Recurrent Neural Networks

Recurrent Neural Networks

Edited by

Xiaolin Hu and P. Balasubramaniam

I-Tech

Published by In-Teh

In-Teh is Croatian branch of I-Tech Education and Publishing KG, Vienna, Austria.

First published September 2008
Printed in Croatia

A catalogue record for this book is available from the University Library Rijeka under no. 111225073
Recurrent Neural Networks, Edited by Xiaolin Hu and P. Balasubramaniam
p. cm.
ISBN 978-953-7619-08-4
1. Recurrent Neural Networks, Xiaolin Hu and P. Balasubramaniam

Preface

The research of neural networks has experienced several ups and downs in the 20th century. The last resurgence is believed to be initiated by several seminal works of Hopfield and Tank in the 1980s, and this upsurge has persisted for three decades. The Hopfield neural networks, either discrete type or continuous type, are actually recurrent neural networks (RNNs). The hallmark of an RNN, in contrast to feedforward neural networks, is the existence of connections from posterior layer(s) to anterior layer(s) or connections among neurons in the same layer. Because of these connections, the networks become dynamic systems, which bring many promising capabilities that the feedforward counterparts do not possess. One of the obvious capabilities of RNNs is that they can handle temporal information directly and naturally, whereas feedforward networks have to convert the patterns from temporal domain into spatial domain first for further processing. Other two distinguished capabilities possessed by RNNs refer to associative memory and optimization, which were initially revealed by Hopfield and Tank.

The field of RNNs has evolved rapidly in recent years. It has become a fusion of a number of research areas in engineering, computer science, mathematics, artificial intelligence, operations research, systems theory, biology, and neuroscience. RNNs have been widely applied for control, optimization, pattern recognition, image processing, signal processing, etc. The aim of the book is to bring together reputable researchers from different countries in order to provide a comprehensive coverage of advanced and modern topics in RNNs not yet reflected by other books. This collective product comprises 18 contributions submitted by 51 authors from 16 different countries and areas. It covers most of the current main streams of RNN researches, ranging from human cognitive behavior modeling, dynamic system identification and control, temporal pattern recognition and classification, optimization, and stability analysis. According to these themes, the 18 contributions are grouped into five categories, corresponding to five parts of the book.

The concept of neural network originated from neuroscience, and one of its primitive aims is to help us understand the principle of the central nerve system and related behaviors through mathematical modeling. The first part of the book is a collection of three contributions dedicated to this aim. Both Chapter 1 and Chapter 2 address neurodynamics in RNNs that are used to model cognitive processes. It is well-known that nonlinear dynamic systems may possess a variety of properties such as attractors, bifurcations, chaos, etc., which are useful in different circumstances. For modeling continuous thread of natural behaviors of humans, a neural system converging to a small region of the state space such as equilibria or limit circles appears not to be competent, though such a property is desirable in other situations (e.g., see Chapters 13-16). In Chapter 1, adaptive thresholds are proposed to model neural homeostasis into the basic formulation of RNN to produce chaotic behavior, which can be used as a source of behavioral exploration and novelty in embodied neural agents. To understand how the brain controls the human movements, Chapter 2 presents a dynamic recurrent neural network modeling approach. In contrast to Chapter 1, this chapter deals with more practical and concrete problems: oculomotor and arm movements. Chapter

3 attempts to construct RNNs to learn languages. It might be thought that RNNs are suitable for such a task considering that a sentence is a sequence of words and RNNs are good at learning sequences. However, things are often not as easy as they look to be. For instance, the linguistic productivity, a key property of any natural language, is notoriously hard to tackle. In the chapter, the authors discuss the modeling of two features of languages that support linguistic productivity, recursiveness and systamaticity, by means of RNNs. Though building an intelligent agent capable of learning natural languages still seems to be a far away goal, we have at least opened a door with the RNN key allowing further exploration.

The second part of the book consists of seven chapters, all of which are about system identification and control. The success of these approaches largely relies on the RNN's ability of recalling history and predicting future. Chapter 4 proposes a new Kalman filter closed loop topology of RNN for identification and modeling of an unknown hydrocarbon degradation process carried out in a biopile system and a rotating drum. Then, an indirect sliding mode controller and a direct recurrent feedback-feedforward neural controller are designed. In Chapter 5, a neural controller is proposed to approximate an ideal tracking controller, which is capable of both structure learning and parameter learning. Based on that, a robust controller is designed to attenuate the effects of the approximation error on the tracking performance. Fuzzy logic is an efficient method to process imprecise, uncertain and noisy information, which is often encountered in the real world. Integrating fuzzy logic and neural networks may foster powerful schemes that combine their respective advantages. Chapter 6 and Chapter 7 provide two examples. The major difference between them is that the former adopts the conventional fuzzy logic system while the latter adopts an extended version, so-called type-2 fuzzy logic system. Chapter 8 presents an adaptive recurrent neural network controller to prevent rollover in heavy vehicles. The control scheme is composed of a recurrent neural identifier and a controller, where the former is used to build an on-line model for the unknown plant, and the latter to force the unknown plant to track the reference trajectory. For a training algorithm, the stability and convergence speed are two major indicators of performance evaluation, but they are often like two sides of a coin and can not be reconciled. Chapter 9 attempts to seek a tradeoff between them by introducing a robust adaptive gradient descent training algorithm. The proposed algorithm is applied to three quite practical problems including time series prediction, system identification, and attractor learning for pattern association. Good results have been obtained. Chapter 10 introduces a deterministic linearized RNN and its application to rainfall-runoff processes. One of the remarkable merits of the proposed RNN is that, its special structure allows a direct interpretation of the network weights in the language of hydrology, considering that most other neural networks are black-box models that lack physical meaning of weights.

The third part of the book is composed of Chapter 11 and Chapter 12, where two interesting RNNs are discussed, respectively. One is called Elman RNN, which is in principle a regular feedforward network but with a hidden layer holding the output of another hidden layer. The other is called locally recurrent probabilistic neural network (LRPNN) coupling with swarm intelligence algorithms and concepts. The two networks also take advantage of RNN's ability of processing temporal information, similar as the networks presented in the previous parts of the book. However, here the networks are used for (temporal) pattern classification and recognition instead of system parameter identification or system behavior prediction. Specifically, Chapter 11 presents applications of the Elman

RNN in biomedical engineering and nuclear engineering, and Chapter 12 presents applications of the LRPNN in speech recognition and classification.

The fourth part of the book comprises four chapters focusing on optimization problems. Doing optimization in a way like the central nerve systems of advanced animals including humans is promising from some viewpoints. On one hand, it may help us understand more about the working principle of nerve systems, and on the other hand, the inherent dynamics of resulting RNNs makes it possible for analog circuit implementation, which would tremendously accelerate the computing tasks, which is favored in many engineering applications. Chapter 13 proposes a general framework for solving various optimization problems including combinatorial optimization problems, linear programming problems and nonlinear programming problems, by using RNNs. For every problem, two energy functions associated with the objective function and constraints are separately defined and minimized by modified Hopfield-type neural networks. The simulation results are given to demonstrate the effectiveness of the proposed method among the specialized methods. The latter two chapters achieve this desirable feature by making use of optimality conditions such as primal-dual conditions or Karush-Kuhn-Tucker conditions. In particular, Chapter 14 describes several RNN models for linear programming problems, quadratic programming problems, nonlinear programming problems, and variational inequalities. As all of the optimization problems considered in the chapter are convex, and variational inequalities are monotone, it is possible to devise globally convergent RNNs for solving them, which is actually the case for the models presented in this chapter. However, the situation gets complicated when the problems become nonconvex or nonmonotone. In this aspect, progresses made in recent years have been reviewed in Chapter 15. It is found that based on similar ideas one could at best devise locally stable and convergent RNNs for most of such problems, though in some particular cases globally convergent models can be needed. The results are obtained without using the penalty parameters. Chapter 16 presents an interesting application of an RNN for optimization. It is found that under certain conditions, the optimal control of a dynamic system can be transformed into a linear programming problem. An annealing RNN is then applied to solve the problem. As this problem is not specific for the proposed network; it can also be tested to other RNNs in the literature.

The last part of the book consists of two chapters, and each presents stability analysis of a variant of the original Hopfield continuous-time neural network. Chapter 17 considers a class of uncertain stochastic high-order neural networks with time varying delays. Based on the Lyapunov stability theory, some new global asymptotic stability criteria are obtained. Chapter 18 considers a discrete-time Hopfield-type neural network with delays. A complete bifurcation analysis is presented for the two neurons case, uncovering the structure of the stability domain of the null solution, as well as the types of bifurcations occurring at its boundary.

As mentioned in the beginning, RNN has a rather broad coverage of applications and we admit there must be some topics that the book does not address. One of the missing topics refers to associative memory, an important capability of RNNs, though there is a case study in Chapter 9. Anyway, this collective product complements other salient books in the same line.

We would like to express our sincere gratitude to all chapter authors for the time and effort they generously gave to the book. We would also like to thank the publisher, I-Tech Education and Publishing, for cooperation in publication. Special thanks go to Dr. Vedran

Kordic, who initiated this project. The first editor would like to acknowledge the support from the National Natural Science Foundation of China under the grant No. 60621062 and 60605003, the National Key Foundation R&D Projects under the grant No. 2003CB317007, 2004CB318108 and 2007CB311003, and the Basic Research Foundation of Tsinghua National Laboratory for Information Science and Technology (TNList).

July 31, 2008

Editors

Xiaolin Hu
State Key Laboratory of Intelligent Technology and Systems
Tsinghua National Laboratory for Information Science and Technology (TNList)
Department of Computer Science and Technology
Tsinghua University, Beijing 100084,
China

P. Balasubramaniam
Department of Mathematics,
Gandhigram Rural University
Tamilnadu,
India

Contents

Preface V

1. Aperiodic (Chaotic) Behavior in RNN with Homeostasis as a Source of Behavior Novelty: Theory and Applications 001
Jorge Simão

2. Biological Signals Identification by a Dynamic Recurrent Neural Network: from Oculomotor Neural Integrator to Complex Human Movements and Locomotion 021
Guy Cheron, Françoise Leurs, Ana Bengoetxea, Ana Maria Cebolla, Jean-Philippe Draye, Pablo D'alcantara and Bernard Dan

3. Linguistic Productivity and Recurrent Neural Networks 043
Akito Sakurai and Yoshihisa Shinozawa

4. Recurrent Neural Network Identification and Adaptive Neural Control of Hydrocarbon Biodegradation Processes 061
Ieroham Baruch, Carlos Mariaca-Gaspar and Josefina Barrera-Cortes

5. Design of Self-Constructing Recurrent-Neural-Network-Based Adaptive Control 089
Chun-Fei Hsu and Chih-Min Lin

6. Recurrent Fuzzy Neural Networks and Their Performance Analysis 107
R.A. Aliev, B. Fazlollahi, B.G. Guirimov and R.R. Aliev

7. Recurrent Interval Type-2 Fuzzy Neural Network Using Asymmetric Membership Functions 127
Ching-Hung Lee and Tzu-Wei Hu

8. Rollover Control in Heavy Vehicles via Recurrent High Order Neural Networks 151
Luis J. Ricalde, Edgar N. Sanchez, Reza Langari and Danial Shahmirzadi

9. A New Supervised Learning Algorithm of Recurrent Neural Networks and L_2 Stability Analysis in Discrete-Time Domain 169
Wu Yilei, Yang Xulei and Song Qing

10. Application of Recurrent Neural Networks to Rainfall-runoff Processes 203
Tsung-yi Pan, Ru-yih Wang, Jihn-sung Lai and Hwa-lung Yu

11. Recurrent Neural Approach for Solving Several Types of Optimization Problems 229
Ivan N. da Silva, Wagner C. Amaral, Lucia V. Arruda and Rogerio A. Flauzino

12. Applications of Recurrent Neural Networks to Optimization Problems 255
Alaeddin Malek

13. Neurodynamic Optimization: Towards Nonconvexity 289
Xiaolin Hu

14. An Improved Extremum Seeking Algorithm Based on the Chaotic Annealing Recurrent Neural Network and Its Application 309
Yun-an Hu, Bin Zuo and Jing Li

15. Stability Results for Uncertain Stochastic High-Order Hopfield Neural Networks with Time Varying Delays 323
P. Balasubramaniam and R. Rakkiyappan

16. Dynamics of Two-Dimensional Discrete-Time Delayed Hopfield Neural Networks 343
Eva Kaslik and Ştefan Balint

17. Case Studies for Applications of Elman Recurrent Neural Networks 357
Elif Derya Übeyli and Mustafa Übeyli

18. Partially Connected Locally Recurrent Probabilistic Neural Networks 377
Todor D. Ganchev, Konstantinos E. Parsopoulos, Michael N. Vrahatis and Nikos D. Fakotakis

Aperiodic (Chaotic) Behavior in RNN with Homeostasis as a Source of Behavior Novelty: Theory and Applications

Jorge Simão
Computer Science Department, Faculty of Sciences, University of Porto
Center for the Sciences of Computation, Cognition and Complexity
Portugal

1. Introduction

One way to understand cognitive system is to think in terms of dependency relationships between the neural controller or micro level, and the agent's body configuration or macro level. Neural dynamics, as modeled in Recurrent Neural Networks (RNN), is determined by units and connections self-organization rules. This micro dynamics guides body configuration as it commands muscular action. On the other hand, an agent's self-perception causes the body configuration state to influence neural dynamics. Cognitive agents thus work in a multi-level causality loop.

An apparent limitation of RNN to model neural controllers for cognitive agents is that the dynamics may converge to a small region of neural state space. In the extreme case, this includes convergence to a fixed point or to limit cycles where only a few neural states are visited. Since agent's body configuration is mostly determined by neural activity, limited neural dynamics also implies a limited dynamic in an agent's body — as being completely "frozen" or keep doing the the same thing over and over again.

Because natural cognitive agents, understood as animals and humans, maintain an almost continuous thread of behavior while they are awake, one can suggest that neural controllers for cognitive modeling and engineering should also allow for this kind of behavior. RNN with adaptive thresholds, modeling neural homeostasis, provide one possible answer. When units in a RNN are endowed with a rule for dynamically changing units thresholds the neural network as a whole behaves in a complex manner, ranging from a close to periodic behavior to aperiodic (or chaotic) behavior. When coupled to an agent's body the neural dynamics can be used to produce variability in body configuration dynamics — this is the cognitive agent's behavior at the macro level. This variability is a key requisite to allow agents the unaided discovery of possibilities of action (*affordances*) of their body in the context of their environment. Behavior habituation to instantaneous body-environment configurations resulting from neural homeostasis, keeps agent continuously exploring the configuration space, thus producing novel body postures and/or move the agent to new locations in the environment.

This mechanism while essential for the production of creative or novel behavior, may not be enough. Without sensors to perceive their body and environment neural activity can not be influenced by the body-configuration. Thus, neural chaos by itself can not guarantee that the agent performs adequately. For example, it may lunch the agent as whole to enter in a self-destructive non-viable biological, psychological, or social region. Introducing sensors that perturb individual neurons and collective dynamics, offers an additional mechanism to develop structured behavior. At the formal level, one can infer that by having sensors for self and environment perception an agent can change the probability distribution of the neural state space. This change, in turn, changes the dynamics and probability distribution of the agent body configuration, possibly steering the agent into more interesting regions.

To illustrate the application of this principles, we show with concrete examples of simple articulated agents how chaos in neural controllers can be used to generate novel behavior and how self-perception can be used to change neural dynamics. Target applications, included muscular control and visual attention. To make the principles general, we also present a conceptual framework for embodied neural agents as models for cognitive systems.

We divide this article into four main parts: in section 2, we make an abstract theoretical characterization of cognitive systems that is useful for the remaining parts of the discussion. In section 3, we describe and discuss the proposed Recurrent Neural Network model that uses units with adaptive thresholds to model homeostasis. In section 4, we use this neural model to build a particular model of a minimalist cognitive agent, endowed with a single link and a joint with only one-degree-of-freedom. The experimental results obtained with this cognitive agent are used study muscular control and to illustrate the application of RNN with homeostasis. In particular, we compare the behavior of agent at the micro and macro level when neural units have or do not have adaptive thresholds and self-perception. In section 5, we describe another model that uses RNN with homeostasis, this time modeling visual attention. In section 6, we present our conclusions and relate our results with others.

2. A meta-model for cognitive systems

For improved understanding of cognitive system, one needs to have a meta-model or meta-theory that allows one to think in abstract terms and helps to identify the relevant entities and concepts specific to the problem domain of cognition. In particular, the relation between agents and the environments in which they live, and the relation between its neural controller dynamics and body configuration dynamics needs to be put in appropriate perspective. If this is achieved, we are better equipped to see what are the relevant elements that models of neural networks need to take into consideration to work effectively as models of cognition. In this section, we present such a meta-model organizing the components sub-sections as follows: section 2.1 presents the concepts needed to begin understanding cognition; section 2.2 further develops the relation between neural controllers and behavior of agents; section 2.3 formally characterizes cognitive systems as complex dynamical systems.

2.1 Situated cognitive agents and environments

We characterize cognitive agents as complex systems that can be studied at two different complexity levels: the *macro-level* and the *micro-level*. The macro-level is defined by the *configuration state* — a formal description of the agents body posture in space and time, as seen by an external observer or as made apparent to the agent itself through self-perception. A small number of degrees-of-freedom is often required to describe an agent at this level (e.g., the variables of the joint angles plus the parameters of link geometry, as is often used in robotics).

The micro-level is a characterization of the state of agent neural controller. In simple neural models, this may include the activation level of neural units, units' thresholds, and neural connections' weights. Usually, the micro-level requires a much higher number of *degrees-of-freedom* to be fully described than the macro-level, since an agent with few links and joints may have a controller with many neural units. Interfacing the micro and macro-levels, agent descriptions include the way the neural controller is connected to the agents' body — both in muscular connections (efferent) and in the way sensation-perception cells/inputs impinge on the neural controller. In complex articulated agents, the number of macro-level variables and parameters needed may be in high number (e.g., on the order of dozens), but we always assume that the micro-level requires a much high number of variables and parameters to be described. A physical (non-cognitive) systems analogy of this, would be a rigid body (object) described at the macro-level by a few variable and parameters (e.g., for geometry, location and orientation in space, and material properties), and that at the micro-level requires much more variables if one wanted to describe in detail where all its constituent particles/elements are located in space at a given time, assuming, for illustration purposes, that this could be done in practice.

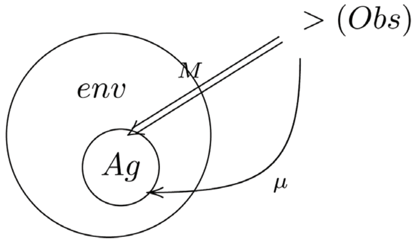


Figure 1. Conceptual diagram of a cognitive agent, its environment and the external observer

Real and virtual agents are often situated in some environment, in such a way that its behavior and interaction with the environment can be observed by an external observer. As pointed out by many classical thinkers and researcher in the AI community, the agent's own perception of the environment may be quite different from an external observer's perspective [10]. Namely, external observers can not make easily educate guesses about the subjective perspective of the observed agent own perception (e.g., the perspective a human and another animal, such as a dog or frog, might have from the same environment, say a tree, might be quite different — assuming for illustrations purposes, that the two of them could somehow be compared). In Figure 1, we make a sketch representation of the relationship between the agent, its environment, the external observer, and the two levels of description. Below, we postulate that an agent can be sensitive to its own actions by means of self-perception, and we use this to provide a causal account of how such self-generated information can be used to guide autonomously the behavior development of the agent through learning at the micro-level.

The activity of neural units often dictates the generation of body movements, by commanding internal force to be made by muscular-like structures. Given this, the dynamics of body movements as captured by the formal configuration definitions, is a

reflection of the dynamics of the neural units (plus whatever mechanical external perturbations the environment might impose in the agent at a given time). This is micro-macro causality mechanism. Moreover, selforganizing mechanisms at the micro-level (e.g., learning and homeostasis) may change the probability distribution of states of the micro-level, and these can also leave a trace at the observed behavior. This macro behavior is emergent from the micro-level activity.
An additional consequence of the above characterization, is that the mapping from the (micro) neural level to the (macro) configuration level is not one-toone, since the number of degrees-of-freedom are different. Many different neural states may mandate the same body configuration. Moreover, since body limbs are pulled by several muscular structures each with possibly many different force generating components (e.g., muscular micro-fibers), the coordinated action of a large number of neurons and muscular cells is usually required to generate strong and high-amplitude body movements. We also explore these aspects below.

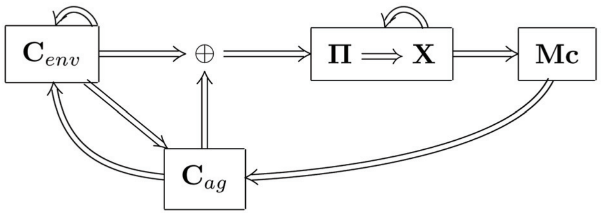


Figure 2. Block diagram of cognitive agents.

Since agents have component units sensitive to the environment (the sensationperception inputs), agents can receive feedback of the "world-state" (as inferred by their sensorial apparatus). This is interpreted as a macro-micro (downward) causality mechanism. Moreover, because agent's sensitivity of the macro world also applies to its own body state, agents can sense the effects of their own actions (e.g., using input from proprio-perceptive cells in muscles and tendons, by visually looking at body limbs — such as hands, or by listing the sounds produced by itself). Below, we call this type of macro-micro causality mechanism as *self-perturbation* or self-perception.

2.2 Multi-level causality

Once we make a micro–macro characterization of cognitive systems, we need to focus on the causal relations between the two levels. Figure 2, represents these causal relation in agent behavior according to the presented meta-model. **X** represents the (activation) state of the neural controller of the agent (part of micro-level or internal state), and $\mathbf{C}_{ag}$ represents the body configuration of the agent (the macro-level or external state). The self-loop in **X** represents the internal dynamics of the controller, such as modeled in recurrent neural network models. The arrow from **X** to the motor units **Mc** represent the commanding of muscular contraction/distention causing the generation or cessation of internal force. The connection from **Mc** to $\mathbf{C}_{ag}$, represents the actual changes made in body configuration caused by changes in internal state (if any).
Due to self-perturbation (in any sensorial modality) the agent configuration $\mathbf{C}_{ag}$ generates input or perturbation to the neural dynamics — represented as **Π**. This represents part of the macro-micro causality. As a "side-effect" of the controller internal dynamics, changes in body configuration may change the state or configuration of the environment, in the

diagram represented as $\mathbf{C}_{env}$ (e.g., as in a manipulation task). Changes to environment state trigger additional perturbation to the neural controller. For individual neural units, the two types of perturbation (self-generated and other) should be considered (mechanistically) indistinguishable. Additionally, in a complex task-environment, environments may also have complicated dynamics of their own (e.g., gravity, dumping, reaction force, etc.) — represented as a self-loop in the box labeled $\mathbf{C}_{env}$ in Figure 2. The environment may also impose macro-perturbation in the configuration of the agent, abstracted as mechanical external forces $\mathbf{F}_{ext}$. We represent this as an additional arrow in Figure 2 from $\mathbf{C}_{env}$ to $\mathbf{C}_{ag}$. We aggregate the agent configuration $\mathbf{C}_{ag}$ and environment configuration $\mathbf{C}_{ag}$, and call it the global configuration or just configuration for short. We represent this as: $\mathbf{C} = \mathbf{C}_{ag} \cdot \mathbf{C}_{env}$.
Changes in neural activity often (but not necessarily always), create body limb movements because they command muscular-like structures that create internal mechanical forces on the body. In most natural situations, body limb movements are also dependent on other mechanical external perturbations on the agent that combine to self-generate internal forces. This may include forces such as gravity, object contact reaction-force, and physical manipulation by social other. This aspects of agent-environment interaction are not developed in the chapter.

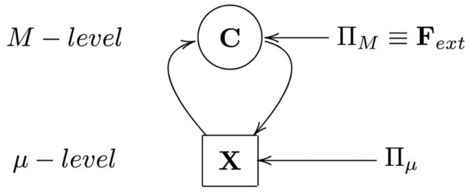


Figure 3. Diagrammatic representation of multi-level causality with two types of perturbation: **X** is the micro state and **C** is the macro state as seen by an external observer or by the agent itself. Π_μ represents the micro-perturbations, mostly due to input to sensorial-perception units/cells, and $\mathbf{F}_{ext}$ represents macroscopic/mechanical perturbations, also represented as Π_M.

Since agents have component units sensitive to the environment (the sensationperception inputs), agents can receive feedback of the "world-state". This input may change internal neural dynamics, and in turn change the internal forces that cause body limb movements. Agents may also have perception of their own body state (e.g. thought proprio-perception of limb displacement, visual perception of own body, or self-produced sounds). In the model presented below, we focus our attention on studies of a simple form of proprioceptive muscular input.
Given this characterization, we see that micro and macro level are connected in a two-way causality loop. The state of the micro-level determines/influences body configuration, and the body configuration perturbs the internal dynamics of the neural controller due to self-perturbation. Figure 3, further illustrates the notion *multi-level causality* in cognitive systems. An upward arrow is used to represent emergence or upward causality, and an downward arrow represents downward causality due to self-perception and perception of the environment.
This characterization of embodied neural agents relates to Ashby classical characterization of adaptive agents and agent-environment couplings as dynamical systems [2], further explored in mainstream situated AI literature [3]. The above presentation, although similar

in general form, makes additional distinctions. In particular, it makes explicit and gives theoretical significance to the difference between the typical number of degrees-of-freedom at the micro or neural level, and the macro or configuration-level. Namely, $|\mathbf{X}| >> |\mathbf{C}_{ag}|$.

2.3 Characterization as complex dynamical system

From a formal point of view, the agent body, neural controller, and environment represent a (complex) dynamical system that can be summarized with two (vectorial) coupled differential equations:

$$\begin{cases} \dot{\mathbf{X}} = f_a[\mathbf{X}, f_\Pi(\mathbf{C})] \\ \dot{\mathbf{C}} = f_C[Mc(\mathbf{X}), \mathbf{C}] \end{cases}$$

where f_a is the neural units activation function, f_Π some (possibly complicated) function that maps agent and world configuration to a particular value of individual units perturbation, and f_C and Mc are functions that relate changes in internal state with changes in agent body and world configuration. We are ignoring here and until the next section, second-order dynamics in the neural controller, such as learning and/or homeostasis.

When we make the assumption that the neural state fully determines body posture (e.g. due to lack of body inertia), than the differential equation above for the configuration can be simplified to a functional equation: $\mathbf{C} = f_C[Mc(\mathbf{X})]$. That is the neural state fully determines the instantaneous body configuration. When no confusion in caused, we abbreviate the above equation to: $\mathbf{C} = f_C(\mathbf{X})$.

For simplicity sake, we leave ambiguous whether the information about configuration state the agent uses is the same or comparable with the information a particular external observer might use to characterize the agent and its environment state. For purposes of neural control, the relevant information is the information the agent uses.

3. A model of RNN with homeostasis

In previous section, we made an abstract characterization of embodied cognitive agents that is independent of the controller and neural model used to generate its behavior. In this section, we propose a model of Recurrent Neural Networks with adaptive threshold capturing homoeostasis behavior in natural neural cell [16]. In section 3.1, we present the equation for neural dynamics. This is a variation of the continuous Hopfield RNN model [7], where units threshold changes to push activation back to a resting value. In section 3.2, we discuss how the non-embodied RNN model can be extended to control an embodied system as is the case of cognitive agents with a body living in some environment.

3.1 Equations for neural dynamics

The neural model consist of a set of units whose activation levels is described b a vector $\mathbf{X} = [x_1, \ldots, x_i, \ldots, x_N]$, with $|\mathbf{X}|$ as the number of units. Neural units activation x_i is constrained to lie in the interval range $[x_{\min}, x_{\max}]$, where $x_{\max}$ is the saturation value and $x_{\min}$ is the lowest/depression value. Units are also assumed to have a rest or natural activation value x_0. In computer simulation we make neural units start in this rest/natural activation state.

Neural units are assumed to be connected in a network/graph as a fully recurrent neural network (all units connect to all) [7]. Connection strengths are represented by a connectivity matrix $\mathbf{M}$, where element c_{ij} represents the connection strength or weight between unit i and

j. In the simulation results presented below we experiment mostly with fixed connection weights. Neural units are assumed to be initially connected with random weights, using a normal distribution with mean value 0 and variance $\sigma^2(M)$.
Neural units have an adaptive threshold that is used to maintain units in a sensitive state. This is equivalent to cellular homeostasis mechanisms in biological neural networks [16]. For unit *i* we represent its threshold as θ_i. When a unit's activation is very high, a slow adaptation process takes place that gradually moves the activation value to a rest or natural activation value x_0. Likewise, when a unit's activation value is low the same adaptation process takes place to raise the activation level to x_0.
The operation of units is formally defined using two ordinary first-order differential equations [approximated by the Euler method in the simulations below]. The first equation below describes the (fast) dynamics of individual unit's activation. The second equation describes the (slower) dynamics of homeostasis.

$$\begin{cases} \tau_1 \dot{x}_i = -x_i + x_0 + f(\sum_j c_{ji} x_i + \pi_i - \theta_i) + \xi \\ \tau_2 \dot{\theta}_i = x_i - x_0 \end{cases}$$

above τ_1, τ_2, with $\tau_1 << \tau_2$, are constants for the characteristic times of the neural processes modelled. x_0 is the resting or natural activation of units. *f* is an activation gain function. The simplest case is to have f a linear function with a constant gain $G = 1$. ξ is some (optional) random noise value.
Solving for equilibrium in the first equation, $\tau_1 \dot{x}_i = 0$, shows that at rest $x_i = x_0 + f(\Sigma_j c_{ji} x_i - \theta_i) + \xi$, which is a fast quiescent/rest state. Solving for equilibrium for the second equation, $\tau_2 \dot{\theta}_i = 0$, show that at rest $x_i = x_0$, which is a slow quiescent/rest state (since $\tau_1 << \tau_2$). Simulation results presented below show that full equilibrium (that is, $x_i = x_0$ for all units) is often not reached due to units' interconnections.
Neural connections can be made to have weights changed similarly to Hopfield networks by using a Hebb-like learning rule. This level of plasticity allow neural agents to have more adaptation possibilities since it introduces a second-order dynamics in the system. In this chapter, we will focus in networks without learning.

3.2 Embodiment neural agents

A straightforward way to give an embodiment to a RNN (with or without homestasic units), is to postulate that each agent actuator is controlled by a sub-set of units. Formally, if agent configuration state and state space is defined by vector $\mathbf{C} = [\psi_1, \ldots, \psi_i, \ldots, \psi_{N'}]$, with $|\mathbf{C}|$ the number-of-degree of freedom of the configuration, then we make each degree-of-freedom ψ_i to be a function of a sub-set of neural units $\mathbf{X}|_i$. In mathematical notation: $\psi_i = f(\mathbf{X}|_i)$. Due to this functional relations, movement in neural state space may produce some kind of movement at the configuration level. On the other hand, since $|\mathbf{X}| >> |\mathbf{C}|$, neural dynamics may be sufficiently confined to make changes in agent configuration minimal.
To make the neural units to receive feedback about behavioral consequence of neural dynamics sensorial mechanisms need to be used. One way to model this is to think that each agent sensor has the ability to produce a perturbation π_i that adds to a units input, with a gain c_i. This slightly changes the equation for neural dynamics, as follows:

$$\tau_1 \dot{x}_i = -x_i + x_0 + f(\sum_j c_{ji} x_i - \theta_i + c_i \pi_i) + \xi$$

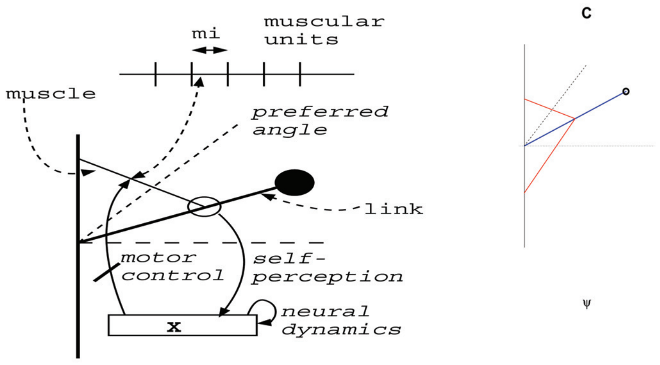


Figure 4. Body configuration of a minimalist articulated agent with a single link and rotational joint in 2D plane (one degree-of-freedom), controlled by an artificial muscle composed of a set of muscular units: **left)**: abstract design; **right)**: visualization in the developed neural agent simulator.

4. A minimalist embodied neural agent

In this section, we present a model of an embodied neural agent that relies on a RNN with homeostasis to control its behavior. In particular, we show that the homeostasis introduces aperiodic (chaotic) behavior in the system preventing the agent to ever reach a stationary regime. This is argued to be useful for characterizing cognitive systems, since behavioral exploration and continuous novelty is a distinguishing feature of this type of systems.

4.1 Model

We consider an embodied articulated agent with a single link and a single joint. The joint angle fully defines the body configuration of the agent. The joint angle is determined by the contraction of a simplified muscle that works like a mechanical lever. The muscle has a large number of muscular units m_i. The contraction/ extension of a muscular unit m_i produces a spatial displacement Δs_i, and the summation of all displacements determines the joint angle. Formally, $\psi = f(\Sigma_i \; \Delta si)$, where f is a function of the detailed geometry of the agent. We assume that the contraction of a single neural units produces a relatively small link displacement. Therefore, the simultaneous contraction of a large proportion of muscular units is required to generate maximum displacement of the link. Additionally, the joint angle ψ is always constrained to lie within a maximum amplitude interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$. The agent also contains proprioceptive mechanisms for muscle contraction/distention or link angular position (discussed below). In Figure 4, we show the abstract design of the agent. We also show the graphical design of the agent as visualized in a developed simulator.

Muscle contraction (and thus body configuration) is controlled by a neural population with N units. We make a simple attachment between this neural population and the muscle units, by making the number of muscular units equal to the number of neural units, and connecting them one-to-one (unidirectional). Muscular contraction is thus proportional to the total activation of the network. When all units are in a rest/natural activation value, ϕ takes value 0 (the link is horizontal).

A fixed proportion f_π of neural units is sensitive to the angular position of the link. Namely, we define a preferred angular position ϕ * and cause that sub-set of units to receive additional excitation the closer the link is to that preferred position/angle. Thus, this units work as proprio-perceptive (proprioceptive) neural cells. The concrete equation we use to model this is:

$$\pi_i = K_1 e^{-\frac{(\phi-\phi^*)^2}{K_2}}$$

where π_i is the external perturbation to the cell due to proprioceptive input, K_1 is a parameter for maximum perturbation value, $(\phi - \phi^*)^2$ is the squared difference of the link's current angular position and the preferred angle ϕ^*, and K_2 is a parameter for how slow perturbation decreases as the current angle moves away from the preferred angle ϕ^*. (K_2 is also represents the variance of a Gaussian curve.) In the simulation results presented below, f_π is always set to .3, ϕ^* always set to 80°.

4.2 Experimental results

We have performed several experiments to study the behavior of the previous presented model. In these experiments, we generate neural controllers with random connections according to the weight matrix **M**, using mean 0 and variance $\sigma^2(M) = 1$. For the presented results, we made connection weights fixed (no learning), and removed internal noise. Parameters for unit's activation were set as follows: $x_0 = 1$, $x_{\max} = 3$, $x_{\min} = .1$. Perturbation parameters where set as $K_1 = 5x_{\max}$, and $K_2 = 2$. Neural activation levels x_i at time $t = 0$ are always set to x_0. In studying model behavior, we look both at the neural (micro) and link configuration (macro) levels. We also look both at the dynamical and the stochastic aspects of model behavior.

Neural Dynamics without Homeostasis

When units homeostasis is not put in the model's operation ($\tau_2 = +\infty$), the neural activation state and the configuration angle converges in most simulation runs to a **fixed point**. In fixed points, a large proportion of units are either fully saturated ($x_i = x_{\max}$) or fully depressed ($x_i = x_{\min}$). In some simulation runs, some units converge to intermediary values (closer to x_0). Simulation runs with different random connection matrices produce different fixed points. Figure 5 shows the evolution in neural state space and link-configuration state spaces, along side with corresponding probability distributions, for a particular simulation run during 200 time steps. (High activation of units is coded as red in color plates, low activation as blue, and values near x_0 as green.) This is a similar behavior to that observed in recurrent neural networks with symmetric connections, as in Hopfield RNN [7]. In a small proportion of simulation runs with different connection weight matrices, neural dynamics converges to a small region of state space usually in the form of a **periodic cycle**. In these scenarios, most neural units are either in fully depressed or saturated regime, as in fixed points solutions, but a proportion of cells oscillates due to non -symmetric and opposite sign connections. In Figure 6 we show the evolution of the configuration state for one of such simulation run, showing a small periodic cycle that corresponds to an oscillation of low amplitude in configuration space. In a set of 10 consecutive runs with the same settings (but different weight matrices), the results obtained were qualitatively similar — following one of these two cases. The results (either fixed points or periodic cycle) also appeared in controllers and networks with different number of neural units, from 5 to 50. Previous work also showed that the introduction of considerable noise is (most of the times) not enough to take the system away from fixed-points or small-regions of state space [14, 13]. Formally, this means that fixed points are either attracting or Lyapunov stable (neural states tend to stay within a small distance of a fixed point when perturbed [15]).

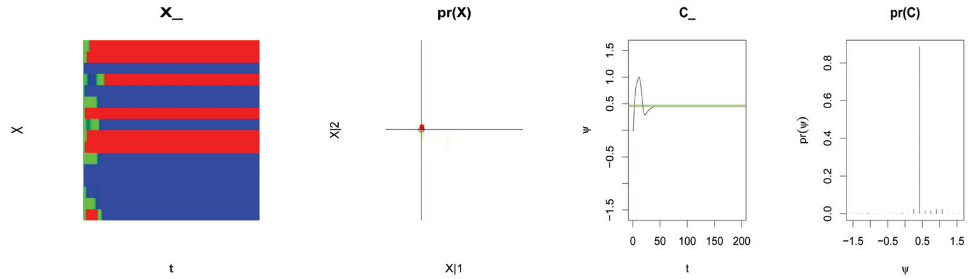


Figure 5. Dynamics without homeostasis with convergence to a fixed point ($\tau_2 = +\infty$, $N = 16$). left-to-right) neural activation history [in (blue, green, red) color code for depressed, rest, and saturated activation levels]; probability distribution of neural state space mapped to two dimensions (**X**|1 is the total activation of units index $[1 : \frac{N}{2}]$, and **X**|2 is the total units activation index $[\frac{N}{2}+1, N]$)); time series for agent-link configuration angle ; probability distribution for angle configuration angle ψ.

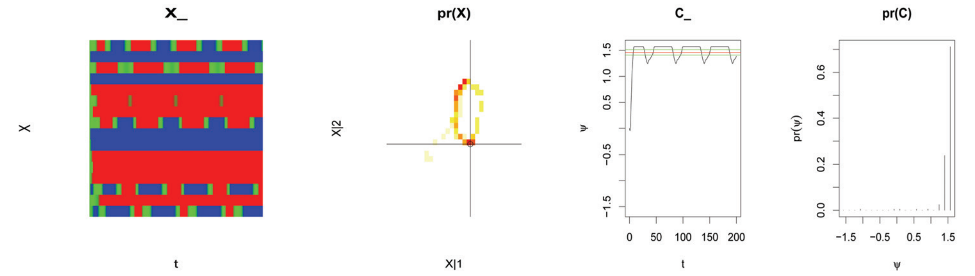


Figure 6. Dynamics without homeostasis — convergence to a small region of state space or periodic cycle ($\tau_2 = +\infty$, $N = 16$).

Neural Dynamics with Homeostasis

When units have homeostasis, the behavior of the system changes considerably. The proportion of time that units are not saturated or depressed increases, as inspection of the differential equation for the threshold above suggests. However, most units do not remain with an activation value near x_0 all the time since they are taken away from homeostasis due to interconnection with other units. Figure 7 and figure 8 shows two qualitatively typical simulation runs. [Noise is absent, $\sigma^2(\xi) = 0$.] The system state does not converge to a fixed point or some simple attractor, but exhibits behavior that qualitatively can be categorized between **non-periodic behavior** and **nearly periodic**, due to threshold adjustments [15]. When the number of units is small, the behavior of systems tends to be closer to periodic behavior (nearly periodic), and when the number of units increases the behavior tends to be more aperiodic. Following Langton [8], such class of qualitative behaviors may be designated as **complex behavior**. This is explained considering that although first-order neural dynamics cause the system to move to a small region of state space, individual units' homeostasis (modeled as threshold adjustments) take the system away from this regions (fixed-points or periodic cycles). This creates the conditions for a wider exploration of state space, when compared with setting where the neural controller as only a first-order dynamics.

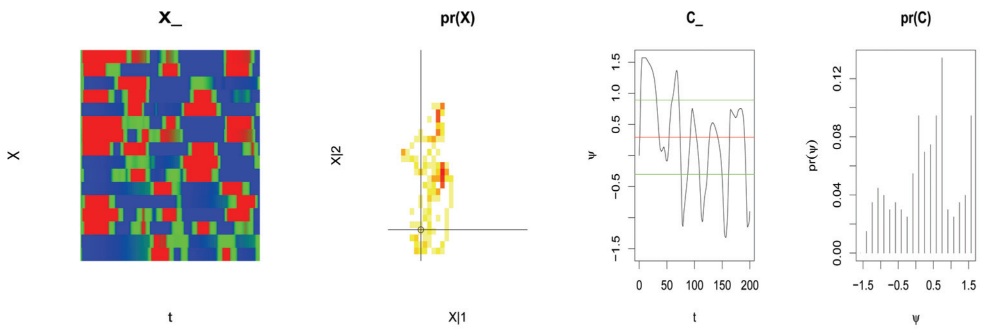


Figure 7. Dynamics with Homeostasis: convergence to aperiodic/chaotic regime ($\tau_2 = \tau_1^{-}1$, N =16).

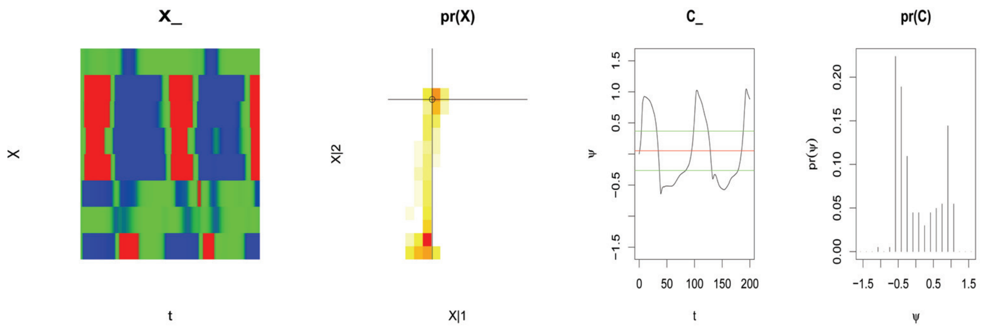


Figure 8. Dynamics with Homeostasis: convergence to a nearly periodic regime ($\tau_2 = \tau_1^{-}1$, N = 8).

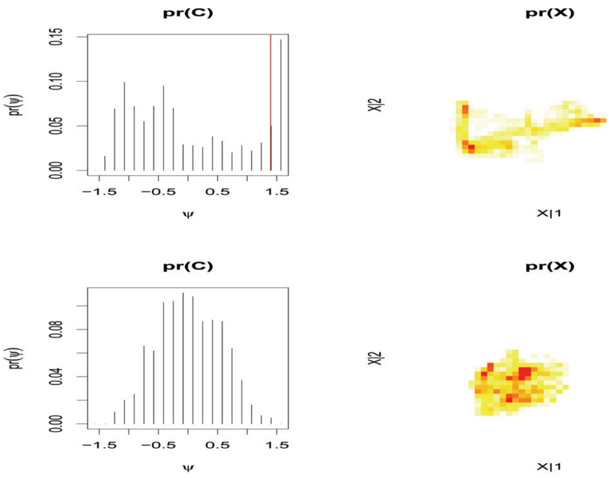


Figure 9. The effects of proprioceptive input in probability distribution of neural and configuration state spaces: **top)** with proprioceptive input the probability distribution become bi-modal, with one high activation region (link up), and one low activation or depressed region (link down); **bottom)**) without proprioceptive input probability distribution become uni-modal, with the region near ψ = 0 of highest probability due to homeostasis.

Neural Dynamics with Proprioceptive Input

To investigate the behavior of the system when proprioceptive input is used, we compared the behavior of the systems with and without proprioception perturbation for neural controllers with the same connection matrices. In particular, we want to see if increasing neural activity when body configuration angle is near a preferred position would increase the probability of the agent to staying near that region. Figure 9 shows diagrams for the probability distribution of the configuration (left) and neural state spaces (right), for one particular neural network with and without (top and bottom) proprioceptive perturbation. The results show that proprioception cause the link angle distribution to become **bimodal**; with the region slightly above the preferred angle, at 1.4 *rad.* (marked with a red vertical bar in the probability distribution graph on the left), to have highest probability, corresponding to a saturated region, and another high probability region corresponding to a region of neural depression (with higher entropy than the saturated region). In contrast to this, for the same connection weights matrix, the neural dynamics without proprioceptive perturbation causes the link distribution to be **uni-modal**. In this case, the region near $\psi = 0$ (link at horizontal position) is the highest probability region. (Previous work, suggests that this distribution can be characterized by a (symmetric) *power-law* distribution [14]).

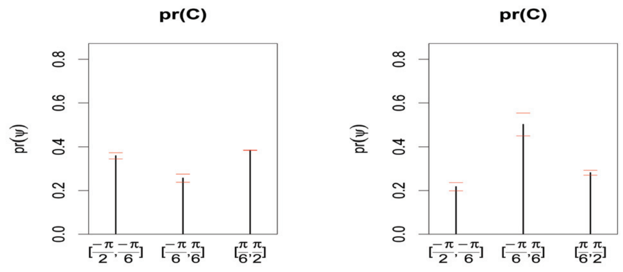


Figure 10. Comparing probability of regions in configuration state space across 10 different simulation runs.

Testing for Robustness

Because, the system's behavior changes considerable with different connection weight matrices, we wanted to test the robustness of these findings across multiple runs with different controllers. For this purpose, we divided the total link-configuration state space, $\psi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, in three regions: a depressed region, corresponding to a interval range of $\psi \in [-\frac{\pi}{2}, -\frac{\pi}{6}]$, a resting region with $\psi \in [-\frac{\pi}{6}, \frac{\pi}{6}]$, and a saturated and high amplitude region, $\psi \in [\frac{\pi}{6}, \frac{\pi}{2}]$. Figure 10 shows the probability distribution of link angle for each of these three intervals (mean and variance) for 10 different simulation runs. The left-hand graph correspond to the settings with proprioceptive input, and in right-hand the graph represents the setting without proprioceptive input. The results confirm the initial observation that introducing proprioceptive input with higher intensity near a preferred region tends to make that region of higher probability.

The bi-modality, induced by proprioception, arrives because positive perturbations tends to increase neural excitation. This can be understood by looking at the equations governing system behavior (here in vector form): $\dot{X} = F(\mathbf{X}, \Pi; \mathbf{M})$, which can be linearized to $\dot{X} \approx F(\mathbf{X};\mathbf{M}) + G \cdot \Pi$, if most units are in linear (non-saturated, non-depressed regime), a condition ensured by homeostasis. Thus, higher input $|\Pi|$ increases the value of first-derivative $\dot{X}$, and higher values of $\mathbf{X}$ correspond to higher configuration angles ψ. Formally, we can say that in the linear regime $\frac{\partial x_i}{\partial \pi_i} > 0$. For link states $\psi = \psi^*$, the first derivative $\dot{X}$ is still positive, so the highest probability angle tends to be higher than ψ^*.

5. Other applications: visual attention

To further illustrate the use of RNN endowed with homeostasis, we describe in this section an additional model this time targeting visual attention. It is a variant of the previous model for muscular control, but now the agent-link as a sensor apparatus for visual perturbation.

5.1 Model

As previously, our agent model description consist of two parts: the description of the agent body and the description of the neural controller. The body of the agent consists of a single link with a tip with visual-input sensitive cells (figure 11). The link position or body configuration is controlled by an antagonistic muscle pair (the left and right muscle), and their contraction-extension depends on the activation of the motor units m_l and m_r, directly connected to them. Activity of motor units fully defines the body configuration of the agent, and consequently the angular position of his visual axes. Formally, we specify the link angle to be:

$$\Phi = k_m \cdot (m_l - m_r)$$

where K_m is a proportionality constant. Therefore, the link will turn to the left when $m_l > m_r$ and to the right when $m_r > m_l$. The link is always constrained to lie within a maximum amplitude interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$.

The visual tip of the agent detects external visual stimuli, modeled here as punctual particles fixed or moving in a direction parallel to the horizontal baseline of the link. The particle position is defined by the angle Φ^*, which is constrained to a maximum amplitude interval $[-\frac{\pi}{4}, \frac{\pi}{4}]$. The presence of the particle imposes a visual input to the neural controller, that includes N_v visual units. A visual unit has a maximum activation value when the stimulus is located at a particular angular position in relation to a preferred position. Formally, visual input is defined as:

$$\pi_i \equiv v_i = \kappa_1 e^{-\kappa_2 (\Delta\Phi + \Phi_i')^2}$$

where k_1 and k_2 are two visual input constants. $\Delta \Phi = \Phi - \Phi^*$ represents the difference between link and particle orientations. $\Phi_i^{'}$ corresponds the preferential angle for each visual unit. With the settings above, $\Delta \Phi$ is constrained to be in the interval $[-\frac{3\pi}{4}, \frac{3\pi}{4}]$, with limits representing the two extreme situations where the particle and the link are as far away as possible.

We made a simple attachment between the visual input units and a neural control population, by making the number of units of each population equal and connecting them one-to-one (unidirectionally). Control units are connected as a fully recurrent neural network — all units connect to all. The control population is connected to the neural motor population such that half of the N_c units connect to the motor unit m_l, and the other half connects to the motor unit m_r. That is:

$$\begin{cases} m_l \propto \sum_{i\in[1:\frac{N}{2}]} x_i \\ m_r \propto \sum_{i\in[\frac{N}{2}+1:N]} x_i \end{cases}$$

5.2 Experimental results

In this section, we present simulation results for basic experimental settings. We focus on experiments with a single particle with a fixed position or simple movements. For more elaborated experimental settings see [1].

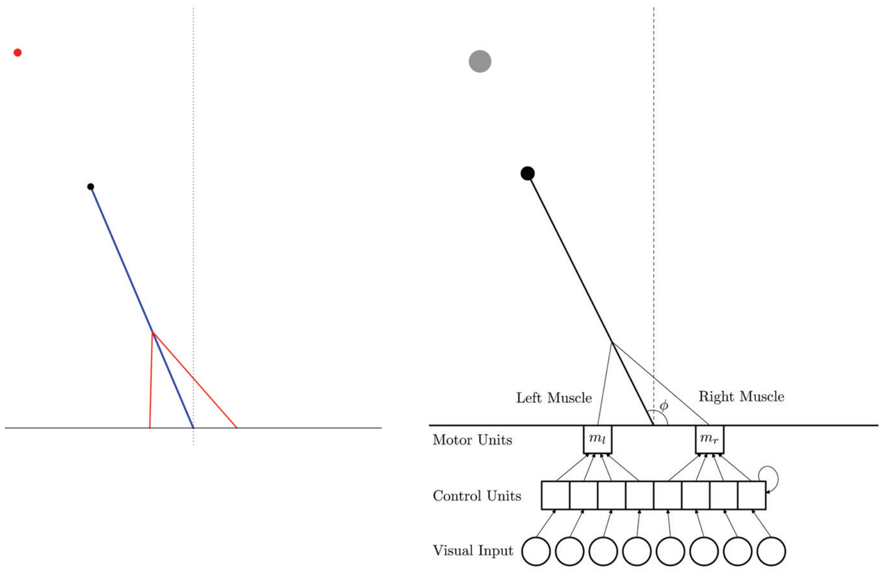


Figure 11. Diagram of a minimalist embodied neural agent with one degree-offreedom for visual attention tasks, subject to a variety of simple visual stimuli (point particles). The agent consists of a single link and joint, representing the orientation of a visual axes, a tip sensitive to visual stimuli, and an antagonistic muscle pair. **left)**: Graphical visualization in the developed neural agent simulator. **right)**: Abstract design representing the agent's body and the set of units controlling muscular contraction-distension, and receiving visual input. The body configuration is defined by the link angle _ as commanded by left and right muscles, whose contraction/extension is set by two motor units m_l and m_r. The motor units are connected to a set of control neural units. Control units are connected in a fully recurrent way (complete connection graph), and each control unit receives input from a corresponding visual input unit.

Neural Dynamics without Homeostasis

The experiments described in this section were used to analyze the effect of homeostasic mechanisms in the visual system, with and without visual perturbation. This study was performed with 8 control units ($N_c = 8$), and then with different sizes of neural populations, $N_c \in \{4, 8, 16, 32, 64\}$.

In the first trials, the neural control population was setup with eight units. Units in the motor population were always set with two units (controlling the left and right muscles).

We set as visual stimulus a point particle situated at a fixed distance from the horizontal basis of the agent. Different simulation runs select a different (randomly selected) angular position for the particle. The weight matrices also take different random values for each simulation run with $V_c = 1$.
Figure 12 shows the system's behavior during 500 time-steps of a particular run, when there is no visual stimuli present in the agent's environment. Here, neural homeostasis is turn off ($\tau_2 = +\infty$). The left-handed side of figure 12 depicts the time-series for configuration angle Φ, with the vertical axes representing time and the horizontal axes representing angular displacement of the link. In the right-handed side is represented the history of neural activation using codes (blue for depression, red for saturation and green for intermediate values). This is represented with the symbol *Xc_*. The results show that the body configuration/visual orientation quickly converge to a **fixed-point**. The same happens with the neural network dynamics. When equilibrium is reached (around t = 20) a variety of individual neural states can be observed. Some cells are in depressed state, some in saturated state (in this run, only one), and some take intermediate activation values. The initial state fluctuation corresponds to a transient period which can be interpreted as a "relaxation" of neural state. The potential energy of the network tends to decrease during this period [7]. Different simulation runs would produce different equilibrium states.

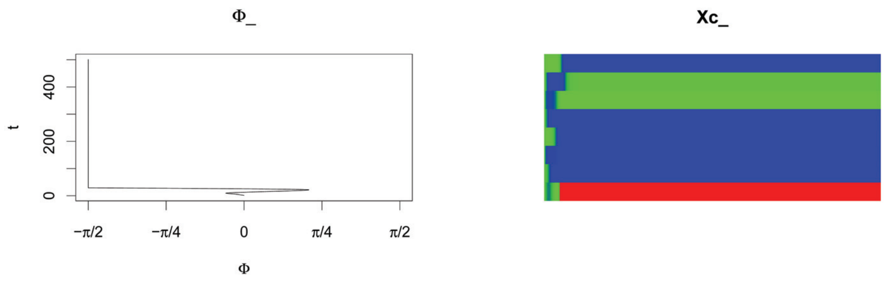


Figure 12. Time-series for the link configuration angle Φ, and control units' activation state X_c over 500 time-steps. Neural controller has 8 units without homeostasis.

Neural Dynamics with Perception but without Homeostasis
In figure 13 we show the behavior of the visual system for two simulation run when a point particle is present in the visual field. The figure includes the time-series for the configuration state and the time-series for the neural state, and also (in the middle) the time-series for the angular difference between link and particle orientations (Δ Φ). The position of the particle is highlighted as a vertical red line in the plot for Φ. The results show that for the first presented simulation run the angular position of the link is close to the position of the particle. This can also be seen by looking at the data plot for Δ Φ which shows that the angular distance to the particle quickly converges to a value close to zero. The second row in the figure 13, shows that this is not always the case. In this second run the link converges to a position far away from the particle position.
The left-handed side of figure 14 shows the time-series of Δ Φ for 10 consecutive simulation runs when the neural controller is configured without homeostasis. The plots confirm the previous observations. Although for an important fraction of simulation runs the angular differences are reduced (6), for several of the simulation runs the link converges to positions far away from the particle. This occurs because the relaxation of the neural state takes the link to certain positions before the visual input is able to significantly influence the neural dynamics.

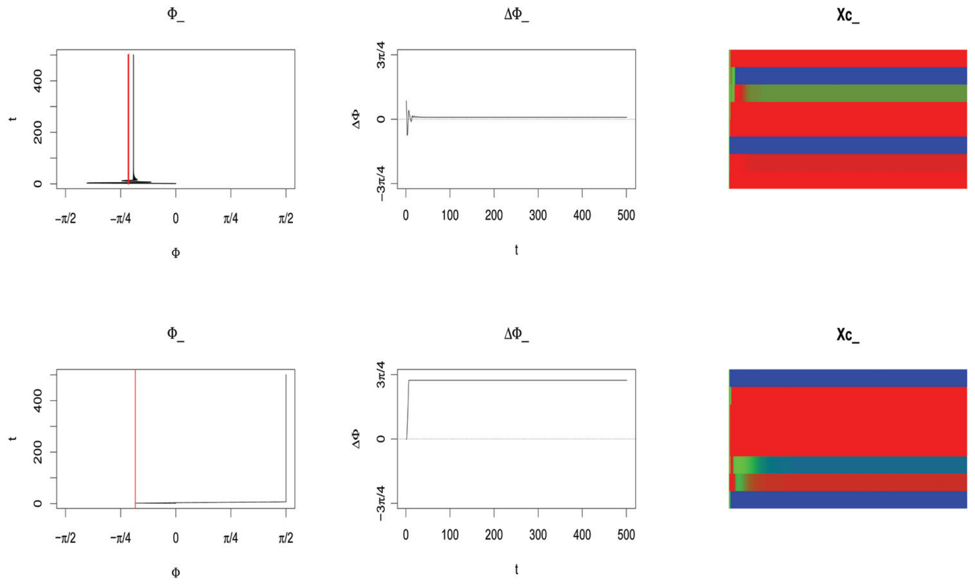


Figure 13. Time-series for the link configuration angle Φ, the difference between link and particle positions Δ Φ, and control units activation state X_c over 50 time-steps. Neural controller has 8 units without homeostasis. The visual stimulus is represented as a vertical red line. Positions were set randomly and were invariant during the simulation time.

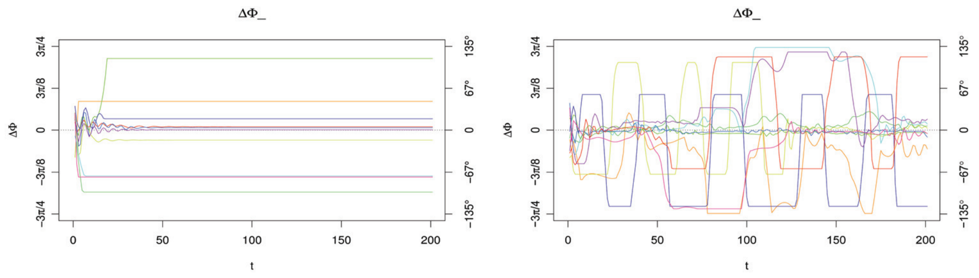


Figure 14. Time-series of Δ Φ over 200 time-steps for 10 consecutive simulation runs. A different particle position was set for each simulation run. (N_c = 8) **left)**: Neural controllers without homeostasis; **right)**: Neural controllers with homeostasis.

Neural Dynamics with Homeostasis

The introduction of an adaptive mechanism in the form of homeostasis completely changes the agent's internal and external dynamics. Figure 15 shows that when homeostasic mechanisms are used ($\frac{\tau_2}{\tau_1}$ = 0.5), the body configuration of the agent exhibits a **non-periodic** or **chaotic** behavior [15]. This means that while the equations for neural dynamics are completely deterministic both the link angle and the neural state seems to move erratically as if a stochastic process is involved. Note that in this trial the point particle is not present yet. Additionally, it can be seen that individual neural units hardly stabilize in particular activation values. This occurs because homeostasis slowly pushes unit's activation to resting value x_0. However, due to unit's interconnections a global equilibrium is never reached [13]. Therefore, the proportion of cells not saturated or depressed at a given time is much less than when homeostasis is not used. Consequently, the activation state of the neural population does not converge to a fixed point. Instead, we can observe complex oscillating patterns of neural activity.

Neural Dynamics with Perception and Homeostasis

In figure 16, we present data plots for two simulation runs with neural controllers working with homeostasis and a point particle is present in two slightly different positions. The results show that in both simulation runs the link orientation converges to a region close to the particle position. This is confirmed in the middle plot of figure 16 where is shown that $\Delta\ \Phi$ converges to values close to zero. Most importantly, the link orientation does not converge to a fixed-point. Instead, it performs small oscillatory movements.

Experimentation with model variations, showed that the use of an antagonistic muscle pair, as apposed to a single muscle, is very useful to give robustness to the model's behavior. Model variations with a single muscle requires parameters to be carefully selected to achieve effective visual fixation behavior. It is interesting to note that while homeostasis tends to move the neural state away from particular regions (e.g. a fixed-point) [14, 13], this does not cause the system to loose track of the particle and increase error. This happens because there is **redundancy** in the neural state–configuration state mapping, with the number of degrees-of-freedom in the former being much higher than the number of degree-of-freedom in the later ($|\mathbf{X}| >> 1$). This explains why in the righthand plots of figure 16 the neural state moves between several states and yet the orientation of the visual axes changes little.

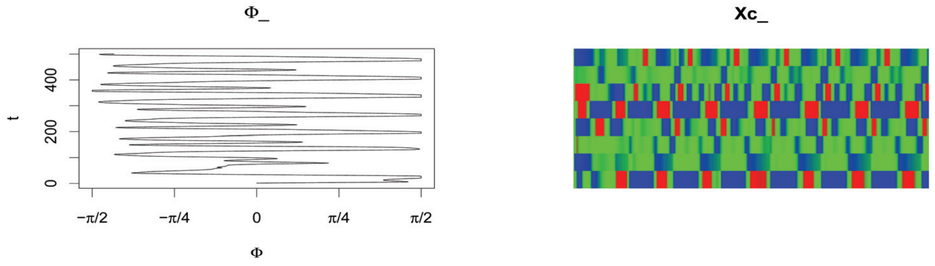


Figure 15. Time-series for the link configuration angle Φ, and control units activation state X_c over 500 time-steps. Neural controller has 8 homeostasic units.

The right-handed side of figure 14 shows time-series of $\Delta\ \Phi$ for 10 consecutive simulation runs when the neural controller is configured with homeostasis. The plot shows that the behavior of the system is qualitatively different from the behavior when no homeostasis is used (left-handed side of figure 14). In a significant proportion of runs the link orientation approximately matches the particle orientation ($\Delta\ \Phi \approx 0$), although we can identify continuous aperiodic oscillations of the link around the particle position. This happens because homeostasis prevents neural state to reach a long-term equilibrium and makes unit's activation to oscillate. On the other hand, the presence of the particle promotes the selection of neural states that corresponds to configuration states of high visual stimulation. Thus, changes at the macro-level are limited while changes in the micro-level can occur. For another important proportion of runs the link orientation moves away from the particle position some proportion of the total number of simulation time-steps (in the case, 200).

Again this happens due to homeostasis, but at particular occasions the neural state moves to regions of the neural states space that do not correspond to a configuration state where the link is aligned with the visual particle. Only in one simulation run (plotted in a blue line) the link is unable to fixate the particle.

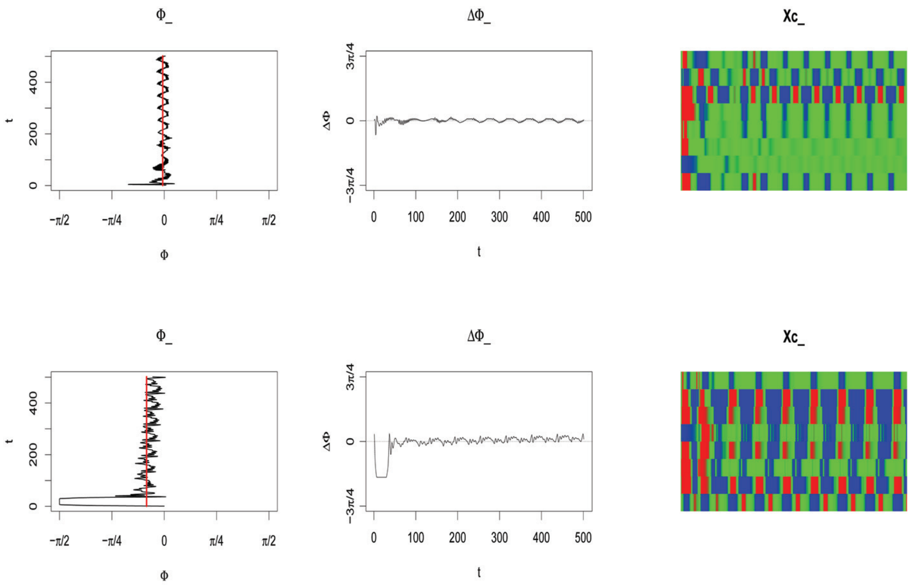


Figure 16 Time-series for the link configuration angle Φ, the difference between link and particle positions Δ Φ and control units activation state X_c over 500 time-steps. Neural controller with 8 homeostasic units. The particle is represented as a vertical red line and is located in random positions.

6. Summary discussion, related work, and conclusions

Research in Recurrent Neural Networks since Hopfield initial contribution [7] as received very much attention specially in exploring its properties as a model for associative memory. Additionally, theoretical and experimental advances in artificial intelligence and robotics research have identified complexity theory as a promising tool to understand how neural agents can self-organize to produce adaptive behavior [11]. Combining RNN models and behavioral research is thus a promising approach to understand cognitive systems and the role played by recurrent connection in the nervous system.

In this article, we make a characterization of cognitive agents that is suggestive of how RNN can control embodied agents, and extend the basic formulation of RNN to include adaptive thresholds to model neural homeostasis. In the proposed approach, adaptive thresholds make neural units to move to a resting activation value although at a slower pace than main activation dynamics. Experimental results show that homeostasis make neural dynamics to produce aperiodic (chaotic) behavior and, for small networks, nearly periodic behavior. We showed that this can be used as a source of behavioral exploration and novelty in embodied neural agents.

Homeostasic mechanisms have been identified in the biological neural networks literature [16], and its behavioral relevance is being explored by other researchers [12]. The emergence of aperiodic behavior in recurrent neural networks as been previously advanced in literature [5], and fits known empirical data about animal and human brain activity [4]. Classical cybernetics has also identified homeostasic behavior as a key characteristic of natural and artificial adaptive/intelligent systems [2]. Experimental methods have been applied to study the role of proprioceptors in neuro-muscular control in animals and humans [6]. The situated AI and ALife community as also identified proprioception as an important mechanism in agent's sensoriomotor coordination [9].

The applicability of the framework and experimental results presented in this chapter are wide. We have provided concrete examples in the domain of muscular control and visual attention, and reported some promising results. Other problem domains in cognitive modeling should also be considered, to see to what extent embodied neural agents and RNN with homeostasis provide a good experimental grounding for research in cognitive modeling.

7. References

R. Abreu and J. P. Simão. Visual attention in embodied RR-ANN without learning. In *Proc. of the EPIA 2007, Workshop on Computational Methods in Bioinformatics and Systems Biology*. 2007.

W. R. Ashby. *Introduction to Cybernetics*. Methuen, London, UK, 1956.

R. D. Beer. Computational and dynamical languages for autonomous agents. In *Mind as motion: explorations in the dynamics of cognition table of contents*, pages 121–147. Bradford Book: The MIT Press, 1996.

W. J. Freeman. *How Brains Make Up Their Minds*. Columbia University Press, 2001.

D. Harter and R. Kozma. Aperiodic dynamics and the self-organization of cognitive maps in autonomous agents. *International Jounral on Artificial Intelligence Tools*, 21(9):955–971, 2005.

Z. Hasan. Role of proprioceptors in neural control. *Current Opinion in Neurobiology*, 2:824–829, 1992.

J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. USA*, 70:2554–2558, 1982.

C. G. Langton. Life at the edge of chaos. In *Artificial Life II, SFI Studies in Sciences of Complexity, vol. X*, pages 41–91. Addison-Wesley, 1991.

M. Maillard, O. Gapenne, L. Hafemeister, and P. Gaussier. Perception as a dynamical sensorio-motor attraction basin. *Advances in Artificial Life (ECAL), LNAI 3630*, pages 37–46, 2005.

H. R. Maturana and F. J. Varela. *Autopoiesis and Cognition: the Realization of the Living*. D. Reidel Publishing, 1980.

S. Nolfi. Behaviour as a complex adaptive system: On the role of selforganization in the development of individual and collective behaviour. *ComplexUs*, 2(3–4):195–203, 2004/2005.

E. A. D. Paolo. Organismically-inspired robotics: Homeostatic adaptation and natural teleology beyond the closed sensorimotor loop. In *Dynamical Systems Approach to Embodiment and Sociality, Advanced Knowledge International, International Series on*

Advanced Intelligence, pages 19–42. Magill, Australia: Advanced Knowledge International Press, 2003.

J. P. Simão. Measuring entropy in embodied neural agents with homeostasic units: A link between complexity and cybernetics. *9th European Conference on Artificial Life*, 2007.

J. P. Simão. Self-perturbation and homeostasis in embodied recurrent neural networks: A meta-model and some explorations with mechanisms for sensorimotor coordination. *International Conference on Artificial Neural Networks*, 2007.

S. H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry and Engineering*. Perseus Books Group, 1994.

G. Turrigiano and S. B. Nelson. Homeostatic plasticity in the developing nervous system. *Nature Reviews Neuroscience*, pages 97–101, 2004.

2

Biological Signals Identification by a Dynamic Recurrent Neural Network: from Oculomotor Neural Integrator to Complex Human Movements and Locomotion

Guy CHERON[a,b], Françoise LEURS[a], Ana BENGOETXEA[a], Ana Maria CEBOLLA[a], Jean-Philippe DRAYE[a], Pablo D'ALCANTARA[a] and Bernard DAN[a,c]
[a]Laboratory of Neurophysiology and Movement Biomechanics,
[b]Laboratory of Electrophysiology, Université de Mons-Hainaut,
[c]Department of Neurology, Hopital Universitaire des Enfants reine Fabiola,
Université Libre de Bruxelles,
Belgium

1. Introduction

The recent advances in the application of artificial neural networks in the biological field have been inspired by the functional organization of real biological structures (Draye et al.,1997a; Anastasio & Gad, 2007). The fascination exerted by the oculomotor system upon both engineers and neuroscientists have played an important role in this issue. In particular, since the definitive evidence of the existence of a neural integrator in the brainstem (Cheron et al., 1986a; Cannon & Robinson, 1987; Robinson, 1989 for a review) performing mathematical integration of the eye velocity into eye position signals, numerous artificial networks have been developed allowing a better understanding of the fundamental question of how the brain control movement. Such bio-mimetic strategy has recently permitted to elaborate different dynamic recurrent neural networks (DRNN) specifically dedicated to the command of humanoid robot (Tani et al., 2008). Hierarchical neural-inspired modules have also been proposed forming cascades of forward dynamics models (Jordan & Rumelhart, 1992; Kawato et al., 1987; Tani, 2003) in which top-down and bottom-up influences allowed generating behavioural primitives. This Chapter describes the main steps performed in the development of our DRNN from the neural integrator models to those applied in the field of human movement control.

2. DRNN simulation of the oculomotor neural integrator

The interest for neural integrator models outpaces the oculomotor field because the processes involved in the maintenance of eye position presents an analogy with the information held in short-term or working memory (Aksay et al., 2001, 2003; McCormick et al., 2003). When the neuron of the neural integrator persistently discharge for encoding the

time-integral of the eye velocity signals during the saccade, this tonic activity may be interpreted as an internal memory of the eye position in space (Godaux & Cheron, 1996; Chan & Galiana, 2005). The analogy with working memory was thus easily accomplished (McCormick, 2001).
The first neural network approach of the neural integrator was made by Cannon et al. (1983, 1985). Their hard-wired model in which the synaptic weights were explicitly specified can integrate a push-pull input signal without integrating the background rates and has the appealing property that localized artificial lesions produced a decrease in the time constant of the whole network. Later, Anastasio & Robinson (1991) proposed the first learning model for the neural integrator. In this context, we have upgraded the Anastasio-Robinson model in order to work in the continuous-time domain (in opposition to the discrete-time domain). Additionally, we improved the biologically plausible features by (1) the introduction of a strong constraint on the synaptic weight and (2) the introduction of an artificial distance between the neurons by generating delays proportional to the proximity.

2.1 The basic DRNN models

The basic model is a dynamic recurrent neural network governed by the following equations:

$$\mathrm{T_i}\,\frac{\mathrm{dy_i}}{\mathrm{dt}} = -\mathrm{y_i} + F(x_i) + \mathrm{I_i} \tag{1}$$

where *F*(α) is the squashing function *F*(α) = *(1+e -a)*$^{-1}$, y_i is the state or activation level of unit *i*, *It* is an external input (or bias), and x_i is given by:

$$\mathrm{x_i} = \sum_{\mathrm{j}} \mathrm{w_{ij}}\ \mathrm{y_j} \tag{2}$$

which is the propagation equation of the network (x_i is called the total or effective input of the neuron, W_{ij} is the synaptic weight between units *i* and *j*). The time constants T_i will act like a relaxation process. The correction of the time constants will be included in the learning process in order to increase the dynamical features of the model. Introduction of T_i allows more complex frequential behaviour, improves the non-linearity effect of the sigmoid function and the memory effect of time delays (Draye et al., 1996; 1997a).

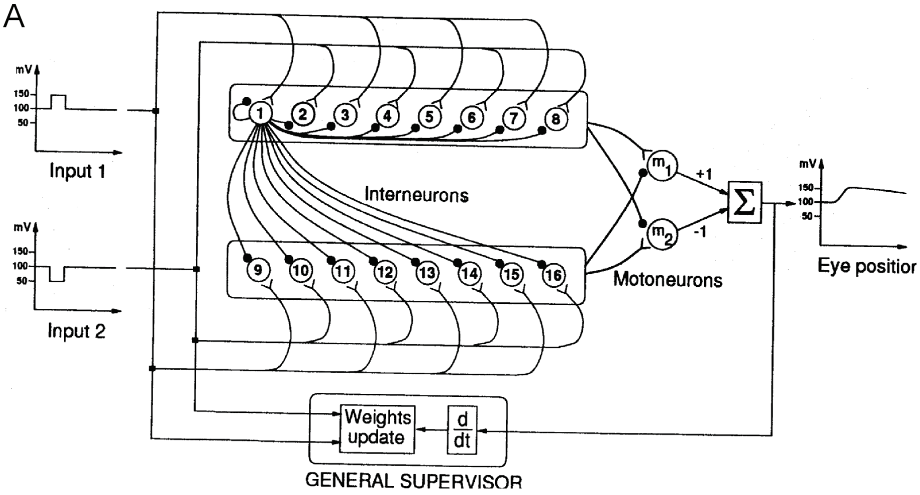
2.2 Fixed-sign connection weights

Traditionally, artificial network represent the synaptic weight by a real number. This number is modified by the learning process (often a gradient-descent kind of minimization) which frequently leads to a sign change at different location. This sign change is in conflict with biological reality and Dale's Principle. Thus we fixed the sign of all connections by the introduction of a variable s_{ij} associated with every weight unit w_{ij}. The variable s_{ij} take their value in the set {-1,0,+1}, and in the classical network propagation equation (2), the weights w_{ij} are replaced by the equation:

$$s_{ij} \cdot \left| w_{ij} \right| \tag{3}$$

The architecture of the network is consistent with the neuroanatomy of the brainstem circuitry of the neural integrator devoted to saccade and vestibulo-ocular systems for the horizontal movements. The figure 1 illustrates the neural integrator DRNN comprising a fully connected hidden layer of 16 inhibitory units, two output units representing the motoneurons of the median and lateral rectus muscle of the left eye and two afferents inputs from the horizontal canal (for the vestibulo-ocular reflex) or from the eye-velocity saccade generator.

A



B

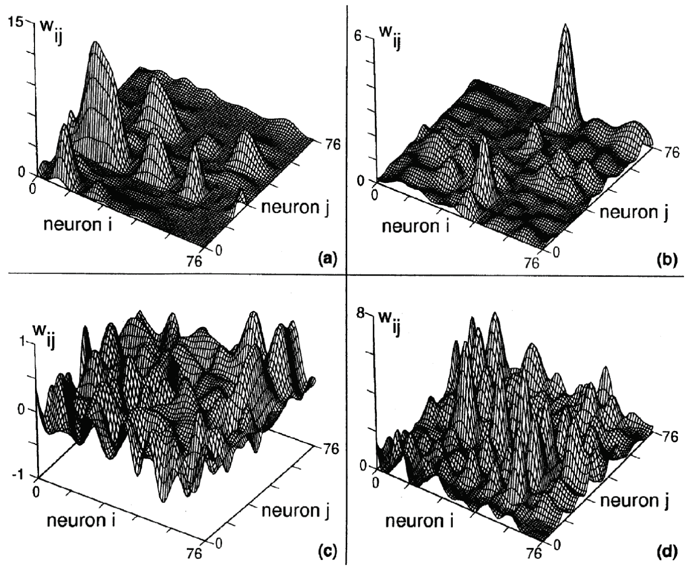


Figure 1. Architecture of the DRNN dedicated to the neural integrator. (A) The 16 interneurons of the hidden layer are divided into two groups of 8 and are fully connected

with inhibitory connections (only the connections from interneuron 1 are depicted). Each interneuron is connected to both motoneurons (output of the network) with a connection whose sign is indicated in the figure. The signals inputs are represented by pulses representing eye-velocity commands of opposite signs. (B) 3D surface plots of the weights distribution. The 16 X 16 weights surface were treated with cubic splines for better visualization. Values of the weights are plotted versus indexes *i* and *j*. Even if the lateral layer has inhibitory connections (except for **c**), the weights are plotted as positives values. **a, b** Two clustered structures of the weights distribution. **c**, The weights distribution of the Arnold-Robinson network trained with the general supervisor without any constraints on the weight signs. **d,** The weights distribution in which each interneuron has its own muscle. (*modified from Draye et al. 1997a Biol Cybern)*

2.3 Artificial distance between neurons

Classically, the distances between an artificial neuron labelled 6 and two other neurons labelled 7 and 16 are the same. In order to introduce a real notion of distance in our device (between digits that are memorized in computer memory) we generate delays between these units. The delay between neurons N_i and N_j is defined in order to keep the proportionality to the difference of index $|i-j|$. By this way, the information is *artificially delayed* during its propagation in the network. It will take $|i-j|$ time steps for the information from neuron N_i to reach the neuron N_j.

2.4 Numerical discretization of the continuous-time model

The discrete-time model with a step Δt was defined as:

$$y_i(t+\Delta t) = (1-\frac{\Delta t}{T})\cdot y_i(t) + \frac{\Delta t}{T}\cdot F(x_i(t)) \tag{4}$$

where

$$x_i(t+\Delta t) = \sum_j w_{ij}\cdot y_j(t) \tag{5}$$

where we assume that the terms $I_i(t)$[see (1)] has been replaced by adaptative weights w_{0i} connected to a fixed input which is set to 1. The discretized equation (5) becomes:

$$x_i(t+\Delta t) = \sum_j \underbrace{(s_{ji}\cdot|w_{ji}|)}_{Fixedsign}\cdot y_j \underbrace{\left[t-|i-j|\cdot\Delta t^*\right]}_{Artificialdis\tan ce} \tag{6}$$

For the learning we introduced a *general supervisor* responsible for the modifications of the network weights w_{ij}. In this particular case of the oculomotor integrator simulation, the sign of the connections must be take into account and strictly conserved. This general supervisor continuously computes the amount of the positional deviation (corresponding to the retinal slip) and uses it as an error signal to minimize. The Levenberg-Marquardt minimization technique has been used. The training of the network was done with pulse signals of 50 ms of duration. After this phase the network produces a position signal compatible with the physiological behavior of the oculomotor neural integrator presenting a time constant of 20s.

2.5 Emergence of clusters

The DRNN was trained a great number of times and each time a clustered structure of the type illustrated in the 3D weights distribution map has emerged (Fig. 1B). A cluster is a region of large weights between a particular group of neurons of index i centred on i^* and another group of neurons j centred on j^*, where the point $[i^*, j^*]$ is considered as the "centre" of the cluster. The interpretation of a cluster is the following: if the connection weight w_{ij} between two hidden units N_i and N_j is high, *the probability* is high that the connection weight $w_{j(i+1)}$ between one of the neighbours of the source neuron N_{i+1} and N_j is *large.* The same conclusion can be made for the weight $w_{(j+i)I}$ between N_i and one of the neighbours of the destination neuron N_{j+1}. The mathematical description of the cluster was developed in Draye et al., 1997a. The process of emergence of such clusters during the training phase remains unknown. However, we have studied the conditions for this emergence. Clusters appeared when (1) the sign of the connections was fixed, (2) a lateral inhibitory layer of interneurons, (3) the introduction of an artificial distance between these units and (4) a convergence of information from the hidden layer to the motoneurons. Indeed, when there are no constraints on the weights sign and no delay between the units, there is no clustering structure in the weight distribution (Fig. 1B,c). When we suppressed the convergence of the hidden units on the 2 motoneurons (each interneuron was in this case linked to a muscle), organization in clusters did not appeared anymore (Fig. 1B,d). As we have found that the behaviour (represented by their phase value when sinusoidal input were used) of the units participating to a same cluster was the same (e.g. units presenting eye position sensitivity) (Draye et al., 1997a), an interesting analogy between the artificial DRNN integrator and the electrophysiological recordings can be made. For example, clusters of position neurons have been found in the neural integrator of the cat (Delgado-Garcia et al., 1989; Escudero et al., 1992, 1996; Godaux & Cheron, 1996). We can thus conclude that emergence of clusters in a DRNN performing a well-defined mathematical task (here a temporal integration) is due to computational constraints with a restricted space of solutions. This also suggests that information processing constraints could be a plausible factor inducing the emergence of iterated patterns in biological neural networks.

3. The DRNN application in the field of human movement control

3.1 Introduction

In human, the electromyographic activity (EMG) is the only non-invasively accessible signal directly related to the final command of movement. EMG signal, though not ideal, is a reasonable reflection of the firing rate of a motoneuronal pool (Soechting & Flanders, 1997), and the analysis of rectified EMG envelopes of multiple muscles may reveal the basic motor coordination dynamics (Scholz & Kelso, 1990; Cheron et al., 1996; Bengoetxea et al., 2008).

Our DRNN approach has been firstly applied to the problem of identification of the relationship between EMG signals of the shoulder muscles and the corresponding kinematics of the arm. This identification task is quite complex because the state variables of the system are unknown and identification has to be carried out using only input–output data. Moreover, the EMG–motion relationship identification task is highly nonlinear. This latter fact complicates the task because it is well known that even in the linear case where the state variables are unknown, a unique parameterization of the system no longer exists (Kalman et al., 1969). The success of nonlinear identification techniques therefore strongly depends upon specific parameterizations used (Wang, 1993).

3.2 Methodological adaptations

The network defined by (1) can be trained using different learning algorithms; the learning algorithm tunes the free parameters to minimize an error measure which is computed as the temporal integration between the real curve and the learned curve. The most famous ones are the real-time recurrent learning algorithm presented by Williams & Zipser (1989) and the time-dependent recurrent backpropagation algorithm derived by Pearlmutter (1989, 1995). The reader can find more details about the learning algorithm in Pearlmutter (1995), Draye et al. (1996, 1997a,c).

In order to make the temporal behaviour of the network explicit, an error function is defined as:

$$E = \int_{t_0}^{t_1} q(\mathbf{y}(t),t)\,dt \tag{7}$$

where t_0 and t_1 give the time interval during which the correction process occurs. The function $q(\ y(t),\ t)$ is the cost function at time t which depends on the vector of the neurone activations y and on time. We then introduce new variables p_i (called adjoint variables) that will be determined by the following system of differential equations:

$$\frac{dp_i}{dt} = \frac{1}{T_i}\, p_i - e_i - \sum_j \frac{1}{T_i}\, w_{ij}\, F'(x_j)\, p_j \tag{8}$$

with boundary conditions $p_i\ (t_1)$=0. After the introduction of these new variables, we can derive the learning equations:

$$\frac{\delta E}{\delta w_{ij}} = \frac{1}{T_i} \int_{t_0}^{t_1} y_i\, F'(x_j)\, p_j\, dt \tag{9}$$

$$\frac{\delta E}{\delta T_i} = \frac{1}{T_i} \int_{t_0}^{t_1} p_i\, \frac{dy_i}{dt} \tag{10}$$

The training is supervised; involving learning rule adaptations of synaptic weights and time constant of each unit (see for more details, Draye et al., 1996). Due to the integration of the system of (8) backward through time, this algorithm is sometimes called ‘backpropagation through time’. In order to reduce the time of the learning process, the acceleration method of Silva & Almeida, (1990) was used, where each weight and time constant has its own adaptative learning rate.

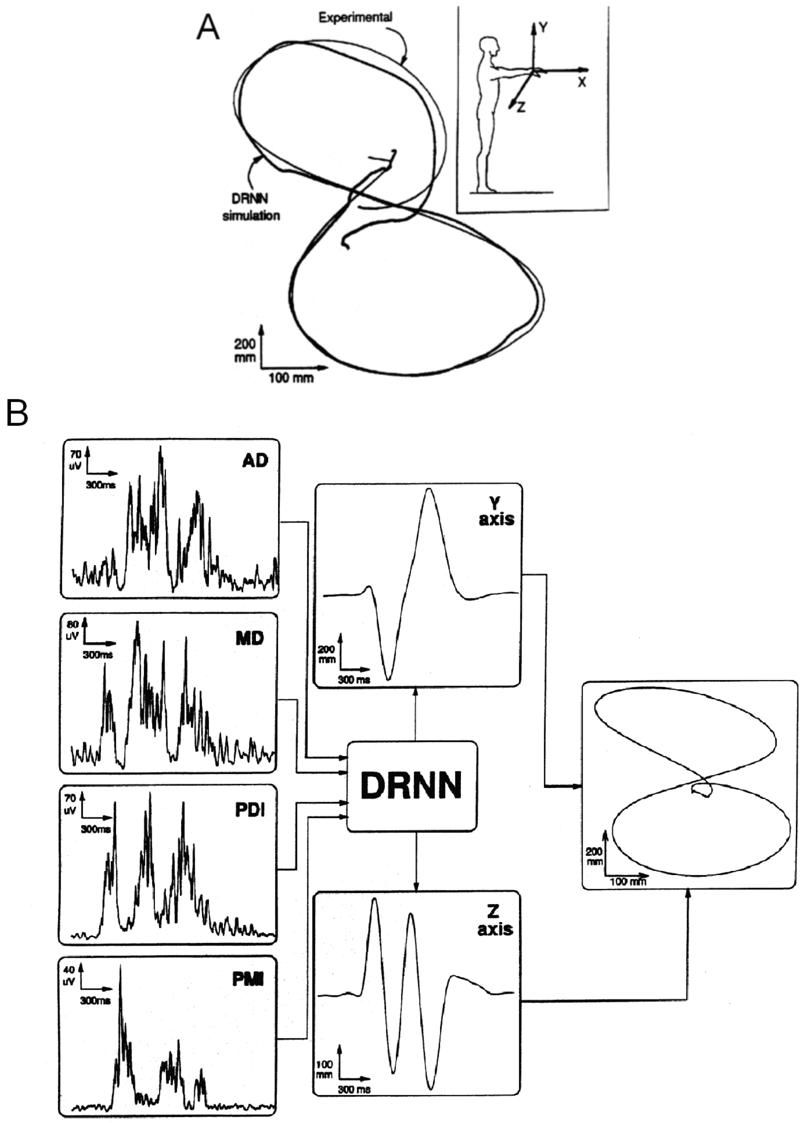


Figure 2. Input-output organization of the DRNN. In this configuration the inputs consist of seven full-wave rectified EMG signals (four of them are depicted). The outputs are the Y and Z coordinates of the index marker during the drawing of the figure eight. The position of the subject and the reference axis are shown on the upper right side. This figurative movement is characterized by two main components in the vertical direction (Y axis-down and up) and by four main components in the horizontal direction (Z axis-right, left, right, and left). In the upper-right inset, superimposition of the experimental trajectory recorded by the ELITE system (thin line) and the simulated curve generated by the DRNN (thick lines). (*Adapted from Cheron et al., 1996 IEEE*)

3.3 EMG and movements recordings

In all experimental situations explored by our group, the DRNN was trained to reproduce the movement performed by the subject in response to the EMG signals as depicted in Fig. 2. In a first set of studies (Cheron et al., 1996; Draye et al., 2002; Bengoetxea et al., 2005), the subjects were asked to draw as fast as possible figures 'eight' with the right extended arm in free space (the initial directions of the movements were up–right, up–left, down–left, and down–right, in that order). We have to note that in this case the kinematics data are given by the position signals of the index finger (the outputs of the DRNN were the vertical and the horizontal position of the index).

In a second set of studies (Cheron et al., 2007) the subjects were asked to perform 'as fast as possible' flexion movements of the elbow in the vertical plane. In this case the angular acceleration of the elbow was used as the output. In the third set of experiments the subjects locomotion was recorded (Cheron et al., 2003; Leurs et al., 2005) and the DRNN presented 3 different output signals corresponding to the kinematics (elevation angle) of the thigh, shank and foot.

These different movements were recorded and analyzed using the optoelectronic ELITE system including two to six TV cameras working at a sampling rate of 100 Hz (BTS, Milano, Italy). Surface EMG patterns were recorded using pairs of silver–silver-chloride surface electrode and measured using telemetry. Raw EMG signals (differential detection) were amplified (1000 times) and band-pass filtered (10–2000 Hz). After this, the EMGs were digitized at 2 kHz, full-wave rectified and smoothed by means of a third-order averaging filter with a time constant of 20 ms. The following muscles were recorded during the figure-eight movement and the elbow flexion: posterior deltoid external and internal (PDE and PDI), anterior deltoid (AD), median deltoid (MD), pectoralis major superior and inferior (PMS and PMI), latissimus dorsi (LD), biceps and triceps brachii. For the locomotion: rectus femoris (RF), vastus lateralis (VL), biceps femoris (BF), tibialis anterior (TA), gastrocnemius lateral (GL), soleus (SOL). The basic mapping between EMG signals input and kinematics output is illustrated in the figure 2 during the execution of the figure eight movement. The superimposition of the real (experimental) and simulated movement well illustrated the DRNN performance.

3.4 From DRNN performance to biological plausibility

For each type of the movement studied, the DRNN has successfully learned the task and was able to reproduce correct output signals with the same type of unlearned EMG signals as input. The learning performance was firstly examined on-line by inspection of the error curve as those illustrated in the case of walking movement (Fig. 3A) (Cheron et al., 2003). Successful learning was commonly ascertained on the basis of the comparison between the DRNN output and the actual output (provided by experimental data). Figure 3 illustrates the superimposition of these data (Fig. 3B-D) when the training has reached an error value of 0.001. The learning process (performed in this case by means of 35 fully connected units) was carried out for 5000 iterations which takes about 5 min on a Intel Core2 at 2 GHz

In order to test the physiological plausibility of the DRNN identification, the basic idea was to compare the angular directional change induced by artificial EMG suppression or potentiation of a single muscle with the physiological knowledge of the pulling direction of the muscle. This method is illustrated for the figure eight movement where a small artificial lesion was performed on the first burst of the PMI muscle. In this case, the last part of this

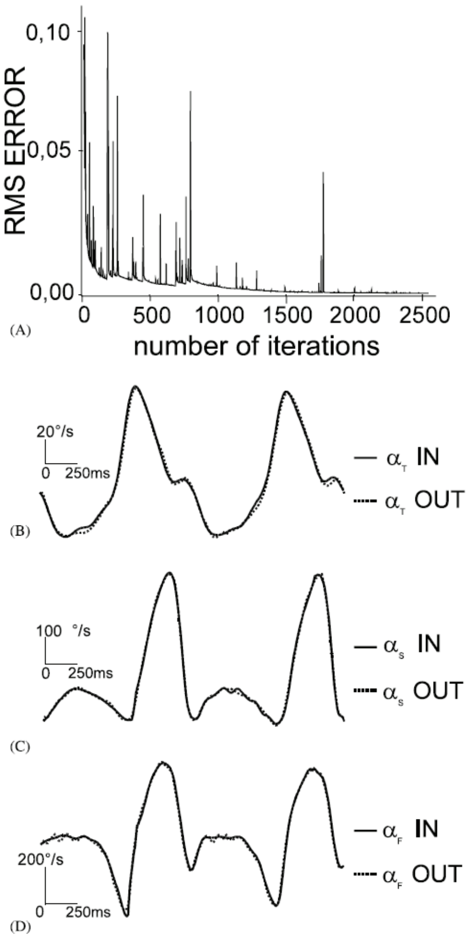


Figure. 3. Assessment of successful learning. (A) Error curve of one learning trial reaching an error value of 0.001 after 5000 iterations. (B, C and D) Superimposition of experimental (continuous line) and DRNN (pointed line) output signals when training reaches an error value of 0.001. (*with permission of Elsevier, Cheron et al. 2003 J Neurosci Meth*)

burst (Fig. 4 a) has been cut off during 50 ms. This altered signal, and the six other unaltered EMG signals are fed to the DRNN previously trained with the normal one. The resulting trajectory is compared to the normal one (Fig. 4 b). This shows that in this case the arm is not able to reach the lower part of the normal trajectory, which is compatible with the physiological action of the PMI acting as extensor-flexor of the shoulder. The quantification of these effects was performed by the computation of the error vector of the arm velocity (Fig. 4 c). For the majority of these lesion experiments performed in the EMG signals of different muscles the direction of the error vector coincided with the preferential field of activation of the corresponding muscle (Cheron et al., 1996). In this context the treatment of the EMG signals by means of different biological filters (Hill-type muscle model) including tension-length and force-velocity relationships of muscle-tendon actuators can provide a good approximation of muscle force and facilitate the DRNN learning (Draye et al., 1997b). However, contamination of the original neuronal input (raw EMG) by output kinematics-related data (muscle length changes) would bias the spontaneous emergence of multiple attractor states linked to the basic input–output mapping.

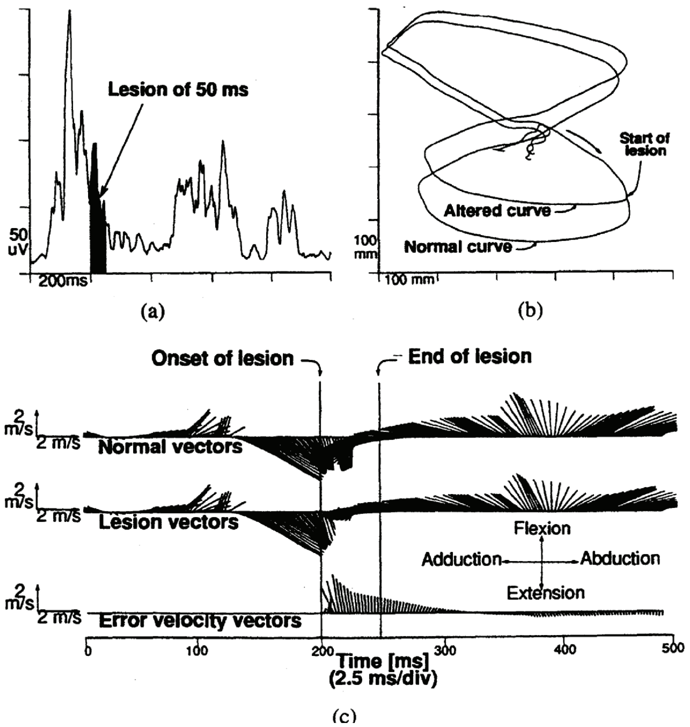


Figure 4. Artificial lesion of the EMG input in order to test the physiological plausibility of the DRNN. (a) Small lesion of 50 ms of duration on the EMG signal recorded on the PMI. (b) Superimposition of the normal and altered trajectories. (c) Velocity vectors of the normal and altered trajectories. The error velocity vectors are obtained by the difference between the preceding ones. (*From Cheron et al.,1996 IEEE*).

The physiological plausibility of our DRNN methods has recently be tested for the identification of the triphasic EMG patterns sub serving the execution of ballistic movements (Cheron & Godaux, 1986b). This pattern comprise a first burst of activity in agonist muscle (AG1) followed by a burst in the antagonist muscle (ANT) and again by second burst in the agonist (AG2). Figure 5 shows that the DRNN is able to perfectly reproduce the acceleration profile of the ballistic movements. The physiological plausibility was tested on all the networks that reached an error level below 0.001 by selectively increasing the amplitude of each burst of the triphasic pattern and evaluating the effects on the simulated accelerating profile. Nineteen percent of these simulations reproduced the physiological action classically attributed to the 3 EMG bursts: AG1 increase showed an increase of the first accelerating pulse, ANT an increase of the braking pulse and AG2 an increase of the clamping pulse. Another important result was that the DRNN also recognized the physiological function of the time interval between AG1 and ANT, reproducing the linear relationship between this time interval and movement amplitude (Fig. 6). Experimental (Cheron & Godaux, 1986b) and clinical evidence from cerebellar patients (Manto et al., 1995) demonstrated that this time interval is one of the main parameters underlying hypermetria, the other parameter being impaired control of ANT amplitude when inertia is increased.

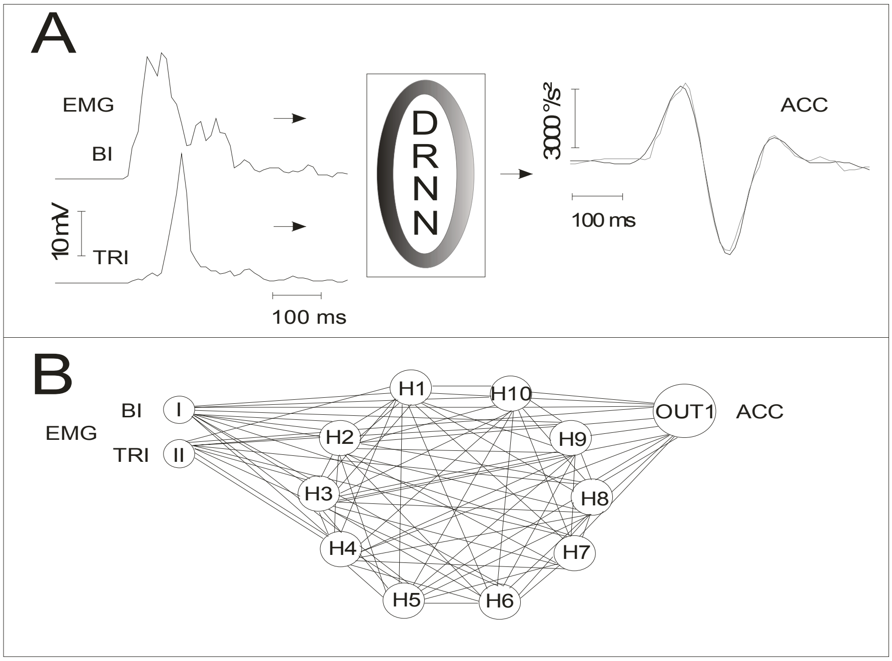


Figure 5. A, Input-output configuration of the DRNN, symbolised by the ring in the central box, with the triphasic EMG pattern as the input and the angular acceleration of the elbow (ACC) used as output. The experimental (grey) and simulated (black) acceleration curves are superimposed. B, DRNN fully connected architecture is represented in case of only 13 artificial neurons (10 hidden neurons, H1-H10; 2 input neurons, I and II, and one output neuron, OUT1).(*Modified from Cheron et al., 2007 Neurosci. Let)*

If the biomechanical knowledge about effect of artificial modifications of the EMG profiles is easily accessible for mono-articular muscles, it is less straightforward for the pluri-articular muscles. In the latter, the muscle force can be involved in a force regulation process for which the directional action is not directly defined by the pulling direction of the muscle. Moreover, dynamical coupling between the three joint segments can be implicated in the evoked movement. For example, in the figure 7 we illustrates the effect of SOL and TA artificial potentiation applied throughout the walking sequence on the sagittal lower limb kinogram over two steps. Whereas the former results in digitigrade gait (explained by the pulling action of SOL) with increased knee flexion (explained by a coupling action) more marked during the swing phase, the latter results in increased ankle dorsiflexion (walking on the heel explained by the pulling action of TA) and knee hyperextension (coupling action) more marked during the stance phase. The implications of such complex dynamical simulations of biomechanics and muscle coordination in human walking have been recently revisited by Zajac et al. (2003).

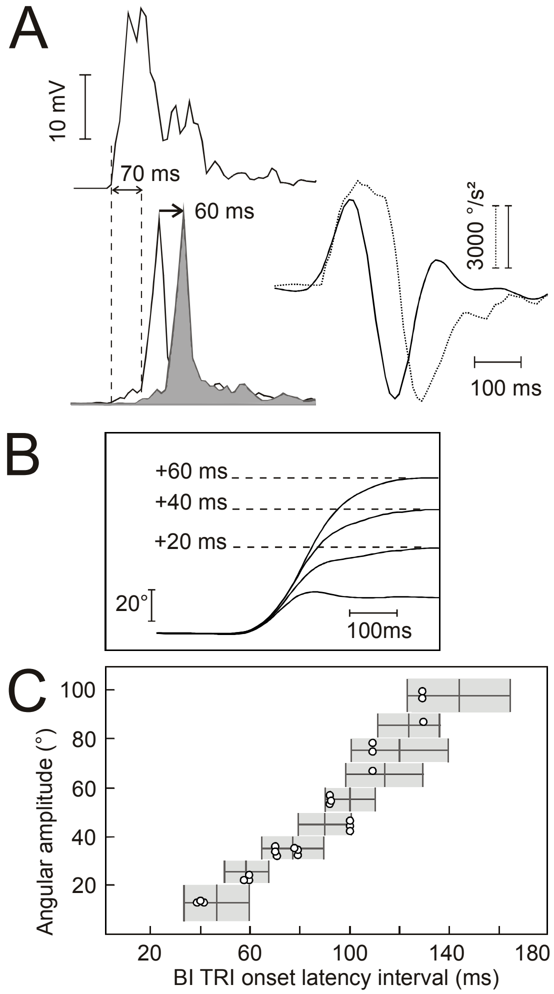


Figure 6. Simulation of AG1-ANT time interval increase on movement amplitude. A, example of a time shift of ANT burst (delayed from 60 ms, grey shading of ANT burst superimposed to the experimental pattern). In the left side, the corresponding ACC curves are superimposed (simulated curve in pointed line and experimental curve in continuous line). B, progressive increase of angular amplitude when the AG1-ANT interval is increased from 20 ms. C, AG1-ANT time interval and the related movement amplitude. Superimposition of the experimental relationship (the mean and SD are represented by the centre and the borders of the grey area, respectively) and the DRNN simulated data (open circles). (*From Cheron et al., 2007 Neurosci. Let)*

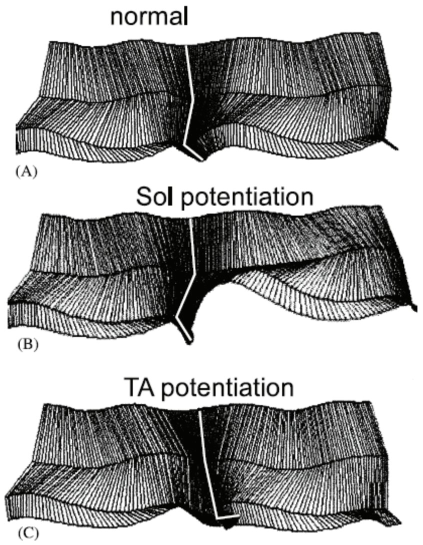


Figure 7. (A–C) Sagittal stick diagrams of the lower limb kinematics obtained after DRNN learning of normal locomotion (A) and after artificial EMG potentiation of SOL (B) and TA (C) muscles. (*From Cheron et al., 2003 J Neurosc Meth, copyright Elsevier*)

4. DRNN with modular architecture

4.1 Introduction of position and inertial subnetworks

We modified the structure of our neural network in order to cope with one of the main drawbacks of trained neural networks: the "solution" appears as a black box from which it is difficult to retrieve any information. For this we modified the network architecture in order to include two distinct sets of output neurons: one set related to posture and the other one related to inertia. The postural–output neurons were trained to produce position reference signals, i.e., postural-related data. The inertial–output neurons were trained to produce inertial related data: acceleration signals. The input neurons remain unchanged: they feed the network with the EMG signals. Postural output neurons are fed by 20 fully connected neurons which form a subnetwork that will be called "the postural subnetwork" (its neurons are labeled "1" to "20" in Fig. 2). Similarly, inertial–output neurons are fed by another set of 20 fully connected neurons which form the "inertial subnetwork" (its neurons are labeled "21" to "40" on Fig. 8).

We imposed a communication channel between both "sub-networks" This consists of 40 interconnections between corresponding neural units. In other words, the learning algorithm is allowed to adapt the interconnection weights between neurons 1 and 21 (and accordingly between 21 and 1), 2 and 22 (22 and 2), . . ., 19 and 39 (39 and 19), and finally 20 and 40 (40 and 20). These weights are shown as dashed lines in Fig. 8. In this configuration, the entire network has 1172 free parameters (interconnection weights and time constants). The modified architecture as presented above is trained as a single homogeneous network which includes seven input neurons (EMGs) and four output neurons (postural Y and Z, inertial Y and Z). The error evaluation criterion is the same for all four output neurons.

These four identical error signals are used by the TDRBP algorithm to adapt all the free parameters of the network. This type of modular DRNN has been used for the figure eight movement and for the straightening-up movement. The architecture of the network in the latter case is slightly different since the network exhibits eight EMG input signals (versus seven) and three output neurons (versus two). Each subnetwork still consists of 20 neurons. The postural subnetwork generates the angular position signals of the three joints, whereas the inertial subnetwork provides their angular acceleration signals. This network has 1192 free parameters.

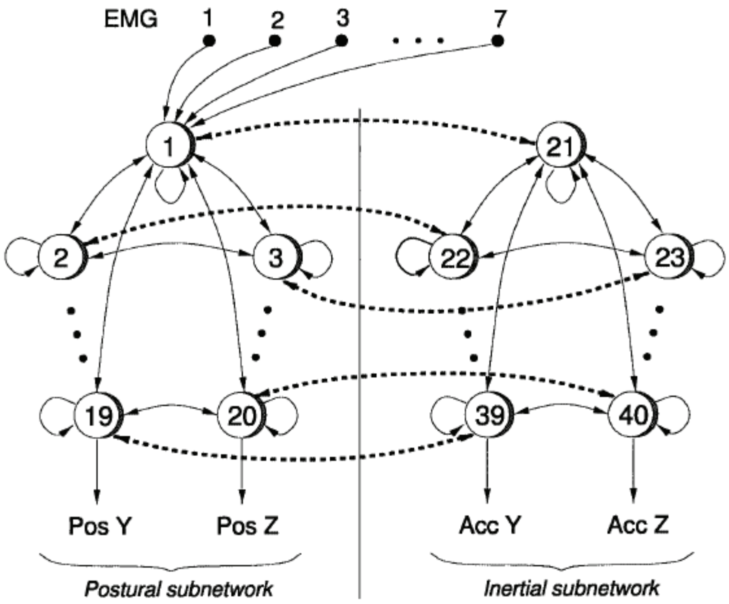


Figure 8. Modular neural architecture with two subnetworks: the first one is related to posture and generates position signals, the second one is related to inertia and generates acceleration signals. The network shows the simulation network for the figure-eight movement case (seven EMG input signals and two kinematics output signals). Note that only some representative connections are depicted (e.g., the input connections are only depicted from input neurons 1 to 7 to neuron 1; the same connections exist between all the input neurons and neurons 2–40). The dashed lines show the interconnections corresponding to the communication channels between both subnetworks. Pos, position; Acc, acceleration. (*From Draye et al., 2002 Biol. Cyber*)

In order to quantify the efficiency of the 40-neuron modular architecture, we compared its performance with the same network without communication channels (in this case, the error signals measured on the postural output signals only affect the postural subnet, and the error signals measured on the inertial output only affect the inertial subnet). We trained this network for 20 times (each new training process started with a new initial random weight distribution). The corresponding error curves were averaged over these 20 learning phases. We have found that the modular network (40-neuron with two communicating subnetworks) gives much better results than the two independent 20 neurons (Fig. 9).

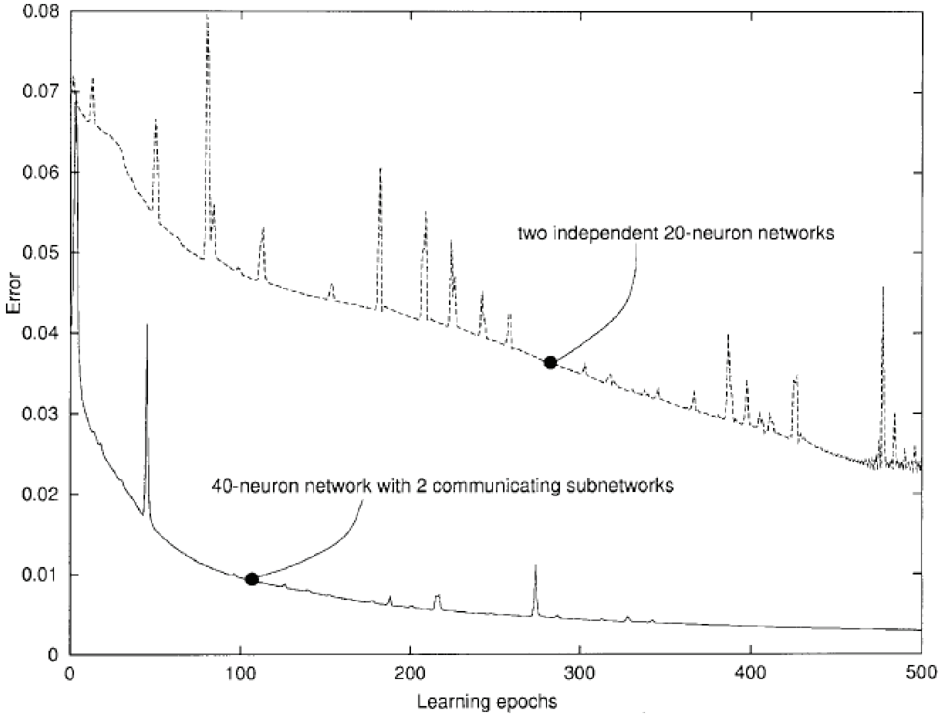


Figure 9. Averaged error curve for the 40-neuron modular architecture (solid line) compared to the cumulative averaged error curves of two independently trained 20-neuron networks (dashed line).

4.2. A Gaussian factor for the artificial distance

In order to improve the biologically plausible features and as previously explained, we decided to introduce a notion of distance in the network using a Gaussian factor *aG* that modulates all the interconnection weights (from neuron i to j) is replaced by aGw_{ij} where *aG* is computed using the absolute value of the difference between the indexes *i* and *j* (see Draye et al., 2002 for more details). The impact of a particular neuron is greater for neurons with close indexes. We assumed that the hidden neurons were distributed along a circumference; this means that the last hidden neuron (neuron 18) has two nearest neighbors: neurons 17 and 1. Thus, the largest distance between two neurons is 9. In contrast to the artificial distance presented in the simulation of the neural integrator (Draye et al., 1997a) which is based on temporal concepts, the distance presented for the identification of EMGs-movement simulation was based on spatial aspects.

4.2 Emergence of a reduced modular architecture

Following the introduction of an artificial distance a reduced modular architecture has emerged. When only the free parameters that were set to a nonzero value by the learning algorithm have been taken into account, a reduced architecture appeared. It has 524 free parameters (compared with the original configuration of 1172 parameters). Moreover, after several attempts, we found out that the minimal architecture to solve the first identification task (figure–eight movement) is composed of 20 neurons (two ten-neuron subnetworks). This model includes 244 free parameters. The same network (with eight input neurons and three output units) can solve the second task (standing-up movement); it consists of 274 free parameters. In summary, we showed that with the minimal architecture, the analysis of the network is much easier; there was less redundancy in the network and the number of free parameters has decreased drastically (a factor of about 5!).

4.3 Emergence of inhibitory feedback connections

We have demonstrated that the reduced architecture always exhibited strong feedback inhibitory interconnections between the two sub-network units. The signs of these feedback connection weights have been selected by the learning algorithm. We also noticed that the learning algorithm was more efficient when we initialized the feedback connection weights with negative values prior to the training phase. The emergence of a reduced lateral inhibitory output layer between both subnetworks is consistent with the model proposed by Cannon & Robinson (1983, 1985) to simulate the neural integrator of the human oculomotor system. It is interesting to point out the fact that in our case, the lateral inhibitory connections appeared during the learning process (and was not forced analytically as in Cannon & Robinson (1983, 1985).

4.4 Time-constant and tonic-phasic behaviour of the hidden neurons

The minimal aspect of the reduced DRNN organization allowed studying the distribution of the time constant and the temporal evolutions of the hidden neurons' output. A clear distinction has appeared between the time constants of the position subnetwork units and the inertial subnetwork ones.

This bimodal distribution of the time constants is depicted in Fig. 10 A,B for the figure eight and the straightening-up movement, respectively. These values were averaged over ten different trained networks for each task. The difference between the values of the subnetwork's time constants proves that the individual role of each subnetwork (postural and inertial) has been clearly identified. A higher time-constant mean value in the postural subnetwork is compatible with the task assigned to this subnetwork. In the same line of evidence, we noticed that some the hidden neurons exhibited a phasic behavior (Fig. 10C) while others present a tonic behavior (Fig. 10D). This result is in accordance with the existence of tonic and phasic neurons found in different brain nuclei (such as in the oculomotor system). For example, in the paramedian reticular formation, the eye velocity signal (excitatory or inhibitory burst neurons, Henn et al., 1982) are phasic neurons whereas a group of tonic neurons encode a pure position-related signal in the prepositus hypoglossi nucleus (Escudero & Delgado-Garcia, 1988; Escudero et al., 1992; Godaux & Cheron, 1996). The coding of movement parameters of the premotoneuronal cells recorded in the motor cortex of behaving animals (Fetz et al. 1989) and in the red nucleus revealed that they exhibit at least a pure tonic, a phasic-tonic, or a pure phasic discharge pattern during a ramp-and-hold movement (Fetz, 1992). It is interesting to note that a comparable separation of phasic and tonic drives was obtained by principal component analysis of raw EMG signals (Flanders, 1991). Although, the phasic and tonic EMG patterns are mixed in the raw EMG signals, they might be implemented by distinct neural subnetworks as suggested by Pelligrini

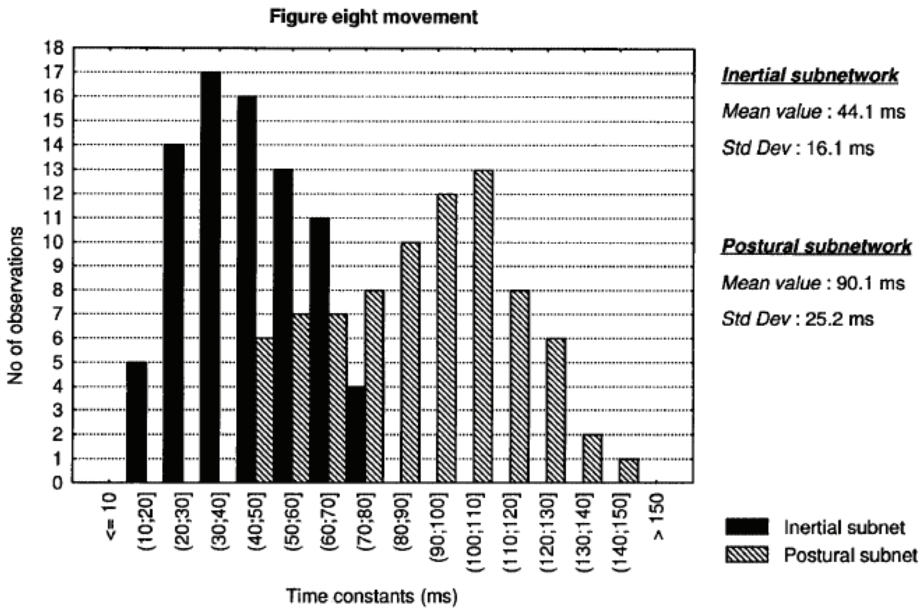
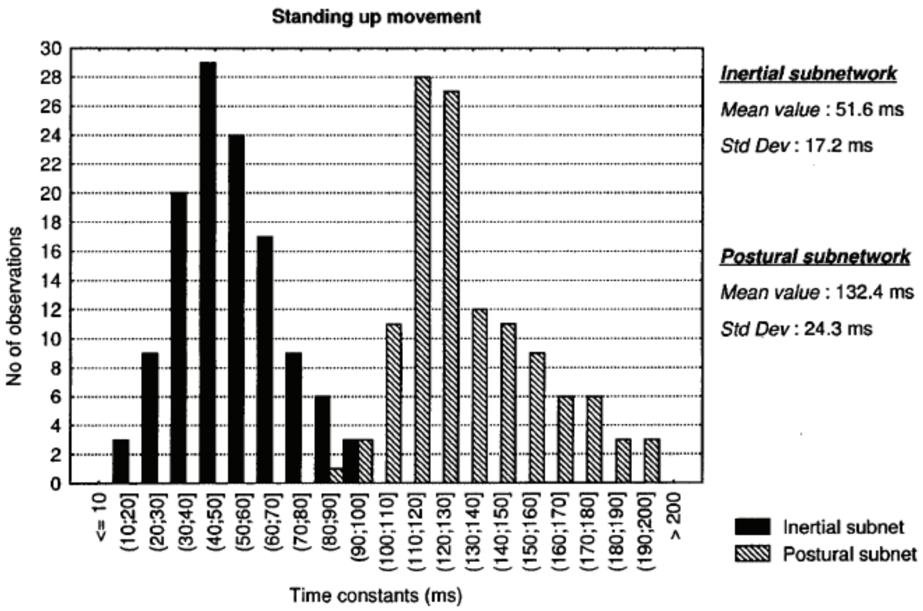


Figure eight movement
No of observations
Time constants (ms)
<= 10
(10;20]
(20;30]
(30;40]
(40;50]
(50;60]
(60;70]
(70;80]
(80;90]
(90;100]
(100;110]
(110;120]
(120;130]
(130;140]
(140;150]
> 150
Inertial subnetwork
Mean value : 44.1 ms
Std Dev : 16.1 ms
Postural subnetwork
Mean value : 90.1 ms
Std Dev : 25.2 ms
Inertial subnet
Postural subnet


Standing up movement
No of observations
Time constants (ms)
<= 10
(10;20]
(20;30]
(30;40]
(40;50]
(50;60]
(60;70]
(70;80]
(80;90]
(90;100]
(100;110]
(110;120]
(120;130]
(130;140]
(140;150]
(150;160]
(160;170]
(170;180]
(180;190]
(190;200]
> 200
Inertial subnetwork
Mean value : 51.6 ms
Std Dev : 17.2 ms
Postural subnetwork
Mean value : 132.4 ms
Std Dev : 24.3 ms
Inertial subnet
Postural subnet

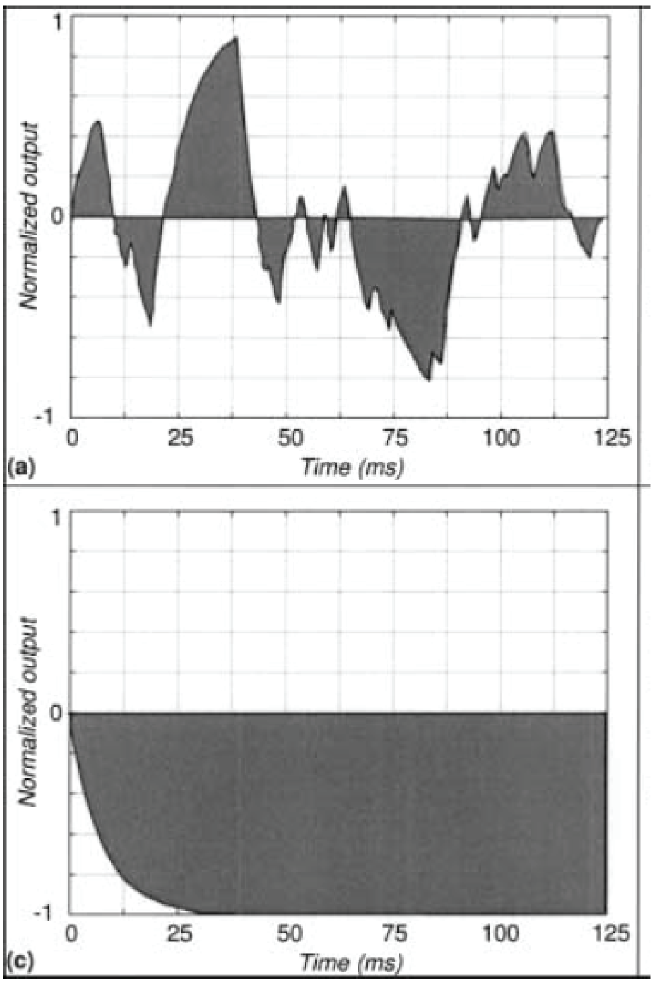


Figure 10. Histograms of the time constants in both postural and inertial subnetworks in the cases of the figure-eight movement (top) and of the standing-up movement (bottom). Temporal evolutions of the hidden neurons output: a,b The output signals of a t phasic hidden neuron and a tonic neuron.

& Flanders (1996). This idea is indirectly confirmed by the present result: The fact that the DRNN mapping between raw EMG signals and the related human movements gives rise to phasic and tonic artificial neuronal substrates is consistent with the neurophysiology of movement control.

In conclusion, the physiological plausibility obtained by our DRNN approach in different aspect of movement control might be of benefit for the potential use of the DRNN in prosthetic control (Craelius, 2002). In particular, the emergence of artificial structures within the DRNN architecture resembling to biological network could be used as a dynamically adaptive interface between EMG signals from residual muscles and artificial actuators (Cheron et al., 2003). New DRNNs would be dedicated to a larger repertoire of learned movements with generalized properties for the building of a patient-specific dynamical memory of motor actions.

5. References

Aksay, E.; Gamkrelidze, G.; Seung, H.S.; Baker, R. & Tank, D.W. (2001). In vivo intracellular recording and perturbation of persistent activity in a neural integrator. *Nat Neurosci*., 4(2),184-93.

Aksay, E.; Baker, R.; Seung, H.S.& Tank, D.W. (2003). Correlated discharge among cell pairs within the oculomotor horizontal velocity-to-position integrator. *J Neurosci*., 23(34), 10852-8.

Anastasio, T.J. & Robinson, D.A. (1991). Failure of the oculomotor neural integrator from a discrete midline lesion between the abducens nuclei in the monkey. *Neurosci Lett*., 127(1), 82-6.

Anastasio, T.J. & Gad, Y.P. (2007). Sparse cerebellar innervation can morph the dynamics of a model oculomotor neural integrator. *J Comput Neurosci*., 22(3), 239-54.

Bengoetxea A, Leurs F, Cebolla A, Wellens S, Draye Jp, Cheron G.(2005) A dynamic recurrent neural network for drawing multi-directional trajectories.*Comput Methods Biomech Biomed Engin*. Supp. 8:29-30.

Bengoetxea, A., Dan, B., Pozzo, T., Gillis, P., Leurs, F. And Cheron, G. (2008) Fast drawing movements: reciprocal muscle patterns encoded in a figure-centered coordinate frame (submitted)

Cannon, S.C.; Robinson, D.A. & Shamma, S. (1983) A proposed neural network for the integrator of the oculomotor system. *Biol Cybern.,* 49, 127–36.

Cannon, S.C. & Robinson, D.A. (1985). An improved neural-network model for the neural integrator of the oculomotor system: more realistic neuron behavior. *Biol Cybern*., 53, 93–108.

Cannon, S.C. & Robinson, D.A. (1987). Loss of the neural integrator of the oculomotor system from brain stem lesions in monkey. *J Neurophysiol*., 57, 1383–409.

Chan, W.W. & Galiana, H.L. (2005). Integrator function in the oculomotor system is dependent on sensory context. *J Neurophysiol*., 93(6), 3709-17.

Cheron, G.; Godaux, E.; Laune, J.M. & Vanderkelen, B. (1986a). Lesions in the cat prepositus complex: Effects on the vestibulo-ocular reflex and saccades. *J Physiol.,* 372, 75–94.

Cheron, G. & Godaux, E. (1986b). Self-terminated fast movement of the forearm in man : amplitude dependence of the triple burst pattern. *J. Biophys. Biom*., 10, 109-17.

Cheron, G.; Draye, J.P.; Bourgeois, M. & Libert, G. (1996) Dynamical neural network identification of electromyography and arm trajectory relationship during complex movements. *IEEE Trans Biomed Eng*., 43, 552–8.

Cheron, G.; Leurs, F.; Bengoetxea, A.; Draye, J.P.; Destree, M. & Dan, B. (2003). A dynamic recurrent neural network for multiple muscles electromyographic mapping to elevation angles of the lower limb in human locomotion. *J. Neurosci. Methods.*, 30, 95-104.

Cheron, G.; Cebolla, A.M.; Bengoetxea, A.; Leurs, F. & Dan. B, (2007). Recognition of the physiological actions of the triphasic EMG pattern by a dynamic recurrent neural network. *Neurosci Lett.*, 414(2), 192-6.

Craelius, W. (2002). The bionic man: restoring mobility. *Science*, 295(5557), 1018-21.

Delgado-Garcia, J.M.; Vidal, P.P.; Gomez, C. & Berthoz, A. (1989). A neurophysiological study of prepositus hypoglossi neurons projecting to oculomotor and preoculomotor nuclei in the alert cat. *Neuroscience*, 29(2), 291-307.

Draye, J.P.; Pavisic, D.; Cheron G. & Libert G. (1996). Dynamic recurrent neural networks: a dynamical analysis. *IEEE Transactions on Systems Man, and Cybernetics.*, 26, 692-706.

Draye, J.P.; Cheron, G.; Libert, G. & Godaux, E. (1997a) Emergence of clusters in the hidden layer of a dynamic recurrent neural network. *Biol Cybern.*, 76, 365–74.

Draye, J.P.; Cheron, G.; Pavisic, D. & Libert, G. (1997b). Improved identification of the human shoulder kinematics with muscle biological filters. *Lecture Note Series in Computer Science.*, 1211, 417-28.

Draye J.P.; Pavisic D.; Cheron G. & Libert G. (1997c) An inhibitory weight initialization improves the speed and quality of recurrent neural networks learning. *Neurocomputing,* 16, 207–24.

Draye, J.P.; Winters, J.M.; & Cheron, G. (2002). Self-selected modular recurrent neural networks with postural and inertial subnetworks applied to complex movements. *Biol. Cybern.*, 87, 27-39.

Escudero, M.; de la Cruz, R.R. & Delgado-Garcia, J.M. (1992). A physiological study of vestibular and prepositus hyglossi neurones projecting to the abducens nucleus in the alert cat. *J Physiol.*, 458, 539–60.

Escudero, M.; Cheron, G. & Godaux, E. (1996). Discharge properties of brain stem neurons projecting to the flocculus in the alert cat. II. Prepositus hypoglossal nucleus.*J Neurophysiol.*, 76(3), 1775-85.

Escudero, M. & Delgado-Garcia, J.M., (1988). Behaviour of reticular, vestibular and prepositus neurons terminating in the abducens nucleus of the alert cat. *Exp Brain Res.*, 71, 218–22.

Fetz, E.E.; Cheney, P.D.; Mewes, K. & Palmer, S. (1989). Control of forelimb activity by populations of corticomotoneuronal and rubromotoneuronal cells. *Prog Brain Res.*, 80, 437–49.

Fetz, E.E. (1992). Are movement parameters recognizably coded in the activity of single neurons? *Behav Brain Sci*, 15, 679–90.

Flanders, M. (1991). Temporal patterns of muscle activation for arm movements in the three-dimensional space. *J Neurosci.*, 11, 2680–93.

Godaux, E. & Cheron, G. (1996). The hypothesis of the uniqueness of the oculomotor integrator: direct experimental evidence in the cat. *J. Physiol.*, 492: 517–27.

Henn, V.; Hepp, K. & Buttner-Ennever, J.A. (1982) The primate oculomotor system. II. Premotor system. *Hum Neurobiol.*, 1, 87–95.

Jordan, M. I. & Rumelhart, D. E. (1992). Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16, 307–54.

Kalman, R.E.; Falb, P.L. & Arbib, M.A. (1969). *Topics in mathematical system theory*, McGraw-Hill, New York.

Kawato, M.; Furukawa, K. & Suzuki, R. (1987). A hierarchical neural network model for the control and learning of voluntary movement. *Biol Cybern.*, 57, 169–85.

Leurs, F.; Bengoetxea, A.; Cebolla, A. & Cheron, G. (2005). Reproducibility of the identification process of stump muscle EMG in prosthetic gait by a dynamic recurrent neural network, *Comput. Methods Biomech. Biomed. Engin*. Supp 1, 181.

Manto, M.; Jacquy, J.; Hildebrand, J. & Godaux, E. (1995). Recovery of hypermetria after a cerebellar stroke occurs as a multistage process. *Ann Neurol.*, 38, 437-45.

McCormick, D.A. (2001). Brain calculus: neural integration and persistent activity. *Nat. Neurosci.*, 4(2), 113-4.

McCormick, D.A.; Shu, Y.; Hasenstaub, A.; Sanchez-Vives, M.; Badoual, M. & Bal, T. (2003). Persistent cortical activity: mechanisms of generation and effects on neuronal excitability. *Cereb Cortex.*, 13(11), 1219-31.

Pearlmutter B.A. (1989). Learning state space trajectories in recurrent neural networks. *Neural Comput.,* 1, 263–69.

Pearlmutter B.A. (1995). Gradient calculations for dynamic recurrent neural networks: a survey. *IEEE Trans Neural Netw.*, 6, 1212–28.

Pelligrini, J.J. & Flanders, M. (1996). Force path curvature and conserved features of muscle activation. *Exp Brain Res*, 110, 80–90.

Robinson, D.A. (1989). Integrating with neurons. *Annu Rev Neurosci.,* 12, 33–45.

Scholz J.P. & Kelso J.A. (1990). Intentional switching between patterns of bimanual coordination depends on the intrinsic dynamics of the patterns. *J Mot Behav.,* 22(1), 98-124.

Silva, F.M. & Almeida, L.B. (1990). Speeding up backpropagation, In: *Advanced Neural Computers*, R. Eckmiller, 151-158, Elsevier, Amsterdam.

Soechting, J.F. & Flanders, M. (1997). Evaluating an integrated musculoskeletal model of the human arm. *J Biomech Eng.,* 119, 93– 102.

Tani J. (2003). Learning to generate articulated behaviour through the bottom-up and the top-down interaction processes. *Neural Netw.*, 16(1), 11-23.

Tani, J.; Nishimoto, R.; Namikawa, J. & Ito, M. (2008). Codevelopmental learning between human and humanoid robot using a dynamic neural-network model. *IEEE Trans Syst Man Cybern B Cybern.*, 38(1), 43-59.

Wada, Y. & Kawato, M. (1992). A neural network model for arm trajectory formation using forward and inverse dynamics models. *Neural Netw.,* 6, 919–932.

Wang, L. (1993). Singular value decomposition based space modelling and Kalman filtering techniques. *PhD thesis, Faculté Polytechnique de Mons*, Mons, Belgium.

Williams, R.J. & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.,* 1, 270–80.

Zajac, F.E.; Neptune, R.R. & Kautz, S.A. (2003). Biomechanics and muscle coordination of human walking: part II: lessons from dynamical simulations and clinical implications. *Gait Posture.*, 17(1), 1-17.

Linguistic Productivity and Recurrent Neural Networks

Akito Sakurai [1, 2] and Yoshihisa Shinozawa [1]
[1] *Keio University, and*
[2] *CREST, JST*
Japan

1. Introduction

Productivity is the defining property of a natural language. Any native speaker of a natural language utters a sentence that has never been heard and understands a sentence that has been heard for the first time. Chomsky claimed that the purpose of linguistics is to account for the productivity of natural languages (Chomsky, 1980).

The learnability of a productive language by computational mechanisms is hindered by the inherent nature of the language. For many years, researchers have attempted to devise a learning mechanism by which productive languages can be learnt in a manner similar to that adopted by a child learning from scratch or a student learning a second language. However, there are many problems resisted to be solved. Productivity is one of their causes.

This chapter is devoted to efforts undertaken to understand productivity in terms of language learning by means of simple but powerful methods such as neural networks, because neural networks are the simplest (maybe over-simplified) models of our brain mechanism we have obtained thus far.

It is natural to expect that a recurrent neural network (RNN) among them is capable of learning languages, specifically a subset of a natural language, because a sentence is a sequence of words and an RNN is capable of learning sequences.

The chapter consists of two parts, each of which is devoted to one of the two unmatched features of human languages—the recursive or self-embedding structure of human languages, and the syntactic or combinatorial systematicity of human languages. Both these features constitute the syntactic productivity of human languages.

2. Linguistic productivity and learnability

The difference between a natural language and other discrete symbolic systems is that a natural language is productive (Chomsky, 1959). The productivity of a language entails that a language is an infinite set of sentences; theoretically, the fact that a language is infinite refutes the learnability of a language. Strangely enough, any natural language can be learned by humans simply by hearing a finite number of, sometimes a very limited number of, sentences; furthermore, sometimes the sentences are ungrammatical (hence, they are not sentences linguistically), and it may not be indicated that they are ungrammatical (for example, refer (Pinker, 1984)).

If a natural language is learnable, it should be generated by some set of rules, which must be finite; otherwise, the language can never be learned. Gold's well-known theorems (Gold, 1967) state that even if the generating rules are finite, we would not be able to learn the language without negative examples (ungrammatical sentences).

Although productivity is the defining property of a natural language, a language should be characterized better by its learnability. Productive systems are easily built by, for example, rewriting systems; however, a learnable productive system is prohibitively difficult to build. Our natural language is, though, a proof of existence of the learnable productive system.

We could mention two key factors that consist the productivity of a language. One of the factors is systematicity and the other is recursiveness. Systematicity is related to the lexical category and recursiveness is related to the phrasal category.

Syntactic systematicity is a property by which a valid sentence remains valid when a word in it is replaced with another word belonging to the same lexical category. Since a lexical category is one of the syntactic categories and is defined as a group of words that are replaceable with each other in a sentence without invalidating the grammaticality of the sentence, the claim is a tautology. Therefore, syntactic systematicity can be better defined as a property by which the syntax is described using lexical categories. The merit of having systematicity is that the number of categories is maintained far fewer than the size of the lexicon, and therefore the rules are simpler. Since a word, specifically a noun, is coined quite easily and infinitely, systematicity ensures that an infinite number of new sentences are obtained from a sentence.

In general, systematicity is a wider concept, which has been under argument for years (for example, refer (Fodor & Pylyshyn, 1988) and the literatures citing it). We consider only syntactic systematicity, specifically weak systematicity and strong systematicity, as defined by Hadley (Hadley, 1994).

Recursiveness is a property by which a syntax is (exactly or approximately) modelled by a set of rewriting rules in which some phrasal category is directly or indirectly defined by itself. By the recursive rules, we can obtain an infinite number of different and valid sentences from a set of finite rules and lexicons. It must be noted that in recursive function theory or computational theory, the concept of recursiveness includes or is equivalent to that of repetition, which is easily observed from the definitions of recursive functions or Turing machines. Recursive rules of the form $A \rightarrow aA$ or $A \rightarrow Aa$ are called regular rules; they are not fully qualified "recursive" rules because the rules $A \rightarrow aA$ or $A \rightarrow Aa$ represent repetitions. Recursive rules generally refer to rules such as $A \rightarrow aAb$, which requires more than simply a finite state machine to parse the resultant sentences.

For language generation, when we use a symbolic representation of grammar, systematicity and recursiveness are easy to implement, and we can observe the variety or postulated infiniteness of the resultant languages.

For language learning or grammatical inference, systematicity and recursiveness pose challenges. We will first consider recursiveness, specifically the possibility or impossibility of representing recursive rules by RNNs; then, we will study recursiveness, specifically the possibility or impossibility of representing recursive rules by RNNs.

3. Recursion

In this section, we state a necessary condition: a simple RNN with two sigmoidal hidden units is a recognizer of the language $\{a^n b^n \mid n > 0\}$, whose grammar is expressed by $\{S \rightarrow aSb,$

$S \rightarrow ab\}$. Further, we show that there exists a set of parameters that satisfy the above condition. Then, we realize the language recognizer for the type of language mentioned above, although the stated condition implies the instability in learning, as has been reported in previous studies. Furthermore, the condition, contrary to its success in implementing the recognizer, implies the difficulty in obtaining a recognizer for more complicated languages.

3.1 Background

The researches conducted on the induction of a grammar that includes a recursive or self-embedding rule by neural networks have employed the following features:

- simple recurrent neural networks (SRN) adopted as the basic or core mechanism
- languages such as $\{a^n b^n \mid n>0\}$, $\{a^n b^n c^n \mid n>0\}$, which are clearly the results of, although not a representative of, recursive rules, adopted as target languages.

Here, a^n denotes a string of n-times repetition of a character a; the language $\{a^n b^n \mid n > 0\}$ is generated by a context-free grammar $\{S \rightarrow aSb, S \rightarrow ab\}$, and $\{a^n b^n c^n \mid n > 0\}$ is generated by a context-sensitive grammar. The adoption of these simple languages as target languages is inevitable because it is not easy to determine whether or not sufficient generalization is achieved by the learning, if realistic grammars are used.

Although these target languages were simple, it has been pointed out that they were learned but their grammars were not, because sufficient generalization by the learned network was not observed and the resultant network appeared to be almost similar to the result of rote learning. Further, the resultant networks were unstable in the sense that when they were given new training sentences, which were longer than the ones they had learned, the learned network changed to completely new networks that were more than simple refinements of the learned network.

Bodén et al. (Bodén et al., 1999; Bodén & Wiles, 2000; Bodén & Blair, 2003), Rodriguez et al. (Rodriguez et al., 1999; Rodriguez, 2001), Chalup (Chalup & Blair, 2003), and others conducted investigations during exploration for the possibility of network learning of the languages. However, they have not succeeded in clearly stating the conditions that the learned networks should satisfy.

In this section, we state a property describing SRNs with two hidden units that learned to recognize a language $\{a^n b^n \mid n>0\}$, that is, a necessary condition for an SRN to be qualified as a successful language recognizer. The stated condition implies the instability in learning. We also show the realization of the condition and obtain the recognizer for the language. However, the question— if there really exists a solution—remains unanswered.

3.2 Preliminaries

A recurrent neural network (RNN) is a network that has recurrent connections added to a feed-forward network. The calculation proceeds at first for the feed-forward part and after a single time-unit delay for the recurrent connection part; hence, the inputs for the calculations in the feed-forward part are supplemented with the outputs of the recurrent connections.

An RNN is considered to be a discrete time system. Starting with an initial state (initial outputs of the feed-forward part, i.e., outputs without external inputs), the network proceeds to accept the subsequent character in a string given to the external inputs, reaches the final state, and obtains the final external output from the final state.

A simple recurrent network (SRN) is a simple type of RNN and has only one layer of hidden units in its feed-forward part.

Rodriguez et al. (Rodriguez et al., 1999) showed that an SRN learns languages $\{a^nb^n \mid n > 0\}$ and $\{a^nb^nc^n \mid n > 0\}$. For $\{a^nb^n \mid n > 0\}$, an SRN successfully accepted a language $\{a^nb^n \mid 0 < n \leq 16\}$ after processing sentences language $\{a^nb^n \mid 0 < n \leq 11\}$. They analyzed and described the manner in which the input sentences were processed by the SRN. However, the analysis was limited to the results obtained by simulations and did not proceed into indicating how and when the learning is possible. Moreover, the existence of the language recognizers was not yet shown.

Siegelmann (Siegelmann, 1999) showed that the computational ability of an RNN is superior to that of a Turing machine, thereby implying that the recoginizers for the languages $\{a^nb^n \mid n>0\}$ and $\{a^nb^nc^n \mid n>0\}$ exist. However, based on unsatisfactory generalization obtained by the learning experiments, we would state that the possibility of learning and existence of realization is different or it might be the case that the solution does not exist seemingly contradictive to Siegelmann's result because there is a difference in formulation: the output functions of the units are piecewise linear functions; the inversibility of the function is utilized in Siegelmann's case and sigmoidal functions are utilized in standard SRN cases. It must be noted that piecewise linear functions have non-differentiable points, which makes them infeasible to utilize error-backpropagation algorithm.

On the other hand, Casey (Casey, 1998) and Maass (Maass & Orponen, 1998) showed that in noisy environments, an RNN is as powerful as finite automaton. The results suggest that we should consider infinite precision computation when we have to search the possibility of computations by an RNN or specifically an SRN.

In summary, the learnability of RNN and SRN recognizers for the language $\{a^nb^n \mid n > 0\}$ is not yet demonstrated, and moreover the existence of the recognizers is not yet proved. To conduct further research, we need to suppose that the computations performed by the RNN and SRN should be formulated with infinite precision, and the units should use sigmoidal functions. Therefore, in this research, we have adopted RNN models with infinite precision calculations and the sigmoidal function ($\tanh(x)$) as the output function for the units.

On the basis of the viewpoints mentioned above, we discuss two points in this section: a necessary condition for an SRN with two hidden units to be a recognizer for the language $\{a^nb^n \mid n>0\}$, and whether or not the stated condition is sufficient to guide us to build an SRN language recognizer.

3.3 Symbols and terminology

An SRN is a simple type of RNN and its function is expressed as follows:

$$\boldsymbol{s}_{n+1} = \sigma(\, \boldsymbol{w}_s \cdot \boldsymbol{s}_n + \boldsymbol{w}_x \cdot \boldsymbol{x}_n \,) \tag{1}$$

$$N_n(\, \boldsymbol{s}_n \,) = \boldsymbol{w}_{os} \cdot \boldsymbol{s}_n + \boldsymbol{w}_{oc} \tag{2}$$

Here, σ is a standard sigmoid function ($\tanh(x) = (1 - \exp(-x))/(1 + \exp(-x))$), which is applied componentwise.

A *counter* is a device that stores an integer and allows +1 or −1 operation and answers yes or no to an inquiry if the content is 0 (*0-test*). A *stack* is a device that allows the operations *push i* (*store i*) and *pop up* (recover the last-stored content, discard it and restore the device to its state immediately before the corresponding push operation). Clearly, a stack is a more powerful device as compared to a counter; hence, if a counter is not implementable, a stack, too, is not implementable.

To represent an input or output word, we adopt a *localist representation* or *one-hot vector*, a vector representation in which a single element is 1 and the other elements are 0. The network is trained so that the sum of the squared error (difference between the actual output and desired output) is minimized.

In learning experiments of languages, it is natural to assume there is no negative, i.e., ungrammatical sentence; therefore, we have to devise some teaching information from the sentences. Elman proposed to train an SRN as a predictor, i.e., to teach the network to predict a word when it sees a sub-string of words up to the word (refer (Elman, 1991)). By this method, if two possible outputs with the same frequency of occurrence for the same input exist in a training data, the network would learn to output the same value for two elements in the output with those for the other elements being 0; this is because the output vector should provide the minimum value of the sum of the squared error.

For the language $\{a^n b^n \mid n>0\}$, when an SRN is trained to predict the subsequent word, it behaves internally as a counter. Clearly, if SRN could count the number of *a*s and *b*s, it would be able to recognize $\{a^n b^n \mid n>0\}$. In fact, Rodriguez et al. (Rodriguez et al., 1999) and others analyzed trained networks and found that they behave like counters.

It is clear that if a network correctly predicts the subsequent character in a string in the language $\{a^n b^n \mid n>0\}$, we could see it as a counter, although its capability is limited. Let us add an auxiliary network output whose value is positive if the original network predicts only "*a*" (which happens only when the input string up to the time was $a^n b^n$ for some n) and is negative otherwise.

The modified network behaves as if it counts up for a character "*a*" and counts down for "*b*," because it outputs positive values when the number of "*a*"s and "*b*"s coincide and outputs negative values otherwise. However, the counting capability of the network may be limited because it would output any value when "*a*" is fed before the due number of "*b*"s are fed, that is, when a counting up action is required before the counter returns back to the 0-state.

As suggested by Rodriguez et al. (Rodriguez et al., 1999), we consider an SRN to be a dynamical system. For the terminology related to dynamical systems, specifically terms such as ω-limit set and stable/unstable manifold, we suggest that the readers refer to (Katok & Hasselblatt, 1996) or (Guckenheimer & Holmes, 1997). Concise definitions are provided in the Appendix of (Rodriguez et al., 1999).

A (discrete-time) *dynamical system* is represented as the iteration of a function application: $\boldsymbol{s}_{i+1} = f(\boldsymbol{s}_i)$, where $i \in N$, $\boldsymbol{s}_i \in R^n$. A point s is called a fixed point of f if $f(s) = s$. A point s is an attracting fixed point of f if s is a fixed point and there exists a neighbourhood U_s around s such that $\lim_{i\to\infty} f^{\,i}(x) = s$ for all $x \in U_s$. A point s is a repelling fixed point of f if s is an attracting fixed point of f^{-1}. A point s is called a periodic point of f if $f^n(s) = s$ for some n.

A point s is a ω-limit point of x for f if $\lim_{i\to\infty} f^{n_i}(x) = s$ for $\lim_{i\to\infty} n_i = \infty$. A fixed point x of f is hyperbolic if all of the eigenvalues of Df at x have absolute values other than one, where $Df = \partial f_i / \partial x_j$ is the Jacobian matrix of the first partial derivatives of the function f. A set D is invariant under f if for any $s \in D$, $f(s) \in D$.

Theorem 1 (Stable Manifold Theorem for a Fixed Point (Guckenheimer & Holmes, 1997))

Let $f: R^n \to R^n$ be a C^r $(r \geq 1)$ diffeomorphism with a hyperbolic fixed point x. Then there exist local stable and unstable manifolds $W^{s,f}_{\mathrm{loc}}(x)$, $W^{u,f}_{\mathrm{loc}}(x)$, tangent to the eigenspaces $E^{s,f}_{x}$, $E^{u,f}_{x}$ of Df at x and of corresponding dimension. $W^{s,f}_{\mathrm{loc}}(x)$ and $W^{u,f}_{\mathrm{loc}}(x)$ are as smooth as the map f, i.e., of class C^r.

Local stable/unstable manifolds for f are defined as follows (Corollary 6.2.5 in (Katok & Hasselblatt, 1996)):

$$W^{s,f}_{\mathrm{loc}}(q) = \{ y \in U_q \mid \lim_{m\to\infty} \mathrm{dist}(f^m(y), q) = 0 \}$$

$$W^{u,f}_{\mathrm{loc}}(q) = \{ y \in U_q \mid \lim_{m\to\infty} \mathrm{dist}(f^{-m}(y), q) = 0 \}$$

where U_q represents the neighbourhood of q, and dist is the distance function. Then, the global stable and unstable manifolds for f are defined as follows:

$$W^{s,f}(q) = \cup_{i\geq 0} f^{-i}(W^{s,f}_{\mathrm{loc}}(q))$$

$$W^{u,f}(q) = \cup_{i\geq 0} f^{i}(W^{u,f}_{\mathrm{loc}}(q))$$

As defined, an SRN is a pair of a discrete-time dynamical system $\boldsymbol{s}_{n+1} = \sigma(\boldsymbol{w}_s\cdot\boldsymbol{s}_n+\boldsymbol{w}_x\cdot\boldsymbol{x}_n;)$ and an external output part $N_n(\boldsymbol{s}_n) = \boldsymbol{w}_{os}\cdot\boldsymbol{s}_n+\boldsymbol{w}_{oc}$. We simply express the former (dynamical system part) as $\boldsymbol{s}_{n+1} = f(\boldsymbol{s}_n,\boldsymbol{x}_n)$ and the external output part as $h(\boldsymbol{s}_n)$.

When an RNN (or SRN) is considered to be a counter, an input $\boldsymbol{x}_+$ plays the role of the "+1" count-up operation and another input $\boldsymbol{x}_-$ performs the "−1" or count-down operation. For simplicity, hereafter, let $f_+ = f(\cdot,\boldsymbol{x}_+)$ denote the "+1" operation and $f_- = f(\cdot,\boldsymbol{x}_-)$ denote the "−1" operation. It must be noted that when the network is used as a recognizer of the language $\{a^nb^n \mid n>0\}$, the input character "a" corresponds to $\boldsymbol{x}_+$ and "b" corresponds to $\boldsymbol{x}_-$. Further, f_- is undefined for the point outside and on the border of the square I[−1,+1] × I[−1,+1], where I[−1,+1] is the closed interval [−1,+1]; however, we do not mention it for simplicity.

D_0 is a set $\{\boldsymbol{s} \mid h(\boldsymbol{s}) \leq 0\}$, that is, a region where the counter value is 0 and which is simply connected when the network is an SRN because h is effectively a linear function. Let $D_i = f_-^{-i}(D_0)$, that is, a region where the counter value is i.

We postulate that $f_+(D_i)\subseteq D_{i+1}$. This means that any point in D_i is eligible for a state which designates that the counter content is i. This may appear to be rather demanding. An alternative approach would be that in which the point p corresponds to counter content c if and only if $p = f_-^{m1} f_+^{p1} \cdots f_-^{mi} f_+^{pi}(s_0)$ for a predefined s_0, some $m_j \geq 0$ and $p_j \geq 0$ for $1 \leq j \leq i$, and $i \geq 0$ such that $\sum_{j=1}^{i}(p_j - m_j) = c$. However, this approach has not resulted in a fruitful result.

We also postulate that the closures of D_i are disjoint. Since we defined D_i as a closed set, the postulate is natural. Our consideration was to select D_i to be closed. The postulate requires that we should maintain a margin between D_0 and D_1 and any other D_is.

3.4 Necessary condition

In this subsection, we consider only an SRN with two hidden units, i.e., all the vectors concerning $\boldsymbol{s}$ such as $\boldsymbol{w}_s$, $\boldsymbol{s}_n$, $\boldsymbol{w}_{os}$ are two-dimensional vectors.

Definition 2. D_ω is the set of the accumulation points of $\{D_i \mid i > 0\}$, i.e., $s \in D_\omega$ iff $s = \lim_{i\to\infty} s_{ki}$ for some $\boldsymbol{s}_{ki} \in D_{ki}$.

Definition 3. P_ω is the set of ω-limit points of points in D_0 for f_+, i.e., $s \in P_\omega$ iff $s = \lim_{i\to\infty} f_+^{ki}(s_0)$ for some k_i and $s_0 \in D_0$. Q_ω is the set of ω-limit points of points in D_0 for f_-^{-1}, i.e., $s \in Q_\omega$ iff $s = \lim_{i\to\infty} f_-^{-k}(s_0)$ for some k_i and $s_0 \in D_0$.

With regard to the results obtained by Bodén et al. (Bodén et al., 1999; Bodén & Wiles, 2000; Bodén & Blair, 2003), Rodriguez et al. (Rodriguez et al., 1999; Rodriguez, 2001), Chalup (Chalup & Blair, 2003), it is natural, at least during the first consideration, to postulate that for any x, $f_+^{i}(x)$ and $f_-^{i}(x)$ do not wonder and therefore will converge to periodic points.

Therefore, P_ω and Q_ω are postulated as finite sets of hyperbolic periodic points for f_+ and f_-, respectively. For simplicity in presentation, we postulate that P_ω and Q_ω are finite sets of hyperbolic fixed points for f_+ and f_-, respectively.
Moreover, the points in Q_ω are saddle points for f_-; hence, we further postulate that $W^{u,f^-}{}_{\mathrm{loc}}(q)$ for $q \in Q_\omega$ and $W^{s,f^-}{}_{\mathrm{loc}}(q)$ for $q \in Q_\omega$ are one-dimensional space and their existence is guaranteed by Theorem 1.
Postulate 4. We postulate that $f_+ (D_i) \subseteq D_{i+1}$, the closures of D_i are disjoint, P_ω and Q_ω are finite sets of hyperbolic fixed points for f_+ and f_-, respectively, and $W^{u,f^-}{}_{\mathrm{loc}}(q)$ for $q \in Q_\omega$ and $W^{s,f^-}{}_{\mathrm{loc}}(q)$ for $q \in Q_\omega$ are one-dimensional spaces.
Lemma 5. $f_-^{-1} \circ f_+(D_\omega) = D_\omega$, $f_-^{-1}(D_\omega \cap \mathrm{I}(-1,1) \times \mathrm{I}(-1,1)) = D_\omega$ and $D_\omega \cap \mathrm{I}(-1,1) \times \mathrm{I}(-1,1) = f_-(D_\omega)$, and $f_+(D_\omega) \subseteq D_\omega$. $P_\omega \subseteq D_\omega$ and $Q_\omega \subseteq D_\omega$.
Definition 6. $W^{u,-1}(q)$ is the global unstable manifold at $q \in Q_\omega$ for f_-^{-1}, i.e., $W^{u,-1}(q) = W^{u,(f^-)^{-1}}(q) = W^{s,f^-}(q)$.
Lemma 7. For any $p \in D_\omega$, any accumulation point of $\{ f_-^i (p) \mid i > 0 \}$ lies in Q_ω
Proof. Since p lies in D_ω, there exist $p_{ki} \in D_{ki}$ such that $p = \lim_{i\to\infty} f_+^{ki}(p_{ki})$. Suppose q in D_ω is the accumulation point stated in the theorem statement, i.e., $q = \lim_{j\to\infty} f_-^{hj}(p)$. We set k_i to be sufficiently large for any h_j so that p_{ki} exists in any neighbourhood of q with $f_-^{hj} (p)$. Then, $q = \lim_{j\to\infty} f_-^{hj} (p_{ki}) = \lim_{j\to\infty} f_-^{hj-ki} (s_{ki})$, where k_i is a function of h_j with $k_i > h_j$. Let $s_{ki} = f_-^{-ki}(p_{ki}) \in D_0$ and $s_0 \in D_0$ be an accumulation point of $\{s_{ki} \}$. Then, since f_-^{-1} is continuous, by setting $n_j = -h_j + k_i > 0$, we get $q = \lim_{j\to\infty} f_-^{nj}(s_0)$, i.e., $q \in Q_\omega$. □
Lemma 8. $D_\omega = \cup_{q \in Q_\omega} W^{u,-1}(q)$
Proof. Let p be any point in D_ω. Since $f_-(D_\omega) \subseteq \mathrm{I}[-1,1] \times \mathrm{I}[-1,1]$ where $\mathrm{I}[-1,1]$ is the closed interval $[-1,1]$, i.e., $f_-(D_\omega)$ is bounded, and $f_-(D_\omega) \subseteq D_\omega$, $f_-^n(p)$ has an accumulation point q in D_ω, which is, by Lemma 7, in Q_ω. Then, q is expressed as $q = \lim_{j\to\infty} f_-^{nj}(p)$. Since Q_ω is a finite set of a hyperbolic fixed point, $q = \lim_{n\to\infty} f_-^n (p)$, i.e., $p \in W^{s,f}(q) = W^{u,f^{-1}}(q) = W^{u,-1}(q)$. □
Since $P_\omega \subseteq D_\omega$, the next theorem holds.
Theorem 9. A point in P_ω is either a point in Q_ω or in $W^{u,-1}(q)$ for some $q \in Q_\omega$.
It must be noted that since $q \in W^{u,-1}(q)$, the theorem statement simply states that "If $p \in P_\omega$, then $p \in W^{u,-1}(q)$ for some $q \in Q_\omega$."

3.5 An Example of a recognizer

To construct an SRN recognizer for $\{ a^n b^n \mid n>0 \}$, the SRN should satisfy the conditions stated in Theorem 9 and Postulate 4, which are summarized as follows:

1. $f_+ (D_i) \subseteq D_{i+1}$,
2. the closures of D_i are disjoint,
3. P_ω and Q_ω are finite sets of hyperbolic fixed points for f_+ and f_-, respectively,
4. $W^{u,f^-}{}_{\mathrm{loc}}(q)$ for $q \in Q_\omega$ and $W^{s,f^-}{}_{\mathrm{loc}}(q)$ for $q \in Q_\omega$ are one-dimensional spaces, and
5. if $p \in P_\omega$ then $p \in W^{u,-1}(q)$ for some $q \in Q_\omega$.

To find a solution as simply as possible, let us try to suppose that $p \in P_\omega$ and $q \in Q_\omega$, that is, $f_+(p) = p$ and $f_-(q) = q$. Since p cannot be the same point as q (because $f_-^{-1} \circ f_+(p) = p + \boldsymbol{w}_s^{-1} \cdot \boldsymbol{w}_x \cdot (\boldsymbol{x}_+ - \boldsymbol{x}_-) \neq p$), we have to find a way to cause $p \in W^{u,-1}(q)$.
Since it is generally very difficult to calculate stable or unstable manifolds from a function and its fixed point, we attempt to allow $W^{u,-1}(q)$ to be a "simple" manifold; if $W^{u,-1}(q)$ is simple, it is easy to define $D_0 = \{\boldsymbol{x} \mid h(\boldsymbol{x}) \geq 0\}$; on the other hand, if $W^{u,-1}(q)$ is not simple, a suitable h may not exist.

We have decided that $W^{u,-1}(q)$ is a line (if possible). Considering the function form $f_-(\boldsymbol{s}) = \sigma(\boldsymbol{w}_s \cdot \boldsymbol{s} + \boldsymbol{w}_x \cdot \boldsymbol{x}_-)$, it is not difficult to observe that the line could be one of the axes or one of the bisectors of the right angles at the origin (i.e., one of the lines $y = x$ and $y = -x$). We have selected the bisector in the first (and the third) quadrant (i.e., the line $y = x$). q is selected as the origin, and p is selected arbitrarily as (0.8, 0.8).

The condition stated in Item 4 in the above is satisfied by setting one of the two eigenvalues of Df_- at the origin to be greater than 1 and the other eigenvalue smaller than 1. We have selected $1/0.6$ and $1/\mu$ for the two eigenvalues so that the conditions stated in Item 1 and 2 in the above are satisfied by considering the eigenvalues of Df_+ at p for f_+.

The design consideration that we have ignored is the design of $D_0 = \{\boldsymbol{x} \mid h(\boldsymbol{x}) \geq 0\}$. A simple method is to make the boundary $h(\boldsymbol{x}) = 0$ parallel to $W^{u,-1}(q)$ for our intended $q \in Q_\omega$; if we do so, by setting the largest eigenvalue of Df_- at q to be equal to the inverse of the eigenvalue of Df_+ at p along the normal to $W^{u,-1}$, we can obtain the points $s \in D_0$, $f_- \circ f_+(s)$, $f_-^2 \circ f_+^2(s), \ldots$, $f_-^i \circ f_+^i(s), \ldots$ that belong to $\{a^n b^n \mid n>0\}$ and reside at approximately equal distances from $W^{u,-1}$. It is apparent that the points belonging to, say, $\{a^{n+1} b^n \mid n > 0\}$ have approximately equal distances from $W^{u,-1}$, and this distance is different from that for $\{a^n b^n \mid n>0\}$

Let $f_-(x) = \sigma(Ax + B_0)$, $f_+(x) = \sigma(Ax + B_1)$. We plan to set $Q_\omega = \{(0,0)\}$, $P_\omega = \{(0.8,0.8)\}$, $W^{u,-1} = \{(x,y) \mid y = x\}$; the eigenvalues of the tangent space of f_-^{-1} at (0,0) are $1/\lambda = 1/0.6$ and $1/\mu$ (where the eigenvector on $y = x$ is expanding), and the eigenvalues of the tangent space of f_+ at (0.8,0.8) are $1/\mu$ and any value. Then, considering the derivatives at (0,0) and (0.8,0.8), it is easy to determine that

$$\frac{1}{2}A = \rho\left(\frac{\pi}{4}\right)\begin{pmatrix} \lambda & 0 \\ 0 & \mu \end{pmatrix}\rho\left(-\frac{\pi}{4}\right), \quad \frac{1}{\mu} = \left(1 - 0.8^2\right)\mu$$

where $\rho(\theta)$ is a rotation by θ. Then

$$A = \begin{pmatrix} \lambda + \mu & \lambda - \mu \\ \lambda - \mu & \lambda + \mu \end{pmatrix}$$

Next from $\sigma(B_0) = (0,0)^T$ and $\sigma((0.8\lambda, 0.8\lambda)^T + B_1) = (0.8,0.8)^T$, we get

$$B_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad B_1 = \begin{pmatrix} \sigma^{-1}(0.8) - 0.8\lambda \\ \sigma^{-1}(0.8) - 0.8\lambda \end{pmatrix}$$

These give us $\mu=5/3$, $\lambda=0.6$, $B_1 \approx (1.23722, 1.23722)^T$.

In Fig. 1, the left-hand side image shows the vector field of f_+, where the arrows starting at the x end at $f_+(x)$ and the right image shows the vector field of f_-. In the left-hand pane of Fig.2, the group of points at the centre (red) correspond to $\{a^n b^n \mid n>0\}$, those at the top (blue) correspond to $\{a^{n+1} b^n \mid n > 0\}$, and those at the bottom (green) correspond to $\{a^n b^{n+1} \mid n > 0\}$. The initial point is set to $p = (0.5,0.95)$ in Fig. 2. All the points correspond to $n = 1$ to $n = 40$, and when n grows, the points group together to points on $y = -x$, forming narrow stripes, i.e., D_n, for some n. As shown in the right-hand pane of Fig. 2, numerical computations of $f_-^n \circ f_+^n$ are sensitive to small truncation errors. In the case of Mathematica, the points start straying away for $n \geq 47$.

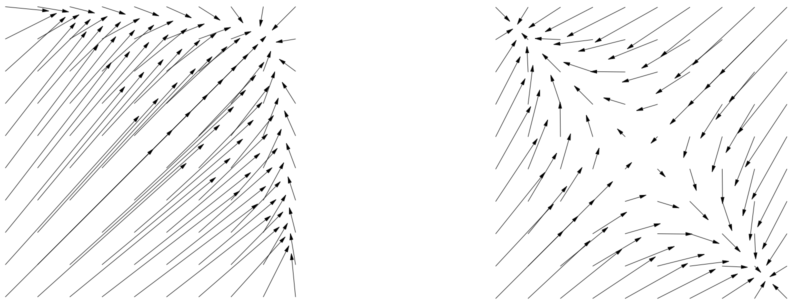


Fig. 1. Vector field representation of f_+ (left) and f_- (right)

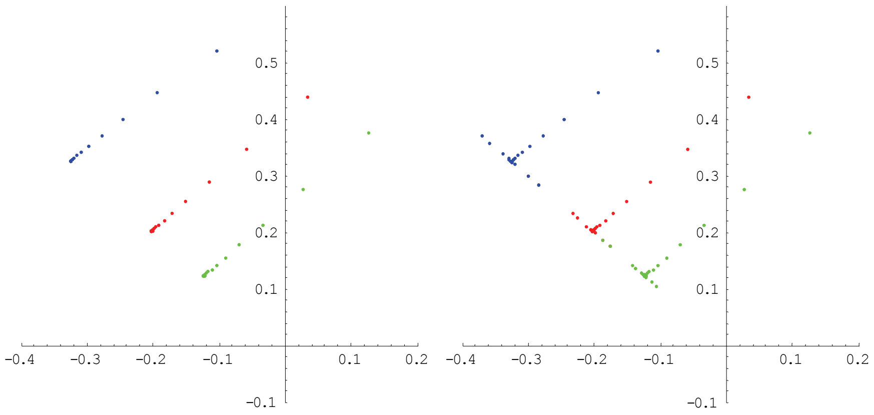


Fig. 2. On the left-hand side, $\{ f_-^n \circ f_+^n(p) \mid 40 \geq n \geq 1\}$ (middle; red), $\{ f_-^{n+1} \circ f_+^n(p) \mid 40 \geq n \geq 1\}$ (top; blue), and $\{ f_-^n \circ f_+^{n+1}(p) \mid 40 \geq n \geq 1\}$ (bottom; green), where $p = (0.5,0.95)$ are plotted. On the right side, the plots are identical to those on the left-hand side, except for $70 \geq n \geq 1$.

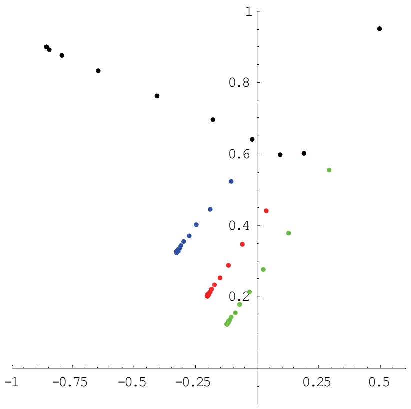


Fig. 3. Points of $\{(f_- \circ f_+)^n(p) \mid 11 \geq n \geq 1\}$, where $p = (0.5,0.95)$ are plotted over the left-hand pane of Fig. 2, which shows that the points do not stay in D_0. The points start from p and converge at around (–0.8595, 0.8984).

3.6 Discussion for recursiveness

We obtained a necessary condition for an SRN to be used as a recognizer for the language $\{a^n b^n \mid n > 0\}$ by analyzing its proper behavior from the viewpoint of discrete dynamical systems. The stated condition supposes that the closures of D_i are disjoint, $f_+(D_i) \subseteq D_{i+1}$, and Q_ω is finite.

This suggests a possibility for the implementation of the recognizer; in fact, we have successfully built a recognizer for the language, thereby showing that the learning problem of the language has at least one solution. However, it is worthwhile to be cautious. As shown in Fig. 3, $f_- \circ f_+(p)$ does not return to around p unless there is no proper setting for D_0 to which p belongs. This means that we have to "reset" the counter when it returns to the 0-state in order to reuse it. The experimental setting used, for example, (Elman, 1990), i.e., a setting where a string *aabbaaaabbbbab…* is used is not appropriate to obtain the recognizer.

It is suggested (but not logically derived) that the instability of any solution for learning occurs due to the necessity of P_ω being in an unstable manifold $W^{u,-1}(q)$ for some $q \in Q_\omega$. Since P_ω is an attractive fixed point in the above example, $f_+^n(s_0)$ for $s_0 \in D_0$ approaches exponentially close to P_ω for n. Even a small fluctuation in the position of P_ω, since $f_+^n(s_0)$, too, is close to $W^{u,-1}(q)$, $f_-^n(f_+^n(s_0))$, which should be in D_0, is significantly disturbed. This means that even if an temporary solution is close to a correct solution, due to a small fluctuation in the position of P_ω caused by a new training data, $f_-^n(f_+^n(s_0))$ may easily be pushed out of D_0.

Since Rodriguez et al. (Rodriguez, 2001) showed that the languages that do not belong to the context-free class could be learned to some degree, we have to conduct further study on the discrepancies.

These instabilities of grammar learning by SRN mentioned above might not be visible in our natural language learning; this suggests that an SRN might not be appropriate for a model of language learning.

4. Systematicity

Hadley defines three degrees of syntactic systamaticity (Hadley, 1994). We focus on two of the degrees, namely weak systematicity and strong systematicity. According to Hadley, supposing that a training corpus is "representative" in the sense that every word (noun, verb, etc.) that occurs in some sentence of the corpus also occurs (at some point) in every permissible syntactic position, if a set of test sentences is the one that contains only grammatical sentences which are syntactically isomorphic to sentences in the training corpus, and that no new vocabulary is present, and a network is capable of successfully processing (by recognizing or interpreting) novel test sentences, then the network is said to be (at least) weakly systematic. Hadley defines that a system is strongly systematic if (i) it can exhibit weak systematicity and (ii) it can accurately process a variety of novel simple sentences and novel embedded sentences containing previously learned words in positions where they do not appear in the training corpus (i.e., the word within the novel sentence does not appear in that same syntactic position within any simple or embedded sentence in the training corpus). For subtleties in the above definition, refer to (Christiansen & Chater, 1994).

It is clear that if a training set is sufficiently large and syntactic sentence patterns are limited (for example, the length of the sentences is limited to at most ten words), we could state that a Bayesian method with lexical categories as a latent variable would provide a satisfactory result. In fact, many researches have targeted unsupervised language learning thus far. (Schütze, 1993) is an early research conducted on unsupervised language learning.

These researches show that the lexical categories could be induced from an unlabeled corpus if we adopt an approach based on symbolic paradigm and statistics. A question remains: is it possible for a simple method such as an RNN to induce lexical categories with, for example, an error-backpropagation algorithm?

If an SRN or RNN is able to learn the grammar of a language, they would be able to learn the lexical categories. However, since a large lexicon implies a large number of connections and possibly the necessity for a large number of internal nodes, it might make learning difficult for a network. Therefore, we must develop some mechanism, other than the SRN or RNN, to group similar words into clusters, which are input to the SRN or RNN. The clustering algorithm to be used may be explicit or implicit; here, "explicit" means that a clustering mechanism other than SRN or RNN is used, whereas "implicit" means that a sub-network is added to the SRN or RNN and is trained with the main SRN or RNN. (Elman, 1991) adopts the latter approach, whereas (Farkaš & Crocker, 2008) adopts the former one. Frank adopts an approach that does not use additional networks (Frank, 2006).

Farkaš and Crocker approached the problem by adding a type of self-organizing map (SOM). By means of the SOM, they successfully constructed a distributional representation of words, as done explicitly in probabilistic approaches (e.g. (Schütze, 1993)). The SOM is used not only for mapping from an input word to its distributional representation (a type of category) but also vise versa. The success of their approach clearly shows the possibility of existence of two types of networks, i.e., networks for clustering and networks for grammar.

One of the problems that remains is that since an SOM provides graded responses, it may be the case that frequency-related information creeps in the SOM output and SRN is trained not only on categories but also on frequency. Since we may not know how much of the result depends on the frequency-related information, we would be a little hesitant to say that the SRN has learned the grammar described in the lexical categories

One of the issues that we have to consider when we adopt the approach of "graded responses" such as statistics and neural networks with real value outputs is the performance criteria for the results. In this chapter, we consider an RNN with linear input functions and sigmoidal activation functions, which assume values between 0 and 1. Although the words are represented by localist representation or one-hot vector and therefore for an input, exactly one node is 1 and others are 0, the outputs assume real values, that is, possibly all the nodes have graded predictions for words.

Networks are trained so that an output designates the subsequent word. A training data set may be ambiguous in a sense that the possible number of subsequent words after a sequence may exceed one. Then, as usual, the output values are considered to be a distributional representation of the subsequent words or the likelihood of the network prediction. Suppose that there are training sequences that consist of a common prefix for a string of words but more than one word comes after the prefix. If a network is properly trained, we could expect that the network's output prediction is uniform for the subsequent words after the common prefix string. If some word appears more often than another word, the output activation for the former word must be higher than that for the latter word, although the activations may not be proportional to their frequency.

Consequently, in a manner different from many neural network classification researches, which employ the sum of errors for each training samples, we must measure the distributional differences.

Measuring the degree of systematicity in a trained network is a problem.
Elman did not conduct a direct test to observe whether the trained network exhibited weak systematicity or equivalent property, because Elman's pioneering study was conducted before Hadley's proposal of weak systematicity. However, Elman showed the result of analysis on activations of hidden units, which effectively showed the categorization of words. The dendrogram in (Elman, 1991) shows the hierarchical structure among words and the similarity between word groups, although it is constructed on the basis of activations summed over all the contexts before the target word. Therefore, there remains the question whether the network really learned the hierarchy or the result is simply a result of statistical analysis of the network's behaviour. In other words, although the network was trained to predict a word, it had not acquired the capability to predict a category (or distribution of possibilities of words) of the subsequent word.
Frank, in (Frank, 2006), adopted a criterion that compares the sum of activations for words in the expected category and the unexpected category. However, a malicious network may not predict a word in an incorrect category but predict only one word among many words in the correct category.
One and only one alternative is to measure the distributional responses in terms of, for example, χ^2 distance or KL divergence.
We have another problem concerning training data. To examine the systematicity of an RNN, Frank selected a rather difficult problem. In (Frank, 2006), the training data include data of type N1 V1 N1 and N2 V2 N2, where N1 and N2 represent the noun categories and V1 and V2 verb category, whereas the test data is of N2 V1 N2 and N1 V2 N2, requiring that a network should induce that words in N1 and N2 are in the same category. The requirement appears to be rather demanding, because no suggestion of equating the position of N1 in the first sentence and N2 in the second and that of V1 and V2 is provided. Since the network inevitably learns the frequencies of words, it may learn to predict N2 after V2, irrespective of the first N1, or to predict N2 in the third place when N2 appears in the first place regardless of V1 in the second place.
It is better to select simpler settings for the training data and test data.
In this section, we examine the learnability of systematicity in a simpler situation. It is hypothesized that the systems of categories exist *a priori*; hence, learners need to only select the best one among them. Suppose that SRNs are furnished with an input conversion sub-network that converts an input word into its category and undergoes learning to predict the subsequent word. We found that an SRN with a conversion sub-network consistent with the grammatical categories of words in input sentences has the smallest learning error in terms of predicting frequency. Consequently, we can recover correct categorization of words by observing the SRN outputs, calculating learning errors, and selecting the SRN with the least learning error.

4.1 Experimental settings: network architecture

In systematicity experiments, SRNs are augmented with a new input layer (Fig. 4 (left)) which corresponds to the primary networks in (Bodén, 2004). The connection weights between the first and second input layers are fixed during learning with 0 (disconnected) or 1 (connected). The activation function of the second input layer is linear and that of the hidden and output units is the standard sigmoid function $1/(1 + \exp(-x))$

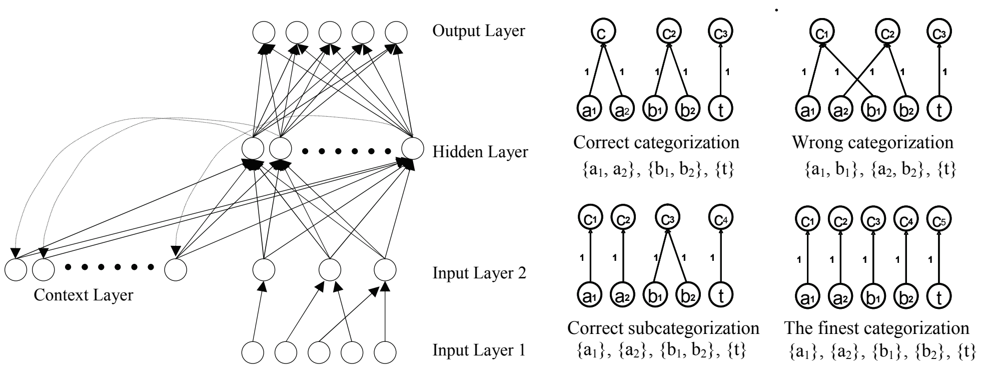


Fig. 4. Architecture experimented (left) and categorization examples (right).

A localist representation or the so-called one-hot vector (a binary vector with a single 1) is adopted for input words as well as categories. The connections between Input Layer 1 and Input Layer 2 convert the former to the latter. These connections are set and fixed to 1 or 0 according to the conversion, because a word belongs to a category or not (the corresponding weight is 1 or 0, respectively). As shown in Fig. 4 (left), the information that the SRN part of the network would know is limited to the categories of the input words.

In the current experiment, the category refers to a mutually exclusive partition of words. Since words and categories are represented by localist representation, each node in Input Layer 1 has exactly one outgoing connection to Input Layer 2, and each node in Input Layer 2 has incoming connections from the nodes designating words of the category. In other words, in this experiment, we suppose that the word in the example sentences belongs to only one category or is derived from at most one non-terminal symbol in derivation. Examples of the categorization are shown in Fig. 4 (right).

As many networks as the possible categorizations were prepared. Then, all the networks were trained while observing their learning errors. It must be noted that the network is robust to overtraining and that a network with the least learning error might possibly be a network with the least generalization error because the networks accept only categorized inputs so that a network with incorrect categorization connections receives inconsistent learning data.

4.2 Experimental settings: target language and learning method

In the current experiments, the task is to learn a diversified centre-embedded language (DCEL) {abt, aabbt, aaabbbt | a = a_1 or a_2, b = b_1 or b_2} (Bodén, 2004), where "t" is the terminal of a string and "a" and "b" correspond to categories. Hereafter, the terms character and string are used instead of word and sentence; further, the density of a category is the number of terminal symbols derived from a non-terminal symbol or equivalently the number of elements or characters belonging to the category; for example, the density of "a" in the above example is 2. The length of the language is the length of the longest sequence in the language (excluding the termination symbol "t"); hence, the length of the language in the above example is 6.

A simple error-backpropagation algorithm with the squared error function is adopted for the learning as in SRN experiments. The learning rate is 0.10 and no acceleration is used.

The generalization capability of the learned networks is evaluated by the Kullback-Leibler divergence (KLd) between two distributions: expectation and realized network output. The former distribution is obtained by supposing the uniform distribution for the above three strings and the characters in the same categories (e.g. {a_1,a_2}). {1/4, 1/4, 1/4, 1/4, 0} is an example distribution for {a_1,a_2,b_1,b_2,t} that appears after a string a_1a_2 or a_2a_1. The latter distribution is obtained by normalizing the network outputs to 1. It must be noted that KLd is used only for evaluation and never used for learning.
The learning data were obtained by first randomly generating a set of strings of terminal symbols, and then randomly re-sampling from the set until the due number of strings is obtained. 90,000 strings were used for each run of training and 10,000 strings were used to calculate the resultant learning error and the generalization error.
15 training sessions for each network were conducted by changing the learning data and its presentation order; the average and the standard deviation of the resultant learning and generalization errors were obtained. Categorizations were varied for each network.
The parameters were varied; the length of the language was 6, 8, and 10; the number of training data was 30 and 60; the density of category was 2, 3, 4, and 5.

4.3 Results

Figs. 5 and 6 summarize some of the results obtained; L denotes the length of the language and D denotes the density of the correct category for "a" and "b" (set equal); Nc denotes the number of categories including "t"; N denotes the number of learning examples. An error bar shows the standard deviation averaged over experiments.

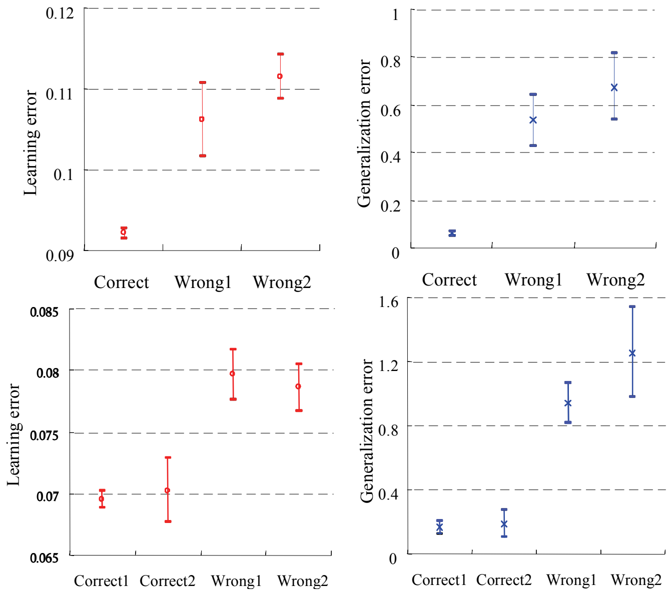


Fig. 5. Top two figures. L = 6, D = 2, Nc = 3, and N = 60. On the horizontal axis, Correct is {a_1,a_2}, {b_1,b_2}, {t}, Wrong1 is {a_1}, {a_2,b_1,b_2}, {t} and Wrong2 is {a_1,b_1}, {a_2,b_2}, {t}. Bottom two figures. L = 8, D = 4, Nc = 4, and N = 60. On the horizontal axis, Correct1 is {a1,a2,a3,a4}, {b_1,b_2}, {b_3,b_4}, {t}, Correct2 is {a_1,a_2,a_3,a_4}, {b_1,b_2,b_3}, {b_4}, {t}, Wrong1 is {a_1,a_2,b_1}, {a_3,a_4,b_2}, {b_3,b_4}, {t} and Wrong2 is {a_1,a_2,b_3,b_4}, {a_1,b_1}, {a_2,b_2}, {t}

The correct categorization consistently gives the smallest of the resultant learning and generalization errors. The correct sub-categorization also gives the smallest values as shown in Fig. 5 (bottom). It is observed that the two different sub-categorizations give the similar learning errors and generalization errors.
In Fig. 6, some results of the experiments for categories with higher density are shown. The difference between the correct categories and the wrong categories in these cases is clearer than that that found in the cases in Fig. 5.

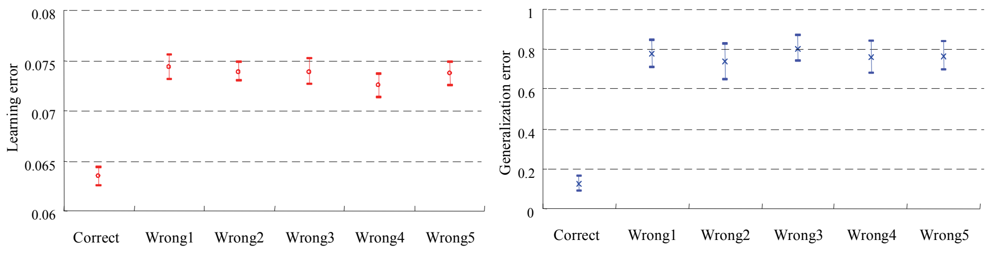


Fig. 6. L = 10, D = 5, Nc = 3, and N = 60. On the horizontal axis, Correct is $\{a_1,a_2,a_3,a_4,a_5\}$, $\{b_1,b_2,b_3,b_4,b_5\}$, {t}; Wrong1 is $\{a_1,a_2,a_3,b_1,b_2,b_3\}$, $\{a_4,a_5,b_4,b_5\}$, {t}; Wrong2 is $\{a_1,a_2,b_3,b_4,b_5\}$, $\{a_3,a_4,a_5,b_1,b_2\}$, {t}; Wrong3 is $\{a_1,a_2,b_1,b_2\}$, $\{a_3,a_4,a_5,b_3,b_4,b_5\}$, {t}; Wrong4 is $\{a_1,a_2\}$, $\{a_3,a_4,a_5,b_1,b_2,b_3,b_4,b_5\}$, {t}; Wrong5 is $\{a_1,b_1\}$, $\{a_2,a_3,a_4,a_5,b_2,b_3,b_4,b_5\}$, {t}.

4.4 Discussion on systematicity

Networks with systematicity that conforms to the systematicity in the target language can be differentiated from the networks without it, because networks with correct categorizations or sub-categorizations can be found by referring to their learning error, simple and observable quantity, as stated in the previous section.
To acquire word meanings (or the extent of category of objects/movements/events that a word means), it has been argued that humans use a lot of tactics that constrain generalizations to an appropriate level (e.g., (Markman, 1990)). It is noteworthy that no workable model is possible if we require that humans should interact with external environments to use the tactics.
Acquisition of grammatical categories of word is a different problem than acquisition of word meanings, because grammatical categories exist only for grammars, which suggests that humans use different tactics. Pinker suggests that humans utilize bootstrapping interactions between syntax and semantics development as a key factor (Pinker, 1984). However, if the semantics should concern with external environments, no workable model is possible.
Conventional approaches have not yet provided satisfactory results. For example, although distributional clustering is one of the promising approaches, the clusters corresponding to grammatical categories that the algorithm provides are unsatisfactory (Clark, 2000). However, if we could utilize the fact that grammatical categories exist only for grammars, as in the case, it may be possible to induce grammatical categories solely from written texts.
In the current experiments, based on these observations, we proposed to hypothesize the existence of innate systems of categorizations. Although the method and results reported here are still very primitive, they suggest that the hypothesis is promising.

5. Conclusion

Productivity is the key property of a natural language. Learnability is an equally important property, since productivity without learnability will not help the transfer of a language beyond generations. Linguistic productivity is supported by recursiveness described in terms of phrasal categories and systamaticity described in terms of lexical categories.
Recursiveness is realizable by an SRN; however, the realized function is limited to a counter capable of counting up and down just once. Hence, for example, parsing consecutive embedded sentential phrases requires a reset of counters or stacks. The difficulty of learning of recursiveness is observed experimentally and theoretically.
Human learns recursive grammar in his/her first language which requires stacks to parse. We have to determine an alternative mechanism or appropriate a priori knowledge along with an additional method that embeds the knowledge in the learning mechanism.
Systematicity is learnable with an RNN by using added sub-networks, which is equivalent to one-stage learning algorithms composed of clustering of words into categories and learning of grammars written in lexical (and phrasal) categories. Learnability is observed when frequency information that exist in training corpus and influence learning results is not transferred to the SRN; this shows the possibility of grammatical induction when systematicity exists.
The difficulty in learning systematicity in more realistic situations is rooted in ambiguous words and varied contexts. An ambiguous word is a word, for example, which could be a noun in a sentence but a verb in another sentence. Varied contexts are contexts which induce the same category but have very different forms, for example, a noun could be followed by almost any lexical category.
Research is being conducted on the learnability of productive grammars.

6. References

Bodén, M.; Wiles, J.; Tonkes, B. & Blair, A. D. (1999). Learning to predict a context-free language: Analysis of dynamics in recurrent hidden units, *Proceedings of ICANN'99*, pp. 359-364, Edinburgh, 1999

Bodén, M. & Wiles, J. (2000). Context-free and context-sensitive dynamics in recurrent neural networks, *Connection Science*, Vol. 12, No. 3-4, (Dec. 2000), pp. 197-210

Bodén, M. & Blair, A. (2003). Learning the dynamics of embedded clauses, *Applied Intelligence*, Vol. 19, No. 1-2, (July 2003), pp. 51-63

Bodén, M. (2004). Generalization by symbolic abstraction in cascaded recurrent networks, *Neurocomputing*, Vol. 57, pp. 87-104

Casey, M. (1998). Correction to proof that recurrent neural networks can robustly recognize only regular languages, *Neural Computation*, Vol. 10, No. 5, (July 1998), pp. 1067-1069

Chalup, S. K.. & Blair, A. D. (2003). Incremental training of first order recurrent neural networks to predict a context-sensitive language, *Neural Networks*, Vol. 16, pp. 955-972

Chomsky, N. (1959a). A Review of B.F. Skinners verbal behavior, *Language*, Vol. 35, pp. 26-58, Reprinted in: Chomsky, N. (1965). *Aspects of the theory of syntax*, M.I.T. Press, Cambridge, Mass

Chomsky, N. (1959b). On certain formal properties of grammars, *Information and Control*, Vol. 2, pp. 137–167

Chomsky, N. (1980). *Rules and representations*, Columbia University Press, New York

Christiansen, M. H. & Chater, N. (1994). Generalization and connectionist language learning, *Mind and Language*, Vol. 9, pp. 273-287

Clark, A. (2000). Inducing syntactic categories by context distribution clustering, *Proceedings of CoNLL-2000 and LLL-2000*, Lisbon, Portugal, (Sept., 2000)

Elman, J. L. (1990). Finding structure in time, *Cognitive Science*, Vol. 14, pp. 179-211

Elman, J. L. (1991). Distributed representations, simple recurrent networks, and grammatical structure, *Machine Learning*, Vol. 7, pp. 195-224

Elman, J. L. (1995). Language as a dynamical system, In: *Mind as Motion: Explorations in the Dynamics of Cognition*, Port, R. F. & van Gelder, T. (Eds.), pp. 195—223, MIT Press, Cambridge, MA

Farkaš, I. & Crocker, M. W. (2008). Syntactic systematicity in sentence processing with a recurrent self-organizing network, *Neurocomputing*, Vol. 71, No. 7-9, pp. 1172-1179

Fodor, J. A. & Pylyshyn, Z. (1988). Connectionism and cognitive architecture: a critical analysis, *Cognition*, Vol. 28, pp. 3-71

Frank, S. L. (2006). Learn more by training less: Systematicity in sentence processing by recurrent networks, *Connection Science*, Vol. 18, pp. 287-302

Gers, F. A. & Schmidhuber, J. (2001). LSTM Recurrent Networks Learn Simple Context Free and Context Sensitive Languages, *IEEE Transactions on Neural Networks*, Vol. 12, No. 6, pp. 1333-1340

Gold, E. M. (1967). Language identification in the limit, *Information and Control*, Vol. 10, No. 5, pp. 447-474.

Guckenheimer, J. & Holmes, P. (1997). *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*, Corr. 5th print., Applied Mathematical Sciences, 42, Springer

Hadley, R. (1994). Systematicity in connectionist language learning, *Mind and Language,* Vol. 9, No. 3, pp. 247–272

Hopcroft, J. & Ullman, J. (1979). *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley

Iwata, A.; Shinozawa, Y. & Sakurai, A. (2007). A Characterization of simple recurrent neural networks as a language recognizer, *Proceedings of 14th International Conference on Neural Information Processing*, (Nov. 2007), Kitakyushu, Japan

Katok, A. & Hasselblatt, B. (1996). *Introduction to the Modern Theory of Dynamical Systems*, Cambridge University Press

Maass, W. & Orponen, P. (1998). On the effect of analog noise in discrete-time analog computations, *Neural Computation*, vol. 10, pp. 1071-1095

Markman, E. M. (1990). Constraints children place on word meanings, *Cognitive Science*, Vol. 14, pp. 154-173

Pinker, S. (1984). *Language learnability and language development*, Harvard University Press, (1984), Cambridge

Rodriguez P.; Wiles J. & Elman J. L. (1999). A Recurrent neural network that learns to count, *Connection Science*, Vol. 11, No. 1, (March 1999), pp. 5-40

Rodriguez, P. (2001). Simple recurrent networks learn context-free and context-sensitive languages by counting, *Neural Computation*, Vol. 13, No. 9, pp. 2093–2118

Schmidhuber, J.; Gers, F. & Eck D. (2002). Learning nonregular languages: a comparison of simple recurrent networks and LSTM, *Neural Computation*, Vol. 14, No. 9, (Sept. 2002), pp. 2039-2041

Schütze, H. (1993). Part-of-speech induction from scratch. *Proceedings of ACL 31*, pp. 251–258, Ohio State University

Siegelmann, H. T. (1998). *Neural Networks and Analog Computation: Beyond the Turing Limit*, Birkhauser, ISBN 0-8176-3949-7, Boston

Suhara, Y. & Sakurai, A. (2006). Generalization by categorical nodes in recurrent neural networks, In: *International Congress Series*, Vol. 1291, pp. 161-164

Van derVelde, F.; Van derVoort van der Kleij, G. T. & De Kamps, M. (2004). Lack of combinatorial productivity in language processing with simple recurrent networks, *Connection Science*, Vol. 16, pp. 21–46

Wiles, J.; Blair, A. D. & Bodén, M. (2001). Representation beyond finite states: alternatives to push-down automata. In: A Field Guide to Dynamical Recurrent Networks, Kolen, J.F. & Kremer, S.C. (Eds.), pp. 129-142, IEEE Press, New York

Recurrent Neural Network Identification and Adaptive Neural Control of Hydrocarbon Biodegradation Processes

Ieroham Baruch[1], Carlos Mariaca-Gaspar[1], and Josefina Barrera-Cortes[2]
CINVESTAV-IPN, Mexico City, [1]Department of Automatic Control,
[2]Department of Biotechnology & Bioengineering,
Mexico

1. Introduction

The Recent advances in understanding of the working principles of artificial neural networks has given a tremendous boost to identification, prediction and control tools of nonlinear systems, (Narendra & Parthasarathy, 1990; Chen & Billings, 1992; Hunt et al., 1992; Miller III et al., 1992; Pao et al., 1992; Su et al., 1992; Narendra & Mukhopadhyay, 1994; Boskovic & Narendra, 1995; Ku & Lee, 1995; Baruch et al., 1996; Jin & Gupta, 1999, Haykin, 1999; Mastorocostas & Theocharis, 2006; Kazemy et all., 2007). The main network property namely the ability to approximate complex non-linear relationships without prior knowledge of the model structure makes them a very attractive alternative to the classical modeling and control techniques. This property has been proved by the universal approximation theorem, (Haykin, 1999). Among several possible network architectures the ones most widely used are the feedforward and the recurrent neural networks. In a feed-forward neural network the signals are transmitted only in one direction, starting from the input layer, subsequently through the hidden layers to the output layer, which requires applying a tap delayed global feedbacks and a tap delayed inputs to achieve a nonlinear autoregressive moving average neural dynamic plant model. A recurrent neural network has local feedback connections to some of the previous layers. Such a structure is suitable alternative to the first one when the task is to model dynamic systems, and the universal approximation theorem has been proved for the recurrent neural networks too. The preferences given to recurrent neural network identification with respect to the classical methods of process identification are clearly demonstrated in the solution of the "bias-variance dilemma", (Haykin, 1999). Furthermore, the derivation of an analytical plant model, the parameterization of that model and the Least Square solution for the unknown parameters have the following disadvantages: (a) the analytical model did not include all factors having influence to the process behavior; (b) the analytical model is derived taking into account some simplifying suppositions which not ever match; (c) the analytical model did not described all plant nonlinearities, time lags and time delays belonging to the process in hand; (d) the analytical model did not include all process and measurement noises which are sensor and actuator dependent. In (Sage, 1968) the method of invariant imbedding has been described. This method seemed to be a universal tool for simultaneous state and

parameter estimation of nonlinear plants but it suffer for the same drawbacks because a complete nonlinear plant model description is needed.
So, the unknown nonlinear technological processes needed a new tool for modeling and identification capable to correlate experimental data and to estimate parameters and states in the same time, processing noisy measurements. Such efficient tool is the recurrent neural Kalman filter, where the estimated parameters and states could be used directly for control.

2. Description of the recurrent neural Kalman filter

2.1 Topology and learning of the recurrent trainable neural network

The Recurrent Trainable Neural Network (RTNN) topology, given on Fig. 1, is a hybrid one. It has one recurrent hidden layer and one feedforward output layer. This topology is inspired from the Jordan canonical form of the state-space representation of linear dynamic systems (Baruch et al., 1996) adding activation functions to the state and the output equations so to convert it to recurrent neural network named Recurrent Trainable Neural Network described by the equations:

$$X(k+1) = AX(k) + BU(k) \tag{1}$$

$$B = [B_1 ; B_0];\ U^T = [U_1^T ; U_2^T] \tag{2}$$

$$A = \text{block-diag}\ (Ai),\ |Ai\ | < 1 \tag{3}$$

$$Z_1(k) = G[X(k)] \tag{4}$$

$$C = [C_1 ; C_0];\ Z^T = [Z_1^T ; Z_2^T] \tag{5}$$

$$V(k) = CZ(k) \tag{6}$$

$$Y(k) = F[V(k)] \tag{7}$$

Where: X, Y, U are vectors of state, output, and augmented input with dimensions N, L, (M+1), respectively, Z is an (L+1) –dimensional input of the feedforward output layer, where Z_1 and U_1 are the (Nx1) output and (Mx1) input of the hidden layer; the constant scalar threshold entries are $Z_2 = -1$, $U_2 = -1$, respectively; V is a (Lx1) pre-synaptic activity of the output layer; the super-index T means vector transpose; A is (NxN) block-diagonal weight matrix; B and C are [Nx(M+1)] and [Lx(N+1)]- augmented weight matrices; B_0 and C_0 are (Nx1) and (Lx1) threshold weights of the hidden and output layers; F[.], G[.] are vector-valued tanh(.) or sigmoid(.) -activation functions with corresponding dimensions.
The RTNN topology has been derived independently of the hybrid Diagonal Recurrent Neural Network (DRNN) (see Ku & Lee, 1995) with which it have the following differences: (a) the state equation (1) of the RTNN is linear and the state equation of the DRNN is nonlinear (the activation functions are in the closed loop). This made the RTNN completely controllable and the state, and parameter information X, A, B of RTNN directly applicable for state-space control purposes. On the other side, the controllability of the DRNN depends on the type of the activation functions (see Sontag & Sussmann, 1997); (b) the state weight matrix A of the RTNN is defined as block- diagonal (3) and some stability bounds have been imposed to it which preserved the RTNN stability during the learning. The DRNN was defined as block-diagonal later (Mastorocostas & Theocharis, 2006; Kazemy et al., 2007) and

some algorithmic measures have been taken to maintained the stability of DRNN during the learning. For the RTNN the learning of the Jordan blocks is resolved in universal manner, defining diagonal and full-matrix Backpropagation (BP) learning options; (c) the RTNN include thresholds in the inputs of both layers which facilitated the nonlinear systems identification, especially in lack of a-priory information about the approximated nonlinear plant. The DRNN did not apply thresholds; (d) the output layer of the DRNN is linear, and that of the RTNN is nonlinear, which permitted it to perform better approximation of nonlinear plants. Furthermore, depending on the plant structure, the topology of the RTNN could be extended with additional feedforward output or input layers which augmented the approximation ability of the RTNN. The observability of the DRNN has been proved by (Albertini & Sontag, 1994). The observability of the RTNN is assumed and it is fulfilled when the reference signal entered in the limits of the activation functions. The dynamic BP algorithm of RTNN learning is derived using the adjoint RTNN topology, predicting the output error (see Fig. 2). The adjoint RTNN is built applying the Separation theorem (Sage, 1968) and the diagrammatic method of (Wan & Beaufays, 1996). The BP algorithm is:

$$W(k+1) = W(k) + \eta\, \Delta W(k) + \alpha\, \Delta W(k-1);\ |W_{ij}| < W_0 \tag{8}$$

$$E(k) = Y_d(k) - Y(k);\ E_1(k) = F'[Y(k)]\, E(k) \tag{9}$$

$$F'[Y(k)] = [1-Y^2(k)] \tag{10}$$

$$\Delta C(k) = E_1(k)\, Z^T(k) \tag{11}$$

$$E_3(k) = G'[Z(k)]\, E_2(k);\ E_2(k) = C^T(k)\, E_1(k) \tag{12}$$

$$G'[Z(k)] = [1-Z^2(k)] \tag{13}$$

$$\Delta B(k) = E_3(k)\, U^T(k) \tag{14}$$

$$\Delta A(k) = E_3(k)\, X^T(k) \tag{15}$$

$$Vec(\Delta A(k)) = E_3(k) \square X(k) \tag{16}$$

Where: F'[.], G'[.] are derivatives of the tanh(.) functions; W is a general weight, denoting each weight matrix (C, A, B) in the RTNN model, to be updated; ΔW (ΔC, ΔA, ΔB), is the weight correction of W; Y_d is an L-dimensional output of the approximated plant taken as a reference for RTNN learning; η, α are learning rate parameters; ΔC is a weight correction of C; ΔB is a weight correction of B; ΔA is a weight correction of A; the diagonal of the matrix A is denoted by Vec (A(k)) where (16) represents its learning as an element-by-element vector product; E, E_1, E_2, E_3, are error vectors (see Fig. 2).

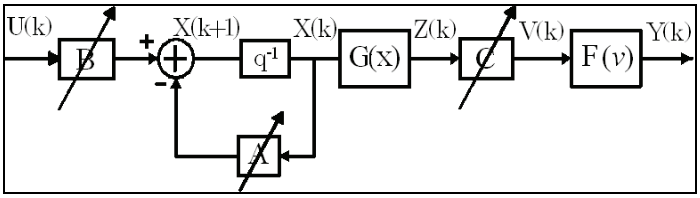


Fig. 1. Block diagram of the RTNN model

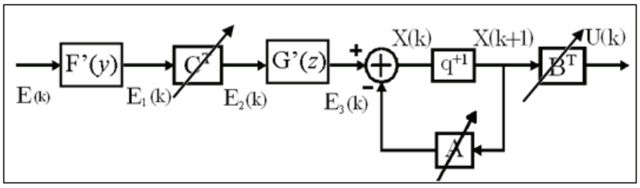


Fig. 2. Block diagram of the adjoint RTNN model

The dimension of the state vector X of the RTNN is chosen using the simple rule of thumb which is: N=L+M. The initial values of the weight matrices during the learning are chosen as arbitrary numbers inside a small range. The stability of the RTNN model is assured by the activation functions [-1, 1] bounds and by the local stability weight bound conditions given by (3), (8). The stability of the RTNN movement around the optimal weight point will be proven extending the proof of (Nava et al., 2004), as it is stated below.

Theorem of stability of the RTNN. Let the RTNN with Jordan Canonical Structure is given by equations (1)-(7) (see Fig.1) and the nonlinear plant model, is as follows:

$$X_{d.}(k+1) = G[\,X_d(k), U(k)\,] \tag{17}$$

$$Y_d(k) = F[\,X_d(k)\,] \tag{18}$$

Where: $\{Y_d(.), X_d(.), U(.)\}$ are output, state and input variables with dimensions L, N_d, M, respectively; F(.), G(.) are vector valued nonlinear functions with respective dimensions. Under the assumption of RTNN identifiability made, the application of the BP learning algorithm for A(.), B(.), C(.), in general matricial form, described by equation (8)-(16), and the learning rates η (k), α (k) (here they are considered as time-dependent and normalized with respect to the error) are derived using the following Lyapunov function:

$$L(k) = L_1(k)+L_2(k) \tag{19}$$

Where: $L_1(k)$ and $L_2(k)$ are given by:

$$L_1(k) = \tfrac{1}{2}e^2(k)$$

$$L_2(k) = tr\left(\tilde{W}_A(k)\tilde{W}_A^T(k)\right)+tr\left(\tilde{W}_B(k)\tilde{W}_B^T(k)\right)+tr\left(\tilde{W}_C(k)\tilde{W}_C^T(k)\right)$$

Where:

$$\tilde{W}_A(k) = \hat{A}(k) - A^*, \tilde{W}_B(k) = \hat{B}(k) - B^*, \tilde{W}_C(k) = \hat{C}(k) - C^*$$

are vectors of the parameter estimation error and (A^*, B^*, C^*) and $(\hat{A}(k), \hat{B}(k), \hat{C}(k))$ denote the ideal neural weight and the estimate of the neural weight at the k-th step, respectively, for each case.

Then the identification error is bounded, i.e.:

$$\begin{aligned} L(k+1) &= L_1(k+1)+L_2(k+1) < 0 \\ \Delta L(k+1) &= L(k+1) - L(k) \end{aligned} \tag{20}$$

Where the condition for $L_1(k+1)<0$ is that:

$$\frac{\left(1-\frac{1}{\sqrt{2}}\right)}{\psi_{\max}} < \eta_{\max} < \frac{\left(1+\frac{1}{\sqrt{2}}\right)}{\psi_{\max}}$$

and for $L_2(k+1)<0$ we have:

$$\Delta L_2(k+1) < -\eta_{\max}|e(k+1)|^2 - \alpha_{\max}|e(k)|^2 + d(k+1)$$

Note that $\eta_{\max}$ changes adaptively during the RTNN learnig and:

$$\eta_{\max} = \max_{i=1}^{3}\{\eta_i\}$$

Where all: the unmodelled dynamics, the approximation errors and the perturbations, are represented by the d-term, and the complete proof of that theorem, is given in Apendix A. The Rate of Convergence Lemma, used, is given in (Nava et al., 2004).

2.2 Topology and learning of the Kalman filter recurrent neural network

Let us consider the linearized plant model (17), (18), represented in a state-space form:

$$X_{d.}(k+1) = A_d(k)\,X_d(k) + B_d(k)\,U(k) + \Theta_1(k) \tag{21}$$

$$Y_d(k) = C_d(k)\,X_d(k) + \Theta_2(k) \tag{22}$$

Where: E [.] means mathematical expectation; the process and measurement noises Θ_1 (.), Θ_2 (.) are white, with $\Theta_1(k)$, Θ_2 (s) and the initial state X_d (k_0) independent and zero mean for all k, s, with known variances E [X_d (k) X_d^T (k)] = P_0 , E[$\Theta_1(k)$ Θ_1^T (k)] = Q(k) δ (k-τ), E[$\Theta_2(k)$ Θ_2^T (k)] = R(k) δ (k-τ), where δ (k-τ)=1 if k= τ, and 0 otherwize. The optimal Kalman filter theory is completely described in (Sage, 1968) and we would not repeated it here.

For us the Kalman Filter (KF) is a full rank optimal state estimator capable to estimate the systems state, to filter the process and measurement noises taking in hand all plant information available like: input/output plant data, all parameters of the plant model (21), (22), and the given up noise and initial state statistics (mean and variance). The basic Kalman filter equations for the estimated state and output variables are given by:

$$X_{e.}(k+1) = A_e(k)\,X_e(k) + K_e(k)\,Y_d(k) + B_d(k)\,U(k) \tag{23}$$

$$A_e(k) = A_d(k) - K_e(k)\,C_d(k) \tag{24}$$

$$Y_e(k) = C_d(k)\,X_e(k) \tag{25}$$

Where: X_e (k) is the estimated state vector with dimension N_e; A_e (k) is a (N_e x N_e) closed-loop KF state matrix; Y_e (k) is the estimated plant output vector variable with dimension L; $K_e(k)$ is the optimal Kalman filter gain matrix with dimension (N_e x L). This gain matrix is computed applying the optimal Kalman filtering methodology given in (Sage, 1968). So, the KF performed noise filtration by means of an optimal closed-loop feedback which has the drawback that the feedback amplified the noise components of the error, especialy when the

feedback gain is high. The second draback is that the KF design needs a complete plant parameter and noise information, which means that if the plant data are incomplete the process noise level is augmented. To overcome this we need to take special measures like to augment the filtering capabilities of the KF.
So, the Kalman filter could not estimate parameters and states in the same time processing noisy measuremets with unknown noise statistics, and it will be our task. To resolve this task we need to derive the topology and the BP learning algorithm of a new recurrent Kalman filter neural network. First of all we could rewrite the equation (23) defining a new extended input vector, containing all available input/output information issued by the plant, and second – we could modify the output equation (25) so to convert it to an output noise filter. After that we obtain:

$$X_{e.}(k+1) = A_d(k)\, X_e(k) - K_e(k)\, Y_e(k) + B_2(k)\, U_e(k) \tag{26}$$

$$B_2 = [B_d\, ;\, K_e];\; U_e^T = [U\, ;\, Y_d] \tag{27}$$

$$Z(k) = C_d(k)\, X_e(k) \tag{28}$$

$$Y_e(k+1) = A_2\, Y_e(k) + Z(k) \tag{29}$$

Now comparing the equations (26)-(29) with the RTNN topology (1)-(7) we decided to extend the RTNN topology adding local and global feedbacks so that to fulfil KF requirements. The obtained new Kalman Filter Recurrent Neural Network (KF RNN) topology is given on Fig. 3, where the first layer represented the plant model, the second layer represented the output noise filtering model, and it has also a global output feedback. The KF RNN topology is described by the equations:

$$X(k+1) = A_1X(k) + BU(k) - DY(k) \tag{30}$$

$$B = [B_1\, ;\, B_0];\; U^T = [U_1\, ;\, U_2] \tag{31}$$

$$A_1 = \text{block-diag}\,(A_{1,i}),\; |\, A_{1,i}\, | < 1 \tag{32}$$

$$Z_1(k) = G[X(k)] \tag{33}$$

$$C = [C_1\, ;\, C_0];\; Z^T = [Z_1\, ;\, Z_2] \tag{34}$$

$$V_1(k) = CZ(k) \tag{35}$$

$$V(k+1) = V_1(k) + A_2V(k) \tag{36}$$

$$A_2 = \text{block-diag}\,(A_{2,i}),\; |\, A_{2,i}\, | < 1 \tag{37}$$

$$Y(k) = F[V(k)] \tag{38}$$

Where: X, Y, U are vectors of state, output, and augmented input with dimensions N, L, (M+1), respectively, Z is an (L+1) –dimensional input of the feedforward output layer, where Z_1 and U_1 are the (Nx1) output and (Mx1) input of the hidden layer; the constant scalar threshold entries are $Z_2 = -1$, $U_2 = -1$, respectively; V is a (Lx1) pre-synaptic activity of the output layer; the super-index T means vector transpose; A_1, A_2 are (NxN) and (LxL) block-diagonal weight matrices; B and C are [Nx(M+1)] and [Lx(N+1)]- augmented weight matrices; B_0 and C_0 are (Nx1) and (Lx1) threshold weights of the hidden and output layers; F[.], G[.] are vector-valued tanh(.) or sigmoid(.) -activation functions with corresponding dimensions. Here the input vector U and the input matrix B of the KF RNN are augmented so to fulfill the specifications (27) and the matrix D corresponded to the feedback gain matrix of the KF. So the KF RNN topology corresponded functionally to the KF definition (26)-(29) and ought to be learnt applying the BP learning algorithm which is in fact an unrestricted optimization procedure, derived using the adjoint KF RNN (see Fig.4) for KF RNN topology, applying the Separation theorem (Sage, 1968) and the diagrammatic method (Wan & Beaufays, 1996). The BP learning algorithm, expressed in vector-matricial form is:

$$W(k+1) = W(k) + \eta\, \Delta W(k) + \alpha\, \Delta W(k-1);\ |W_{ij}| < W_0 \tag{39}$$

$$E(k) = Y_d(k) - Y(k);\ E_1(k) = F'[Y(k)]\, E(k) \tag{40}$$

$$F'[Y(k)] = [1 - Y^2(k)] \tag{41}$$

$$\Delta C(k) = E_1(k)\, Z^T(k) \tag{42}$$

$$\Delta A_2(k) = E_1(k)\, V^T(k) \tag{43}$$

$$\mathrm{Vec}(\Delta A_2(k)) = E_1(k) \square X(k) \tag{44}$$

$$E_3(k) = G'[Z(k)]\, E_2(k);\ E_2(k) = C^T(k)\, E_1(k) \tag{45}$$

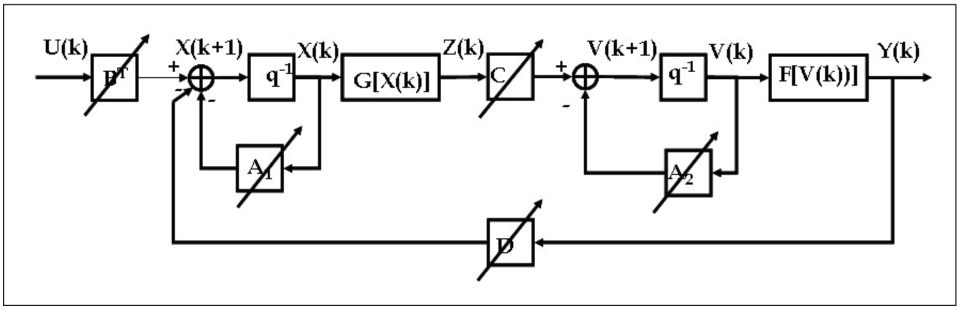


Fig. 3. Block diagram of the KF RNN model

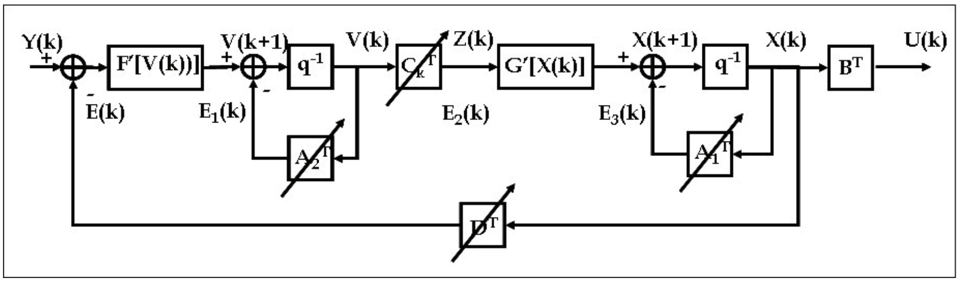


Fig. 4. Block diagram of the adjoint KF RNN model

$$G'[Z(k)] = [1-Z^2(k)] \tag{46}$$

$$\Delta B(k) = E_3(k)\, U^T(k) \tag{47}$$

$$\Delta D(k) = E_3(k)\, Y^T(k) \tag{48}$$

$$\Delta A_1(k) = E_3(k)\, X^T(k) \tag{49}$$

$$\mathrm{Vec}(\Delta A_1(k)) = E_3(k) \square X(k) \tag{50}$$

Where: F'[.], G'[.] are derivatives of the tanh(.) functions; W is a general weight, denoting each weight matrix (C, A_1, A_2, B, D) in the KF RNN model, to be updated; ΔW (ΔC, ΔA_1, ΔA_2, ΔB, ΔD), is a weight correction of W; Y_d is an L-dimensional output of the approximated plant taken as a reference for KF RNN learning; η, α are learning rate parameters; ΔC is a weight correction of C; ΔB is a weight correction of B; ΔD is a weight correction of D, ΔA_1 is a weight correction of A_1 , ΔA_2 is a weight correction of A_2; the diagonals of the matrices A_1, A_2 are denoted by Vec ($A_1(k)$), Vec ($A_2(k)$), respectively, where (44), (50) represented their learning as an element-by-element vector products; E, E_1, E_2, E_3, are error vectors (see Fig. 4), predicted by the adjoint KF RNN model.
So, the KF RNN is capable to issue parameter and state estimations for control purposes, thanks to the optimization capabilities of the BP learning algorithm, applying the "correction for error" delta rule of learning (see Haykin, 1999). The stability of the KF RNN model is assured by the activation functions [-1, 1] bounds and by the local stability weight bound conditions given by (32), (37). The stability of the KF RNN movement around the optimal weight point has been proved by one theorem and the Rate of Convergence Lemma (see Nava et al., 2004), following the same way as for the RTNN. It is stated below.
Theorem of stability of the KF RNN. Let the KF RNN is given by equations (30)-(38) (see Fig.3) and the nonlinear plant model, is given by equations (17), (18). Under the assumption of KF RNN identifiability made, the application of the BP learning algorithm for C, A_1, A_2, B, D, in general matricial form, described by equation (39)-(50), and the learning rates η (k), α (k) (here they are considered as time-dependent and normalized with respect to the error) are derived using the following Lyapunov function:

$$L(k) = L_1(k)+L_2(k) \tag{51}$$

Where: $L_1(k)$ and $L_2(k)$ are given by:

$$L_1(k) = \tfrac{1}{2}e^2(k)\,;$$

$$L_2(k) = \mathrm{tr}\left(\tilde{W}_{A1}(k)\tilde{W}_{A1}^T(k)\right)+ \mathrm{tr}\left(\tilde{W}_{A2}(k)\tilde{W}_{A2}^T(k)\right)+\mathrm{tr}\left(\tilde{W}_{B}(k)\tilde{W}_{B}^T(k)\right)+\mathrm{tr}\left(\tilde{W}_{C}(k)\tilde{W}_{C}^T(k)\right)$$
$$+\mathrm{tr}\left(\tilde{W}_{D}(k)\tilde{W}_{D}^T(k)\right)$$

Where:
$\tilde{W}_{A1}(k) = \hat{A}_1(k)-A_1^*, \tilde{W}_{A2}(k) = \hat{A}_2(k)-A_2^*, \tilde{W}_B(k) = \hat{B}(k)-B^*, \tilde{W}_C(k) = \hat{C}(k)-C^*, \tilde{W}_D(k) = \hat{D}(k)-D^*$

are vectors of the estimation error and $(A_1^*,A_2^*,B^*,C^*,D^*)$ and $(\hat{A}_1(k),\hat{A}_2(k),\hat{B}(k),\hat{C}(k),\hat{D}(k))$ denote the ideal optimal neural weight and the estimate of the neural weight at the k-th step, respectively, for each case.

Then the identification error is bounded, i.e.:

$$\begin{aligned} L(k+1) &= L_1(k+1)+L_2(k+1)<0 \\ \Delta L(k+1) &= L(k+1) - L(k) \end{aligned} \tag{52}$$

Where the condition for $L_1(k+1)<0$ is that

$$\frac{\left(1-\frac{1}{\sqrt{2}}\right)}{\psi_{max}}<\eta_{max}<\frac{\left(1+\frac{1}{\sqrt{2}}\right)}{\psi_{max}}$$

and for $L_2(k+1)<0$ we have:

$$\Delta L_2(k+1)< -\eta_{max}\left|e(k+1)\right|^2 - \alpha_{max}\left|e(k)\right|^2+d(k+1)$$

Note that η_{max} changes adaptively during learnig process of the network and

$$\eta_{max}=\max_{i=1}^{5}\{\eta_i\}$$

Where all: the unmodeled dynamics, the approximation errors and the perturbations, are represented by the d-term, and the complete proof of that theorem can be easily obtained following the same procedure detailed in Appendix A derived for the RTNN.

3. Description of the adaptive control schemes

3.1 Indirect adaptive control scheme (sliding mode control)

The indirect adaptive control using the RTNN as plant identifier has been described in (Baruch et al., 2001a; Baruch et al., 2001b; Baruch et al., 2005). Later the proposed indirect control has been derived as a Sliding Mode Control (SMC) and some preliminary results of SMC of unknown hydrocarbon biodegradation processes have been reported (see Baruch et al., 2007a; Baruch et al., 2007b). The block diagram of the indirect adaptive control scheme is shown on Fig. 5. It contained identification and state estimation KF RNN and a sliding mode controller. The stable nonlinear plant is identified by a KF RNN model with topology, given by equations (30)-(38) learned by the stable BP-learning algorithm, given by equations (39)-(50), where the identification error tends to zero. The simplification and linearization of the neural identifier equations (30)-(33), omitting the DY(.) and $K_eY_d(.)$, (27) parts, leads to the next local linear plant model, extracted from the complete KF RNN model:

$$X(k+1) = A_1X(k) + BU(k) \tag{53}$$

$$Z(k) = H\,X(k);\; H = C\,G'(Z) \tag{54}$$

Where G'(.) is the derivative of the activation function and L = M, is supposed.
In (Young et al., 1999), the sliding surface is defind with respect to the state variables and the SMC objective is to move the states form an arbitrary space position to the sliding surface in finite time. In (Levent, 2003), the sliding surface is also defined with respect to the states but the states of a SISO systems are obtained from the plant outputs by differentiation. In (Eduards et al., 2003), the sliding surface definition and the control objectives are the same. The equivalent control systems design is done with respect to the plant output, but

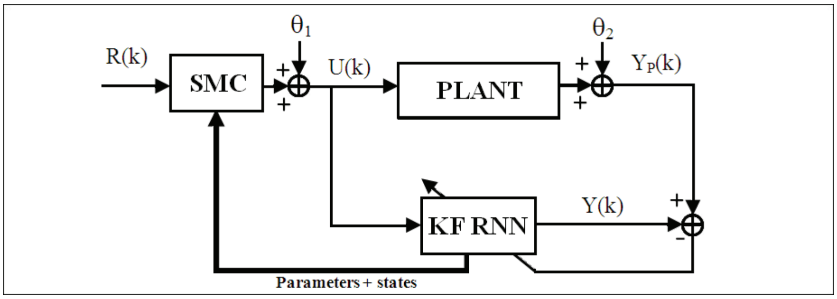


Fig. 5. Block diagram of the closed-loop system containing KF RNN identifier and a SMC

the reachability of the stable output control depended on the plant structure. In (Baruch et al., 2007a; Baruch et al., 2007b), the sliding surface is derived directly with respect to the plant outputs which facilitated the equivalent SMC systems design.
Let us define the following Sliding Surface (SS) as an output tracking error function:

$$S(k+1)=E(k+1)+\sum_{i=1}^{P}\gamma_i\, E(k-i+1);\quad |\gamma_i| < 1 \tag{55}$$

Where: S(.) is the Sliding Surface Error Function (SSEF) defined with respect to the plant output; E(.) is the systems output tracking error; γ_i are parameters of the desired stable SSEF; P is the order of the SSEF. The tracking error and its iterate are defined as:

$$E(k) = R(k) - Z(k);\ E(k+1) = R(k+1) - Z(k+1) \tag{56}$$

Where R(k), Z(k) are L-dimensional reference and output vectors of the local linear plant model. The objective of the sliding mode control systems design is to find a control action which maintains the systems error on the sliding surface which assure that the output tracking error reaches zero in P steps, where P < N. So, the control objective is fulfilled if:

$$S(k+1) = 0 \tag{57}$$

Now, let us to iterate (54) and to substitute (53) in it so to obtain the input/output local plant model, which yields:

$$Z(k+1) = H\,X(k+1) = H\,[AX(k) + BU(k)] \tag{58}$$

From (55), (56), and (57) it is easy to obtain:

$$R(k+1) - Z(k+1) + \sum_{i=1}^{P}\gamma_i\, E(k-i+1) = 0 \tag{59}$$

The substitution of (58) in (59) gives:

$$R(k+1) - HAX(k) - HBU(k) + \sum_{i=1}^{P}\gamma_i\, E(k-i+1) = 0 \tag{60}$$

As the local approximation plant model (53), (54), is controllable, observable and stable, (see the proofs of the preceeding paragraph), the matrix A_1 is diagonal, and L = M, then the matrix product (HB), representing the plant model static gain, is nonsingular, and the plant states X(k) are smooth non-increasing functions. Now, from (60) it is easy to obtain the equivalent control capable to lead the system to the sliding surface which yields:

$$U_{eq}(k) = (HB)^{-1} [- HAX(k) + R(k+1) + \sum_{i=1}^{P} \gamma_i \, E(k-i+1)] \qquad (61)$$

Following (Young et al., 1999), the SMC avoiding chattering is taken using a saturation function instead of sign one. Here the saturation level Uo is chosen with respect to the load level perturbation. So the SMC takes the form:

$$U^*(k) = \begin{cases} U_{eq}(k), & \text{if } \|U_{eq}(k)\| < U_0 \\ -U_0 U_{eq}(k) / \|U_{eq}(k)\|, & \text{if } \|U_{eq}(k)\| \geq U_0 \end{cases} \qquad (62)$$

It is easy to see that the substitution of the equivalent control (61) in the linear plant model (53), (54) show an exact complete plant dynamics compensation which avoided oscillations, so that the chattaring effect is not observed. Furthermore, the designed plant output sliding mode equivalent control substituted the multi-input multi-output coupled high order dynamics of the linearized plant with desired decoupled low order one.

3.2 Direct adaptive neural control scheme

The Direct Adaptive Neural Control (DANC) using the RTNN as plant identifier and plant controller has been described in (Baruch et al., 2001b; Flores et al., 2001; Baruch et al., 2004; Baruch et al., 2005). The block-diagram of the control system is given on Fig. 6. It contains a recurrent neural identifier, and two recurrent neural controllers (feedback and feedforward). Let us to write the following *z*-transfer- function representations of the plant, the state estimation part of the KF RNN, the feedback and the feedforward controllers:

$$W_p(z) = C_p \, (zI - A_p)^{-1} \, B_p \qquad (63)$$

$$P_i(z) = (zI - A_i)^{-1} \, B_i \qquad (64)$$

$$Q_1(z) = C_{cfb} \, (zI - A_{cfb})^{-1} \, B_{cfb} \qquad (65)$$

$$Q_2(z) = C_{cff} \, (zI - A_{cff})^{-1} \, B_{cff} \qquad (66)$$

The control systems *z*-transfer functions (63)-(66) are connected by the following equation, derived from the Fig. 6, and given in *z*-operational form:

$$Y_p(z) = W_p(z) \, [I + Q_1(z) \, P_i(z)]^{-1} \, Q_2(z) \, R(z) + \theta(z) \qquad (67)$$

$$\theta(z) = W_p(z) \, \theta_1(z) + \theta_2(z) \qquad (68)$$

Where θ(z) is a noise term. The RTNN and the KF RNN topologies are controllable and observable. The BP algorithm of learning is convergent (Baruch et al., 2002; Nava et al., 2004). Then the identification and control errors tend to zero.

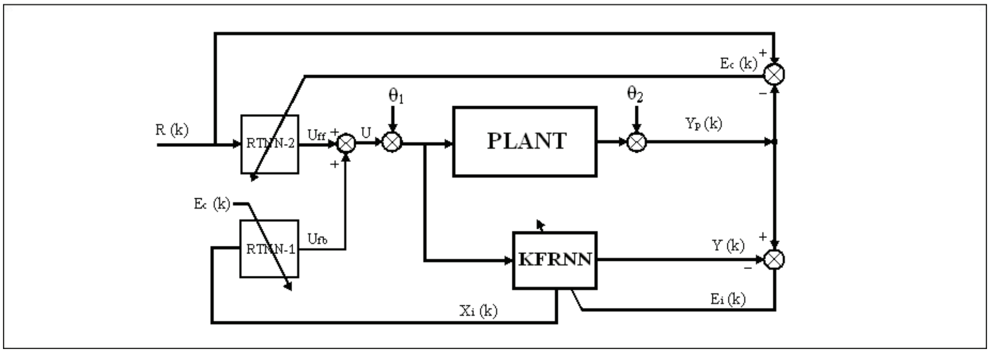


Fig. 6. Block - diagram of the control system containing neural identifier and two adaptive neural controllers.

$$E_i(k) = Y_p(k) - Y(k) \rightarrow 0; k \rightarrow \infty \tag{69}$$

$$E_c(k) = R(k) - Y_p(k) \rightarrow 0; k \rightarrow \infty \tag{70}$$

This means that each transfer function given by equations (63)-(66) is stable with minimum phase. The closed-loop system is stable and the RTNN-1 feedback controller compensates the plant dynamics. The RTNN-2 feedforward controller dynamics is an inverse dynamics of the closed-loop system one, which assure a precise reference tracking in spite of the presence of process and measurement noises.

4. Experimental and simulation results

A time ago the KF RNN has been applied for prediction of various bioprocesses like the Fed-Batch fermentation kinetics of Bacillus Thuringiensis (Valdes-Castro et al., 2003), the osmotic dehydration process (Baruch et al., 2005), and the hydrocarbon degradation profiles in a biopile system (De la Torre-Sanchez et al., 2006). Some preliminary results of application of the KF RNN used as systems identifier in a sliding mode controlled bioremediation processes have been presented in various scientific conferences like (Baruch et al., 2007 a; Baruch et al., 2007b). In this part those results would be described with more details. The bioremediation process at hand is considered as completely unknown and represented by input/output records of normalized noisy data.

4.1 Experimental and simulation results obtained for the biopile system

Description of the Process and the Experiment. Biological treatment is attractive as a potentially low-cost technology, which converts toxic organic contaminants into CO_2 and biomass. Since the 70's, this technology has been applied for the hydrocarbon degradation, and today, it is considered as the best alternative to cleanup polluted soils. Bioremediation in biopile system is an *ex-situ* Solid Substrate Fermentation (SSF) technology, based on the ability of micro-organisms to degrade pollutant hydrocarbon compounds (Alexander, 1994). The often used bio-stimulation technique consists on the activation of the native soil micro-organisms by addition of nutrients, water, oxygen (for aerobic process) and a bulking agent that let it to improve the oxygen supplied to the microorganisms. The Solid Substrate

Fermentation takes place in the absence of free water, so it offers the advantage of reducing the place and cost requirements. The SSF disadvantage consists of the complexity and heterogeneity of the solid matrix, which makes quite difficult the measurement and control of process variables. The interest of the biopile technology is an inherent temperature increase inside the biopile - from the centre to the surface, which favors the sequential development of a microbial population growth associated to the temperature profile and residual pollution. Temperature increase can reach 60ºC, so it is frequently controlled by an air flux supplied to the biopile columns. Besides, controlling the temperature, the air flux is a source of fresh oxygen to the microorganisms. The next environmental conditions are recommended for an adequate hydrocarbon biodegradation in biopile system: pH $\approx$ 7; humidity at 50-60% of the water holding capacity of soil; average temperature of 30ºC. It is important to supply an adequate air flux, since a low one could not be enough for satisfying the microbial requirements, but a high air one could dry the solid matrix. In this study, it is used a crumb-limose soil from a site polluted near a refinery in México. The pollution of 165000 ppm, consist on different residues of crude oil process and refining. The soil was dried and blended with ocorn used as a bulking agent 10:1 (% v/v), which was milled and sterilized. The moisture was adjusted at 60% of water retention capacity, and C:N:P ratio at 100:10:1 according to analyses done. Tergitol 1% (p/p) was used as surfactant to enhance contaminant desorption from soil. The equipment used is shown on Fig. 7a, and the Input/Output full KF RNN learning pattern in shown on Fig. 7b. The biopile system consists of twenty one columns (1.0 m height x 3.81 cm i.d.), constructed to allow the monitoring through 28 days, almost each other day. Each column has sample ports located at the sites every 25 cm, and was fitted with water vessels to humidify the air entering the columns. The columns were housed in a chamber provided with temperature control, and the air was supplied at a constant pressure via a manifold. The experiment consists of seven sets of fermentation data taken for different air flux (180, 360, 450 and 540 ml/min) and different temperature (20 and 40ºC). The duration of the bioremediation process depends on the volume of the soil under treatment and the type and concentration of the contaminants in it. In our case 28 days are sufficient to degrade 60% of the contaminants which is considered sufficient for our experiment. The evolution of the hydrocarbon removal was evaluated from solid samples periodically extracted from the biopile for analysis of pH (potentiometer), humidity (gravimetric method), oxygen consumption and carbon dioxide production - by gas chromatography, Total Petroeum Hydrocarbons (TPH) - by infrared spectroscopy, following soxhlet extraction with dichloromethane (EPA Method 3540C).

Process Identification. The graphical results of the experimental neural biodegradation process identification are given on Fig. 8 a – for KF RNN learning, and on Fig. 8 b – for KF RNN generalization. The Input Learning Pattern (ILP) proposed is conformed by the: ILP(AF, TEMP, pH, HU, O2, CO2, TPH). The Output Learning Pattern (OLP) includes: OLP(pH, HU, O2, CO2, TPH). The KF RNN used for modeling and identification of the hydrocarbon degradation process in biopile system has seven inputs, twelve neurons in the hidden layer and five outputs. The number of neurons (twelve) in the hidden layer was determined in an experimental way, applying the rule of thumb and according to the Mean Square Error (MSE%) of learning. The learning algorithm is a version of the dynamic BP one, specially designed for this KF RNN topology. The described above learning algorithm is applied simultaneously to 7 degradation kinetic data sets (patterns), realized below different conditions of air flow and temperature in the ranges 180-540 mi/min and 25-50ºC,

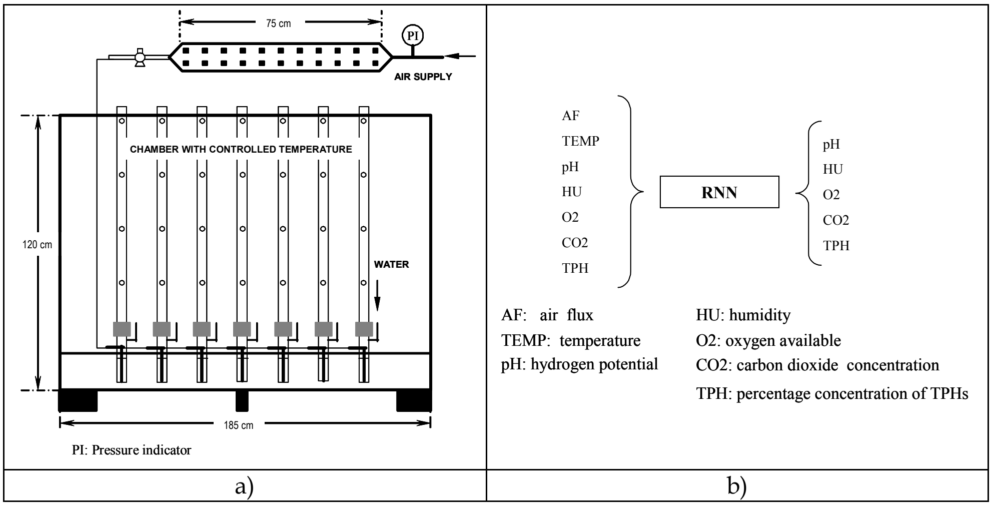


Fig. 7. a) Sketch of the biopile system; b) Learning pattern of the full KF RNN model

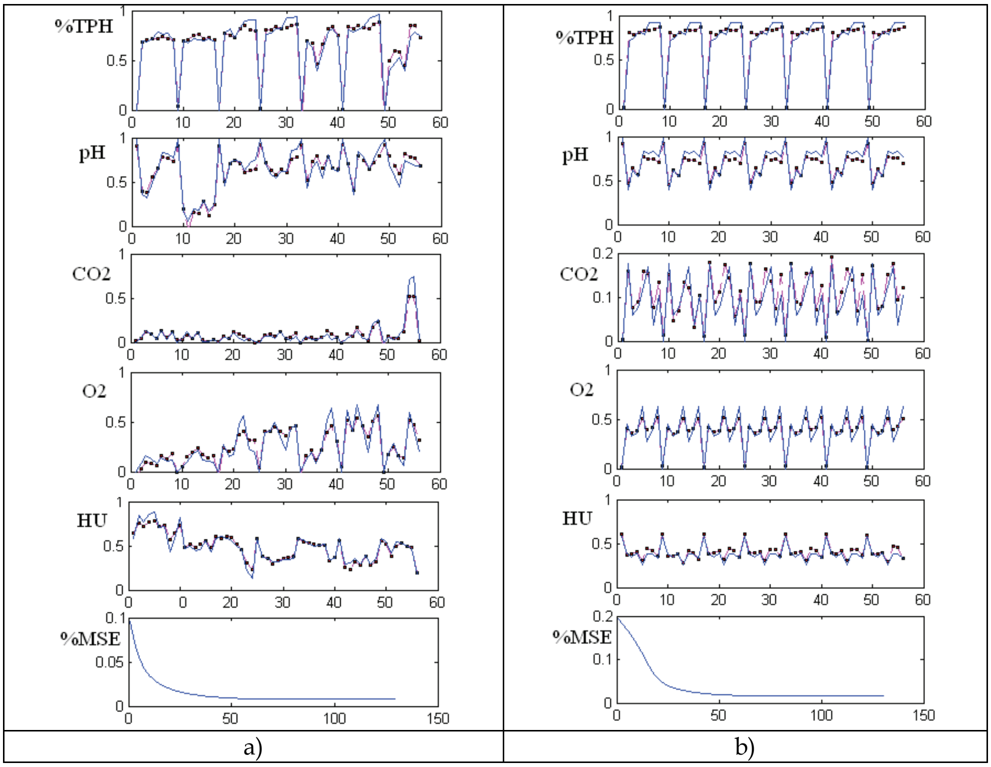


Fig. 8. Graphical results of experimental biodegradation process identification; a) graphical results of KF RNN learning (%TPH, pH, CO2, O2, HU, and MSE%); b) graphical results of KF RNN generalization (%TPH, pH, CO2, O2, HU, and MSE%)

and containing 8 points each one. The experimental data were normalized in the range 0-1 due to the great difference in magnitude between them. The 7 data sets are considered as an epoch of learning, containing 56 points. After each epoch of learning, the 7 pattern sets are interchanged in an arbitrary manner from one epoch to another. An unknown kinetic data set, repeated 7 times, is used as a generalization data set. The learning is stopped when the MSE% of learning and generalization reached values below 2%, and the relationship $|\Delta W_{ij}(k)|/|W_{ij}(k)|*100\%$ reached values below or equal of 2% for all updated parameters**.** This error was attained after 131 epochs of learning. The graphical results shown on Fig. 8 a. compared the experimental data for the 7 degradation kinetics with the outputs of the KF RNN during the last epoch of learning. The variables compared and plotted subsequently for the last epoch of learning are % degradation in TPH, pH, carbon dioxide (CO2), oxygen available (O2), % of humidity (HU) and the mean square error (MSE%) given for 131 epochs of learning. The learning rate is 0.9, the momentum rate is 0.8, the epoch size contains 56 points, the convergence is obtained after 131 epochs of learning. The final MSE% of learning is below 2%. The generalization of the KF RNN was carried out reproducing a degradation kinetics which is not included in the training process. This degradation process was carried out at AF = 360 ml/min and temperature of 20°C. The operational conditions of this degradation process are in the range of operational conditions studied. The generalization results shown on Fig. 8 b. compare the experimental data for the one unknown degradation kinetics (repeated 7 times so to maintain the epoch size) with the output of the KF RNN. The same experimental data %TPH, pH, CO2, O2, HU, MSE% (continuous line) are compared with the KF RNN outputs (pointed line) and are plotted subsequently for the last epoch of generalization. The final MSE% of KF RNN generalization is below 2%.

Simulation Results Obtained with the Sliding Mode Control and the Direct Adaptive Neural Control. The graphical simulation results of the controlled system with both controls are given on Fig. 9a,b and the MSE% of control is given in Table 1, Table 2 for 20 runs of the control program (SMC and DANC) with data mixed with 10% measurement Gaussian noise with different variance for each run. A simplified RTNN process model, extracted from the complete KF RNN model has been used to design both control systems and to issue the state vector. The RTNN particular model used as a feedforward controller has 2 inputs or references (%TPH, CO2), two outputs as control signals (AF, HU) and 9 states. The RTNN feedback controller has the topology (12, 9, 2). The RTNN particular plant model has 2 inputs (AF, HU), two outputs (%TPH, CO2) and 12 states. In that reduced model, depending on the available measurements, the input and output patterns are chosen as: ILP(AF, HU, CO2, TPH); OLP(CO2, TPH). For both control schemes, the two system set points (continuous line) are compared with the two plant outputs (%TPH, CO2) (pointed line) and are plotted subsequently for seven sets of set point data. The control variables shown are: AF, HU. However the lost of water is pretended to be compensated by the wet saturated air flux with controlled humidity introduced, which could accelerate the bioremediation process in the biopile system. The obtained MSE% of control in the end of the process for both control schemes is below 1%. The behaviour of the control system in the presence of 10% white Gaussian noise (with different SEED parameter at each run) added to the plant outputs could be studied acumulating some statistics of the final MSE% (ξ_{av}) for multiple run of the control program (see Table 1 for SMC and Table 2 for DANC). The mean average cost for all runs (ε) of control, the standard deviation (σ) with respect to the mean value and the deviation (Δ) are computed by means of the following formulas:

No	1	2	3	4	5
MSE%	1.106	1.0035	1.001	1.0951	0.93454
No	6	7	8	9	10
MSE%	1.146	1.3214	1.225	1.4721	1.1206
No	11	12	13	14	15
MSE%	1.3185	1.1544	1.1821	1.0316	1.1267
No	16	17	18	19	20
MSE%	1.1295	1.3268	1.1842	1.2858	1.1993

Table 1. Final MSE (%) of control (ξ_{av}) for 20 runs of the SMC control program

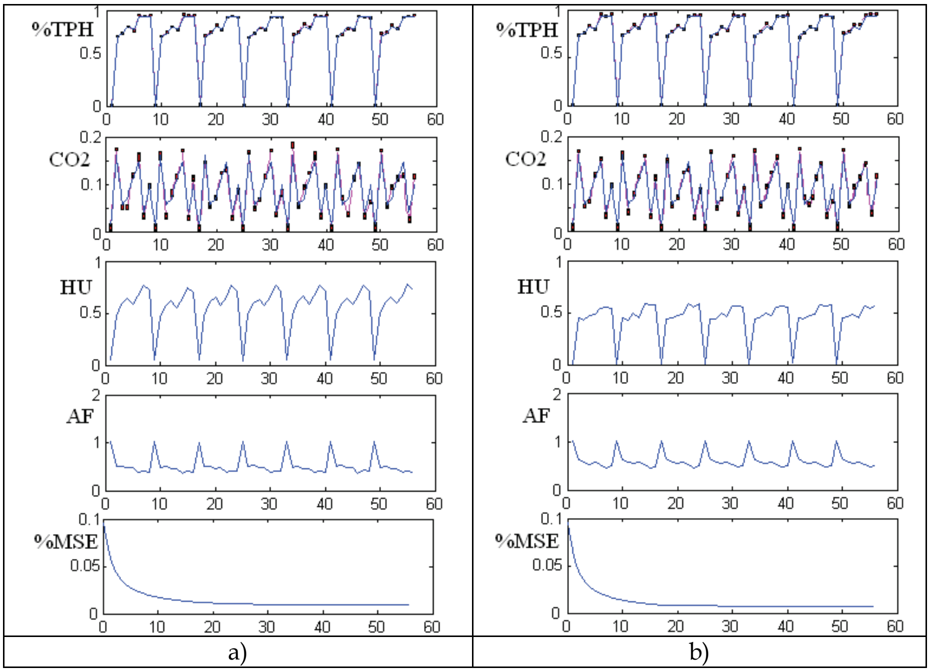


Fig. 9. a) Graphical results of the biodegradation process SMC; b) Graphical results of the biodegradation process DANC; for both schemes the variables shown are (%TPH, CO2, AF, HU, MSE%)

No	1	2	3	4	5
MSE%	0.9805	0.8207	1.0421	0.8148	0.8813
No	6	7	8	9	10
MSE%	0.8227	1.0959	0.8990	0.8100	1.0881
No	11	12	13	14	15
MSE%	1.0551	0.9569	0.8227	1.0619	1.0891
No	16	17	18	19	20
MSE%	1.0518	1.1173	0.8045	1.0454	1.1012

Table 2. Final MSE (%) of control (ξ_{av}) for 20 runs of the DANC control program

$$\varepsilon = \frac{1}{n}\sum_{k=1}^{n} \xi_{av\ k}\ ;\ \sigma = \sqrt{\frac{1}{\mathrm{n}}\sum_{i=1}^{n} \Delta_i^2}\ ;\ \Delta = \xi_{av} - \varepsilon \tag{71}$$

Where k is the run number and n is equal to 20. The mean and standard deviation values of process error, obtained for the SMC, are respectively: $\varepsilon = 1.1682\%$; $\sigma = 0.1276\ \%$. The mean and standard deviation values of process control, obtained for DANC, are respectively: $\varepsilon = 0.9680\ \%$; $\sigma = 0.0583\ \%$ which is a little bit smaller that the results, obtained for the SMC due to the nonlinearity and adaptivity of the DANC, which contained two RTNN controllers.

4.2 Simulation results obtained for the rotating drum

Description of the Process and the Experiment. For the bioremediation process one challenge is to provide enough O_2 and nutrients to enable rapid conversion of contaminants by either indigenous microorganisms or inoculated species (Alexander, 1994). Another challenge is to achieve efficient contact between the active micro-organisms and the contaminants, which may represented a problem with *in-situ* treatment. An attractive alternative to overcome this problem is to apply a biological treatment in slurry phase using Horizontal Rotating Drum (HRD) (see the schematic diagram given on Fig. 10a). The HRD can effectively mix heterogeneous blends of a wide range of particle sizes and high solid concentration (more than 60 %), (Alexander, 1994). The HRD operated with oxygen supply or aeration. Independently of the type of HRD operation (open or close), the insufficiency of water decreased the efficiency of hydrocarbon degradation in HRD favouring the formation of hydrocarbon balls (Alexander, 1999). So one objective of the process control is to maintain the humidity at 60%, which is the maximal solid concentration determined as the best for hydrocarbon removal from polluted soils treated in open rotating slurry bioreactors. Nowadays, semi empirical models, based on the Monod equation, have been developed to predict micro-organism growth as a function of available contaminants concentration. However, as the application of such models requires experimental work for calculating the kinetics parameters involved, so an alternative modelling technique is required. The KF RNN model offers many advantages as the possibility to approximate complex non linear high order multivariable processes, as the biodegradation process is. The bioremediation of polluted soils selected for modelling purpose was carried out by bio-stimulation in slurry phase using an open HRD. A silt loam (sand 36.5%w/w, silt 62.5% w/w and clay 1% w/w) soil of an average diameter of 15 µm, particle diameter in the range 2 - 75 µm, was used. The soil was contaminated with 50000 ppm of crude oil collected from a contaminated zone located near from a petroleum refinery. The slurry was prepared with 40% weight of soil (715 g) and 60% weight of a mineral solution (formula in $kg \cdot m^{-3}$: $(NH_4)_2SO_4$, 19; KH_2PO_4, 1.7; $MgSO_4$, 1; $CaCl_2 \cdot 2H_2O$, 0.005; $FeCl_3 \cdot 6H_2O$, 0.0025; yeast extract, 0.59; tergitol - 0.5%). The slurry was added to a HRD of 4 litters (13 cm diameter by 30 cm long), which was opened, on the flat faces, for a natural air supply (see Fig. 10a). The drum was operated during 19 days at a fix turning in the interval 3.5-20 RPM. During this time, the reactor was daily weighted in order to replace the water lost, so to maintain constant the water concentration. Samples were removed each day for analysis of residual hydrocarbons, pH, water concentration and slurry viscosity. The hydrocarbon concentration was determined by an infrared spectrometer; the pH was measured with a Beckman Φ potentiometer; water concentration was calculated by difference of two sequence data of the drum weight; finally, slurry viscosity was measured with an AND Vibro-viscometer SV-10 (MED BY A&D LTD).

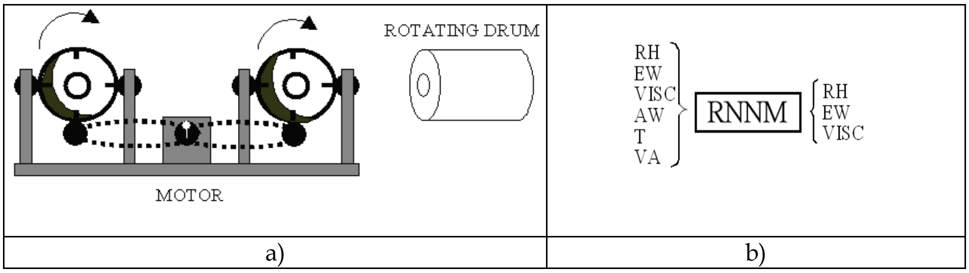


Fig. 10. a) Schematic diagram of a rotating drum system. b) Learning pattern

The biodegradation process was repeated at a different turning value (3.5, 5, 7.5, 10, 15, 20 RPM) in order to vary the oxygen available into the HRD. The learning pattern (see Fig. 10b) used for KF RNN training is composed by six input variables and three output variables. In order to avoid saturation problems in the RNNM training, the variables of the learning pattern are normalized in the interval 0-1. The measured variables are: Residual Hydrocarbon Concentration (RH), Evaporated Water (EW); Soil Viscosity (VISC), Added Water (AW); Temperature (T); Velocity of Agitation (VA). The RNNM outputs are: OUT (RH, EW, VISC). Depending on the available measurements and the control objectives, this model could be simplified, where the input- output pattern is chosen as: ILP (RH, EW, AW, VA); OLP (RH, EW). This reduced model will be used for SMC and DANC system design.

Process Identification. The described above learning algorithm is applied simultaneously to four fermentation kinetic data, represented by its input/output learning data patterns, and containing 19 points each (one per day). The total time of learning is 200 epochs, where the epoch size, corresponding to the number of data, is 76 iterations. After each epoch of training, the 4 sets are interchanged in an arbitrary manner from one epoch to another. The learning is stopped when the MSE% of learning reached values below 1.5%, the MSE% of generalization reached valued below 2%, and the relationship $|\Delta W_{ij}(k)|/|W_{ij}(k)|*100\%$ reached values below or equal of 2% for all updated parameters. Graphical results of RNNM training are given in Fig. 11a for the last epoch of learning. In the graphics, the output variables of the KF RNN are compared with the experimental data. The Fig. 11a compared the 4 kinetics experimental data with those, issued by the KF RNN. The output process data of 76 points are the hydrocarbon residual, the water requirements and the soil viscosity (EW, RH, VISC). The last figure in Fig. 11a represents the evolution of the mean squared error of approximation (MSE%) for whole learning process of 200 epochs. An unknown set of kinetic data, containing 19 points and repeated 4 times, so to maintain the same 76-points epoch size, is used as a validation (generalization) set, and these results are given on Fig. 11b. The obtained graphical results of KF RNN training and generalization shows a good convergence with an MSE% below 1.5% for the training and 2% for the generalization.

Simulation Results Obtained with the Sliding Mode Control and the Direct Adaptive Neural Control. A simplified RTNN process model extracted from the KFRNN complete identified model has been used to design SMC and DANC systems. The RTNN particular model has two inputs (AW, VA), two outputs (EW, RH) and nine states. The SMC SSEF is chosen as a first order one (P=1) with parameters Uo=1, γ=0.07, L=M=2. The DANC RTNN particular model used as a feedforward controller has two reference inputs (EW, RH), two outputs as control signals (VA, AW) and six states. The feedback RTNN controller has topology (12, 6, 2). The graphical simulation results of the controlled system outputs (EW,

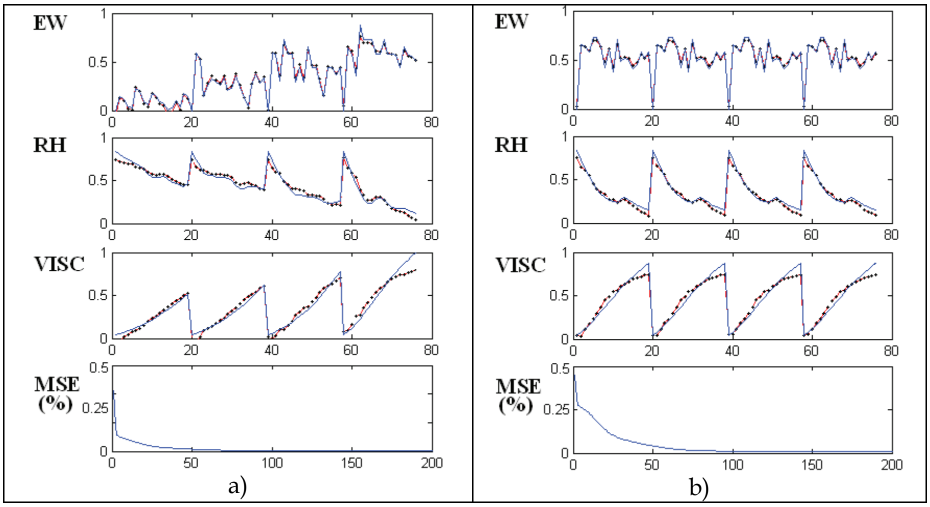


Fig. 11. Graphical results of experimental biodegradation process identification; a) graphical results of KF RNN learning (EW, RH, VISC, and MSE%.); b) graphical results of KF RNN generalization (EW, RH, VISC, and MSE%.)

RH), and the control variables (AW, VA) for both control schemes are given on Fig. 12 a,b for 76 points (one epoch of learning). For both control schemes, the two system set points (continuous line) are compared with the plant outputs (EW, RH) (data point line) and are plotted subsequently for four sets of set point data. The MSE% of control is given also in Fig. 12 a,b for all 200 epochs of learning. For both control schemes, the obtained MSE% of control at the end of the process is below 1%. The behaviour of the control system in the presence of 5% white Gaussian noise (with different SEED parameter at each run) added to the plant output has been studied accumulating some statistics of the final MSE% (ξ_{av}) for multiple run of the control program (SMC and DANC), which results are given on Table 3, Table 4 for 20 runs. The mean average cost for all runs (ε) of control, the standard deviation (σ) with respect to the mean value and the deviation (Δ) are computed using the formulas (71). The mean and standard deviation values of process error, obtained for the SMC are respectively: ε = 0.6663 %; σ= 0.0593 %. The mean and standard deviation values of process control, obtained for the DANC are respectively: ε = 0.5456 %; σ= 0.0124 %, which is slightly smaller with respect to the SMC, due to the nonlinearity and the adaptivity of the DANC, which contained two RTNN controllers.

5. Conclusion

The chapter proposes a new Kalman filter closed loop topology of recurrent neural network for identification and modeling of an unknown hydrocarbon degradation process carried out in a biopile system and a rotating drum. The proposed KF RNN contained a recurrent neural plant model, a recurrent neural output plant filter and posses global and local feedbacks. The learning algorithm is a modified version of the dynamic Backpropagation one derived using the adjoint KF RNN topology by means of the diagramatic method. The obtained KF RNN model issued parameters and states information appropriate for control systems design purposes.

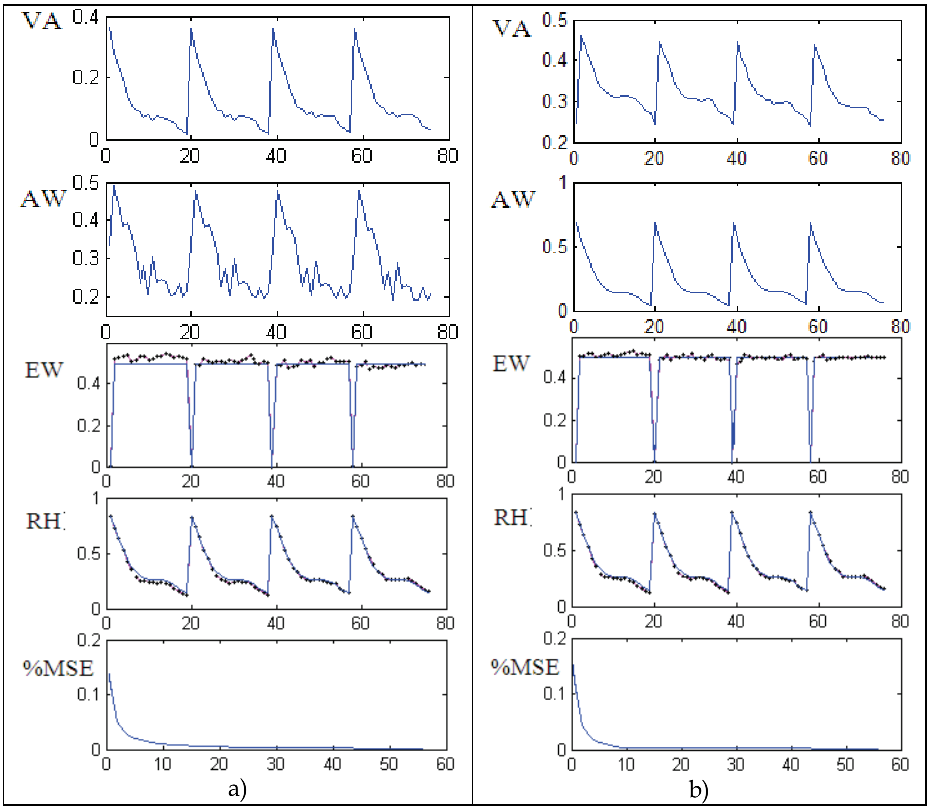


Fig. 12. a) Graphical results of the biodegradation process SMC; b) Graphical results of the biodegradation process DCD; for both schemes the variables shown are (VA, AW, EW, RH, MSE%)

No	1	2	3	4	5
MSE%	0.6434	0.6577	0.7669	0.6805	0.6662
No	6	7	8	9	10
MSE%	0.5757	0.5835	0.7043	0.7040	0.6350
No	11	12	13	14	15
MSE%	0.6602	0.7759	0.7732	0.6566	0.6408
No	16	17	18	19	20
MSE%	0.6481	0.6061	0.7240	0.6514	0.5725

Table 3. Final MSE (%) of control (ξ_{av}) for 20 runs of the control program

The obtained complete KF RNN model is simplified and used to design an indirect sliding mode control and a direct recurrent feedback-feedforward neural control. The simulation results obtained with the recurrent neural model learning and control exhibited a good convergence and precise reference tracking. The MSE% of the KF RNN learning and generalization is below 2% and the MSE% of the indirect and direct control is below 1%.

No	1	2	3	4	5
MSE%	0.5187	0.5449	0.5788	0.5738	0.5496
No	6	7	8	9	10
MSE%	0.5208	0.5732	0.5418	0.5672	0.5576
No	11	12	13	14	15
MSE%	0.5619	0.5040	0.5468	0.5471	0.5029
No	16	17	18	19	20
MSE%	0.5752	0.5744	0.5228	0.5065	0.5440

Table 4. Final MSE (%) of control (ξ_{av}) for 20 runs of the control program

Some statistical results of multiple run of the control program with noisy data, obtained with both control schemes are also given. The results show a slight priority of the DANC with respect to the SMC due to the better adaptation abilities to the first one.

6. References

Albertini, F. & Sontag, E. (1994). State observability in recurrent neural networks. *System and Control Letters,* Vol. 22, No 4, (April 1994) page numbers (235-244), ISSN 0167-6911

Alexander, M. (1999). *Biodegradation and Bioremediation*. Academic Press, ISBN 0-12-049861, New York

Baruch, I.S.; Stoyanov, I.P. & Gortcheva, E. (1996). Topology and learning of a class RNN, *ELEKTRIK (Turkish Journal of Electrical Engineering and Computer Sciences),* Vol. 4 Supplement, No 1, (January 1996) page numbers (35-42), ISSN: 1300-0632

Baruch, I.S.; Gortcheva, E. & Garrido, R. (1999). Redes neuronales recurrentes para la identificacion de objetos no lineales (in spanish). *Cientifica, (The Mexican Journal of Electromechanical Engineering, ESIME-IPN)*, Vol. 3, No 14, (March-April 1999) page numbers (39-45), ISSN 1665-0654

Baruch, I.S.; Flores, J.M.; Thomas, F. & Garrido, R. (2001). Adaptive neural control of nonlinear systems. In: *Artificial Neural Networks-ICANN 2001, LNCS 2130*, Dorffner, G., Bischof, H., Hornik, K. (Eds.), page numbers (930-936), Springer, ISBN 3-540-42486-5, Berlin

Baruch, I.S.; Flores, J.M. & Nenkova, B. (2001). Design of indirect adaptive neural control systems. *Cybernetics and Information Technologies (Bulgarian Academy of Sciences),* Vol. 1, No 1, (January 2001) page numbers (81- 94), ISSN 1311-9702

Baruch, I.S.; Flores, J.M.; Nava, F.; Ramirez, I.R. & Nenkova, B. (2002). An Advanced neural network topology and learning applied for identification and control of a D.C. motor, *Proceedings of the First International IEEE Symposium on Intelligent Systems*, pp. 289-295, ISBN 0-7803-7601-3, Varna Bulgaria, September 2002, IEEE Inc., New York

Baruch, I.S.; Barrera-Cortes, J. & Hernandez, L.A. (2004). A fed-batch fermentation process identification and direct adaptive neural control with integral term. In: *MICAI 2004: Advances in Artificial Intelligence, LNAI 2972,* Monroy, R., Arroyo-Figueroa, G., Sucar, L.E., Sossa, H. (Eds.), page numbers (764-773), Springer-Verlag, ISBN 3-540-21459-3, Berlin Heidelberg New York

Baruch, I.S.; Georgieva, P.; Barrera-Cortes, J. & Feyo de Azevedo, S. (2005). Adaptive recurrent neural network control of biological wastewater treatment. *International*

Journal of Intelligent Systems, Special issue on Soft Computing for Modelling, Simulation and Control of Nonlinear Dynamical Systems, (O.Castillo, and P.Melin - guest editors), Vol. 20, No 2, (February 2005) page numbers (173-194), ISSN 0884-8173

Baruch, I.S.; Genina-Soto, P. & Barrera-Cortes, J. (2005). Predictive neural model of an osmotic dehydration process. *Journal of Intelligent Systems, Special Issue on Hybrid Intelligent Systems for Time Series Prediction, (O.Castillo, and P.Melin - guest editors),* Vol. 14, No 2-3, (February-March 2005) page numbers (143-155), ISSN 0334-1860

Baruch, I.S.; Mariaca-Gaspar, C.R. & Barrera-Cortes, J. (2007). Neural modelling and sliding mode control of bio-degradation process in a rotating bioreactor, In: *Preprints of the 8-th IFAC International Symposium on Dynamics and Control of Process Systems, DYCOPS*, vol. 2, Foss, B., Alvarez, J. (Eds.), pp. 261-266, Cancun Mexico, June 6-8, 2007, IFAC

Baruch, I.S.; Mariaca-Gaspar, C.R.; Cruz-Vega, I. & Barrera-Cortes, J. (2007). Sliding mode control of a hydrocarbon degradation in biopile system using recurrent neural network model, In: *MICAI 2007: Advances in Artificial Intelligence, LNAI 4827*, Gelbukh, A., Kuri-Morales, A.F. (Eds.), page numbers (1184-1194), Springer, ISBN-10 3-540-76630-8, Berlin Heidelberg New York

Boskovic, J.D. & Narendra, K. S. (1995). Comparison of linear, nonlinear and neural-network-based adaptive controllers for a class of fed-batch fermentation processes. *Automatica*, Vol. 31, No 6, (June 1995) page numbers (817-840), ISSN 0005-1098

Chen, S. & Billings, S.A. (1992). Neural networks for nonlinear dynamics system modeling and identification. *International Journal of Control,* Vol. 56, No 2, (August 1992) page numbers (319-346), ISSN 0020-7179

De la Torre-Sanchez, R.; Baruch, I.S. & Barrera –Cortes, J. (2006). Neural prediction of hydrocarbon degradation profiles developed in a biopile. *Expert Systems with Applications*, Vol. 31, No 3 (April 2006) page numbers (283-389), ISSN 0957-4174

Eduards, C.; Spurgeon, S.K. & Hebden, R.G. (2003). On the design of sliding mode output feedback controllers. *International Journal of Control, Special Issue Dedicated to Vadim Utkin on the Occasion of his 65th Birthday (Guest editor: Leonid M. Fridman),* Vol. 76, No 9/10, (15 June/10 July 2003) page numbers (893-905), ISSN 0020-7179

Flores, J. M.; Baruch, I. S. & Garrido, R. (2001). Red neuronal recurrente para identificación y control de sistemas no lineales (in spanish). *Científica, (The Mexican Journal of Electromechanical Engineering, ESIME-IPN),* Vol. 5, No 1, (January-March 2001) page numbers (11-20), ISSN 1665-0654

Haykin, S. (1999). *Neural Networks, a Comprehensive Foundation. Second Edition.* Section 2.13, pp. 84-89; Section 4.13, pp. 208-213. Prentice-Hall, ISBN 0-13-273350-1, Upper Saddle River New Jersey

Hunt, K. J.; Sbarbaro, D.; Zbikowski, R., & Gawthrop, P. J. (1992). Neural network for control systems - a survey. *Automatica*, Vol. 28, No 6 (December 1992) page numbers (1083-1112), ISSN 0095-0963

Jin, L. & Gupta, M. (1999). Stable dynamic backpropagation learning in recurrent neural networks. *IEEE Transactions on Neural Networks*, Vol. 10, No 6, (November 1999) page numbers (1321-1334), ISSN 1045-9227

Kazemy, A.; Hosseini, S.A. & Farrokhi, M. (2007) Second order diagonal recurrent neural network, *Proceedings of the IEEE International Symposium on Industrial Electronics, ISIE,* pp. 251-256, ISBN 978-1-4244-0755-2, 4-7 June 2007, Vigo, Spain, IEEE Inc., New York

Ku, C.C. & Lee, K.Y. (1995). Diagonal recurrent neural networks for dynamic systems control. *IEEE Transactions on Neural Networks*, Vol. 6, No 1, (January 1995) page numbers (144-156), ISSN 1045-9227

Levent, A. (2003). Higher order sliding modes, differentiation and output feedback control. *International Journal of Control, Special Issue Dedicated to Vadim Utkin on the Occasion of his 65th Birthday (Guest editor: Leonid M. Fridman),* Vol. 76, No 9/10, (15 June/10 July 2003) page numbers (924-941), ISSN 0020-7179

Mastorocostas, P.A. & Theocharis, J.B. (2006). A stable learning algorithm for block-diagonal recurrent neural networks: application to the analysis of lung sounds. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics,* Vol. 36, No 2 (April 2006) page numbers (242-254), ISSN 1083-4419

Miller, W.T., III; Sutton, R.S. & Werbos, P.J. (1990). *Neural Networks for Control*, A Bradford Book, MIT Press, ISBN 0-262-13261-3, Cambridge, Massachusetts, London, England

Narendra, K.S. & Parthasarathy, K. (1990). Identification and control of dynamic systems using neural networks. *IEEE Transaction on Neural Networks,* Vol. 1, No. 1, (January 1990) page numbers (4-27), ISSN 1045-9227

Narendra, K.S. & Mukhopadhyay, S. (1994). Adaptive control of nonlinear multivariable systems using neural networks. *Neural Networks*, Vol. 7, No 5, (May 1994) page numbers (737-752), ISSN 0893-6080

Nava, F.; Baruch, I.S.; Poznyak, A. & Nenkova, B. (2004). Stability proofs of advanced recurrent neural networks topology and learning. *Comptes Rendus (Proceedings of the Bulgarian Academy of Sciences)*, Vol. 57, No 1, (January 2004) page numbers (27-32), ISSN 0861-1459

Pao, S.A.; Phillips, S.M. & Sobajic, D. J. (1992). Neural net computing and intelligent control systems. *International Journal of Control*, *Special Issue on Intelligent Control,* Vol. 56, No 3, (August 1992) page numbers (263-289), ISSN 0020-7179

Sage, A.P. (1968) *Optimum Systems Control*. Prentice-Hall Inc., Library of Congress Catalog Number 68-20862, Englewood Cliffs, New Jersey

Sontag, E. & Sussmann, H. (1997). Complete controlability of continuos time recurrent neural network. *System and Control Letters*, Vol. 30, No 4, (May 1997) page numbers (177-183), ISSN 0167-6911

Su, Hong-Te; McAvoy, Th. J. & Werbos, P. (1992). Long-term predictions of chemical processes using recurrent neural networks: a parallel training approach. *Industrial Engineering Chemical Research*, Vol. 31, No 5, (May 1992) page numbers (1338-1352), ISSN 0888-5885

Valdez-Castro, L.; Baruch, I.S. & Barrera-Cortes, J. (2003). Neural networks applied to the prediction of fed-batch fermentation kinetics of Bacillus Thuringiensis. *Bioprocess and Biosystems Engineering*, Vol. 25, No 4, (January 2003) page numbers (229-233), ISSN: 1615-7591

Wan, E. & Beaufays, F. (1996). Diagrammatic method for deriving and relating temporal neural networks algorithms. *Neural Computation,* Vol. 8, No 2, (February 1996) page numbers (182-201), ISSN 0899-7667

Young, K.D.; Utkin, V.I. & Ozguner, U. (1999). A control engineer's guide to sliding mode control. *IEEE Trans. on Control Systems Technology*, Vol. 7, No 3, (May 1999) page numbers (328-342), ISSN 1063-6536

7. Appendix: Proof of the Theorem of RTNN Stability

Let the Recurrent Trainable Neural Network with Jordan Canonical Structure given by (1), (2), (3), (4), (5), (6), (7) and the nonlinear plant model as follows:

$$x(k+1)=g[x(k),u(k)] \tag{A.1}$$

$$y(k)=f[x(k)] \tag{A.2}$$

and the plant and activation functions fulfill the following assumptions:
Assumption 1: The plant dynamics is locally Lipchitz, so the functions $g(\cdot)$, $f(\cdot)$ are as:

$$f:=\{f \mid f=\sigma+\Delta f, \|\Delta f\| \le f_0+f_1\|x(k)\|\}$$

$$g:=\{g \mid g=\sigma+\Delta g, \|\Delta g\| \le g_0+g_1\|x(k)\|\}$$

and Δg , Δf are modeling errors, which reflex the effect of unmodeled dynamics.
Assumption 2: The activation function has the following Taylor approximation:

$$\sigma(\bar{\theta})=\sigma(\theta)+\frac{\partial\sigma(\bar{\theta})}{\partial\bar{\theta}}(\bar{\theta}-\theta)+\varsigma$$

with the approximation error bound given by:

$$\|\varsigma\|^2 \le \frac{L}{2}\|\bar{\theta}-\theta\|^2$$

and the signal error defined by:

$$e(k)=\hat{y}(k)-y(k)$$

$$e(k+1)=\hat{y}(k+1)-y(k+1)=F[C(k)\hat{x}(k)]-F[C^{*}\hat{x}(k)]-\Delta f[x(k)]$$

Now, let us define the state estimation error, add and subtract the RTNN to the last equation and apply the Assumption 2, then:

$$\Delta(k)=\hat{x}(k)-x(k)$$

$$\Delta(k+1)=\hat{x}(k+1)-x(k+1)=G[A(k)\hat{x}(k)+B(k)u(k)]-G[A^{*}\hat{x}(k)+B^{*}u(k)]-\Delta g[x(k),u(k)]$$

Let us now define the output identification error and put it in terms of the state estimation error as:

$$e(k+1)=F'(k)\delta C(k)[G'(k)(\delta A(k)\hat{x}(k)+\delta B(k)u(k))+\Theta_3]+F'(k)\delta C^*\left[G'(k)\left(A^*\Delta(k)-B^*O_F\right)+\Theta_4\right] \\ +\Theta_1+\Theta_2-\Delta f(x(k),\bar{u}(k))$$

Where: the term $\bar{u}(k)=u(k)+O_F$; the $\Theta_{1,2,3,4}$ are the higher order terms in the Taylor series approximation; $\Delta f(x(k),\bar{u}(k))$ is the unmodeled dynamics; O_F is an offset.

If Assumptions 1 and 2 fulfil, the learning algorithm for the RTNN is given by (8) and the learning parameters η_k, α_k are normalized and depended on the output error structure.

Then, the approximation error is bounded.

Consider a Lyapunov candidate function as

$$L(k)=L_1(k)+L_2(k) \tag{A.3}$$

In which $L_1(k)$ and $L_2(k)$ are given by:

$$L_1(k)=\frac{1}{2}e^2(k) \tag{A.4}$$

$$L_2(k)=tr\left(\tilde{W}_A(k)\tilde{W}_A^T(k)\right)+tr\left(\tilde{W}_B(k)\tilde{W}_B^T(k)\right)+tr\left(\tilde{W}_C(k)\tilde{W}_C^T(k)\right) \tag{A.5}$$

Where:

$$\tilde{W}_{A1}(k)=\hat{A}(k)-A^*,\tilde{W}_B(k)=\hat{B}(k)-B^*,\tilde{W}_C(k)=\hat{C}(k)-C^*$$

are vectors of the estimation error and (A^*,B^*,C^*) and $(\hat{A}_k,\hat{B}_k,\hat{C}_k)$ denote the ideal neural weight and the estimate of neural weight at the k-th step, respectively, for each case.

Let us consider the equation (A.4). The change of the Lyapunov function in two consecutive samples due of the training process is obtained by:

$$\Delta L_1(k)=L_1(k+1)-L_1(k)=[e(k+1)-e(k)]\left[e(k)+\tfrac{1}{2}e(k+1)-\tfrac{1}{2}e(k)\right] \tag{A.6}$$

Then, defining $\Delta e(k)$ as the difference between two consecutive error samples, the equation (A.6) becomes:

$$\Delta L_1(k)=\Delta e(k)\left[e(k)+\tfrac{1}{2}\Delta e(k)\right] \tag{A.7}$$

Where: $\Delta e(k)$ can be defined as:

$$\Delta e(k)=\left[\frac{\partial e(k)}{\partial W}\right]\underline{\Delta\vec{W}} \tag{A.8}$$

Putting all weights into one vector as

$$\vec{W}=\left[\left[\vec{A}\right]^T\left[\vec{B}\right]^T\left[\vec{C}\right]^T\right]^T \tag{A.9}$$

Where:

$$\vec{A}=\left[\left[\vec{A}_1\right]^T\left[\vec{A}_2\right]^T L\left[\vec{A}_n\right]^T\right]^T,\vec{B}=\left[\left[\vec{B}_1\right]^T\left[\vec{B}_2\right]^T L\left[\vec{B}_m\right]^T\right]^T,\vec{C}=\left[\left[\vec{C}_1\right]^T\left[\vec{C}_2\right]^T L\left[\vec{C}_n\right]^T\right]^T$$

which represents the weight vectors constructed by their columns. Also let:

$$\eta=\begin{bmatrix}\eta^A & & \\ & \eta^B & \\ & & \eta^C\end{bmatrix},\quad \alpha=\begin{bmatrix}\alpha^A & & \\ & \alpha^B & \\ & & \alpha^C\end{bmatrix} \tag{A.10}$$

Where: (η^A,η^B,η^C) and $(\alpha^A,\alpha^B,\alpha^C)$ represented the learning rate matrix, the momentum rate matrix corresponding to (A,B,C), respectively, and $\eta^A=\eta_1 I_A$, $\eta^B=\eta_2 I_B$, $\eta^C=\eta_3 I_C$, $\alpha^A=\alpha_1 I_A$, $\alpha^B=\alpha_2 I_B$, $\alpha^C=\alpha_3 I_C$. Moreover, $\eta_i(i=1,\ldots,3)$ and $\alpha_i(i=1,\ldots,3)$ are two positive constants, and I_Z is an identity matrix with Z representing A,B,C, respectively. Now, we could define $\underline{\Delta}W$ and derive an expression for $\Delta L_1(k)$:

$$\underline{\Delta}W=\eta\Delta W(k)+\alpha\Delta W(k-1) \tag{A.11}$$

$$\begin{aligned}\underline{\Delta}W&=-\eta\frac{\partial L_1(k)}{\partial W}-\alpha\frac{\partial L_1(k-1)}{\partial W}=-e(k)\eta\frac{\partial e(k-1)}{\partial W}-e(k-1)\alpha\frac{\partial e(k-1)}{\partial W}\\ &=-e(k)\begin{bmatrix}\eta^A & & \\ & \eta^B & \\ & & \eta^C\end{bmatrix}\times\left[\left[\frac{\partial e(k)}{\partial A}\right]^T\left[\frac{\partial e(k)}{\partial B}\right]^T\left[\frac{\partial e(k)}{\partial C}\right]^T\right]^T\\ &\quad-e(k-1)\begin{bmatrix}\alpha^A & & \\ & \alpha^B & \\ & & \alpha^C\end{bmatrix}\times\left[\left[\frac{\partial e(k-1)}{\partial A}\right]^T\left[\frac{\partial e(k-1)}{\partial B}\right]^T\left[\frac{\partial e(k-1)}{\partial C}\right]^T\right]^T\end{aligned} \tag{A.12}$$

$$\begin{aligned}\Delta e(k)&=\left[\frac{\partial e(k)}{\partial W}\right]^T\underline{\Delta}W\\ &=-e(k)\times\left(\eta_1\left\|\frac{\partial e(k)}{\partial A}\right\|^2+\eta_2\left\|\frac{\partial e(k)}{\partial B}\right\|^2+\eta_3\left\|\frac{\partial e(k)}{\partial C}\right\|^2\right)\\ &\quad-e(k\text{-}1)\times\left(\alpha_1\left\|\frac{\partial e(k-1)}{\partial A}\right\|^2+\alpha_2\left\|\frac{\partial e(k-1)}{\partial B}\right\|^2+\alpha_3\left\|\frac{\partial e(k-1)}{\partial C}\right\|^2\right)\end{aligned} \tag{A.13}$$

$$\begin{aligned}\gamma&=\eta_1\left\|\frac{\partial e(k)}{\partial A}\right\|^2+\eta_2\left\|\frac{\partial e(k)}{\partial B}\right\|^2+\eta_3\left\|\frac{\partial e(k)}{\partial C}\right\|^2\\ \lambda&=\alpha_1\left\|\frac{\partial e(k-1)}{\partial A}\right\|^2+\alpha_2\left\|\frac{\partial e(k-1)}{\partial B}\right\|^2+\alpha_3\left\|\frac{\partial e(k-1)}{\partial C}\right\|^2\end{aligned} \tag{A.14}$$

$$\Delta e(k+1) = -\gamma e(k+1) - \lambda e(k) \tag{A.15}$$

$$\begin{aligned}\Delta L_1(k+1) &= \Delta e(k+1)\left[e(k+1)+\tfrac{1}{2}\Delta e(k+1)\right] \\ &= -\tfrac{1}{2}e^2(k+1)\left[2\gamma-\gamma^2\right]+e(k+1)e(k)[\gamma-1]\lambda+\tfrac{1}{2}\lambda^2 e^2(k)\end{aligned} \tag{A.16}$$

Proposing: $\lambda = \gamma - 1$, then:

$$\Delta L_1(k+1) = -\tfrac{1}{2}e^2(k+1)\left[-2\gamma^2+4\gamma-1\right]-\tfrac{1}{2}\lambda^2[\Delta e(k)]^2 \tag{A.17}$$

According to the Lyapunov's stability theory, the convergence could be be guaranteed, if $\Delta L(k+1)<0$, thus $-2\gamma^2+4\gamma-1>0$, and

$$\left(1-\frac{1}{\sqrt{2}}\right)<\gamma<\left(1+\frac{1}{\sqrt{2}}\right) \tag{A.18}$$

That is :

$$\left(1-\frac{1}{\sqrt{2}}\right)<\eta_1\left\|\frac{\partial e(k)}{\partial A}\right\|^2+\eta_2\left\|\frac{\partial e(k)}{\partial B}\right\|^2+\eta_3\left\|\frac{\partial e(k)}{\partial C}\right\|^2<\left(1+\frac{1}{\sqrt{2}}\right) \tag{A.19}$$

Let $\eta_{max} = \max_{i=1}^{3}\{\eta_i\}$; thus, as long as:

$$\frac{\left(1-\frac{1}{\sqrt{2}}\right)}{\left\|\frac{\partial e(k)}{\partial A}\right\|^2+\left\|\frac{\partial e(k)}{\partial B}\right\|^2+\left\|\frac{\partial e(k)}{\partial C}\right\|^2}<\eta_{max}<\frac{\left(1+\frac{1}{\sqrt{2}}\right)}{\left\|\frac{\partial e(k)}{\partial A}\right\|^2+\left\|\frac{\partial e(k)}{\partial B}\right\|^2+\left\|\frac{\partial e(k)}{\partial C}\right\|^2} \tag{A.20}$$

Note that $\|\cdot\|$ is the Euclidean norm, therefore:

$$\left\|\frac{\partial e(k)}{\partial A}\right\|^2+\left\|\frac{\partial e(k)}{\partial B}\right\|^2+\left\|\frac{\partial e(k)}{\partial C}\right\|^2=\left\|\frac{\partial e(k)}{\partial W}\right\|^2 \tag{A.21}$$

Now let : $\psi(k) = \frac{\partial e(k)}{\partial W} = -\frac{\partial y(k)}{\partial W}$ and $\psi_{max} = \max_k \|\psi(k)\|$, then:

$$\frac{\left(1-\frac{1}{\sqrt{2}}\right)}{\psi_{max}}<\eta_{max}<\frac{\left(1+\frac{1}{\sqrt{2}}\right)}{\psi_{max}} \tag{A.22}$$

Now, working with equation (A.5), we have:

$$L_2(k) = tr\left(\tilde{W}_A(k)\tilde{W}_A^T(k)\right)+tr\left(\tilde{W}_B(k)\tilde{W}_B^T(k)\right)+tr\left(\tilde{W}_C(k)\tilde{W}_C^T(k)\right) \tag{A.23}$$

Considering the change of the Lyapunov function in two consecutive samples of the training process, and substituting the quantities: $\tilde{W}_{A(k)}=\hat{A}(k)-A^*, \tilde{W}_B(k)=\hat{B}(k)-B^*, \tilde{W}_C(k)=\hat{C}(k)-C^*$, we get:

$$\Delta L_2(k) = L_2(k+1)-L_2(k) = \mathrm{tr}\begin{pmatrix} \hat{A}(k+1)\hat{A}^T(k+1) - \hat{A}(k+1)A^{*T} - A^*\hat{A}^T(k+1)+A^*A^{*T} \\ -\hat{A}(k)\hat{A}^T(k)+\hat{A}(k)A^{*T}(k)+A^*\hat{A}^T(k) - A^*A^{*T} \end{pmatrix}$$
$$+\mathrm{tr}\begin{pmatrix} \hat{B}(k+1)\hat{B}^T(k+1) - \hat{B}(k+1)B^{*T} - B^*\hat{B}^T(k+1)+B^*B^{*T} \\ -\hat{B}(k)\hat{B}^T(k)+\hat{B}(k)B^{*T}+B^*\hat{B}^T(k) - B^*B^{*T} \end{pmatrix} \tag{A.24}$$
$$+\mathrm{tr}\begin{pmatrix} \hat{C}(k+1)\hat{C}^T(k+1) - \hat{C}(k+1)C^{*T} - C^*\hat{C}^T(k+1)+C^*C^{*T} \\ -\hat{C}(k)\hat{C}^T(k)+\hat{C}(k)C^{*T}+C^*\hat{C}^T(k) - C^*C^{*T} \end{pmatrix}$$

Applying the learning law (8) and the trace properties we obtained:

$$\Delta L_2(k) = \eta_{max}^2\left[\|\Delta\hat{A}(k)\|^2+\|\Delta\hat{B}(k)\|^2+\|\Delta\hat{C}(k)\|^2\right]+\alpha_{max}^2\left[\|\Delta\hat{A}(k-1)\|^2+\|\Delta\hat{B}(k-1)\|^2+\|\Delta\hat{C}(k-1)\|^2\right]$$
$$+2\eta_{max}\mathrm{tr}\left(\tilde{A}(k)\Delta\hat{A}^T(k)+\tilde{B}(k)\Delta\hat{B}^T(k)+\tilde{C}(k)\Delta\hat{C}^T(k)\right)$$
$$+2\alpha_{max}\mathrm{tr}\left(\tilde{A}(k)\Delta\hat{A}^T(k-1)+\tilde{B}(k)\Delta\hat{B}^T(k-1)+\tilde{C}(k)\Delta\hat{C}^T(k-1)\right) \tag{A.25}$$
$$+2\alpha_{max}\eta_{max}\mathrm{tr}\left(\Delta\hat{A}(k)\Delta\hat{A}^T(k-1)+\Delta\hat{B}(k)\Delta\hat{B}^T(k-1)+\Delta\hat{C}(k)\Delta\hat{C}^T(k-1)\right)$$

Substituting the learning values and errors (9)-(15), we obtained terms like:

$$2\eta_{max}|e(k)|^2 - 2\eta_{max}\xi(k)e(k) \;;\; 2\eta_{max}|e(k-1)|^2 - 2\eta_{max}\xi(k-1)e(k-1) \tag{A.26}$$

Applying the following inequality: $X^TY+(X^TY)^T \leq X^T\Lambda X+Y^T\Lambda^{-1}Y$, which is valid for any $X,Y \in \Re^{n\times m}$, and for any positive definite matrix $0 < \Lambda = \Lambda^T \in \Re^{n\times n}$, we obtained:

$$2\eta_{max}e(k)\xi(k) = (\eta_{max}e(k))\xi(k)+\xi(k)(\eta_{max}e(k)) \leq \eta_{max}^2\|e(k)\|_{\Lambda 1}^2+\|\xi(k)\|_{\Lambda_1^{-1}}^2 \;;$$

$$2\alpha_{max}e(k-1)\xi(k-1) = (\alpha_{max}e(k-1))\xi(k-1)+\xi(k-1)(\alpha_{max}e(k-1)) \leq \alpha_{max}^2\|e(k-1)\|_{\Lambda 2}^2+\|\xi(k-1)\|_{\Lambda_2^{-1}}^2 \tag{A.27}$$

Analyzing (A.27) term by term and applying the Rayleigh inequality: $\lambda_{min}(\Lambda)\|x\|^2 \leq x^T\Lambda x \leq \lambda_{max}(\Lambda)\|x\|^2$ we obtained a statement for $\Delta L_2(k)$. Making inner terms equal to one as in the unit circle condition for discrete time, at last we get the final condition:

$$\Delta L_2(k) \leq -\eta_{max}|e(k)|^2 - \alpha_{max}|e(k-1)|^2 + d(k) \tag{A.28}$$

$$d(k) = \|\xi(k)\|_{\Lambda_1^{-1}}^2+\|\xi(k-1)\|_{\Lambda_2^{-1}}^2 \tag{A.29}$$

Where $d(k)$ represented the unmodeled dynamics and/or perturbations term. Applying the Rate of Convergence Lemma (Nava et al., 2004) for the result (A.28) we could conclude that: the $d(k)$ - term must be bounded by the weight matrices and the learning parameter in order to obtain the final result: $\Delta L_2(k) \in L_\infty$. As a consequence we get : $A_{(k)} \in L_\infty, B_{(k)} \in L_\infty, C_{(k)} \in L_\infty$. From equations (A.22) and (A.28) we easily could get the inequality (A.20). Therefore the boundedness of $L(k)$, $k \in Z_0^+$ is guaranteed.

Design of Self-Constructing Recurrent-Neural-Network-Based Adaptive Control

Chun-Fei Hsu [1] and Chih-Min Lin [2]
Chung Hua University [1], Yuan Ze University [2]
Taiwan, Republic of China

1. Introduction

Recently, neural-network-based adaptive control technique has attracted increasing attentions, because it has provided an efficient and effective way in the control of complex nonlinear or ill-defined systems (Duarte-Mermoud et al., 2005; Hsu et al., 2006; Lin and Hsu, 2003; Lin et al., 1999; Peng et al. 2004). The key elements of this success are the approximation capabilities of the neural networks. The parameterized neural networks can approximate the unknown system dynamics or the ideal tracking controller after learning. One must distinguish between two classes of control applications – open-loop identification and closed-loop feedback control. Identification applications are similar to signal processing/classification, so that the same open-loop algorithms may often be used. Therefore, a tremendous amount of training data must be used and considerable training time undertaken is required. On the other hand, in closed-loop feedback applications the neural network is inside the control loop, so that special steps must be taken to ensure that the tracking error and the neural network weights remain bounded in the closed-loop system. The basic issues in neural network closed-loop feedback control are to provide on-line learning algorithms that do not require preliminary off-line tuning. Some of these learning algorithms are based on the backpropagation algorithm. However, these approaches have difficulties to guarantee the stability and robustness of closed-loop system (Duarte-Mermoud et al., 2005; Lin et al., 1999). Another learning algorithms are based on the Lyapunov stability theorem. The tuning laws have been designed to guarantee the system stability in the Lyapunov sense (Hsu et al., 2006; Lin & Hsu, 2003; Peng et al., 2004).
However, these neural networks are feedforward neural networks; they belong to static mapping networks. Without aid of tapped delay, a feedforward neural network is unable to represent a dynamic mapping. The recurrent neural network (RNN) has superior capabilities as compared to feedforward neural networks, such as their dynamic response and their information storing ability (Lee & Teng, 2000; Lin & Hsu, 2004). Since an RNN has an internal feedback loop, it captures the dynamic response of a system with external feedback through delays. Thus, an RNN is a dynamic mapping network. Due to its dynamic characteristic and relatively simple architecture, the recurrent neural network is a useful tool for most real-time applications (Lin & Chen, 2006; Lin & Hsu, 2004; Tian et al., 2004; Wai et al. 2004).
Although the neural-network-based adaptive control performances are acceptable in above literatures; however, the learning algorithm only includes the parameter learning, and they

have not considered the structure learning of the neural network. If the number of hidden neurons is chosen too large, the computation load is heavy so that they are not suitable for practical applications. If the number of hidden neurons is chosen too small, the learning performance may be not good enough to achieve desired control performance. To tackle this problem, several self-structuring neural networks, consisting of structure and parameter learning phases, have been proposed (Huang et al., 2004; Leung & Tsoi, 2005; Lin et al, 2005). These learning phases not only decide the structure of neural network but also adjust the parameters of neural network. Recently, some self-structuring neural networks have been applied to solve several control problems (Lin et al., 2001; Gao & Er, 2003; Park et al., 2005). Lin et al. (2001) used a similarity measure method to avoid the newly generated membership function being too similar to the existing ones; however, the structure would grow large as the input data has large variations. Gao & Er (2003) proposed an error reduction ratio with QR decomposition to prune the hidden neurons; however, the design procedure is overly complex. Park et al. (2005) proposed a self-structuring neural network which can create new hidden neurons to increase the learning ability; unfortunately, the proposed approach can not avoid the structure of neural network growing unboundedly.

This paper proposes a recurrent-neural-network-based adaptive control (RNNAC) method, which combines neural-network-based adaptive control, robust control and self-structuring approach, for a class of unknown nonlinear systems. The proposed RNNAC system is composed of a neural controller and a robust controller. The neural controller uses a self-structuring recurrent neural network (SRNN) to approximate an ideal tracking controller. The learning process of SRNN includes the structure learning and parameter learning. In the structure learning, the SRNN can online create new hidden neurons as the incoming data is far away the existing hidden neurons, and cancel hidden neurons as the hidden neurons is inappropriate. Thus the learning capability and flexibility can be upgraded. In the parameter learning, the controller parameters can be online tuned based on the Lyapunov function, so that the stability of the closed-loop system can be guaranteed. The robust controller is designed to recover the residual of the approximation error to achieve L^2 tracking performance with desired attenuation level. Finally, the proposed RNNAC system is applied to control a nonlinear dynamic system. Simulation results are performed to demonstrate the effectiveness of the proposed design method.

2. Problem statement and ideal tracking control

The model of many practical nonlinear systems can be expressed in the nth-order form as

$$x^{(n)} = f(\mathbf{x}) + u \tag{1}$$

where $\mathbf{x} = [x, \dot{x}, \dots, x^{(n-1)}]^T$ is the state vector of the system, which is assumed to be available for measurement, $f(\mathbf{x})$ is the nonlinear system dynamics which can be unknown, and u is the input of the system. The tracking control problem of the system is to find a control law so that the state trajectory x can track a reference command x_c closely. The tracking error is defined as

$$e = x_c - x\ . \tag{2}$$

If the exact model of the controlled system is well known, there exists an ideal tracking controller to achieve favorable control performance by possible canceling all the system uncertainties (Slotine and Li, 1991). Assume that the parameters of the controlled system in (1) are well known, there exits an ideal tracking controller

$$u^* = -f(\mathbf{x}) + x_c^{(n)} + \mathbf{K}^T\mathbf{E} \tag{3}$$

where $\mathbf{E} = [e, \dot{e}, \ldots, e^{(n-1)}]^T$ and $\mathbf{K} = [k_n, \cdots, k_2, k_1]^T$. Applying the ideal tracking controller (3) to system (1) results in the following error dynamics

$$e^{(n)} + k_1 e^{(n-1)} + \cdots + k_n e = 0\,. \tag{4}$$

If k_i, $i = 1, 2, \cdots, n$ are chosen such that all roots of the polynomial $h(s) \underline{\underline{\Delta}}\, s^n + k_1 s^{n-1} + \cdots + k_n$ lie strictly in the open left half of the complex plane, then it implies that $\lim_{t\to\infty} e = 0$ for any starting initial conditions. The error dynamics (4) can be rewritten in a vector form as

$$\dot{\mathbf{E}} = \mathbf{AE} \tag{5}$$

where $\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & \cdots 0 \\ \vdots & \ddots & \ddots & \ddots 0 \\ 0 & \cdots & 0 & 1 \\ -k_n & -k_{n-1} & \cdots & -k_1 \end{bmatrix}$. However, since the system dynamics $f(\mathbf{x})$ may be unknown or perturbed in practical applications, the ideal tracking controller (3) can not be precisely obtained.

3. Design of RNNAC

For achieving a favorable tracking performance and a specified attenuation level simultaneously, the developed recurrent-neural-network-based adaptive control (RNNAC) system with structure adaptation algorithm shown in Fig. 1 is assumed to take the following form

$$u_{anc} = u_{nc} + u_{rc} \tag{6}$$

where u_{nc} is the neural controller and u_{rc} is the robust controller. The neural controller using a self-structuring recurrent neural network (SRNN) to approximate the ideal tracking controller is the principal controller; and the robust controller is designed to achieve a specified L^2 robust tracking performance. The detail will be described as follows:

3.1 Description of SRNN

Radial basis function (RBF) networks have gained much popularity due to their ability to approximate complex nonlinear mappings directly from the input-output data with a simple topological structure. RBF is different from neural network with sigmoidal activation functions utilizing basis functions, which are locally responsive to input stimulus. Each output of RBF has a radially symmetrical response around the center vector. Although the RBF neural-network-based adaptive control performances are acceptable, the structure of the RBF network is determined by trial-and-error, and RBF network is unable to represent a dynamic mapping. To tackle this problem, a three-layer SRNN is shown in Fig. 2, which comprises of an input layer, a hidden layer with a feedback unit, and an output layer.

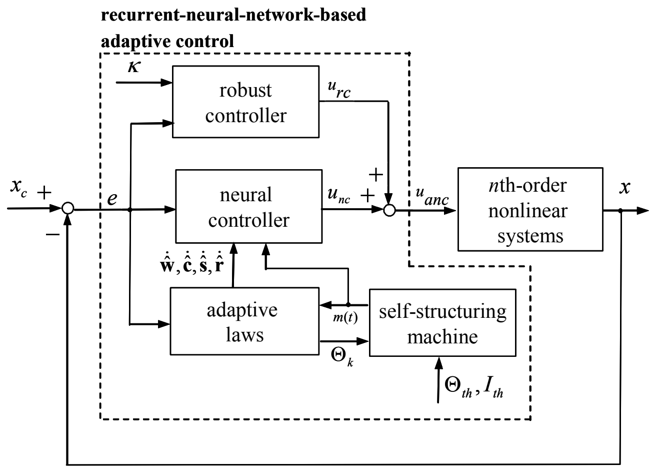

recurrent-neural-network-based adaptive control
κ
robust controller
u_{rc}
x_c +
−
e
neural controller
u_{nc} +
+
u_{anc}
nth-order nonlinear systems
x
$\dot{\hat{\mathbf{w}}}, \dot{\hat{\mathbf{c}}}, \dot{\hat{\mathbf{s}}}, \dot{\hat{\mathbf{r}}}$
adaptive laws
$m(t)$
Θ_k
self-structuring machine
Θ_{th}, I_{th}

Fig. 1 Block diagram of self-constructing RNNAC system.

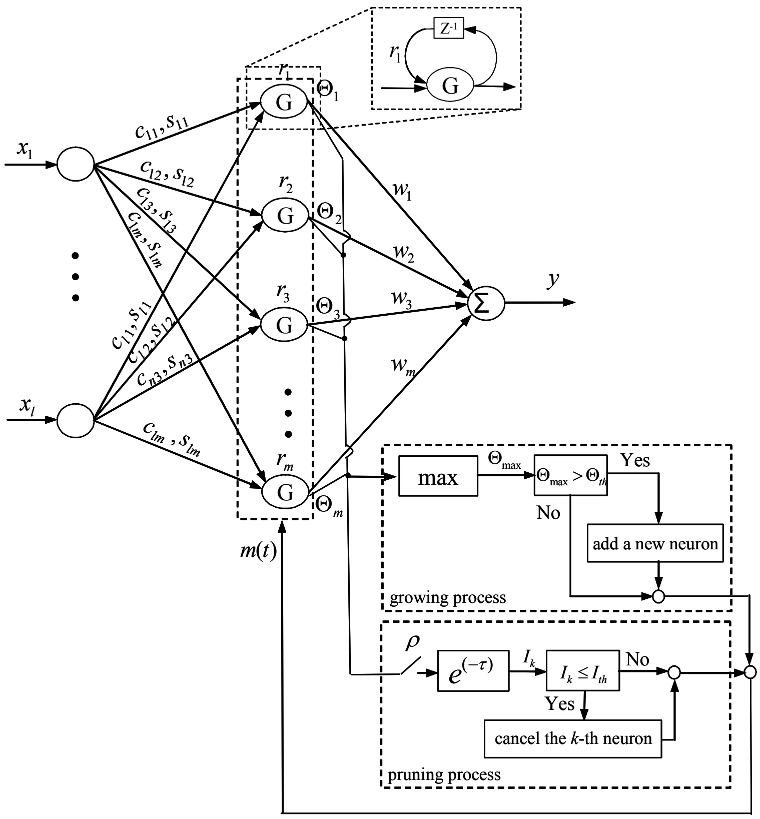


Fig. 2 The structure of self-structuring recurrent neural network.

The recurrent feedback is embedded in the network by adding feedback connections in the hidden layer. Then, the developed SRNN captures the dynamic response with external feedback through delays. The output of SRNN with m neurons for an input vector $\mathbf{x} = [x_1, x_2, ..., x_l]^T$ is given by

$$y = \sum_{k=1}^{m} w_k \Theta_k (\|\mathbf{x} - \mathbf{c}_k\|, \mathbf{s}_k, r_k) \tag{7}$$

where $\mathbf{c}_k = [c_k^1 \; c_k^2 \; .... \; c_k^l]^T$ and $\mathbf{s}_k = [s_k^1 \; s_k^2 \; .... \; s_k^l]^T$ are the center and width vectors of RBF, respectively; r_k is the internal feedback gain of RBF; w_k represents the connection weights between the hidden layer; and $\Theta_k (\|\mathbf{x} - \mathbf{c}_k\|, \mathbf{s}_k, r_k)$ represents the firing weight of the *k*-th hidden neuron which is given as

$$\Theta_k (\|\mathbf{x} - \mathbf{c}_k\|, \mathbf{s}_k, r_k) = \prod_{i=1}^{l} \exp[-(x_i + \Theta_{kp} r_k - c_k^i)^2 / {s_k^i}^2] \tag{8}$$

where c_k^i and s_k^i are the center and width of RBF in the *k*-th term of the *i*-th input variable x_i, respectively; and Θ_{kp} is the output signal of the *k*-th hidden neuron in the previous time. Define the vectors $\mathbf{c}$, $\mathbf{s}$ and $\mathbf{r}$ collecting all parameters of the hidden layer as

$$\mathbf{c} = [\mathbf{c}_1^T \; \mathbf{c}_2^T \; .... \; \mathbf{c}_m^T]^T \tag{9}$$

$$\mathbf{s} = [\mathbf{s}_1^T \; \mathbf{s}_2^T \; .... \; \mathbf{s}_m^T]^T \tag{10}$$

$$\mathbf{r} = [r_1 \cdots r_m]^T . \tag{11}$$

Then, the output of the SRNN can be represented in a vector form

$$y(\mathbf{x}, \mathbf{c}, \mathbf{s}, \mathbf{r}, \mathbf{w}) = \mathbf{w}^T \mathbf{\Theta}(\mathbf{x}, \mathbf{c}, \mathbf{s}, \mathbf{r}) \tag{12}$$

where $\mathbf{w} = [w_1 \; w_2 \; ... w_m]^T$ and $\mathbf{\Theta} = [\Theta_1 \; \Theta_2 \; ... \; \Theta_m]^T$.

If the number of the hidden neurons m is chosen too large, the computation load is heavy so that they are not suitable for online practical applications. If the number of the hidden neurons m is chosen too small, the learning performance may be not good enough to achieve desired performance.

To solve this problem, this paper proposes an online structuring learning algorithm. The first step of the structure learning is to determine whether or not to add a new hidden neuron (Lin et al., 2001). In the growing process, the firing weight of a hidden neuron for each incoming data x_i can be represented as the degree to which the incoming data belong to the existing hidden neurons. According to the degree measure, the criterion of generating a new hidden neuron for new incoming data is described as follows. Find the maximum degree $\Theta_{\max}$ defined as

$$\Theta_{\max} = \max_{1 \le k \le m(t)} \Theta_k \tag{13}$$

where $m(t)$ is the number of the existing hidden neurons at the time *t*. It can be observed that if the maximum degree $\Theta_{\max}$ is small as the incoming data is far away the existing hidden neurons. If $\Theta_{\max} \le \Theta_{th}$ is satisfied, where $\Theta_{th} \in (0,1)$ is a pre-given threshold, then a new hidden neuron is generated. The Θ_{th} denotes the adding threshold value. If Θ_{th} is chosen to be large, the hidden neurons of SRNN can be easily created; on the other hand, if Θ_{th} is chosen to be small, the hidden neurons of SRNN can be difficulty created. For the practical implement, as the unknown control system dynamics are too complex, the Θ_{th} should be chosen as a large value so that more hidden neurons can be created to increase the learning ability. The number $m(t)$ is incremented

$$m(t+1) = m(t) + 1 . \tag{14}$$

The parameters associated with the new hidden neuron are given by

$$c_{i(m+1)}^{new} = x_i \tag{15}$$

$$s_{i(m+1)}^{new} = \sigma \tag{16}$$

$$w_{(m+1)}^{new} = r_{(m+1)}^{new} = 0 \tag{17}$$

where x_i is the new incoming data and σ is the width of a radial basis function.

Then, to prevent the structure growing unboundedly, the structure learning considers whether or not to prune the existing hidden neurons which are inappropriate. A significance of the *k*-th hidden neuron is defined as (Hsu, 2007)

$$I_k(t+1) = \begin{cases} I_k(t)\exp(-\tau), & if\ \Theta_k < \delta \\ I_k(t) & , if\ \Theta_k \ge \delta \end{cases}, \quad k = 1, ,2, \dots, m(t) \tag{18}$$

where the initial value of I_k is 1; δ is the threshold value; and τ is the elimination speed constant. The pruning algorithm is derived from the observation that if the significance gets fading when the firing weight Θ_k is smaller than the threshold value δ. If $I_k \le I_{th}$ is satisfied, where I_{th} a pre-given threshold, then the *k*-th hidden neuron is cancelled. I_{th} denotes the significance threshold value. If I_{th} is chosen to be large, the neurons of SRNN can be easily canceled. For practical implement, as the computation load is the important issue, I_{th} should be chosen as a large value so that more hidden neurons can be pruned. Hence, the computation load can be decreased. In summary, the flow chart of the structure learning algorithm is shown in Fig. 3. The major contribution of SRNN is that it can operate directly without spending much time on pre-determining the structuring of neural network.

3.2 SRNN approximation

Let the number of optimal hidden neurons be m^* and can divide into two parts. The first part contains m hidden neurons which are the activated part, and the secondary part contains $m^* - m$ hidden neurons which do not exist yet. Thus, by the universal approximation theorem, an optimal SRNN approximator can be designed to approximate y, such that (Park et al., 2005)

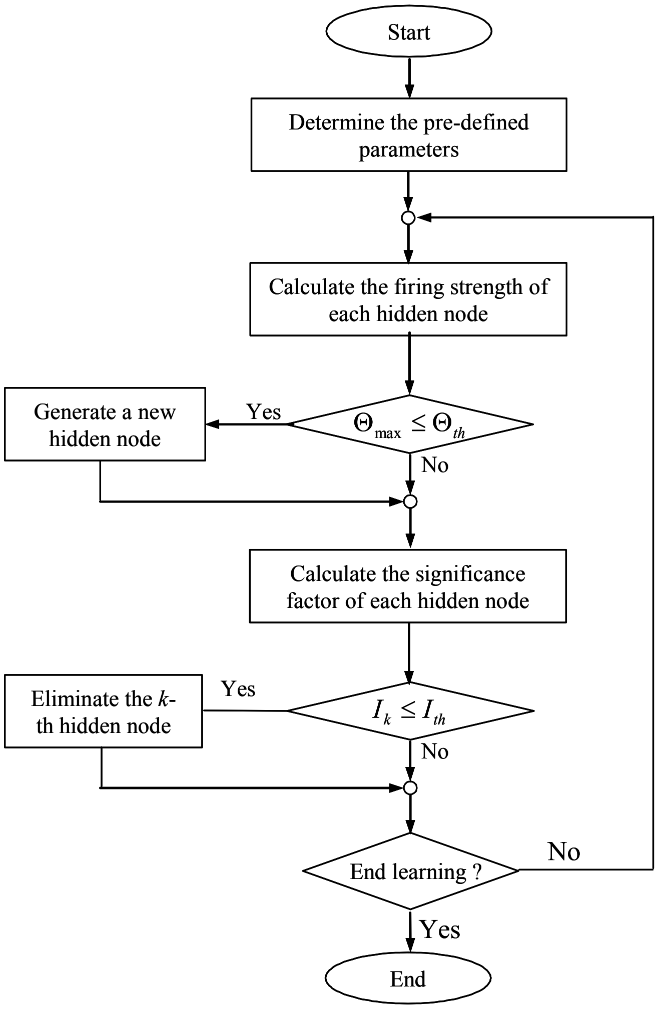

Start
Determine the pre-defined parameters
Calculate the firing strength of each hidden node
Generate a new hidden node
Yes
$\Theta_{\max} \le \Theta_{th}$
No
Calculate the significance factor of each hidden node
Eliminate the k-th hidden node
Yes
$I_k \le I_{th}$
No
End learning ?
No
Yes
End

Fig. 3 The flow chart of the structure learning algorithm for SRNN.

$$y = \mathbf{w}^{*T}\mathbf{\Theta}^{*}(\mathbf{x},\mathbf{c}^{*},\mathbf{s}^{*},\mathbf{r}^{*}) + \mathbf{w}_{u}^{*T}\mathbf{\Theta}_{u}^{*}(\mathbf{x},\mathbf{c}_{u}^{*},\mathbf{s}_{u}^{*},\mathbf{r}_{u}^{*}) + \Delta \tag{19}$$

where $\mathbf{w}^{*}$, $\mathbf{\Theta}^{*}$, $\mathbf{c}^{*}$, $\mathbf{s}^{*}$ and $\mathbf{r}^{*}$ are activated parts of optimal weights; $\mathbf{w}_{u}^{*}$, $\mathbf{\Theta}_{u}^{*}$, $\mathbf{c}_{u}^{*}$, $\mathbf{s}_{u}^{*}$ and $\mathbf{r}_{u}^{*}$ are inactivated parts of optimal weights; and Δ is the approximation error. Since these optimal parameters are unobtainable, a SRNN estimator $\hat{y}$ is defined as

$$\hat{y} = \hat{\mathbf{w}}^{T}\hat{\mathbf{\Theta}}(\mathbf{x},\hat{\mathbf{c}},\hat{\mathbf{s}},\hat{\mathbf{r}}) \tag{20}$$

where $\hat{\mathbf{w}}$, $\hat{\mathbf{\Theta}}$, $\hat{\mathbf{c}}$, $\hat{\mathbf{s}}$ and $\hat{\mathbf{r}}$ are the estimated values of $\mathbf{w}^{*}$, $\mathbf{\Theta}^{*}$, $\mathbf{c}^{*}$, $\mathbf{s}^{*}$ and $\mathbf{r}^{*}$, respectively. Define the estimated error $\tilde{y}$ as

$$\tilde{y} = y - \hat{y} = \mathbf{w}^{*T}\mathbf{\Theta}^{*} + \mathbf{w}_{\mathrm{u}}^{*T}\mathbf{\Theta}_{\mathrm{u}}^{*} + \Delta - \hat{\mathbf{w}}^{T}\hat{\mathbf{\Theta}} = \tilde{\mathbf{w}}^{T}\hat{\mathbf{\Theta}} + \hat{\mathbf{w}}^{T}\tilde{\mathbf{\Theta}} + \tilde{\mathbf{w}}^{T}\tilde{\mathbf{\Theta}} + \mathbf{w}_{\mathrm{u}}^{*T}\mathbf{\Theta}_{\mathrm{u}}^{*} + \Delta \tag{21}$$

where $\tilde{\mathbf{w}} = \mathbf{w}^{*} - \hat{\mathbf{w}}$ and $\tilde{\mathbf{\Theta}} = \mathbf{\Theta}^{*} - \hat{\mathbf{\Theta}}$. In this study, a method is proposed to guarantee the closed-loop stability and perfect tracking performance, and to tune the center and the width of the radial basis function and the recurrent weight on line. For achieving this goal, linearization technique is employed to transform the nonlinear functions into partially linear form so that the expansion of $\tilde{\mathbf{\Theta}}$ in a Taylor series to obtain (Lin and Chen, 2006)

$$\tilde{\mathbf{\Theta}} = \mathbf{T}_{\mathrm{c}}^{T}\tilde{\mathbf{c}} + \mathbf{T}_{\mathrm{s}}^{T}\tilde{\mathbf{s}} + \mathbf{T}_{\mathrm{r}}^{T}\tilde{\mathbf{r}} + \mathbf{h} \tag{22}$$

where $\tilde{\mathbf{c}} = \mathbf{c}^{*} - \hat{\mathbf{c}}$, $\tilde{\mathbf{s}} = \mathbf{s}^{*} - \hat{\mathbf{s}}$; $\tilde{\mathbf{r}} = \mathbf{r}^{*} - \hat{\mathbf{r}}$; $\mathbf{T}_{\mathrm{c}} = \left[\frac{\partial \Theta_1}{\partial \mathbf{c}} \cdots \frac{\partial \Theta_m}{\partial \mathbf{c}}\right]\Big|_{\mathbf{c}=\hat{\mathbf{c}}}$; $\mathbf{T}_{\mathrm{s}} = \left[\frac{\partial \Theta_1}{\partial \mathbf{s}} \cdots \frac{\partial \Theta_m}{\partial \mathbf{s}}\right]\Big|_{\mathbf{s}=\hat{\mathbf{s}}}$; $\mathbf{T}_{\mathrm{r}} = \left[\frac{\partial \Theta_1}{\partial \mathbf{r}} \cdots \frac{\partial \Theta_m}{\partial \mathbf{r}}\right]\Big|_{\mathbf{r}=\hat{\mathbf{r}}}$; and $\mathbf{h}$ is a vector of higher-order terms. Substituting (22) into (21), it is obtained that

$$\begin{aligned}\tilde{y} &= \tilde{\mathbf{w}}^{T}\hat{\mathbf{\Theta}} + \hat{\mathbf{w}}^{T}(\mathbf{T}_{\mathrm{c}}^{T}\tilde{\mathbf{c}} + \mathbf{T}_{\mathrm{s}}^{T}\tilde{\mathbf{s}} + \mathbf{T}_{\mathrm{r}}^{T}\tilde{\mathbf{r}} + \mathbf{h}) + \tilde{\mathbf{w}}^{T}\tilde{\mathbf{\Theta}} + \mathbf{w}_{\mathrm{u}}^{*T}\mathbf{\Theta}_{\mathrm{u}}^{*} + \Delta \\ &= \tilde{\mathbf{w}}^{T}\hat{\mathbf{\Theta}} + \tilde{\mathbf{c}}^{T}\mathbf{T}_{\mathrm{c}}\hat{\mathbf{w}} + \tilde{\mathbf{s}}^{T}\mathbf{T}_{\mathrm{s}}\hat{\mathbf{w}} + \tilde{\mathbf{r}}^{T}\mathbf{T}_{\mathrm{r}}\hat{\mathbf{w}} + \varepsilon\end{aligned} \tag{23}$$

where $\hat{\mathbf{w}}^{T}\mathbf{T}_{\mathrm{c}}^{T}\tilde{\mathbf{c}} = \tilde{\mathbf{c}}^{T}\mathbf{T}_{\mathrm{c}}\hat{\mathbf{w}}$, $\hat{\mathbf{w}}^{T}\mathbf{T}_{\mathrm{s}}^{T}\tilde{\mathbf{s}} = \tilde{\mathbf{s}}^{T}\mathbf{T}_{\mathrm{s}}\hat{\mathbf{w}}$ and $\hat{\mathbf{w}}^{T}\mathbf{T}_{\mathrm{r}}^{T}\tilde{\mathbf{r}} = \tilde{\mathbf{r}}^{T}\mathbf{T}_{\mathrm{r}}\hat{\mathbf{w}}$ are used since they are scales; and the uncertain term $\varepsilon \equiv \hat{\mathbf{w}}^{T}\mathbf{h} + \tilde{\mathbf{w}}^{T}\tilde{\mathbf{\Theta}} + \mathbf{w}_{\mathrm{u}}^{*T}\mathbf{\Theta}_{\mathrm{u}}^{*} + \Delta$.

3.3 RNNAC design

By substituting (6) into (1) and using (3) and (23), the tracking error dynamic equation can be obtained as follows

$$\begin{aligned}\dot{\mathbf{E}} &= \mathbf{A}\mathbf{E} + \mathbf{b}(u^{*} - u_{nc} - u_{rc}) \\ &= \mathbf{A}\mathbf{E} + \mathbf{b}(\tilde{\mathbf{w}}^{T}\hat{\mathbf{\Theta}} + \tilde{\mathbf{c}}^{T}\mathbf{T}_{\mathrm{c}}\hat{\mathbf{w}} + \tilde{\mathbf{s}}^{T}\mathbf{T}_{\mathrm{s}}\hat{\mathbf{w}} + \tilde{\mathbf{r}}^{T}\mathbf{T}_{\mathrm{r}}\hat{\mathbf{w}} + \varepsilon - u_{rc})\end{aligned} \tag{24}$$

where $\mathbf{b} = [0 \ldots 0\,1]^{T}$. In case of the existence of ε, consider a specified L^{2} tracking performance (Lee et al., 2005; Lin and Lin 2002; Wang et al., 2002)

$$\int_0^T \mathbf{E}^{T}\mathbf{Q}\mathbf{E}\,dt \le \mathbf{E}^{T}(0)\mathbf{P}\mathbf{E}(0) + \frac{\tilde{\mathbf{w}}^{T}(0)\tilde{\mathbf{w}}(0)}{\eta_1} + \frac{\tilde{\mathbf{c}}^{T}(0)\tilde{\mathbf{c}}(0)}{\eta_2} + \frac{\tilde{\mathbf{s}}^{T}(0)\tilde{\mathbf{s}}(0)}{\eta_3} + \frac{\tilde{\mathbf{r}}^{T}(0)\tilde{\mathbf{r}}(0)}{\eta_4} + \rho^{2}\int_0^T \varepsilon^{2}\,dt \tag{25}$$

where η_1, η_2, η_3 and η_4 are the positive constants, $T \in [0,\infty]$ and $\varepsilon \in L^{2}$. The κ is a design gain, ρ is a prescribed attenuation level, and the positive definite matrices $\mathbf{P}$ and $\mathbf{Q}$ satisfy the following Riccati-like equation

$$\mathbf{P}\mathbf{A} + \mathbf{A}^{T}\mathbf{P} + \mathbf{Q} + \mathbf{P}\mathbf{b}\left(\frac{1}{\rho^{2}} - \frac{2}{\kappa}\right)\mathbf{b}^{T}\mathbf{P} = \mathbf{0} \tag{26}$$

with $2\rho^2 \geq \kappa$. The design objective is to tune the parameters of SRNN to specify an adequate control law so that the worst effect of approximation error ε on tracking error vector $\mathbf{E}$ is guaranteed to be less than or equal to prescribed attenuation level ρ. If the system starts with initial conditions $\mathbf{E}(0)=0$, $\tilde{\mathbf{w}}(0)=0$, $\tilde{\mathbf{c}}(0)=0$, $\tilde{\mathbf{s}}(0)=0$ and $\tilde{\mathbf{r}}(0)=0$, then the L^2 tracking performance in (25) can be rewritten as

$$\sup_{\varepsilon \in L^2[0,T]} \frac{\int_0^T \mathbf{E}^T \mathbf{Q} \mathbf{E}\, dt}{\int_0^T \varepsilon^2\, dt} \leq \rho \,. \tag{27}$$

where the L^2-gain from ε to the tracking error $\mathbf{E}$ must be equal to or less than ρ. The following theorem can be stated and proved.

Theorem 1: Consider an *n*th-order nonlinear system expressed by (1). The control system is designed as (6), in which the adaptation laws of the neural controller are designed as

$$\dot{\hat{\mathbf{w}}} = -\dot{\tilde{\mathbf{w}}} = \eta_1 \mathbf{E}^T \mathbf{P} \mathbf{b} \hat{\mathbf{\Theta}} \tag{28}$$

$$\dot{\hat{\mathbf{c}}} = -\dot{\tilde{\mathbf{c}}} = \eta_2 \mathbf{E}^T \mathbf{P} \mathbf{b} \mathbf{T}_{\mathbf{c}} \hat{\mathbf{w}} \tag{29}$$

$$\dot{\hat{\mathbf{s}}} = -\dot{\tilde{\mathbf{s}}} = \eta_3 \mathbf{E}^T \mathbf{P} \mathbf{b} \mathbf{T}_{\mathbf{s}} \hat{\mathbf{w}} \tag{30}$$

$$\dot{\hat{\mathbf{r}}} = -\dot{\tilde{\mathbf{r}}} = \eta_4 \mathbf{E}^T \mathbf{P} \mathbf{b} \mathbf{T}_{\mathbf{r}} \hat{\mathbf{w}} \tag{31}$$

and the robust controller is designed as

$$u_{rc} = \frac{1}{\kappa} \mathbf{b}^T \mathbf{P} \mathbf{E} \tag{32}$$

then the stability of the system can be guaranteed.

Proof:

Consider a Lyapunov function in the following form

$$V(\mathbf{E}, \tilde{\mathbf{w}}, \tilde{\mathbf{c}}, \tilde{\mathbf{s}}, \tilde{\mathbf{r}}) = \frac{1}{2} \mathbf{E}^T \mathbf{P} \mathbf{E} + \frac{\tilde{\mathbf{w}}^T \tilde{\mathbf{w}}}{2\eta_1} + \frac{\tilde{\mathbf{c}}^T \tilde{\mathbf{c}}}{2\eta_2} + \frac{\tilde{\mathbf{s}}^T \tilde{\mathbf{s}}}{2\eta_3} + \frac{\tilde{\mathbf{r}}^T \tilde{\mathbf{r}}}{2\eta_4} \,. \tag{33}$$

Differentiating (33) with respect to time and using (24) and (28) ~ (31), it can be obtained that

$$\dot{V}(\mathbf{E}, \tilde{\mathbf{w}}, \tilde{\mathbf{c}}, \tilde{\mathbf{s}}, \tilde{\mathbf{r}}) = \frac{1}{2} \dot{\mathbf{E}}^T \mathbf{P} \mathbf{E} + \frac{1}{2} \mathbf{E}^T \mathbf{P} \dot{\mathbf{E}} + \frac{\tilde{\mathbf{w}}^T \dot{\tilde{\mathbf{w}}}}{\eta_1} + \frac{\tilde{\mathbf{c}}^T \dot{\tilde{\mathbf{c}}}}{\eta_2} + \frac{\tilde{\mathbf{s}}^T \dot{\tilde{\mathbf{s}}}}{\eta_3} + \frac{\tilde{\mathbf{r}}^T \dot{\tilde{\mathbf{r}}}}{\eta_4}$$

$$= \frac{1}{2} \mathbf{E}^T (\mathbf{A}^T \mathbf{P} + \mathbf{P} \mathbf{A}) \mathbf{E} + \mathbf{E}^T \mathbf{P} \mathbf{b} (\tilde{\mathbf{w}}^T \hat{\mathbf{\Theta}} + \tilde{\mathbf{c}}^T \mathbf{T}_{\mathbf{c}} \hat{\mathbf{w}} + \tilde{\mathbf{s}}^T \mathbf{T}_{\mathbf{s}} \hat{\mathbf{w}} + \tilde{\mathbf{r}}^T \mathbf{T}_{\mathbf{r}} \hat{\mathbf{w}} + \varepsilon - u_{rc})$$

$$+ \frac{\tilde{\mathbf{w}}^T \dot{\tilde{\mathbf{w}}}}{\eta_1} + \frac{\tilde{\mathbf{c}}^T \dot{\tilde{\mathbf{c}}}}{\eta_2} + \frac{\tilde{\mathbf{s}}^T \dot{\tilde{\mathbf{s}}}}{\eta_3} + \frac{\tilde{\mathbf{r}}^T \dot{\tilde{\mathbf{r}}}}{\eta_4}$$

$$= \frac{1}{2}\mathbf{E}^T(\mathbf{A}^T\mathbf{P}+\mathbf{PA})\mathbf{E}+\tilde{\mathbf{w}}^T(\mathbf{E}^T\mathbf{Pb}\hat{\mathbf{\Theta}}+\frac{\dot{\tilde{\mathbf{w}}}}{\eta_1})+\tilde{\mathbf{c}}^T(\mathbf{E}^T\mathbf{PbT_c}\hat{\mathbf{w}}+\frac{\dot{\tilde{\mathbf{c}}}}{\eta_2})$$

$$+\tilde{\mathbf{s}}^T(\mathbf{E}^T\mathbf{PbT_s}\hat{\mathbf{w}}+\frac{\dot{\tilde{\mathbf{s}}}}{\eta_3})+\tilde{\mathbf{r}}^T(\mathbf{E}^T\mathbf{PbT_r}\hat{\mathbf{w}}+\frac{\dot{\tilde{\mathbf{r}}}}{\eta_4})+\mathbf{E}^T\mathbf{Pb}(\varepsilon-u_{rc})$$

$$= \frac{1}{2}\mathbf{E}^T(\mathbf{A}^T\mathbf{P}+\mathbf{PA})\mathbf{E}+\mathbf{E}^T\mathbf{Pb}(\varepsilon-u_{rc}) \,. \tag{34}$$

Using (26) and (32), equation (34) can be rewritten as

$$\dot{V}(\mathbf{E},\tilde{\mathbf{w}},\tilde{\mathbf{m}},\tilde{\mathbf{s}},\tilde{\mathbf{r}}) = \frac{1}{2}\mathbf{E}^T(\mathbf{A}^T\mathbf{P}+\mathbf{PA}-\frac{2}{\kappa}\mathbf{Pbb}^T\mathbf{P})\mathbf{E}+\frac{1}{2}\varepsilon^T\mathbf{b}^T\mathbf{PE}+\frac{1}{2}\mathbf{E}^T\mathbf{Pb}\varepsilon$$

$$= \frac{1}{2}\mathbf{E}^T(-\mathbf{Q}-\frac{1}{\rho^2}\mathbf{Pbb}^T\mathbf{P})\mathbf{E}+\frac{1}{2}\varepsilon^T\mathbf{b}^T\mathbf{PE}+\frac{1}{2}\mathbf{E}^T\mathbf{Pb}\varepsilon$$

$$= -\frac{1}{2}\mathbf{E}^T\mathbf{QE}-\frac{1}{2}(\frac{1}{\rho}\mathbf{b}^T\mathbf{PE}-\rho\varepsilon)^T(\frac{1}{\rho}\mathbf{b}^T\mathbf{PE}-\rho\varepsilon)+\frac{1}{2}\rho^2\varepsilon^2$$

$$\le -\frac{1}{2}\mathbf{E}^T\mathbf{QE}+\frac{1}{2}\rho^2\varepsilon^2 \tag{35}$$

where $(\frac{1}{\rho}\mathbf{b}^T\mathbf{PE}-\rho\varepsilon)^T(\frac{1}{\rho}\mathbf{b}^T\mathbf{PE}-\rho\varepsilon)\ge 0$ and $\varepsilon^T\mathbf{b}^T\mathbf{PE}=\mathbf{E}^T\mathbf{Pb}\varepsilon$ are used. Integrating the above equation from $t=0$ to $t=T$, yields

$$V(T)-V(0)\le -\frac{1}{2}\int_0^T \mathbf{E}^T\mathbf{QE}\,dt+\frac{1}{2}\rho^2\int_0^T \varepsilon^2\,dt \tag{36}$$

Since $V(T)\ge 0$, the above inequality implies the following inequality

$$\frac{1}{2}\int_0^T \mathbf{E}^T\mathbf{QE}\,dt \le V(0)+\frac{1}{2}\rho^2\int_0^T \varepsilon^T\varepsilon\,dt \tag{37}$$

Using (34), this inequality is equivalent to inequality (25). Since $V(0)$ is finite if the approximation error $\varepsilon\in L^2$, that is $\int_0^T \varepsilon^2 d\tau<\infty$, it implies that $\lim_{t\to\infty}|\mathbf{E}|=0$.

In the following, the design algorithm of RNNAC with structure adaptation algorithm is summarized as follows:

Step 1: Initialize the pre-defined parameters of RNNAC.

Step 2: The tracking error is given in (2).

Step 3: The neural controller is given as (20), where the parameter are estimated by (28)-(31), respectively.

Step 4: The robust controller is given as (32).

Step 5: The control law is given as (6).

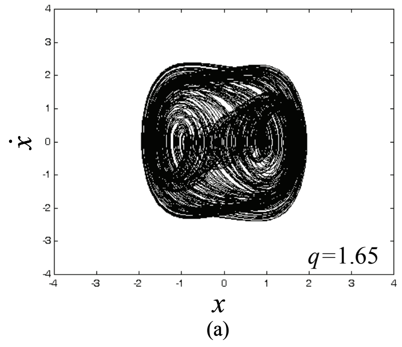
Step 6: Determine whether or not to add a new hidden neuron by $\Theta_{\max} \le \Theta_{th}$ condition, and determine whether or not to cancel a existing node by a significance index I_k.
Step 7: Return to Step 2.

4. Simulation results

Consider a second-order chaotic system such as the Duffing's equation describing a special nonlinear circuit or a pendulum moving in a viscous medium (Chen and Dong, 1993; Jiang, 2002)

$$\ddot{x} = f(\mathbf{x}) + u \tag{38}$$

where $f(\mathbf{x}) = -p\dot{x} - p_1 x - p_2 x^3 + q\cos(\omega t)$ is the system dynamics, *t* is the time variable, ω is the frequency, u is the control effort and p , p_1, p_2 and q are real constants. The chaotic dynamic system can be observed in many nonlinear circuits and mechanical systems.



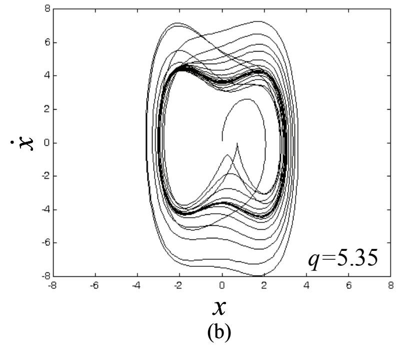


Fig. 4 Phase plane of uncontrolled chaotic system.

Recently, control of the chaotic dynamic system has become a significant research topic in the physics, mathematics and engineering communities. Chaotic dynamic system is a nonlinear deterministic system that displays complex, noisy-like and unpredictable behavior. Depending on the choice of these constants, it is known that the solutions of (38) may exhibit periodic, almost periodic and chaotic behavior. For observing the chaotic unpredictable behavior, the open-loop system behavior with $u = 0$ was simulated with

$p = 0.4$, $p_1 = -1.1$, $p_2 = 1.0$ and $\omega = 1.8$. The phase plane plots from an initial condition point (0, 0) are shown in Figs. 4(a) and 4(b) for $q = 1.65$ (chaotic) and $q = 5.35$ (period 1), respectively (Chen and Dong, 1993). It is shown that the uncontrolled chaotic dynamic system has different chaotic trajectories with different q values. The interest in the chaotic equation is the problem of how to design a controller to drive a chaotic trajectory to track a reference command closely.

The proposed RNNAC with structure adaptation algorithm is applied to control a nonlinear dynamic system. It should be emphasized that the development of the proposed control method does not need to know the system dynamics of the control system. A SRNN approximator is used to online estimate an ideal tracking controller with the online structuring and parameter learning algorithms. The structure learning possesses the ability of both adding and pruning hidden neurons, and the parameter learning adjusts the interconnection weights of neural network to achieve favorable approximation performance. The parameters of RNNAC are selected as $k_1 = 1$, $k_2 = 2$, $\eta_1 = 50$, $\eta_2 = \eta_3 = \eta_4 = 10$, $\sigma = 2.0$, $\Theta_{th} = 0.5$, $\tau = 0.01$, $\delta = 0.2$, and $I_{th} = 0.1$. The choices of these values are through some trials to achieve satisfactory control performance considering the requirement of stability and possible operating conditions. Properly choosing the values of k_1 and k_2, the desired system dynamics such as rise time, overshoot, and settling time can be easily designed by the second-order system shown in (4). The parameters η_1, η_2, η_3 and η_4 are the leaning rates of the interconnection weights. If the leaning rates are chosen to be small, then the parameters convergence of RNNAC will be easily achieved; however, this will result in slow learning speed. On the other hand, if the leaning rates are chosen to be large, then the learning speed will be fast; however, the RNNAC system may become more unstable for the parameter convergence. For a choice of $\mathbf{Q} = \mathbf{I}$, solve the Riccati-like equation (26) with $2\rho^2 = \kappa$, then

$$\mathbf{P} = \begin{bmatrix} 1.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}. \tag{39}$$

The simulation results of the RNNAC system with $\kappa = 1.0$ for $q = 1.65$ and $q = 5.35$ are shown in Figs. 5 and 6, respectively. The tracking responses of state x are shown in Figs. 5(a) and 6(a); the tracking responses of state $\dot{x}$ are shown in Figs. 5(b) and 6(b); the associated control efforts are shown in Figs. 5(c) and 6(c); and the numbers of hidden neurons are shown in Figs. 5(d) and 6(d), respectively. Simulation results show that the robust tracking performance of the proposed RNNAC system has been achieved. To attenuate an arbitrarily desired level via L^2 tracking design technique as small as possible. The simulation results of the proposed RNNAC system with $\kappa = 0.1$ for $q = 1.65$ and $q = 5.35$ are shown in Figs. 7 and 8, respectively. The tracking responses of state x are shown in Figs. 7(a) and 8(a); the tracking responses of state $\dot{x}$ are shown in Figs. 7(b) and 8(b); the associated control efforts are shown in Figs. 7(c) and 8(c); and the numbers of hidden neurons are shown in Figs. 7(d) and 8(d), respectively. From these simulation results, it can be seen that robust tracking performance can be also achieved without any knowledge of system dynamic functions; moreover, better system performance can be achieved as soon as the robust gain κ is decreased.

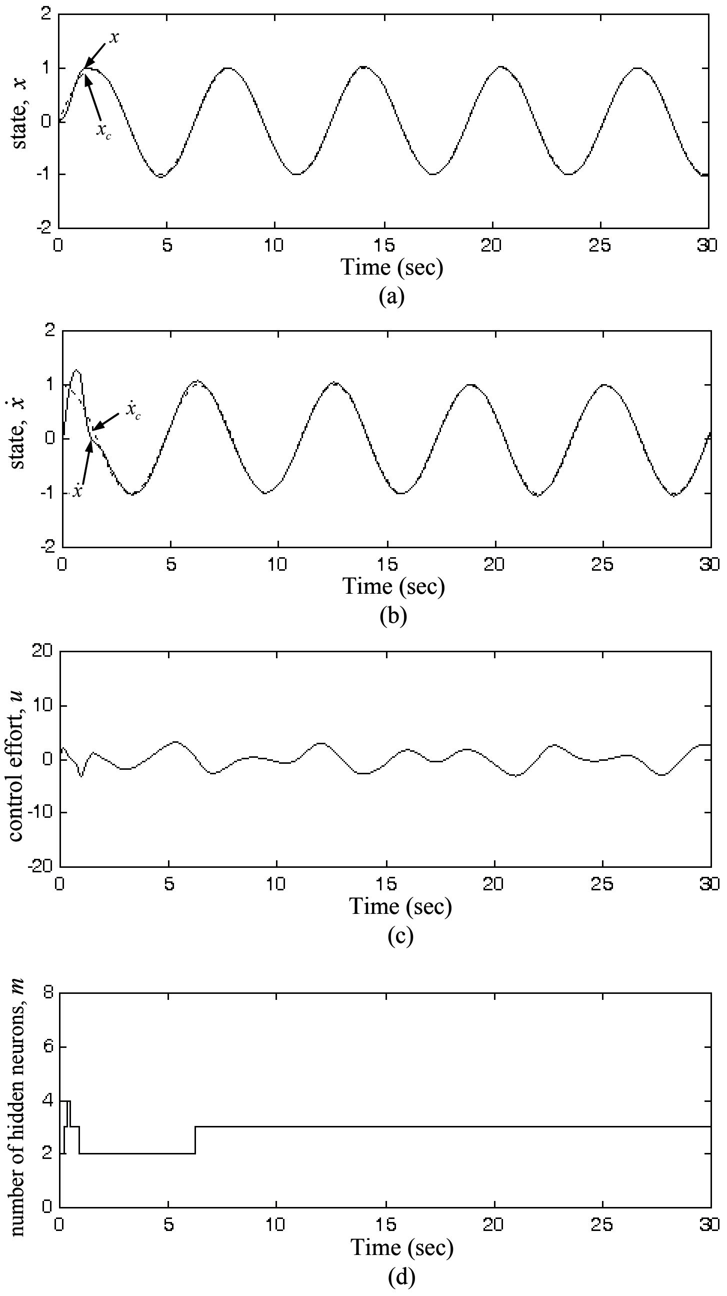


Fig. 5 Simulation results for $q = 1.65$ with $\kappa = 1.0$.

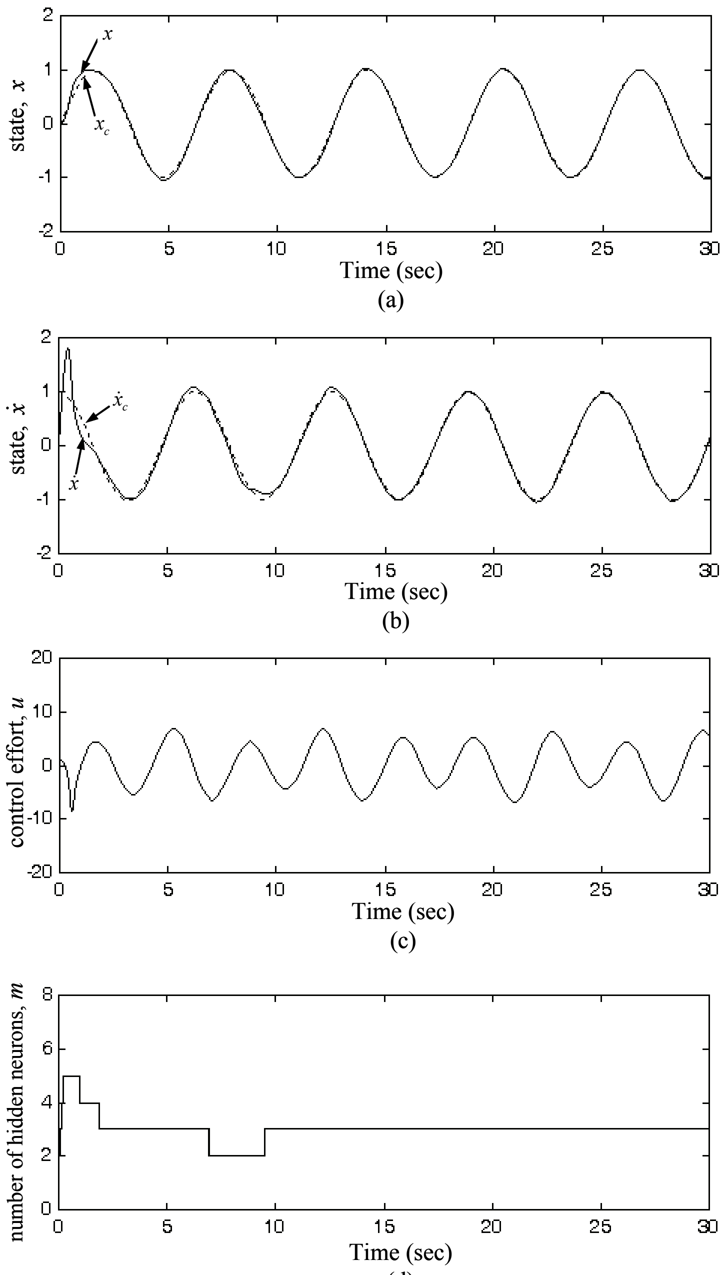


Fig. 6 Simulation results for $q = 5.35$ with $\kappa = 1.0$.

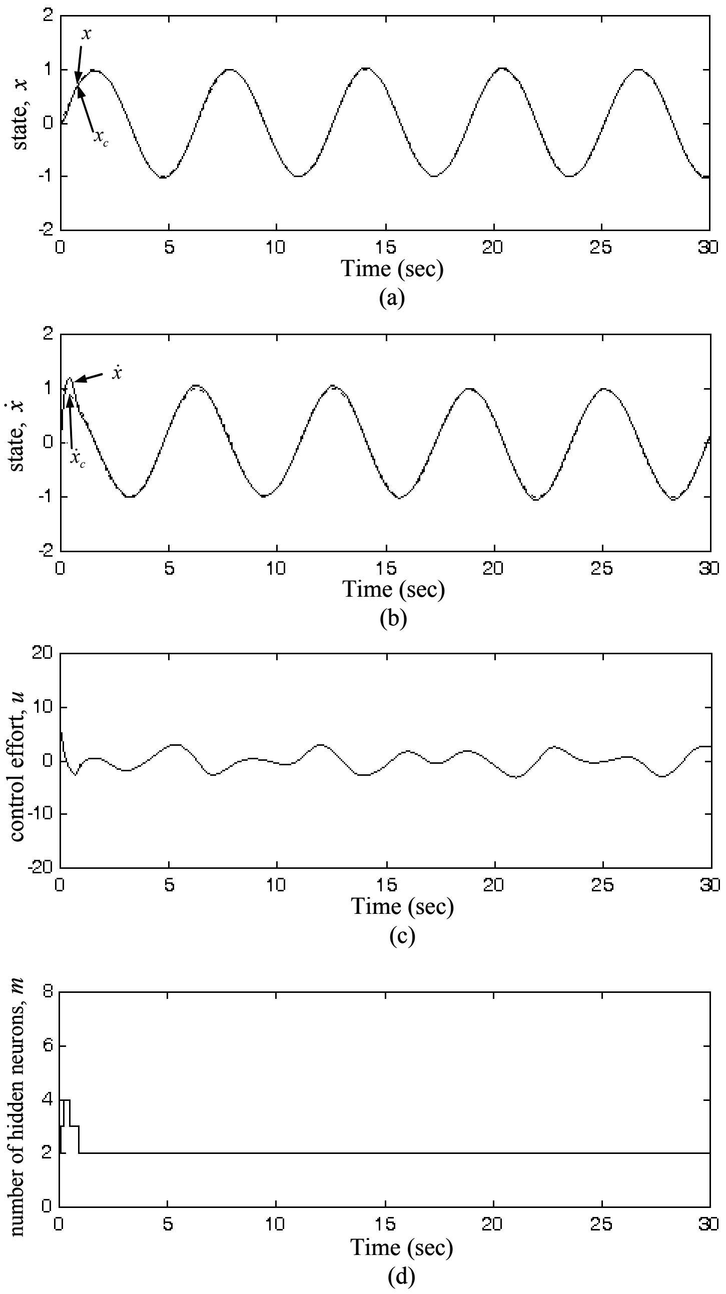
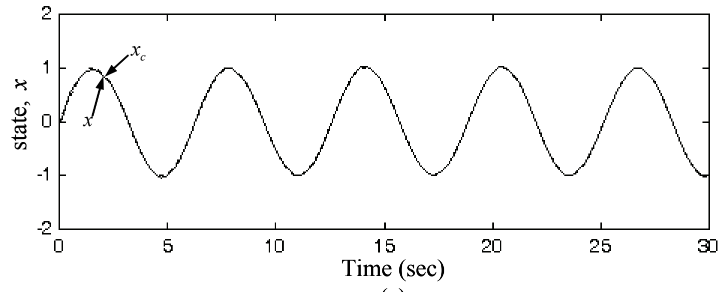
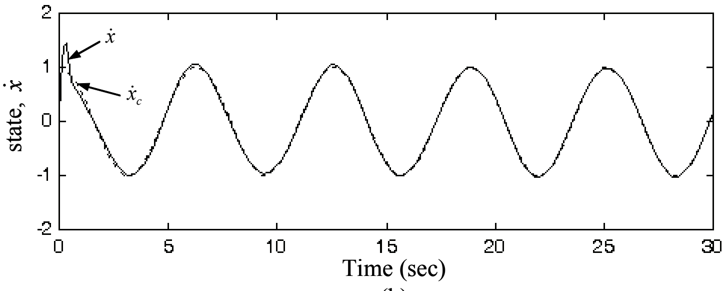


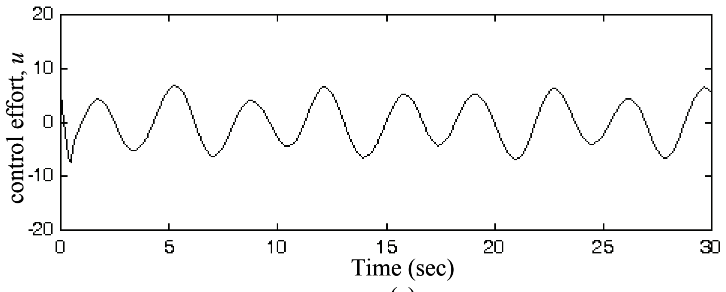
Fig. 7. Simulation results for $q = 1.65$ with $\kappa = 0.1$.



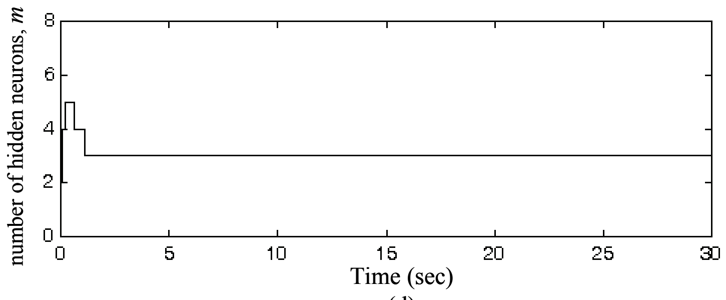
(a)



(b)



(c)



(d)

Fig. 8 Simulation results for $q = 5.35$ with $\kappa = 0.1$.

5. Conclusions

This paper develops a recurrent-neural-network-based adaptive control (RNNAC) system with structure adaptation algorithm, which is composed of a neural controller and a robust controller. In the neural controller design, a self-structuring recurrent neural network (SRNN) is utilized to mimic an ideal tracking controller. In the SRNN approximator, a dynamic generating and pruning mechanism of the neural stricture is developed to cope with the tradeoff between the approximation accuracy and computation load. The robust controller is designed to attenuate the effects of the approximation error on the tracking performance using L^2 tracking technique. Finally, the developed RNNAC system is used to control a nonlinear chaotic dynamic system to demonstrate its effectiveness. Simulation results indicate that a small attenuation level can be achieved if the magnitude of weighting factor κ is chosen small.

6. Acknowledgment

The authors appreciate the partial financial support from the National Science Council of Republic of China under grant NSC 95-2622-E-155-004-CC3.

7. References

Chen G, Dong X (1993) On feedback control of chaotic continuous-time systems. IEEE Trans Circuits Syst I 40 (9): 591-601

Duarte-Mermoud MA, Suarez AM, Bassi DF (2005) Multivariable predictive control of a pressurized tank using neural networks. Neural Comput Appl 15 (1): 18-25

Gao Y, Er MJ (2003) *Online adaptive fuzzy neural identification and control of a class of MIMO nonlinear systems.* IEEE Trans Fuzzy Syst 11 (4): 462-477

Huang GB, Saratchandran P, Sundararajan N (2004) An efficient sequential learning algorithm for growing and pruning RBF (GAP-RBF) networks. IEEE Trans Syst Man Cybern B Cybern 34 (6): 2284-2292

Hsu CF, Lin CM, and Lee TT (2006) Wavelet adaptive backstepping control for a class of nonlinear systems. IEEE Trans Neural Netw 17 (5): 1175-1183

Hsu CF (2007) Self-organizing adaptive fuzzy neural control for a class of nonlinear systems. IEEE Trans Neural Netw 18 (4): 1232-1241

Jiang ZP (2002) Advanced feedback control of the chaotic Duffing equation. IEEE Trans Circuits Syst I 49 (2): 244-249

Lee CH, Teng CC (2000) Indentification and control of dynamic systems using recurrent fuzzy neural networks. IEEE Trans Fuzzy Syst 8 (4): 349-366

Lee TS, Lin CH, Lin FJ (2005) An adaptive H^∞ controller design for permanent magnet synchronous motor drives. Control Eng Pract 13 (4): 425-439

Leung CS, Tsoi AC (2005) Combined learning and pruning for recurrent radial basis function networks based on recursive least square algorithms. Neural Comput Appl 15 (1): 62-78

Lin CL, Lin TY (2002) Approach to adaptive neural net-based H^∞ control design. IEE Proc Control Theory Appl 149 (4): 331-342

Lin CM, Hsu CF (2003) Neural network hybrid control for antilock braking systems. IEEE Trans Neural Netw 14 (2): 351-359

Lin CM, Hsu CF (2004) Supervisory recurrent fuzzy neural network control of wing rock for slender delta wings. IEEE Trans Fuzzy Syst 12 (5): 733-742

Lin CM, Chen, CH (2006) Adaptive RCMAC sliding mode control for uncertain nonlinear systems. Neural Comput Appl 15 (3): 253-267

Lin CT, Cheng WC, Liang SF (2005) *An on-line ICA-mixture-model-based self-constructing fuzzy neural network.* IEEE Trans Circuits Syst I 52 (1): 207-221

Lin FJ, Hwang WJ, Wai RJ (1999) A supervisory fuzzy neural network control system for tracking periodic inputs. IEEE Trans Fuzzy Syst 7 (1): 41-52

Lin FJ, Lin CH, Shen PH (2001) Self-constructing fuzzy neural network speed controller for permanent-magnet synchronous motor drive. IEEE Trans Fuzzy Syst 9 (5): 751-759

Park JH, Huh SH, Kim SH, Seo SJ, Park GT (2005) Direct adaptive controller for nonaffine nonlinear systems using self-structuring neural networks. IEEE Trans Neural Netw 16 (2): 414-422

Peng YF, Wai RJ, Lin CM (2004) Implementation of LLCC-resonant driving circuit and adaptive CMAC neural network control for linear piezoelectric ceramic motor. IEEE Trans Ind Electron 51 (1): 35-48

Tian L; Wang J, Mao Z (2004) *Constrained motion control of flexible robot manipulators based on recurrent neural networks.* IEEE Trans Syst Man Cybern B Cybern 34 (3): 1541-1552

Slotine J-JE, Li WP (1991) Applied Nonlinear Control. Prentice-Hall, Englewood Cliffs, NJ

Wai RJ, Lin CM, Peng YF (2004) Adaptive hybrid control for linear piezoelectric ceramic motor drive using diagonal recurrent CMAC network. IEEE Trans Neural Netw 15 (6): 1491-1506

Wang WY, Chan ML, Hsu CCJ, Lee TT (2002) H^{∞} tracking-based sliding mode control for uncertain nonlinear systems via an adaptive fuzzy-neural approach. IEEE Trans Syst Man Cybern B Cybern 32 (4): 483-492

Recurrent Fuzzy Neural Networks and Their Performance Analysis

R.A. Aliev[1], B. Fazlollahi[2], B.G. Guirimov[3] and R.R. Aliev[4]
[1,3]*Azerbaijan,*
[2]*USA,*
[4]*North Cyprus*

1. Introduction

There are many papers that consider different structure and training algorithms of FNN. Within their structural range, the networks may differ by type of signals (singleton, interval, general fuzzy, triangle shaped or other), topology (layered, fully-connected, with or without feed-back connections, feed-back connections in all or some of the layers etc), type of neurons (transfer function, same type in all layers or different depending on layer).
Note also that FNN is further complicated when we deal with applications of temporal character such as dynamic control, forecasting, identification, recognition of temporal sequences (e.g. voice recognition). It is obvious that in this case classical FNN with feed-forward structure, operable mainly for memory-less problems, would be ineffective. In this respect there is a strong demand for recurrent fuzzy neural networks (RFNN) with dynamic mapping capability, temporal information storage, dynamic fuzzy inference, and as a result, capable of solving temporal problems [7,24,27].
Paper [48] discusses delay feedback neuro-fuzzy networks and their usability to effectively tackle dynamic systems. This is a simplified version of recurrent network with feedback connections at only one layer of the network. To train unknown parameters of RFNN the author of [1] uses a supervised learning algorithm that requires differentiability of the membership functions that is not always possible.
In [25] a recurrent self-organizing neuro-fuzzy inference network is proposed. The main characteristic of this system is the ability to deal with temporal problems including dynamic fuzzy inference. The system with on-line learning feature is capable also of building the structure and (crisp) parameters of the network. The learning algorithm is based on the use of the ordered derivative (partial derivative) produced with the use of an ordered set of equations. The efficiency of the proposed neuro-fuzzy system is verified on the basis of various simulations on benchmark temporal problems, including time-sequence prediction, adaptive noise cancellation, dynamic plant identification, and non-linear plant control.
In [27] a recurrent multi-layered connectionist network for realizing fuzzy inference using dynamic fuzzy rules is presented. The paper distinguishes as containing good methodological support encompassing important aspects of neuro-fuzzy systems class. The back-propagation algorithm is used as the learning algorithm minimizing the cost function to achieve necessary connection weights and biases. As in [25], several examples and

performance comparisons with the existing works are presented including time sequence prediction, identification of non-linear dynamic system, identification of a chaotic system, and adaptive control of a non-linear system.

It should be noted that in [27] feedback links in the second layer only are added to the fuzzy feed-forward neural network. This rather simplified version of neuro-fuzzy network has crisp feed-forward connection weights and non-adjustable recurrent connection weights in the second layer. These simplifications undoubtedly lead to some decrease in the efficiency of the proposed neuro-fuzzy network.

In [35] a dynamic neuro-fuzzy system consisting of recurrent TSK rules is investigated. The suggested network is trained by dynamic fuzzy neural constrained optimization method based on the concept of constrained optimization. The proposed dynamic neuro-fuzzy system is tested on two temporal examples and the noise cancellation problem.

In [31] a hybrid supervisory control system using a recurrent neuro-fuzzy network, with the network output feeding back to the network input through time-delay units, is proposed. An on-line training methodology which is based on Lyapunov stability theorem and the gradient descent method is proposed. Some simulated and experimental results are provided to demonstrate the efficiency of the proposed neuro-fuzzy system.

Recurrent neuro-fuzzy systems for implementation of long-range prediction fuzzy model is investigated in [54]. In this recurrent neuro-fuzzy model the network output is fed back to the network input through one or more time delay units. Levenberg-Marquardt algorithm with regularization is used for adjusting crisp weights and biases of the feed-forward and feed-back connections of the recurrent neuro-fuzzy network. The suggested neuro-fuzzy network is applied to modeling and control of a neutralization process.

In [50] a direct adaptive iterative learning control system based recurrent neuro-fuzzy network is presented. The analysis of stability and learning is studied. A computer simulation for an inverted pendulum system and Chua's chaotic circuit is demonstrated.

A sliding mode recurrent neuro-fuzzy network based control system is proposed in [30] to control the mover of a permanent-magnet linear synchronous motor. The learning algorithm used is the same as in [31].

Interesting design methods and applications of FRNN are discussed in [18,26,29,34,55]. In paper [34] a discrete mathematical model of RFNN is constructed and a learning algorithm adopting a recursive least square approach is used to identify the unknown parameters in the model. In [18] the authors propose an efficient algorithm for determination of structure of model and identification of its parameters with the aim of producing improved predictive performance for NARMAX (nonlinear autoregressive moving average with exogenous inputs) time series models. A fuzzified TSK (Takagi-Sugeno-Kang) type recurrent fuzzy network is developed in paper [26] for one-dimensional and two-dimensional fuzzy temporal sequence prediction. Paper [29] considers a design method of recurrent fuzzy neural network based adaptive hybrid control for multi-input multi-output linearized dynamic systems. The proposed control system is applied to aircraft flight control system. Paper [55] deals with adaptive nonlinear noise control systems using recurrent fuzzy neural networks, the feedback connections of which are used to create dynamic fuzzy rules trained using dynamic back-propagation learning algorithm. The learning of fuzzy weights of FRNN is not considered in these works as all the papers assume network weights to be crisp numbers.

In [20] a self-organizing adaptive fuzzy neural network for nonlinear systems is proposed. The identifier is used to estimate the controlled system's dynamic with the learning of fuzzy neural network. The parameter learning algorithms are derived based on Lyapunov function candidate.

A very important role in designing fuzzy neural networks takes its learning method and the problem of how to train fuzzy neural networks (FNN) has great scientific and practical interest and is becoming challenging and important research area.

The training methods for neural networks can be divided into two large categories: gradient-based algorithms and evolutionary algorithms. The overview of the works on training methods for fuzzy feed-forward neural networks is given in [10]. Work [32] needs special note as presenting some methodological support from the viewpoint of fuzzy neural networks. Paper [32] develops two learning algorithms for fuzzy feed-forward neural networks that is the fuzzy back-propagation algorithm and the fuzzy conjugate gradient (CG) algorithm for determination of fuzzy weights and biases represented as Π -type fuzzy numbers. The authors use GA for determination of optimal learning rate at each iteration step of fuzzy CG algorithm. Some real simulations realizing non-dynamic fuzzy inference rules and fuzzy functions are demonstrated.

The evolutionary algorithms based approach to training of FNN involves application of genetic algorithms and other population-based natural evolution inspired algorithms to minimize error function and determine the fuzzy connection weights and biases [10,28]. In contrast to BP and other supervised learning algorithms, evolutionary algorithms do not use the derivative information, and hence, they are most effective in case where the derivative is very difficult to obtain or even unavailable. Moreover, the calculation complexity of BP algorithms is high due to the need for computing complex Hessian or Jacobian matrices.

In [47] nonlinear neural network predictive control strategy based on chaotic particle swarm optimization is presented. It is shown that since the back-propagation algorithm is easily trapped in local minima and its convergence performance greatly depends on its learning rate parameter and initial conditions, the weights and biases of the neural network are optimized by particle swarm optimization algorithm. Learning of crisp weights and biases is considered in this work.

TSK-type recurrent neuro-fuzzy system trained by GA is proposed in [24]. In this network internal variables, derived from fuzzy firing strengths are fed back to both network input and output layers. To train the proposed TSK-type recurrent neuro-fuzzy network, a GA based method is developed. The recurrent neuro-fuzzy network with genetic learning is applied to dynamic system control problem. The research in this field is at its infancy and many fundamental problems such as choosing the most efficient error function, coding technique, and genetic strategies remain to be solved [33].

Unfortunately, little progress has been made in the development of recurrent fuzzy neural networks processing directly fuzzy information and using fuzzy weights and biases as adjustable parameters. For the first time some attempts were made in [5,6,8-10,22] to develop an efficient RFNN with fuzzy inputs, fuzzy weights expressed as fuzzy numbers, and fuzzy outputs. In this study we consider the structure, operation, and DE-based training algorithm for multi-layer recurrent fuzzy neural network processing fuzzy signals and demonstrate its efficiency on a number of benchmark and application problems.

The rest of this paper is organized as follows. In section 2 we cover prerequisite material (such as fuzzy function, Hamming distance, fuzzy neural networks, differential evolution optimization, etc.) to be used in the study. Section 3 formulates the statement of problem of creating RFNN with efficient learning algorithm. Section 4 illustrates the structure and computational procedure of the investigated recurrent fuzzy neural network. In section 5 the recurrent fuzzy neural network learning algorithm using DEO is described. Simulations and experimental results are discussed in section 6. Section 7 gives the conclusion of this paper.

2. Preliminaries

In this section, we briefly review some prerequisite material which will be of help in the development of the concepts of evolutionary computing based learning of RFNN. While the reader may find some of the definitions in the literature, we augment them with some interpretation which could be useful in the context of our considerations.

2.1 Fuzzy function

Briefly speaking, by a fuzzy function we mean a function, whose values are fuzzy numbers. Let *f* be a fuzzy function, $\mu_{f(x)}$ denotes the membership function of the fuzzy number $f(x)$, and for $0<\alpha\le 1$, $f_+^\alpha(x)$ will denote sup$\{z\in dom(\mu_{f(x)}):\ \mu_{f(x)}(z)\ge\alpha\ \}$ and $f_-^\alpha(x)$ will denote inf$\{z\in dom(\mu_{f(x)}):\mu_{f(x)}(z)\ge\alpha\ \}$. Functions $f_-^\alpha(x)$ and $f_+^\alpha(x)$ are level functions of *f*.

A fuzzy subset *A* of R^n is defined in terms of its membership function $\mu_A(x):R^n\to[0,1]$

For each $\alpha\in(0,1]$ the α -level set $[\mu_A(x)]^\alpha$ of a fuzzy set *A* is the subset of points $x\in R^n$ with membership values $\mu_A(x)$ of at least α, that is $[\mu_A(x)]^\alpha=\{x\in R^n:\ \mu_A(x)\ge\alpha\}$.

2.2 Distance

Formally, the distance *d(x,y)* between *x* and *y* in R^n is considered to be a two-argument function satisfying the conditions: *d(x,y)* ≥ 0, for every *x* and *y*; *d*(*x,x*)=0, for every *x*; $d(x,z)\ge d(x,y)+d(y,z)$ for every pattern *x*, *y* and *z*. In the case of continuous variables we have a long list of distance functions [7, 40].

Let us consider the space E^n of all fuzzy subsets of R^n which satisfy the conditions of normality, convexity and are upper semicontinuous with compact supports $[\mu_A(x)]^{\alpha=0}$. For fuzzy sets *A* and *B* in E^n, in general, the Minkowski distance defined as follows

$$d(A,B)=\sqrt[p]{\int_X |\mu_A(x)-\mu_B(x)|^p\,dx},\ p\ge 1, \tag{1}$$

where **X** - is a universe of discourse. This distance satisfies the above mentioned conditions. In particular, when $p=1$ we get Hamming distance.

2.3 Fuzzy neural networks and neuro-fuzzy systems

Fuzzy neural network (FNN) approach has become a powerful tool for solving real-world problems in the area of forecasting, identification, control, image recognition and others that are associated with high level of uncertainty [2,7,10,11,14,23,24,23]. This is related with the fact that the FNN paradigm combines the capability of fuzzy reasoning in handling uncertain information and the capability of pure neural networks in learning from experiments [48]. An advantage of FNN is that it allows automation of design of fuzzy rules and combined learning of numerical data as well as expert knowledge expressed as fuzzy IF-THEN rules [7]. FNN may have smaller network size and be faster in convergence speed as compared with ordinary NN.

There are two different approaches in academic literature. First approach is neuro-fuzzy systems whose main task is to process numerical relationships [27]. Many papers, including papers [31,37,48] combine features of neural and fuzzy approaches into Neuro-Fuzzy systems. Second approach is fuzzy neural systems with the objective to process both numerical (measurement based) information and perception based information. The FNN of this (second) class are oriented for real-world problems that are inherently uncertain and imprecise [10,19,32]. It is necessary to point out that neuro-fuzzy systems cannot replace fuzzy neural systems because unlike the former which perform mapping from non-fuzzy input signals to non-fuzzy outputs, the latter process linguistic information directly.

When we deal with linguistic information, i.e. work at a higher data abstraction level, we should employ fuzzy neural networks, not neuro-fuzzy networks, to solve the considered problem approximately [21].

2.4 Differential evolution optimization method

Recently many heuristic algorithms have been proposed for global optimization of nonlinear, non-convex, and non-differential functions [3,12,41,51]. These methods are more flexible than classical as they do not require differentiability, continuity, or other properties to hold for optimizing functions. Some of such methods are genetic algorithm, evolutionary strategy, particle swarm optimization, and differential evolution (DE) optimization. In this study we consider the use of the DE algorithm.

As a stochastic method, DE algorithm uses initial population randomly generated by uniform distribution, differential mutation, probability crossover, and selection operators [42]. The population with *ps* individuals are maintained with each generation. A new vector is generated by mutation which in this case is randomly selecting from the population 3 individuals: $r_1 \neq r_2 \neq r_3$ and adding a weighted difference vector between two individuals to a third individual (population member).

The mutated vector is then undergone crossover operation with another vector generating new offspring vector.

The selection process is done as follows. If the resulting vector yields a lower objective function value than a predetermined population member, the newly generated vector will replace the vector with which it was compared in the following generation.

Extracting distance and direction information from the population to generate random deviations results in an adaptive scheme with excellent convergence properties. DE has been successfully applied to solve a wide range of problems such as image classification, clustering, optimization etc.

Figure 1 shows the process of generation new trial solution vector from randomly selected population members. Here we assume that the solution vectors are of dimension 2 (i.e. 2 optimization parameters).

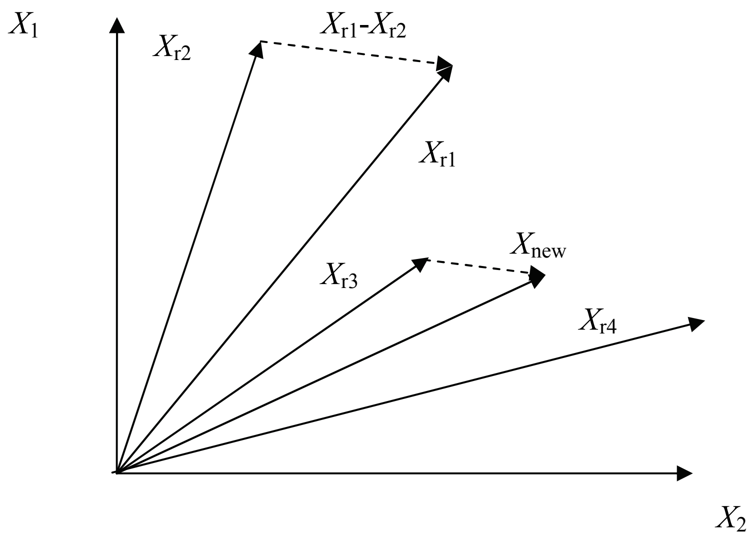


Figure 1

3. Statement of problem

Assume that an unknown nonlinear system is expressed as follows [4]:

$$\tilde{y}(t) = \tilde{g}\big(\tilde{y}(t-1),...,\tilde{y}(t-n),\tilde{u}(t-1),...,\tilde{u}(t-m)\big), \tag{2}$$

where $\tilde{y}(t)$ and $\tilde{u}(t)$ are the output and input of the system, respectively, represented as fuzzy valued function, $\tilde{g}(.)$ is an unknown nonlinear fuzzy mapping to be estimated by RFNN, *n* and *m* are order of the system. It is required to design RFNN such that its output $\tilde{y}_N(t)$ determined as

$$\tilde{y}_N(t) = \tilde{g}_N\left(\tilde{y}(t-1),\tilde{u}(t-1),\tilde{W},\tilde{V},\tilde{\theta}\right), \tag{3}$$

will be as close as possible to $\tilde{y}(t)$ (1), where $\tilde{W},\tilde{V},\tilde{\theta}$ collectively define the structure and set of parameters of RFNN: forward connection weights, backward (recurrent) connection weights, and biases, respectively.

As measure of closeness between $\tilde{y}(t)$ and $\tilde{y}_N(t)$ we need to define a suitable error function serving as a distance (metric). For continuous variables there is a long list of distance functions [4,40]. In this paper we will use the well-known and commonly used Hamming distance. Therefore the problem of learning of FRNN is an optimization problem with the purpose of adjusting fuzzy parameters $\tilde{W} = \{\tilde{w}_{lij}\}$, $\tilde{V} = \{\tilde{v}_{lij}\}$, and $\tilde{\theta} = \{\tilde{\theta}_{li}\}$ to minimize the error function

$$\widetilde{E} = \sum\sum\left|\widetilde{y}_{pi} - \widetilde{y}_{Npi}\right|, \tag{4}$$

where $\widetilde{y}_{pi}$ is the desired value and $\widetilde{y}_{Npi}$ is the actual value of RFNN output layer's neuron i when applied training patter p, $\widetilde{E}$ is hamming distance.

Training algorithm is critical to RFNN as it will affect RFNN approximation capability. Due to the type of error function, we cannot use here the BP algorithm. Another problem with the BP is that it is easily trapped in local minimima and its convergence performance greatly depends on its learning rate parameter and the initial conditions. As optimization strategy for training RFNN we will use an evolutionary computing strategy, namely, DEO method.

During the training, the weights of feed-forward and feed-back connections and biases of RFNN are optimized by the differential evolution algorithm which would lead to the minimum of error function (4).

It is worth to note that using clustering based differential evolution algorithm for training of RFNN may give a higher performance [51]. Such an approach will be used on considering a petrol production forecasting example.

4. Recurrent fuzzy neural network structure and computation

The general structure of a recurrent fuzzy neural network is presented in Figure 2. The box elements represent memory cells that store values of activation of neurons at previous time step, which is fed back to the input at the next time step.

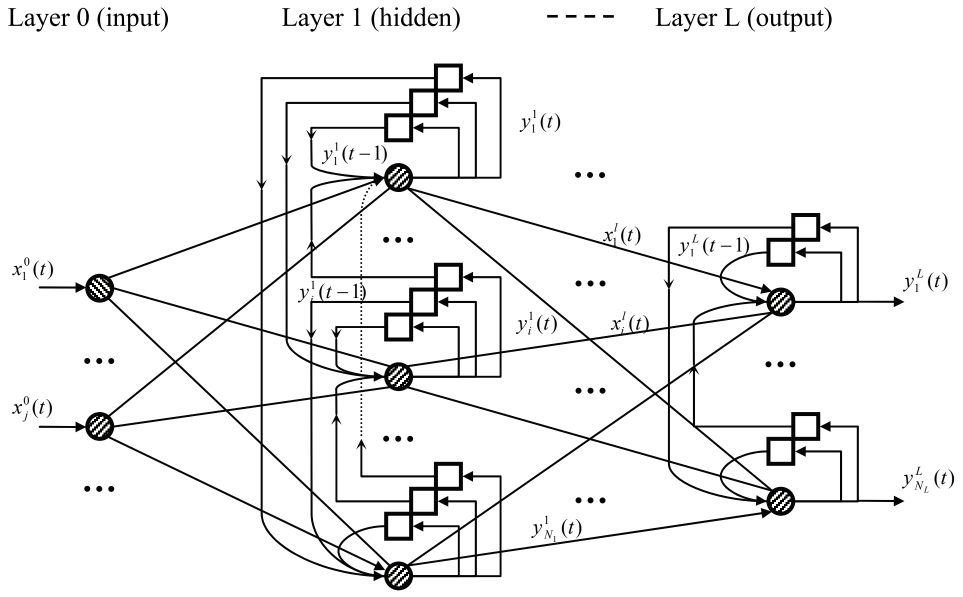


Figure 2. The structure of RFNN

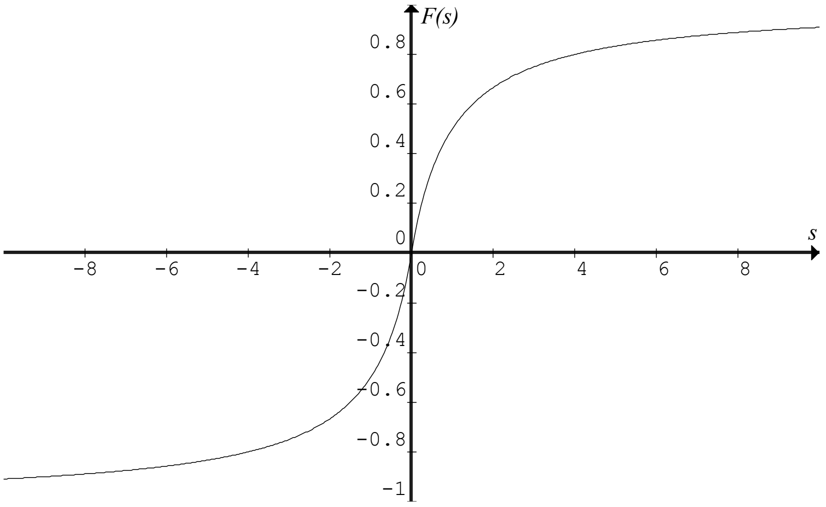


Figure 3. The activation function *F*(*s*)

In general, the network may have virtually any number of layers. We number the layers successively from 0 (the first or input layer) to *L* (last or output layer). The neurons in the input layer (layer 0) only distribute the input signals without modifying their values.

$$\tilde{y}_i^0(t) = \tilde{x}_i^0(t) \tag{5}$$

The neurons in the remaining layers (layer 1 to layer *L*-1) are dynamic and compute their output signals as follows:

$$\tilde{y}_i^l(t) = F\left(\tilde{\theta}_i^l + \sum_j \tilde{x}_j^l(t)\tilde{w}_{ij}^l + \sum_j \tilde{y}_j^l(t-1)\tilde{v}_{ij}^l\right), \tag{6}$$

where $\tilde{x}_j^l(t)$ is *j*-th fuzzy input to the neuron *i* at layer *l* at the time step *t*, $\tilde{y}_i^l(t)$ is the computed output signal of the neuron at the time step *t*, $\tilde{w}_{ij}$ is the fuzzy weight of the connection to neuron *i* from neuron *j* located at the previous layer, $\tilde{\theta}_i$ is the fuzzy bias of neuron *i*, and $\tilde{y}_j^l(t-1)$ is the activation of neuron *j* at the time step (*t*-1), $\tilde{v}_{ij}$ is the recurrent connection weight to neuron *i* from neuron *j* at the same layer.
The neurons at the last layer (layer *L*) are linear and their outputs are calculated as:

$$y_i^L(t) = \theta_i^L + \sum_j x_j^L(t)w_{ij}^L + \sum_j y_j^L(t-1)v_{ij}^L \tag{7}$$

The activation *F* for a total input to the neuron *s* (figure 3) is calculated as:

$$F(s) = \frac{s}{1+|s|} \tag{8}$$

So, the output of neuron *i* at layer *l* ($l = \overline{1, L-1}$) is calculated as follows:

$$y_i^l(t) = \frac{\tilde{\theta}_i^l + \sum_j \tilde{x}_j^l(t)\tilde{w}_{ij}^l + \sum_j \tilde{y}_j^l(t-1)\tilde{v}_{ij}^l}{1 + \left| \tilde{\theta}_i^l + \sum_j \tilde{x}_j^l(t)\tilde{w}_{ij}^l + \sum_j \tilde{y}_j^l(t-1)\tilde{v}_{ij}^l \right|} \tag{9}$$

The total number of connections (including forward, recurrent, and biases) is equal to $(N_I + 1)N_H + (N_H + 1)N_O + N_H^2 + N_O^2$,

where N_I is number of inputs (here we use 1 input for simplicity), N_H is the number of neurons in the second (hidden) layer of RFNN and N_O is the number of outputs. Thus, for a network with 1 input, 3 hidden neurons, and 1 output, there will be 20 connections overall. Every forward connection weight, recurrent connection weight, and bias value are represented as a triangular fuzzy number [16,17,37-39]: $\tilde{w}_{ij}^l = T(w_{Lij}^l, w_{Aij}^l, w_{Rij}^l)$, $\tilde{v}_{ij}^l = T(v_{Lij}^l, v_{Aij}^l, v_{Rij}^l)$, $\tilde{\theta}_i^l = T(\theta_{Li}^l, \theta_{Ai}^l, \theta_{Ri}^l)$, respectively. Note that the forward connection is from neuron j at layer (*l*-1) to neuron *i* at layer *l*, while the recurrent connections are between the outputs and inputs of the neurons at the same layer. Triangle fuzzy numbers are described as *T*(*a*,*b*,*c*), where [*a*,*c*] is the fuzzy number support and *b* is the value with membership equal to 1 (the average value).

Inputs to RFNN x_i can accept system outputs at previous stages, $x_1 = y_N(t-1),\ x_2 = y_N(t-2), \ldots, x_r = y_N(t-r)$, etc., as well as exogenous signals $x_{r+1} = u(t),\ x_{r+2} = u(t-1),\ x_{r+3} = u(t-2), \ldots$.

For example, for constructing a RFNN based time series predictor ($\hat{F}_{NN}$) $\hat{y}_{t+1} = \hat{F}_{NN}(y_t, y_{t-1}, \ldots, y_{t-n+1}, u_t, \ldots, u_{t-1}, u_{t-m+1})$, where y_t is the value of time series data at time interval *t*, u_t is the value of an additional (second) factor at time interval *t*, the above presented structure could be modified as given in Figure 4.

In case the original learning patterns are crisp, we need to sample data into fuzzy terms, i.e. to fuzzify the learning patterns. The fuzzifiers can be created independently for specific problems. A different approach that is used in this paper is to convert the numeric data into information granules by fuzzy clustering [40]. In this case the receptive fields forming the input layer of RFNN are constructed using clustering. Fuzzy clusters fully reflect the character of the data.

5. Differential evolution optimization based learning of RFNN

The considered RFNN requires the global parameter optimization method suitable for nonlinear, non-convex, and non-differentiable mapping functions. Ideally, we want to find the global minimum of (4) and this requires more careful selection of the optimization engine. Despite the fact that the gradient descent based methods are predominant, they are not global optimizers. Most suitable are population based optimization techniques including genetic algorithms, evolutionary strategy, particle swarm optimization, DEO, etc. In this paper we use DEO method which has many advantages over other evolutionary algorithms and GA [3,42,51]. There are some reasons for using DEO in RFNN learning problem. First, DEO supports a search mechanism of global nature. DEO is useful when dealing with different distance functions including Hamming distance, Tschebyshev distance (gradient descent based methods require distance functions be differentiable, e.g. Euclidean distance function).

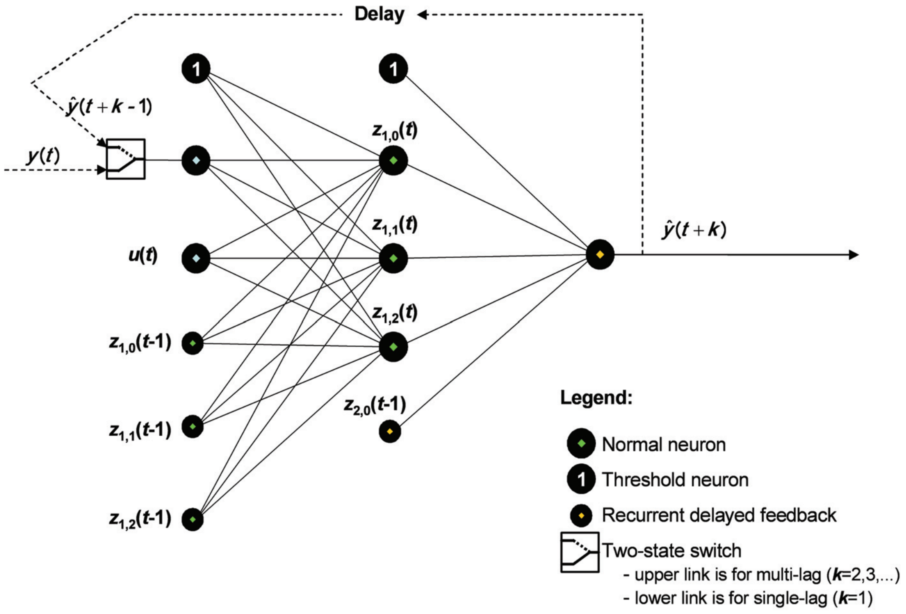


Figure 4. The structure of a simple FRNN

For application of an evolutionary algorithm for learning RFNN we consider the population individ to represent a whole combination of weights ($\widetilde{W} = \{\widetilde{w}_{lij}\}$, $\widetilde{V} = \{\widetilde{v}_{lij}\}$) and biases ($\widetilde{\theta} = \{\widetilde{\theta}_{li}\}$) (i.e. parameters of RFNN) defining the input/output mapping (3). The population maintains a number of popential parameter sets defining different RFNN solutions and recognizes one of these solutions to be the best solution. This best solution is the one with minimum training error. After a series of generations, the best solution may converge to a near-optimum solution, which would represent in our case a RFNN with the required accuracy.

To apply an evolutionary population based optimization algorithm we first should identify the optimized parameter vector. For training RFNN we need to optimize values of: forward connection weights, processing (hidden and output) neuron biases, and recurrent weights. According to the structure of RFNN given in section 2, the number of all parameters to be adjusted during the learning process and therefore the dimension of the optimized parameter vector (for a FRNN with one hidden layer) is

$$N_{par} = (N_I + 1)N_H + (N_H + 1)N_O + N_H^2 + N_O^2 . \tag{10}$$

Before starting training all the parameters are initialized by randomly chosen values, usually not beyond the interval [-1,1]. This constraint is further enforced to the parameters associated to backward connections. It means that during further training steps the values of forward weights and biases can go beyond the interval [-1,1] while the values of backward connection are kept within this interval. This additional constraint is added to make RFNN stable which means that under the constant input the value of output will converge to a constant value (either crisp or fuzzy).

Prior launching the optimization we set parameter *f* of DEO to a positive value (typically about 0.9), define the DEO cost function to be the RFNN error function (4), and choose the population size (typically ten times the number of optimization parameters, i.e. $10N_{par}$).

Then the differential evolution optimization is started.

DEO based RFNN training algorithm can be summarized as follows:

```
Step 0. Initialize DE
        Step 0.0 Define the structure of RFNN: $N_i$, $N_h$, $N_o$
        Step 0.1. Construct template parameter vector X of dimension $N_{par}$ according (10) for
                  holding RFNN weights and biases: $X=\{\widetilde{W},\widetilde{V},\widetilde{\theta}\}$
        Step 0.2. Set algorithm parameters: $f$ (mutation rate), $cr$ (crossover rate), and $ps$ (size
                  of population)
        Step 0.3. Define the cost function as function of error function of current RFNN
                  parameters: $\widetilde{E} = \sum\sum\left|\widetilde{y}_{pi} - \widetilde{y}_{Npi}\right|$
Step 1. Randomly generate $ps$ parameter vectors (from respective parameter spaces (e.g. in
        the range [-1, 1]) and form a population $P=\{X_1, X_2, ..., X_{ps}\}$
Step 2. While Termination condition (number of predefined generations reached or required
        error level obtained) is not met generate new parameter sets:
        Step 2.1. Choose a next vector $X_i$ ($i$=1,...,$ps$)
        Step 2.2. Choose randomly different 3 vectors from $P$: $X_{r1}$, $X_{r2}$, $X_{r3}$ each of which is
                  different from current $X_i$
        Step 2.3. Generate trial vector $X_t=X_{r1}+f(X_{r2}-X_{r3})$
        Step 2.4. Generate new vector from trial vector $X_t$. Individual vector parameters of $X_t$
                  are inherited with probability $cr$ into the new vector $X_{new}$. If the cost function
                  from $X_{new}$ is better (lower) than the cost function from $X_i$, current $X_i$ is
                  replaced in population $P$ by $X_{new}$
         Next i
Step 3. Select the parameter vector $X_{best}$ (RFNN parameter set) with best cost (training error
        $\widetilde{E}$) function from population $P$. Extract from $X_{best}$ vectors $\widetilde{W},\widetilde{V},\widetilde{\theta}$ defining weights
        and thresholds for RFNN
Step 4. Stop the algorithm
```

If the obtained total error performance index or the behavior of the obtained network is not desired, we can restructure the network by adding new hidden neurons, or do better granulation of the learning patterns.

During the DE optimization process the solutions resulting in lower cost values have more chances to survive and be saved into a new population for participation in future generations. The process is repeated iteratively. During succeeding generations we keep into the population the solution that produced the lowest value of cost function of all previous generations. The farther we go with generations the higher is the chance to find a better solution.

6. Experiments and application

In this section, we report on the results of simulation of the suggested RFNN with DEO learning and compare the performance of RFNN and existing approaches. The performance of the proposed algorithm is examined on 3 benchmark problems in literature [13,27,46].

6.1 Non-linear system identification

We start with non-linear system studied in [13,28,32,45] as a benchmark identification problem.
The dynamic system is described by the equation:

$$y(k)=g(y(k\text{-}1),\ y(k\text{-}2))+u(k) \tag{11}$$

where:

$$g(y(k-1),y(k-2))=\frac{y(k-1)y(k-2)(y(k-1)-0.5)}{1+y^2(k-1)+y^2(k-2)} \tag{12}$$

The system output depends on both its past values and current input. The goal is to approximate the model (11)-(12) by RFNN.
The RFNN for this example has 2 input neurons, 6 neurons at layer 1 and one output neuron. The number of all connections (including forward, backward, and biases) was 62. On the basis of (12) 400 data were created using random (in interval [-1,1] signal u and used for training. The trained network was tested on the basis of 200 test data created using (12) by applying sinusoidal signal $u=\sin(2\pi k/25)$.
DE Optimization progress (MSE vs. successful iterations) is shown in Figure 5.
Table 1 shows a fragment of results (reached MSE) from intensive simulation experiments.

Experiment	MSE on train data	MSE on test data
1	0.0000818477	0.000342153
2	0.000116419	0.000738948
3	0.0000761331	0.000361591
4	0.0000235756	0.0000656312
5	0.000373168	0.00105125
6	0.000103995	0.000215761
7	0.0000696317	0.000204256

Table 1. RFNN training simulations for non-linear system identification

The final reached MSE at the best experiment was 0.000024 on training data and 0.000066 on test data. Table 2 presents comparative results obtained by different methods given in literature.

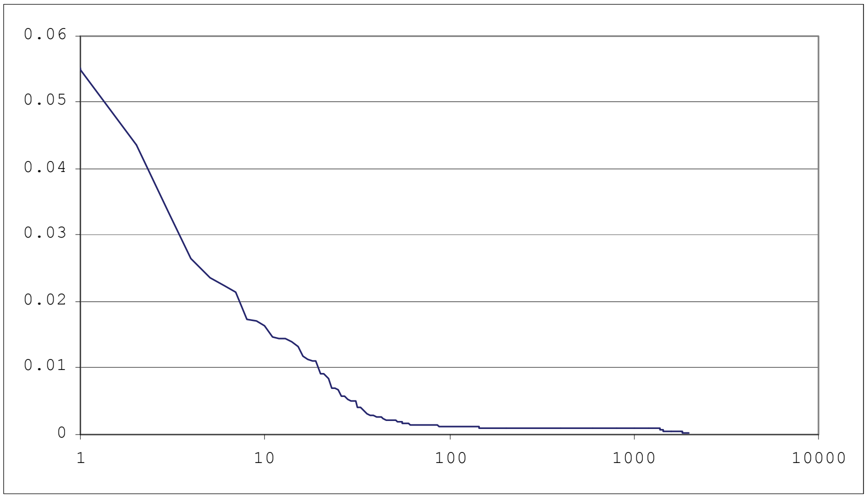


Figure 5. RFNN error convergence

Reference fuzzy model	MSE on training set	MSE on test set
[53]	-	0.00080
[45]	0.00075	0.00035
[13]	0.00010	0.00032
RFNN (our approach)	0.000024	0.000066

Table 2. Comparative results by different methods

The comparison of the actual and identified curves for y(k) is illustrated in Figure 6.

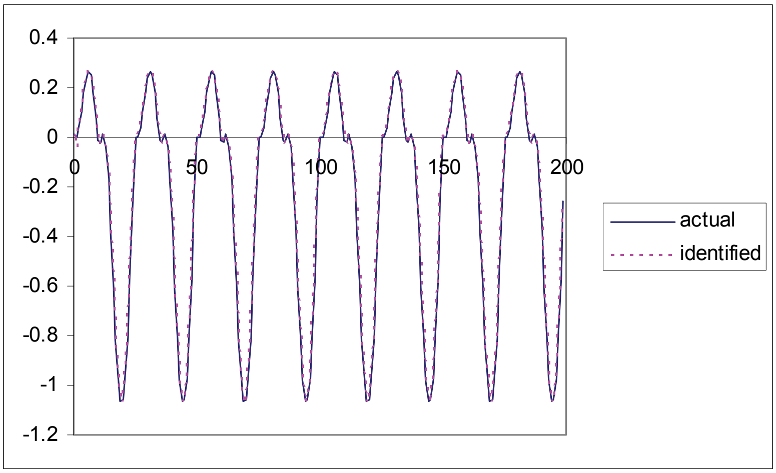


Figure 6. RFNN identification performance

6.2 Dynamic plant identification

This example is taken from [25,27] in which a nonlinear plant with multiple time-delay is identified. The nonlinear plant is described as follows:

$$y_p(k+1) = f(y_p(k), y_p(k-1), y_p(k-2), u(k), u(k-1)) \tag{13}$$

where

$$f(x_1, x_2, x_3, x_4, x_5) = \frac{x_1 x_2 x_3 x_5 (x_3 - 1) + x_4}{1 + x_2^2 + x_3^2} \tag{14}$$

In this example the output depends on three previous outputs and two previous inputs. For better results 2000 data were used for training generated by applying random $u(k)$ in interval [-1, 1].

For the testing signal $u(k)$ the following equation was used:

$$u(k) = \begin{cases} \sin\left(\dfrac{\pi k}{25}\right), & 0 \le k < 250 \\ 1, & 250 \le k < 500 \\ -1, & 500 \le k < 750 \\ 0.3\sin\left(\dfrac{\pi k}{25}\right) + 0.1\sin\left(\dfrac{\pi k}{32}\right) + 0.6\sin\left(\dfrac{\pi k}{10}\right), & 750 \le k < 1000 \end{cases} \tag{15}$$

Figure 7 shows the comparison of the desired test and RFNN output curves of the considered dynamic system.

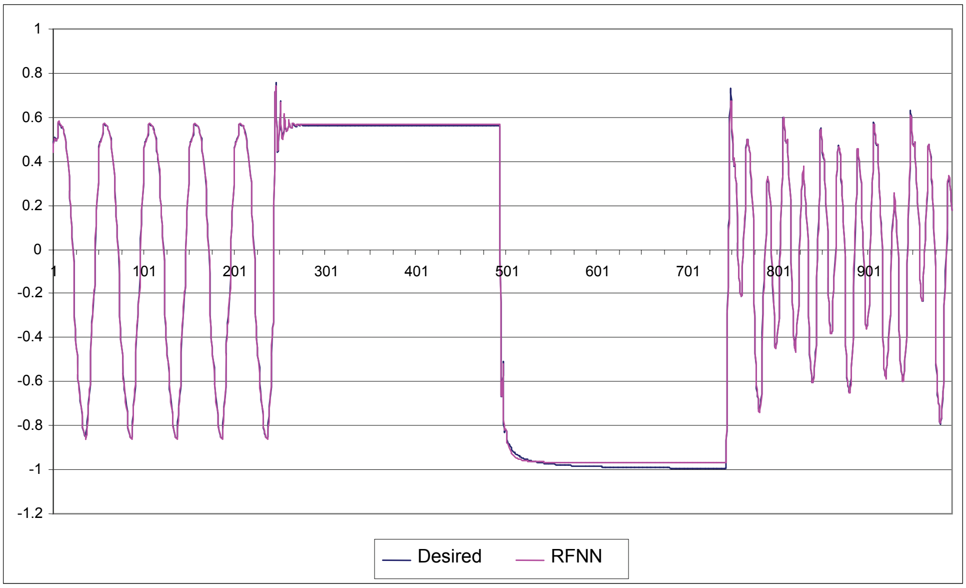


Figure 7. Comparison of output of RFNN with the desired output

The network used two input neurons ($y(k)$ and $u(t)$, respectively), 8 hidden neurons, and one output neuron ($y(k+1)$). In case of a non-recurrent FNN we would need 5 input neurons ($y(k)$, $y(k-1)$, $y(k-2)$, $u(t-1)$) and 1 output neuron ($y(k+1)$). In comparison with a regular FNN, the use of RFNN allows significant simplification of the network structure. Table 3 below shows the comparison of characteristics of fuzzy neural networks suggested in [27] and our RFNN.

	RFNN [27]	FNN [27]	RFNN (our approach)
No of inputs	2	5	2
No of outputs	1	1	1
Nodes	51	112	11
Parameters	112 (crisp)	176 (crisp)	96 (fuzzy triangle numbers)
MSE	0.00013	0.003	0.00004048

Table 3. Comparison performance of different FNN models for dynamic plant identification

In addition, RFNN is more accurate. The simulation using the suggested RFNN demonstrates that the identification error (MSE) is less than the error with other approaches (Table 3).
It can be concluded that DEO learning based RFNN outperform its comparing rivals [25,27] exhibiting considerably lower MSE. In terms of model complexity, the considered RFNN model has lower number of nodes than the models presented in [27].

6.3 Sun-spot prediction

The performance of FRNN was also tested on a well-known problem of sun-spot prediction [8,46]. Sunspot numbers rise and fall with an irregular cycle with a length of approximately 11 years. In addition to this, there are variations over longer periods. The recent trend is upward from 1900 to the 1960s, then somewhat downward. The historical data for this problem were taken from the Internet. Several data sets were prepared as in [8,46]. The data used for training were sun-spot data from years 1700 to 1920. Two unknown prediction sets used for testing were from 1921 to 1955 (PR1) and from 1956 to 1979.
The comparison of performance of the FRNN approach with other existing methods for two different datasets (PR1, PR2) is presented in Table 4 (NMSE i.e. the Normalized Mean Square Error measure is used in these experiments). The last two rows in Table 4 were obtained by two networks trained on the same data sets by two different persons independently (indicated RFNN-1 and RFNN-2, respectively). In RFNN-1 and RFNN2 the total numbers of neurons were 9 (1+7+1) and 13 (1+11+1), respectively. The numbers of connections for RFNN-1 and RFNN-2 were 148 and 179, respectively.
Table 4 presents comparative results on performance of different forecasting methods for sun-spot prediction problem.
As can be seen from Table 4, the suggested RFNN has simpler structure (having only 1 input neuron) than other models. The identification error of the RFNN is less than that of existing models applied to sun-spot forecasting problem.

Author (Method)	Number of inputs	PR1	PR2
Rementeria (AR) [44]	12	0.126	0.36
Tong (TAR) [49]	12	0.099	0.28
Subba Rao (Bilinear) [43]	9	0.079	-
DeGroot (ANN)[15]	4	0.092	-
Nowland (ANN) [36]	12	0.077	-
Rementeria (ANN) [44]	12	0.079	0.34
Waterhouse (HME) [52]	12	0.089	0.27
(RFNN-1)	1	0.066	0.22
(RFNN-2)	1	0.074	0.21

Table 4. MSE obtained by different models for sun-spot prediction

6.4 Application of RFNN to forecast demand for petrol

In this example the problem is to forecast demand for petrol (A92) for optimal scheduling of an oil refinery plant [56]. In our fuzzy forecasting model we assumed the relationship:

$$y(k+1)=F(y(k-2), y(k-1), y(k)) \tag{16}$$

For this example we used actual daily data from existing oil refinery plant for a month period. Approximately 80% of the data (chosen randomly) were used for clustering and training and the remaining data were used for testing of RFNN.

Usually, the structure of RFNN is determined by trial-and-error in advance for the reason that it is difficult to consider the balance between the number of rules and desired performance [20]. In this study, to determine the structure of RFNN, first we convert numeric data into information granules by fuzzy clustering. The number of clusters defines the number of fuzzy rules. By applying the fuzzy C-means clustering method [13,40] on the training data and checking the validity measure suggested in [13] it was identified that an adequate number of clusters is 4. Therefore 4 fuzzy rules were used for the basis for training and further refining. The clustering algorithm identified the following cluster centers for the presented data.

IF y(t-2) is A1 AND y(t-1) is B1 AND y(t) is C1 THEN y(t+1) is D1
IF y(t-2) is A2 AND y(t-1) is B2 AND y(t) is C2 THEN y(t+1) is D2
IF y(t-2) is A3 AND y(t-1) is B3 AND y(t) is C3 THEN y(t+1) is D3 (17)
IF y(t-2) is A4 AND y(t-1) is B4 AND y(t) is C4 THEN y(t+1) is D4

Initial fuzzy terms A1, A2, A3, A4 were created from the component y(t-2) of the cluster vectors 1, 2, 3, and 4, respectively. Similarly, terms B1, B2, B3, B4 – from y(t-1), C1, C2, C3, C4 – from y(t), and D1, D2, D3, D4 – from y(t+1). The terms A1, A2, ...,B1, B2, ..., C1, C2,...D1, D2, ... are described linguistically.

DEO based training allowed to further decrease MSE of output (forecasting of petrol) after clustering making it ten times lower. The final MSE after training was 0.0008.

6.5 Application of RFNN to control battery charging process

The FRRN designed for battery charging control has 4 inputs, 20 hidden neurons, and 1 output. The four used inputs represent temperature (*T*), change of temperature (*dT*), voltage

(U) and change of voltage (dU). The output of the controller is the current (I) applied for charging the battery.
The network has been trained on the basis of the data base (collected by a separate work group over a year period) contained data series formed of measured temperature, voltage, and current readings from many charging experiments with different batteries.
The proposed control system allows very quick and effective charge of the battery: the charging time is reduced from more than 2000 seconds (with applied constant charge current 2A) to 860 seconds (or even less, if the temperature limit is set higher than 25°C) with dynamically changed (under the control of the proposed intelligent controller) input current. Also the battery is protected from overheating and a long utilization time of the battery can be provided by adequately adjusting the fuzzy rules describing the desired charging process. The results of proposed charging controller compared with other battery chargers for a particular charging experiment (with the same initial conditions) are given in Table 5. The value of decrease in charging time and heating level was $18.0 \pm 2.2\%$ and $5 \pm 0.6\%$, respectively, compared to other methods.

Charging controller	Time (sec)	T_{end}-T_{start}
Proposed approach	860	2,85
FL [4]	(no data)	35-60
FG [14]	959	9
ANFIS [9]	900	50
NeuFuz [16]	1200-1800	5

Table 5. Comparison of different charging controllers

7. Conclusions

In spite of great importance of fuzzy neural networks for solving wide range of real-world problems, unfortunately, little progress has been made in their development.
In this study we have discussed recurrent neural networks with fuzzy weights and biases as adjustable parameters and internal feedback loops, which allows capturing dynamic response of a system without using external feedback through delays. In this case all the nodes are able to process linguistic information.
As the main problem regarding fuzzy and recurrent fuzzy neural networks that limits their application range is the difficulty of proper adjustment of fuzzy weights and biases, we put an emphasize on the RFNN training algorithm.
We have proposed the standard DEO-based method for learning of recurrent fuzzy neural network. The optimization method, customized for RFNN training, compares favorably with the existing gradient-based error minimization method as it is less complex and is more likely to locate the global minimum of network error. As the method does not require derivative information, it is very effective in case when dealing with different distance functions. Also, the considered global optimization algorithm can provide high accuracy of fuzzy mapping with relatively smaller network size.
The RFNN was tested on a number of benchmark identification and time-series forecasting problems well-known in the literature as well as on application problems. Experimental results demonstrated very good performance on all considered problems.

8. References

R.H. Abiyev, "Recurrent Neural Network Based Fuzzy Inference System for Identification and Control of Dynamic Plants", in: Proceedings of International XII Turkish Symposium on Artificial Intelligence and Neural Networks, Vol. 1, No. 1, 2003, 31-39.

D.F. Akhmetov, Y. Dote, S. Ovaska, "Fuzzy Neural Network with General Parameter Adapation for Modelig of Nonlinear Time-Series". IEEE Transactions on Neural Networks, Vol. 12, No. 1, (2001).

F.S.Al-Anzi, A.Allahverdi, "A self-adaptive differential evolution heuristic for two-stage assembly scheduling problem to minimize maximum lateness with setup times", European Journal of Operational Research, 182 (2007), 80-94.

R.A.Aliev, R.R.Aliev, "Soft Computing and Its Applications", World Scientific, 2001, 465 p.

R.A.Aliev, R.R.Aliev, B.G.Gurimov, K.Uyar, "Dynamic Data Mining Technique for Battery Charging Rules Extraction", Applied Soft Computing Journal (2006), http://dx.doi.org/10.1016/j.asoc.2007.02.015.

R.A.Aliev, R.R.Aliev, B.G.Gurimov, K.Uyar, "Reccurent Neural Network Based System for Battery Charging", Lecture Notes in Computer Science, Springer Berlin, Volume 4492 (2007), 307-316.

R.A. Aliev, B. Fazlollahi, R.R. Aliev, "Soft Computing and Its Applications in Business and Economics", Springer Verlag, 2004, 420 p.

R.A.Aliev, B.Fazlollahi, R.R.Aliev, B.G.Gurimov, "Fuzzy Time Series prediction Method Based on Fuzzy Recurrent Neural Network", Lecture Notes in Computer Science (2006), 860-869.

R.A.Aliev, B.Fazlollahi, R.R.Aliev, B.Gurimov, "Linguistic time series forecasting using fuzzy recurrent neural network", International Soft Computing Journal, Volume 12, Issue 2 (2007), 183-190.

R.A.Aliev, B.Fazlollahi, R.Vahidov, "Genetic Algorithm-Based Learning of Fuzzy Neural Networks. Part 1: Feed-Forward Fuzzy Neural Networks", Fuzzy Sets and Systems 118 (2001), 351-358.

G. Bortolan, "An Architecture of Fuzzy Neural Networks for Linguistic Processing", Fuzzy Sets and Systems, 100 (1998), 197-215.

W.-D. Chang, "Nonlinear system identification and control using a real-coded genetic algorithm", Applied mathematical Modeling 31 (2007), 541-550.

M.-Y. Chen, D.A.Linkens, "Rule-base self-generation and simplification for data-driven fuzzy models", Fuzzy Sets and Systems 142 (2004), 243-265.

O. Cordon, F. Herrera, M. Lozano, "On the Combination of Fuzzy Logic and Evolutionary Computation: a Short Review and Bibliography", in: W.Pedrycz (Ed.), Fuzzy Evolutionary Computation, Kluwer Academic Publishers, Dordrecht, 1997, 33-56.

W.D.DeGroot, "Analysis of Univariate Time Series with Connectionist Nets: A Case Study of Two Classical Examples", Neurocomput., Vol. 3, (1991), 177-192.

D. Dubois, H. Prade (Eds.), "The Handbooks of Fuzzy Sets Series", Kluwer Acad. Publ., 2000.

D. Dubois, H. Prade, R.R. Yager (Eds.), "Fuzzy Information Engineering: A Guided Tour of Applications", Wiley, New York.

Y.Gao, M.J.Er, "NARMAX time series model prediction: Feedforward and recurrent fuzzy neural network approaches", Fuzzy Sets and Systems, Vol. 150, Issue 2 (2005), 331-350.

Y. Hayashi, J. Buckley, E. Czogola, "Fuzzy Neural Networks with Fuzzy Signals and Weights", Internat. J. Intell. Systems 8 (1993), 527-537.

C.-F.Hsu, "Self-organizing adaptive fuzzy neural control for a class of nonlinear systems", IEEE Trans. on Neural Networks, Vol. 18, No. 4 (2007), 1232-1241.

J.-R. Hwang, S.-M. Chan, C.-H. Lee, "Handling Forecasting Problems Using Fuzzy Time Series", Fuzzy Sets and Systems 100 (1998), 217-228.

P.D. Ionescu, M. Moscalu, A. Moscalu, "Intelligent Charger with Fuzzy Logic", in: Proc. of Int. Symp. on Signals, Circuits and Systems, 2003.

H.Ishubuchi, K.Morioka, H.Tanaka, "A Fuzzy Neural Network with Trapezoid Fuzzy Weights", IEEE, New York, 1994, 228-233.

C-F.Juang, "A TSK-Type Recurrent Fuzzy Network for Dynamic Systems Processing by Neural Network and Genetic Algorithms", IEEE Transactions on Fuzzy Systems, Vol. 10, No. 2 (2002), 155-170.

C.-F. Juang, C.-T. Lin. "A Recurrent Self-Organizing Neural Fuzzy Inference Network", IEEE Transactions on Neural Networks, Vol. 10, No. 4 (1999).

C.-F. Juang, K.-C. Ku, "A recurrent network for fuzzy temporal sequence processing and gesture recognition", IEEE Trans. On Systems, Man, and Cybernetics, Part. B: Cybernetics, Vol. 35, Issue 4 (2005), 646-658

C.-H. Lee, C.-C. Teng, "Identification and Control of Dynamic Systems Using Recurrent Fuzzy Neural Networks", IEEE Transactions on Fuzzy Systems, Vol. 8, No. 4 (2000), 349-366.

F.H.F. Leung, H.K. Lam, S.H. Ling, P.K.S. Tam, "Tuning of the Structure and Parameters of a Neural Network Using an Improved Genetic Algorithm", IEEE Transactions on Neural Networks, Vol. 14, No. 1 (2003).

C.M. Lin, C.H. Chen, Y.F.Lee, "Recurrent fuzzy neural network adaptive hybrid control for linearized multivariable systems", Journal of Intelligent and Fuzzy Systems, Vol. 17, Number 5 (2006), 479-491.

F.-J.Lin, P.-K.Huang, "Recurrent fuzzy neural network controller design using sliding-mode control for linear synchronous motor drive", IEE Proc.-Control Theory Appl., Vol. 151, No. 4 (2004).

F.-J.Lin, R.-J. Wai, C.-M. Hong, "Hybrid Supervisory Control Using Recurrent Fuzzy Neural Network for Tracking Periodic Inputs", IEEE Transactions on Neural Networks, Vol. 12, No. 1 (2000).

P.Liu, H. Li. "Efficient Learning Algorithms for Three-Layer Regular Feedforward Fuzzy Neural Network", IEEE Transactions on Neural Networks, Vol. 15, No 3, (2004), 545-558.

P.Liu, H.Li, "Fuzzy neural network theory and applications", World Scientific, Singapore (2004), 376 p.

C.-H. Lu, C.-C Tsai, "Generalized predictive control using recurrent fuzzy neural networks for industrial purposes", Journal of Process Control 17 (2007), 83-92

P.Mastorocostas, "A Recurrent Fuzzy Neural Model for Dynamic System Identification", IEEE Transactions on Systems, Man and Cybernetics—Part B: Cybernetics, Vol. 32, No. 2 (2002) 176-190.

S.Nowland., G.Hinton, "Simplifying Neural Networks by Soft Weight-Sharing", Neural Comput., Vol. 4, No. 4 (1992), 473-493.

B.-J.Park, S.-K. Oh, W. Pedrycz, H.-K. Kim, "Design of Evolutionary Optimized Rule-Based Fuzzy Neural Networks Based on Fuzzy Relation and Evolutionary Optimization", in: International Conference on Computational Science (3), 2005, 1100-1103.

W.Pedrycz, "Computational Intelligence. An Introduction", CRC Press, 1997.

W.Pedrycz, "Fuzzy Control and Fuzzy Systems" (second, extended, edition). John Wiley and Sons, New York, 2003.

W.Pedrycs, "Knowledge-Based Clustering: From Data to Information Granules", John Wiley & Sons (2005), 316 p.

W.Pedrycz, K.Hirota, "Fuzzy vector quantization with particle swarm optimization: A study in fuzzy granulation-degranulation information processing", Signal Process (2007), doi:10.1016/j.sigpro.2007.02.001.

K.Price, R.Storn, J.Lampinen, "Differential evolution – a practical approach to global optimization, Springer, Berlin (2005).

S.Rao., M.M.Gabr, "An Introduction to Bispectral Analysis and Bilinear Time Series Models", in Lecture Notes in Statistics. Springer-Verlag, Vol. 24 (1984).

S.Rementeria., X.Olabe, "Predicting Sunspots with a Self-Configuring Neural System", in Proc. 8th Int. Conf. Information Processing Management Uncertainty Knowledge-Based Systems, 2000.

M. Setnes, H. Roubos, "GA-fuzzy modeling and classi&cation: complexity and performance", IEEE Trans. Fuzzy Systems 8 (5) (2000) 509–522.

A.Sfetsos, C.Siriopoulos, "Time Series Forecasting with Hybrid Clustering Scheme and Pattern Recognition", IEEE Trans. on Systems, Man, and Cybernetics – part A: Systems and Humans, Vol. 34, No. 3 (2004), 399-405.

Y.Song, Z.Chen, Z.Yuan, "New chaotic PSO-based neural network predictive control for nonlinear process", IEEE Trans. on Neural Networks, Vol. 18, No. 2 (2007), 595-600.

S.-F.Su, F.-Y. P. Yang, "On the Dynamic Modeling with Neural Fuzzy Networks", IEEE Transactions on Neural Networks, Vol. 13, No. 6 (2002).

H.Tong, K.S.Lim, "Threshold Autoregression, Limit Cycle and Cyclical Data", Int. Rev. Statist. Soc. B, Vol. 42 (1980).

Y.-C. Wang, C.-J. Chien, and C.-C. Teng, "Direct Adaptive Iterative Learning Control of Nonlinear Systems Using an Output-Recurrent Fuzzy Neural Networks", IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, Vol. 34, No. 3 (2004), 1348-1359.

Y.-J.Wang, J.-S.Zhang, G.-Y. Zhang, "A dynamic clustering based differential evolution algorithm for global optimization", European Journal of Operational Research 183 (2007), 56-73.

S.R.Waterhouse, A.J.Robinson, "Non-Linear Prediction of Acoustic Vectors Using Hierarchical Mixtures of Experts", in Advances of Neural Information Processing Systems. Cambridge, MA: MIT Press, Vol. 7 (1995).

S.Wu, M.J.Er, "Dynamic fuzzy neural networks—a novel approach to function approximation", IEEE Trans. SMC-B 30 (2) (2000) 358–364.

J. Zhang, "Recurrent Neuro-Fuzzy Networks for Nonlinear Process Modeling", IEEE Transactions on Neural Networks, Vol. 10, No. 2 (1999), 313-325.

Q.-Z. Zhang, W.-S. Gan, Y.-L. Zhou, "Adaptive recurrent fuzzy neural networks for active noise control", Journal of Sound and Vibration, Vol. 296, Issue 4-5 (2006), 935-948.

Sh.Mehdi, "Fuzzy time series forecasting", Fourth World Conference on Intelligent Systems for Industrial Automation, 2006, 342-347.

Recurrent Interval Type-2 Fuzzy Neural Network Using Asymmetric Membership Functions

Ching-Hung Lee and Tzu-Wei Hu
Department of Electrical Engineering
Yuan-Ze University
Taiwan

1. Introduction

The fuzzy systems and control are regarded as the most widely used application of fuzzy logic systems in recent years (Jang, 1993; John & Coupland, 2007; Lin & Lee, 1006; Mendel, 2001; Wang, 1994). The structure of traditional fuzzy system models that is characterized by using type 1 fuzzy sets, which are defined on a universe of discourse, map an element of the universe of discourse onto a precise number in the unit interval [0, 1]. The concept of type-2 fuzzy sets was initially proposed by Zadeh as an extension of typical fuzzy sets (called type-1) (Zadeh, 1975). Mendel and Karnik developed a complete theory of interval type-2 fuzzy logic systems (iT2FLSs) (Karnik et al, 1999; Liang & Mendel, 2000; Mendel, 2001). Recently, T2FLSs have attracted more attention in many literatures and special issue of IEEE Transactions on Fuzzy systems (Baldwin & Karake, 2003; John & Coupland, 2007; Lee & Lin, 2005; Liang & Mendel, 2000; Mendel, 2001, Hagras, 2007; Ozen & Garibaldi, 2004; Pan et al, 2007; Wang et al, 2004).
T2FLSs are more complex than type-1 ones, the major difference being the present of type- is their antecedent and consequent sets. T2FLSs result better performance than type-1 Fuzzy Logic Systems (T1FLSs) on the applications of function approximation, modeling, and control. In addition, neural networks have found numerous practical applications, especially in the areas of prediction, classification, and control (Lee & Teng, 2000; Lin & Lee, 1996; Narendra & Parthasarathy, 1990). The main aspect of neural networks lies in the connection weights which are obtained by training process. Based on the advantages of T2FLSs and neural networks, the type-2 neural fuzzy systems are presented to handle the system uncertainty and reduce the rule number and computation (Castillo & Melin, 2004; Lee & Lin, 2005; Mendel, 2001; Pan et al, 2007; Wang et al, 2004). Besides, recurrent neural network has the advantages of store past information and speed up convergence (Lee & Teng, 2000).
The design of a fuzzy partition and rules engine normally affects system performance. To simplify the design procedure, we usually use the symmetric and fixed membership functions (MFs), such as Gaussian, triangular. However, a large rule number should be used to achieve the specified approximation accuracy (or result larger approximated error) (Lee & Teng, 2001; Lotfi & Tsoi, 1996). Several approaches have been introduced to optimize fuzzy MFs and choose an efficient scheme for structure and parameter learning. Nevertheless, asymmetric fuzzy MFs (AFMFs) has been discussed and analyzed for this problem (Baldwin

& Karake, 2003; Kim et al, 2003; Lee & Teng, 2001; Li et al, 2005; Lin & Ho, 2005; Ozen & Garibaldi, 2004; Pan et al, 2007). The results showed that using AFMFs can improve the approximation capability. According to the results above, our purpose is to introduce a recurrent interval type-2 fuzzy neural network with asymmetric membership functions (RiT2FNN-A). The asymmetric Gaussian function is a new type of membership function due to excellent approximation results. It also provides a fuzzy-neural network with higher flexibility to easily approach the optimum result more accurately. Literature (Lee & Pan, 2007; Pan et al, 2007) proposed that a T2FNN with AFMFs (T2FNN-A) can improve the system performance and obtain better approach ability. However, the structure of network was a static model. In this article, we proposed a combining interval type-2 fuzzy asymmetric membership functions with recurrent neural network system, called RiT2FNNA. The proposed RiT2FNN-A is a modified version of the T2FNN (Lee & Lin, 2005; Lee et al, 2003; Lee & Pan, 2007; Pan et al, 2007; Wang et al, 2004), which provides memory elements to capture system dynamic information (Lee & Teng, 2000). The RiT2FNN-A system capability for temporarily storing information allowed us to extend the application domain to include temporal problem. Simulations are shown to illustrate the effectiveness of the RiT2FNN-A system.
This article is organized as follows. Section 2 introduces the interval type-2 fuzzy neural systems and construction of interval type-2 AFMFs. The proposed RiT2FNN-A system is described in Section 3. Simulation results about handling nonlinear system identification is done and introduced in Section 4. Finally, conclusion is given.

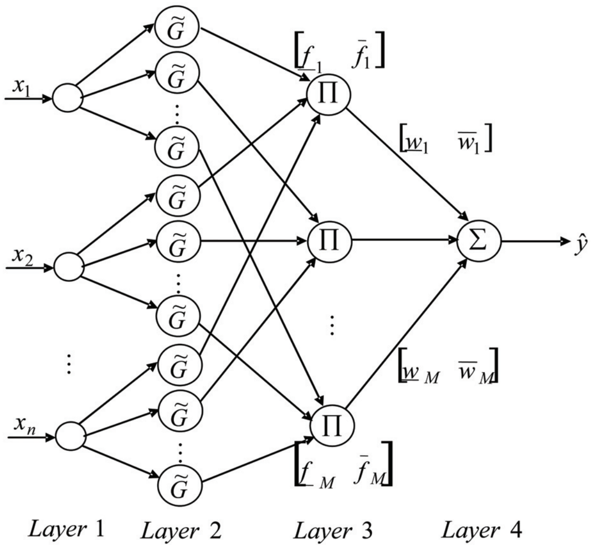

x_1
x_2
x_n
$\tilde{G}$
$[\underline{f}_1 \ \bar{f}_1]$
$[\underline{w}_1 \ \bar{w}_1]$
Π
Σ
$\hat{y}$
$[\underline{w}_M \ \bar{w}_M]$
$[\underline{f}_M \ \bar{f}_M]$
Layer 1
Layer 2
Layer 3
Layer 4

Figure1, Diagram of MISO T2FNN system with *M* fuzzy rules (Lee & Lin, 2005).

2. Interval type-2 fuzzy neural systems

The concept of type-2 fuzzy set was initially proposed as an extension of ordinary one (called type-1) by Zadeh (Zadeh 1975). In recent years, Mendel and Karnik have developed a complete theory of T2FLSs (John & Coupland, 2007; Karnik et al, 1999; Lee & Lin, 2005; Mendel, 2001). These systems are as an extension of general FLSs (called type-1) which is characterized by IF-THEN rules (Lin & Lee, 1996). The computation of iT2FLSs is more complex than the T1FLSs because of the antecedent and consequent type-2 fuzzy sets (John & Coupland, 2007; Karnik et al, 1999; Mendel, 2001). In our previous results, we successfully constructed the T2FNN to identify the nonlinear system (Lee & Lin, 2005; Lee et al, 2003; Lee & Pan, 2007). They perform as well as the general T1FNNs, even better. In this section, we first introduce the interval type-2 fuzzy neural network (iT2FNN) systems, a type of fuzzy inference system in neural network structures, followed by the construction of interval type-2 AFMFs (iT2AFMFs) which is used to develop the recurrent interval type-2 fuzzy neural network (RiT2FNN).

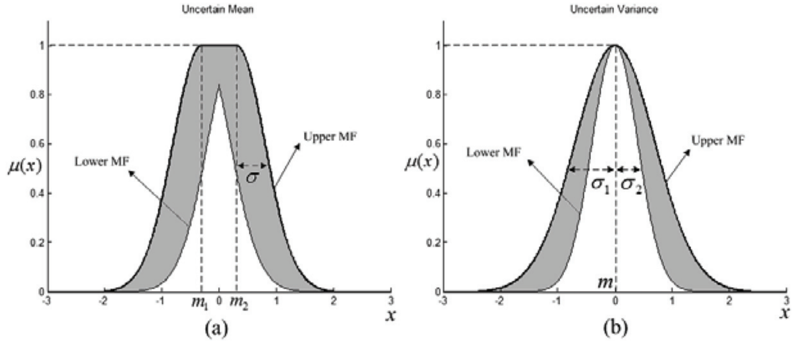


Figure 2. Symmetric interval type-2 fuzzy MFs: (a) Gaussian MF with uncertain mean and (b) Gaussian MF with uncertain variance.

2.1 Interval type-2 fuzzy neural network systems

In general, given an system input data set x_i , i=1, 2, …, n, and the desired output y_p , p=1, 2, …, m, the jth type-2 fuzzy rule has the form

$$\begin{aligned} &\text{Rule j: IF } x_1 \text{ is } \tilde{G}_1^j \text{ and } \dots x_n \text{ is } \tilde{G}_n^j \\ &\qquad \text{THEN } y_1 \text{ is } \tilde{w}_1^j \text{ and } \dots y_m \text{ is } \tilde{w}_m^j , \end{aligned} \tag{1}$$

where j is the number of rules, $\tilde{G}_i^j$ represents the linguistic term of the antecedent part, $\tilde{w}_p^j$ represents the real number of the consequent part; n and m are the numbers of the input and output dimensions, respectively. Based on the iT2FLSs, the construction of multi-inputsingle- output (MISO) type of the iT2FNN system is shown in Fig. 1 (Lee & Lin, 2005). Obviously, it is a static model and the structure uses interval type-2 fuzzy sets ($\tilde{G}$ and $\tilde{w}$). Figure 2 shows a commonly used two-dimensional interval type-2 Gaussian MF. Figure 2(a) is an interval type-2 Gaussian MF with an interval mean in [m_1,m_2] and fixed variance σ, and Fig. 2(b) is an interval type-2 Gaussian MF with an interval variance in [σ_1, σ_2] and fixed mean m.

It can be found that the iT2FNN uses the interval type-2 fuzzy sets and it implements the FLS in a four layer neural network structure. Layer-1 nodes are input nodes representing input linguistic variables, and layer-4 nodes are output nodes. The nodes in layer 2 are term nodes that act as T2MFs. All of the layer-3 nodes together formulate a fuzzy rule basis, and the links between layers 3 and 4 function as a connectionist inference engine. Herein, we introduce the iT2FNN system.

Layer 1: Input Layer

For the *i*th node of layer 1, the net input and the net output are represented as:

$$O_i^{(1)} = x_i^{(1)} \tag{2}$$

where $x_i^{(1)}$ represents the *i*th input to the *i*th node of layer 1. The subscript $_i$ denotes the *i*th input and the super-script $^{(1)}$ denotes the first layer.

Layer 2: Membership Layer

In this layer, each node performs a type-2 membership function (T2MF). Two kinds of T2MF are introduced (Liang & Mendel, 2000; Mendel, 2001). For case 1- Gaussian MFs with uncertain mean, shown in Fig. 2(a), we have

$$O_{ij}^{(2)} = \exp\left[-\frac{1}{2}\frac{\left(O_i^{(1)} - m_{ij}\right)^2}{\left(\sigma_{ij}\right)^2}\right] = \begin{cases} \overline{O}_{ij}^{(2)} & as\ m_{ij} = \overline{m}_{ij} \\ \underline{O}_{ij}^{(2)} & as\ m_{ij} = \underline{m}_{ij}. \end{cases} \tag{3}$$

Case 2- Gaussian MFs with uncertain variance, shown in Fig. 2(b), we have

$$O_{ij}^{(2)} = \exp\left[-\frac{1}{2}\frac{\left(O_i^{(1)} - m_{ij}\right)^2}{\left(\sigma_{ij}\right)^2}\right] = \begin{cases} \overline{O}_{ij}^{(2)} & as\ \sigma_{ij} = \overline{\sigma}_{ij} \\ \underline{O}_{ij}^{(2)} & as\ \sigma_{ij} = \underline{\sigma}_{ij}, \end{cases} \tag{4}$$

where m_{ij} and σ_{ij} represent the center (or mean) and the width (or variance), respectively The subscript $_{ij}$ indicates the *j*th term of the *i*th input $O_i^{(1)}$, where *j*=1, …, *M*, and the superscript $_{(2)}$ means the secondary layer. Therefore, the output $O_{ij}^{(2)}$ is represented as $[\underline{O}_{ij}^{(2)}\ \overline{O}_{ij}^{(2)}]$.

Layer 3: Rule Layer

In this layer, the operation is chosen as simple PRODUCT operation, i.e.,

$$O_j^{(3)} = \prod_{i=1}^{n}\left(O_{ij}^{(2)}\right) = \begin{cases} \overline{O}_j^{(3)} = \prod_{i=1}^{n}\left(\overline{w}_{ij}^{(3)}\overline{O}_{ij}^{(2)}\right) \\ \underline{O}_j^{(3)} = \prod_{i=1}^{n}\left(\underline{w}_{ij}^{(3)}\underline{O}_{ij}^{(2)}\right), \end{cases} \tag{5}$$

where the weights $w_{ij}^{(3)}$ are assumed to be unity, and the subscript $_j$ indicates the *j*th rule, *j* = 1,…,*M*, and the super-script $^{(3)}$ means the third layer. Thus, the output $O_{ij}^{(3)}$ is represented as $[\underline{O}_j^{(3)}, \overline{O}_j^{(3)}]$.

Layer 4: Output Layer

Links in this layer are used to implement the consequence matching, type-reduction and defuzzification (Lee & Lin, 2005; Mendel, 2001). Thus,

$$\hat{y} = O^{(4)} = \frac{O_R^{(4)} + O_L^{(4)}}{2}, \tag{6}$$

where

$$O_R^{(4)} = \sum_{j=1}^{M}\left(f_j^R \overline{w}_j^{(4)}\right) = \sum_{j=1}^{R}\left(\underline{O}_j^{(3)}\overline{w}_j^{(4)}\right) + \sum_{j=R+1}^{M}\left(\overline{O}_j^{(3)}\overline{w}_j^{(4)}\right), \tag{7}$$

$$O_L^{(4)} = \sum_{j=1}^{M}\left(f_j^L \underline{w}_j^{(4)}\right) = \sum_{j=1}^{L}\left(\overline{O}_j^{(3)}\underline{w}_j^{(4)}\right) + \sum_{j=L+1}^{M}\left(\underline{O}_j^{(3)}\underline{w}_j^{(4)}\right), \tag{8}$$

and

$$R = \underset{j\in[1,\cdots,M-1]}{\arg}\ \max\left(\overline{O}_R^{(4)}\right), \qquad L = \underset{j\in[1,\cdots,M-1]}{\arg}\ \min\left(\underline{O}_L^{(4)}\right).$$

In order to get $O_L^{(4)}$ and $O_R^{(4)}$, we first need to find coefficients R and L. Without loss of generality, we assume that the pre-computed $\overline{w}_j^{(4)}$ and $\underline{w}_j^{(4)}$ are arranged in ascending order, i.e., $\overline{w}_1^{(4)} \le \overline{w}_2^{(4)} \le \cdots \le \overline{w}_M^{(4)}$ and $\underline{w}_1^{(4)} \le \underline{w}_2^{(4)} \le \cdots \le \underline{w}_M^{(4)}$ (Mendel, 2001). Then,

R1: Compute $O_R^{(4)}$ in (7) by initial setting $f_j^R = \frac{1}{2}\left(\overline{O}_j^{(3)} + \underline{O}_j^{(3)}\right)$ for $j = 1,\ldots,M$, and let $y_r \equiv O_R^{(4)}$.

R2: Find $R\left(1 \le R \le M-1\right)$ such that $\overline{w}_R^{(4)} \le y_r \le \overline{w}_{R+1}^{(4)}$.

R3: Compute $O_R^{(4)}$ in (7) with $f_j^R = \underline{O}_j^{(3)}$ for $j \le R$ and $f_j^R = \overline{O}_j^{(3)}$ for $j > R$, and let $y_r' = O_R^{(4)}$.

R4: If $y_r' \ne y_r$, then go to step R5. If $y_r' = y_r$, then stop and set $O_R^{(4)} = y_r'$.

R5: Set y_r' to be y_r and return to step R2.

Subsequently, the computation of $O_L^{(4)}$ is similar to the above procedure.

L1: Compute $O_L^{(4)}$ in (8) by initial setting $f_j^L = \frac{1}{2}\left(\overline{O}_j^{(3)} + \underline{O}_j^{(3)}\right)$ for $i = 1,\cdots,M$, and let $y_r \equiv O_L^{(4)}$.

L2: Find $L\left(1 \le L \le M-1\right)$ such that $\underline{w}_L^{(4)} \le y_l \le \underline{w}_{L+1}^{(4)}$.

L3: Compute $O_L^{(4)}$ in (2.7) with $f_j^L = \overline{O}_j^{(3)}$ for $j \le L$ and $f_j^L = \underline{O}_j^{(3)}$ for $j > L$, and let $y_l' = O_L^{(4)}$.

L4: If $y_l' \ne y_l$, then go to step L5. If $y_l' = y_l$, then stop and set $O_L^{(4)} = y_l'$.

L5: Set y_l' to be y_l and return to step L2.

This five-step iterative procedure is called the Karnik-Mendel procedure (Liang & Mendel, 2000; Mendel, 2001).Thus, the input/output representation of iT2FNN system with uncertain mean is

$$\hat{y}\left(\overline{m}_{ij}, \underline{m}_{ij}\sigma_{ij}, \overline{w}_j, \underline{w}_j\right) = \frac{1}{2}\left[\sum_{j=1}^{R}\left(\underline{O}_j^{(3)}\overline{w}_j^{(4)}\right) + \sum_{k=R+1}^{M}\left(\overline{O}_k^{(3)}\overline{w}_k^{(4)}\right) + \sum_{j=1}^{L}\left(\overline{O}_j^{(3)}\underline{w}_j^{(4)}\right) + \sum_{k=L+1}^{M}\left(\underline{O}_k^{(3)}\underline{w}_k^{(4)}\right)\right]. \tag{9}$$

Similarly, the iT2FNN using T2MFs with uncertain variance can be simplified as (Lee & Lin, 2005)

$$\hat{y}\left(m_{ij},\overline{\sigma}_{ij},\underline{\sigma}_{ij},w_j\right)=\frac{1}{2}\sum_{j=1}^{M}\left[\left(\underline{O}_j^{(3)}+\overline{O}_j^{(3)}\right)w_j^{(4)}\right] \tag{10}$$

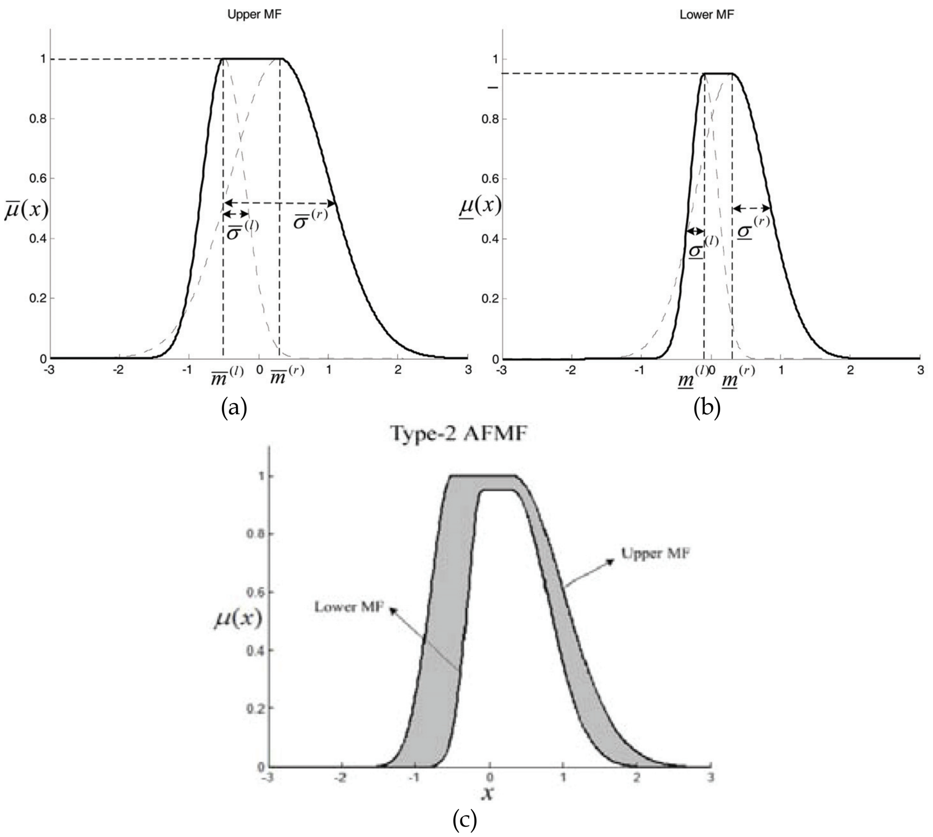


Figure 3. Construction of a type-2 AFMF: (a) upper MF (solid line), (b) lower MF (solid line), and (c) constructed iT2AFMF.

2.2 Construction of interval type-2 asymmetric fuzzy membership functions

The interval T2MFs of the precondition part discussed in this article are of asymmetric type, iT2AFMFs, as described below (see Fig. 3). Each MF is replaced by an asymmetric one constructed from parts of four Gaussian functions; that is, each upper and lower MF is constructed by two Gaussian MFs and one segment. Here we use the superscripts (l) and (r) to denote the left and right curves of a Gaussian MF. The parameters of lower and upper MFs are denoted by an underline (_) and bar (¯), respectively. Thus, the upper MF is constructed as

$$\overline{\mu}_{\tilde{G}}(x)=\begin{cases}\exp\left[-\frac{1}{2}\left(\frac{x-\overline{m}^{(l)}}{\overline{\sigma}^{(l)}}\right)^2\right] & \text{, for } x\le\overline{m}^{(l)} \\ 1 & \text{, for } \overline{m}^{(l)}\le x\le\overline{m}^{(r)} \\ \exp\left[-\frac{1}{2}\left(\frac{x-\overline{m}^{(r)}}{\overline{\sigma}^{(r)}}\right)^2\right] & \text{, for } \overline{m}^{(r)}\le x\end{cases} \tag{11}$$

where $\overline{m}^{(l)}$ and $\overline{m}^{(r)}$ denote the means of two Gaussian MFs satisfying $\overline{m}^{(l)}\le\ \overline{m}^{(r)}$, and $\overline{\sigma}^{(l)}$ and $\overline{\sigma}^{(r)}$ denotes the deviation (i.e., width) of two Gaussian MFs. Figure 3(a) shows the upper iT2AFMF constructed using $\overline{m}^{(l)}$, $\overline{m}^{(r)}$, $\overline{\sigma}^{(l)}$, and $\overline{\sigma}^{(r)}$. Similarly, the lower asymmetric MF is defined as

$$\underline{\mu}_{\tilde{G}}(x)=\begin{cases}\underline{r}\cdot\exp\left[-\frac{1}{2}\left(\frac{x-\underline{m}^{(l)}}{\underline{\sigma}^{(l)}}\right)^2\right] & \text{, for } x\le\underline{m}^{(l)} \\ \underline{r} & \text{, for } \underline{m}^{(l)}\le x\le\underline{m}^{(r)} \\ \underline{r}\cdot\exp\left[-\frac{1}{2}\left(\frac{x-\underline{m}^{(r)}}{\underline{\sigma}^{(r)}}\right)^2\right] & \text{, for } \underline{m}^{(r)}\le x\end{cases} \tag{12}$$

where $\underline{m}^{(l)}\le\ \underline{m}^{(r)}$ and $0.5\le r\le 1$. The corresponding widths of the MFs are $\underline{\sigma}^{(l)}$ and $\underline{\sigma}^{(r)}$. To avoid unreasonable MFs, the following constrains are added:

$$\begin{cases}\overline{m}^{(l)}\le\underline{m}^{(l)}\le\underline{m}^{(r)}\le\overline{m}^{(r)} \\ \underline{\sigma}^{(l)}\le\overline{\sigma}^{(l)}\ ,\ \underline{\sigma}^{(r)}\le\overline{\sigma}^{(r)}. \\ 0.5\le\underline{r}\le 1\end{cases} \tag{13}$$

Figure 3(b) sketches the lower type-2 AFMF. The corresponding constructed iT2AFMF is shown in Fig. 3(c). This introduces the properties of uncertain mean and variance (Karnik et al, 1999). Additionally, we can construct other iT2AFMFs by tuning the parameters. The corresponding tuning algorithm is derived to improve system accuracy and approximation ability.

3. RiT2FNN-A system and learning

3.1 Network structure of RiT2FNN-A system

In this section, the structure of RiT2FNN-A system is introduced. The MISO case I considered here for convenience. The proposed RiT2FNN-A is modified and extended from previous results of literature (Juang, 2002; Karnik et al, 1999; Lee & Lin, 2005; Lee & Pan, 2007; Lin & Ho, 2005). It uses the interval asymmetric type-2 fuzzy sets and it implements the FLS in a five-layer neural network structure which contains four-layer forward network and a feedback layer. Layer-1 nodes are input nodes representing input linguistic variables, and layer-4 nodes are output nodes representing output linguistic variables. The nodes in layer 2 are term nodes that act as MFs, where each membership node is responsible for mapping an input linguistic variable into a corresponding linguistic value for that variable. All of the layer-3 nodes together formulate a fuzzy rule basis, and the links between layers 3 and 4 function as a connectionist inference engine. The rule nodes reside in layer 3, and layer 5 is the recurrent part in type-2 fuzzy sets.

In general, given system input data x_i, $i = 1, 2,\dots, n$, the internal variables g_j, $j = 1, 2,\dots, M$, and the desired output y_p, $p = 1, 2,\dots, m$, the jth type-2 fuzzy rule for RiT2FNN-A has the form:

$$\begin{aligned}&\text{Rule } j\text{: IF } x_1 \text{ is } \tilde{G}_{1j} \text{ and } \dots x_n \text{ is } \tilde{G}_{nj} \text{ and } g_j \text{ is } \tilde{G}_j^F\\ &\quad\text{THEN } y_1 \text{ is } \tilde{w}_1^j \text{ and } \dots y_m \text{ is } \tilde{w}_m^j, g_1 \text{ is } \tilde{a}_1^j, g_2 \text{ is } \tilde{a}_2^j, \dots, \text{and } g_M \text{ is } \tilde{a}_M^j.\end{aligned} \tag{14}$$

where $\tilde{G}$ represents the linguistic term of the antecedent part, $\tilde{w}$ and $\tilde{a}$ represents the interval real number of the consequent part; and M is the total rule number. Here the fuzzy MFs of the antecedent part $\tilde{G}$ are of iT2AFMFs, which represent the different from typical Gaussian MFs. The diagram of RiT2FNN-A is shown in Fig. 4. Below we indicate the signal propagation and the operation functions of the nodes in each layer. In the following description, $O_i^{(l)}$ denotes the ith output of a node in the lth layer.

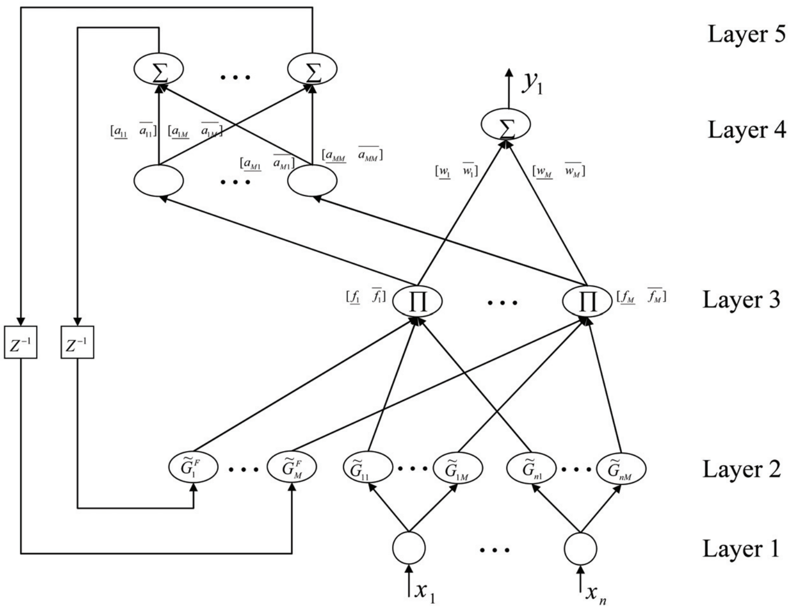

Layer 5
Layer 4
Layer 3
Layer 2
Layer 1
y_1
$[\underline{a}_{11}\ \overline{a}_{11}]$
$[\underline{a}_{1M}\ \overline{a}_{1M}]$
$[\underline{a}_{M1}\ \overline{a}_{M1}]$
$[\underline{a}_{MM}\ \overline{a}_{MM}]$
$[\underline{w}_1\ \overline{w}_1]$
$[\underline{w}_M\ \overline{w}_M]$
$[\underline{f}_1\ \overline{f}_1]$
$[\underline{f}_M\ \overline{f}_M]$
Z^{-1}
Z^{-1}
$\tilde{G}_1^F$
$\tilde{G}_M^F$
$\tilde{G}_{11}$
$\tilde{G}_{1M}$
$\tilde{G}_{n1}$
$\tilde{G}_{nM}$
x_1
x_n

Fig. 4. Diagram of the proposed RiT2FNN-A system.

Layer 1: Input Layer

For the ith node of layer 1, the net input and output are represented as

$$O_j^{(1)} = x_i^{(1)}, \tag{15}$$

where $x_i^{(1)}$ represents the *i*th input to the *j*th node. Obviously, the nodes in this layer only transmit input values to the next layer directly.

Layer 2: Membership Layer

In layer 2, each node performs an iT2AFMF introduced by (11)–(13) (shown in Fig. 3). The following simplified notation is adopted

$$O_{ij}^{(2)} = \tilde{\mu}_{\tilde{G}_i^j}(O_i^{(1)}) \tag{16}$$

It is clear that there are two parts in this layer, regular nodes and feedback nodes. Their input are $O_j^{(1)}$ and $g_j(k)$. Therefore, for network input x_i, the output is

$$O_{ij}^{(2)} = \left[\underline{O}_{ij}^{(2)} \quad \overline{O}_{ij}^{(2)}\right]^T = \left[\underline{\mu}_{\tilde{G}_{ij}}(O_i^{(1)}) \quad \overline{\mu}_{\tilde{G}_{ij}}(O_i^{(1)})\right]^T \tag{17}$$

For internal or feedback variable g_j,

$$O_j^{F(2)} = \left[\underline{O}_j^{F(2)} \quad \overline{O}_j^{F(2)}\right]^T = \left[\underline{\mu}_{\tilde{G}_j^F}(g_j(k)) \quad \overline{\mu}_{\tilde{G}_j^F}(g_j(k))\right]^T, \tag{18}$$

where the subscript *ij* indicates the *j*th term of the *i*th input $O_i^{(1)}$. The superscript *F* indicates the feedback layer.

Layer 3: Rule Layer

The links in layer 3 are used to implement the antecedent matching, and these are equal to the work in the rule layer. Using the product *t*-norm, the firing strength associated with the *j*th rule is

$$\underline{f}^j = \underline{\mu}_{\tilde{G}_1^j}(x_1) * \cdots * \underline{\mu}_{\tilde{G}_n^j}(x_n) * \underline{\mu}_{\tilde{G}_j^{jF}}(\cdot) \tag{19}$$

$$\overline{f}^j = \overline{\mu}_{\tilde{G}_1^j}(x_1) * \cdots * \overline{\mu}_{\tilde{G}_n^j}(x_n) * \overline{\mu}_{\tilde{G}_j^{jF}}(\cdot) \tag{20}$$

where $\underline{\mu}(\cdot)$ and $\overline{\mu}(\cdot)$ are the lower and upper membership grades of $\tilde{G}(\cdot)$, respectively. Therefore, a simple product operation is used. Then, for the *j*th input rule node:

$$O_j^{(3)} = \begin{cases} \overline{O}_j^{F(2)} \prod_i w_{ij}^{(3)} \cdot \overline{O}_{ij}^{(2)} \\ \underline{O}_j^{F(2)} \prod_i w_{ij}^{(3)} \cdot \underline{O}_{ij}^{(2)} \end{cases} \tag{21}$$

where weights $w_{ij}^{(3)}$ are assumed to be unity and

$$O_j^{(3)} = \left[\underline{O}_j^{(3)} \quad \overline{O}_j^{(3)}\right]^T = \left[\underline{O}_j^{F(2)} \prod_{i=1}^{n} \underline{O}_{ij}^{(2)} \quad \overline{O}_j^{F(2)} \prod_{i=1}^{n} \overline{O}_{ij}^{(2)}\right]^T. \tag{22}$$

Layer 4: Output Layer

Without loss of generality, the consequent part of the iT2FLS is $\tilde{w}_j = \left[\underline{w}_j \quad \overline{w}_j\right]^T$, $\underline{w}_j \le \overline{w}_j$. The vector notations $\underline{w} = [\underline{w}_1 \quad \cdots \quad \underline{w}_M]^T$ and $\overline{w} = [\overline{w}_1 \quad \cdots \quad \overline{w}_M]^T$ are used for clarity. The remaining works are type reduction and defuzzification. For type reduction, we should calculate the lower and upper bounds [y_l, y_r] (Karnik et al, 1999; Mendel, 2001). Modifying from the Karnik-Mendel procedure (Karnik et al, 1999; Mendel, 2001), let

$$O_{TR}^{(4)} = \int_{w_1 \in [\underline{w}_1, \overline{w}_1]} \cdots \int_{w_M \in [\underline{w}_M, \overline{w}_M]} \int_{f_1 \in [\underline{f}_1, \bar{f}_1]} \cdots \int_{f_M \in [\underline{f}_M, \bar{f}_M]} 1 \Big/ \sum_{i=1}^{M} f_i w_i \, . \tag{23}$$

Note that the normalization ($\sum_{i=1}^{M} f_i$) is removed here to simplify the type reduction procedure, computation, and the derivation of the learning algorithm by the gradient method. We denote the maximum and minimum of $\sum_{i=1}^{M} f_i w_i$ as $\overline{O}^{(4)}$ and $\underline{O}^{(4)}$,

$$\underline{O}^{(4)} = \underline{w}^T f_L = \sum_{j=1}^{L} \left(\overline{O}_j^{(3)} \underline{w}_j \right) + \sum_{j=L+1}^{M} \left(\underline{O}_j^{(3)} \underline{w}_j \right), \tag{24}$$

$$\overline{O}^{(4)} = \overline{w}^T f_R = \sum_{j=1}^{R} \left(\underline{O}_j^{(3)} \overline{w}_j \right) + \sum_{j=R+1}^{M} \left(\overline{O}_j^{(3)} \overline{w}_j \right), \tag{25}$$

where

$$f_L = \left[\bar{f}_1, \cdots, \bar{f}_L, \underline{f}_{L+1}, \cdots, \underline{f}_M \right]^T = \left[\overline{O}_1^{(3)}, \cdots, \overline{O}_L^{(3)}, \underline{O}_{L+1}^{(3)}, \cdots, \underline{O}_M^{(3)} \right]^T, \tag{26}$$

$$f_R = \left[\underline{f}_1, \cdots, \underline{f}_R, \bar{f}_{R+1}, \cdots, \bar{f}_M \right]^T = \left[\underline{O}_1^{(3)}, \cdots, \underline{O}_R^{(3)}, \overline{O}_{R+1}^{(3)}, \cdots, \overline{O}_M^{(3)} \right]^T . \tag{27}$$

It is obvious that *R* and *L* should be calculated first. The weights are arranged in order as $\underline{w}_1 \le \underline{w}_2 \le \cdots \underline{w}_M$and $\overline{w}_1 \le \overline{w}_2 \le \cdots \overline{w}_M$. According to the Karnik-Mendel procedure (Karnik et al, 1999; Liang & Mendel, 2000; Mendel, 2001), *L* and *R* are

$$L = \underset{j \in [1, \cdots, M-1]}{\arg} \min\left(\underline{O}^{(4)} \right), \qquad R = \underset{j \in [1, \cdots, M-1]}{\arg} \max\left(\overline{O}^{(4)} \right). \tag{28}$$

According to the above introduction, only the minimum of $\underline{O}^{(4)}$ and the maximum of $\overline{O}^{(4)}$ should be calculated; which therefore simplifies the type-reduction computation. Finally, the crisp output is

$$O^{(4)} = \frac{\underline{O}^{(4)} + \overline{O}^{(4)}}{2} . \tag{29}$$

Layer 5: Feedback Layer

This layer contains the context nodes, which is used to produce the internal variable $O_j^{(5)}$. Each rule is associated with a particular internal variable. Hence, the number of the context nodes is equal to the number of rules. The same operations (type-reduction and defuzzifcation) as layer 4 are performed here.

$$g_j(k+1) = O_j^{(5)}(k+1) = \frac{1}{2} \left[\underline{O}_j^{(5)}(k+1) + \overline{O}_j^{(5)}(k+1) \right] \tag{30}$$

$$\underline{O}_j^{(5)}(k+1) = \underline{a}_j^T f_L = \sum_{h=1}^{L_j^F} \left(\overline{O}_h^{(3)} \underline{a}_{jh} \right) + \sum_{h=L_j^F+1}^{M} \left(\underline{O}_h^{(3)} \underline{a}_{jh} \right) \tag{31}$$

$$\overline{O}_j^{(5)}(k+1) = \overline{a}_j^T f_R = \sum_{h=1}^{R_j^F} \left(\underline{O}_h^{(3)} \overline{a}_{jh} \right) + \sum_{h=R_j^F+1}^{M} \left(\overline{O}_h^{(3)} \overline{a}_{jh} \right) \tag{32}$$

$$L_j^F = \underset{j\in[1,\cdots,M-1]}{\arg} \min\left(\underline{O}_j^{(5)}(k+1) \right) \tag{33}$$

$$R_j^F = \underset{j\in[1,\cdots,M-1]}{\arg} \max\left(\overline{O}_j^{(5)}(k+1) \right) . \tag{34}$$

Note that the delayed value of g_j is fed into layer 2, and it acts as an input variable to the precondition part of a rule. Each fuzzy rule has the corresponding internal variable g_j which is used to decide the influence degree of temporal history to the current rule.

3.2 Learning algorithm for RiT2FNN-A

The gradient descent method is adopted to derive learning algorithm of the RiT2FNN-A system. For clarification, we consider the single-output system and define the error cost function as

$$E(k) = \frac{1}{2}[y_d(k) - \hat{y}(k)]^2 \tag{35}$$

where y_d is the desired output and $\hat{y}$ is the RiT2FNN-A's output. Using the gradient descent algorithm, the parameters updated law is

$$\mathbf{W}(k+1) = \mathbf{W}(k) + \Delta\mathbf{W}(k) = \mathbf{W}(k) + \eta\left(-\frac{\partial E(k)}{\partial \mathbf{W}(k)} \right) \tag{36}$$

in which η is the learning rate ($0 < \eta \le 1$). $\mathbf{W} = [\mathbf{W}_w \quad \underline{\mathbf{W}} \quad \underline{\mathbf{W}}^F \quad \overline{\mathbf{W}} \quad \overline{\mathbf{W}}^F \quad \mathbf{W}_a \quad \underline{r} \quad \underline{r}^F]$are the adjustable parameters, where $\mathbf{W}_w$ is consequent weights, $\underline{\mathbf{W}}$ and $\underline{\mathbf{W}}^F$ are parameters of lower MFs, $\overline{\mathbf{W}}$ and $\overline{\mathbf{W}}^F$ are upper MFs parameters, $\mathbf{W}_a$ is parameter in feedback layer, and $\underline{r}$ and $\underline{r}^F$ are the column vectors, i.e.,

$$\mathbf{W}_w = \left[\underline{w} \quad \overline{w} \right]^T \tag{37}$$

$$\mathbf{W}_a = \left[\underline{a} \quad \overline{a} \right]^T \tag{38}$$

$$\underline{\mathbf{W}} = \left[\underline{m}^{(l)} \quad \underline{m}^{(r)} \quad \underline{\sigma}^{(l)} \quad \underline{\sigma}^{(r)} \right]^T \tag{39}$$

$$\overline{\mathbf{W}} = \left[\overline{m}^{(l)} \quad \overline{m}^{(r)} \quad \overline{\sigma}^{(l)} \quad \overline{\sigma}^{(r)} \right]^T \tag{40}$$

$$\underline{\mathbf{W}}^F = \left[\underline{m}^{F(l)} \quad \underline{m}^{F(r)} \quad \underline{\sigma}^{F(l)} \quad \underline{\sigma}^{F(r)}\right]^T \tag{41}$$

$$\overline{\mathbf{W}}^F = \left[\overline{m}^{F(l)} \quad \overline{m}^{F(r)} \quad \overline{\sigma}^{F(l)} \quad \overline{\sigma}^{F(r)}\right]^T . \tag{42}$$

Considering the term of $\partial E(k)/\partial \mathbf{W}(k)$, we have

$$\frac{\partial E(k)}{\partial \mathbf{W}(k)} = \frac{\partial E(k)}{\partial \hat{y}(k)} \frac{\partial \hat{y}(k)}{\partial \mathbf{W}(k)} = -\left[y_d(k) - \hat{y}(k)\right] \frac{\partial \hat{y}(k)}{\partial \mathbf{W}(k)} \tag{43}$$

Thus, (36) can be rewritten as

$$\mathbf{W}(k+1) = \mathbf{W}(k) + \eta e(k) \frac{\partial \hat{y}(k)}{\partial \mathbf{W}(k)} \tag{44}$$

where $e(k) = y_d(k) - \hat{y}(k)$. The remaining work involves finding the corresponding partial derivatives with respect to each parameter.

Observing equation (24) and if $j \le L$, only the term of $\sum_{j=1}^{L}\left(\overline{O}_j^{(3)} \underline{w}_j\right)$ should be considered, and only consider $\sum_{j=L+1}^{M}\left(\underline{O}_j^{(3)} \underline{w}_j\right)$ if $j>L$. Moreover, we consider $\sum_{j=1}^{R}\left(\underline{O}_j^{(3)} \overline{w}_j\right)$ if $j \le R$ in (25), as well as $\sum_{j=R+1}^{M}\left(\overline{O}_j^{(3)} \overline{w}_j\right)$ where $j>R$. Thus, we should notice the values of j, R, and L in deriving the update laws.

In order to avoid the unnecessary tuning, we must also consider the firing regions of MFs for input variable xi. For example, considering an upper MF as shown in Fig. 5, region (I)- $x_i \le \overline{m}_{ij}^{(l)}$, only $\overline{m}_{ij}^{(l)}$ and $\overline{\sigma}_{ij}^{(l)}$ are updated; region (II)- $\overline{m}_{ij}^{(r)} \le x_i$, only $\overline{m}_{ij}^{(r)}$ and $\overline{\sigma}_{ij}^{(r)}$ must be updated as well. Finally, region (III)- $\overline{m}_{ij}^{(l)} < x_i < \overline{m}_{ij}^{(r)}$, nothing should be done. Therefore, we can tune one side of MF for each training pattern. The results of lower MFs are the same as above discussion. Besides, parameter r must be updated for all three regions. Owing the recurrent property, the real time recurrent learning algorithm (RTRL) is used.

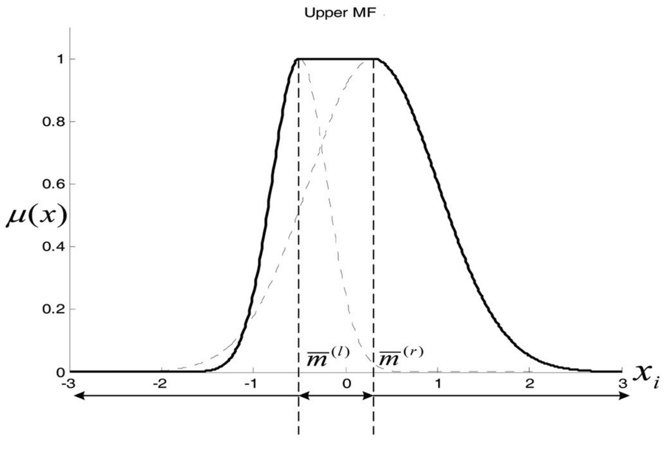


Fig. 5. Definitions of firing regions of input variable xi (upper MF).

By the gradient method, we derive the parameters update laws. Consider equations (24), (25), and (29), the output of RiT2FNN-A is rewritten as

$$O^{(4)} = \frac{1}{2}\left[\underline{O}_1^{(4)} + \overline{O}_1^{(4)}\right] = \frac{1}{2}\left[\sum_{j=1}^{L}\left(\overline{O}_j^{(3)}\underline{w}_j\right) + \sum_{j=L+1}^{M}\left(\underline{O}_j^{(3)}\underline{w}_j\right) + \sum_{j=1}^{R}\left(\underline{O}_j^{(3)}\overline{w}_j\right) + \sum_{j=R+1}^{M}\left(\overline{O}_j^{(3)}\overline{w}_j\right)\right]. \tag{45}$$

From equations (36) and (44), our major work is to find the partial derivation of RiT2FNN-A with respect to each parameter which can be obtained using the chain rule. We will show the update rule of $\mathbf{W}_w$ and $\overline{\mathbf{W}}$ only. Other parameter's updated rule can be derived the same way and are omitted.

-Parameters $\mathbf{W}_w$

$$\begin{bmatrix}\underline{w}_j(k+1)\\ \overline{w}_j(k+1)\end{bmatrix} = \begin{bmatrix}\underline{w}_j(k)\\ \overline{w}_j(k)\end{bmatrix} + \eta_w e(k)\begin{bmatrix} f_{L,j}\\ f_{R,j}\end{bmatrix} \tag{46}$$

where $f_{L\,j}$, and $f_{R\,j}$, are introduced previously in (26) and (27), and η_w is the corresponding learning rate.

-Parameters $\overline{\mathbf{W}}$

Region (I): $x_i \leq \overline{m}_{ij}^{(l)}$

$$\begin{bmatrix}\overline{m}_{ij}^{(l)}(k+1)\\ \overline{\sigma}_{ij}^{(l)}(k+1)\end{bmatrix} = \begin{bmatrix}\overline{m}_{ij}^{(l)}(k)\\ \overline{\sigma}_{ij}^{(l)}(k)\end{bmatrix} + \frac{1}{2}\overline{\eta}e(k)\begin{cases}\underline{w}_j\begin{bmatrix}\overline{P}_{ij}^{(l)} + \overline{\omega}_j^{F(l)} \times \overline{H}_{jh}^{P(l)}\\ \overline{Q}_{ij}^{(l)} + \overline{\omega}_j^{F(l)} \times \overline{H}_{jh}^{Q(l)}\end{bmatrix}, \text{ if } j \leq L\\ \overline{w}_j\begin{bmatrix}\overline{P}_{ij}^{(l)} + \overline{\omega}_j^{F(l)} \times \overline{H}_{jh}^{P(l)}\\ \overline{Q}_{ij}^{(l)} + \overline{\omega}_j^{F(l)} \times \overline{H}_{jh}^{Q(l)}\end{bmatrix}, \text{ if } j > R\end{cases} \tag{47}$$

where $\overline{\eta}$ denotes the corresponding learning rate,

$$\overline{P}_{ij}^{(l)} = \overline{O}_j^{(3)}\frac{\left(x_i - \overline{m}_{ij}^{(l)}\right)}{\left(\overline{\sigma}_{ij}^{(l)}\right)^2},\ \overline{Q}_{ij}^{(l)} = \overline{O}_j^{(3)}\frac{\left(x_i - \overline{m}_{ij}^{(l)}\right)^2}{\left(\overline{\sigma}_{ij}^{(l)}\right)^3},$$

$$\overline{\omega}_j^{F(l)} = \overline{O}_j^{(3)}\frac{\left(g_j - \overline{m}_j^{F(l)}\right)}{\left(\overline{\sigma}_j^{F(l)}\right)^2},$$

$$\overline{H}_{jh}^{P(l)} = \sum_{h=1}^{M}\overline{C}_{jh}^{F}\left(\overline{P}_{ij}^{(l)}(k-1) + \omega_h^{F(l)}(k-1)\cdot\overline{H}_{jh}^{P(l)}(k-1)\right),$$

$$\overline{H}_{jh}^{Q(l)} = \sum_{h=1}^{M}\overline{C}_{jh}^{F}\left(\overline{Q}_{ij}^{(l)}(k-1) + \overline{\omega}_j^{F(l)}(k-1)\cdot\overline{H}_{jh}^{Q(l)}(k-1)\right), \text{ and}$$

$$\overline{C}_{jh}^{F} = \begin{cases} 0.5\times\underline{a}_{jh}, & \text{for } (h \leq L_j^F) \text{ and } (h \leq R_j^F)\\ 0.5\times\overline{a}_{jh}, & \text{for } (h > L_j^F) \text{ and } (h > R_j^F)\\ 0, & \text{for } (h > L_j^F) \text{ and } (h \leq R_j^F)\\ 0.5\times(\underline{a}_{jh} + \overline{a}_{jh}), & \text{else.}\end{cases}$$

Region (II): $\overline{m}_{ij}^{(r)} \le x_i$

$$\begin{bmatrix} \overline{m}_{ij}^{(r)}(k+1) \\ \overline{\sigma}_{ij}^{(r)}(k+1) \end{bmatrix} = \begin{bmatrix} \overline{m}_{ij}^{(r)}(k) \\ \overline{\sigma}_{ij}^{(r)}(k) \end{bmatrix} + \frac{1}{2}\overline{\eta} e(k) \begin{cases} \underline{w}_j \begin{bmatrix} \overline{P}_{ij}^{(r)} + \overline{\omega}_j^{F(r)} \times \overline{H}_{jh}^{P(r)} \\ \overline{Q}_{ij}^{(r)} + \overline{\omega}_j^{F(r)} \times \overline{H}_{jh}^{Q(r)} \end{bmatrix}, \text{ if } j \le L \\ \overline{w}_j \begin{bmatrix} \overline{P}_{ij}^{(r)} + \overline{\omega}_j^{F(r)} \times \overline{H}_{jh}^{P(r)} \\ \overline{Q}_{ij}^{(r)} + \overline{\omega}_j^{F(r)} \times \overline{H}_{jh}^{Q(r)} \end{bmatrix}, \text{if } j > \mathrm{R} \end{cases} \tag{48}$$

where

$$\overline{P}_{ij}^{(r)} = \overline{O}_j^{(3)} \frac{\left(x_i - \overline{m}_{ij}^{(r)}\right)}{\left(\overline{\sigma}_{ij}^{(r)}\right)^2}, \overline{Q}_{ij}^{(r)} = \overline{O}_j^{(3)} \frac{\left(x_i - \overline{m}_{ij}^{(r)}\right)^2}{\left(\overline{\sigma}_{ij}^{(r)}\right)^3},$$

$$\overline{\omega}_j^{F(r)} = \overline{O}_j^{(3)} \frac{\left(g_j - \overline{m}_j^{F(r)}\right)}{\left(\overline{\sigma}_j^{F(r)}\right)^2},$$

$$\overline{H}_{jh}^{P(r)} = \sum_{h=1}^{M} \overline{C}_{jh}^{F} \left(\overline{P}_{ij}^{(r)}(k-1) + \omega_j^{F(r)}(k-1) \cdot H_{jh}^{P(r)}(k-1)\right), \text{ and}$$

$$\overline{H}_{jh}^{Q(r)} = \sum_{h=1}^{M} \overline{C}_{jh}^{F} \left(\overline{Q}_{ij}^{(r)}(k-1) + \overline{\omega}_j^{F(r)}(k-1) \cdot \overline{H}_{jh}^{Q(r)}(k-1)\right).$$

Region (III): $\overline{m}_{ij}^{(l)} < x_i < \overline{m}_{ij}^{(r)}$

$$\begin{bmatrix} \overline{m}_{ij}^{(l)}(k+1) \\ \overline{\sigma}_{ij}^{(l)}(k+1) \\ \overline{m}_{ij}^{(r)}(k+1) \\ \underline{\overline{\sigma}}_{ij}^{(r)}(k+1) \end{bmatrix} = \begin{bmatrix} \overline{m}_{ij}^{(l)}(k) \\ \overline{\sigma}_{ij}^{(l)}(k) \\ \overline{m}_{ij}^{(r)}(k) \\ \overline{\sigma}_{ij}^{(r)}(k) \end{bmatrix}. \tag{49}$$

Note that $\overline{H}_{jh}^{P(l)}, \overline{H}_{jh}^{P(r)}, \overline{H}_{jh}^{Q(l)}$, and $\overline{H}_{jh}^{Q(r)}$ are recurrent factors and equal to zero initially and are reset to zero after a period of time. $\overline{C}_{jh}^{F}$ is the recurrent weighting factor.

By using the Lyapunov stability approach, we have the following convergence theorem.

Theorem 1: Let $[\eta_w \quad \underline{\eta} \quad \overline{\eta} \quad \eta_{\underline{r}}]$ be the learning rates of the tuning parameters for RiT2FNNA The asymptotic convergence of RiT2FNN-A is guaranteed if proper learning rates $[\eta_w \quad \underline{\eta} \quad \overline{\eta} \quad \eta_{\underline{r}}]$ are chosen satisfying the following condition

$$\left(\lambda_w + \underline{\lambda} + \overline{\lambda} + \lambda_{\underline{r}} + \underline{\lambda}_a + \overline{\lambda}_a + \underline{\lambda}^F + \overline{\lambda}^F + \lambda_{\underline{r}^F}\right) < 2 \tag{50}$$

where

$$\lambda_w = \eta_w \left[\frac{\partial \hat{y}(k)}{\partial \mathbf{W}_w}\right]^T \left[\frac{\partial \hat{y}(k)}{\partial \mathbf{W}_w}\right] > 0, \underline{\lambda} = \underline{\eta} \left[\frac{\partial \hat{y}(k)}{\partial \underline{\mathbf{W}}}\right]^T \left[\frac{\partial \hat{y}(k)}{\partial \underline{\mathbf{W}}}\right] > 0$$

$$\overline{\lambda} = \overline{\eta} \left[\frac{\partial \hat{y}(k)}{\partial \overline{\mathbf{W}}}\right]^T \left[\frac{\partial \hat{y}(k)}{\partial \overline{\mathbf{W}}}\right] > 0, \lambda_{\underline{r}} = \eta_{\underline{r}} \left[\frac{\partial \hat{y}(k)}{\partial \underline{r}}\right]^T \left[\frac{\partial \hat{y}(k)}{\partial \underline{r}}\right] > 0$$

$$\underline{\lambda}_a = \eta_a \left[\frac{\partial \hat{y}(k)}{\partial \underline{\mathbf{W}}_a}\right]^T \left[\frac{\partial \hat{y}(k)}{\partial \underline{\mathbf{W}}_a}\right] > 0, \overline{\lambda}_a = \eta_a \left[\frac{\partial \hat{y}(k)}{\partial \overline{\mathbf{W}}_a}\right]^T \left[\frac{\partial \hat{y}(k)}{\partial \overline{\mathbf{W}}_a}\right] > 0$$

$$\underline{\lambda}^F = \underline{\eta} \left[\frac{\partial \hat{y}(k)}{\partial \underline{\mathbf{W}}^F}\right]^T \left[\frac{\partial \hat{y}(k)}{\partial \underline{\mathbf{W}}^F}\right] > 0, \overline{\lambda}^F = \overline{\eta} \left[\frac{\partial \hat{y}(k)}{\partial \overline{\mathbf{W}}^F}\right]^T \left[\frac{\partial \hat{y}(k)}{\partial \overline{\mathbf{W}}^F}\right] > 0$$

$$\lambda_{\underline{r}^F} = \eta_{\underline{r}} \left[\frac{\partial \hat{y}(k)}{\partial \underline{r}^F}\right]^T \left[\frac{\partial \hat{y}(k)}{\partial \underline{r}^F}\right] > 0.$$

Proof:
First, we define the Lyapunov function as follows:

$$V(k) = \frac{1}{2}\left[y_d(k) - \hat{y}(k)\right]^2 = \frac{1}{2}e^2(k) \tag{51}$$

where $\hat{y}\,(k)$ is RIT2FNN-A's system output, $y_d(k)$ is desired output and $e(k)$ denotes the approximated error. Thus, the change of V(k) is

$$\Delta V(k) = \frac{1}{2}\left[e^2(k+1) - e^2(k)\right] = \frac{1}{2}\left[e(k+1) + e(k)\right]\left[e(k+1) - e(k)\right]. \tag{52}$$

The error difference due to the learning can be represented by

$$\Delta e(k) = e(k+1) - e(k) \approx \left[\frac{\partial e(k)}{\partial \mathbf{W}}\right]^T \Delta\mathbf{W} \tag{53}$$

where $\left[\frac{\partial e(k)}{\partial \mathbf{W}}\right]^T = \left[\frac{\partial e(k)}{\partial \mathbf{W}_w} \;\; \frac{\partial e(k)}{\partial \underline{\mathbf{W}}} \;\; \frac{\partial e(k)}{\partial \overline{\mathbf{W}}} \;\; \frac{\partial e(k)}{\partial \underline{r}} \;\; \frac{\partial e(k)}{\partial \underline{\mathbf{W}}_a} \;\; \frac{\partial e(k)}{\partial \overline{\mathbf{W}}_a} \;\; \frac{\partial e(k)}{\partial \underline{\mathbf{W}}^F} \;\; \frac{\partial e(k)}{\partial \overline{\mathbf{W}}^F} \;\; \frac{\partial e(k)}{\partial \underline{r}^F}\right]^T$,

$\Delta\mathbf{W}^T = [\Delta\mathbf{W}_w \; \Delta\underline{\mathbf{W}} \; \Delta\overline{\mathbf{W}} \; \Delta r \; \Delta\underline{\mathbf{W}}_a \; \Delta\overline{\mathbf{W}}_a \; \Delta\underline{\mathbf{W}}^F \; \Delta\overline{\mathbf{W}}^F \; \Delta r^F]$

$$\Delta\mathbf{W}_w = e(k)\eta_w \left[\frac{\partial \hat{y}(k)}{\partial \mathbf{W}_w}\right]^T = e(k)\eta_w \begin{bmatrix} \frac{\partial \hat{y}(k)}{\partial \underline{w}} \\ \frac{\partial \hat{y}(k)}{\partial \overline{w}} \end{bmatrix}, \qquad \Delta\underline{\mathbf{W}} = e(k)\underline{\eta}\left[\frac{\partial \hat{y}(k)}{\partial \underline{\mathbf{W}}}\right]^T = e(k)\underline{\eta}\begin{bmatrix} \frac{\partial \hat{y}(k)}{\partial \underline{m}^{(l)}} \\ \frac{\partial \hat{y}(k)}{\partial \underline{m}^{(r)}} \\ \frac{\partial \hat{y}(k)}{\partial \underline{\sigma}^{(l)}} \\ \frac{\partial \hat{y}(k)}{\partial \underline{\sigma}^{(r)}} \end{bmatrix},$$

$$\Delta\overline{\mathbf{W}} = e(k)\overline{\eta}\left[\frac{\partial \hat{y}(k)}{\partial \overline{\mathbf{W}}}\right]^T = e(k)\overline{\eta}\begin{bmatrix} \frac{\partial \hat{y}(k)}{\partial \overline{m}^{(l)}} \\ \frac{\partial \hat{y}(k)}{\partial \overline{m}^{(r)}} \\ \frac{\partial \hat{y}(k)}{\partial \overline{\sigma}^{(l)}} \\ \frac{\partial \hat{y}(k)}{\partial \overline{\sigma}^{(r)}} \end{bmatrix}, \qquad \Delta\underline{r} = e(k)\eta_{\underline{r}}\frac{\partial \hat{y}(k)}{\partial \underline{r}},$$

$$\Delta\mathbf{W}_a = e(k)\eta_a\left[\frac{\partial\hat{y}(k)}{\partial\mathbf{W}_a}\right]^T = e(k)\eta_a\begin{bmatrix}\dfrac{\partial\hat{y}(k)}{\partial\underline{W}_a}\\ \dfrac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}_a}\end{bmatrix}, \qquad \Delta\overline{\mathbf{W}}_a = e(k)\eta_a\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}_a}\right]^T,$$

$$\Delta\underline{\mathbf{W}}^F = e(k)\underline{\eta}\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}^F}\right]^T = e(k)\underline{\eta}\begin{bmatrix}\dfrac{\partial\hat{y}(k)}{\partial\underline{m}^{F(l)}}\\ \dfrac{\partial\hat{y}(k)}{\partial\underline{m}^{F(r)}}\\ \dfrac{\partial\hat{y}(k)}{\partial\underline{\sigma}^{F(l)}}\\ \dfrac{\partial\hat{y}(k)}{\partial\underline{\sigma}^{F(r)}}\end{bmatrix}, \qquad \Delta\overline{\mathbf{W}}^F = e(k)\overline{\eta}\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}^F}\right]^T = e(k)\overline{\eta}\begin{bmatrix}\dfrac{\partial\hat{y}(k)}{\partial\overline{m}^{F(l)}}\\ \dfrac{\partial\hat{y}(k)}{\partial\overline{m}^{F(r)}}\\ \dfrac{\partial\hat{y}(k)}{\partial\overline{\sigma}^{F(l)}}\\ \dfrac{\partial\hat{y}(k)}{\partial\overline{\sigma}^{F(r)}}\end{bmatrix},$$

$$\Delta\underline{r}^F = e(k)\eta_{\underline{r}}\frac{\partial\hat{y}(k)}{\partial\underline{r}^F}.$$

Therefore, the change in the Lyapunov function is

$$\begin{aligned}
\Delta V(k) &= \frac{1}{2}\Delta e(k)\left[2e(k)+\Delta e(k)\right]\\
&= \frac{1}{2}\Bigg\{e(k)\eta_w\left[\frac{\partial e(k)}{\partial\mathbf{W}_w}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\mathbf{W}_w}\right] + e(k)\underline{\eta}\left[\frac{\partial e(k)}{\partial\underline{\mathbf{W}}}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}}\right] + e(k)\overline{\eta}\left[\frac{\partial e(k)}{\partial\overline{\mathbf{W}}}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}}\right]\\
&\quad + e(k)\eta_{\underline{r}}\left[\frac{\partial e(k)}{\partial\underline{r}}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{r}}\right] + e(k)\eta_a\left[\frac{\partial e(k)}{\partial\underline{\mathbf{W}}_a}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}_a}\right] + e(k)\eta_a\left[\frac{\partial e(k)}{\partial\overline{\mathbf{W}}_a}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}_a}\right]\\
&\quad + e(k)\underline{\eta}\left[\frac{\partial e(k)}{\partial\underline{\mathbf{W}}^F}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}^F}\right] + e(k)\overline{\eta}\left[\frac{\partial e(k)}{\partial\overline{\mathbf{W}}^F}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}^F}\right] + e(k)\eta_{\underline{r}}\left[\frac{\partial e(k)}{\partial\underline{r}^F}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{r}^F}\right]\Bigg\}\\
&\quad \times\Bigg\{2e(k) + e(k)\eta_w\left[\frac{\partial e(k)}{\partial\mathbf{W}_w}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\mathbf{W}_w}\right] + e(k)\underline{\eta}\left[\frac{\partial e(k)}{\partial\underline{\mathbf{W}}}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}}\right]\\
&\quad + e(k)\overline{\eta}\left[\frac{\partial e(k)}{\partial\overline{\mathbf{W}}}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}}\right] + e(k)\eta_{\underline{r}}\left[\frac{\partial e(k)}{\partial\underline{r}}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{r}}\right] + e(k)\eta_a\left[\frac{\partial e(k)}{\partial\underline{\mathbf{W}}_a}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}_a}\right]\\
&\quad + e(k)\eta_a\left[\frac{\partial e(k)}{\partial\overline{\mathbf{W}}_a}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}_a}\right] + e(k)\underline{\eta}\left[\frac{\partial e(k)}{\partial\underline{\mathbf{W}}^F}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}^F}\right] + e(k)\overline{\eta}\left[\frac{\partial e(k)}{\partial\overline{\mathbf{W}}^F}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}^F}\right]\\
&\quad + e(k)\eta_{\underline{r}}\left[\frac{\partial e(k)}{\partial\underline{r}^F}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{r}^F}\right]\Bigg\}\\
&= -\left[e(k)\right]^2\frac{1}{2}\Bigg\{\eta_w\left[\frac{\partial\hat{y}(k)}{\partial\mathbf{W}_w}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\mathbf{W}_w}\right] + \underline{\eta}\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}}\right] + \overline{\eta}\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}}\right]
\end{aligned}$$

$$
\begin{aligned}
&+\eta_{\underline{r}}\left[\frac{\partial\hat{y}(k)}{\partial\underline{r}}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{r}}\right]+\eta_a\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}_a}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}_a}\right]+\eta_a\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}_a}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}_a}\right]\\
&+\underline{\eta}\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}^F}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}^F}\right]+\overline{\eta}\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}^F}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}^F}\right]+\eta_{\underline{r}}\left[\frac{\partial\hat{y}(k)}{\partial\underline{r}^F}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{r}^F}\right]\Bigg\}\\
&\times\Bigg\{2-\eta_w\left[\frac{\partial\hat{y}(k)}{\partial\mathbf{W}_w}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\mathbf{W}_w}\right]-\underline{\eta}\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}}\right]-\overline{\eta}\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}}\right]\\
&-\eta_{\underline{r}}\left[\frac{\partial\hat{y}(k)}{\partial\underline{r}}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{r}}\right]-\eta_a\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}_a}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}_a}\right]-\eta_a\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}_a}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}_a}\right]\\
&-\underline{\eta}\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}^F}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}^F}\right]-\overline{\eta}\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}^F}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}^F}\right]-\eta_{\underline{r}}\left[\frac{\partial\hat{y}(k)}{\partial\underline{r}^F}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{r}^F}\right]\Bigg\}\\
&=-[e(k)]^2\frac{1}{2}\left(\lambda_w+\underline{\lambda}+\overline{\lambda}+\lambda_{\underline{r}}+\underline{\lambda}_a+\overline{\lambda}_a+\underline{\lambda}^F+\overline{\lambda}^F+\lambda_{\underline{r}^F}\right)\times\left(2-\lambda_w-\underline{\lambda}\right.\\
&\left.-\overline{\lambda}-\lambda_{\underline{r}}-\underline{\lambda}_a-\overline{\lambda}_a-\underline{\lambda}^F-\overline{\lambda}^F-\lambda_{\underline{r}^F}\right)\\
&=-[e(k)]^2\frac{1}{2}\lambda
\end{aligned}
\tag{54}
$$

where

$$\lambda_w=\eta_w\left[\frac{\partial\hat{y}(k)}{\partial\mathbf{W}_w}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\mathbf{W}_w}\right]>0,\qquad \underline{\lambda}=\underline{\eta}\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}}\right]>0,$$

$$\overline{\lambda}=\overline{\eta}\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}}\right]>0,\qquad \lambda_{\underline{r}}=\eta_{\underline{r}}\left[\frac{\partial\hat{y}(k)}{\partial\underline{r}}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{r}}\right]>0,$$

$$\underline{\lambda}_a=\eta_a\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}_a}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}_a}\right]>0,\qquad \overline{\lambda}_a=\eta_a\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}_a}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}_a}\right]>0,$$

$$\underline{\lambda}^F=\underline{\eta}\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}^F}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{\mathbf{W}}^F}\right]>0,\qquad \overline{\lambda}^F=\overline{\eta}\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}^F}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\overline{\mathbf{W}}^F}\right]>0,\ \text{and}$$

$$\lambda_{\underline{r}^F}=\eta_{\underline{r}}\left[\frac{\partial\hat{y}(k)}{\partial\underline{r}^F}\right]^T\left[\frac{\partial\hat{y}(k)}{\partial\underline{r}^F}\right]>0.$$

Let

$$
\begin{aligned}
\lambda=&\left(\lambda_w+\underline{\lambda}+\overline{\lambda}+\lambda_{\underline{r}}+\underline{\lambda}_a+\overline{\lambda}_a+\underline{\lambda}^F+\overline{\lambda}^F+\lambda_{\underline{r}^F}\right)\times\left(2-\lambda_w-\underline{\lambda}-\overline{\lambda}-\lambda_{\underline{r}}\right.\\
&\left.-\underline{\lambda}_a-\overline{\lambda}_a-\underline{\lambda}^F-\overline{\lambda}^F-\lambda_{\underline{r}^F}\right).
\end{aligned}
$$

The convergence of RiT2FNN-A is guaranteed if $\Delta V(k) < 0$, i.e., $\lambda > 0$, and

$$\left(\lambda_w + \underline{\lambda} + \overline{\lambda} + \lambda_{\underline{r}} + \underline{\lambda}_a + \overline{\lambda}_a + \underline{\lambda}^F + \overline{\lambda}^F + \lambda_{\underline{r}^F}\right) < 2 .$$

This completes the proof.

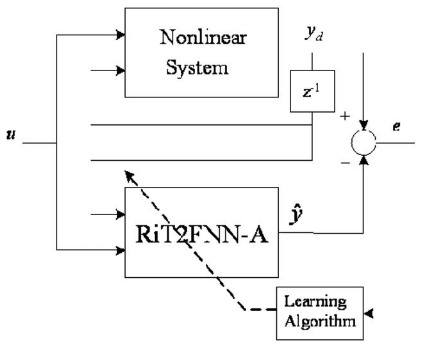


Fig. 7. Series-parallel identification scheme using RiT2FNN-A.

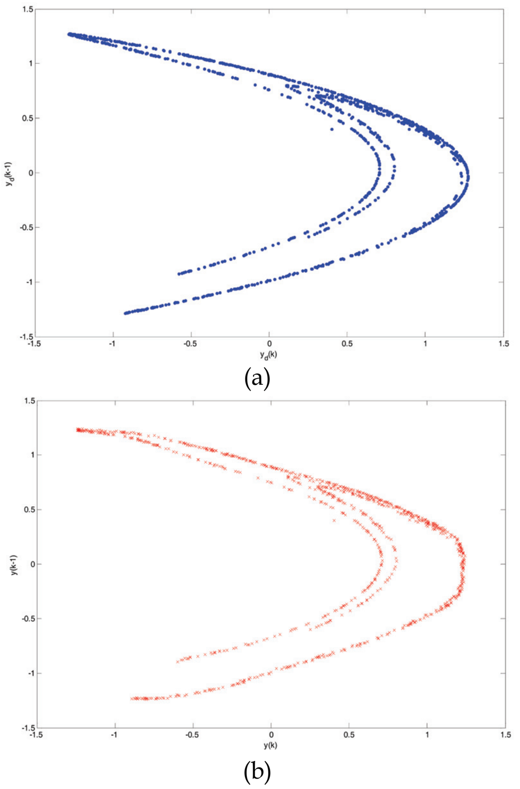


Fig. 8. Phase plane plot of chaotic system; (a) the chaotic system, (b) identification result of RiT2FNN-A.

4. System identification using RiT2FNN-A system

Consider the following non-linear system

$$y_d(k+1) = f(y_d(k), \cdots, y_d(k-n+1), u(k), \cdots, u(k-m+1)) \tag{55}$$

where *u* and y_d are systematic input and output; function f(.) is the unknown function which is approximated by the RiT2FNN-A. And then *m* and *n* are all positive integer number. Here, the series-parallel training scheme is adopted, as shown in Fig. 7. The approximated error is defined as follows

$$e(k) \equiv y_d(k) - \hat{y}(k) \tag{56}$$

where $\hat{y}$ (*k*) denotes the RiT2FNN-A's output. Clearly, the inputs of RiT2FNN-A are contro input u and system past input $y_d(k\text{-}1)$. If a static network system (or feed-forward neural network) is used, such as, neural network, fuzzy neural network, T2FNN, T2FNN-A, the input number of *n*+*m* should be used. This is due to the dynamic property (feedback layer) of RiT2FNN-A system.
In general, the following Training-Mean-Square-Error (TMSE) is adopted to be the performance index.

$$TMSE \equiv \frac{1}{N}\sum_{k=1}^{N} e^2(k) \tag{57}$$

where *N* is the number of training pattern.
In this article, the following nonlinear chaotic system is considered

$$y_d(k) = -P \cdot y_d^2(k-1) + Q \cdot y_d(k-2) + 1.0 \tag{58}$$

where *P*=1.4 and *Q*=0.3.
The feed-forward type-2 fuzzy neural network- T2FNN and T2FNN-A, are used to have comparisons in nonlinear system identification for illustrating the performance of RiT2FNNA. It is clear that the feed-forward T2FNN with three input nodes for feeding appropriate past values of yd and u were used. In this article, only two values, $y_d(k\text{-}1)$ and u(k), are fed into the RiT2FNN-A to predict the system output. In training the RiT2FNN-A, we first randomly choose the training data (1000 pairs) from system over the interval [-1.5 1.5]. Then, the RiT2FNN-A is used to approximate the chaotic system. In this simulation, we use 3 rules to construct the RiT2FNN-A. Learning rate is selected as 0.1.
The simulation results are described in Figs. 8 and 9. Figure 8(a) shows the phase plane of this chaotic system, whereas Fig. 8(b) shows the result of RiT2FNN-A system after training (10 epochs). The initial point is $[y_d(1), y_d(0)]$T=[0.4, 0.4]T and the TMSE is 0.00019886, which is less than the results of T2FNN-A and T2FNN (as shown in Fig. 9). The initial interval T2MFs for input and internal variables *x* and *g* are empirically designed as Figs. 10(a) and 10(b), respectively. After training, the final iT2AFMFs are shown in Figs. 10(c) and 10(d). Obviously, the iT2AFMFs are obtained for better performance.
In order to make sure RiT2FNN-A system to be stable in training, we need to check the condition (50). Figure 11 shows the values of $\beta = \lambda_w + \underline{\lambda} + \overline{\lambda} + \lambda_{\underline{r}} + \underline{\lambda}_a + \overline{\lambda}_a + \underline{\lambda}^F + \overline{\lambda}^F + \lambda_{\underline{r}^F}$

which were introduced previously in (50), the stable condition hold if β<2. Obviously, condition (50) holds in training epochs.

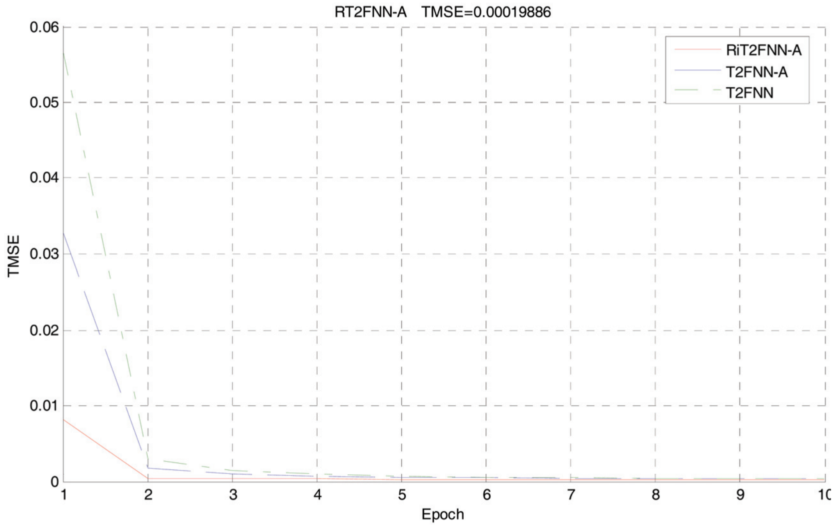
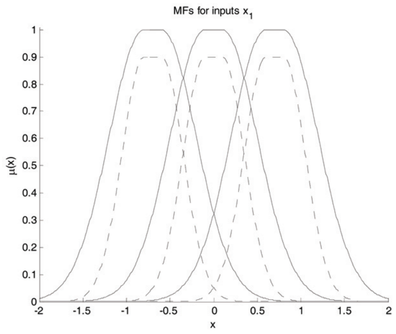
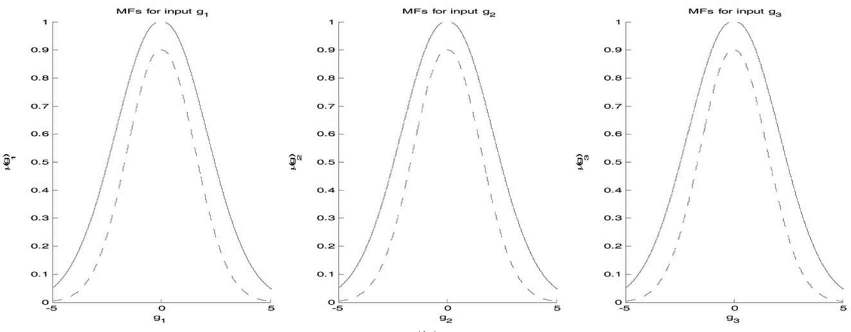


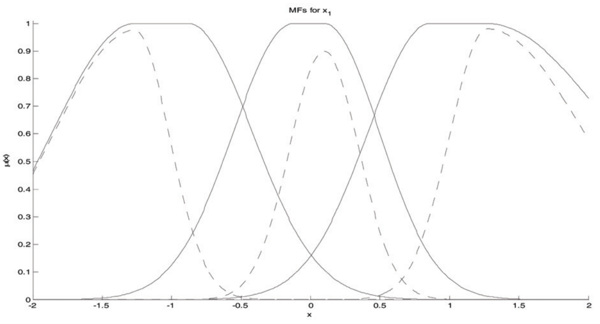
Fig. 9. Simulation results of system identification; (a) system output, (b) learning curves of the T2FNN (dotted-line), T2FNN-A (dashed-line) and RiT2FNN-A (solid-line).



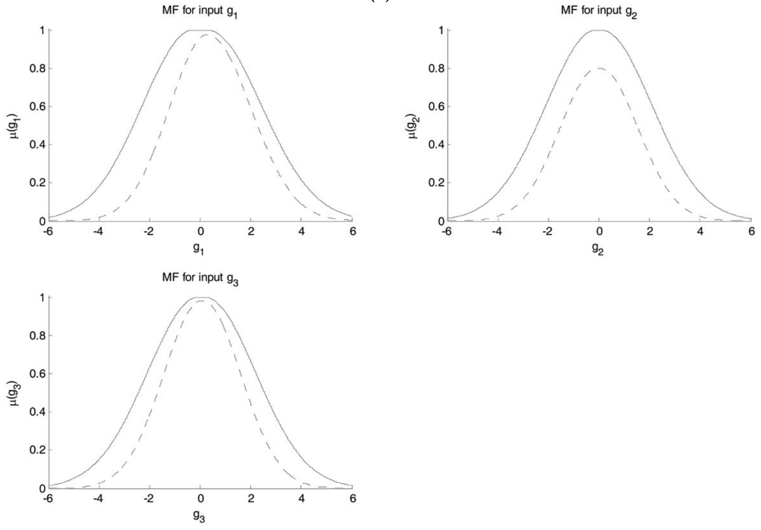
(a)



(b)



(c)



(d)

Fig. 10. Membership functions; (a) initial MFs for x_1, (b) initial MFs for g_1, g_2, and g_3, (c) MFs for x_1 after training, and (d) MFs for g_1, g_2, and g_3 after training.

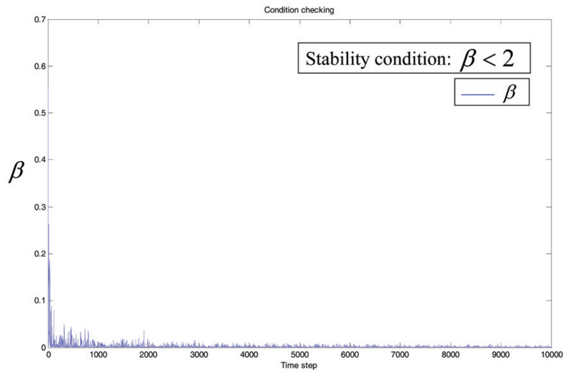


Fig. 11. Condition checking of (50).

This simulation demonstrates that the RiT2FNN-A has the smaller network structure for identification. In addition, we observe that the identification error of the RiT2FNN-A is less than that of T2FNN-A for each epoch.

	Network structure	Rule number (M)	Parameter number	TMSE ($\times 10^{-3}$)
T2FNN (Lee & Lin, 2005)	2-10-5-1	5	40	0.35
	1-3-3-1	3	15	215.5
T2FNN-A (Lee & Pan, 2007)	2-6-3-1	3	36	0.34
	1-3-3-1	3	21	195.9
RiT2FNN-A	1-3-3-1	3	24	0.20

Table 1. Comparison results of network structure, rule number, parameter number, and TMSE.

Table 1 shows the comparison results of network structure, rule number, parameter number, and TMSE. Obviously, the asymmetric MFs improve the approximation accuracy of the iT2FLSs. On the other hand, for a given approximation accuracy, RiT2FNN-A can achieve by using less fuzzy rules and tuning parameters with simplified structure.

5. Conclusion

This article has introduced a novel recurrent interval type-2 fuzzy neural network with asymmetric membership functions, which utilizes Lyapunov stability theorem to prove the stability of the system. The novel RiT2FNN-A use the interval asymmetric type-2 fuzzy sets implements the FLS in a five-layer neural network structure which contains four layer forward network and a feedback layer. According to the Lyapunov theorem and gradient descent method, the convergence of RiT2FNN-A is guaranteed and the corresponding learning algorithm is derived. Moreover, the RIT2FNN-A capability to temporarily store information allowed us to extend the application domain to include temporal problem. In application, We have found that the proposed RiT2FNN-A can use a smaller network structure and a small number of tuning parameters than the feed-forward fuzzy neural networks to obtain similar or better performance. It can successfully also approximate to a dynamic system mapping as accurately as desired.

6. References

Baldwin, J. F. & Karake, S. B. (2003). Asymmetric Triangular Fuzzy Sets for Classification Models, *Lecture Notes in Artificial Intelligence*, Vol. 2773, pp. 364-370, 2003.

Castillo, O. & Melin, P. (2004). Adaptive Noise Cancellation Using Type-2 Fuzzy Logic and Neural Networks, *IEEE International Conf. on Fuzzy Systems*, Vol. 2, pp. 1093-1098, 2004.

Jang, J. S. (1993). ANFIS: Adaptive-Network-Based Fuzzy Inference System, *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. 23, No. 3, pp. 665-685, 1993.

John, R. & Coupl, S. (1999). Geometric Type-1 and Type-2 Fuzzy Logic Systems, *IEEE Trans. on Fuzzy Systems, Special Issue on Type-2 Fuzzy Systems*, Vol. 15, No. 1, pp. 3-15, 2007.

Juang, C. F. (2002), A TSK-type recurrent fuzzy network for dynamic systems processingby neural network and genetic algorithms, *IEEE trans. on Fuzzy Systems*, Vol. 10, No. 2, pp. 155- 170, 2002.

Karnik, N. N., Mendel, J. & Liang, Q. (1999). Type-2 Fuzzy Logic Systems, *IEEE Trans. on Fuzzy Systems*, Vol. 7, No. 6, pp. 643-658, 1999.

Kim, M. S., Kim, C. H. & Lee, J. J. (2003). Evolutionary Optimization of Fuzzy Models with Asymmetric RBF Membership Functions Using Simplified Fitness Sharing, *Lecture Notes in Artificial Intelligence*, Vol. 2715, pp. 628-635, 2003.

Lee, C. H. & Lin, Y. C. (2005). An Adaptive Type-2 Fuzzy Neural Controller for Nonlinear Uncertain Systems, *International Journal of Control and Intelligent*, Vol. 12, No. 1, pp. 41-50, 2005.

Lee, C. H. & Teng, C. C. (2000). Identification and Control of Dynamic Systems Using Recurrent Fuzzy Neural Networks, *IEEE Trans. on Fuzzy Systems*, Vol. 8, No. 4, pp. 349-366, 2000.

Lee, C. H. & Teng, C. C. (2001).Fine Tuning of Membership Functions for Fuzzy Neural Systems, *Asian Journal of Control*, Vol. 3, No. 3, pp. 216-225, 2001.

Lee, C. H. Lin, Y. C. & Lai, W. Y. (2003). Systems Identification Using Type-2 Fuzzy Neural Network (Type-2 FNN) Systems, *IEEE International Sym. on Computational Intelligence in Robotics and Automation*, Vol. 3, pp. 1264-1269, 2003.

Lee, C. H. & Pan, H. Y. (2007). Enhancing the Performance of Neural Fuzzy Systems Using Asymmetric Membership Functions, Revised in *Fuzzy Sets and Systems*, 2007.

Li, C. Cheng, K. H. & Lee, J. D. (2005). Hybrid Learning Neuro-fuzzy Approach for Complex Modeling Using Asymmetric Fuzzy Sets, *Proc. of the 17th IEEE International Conf. on Tools with Artificial Intelligence*, pp. 397-401, 2005.

Li, C. & Lee, C. Y. (2003). Self-organizing Neuro-fuzzy System for Control of Unknown Plants, *IEEE Trans. on Fuzzy Systems*, Vol. 11, No. 1, pp. 135-150, 2003.

A. Q. Liang & J. M. Mendel, Interval Type-2 Fuzzy Logic Systems: Theory and Design, *IEEE Trans. on Fuzzy Systems*, Vol. 8, No. 5, pp. 535-550, 2000.

Lin, C. T. & Lee, C. S. G. *Neural Fuzzy Systems: A Neuro-fuzzy Synergism to Intelligent Systems*, Prentice-Hall, Englewood Cliffs, 1996.

Lin, C. J. & Ho, W. H. (2005). An Asymmetric-similarity-measure-based Neural Fuzzy Inference System, *Fuzzy Sets and Systems*, Vol. 152, pp. 535-551, 2005.

Lotfi, A. & Tsoi, A. C. (1996). Learning Fuzzy Inference Systems Using An Adaptive Membership Function Scheme, *IEEE Trans. on Systems, Man, and Cybernetics, Part-B*, Vol. 26, No. 2, pp. 326-331, 1996.

Mendel, J. M. (2001). *Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions*, Upper Saddle River, Prentice-Hall, NJ, 2001.

Hagras, H. (2007). Type-2 FLCs: A New Generation of Fuzzy Controllers, *IEEE Computational Intelligence Magazine*, Vol. 2, No. 1, pp. 30-43, 2007.

Narendra, K. S. & Parthasarathy, K. (1990). Identification and Control of Dynamical Systems Using Neural Networks, *IEEE Trans. on Neural Networks*, Vol. 1, No. 1, pp. 4-27, 1990.

Ozen, T. & Garibaldi, J. M. (2004). Effect of Type-2 Fuzzy Membership Function Shape on Modeling Variation in Human Decision Making, *IEEE International Conf. on Fuzzy Systems*, Vol.2, pp. 971-976, 2004.

Pan, H. Y., Lee, C. H., Chang, F. K., & Chang, S. K. (2007). Construction of Asymmetric Type-2 Fuzzy Membership Functions and Application in Time Series Prediction, *International Conf. on Machine Learning and Cybernetics,* Vol. 4, pp. 2024-2030, 2007.

Wang, C. H. Cheng, C. S. & Lee, T. T. (2004). Dynamical Optimal Training for Interval Type-2 Fuzzy Neural Network (T2FNN), *IEEE Trans. on Systems, Man, Cybernetics Part-B*, Vol. 34, No. 3, pp. 1462-1477, 2004.

Wang, L. X. (1994). *Adaptive Fuzzy Systems and Control: Design and Stability Analysis*, Prentice -Hall, Englewood Cliffs, NJ, 1994.

Zadeh, L. A. (1975). The Concept of A Linguistic Variable and Its Application to Approximate Reasoning, *Information Sciences*, Vol. 8, No.3, pp. 199-249, 1975.

Rollover Control in Heavy Vehicles via Recurrent High Order Neural Networks

Luis J. Ricalde[1], Edgar N. Sanchez[2], Reza Langari[3] and Danial Shahmirzadi[3]
[1] Universidad Autonoma de Yucatan, Faculty of Engineering, Merida, Yuc.,
[2] CINVESTAV, Unidad Guadalajara, Guadalajara, Jalisco,
[3] Texas A&M University, College Station, TX,
[1,2]Mexico
[3]USA

1. Introduction

Heavy vehicles, such as tractor-semitrailers, play an important role in transportation systems. They present more complex dynamical behavior than passenger cars, due to their high centers of gravity, which can vary depending on the load conditions, and are highly susceptible to rollover during cornering. Heavy vehicle rollover on highways occurs as a result of cornering with excessively high speed, cornering on a small radius curve or sudden lane change. However, if rollover threat is predicted using an appropriate algorithm, then the accident can be prevented by the driver's corrective maneuvers. For situations where rollover warning is ineffective, active rollover control is necessary.

Most of the rollover warning algorithms use instantaneous rollover-threat index to identify the rollover threat. Since a rollover warning may be issued at 75 % of the rollover threshold acceleration, the time from warning to rollover is too short for the driver to respond effectively. However, if the rollover threat is predicted using the expected maneuvers, a warning can be issued sufficiently in advance of the event. This fact implies that warning systems based on predicted rollover threat can be more effective.

Many control strategies have been designed to prevent rollover, most of them based on active speed control and active roll control. However, active roll control is ineffective for sharp turns, since it does not reduce the lateral acceleration, and requires hydraulic actuators which increase the cost considerably. On the other hand, the use of differential braking prevent jack-knifing and loss of direction generated by sudden braking during cornering.

Different loading configurations produce different reaction forces on each wheel. This motivates the use of nonlinear robust controllers which have to be able to deal with parametric uncertainties, but most controllers are based on reduced models, in order to lessen the computational requirements. Many mathematical models for tractor semitrailers have been developed in order to derive active control algorithms. The Automotive Research Center of the University of Michigan developed the 33 degrees-of-freedom ArcSim model (UMTRI, 1997) to study the acceleration/braking and handling responses of an US Army 6-axle tractor-semitrailer. In (Hyun & Langari, 2003), the vehicle model for single-unit heavy

vehicles and tractor-semitrailers was derived using Lagrange's equations and Newtonian mechanics; this model was validated by examining its steady-state response characteristics and comparing it with ArcSim obtaining similar results but with less computational complexity. Then, an algorithm to identify the rollover threshold, the measure of roll stability, in terms of vehicle lateral acceleration or roll angle is established. In this paper we used the model presented in (Hyun & Langari, 2003) for simulations.
On the other hand, since the seminal paper (Narendra & Parthasarathy, 1990), there has been continually increasing interest in applying neural networks to identification and control of nonlinear systems. Lately, the use of recurrent neural networks is being developed, which allows more efficient modeling of the underlying dynamical systems (Poznyak et al. 1999). Three representative books (Poznyak et al. 2000), (Rovitahkis & Christodoulou, 2000) and (Suykens et al., 1996) have reviewed the application of recurrent neural networks for nonlinear system identification and control. In particular, (Suykens et al., 1996) uses off-line learning, while (Rovitahkis & Christodoulou, 2000) analyzes adaptive identification and control by means of on-line learning, where stability of the closed-loop system is established based on the Lyapunov function method. In (Rovitahkis & Christodoulou, 2000), the trajectory tracking problem is reduced to a linear model following problem, with application to DC electric motors. In (Poznyak et al. 2000), analysis of Recurrent Neural Networks for identification, estimation and control are developed, with applications on chaos control, robotics and chemical processes.

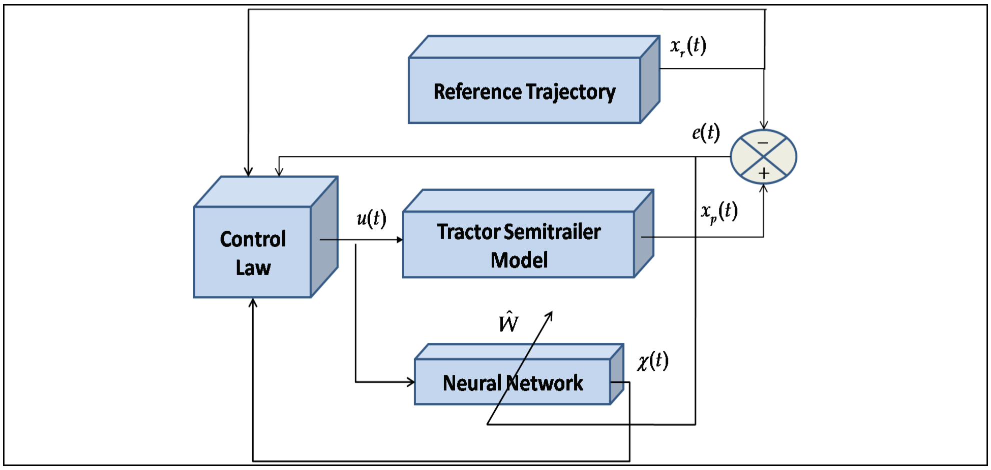


Fig. 1. Recurrent neural control scheme

Control methods which are applicable to general nonlinear systems have been intensely developed since the early 1980's. Recently, the passivity approach has generated increasing interest for synthesizing control laws (Hill & Moylan, 1996). An important problem for these approaches is how to achieve robust nonlinear control in the presence of unmodelled dynamics and external disturbances. In this direction, there exists the so-called H_∞ nonlinear control approach (Basar & Bernhard, 1995). One major difficulty with this approach, alongside its possible system structural instability, seems to be the requirement of solving some resulting partial differential equations. In order to alleviate this computational problem, the so-called inverse optimal control technique was recently developed, based on

the input-to-state stability concept (Krstic & Deng, 1999). In (Sanchez et al., 2002), a robust adaptive neural controller for nonlinear systems with uncertainties is considered, in order to guarantee stability and trajectory tracking; a direct control approach is considered, where a recurrent neural network is assumed to model the unknown system and a control law is designed using the Lyapunov methodology and the inverse optimal control approach (Krstic & Deng, 1999).
In this article we use Recurrent Neural Networks for applications to rollover prevention on heavy vehicles where we consider the presence of uncertainties and unmodeled dynamics. An active control algorithm is developed to prevent rollover if corrective actions from the driver are not done after receiving alarm signals for rollover threats. The proposed adaptive control scheme, as shown in Fig. 1, is composed of a recurrent neural identifier and a controller, where the former is used to build an on-line model for the unknown plant, and the latter to force the unknown plant to track the reference trajectory. An update law for the high order recurrent neural network weights is proposed via the Lyapunov methodology. The control law is synthesized using the Lyapunov methodology and the inverse optimal control approach. The algorithm is tested, via simulations, for prevention of rollover of the tractor semitrailer model developed in (Hyun & Langari, 2003). Speed only control and Speed-Yaw rate control are applied in order to reduce the lateral acceleration and roll angle of the trailer. The list of symbols that appear in this chapter are presented in Table 1 and Table 2.

A	Lipschitz matrix in the Recurrent Neural Network system
a_{yt}	Lateral acceleration rollover threshold
e	Tracking error
$f_p(\cdot)$, $f_r(\cdot)$	Vector field for the vehicle and reference dynamics
F_{Ti}	Normal tire forces for wheel i-th
$g_p(\cdot)$	Input vector field for the vehicle dynamics
k	Sigmoid slope parameter
L	Number of high order connections
L_f, L_r	Front and rear segments of tractor wheelbase
L_fV, L_gV	Lie derivatives of the Lyapunov function respect of $f_p(\cdot)$ and $g_p(\cdot)$
$l(\cdot)$	Positive semidefinite function for Hamilton-Jacobi-Bellman system
$R(\cdot)$	Positive definite function for cost function evaluation
$S(\cdot)$	Sigmoid function
u	Applied input
v_x	Longitudinal speed
$\dot{v}_y$	Lateral acceleration

Table 1. List of symbols

W, W_g	Estimated weights matrices
W^*, W_g^*	Optimal weights matrices
$\tilde{W}, \tilde{W}_g$	Weight error matrices
x	Plant state to be identified
x_p	Unknown nonlinear state
x_r	Reference signal state
x_p, y_p, z_p	Longitudinal, lateral and vertical position for tractor sprung mass
x_N, y_N	Longitudinal and lateral reference coordinates
z_r	Vertical position of the tractor unsprung mass
$z(\cdot), z_g(\cdot)$	Sigmoid high order vectors
$\alpha_r(\cdot)$	Applied input forces for reference tracking of the neural network
β	Positive parameter for cost function
Γ, Γ_g	Learning rate matrices
δ	Steer input
ε	Relative pitch angle of the fifth wheel
ζ	Sigmoid function parameter
η	Relative yaw angle of the trailer
θ	Tractor pitch
λ	HORNN system parameter
μ	Gain matrix for the control law
ϖ	Vector of tractor states
τ	Parameter for sigmoid function
ϕ	Tractor roll angle
ϕ_t	Roll angle rollover threshold
χ	Neural network state
ψ	Tractor yaw angle
φ_d	Reference yaw angle
ω_i	Wheel i spin i =1,…,6

Table 2. List of symbols

2. System model description

In this paper, we consider as the simulation tool, the tractor-semitrailer model presented in (Hyun & Langari, 2003), which has 14 degrees of freedom:

x_N, y_N, z_r	Longitudinal, lateral and vertical position with respect to a coordinate system fixed to the ground
ψ	Tractor yaw angle
θ	Tractor pitch angle
ϕ	Tractor roll angle
ε	Relative pitch angle of the fifth wheel with respect to the tractor sprung mass coordinates (x_p, y_p, z_p)
η	Relative yaw angle of the trailer with respect to the tractor sprung mass coordinates (x_p, y_p, z_p)
ω_i	Wheel i spin i=1,...,6

This model is derived using Lagrange's equations as well as Newtonian mechanics. Nonlinear suspension and tire-force models are considered in the vehicle model. Fig. 2 and Fig. 3 display side, rear and yaw plane view of the trailer under consideration.

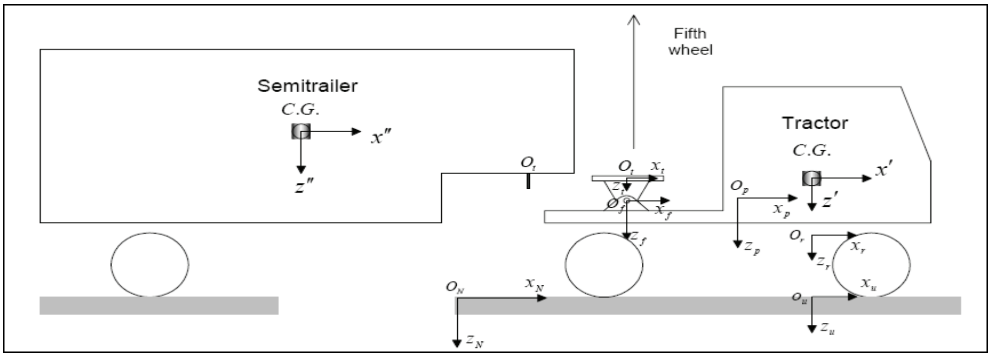


Fig. 2. Side view of the tractor-semitrailer

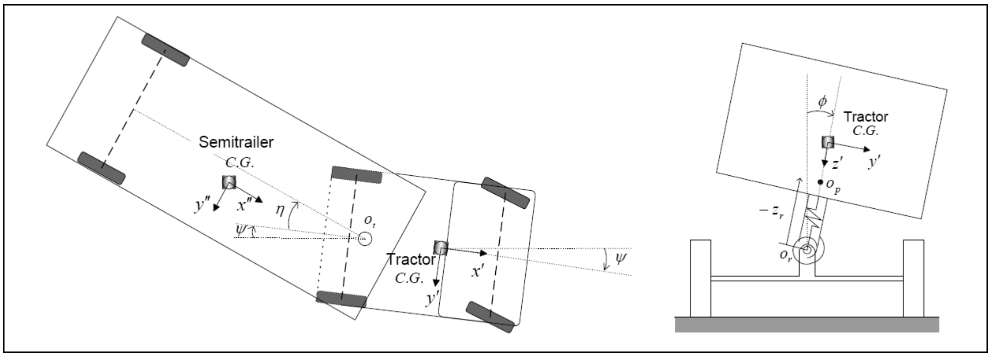


Fig. 3. Rear view and Yaw-plane view of the tractor-semitrailer.

3. Mathematical preliminaries

3.1 Artificial neural networks

Artificial neural networks have become an useful tool for control engineering thanks to their applicability on modelling, state estimation and control of complex dynamic systems. Using neural networks, control algorithms can be developed to be robust to uncertainties and modelling errors.

Neural Networks consist of a number of interconnected processing elements or neurons. The way in which the neurons are interconnected determines its structure. For identification and control, the most used structures are:

Feedforward networks. In feedforward networks, the neurons are grouped into layers. Signals flow from the input to the output via unidirectional connections. The network exhibits high degree of connectivity, contains one or more hidden layers of neurons and the activation function of each neuron is smooth, generally a sigmoid function.

Recurrent networks. In a recurrent neural network, the outputs of the neuron are fed back to the same neuron or neurons in the preceding layers. Signals flow in forward and backward directions.

3.2 Recurrent higher-order neural networks

Artificial Recurrent Neural Networks are mostly based on the Hopfield model (Hopfield, 1984). These networks are considered as good candidates for nonlinear systems applications which deal with uncertainties and are attractive due to their easy implementation, relatively simple structure, robustness and the capacity to adjust their parameters on line.

In (Kosmatopoulos, et al. 1997), Recurrent Higher-Order Neural Networks (RHONN) are defined as

$$\dot{\chi}_i = -\alpha_i \chi_i + \sum_{k=1}^{L} \omega_{ik} \prod_{j \in I_k} y_j^{d_j(k)}, \qquad i = 1, \ldots, n \tag{1}$$

where χ_i is the ith neuron state, L is the number of higher-order connections, $\{I_1, I_2, \ldots, I_L\}$ is a collection of non-ordered subsets of $\{1, 2, \ldots, m+n\}$, $a_i > 0$, w_{ik} are the adjustable weights of the neural network, $d_j(k)$ are nonnegative integers, and y is a vector defined by $y = [y_1, \ldots, y_n, y_{n+1}, \ldots, y_{n+m}]^T = [S(\chi_1), \ldots, S(\chi_n), S(u_1), \ldots, S(u_m)]^T$, with $u = [u_1, u_2, \ldots, u_m]^T$ being the input to the neural network, and $S(\bullet)$ a smooth sigmoid function formulated by $S(\chi) = \dfrac{1}{1 + \exp(-\tau\chi)} + \zeta$. For the sigmoid function, τ is a positive constant and ζ is a small positive real number. Hence, $S(\chi) \in [\zeta, \zeta + 1]$.

As can be seen, (1) allows the inclusion of higher-order terms.

By defining a vector

$$z(\chi, u) = [z_1(\chi, u) \ldots\ldots, z_L(\chi, u)]^T = \left[\Pi_{j \varepsilon I_1} y_j^{d_j(1)}, \ldots., \Pi_{j \varepsilon I_L} y_j^{d_j(L)}\right]^T$$

(1) can be rewritten as

$$\dot{\chi}_i = -\alpha_i \chi_i + \sum_{k=1}^{L} \omega_{ik} z_k(\chi, u) \quad , \qquad\qquad i = 1,...,n$$
$$\dot{\chi}_i = -\alpha_i \chi_i + \omega_i z(\chi, u) \quad , \tag{2}$$

where $w_i = \left[w_{i,1} .....w_{i,L} \right]^T$.

In this paper, terms as $y = \left[y_1 ,... y_n , y_n + 1,...., y_{n+m} \right]^T = \left[S(\chi_1),...,S(\chi_n),u_1,..,u_n \right]^T$ are considered. This means that the same number of inputs and states is used. We also assume that the RHONN is affine in the control, so that (2) can be rewritten as

$$\dot{\chi}_i = -\alpha_i \chi_i + \omega_i^T z(\chi) + \omega_{gi} u_i \quad , \tag{3}$$

Reformulating (3) in matrix form yields

$$\dot{\chi}_i = A\chi + Wz(\chi) + W_g u \tag{4}$$

where $\chi \in \Re^n , W \in \Re^{n\times L}, W_g \in \Re^{n\times n}, z(x) \in \Re^L, u \in \Re^n$, and $A = -\lambda I, \lambda > 0$.

4. Adaptive recurrent neural control for tractor-semitrailer

4.1 Problem formulation

The nonlinear system (tractor-semitrailer) model can be described as

$$\dot{x}_p = f_p(x_p) + g(x_p)u \tag{5}$$

We propose to model the unknown nonlinear plant by the recurrent neural network

$$\begin{aligned} \dot{x}_p &= \dot{\chi} + \omega_{per} \\ &= A\chi + W^* z(\chi) + (\chi - x_p) + W_g^* u \end{aligned} \tag{6}$$

where $A = -\lambda\chi, x_p \in \Re^n, x \in \Re^n, z(x) \in \Re^L, W_g{}^* \in \Re^{nxL}, W_g{}^* \in \Re^{nxm}, u \in \Re^m$ and $\omega_{per} = x - \chi_p$ represents the modelling error, with W^*, W_g^* being the unknown values of the neural network weights which minimize the modelling error.
We will design a robust controller which enforces asymptotic stability of the tracking error between the plant and the reference signal

$$\dot{x}_r = f_r(x_r, u_r) \tag{7}$$

namely,

$$e = x_p - x_r \tag{8}$$

Its time derivative is

$$\dot{e} = A\chi + W^* z(\chi) + (\chi - x_p) + W_g^* u - f_r(x_r, u_r) \tag{9}$$

Now, we proceed to add and subtract the terms $\hat{W}z(x_r), Ae, Ax_r, x_r$ and Ae, so that

$$\begin{aligned}\dot{e} = Ae + W^{*}\Gamma z(\chi) + W_g^{*}u + \left(-f_r(x_r,u_r) + Ax_r + \hat{W}z(x_r) + x_r - x_p\right) \\ -Ae - \hat{W}z(x_r) - Ax_r - x_r + \chi + A\chi\end{aligned} \tag{10}$$

where $\hat{W}$ is the estimated value for the unknown weight matrix W^{*}.
Let us assume that there exists a function $\alpha_r(t,\hat{W},\hat{W}_g)$ such that

$$\alpha_r(t,\hat{W}) = (\hat{W}_g)^{-1}\left(f_r(x_r,u_r) - Ax_r - \hat{W}\Gamma z(x_r) - (x_r - x_p)\right) \tag{11}$$

where $\hat{W}_g$ is the estimated value for the unknown weight matrix W_g^{*}.
Then, adding and subtracting to (10) the term $\hat{W}_g\alpha_r(t,\hat{W},\hat{W}_g)$ and simplifying we obtain

$$\dot{e} = Ae + W^{*}z(\chi) + W_g^{*}u - \hat{W}_g\alpha_r(t,\hat{W},\hat{W}_g) - A(x_p - x_r) - \hat{W}z(x_r) + (A+I)(\chi - x_r) \tag{12}$$

Next, let us define

$$\begin{aligned}\hat{W} &= W^{*} - \hat{W} \\ \hat{W}_g &= W_g{}^{*} - \hat{W}_g \\ \tilde{u} &= u - \alpha_r\left(t,\hat{W}\right)\end{aligned}$$

so that (12) is reduced to

$$\begin{aligned}\dot{e} &= Ae + (\tilde{W} + \hat{W})z(\chi) + (\tilde{W}_g + \hat{W}_g)u - \hat{W}_g\alpha_r(t,\hat{W},\hat{W}_g) - A(x_p - x_r) - \hat{W}z(x_r) \\ &\quad + (A+I)(\chi - x_r) \\ \dot{e} &= Ae + \tilde{W}z(\chi) + \hat{W}(z(\chi) - z(x_r)) + \tilde{W}_g u + \hat{W}_g\tilde{u} - A(x_p - x_r) + (A+I)(\chi - x_r)\end{aligned} \tag{13}$$

Adding and subtracting to (13) the terms $z\left(x_p\right)$ and x_p, we obtain

$$\begin{aligned}\dot{e} = Ae + \tilde{W}z(\chi) + \hat{W}(z(\chi) - z(x_p) + z(x_p) - z(x_r)) + \tilde{W}_g u + \hat{W}_g\tilde{u} \\ - A(x_p - x_r) + (A+I)(\chi - x_p + x_p - x_r)\end{aligned} \tag{14}$$

Then, by defining

$$\tilde{u} = u_1 + u_2 \tag{15}$$

with

$$u_1 = (\hat{W}_g)^{-1}\left(-\hat{W}(z(\chi) - z(x_p)) - (A+I)(\chi - x_p)\right) \tag{16}$$

equation (14) reduces to

$$\dot{e} = (A+I)e + \tilde{W}z(\chi) + \hat{W}(z(x_p) - z(x_r)) + \tilde{W}_g u + \hat{W}_g u_2 \tag{17}$$

Therefore, the tracking problem reduces to a stabilization problem for the error dynamics (17). To solve this problem, we next apply the inverse optimal control approach.

4.2 Tracking error stabilization

Once (17) is obtained, we proceed to study its stabilization. Note that $e = 0, \hat{W} = 0, \hat{W}_g = 0$ is an equilibrium point for the system without disturbances.
In order to perform the stability analysis for the system, the following Lyapunov function is formulated

$$\begin{gathered}\dot{V} = \frac{1}{2}\|e\|^2 + \frac{\Gamma^{-1}}{2} tr\left\{\tilde{W}^T \tilde{W}\right\} + \frac{\Gamma_g^{-1}}{2} tr\left\{\tilde{W}_g^T \tilde{W}_g\right\} \\ \Gamma = diag\{\gamma_1, \ldots, \gamma_n\}, \quad \Gamma_g = diag\{\gamma_{g1}, \ldots, \gamma_{gn}\}\end{gathered} \tag{18}$$

Its time derivative, along the trajectories of (17), is

$$\begin{gathered}\dot{V} = (A + I)\|e\|^2 + e^T \hat{W} z(\chi) + e^T \hat{W}(z(x_p) - z(x_r)) + e) + e^T \tilde{W}_g u \\ + e^T \hat{W}_g u_2 + \Gamma^{-1} tr\left\{\dot{\tilde{W}}^T \tilde{W}\right\} + \Gamma_g^{-1} tr\left\{\dot{\tilde{W}}_g^T \tilde{W}_g\right\}\end{gathered} \tag{19}$$

Replacing the learning laws

$$\begin{gathered}tr\left\{\dot{\tilde{W}}^T \tilde{W}\right\} = -\Gamma e^T \tilde{W} z(x) \\ \dot{\hat{\omega}}_{ij} = -\gamma_i e z(x_j) \\ tr\left\{\dot{\tilde{W}}_g^T \tilde{W}_g\right\} = -\Gamma_g e^T \tilde{W}_g u \\ \dot{\hat{\omega}}_{gij} = -\gamma_{ig} e_i u_j\end{gathered} \tag{20}$$

in (19), we obtain

$$\dot{V} = -(\lambda - 1)\|e\|^2 + e^T \hat{W} \phi_z(e, x_r) + e^T \hat{W}_g u_2 \tag{21}$$

where

$$\phi_z(e, x_r) = z(x_p) - z(x_r) = z(e + x_r) - z(x_r) \tag{22}$$

Next, we consider the following inequality (Poznyak et al., 1999),

$$X^T Y + Y^T X \leq X^T \Lambda X + Y^T \Lambda^{-1} Y \tag{23}$$

which holds for all matrices $X, Y \in \Re^{n \times k}$ and $\Lambda \in \Re^{n \times n}$ with $\Lambda = \Lambda^T > 0$.

Applying (23) to $e^T \hat{W} \phi(e, x_r)$ with $\Lambda = I$, we obtain

$$\dot{V} = -(\lambda - 1)\|e\|^2 + \frac{1}{2} e^T e + \frac{1}{2}\left\|\hat{W}\right\|^2 \|\phi_z(e, x_r)\|^2 + e^T \hat{W}_g u_2 \tag{24}$$

where $\left\|\hat{W}\right\|$, is any matrix norm for $\hat{W}$.

Since $\phi_z(e, x_r)$ is Lipschitz with respect to e, then, there exists a positive constant L_ϕ such that

$$\phi_z(e,x_r)^{\mathrm{T}} \le L_\phi \|e\|$$

Hence (24) can be rewritten as

$$\dot{V} = -(\lambda-1)\|e\|^2 + \frac{1}{2}(1+L_\phi^2)\left\|\hat{W}\right\|^2)\|e\|^2 + e^T\hat{W}_g u_2 \tag{25}$$

To this end, we define the following control law

$$\begin{aligned} u_2 &= -(\hat{W}_g)^{-1}\mu(1+L_{\phi z}^2\left\|\hat{W}\right\|^2)e \\ \mu &= diag\{\mu_1,\mu_2,\ldots,\mu_n\}, \qquad \mu_i > \frac{1}{2}, \qquad i=1,\ldots,n \end{aligned} \tag{26}$$

which renders

$$\begin{aligned} \dot{V} &\le -(\lambda-1)\|e\|^2 - \left(1+L_\phi^2\left\|W^* - \tilde{W}\right\|^2\right)\sum_{i=1}^{n}\left(\mu_i - \tfrac{1}{2}\right)e_i^2 \\ &\le 0 \end{aligned} \tag{27}$$

We now apply the Barbalat's lemma (Khalil, 1996), (Khalil, 2002). Since $V>0 \quad \forall e_i, \tilde{W}, \tilde{W}_g \ne 0$ and $\dot{V}(t) \le 0$, V is bounded. Hence, $\|e\|$ is bounded on $[0,T]$, the maximal interval of existence of the solution for any given initial state. V is nonincreasing and bounded from below by zero, and converges as $t \to \infty$. Integrating both sides of (27) we obtain

$$\lim_{t\to\infty}\left(-\int_0^t \dot{V}(\tau)d\tau\right) \ge \lim_{t\to\infty}\int_0^t\left((\lambda-1)\|e\|^2 + \left(1+L_\phi^2\left\|W^*+\tilde{W}\right\|\right)\sum_{i=1}^{n}\left(\mu_i - \frac{1}{2}\right)e_i^2\right)d\tau$$

which exists and is finite. Then,

$$(\lambda-1)\|e\|^2 + \left(1+L_\phi^2\left\|W^*-\tilde{W}\right\|^2\right)\sum_{i=1}^{n}\left(\mu_i - \frac{1}{2}\right)e_i^2 \to 0 \text{ as } t\to\infty$$

which implies that $e \to 0$ as $t \to \infty$.
From the learning laws (20), we have

$$\dot{\hat{w}}_{ij} \to 0 \text{ as } t\to\infty$$

$$\dot{\hat{w}}_{g\,ij} \to 0 \text{ as } t\to\infty$$

Therefore,

$$\dot{\hat{W}}(t) \to 0 \text{ as } t\to\infty$$

$$\dot{\hat{W}}_g(t) \to 0 \text{ as } t\to\infty$$

then

$$\lim_{t\to\infty}\hat{W} \to \hat{W}_{\infty}, \qquad \lim_{t\to\infty}\tilde{W} \to \tilde{W}_{\infty}$$

$$\lim_{t\to\infty}\hat{W}_g \to \hat{W}_{g\infty}, \qquad \lim_{t\to\infty}\tilde{W}_g \to \tilde{W}_{g\infty}$$

where $\hat{W}_\infty, \tilde{W}_\infty, \hat{W}_{g\infty}, \tilde{W}_{g\infty}$ are constant values.

Taking into account that W^*, W_g^* are constant matrices, $\hat{W}(t)$ and $\hat{W}_g(t)$ are bounded when $t \to \infty$. Since χ and x_p are assumed to be bounded on $[0,T]$, this implies that $T = \infty$. This ensures asymptotic stability of the tracking error.

Then, the control law to apply to the nonlinear system is defined by

$$u = \alpha_r(x_r) + u_1 + u_2 \tag{28}$$

where $\alpha_r(x_r), u_1, u_2$ are defined in equations (11), (16), (26). This control law guarantees asymptotic stability of the error dynamics and therefore ensures the tracking of the reference signal.

4.3 Optimization with respect to a cost function

Once the control law (26) is obtained, we proceed to analyze its optimality with respect to a cost function which considers the tracking error and the magnitude of the applied input.

Next, we prove that the control law (26), minimizes the cost function given by (Sanchez et al., 2002)

$$J(\tilde{u}) = \lim_{t\to\infty}\left\{2\beta V + \int_0^t \left(l(e,\hat{W},\hat{W}_g) + u_2^T R(e,\hat{W},\hat{W}_g)u_2\right)dt\right\} \tag{29}$$

where the Lyapunov function solves the following Hamilton-Jacobi-Bellman family of partial derivative equations parametrized with $\beta > 0$

$$l(e,\hat{W},\hat{W}_g) + 2\beta L_f V - \beta^2 L_g V R(e,\hat{W},\hat{W}_g)^{-1} L_g V^T = 0 \tag{30}$$

Note that $2\beta V$ in (30) is bounded when $t \to \infty$, since by (25) and (26), is decreasing and bounded from below by $V(0)$. Therefore, $\lim_{t\to\infty} V(t)$ exists and is finite. V

In (Krstic & Deng, 1998), $l\left(e,\hat{W}\right)$ is required to be positive definite and radially unbounded with respect to e. Here, from (30) we have

$$l(e,\hat{W},\hat{W}_g) = -2\beta L_f V + \beta^2 L_g V R(e,\hat{W},\hat{W}_g)^{-1} L_g V^T \tag{31}$$

Substituting (26) into (31) and then applying (23) to the second term on the right side of $L_f V$, we have

$$l(e,\hat{W},\hat{W}_g) \geq (\lambda - 1)\|e\|^2 + \left(1 + L_\phi^2\left\|\hat{W}\right\|^2\right)\sum_{i=1}^{n}(\mu_i - 1)e_i^2 \tag{32}$$

Selecting $\lambda > 1$ and $\mu_i > 1$, we ensure that $l(e,\hat{W},\hat{W}_g)$ satisfies the condition of being positive definite and radially unbounded. Hence, (29) is a suitable cost function.

The integral term in (29) can be written as,

$$l(e,\hat{W}) + u_2^T R(e,\hat{W})u_2 = -2\beta(L_f V) + 2\beta^2(L_g V)\left[R(e,\hat{W})\right]^{-1}(L_g V)^T \tag{33}$$

The Lyapunov time derivative is defined as

$$\dot{V} = L_f V + L_g V_u \tag{34}$$

and substituting in , we obtain

$$\dot{V} = L_f V + L_g V\left[-\beta(R(e,\hat{W},\hat{W}_g)^{-1}\right](L_g V)^{\tau}$$

Then, multiplying V by -2β we have

$$-2\beta\dot{V} = -2\beta(L_f V) + 2\beta^2(L_g V)\left[R(e,\hat{W},\hat{W}_g)\right]^{-1}(L_g V)^{\mathrm{T}}$$

Hence,

$$l(e,\hat{W},\hat{W}_g) + u_2^T R(e,\hat{W},\hat{W}_g)u_2 = -2\beta\dot{V} \tag{35}$$

Replacing (35) in the cost function (29), we obtain

$$\begin{aligned} J(u_2) &= \lim_{t\to\infty} 2\beta V - 2\beta\int_0^t \dot{V}dt = \lim_{t\to\infty}\{2\beta V(t) - 2\beta V(t) + 2\beta V(0)\} \\ &= 2\beta V(0) \end{aligned} \tag{36}$$

The cost function optimal value is given by $J^* = 2\beta V(0)$. This is achieved by the control law (26).

Selecting $\lambda > 1$ and $\mu_i > 1$, we ensure that $l(e,\hat{W},\hat{W}_g)$ satisfies the condition of being positive definite and radially unbounded. Hence, (29) is a suitable cost function.

4.4 Simulation results for rollover active control

We now apply the developed approach on rollover active control for cornering situations, where the features of the road can be assumed to be known by means of a system such as GPS, in order to determine the steering input for the vehicle. A prediction model can be introduced in the control scheme in order to predict the rollover threat and to produce a warning signal. For the cases where the driver can not respond to warning signals, active rollover control is necessary in order to prevent rollover. We consider two control approaches. First we develop a speed control which would be activated before cornering using differential braking, which could be available for implementation purposes. For the second approach, we consider the case where the road could be slippery, thus the braking would not be the same on each wheel, so the braking process would produce undesirable roll, yawing and lateral acceleration response, which would reduce the rollover threshold (Hyun & Langari, 2003). The tractor roll motion is governed by its lateral acceleration, which is generated by longitudinal speed and vehicle steering angle. In order to have reference values for the desired yawing response, the roll threshold and lateral acceleration threshold, we consider the values given in (Hyun & Langari, 2003).

The approach is based on building a recurrent neural network identifier which models the longitudinal speed v_x and yaw rate, which considers two inputs: longitudinal force F_T and yawing moment T_z. The model is described by the following RHONN

$$\dot{\chi}_1 = -\lambda\chi_1 + W_{1*}z(\chi) + W_{g11}F_T \tag{37}$$

$$\dot{\chi}_2 = -\lambda\chi_2 + W_{2*}z(\chi) + W_{g22}T_z \tag{38}$$

or in matrix form

$$\dot{\chi}_1 = -\lambda\chi + \hat{W}z(\chi) + W_g u \tag{39}$$

where $\lambda > 0, W \in \Re^{2\times 12}, W_g = diag\left\{W_{g11}, W_{g22}\right\}$ and

$$\begin{aligned} z(\chi) = \big[&\tanh\chi_{k1}, \tanh\chi_{k2}, \tanh\chi_{k1}\tanh\chi_{k2}, \tanh^2\chi_{k1}, \tanh^2\chi_{k2}, \\ &\tanh^2\chi_{k1}\tanh^2\chi_{k2}, \tanh^3\chi_{k1}, \tanh^3\chi_{k2}, \\ &\tanh^3\chi_{k1}\tanh^3\chi_{k2}, \tanh^4\chi_{k1}, \tanh^4\chi_{k2}, \tanh^4\chi_{k1}\tanh^4\chi_{k2} \big] \end{aligned} \tag{40}$$

where $\chi_{k1} = k_1\chi_1$ and $\chi_{k2} = k_2\chi_2$.
As in (Hyun & Langari, 2003), the reference yaw response can be obtained as a function of the desired speed and the steer angle using the Ackermann angle (Gillespie, 1992)

$$\dot{\psi}_d = \delta\frac{\upsilon_x}{L_f + L_r} \tag{41}$$

where δ is the steer angle and L_f, L_r are the front and rear vehicle wheelbase segments.
We consider for the tractor semitrailer model, the heavy payload condition model given in (Hyun & Langari, 2003), with the rollover threshold values given in function of the roll angle and lateral acceleration as

$$a_{yt} = 2.6m/s^2$$
$$\phi_t = 2.87\deg$$

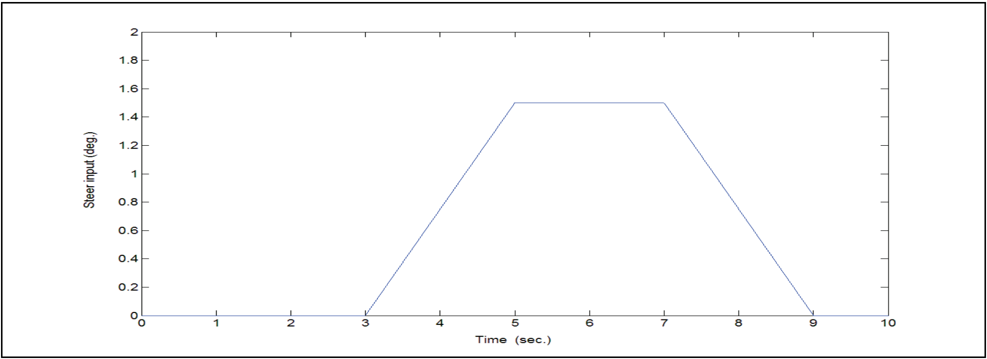


Fig. 4. Steer input for cornering maneuver

For the cornering situation given in Fig. 4, a speed reduction is desirable as given in Fig. 5. This speed reference is arbitrarily selected only for simulation purposes.
For the speed control, we consider the simplified RHONN given by (39), and we select

$$\lambda = 15, \quad \Gamma = diag\{10\}, \quad \Gamma_g = diag\{1 \times 10^{-3}\}$$
$$k_1 = 0.085, \quad k_2 = 70$$

For the control law (26), we choose

$$\mu = 0.5 \times 10^3$$

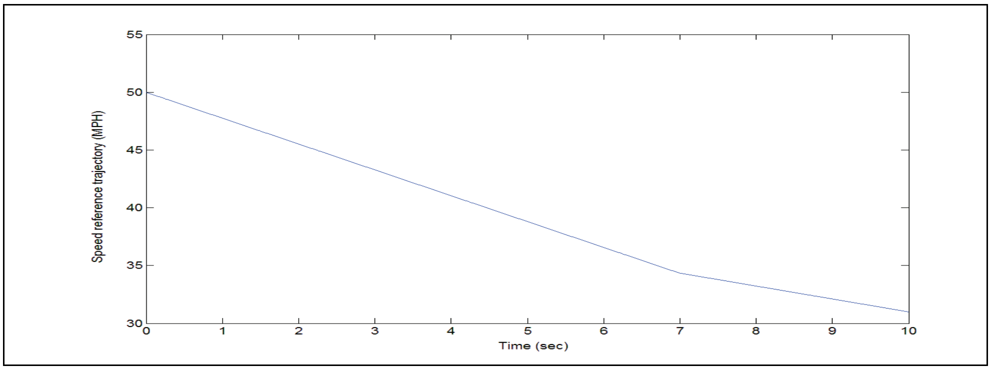


Fig. 5. Speed reference trajectory

The results for the speed-only control in Fig. 6 and Fig. 7 show that the speed is decreased successfully, but the yaw response deviates from the desired one, and the trailer presents high values of the roll angle. In order to reduce these effects we now apply a speed-yaw rate control.

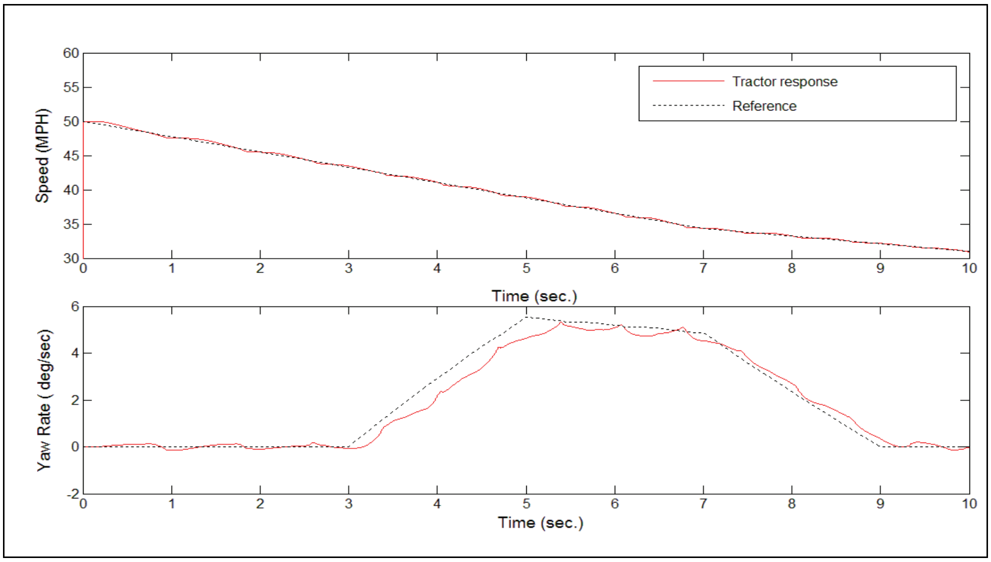


Fig. 6. Vehicle speed and yaw rate for speed only control

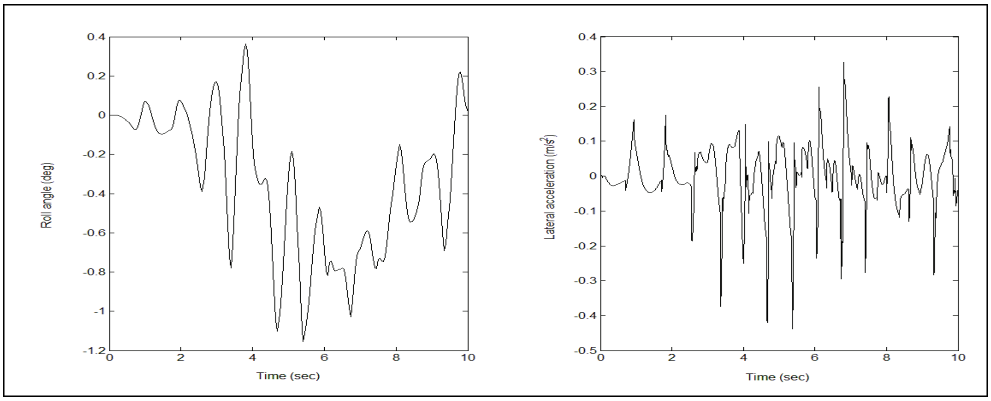


Fig. 7. Trailer roll angle and lateral acceleration for speed-only control

For the speed-yaw rate control, we consider the RHONN build from (37) and (38) the same cornering situation as in the previous application.
The RHONN parameters are selected as

$$\lambda = 15, \quad \Gamma = diag\left\{10, 2\times 10^{4}\right\}, \quad \Gamma_g = diag\left\{1\times 10^{-3}\right\}$$
$$k_1 = 0.085, \quad k_2 = 70$$

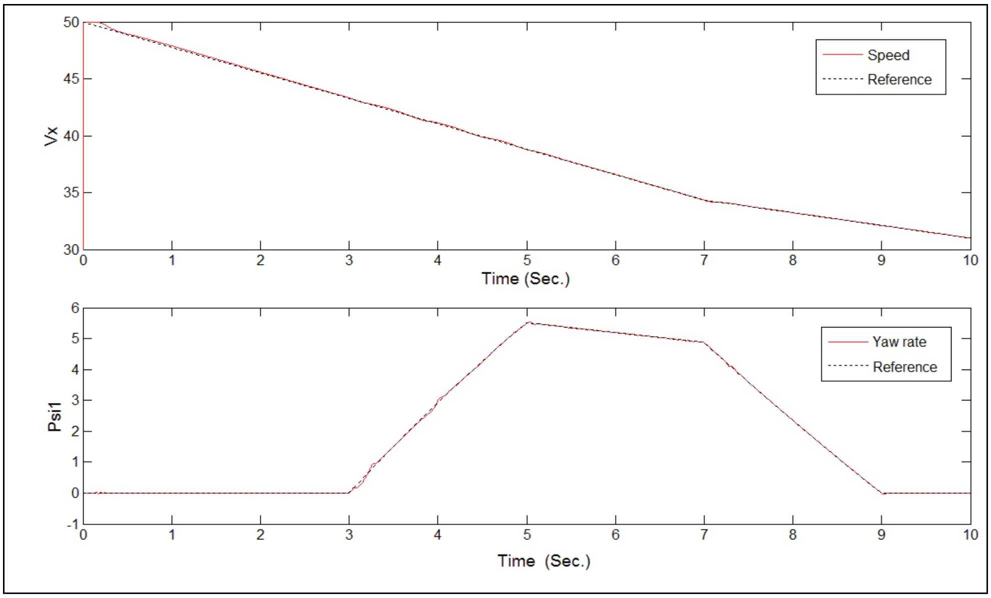


Fig. 8. Vehicle speed and yaw rate for speed-yaw rate control

For the control law, (26) we choose

$$\mu = diag\left\{0.5\times 10^{3}, 5\times 10^{4}\right\}$$

The results for trajectory tracking are shown in Fig. 8 to Fig. 10, where the tracking error is decreased considerably. The value for the roll angle decreased compared to the speed-only control simulation. The lateral acceleration presents an improved response. The speed-yaw rate control scheme prevents the rollover threat by forcing the values for roll and lateral acceleration to be far from the rollover threshold parameters.

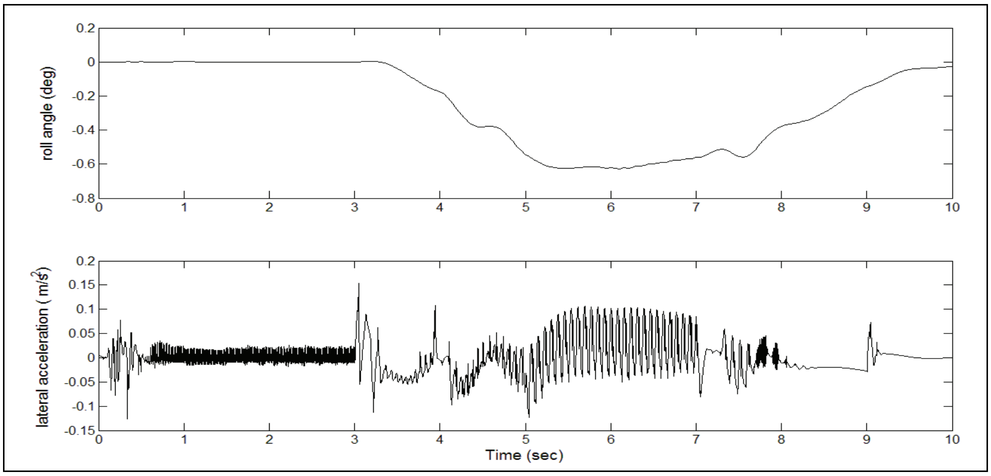


Fig. 9. Trailer roll angle and lateral acceleration for speed-yaw rate control

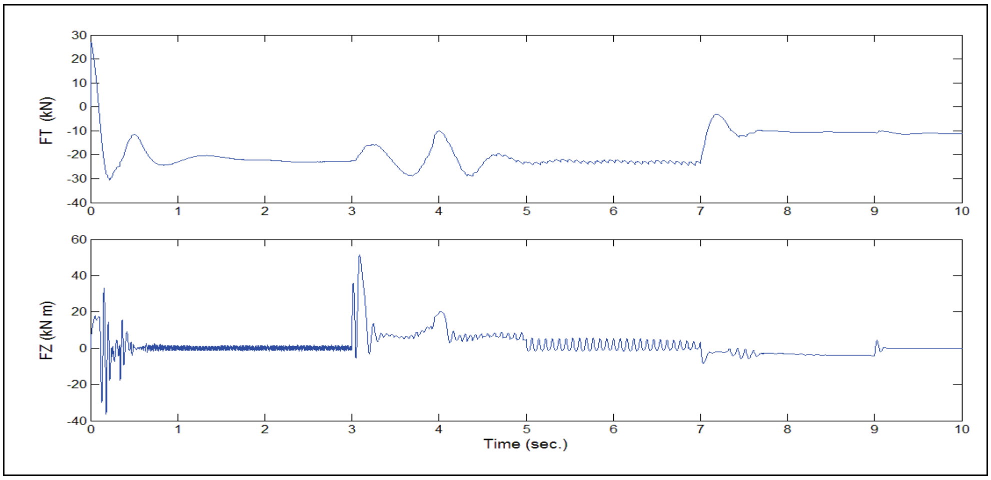


Fig. 10. Applied total braking torque and yawing moment for speed-yaw rate control

5. Conclusions

In this paper an adaptive recurrent neural network controller is developed in order to prevent rollover in heavy vehicles. The control scheme is composed of an Recurrent Neural

Network predictor which estimates the future behavior of the roll angle and lateral acceleration. A neural identifier builds an on-line model for the trailer-semitrailer model of 14 degrees of freedom which is assumed to be unknown. A learning adaptation law is derived using the Lyapunov methodology. Asymptotic stability of the tracking error is ensured by means of the inverse optimal control approach. The proposed scheme is tested, via simulations, to prevent rollover of a tractor-semitrailer. Two different control strategies are applied: speed-only control and speed-yaw rate control. The neural controller for speed and yaw rate presented the best performance by reducing the roll angle and lateral acceleration of the trailer.

6. References

UMTRI (1997). *ArcSim User Reference Manual,* The University of Michigan Transportation Research Institute, Ann Arbor, MI.

Basar, T. & Bernhard, P. (1995), *H-Infinity Optimal Control and Related Minimax Design Problems*, Birkhauser, Boston, USA.

Gillespie, T.D. (1992). *Fundamentals of Vehicle Dynamics*, Society of Automotive Engineers, Inc., Warrendale, PA.

Hill, D. J. & Moylan, P. (1996). "The Stability of nonlinear dissipative systems", *IEEE Trans. on Automatic Control*, Vol. 21, 708-711.

Hopfield, J. (1984). "Neurons with graded responses have collective computational properties like those of two state neurons", *Proc. Nat. Acad. Sci.,* USA, Vol. 81, pp. 3088-3092.

Hyun, D. & Langari, R. (2003). "Predictive Modelling for Rollover Warning of Heavy Vehicles," *Vehicle System Dynamics*, Vol. 39, No. 6, pp. 401-414.

Khalil, H. (1996), "Adaptive Output Feedback Control of Nonlinear Systems Represented by Input-Output Models", *IEEE Trans. on Automatic Control*, Vol. 41, No. 2, pp. 177-188.

Khalil, H. (2002). *Nonlinear Systems*, 3rd Ed., Prentice Hall, Upper Saddle River, NJ, USA.

Kosmatopoulos, E. B.; Christodoulou, M. A. & Ioannou, P. A. (1997). "Dynamical neural networks that ensure exponential identification error convergence", *Neural Networks*, Vol. 10, No. 2, pp. 299-314.

Krstic, M. & Deng, H. (1998). *Stabilization of Nonlinear Uncertain Systems*, Springer Verlag, New York, USA.

Narendra, K. S. & Parthasarathy, K. (1990). "Identification and control of dynamical systems using neural networks", *IEEE Trans. on Neural Networks*, Vol. 1, No. 1, pp. 4-27.

Poznyak, A. S.; Yu, W.; Sanchez, E. N. & Perez, J. P. (1999). "Nonlinear adaptive trajectory tracking using dynamic neural networks", *IEEE Trans. on Neural Networks*, Vol. 10, No. 6 Nov. 1999, pp. 1402-1411.

Poznyak, A. S.; Sanchez, E. N. & Yu, W. (2000). *Differential Neural Networks for Robust Nonlinear Control*, World Scientific, USA.

Rovitahkis, G. A. & Christodoulou, M. A. (2000), *Adaptive Control with Recurrent High-Order Neural Networks*, Springer Verlag, New York, USA.

Sanchez, E. N.; Perez, J. P. & Ricalde, L. (2002). "Recurrent neural control for robot trajectory tracking", *Proceedings of the 15th World Congress International Federation of Automatic Control*, Barcelona Spain, July, 2002.

Suykens, K.; Vandewalle, L. & De Moor, R. (1996). *Artificial Neural Networks for Modelling and Control of Nonlinear Systems*, Kluwer Academic Publishers, Boston, USA, 1996.

9

A New Supervised Learning Algorithm of Recurrent Neural Networks and L_2 Stability Analysis in Discrete-Time Domain

Wu Yilei, Yang Xulei and Song Qing
School of Electrical and Electronic Engineering
Nanyang Technological University,
Singapore

1. Introduction

In the past decades, Recurrent Neural Network (RNN) has attracted extensive research interests in various disciplines. One important motivation of these investigations is the RNN's promising ability of modeling time-behavior of nonlinear dynamic systems. It has been theoretically proved that RNN is able to map arbitrary input sequences to output sequences with infinite accuracy regardless underline dynamics with sufficient training samples [1]. Moreover, from biological point of view, RNN is more plausible to the real neural models as compared to other adaptive methods such as Hidden Markov Models (HMM), feed-forward networks and Support Vector Machines (SVM). From the practical point of view, the dynamics approximation and adaptive learning capability make RNN a highly competitive candidate for a wide range of applications. See [2] [3] [4] for examples.
Among the various applications, the realtime signal processing has constantly been one of the active topics of RNN. In such kind of applications, the convergence speed is always an important concern because of the tight timing requirement. For example, the conventional training algorithms of RNN, such as the Backpropagation Through Time (BPTT) and the Real Time Recurrent Learning (RTRL) always suffer from slow convergence speed. If a large learning rate is selected to speed up the weight updating, the training process may become unstable. Thus it is desirable to develop robust learning algorithms with variable or adaptive learning coe±cients to obtain a tradeoff between the stability and fast convergence speed.
The issue has already been extensively studied for linear adaptive filters, e.g., the famous Normalized Least Mean Square (N-LMS) algorithm. However, for online training algorithms of RNN this is still an open topic. Due to the inherent feedback and distributive parallel structure, the adjustments of RNN weights can affect the entire neural network state variables during network training. Hence it is difficult to obtain the error derivative for gradient type updating rules, and in turn difficulty in the analysis of the underlying dynamics of the training. So far, a great number of works have been carried out to solve the problem. To name a few, in [5], B. Pearlmutter presented a detail survey on gradient calculation for RNN training algorithms. In [6] [7] , M. Rupp et al introduced a robustness

analysis of RNN by the small gain theorem. The stability was explained from the energy point of view that the ratio of output noise against input noise was guaranteed to be smaller than unity. In [8], J. Liang and M. Gupta studied the stability of dynamic back-propagation training algorithm by the Lyapunov method. An auxiliary term was appended to augment the learning error. The convergence speed was improved by introducing an extra increment in the updating rule. Later, A. Atiya and A. Parlos used a generalized steepest descent method to obtain a unified error gradient algorithm [9]. Recently, Q. Song et al proposed a simultaneous perturbation stochastic approximation training method for neural networks and robust stability is established by the conic sector theorem [10] [11].

The work presented in this chapter investigate the stability and robustness of the gradient-type training algorithms of RNN in the discrete-time domain. A Robust Adaptive Gradient Descent (RAGD) training algorithm is introduced to improve the RNN training speed as compared to those conventional algorithms, such as the BPTT, the RTRL and the Normalized RTRL (N-RTRL). The main feature of the RAGD is the novel hybrid training concept, which switches the training patterns between the standard online Back Propagation (BP) and the N-RTRL algorithm via three adaptive parameters, the hybrid adaptive learning rates, the adaptive dead zone learning rates, and the normalization factors. These parameters allow RAGD to locate relatively deeper local attractors of the training and hence obtain a faster transient response. Different from the N-RTRL, the RAGD uses a specifically designed error derivatives based on the extended recurrent gradient to approximate the true gradient for realtime learning. Also the RAGD is different from the static BP in terms that the former uses the extended recurrent gradient to extend the instantaneous squared estimation error minimization into recurrent mode, while the latter is strictly based on the instantaneous squared estimation error minimization without specifically considering the recurrent signal.

Weight convergence and robust stability of the RAGD are proved respectively based on the Lyapunov function and the Cluett's law, which is developed from the conic sector theorem of input- output system theory. Sufficient boundary conditions of the three adaptive parameters are derived to guarantee the L_2 stability of the training. Different from precedent results [12], the present work employs the input-output systematic approach in analysis. This is because the input-output theory on basis of functional analysis requires minimal assumptions about the training statistics. Although the results are also derivable from conventional analysis method, we emphasize that input-output systematic scheme can provide an in-depth understanding of RNN training dynamics from different aspect.

In addition to the theoretical analysis, we carried out three case studies of the applications in realtime signal processing via computer simulations, including time series prediction, system identification, and attractor learning for pattern association. With these case studies, we are able to qualify the effectiveness of the RAGD and hence justify that the algorithm outperforms other counterparts.

The overall chapter is organized as follows: In Sections 2, we briefly introduce the structure of the RNN and the RAGD training algorithm. In Section 3, the robustness analysis of the RAGD is carried out for the Single-input Single-Output and Multi-input Multi-output RNN respectively. In addition, the conic sector theorem is introduced as the theoretical foundation of the analysis. Computer simulations are presented in Section 4 to show the efficiency of our proposed RAGD. Section 5 draws the final conclusions.

2. RAGD learning algorithm

Consider a RNN with l output nodes and m hidden neurons. In discrete-time domain, the network output $\hat{y}$ at time instant k can be written as

$$\hat{y}(k) = \hat{V}(k)\Phi(\hat{W}(k)\hat{x}(k)) \tag{1}$$

where $\hat{V}(k) \in R^{l\times m}$ and $\hat{W}(k) \in R^{m\times n}$ are output and hidden layer weights respectively (in matrix form), $\Phi(\cdot) \in R^{m\times 1}$ is a vector of nonlinear activation functions, and $\hat{x}(k) \in R^{n\times 1}$ is the state vector that consists of external input $u(k)$ and n - 1 delayed output feedback entries

$$\hat{x}(k) = [u(k), \hat{y}(k-1), \cdots, \hat{y}(k-n+1)]^T \tag{2}$$

in which T denotes transpose operation. To simplify the expression, we use notation $\Phi(k)$ instead of $\Phi(\hat{W}(k)\hat{x}(k))$ hereafter. When estimating a command signal $d(k)$, the instantaneous modeling error of RNN can be defined by

$$e(k) = d(k) - \hat{y}(k) + \varepsilon(k) \tag{3}$$

Note a disturbance term $\varepsilon(k) \in R^{l\times 1}$ is taken into account in (3). Without loss of generality, there is no assumption on the prior knowledge of $\varepsilon(k)$ and its statistics. The training objective of RNN is to update the weight parameters step by step to minimize certain cost function $f(e(k))$, with the most convenient form being the squared instantaneous error $e^2(k)/2$. Specifically, in an environment of time-varying signal statistics, a gradient based sequential training algorithm can be used to recursively reduce the $f(e(k))$ by estimating the weights at each time instant

$$\begin{cases} \hat{V}(k+1) = \hat{V}(k) - \alpha\frac{\partial f(e(k))}{\partial \hat{V}(k)} \\ \hat{W}(k+1) = \hat{W}_i(k) - \alpha\frac{\partial f(e(k))}{\partial \hat{W}_i(k)} \end{cases} \tag{4}$$

where α is the learning rate of RNN, and $\hat{W}_i(k)$ is the ith row of hidden layer weight matrix, with i = 1, 2, …, m. Note subscript i denotes ith row for matrices or ith entry for vectors. As for the above algorithm, a widely recognized problem is the slow convergence speed because of small learning rates for purpose of preserving weight convergence. So far the commonly accepted solution of this problem is to employ normalization, e.g., the N-RTRL algorithm [13] [1]. Indeed, the solution can be further improved if we can find effective boundary conditions of learning rates and normalization factors as will be shown in later sections. Moreover, hybrid learning rates can be employed to obtain the tradeoff between the transient and steady state response. Now based on the RNN model (1) and the gradient-based training equation (4), we propose the RAGD learning algorithm as follows

$$\begin{cases} \hat{V}(k+1) = \hat{V}(k) + \frac{\alpha^v(k)}{\rho^v(k)}e(k)(\Phi(k)^T + \beta^v(k)\hat{A}(k)) \\ \hat{W}(k+1) = \hat{W}(k) + \frac{\alpha^w(k)}{\rho^w(k)}diag\{\Phi'(k)\}\hat{V}(k)^T e(k)(\hat{x}(k)^T + \beta^w(k)\hat{B}(k)) \end{cases} \tag{5}$$

where $\Phi'(k)$ is the vector of activation function derivatives, $\alpha^v(k)$, $\alpha^w(k)$ are adaptive dead zone learning rates, $\beta^v(k)$, $\beta^w(k)$ are hybrid learning rates, $\rho^v(k)$, $\rho^w(k)$ are normalization factors, and $\hat{A}(k)$, $\hat{B}(k)$ are residual error gradients. These variables are defined in the following.

(a) $\Phi'(k) \in R^{m\times 1}$

$$\Phi'(k) = [\ \phi'(\hat{W}_1(k)\hat{x}(k)) \quad \phi'(\hat{W}_2(k)\hat{x}(k)) \quad \cdots \quad \phi'(\hat{W}_m(k)\hat{x}(k))\ ]^T \tag{6}$$

(b) $\hat{A}(k) \in R^{1\times m}$ and $\hat{B}(k) \in R^{1\times n}$

$$\hat{A}(k) \quad = \quad \underline{\hat{V}}(k) \cdot [diag\{\Phi'(k)\}]_l \cdot \left[\hat{W}(k)\right]_l \cdot \hat{D}^v(k) \tag{7}$$

$$\hat{B}(k) \quad = \quad \underline{\hat{W}}(k)\hat{D}^w(k) \tag{8}$$

where $[diag\{\Phi'(k)\}]_l \in R^{(l\times m)\times(l\times m)}$ and $\left[\hat{W}(k)\right]_l \in R^{(l\times m)\times(l\times n)}$ are block diagonal matrices with sub-matrix $diag\{\Phi'(k)\}$ and $\hat{W}(k)$ on the diagonal respectively

$$[diag\{\Phi'(k)\}]_l = \begin{bmatrix} diag\{\Phi'(k)\} & & & 0 \\ & diag\{\Phi'(k)\} & & \\ & & \ddots & \\ 0 & & & diag\{\Phi'(k)\} \end{bmatrix}$$

$$\left[\hat{W}(k)\right]_l = \begin{bmatrix} \hat{W}(k) & & & 0 \\ & \hat{W}(k) & & \\ & & \ddots & \\ 0 & & & \hat{W}(k) \end{bmatrix}$$

$\underline{\hat{V}}(k) \in R^{1\times(l\times m)}$ and $\underline{\hat{W}}(k) \in R^{1\times(m\times n)}$ are long vector versions of the weight matrices $\hat{V}(k)$ and $\hat{W}(k)$ respectively

$$\begin{cases} \underline{\hat{V}}(k) = \left[\hat{V}_1(k) \quad \hat{V}_2(k) \quad \cdots \quad \hat{V}_l(k)\right] \\ \underline{\hat{W}}(k) = \left[\hat{W}_1(k) \quad \hat{W}_2(k) \quad \cdots \quad \hat{W}_m(k)\right] \end{cases}$$

and the Jacobian $\hat{D}^v(k) \in R^{(l\times n)\times m}$ and $\hat{D}^w(k) \in R^{(m\times n)\times n}$

$$\begin{cases} \hat{D}^v(k) = [\hat{D}_1^v(k)^T \quad \hat{D}_2^v(k)^T \quad \cdots \quad \hat{D}_l^v(k)^T]^T \\ \hat{D}^w(k) = [\hat{D}_1^w(k)^T \quad \hat{D}_2^w(k)^T \quad \cdots \quad \hat{D}_m^w(k)^T]^T \end{cases}$$

in which $\hat{D}_i^v(k) = \frac{\partial \hat{x}(k)}{\partial \hat{V}_i(k)} \in R^{n\times m}$, $\hat{D}_i^w(k) = \frac{\partial \hat{x}(k)}{\partial \hat{W}_i(k)} \in R^{n\times n}$ are sub-matrices.

(c) $\beta^v(k)$ and $\beta^w(k)$

$$\beta^v(k) = sgn\{\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T\} \tag{9}$$

$$\beta^w(k) = sgn\{\hat{x}(k)^T(\delta I + \hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T\} \tag{10}$$

where δ is a small positive constant, *I* is the identity matrix, and δ*I* is employed to ensure the matrix $\delta I + \Phi(k)\Phi(k)^T$ and $\delta I + \hat{x}(k)\hat{x}(k)^T$ positive definite.

(d) $\rho^v(k)$ and $\rho^w(k)$

$$\rho^v(k) = \nu\rho^v(k-1) + \max\{\bar{\rho}^v, \|\Phi(k)^T + \beta^v(k)\hat{A}(k)\|^2\} \tag{11}$$

$$\rho^w(k) = \nu\rho^w(k-1) + \max\{\bar{\rho}^w, \frac{\mu_{max}\|diag\{\Phi'(k)\}\hat{V}(k)^T\|_F^2 \cdot \|\hat{x}(k)^T + \beta^w(k)\hat{B}(k)\|^2}{\phi'_{min}(k)}\} \tag{12}$$

where $\nu < 1$, $\bar{\rho}^v$ and $\bar{\rho}^w < 1$ are positive constants, μ_{max} is the maximu value of the activation function, and $\phi'_{min}(k) = \min\{\Phi'_1(k), \cdots, \Phi'_m(k)\}$. Note we are using an inner product induced norm, the Frobenius norm, as the norm of weight matrices in this work.

(e) $\alpha^v(k)$ and $\alpha^w(k)$

$$\alpha^v(k) = sgn\{\|e(k)\| - \varepsilon_{max}^v / \sqrt{1 - \frac{\|\Phi(k)^T + \beta^v(k)\hat{A}(k)\|^2}{\rho^v(k)}}\} \tag{13}$$

$$\alpha^w(k) = sgn\{\|e(k)\| - \varepsilon_{max}^w / \sqrt{1 - \frac{\mu_{max}\|diag\{\Phi'(k)\}\hat{V}(k)^T\|_F^2 \cdot \|\hat{x}(k)^T + \beta^w(k)\hat{B}(k)\|^2}{\phi'_{min}(k)\cdot\rho^w(k)}}\} \tag{14}$$

where $\varepsilon_{max}^v = \max\{\|\tilde{\varepsilon}^v(k)\|\}$, $\varepsilon_{max}^w = \max\{\|\tilde{\varepsilon}^w(k)\|\}$, and *sgn*(•) function is defined by

$$sgn(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{15}$$

Remark 1 The RAGD algorithm uses the specific designed derivative as shown in (5). The state estimators are taken into account in the second terms of the partial derivatives on the right side of the equation. Further, to make the proposed algorithm realtime adaptive and recurrent, the $\hat{D}^v(k)$ *and the* $\hat{D}^w(k)$ *in the partial derivatives are calculated on basis of the data from previous training steps, which is similar to that of the N-RTRL algorithm [14]. It is noteworthy only when the convergence and stability requirements (details will be given in Section 3) are met, they hybrid learning rate β will be turned on. In this case, since we have estimated the best available gradient at each step k, the combination of weights and state estimates in (5) should provide a relatively deeper local attractor of the nonlinear iteration, and hence to speed up the training.*

3. Robust stability analysis

In this section, we present detail analysis of robust stability of the RAGD algorithm. Proofs of weight convergence and L_2 stability are derived on basis of Lyapunov function and input-output systematic approach respectively. The boundary conditions on the three adaptive parameters, the hybrid learning rate, the adaptive dead zone learning rates, and the normalization factors, are obtained for the optimized transient response of the training. For better understanding of the algorithm, a simple case of Single-input Single-output (SISO) RNN is firstly given as an example. Then the results are extended to the more complicated case of Multi-input Multi-output (MIMO) RNN. Before proceeding, we introduce the Cluett's law and mathematical preliminaries.

3.1 Cluett's laws

The main concern of this work is discrete signals which are infinite sequences of real numbers. Each signal may be considered an element of a set known as a linear vector space. To provide a clear explanation, an immediate review is given on several mathematical notations. Let the $x(k) \in R^{n^{\times 1}}$ denotes the series $\{x(1), x(2), \ldots\}$, then

i) The L_2 norm of $x(k)$ is defined as $\|x(k)\|_2 = \sqrt{\sum_{k=1}^{\infty} \|x(k)\|^2}$

ii) If the L_2 norm of $x(k)$ exists, the corresponding normed vector spaces are called L_2 spaces;

iii) The truncation of $x(k)$ is defined as $\|x(k)\|_{2,N} = \sqrt{\sum_{k=1}^{N} \|x(k)\|^2}$

iv) The extension of a space L_2, denoted by L_{2e} is the space consisting of those elements $x(k)$ whose truncations are all lie in L_2, i.e., $\|x(k)\|_{2,N} < \infty$, for all $N \in Z_+$ (the set of positive integers).

Note $\|\bullet\|$ denotes the Euclidean norm of a vector, and $\|\bullet\|_2$ for the L_2 norm of a signal (could be either a vector or a scalar). Let's consider the closed loop system shown in Figure 1

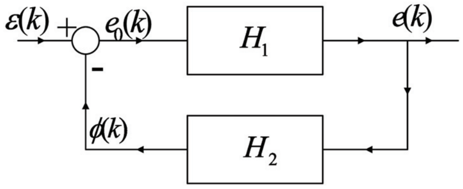


Figure 1. A general closed loop feedback system

$$\begin{cases} e_0(k) = \varepsilon(k) - \phi(k) \\ e(k) = H_1 e_0(k) \\ \phi(k) = H_2 e(k) \end{cases} \tag{16}$$

where operators $H_1;H_2 : L_{2e} \rightarrow L_{2e}$, discrete time signals $e_0(k); e(k); \phi(k) \in L_{2e}$ and $\varepsilon(k) \in L_2$.

Theorem 1 *(Cluett's Law-1) If the following two conditions hold* *i)* $H_1 : e_0(k) \to e(k)$ *satisfies* $\sum_{k=0}^{N} [e^2(k) + \alpha e_0(k)e(k) + \beta e_0^2(k)] \geq -\gamma,\ \forall N \in Z^+$

ii) $H_2 : e(k) \to \phi(k)$ *satisfies* $\sum_{k=0}^{N} [\beta\phi^2(k) - \alpha\phi(k)e(k) + e^2(k)] \leq -\eta\{\|(\phi(k), e(k))\|_2^2\}_N,\ \forall N \in Z^+$

for some $\alpha,\ \beta \in R$, which are independent of k and N, and $\gamma \geq 0$, $\eta > 0$, which are independent of N, then the closed loop feedback system of (16) is stable in the sense of $e(k)$, $\phi(k) \in L_2$.

Proof: By the inequality i) and using $e_0(k) = \varepsilon(k) - \phi(k)$

$$\sum_{k=0}^{N}[\beta\phi^2(k) - \alpha\phi(k)e(k) + e^2(k)] + \sum_{k=0}^{N}[\alpha\varepsilon(k)e(k) - 2\beta\varepsilon(k)\phi(k) + \beta\varepsilon^2(k)] \geq -\gamma \quad (17)$$

Combining inequality ii) and equation (17)

$$-\eta\{\|(\phi(k), e(k))\|_2^2\}_N + \sum_{k=0}^{N}[\alpha\varepsilon(k)e(k) - 2\beta\varepsilon(k)\phi(k) + \beta\varepsilon^2(k)] \geq -\gamma \quad (18)$$

Using the Schwartz inequality

$$\eta\{\|(\phi(k), e(k))\|_2^2\}_N - |\alpha| \cdot \{\|\varepsilon(k)\|_2\}_N \cdot \{\|e(k)\|_2\}_N - 2|\beta| \cdot \{\|\varepsilon(k)\|_2\}_N \cdot \{\|\phi(k)\|_2\}_N \leq \gamma + |\beta| \cdot \{\|\varepsilon(k)\|_2^2\}_N \quad (19)$$

Assume $\{\|(\phi(k), e(k))\|_2^2\}_N \to \infty$as $N \to \infty$, then from equation (19) we derive $\eta \leq 0$. This is a contradiction. Therefore $\{\|(\phi(k), e(k))\|_2^2\}_N$is bounded for all $N \in Z_+$, i.e., $\phi(k)$, $e(k) \in L_2$. ■

Theorem 2 *(Cluett's Law{2) For the feedback system (16), if*

i) H_1 : $e_0(k)$ - $e(k)$ satisfies

$$\sum_{k=1}^{N} \left(e_0(k)e(k) + \sigma e_0(k)^2/2\right) \geq -\gamma$$

ii) H_2 : $e(k)$ - ϕ (k) satisfies

$$\sum_{k=1}^{N} \left(\sigma\phi(k)^2/2 - \phi(k)e(k)\right) \leq -\eta\|(\phi(k), e(k))\|_{2,N}^2$$

for some $\gamma \geq 0$, $\eta > 0$, which are independent of N, and $\sigma \in (0, 1]$, which is independent of k and N, then the closed loop signals $e(k)$, ϕ (k) $\in L_2$.

Proof: See corollary 2.1 in [15]. ■

Remark 2 *As a matter of fact, the operator H_1 represents the nonlinear mapping and H_2 is a dynamic linear transfer function. When condition (i) and (ii) are satisfied, H_2 is guaranteed to be passive and H_1^{-1} is strictly interior conic (c_1, r_1), where c_1 = 1 and r_1 = (1- σ)$^{1/2}$, or equivalently H_1 is strictly interior the conic (c_2, r_2) where c_2 = σ $^{-1}$ and r_1 = σ $^{-1}$ (1 - σ)$^{1/2}$ as long as σ < 1 holds. Hence the feedback loop is L_2-stable by the conic sector theorem. This conic relation is illustrated in Figure 2*

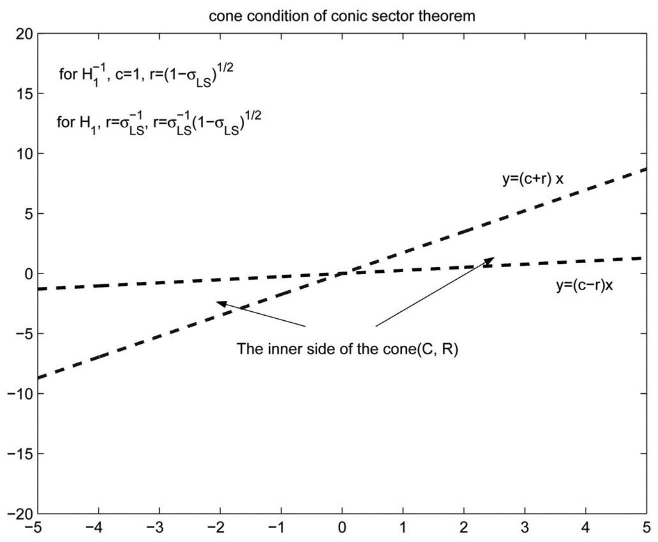


Figure 2. Illustration of interior and exterior conic relations of H_1

3.2 Output layer analysis of SISO RNN

In this and next section, we consider the RNN model of (1) with only one output node, i.e., l = 1. Such simplification is favorable for us to put more concentration on the basic ideas of the proof rather than the pure mathematics. Moreover, the results for SISO RNN will also be extended to the more general case of MIMO RNN in later sections. On the other hand, in a multi-layered RNN, it may not be able to update all the estimated weights within a single gradient approximation function. Hence we shall partition the training into different layers. Now with the assumption of SISO RNN, the training for output layer can be re-written as

$$\hat{V}(k+1) = \hat{V}(k) + \frac{\alpha^v(k)}{\rho^v(k)} e(k) \left(\Phi(k)^T + \beta^v(k) \hat{V}(k) diag\{\Phi'(k)\} \hat{W}(k) \hat{D}^v(k) \right) \quad (20)$$

In order to analyze the dynamics of this training equation via input-output approach, the first step is to restructure (20) into an error feedback loop, which should be the same as that in Figure 1. Further, the weight estimation error must be referred as the output signal. For this purpose, define the estimation error

$$e^v(k) = \hat{V}(k)\Phi(k) - V^*\Phi(k) = \tilde{V}(k)\Phi(k) \quad (21)$$

where $V^* \in R^{1\times m}$ and $\tilde{V}$ (k) = V (k) - V^* are the ideal weight vector and estimation error vector of output layer respectively, and $\Phi^*(k)$ is defined in analogous to Φ (k) as

$$\Phi^*(k) = [\phi(W_1^* x^*(k)) \quad \phi(W_2^* x^*(k)) \quad \cdots \quad \phi(W_m^* x^*(k))]^T \quad (22)$$

where $x^*(k) \in R^{n\times 1}$ is the ideal input state, $W^* \in R^{m\times n}$ is the ideal weight matrix of hidden layer of the RNN. Then the training error of RNN can be expanded as

$$\begin{aligned} e(k) &= d(k) - \hat{y}(k) + \varepsilon(k) \\ &= V^*\Phi^*(k) - \hat{V}(k)\Phi(k) + \varepsilon(k) \\ &= [V^*\Phi^*(k) - V^*\Phi(k)] - [\hat{V}(k)\Phi(k) - V^*\Phi(k)] + \varepsilon(k) \end{aligned} \tag{23}$$

Because the term $V^* \Phi^*(k)$ - $V^* \Phi (k)$ is temporarily constant in case of output layer training, we can define $\tilde{\varepsilon}^v(k) = \varepsilon(k) + V^* \Phi^*(k) - V^* \Phi(k)$. Then (23) can be transformed as

$$\tilde{\varepsilon}^v(k) - e^v(k) = e(k) \tag{24}$$

Equation (24) has a similar form as the feedback path of the system (16), with $e^v(k)$ and $e(k)$ corresponding to $e(k)$ and $e_0(k)$ in Figure 1 respectively, and here the feedback gain is unity, i.e., $H_2 = 1$.

There is an important implication in the relation of (24). The $e^v(k)$, $e(k)$ and $\tilde{\varepsilon}^v(k)$ correspond to the weight estimation error, the RNN modeling error and the disturbance, respectively. Hence the training error is directly linked to the disturbance, and in turn, the parameter estimating error of the RNN output layer. If we further establish a nonlinear mapping from the original disturbance $\tilde{\varepsilon}^v(k)$ to the parameter estimation error $e^v(k)$, the relationship between L_2-stability of training algorithm and learning parameters can subsequently be studied by imposing the conditions of Theorem 2.

Theorem 3 *If the output layer of the RNN is trained by the adaptive normalized gradient algorithm (20), the weight $\hat{V}(k)$ is guaranteed to be stable in the sense of Lyapunov*

$$\|\tilde{V}(k+1)\|^2 - \|\tilde{V}(k)\|^2 \le 0, \quad \forall k \tag{25}$$

with $\tilde{V}(k) = V(k) - V^*$. Also the training will be L_2-stable in the sense of $e^v(k) \in L_2$ if $\alpha^v(k) \neq 0$ for all $k \in Z_+$.

Proof: Subtracting $V^*$and then squaring both sides of (20)

$$\begin{aligned} &\|\tilde{V}(k+1)\|^2 - \|\tilde{V}(k)\|^2 \\ =\ & \frac{2\alpha^v(k)e(k)}{\rho^v(k)} \cdot \tilde{V}(k)(\Phi(k)^T + \beta^v(k)\hat{A}(k))^T + (\frac{\alpha^v(k)e(k)}{\rho^v(k)})^2\|\Phi(k)^T + \beta^v(k)\hat{A}(k)\|^2 \\ =\ & \frac{2\alpha^v(k)e(k)}{\rho^v(k)} \cdot \tilde{V}(k)(\Phi(k) + \beta^v(k)\hat{A}(k)^T) + (\frac{\alpha^v(k)e(k)}{\rho^v(k)})^2\|\Phi(k)^T + \beta^v(k)\hat{A}(k)\|^2 \end{aligned} \tag{26}$$

Regarding the first term on the right side of (26), we find that it may be easily associated with the term $e^v(k)$ due to the explicit appearance of $\tilde{V}(k)$ and $\Phi(k)$. Following this idea, we need to apply certain transformation to $\beta^v(k)\hat{A}(k)^T$, such that $\Phi(k)$ can be extracted from the summation. When it comes to this point, our first thought is to left multiply $\beta^v(k)\hat{A}(k)^T$ by $\Phi(k)\Phi(k)^T(\Phi(k)\Phi(k)^T)^{-1}$. However, the transformation is not valid because $\Phi(k)\Phi(k)^T$ is not an invertible matrix ($\Phi(k)$ is a column vector). Fortunately, inspired by the approximation method of classical Gauss-Newton iteration algorithm [2] (pp.126-127), we can add the term $\Phi(k)\Phi(k)^T$ by a small positive constant δ to expand it into

$$\delta I + \Phi(k)\Phi(k)^T : \text{ positive definite for all } k \tag{27}$$

Such that the singular matrix problem can be avoided. On this basis, we have the following derivations

$$\begin{aligned}
& \|\tilde{V}(k+1)\|^2 - \|\tilde{V}(k)\|^2 \\
= \; & \frac{2\alpha^v(k)e(k)}{\rho^v(k)} \cdot \tilde{V}(k)\Phi(k)(1+\beta^v(k)\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T) \\
& +(\frac{\alpha^v(k)e(k)}{\rho^v(k)})^2\|\Phi(k)^T + \beta^v(k)\hat{A}(k)\|^2 \\
= \; & \frac{2\alpha^v(k)e(k)}{\rho^v(k)} e^v(k)(1+\beta^v(k)\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T) \\
& +(\frac{\alpha^v(k)e(k)}{\rho^v(k)})^2\|\Phi(k)^T + \beta^v(k)\hat{A}(k)\|^2
\end{aligned} \tag{28}$$

$$\begin{aligned}
= \; & \frac{2\alpha^v(k)(\tilde{\varepsilon}^v(k)e(k) - e^2(k))}{\rho^v(k)}(1+\beta^v(k)\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T) \\
& +(\frac{\alpha^v(k)e(k)}{\rho^v(k)})^2\|\Phi(k)^T + \beta^v(k)\hat{A}(k)\|^2
\end{aligned} \tag{29}$$

where (29) is obtained by substituting (24) into (28). Then based on the triangular inequality $2\tilde{\varepsilon}^v(k)e(k) \leq (\tilde{\varepsilon}^v(k))^2 + e^2(k)$, (29) can be further deducted as

$$\begin{aligned}
& \|\tilde{V}(k+1)\|^2 - \|\tilde{V}(k)\|^2 \\
\leq \; & \frac{\alpha^v(k)((\tilde{\varepsilon}^v(k))^2 - e^2(k))}{\rho^v(k)}(1+\beta^v(k)\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T) \\
& +(\frac{\alpha^v(k)}{\rho^v(k)})^2 e^2(k)\|\Phi(k)^T + \beta^v(k)\hat{A}(k)\|^2 \\
= \; & \frac{\alpha^v(k)}{\rho^v(k)}(1+\beta^v(k)\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T)\big((\tilde{\varepsilon}^v(k))^2 \\
& -(1 - \frac{\alpha^v(k)\|\Phi(k)^T + \beta^v(k)\hat{A}(k)\|^2}{\rho^v(k)(1+\beta^v(k)\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T)})\, e^2(k)\big)
\end{aligned}$$

By the definition of $\beta^v(k)$, we may derive that $\beta^v(k)\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T \geq 0$. Furthermore, because that $\rho^v(k) \geq \|\Phi(k)^T + \beta^v(k)\hat{A}(k)\|^2$ as defined in (11) which lead to 1 - $\frac{\|\Phi(k)^T+\beta^v(k)\hat{A}(k)\|^2}{\rho^v(k)}$ > 0, and by the definition of $\alpha^v(k)$, the convergence of $\tilde{V}$ (k) can be derived

$$\begin{aligned}
&\|\tilde{V}(k+1)\|^2 - \|\tilde{V}(k)\|^2 \\
\leq\ & \frac{\alpha^v(k)}{\rho^v(k)}(1+\beta^v(k)\Phi(k)^T(\delta I+\Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T)\left((\tilde{\varepsilon}^v(k))^2\right. \\
& \left.-(1-\frac{\alpha^v(k)\|\Phi(k)^T+\beta^v(k)\hat{A}(k)\|^2}{\rho^v(k)})\ e^2(k)\right) \\
\leq\ & \frac{\alpha^v(k)}{\rho^v(k)}(1+\beta^v(k)\Phi(k)^T(\delta I+\Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T)\left((\varepsilon^v_{max})^2\right. \\
& \left.-(1-\frac{\alpha^v(k)\|\Phi(k)^T+\beta^v(k)\hat{A}(k)\|^2}{\rho^v(k)})\ e^2(k)\right) \\
\leq\ & 0
\end{aligned} \tag{30}$$

Next considering the case that the assumption $\alpha^v(k) \neq 0$ holds for all $k \in Z_+$, we can divide both sides of (28) by $2\alpha^v(k)(1+\beta^v(k)\Phi(k)^T(\delta I+\Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T)$ and then sum up to N steps

$$\begin{aligned}
-\Delta V &= \sum_{k=1}^{N}\left(\frac{e(k)e^v(k)}{\rho^v(k)} + \frac{\alpha^v(k)\|\Phi(k)^T+\beta^v(k)\hat{A}(k)\|^2}{2(\rho^v(k))^2(1+\beta^v(k)\Phi(k)^T(\delta I+\Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T)}e^2(k)\right) \\
&\leq \sum_{k=1}^{N}(\bar{e}(k)\bar{e}^v(k)+\frac{1}{2}\bar{\sigma}^v(\bar{e}(k))^2) \qquad \forall k \in \{k|\ \alpha^v(k)\neq 0\}
\end{aligned} \tag{31}$$

where the normalized error signals are defined as

$$\bar{e}(k) = \frac{e(k)}{\sqrt{\rho^v(k)}}, \quad \bar{e}^v(k) = \frac{e^v(k)}{\sqrt{\rho^v(k)}}$$

and the cone satisfies

$$\bar{\sigma}^v = \sup_k\{\frac{\|\Phi(k)^T+\beta^v(k)\hat{A}(k)\|^2}{\rho^v(k)}\} < 1$$

which prevents the vanishing radius problem, i.e., $\bar{\sigma}^v$ is strictly smaller than one [15]. Because for each k the Lyapunov function (30) is guaranteed smaller or equal to zero, we have

$$\begin{aligned}
0\ \leq\ \Delta V &= -\sum_{k=1}^{N}\frac{\|\tilde{V}(k+1)\|^2-\|\tilde{V}(k)\|^2}{2(1+\beta^v(k)\Phi(k)^T(\delta I+\Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T)} \\
&\leq -\frac{1}{2}\sum_{k=1}^{N}(\|\tilde{V}(k+1)\|^2-\|\tilde{V}(k)\|^2) \\
&= \frac{1}{2}(\|\tilde{V}(1)\|^2-\|\tilde{V}(N+1)\|^2)
\end{aligned}$$

Due to the specific selection of the normalization factor in (11), the normalized error signals guarantee that the original signals $e(k)$ and $e^v(k)$ are bounded according to the original operators H_1^v and H_2 [15]. Now the operator H_1^v represented by (31) satisfies the condition (i) of Theorem 2, and condition (ii) is guaranteed to hold due to $H_2 = 1$. Thus we conclude that $e^v(k) \in L_2$. ■

Remark 3 *According to the theoretical analysis, the three adaptive parameters $\alpha^v(k)$, $\beta^v(k)$ and $\rho^v(k)$ play important roles in the design of the RAGD. The adaptive learning rate $\alpha^v(k)$ is based on the standard adaptive control system to solve the weight drift problem [10]. The normalization factor $\rho^v(k)$ prevents the so-called vanishing cone problem of the conic sector theorem [15], which also has a similar role to the local stability condition as in [8] to bound the gradient in (20). The specific designed hybrid adaptive learning rate $\beta^v(k)$ can be further interpreted as activating the recurrent learning fashion in case $\Phi(k)^T\{\delta I + \Phi(k)\Phi(k)^T\}^{-1}\hat{A}(k)^T \geq 0$. It implies that the recurrent training of the RAGD will be active only if the second term of the derivative in (20) gives the negative gradient direction, i.e., a relatively deeper local attractor, otherwise the RAGD training procedure will be the same as a static BP algorithm and likely escape this undesired local attractor since it is unfavorable in the recurrent training. This design is especially effective for accelerating the training of the RNN when the iteration is near the bottom of basin of a local attractor, where the derivatives are changed slowly. With $\beta^v(k) = 1$, the approximation of $\hat{D}^v(k)$ is more accurate to meet the convergence and stability requirements.*

Remark 4 *The idea of the RAGD is similar to the existing works [16] [17] [14]. If we calculate the derivative in (20) exactly by unfolding the recurrent structure and force $\beta^v(k) = 0$, i.e, pursuing all N steps back in the past, then the algorithm will recover the static BP [17] [18]. Moreover, based on the assumption that the model parameters do not change apparently between each iteration [16], then we can derive a similar approach as the N-RTRL [14]. However, the key difference between the RAGD and the N-RTRL is that we use the hybrid learning rate $\beta^v(k)$ to guarantee the weight convergence and system stability.*

3.3 Hidden layer analysis of SISO RNN

This section presents the stability analysis for the hidden layer training of the RAGD. Apparently the analysis for the hidden layer is more di±cult than the one of the output layer, because the dynamics between the weight and modeling error is nonlinear. The derivation of error gradient must be carried out through one layer backward, which involves the derivative of activation function. In the following analysis, we show that the nonlinearity can actually be avoided by using the mean value theorem. On the other hand, as mentioned in section 2, the Frobenius norm is employed as weight matrix norm in the proof, e.g., $\left\|\hat{W}(k)\right\|_F$. A direct benefit of this expression is that the proof and the training equation can be presented in matrix forms, while not in a manner of row by row. However question arises, it is difficult to derive the Jacobian in this framework. We find that it is feasible to extend the Jacobian into a long vector form on the row basis. Next, similar to the output layer analysis, the hidden layer training of the RAGD of SISO RNN can be simplified as follows

$$\hat{W}(k+1) = \hat{W}(k) + \frac{\alpha^w(k)}{\rho^w(k)} \cdot e(k) diag\{\Phi'(k)\}\hat{V}(k)^T \left(\hat{x}(k)^T + \beta^w(k)\underline{W}(k)\hat{D}^w(k)\right) \quad (32)$$

Expanding the modeling error around the hidden layer weight

$$\begin{aligned}
e(k) &= d(k) - \hat{y}(k) + \varepsilon(k) \\
&= V^*\Phi^*(k) - \hat{V}(k)\Phi(k) + \varepsilon(k) \\
&= V^*\Phi^*(k) - \hat{V}(k)\Phi(W^*\hat{x}(k)) + \hat{V}(k)\Phi(W^*\hat{x}(k)) - \hat{V}(k)\Phi(\hat{W}(k)\hat{x}(k)) + \varepsilon(k) \\
&= \hat{V}(k)\Phi(W^*\hat{x}(k)) - \hat{V}(k)\Phi(\hat{W}(k)\hat{x}(k)) + \tilde{\varepsilon}^w(k) \\
&= -\hat{V}_1(k)\mu_1(k)\tilde{W}_1(k)\hat{x}(k) - \hat{V}_2(k)\mu_2(k)\tilde{W}_2(k)\hat{x}(k) \cdots - \hat{V}_m(k)\mu_m(k)\tilde{W}_m(k)\hat{x}(k) + \tilde{\varepsilon}^w(k) \\
&= -\sum_{i=1}^{m} \hat{V}_i(k)\mu_i(k)\tilde{W}_i(k)\hat{x}(k) + \tilde{\varepsilon}^w(k) \\
&= -\hat{V}(k)diag\{\Psi(k)\}\tilde{W}(k)\hat{x}(k) + \tilde{\varepsilon}^w(k)
\end{aligned} \tag{33}$$

where $\tilde{\varepsilon}^w(k) = V^*\Phi^*(k) - \hat{V}(k)\Phi(W^*\hat{x}(k)) + \varepsilon(k)$, $\tilde{W}_i(k) \in R^{1\times n}$is the vector difference between the *i*th row of \hat W(k) and the ideal weight W^*, $\mu_i(k)$ is the mean value of the *i*th nonlinear activation function, and $\Psi(k)$ is

$$\Psi(k) = [\mu_1(k), \mu_2(k), \cdots \mu_m(k)]^T$$

Defining

$$e^w(k) = \hat{V}(k)diag\{\Psi(k)\}\tilde{W}(k)\hat{x}(k) \tag{34}$$

then equation (33) can be simplified as

$$e(k) = -e^w(k) + \tilde{\varepsilon}^w(k) \tag{35}$$

Because the output layer weight is always updated before the hidden layer weight, and $\hat{V}(k)$ of the RAGD is bounded as already proved in Section 3.2, then definitely the error signal $\tilde{\varepsilon}^w(k)$ is also bounded for every step *k*. Furthermore, since $H_2 = 1$ is inside any cone, thus we only need to study the operator H_1 to analyze the stability of the training.

Theorem 4 *If the output layer of the RNN is trained by the adaptive normalized gradient algorithm (32), the weight matrix* $\hat{W}(k)$ *is guaranteed to be stable in the sense of Lyapunov*

$$\|\tilde{W}(k+1)\|_F^2 - \|\tilde{W}(k)\|_F^2 \le 0, \qquad \forall k$$

with $\tilde{W}(k) = \hat{W}(k) - W^*$. *Also the hidden layer training of the RAGD will be* L_2*-stable in the sense of* $e^w(k) \in L_2$ *if* $\alpha^w(k) \neq 0$ *for all* $k \in Z_+$.

Proof: Subtracting W^* from both sides of (32)

$$\tilde{W}(k+1) = \tilde{W}(k) + \frac{\alpha^w(k)}{\rho^w(k)} \cdot e(k)\frac{d\hat{y}(k)}{d\hat{W}(k)} \tag{36}$$

Squaring both sides of (36)

$$\begin{aligned}
& \tilde{W}(k+1)^T\tilde{W}(k+1) \\
= & (\tilde{W}(k) + \frac{\alpha^w(k)}{\rho^w(k)} \cdot e(k)\frac{d\hat{y}(k)}{d\hat{W}(k)})^T(\tilde{W}(k) + \frac{\alpha^w(k)}{\rho^w(k)} \cdot e(k)\frac{d\hat{y}(k)}{d\hat{W}(k)}) \\
= & \tilde{W}(k)^T\tilde{W}(k) + \frac{\alpha^w(k)e(k)}{\rho^w(k)}\tilde{W}(k)^T \cdot \frac{d\hat{y}(k)}{d\hat{W}(k)} + \frac{\alpha^w(k)e(k)}{\rho^w(k)}\frac{d\hat{y}(k)}{d\hat{W}(k)^T} \cdot \tilde{W}(k) \\
& + \frac{(\alpha^w(k))^2e^2(k)}{(\rho^w(k))^2}\frac{d\hat{y}(k)}{d\hat{W}(k)^T} \cdot \frac{d\hat{y}(k)}{d\hat{W}(k)}
\end{aligned} \tag{37}$$

By the definition of Frobenius norm

$$\begin{cases}
Trace\{\tilde{W}(k+1)^T\tilde{W}(k+1)\} = \|\tilde{W}(k+1)\|_F^2 \\
Trace\{\tilde{W}(k)^T\tilde{W}(k)\} = \|\tilde{W}(k)\|_F^2 \\
Trace\{\frac{d\hat{y}(k)}{d\hat{W}^T(k)} \cdot \frac{d\hat{y}(k)}{d\hat{W}(k)}\} = \left\|\frac{d\hat{y}(k)}{d\hat{W}(k)}\right\|_F^2 = \|diag\{\Phi^{'}(k)\}\hat{V}(k)^T(\hat{x}(k)^T + \beta^w(k)\hat{B}(k))\|_F^2 \\
Trace\{\tilde{W}(k)^T \cdot \frac{d\hat{y}(k)}{d\hat{W}(k)}\} = Trace\{\frac{d\hat{y}(k)}{d\hat{W}(k)^T} \cdot \tilde{W}(k)\}
\end{cases}$$

where *Trace* {•} function is defined as the sum of the entries on the main diagonal of the associated matrix. The following equation can be derived then

$$\begin{aligned}
\|\tilde{W}(k+1)\|_F^2 - \|\tilde{W}(k)\|_F^2 = & \frac{2\alpha^w(k)e(k)}{\rho^w(k)}Trace\{\frac{d\hat{y}(k)}{d\hat{W}(k)^T}\tilde{W}(k)\} \\
& + \frac{(\alpha^w(k))^2e^2(k)}{(\rho^w(k))^2} \cdot \|diag\{\Phi^{'}(k)\}\hat{V}(k)^T(\hat{x}(k)^T + \beta^w(k)\hat{B}(k))\|_F^2
\end{aligned} \tag{38}$$

Using the trace properties, the first term on the right side of (38) can be transformed as

$$\begin{aligned}
& e(k)Trace\{\frac{d\hat{y}(k)}{d\hat{W}(k)^T}\tilde{W}(k)\} \\
= & e(k)Trace\{\left(\hat{x}(k) + \beta^w(k)\hat{B}(k)^T\right)\hat{V}(k)diag\{\Phi^{'}(k)\}\tilde{W}(k)\} \\
= & e(k)Trace\{\left(\hat{x}(k)\hat{V}(k)diag\{\Phi^{'}(k)\} + \beta^w(k)\hat{B}(k)^T\hat{V}(k)diag\{\Phi^{'}(k)\}\right)\tilde{W}(k)\} \\
= & e(k)Trace\{\hat{x}(k)\hat{V}(k)diag\{\Phi^{'}(k)\}\tilde{W}(k)\} + e(k)\beta^w(k)Trace\{\hat{B}(k)^T\hat{V}(k)diag\{\Phi^{'}(k)\}\tilde{W}(k)\} \\
= & e(k)Trace\{\hat{V}(k)diag\{\Phi^{'}(k)\}\tilde{W}(k)\hat{x}(k)\} + e(k)\beta^w(k)Trace\{\hat{V}(k)diag\{\Phi^{'}(k)\}\tilde{W}(k)\hat{B}(k)^T\} \\
= & e(k)\hat{V}(k)diag\{\Phi^{'}(k)\}\tilde{W}(k)\hat{x}(k) + e(k)\beta^w(k)V(k)diag\{\Phi^{'}(k)\}\tilde{W}(k)\hat{x}(k)\hat{x}(k)^T(\delta I + \hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k) \\
= & e(k)\hat{V}(k)diag\{\Phi^{'}(k)\}\tilde{W}(k)\hat{x}(k)(1 + \beta^w(k)\hat{x}(k)^T(\delta I + \hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T) \\
= & e(k)(\sum_{i=1}^{m}\hat{V}_i(k)\Phi_i^{'}(k)\tilde{W}_i(k)\hat{x}(k))(1 + \beta^w(k)\hat{x}(k)^T(\delta I + \hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T)
\end{aligned} \tag{39}$$

where the third equality to the last is derived by the similar perturbation method as the one in the output layer training (adding a small constant diagonal matrix δ*I* to $\hat{x}(k)\hat{x}(k)^T$to make it invertible, see the proof in Section 3.2).

Before proceeding, let's consider a RNN with scalar weight $\hat{W}$ (k). The relation of the local attractor basin of the instantaneous square error against the $\tilde{W}$ (k) can be presented by $-\frac{df(e(k))}{\hat{W}(k)}\tilde{W}(k)$, as illustrated in Figure 3 [10]. Extend this result to the RNN with a matrix weight $\hat{W}$ (k), we have a similar presentation by the local attractor basin concept

$$-\frac{df(e(k))}{\hat{W}_i(k)^T}\tilde{W}_i(k) \le 0, \qquad \forall k, i \tag{40}$$

By the local attractor basin properties in (40)

$$\begin{aligned} & e(k)(\hat{V}_i(k)\Phi_i'(k)\tilde{W}_i(k)\hat{x}(k))(1+\beta^w(k)\hat{x}(k)^T(\delta I+\hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T) \\ & = e(k)\frac{d\hat{y}(k)}{\hat{W}_i(k)^T}\tilde{W}_i(k) = -\frac{df(e(k))}{\hat{W}_i(k)^T}\tilde{W}_i(k) \le 0 \end{aligned} \tag{41}$$

The right side of (39) can be enlarged as

$$\begin{aligned} & e(k)(\sum_{i=1}^{m}\frac{\Phi_i'(k)}{\mu_i(k)}\cdot\hat{V}_i(k)\mu_i(k)\tilde{W}_i(k)\hat{x}(k))(1+\beta^w(k)\hat{x}(k)^T(\delta I+\hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T) \\ \le\ & e(k)(\frac{\phi_{min}'(k)}{\mu_{max}})\cdot(\sum_{i=1}^{m}\hat{V}_i(k)\mu_i(k)\tilde{W}_i(k)\hat{x}(k))(1+\beta^w(k)\hat{x}(k)^T(\delta I+\hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T) \\ =\ & \frac{\phi_{min}'(k)}{\mu_{max}}\cdot e(k)e^w(k)(1+\beta^w(k)\hat{x}(k)^T(\delta I+\hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T) \end{aligned} \tag{42}$$

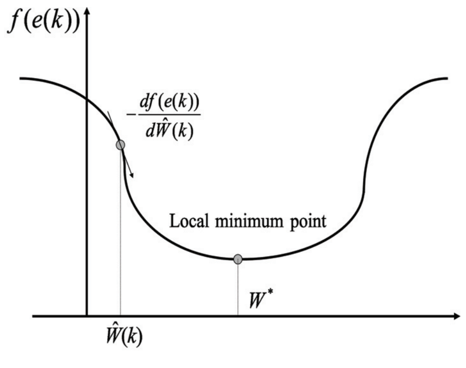


Figure 3. Illustration of a local attractor basin of the RNN against a scalar estimated weight $\hat{W}$ (k)

Substituting (42) into (39)

$$\begin{aligned} & \|\tilde{W}(k+1)\|_F^2 - \|\tilde{W}(k)\|_F^2 \\ \le\ & \frac{2\alpha^w(k)\phi_{min}'(k)e(k)e^w(k)}{\rho^w(k)\mu_{max}}(1+\beta^w(k)\hat{x}(k)^T(\delta I+\hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T) \end{aligned}$$

$$+\frac{(\alpha^w(k))^2 e^2(k)}{(\rho^w(k))^2} \cdot \|diag\{\Phi^{'}(k)\}\hat{V}(k)^T(\hat{x}(k)^T + \beta^w(k)\hat{B}(k))\|_F^2 \tag{43}$$

Substituting (35) into (43)

$$\begin{aligned}
& \|\tilde{W}(k+1)\|_F^2 - \|\tilde{W}(k)\|_F^2 \\
\leq\; & \frac{2\alpha^w(k)\phi'_{min}(k)(\tilde{\varepsilon}^w(k)e(k) - e^2(k))}{\rho^w(k)\mu_{max}}(1+\beta^w(k)\hat{x}(k)^T(\delta I + \hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T) \\
& +\frac{(\alpha^w(k))^2 e^2(k)}{(\rho^w(k))^2} \cdot \|diag\{\Phi^{'}(k)\}\hat{V}(k)^T(\hat{x}(k)^T + \beta^w(k)\hat{B}(k))\|_F^2 \\
\leq\; & \frac{\alpha^w(k)\phi'_{min}(k)((\tilde{\varepsilon}^w(k))^2 - e^2(k))}{\rho^w(k)\mu_{max}}(1+\beta^w(k)\hat{x}(k)^T(\delta I + \hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T) \\
& +\frac{(\alpha^w(k))^2 e^2(k)}{(\rho^w(k))^2} \cdot \|diag\{\Phi^{'}(k)\}\hat{V}(k)^T(\hat{x}(k)^T + \beta^w(k)\hat{B}(k))\|_F^2 \\
=\; & \frac{\alpha^w(k)\phi'_{min}(k)}{\rho^w(k)\mu_{max}}(1+\beta^w(k)\hat{x}(k)^T(\delta I + \hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T)((\tilde{\varepsilon}^w(k))^2 \\
& -(1-\frac{\alpha^w(k)\mu_{max}\|diag\{\Phi^{'}(k)\}\hat{V}(k)^T(\hat{x}(k)^T + \beta^w(k)\hat{B}(k))\|_F^2}{\rho^w(k)\phi'_{min}(k)(1+\beta^w(k)\hat{x}(k)^T(\delta I + \hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T)})e^2(k)) \\
\leq\; & \frac{\alpha^w(k)\phi'_{min}(k)}{\rho^w(k)\mu_{max}}(1+\beta^w(k)\hat{x}(k)^T(\delta I + \hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T)((\varepsilon^w_{max})^2 \\
& -(1-\frac{\alpha^w(k)\mu_{max}\|diag\{\Phi^{'}(k)\}\hat{V}(k)^T(\hat{x}(k)^T + \beta^w(k)\hat{B}(k))\|_F^2}{\rho^w(k)\phi'_{min}(k)})e^2(k))
\end{aligned} \tag{44}$$

By the definition of $\rho^w(k)$ and $\alpha^w(k)$ in (12) and (14) respectively, we can draw that

$$\|\tilde{W}(k+1)\|_F^2 - \|\tilde{W}(k)\|_F^2 \leq 0 \tag{45}$$

Again, consider the extreme case with the assumption of nonzero $\alpha^w(k)$. Dividing both sides of (43) by

$$\frac{2\alpha^w(k)\phi'_{min}(k)}{\mu_{max}}(1+\beta^w(k)\hat{x}(k)^T(\delta I + \hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T)$$

and then summing up to N steps

$$\begin{aligned}
-\Delta W \leq & \sum_{k=1}^{N}\left(\frac{e(k)e^w(k)}{\rho^w(k)} + \frac{\alpha^w(k)\mu_{max}\|diag\{\Phi^{'}(k)\}\hat{V}(k)^T(\hat{x}(k)^T+\beta^w(k)\hat{B}(k))\|_F^2}{2(\rho^w(k))^2\phi'_{min}(k)(1+\beta^w(k)\hat{x}(k)^T(\delta I+\hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T)}e^2(k)\right) \\
& \leq \sum_{k=1}^{N}\{\bar{e}(k)\bar{e}^w(k) + \tfrac{1}{2}\bar{\sigma}^w\bar{e}^2(k)\}
\end{aligned} \tag{46}$$

where the normalized error signals are $\bar{e}(k) = e(k)/\sqrt{\rho^w(k)}$, $\bar{e}^w(k) = e^w(k)/\sqrt{\rho^w(k)}$, and the cone is

$$\bar{\sigma}^w = \sup_k \{ \frac{\mu_{max} \| diag\{\Phi'(k)\} \hat{V}(k)^T (\hat{x}(k)^T + \beta^w(k)\hat{B}(k)) \|_F^2}{\rho^w(k)\phi'_{min}(k)} \} < 1 \tag{47}$$

and ΔW is greater than zero because for each k the Lyapunov function (45) is guaranteed smaller than or equal to zero, i.e.

$$\begin{aligned} 0 \quad \leq \quad & \Delta W = -\sum_{k=1}^{N} \frac{\mu_{max}(\|\tilde{W}(k+1)\|_F^2 - \|\tilde{W}(k)\|_F^2)}{2\phi'_{min}(k)(1+\beta^w(k)\hat{x}(k)^T(\delta I + \hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T)} \\ \leq \quad & \frac{\mu_{max}}{2min\{\phi'_{min}(k)\}} \sum_{k=1}^{N} (\|\tilde{W}(k)\|_F^2 - \|\tilde{W}(k+1)\|_F^2) \\ = \quad & \frac{\mu_{max}}{2min\{\phi'_{min}(k)\}} (\|\tilde{W}(1)\|_F^2 - \|\tilde{W}(N+1)\|_F^2) \end{aligned} \tag{48}$$

Due to the specific selection of the normalization factor in (12), the original signals $e(k)$ and $e^w(k)$ are guaranteed to be bounded according to the original operators H_1^w and H_2 [11] [15]. Now the operator H_1^w represented by (46) satisfies the condition (i) of Theorem 2. Thus we conclude that $e^w(k) \in L_2$ in case of $\alpha^w(k) \neq 0$, $\forall k \in Z_+$. ■

3.4 Robustness analysis of MIMO RNN

In this section, we discuss the RAGD training for the RNN of Multi-Input Multi-Output (MIMO) types. As mentioned in the introduction, the RNN with multiple output neurons can be regarded as consisting of several single output RNNs. Thus the training of MIMO RNN can be studied by decomposition. In detail, for the output layer training, we may calculate the gradient of each output neuron with respect to weight parameters, and then obtain the total weight updating by summing these individual gradient. As for the hidden layer, we also use this method to take into account the influence of multi-output neurons on total weight updating. Following this idea, the extension of the stability analysis from SISO to MIMO is straight forward.

Theorem 5 *If the RNN is trained by the adaptive normalized gradient algorithm (5)-(15), then the weight $\hat{V}$ (k) and $\hat{W}$ (k) are guaranteed to be stable in the sense of Lyapunov.*

Proof: *(i) Output layer analysis:* To study the stability of the RAGD, we need to establish the error dynamics of the training algorithm. First of all, define the estimation error

$$e^v(k) = \hat{V}(k)\Phi(k) - V^*\Phi(k) = \tilde{V}(k)\Phi(k) \tag{49}$$

where $V^* \in R^{l\times m}$ is the ideal output layer weight, and

$$\begin{cases} \tilde{V}(k) = V(k) - V^* \\ \Phi^*(k) = [\quad \cdots \quad h(W_i^*(k)x^*(k)) \quad \cdots \quad]^T \end{cases}$$

Then we expand $e(k) \in R^{l\times 1}$ with respect to the output layer weight as

$$\begin{aligned} e(k) &= d(k) - \hat{y}(k) + \varepsilon(k) \\ &= [V^*\Phi^*(k) - V^*\Phi(k)] - [\hat{V}(k)\Phi(k) - V^*\Phi(k)] + \varepsilon(k) \\ &= \tilde{\varepsilon}^v(k) - e^v(k) \end{aligned} \tag{50}$$

with $\tilde{\varepsilon}^v(k) = V^*\Phi^*(k) - V^*\Phi(k) + \varepsilon(k)$. In (50), we restructure the output layer training of the RAGD algorithm into a closed loop form same as that of (16) , by which the weight estimation error $e^v(k)$ is referred as the output signal. Subtracting V^* and squaring both sides of the output layer training equation in (5)

$$\begin{aligned} \|\tilde{V}(k+1)\|_F^2 &= \|\tilde{V}(k)\|_F^2 + \frac{2\alpha^v(k)}{\rho^v(k)} Trace\{(\Phi(k)^T + \beta^v(k)\hat{A}(k))^T e(k)^T \tilde{V}(k)\} \\ &\quad + (\frac{\alpha^v(k)}{\rho^v(k)})^2 \|e(k)\Phi(k)^T + e(k)\beta^v(k)\hat{A}(k)\|_F^2 \end{aligned} \tag{51}$$

By the matrix trace properties

$$\begin{aligned} & Trace\{(\Phi(k)^T + \beta^v(k)\hat{A}(k))^T e(k)^T \tilde{V}(k)\} \\ = & Trace\{\Phi(k)e(k)^T\tilde{V}(k)\} + \beta^v(k)Trace\{\hat{A}(k)^T e(k)^T \tilde{V}(k)\} \\ = & Trace\{e(k)^T\tilde{V}(k)\Phi(k)\} + \beta^v(k)Trace\{e(k)^T\tilde{V}(k)\hat{A}(k)^T\} \\ = & e(k)^T\tilde{V}(k)\Phi(k) + \beta^v(k)e(k)^T\tilde{V}(k)\hat{A}(k)^T \end{aligned}$$

Again, we employ the customary practice by using a small positive perturbation constant δ to make $\delta I + \Phi(k)\Phi(k)^T$ full rank and then apply the approximation as

$$\begin{aligned} & Trace\{(\Phi(k)^T + \beta^v(k)\hat{A}(k))^T e(k)^T \tilde{V}(k)\} \\ = & e(k)^T\tilde{V}(k)\Phi(k) + \beta^v(k)e(k)^T\tilde{V}(k)\Phi(k)\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T \\ = & e(k)^T e^v(k) + \beta^v(k)e(k)^T e^v(k)\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T \\ = & e(k)^T e^v(k)(1 + \beta^v(k)\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T) \end{aligned} \tag{52}$$

Substituting (50) and (52) into (51)

$$\begin{aligned} & \|\tilde{V}(k+1)\|_F^2 - \|\tilde{V}(k)\|_F^2 \\ = & \frac{2\alpha^v(k)}{\rho^v(k)} e(k)^T e^v(k)(1 + \beta^v(k)\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T) \\ & + (\frac{\alpha^v(k)}{\rho^v(k)})^2 \|e(k)\Phi(k)^T + e(k)\beta^v(k)\hat{A}(k)\|_F^2 \\ \leq & \frac{2\alpha^v(k)}{\rho^v(k)} e(k)^T e^v(k)(1 + \beta^v(k)\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T) \\ & + (\frac{\alpha^v(k)}{\rho^v(k)})^2 \cdot \|e(k)\|^2 \cdot \|\Phi(k)^T + \beta^v(k)\hat{A}(k)\|^2 \end{aligned} \tag{53}$$

$$\begin{aligned}
&= \frac{2\alpha^v(k)}{\rho^v(k)}(e(k)^T\tilde{\varepsilon}^v(k) - \|e(k)\|^2)(1+\beta^v(k)\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T) \\
&\quad +(\frac{\alpha^v(k)}{\rho^v(k)})^2 \cdot \|e(k)\|^2 \cdot \|\Phi(k)^T + \beta^v(k)\hat{A}(k)\|^2 \\
&\le \frac{\alpha^v(k)}{\rho^v(k)}(\|\tilde{\varepsilon}^v(k)\|^2 - \|e(k)\|^2)(1+\beta^v(k)\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T) \\
&\quad +(\frac{\alpha^v(k)}{\rho^v(k)})^2 \cdot \|e(k)\|^2 \cdot \|\Phi(k)^T + \beta^v(k)\hat{A}(k)\|^2 \qquad (54)
\end{aligned}$$

$$\begin{aligned}
&= \frac{\alpha^v(k)}{\rho^v(k)}(1+\beta^v(k)\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T)(\|\tilde{\varepsilon}^v(k)\|^2 \\
&\quad -(1-\frac{\alpha^v(k)\|\Phi(k)^T + \beta^v(k)\hat{A}(k)\|^2}{\rho^v(k)(1+\beta^v(k)\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T)})\|e(k)\|^2) \\
&\le \frac{\alpha^v(k)}{\rho^v(k)}(1+\beta^v(k)\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T)(\|\tilde{\varepsilon}^v(k)\|^2 \\
&\quad -(1-\frac{\alpha^v(k)\|\Phi(k)^T + \beta^v(k)\hat{A}(k)\|^2}{\rho^v(k)})\|e(k)\|^2) \\
&\le \frac{\alpha^v(k)}{\rho^v(k)}(1+\beta^v(k)\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T)((\varepsilon^v_{max})^2 \\
&\quad -(1-\frac{\alpha^v(k)\|\Phi(k)^T + \beta^v(k)\hat{A}(k)\|^2}{\rho^v(k)})\|e(k)\|^2) \qquad (55)
\end{aligned}$$

where (54) is because of the triangular inequality $\tilde{\varepsilon}^v(k)^T e(k) \le (\|\tilde{\varepsilon}^v(k)\|^2 + \|e(k)\|^2)/2$, and (55) is due to

$$\beta^v(k)\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T \ge 0$$

Combining the inequality (55) with the definition of $\rho^v(k)$ and $\alpha^v(k)$ in (11) and (13) respectively, the Lyapunov equation of output layer estimation error can be derived

$$\|\tilde{V}(k+1)\|_F^2 - \|\tilde{V}(k)\|_F^2 \le 0 \qquad (56)$$

(ii) Hidden layer analysis: Expanding $e(k)$ with respect to the estimation error of hidden layer weight

$$\begin{aligned}
e^w(k) &= \hat{V}(k)\Phi(\hat{W}(k)\hat{x}(k)) - \hat{V}(k)\Phi(W^*\hat{x}(k)) \\
&= \sum_{i=1}^{l}\sum_{j=1}^{m} e_i(k)\hat{V}_{i,j}(k)\mu_j(k)\tilde{W}_j(k)\hat{x}(k) \qquad (57)
\end{aligned}$$

where $W^*(k) \in R^{m\times n}$ is the ideal hidden layer weight matrices, $x^*(k) \in R^{n\times 1}$ is the ideal state vector, $\mu_i(k)$ is the mean value of the ith nonlinear activation function at instant k, and $\tilde{W}_j(k) = \hat{W}_j(k) - W_j^*$. Using the local attractor basin concept that similar to (40)

$$
\begin{aligned}
& Trace\{\left(\hat{x}(k)^T+\beta^w(k)\hat{B}(k)\right)^T e(k)^T\hat{V}(k)diag\{\Phi'(k)\}\tilde{W}(k)\} \\
= \; & Trace\{\hat{x}(k)e(k)^T\hat{V}(k)diag\{\Phi'(k)\}\tilde{W}(k)\}+\beta^w(k)Trace\{\hat{B}(k)^Te(k)^T\hat{V}(k)diag\{\Phi'(k)\}\tilde{W}(k)\} \\
= \; & Trace\{e(k)^T\hat{V}(k)diag\{\Phi'(k)\}\tilde{W}(k)\hat{x}(k)\}+\beta^w(k)Trace\{e(k)^T\hat{V}(k)diag\{\Phi'(k)\}\tilde{W}(k)\hat{B}(k)^T\} \\
= \; & e(k)^T\hat{V}(k)diag\{\Phi'(k)\}\tilde{W}(k)\hat{x}(k)+\beta^w(k)e(k)^T\hat{V}(k)diag\{\Phi'(k)\}\tilde{W}(k)\hat{B}(k)^T \\
= \; & e(k)^T\hat{V}(k)diag\{\Phi'(k)\}\tilde{W}(k)\hat{x}(k) \\
& +\beta^w(k)e(k)^T\hat{V}(k)diag\{\Phi'(k)\}\tilde{W}(k)\hat{x}(k)\hat{x}(k)^T(\delta I+\hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T \\
= \; & e(k)^T\hat{V}(k)diag\{\Phi'(k)\}\tilde{W}(k)\hat{x}(k)\left(1+\beta^w(k)\hat{x}(k)^T(\delta I+\hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T\right) \\
= \; & (\sum_{i=1}^{l}\sum_{j=1}^{m}e_i(k)\hat{V}_{i,j}(k)\Phi'_j(k)\tilde{W}_j(k)\hat{x}(k))\left(1+\beta^w(k)\hat{x}(k)^T(\delta I+\hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T\right) \\
= \; & (\sum_{i=1}^{l}\sum_{j=1}^{m}\frac{\Phi'_j(k)}{\mu_j(k)}e_i(k)\hat{V}_{i,j}(k)\mu_j(k)\tilde{W}_j(k)\hat{x}(k))\left(1+\beta^w(k)\hat{x}(k)^T(\delta I+\hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T\right) \\
\leq \; & \frac{\phi'_{min}(k)}{\mu_{max}}(\sum_{i=1}^{l}\sum_{j=1}^{m}e_i(k)\hat{V}_{i,j}(k)\mu_j(k)\tilde{W}_j(k)\hat{x}(k))(1+\beta^w(k)\hat{x}(k)^T(\delta I+\hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T) \\
= \; & \frac{\phi'_{min}(k)}{\mu_{max}}e^w(k)^Te(k)\left(1+\beta^w(k)\hat{x}(k)^T(\delta I+\hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T\right) \qquad (58)
\end{aligned}
$$

Substituting *W**and squaring both sides of hidden layer training equation of the RAGD in (5), we can derive the Lyapunov function of the hidden layer weight of MIMO RNN based upon (58) as

$$
\begin{aligned}
& \|\tilde{W}(k+1)\|_F^2-\|\tilde{W}(k)\|_F^2 \\
= \; & \frac{2\alpha^w(k)}{\rho^w(k)}Trace\{(\hat{x}(k)^T+\beta^w(k)\hat{B}(k))^Te(k)^T\hat{V}(k)diag\{\Phi'(k)\}\tilde{W}(k)\} \\
& +(\frac{\alpha^w(k)}{\rho^w(k)})^2\|diag\{\Phi'(k)\}\hat{V}(k)^Te(k)(\hat{x}(k)^T+\beta^w(k)\hat{B}(k))\|_F^2 \\
\leq \; & \frac{2\alpha^w(k)\phi'_{min}(k)}{\rho^w(k)\mu_{max}}e^w(k)^Te(k)(1+\beta^w(k)\hat{x}(k)^T(\delta I+\hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T) \\
& +(\frac{\alpha^w(k)}{\rho^w(k)})^2\|diag\{\Phi'(k)\}\hat{V}(k)^T\|_F^2\cdot\|e(k)\|^2\cdot\|\hat{x}(k)^T+\beta^w(k)\hat{B}(k)\|^2 \qquad (59)
\end{aligned}
$$

$$
\begin{aligned}
= \; & \frac{2\alpha^w(k)\phi'_{min}(k)}{\rho^w(k)\mu_{max}}(1+\beta^w(k)\hat{x}(k)^T(\delta I+\hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T)(\tilde{\varepsilon}^w(k)^Te(k)-e(k)^Te(k)) \\
& +(\frac{\alpha^w(k)}{\rho^w(k)})^2\|diag\{\Phi'(k)\}\hat{V}(k)^T\|_F^2\cdot\|e(k)\|^2\cdot\|\hat{x}(k)^T+\beta^w(k)\hat{B}(k)\|^2 \\
\leq \; & \frac{\alpha^w(k)\phi'_{min}(k)}{\rho^w(k)\mu_{max}}(1+\beta^w(k)\hat{x}(k)^T(\delta I+\hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T)(\|\tilde{\varepsilon}^w(k)\|^2-\|e(k)\|^2)
\end{aligned}
$$

$$
\begin{aligned}
&+(\frac{\alpha^w(k)}{\rho^w(k)})^2\|diag\{\Phi'(k)\}\hat{V}(k)^T\|_F^2 \cdot \|e(k)\|^2 \cdot \|\hat{x}(k)^T+\beta^w(k)\hat{B}(k)\|^2 \\
= \quad & \frac{\alpha^w(k)\phi'_{min}(k)}{\rho^w(k)\mu_{max}}(1+\beta^w(k)\hat{x}(k)^T(\delta I+\hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T)(\|\tilde{\varepsilon}^w(k)\|^2 \\
&-(1-\frac{\alpha^w(k)\mu_{max}\|diag\{\Phi'(k)\}\hat{V}(k)^T\|_F^2\cdot\|\hat{x}(k)^T+\beta^w(k)\hat{B}(k)\|^2}{\rho^w(k)\phi'_{min}(k)(1+\beta^w(k)\hat{x}(k)^T(\delta I+\hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T)})\|e(k)\|^2) \\
\le \quad & \frac{\alpha^w(k)\phi'_{min}(k)}{\rho^w(k)\mu_{max}}(1+\beta^w(k)\hat{x}(k)^T(\delta I+\hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T)((\varepsilon^w_{max})^2 \\
&-(1-\frac{\alpha^w(k)\mu_{max}\|diag\{\Phi'(k)\}\hat{V}(k)^T\|_F^2\cdot\|\hat{x}(k)^T+\beta^w(k)\hat{B}(k)\|^2}{\rho^w(k)\phi'_{min}(k)})\|e(k)\|^2) \\
\le \quad & 0
\end{aligned}
\tag{60}
$$

Summarizing (56) and (60), we can conclude the proof. ∎

Theorem 6 *If a MIMO RNN is trained by the adaptive normalized gradient algorithm (5)-(15), and $\alpha^v(k)$, $\alpha^w(k)$ are nonzero for all $k \in Z_+$, then the training will be L_2-stable in the sense of $e^v(k), e^w(k) \in L_2$.*

Proof: Respectively, dividing both sides of (53) and (59) by the following two factors (since $\alpha^v(k); \alpha^w(k) \neq 0$)

$$
\begin{cases}
2\alpha^v(k)(1+\beta^v(k)\Phi(k)^T(\delta I+\Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T) \\
\frac{2\alpha^w(k)\phi'_{min}(k)}{\mu_{max}}(1+\beta^w(k)\hat{x}(k)^T(\delta I+\hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T)
\end{cases}
\tag{61}
$$

Summing both inequalities up to N steps, then for the output layer

$$
\begin{aligned}
-\Delta V_N = \sum_{k=1}^{N}\{\frac{e^v(k)^Te(k)}{\rho^v(k)} + \frac{\alpha^v(k)\|\Phi(k)^T+\beta^v(k)\hat{A}(k)\|^2}{2(\rho^v(k))^2(1+\beta^v(k)\Phi(k)^T(\delta I+\Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T)}\|e(k)\|^2\} \\
\le \sum_{k=1}^{N}\{\bar{e}^v(k)^T\bar{e}_v(k)+\tfrac{1}{2}\bar{\sigma}^v\|\bar{e}_v(k)\|^2\}
\end{aligned}
\tag{62}
$$

and for the hidden layer

$$
\begin{aligned}
-\Delta W_N \le \sum_{k=1}^{N}\{\frac{e^w(k)^Te(k)}{\rho^w(k)} + \frac{\mu_{max}\alpha^w(k)\|diag\{\Phi'(k)\}\hat{V}(k)^T\|_F^2\cdot\|\hat{x}(k)^T+\beta^w(k)\hat{B}(k)\|^2}{2(\rho^w(k))^2\phi'_{min}(k)(1+\beta^w(k)\hat{x}(k)^T(\delta I+\hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T)}\|e(k)\|^2\} \\
\le \sum_{k=1}^{N}\{\bar{e}^w(k)^T\bar{e}_w(k)+\tfrac{1}{2}\bar{\sigma}^w\|\bar{e}_w(k)\|^2\}
\end{aligned}
\tag{63}
$$

where

$$0 \leq \Delta V_N \leq \frac{1}{2}(\|\tilde{V}(1)\|_F^2 - \|\tilde{V}(N+1)\|_F^2)$$
$$0 \leq \Delta W_N \leq \frac{\mu_{max}}{2min\{\phi'_{min}(k)\}}(\|\tilde{W}(1)\|_F^2 - \|\tilde{W}(N+1)\|_F^2)$$

and the normalized signals are defined by

$$\bar{e}_v(k) = \frac{e(k)}{\sqrt{\rho^v(k)}} \qquad \bar{e}_w(k) = \frac{e(k)}{\sqrt{\rho^w(k)}}$$
$$\bar{e}^v(k) = \frac{e^v(k)}{\sqrt{\rho^v(k)}} \qquad \bar{e}^w(k) = \frac{e^w(k)}{\sqrt{\rho^w(k)}}$$

$$\bar{\sigma}^v(k) = \sup_k \{\frac{\|\Phi(k)^T + \beta^v(k)\hat{A}(k)\|^2}{\rho^v(k)(1 + \beta^v(k)\Phi(k)^T(\delta I + \Phi(k)\Phi(k)^T)^{-1}\hat{A}(k)^T)}\}$$
$$\bar{\sigma}^w(k) = \sup_k \{\frac{\mu_{max}\|diag\{\Phi'(k)\}\hat{V}(k)^T\|_F^2 \cdot \|\hat{x}(k)^T + \beta^w(k)\hat{B}(k)\|^2}{\rho^w(k)\phi'_{min}(k)(1 + \beta^w(k)\hat{x}(k)^T(\delta I + \hat{x}(k)\hat{x}(k)^T)^{-1}\hat{B}(k)^T)}\}$$

Due to the specific selection of the normalization factor $\rho^v(k)$ and $\rho^w(k)$ as in (11) and (12), the normalized error signals $\bar{e}_v(k)$, $\bar{e}^v(k)$, $\bar{e}_w(k)$, and $\bar{e}^w(k)$ are guaranteed to be bounded. Now, for each $\hat{V}(k)$ and $\overline{W}(k)$, applying the Cluett's law, we found that the operator H_1^v and H_1^w represented by (62) and (63) satisfy the condition (i). Further, $H_2 = 1$ ensures condition (ii) holds, thus $e^v(k)$and $e^w(k)$ are L_2 stable with $\alpha^v(k)$, $\alpha^w(k) \neq 0$, $\forall k \in Z_+$. ■

3.5 Summary

In Section 3, we introduce a novel RAGD training algorithm of RNN. Because conventional gradient type algorithms most likely suffer from slow convergence when dealing with statistically non-stationary inputs, the RAGD aims at overcoming such shortcomings via a series of new training parameters. Moreover, the robust local stability of the RAGD has been addressed for three layer RNN based upon the Cluett's law. Theoretical analysis shows that the proposed adaptive parameters improve the training performance in terms of a deeper gradient descent direction updating, which leads to a better transient response. Further, compared to BPTT, the RAGD algorithm requires limited backward unfolding, which reduces the computational complexity. The flow chart of the overall training procedure of the RAGD is summarized in Figure 4.

4. Applications in realtime signal processing

This section presents quantitative studies of the RAGD algorithm via computer simulations. We choose three of the most representative applications of RNN to verify the effectiveness of the RAGD. By default, the RNN is constructed with 50 hidden neurons and 5 input nodes. The 5- dimensional input vector consists of current and last sample of time sequence $u(k)$ and RNN output feedback with 1 to 3 steps delay respectively. Both hidden and output layer weights are initialized as uniformly distributed in the interval of (-1, 1). Sigmoid function is chosen as activation function, which is monotonic increasing, and both first and second order differentiable. The function and its first order derivative are given in equation (64), including the boundaries

$$\begin{cases} 0 \le \phi(x) = \frac{1}{1+e^{-\lambda x}} \le 1 \\ 0 \le \phi'(x) = \frac{\lambda e^{-\lambda x}}{(1+e^{-\lambda x})^2} \le \frac{\lambda}{4} \end{cases} \tag{64}$$

For the purpose of comparison, in most of the simulations we also provide the results of other training algorithms, such as the Truncated BPTT (T-BPTT) and the N-RTRL etc.

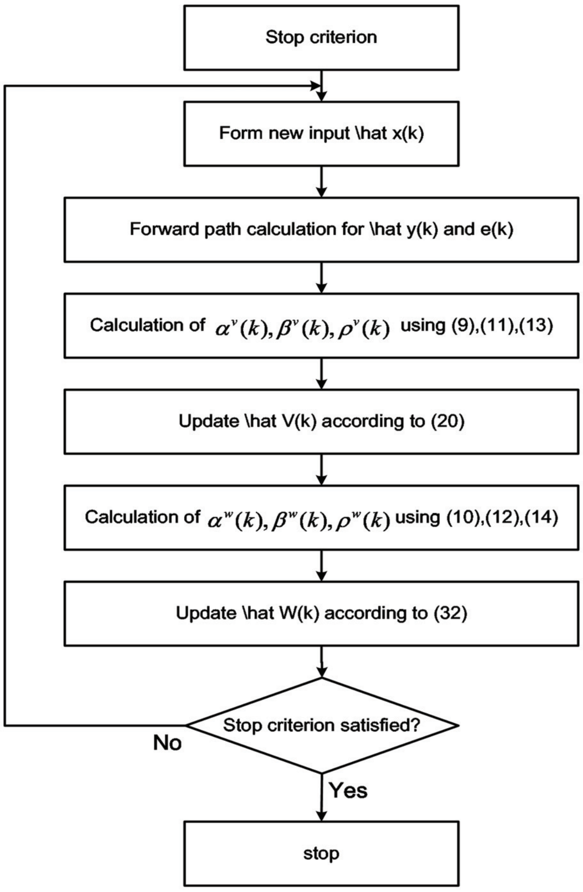

Stop criterion
Form new input \hat x(k)
Forward path calculation for \hat y(k) and e(k)
Calculation of $\alpha^v(k), \beta^v(k), \rho^v(k)$ using (9),(11),(13)
Update \hat V(k) according to (20)
Calculation of $\alpha^w(k), \beta^w(k), \rho^w(k)$ using (10),(12),(14)
Update \hat W(k) according to (32)
Stop criterion satisfied?
No
Yes
stop

Figure 4. Flow chart of the RAGD training algorithm for SISO RNN

4.1 Time series prediction

In the first simulation, the performance of the RAGD is evaluated via time series prediction problems. The RNN is employed to predict the next sample (one step) of a real sequence $\{y(k)\}$, which is generated by the following process

$$y(k+1) = \frac{y(k)y(k-1)y(k-2)u(k-1)(y(k-2)-1)+u(k)}{1+y^2(k-1)+y(k-2)} \tag{65}$$

where $u(k)$ is white Gaussian input sequence. The model of (65) is chosen from the benchmark problem in [1] (pp.159). Two data groups are generated in simulations. One is the training data set, and the other is for evaluation purpose. The traces of the time series for training and evaluation are displayed in Figure 5.

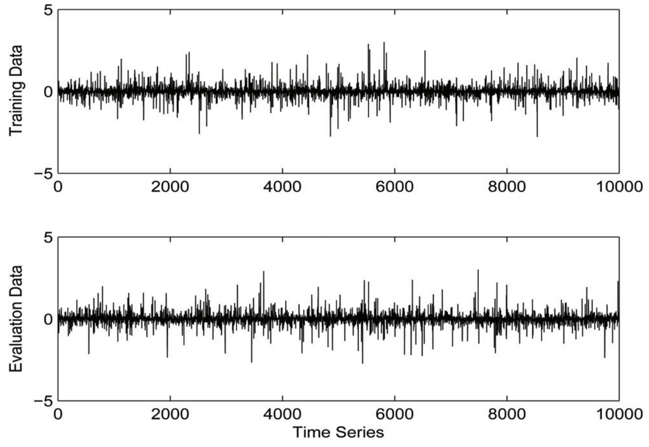


Figure 5. Sequences of the time series for training and evaluation

To provide a comparative idea, we have also implemented the N-RTRL in simulations with constant C = 0 and C = 0.2 respectively. All the simulations run for 10000 steps. In order to present a clear illustration on both transient and steady state performance of each training algorithm, the training errors are displayed by the first 100 steps and the full 10000 steps separately as shown in Figure 6 and 7. Moreover, the squared training errors of the first 100

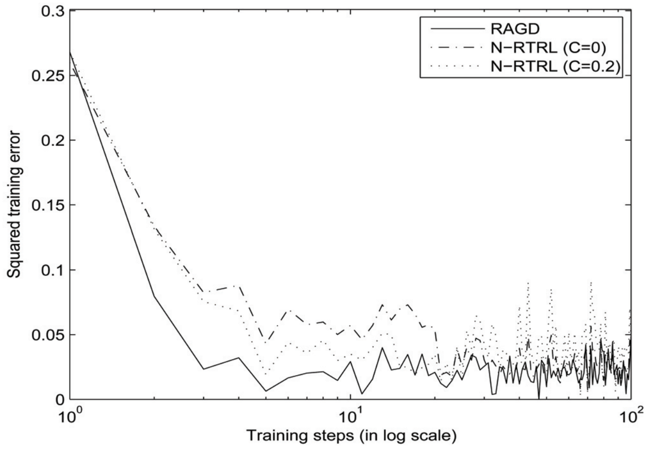


Figure 6. Squared training errors of the first 100 steps with the same set of random initializations for different algorithms

steps are plot in logarithmic format to provide a further better comparison. The steady state training errors are expressed in dB (20 times the logarithm of the amplitude ratio between error and signal) such that performance difference between the RAGD and the N-RTRL can be more explicit. The traces of the normalization factors are shown in Figure 8. The trajectories of the Frobenius norms of RNN weights with the RAGD training are displayed in Figure 9.

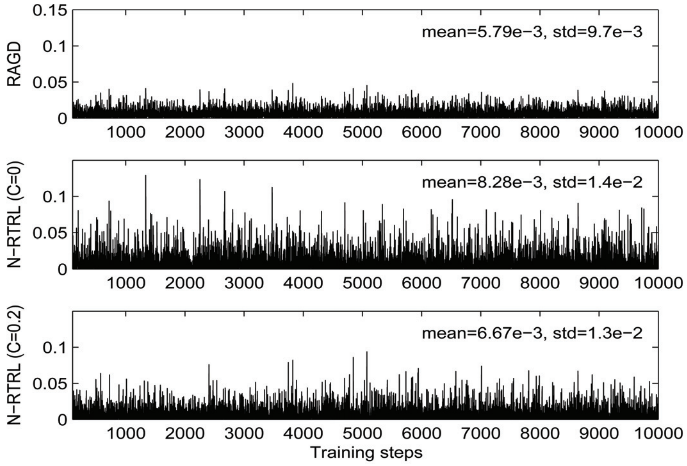


Figure 7. Squared training errors of full 3000 steps for different algorithms

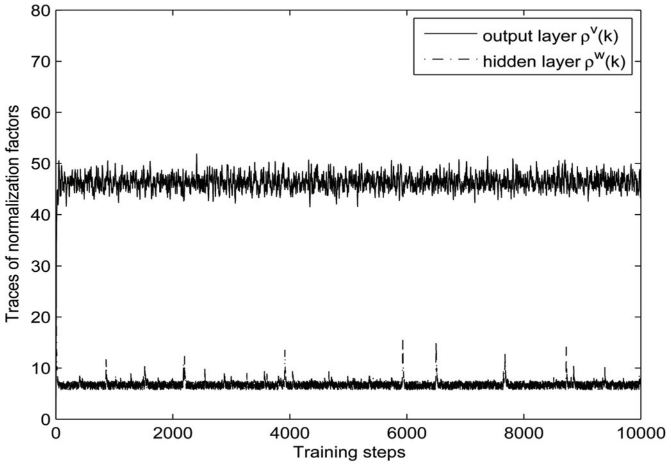


Figure 8. Traces of normalization factors $\rho^v(k)$ and $\rho^w(k)$

The results show that the RAGD algorithm is successfully stabilized in the sense that the Frobenius norms of the weights converge. The convergence of the RAGD is faster than the N- RTRL with both parameter values. Moreover, the RAGD can achieve better steady state error (mean squared training error 5.79*e*-3) than the N-RTRL (mean squared training errors 6.67*e*-3 and 8.28*e* - 3 respectively).

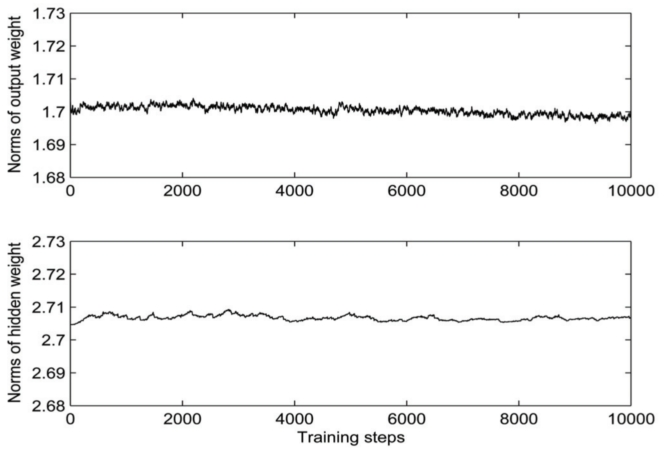


Figure 9. Traces of the Frobenius norms of RNN weights with the RAGD training

In addition to the proposed adaptive training parameters, we also investigate how the training is affected by the number of hidden layer neurons and the exponential factor of activation functions. The statistics with respect to various values of this two parameters are given in Table 1 and 2 respectively. The data are obtained by averaging the results of 50 runs (each have 10000 steps).

All simulations start with same initial weights, which can make a same starting point of the training error such that we can make a convincing comparison. The results indicate that the steady state performance is slightly improved as the λ increases. A possible reason is that transition slope of linear region of activation function becomes higher (faster) with larger λ. A similar phenomena is also observed in [7] (pp.617). In contrast, there is no obvious influence of the neuron number on the training performance.

exponential factor λ	RAGD		N-RTRL (C=0.2)		N-RTRL($C=0$)	
	Mean	Std	Mean	Std	Mean	Std
1	5.72e-3	1.0e-2	6.61e-3	1.2e-2	8.24e-3	1.4e-2
4	5.79e-3	1.0e-2	6.67e-3	1.3e-2	8.28e-3	1.4e-2
8	5.82e-3	1.0e-2	6.73e-3	1.2e-2	8.29e-3	1.5e-2
12	5.82e-3	1.1e-2	6.75e-3	1.3e-2	8.31e-3	1.2e-2

Table 1. Statistics of squared training errors of the RAGD with different λ

number of nodes	RAGD		N-RTRL (C=0.2)		N-RTRL($C=0$)	
	Mean	Std	Mean	Std	Mean	Std
10	5.78e-3	1.0e-2	6.69e-3	1.3e-2	8.30e-3	1.3e-2
20	5.77e-3	1.0e-2	6.64e-3	1.3e-2	8.28e-3	1.4e-2
50	5.79e-3	1.0e-2	6.67e-3	1.3e-2	8.28e-3	1.4e-2
80	5.81e-3	1.0e-2	6.68e-3	1.3e-2	8.29e-3	1.3e-2

Table 2. Statistics of squared training errors of the RAGD with di®erent neurons

4.2 Output tracking of Hammerstein-Wiener model

In the second example, the RAGD is evaluated using a system identification problem. In this simulation, the "unknown" plant consists of a dynamic linear block followed by a static nonlinearity, which is a so-called Hammerstein-Wiener model. Furthermore, the model dynamics is supposed to vary with time in terms of the time-varying coefficients of linear part, which can be expressed in a polynomial form as [19]

$$\begin{cases} x(k) = 0.3tanh(0.5u(k)) + 0.5tanh(0.8u(k)) \\ d(k) = 0.7680x(k) + 0.2872e^{-0.006k}x(k-1) - 0.1147e^{-0.03k}x(k-2) + 0.2140x(k-3) \\ \qquad -0.6435sin(0.008k)u(k-4) + 0.3548x(k-5) + 0.0197x(k-6) \\ \qquad -0.0201x(k-7) + 0.0954x(k-8) \end{cases} \tag{66}$$

The objective of the simulation is to model the plant's input-output behavior by the RNN. The command signal was given by $u(k)$, and the RNN attempts to emulate the plant output $d(k)$ as close as possible. The estimation error between actual plant output and reference signal $e(k) = d(k)\text{-}y(k)$ is fed back to RNN to adjust the weight parameters. One of the most crucial tasks in system identification is the design of appropriate excitation signals. It is important that the training data cover the entire range of plant operation due to non accurate extrapolation of RNN. In this simulation, Amplitude Modulated Pseudo Random Permutation (AMPRP) sequence are generated as training set, with the data uniformly distributed in the range of (0, 1), see Figure 10. We have also implemented the T-BPTT algorithm in simulations. The learning rate α = 0.05 (tuned by trial-and-error) was used for T-BPTT.We present the squared training error of first 1000 (transient) and 1000-5000 (steady state) steps separately in Figure 11 and 12. Results show that the RAGD converges within 200 steps while T-BPTT takes around 1000 steps. In addition, the steady state error of the RAGD is smaller than T-BPTT. Hence we say the RAGD is capable of providing a faster response to the changes of system dynamics. The traces of the normalization factors of the RAGD are provided in Figure 13.

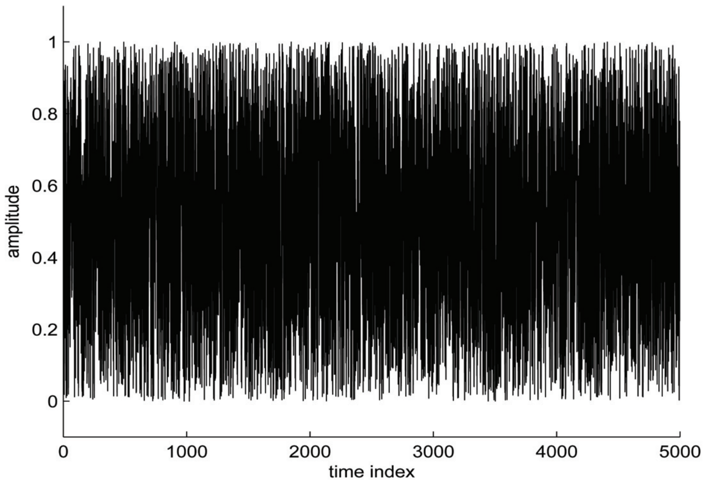


Figure 10. Trace of AMPRP input for model identification

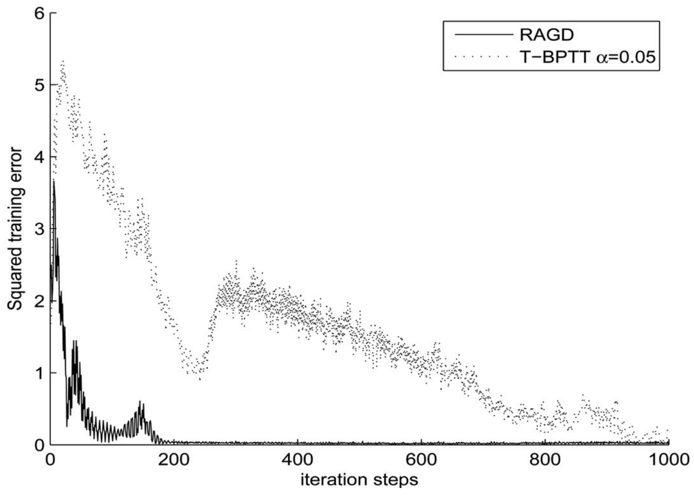


Figure 11. Squared training errors of the first 1000 steps

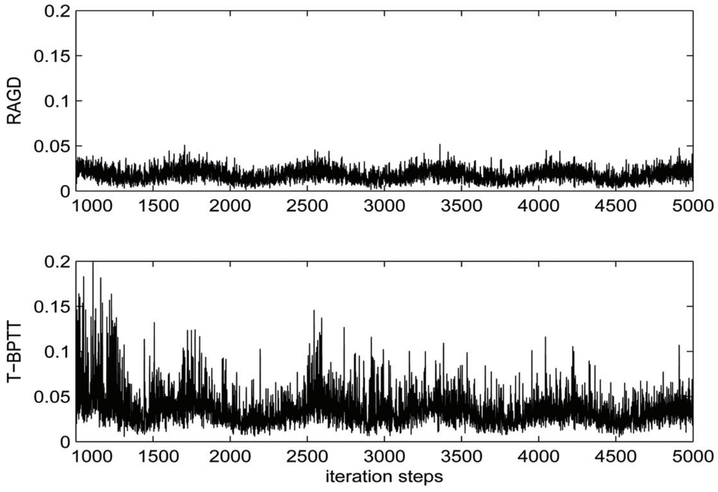


Figure 12. Squared training errors of steady state: 1000-5000 steps

4.3 Pattern association of binary image

In the last simulation, we study the problem of stable equilibrium point learning associated with a discrete-time RNN using the RAGD algorithm. In the applications of visual processing and pattern recognition, RNN plays an important role due to the feature of associative memory. The work presented in this section is inspired by an earlier paper of Liang and Gupta [8]. In [8], the authors considered absolute stability of BPTT for a general class of discrete time RNN by the Lyapunov first method. In this work the RAGD will be incorporated in place of BPTT to develop a stable learning process. To present comparison with the precedent works [20], we implement a similar simulation case of binary pattern

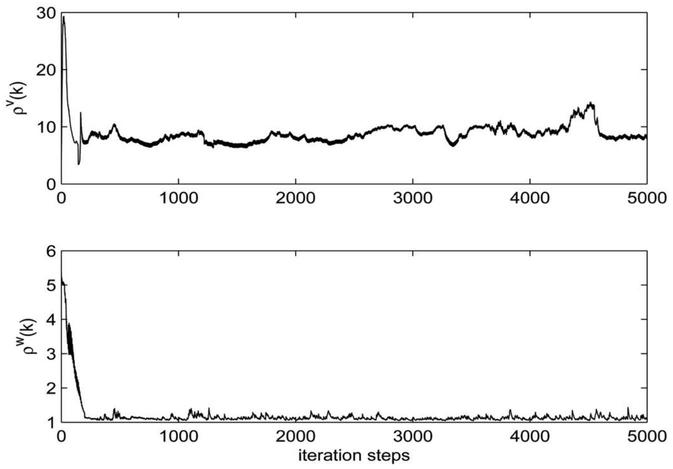


Figure 13. Traces of the normalization factors $\rho^v(k)$ and $\rho^w(k)$

association as well as BPTT algorithm, where the target pattern is a 10×10 binary image as shown in the first picture of Figure 14. The training of RNN is to store the target pattern directly as a local attractor, i.e., an equilibrium point of RNN. Since the state vector is 100 dimensional (number of pixels in target pattern) and there are no external inputs, RNN is configured with 100 inputs and outputs. As a matter of fact, this structure is analogous to the conventional Hopfield type network. RNN is utilized as an auto-associator and we aim at studying self-organizing behavior with the RAGD training algorithm. In order to demonstrate the changing of the binary image corresponding to the state of RNN during learning process, a filter layer based on sign function is added to observe the RNN output pattern, which represents the binary image at the iterative instant. The training process of the RAGD is shown in Figure 14. As mentioned, we also implement the BPTT algorithm to

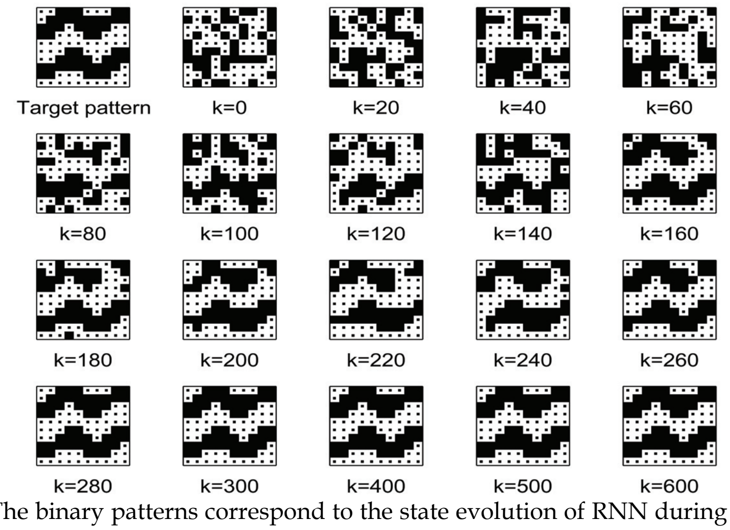


Figure 14. The binary patterns correspond to the state evolution of RNN during the training process using the RAGD algorithm.

provide comparison. The learning rate for the BPTT is 0.028. Similar to previous sections, this value is obtained by trial-and-error tuning method without violating stability constraint. The changing process of the binary image corresponding to the state vector of RNN is shown in Figure 15.

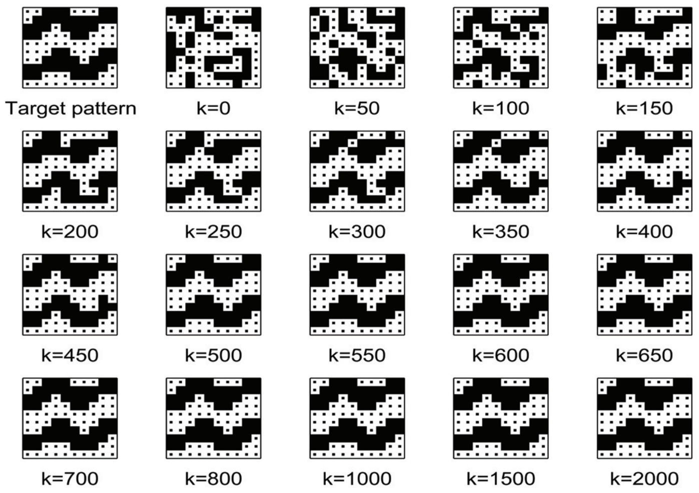


Figure 15. The binary patterns correspond to the state evolution of RNN during the training process using the BPTT algorithm.

From Figure 14 and 15, we see that using the RAGD training method, the dynamic learning process is completed within 300 steps, which is superior to the 500 steps of the BPTT algorithm. Further, we provide the squared error during the dynamic learning process of the RAGD and BPTT in Figure 16. The results indicate that the convergent process of the BPTT (about 450 iterations) is longer than the RAGD (about 280 iterations).

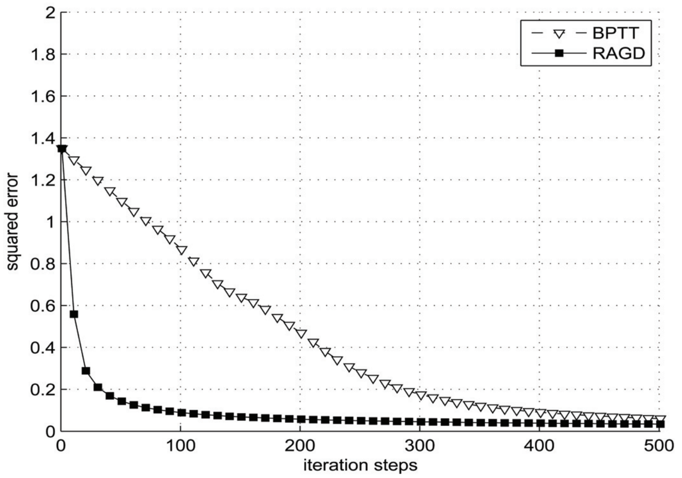


Figure 16. Comparison of the squared error curves between the RAGD and BPTT training procedures.

With these training results, we evaluate the association performance upon a distorted test pattern. The target image pattern is assumed to be disturbed by a white Gaussian noise with the noise level about 40% pixels, as shown in the first picture of Figure 17. This image is utilized as initial state of RNN to test the capability of recalling the associative memory. The recovered binary images at each time instant during recalling procedure of the two RNN trained by the RAGD and BPTT are given in Figure 17 and 18 respectively. The results show

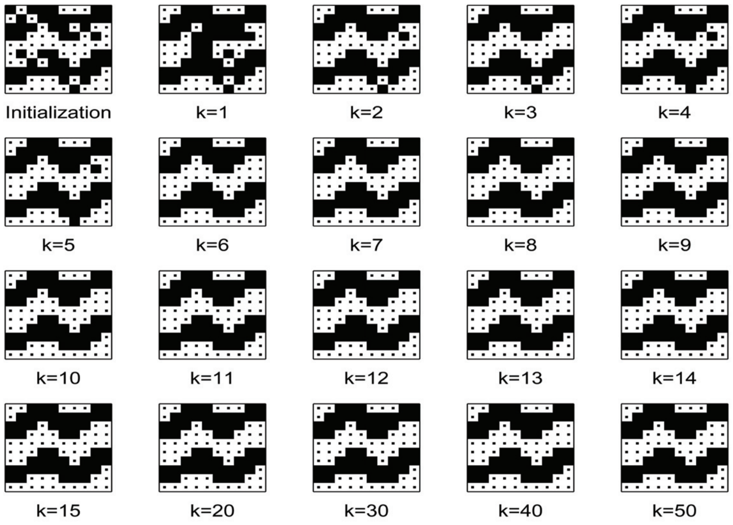


Figure 17. The binary patterns correspond to the state evolution of association process of RNN trained b the BPTT algorithm.

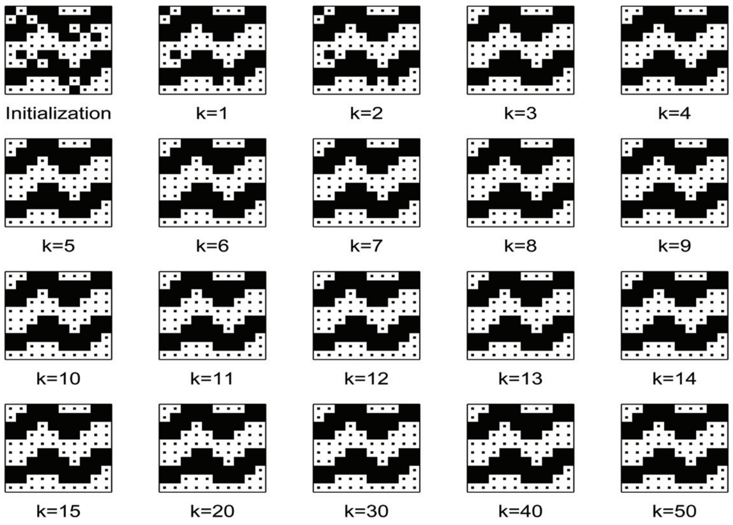


Figure 18. The binary patterns correspond to the state evolution of association process of RNN trained b the RAGD algorithm.

that the 10×10 binary pattern is successfully stored as a stable equilibrium point of the RNN by both algorithms. And there is no obvious difference of recall duration between two schemes (both within 10 iterations).

4.4 Summary

We have presented quantitative studies of the proposed RAGD algorithm in this section. Computer simulations are synthesized to justify the effectiveness of the RAGD. We give three examples which are the most frequent application areas of RNN: i) One-step prediction of non-statistical time series, which is generated by benchmark process model; ii) Identification of a nonlinear dynamic plant and the training data set is generated by a time-varying Hammerstein-Wiener model; iii) Pattern association of binary images. Further, we provide comprehensive comparisons between the RAGD and various other algorithms such as the N-RTRL, the T-BPTT, and the BPTT. In most results of these simulations, RNN trained by the RAGD demonstrates explicit advantages in the transient response speed, e.g., see Figure 6, 11 and 16. Some of the results also indicates that the RAGD can achieve better steady state responses, such as those in Figure 7 and 12. Hence by these experiment results, we conclude that the performance of the RAGD training algorithm of RNN is improved.

5. Conclusion

In this chapter, a Robust Adaptive Gradient Descent training algorithm of RNN with improved convergence speed is investigated. The major feature of the RAGD is the three adaptive parameters that switch the training patterns in a hybrid learning mode. Weight convergence and robust stability of the algorithm are analyzed via Lyapunov and input-output systematic approach respectively. We show how the training algorithm can be decomposed into a nonlinear feedforward operator H_1 and a linear feedback operator H_2, and thus form a closed loop (H_1, H_2). Then, by restricting the cone conditions of each operator, sufficient boundary conditions of L_2 stability of the training are obtained. In addition, we obtain the knowledge in which way we can adaptively change the learning rates of gradient training algorithms, or equivalently re-scale the corresponding error derivatives under stability preservation, such that the learning is ensured to be within the stable range. Such techniques are specially important for deriving a fast transient response. Another important contribution of this work lies in that we obtain a unified framework for the analysis of training algorithms of RNN by taking this systematic approach. Such an approach avoids the direct analysis of nonlinear functions in the feedforward path by applying the sector conditions. Computer simulations are also synthesized to justify the effectiveness of the RAGD. We give three examples which are most frequent application areas of RNN. The evaluation results indicate that with the proposed adaptive training parameters, the RAGD can obtain better transient and steady state responses than that of the conventional algorithms such as the BPTT, the RTRL, and the N-RTRL etc.

6. References

D. P. Mandic and J. A. Chambers, Recurrent Neural Networks for Prediction: Learning Algorithms, Architecture and Stability. *Chichester: John Wiley & Sons, 2001.*

S. Haykin, Neural Networks. *Upper saddle River, NJ: Prentice-Hall, 1999.*

M. Anthony and P. Bartlett, Neural Network Learning: Theoretical Foundations. *Cambridge, MA: Cambridge University Press, 1999.*

Y. L. Wu, Q. Song, and X. L. Yang, "Robust recurrent neural network control of biped robot," J. Intell. Robot. Syst.*, vol. 49, pp. 151-169, Jun. 2007.*

B. A. Pearlmutter, "Gradient calculations for dynamic recurrent neural networks: a survey," IEEE Trans. Neural Networks*, vol. 6, no. 5, pp. 1212-1228, 1995.*

M. Rupp and A. H. Sayed, "A time-domain feedback analysis of filtered-error adaptive gradient algorithms," IEEE Trans. Signal Process.*, vol. 44, no. 6, pp. 142-1439, 1996.*

M. Rupp and A. H. Sayed, "Supervised learning of perceptron and output feedback dynamic networks: a feedback analysis via the small gain theorem," IEEE Trans. Neural Networks, *vol. 8, no. 3, pp. 612-622, 1997.*

J. Liang and M. M. Gupta,"Stable dynamic backpropagation learning in recurrent neural networks," IEEE Trans. Neural Networks*, vol. 10, no. 6, pp. 1321-1334, 1999.*

A. F. Atiya and A. G. Parlos, "New results on recurrent network training: unifying the algorithms and accelerating convergence," IEEE Trans. Neural Networks*, vol. 11, pp. 697-709, May 2000.*

Q. Song, J. Xiao, and Y. C. Soh, "Robust backpropagation training algorithm for multi-layered neural tracking controller," IEEE Trans. Neural Networks*, vol. 10, no. 5, pp. 1133-1141, 1999.*

Q. Song, J. C. Spall, and Y. C. Soh, "Robust neural network tracking controller using simultaneous perturbation stochastic approximation," in Proc. IEEE Conf. Dec. Cont.*, vol. 42, pp. 6194-6199, Dec. 2003.*

Y. L. Wu, Q. Song, and S. Liu, "A normalized adaptive training of recurrent neural networks with augmented error gradient," IEEE Trans. Neural Networks, vol. 19, pp. 351-356, Feb. 2008.

D. P. Mandic, "Data-reusing recurrent neural adaptive filters," *Neural Comput.*, vol. 14, pp. 2693-2707, 2002.

R. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Comput.*, vol. 1, pp. 270-280, 1989.

V. R. Cluett, L. Shah, and G. Fisher, \Robustness analysis of discrete-time adaptive contro systems using input-output stability theory: a tutorial," in *IEE Proc. Part-D*, vol. 135, pp. 133- 141, Mar. 1988.

O. Nelles, *Nonlinear System Identification: From Classic Approaches to Neural Network and Fuzzy Models*. Berlin: Springer-Velag, 2001.

D. Rumelhart, G. E. Hinton, and R. Williams, "Learning internal representation by error propagation," *Parall. Distr. Process.*, vol. 1, pp. 714-728, 1986.

P. J. Werbos, "Generalisation of backpropagation with application to a recurrent gas market model," *Neural Networks*, vol. 1, no. 1, pp. 339-356, 1988.

A. E. Nordsjo and L. H. Zetterberg, "Identification of certain time-varying nonlinear wiener and hammerstein systems," *IEEE Trans. Signal Process.*, vol. 49, pp. 577-592, Mar. 2001.

Y. L. Wu and Q. Song, "A frequency domain analysis for incremental gain of chaotic recurrent neural network," in *Proc. Int. Joint Conf. Neural Networks*, (Vancouver, BC, Canada), Jul. 2006.

Application of Recurrent Neural Networks to Rainfall-runoff Processes

Tsung-yi Pan, Ru-yih Wang, Jihn-sung Lai and Hwa-lung Yu
National Taiwan University
Taiwan

1. Introduction

Knowledge of the hydrological process is essential to the watershed and flood management. Due to the complexity of the interactions among the hydrological process, hydormeterological and geomorphological processes, a rigorous dynamic system model is required for the modelling purpose. Among them, the rainfall-runoff modelling is always considered as one of the most challenging part of hydrological process modelling. It has been shown in a variety of research fields that the application of recurrent neural network (RNN) can perform superior in dynamic system modelling (Pan and Wang, 2005). However, Maier and Dandy (2000) reviewed 43 hydrology journal articles with modelling of artificial neural networks (ANNs) published before 1998, where only Chow and Cho (1997) applied RNNs to forecast rainfall.

The application of RNNs to hydrological modelling is rapidly growing these years. Published between 2000 and 2008 spring, 14 papers in which RNNs have been used for simulation or forecasting of water resources variables are reviewed in terms of the modelling process. Due to the rapid increase in journals, it is unlikely that complete coverage has been achieved. Following the form of Maier and Dandy (2000), the major features of the models investigated are summarised in Tables 1 and 2, including background information (variable modelled, location of application, model time step, and forecast length), information about the data used (data type, normalization range, number of training samples, and number of testing samples), information about network architecture (connection type, method used to obtain optimal network geometry, and number of nodes per layer), information about the optimization algorithm used (optimization method, internal network parameters (hidden layer transfer function, learning rate, momentum value, epoch size, and initial weight distribution range)) and the stopping criterion adopted. While hydrologists have not made an effort to construe the knowledge embedded in the trained RNN models, the recent studies strive to interpret physical significance from the internal architecture of RNN hydrological models, like Pan et al. (2004, 2005, and 2007). Therefore, this chapter will introduce the deterministic linearized recurrent neural network (denoted as DLRNN) and its application to rainfall-runoff processes.

Background information					Data						
Ref.	Author(s) and year	Input variable	Output variable	Location(s)	Time step	Forecast length	Data type	Normali-zation range	No. train. Samples	No. test samples	Software
1	Coulibaly et al., 2000	Monthly mean flow, climatic indices	Annual flow	North-eastern Canada	Year	+1~11	Real	?	36 years	11 years	?
2	Walter et al., 2001	Phosphorus, ammonia, nitrogen, water temperature, solar radiation, volume, area, water depth	Chlorophyll-a	Burrinjuck Reservoir (Australia)	Day	+7	Real	0~1	15 years	3 years	?
3	Chang et at., 2004	Precipitation, flow	Flow	Da-chia River (Taiwan)	Hour	+1~2	Real	?	1 year	6 years	MATLAB
4	Chiang et al., 2007b	Precipitation	Flow	Keelung River (Taiwan)	Hour	+1	Real	?	13 typhoons	2 typhoons	?
5	Nagsh Kumar et al., 2004	Flow	Flow	India	Month	+1~12	Real/Synthetic	0~1	50 years	7 years	?
6	Pan and Wang, 2004	Precipitation	Flow	Keelung River (Taiwan)	Hour	+1~3	Real	0~0.9	10 events	37 events	MATLAB
7	Pan and Wang, 2005	Precipitation	Flow	Keelung River (Taiwan)	Hour	0	Real	0~0.9	10 events	31 events	MATLAB
8	Pan et al., 2007	Precipitation	Flow	Keelung River (Taiwan)	Hour	0	Real	0~0.9	10 events	28 events	MATLAB
9	Coulibaly & Baldwin, 2005	Flow, lake volume	Flow, lake volume	Nile River (Egypt), Saint-lawrence River (British), Great Salt Lake (U.S.A)	Month	+12	Real	?	1995-1999 for SLR & GSL, 1985-1989 for Nile River	1861-1994 (SLR), 1847-1994 (GSL), 1871-1984 (Nile River)	MATLAB
10	Chiang et al., 2007a	Precipitation	Precipitation	Keelung River (Taiwan)	Hour	+1	Real	?	4 typhoons	2 typhoons	?
11	Karamouz et al.,	Climate signals	Precipitation	Iran	Season	+2	Real	-1~1	22 years	10 years	?
12	Chiang et al., 2004	Flow	Precipitation, flow	Lan-yang River (Taiwan)	Hour	+1	Real	?	15 events	3 events	?
13	Coulibaly and Evora, 2007	Precipitation, max and min temperature	Precipitation, max and min temperature	Gatineau river basin (Canada)	Day	0	Real	?	3660 for precipitation	915 for precipitation	Neural Solutions v4
									4285 for max temperature	880 for max temperature	
									2683 for min temperature	671 for min temperature	
14	Coulibaly et al., 2001	Water table depth, precipitation, mouboun water level, max temperature, mean	Water table depth	Northwestern Burkina Faso (West Africa)	Month	+1	Real	?	7 years	3 yeaers	MATLAB

?, no specified ; SLR, Saint-lawrence River; GSL, Great Salt Lake; NeuralSolutions v4 (NeuroDimension Inc., Gainesville, FL); MATLAB (The Mathworks Inc., Natick, MA).

Table 1. Details of papers reviewed (background information and data).

Network architecture				Optimization algorithm						
Ref.[a]	Connect. Type	Geometry method	Optimum I-H1-H2-O	Optim. Method	Transfer function	Learn. Rate	Momen-tum	Epoch size	Initial weights	Stopping criterion
1	RNN	T&E	Problem dependent	LMBP	?	?	?	?	?	VE
2	ANNA	T&E	8-8-1	IRBP	Sigmoid	?	0.7	16	?	VE
3	RNN	T&E	5-5-1	RTRL	?	40, 80	?	?	?	?
4	RNN	T&E	2-1-3-1	RTRL	?	?	?	?	?	?
5	Context model	T&E	5-10-10-1	OPD	Sigmoid	0.9	0	All	Random	Training iteration
6	SSNN	T&E	1-6-1	MGL	Linear	?	0	?	Random	VE
7	RNN	ISI	1-6-1	MGL	Linear	?	0	?	Random	VE
8	DLRNN	MSI	1-6-1	MGL	Linear	?	0	?	Random	VE
9	CRNN	T&E	1-9-1	LMBP, BPTT, BR	Logistic	?	?	?	?	VE
10	RNN	T&E	12-5-1	RTRL	?	?	?	?	?	?
11	TDRNN	T&E	4-4-1	Adaptive training	Sigmoid	?	?	?	-0.5~0.5	VE
12	RNN	T&E	5-3-1	RTRL	?	?	?	?	?	?
13	RNN	T&E	4-20-1	BPTT	Logistic	?	?	?	?	VE
	TDRNN	T&E	20-20-1	BPTT	Logistic	?	?	?	?	VE
14	RNN	T&E	5-3-1	BPTT	Logistic	?	?	?	?	TE

[a]Refer to Table 1 for details of reference.
?, not specified; RNN, recurrent nerual network; ANNA, Artificial Neural Network model for predicting species abundance and succession of blue-green Algae; BPTT, backpropagation through time; BR, Bayesian regularization; CRNN, competitive recurrent neural network; IRBP, internally recurrent backpropagation; ISI, indirect system identification; LMBP, Levenberg-Marquardt backpropagation; MGL, modified gradient learning; MSI, modified system identification; OPD, ordered partial derivatives; RTRL, real-time recurrent learning; SSNN, state space neural network; TDRNN, time delay recurrent neural network; T&E, trial and error; TE, training error; VE, validation error.

Table 2. Details of papers reviewed (RNN architecture and optimization).

2. Deterministic linearized recurrent neural network

The RNN introduced in this chapter is to integrate a state space form into the neural network framework. The integration can provide not only the flexibility to represent any nonlinear functions but also the parallel inputs/outputs (causes/effects) relationships established between the neural model and the physical system (Pan & Wang, 2004). The presented RNN has five layers: input layer, hidden layer S, state layer, hidden layer O, and output layer. The input layer takes the input signals and delivers these inputs to every neuron in the next layer, hidden layer S, which represents any function that specifies the behaviour of states. State layer receives the signals from hidden layer S, and each neuron in this layer represents one state whose output value is the value of the state. After hidden layer O, which represents the features that relates the outputs of the neural network to the states, gets the signals from state layer, output layer takes the hidden layer O signals adds them to each output neuron. These outputs are, finally, the outputs of the RNN embedded in a state space form as Fig. 1.
The mathematical representation of a deterministic non-linear system in state space form is:

$$x_{k+1} = F(x_k, u_k) \tag{1}$$

$$y_k = G(x_k) \tag{2}$$

where u_k, y_k, and x_k with m, l, and n ranks denote, respectively, the input, output, and state vectors at time k. $F: n \times m \to n$ and $G: n \to l$ are two static linear/nonlinear mappings.

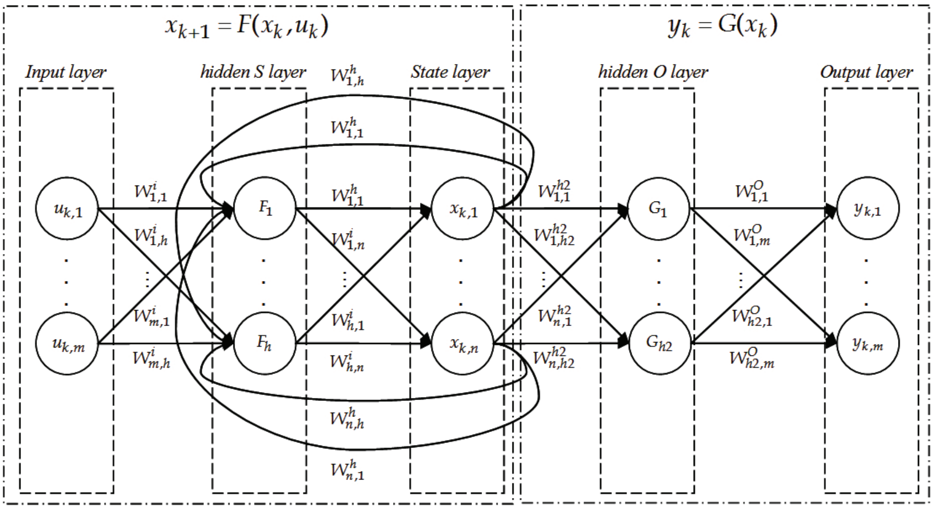


Fig. 1. The RNN embedded in a state space form.

A neural network containing a single hidden layer with bounded transfer functions in its neurons can be used for the representation of a variety of linear/nonlinear functions (Zarmarreño et al., 2000). Therefore, to apply the neural network for the linear/nonlinear mappings in Eqs. (1) and (2), the mathematical form of this special RNN can be written as:

$$\hat{x}_{k+1} = W^h \cdot f_1(W^r \cdot x_k + W^i \cdot u_k + B^h) \tag{3}$$

$$\hat{y}_k = W^o \cdot f_2(W^{h2} \cdot x_k + B^{h2}) \tag{4}$$

where W^h, W^r, W^i, W^o, and W^{h2} are matrices with dimensions $n \times h$, $h \times n$, $h \times m$, $m \times h2$, and $h2 \times n$ as the weights of the RNN, respectively. B^h and B^{h2} are two vectors with h and $h2$ elements as biases. f_1 and f_2 are linear/nonlinear functions depending on the behaviour of the system.

Previous works have established that linearized neural networks suffice to capture nonlinear systems. Botto and Costa (1998) designed a linear predictive control using a linearized neural network model. Henriques and Kuanyi (1998) stated that control design for linear systems has been well developed, and it is natural to make use of it in nonlinear plants. Hence, they applied as a linearized neural model. Furthermore, Rahman and Kuanyi (2000) studied a neural network method to linearizing control of nonlinear process plants, and used neural networks to model the process plant and to linearize the neural network model in a novel way. Additionally, the difference between a RNN and a linearized one is the linearity of the active function of each neuron in the hidden layer. In fact, however, it is not strictly necessary that a neural interpretation of the neuron contains a non-linear function because the reduction of the diversity of activation functions, such as the sigmoid function, is beneficial (Ptitchkin, 2001). Although neural networks are known to be universal function approximators, except for unchanged the active functions, the weights and structure of the neural network are updated or modified during the entire approximating process. Moreover, a high-dimensional space nonlinearity problem can be suitably approximated by modifying the weights in the linear combinations of state variables with time. Consequently, the linear transfer function of the RNN applied herein is capable of simulating nonlinear rainfall-runoff process.

Considering the transfer functions of the RNN applied herein are set as linear functions and the biases are set at zero. Consequently, the Eqs. (3) and (4) are rewritten as:

$$\hat{x}_{k+1} = W^h \cdot \left(W^r \cdot x_k + W^i \cdot u_k\right)$$

$$= \left(W^h \cdot W^r\right) \cdot x_k + \left(W^h \cdot W^i\right) \cdot u_k = W_1 \cdot x_k + W_2 \cdot u_k \tag{5}$$

$$\hat{y}_k = W^o \cdot \left(W^{h2} \cdot x_k\right) = \left(W^o \cdot W^{h2}\right) \cdot x_k = W_3 \cdot x_k \tag{6}$$

In the recursive equation (5), W_1, W_2, and W_3 are unknown weights to be identified by observed input/output sequences $\{u_0, u_1, \cdots, u_{N-1}\}$ and $\{y_0, y_1, \cdots, y_{N-1}\}$. By replacing the x_k term in the observed equation (6) with the solved recursive equation (5), the output response of the system is given as:

$$y_k = W_3 W_1^{\,k} x_1 + \sum_{p=2}^{k} W_3 W_1^{\,p-1} W_2 u_{k-p} \tag{7}$$

For a system initially at rest, i.e., $x_1 = 0$, Equation (7) is rewritten as:

$$y_k = \sum_{p=1}^{k} h_p u_{k-p} \tag{8}$$

where the unit hydrograph (UH) of the rainfall-runoff processes can be summarized as:

$$h_p = W_3 W_1^{\,p-1} W_2 \quad \text{if } p \ge 2$$

$$h_p = 0 \qquad \text{if } p = 1 \tag{9}$$

The impulse response terms $W_3 W_1^{\,p-1} W_2$ for $p \ge 2$ are known as the Markov parameters.

3. Calibration algorithm for DLRNN

3.1 Indirect system identification

The concept of indirect system identification algorithms is to obtain the UH ordinates first, called the constrained deconvolution step. The linear programming is selected herein to carry out the UH from the rainfall and runoff data. Then, the system matrices $[W_1, W_2, W_3]$ are identified from the UH ordinates via singular value decomposition (SVD), entitled the realization step.

In the realization step, the state space model can be represented as follows if $\bar{x}_k = T x_k$ for some nonsingular transformation matrix T (Romos et al., 1995):

$$Tx_{k+1} = [TW_1T^{-1}]Tx_k + [TW_2]u_k \tag{10}$$

$$y_k = [W_3T^{-1}]Tx_k \tag{11}$$

By considering $[TW_1T^{-1}]$ as $\overline{W}_1$, $[TW_2]$ as $\overline{W}_2$, and $[W_3T^{-1}]$ as $\overline{W}_3$, the system matrices of the transformed system are now $\left[\overline{W}_1, \overline{W}_2, \overline{W}_3\right]$, and these parameter matrices $\left[\overline{W}_1, \overline{W}_2, \overline{W}_3\right]$ are identified based on the deconvoluted impulse response sequence $\{\hat{h}_p\}$. Specifically, SVD is performed on the following Hankel matrix:

$$H_{t,t} = \begin{bmatrix} \hat{h}_1 & \hat{h}_2 & \hat{h}_3 & \cdots & \hat{h}_k \\ \hat{h}_2 & \hat{h}_3 & \hat{h}_4 & \cdots & \hat{h}_{k+1} \\ \hat{h}_3 & \hat{h}_4 & \hat{h}_5 & \cdots & \hat{h}_{k+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \hat{h}_k & \hat{h}_{k+1} & \hat{h}_{k+2} & \cdots & \hat{h}_{2k-1} \end{bmatrix} = U \cdot S \cdot V^T = \left(U \cdot S^{1/2}\right) \cdot \left(S^{1/2} \cdot V^T\right) = \overline{\mathbf{o}}_k \cdot \overline{\mathbf{c}}_k \tag{12}$$

where $2k - 1 \le M$. M is the memory of system. The transformed parameter matrices are identified from:

$$\overline{W}_1 = TW_1T^{-1} = \overline{\mathbf{c}}_{k,2}\overline{\mathbf{c}}_{k,1}^{*};\ \overline{W}_2 = TW_2 = \overline{\mathbf{c}}_k \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix};\ \overline{W}_3 = W_3T^{-1} = [1, 0, \cdots, 0]\overline{\mathbf{o}}_k$$

where $\overline{\mathbf{c}}_{k,1}$ and $\overline{\mathbf{c}}_{k,2}$ denote the first and last $(k-1)$ columns of $\overline{\mathbf{c}}$, and the star denotes a pseudoinverse.

3.2 Subspace algorithm

Above indirect system identification algorithm computes the weights of a RNN from a Hankel matrix constructed using Markov parameters. However, using the Markov parameters as a starting point would be rather difficult to measure in some fields (Abdelghani & Verhaegen, 1998). The subspace algorithms are the automatic structure identification, and derive the model directly from the input-output data without estimating the Markov parameters as an intermediate step (Gustafsson, 2001; Ramos et al., 1995).

Before description of subspace algorithm, the past and future highly rectangular input/output Hankel matrices, H₁ and H₂ respectively, are defined by input-output data:

$$H_1 = \begin{bmatrix} U_1 \\ \hline Y_1 \end{bmatrix} = \left[\begin{array}{cccc} u_1 & u_2 & \cdots & u_j \\ u_2 & u_3 & \cdots & u_{j+1} \\ \vdots & \vdots & \ddots & \vdots \\ u_i & u_{i+1} & \cdots & u_{i+j-1} \\ \hline y_1 & y_2 & \cdots & y_j \\ y_2 & y_3 & \cdots & y_{j+1} \\ \vdots & \vdots & \ddots & \vdots \\ y_i & y_{i+1} & \cdots & y_{i+j-1} \end{array}\right] \text{ for j>>i>n} \tag{13}$$

$$H_2 = \left[\frac{U_2}{Y_2}\right] = \left[\begin{array}{cccc} u_{i+1} & u_{i+2} & \cdots & u_{i+j} \\ u_{i+2} & u_{i+3} & \cdots & u_{i+j+1} \\ \vdots & \vdots & \ddots & \vdots \\ u_{2i} & u_{2i+1} & \cdots & u_{2i+j-1} \\ \hline y_{i+1} & y_{i+2} & \cdots & y_{i+j} \\ y_{i+2} & y_{i+3} & \cdots & y_{i+j+1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{2i} & y_{2i+1} & \cdots & y_{2i+j-1} \end{array}\right] \quad \text{for j>>i>n} \tag{14}$$

Two state vector sequences X_1 and X_2 are defined as $X_1 = [x_1 \,|\, x_2 \,|\cdots|\, x_j]$ and $X_2 = [x_{i+1} \,|\, x_{i+2} \,|\cdots|\, x_{i+j}]$. The subspace algorithm is presented as follows:

a) Compute the SVD of the concatenation of H_1 and H_2:

$$\left[\frac{H_1}{H_2}\right] = U_H \Sigma_H V_H^T = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix} \cdot \begin{bmatrix} \Sigma_{11} & 0_{(2mi+n)\times j} \\ 0_{(2li-n)\times(2mi+n)} & 0_{(2li-n)\times j} \end{bmatrix} \cdot V_H^T \tag{15}$$

where u_{11}, u_{12}, u_{21}, u_{22}, Σ_{11}, and V_H are the matrices with dimensions $(mi+li)\times(2mi+n)$, $(mi+li)\times(2li-n)$, $(mi+li)\times(2mi+n)$, $(mi+li)\times(2li-n)$, $(2mi+n)\times(2mi+n)$, and $j\times j$ respectively.

b) Compute the SVD of $u_{12}^T u_{11} \Sigma_{11}$ in order to determine the system order, n:

$$u_{12}^T u_{11} \Sigma_{11} = \left[U_q \,|\, U_q^{\perp}\right] \cdot \begin{bmatrix} \Sigma_q & 0_{n\times(2mi)} \\ 0_{2(li-n)\times n} & 0_{2(li-n)\times(2mi)} \end{bmatrix} \cdot V_q^T \tag{16}$$

where U_q, $U_q^{\perp}$, Σ_q, and V_q are the matrices with dimensions $(2li-n)\times n$, $(2li-n)\times 2(li-n)$, $n\times n$, and $(2mi+n)\times(2mi+n)$ respectively.

c) Compute the transformed state vector sequence:

$$\overline{X}_2 = U_q^T u_{12}^T H_1 = [\overline{x}_{i+1} \,|\, \overline{x}_{i+2} \,|\cdots|\, \overline{x}_{i+j}] \tag{17}$$

where $\overline{X}_2$ is the matrix with dimensions $n\times j$.

d) Compute the weights of the RNN by solving the overdetermined system of equations:

$$\begin{bmatrix} \overline{x}_{i+2} & \overline{x}_{i+3} & \cdots & \overline{x}_{i+j} \\ y_{i+1} & y_{i+2} & \cdots & y_{i+j-1} \end{bmatrix} = \begin{bmatrix} \overline{W}_1 & \overline{W}_2 \\ \overline{W}_3 & 0 \end{bmatrix} \cdot \begin{bmatrix} \overline{x}_{i+1} & \overline{x}_{i+2} & \cdots & \overline{x}_{i+j-1} \\ u_{i+1} & u_{i+2} & \cdots & u_{i+j-1} \end{bmatrix} \tag{18}$$

In the past few years, much attention has been paid recently to subspace algorithms when various time domain methods for identifying dynamic models of systems from modal experimental data appeared. However, this algorithm was seldom applied in the scope of hydrology. Except Ramos et al. (1995), they used one event of 29 data points (each 30 minutes long) and 365 daily data to evaluate the algorithm. To compare with daily data, hourly data used herein have more uncertainty and noisy. The suitability of subspace algorithm with hourly rainfall-runoff data, therefore, is re-evaluated based on a real typhoon event of the Keelung River in Taiwan as follows:

Firstly, a sequence of 100 data is generated from a state space model that was identified from rainfall-runoff data observed on Sep. 27, 1996. Indirect system identification algorithm was used to check if the subspace algorithm could identify the original system. The state space model is a 3-order system as following equations:

$$X_{k+1} = \begin{bmatrix} 0.9600 & -0.1087 & -0.0383 \\ 0.1087 & 0.8333 & -0.3317 \\ -0.0383 & 0.3316 & 0.6772 \end{bmatrix} \cdot X_k + \begin{bmatrix} -0.2274 \\ 0.2144 \\ -0.1485 \end{bmatrix} \cdot U_k \tag{19}$$

$$Y_k = \begin{bmatrix} -0.2274 & -0.2144 & -0.1485 \end{bmatrix} \cdot X_k \tag{20}$$

The generated sequence was identified as Equations (21) and (22). The results show that the system order determination in the step 2 of subspace algorithm is correct so that the impulse response can be simulated accurately.

$$X_{k+1} = \begin{bmatrix} 0.9559 & -0.1857 & -0.0944 \\ 0.0693 & 0.8517 & 0.5704 \\ 0.0237 & -0.1880 & 0.6628 \end{bmatrix} \cdot X_k + \begin{bmatrix} -0.1190 \\ 0.0716 \\ 0.0386 \end{bmatrix} \cdot U_k \tag{21}$$

$$Y_k = \begin{bmatrix} -0.4403 & -0.6683 & 0.6034 \end{bmatrix} \cdot X_k \tag{22}$$

The second test used the original observed data to identify a rainfall-runoff system. However, according to the identified UHs shown in Fig. 2, the subspace algorithm performed poorly because it was very sensitive to the noise in observed data. Therefore, the modified system identification combined with indirect system identification and subspace algorithm is introduced.

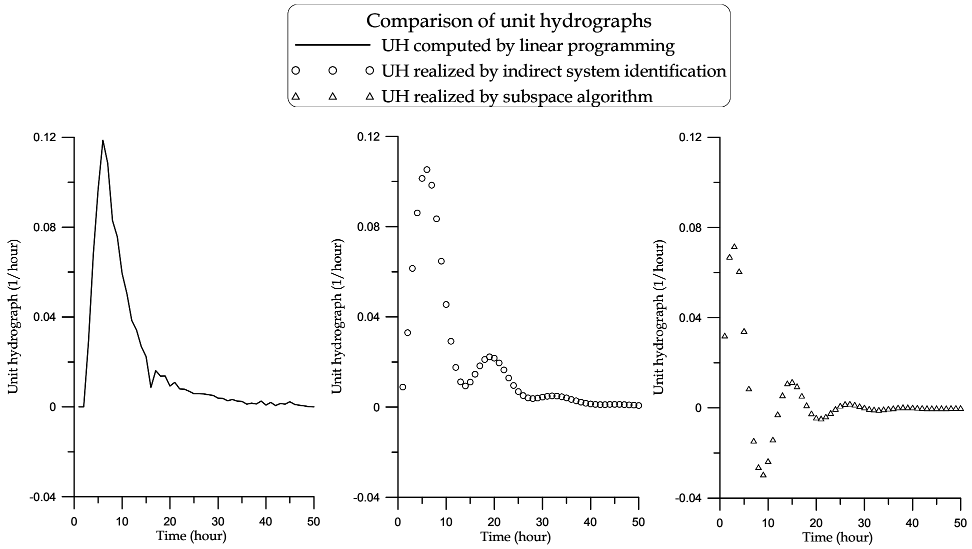


Fig. 2. UHs carried out via linear programming, indirect system identification, and subspace algorithm.

3.3 Modified system identification for hydrology

Figure 3 and the left part of Fig. 4 are the flowcharts of indirect system identification algorithm and subspace algorithm respectively. To compare with these two flowcharts, indirect system identification algorithm needs to subjectively decide the system order from a sequence of singular values in Equation (16). In practice, the singular values are not easily classified into significant and insignificant groups when the singular values descend slowly. Additionally, subspace algorithm can determine the system order objectively, but it is sensitive. Therefore, the constrained deconvolution step is considered, firstly, to compute a discrete UH from rainfall-runoff events for calibration. Secondly, a sequence of rainfall-runoff data generated form the discrete UH via convolution is synthesized. This synthesized data are without noise that helps subspace algorithm to get the system order. The right part of Fig. 4 surrounded by dotted line is the modified system identification for hydrology.

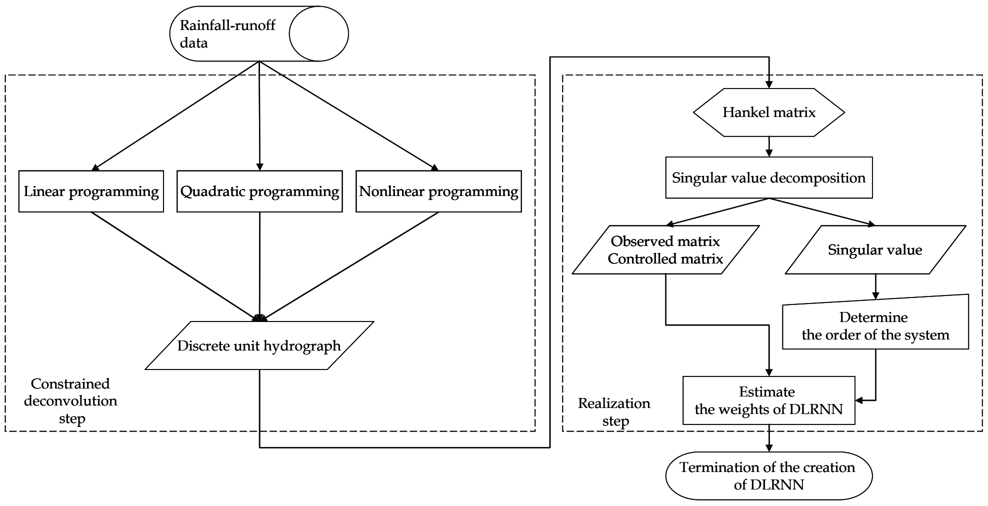

Rainfall-runoff data
Linear programming
Quadratic programming
Nonlinear programming
Discrete unit hydrograph
Constrained deconvolution step
Hankel matrix
Singular value decomposition
Observed matrix Controlled matrix
Singular value
Determine the order of the system
Estimate the weights of DLRNN
Realization step
Termination of the creation of DLRNN

Fig. 3. Flowchart of indirect system identification.

4. On-line learning algorithm for DLRNN

Dynamic RNN learning algorithms can be grouped into five major categories (Parlos et al., 2000), such as (1) the real time recurrent learning; (2) the backpropagation through time (BTT) method; (3) the fast forward propagation method; (4) the Green's function method; and (5) the block update method. All training algorithms above are gradient-based by which the learning trajectory is represented into the changes of weights of neurons.

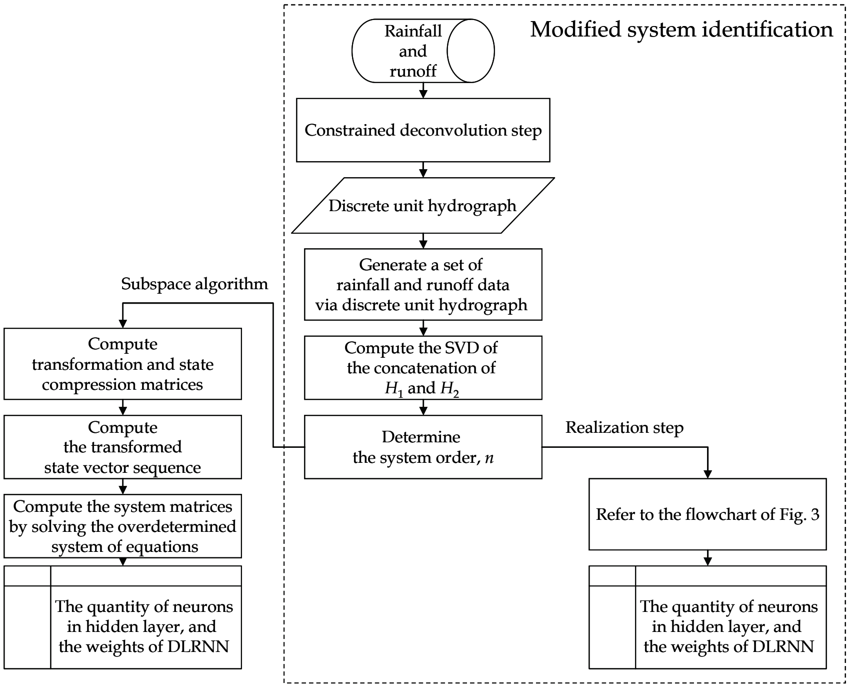
The weights updated via gradient-based learning algorithms can be written as:

$$W_{new} = W_{old} - \eta \frac{dE}{dW} \tag{23}$$

where η denotes the learning rate, and E is the sum of square errors.

$$E = \frac{1}{2}\sum_{k=1}^{K}(y_k - d_k)^T(y_k - d_k) \tag{24}$$

where y_k is the output of the model, and d_k represents the desired output at time index k. The algorithm introduced herein is based on the gradient-based learning method developed by Atiya and Parlos (2000).



Modified system identification
Rainfall and runoff
Constrained deconvolution step
Discrete unit hydrograph
Generate a set of rainfall and runoff data via discrete unit hydrograph
Compute the SVD of the concatenation of H_1 and H_2
Determine the system order, n
Subspace algorithm
Compute transformation and state compression matrices
Compute the transformed state vector sequence
Compute the system matrices by solving the overdetermined system of equations
The quantity of neurons in hidden layer, and the weights of DLRNN
Realization step
Refer to the flowchart of Fig. 3
The quantity of neurons in hidden layer, and the weights of DLRNN

Fig. 4. Flowchart of modified system identification.

4.1 DLRNN learning algorithm

The idea of the algorithm adopted herein is to obtain an approximation for the gradient that can be efficiently computed via the interchange of the roles of the network states x_k and the weight matrix W. Let the states be considered as the control variables, and the change in the weights is determined upon the changes in x_k. The details of the algorithm are as follows: First, the network learning is formulated as constrained minimization problem, with the objective to minimize the sum of square error, E, given by Equation (24), and the constraints.

$$g_{k+1} \equiv W_1 \cdot x_k + W_2 \cdot u_k - \hat{x}_{k+1} = 0, \quad k = 0, \cdots, K-1 \tag{25}$$

According to the Equations (10) and (11), the error gradient can be written as follows:

$$\frac{dE}{dw_1} = \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w_1} \tag{26a}$$

$$\frac{dE}{dw_2} = \frac{\partial E}{\partial w_2} + \frac{\partial E}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w_2} \tag{26b}$$

$$\frac{dE}{dw_3} = \frac{\partial E}{\partial w_3} + \frac{\partial E}{\partial y}\frac{\partial y}{\partial w_3} \tag{26c}$$

where $\frac{dE}{dw_n}$ for n = 1, 2, 3 equals 0 since E is the function of y. Consequently, the updated weights of W_2 and W_3 for time K can be derived ffom Equation (23), (24), (26b), and (26c) as follows:

$$W_{2,K+1} = W_{2,K} - \eta \cdot (y_K - d_K) \cdot W_{3,K} \cdot u_K \tag{27}$$

$$W_{3,K+1} = W_{3,K} - \eta \cdot (y_K - d_K) \cdot x_K \tag{28}$$

By taking the derivative of the Equation (25), one can get:

$$\frac{\partial g}{\partial w_1} + \frac{\partial g}{\partial x}\frac{\partial x}{\partial w_1} = 0 \Rightarrow \frac{\partial x}{\partial w_1} = -\left(\frac{\partial g}{\partial x}\right)^{-1}\frac{\partial g}{\partial w_1} \tag{29}$$

Solving Equations (26a) and (29), one can get:

$$\frac{dE}{dw_1} = -\frac{\partial E}{\partial y}\frac{\partial y}{\partial x}\left(\frac{\partial g}{\partial x}\right)^{-1}\frac{\partial g}{\partial w_1} \tag{30}$$

According to the convention that $(\partial u/\partial v)$ for two vectors u and v is the matrix whose (i, j)th element is $(\partial u_i/\partial v_j)$, the matrices in (28) can be evaluated from Equations (6) and (24) as follows:

$$\frac{\partial E}{\partial y}\frac{\partial y}{\partial x} = (e_1, e_2, \cdots, e_K)^T \cdot W_3 = W_3^T \cdot (e_1, e_2, \cdots, e_K) \tag{31}$$

where e_k is the error at time k: $e_k = y_k - d_k$,

$$\frac{\partial g}{\partial x} = \begin{bmatrix} -I & 0 & 0 & \cdots & 0 & 0 \\ W_1 & -I & 0 & \cdots & 0 & 0 \\ 0 & W_1 & -I & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & W_1 & -I \end{bmatrix} \tag{32}$$

and

$$\frac{\partial g}{\partial w} = \begin{bmatrix} X_0 \\ X_1 \\ \vdots \\ X_{K-1} \end{bmatrix} \tag{33}$$

where

$$X_k = \begin{bmatrix} x_k^T & 0 & 0 & \cdots & 0 \\ 0 & x_k^T & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & x_k^T \end{bmatrix} \tag{34}$$

I is the identity matrix, and 0 in Equations (32) and (34) is a matrix (or vector) of zeros. After calculating the gradient of *E* with respect to the states x_k, a small change at the states x_k in the negative direction of that gradient can be written as:

$$\Delta x = -\eta\left(\frac{\partial E}{\partial x}\right) \tag{35}$$

Replace $\frac{\partial E}{\partial x}$ by Equation (29), and Equation (33) can be rewritten as:

$$\Delta x = -\eta e = -\eta W_3^T\left(e_1, \cdots, e_K\right) \tag{36}$$

Since g, given by Equation (25), equals zero, one can get:

$$\frac{\partial g}{\partial w_1}\Delta w_1 + \frac{\partial g}{\partial x}\Delta x = 0 \tag{37}$$

After applying the transposition and the pseudoinverse in Equation (37), the change in weights can be determined as:

$$\Delta w_1 = -\left[\left(\frac{\partial g}{\partial w_1}\right)^T\left(\frac{\partial g}{\partial w_1}\right)\right]^{-1}\left(\frac{\partial g}{\partial w_1}\right)^T\frac{\partial g}{\partial x}\Delta x_1 \tag{38}$$

where

$$\left[\left(\frac{\partial g}{\partial w_1}\right)^T\left(\frac{\partial g}{\partial w_1}\right)\right]^{-1} = \begin{bmatrix} \sum_{k=0}^{K-1} x_k x_k^T & 0 & \cdots & 0 \\ 0 & \sum_{k=0}^{K-1} x_k x_k^T & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sum_{k=0}^{K-1} x_k x_k^T \end{bmatrix}^{-1}$$

$$= \begin{bmatrix} \left[\sum_{k=0}^{K-1} x_k x_k^T\right]^{-1} & 0 & \cdots & 0 \\ 0 & \left[\sum_{k=0}^{K-1} x_k x_k^T\right]^{-1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \left[\sum_{k=0}^{K-1} x_k x_k^T\right]^{-1} \end{bmatrix} \tag{39}$$

From Equation (36), let

$$\gamma = \frac{\partial g}{\partial x} e^T = \frac{-1}{\eta} \frac{\partial g}{\partial x} \Delta x \tag{40}$$

and partition the vector γ into the K vectors as follows:

$$\gamma = \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_K \end{pmatrix} \tag{41}$$

Using Eqs. (32) and (40), γ can be evaluated by following recursions:

$$\gamma_1 = -e_1^T \tag{42a}$$

$$\gamma_2 = -e_2^T + W_1 e_1^T \tag{42b}$$

$$\vdots$$

$$\gamma_K = -e_K^T + W_1 e_{K-1}^T \tag{42c}$$

Let

$$V_K' = \sum_{k=0}^{K-1} x_k x_k^T \tag{43}$$

Substituting Equations (33), (39), (40), and (43) into (38), one can get after some manipulation.

$$\Delta W_1 = \eta \left[\sum_{k=1}^{K} \gamma_k x_{k-1}^T \right] \cdot V_K'^{-1} \tag{44}$$

In order to alleviate the effect of most likelihood ill-conditioning problems caused by the matrix inversion in Equation (44), a small matrix εI is added to the outer product matrix V_K' as follows:

$$V_K = \varepsilon I + \sum_{k=0}^{K-1} x_k x_k^T \ , \tag{45}$$

where ε is a small positive constant. Then Equation (44) is rewritten as follows:

$$\Delta W_1 = \eta \left[\sum_{k=1}^{K} \gamma_k x_{k-1}^T \right] \cdot V_K^{-1} \tag{46}$$

Since the passed inputs, state variables, and observed outputs (u_1, x_1, d_1, …, u_{K-1}, x_{K-1}, d_{K-1}) are already available to get $\Delta W_{1,K-1}$, the on-line updated change in weights $\Delta W_{1,K}$ based on a new data point (u_K, x_K, d_K) can be written as follows:

$$\Delta W_{1,K} = \eta \left[\sum_{k=1}^{K-1} \gamma_k x_{k-1}^T + \gamma_K x_{K-1}^T \right] \cdot \left[V_{K-1} + x_{K-1} x_{K-1}^T \right]^{-1} \tag{47}$$

Furthermore, using the small rank adjustment matrix inversion lemma, the inverse of V_K can be obtained recursively in terms of the inverse of V_{K-1} as follows:

$$V_K^{-1} = \left(V_{K-1} + x_{K-1}x_{K-1}^T\right)^{-1} = V_{K-1}^{-1} - \frac{\left(V_{K-1}^{-1}x_{K-1}\right)\left(V_{K-1}^{-1}x_{K-1}\right)^T}{1 + x_{K-1}^T V_{K-1}^{-1} x_{K-1}} \tag{48}$$

and let

$$B_K = \sum_{k=1}^{K} \gamma_k x_{k-1}^T \tag{49}$$

Substituting Eq. (48) into Eq. (47), after simplification one can get the final on-line updated formula of W_1 as follows:

$$\Delta W_{1,K} = \Delta W_{1,K-1} + \eta \frac{\gamma_K x_{K-1}^T V_{K-1}^{-1} - B_{K-1} V_{K-1}^{-1} x_{K-1} \left[V_{K-1}^{-1} x_{K-1}\right]^T}{1 + x_{K-1}^T V_{K-1}^{-1} x_{K-1}} \tag{50}$$

5. Application

5.1 Study area and data pre-processing

With a length of 86 km and an area of 501 km^2, the Keelung River has a U-turn in the northeast Taipei county, and runs through Taipei city, where it joins the Dansuie River and flows out to sea, as shown in Fig. 5. The watershed upstream of Wu-tu with about 204 km^2 surrounding the city of Taipei in northern Taiwan was chosen for evaluating the simulation ability of the DLRNN for recognizing the transition of rainfall-runoff processes. Due to the northeast monsoon in winter and the typhoons in summer, the mean annual precipitation, runoff depth, and runoff coefficient are 2865 mm, 2177 mm, and 0.76, respectively. Owing to the rugged topography of the watershed, large floods caused by the short and steep runoff path-line arrive rapidly in the middle-to-downstream reaches of the watershed, and cause serious damage.

According to the records of three rain gauges (Wu-tu, Jui-fang, and Huo-shao-liao) and on discharge site (Wu-tu) in Wu-tu watershed, as shown in Fig. 5, 38 rainfall-runoff events from 1966 to 1997 were selected as study cases including 13 multi-peak and 25 single-peak events (Table 3). With 766 rainfall-runoff observations, the earliest 10 events, from 1966 to 1972, were used for calibration while the remainder events were used for validation. Through the Kriging method to calculate the average effective rainfall based on effective rainfall measurements from three rain gauges, current average effective rainfall (mm) and direct hourly runoff (m^3/s) are the input and output with no lead-time considered after be normalized between 0 and 0.9.

5.2 Criteria

The performances of rainfall-runoff simulations were evaluated by four criteria as follows:

(1) Coefficient of efficiency, *CE*, is defined as follows:

$$CE = 1 - \frac{\sum_{k=1}^{K}\left[Q_{obs,k} - Q_{est,k}\right]^2}{\sum_{k=1}^{K}\left[Q_{obs,k} - \overline{Q}_{obs}\right]^2} \tag{51}$$

where $Q_{est,k}$ denotes the discharge of the simulated hydrograph for time index k (m^3/s), $Q_{obs,k}$ is the discharge of the observed hydrograph for time index k (m^3/s), and $\overline{Q}_{obs}$ is the mean of the discharge of the observed hydrograph during whole event period K. The better the fit, the closer CE is to 1.

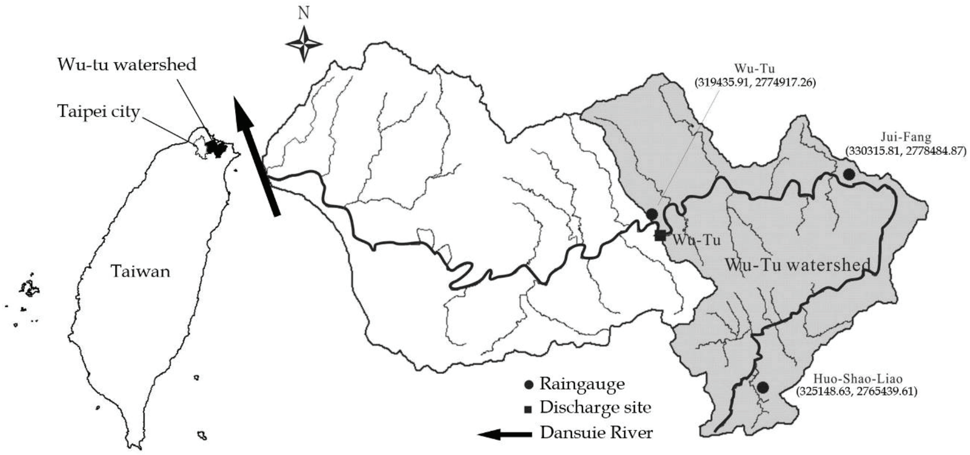


Fig. 5. The maps of Wu-tu watershed showing the study area near Taipei, Taiwan (the coordinates are TWD67 2-degree wide Transverse Mercator projection).

Typhoon name	Time (y/m/d)	Rainfall duration (h)	Rainfall depth (mm)	Max rainfall intensity (mm/h)	Max discharge (m^3/s)	Typhoon name	Time (y/m/d)	Rainfall duration (h)	Rainfall depth (mm)	Max rainfall intensity (mm/h)	Max discharge (m^3/s)
Cora	1966/09/06	48	247.9	20.0	770.7	*Gerald	1984/08/14	127	513.5	23.4	586.4
*Carla	1967/10/17	72	1088.0	52.9	921.2	Nelson	1985/08/22	46	341.4	25.0	1177.0
*Gilda	1967/11/16	59	339.5	29.1	706.9	Brenda	1985/10/03	38	248.4	15.1	626.7
Nadine	1968/07/26	61	252.1	15.1	219.7	*Abby	1986/09/17	91	521.3	28.8	579.0
Elaine	1968/09/29	72	686.6	44.5	1037.7	Alex	1987/07/27	30	187.0	40.7	519.8
*Storm	1969/09/09	89	678.5	24.1	848.4	*Gerald	1987/09/09	33	321.2	47.2	553.9
Elsie	1969/09/26	38	288.5	38.0	662.5	*Storm	1988/09/29	101	627.3	22.7	670.2
Agnes	1971/09/18	69	411.3	31.5	466.3	*Sarah	1989/09/10	61	322.5	27.7	401.2
Bess	1971/09/22	54	353.3	32.0	994.1	*Offlia	1990/06/22	49	251.0	20.6	500.0
Betty	1972/08/16	40	177.2	15.2	677.9	Yancy	1990/08/19	44	259.5	46.3	824.5
Storm	1973/09/20	22	292.5	37.3	862.3	Abe	1990/08/30	35	239.1	15.7	764.4
Wendy	1974/09/28	57	321.2	16.7	822.0	Storm	1990/09/02	26	192.8	32.2	842.5
Vera	1977/07/31	46	264.7	16.9	735.7	*Polly	1992/08/29	98	500.6	17.8	278.9
Storm	1977/11/15	72	292.2	15.2	538.4	Gladys	1994/09/01	18	184.1	31.3	434.2
Irving	1979/08/14	56	340.3	24.4	974.1	*Seth	1994/10/09	48	300.7	12.2	451.3
Storm	1980/11/19	42	266.9	21.9	687.1	Herb	1996/07/31	44	313.6	31.8	1082.9
*Cecil	1982/08/09	34	235.7	23.9	626.4	*Zane	1996/09/27	84	440.6	29.9	666.0
Storm	1984/06/02	18	212.7	46.1	1403.5	Winnie	1997/08/17	47	343.5	24.1	1034.8
Freda	1984/08/06	30	242.1	30.7	501.5	Amber	1997/08/29	42	329.8	30.2	953.5

* Multi-peak event

Table 3. Information about the 38 events selected from Wu-tu watershed.

(2) The error of peak discharge, EQ_p (%), is defined as follows:

$$EQ_p(\%) = \frac{Q_{p,est} - Q_{p,obs}}{Q_{p,obs}} \times 100\% \tag{52}$$

where $Q_{p,est}$ denotes the peak discharge of the simulated hydrograph (m^3/s) and $Q_{p,obs}$ is the peak discharge of the observed hydrograph (m^3/s).
(3) The error of the time for peak to arrive, ET_p, is defined as follows:

$$ET_p = T_{p,est} - T_{p,obs} \tag{53}$$

where $T_{p,est}$ denotes the time for the simulated hydrograph peak to arrive (*hours*) and $T_{p,obs}$ represents the time required for the observed hydrograph peak to arrive (*hours*).
(4) The error of total discharge volume, $VER(\%)$, is defined as follows:

$$VER(\%) = \frac{\left(\sum_{k=1}^{K} Q_{est,k} - \sum_{k=1}^{K} Q_{obs,k}\right)}{\sum_{k=1}^{K} Q_{obs,k}} \times 100\% \tag{54}$$

where $Q_{est,k}$ denotes the discharge of the simulated hydrograph for time index k (m^3/s) and $Q_{obs,k}$ is the discharge of the observed hydrograph for time index k (m^3/s). The better the fit, the closer EQ_p, ET_p and VER are to 0.

6. Result and discussion

A developed DLRNN is applied to perform rainfall-runoff simulation and recognize the transition of rainfall-runoff processes using UHs realized from the DLRNN weights. First, the DLRNN is compared with a forward neural network to demonstrate the advantage of RNNs. DLRNNs identified using indirect system identification and modified system identification then are compared. Furthermore, control system theory is employed to consider a DLRNN in canonical form and compare it with that identified using modified system identification. Finally, rainfall-runoff processes recognition using DLRNN is described.

6.1 Comparison between DLRNN and FNN (Pan et al., 2007)

Through the modified system identification based on the earliest 10 events, a DLRNN with 4 neurons in the hidden layer is calibrated, as shown in Fig. 6. Due to the full connection between neurons in hidden layer, the DLRNN totally has 24 weights for storing information. Therefore, it is fair to have the same control on the quantity of weights for comparing the DLRNN with the feed-forward neural networks (FNNs) although the structures of FNNs with inputting information as a time delay pattern that constitutes the tapped delay line information are classified as local or global RNNs according to the definition by Tsoi and Back (1997). Based on the rule of Equations (55) and (56), observed runoff and rainfall data are used in sequence to constitute the tapped delay line inputs as the input layer illustrated in Fig. 7. In hidden layer of Fig. 7, a bias neuron always delivers a negative impulse as a threshold to each hidden neuron. All FNNs compared with the DLRNN herein are trained using the same calibrated data via the back-propagation learning algorithm, the most common learning algorithm for FNNs.

$$\text{input neuron} = \text{rainfall}\big(k - (\text{int}((n+1)/2)+1)\big), \text{ if } n \text{ is odd;} \tag{55}$$

$$\text{input neuron} = \text{runoff}\left(k - \left(\text{int}(n/2)\right)\right), \text{ if } n \text{ is even} \tag{56}$$

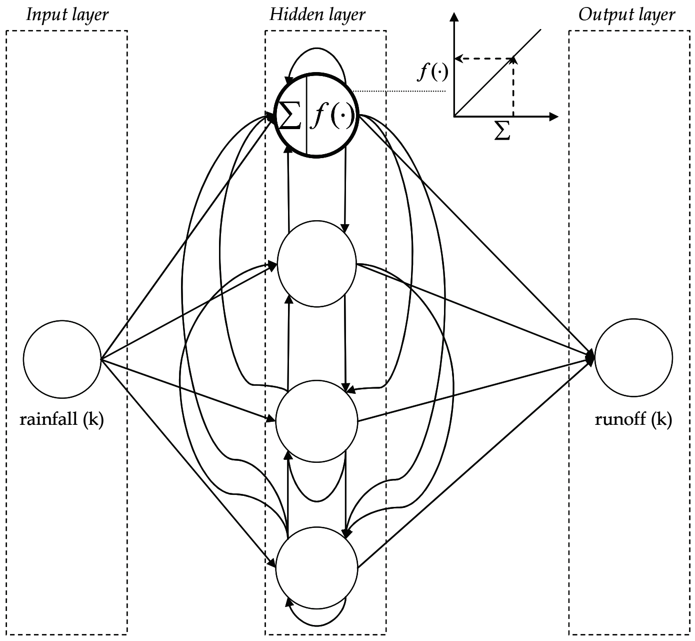


Fig. 6. Architecture of DLRNN identified via modified system identification.

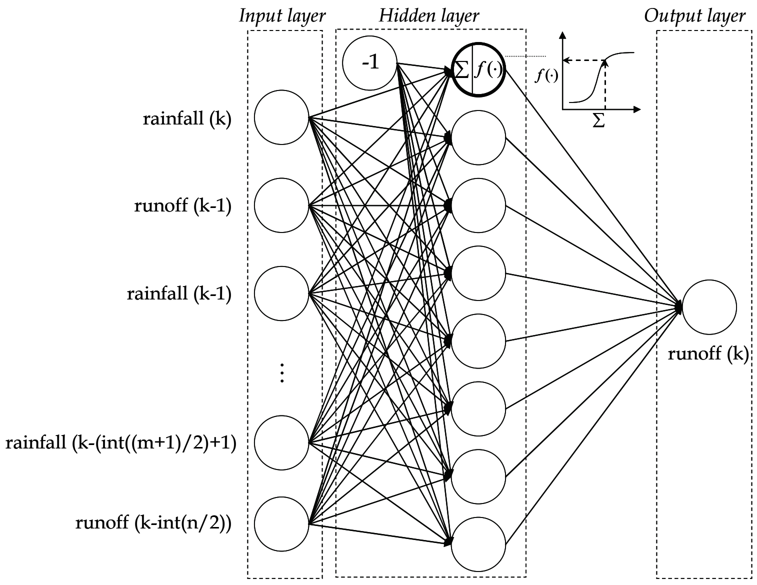


Fig. 7. The structure of FNNs with the tapped delay line inputs.

Model Form	Feed-forward neural network								DLRNN
Number of neurons in hidden layer	1	2	3	4	5	6	7	8	4
Number of neuraons in input layer	22	10	6	4	3	2	2	1	1
Number of neural network's weights	24	24	24	24	25	24	28	24	24
CE	0.943	0.982	0.975	0.974	0.957	0.952	0.950	-0.074	0.926
EQp (%)	10.363	4.377	4.432	4.445	4.545	4.687	5.322	48.936	12.438
ETp (hour)	2.079	1.184	1.184	1.526	1.895	1.921	1.921	7.474	1.036
VER (%)	5.504	2.088	2.242	2.383	2.513	3.101	3.295	24.509	4.769

Table 4. The averages of absolute criteria of the DLRNN and FNNs to simulate the rest 28 events (Pan et al., 2007).

Table 4 shows the averages of the absolute criteria of the DLRNN and the FNNs in which FNN(1-8-1) is the only neural network without any feedback connection. According to the average absolute criteria, the FNN(1-8-1) performs poorly because it is merely a static system without memory and only executes mapping from rainfall to runoff. However, the FNNs with tapped delay line inputs, such as FNN(2-7-1) to FNN(22-1-1), perform superiorly. The result shows the importance of a feedback connection and using tapped delay line inputs to the FNN. Chiang et al. (2004) also noticed that the feature of feedback connections is especially important and useful for grasping the extraordinary time-varying characteristics of the rainfall-runoff processes. The neural network with only one rainfall input can not achieve a satisfactory mapping to the current runoff because the rainfall-runoff processes are dynamic systems. One more tapped delay line input, like FNN(2-7-1), gives the feed-forward neural network the last-time-step status of the runoff, and raises the CE over 0.94. However, the DLRNN only needs the current rainfall as the input to get a satisfactory simulation because the feedback connections in hidden layer give the DLRNN the function to calculate the state of the rainfall-runoff process recurrently.

6.2 Comparison between DLRNNs based on two identification methods

Vos et al. (2005) commented that a disadvantage of artificial neural networks is that the optimal form or value of most network design parameters differ for each application and cannot be theoretically defined, which is why they are commonly found using trial-and-error approaches. However, the identification methods mentioned herein provide a deterministic solution. This chapter considers the indirect and modified system identification for identifying DLRNNs. In the realization step of the indirect system identification, a series of singular values is carried out through the singular value decomposition, and it can be illustrated in Fig. 8. If the singular values can be separated distinctly into two groups, namely the significant and the neglected groups, the number of neurons in the hidden layers of a RNN equals to the size of the significant group. From Fig. 8, the first two singular values are relatively significant and the number of neurons in the hidden layers are at least 2. However, the other singular values do not decrease noticeably, making it difficult to optimize the number of neurons of the hidden layers. Furthermore, the relation between the coefficient of efficiency and the number of neurons in the hidden layers of the DLRNN determined using trial-and-error method is illustrated as the open dots in Fig. 9. The CE increases from 0.70 to over 0.86 while the number of neurons in hidden layers exceeds 2 in Fig. 9. Six neurons in the hidden layer are selected as the optimum DLRNN (denoted as DLRNN(1)), denoted as the solid dot at the right side of Fig. 9) using the best coefficient of efficiency (*CE*=0.87043).

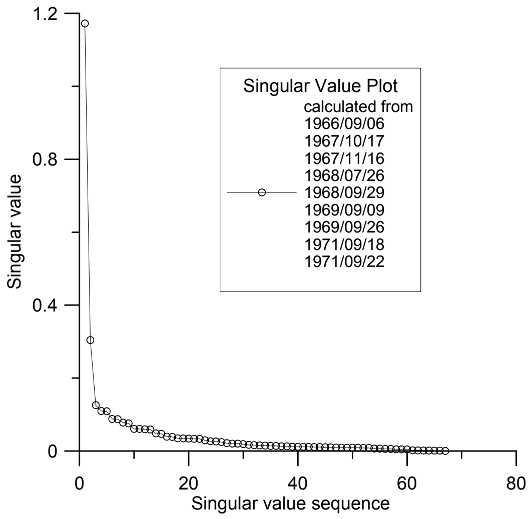


Fig. 8. The singular value plot from the realization step of indirect system identification.

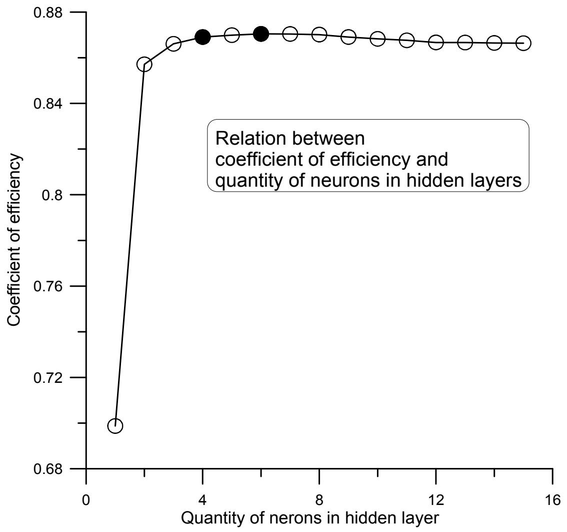


Fig. 9. The relation between coefficient of efficiency and number of neurons in hidden layers of the DLRNN.

Another DLRNN (denoted as DLRNN(2)) has four neurons in the hidden layer, as determined using modified system identification (solid dot at the left side of Fig. 9). Owing to part of the subspace algorithm being included in modified system identification, the four neurons in the hidden layer are chosen without any referable plot, such as singular value plot. From Fig. 9, the *CE* of DLRNN(2) is just 0.00028 less than that of DLRNN(1). However, DLRNN(2) reduces 48 weights of DLRNN(1), to 24 weights. The 50% reduction in weights from DLRNN(2) demonstrates that the combination of modified system identification and the advantages of indirect system identification and subspace algorithm provide an efficient algorithm for applying DLRNN in hydrology.

6.3 Comparison between DLRNNs in different forms

The DLRNN adopted herein is a fully RNN and has full connections between neurons in different layers. However, using a state space model is well known to over parameterize the estimation problem, while using canonical forms, as illustrated in Fig. 10, is far more economical for estimating the linear model. Figures 6 and 10 show that the DLRNNs have the feed-back connections in the hidden layers that belong to the local recurrent structures. The DLRNN in a canonical form has the same number of neurons as the original DLRNN, but the DLRNN in a canonical form has the minimum connections and weights to achieve the same performance. Hence, the comparison between the two DLRNNs in canonical form is of interest in this investigation. Some experiments are designed to clarify this issue. First, in the flowchart illustrated in Fig. 4, the original DLRNN(1) is transformed into a DLRNN in the canonical form after identifying the quantity of neurons in hidden layer and the weights of the DLRNN. Figure 10 shows that the DLRNN in the canonical form is clearly not a fully RNN. 28 validated events are fed to the model, and a new on-line learning method developed by Pan and Wang (2004), is applied to develop the DLRNN into a fully RNN via on-line learning. Table 5 lists the average absolute criteria. The table reveals that the canonical and non-canonical form DLRNNs do not differ significantly, and the on-line learning algorithm always derives a fully RNN from a DLRNN in the canonical form.

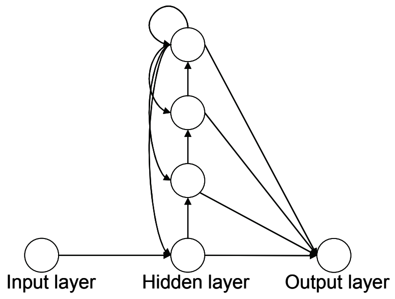


Fig. 10. The DLRNN in canonical form.

model type	*CE*	*EQp* (%)	*ETp* (hour)	*VER* (%)
original model	0.926	12.438	1.036	4.769
canonical form	0.925	12.704	1.071	4.845

original model: a DLRNN identified via modified system identification.

canonical form: a DLRNN in canonical form.

Table 5. The averages of absolute criteria of the DLRNNs in two forms.

6.4 Recognition of the transition of rainfall-runoff processes (Pan et al., 2007)

A streamflow or discharge hydrograph is a graph showing the flow rate as a function of time at a given location on the stream. In effect, the hydrograph is “an integral expression of the physiographic and climatic characteristics that govern the relations between rainfall and runoff of a particular drainage basin" (Chow, 1959). UH is a hypothetical unit response of the watershed to a unit input of rainfall that has been widely adopted by hydrologists to represent the mechanism of rainfall-runoff processes. Through the visualization of the transition of rainfall-runoff processes by UHs, the duration of a storm event, the time to peak flow, and the peak flow can be detected from the UHs. Therefore, the DLRNN learning algorithm is applied to modify the weights of the DLRNN on-line for detecting the transition of UHs based on the connection between the DLRNN and UH representation by treating the weights as Markov parameters. The structure of DLRNN can analogize the rainfall-runoff processes in a simple manner. The number of neurons in the hidden layer calibration by modified system identification describes the dimensions of the state space for the rainfall-runoff processes. Each neuron in the hidden layer represents a state variable that is controlled by rainfall and interacts with all state variables recurrently. Although the state variables can not be measured directly, UH can be represented based on their weights to describe the transition of rainfall-runoff processes.

Equations (8) and (9) reveal the relationship between the UH and the weights of DLRNN. Equation (9) also illustrates the relationship between the system responses to a unit impulse and the weights of DLRNN used herein. The time variance of the weights of a DLRNN can be used to recognize the transition of rainfall-runoff processes. Figure 11 illustrates the transition of UHs of the single-peak typhoon in Aug. 17, 1997, while Fig. 12 shows the simulation of this typhoon through DLRNN with on-line learning. At the beginning of the simulation, the weights of the DLRNN are identified from the earliest 10 events to form a generalized model. When comparing these two figures, the change of the UHs reveals the peak arrival is between the 15th and 30th hours. The time to peak of this typhoon is approximately 8 hours, shown in Fig. 11. The 8-hour duration is significantly increased after the time to peak of UH is calibrated as 3 hours. The rainfall process is fed to DLRNN to simulate runoff, as illustrated in Fig. 12, and the simulated runoff should follow the trends of the rainfall process. The rainfall-runoff simulations are evaluated as effective if the trends of rainfall and runoff are identical.

Another study case, Zane typhoon, is a multi-peak rainfall-runoff process out of the 38 selected events (Table 5). Figure 14 illustrates the variation between observed rainfall and runoff, and shows the excellent simulation performance from DLRNN. Figure 13 characterizes the transition of the rainfall-runoff process as the changes of UHs. During the first 20 hours of Zane typhoon, the simulated runoff is slightly higher than observed runoff (Figure 14), and this phenomenon demonstrates that the peak of the actual UH is lower than the UH realized from DLRNN. Through the on-line learning, the peak of the UH realized from DLRNN decays during first 20 hours. However, the largest peak of observed runoff is higher than the simulated runoff, and this shows that the actual UH of the rainfall-runoff process changes with time. Therefore, the peak of the UH realized from DLRNN increases after on-line learning. Furthermore, the difference between observed and simulated runoffs around the 60th hour demonstrates again the property of DLRNN that simulated runoff goes with the trends of the rainfall process. Additionally, a common conceptual model, called linear reservoir model, is introduced to compare with the DLRNN. It is an objective comparison in which both two models consider rainfalls as inputs. Results show that DLRNN performs better than the linear reservoir model.

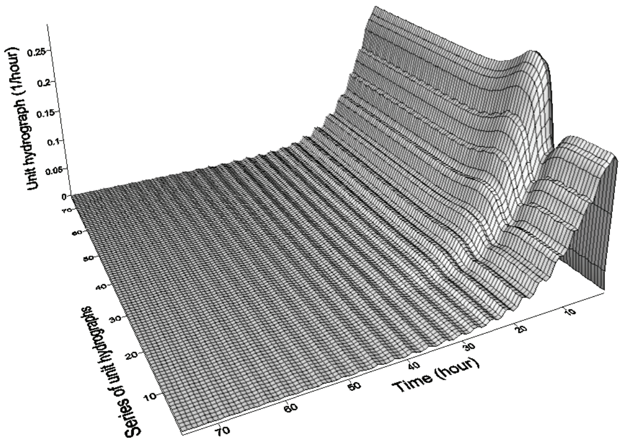


Fig. 11. The transition of UHs of the single-peak typhoon in Aug. 17, 1997.

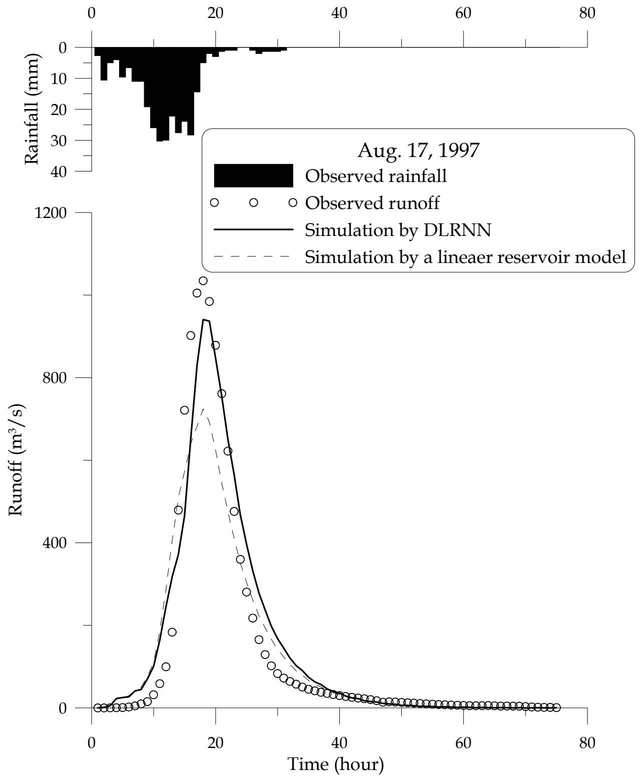


Fig. 12. Simulation of Winnie typhoon in Aug. 17, 1997 via DLRNN with on-line learning and a linear reservoir model.

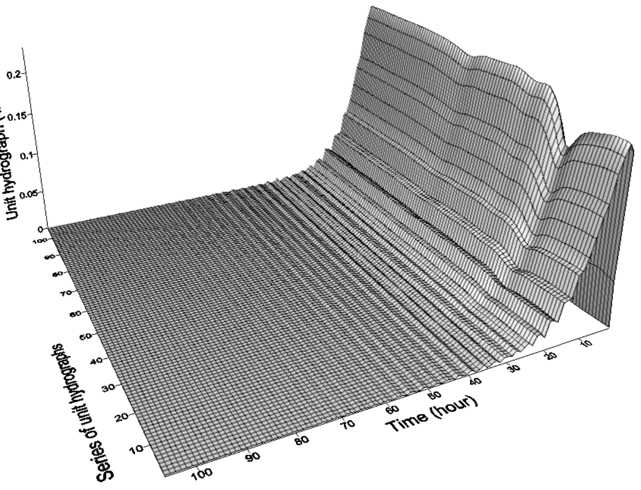


Fig. 13. The transition of UHs of the multi-peak typhoon in Sep. 27, 1996.

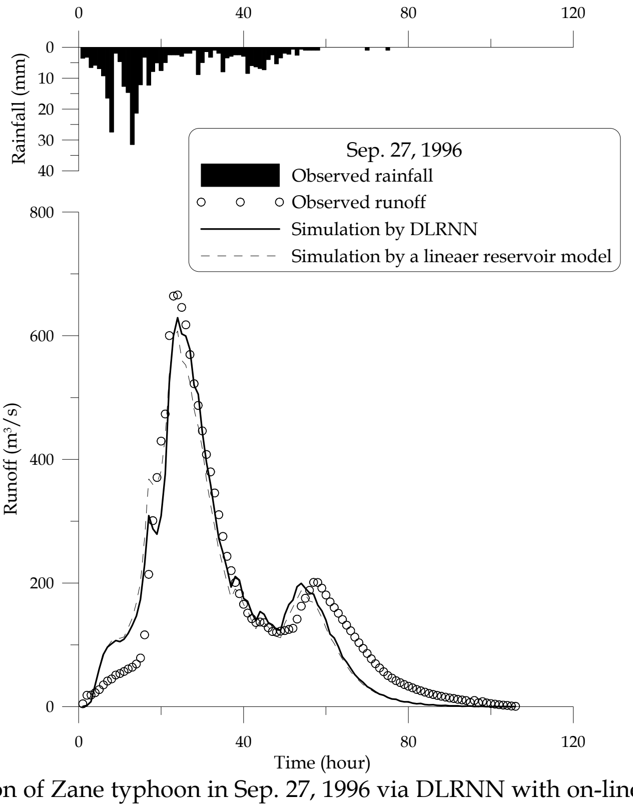


Fig. 14. Simulation of Zane typhoon in Sep. 27, 1996 via DLRNN with on-line learning and a linear reservoir model.

A generalized UH identified from multi-event rainfall-runoff records can represent the hydrological feature of the watershed. However, due to the complex interaction with other hydrometeorological and geomorphological processes within the hydrological cycle, the true UH of a rainfall-runoff process can not be predetermined before the event happens. DLRNN has the capability to shape the generalized UH to catch the transition of rainfall-runoff processes by real time modifying weights. The case study shows that the representation of UHs from DLRNN weights and the tracing ability of the DLRNN. The transition of the rainfall-runoff processes is visualized by the representation of UHs that furthers the interpretation of DLRNN weights.

7. Conclusion

In this chapter, the application of a DLRNN is demonstrated to simulate rainfall-runoff processes and recognize the transition of UHs in hydrology. Although most neural networks are black-box models that lack physical meanings of weights, the DLRNN developed in this chapter connects its weights with UHs that reveal the physical concepts from the network based on the special structure of RNNs. Without trial and error method, the structure and the weights of DLRNN can be quickly determined through a modified form of system identification that combines indirect system identification with the subspace algorithm. Then, the DLRNN learning algorithm based on the interchange of the roles of the network state variables and the weight matrix is derived for on-line training.

In this chapter, the DLRNN introduced can not only simulate rainfall-runoff processes, but also recognize the transition of UHs. Owing to the feedback connections, DLRNN performs rainfall-runoff simulations as dynamic systems, and the advantage of DLRNN's dynamic feature has been proven after the comparison between DLRNN and FNN. The investigation of the connections between weights and physical meanings is an extension of neural networks applied in hydrological field due to the linearization of the RNN. Based on the linearization, weights of DLRNN are treated as Markov parameters to realize the transition of UHs. Through on-line learning, DLRNN modifies the weights to capture the relation between rainfall and runoff every time step, and the transition of rainfall-runoff processes can be emerged based on the changes of UHs.

Furthermore, a modified system identification that combines indirect system identification with subspace algorithm is described to calibrate the DLRNN. This method determines the quantity of neurons in hidden layer and the weights of the network. It overcomes the drawback of costing time by traditional trial and error search for optimum structure of DLRNN. Additionally, the different forms of DLRNN have also been discussed herein. The results show that the performances of DLRNNs in different forms are close. Hence, the transformation of canonical form can be ignored in the flowchart of simulation via DLRNN. Finally, four criteria have been applied to evaluate the performance of rainfall-runoff simulation via DLRNN. The results show that the performance is satisfactory and DLRNN is competent to simulate dynamic systems, like rainfall-runoff processes.

8. Future research

Although feed-forward neural networks are commonly adopted to solve hydrological problems, applying RNNs to deal with the issues of hydrology is still a novel technique because the structure and the learning algorithm of RNN are more complex than those of FNN. This chapter has demonstrated an example to show how RNN applies to hydrological problems. However, further research is necessary. As Sudheer mentioned (2005), hydrologists have not endeavored to construe the knowledge embedded in the trained ANN models, other than the recent research attempts to assign physical significance to the internal architecture of ANN hydrological models. Therefore, how to abstract more physical interpretations from the weights or the architectures of RNN, like the connection between UHs and the weights of DLRNN, is one of the major issues. Furthermore, in order to clarify some opacity in RNN, the DLRNN mentioned herein is only a single-input-single-output (SISO) system with a nonlinearity-interpretation trade-off. With construing the knowledge embedded in, an ideal multi-input-multi-output RNN without any trade-off for rainfall-runoff simulation is needed.

9. References

Abdelghani M. & Verhaegen M. (1998). Comparison study of subspace identification methods applied to flexible structures. *Mechanical Systems and Signal Processing*, Vol. 12, No. 5, pp. 679-692, ISSN 0888-3270.

Anctil F.; Asce M. & Rat A. (2005). Evaluation of neural network streamflow forecasting on 47 watersheds. *Journal of Hydrologic Engineering*, Vol. 10, No. 1, pp. 85-88, ISSN 1084-0699.

Atiya A.F. & Parlos A.G. (2000). New results on recurrent network training: unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks*, Vol. 11, No. 3, pp. 697-709, ISSN 1045-9227.

Boto M.A. & Costa J.S. (1998). A comparison of nonlinear predictive control techniques using neural network models. *Journal of Systems Architecture*, Vol. 44, No. 8, pp. 597-616, ISSN 1383-7621.

Chang L.C.; Chang F.J. & Chiang Y.M. (2004). A two-step-ahead recurrent neural network for stream-flow forecasting. *Hydrological Processes*, Vol. 18, pp. 81-92, ISSN 0885-6087.

Chow T.W.S. & Cho S.Y. (1997). Development of a recurrent sigma-pi neural network rainfall forecasting system in Hong Kong. *Neural Computing and Applications*, Vol. 51, No. 5, pp. 921-927, ISSN 0941-0643.

Chiang Y.M.; Chang L.C. & Chang F.J. (2004). Comparison of static-feedforward and dynamic-feedback neural networks for rainfall-runoff modelling. *Journal of Hydrology*, Vol. 290, pp. 297-311, ISSN 0022-1694.

Chiang Y.M.; Chang F.J.; Jou B.J.D. & Lin P.F. (2007a). Dynamic ANN for precipitation estimation and forecasting from radar observations. *Journal of Hydrology*, Vol. 334, pp. 250-261, ISSN 0022-1694.

Chiang Y.M.; Hsu K.L.; Chang F.J.; Hong Y. & Sorooshian S. (2007b) Merging multiple precipitation sources for flash flood forecasting. *Journal of Hydrology*, Vol. 340, pp. 183-196, ISSN 0022-1694.

Chow V.T. (1959). *Open Channel Hydraulics*, McGraw Hill, ISBN 007085906X, New York.

Coulibaly P; Anctil F.; Rasmussen P. & Bobée B. (2000). A recurrent neural networks approach using indices of low-frequency climatic variability to forecast regional annual runoff. *Hydrological Processes*, Vol. 14, pp. 2755-2777, ISSN 0885-6087.

Coulibaly P.; Anctil F.; Aravena R. & Bobée B. (2001). Artificial neural network modeling of water table depth fluctuations. *Water Resources Research*, Vol. 37, No. 4, pp. 885-896, ISSN 0043-1397.

Coulibaly P. & Baldwin C.K. (2005). Nonstationary hydrological time series forecasting using nonlinear dynamic methods. *Journal of Hydrology*, Vol. 307, pp. 164-174, ISSN 0022-1694.

Coulibaly P. & Evora N.D. (2007). Comparison of neural network methods for infillin missing daily weather records. *Journal of Hydrology*, Vol. 341, pp. 27-41, ISSN 0022-1694.

Gustafsson T. (2001). Subspace identification using instrumental variable techniques. *Automatica*, Vol. 37, No. 12, pp. 2005-2010, ISSN 0005-1098.

Henriques J. & Dourado A. (1998). A multivariable adaptive control using a recurrent neural network. *Proceedings of international conference on engineering applications of neural networks, engineering applications of neural networks*, pp. 118-121, UK, June 1998, Gibraltar.

Karamouz M.; Razavi S. & Araghinejad S. (2008). Long-lead seasonal rainfall forecasting using time-delay recurrent neural networks: a case study. *Hydrological Processes*, Vol. 22, pp. 229-241, ISSN 1099-1085.

Maier H.R. & Dandy G.C. (2000). Neural networks for the prediction and forecasting of water resources variables: a review of modelling issues and applications. *Environmental Modelling & Software,* Vol. 15, pp. 101-124, ISSN 1364-8152.

Nagesh Kumar D.; Srinivasa Raju K. & Sathish T. (2004). River flow forecasting using recurrent neural networks. *Water Resources Management*, Vol. 18, pp. 143-161, ISSN 0920-4741.

Pan T.Y. & Wang R.Y. (2004). State space neural networks for short term rainfall-runoff forecasting. *Journal of Hydrology*, Vol. 297, pp. 34-50, ISSN 0022-1694.

Pan T.Y. & Wang R.Y. (2005). Using recurrent neural networks to reconstruct rainfall-runoff processes. *Hydrological Processes*, Vol. 19, pp. 3603-3619, ISSN 0885-6087.

Pan T.Y.; Wang R.Y. & Lai J.S. (2007). A deterministic linearized recurrent neural network for recognizing the transition of rainfall-runoff processes. *Advances in Water Resources*, Vol. 30, pp. 1797-1814, ISSN 0309-1708.

Parlos A.G.; Rais O.T. & Atiya A.F. (2000). Multi-step-ahead prediction using dynamic recurrent neural networks. *Neural Networks*, Vol. 13, pp. 765-786, ISSN 0893-6080.

Ptitchkin V.A. (2001). Models of dynamic neural networks and automatic control systems, *Proceedings of the second international conference on neural networks and artificial intelligence*, Republic of Belarus, October 2001, Minsk.

Rahman M.H.R.F. & Kuanyi Z. (2000). Neural network approach for linearizing control of nonlinear process plants. *IEEE Transactions on Industrial Electronics*, Vol. 47, No. 2, pp. 470-477, ISSN 0278-0046.

Ramos J.; Mallants D. & Feyen J. (1995). State space identification of linear deterministic rainfall-runoff models. *Water Resources Research*, Vol. 31, No. 6, pp. 1519-1531, ISSN 0043-1397.

Sudheer K.P. (2005). Knowledge extraction from trained neural network river flow models. *Journal of Hydrologic Engineering*, Vol. 10, No. 4, pp. 264-269, ISSN 1084-0699.

Tsoi A.C. & Back A. (1997). Discrete time recurrent neural network architectures: a unifying review. *Neurocomputing,* Vol. 15, pp. 183-223, ISSN 0925-2312.

Vos N.J. & Rientjes T.H.M. (2005). Constraints of artificial neural networks for rainfall-runoff modeling: trade-offs in hydrological state representation and model evaluation. *Hydrology and Earth System Sciences Discussions*, Vol. 2, pp. 365-415, ISSN 1812-2108.

Walter M; Recknagel F.; Carpenter C. & Bormans M. (2001). Predicting eutrophication effects in the Burrinjuck Reservoir (Australia) by means of the deterministic model SALMO and the recurrent neural network model ANNA. *Ecological Modelling*, Vol. 146, pp. 97-113, ISSN 0304-3800.

Zarmarreño J.M.; Vega P.; García L.D. & Francisco M. (2000). State-space neural network for modelling, prediction and control. *Control Engineering Practice*, Vol. 8, pp. 1063-1075, ISSN 0967-0661.

Recurrent Neural Approach for Solving Several Types of Optimization Problems

Ivan N. da Silva, Wagner C. Amaral, Lucia V. Arruda
and Rogerio A. Flauzino
University of São Paulo, USP/EESC/SEL, CP 359, São Carlos, SP
Brazil

1. Introduction

An artificial neural network, more commonly known as neural network, is a mathematical model for information processing based on the biological nervous system, which has a natural propensity for storing experiential knowledge and making it available for use (Haykin, 1999). The main advantage of a neural network is in its ability to approximate functional relationships, particularly nonlinear relationships.
Neural networks have been applied to several classes of optimization problems and have shown promise for solving such problems efficiently. Most of the neural architectures proposed in the literature solve specific types of optimization problems (Dillon & O'Malley, 2002; Kakeya & Okabe, 2000; Xia et al., 2002). In contrast to these neural models, the network proposed here is able to treat several kinds of optimization problems using a unique network architecture.
The approach described in this chapter uses a modified Hopfield network, which has equilibrium points representing the solution of the optimization problems. The Hopfield network is modified by presenting an optimization process carried out in two distinct stages, which are represented by two energy functions. The internal parameters of the network have been computed using the valid-subspace technique (Aiyer et al., 1990; Silva et al., 1997). This technique allows us to define a subspace, which contains only those solutions that represent feasible solutions to the problem analyzed. It has also been demonstrated that with appropriately set parameters, the network confines its output to this subspace, thus ensuring convergence to a valid solution. Also in contrast to other neural approaches that use an energy function for each constraint to be satisfied, the mapping of optimization problems using the modified Hopfield network always consists of determining just two energy functions, which are denoted by E^{conf} and E^{op}. The function E^{conf} is a confinement term that groups all structural constraints associated with the problems, and E^{op} is an optimization term that leads the network output to the equilibrium points corresponding to optimal solutions.
In this chapter, the proposed approach has been applied to solve combinatorial optimization problems, dynamic programming problems and nonlinear optimization problems. In addition to providing a new approach for solving several classes of optimization problems through a unique neural network architecture, the main advantages of using the modified

Hopfield network proposed in this chapter are the following: i) the internal parameters of the network are explicitly obtained by the valid-subspace technique of solutions, which avoids the need to use training algorithm for their adjustments; ii) the application of the valid-subspace technique allows feasible solutions to be found, which are derived from the confinement of all structural constraints by E^{conf}; iii) The optimization and confinement terms are not weighted by penalty parameters, which could affect both precision of the equilibrium points and their respective convergence processes; iv) for all classes of optimization problems, the same methodology is adopted to derive the internal parameters of the network, and v) for industrial application, the modified Hofpfield network offers simplicity of implementation both in analogue hardware, making use of operational amplifiers and in digital hardware by using digital signal processors.
The organization of the present chapter is as follows. In Section 2, the modified Hopfield network is presented, and the valid-subspace technique used to design the network parameters is described. In Section 3, the mapping of optimization problems using the modified Hopfield network is formulated. In Section 4, simulation results are given to demonstrate the performance of the developed approach. In Section 5, the key issues raised in the chapter are summarized and conclusions drawn.

2. The modified Hopfield network

Hopfield networks are single-layer networks with feedback connections between nodes. In the standard case, the nodes are fully connected, i.e., every node is connected to all others nodes, including itself (Hopfield, 1984). The node equation for the continuous-time network with N neurons is given by:

$$\dot{u}_i(t) = -\eta . u_i(t) + \sum_{j=1}^{N} T_{ij} . v_j(t) + i_i^b \tag{1}$$

$$v_i(t) = g(u_i(t)) \tag{2}$$

where $u_i(t)$ is the current state of the i-th neuron, $v_i(t)$ is the output of the i-th neuron, i_i^b is the offset bias of the i-th neuron, $\eta . u_i(t)$ is a passive decay term, T_{ij} is the weight connecting the j-th neuron to i-th neuron.
In Equation (2), $g(u_i(t))$ is a monotonically increasing threshold function that limits the output of each neuron to ensure that the network output always lies in or within a hypercube. It is shown in Hopfield (1984) that if $\boldsymbol{T}$ is symmetric and $\eta=0$, the equilibrium points of the network correspond to values $\boldsymbol{v}(t)$ for which the energy function (3) associated with the network is minimized:

$$E(t) = -\frac{1}{2} \boldsymbol{v}(t)^T . \boldsymbol{T} . \boldsymbol{v}(t) - \boldsymbol{v}(t)^T . \boldsymbol{i}^b \tag{3}$$

Therefore, the mapping of optimization problems using the Hopfield network consists of determining the weight matrix $\boldsymbol{T}$ and the bias vector $\boldsymbol{i}^b$ to compute equilibrium points to represent the problem to be solved.
One of the major difficulties in mapping optimization problems onto a conventional Hopfield network involves deciding how constraints can be included. Basically, most of

these neural networks proposed in the literature for solving optimization problems code the constraints as terms in the energy function that are weighted by penalty parameters. The stable equilibrium points of these networks, which represent a solution of the optimization problem, gave the correct solution only when those parameters are properly adjusted, and both the accuracy and the convergence process can be affected. This weakness of penalty and barrier function methods has, of course, been well known since 1968 when it was discussed by Fiacco and McCormick in Fiacco & McCormick (1968). They investigated the numerical problem associated with the change of parameters in these functions. In such approaches, the energy function given in (3) is represented by:

$$E(t) = E^{op}(t) + c_1 \cdot E_1^{const}(t) + c_2 \cdot E_2^{const}(t) + \cdots + c_m \cdot E_m^{const}(t) \qquad (4)$$

where c_i are positive constants that are weighing each one of the constraints E_i^{const}. Thus, the network is involved with the minimization of a single energy function (E^{op}) correspondent to the objective function of the problem and subject to the several constraints E_i^{const}. If any of these constraints is violated then the solution is not feasible, i.e., the multiple constraints terms E_i^{const} tend to cancel each other out. Moreover, the convergence processes of these networks depend on the correct adjustment of the penalty constants associated with the energy terms.

In this chapter, we have developed a modified Hopfield network that does not depend on penalty or weighting parameters, which overcomes shortcomings associated with the other neural approaches. In contrast to most of the other neural models, the network proposed here is able to treat several kinds of optimization problems using a unique network architecture. A modified energy function $E^m(t)$, composed just by two energy terms is used here, which is defined as follows:

$$E^m(t) = E^{op}(t) + E^{conf}(t) \qquad (5)$$

where $E^{conf}(t)$ is a confinement term that groups the structural constraints associated with the respective optimization problem, and $E^{op}(t)$ is an optimization term that conducts the network output to the equilibrium points corresponding to a cost constraint. Thus, the minimization of $E^m(t)$ of the modified Hopfield network is conducted in two stages:

i) minimization of the term $E^{conf}(t)$:

$$E^{conf}(t) = -\frac{1}{2}\boldsymbol{v}(t)^T.\boldsymbol{T}^{conf}.\boldsymbol{v}(t) - \boldsymbol{v}(t)^T.\boldsymbol{i}^{conf} \qquad (6)$$

where $\boldsymbol{v}(t)$ is the network output, $\boldsymbol{T}^{conf}$ is a weight matrix and $\boldsymbol{i}^{conf}$ is a bias vector belonging to E^{conf}. This results in a solution $\boldsymbol{v}(t)$ in the subspace generated from the structural constraints imposed by the problem. This subspace has been derived from analysis of the Hopfield network dynamics, where it is shown in Hopfield (1984) that the energy functions E_i^{const}(t) given in (4), which are defined by (3), are Lyapunov functions provided matrices $\boldsymbol{T}$ are symmetric. An investigation associating the equilibrium points of those Lyapunov functions with respect to the eigenvalues and eigenvectors of the matrices $\boldsymbol{T}$ shows that all feasible solutions can be grouped in a unique subspace of solutions with equation $\boldsymbol{v}(t+1) = \boldsymbol{T}^{conf}.\boldsymbol{v}(t) + \boldsymbol{i}^{conf}$, where $\boldsymbol{T}^{conf}$ is a projection matrix and $\boldsymbol{i}^{conf}$ is a vector orthogonal to $\boldsymbol{T}^{conf}$. By analyzing the convergence process dynamics, it is revealed that $\boldsymbol{v}$ evolves first along those

eigenvectors of $\boldsymbol{T}^{conf}$ with the large eigenvalues, then along those with negative eigenvalues. As consequence of the application of this subspace approach, which is named the valid-subspace method, a unique energy term can be used to represent all constraints associated with the optimization problem since $\boldsymbol{T}^{conf}$ to be a projection matrix ($\boldsymbol{T}^{conf}.\ \boldsymbol{T}^{conf} = \boldsymbol{T}^{conf}$) and $\boldsymbol{i}^{conf}$ a vector orthogonal to $\boldsymbol{T}^{conf}$, i. e., $\boldsymbol{T}^{conf}.\ \boldsymbol{i}^{conf} = \boldsymbol{0}$. A more detailed analysis of the valid-subspace method can be found in Silva et al. (1997).
ii) minimization of the term $E^{op}(t)$:

$$E^{op}(t) = -\frac{1}{2}\boldsymbol{v}(t)^T.\boldsymbol{T}^{op}.\boldsymbol{v}(t) - \boldsymbol{v}(t)^T.\boldsymbol{i}^{op} \tag{7}$$

where $\boldsymbol{T}^{op}$ is weight matrix and iop is bias vector belonging to E^{op}. This corresponds to move $\boldsymbol{v}(t)$ towards an optimal solution (the equilibrium points). Thus, the operation of the modified Hopfield network consists of three main steps, as shown in Fig. 1:

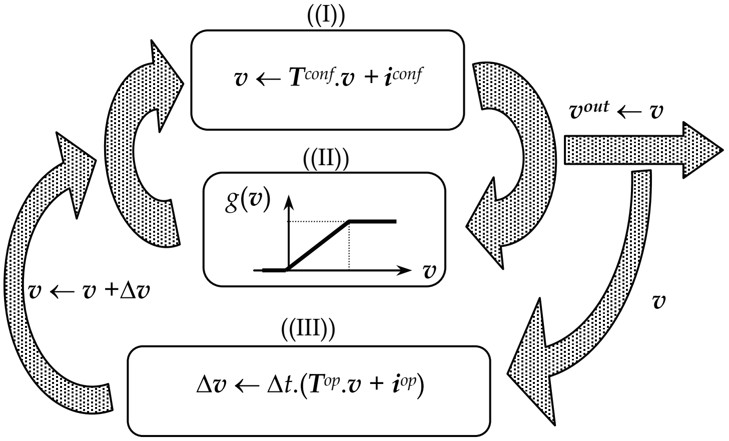


Fig. 1. The modified Hopfield network.

Step ((I)): Minimization of E^{conf}, corresponding to the projection of $\boldsymbol{v}(t)$ in the valid subspace defined by:

$$\boldsymbol{v}(t+1) = \boldsymbol{T}^{conf}\cdot\boldsymbol{v}(t) + \boldsymbol{i}^{conf} \Rightarrow \boldsymbol{v} \leftarrow \boldsymbol{T}^{conf}\cdot\boldsymbol{v} + \boldsymbol{i}^{conf} \tag{8}$$

where: $\boldsymbol{T}^{conf}$ is a projection matrix ($\boldsymbol{T}^{conf}.\boldsymbol{T}^{conf} = \boldsymbol{T}^{conf}$) and the vector $\boldsymbol{i}^{conf}$ is orthogonal to the subspace ($\boldsymbol{T}^{conf}.\boldsymbol{i}^{conf} = \boldsymbol{0}$). This operation corresponds to an indirect minimization of $E^{conf}(t)$. An analysis of the valid-subspace technique is presented in Aiyer et al. (1990) and Silva et al. (1997).
Step ((II)): Application of a nonlinear 'symmetric ramp' activation function constraining $\boldsymbol{v}(t)$ in a hypercube:

$$g_i(v_i) = \begin{cases} lim_i^{\inf} & , \text{ if } \quad v_i < lim_i^{\inf} \\ v_i & , \text{ if } \quad lim_i^{\inf} \le v_i \le lim_i^{\sup} \\ lim_i^{\sup} & , \text{ if } \quad v_i > lim_i^{\sup} \end{cases} \tag{9}$$

where $v_i(t) \in [lim_i^{\inf}, lim_i^{\sup}]$. For combinatorial optimization and dynamic programming problems $g_i(v_i) \in [0, 1]$ and in this case $lim_i^{\inf} = 0$ and $lim_i^{\sup} = 1$. Although v is inside a set with particular structure, the modified
Hopfield network can represent a general problem. For example, if $\boldsymbol{v} \in \Re^n$ for nonlinear optimization problem, then $lim_i^{\inf} = -\infty$ and $lim_i^{\sup} = \infty$.

Step ((III)): Minimization of E^{op}, which involves updating of $\boldsymbol{v}(t)$ in direction to an optimal solution (defined by $\boldsymbol{T}^{op}$ and $\boldsymbol{i}^{op}$) corresponding to network equilibrium points, which are the solutions for the optimization problem considered in a specific application. Using the 'symmetric ramp' activation function defined in (7) and given η=0, equation (2) subsequently becomes:

$$\boldsymbol{v}(t) = g(\boldsymbol{u}(t)) = \boldsymbol{u}(t) \tag{10}$$

By comparison with (1) and (6), we have:

$$\frac{d\boldsymbol{v}(t)}{dt} = \dot{\boldsymbol{v}} = -\frac{\partial E^{op}(t)}{\partial \boldsymbol{v}}$$

$$\Delta \boldsymbol{v} = -\Delta t.\nabla E^{op}(\boldsymbol{v}) = \Delta t.(\boldsymbol{T}^{op}.\boldsymbol{v} + \boldsymbol{i}^{op}) \tag{11}$$

Therefore, minimization of E^{op} consists of updating $\boldsymbol{v}(t)$ in the opposite direction to the gradient of E^{op}. These results are also valid when a 'hyperbolic tangent' activation function is used. In this step, the process used by the modified Hopfield network for solving the corresponding differential equations are identical to Euler's method and in optimization terms it represents a steepest descent algorithm with a fixed step size.

After each optimization step in ((III)), it is necessary to carry out several times the two steps involved with the confinement of constraints in order to ensure the feasibility of the problem is achieved, i.e., the steps ((I)) and ((II)) are continuously applied until the convergence of the output vector $\boldsymbol{v}$. In optimization terminology this method is therefore a gradient restoration algorithm with a fixed step size.

Therefore, according to Fig. 1 each iteration has two distinct stages. First, as described in Step ((III)), $\boldsymbol{v}$ is updated using the gradient of the term E^{op} alone. Second, after each updating, $\boldsymbol{v}$ is projected in the valid subspace. This is an iterative process, in which $\boldsymbol{v}$ is first orthogonally projected in the valid subspace (8), and then thresholded so that its elements lie in the range $[lim_i^{\inf}, lim_i^{\sup}]$. The convergence process is concluded when the values of vout during two successive loops remain practically constant, where the value of vout in this case is equal to $\boldsymbol{v}$.

3. Mapping optimization problems by the modified Hopfield network

In this section, the formulation of three types of optimization problems, namely combinatorial optimization problems, dynamic programming problems and nonlinear optimization problems, is presented.

3.1 Notation and definitions

The notation employed for vectors and matrices, which are used for mapping combinatorial optimization problems and dynamic programming problems, is as follows.

- The vector $\boldsymbol{p} \in \Re^n$ represents the solution set of an optimization problem consisted of n nodes (neurons). Thus, the elements belonging to $\boldsymbol{p}$ have integer elements defined by:

$$p_i \in \{1,...,n\} \text{ where } i \in \{1..n\} \tag{12}$$

The vector $\boldsymbol{p}$ can be represented by a vector $\boldsymbol{v}$, composed of ones and zeros, which represents the output of the network. In the notation using Kronecher products (Graham, 1981), we have:

- δ is a matrix ($\delta \in \Re^{n\text{x}n}$) defined by:

$$\delta_{ij} = \begin{cases} 1, \text{ if } i = j \\ 0, \text{ if } i \neq j \end{cases} \tag{13}$$

$\delta(k) \in \Re^n$ is a column vector corresponding to k-th column of δ.

- $\boldsymbol{v}(\boldsymbol{p})$ is an $n.m$ dimensional vector representing the form of the final network output vector $\boldsymbol{v}$, which corresponds to the problem solution denoted by $\boldsymbol{p}$. The vector $\boldsymbol{v}(\boldsymbol{p})$ is defined by:

$$\boldsymbol{v}(\boldsymbol{p}) = \begin{bmatrix} \delta(p_1) \\ \delta(p_2) \\ \vdots \\ \delta(p_n) \end{bmatrix} \tag{14}$$

- vec($\boldsymbol{U}$) is a function which maps the mxn matrix $\boldsymbol{U}$ to the $n.m$-element vector $\boldsymbol{v}$. This function is defined by:

$$\boldsymbol{v} = \text{vec}(\boldsymbol{U}) = [U_{11}\, U_{21} ... U_{m1} \quad U_{12}\, U_{22} ... U_{m2} \quad U_{1n}\, U_{2n} ... U_{mn}]^T \tag{15}$$

- $\boldsymbol{V}(\boldsymbol{p})$ is an nxn dimensional matrix defined by:

$$\boldsymbol{V}(\boldsymbol{p}) = \begin{bmatrix} \delta(p_1)^T \\ \delta(p_2)^T \\ \vdots \\ \delta(p_n)^T \end{bmatrix} \tag{16}$$

where $[\boldsymbol{V}(\boldsymbol{p})]_{ij} = [\delta(p_i)]_j$.

- $\boldsymbol{P}\otimes\boldsymbol{Q}$ denotes the Kronecher product of two matrices. If $\boldsymbol{P}$ is an nxn matrix, and $\boldsymbol{Q}$ is an mxm matrix, then ($\boldsymbol{P}\otimes\boldsymbol{Q}$) is an $(n.m)$x$(n.m)$ matrix given by:

$$\boldsymbol{P} \otimes \boldsymbol{Q} = \begin{bmatrix} P_{11}\boldsymbol{Q} & P_{12}\boldsymbol{Q} & \cdots & P_{1n}\boldsymbol{Q} \\ P_{21}\boldsymbol{Q} & P_{22}\boldsymbol{Q} & \cdots & \mathrm{P}_{2n}\boldsymbol{Q} \\ \vdots & \vdots & \ddots & \\ P_{n1}\boldsymbol{Q} & P_{n2}\boldsymbol{Q} & \cdots & P_{nn}\boldsymbol{Q} \end{bmatrix} \tag{17}$$

- $\boldsymbol{w}\otimes\boldsymbol{h}$ denotes the Kronecher product of two vectors. If $\boldsymbol{w}$ is an n-element vector and $\boldsymbol{h}$ an m-element vector, then ($\boldsymbol{w}\otimes\boldsymbol{h}$) is an $n.m$-element vector given by:

$$\boldsymbol{w} \otimes \boldsymbol{h} = \begin{bmatrix} w_1 .\boldsymbol{h} \\ w_2 .\boldsymbol{h} \\ \vdots \\ w_n .\boldsymbol{h} \end{bmatrix} \tag{18}$$

The properties of the Kronecher products (Graham, 1981) utilized are:

$$(\lambda \boldsymbol{w} \otimes \gamma \boldsymbol{h}) = \lambda\gamma(\boldsymbol{w} \otimes \boldsymbol{h}) \tag{19}$$

$$(\boldsymbol{w} \otimes \boldsymbol{h})^T(\boldsymbol{x} \otimes \boldsymbol{g}) = (\boldsymbol{w}^T\boldsymbol{x})(\boldsymbol{h}^T\boldsymbol{g}) \tag{20}$$

$$(\boldsymbol{P} \otimes \boldsymbol{Q})(\boldsymbol{w} \otimes \boldsymbol{h}) = (\boldsymbol{Pw} \otimes \boldsymbol{Qh}) \tag{21}$$

$$(\boldsymbol{P} \otimes \boldsymbol{Q})(\boldsymbol{E} \otimes \boldsymbol{F}) = (\boldsymbol{PE} \otimes \boldsymbol{QF}) \tag{22}$$

$$\text{vec}(\boldsymbol{Q}.\boldsymbol{V}.\boldsymbol{P}^{\text{T}}) = (\boldsymbol{P} \otimes \boldsymbol{Q}).\text{vec}(\boldsymbol{V}) \tag{23}$$

- $\boldsymbol{o}^n$ and $\boldsymbol{O}^n$ are respectively the n-element vector and the nxn matrix of ones, that is:

$$\left.\begin{aligned} [\boldsymbol{o}]_i &= 1 \\ [\boldsymbol{O}]_{ij} &= 1 \end{aligned}\right\} \text{for } i, j \in \{1..n\} \tag{24}$$

- $\boldsymbol{R}^n$ is an nxn projection matrix (i.e., $\boldsymbol{R}^n.\boldsymbol{R}^n = \boldsymbol{R}^n$) defined by:

$$\boldsymbol{R}^n = \boldsymbol{I}^n - \frac{1}{n}\boldsymbol{O}^n \tag{25}$$

The sum of the elements of each row of a matrix is transformed to zero by post-multiplication with $\boldsymbol{R}^n$, while pre-multiplication by $\boldsymbol{R}^n$ has the effect of setting the sum of the elements of each column to zero.

3.2 Formulation of combinatorial optimization problems

The combinatorial optimization problem considered in this chapter is the matching problem in bipartite graphs. However, several other types of combinatorial optimization problems, such as the salesman and N-queens problems, can be also solved by the proposed neural approach.

A graph G is a pair $G = (V,E)$, where V is a finite set of $2n$ nodes or vertices and E has as elements subsets of V of cardinality two called edges (Papadimitriou & Steiglitz, 1982). A matching M of a graph $G = (V,E)$ is a subset of the edges with the property that no two edges of M share the same node. The graph $G = (V,E)$ is called bipartite if the set of vertices V can be partitioned into two sets of n nodes, U and W, and each edge in E has one vertex in U and one vertex in W.

For each edge $[u_i, w_j] \in E$ is given a number $P_{ij} \geq 0$ called the connection weight of $[u_i, w_j]$. The goal of the matching problem in bipartite graphs is to find a matching of G with the minimum total sum of weights. Several problems, such as pattern recognition in computational vision, processes involving signal transmission, design of thin film circuits and schedule of operation processes, can be modeled as a matching problem in bipartite graphs.

As an example, for a bipartite graph with four nodes ($2n = 4$) represented in Fig. 2, the sets V, E, U, W and the matrix $\boldsymbol{P}$ are given by:

$$V = \{u_1, u_2, w_1, w_2\} \tag{26}$$

$$E = \{[u_1,w_1], [u_1,w_2], [u_2,w_1], [u_2,w_2]\} \tag{27}$$

$$U = \{u_1, u_2\} \tag{28}$$

$$W = \{w_1, w_2\} \tag{29}$$

$$\boldsymbol{P} = \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix} = \begin{bmatrix} 2.9 & 0.8 \\ 1.3 & 3.5 \end{bmatrix} \tag{30}$$

In this case, the minimum bipartite graph represented by the matching M will be chosen either by the subset $M_1 = \{[u_1,w_1], [u_2,w_2]\}$ or $M_2 = \{[u_1,w_2], [u_2,w_1]\}$. As the sum of the edges of M_2 is lower than that of M_1, then the subset M_2 corresponds to minimum bipartite graph, i. e., $M = M_2$.

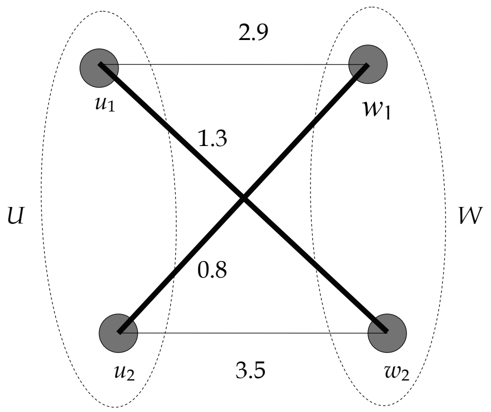


Fig. 2. Bipartite graph composed by four nodes.

In order to represent the association between nodes of U and W belonging to matching M, we have used the vector $\boldsymbol{p} \in \Re^n$, where the element $p_i \in \{1,\ldots,n\}$ represents the edge linking the i-th node of U to respective node of W, which is given by the own value of p_i. Using the definitions presented in subsection 3.1, for the matching problem illustrated in Fig. 2 the values of $\boldsymbol{p}$, $\boldsymbol{v}(\boldsymbol{p})$ and $\boldsymbol{V}(\boldsymbol{p})$ representing the solution given by M are defined by:

$$\boldsymbol{p} = [2 \quad 1]^T \tag{31}$$

$$\boldsymbol{v}(\boldsymbol{p}) = [\underbrace{0 \quad 1}_{\delta(p_1)^T} \quad \underbrace{1 \quad 0}_{\delta(p_2)^T}]^T \tag{32}$$

$$\boldsymbol{V}(\boldsymbol{p}) = \begin{bmatrix} \delta(p_1)^T \\ \delta(p_2)^T \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{33}$$

The equations of $\boldsymbol{T}^{conf}$ and $\boldsymbol{i}^{conf}$ are developed to force the validity of the structural constraints. These constraints mean that each edge in *E* has just one activated node in *U* and one activated node in *W*. Using the matrix $\boldsymbol{V}(\boldsymbol{p})$ to represent the structural constraints, we have:

$$[\boldsymbol{V}(\boldsymbol{p})]_{ij} \in \{1,0\}$$

$$\sum_{j=1}^{n} [\boldsymbol{V}(\boldsymbol{p})]_{ij} = 1 \tag{34}$$

In this case, a valid subspace for the matching problem in bipartite graphs can be represented by the following relationship:

$$\boldsymbol{I}^{conf} = \boldsymbol{V} = \frac{1}{n}\,\boldsymbol{o}^{n}.\boldsymbol{o}^{n^T} \tag{35}$$

It is now necessary to guarantee that the sum of the elements of each line of the matrix $\boldsymbol{V}$ takes value equal to 1. This procedure is represented in the modified Hopfield network by the projection matrix $\boldsymbol{T}^{conf}$, i.e., the multiplication of $\boldsymbol{T}^{conf}$ by $\boldsymbol{V}$ should also guarantee these constraints. Using the properties of the matrix $\boldsymbol{R}^{n}$, we have:

$$\boldsymbol{V}.\boldsymbol{R}^{n} = \boldsymbol{T}^{conf}.\boldsymbol{V} \tag{36}$$

$$\mathbf{I}^{n}.\mathbf{V}.\mathbf{R}^{n} = \mathbf{T}^{conf}.\mathbf{V} \tag{37}$$

Using (35) and (37) in equation of the valid subspace ($\boldsymbol{V} = \boldsymbol{T}^{conf}.\boldsymbol{V} + \boldsymbol{I}^{conf}$),

$$\boldsymbol{V} = \boldsymbol{I}^{n}.\boldsymbol{V}.\boldsymbol{R}^{n} + \frac{1}{n}\,\boldsymbol{o}^{n}.\boldsymbol{o}^{n^T} \tag{38}$$

Applying operator vec(.) given by (23) in (38),

$$\text{vec}(\boldsymbol{V}) = \text{vec}(\boldsymbol{I}^{n}.\boldsymbol{V}.\boldsymbol{R}^{n}) + \frac{1}{n}\,\text{vec}(\boldsymbol{o}^{n}.1.\boldsymbol{o}^{n^T})$$

$$\text{vec}(\boldsymbol{V}) = (\boldsymbol{I}^{n} \otimes \boldsymbol{R}^{n}).\text{vec}(\boldsymbol{V}) + \frac{1}{n}\,(\boldsymbol{o}^{n} \otimes \boldsymbol{o}^{n}) \tag{39}$$

Changing vec($\boldsymbol{V}$) by $\boldsymbol{v}$ in equation (39), we have:

$$\boldsymbol{v}(t+1) = (\boldsymbol{I}^{n} \otimes \boldsymbol{R}^{n}).\boldsymbol{v}(t) + \frac{1}{n}\,(\boldsymbol{o}^{n} \otimes \boldsymbol{o}^{n}) \tag{40}$$

Thus, comparing (40) and (8) the parameters $\boldsymbol{T}^{conf}$ and $\boldsymbol{i}^{conf}$ are given by:

$$\boldsymbol{T}^{conf} = (\boldsymbol{I}^{n} \otimes \boldsymbol{R}^{n}) \tag{41}$$

$$\boldsymbol{i}^{conf} = \frac{1}{n}\,(\boldsymbol{o}^{n} \otimes \boldsymbol{o}^{n}) \tag{42}$$

Equations (41) and (42) satisfy the properties of the valid subspace, i.e., $\boldsymbol{T}^{conf}.\boldsymbol{T}^{conf} = \boldsymbol{T}^{conf}$ and $\boldsymbol{T}^{conf}.\boldsymbol{i}^{conf} = \mathbf{0}$. In relation to example illustrated in Fig. 2 the matrix $\boldsymbol{T}^{conf}$ and the vector $\boldsymbol{i}^{conf}$ are respectively given by:

$$\boldsymbol{T}^{conf} = \begin{bmatrix} 0.5 & -0.5 & 0 & 0 \\ -0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & -0.5 \\ 0 & 0 & -0.5 & 0.5 \end{bmatrix} \tag{43}$$

$$\boldsymbol{i}^{conf} = [0.5 \quad 0.5 \quad 0.5 \quad 0.5]^T \tag{44}$$

The energy function E^{op} of the modified Hopfield network for the matching problem in bipartite graphs is projected in order to find a solution corresponding to the minimum total sum $\xi(\boldsymbol{p})$ referent to the values P_{ij} associated with each edges of M, which is defined by:

$$E^{op} = \xi(\boldsymbol{p}) = \text{trace}(\boldsymbol{V}(\boldsymbol{p})^T.\boldsymbol{P}) \tag{45}$$

In this case, when E^{op} is minimized, the optimal solution corresponds to the minimum energy state of the network. The parameters $\boldsymbol{T}^{op}$ and $\boldsymbol{i}^{op}$ are then obtained from the corresponding cost constraint given by above equation. Using the properties of Kronecher product in (45), we have:

$$E^{op} = \text{vec}(\boldsymbol{V}(\boldsymbol{p})^T).\text{vec}(\boldsymbol{P}) = \boldsymbol{v}(\boldsymbol{p})^T.\text{vec}(\boldsymbol{P}) \tag{46}$$

Comparing (46) and (7), the parameters $\boldsymbol{T}^{op}$ and $\boldsymbol{i}^{op}$ are given by:

$$\boldsymbol{T}^{op} = \mathbf{0} \tag{47}$$

$$\boldsymbol{i}^{op} = -\text{vec}(\boldsymbol{P}) \tag{48}$$

Using the definition of vec(.) provided in (15), the vector $\boldsymbol{i}^{op}$ in relation to example illustrated in Fig. 2 is given by:

$$\boldsymbol{i}^{conf} = [-2.9 \quad -1.3 \quad -0.8 \quad -3.5]^T \tag{49}$$

To illustrate the performance of the proposed neural network, some simulation results involving the matching problem in bipartite graphs are presented in Section 4.

3.3 Formulation of dynamic programming problems

A typical dynamic programming problem can be modeled as a set of source and destination nodes with n intermediate stages, m states in each stage, and metric data $d_{xi,(x+1)j}$, where x is the index of the stages, and i and j are the indices of the states in each stage (Hillier & Lieberman, 1980). The goal of the dynamic programming problem considered in this chapter is to find a valid path which starts at the source node, visits one and only one state node in each stage, reaches the destination node, and has a minimum total length (cost) among all possible paths.

The equations of $\boldsymbol{T}^{conf}$ and $\boldsymbol{i}^{conf}$ are developed to force the validity of the structural constraints. These constraints, for dynamic programming problems, mean that one and only one state in each stage can be actived. Thus, the matrix $\boldsymbol{V}(\boldsymbol{p})$ is defined by:

$$[\boldsymbol{V}(\boldsymbol{p})]_{ij} \in \{1,0\}$$

$$\sum_{j=1}^{m} [\boldsymbol{V}(\boldsymbol{p})]_{ij} = 1 \tag{50}$$

A valid subspace ($\boldsymbol{V}=\boldsymbol{T}^{val}.\boldsymbol{V} + \boldsymbol{I}^{conf}$) for the dynamic programming problem can be represented by:

$$\boldsymbol{I}^{conf} = \boldsymbol{V} = \frac{1}{m}\,\boldsymbol{o}^{n}.\boldsymbol{o}^{m^{T}} \tag{51}$$

Equation (51) guarantees that the sum of the elements of each line of the matrix $\boldsymbol{V}$ takes values equal to 1. Therefore, the term $\boldsymbol{T}^{conf}.\boldsymbol{V}$ must also guarantee that the sum of the elements of each line of the matrix $\boldsymbol{V}$ takes value equal to zero. Using the properties of the matrix $\boldsymbol{R}^{n}$, we have:

$$\boldsymbol{V}.\boldsymbol{R}^{m} = \boldsymbol{T}^{conf}.\boldsymbol{V}$$

$$\boldsymbol{I}^{n}.\boldsymbol{V}.\boldsymbol{R}^{m} = \boldsymbol{T}^{conf}.\boldsymbol{V} \tag{52}$$

Using (51) and (52) in equation of the valid subspace ($\boldsymbol{V} = \boldsymbol{T}^{conf}.\boldsymbol{V} + \boldsymbol{I}^{conf}$),

$$\boldsymbol{V} = \boldsymbol{I}^{n}.\boldsymbol{V}.\boldsymbol{R}^{m} + \frac{1}{m}\,\boldsymbol{o}^{n}.\boldsymbol{o}^{m^{T}} \tag{53}$$

Applying operator vec(.) given by (23) in (53),

$$\mathrm{vec}(\boldsymbol{V}) = \mathrm{vec}(\boldsymbol{I}^{n}.\boldsymbol{V}.\boldsymbol{R}^{m}) + \frac{1}{m}\,\mathrm{vec}(\boldsymbol{o}^{n}.1.\boldsymbol{o}^{m^{T}})$$

$$\mathrm{vec}(\boldsymbol{V}) = (\boldsymbol{I}^{n} \otimes \boldsymbol{R}^{m}).\mathrm{vec}(\boldsymbol{V}) + \frac{1}{m}\,(\boldsymbol{o}^{n} \otimes \boldsymbol{o}^{m}) \tag{54}$$

Changing vec($\boldsymbol{V}$) by $\boldsymbol{v}$ in equation (54), we have:

$$\boldsymbol{v}(t+1) = (\boldsymbol{I}^{n} \otimes \boldsymbol{R}^{m}).\boldsymbol{v}(t) + \frac{1}{m}\,(\boldsymbol{o}^{n} \otimes \boldsymbol{o}^{m}) \tag{55}$$

Thus, comparing (55) and (8) the parameters $\boldsymbol{T}^{conf}$ and $\boldsymbol{i}^{conf}$ are given by:

$$\boldsymbol{T}^{conf} = (\boldsymbol{I}^{n} \otimes \boldsymbol{R}^{m}) \tag{56}$$

$$\boldsymbol{I}^{conf} = \frac{1}{m}\,(\boldsymbol{o}^{n} \otimes \boldsymbol{o}^{m}) \tag{57}$$

Equations (56) and (57) satisfy the properties of the valid subspace, i.e., $\boldsymbol{T}^{conf}.\boldsymbol{T}^{conf} = \boldsymbol{T}^{conf}$ and $\boldsymbol{T}^{conf}.\boldsymbol{i}^{conf} = \boldsymbol{0}$.

The energy function E^{op} of the modified Hopfield network for the dynamic programming problem, which is defined in (58), is projected to find a minimum path among all possible paths. In this equation, the first term defines the weight (metric cost) of the connection linking the *i*-th neuron of stage *x* to the *j*-th neuron of the following stage (*x*+1). The second term defines the weight of the connection linking the *i*-th neuron of stage *x* to the *j*-th neuron of previous stage (*x*–1). The third term provides the weight of the connection linking the source node to all others nodes of the first stage, while the fourth term provides the weight of the connection linking the destination to all other nodes of the last stage. When E^{op} is minimized, the optimal solution corresponds to the minimum energy state of the network.

$$E^{op} = \frac{1}{4}[\sum_{x=1}^{n-1}\sum_{i=1}^{m}\sum_{j=1}^{m}\underbrace{d_{xi,(x+1)j}.v_{xi}.v_{(x+1)j}}_{1st.\text{Term}} + \sum_{x=2}^{n}\sum_{i=1}^{m}\sum_{j=1}^{m}\underbrace{d_{(x-1)j,xi}.v_{xi}.v_{(x-1)j}}_{2nd.\text{Term}}] + \\ +[\sum_{x=1}^{1}\sum_{i=1}^{m}\underbrace{d_{source,xi}.v_{xi}}_{3rd.\text{Term}} + \sum_{x=n}^{n}\sum_{i=1}^{m}\underbrace{d_{xi,destination}.v_{xi}}_{4th.\text{Term}}] \tag{58}$$

Therefore, optimization of E^{op} corresponds to minimizing each term given by (58) in relation to v_{xi}. From (58), the matrix $\boldsymbol{T}^{op}$ and vector $\boldsymbol{i}^{op}$ can be given by:

$$[\boldsymbol{T}^{op}]_{pq} = -[\boldsymbol{P}]_{xi,yj}.[\boldsymbol{Q}]_{xy} \quad \begin{cases} [\boldsymbol{P}]_{xi,yj} = \frac{1}{2}d_{xi,yj} \\ [\boldsymbol{Q}]_{xy} = \delta_{(x+1)y} + \delta_{(x-1)y} \end{cases} \tag{59}$$

$$\boldsymbol{i}^{op} = -[\underbrace{d_{source,11} \;\; \text{d}_{\text{source,12}} \cdots \text{d}_{\text{source,1m}}}_{m} \; \underbrace{0 \;\; 0 \ldots 0 \;\; 0}_{m.(n\text{-}2)} \\ \underbrace{d_{n1,\text{destination}} \;\; \text{d}_{n2,\text{destination}} \cdots d_{nm,destination}}_{m}] \tag{60}$$

where: $\boldsymbol{T}^{op} \in \Re^{nmxnm}$; $\boldsymbol{i}^{op} \in \Re^{n.m}$; $p = m.(x - 1) + i$; $q = m.(y - 1) + j$; $x, y \in \{2..n - 1\}$; $i, j \in \{1..m\}$.
In the next subsection, the formulation of nonlinear optimization problems by the modified Hopfield network is presented.

3.4 Formulation of nonlinear optimization problems

Consider the following general nonlinear optimization problem, with *m*-constraints and *n*-variables, given by the following equations:

$$\text{Minimize: } E^{op}(\boldsymbol{v}) = f(\boldsymbol{v}) \tag{61}$$

$$\text{subject to: } E^{conf}(\boldsymbol{v}): h_i(\boldsymbol{v}) \leq \boldsymbol{0}, \quad i \in \{1..m\} \tag{62}$$

$$\boldsymbol{z}^{min} \leq \boldsymbol{v} \leq \boldsymbol{z}^{max} \tag{63}$$

where $\boldsymbol{v}$, $\boldsymbol{z}^{min}$, $\boldsymbol{z}^{max} \in \Re^n$; $f(\boldsymbol{v})$ and $h_i(\boldsymbol{v})$ are continuous, and all first and second order partial derivatives of $f(\boldsymbol{v})$ and $h_i(\boldsymbol{v})$ exist and are continuous. The vectors $\boldsymbol{z}^{min}$ and $\boldsymbol{z}^{max}$ define the bounds on the variables belonging to the vector $\boldsymbol{v}$. The conditions in (62) and (63) define a bounded polyhedron. The vector $\boldsymbol{v}$ must remain within this polyhedron if it is to represent a valid solution for the optimization problem (61). A solution can be obtained by a modified Hopfield network, whose valid subspace guarantees the satisfaction of condition (62). Moreover, the initial hypercube represented by the inequality constraints in (63) is directly defined by the 'symmetric ramp' function given in (9), which is used as neural activation function, i.e. $\boldsymbol{v} \in [\mathbf{z}^{min}, \mathbf{z}^{max}]$.

The parameters $\boldsymbol{T}^{conf}$ and $\boldsymbol{i}^{conf}$ are calculated by transforming the inequality constraints in (62) into equality constraints by introducing a slack variable $\boldsymbol{w} \in \Re^n$ for each inequality constraint:

$$h_i(\boldsymbol{v}) + \sum_{j=1}^{m} \delta_{ij}.w_j = 0 \tag{64}$$

where w_j are slack variables, treated as the variables v_i , and δ_{ij} is defined by the Kronecker impulse function:

$$\delta_{ij} = \begin{cases} 1\ , \text{ if } i = j \\ 0\ , \text{ if } i \neq j \end{cases} \tag{65}$$

After this transformation, the problem defined by equations (61), (62) and (63) can be rewritten as:

$$\text{Minimize: } E^{op}(\boldsymbol{v}^+) = f(\boldsymbol{v}^+) \tag{66}$$

$$\text{subject to: } E^{conf}(\boldsymbol{v})\text{: } h_i(\boldsymbol{v}^+) \leq 0\ ,\quad i \in \{1..m\} \tag{67}$$

$$z_i^{min} \leq v_i^+ \leq z_i^{max}\ ,\ i \in \{1..n\} \tag{68}$$

$$0 \leq v_i^+ \leq z_i^{max}\ ,\ i \in \{n+1..N\} \tag{69}$$

where $N = n + m$, and $\boldsymbol{v}^+ = [\boldsymbol{v}^T\ \ \boldsymbol{w}^T]^T \in \Re^N$ is a vector of extended variables. Note that E^{op} does not depend on the slack variables $\boldsymbol{w}$. Also an equality constraint of the form $h_i(.) = 0$ is incorporated in the above optimization problem by transforming into two inequalities, i.e., $h_i(.) \leq 0$ and $h_i(.) \geq 0$.

The projection matrix $\boldsymbol{T}^{conf}$ belonging to the valid-subspace equation given in (8) is obtained from the projection of $\boldsymbol{v}^+$, which is obtained after a minimization step of $E^{op}(\boldsymbol{v}^+)$, onto the tangent subspace of the surface bounded by constraints given by (67). In Luenberger (1984), it has been shown that a projection matrix to the system defined in (67) is given by:

$$\boldsymbol{T}^{conf} = \boldsymbol{I} - \nabla\boldsymbol{h}(\boldsymbol{v}^+)^T.(\nabla\boldsymbol{h}(\boldsymbol{v}^+).\nabla\boldsymbol{h}(\boldsymbol{v}^+)^T)^{-1}.\nabla\boldsymbol{h}(\boldsymbol{v}^+) \tag{70}$$

where:

$$\nabla\boldsymbol{h}(\boldsymbol{v}^+) = \begin{bmatrix} \dfrac{\partial h_1(\boldsymbol{v}^+)}{\partial v_1^+} & \dfrac{\partial h_1(\boldsymbol{v}^+)}{\partial v_2^+} & \cdots & \dfrac{\partial h_1(\boldsymbol{v}^+)}{\partial v_N^+} \\ \dfrac{\partial h_2(\boldsymbol{v}^+)}{\partial v_1^+} & \dfrac{\partial h_2(\boldsymbol{v}^+)}{\partial v_2^+} & \cdots & \dfrac{\partial h_2(\boldsymbol{v}^+)}{\partial v_N^+} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial h_m(\boldsymbol{v}^+)}{\partial v_1^+} & \dfrac{\partial h_m(\boldsymbol{v}^+)}{\partial v_2^+} & \cdots & \dfrac{\partial h_m(\boldsymbol{v}^+)}{\partial v_N^+} \end{bmatrix} = \begin{bmatrix} \nabla\boldsymbol{h}_1(\boldsymbol{v}^+)^T \\ \nabla\boldsymbol{h}_2(\boldsymbol{v}^+)^T \\ \vdots \\ \nabla\boldsymbol{h}_m(\boldsymbol{v}^+)^T \end{bmatrix} \tag{71}$$

Inserting the value of (70) in the expression of the valid subspace in (8), we have:

$$\boldsymbol{v}^{+} \leftarrow [\boldsymbol{I} - \nabla \boldsymbol{h}(\boldsymbol{v}^{+})^{T}.(\nabla \boldsymbol{h}(\boldsymbol{v}^{+}).\nabla \boldsymbol{h}(\boldsymbol{v}^{+})^{T})^{-1}.\nabla \boldsymbol{h}(\boldsymbol{v}^{+})].\ \boldsymbol{v}^{+} + \boldsymbol{i}^{conf} \qquad (72)$$

Results of the Lyapunov stability theory (Vidyasagar, 1993) should be used in (72) to guarantee the stability of the nonlinear system, and consequently, to force the network convergence to equilibrium points that represent a feasible solution to the nonlinear system. By the definition of the Jacobean, when $\boldsymbol{v}$ leads to equilibrium point implicates in $\boldsymbol{v}^{e} = \boldsymbol{0}$. In this case, the value of $\boldsymbol{i}^{conf}$ should also be null to satisfy the equilibrium condition, i. e., $\boldsymbol{v}^{e} = \boldsymbol{v}(t) = \boldsymbol{v}(t+1) = \boldsymbol{0}$. Thus, $\boldsymbol{h}(\boldsymbol{v}^{+})$ given in equation (72) can be approximated as follows:

$$\boldsymbol{h}(\boldsymbol{v}^{+}) \approx \boldsymbol{h}(\boldsymbol{v}^{e}) + \boldsymbol{J}.(\ \boldsymbol{v}^{+} - \boldsymbol{v}^{e}) \qquad (73)$$

where $\boldsymbol{J} = \nabla \boldsymbol{h}(\boldsymbol{v}^{+})$ and $\boldsymbol{h}(\boldsymbol{v}^{+}) = [h_1(\boldsymbol{v}^{+})\ \ h_2(\boldsymbol{v}^{+}) \ldots h_m(\boldsymbol{v}^{+})]^{T}$.

In the proximity of the equilibrium point $\boldsymbol{v}^{e} = \boldsymbol{0}$, we obtain the following equation related to the parameters $\boldsymbol{v}^{+}$ and $\boldsymbol{h}(\boldsymbol{v}^{+})$:

$$\lim_{\boldsymbol{v}^{+} \to \boldsymbol{v}^{e}} \left\| \frac{\boldsymbol{h}(\boldsymbol{v}^{+})}{\boldsymbol{v}^{+}} \right\| = \boldsymbol{0} \qquad (74)$$

Finally, introducing the results derived from (73) and (74) in equation given by (72), we obtain:

$$\boldsymbol{v}^{+} \leftarrow \boldsymbol{v}^{+} - \nabla \boldsymbol{h}(\boldsymbol{v}^{+})^{T}.(\nabla \boldsymbol{h}(\boldsymbol{v}^{+}).\nabla \boldsymbol{h}(\boldsymbol{v}^{+})^{T})^{-1}.\boldsymbol{h}(\boldsymbol{v}^{+}) \qquad (75)$$

Therefore, equation (75) synthesizes the valid-subspace expression for treating systems of nonlinear equations. In this case, for nonlinear optimization problems the original valid-subspace equation given in (8), which is represented by step ((I)) in Fig. 1, should be substituted by equation (75). Thus, according to Fig. 1, successive applications of the step ((I)) followed by the step ((II)) make $\boldsymbol{v}^{+}$ convergent to a point that satisfies all constraints imposed to the nonlinear optimization problem.

The parameters $\boldsymbol{T}^{op}$ and $\boldsymbol{i}^{op}$ associated to the energy function E^{op}, which is given by (7) and represented in (66), should be defined so that the optimal solution corresponds to the minimization of E^{op}. This procedure can be implemented by updating the vector $\boldsymbol{v}^{+}$ in the opposite gradient direction that of the energy function E^{op}. Since conditions (66)-(69) define a bounded polyhedron, the objective function (66) has always a minimum. Thus, the equilibrium points of the network can be calculated by assuming the following values to $\boldsymbol{T}^{op}$ and $\boldsymbol{i}^{op}$:

$$\boldsymbol{i}^{op} = -\left[\frac{\partial f(v^{+})}{\partial v_1^{+}}\ \frac{\partial f(v^{+})}{\partial v_2^{+}} \cdots \frac{\partial f(v^{+})}{\partial v_N^{+}} \right]^{T} \qquad (76)$$

$$\boldsymbol{T}^{op} = \boldsymbol{0} \qquad (77)$$

According to mentioned previously, the vector $\boldsymbol{v}^{+}$ is composed by both vectors $\boldsymbol{v}$ and $\boldsymbol{w}$, i. e., $\boldsymbol{v}^{+} = [\boldsymbol{v}^{T}\ \ \boldsymbol{w}^{T}]^{T}$, then the vector $\boldsymbol{i}^{op}$ given in (76) can be also represented by:

$$\boldsymbol{i}^{op} = -\left[\frac{\partial f(\boldsymbol{v}^+)}{\partial v_1} \; \frac{\partial f(\boldsymbol{v}^+)}{\partial v_2} \cdots \frac{\partial f(\boldsymbol{v}^+)}{\partial v_n} \; \frac{\partial f(\boldsymbol{v}^+)}{\partial w_1} \; \frac{\partial f(\boldsymbol{v}^+)}{\partial w_2} \cdots \frac{\partial f(\boldsymbol{v}^+)}{\partial w_m} \right]^T \tag{78}$$

As the optimization process of the cost function does not depend on the slack variables $\boldsymbol{w}$, equation (76) can then be replaced by the following one:

$$\boldsymbol{i}^{op} = -\left[\underbrace{\frac{\partial f(\boldsymbol{v}^+)}{\partial v_1} \; \frac{\partial f(\boldsymbol{v}^+)}{\partial v_2} \cdots \frac{\partial f(\boldsymbol{v}^+)}{\partial v_n}}_{n\text{-components}} \; \underbrace{0 \;\; 0 \;\; 0 \; \ldots \; 0}_{m\text{-components}} \right]^T \tag{79}$$

To illustrate the performance of the proposed neural network, some simulation results involving nonlinear optimizations problems are presented in the next section.

4. Simulation results

In this section, some simulation results are presented to illustrate the application of the neural network approach developed in the previous sections for solving combinatorial optimization problems, dynamic programming problems and nonlinear optimization problems.

4.1 Combinatorial optimization problems

The modified Hopfield network has been used in the solution of the matching problem proposed in Papadimitriou & Steiglitz (1982), with matrix $\boldsymbol{P}$ given by:

$$\boldsymbol{P} = \begin{bmatrix} 7 & 2 & 1 & 9 & 4 \\ 9 & 6 & 9 & 5 & 5 \\ 3 & 8 & 3 & 1 & 8 \\ 7 & 9 & 4 & 2 & 2 \\ 8 & 4 & 7 & 4 & 8 \end{bmatrix}$$

A graphical representation of this problem is illustrated in Fig. 3(a). The parameters $\boldsymbol{T}^{conf}$ and $\boldsymbol{i}^{conf}$ to be used in the modified Hopfield network illustrated in Fig. 1 are obtained using equations given in (41) and (42), while the parameters $\boldsymbol{T}^{op}$ and $\boldsymbol{i}^{op}$ are defined using (47) and (48). The elements of the vector $\boldsymbol{v}$ of the modified Hopfield network were randomly generated between 0 and 1.

The modified Hopfield network converged after 50 iterations, which is considered extremely fast when compared with other neural approaches used in combinatorial optimization. In comparative terms, the simulation of this problem by the conventional Hopfield network proposed in Hopfield & Tank (1985), using the same initial values for the output vector $\boldsymbol{v}$, reaches the final solution in 317 iterations. The edges set, representing the optimal solution, is given by {[1,3];[2,5];[3,1];[4,4];[5,2]}. The vectors $\boldsymbol{p}$ and $\boldsymbol{v}(\boldsymbol{p})$, and the matrix $\boldsymbol{V}(\boldsymbol{p})$ representing the obtained solution is provided by:

$$\boldsymbol{p} = [3 \ \ 5 \ \ 1 \ \ 4 \ \ 2]^T$$

$$\boldsymbol{v}(\boldsymbol{p}) = [\underbrace{0\,0\,1\,0\,0}_{\delta(p_1)^T} \ \underbrace{0\,0\,0\,0\,1}_{\delta(p_2)^T} \ \underbrace{1\,0\,0\,0\,0}_{\delta(p_3)^T} \ \underbrace{0\,0\,0\,1\,0}_{\delta(p_4)^T} \ \underbrace{0\,1\,0\,0\,0}_{\delta(p_5)^T}]^T$$

$$\boldsymbol{V}(\boldsymbol{p}) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Figure 3(b) illustrates the minimum bipartite graph representing the final solution obtained by the modified Hopfield network. Figure 4 shows the evolution of the matrix $\boldsymbol{V}$ during the convergence process of the network. The minimization of the energy term E^{op} guarantees the minimum total sum among all edges (E^{op} = 15), where the value of Δt used in (11) were assumed as 0.01.

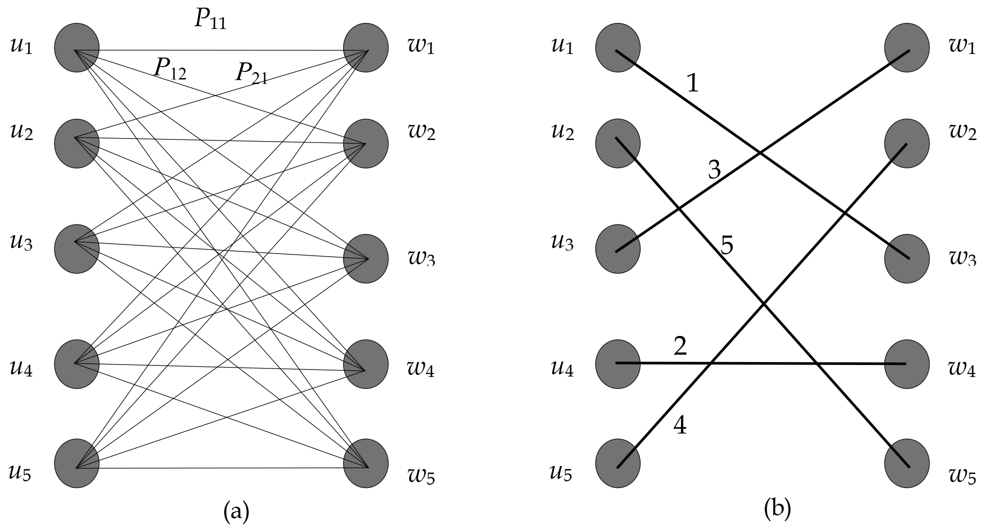


Fig. 3. Bipartite graph composed by ten nodes (a) and minimum bipartite graph (b).

4.2 Dynamic programming problems

The first dynamic programming problem to be solved by the modified Hopfield network is illustrated in Fig. 5, which is composed by three intermediate stages (n = 3) and two states in each stage (m = 2). The values of the weights $d_{xi,(x+1)j}$, which link the ith neuron of stage x to the jth neuron of the following stage (x+1), are also indicated in Fig. 5. The goal is to find the minimum path (from all possible paths), which starts at the source node and reaches the destination node, passing by only one state node in each stage.

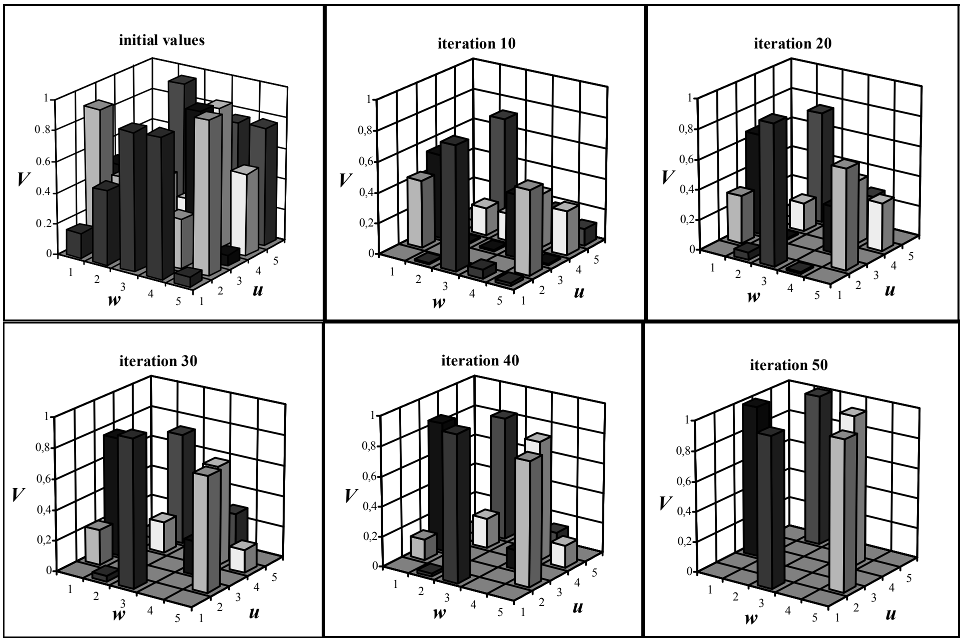


Fig. 4. Evolution of the matrix ***V*** for bipartite graph problem.

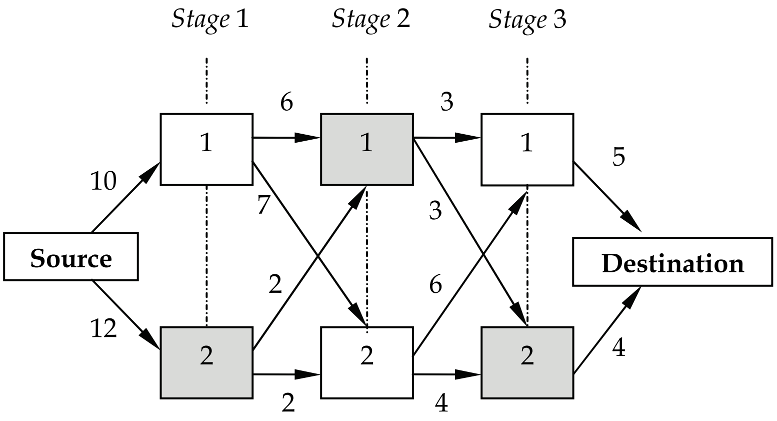


Fig. 5. The dynamic programming problem (m = 3 and n =2)

For this example the total number of possible paths is equal to 8, which is obtained by m^n. The optimal solution is given by the shaded states, i.e., state 2 in stage 1, state 1 in stage 2, and state 2 in stage 3. The modified Hopfield network applied in this problem always converges after three iterations. The vectors $\boldsymbol{p}$ and $\boldsymbol{v}(\boldsymbol{p})$, and the matrix $\boldsymbol{V}(\boldsymbol{p})$ representing the obtained solution are as follows:

$$\boldsymbol{p} = [2 \;\; 1 \;\; 2]^T$$

$$\boldsymbol{v}(\boldsymbol{p})^T = [0 \;\; 1 \quad 1 \;\; 0 \quad 0 \;\; 1]^T$$

$$\boldsymbol{V}(\boldsymbol{p}) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The minimization of the energy term E^{op} guarantees that the solution obtained represents the minimum path ($E^{op} = 21$) from all possible paths.

To illustrate that the proposed network can be used efficiently, various dynamic programming problems were simulated and the results compared with those obtained by the network proposed by Chiu et al. (1991). In this example, the number of stages and number of states has been increased step by step. The number of stages and number of states in each stage for the simulated examples were established by values belonging to the integer set defined by {2, 4, 8, 16, 32, 64}. The goal was to find a valid path, which starts at the source node, visits one and only node in each stage, and reaches the destination node, with the minimum possible total length. For such purposes, we have simulated both networks using the same initial values for the output vectors v, which were randomly generated between 0 and 1 for all instances treated in this comparison.

The weights of the connection $d_{xi,(x+1)j}$ linking nodes (states) of the network were randomly selected from the integer set {1, 3, 5, 7, 9}. For those instances with n and m less than 32, each example was simulated twenty times using random initial conditions. Examples with n and m greater than or equal to 32 were simulated ten times.

The performance analysis for both networks was done using the average normalized path length (D), which is given by:

$$D = \frac{S_c}{n_s.(n+1)} \tag{80}$$

where S_c is the sum of the selected paths after network convergence; n_s is the number of simulations; n is the number of stages.

The simulation results are shown in Table 1. In this table, the D^{MHN} and D^{CN} columns provide, respectively, the results of the average normalized path length for the modified Hopfield network and the one proposed by Chiu et al. (1991). This table shows that the modified Hopfield network presented better results with a shorter normalized path length. For checking the results obtained by the modified Hopfield network, simulations using conventional dynamic programming were also carried out using the same instances described in Table 1. In all analyzed instances, the values reached to the objective functions were practically identical in both approaches. However, the conventional method obtains the final solutions more rapidly than the modified Hopfield network. On the other hand, the implementation of dynamic programming problem to specialist systems in a neural network environment can be more easily made by using the modified Hopfield network. For all problems treated in this subsection, the values of Δt used in (11) were assumed as 0.01.

The adverse facts that can influence on the performance of the network proposed by Chiu et al. (1991) and explain their less accurate results are the following: i) optimization and constraint terms involved in problem mapping are treated in a single stage, ii) interference between optimization and constraint terms affects the precision of the equilibrium points, and iii) the convergence process of the network depend on the correct adjustment of the weighting constants associated with the energy terms. However, the modified Hopfield network presented here treats these terms in different stages. The terms $\boldsymbol{T}^{conf}$ and $\boldsymbol{i}^{conf}$ (belonging to E^{conf}) of the modified Hopfield network were developed to force the validity of the structural constraints associated with the dynamic programming problem, and the terms $\boldsymbol{T}^{op}$ and $\boldsymbol{i}^{op}$ were projected to find a minimum path among all possible paths.

Number of stages (n)	Number of states (m)	D^{MHN}	D^{CN}
2	2	3.13	3.25
4	4	2.03	3.12
8	8	1.34	2.00
16	16	1.06	1.85
32	32	1.03	1.61
64	64	1.02	1.39
16	2	3.14	3.21
16	4	1.79	2.98
16	8	1.26	1.85
16	32	1.13	1.79
2	16	1.17	1.53
4	16	1.02	1.60
8	16	1.09	1.76

Table 1. Simulation results (dynamic programming).

Thus, the main advantages of using a modified Hopfield network to solve dynamic programming problems are i) consideration of optimization and constraint terms in distinct stages with no interference with each other, ii) use of the unique energy term (E^{conf}) to group all constraints imposed on the problem, and iii) lack of need for adjustment of weighting constants for initialization. In all examples, the network output vector $\boldsymbol{v}$ was initialized with small random values defined between 0 and 1. It should be noticed that the increase in the number of states and stages does not degrade the performance of the network, but rather shows its efficiency.

4.3 Nonlinear optimization problems

In this subsection, we provide three examples to illustrate the effectiveness of the proposed architecture to solve nonlinear optimizations problems.

Example 1. Consider the following constrained optimization problem proposed in Bazaraa & Shetty (1979) in page 491, which is composed by inequality constraints and bounded variables:

$$\begin{aligned} \text{Min } f(v) &= e^{v_1} + v_1^2 + 4v_1 + 2v_2^2 - 6v_2 + 2v_3 \\ \text{subject to: } & v_1^2 + e^{v_2} + 6v_3 \le 15 \\ & v_1^4 - v_2 + 5v_3 \le 25 \\ & v_1^3 + v_2^2 - v_3 \le 10 \\ & 0 \le v_1 \le 4 \\ & 0 \le v_2 \le 2 \\ & v_3 \ge 0 \end{aligned}$$

This problem has a unique optimal solution $v^* = [0.0 \quad 1.5 \quad 0.0]^T$, and the minimal value of $f(v^*)$ at this point is equal to –3.5. Using a value of $\Delta t = 0.01$ in (11), which is corresponding to step ((III)) in Fig. 2, the solution vector (equilibrium point) obtained by the modified Hopfield network is given by $v = [0.0002 \quad 1.5001 \quad 0.0000]^T$, with $E(v) = f(v) = -3.499$. However, if we assume the value $\Delta t = 0.0001$ the network reaches the optimal solution v^*. Figure 6 shows the trajectories of the modified Hopfield network starting from $v^0 = [2.33 \quad 0.31 \quad 0.16]^T$ and converging towards the equilibrium point.

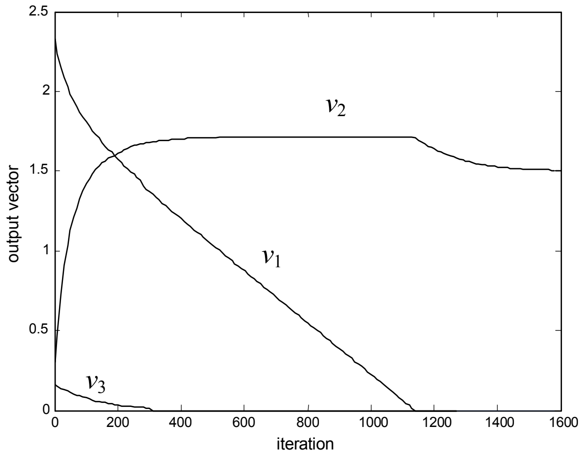


Fig. 6. Transient behavior of the modified Hopfield network in example 1.

To observe the global convergent behavior of the proposed network, we generated 15 initial points randomly distributed between 0 and 5. The bound constraints represented by the last three equations are directly mapped through the piecewise activation function defined in (9). All simulation results obtained by the modified Hopfield network show that the proposed architecture converges to v^*. The trajectories of the objective function starting from several initial points are illustrated in Fig. 7. All trajectories lead towards the same theoretical minimal value provided by $f(v^*) = -3.5$ when assumed $\Delta t = 0.0001$. These results show the efficiency of the modified Hopfield network for solving constrained nonlinear optimization problems.

A comparison using the SQP (Sequential Quadratic Programming) method and the modified Hopfield network was also done for this example. Both methods have found the same final solution. The SQP method reached the final solution in 35 iterations, whereas the modified Hopfield network needed 1587 iterations. However, convergence time to reach the final solution has not been directly proportional to number of iterations. For this example, using a microcomputer Pentium IV, the SQP method and the modified Hopfield network obtained the final solution in 3.65 and 5.86 seconds, respectively. This fact can be explained with respect to simplicity associated with the convergence process used by the modified Hopfield network, which consists of only three main steps as shown in Fig. 1. As well, as observed with the dynamic programming problems, the modified Hopfield network is an alternative method for solving constrained optimization problems and has the advantage of offering simplicity of implementation both in analogue hardware making use of operational amplifiers and in digital hardware by using digital signal processors.

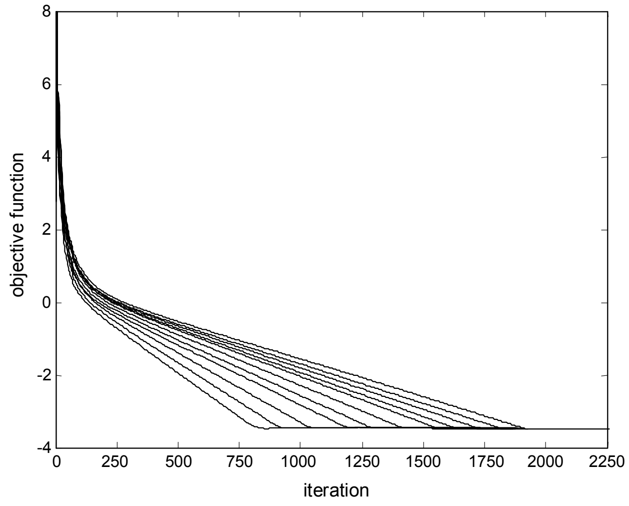


Fig. 7. Evolution of the objective function for 15 initial points in example 1.

To provide a more consistent analysis in relation to the efficiency of the proposed architecture, we make in the next example a comparison between the results produced by the modified Hopfield network with those provided by the network developed in Xia et al. (2002), and also by the topology presented in Kennedy & Chua (1988).

Example 2. Consider the following constrained optimization problem proposed in Xia et al. (2002), which is composed by inequality constraints:

$$\text{Min } f(v) = v_1^3 + (v_1 - 2v_2)^3 + e^{v_1+v_2}$$

$$\text{subject to: } v \in V$$

where $V = \{v \in \Re^2 \mid v_1^2 + v_2^2 \le 1\}$. This problem has a unique optimal solution given by v^* = $[-0.5159\ \ 0.8566]^T$ with $f(v^*) = -9.8075$. All simulation results provided by the modified Hopfield network show that it is convergent to v^*.

In Table 2, the results obtained by the modified Hopfield network using Δt = 0.0001 are compared with those provided by the projection neural network proposed in Xia et al. (2002), and also those given by the nonlinear circuit network developed in Kennedy & Chua (1988). Six different initial points were chosen, where two points {(1, 0); (0, -1)} are located in V and four {(-2, -2); (2, -2); (2, 2); (-2, 2)} are not in V. The results obtained by the modified Hopfield network are very close to the exact solution. The mean error between the solutions obtained by the network and the exact solution is less than 0.02%. We can verify that all solutions produced by the modified Hopfield network are quite stable.

According to Table 2 the nonlinear circuit network proposed in Kennedy & Chua (1988) can apparently approach v^* in only two cases. This was also observed in simulations performed in Xia et al. (2002). The projection neural network developed in Xia et al. (2002) produces solutions for all cases presented in Table 1, and we can observe that the final solutions depend on their initial values. It is also shown in table 1 that the modified Hopfield network, independently of the initial values of v, has converged to the same final values for all simulations. To illustrate the global convergent behavior of the modified Hopfield network, Fig. 8 shows the trajectories of v starting from several initial points.

Initial Vector	Modified Hopfield Network	Projection Neural Network	Nonlinear Circuit Network
$v^{(0)} = [\ 2\ \ 2]^T$	$v = [-0.5160\ \ 0.8566]^T$	$v = [-0.5160\ \ 0.8566]^T$	$v = [-0.5195\ \ 0.8641]^T$
$v^{(0)} = [-2\ \ 2]^T$	$v = [-0.5160\ \ 0.8566]^T$	$v = [-0.5160\ \ 0.8566]^T$	$v = [-0.5196\ \ 0.8641]^T$
$v^{(0)} = [-2\ \ -2]^T$	$v = [-0.5160\ \ 0.8566]^T$	$v = [-0.5161\ \ 0.8564]^T$	$\propto$
$v^{(0)} = [\ 2\ \ -2]^T$	$v = [-0.5160\ \ 0.8566]^T$	$v = [-0.5162\ \ 0.8563]^T$	$\propto$
$v^{(0)} = [\ 1\ \ 0]^T$	$v = [-0.5160\ \ 0.8566]^T$	$v = [-0.5162\ \ 0.8564]^T$	$\propto$
$v^{(0)} = [\ 0\ \ -1]^T$	$v = [-0.5160\ \ 0.8566]^T$	$v = [-0.5161\ \ 0.8564]^T$	$\propto$

Table 2. Comparison of the simulation results in example 2.

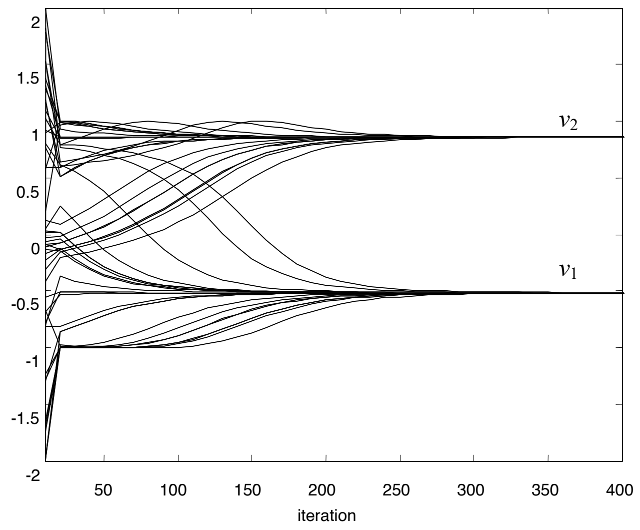


Fig. 8. Trajectories of the modified Hopfield network for 20 initial points in example 2.

It is important to observe that all trajectories starting from the inside or outside of the feasible region V converge to v^*. Thus, the proposed approach always converges to the optimal solution, independently whether the chosen initial point is located in the feasible region or not. Therefore, we can conclude that the modified Hopfield network is of high robustness.

A comparison using the SQP method and the modified Hopfield network was also made for this example. The SQP method has reached the exact solution for all simulations. Table 3 shows the number of iterations and convergence time used in each approach to reach the final solution for different initial values of the output vector v. From this table, although the method SQP obtains the final solution in less iteration, it is verified that convergence time of the modified Hopfield network is close to that required by the SQP method, where for $v^{(0)} = [-2\ \ 2]^T$ and $v^{(0)} = [1\ \ 0]^T$ the network converged more rapidly.

Initial Vector	Modified Hopfield Network		SQP Method	
	Iterations	Convergence time	Iterations	Convergence time
$v^{(0)} = [\ 2\ \ 2]^T$	278	3.86	34	3.72
$v^{(0)} = [-2\ \ 2]^T$	316	2.83	24	2.87
$v^{(0)} = [-2\ \ -2]^T$	297	3.19	24	2.81
$v^{(0)} = [\ 2\ \ -2]^T$	303	5.03	42	4.41
$v^{(0)} = [\ 1\ \ \ 0]^T$	359	2.94	25	3.16
$v^{(0)} = [\ 0\ \ -1]^T$	311	4.76	39	4.15

Table 3. Comparison between SQP method and modified Hopfield network in example 2.

Example 3. Consider the following constrained optimization problem proposed in Bazaraa & Shetty (1979) in page 418, which is composed by inequality and equality constraints:

$$\text{Min } f(v) = v_1^3 + 2v_2^2 \cdot v_3 + 2v_3$$

$$\text{subject to: } v_1^2 + v_2 + v_3^2 = 4$$

$$v_1^2 - v_2 + 2v_3 \le 2$$

$$v_1, v_2, v_3 \ge 0$$

The optimal solution for this problem is given by $v^* = [0.0\ \ 4.0\ \ 0.0]^T$, where the minimal value of $f(v^*)$ at this point is equal to zero. Figure 9 shows the trajectories of the network variables starting from the initial point $v^0 = [1.67\ \ 1.18\ \ 3.37]^T$. All simulation results obtained by the modified Hopfield network using $\Delta t = 0.001$ show that the proposed architecture is globally convergent to v^*.

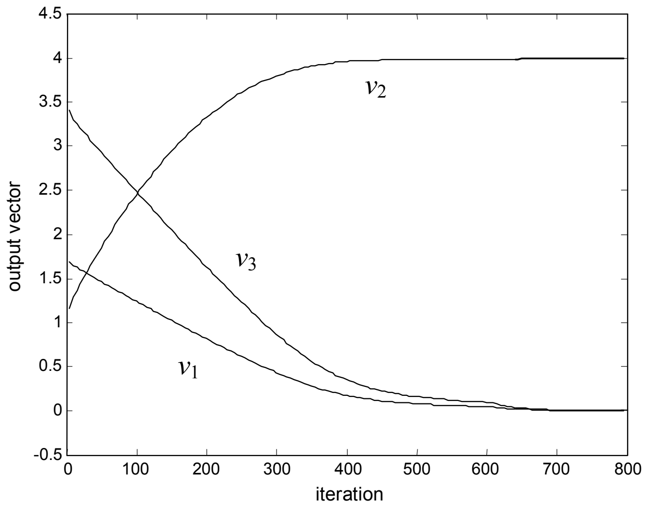


Fig. 9. Transient behavior of the modified Hopfield network in example 3.

The network has also been evaluated for different values of initial conditions. The trajectories of the objective function starting from several initial points are illustrated in Fig. 10. All trajectories lead toward the same equilibrium point. These results show the ability and efficiency of the modified Hopfield network for solving constrained nonlinear optimization when equality and inequality constraints are simultaneously included in the problem.

In comparison with results obtained by using the multilayer perceptron network proposed in Bazaraa & Shetty (1979), and starting from the same initial points, it was observed that the modified Hopfield Network not only converges more quickly, but also results in higher accuracy.

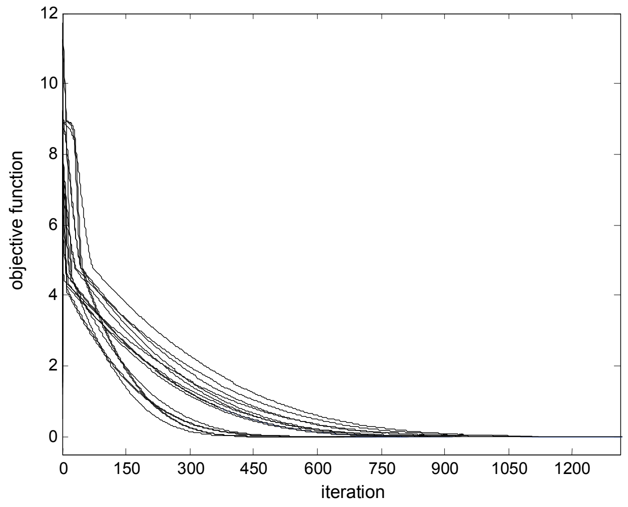


Fig. 10. Evolution of the objective function for 15 initial points in example 3.

In relation to the SQP method, the obtained solution was the same found by the modified Hopfield network. For this example, the SQP method reached the final solution using 30 iterations (3.66 seconds), whereas the modified Hopfield network needed 768 iterations (3.48 seconds). So, for this example, the modified Hopfield network has converged in less time than the SQP method.

5. Conclusions

This chapter presents an approach for solving optimization problems using artificial neural networks. More specifically, a modified Hopfield network is developed and its internal parameters are computed using the valid-subspace technique.

The developed approach allows to solve several classes of optimization problems through a unique neural network architecture. The optimization problems treated in this chapter are the combinatorial optimization problems, dynamic programming problems and nonlinear optimization problems. An energy function E^{op} was designed to conduct the network output

to the equilibrium points corresponding to a cost constraint. All structural constraints associated with the optimization problems can be grouped in E^{conf}.
The simulation results demonstrate that the network is an alternative method to specialist algorithms and has the advantage of being implementable in a neural network environment, which can be mapped in hardware for engineering applications. The internal parameters of the network were explicitly computed using the valid-subspace technique that guarantees the network convergence. All simulation results show that the proposed network is completely stable and convergent to the solutions of the optimization problems considered in this chapter. The network has also been evaluated for different values of initial conditions. All trajectories lead toward the same equilibrium point.

6. References

Aiyer, S. V. B.; Niranjan, M. & Fallside, F. (1990). A Theoretical investigation into the performance of the Hopfield network. *IEEE Transactions on Neural Networks*, Vol.1, 204-215.

Bazaraa, M. S. & Shetty, C. M. (1979). *Nonlinear Programming – Theory and Algorithms*, Wiley, New York.

Chiu, C.; Maa, C. Y. & Shanblatt, M. S. (1991). Energy function analysis of dynamic programming neural networks. *IEEE Transactions on Neural Networks*, Vol. 2, 418-426.

Dillon, J. D. & O'Malley, M. J. (2002). A Lagrangian augmented Hopfield network for mixed integer non-linear programming problems. *Neurocomputing*, Vol. 42, 323-330.

Fiacco, A. V. & McCormick, G. P. (1968). *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, Wiley, New York.

Graham, A. (1981). *Kronecher Products and Matrix Calculus*, Ellis Horwood Ltd., Chichester, UK.

Haykin, S. (1999). *Neural Networks – A Comprehensive Foundation*, Prentice-Hall Inc., Upper Saddle River, New Jersey.

Hillier, F. S. & Lieberman, G. J. (1980). *Introduction to Operations Research*, Holden Day, San Francisco, California.

Hopfield, J. J. & Tank, D. W. (1985). Neural computation of decisions in optimization problems. *Biological Cybernetics*, Vol. 52, 141-152.

Hopfield, J. J. (1984). Neurons with a graded response have collective computational properties like those of two-state neurons. *Proc. of the National Academy of Science*, Vol. 81, 3088-3092.

Kakeya, H. & Okabe, Y. (2000). Fast combinatorial optimization with parallel digital computers. *IEEE Transactions on Neural Networks*, Vol. 11, 1323-1331.

Kennedy, M. P. & Chua, L. O. (1988). Neural networks for nonlinear programming. *IEEE Transactions on Circuits and Systems*, Vol. 35, 554-562.

Luenberger, D. G. (1984). *Linear and Nonlinear Programming*, Addison-Wesley, Reading, MA.

Papadimitriou, C. H. & Steiglitz, K. (1982). *Combinatorial Optimization - Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ.

Silva, I. N.; Arruda, L. V. R. & Amaral, W. C. (1997). Robust estimation of parametric membership regions using artificial neural networks. *International Journal of Systems Science*, Vol. 28, 447-455.

Vidyasagar, M. (1993). *Nonlinear Systems Analysis*, Prentice-Hall, Englewood Cliffs, NJ.

Xia, Y.; Leung, H. & Wang, J. (2002). A projection neural network and its application to constrained optimization problems. *IEEE Trans. on Circuits and Systems*, Vol. 49, 447-458.

12

Applications of Recurrent Neural Networks to Optimization Problems

Alaeddin Malek
Department of Applied Mathematics,
Tarbiat Modares University, Tehran,
Iran

1. Introduction

The work in this chapter presents some applications of recurrent neural networks to general optimization problems. While particular problems presented in this research relates to linear, quadratic and nonlinear programming, monotone variational inequalities and complementarity problems, I fell that the methodology by which one solves these problems are quite general and warrants attention in and of themselves. Correspondingly, I hope that this material will be taken as both a response to a particular problem and a general method.
Constrained optimization problems are defined as the mathematical representation of real world problems concerned with the determination of a minimum or a maximum of a function of several variables, which are required to satisfy a number of constraints. Such function optimization are sought in diverse fields, including mechanical, electrical and industrial engineering, operational research, management sciences, computer sciences, system analysis, economics, medical sciences, manufacturing, social and public planning and image processing.
Although many classical optimization algorithms such as simplex, Karmarkar interior point, direct and indirect techniques are given to solve linear, quadratic and nonlinear optimization problems, in many applications, it is desire to have real-time on-line solutions of corresponding optimization problems. However, traditional optimization algorithms are not suitable for real-time on-line implementation on the computer. The dynamical system approach is one of the promising approaches that can handle these difficulties.
In the recent years many artificial neural networks models developed to solve optimization problems. Several basic and advance questions associated with these models have motivated the studies presented in this chapter.
The goal of this chapter is twofold. The theoretical areas of interest include fundamental methods, models and algorithms for solving general optimization problems using artificial recurrent neural networks. On the other hand, it will try to present and discuss the numerical analysis for the corresponding models, simulations and applications of recurrent neural networks that solve various practical optimization problems.
Recurrent dynamical neural network is an area of neural networks which is one of the fundamental topics of the subject, and combines many mathematical concepts like ordinary and partial differential equations, dynamical systems, unconstrained and constrained

optimization, local and global optima for a function of several variables, sigmoid functions, error estimation, integration and gradient descent methods. Students are often familiar with the local optima of a function with one variable before embarking on an undergraduate course, and in practical way will have integrals which they can not express in closed analytical form. Here we must compute the optimal solution for the constrained optimization problem with objective function of several variables that corresponds with the solution of a system of ordinary differential equations. From mathematical point of the view convergence of the solution and stability of the method has quiet importance, while as an engineer we might look for an algorithm that works for many different problems.

The troublesome problem of just what numerical optimization analysis is arises in recurrent dynamical neural network, as it does in other branches of the field. Should the optimization analysis part be the main aim, or is it the generation of an efficient, tested and validated program which is important? The answer is surely that both areas are important, but at the end of the day numerical analysis and mathematical techniques are some service industry and what the customers want is reliable codes to solve their problems. The theoretical analysis forms part of the reliability assessment, as it determines bounds on errors and levels of stability. These error bounds form the basis of a theoretical justification for the solution convergence of the corresponding numerical algorithm to the actual solution of the original neural network model.

The chapter covers a range of topics from early undergraduate work on constrained linear and quadratic programming through to recent research on nonlinear constrained optimization problems and recurrent neural networks. The source of the optimization work is the lecture notes for graduate students participated in my advance linear programming and optimization courses. The notes have grown in sixteen years of teaching the subject. The work on recurrent neural network models is based partly on my own research. It has taken annual updates as new models have proposed in some of the thesis of my postgraduate students during the last ten years. This research is enriched by the huge literature which has grown in the last two decades.

I am grateful to the applied mathematics department here at Tarbiat Modares University which has made available the technical equipment for the work. The novel models and numerical programs have been tested, compared and improved using the various computers which have been installed over the years.

In the next section we study solution methods for general optimization problems under the assumption that there exists an optimal solution.

2. Optimization problems

In this section, we shall first consider an important class of constrained linear programming problems and their general dual form. Second, we shall introduce primal and dual form of a constrained convex quadratic programming problem. Then we will consider the nonlinear convex programming problems. This, as we shall see, leads to discovering some primal-dual relationships that exists for corresponding class of constrained optimization problems. Among the class of constrained optimization problems, an important and richly studied subclass of problem is that of convex programs.

Definition 1. The problem of maximizing a concave function or minimizing a convex function over a convex set is known as convex programming.

2.1 Constrained linear optimization problems

A problem of the form

$$\text{(PLP)} \qquad \begin{aligned} &\mathit{Maximize} \quad z = c^T x \\ &\mathit{Subject\ to} \\ &\qquad Ax \geq b \\ &\qquad x \geq 0 \end{aligned} \tag{1}$$

is said to be a primal linear programming problem, where $x \in R^n, c \in R^n, A \in R^{m \times n}$, $b \in R^m$. Here $A = (a_{ij})$ is the coefficient matrix of the inequality constraints, $b = (b_1,...,b_m)^T$ is the vector of constants, the components of $c = (c_1,...,c_n)$ are called cost factors, $x = (x_1,...,x_n)^T$ is the vector of variables, called the decision variables. Associated with (PLP) is the linear programming problem (DLP), called the dual of (PLP):

$$\text{(DLP)} \qquad \begin{aligned} &\mathit{Minimize} \quad v = b^T y \\ &\mathit{Subject\ to} \\ &\qquad A^T y \leq C \\ &\qquad y \geq 0 \end{aligned} \tag{2}$$

In (DLP) formulation y is the vector of m dual variables. We can define the dual of any linear problem after writing it in the primal form (PLP), [1].

Remark 1. Primal and dual linear programs (PLP) and (DLP) are convex programs since the set of feasible solutions to a linear program is a convex set and a linear objective function is both convex and concave.

2.2 Constrained quadratic optimization problems

We consider a primal quadratic programming problem in

$$\mathit{Minimize} \qquad f(x) = \frac{1}{2} x^T A x + c^T x$$

$$\mathit{subject\ to} \qquad g(x) = Dx - b = 0, \quad x \geq 0, \tag{3}$$

Where A is a $m \times m$ symmetric positive semidefinite matrix, D is a $n \times m$ matrix and rank (D) = m, $b \in R^n$, $x, c \in R^m$. We define the dual problem (DQP) as follows:

$$\mathit{Minimize} \quad \tilde{f}(x) = -\frac{1}{2} x^T A x + b^T y$$

$$\mathit{Subject\ to} \quad \tilde{g}(x) = D^T y - \nabla f(x) \leq 0, \tag{4}$$

where $\nabla f(x) = Ax + c, \ y \in R^n$.

Lemma 1. The primal quadratic program (PLP) and its dual (DLP) are convex programs. This is because the quadratic forms $\frac{1}{2}x^T Ax + c^T x$ and $-\frac{1}{2}x^T Ax + b^T y$ are convex if and only if A is a positive semidefinite matrix (for example see [2]). Clearly the standard linear programming problem

$$\begin{aligned}&\mathit{Maximize} \quad z = c^T x\\&\mathit{Subject\ to}\\&\qquad\qquad Dx = b\\&\qquad\qquad x \geq 0\end{aligned} \tag{5}$$

and its dual

$$\begin{aligned}&\mathit{Minimize} \quad v = b^T y\\&\mathit{Subject\ to}\\&\qquad\qquad A^T y \geq c\\&\qquad\qquad y\ \mathit{is\ free\ in\ sign},\end{aligned} \tag{6}$$

are special cases of the (3) and (4) respectively, for which $A = 0_{m\times m}$.

2.3 Constrained nonlinear optimization problems

Consider the following nonlinear convex programming problem (NP) with nonlinear constraints:

$$\text{(NP)}\ \mathit{Minimize} \quad f(x)$$

$$\mathit{Subject\ to}\ g(x) \leq 0,\ x \in \Omega \tag{7}$$

where $x = (x_1, \ldots, x_n)^T \in R^n$, $f : R \to R^n$. $g(x) = (g_1(x), \ldots, g_m(x))$ is m-dimensional vector-valued continuous function of *n* variables. The functions f and $g_1, \ldots, g_m$ assumed to be convex and twice differentiable for $\Omega \subseteq R^n$.

Definition 2. A vector x is called a feasible solution to (NP) if and only if *x* satisfies $m+n$ constraints of the (NP).

Definition 3. Any feasible solution x is said to be a regular point if the gradients of $g_i(x)$, $\nabla g_i(x)$ for $(i \in I = \{j \mid g_j(x) = 0\})$, are linearly independent.

Definition 4. The (NP) has at least one optimal solution [3] when

i. the set of all feasible solutions is nonempty and bounded,

ii. the feasible set is unbounded but $f(x)$ has a bound level set.

2.4 Monotone variational inequalities and complementarity problems

The problem of finding a vector point $x^* \in R^n$ such that

$$x^* \in S, \quad \langle F(x^*),\ x - x^* \rangle \geq 0 \quad \text{for all } x \in S \tag{8}$$

where $x = (x_1, \ldots, x_n)^T \in R^n$, is called the monotone variational inequality problem [4]. F is a continuous mapping from R^n into itself, and $S = \{ x \in R^n \mid Ax - b \geq 0,\ Bx = c,\ x \geq 0 \}$ where $A \in R^{m \times n}$, rank $(A) = m$, $B \in R^{r \times n}$, rank$(B) = r$, $0 \leq m$, $r \leq n$, $b \in R^m$, $c \in R^r$, and S is a nonempty closed convex subset of R^n and $\langle\, .\, ,\, .\, \rangle$ denotes the inner product in R^n. In the special case where $S = R^n_+$, problem (8) can be rewritten as the following nonlinear complementarity problem

$$x^* \geq 0, \quad F(x^*) \geq 0, \quad \langle x^*,\ F(x^*) \rangle = 0\,. \tag{9}$$

For $S = R^n$, problem (8) reduces to solving the system of nonlinear equation $F(x) = 0$, [5].

Remark 2. For a continuously differentiable function f, if x^* is a solution of the problem Minimize$\{ f(x) \mid x \in S \}$; $S = \{ x \in R^n \mid Ax - b \geq 0,\ Bx = c,\ x \geq 0 \}$ then x^* is also a solution of (8) with $F(x) = \nabla f(x)$, and $\nabla f(x) = \left(\partial f / \partial x_1, \ldots, \partial f / \partial x_n \right)^T \in R^n$ is the gradient vector of $f(x)$ at point x.

Definition 5. [6] A mapping $F : R^n \to R^n$ is said to be monotone on S if

$$\langle F(x) - F(x'),\, x - x' \rangle \geq 0 \quad \text{for all } x, x' \in S. \tag{10}$$

F is strictly monotone on S, if strict inequality holds in (10) whenever $x \neq x'$.

Lemma 2. If F is continuously differentiable and the Jacobean matrix ∇F is positive definite for all $x \in S$, i.e.

$$\langle d, \nabla F d \rangle > 0 \quad \text{for all } \ x \in S,\ \ d \in R^n \ \ (d \neq 0)\,.$$

then F is strictly monotone on S.

Proof. For example see [7].

The variational inequalities problems have wide variety of scientific and engineering applications (for example see [2], [6], [8] to [11]). In many applications, real-time on-line solutions of (8) and (9) are desired. However, traditional algorithms (see [2], [6], [8], [12] and [13]) are not suitable for real-time on-line implementation on the computer. One promising approach to handle these problems is to employ an artificial neural network based on circuit implementation. Many continuous-time neural networks for constrained optimization problems, have been developed ([14] to [18]) using network parameters. To avoid using penalty parameters, some significant works have been done in recent years. A few primal and dual neural networks with two-layer and one-layer structure were developed in [14], [17] and [18]. These neural networks were proved to be globally convergent to an exact solution.

In the next section, we discuss some general ideas about artificial neural networks.

3. Artificial neural networks

Artificial neural networks consist of a calculation unit called neuron. Every neuron has some real valued inputs. Inside every neuron, each input is multiplied with corresponding neural coefficient defining its value. The sum of all these products adds to a value called bias. Finally, activation function affects this sum and determines the real valued output of the neuron feed forwardly [19] or by some feed back [20].

3.1 Feed forward back propagation neural networks

Primary discussions regarding artificial neural networks introduced in the 40's with presentation of the feed forward neural networks. Artificial neural networks in some extents are modeled from the brain and neural system of the human, which are able to give acceptable solutions based on correct information records from the problem.
The basic structure for the feed forward back propagating neural network (nets without feed back) consists of some number of nodes in the input layer, the hidden layer, and the output layer that has one node. The sigmoid functions approximate linear functions, yet allow the update scheme to propagate backwards through differentiable functions. The manner in which input data generates output data for a given neural network depends on the interconnection weights. These weights are adjusted to reduce the error between the neural network outputs and the actual output values. i.e.

$$E = \frac{1}{2}\sum_{i=1}^{n}(O_i^{actual} - O_i^{net}) \tag{11}$$

where O_i^{actual} is the actual output for the i^{th} training point. O_i^{net} is the estimated value from the neural network for the i^{th} training point from the neural network. n is the total number of training points obtained by taking known data points for a given task. Here the objective is to train the network so that the output from the network minimizes equation (11).

3.2 Recurrent dynamical artificial neural network

Khanna in year 1990 [21], describes associative memory as "the ability to get from one internal representation to another or to infer a complex representation from a portion of it". Effectively our goal in applying neural networks is to create a functional mapping from steady optimization space to either dynamical time dependent space or some parameter space. Two approaches to achieving this mapping have been extensively studied by Xia [14], [15] and [22] to [25], Malek [4], [16], [26] and [27] and their coauthors.
The first approach relies on a structure with adjustable parameters. On the basis of known input/output pairs, these parameters are selected or changed. If this approach is successful, the appropriate selection of these parameters will yield a mapping device which will always provide the associated output values for a given input.
The second approach uses information from the primal and dual optimization problem and applied primarily by Malek in year 2005 [16]. The basis for such systems is a precisely defined set of ordinary differential equations that automatically satisfy the related primal and dual optimization problems simultaneously. These information are defined by the cumulative designing the system and are laid out in a hierarchical fashion. The system then

performs a sequential set of values, using the output from the previous as the input to the next. If successful, a system can be created which will associate input with its correlated output. The challenge is to make the system complete enough (consistent, convergent and stable) to always associate the correct output with a given input.

The primary difference in these two approaches is that adjustable parameters in the first are a prior, i.e., the parameters are settled upon and maintained before data is introduced into the system. The second approach has no adjustable parameters thus its model is simple to use. The advantage of this approach is that in this way, we can obtain a solution for the given real life problem, however we wish to assume a prior knowledge of relationships between constrained optimization problem and dynamical system. Moreover the solution for optimization problem consists of a solution for real life problem, since optimization problem is simulated from the corresponding real life problem.

The work presented in this section applies recurrent dynamical artificial neural network. We shall emphasize on networks that do not use network parameters or penalty parameters in advance. This approach is a metric driven method. i.e., we establish distance between the input and the neural network output. For a given input, the neural network outputs the value whose distance from the given input is smallest using linear constraint least square technique or any other related method. One manner of doing this mapping is to associate the equilibrium points of a dynamical system with the optimal points of constraint optimization problem. When the input is the initial condition of the dynamical system, the system will converge to an equilibrium point. Thus this optimal solution contains a solution that minimizes equation (11), where we use the feed back process to produce corresponding optimal weights. This means that the artificial neural network structure is recurrent.

The structure of the recurrent dynamical artificial neural network is different from the feed forward artificial neural network. However it is possible to make some corresponding relations between these two neural networks (see Rumelhart 1986, [28]). i.e., there is a sense in which the error back propagation scheme may be applied to networks that contain feed back, (see Fig. 3.1). The feed forward network in Fig. 3.1 may be represented to simulate a feed back network with a given set of weight and bias parameters.

Having developed the equivalent structure as shown in Fig. 3.2, it becomes proper to say "the goal for recurrent dynamical artificial neural network, as with the back propagation artificial neural network, is to minimize the error function given by equation (11).

Training of dynamical neural networks has received considerable attention in the last 30 years [20], [29] and [30]. The equations governing the behavior of the simplest supervised recurrent dynamical neural network are

$$\frac{\partial u}{\partial t} = -u + AS(u) + Bx, \qquad u_{initial} = 0 \tag{12}$$

$$u^* = AS(u^*) + Bx \tag{13}$$

$$y = C^T u^* \tag{14}$$

where $$S(u) = \frac{1}{1+e^{-u}}$$

is the sigmoid function. The adjustable parameters in this supervised recurrent dynamical neural network are found in the *A*, *B* matrices and vector *C*. The input x of the neural network corresponds to the input data associated with a training point. This input is then applied to the system governed by equation (12). When equation (13) reaches an equilibrium value *u** for this input, we obtain the output of the neural network by taking the dot product of *C* and *u** by equation (14). This neural network output will then compare with the actual output. To update the elements of *A*, *B*, and Cone may use gradient descent method using $\frac{\partial u}{\partial A_{ij}}, \frac{\partial u}{\partial B_{ij}}$, and $\frac{\partial u}{\partial C_i}$.

This minimization task requires that the neural network possess enough parameter freedom to enable each input set to generate an output close to the actual value. This is not a case in many problems. Thus in the next section we emphasize on the unsupervised recurrent dynamical artificial neural networks.

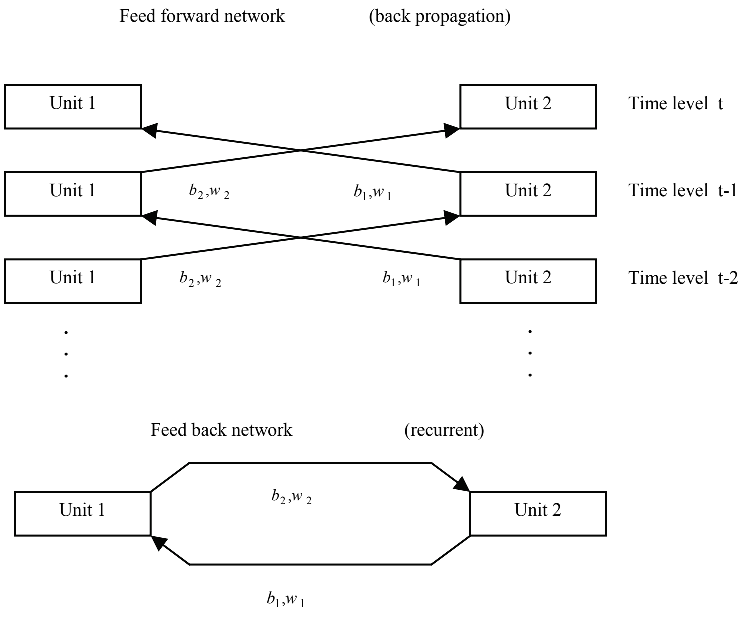


Fig. 3.1 Equivalent structures of a two unit network; Feed forward network, and feed back network for a given biases b_1 and b_2 and weights w_1 and w_2.

4. Networks dynamic analysis

For many times dependent cost functions an online optimizer on the basis of an analog circuit [31], [32] and [33]) is desirable. Dynamic solvers or analog computer, was first proposed by Dennis [34], Rybashov [35] and [36], Karpinskaya [37], and later studied by Kenedy and Chua [38], Rodriguez-Vazquez et al. [39], Tank and Hopfield [31]. These dynamic solvers usually employ neural networks since they have many advantages over the traditional algorithms. Massively parallel processing and fast convergence are two of the most important advantages of the neural networks.

4.1 Models for linear programming

Use of neural network for the solution of linear programming problems goes back to 1985, when Hopfield and Tank [31] provide fast algorithm based on analog electrical components. Chen and Fang [40] in 1998 examined the theoretical properties of a method proposed by Kennedy and Chua in 1987, [38]. Malek and Yari in year 2005 proposed a fully stable artificial recurrent neural network model for the solution of primal linear programming problems of the type (1):

$$\begin{cases} \dfrac{dX(t+\eta)}{dt} = \eta_1[C - A^T(Y + \eta\dfrac{dY}{dt})] \\ \dfrac{dY(t+\eta)}{dt} = \eta_2[A(X + \eta\dfrac{dX}{dt}) - b] \end{cases} \tag{15}$$

where η, η_1 and η_2 are rate of learning (in the neural network dynamic). They are step sizes in the process of optimization computation. η, η_1 and η_2 can stay constant or vary in each iteration.

Model (15) transfers the linear programming problem into a dynamical system of equations and gives approximation solution to the exact solution only for primal variables. This means that by the recurrent neural network model (15) dual optimum value for objective function does not coincide exactly with the optimum value obtained from primal problem.

The second model proposed by Malek in the same article is in the following form [16]:

$$\begin{cases} \dfrac{dU(t+\eta)}{dt} = \eta_1[\overline{C} - \overline{A}^T(V + \eta\dfrac{dV}{dt})] \\ \dfrac{dV(t+\eta)}{dt} = \eta_2[\overline{A}^T(U + \eta\dfrac{dU}{dt}) - \overline{b}] \end{cases} \tag{16}$$

where $U = (X, Y)$ and V is the corresponding dual variable to the dual form of problem

$$\begin{aligned} &Maximize\ \ Z = \left[C^T(X + \eta\frac{dX}{dt}) - b^T(Y + \eta\frac{dY}{dt}) \right] \\ &Subject\ to\ \ A(X + \eta\frac{dX}{dt}) \le b \\ &\qquad -C^T\frac{dX}{dt} \le 0 \\ &\qquad -A^T(Y + \eta\frac{dY}{dt}) \le -C \\ &\qquad b^T\frac{dY}{dt} \le 0 \\ &\qquad X + \frac{dX}{dt} \ge 0,\ Y + \eta\frac{dY}{dt} \ge 0 \end{aligned} \tag{17}$$

$\overline{A}$ is a block matrix of the form

$$\overline{A}=\begin{pmatrix} A & \eta A & 0 & 0 \\ 0 & -C^T & 0 & 0 \\ 0 & 0 & -A^T & -\eta A^T \\ 0 & 0 & 0 & b^T \end{pmatrix}$$

for $A_{m\times n}$, $\overline{C}=(C^T,\ \eta c^T,\ -b^T,\ -\eta b^T)$, $\overline{b}=(b,0,-c,0)$. We shall see that, $\overline{A}$ is a *(m+n+2)* × $(2m+2n)$ matrix and $\overline{C}$ is a vector with $2m+2n$ components and $\overline{b}$ is($m+n+2$)×1 vector.

The following lemma shows that this model solves both primal and dual problems of the type (1) and (2) simultaneously.

Lemma 3. For $X^*=(x_1^*,x_2^*,...,x_n^*)$ the optimum solution $U^*=(X^*,Y^*)$ of problems in the forms (PLP) and (DLP), is the optimum solution for (P-D) *iff* Z^* the maximum value for Z vanishes where $\frac{dX}{dt}\to 0$ and $\frac{dY}{dt}\to 0$.

Proof: See [16].

These models need some network parameters η,η_1 and η_2 that must be fixed in the starting time.

4.2 Models for quadratic programming

Xin-Yu Wu et al. [22] in year 1996 proposed the following neural network model to solve problems (3) and (4)

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix}=-\begin{Bmatrix} \beta(-D^Ty+Ax+c)+\beta A[x-(x+D^Ty-Ax-c)^+]+D^T(Dx-b) \\ \beta\{Dx-b+D[(x+D^Ty-Ax-c)^+-x]\} \end{Bmatrix} \tag{18}$$

where $\beta=\left\|x-(x+D^Ty-Ax-c)^+\right\|_2^2$.

Youshen Xia [14] considered the adjusted form of model (1) as follows

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix}=-\begin{Bmatrix} (I+A)[x-(x+D^Ty-Ax-c)^+]+D^T(Dx-b) \\ -D[x-(x+D^Ty-Ax-c)^+]+Dx-b \end{Bmatrix} \tag{19}$$

where I is the identity matrix.

Malek and Oskoei [26] proposed three novel models based on model (1) in the following forms:

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix}=\begin{Bmatrix} D^Ty-Ax-c-A[x-(x+D^Ty-Ax-c)^+]-D^T(Dx-b) \\ -D(x+D^Ty-Ax-c)^++b \end{Bmatrix} \tag{20}$$

Model (20) is a simplified model (18) of Xin-Yu Wu et al. Here one may concerne of obtaining better accuracy for the final solutions, while we do not use expensive analog multipliers of Xin-Yu Wu et al. Therefore the relative question might be: is there a simpler neural network models in the manipulation of hardware tools. Malek & Oskoei [26] show that for some examples model (20) converges to the exact solution with 13 exact decimal points. While in the same conditions the solutions for neural network proposed by Xin-Yu Wu agrees with the corresponding exact solution only up to 3 decimal points.
It is still possible to simplify model (20). Model (21) has the advantage of serious simplification and good accuracy in the same time. It is in the form [26]:

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = -\begin{Bmatrix} (I+A)[x-(x+D^T y-Ax-c)^+] \\ D(x+D^T y-Ax-c)^+ - b \end{Bmatrix} \tag{21}$$

Let us assume that

$$(x,y)\in\Psi,\ \Psi=\left\{(x,\ y)\,\middle|\, y\in R^n, x\in R^m, x\geq 0\right\},$$

$(x)^+ = \left[(x_1)^+,\ldots,(x_m)^+\right]^T$ and $(x_i)^+ = \max\{0, x_i\}$, for $i=1,\ldots,m.$ We proposed following model:

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = -\begin{Bmatrix} x-(x+D^T y-Ax-c)^+ \\ D[(x+D^T y-Ax-c)^+]-b \end{Bmatrix} \tag{22}$$

in [26] which appears to be more efficient than the models (20) and (21) when we investigate the complexity, complexity of individual neurons, stability, and accuracy of the solutions, (see Tables 1 and 2 in section 5).
Model (22) does not use any projection operator in practice thus it is different and simpler from the model proposed by Qing Tao et al. Here in model (9), unlike the Qing Tao's model we do not use any extension of Newton's optimal descent flow equation to solve the problem.
If we assume that $\alpha=(x+D^T y-Ax-c)^+$ and $\beta=D^T(Dx-b)$, then models (22) and (19) are in the following forms respectively [41]:

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{cases} \alpha - x, \\ \\ -D\alpha + b. \end{cases} \tag{23}$$

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{cases} (I+A)\ (\alpha - x) - \beta, \\ \\ -D\alpha + b. \end{cases} \tag{24}$$

The network circuit implementation for solving problems (3) and (4) whose dynamics are governed by (23) are given in the Fig. 3.2. The circuit consists of adders (summing amplifiers) and integrators. In the Fig. 3.2, vectors c and b are external input vectors, while

x and y are the network outputs. In this diagram dynamical process of vector α is the same as what is given in [14]. A simplified block diagram of α is illustrated in Fig. 3.3 to show how expensive it is using vector α in the arbitrary model.
Malek & Alipour, Applied Mathematics and Computation 192 (2007) 27-39 We now compare the network (24) with our proposed network in (23) for solving problems (2) and (3). The network (24) is stable to exact solution and there are no parameters to set, but the main disadvantage of it is that too many expensive analog multipliers (α, β) are required for large scale quadratic programming problems, thus the set of hardware implementation is expensive and therefore greatly affect the accuracy of solutions. Neural network model (23) does not need to use β and therefore in practice needs relatively less computational efforts. Moreover, this model is globally convergence to the corresponding exact solution independent of where and how to choose the starting input initial values. Model (23) not only has the same global convergence property as the model (24), but also has some more advantages, plus simplicity. Network (23) is better than network (24) in the sense of complexity, i.e. usage analog multipliers and hardware implementations.

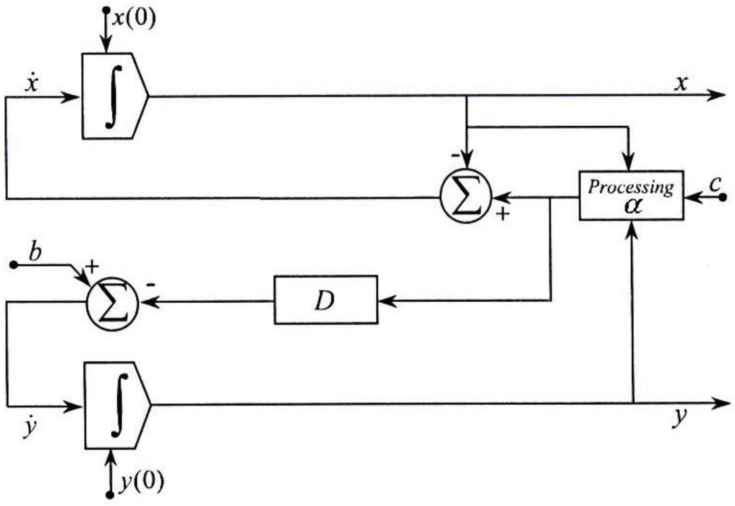


Fig. 3.2. A simplified neural network diagram for model (23): Malek & Alipour, Applied Mathematics and Computation 192 (2007) 27-39

Remark 3. Model in (23) may be used for solving general standard linear programming problems by setting $A = 0_{m\times m}$.
Simulation and numerical results are discussed in the next section.
Theorem 1. The recurrent dynamic artificial neural network (23) is globally convergent to the solution set of the primal and dual quadratic programming problems (3) and (4).
Proof. Let in the proposed model of Qing Tao et al. [17], general projection operator to be the identity operator. Then the proof is similar to Qing Tao's proof. (see [26] and also see Theorem 4)
In the reminder of this subsection we will try to clarify the ideas in Theorem 1 from theoretical point of view (see [41]).

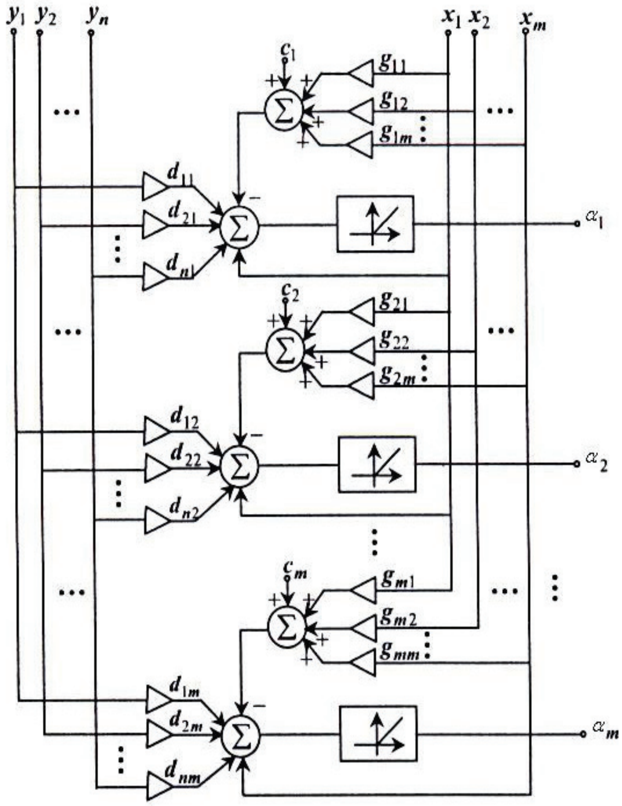


Fig. 3.3. A simplified block diagram for α , where $A = (a_{ij})$ and $D = (d_{ij})$:

In this section, we shall study the dynamics of network (23).We define a specific Liapunov function and get the global convergence of network (23).We first discuss some prerequisites.

Definition 6. A continuous-time neural network is said to be globally convergent if for any given initial point, the trajectory of the dynamic system converges to an equilibrium point.

Lemma 4. Let Ψ be a closed convex set of R^m .Then

$$\left[v - p_\Psi(v)\right]^T \left[p_\Psi(v) - x\right] \geq 0, v \in R^m, x \in \Psi$$

and $\left\|p_\Psi(v) - p_\Psi(u)\right\| \leq \|v - u\|, v, u \in R^m$

where $\|.\|$ denote l_2 norm and the projection operator $p_\Psi(u)$ is defined by

$$p_\Psi(u) = \arg\min_{v \in \Psi} \|u - v\|.$$ Proof. See [42].

Remark 4. Since $R_+^m = \{x \mid x \geq 0\}$ is a closed convex and by the property of a projection on a

$$\left[v - (v)^+\right]^T \left[(v)^+ - \tilde{x}\right] \geq 0, \tilde{x} \in R_+^m, v \in R^m.$$

Theorem 2. x^*, y^* are solutions of problems (3) and (4), respectively, if and only if (x^*, y^*) satisfies

$$\begin{cases} (x^* + D^T y^* - Ax^* - c)^+ = x^*, \\ D(x^* + D^T y - Ax^* - c)^+ = b. \end{cases} \tag{25}$$

Proof . By Karush-Kuhn-Tucker theorem for convex programming problem [43] we have x^*, y^* are solutions of problems (3) and (4), respectively, if and only if (x^*, y^*) satisfies

$$\begin{cases} Dx^* = b, x^* \geq 0, \\ x^{*T}(D^T y^* - Ax^* - c) = 0, \\ D^T y^* - Ax^* - c \leq 0. \end{cases} \tag{26}$$

Clearly, that (26) is equivalent to (25).
We will now prove a theorem that is a base for proving the global convergence of model (23).

Theorem 3. Let $F_1(x,y) = \frac{1}{2}(x - x^*)^T A(x - x^*) + \frac{1}{2}\|x - x^*\|^2$ and

$F_2(x,y) = \frac{1}{2}\|y - y^*\|^2$ and $F(x,y) = F_1(x,y) + F_2(x,y)$. Then

$$\frac{d}{dt}F(x,y) \leq -(x - x^*)^T A(x - x^*) - \left\|x - (x + D^T y - Ax - c)^+\right\|^2.$$

Proof .

$$\frac{d}{dt}F_1(x,y) = \left(A(x - x^*)\right)^T \frac{dx}{dt} + \left(x - x^*\right)^T \frac{dx}{dt}$$

$$= \left(A(x - x^*)\right)^T \left((x + D^T y - Ax - c)^+ - x\right) + \left(x - x^*\right)^T \left((x + D^T y - Ax - c)^+ - x\right)$$

Note that

$$\left(A(x - x^*)\right)^T \left((x + D^T y - Ax - c)^+ - x\right)$$

$$= \left(A(x - x^*)\right)^T \left((x + D^T y - Ax - c)^+ - x^* + x^* - x\right)$$

$$= \left(A(x - x^*)\right)^T \left(x^* - x\right) + \left((x + D^T y - Ax - c)^+ - x^*\right)^T \left(Ax + c\right)$$
$$- \left((x + D^T y - Ax - c)^+ - x^*\right)^T \left(Ax^* + c\right)$$

On the other hand,

$$
\begin{aligned}
&\left((x+D^T y-Ax-c)^+-x^*\right)^T (Ax+c)\\
&=\left((x+D^T y-Ax-c)^+-x^*\right)^T \left(Ax+c-D^T y-x+(x+D^T y-Ax-c)^+\right)\\
&+\left((x+D^T y-Ax-c)^+-x^*\right)^T \left(x-(x+D^T y-Ax-c)^+\right)\\
&+\left((x+D^T y-Ax-c)^+-x^*\right)^T D^T y\\
&=\left((x+D^T y-Ax-c)^+-x^*\right)^T \left(Ax+c-D^T y-x+(x+D^T y-Ax-c)^+\right)\\
&+\left((x+D^T y-Ax-c)^+-x\right)^T \left(x-(x+D^T y-Ax-c)^+\right)\\
&+\left(x-x^*\right)^T \left(x-(x+D^T y-Ax-c)^+\right)\\
&+\left((x+D^T y-Ax-c)^+-x^*\right)^T D^T y
\end{aligned}
$$

and

$$\left((x+D^T y-Ax-c)^T-x^*\right)^T (Ax^*+c)$$

$$=\left((x+D^T y-Ax-c)^T-x^*\right)^T \left(Ax^*+c-D^T y^*\right)+\left((x+D^T y-Ax-c)^T-x^*\right)^T D^T y^*$$

So

$$
\begin{aligned}
&\frac{d}{dt}F_1\left(x,y\right)=\left(A(x-x^*\right)^T\left(x-x^*\right)\\
&+\left((x+D^T y-Ax-c)^+-x^*\right)^T \left(Ax+c-D^T y-x+(x+D^T y-Ax-c)^+\right)\\
&-\left\|x-(x+D^T y-Ax-c)^+\right\|^2-\left((x+D^T y-Ax-c)^+-x^*\right)^T \left(Ax^*+c-D^T y^*\right)\\
&+\left((x+D^T y-Ax-c)^+-x^*\right)^T \left(D^T y-D^T y^*\right).
\end{aligned}
$$

Thus by (22) we have

$$
\begin{aligned}
&\left((x+D^T y-Ax-c)^+-x^*\right)^T \left(D^T y^*-Ax^*-c\right)\\
&=\left((x+D^T y-Ax-c)^+\right)^T \left(D^T y^*-Ax^*-c\right)-x^{*T}\left(D^T y^*-Ax^*-c\right)\\
&=\left((x+D^T y-Ax-c)^+\right)^T \left(D^T y^*-Ax^*-c\right)\le 0
\end{aligned}
$$

Using lemma 4 we have

$$
\begin{aligned}
\frac{d}{dt}F_1\left(x,y\right)\le&-\left(x-x^*\right)^T A\left(x-x^*\right)-\left\|x-(x+D^T y-Ax-c)^+\right\|^2\\
&+\left((x+D^T y-Ax-c)^+-x^*\right)^T\left(D^T y-D^T y^*\right).
\end{aligned}
$$

Since

$$\frac{d}{dt}F_2(x,y)=\left(y-y^*\right)^T\frac{dy}{dt}$$

$$=\left(y-y^*\right)^T\left(-D(x+D^Ty-Ax-c)^++Dx^*\right)$$

then

$$\frac{d}{dt}F(x,y)\le-\left(x-x^*\right)^T A\left(x-x^*\right)-\left\|x-\left(x+D^Ty-Ax-c\right)^+\right\|^2.$$

The proof is complete.

Theorem 4. Network (4) is globally convergent to the solutions set of problems (3) and (4).

Proof . Using lemma 4, the right hand side of (23) is a Lipschitz mapping. From the existence theory of ordinary differential equations [44], we can assume that for any $(x_0,y_0)\in R^m\times R^n$ there exists a unique solution $\left(x(t),y(t)\right)$ of (4) and its maximal existence interval $\left[0,\lambda(x_0,y_0)\right)$.

Let x^*,y^* be solutions of problems (3) and (4) respectively. Let

$$V=\left\{(x,y)\in R^m\times R^n\left|F(x,y)\le\frac{1}{2}(x_0-x^*)^TA(x_0-x^*)+\frac{1}{2}\left\|x-x^*\right\|^2+\frac{1}{2}\left\|y-y^*\right\|^2\right.\right\}$$

Using theorem 3, $F(x,y)$ is a Liapunov function of system (23) on *V*. Since $(x-x^*)^TA(x-x^*)\ge0$ we have

$$F(x,y)\ge\frac{1}{2}\left\|x-x^*\right\|^2+\frac{1}{2}\left\|y-y^*\right\|^2.$$

This proves that *V* is bounded. By the extension theory of ordinary differential equations [], $\lambda(x_0,y_0)=+\infty$ .Using the LaSalle invariant principle [45], there exists a constant k, such that $\left(x(t),y(t)\right)\to M\cap F^{-1}(k),t\to+\infty$, where *M* is the maximal invariant set in

$$\Omega=\left\{(x,y)\left|\frac{d}{dt}F(x,y)=0,(x,y)\in\bar{V}\right.\right\}.$$

Now we will prove that every point in set *M* is a solution of problems (3) and (4). $\forall(x_1,y_1)\in N$, let $\left(x_1(t),y_1(t)\right)$ be a solution of equation (23) with initial point (x_1,y_1), its maximal existence interval is $\left[0,\lambda(x_1,y_1)\right)$. By the invariant of *M* and bounded ness of *V*, we have $\lambda(x_1,y_1)=+\infty,\ x_1(t)=x_1$. If (x_1,y_1) is not a solution of problems (3) and (4), using theorem 2 and 3 $D\left(x_1+D^Ty_1-Ax_1-c\right)^+\ne b$. From (23)

We have $\left\|y_1(t)\right\|\to\infty$ as $t\to\infty$. It is contradictory to the bound ness of *V*. Thus (x_1,y_1) is a solution of problems (3) and (4). Since (x_1,y_1) is arbitrary the proof is completed.

4.3 Models for nonlinear programming

Malek and Yashtini proposed the following recurrent dynamical artificial neural network [46]

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} P_\Omega[x-\nabla f(x)-\nabla g(x)y]-x \\ [y+g(x)]^+ - y \end{pmatrix}, \tag{27}$$

for the solution of nonlinear programming problem:

$$\begin{aligned} &Minimize \quad f(x) \\ &Subject\ \ to \quad Ax \le b \\ &\qquad\qquad x \in \Omega \end{aligned} \tag{28}$$

where $A \in R^{m\times n}$, $b \in R^m$.

4.4 Models for variational inequalities

The systems governing the behavior of the recurrent dynamical artificial network corresponding to the variational inequalities problem (8) are [4]

$$\frac{du}{dt} = \frac{d}{dt}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} (x-F(x)+A^T y + B^T z)^+ - x \\ (y-Ax+b)^+ - y \\ -Bx+c \end{pmatrix} \tag{29}$$

$$\begin{cases} x^* = (x^* - F(x^*) + A^T y^* + B^T z^*)^+ \\ y^* = (y^* - Ax^* + b)^+ \\ Bx^* = c \end{cases} \tag{30}$$

where $(x_i)^+ = \max\{0, x_i\}$ for all $i = 1,\ldots,n$ and $(y_j)^+ = \max\{0, y_j\}$ for all $j = 1,\ldots,m$, and x^* is the solution of monotone variational inequalities problem (…).

Now, let $x(.)$, $y(.)$ and $z(.)$ be some dependent variables to time *t*. We initiate $u_{initial} = 0$ to the system governed by (29), when system (30) reaches an equilibrium value u^* for this input, we obtain the output of the neural network. The goal for the continuous time based dynamical system described by two systems (29) and (30), is to minimize the error function given by equation (11).

Yashtini and Malek [4] proved that the recurrent neural network based on the systems (29) and (30) are stable in the sense of Lyapunov and globally convergent to an optimal solution.

5. Work examples

For the following three models proposed by Xia, Malek and their coauthors solve quadratic programming problem in Example 1.

(Model 1): $$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = -\begin{Bmatrix} (I+A)(x-(x+D^T y - Ax - c)^+) + D^T(Dx-b) \\ D(x+D^T y - Ax - c)^+ - b \end{Bmatrix}$$

(Model 2): $$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = -\begin{Bmatrix} (I+A)(x-(x+D^T y - Ax - c)^+) \\ D(x+D^T y - Ax - c)^+ - b \end{Bmatrix}$$

(Model 3): $$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = -\begin{Bmatrix} x-(x+D^T y - Ax - c)^+ \\ D(x+D^T y - Ax - c)^+ - b \end{Bmatrix}$$

Example 1. Consider the following (QP) problem:

$$\begin{aligned} \textit{Minimize} \quad & x_1^2 + x_2^2 + x_1 x_2 - 30x_1 - 30x_2 \\ \textit{Subject to} \quad & -x_1 + x_2 + x_3 = 2 \\ & x_1 + x_2 + x_4 = 2 \\ & x_1 - x_2 + x_5 = 8 \\ & x_1 + x_2 + x_6 = 12 \\ & x_1,\ x_2 \geq 0 \end{aligned}$$

x*=(1,1,2,0,8,10) z*=-57

Figs. 5.3 to 5.8 displays the transient behavior of $x(t)$ with five feasible initial points $A = (5,\ 0, 7, \text{-}3, 3, 7)$,
$B = (9,\ 1, 10, \text{-}12, 0, 2)$, $C = (8,\ 4, 6, \text{-}10, 4, 0)$, $D = (4,\ 6, 0, \text{-}8, 10, 2)$ and
$E = (5,\ 4, 3, \text{-}7, 7, 3)$ where y=(0,-1,0-2).

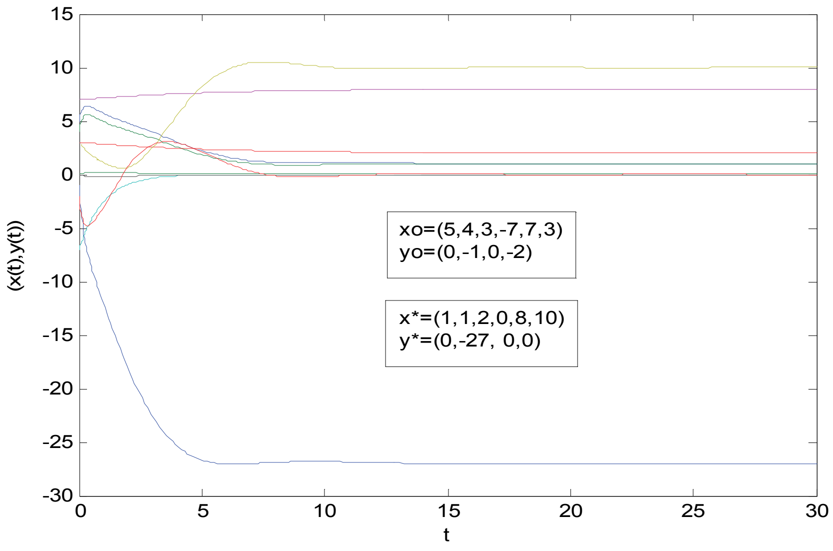


Fig. 5.1. Trajectories of example 1. for the given x and y initial vectors (feasible) using (Model 3).

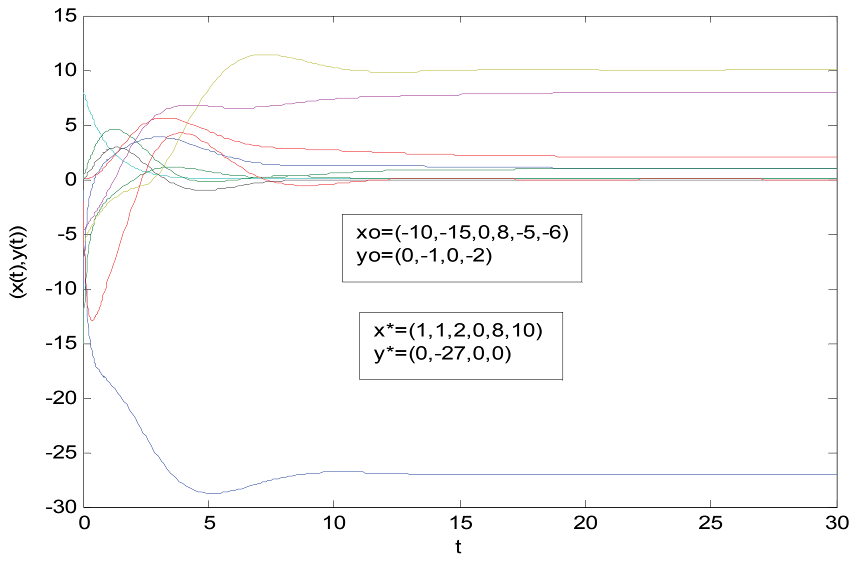


Fig. 5.2. Trajectories of example 1. for the given x and y initial vectors (infeasible) using (Model 3).

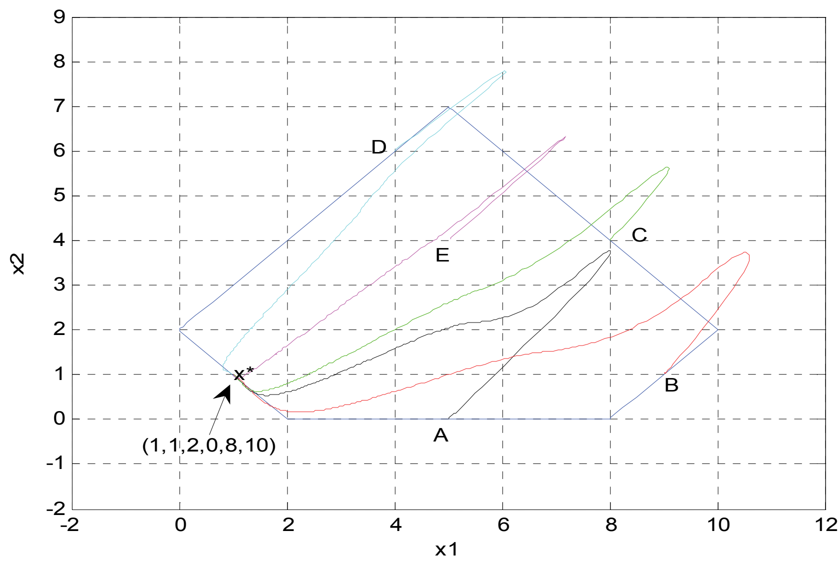


Fig. 5.3. Example 1: trajectories with initial points inside the feasible region using (Model 1).

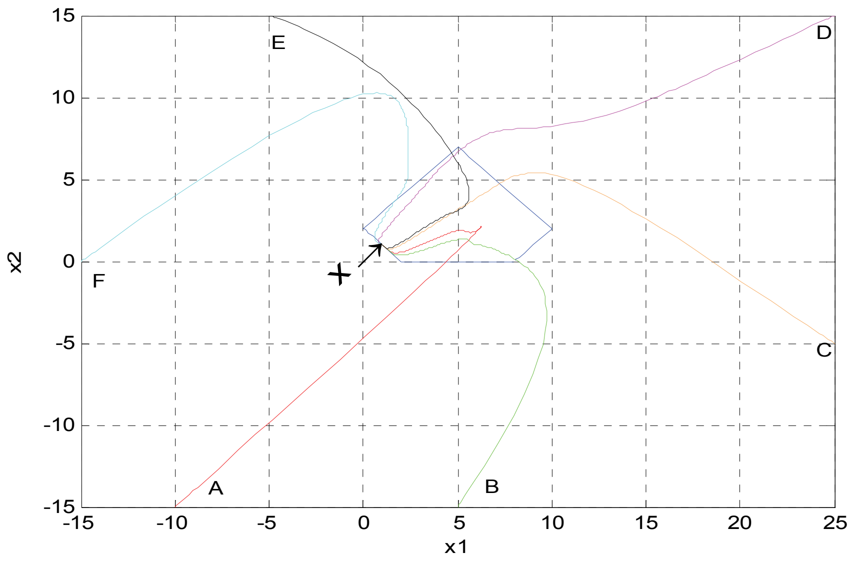


Fig. 5.4. Example 1: trajectories with initial points outside the feasible region using (Model 1).

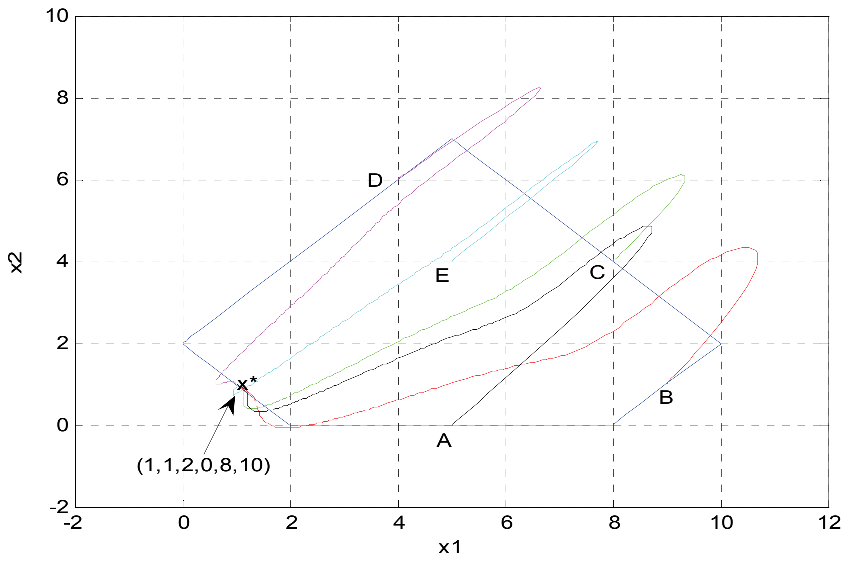


Fig. 5.5. Example 1: trajectories with initial points inside the feasible region using (Model 2).

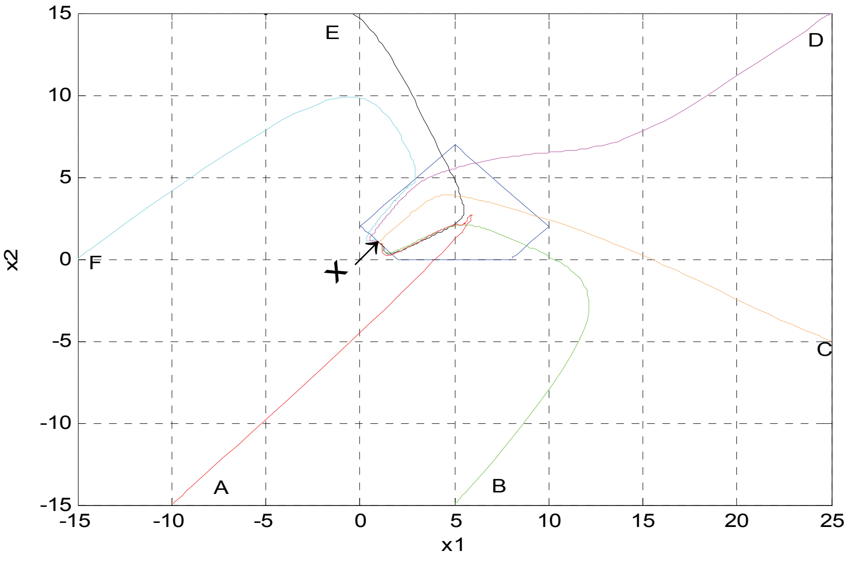


Fig. 5.6. Example 1: trajectories with initial points outside the feasible region using (Model 2).

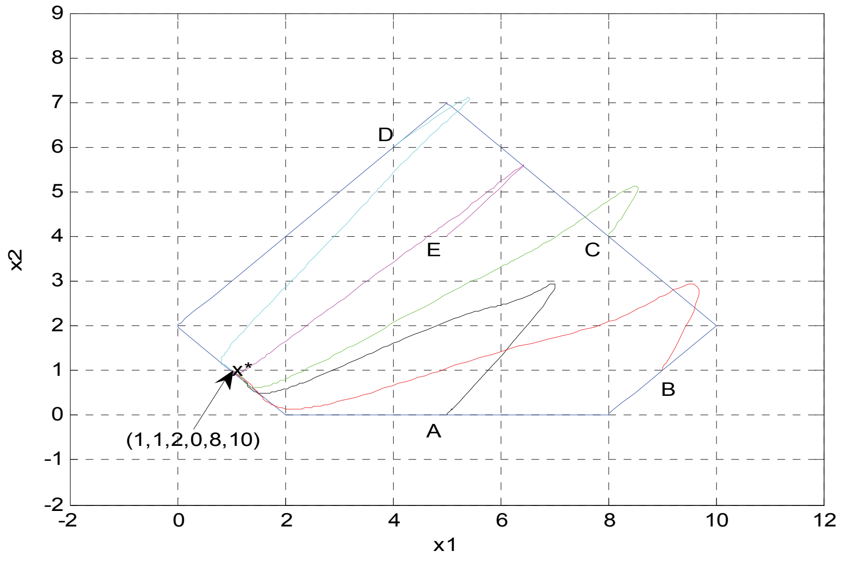


Fig. 5.7. Example 1: trajectories with initial points inside the feasible region using (Model 3).

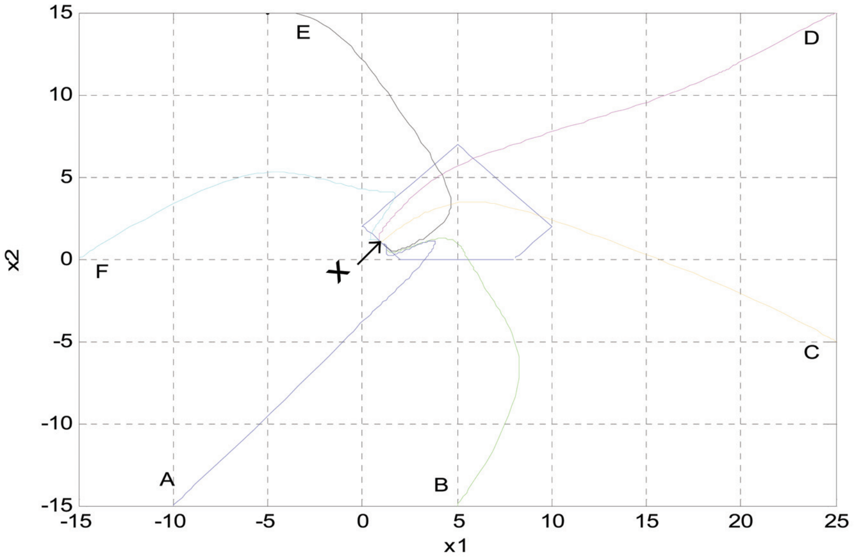


Fig. 5.8. Example 1: trajectories with initial points outside the feasible region using (Model 3).

Example 2. Consider the following nonlinear programming problem:

$$\textit{Minimize} \quad \frac{1}{4}x_1^4 + 0.5x_1^2 + \frac{1}{4}x_2^4 - 0.5x_2^2 - 30x_1x_2$$

$$\textit{Subject to} \quad x_1 - x_2 \geq -2\,,$$

$$\frac{1}{4}x_1 - x_2 \geq -\frac{13}{2}\,,$$

$$-4x_1 - x_2 = -4$$

$$-x_1 + x_2 = 1$$

$$x_1,\ x_2 \geq 0$$

Fig. 5.9 displays the transient behavior of $x(t)$ with seven initial points A(-5,-5), B(5,-5), C(15,0) D(15,10), E(5,15), F(-5,10) and G(-5,5).

Example 3. Consider the following convex nonlinear programming problem:

$$\textit{Minimize} \quad 0.4x_1 + x_1^2 + x_2^2 + 0.5x_3^2 - x_1x_2 + 0.5x_4^2 + \frac{x_1^3}{30}$$

$$\textit{Subject to} \quad x_1 - x_2 + x_3 \geq -2$$

$$-3x_1 - x_2 + x_3 + x_4 \geq -18$$

$$\frac{1}{3}x_1 + x_2 - x_4 = 2$$

$$x_1,\ x_2,\ x_3,\ x_4 \geq 0$$

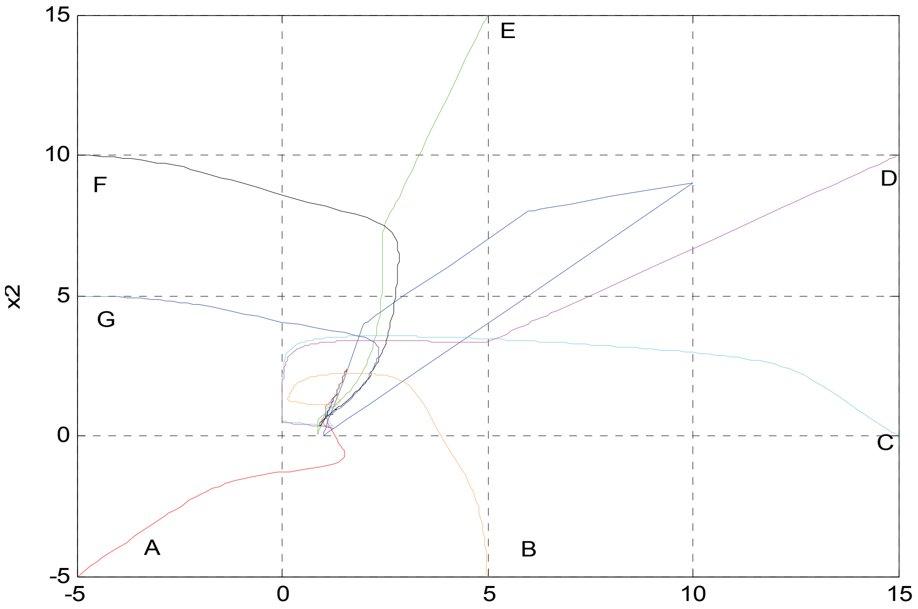


Fig. 5.9. Example 2: The transient behavior of $x(t)=(x_1(t),x_2(t))$, with initial points outside the feasible region.

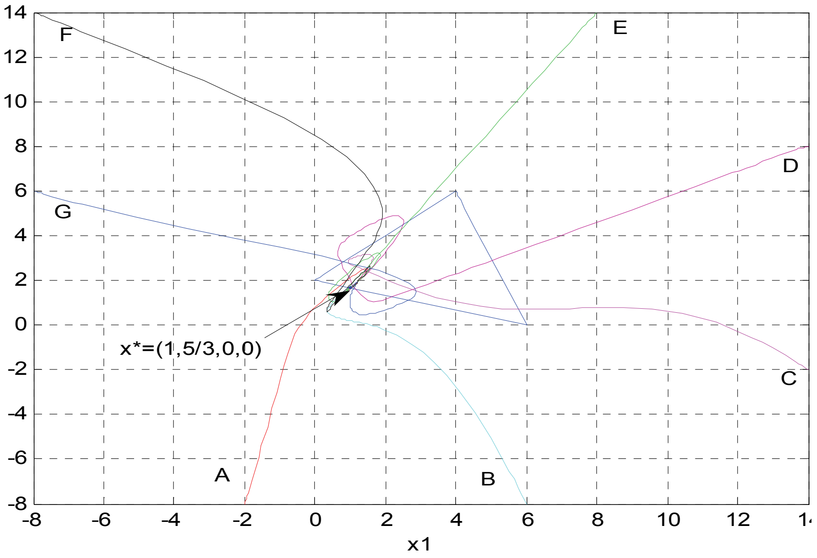
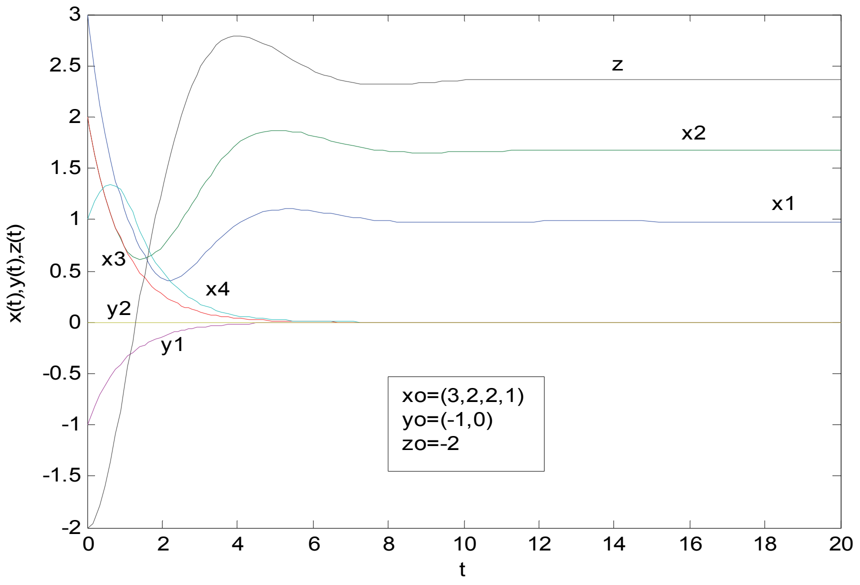


Fig. 5.10. Example 3: The transient behavior of $x(t)=(x_1(t),x_2(t))$, using the recurrent neural network model proposed by Yashtini and Malek [4].

(a)



(b)

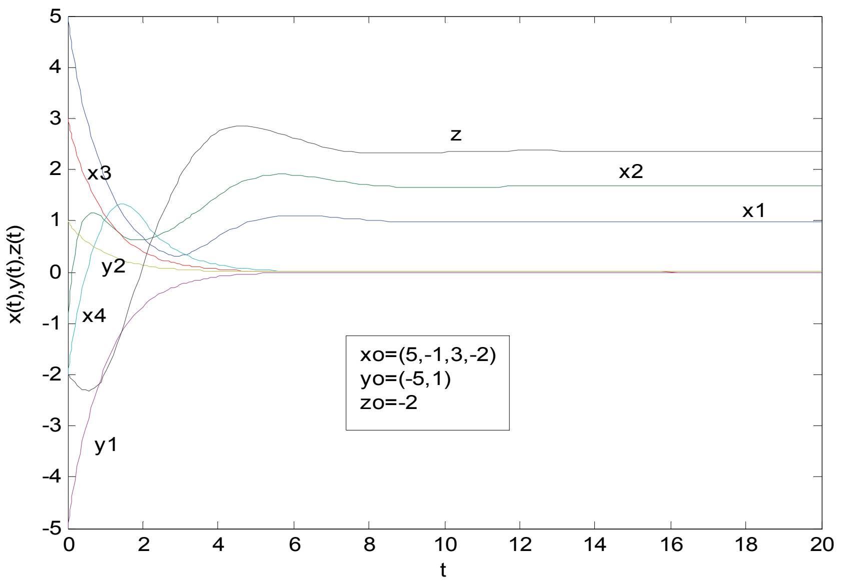


Fig. 5.11. Example 3: The transient behavior of the neural network model, Yashtini and Malek [4], for two different cases: (a) the feasible initial points and (b) the infeasible initial points.

Example 4. Consider the nonlinear variational inequalities problem. The mapping *F* and constraint set $\tilde{S}$ defined by

$$F(x) = \begin{bmatrix} 4x_1 - \dfrac{1}{x_2} + 2x_2 - 1 \\ x_1 + \dfrac{1}{2}x_2 \\ 2x_3 + 6x_4 \\ \dfrac{1}{3}x_3 + x_4 - \dfrac{1}{x_3} - 2 \end{bmatrix}$$

and $\tilde{S} = \{x \in R^4 \,|2x_1 + x_2 = 2,\ x_3 + 3x_4 \ge 2,\ l \le x \le h\}$ where $l = (0,\ 0.1,\ 0.3,\ 0)^T$ and $h = (8,\ 8,\ 8,\ 8)^T$.

In both cases trajectories converge to the *x**=(0.95, 0.1, 0.3, 5.233). Here y*=0, z*=-3.5.

Example 5. Consider the following linear variational inequality problem. The mapping F and constraint set $\tilde{S}$ defined by

$$F(x) = \begin{bmatrix} 4x_1 - 2x_2 + 8x_3 + 5 \\ -2x_1 + 8x_2 - 6x_3 + 6 \\ 8x_1 - 6x_2 + 12x_3 - 12 \end{bmatrix}$$

and

$$\tilde{S} = \{x \in R^4 \,|x_1 + 2x_2 + x_3 \ge 6,\ -x_1 - 2x_2 - x_3 \ge -16,\ \text{-}x_1 + 2x_2 = 4,\ l \le x \le h\},$$

where $l = (-7,\ -7,\ -7)^T$ and $h = (5,\ 5,\ 5)^T$.

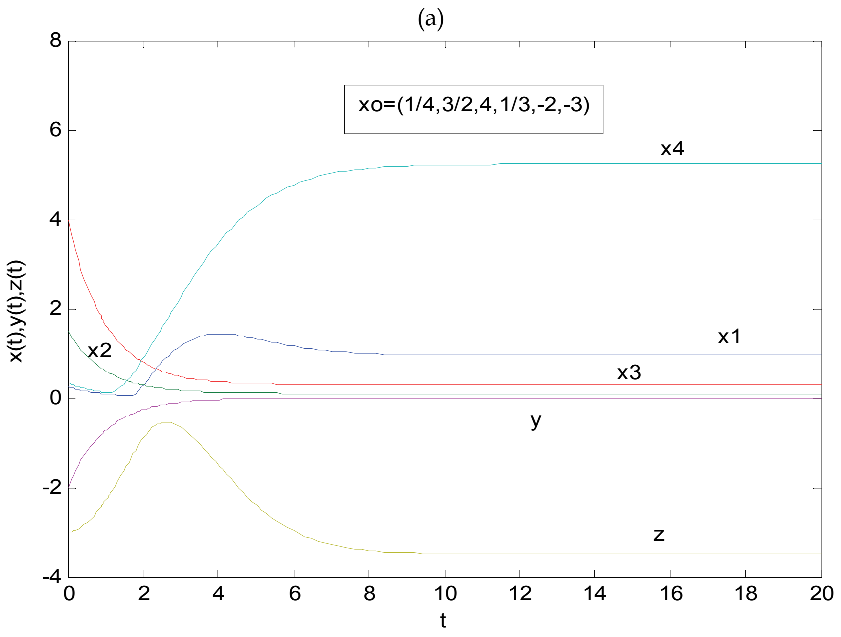
Example 6. Consider the following linear complementarity problem:

$$x \ge 0, \quad Qx + \theta \ge 0, \quad x^T(Qx + \theta) = 0,$$

Where

$$Q = \begin{bmatrix} 2 & -6 & 2 & -3 \\ 0 & 3 & -2 & 6 \\ -2 & 3 & 4 & -9 \\ 2 & -6 & -2 & 6 \end{bmatrix} \text{ and } .\theta = \begin{bmatrix} -5 \\ 6 \\ -3 \\ 4 \end{bmatrix}$$

This problem has one solution x*= (1.1668, 0, 1.333, 0). Fig. 5.12 depict the trajectories of neural network model (19) with initial points $(8,3,2,0)^T$ and $(3,1,2,6)^T$.



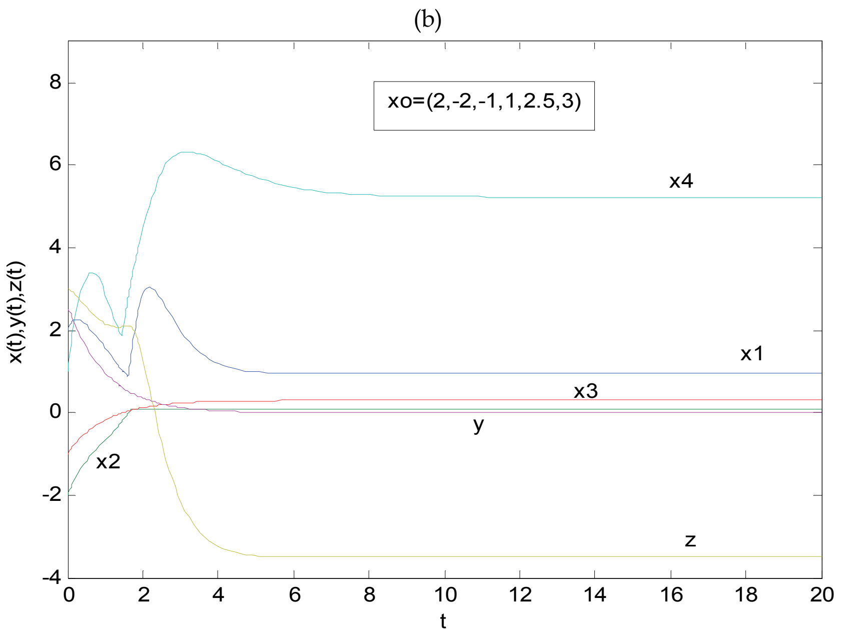


Fig. 5.12. Example 4: The transient behavior of the neural network model, Yashtini and Malek [4], for two different cases: (a) the feasible initial points and (b) the infeasible initial points.

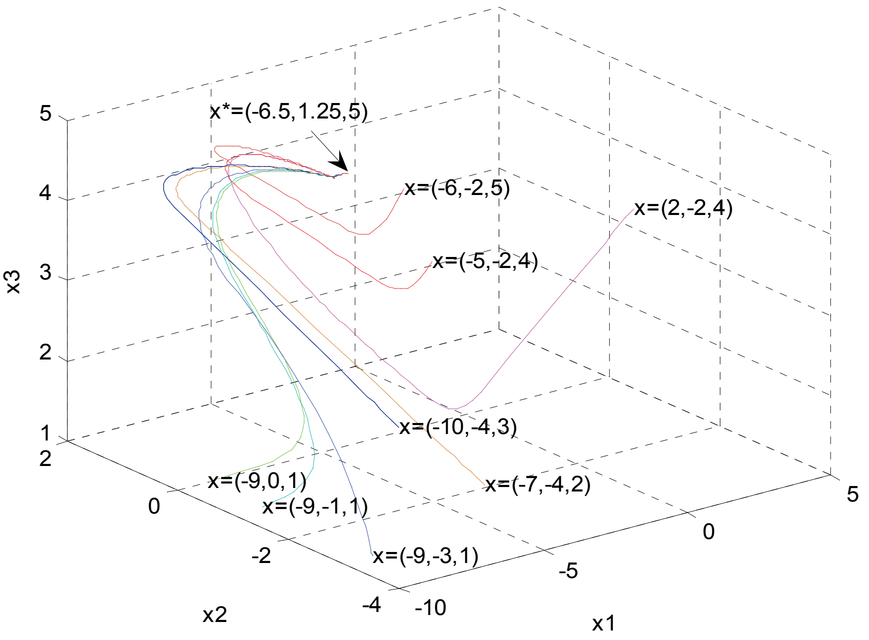


Fig. 5.13. Example 5: Simulation results for the neural network model Yashtini and Malek [4], with eight various initial points.

Example 7. Consider the following quadratic programming problem:

$$
\begin{aligned}
&\textit{Minimize} \quad x_1^2 + x_2^2 + x_1x_2 - 30x_1 - 30x_2,\\
&\textit{Subject to} \quad \frac{5}{12}x_1 - x_2 + x_3 = \frac{35}{12},\\
&\qquad \frac{5}{2}x_1 + x_2 + x_4 = \frac{35}{2},\\
&\qquad x_5 - x_1 = 5,\\
&\qquad x_2 + x_6 = 5,\\
&\qquad x_i \ge 0, (i = 1,2,\dots,6).
\end{aligned}
$$

and its dual:

$$
\begin{aligned}
&\textit{Minimize} \quad \frac{35}{12}y_1 + \frac{35}{2}y_2 + 5y_3 + 5y_4 - x_1^2 - x_2^2 - x_1x_2,\\
&\textit{Subject to} \quad \frac{5}{12}y_1 + \frac{5}{2}y_2 - y_3 - 2x_1 - x_2 \le -30,\\
&\qquad -y_1 + y_2 + y_4 - x_1 - 2x_2 \le -30.
\end{aligned}
$$

This problem is solved using models (23) and (24). Numerical results are shown in tables 1 and 2. These tables show that both models (23) and (24) are converging to the exact solution while model (23) is simpler to use and uses less expensive analog multipliers (see Malek & Alipour 2007).

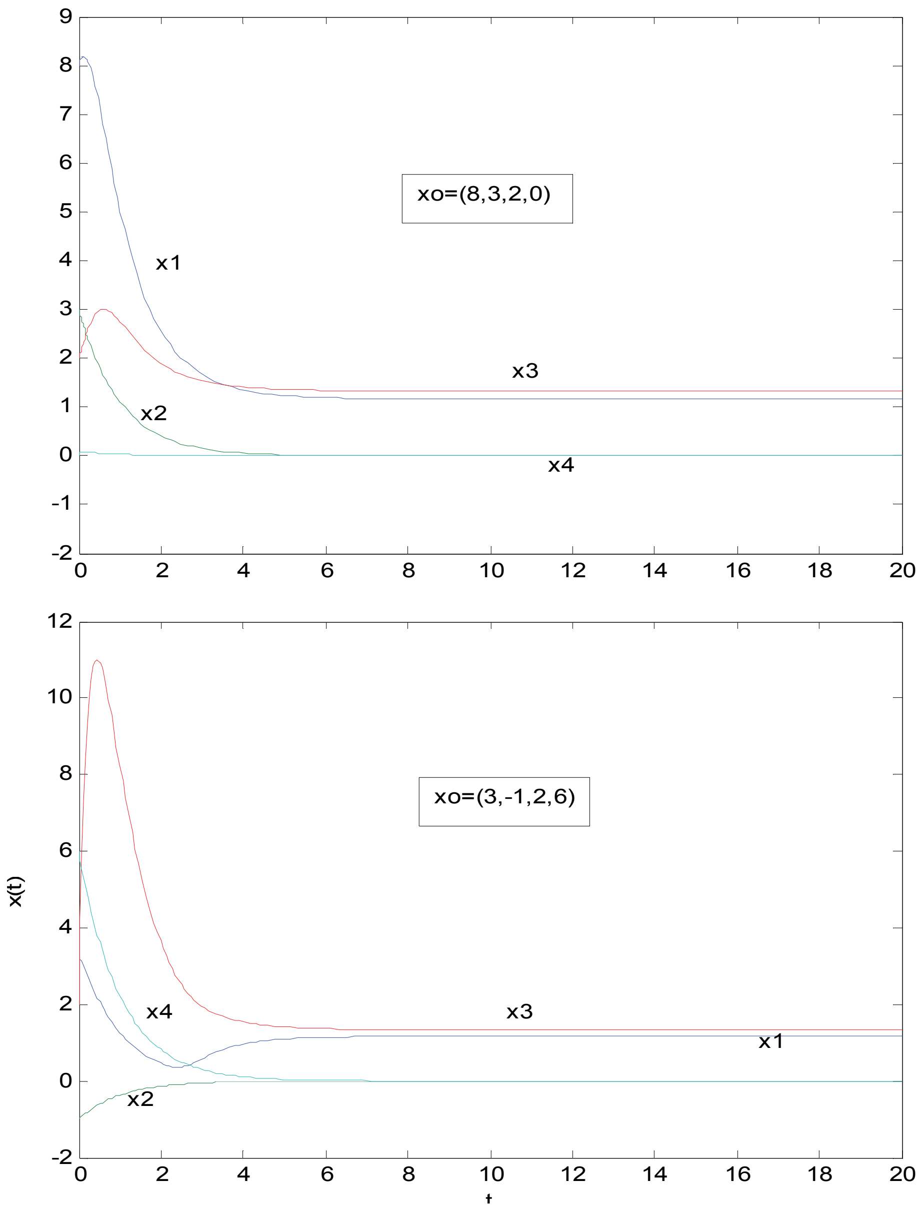


Fig. 5.14. Example 6: The transient behavior of the neural network model, Yashtini and Malek [4] using two different set of the initial points.

6. Exercises for the Reader

6.1 Smoothing filters:

It is often desirable to apply a smoothing filter to the measured date in order to reconstruct the underlying smooth function, where the noise is independent of the observed variable. We denote by $f = [f_1, ..., f_n]$ and $g = [g_1, ..., g_n]$ as measured data and smoothed data respectively. For a given vector f of length n consisting of measured data corrupted by random noise of δ =constant mean derivation, the smoothing filters problem is to find g_i, $i = 1, ..., n$ such that $|f_i - g_i| \le \delta$, on average. For n samples this condition can be written as

$$\sum_{i=1}^{n} (f_i - g_i)^2 \le n\delta^2 \text{ (Noise limiting condition)} \tag{31}$$

Now since our filtering problem consists in requiring that the continuous filtered curve $g(x)$ be as smooth as possible, we would require that

$$\mathit{Minimize} \int_{x_{\min}}^{x_{\max}} g''(x)^2 \, dx \tag{32}$$

A finite difference scheme for second derivative of $g(x)$ is [47]

$$g''(x) = \frac{g_{i+1} - 2g_i + g_{i-1}}{\Delta x^2}, \qquad \Delta x = x_{i+1} - x_i \quad \forall i \;\; i = 1, ..., n$$

Thus condition (32) is replaced by

$$\mathit{Minimize} \;\; \sum_{i=2}^{n-1} (g_{i+1} - 2g_i + g_{i-1})^2 \text{ (Smoothness condition)} \tag{33}$$

Then, by restating the optimization problem (33) and (31) in matrix notation we will have objective of

$\mathit{Minimize} \;\; \|Ag\|_2$

subject to the quadratic inequality constraint

$\mathit{Minimize} \;\; \|g - f\|_2^2 \le n\delta^2$

where $A_{(n-2)\times n}$ matrix is defined as

$$A = \begin{bmatrix} 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & . & . & . & & \\ & & & . & . & . & \\ & & & & . & . & . \\ & & & & 1 & -2 & 1 \end{bmatrix}$$

This is a quadratic optimization problem with quadratic conditions [48].

i. Solve this problem for $\delta = 0.1$, $n = 1000$ by the neural network models given in this chapter (Malek & coauthors).
ii. Compare your results with method of A. Savitzky and M.J.E Golay [49] and J. Steinier et al. [50].
iii. Use MATLAB or MAPEL to solve this problem.

6.2 Nonlinear programming via variational inequalities

Consider the nonlinear programming problem:

$$\begin{array}{ll} \textit{Minimize} & f(x) = 0.4x_1 + x_1^2 + x_2^2 - x_1x_2 + 0.5x_3^2 + 0.5x_4^2 + \dfrac{x_1^3}{30} \\ \textit{Subject to} & -0.5x_1 - x_2 + x_4 \geq -0.5, \\ & x_1 + 0.5x_2 - x_3 = 0.4, \\ & x_1, x_2, x_3, x_4 \geq 0. \end{array}$$

i. Show that this is a convex nonlinear programming problem.
ii. Use MATLAB or MAPEL to show that $x^* = (0.257, 0.258, 0, 0)^T$ is an optimal solution for this problem.
iii. Show that x^* In (ii) is also a solution for the monotone variational inequalities described in section 2.4, where $F(x) = \nabla f(x)$, and

$$S = \left\{ x \in R^4 \,\middle|\, -0.5x_1 - x_2 + x_4 \geq -0.5,\ x_1 + 0.5x_2 - x_3 = -0.4,\ x \geq 0 \right\}$$

iv. Use the dynamical system

$$\frac{d}{dt}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} (x - F(x) + A^T y + B^T y) - x \\ y - Ax + b \\ -Bx + c \end{pmatrix}$$

proposed in [4], in order to find the equilibrium value $u^*=(x^*, y^*, z^*)$, starting from (a) feasible initial point $(0.2, 1, 0.3, 0.75, -0.3, -0.5,)^T$ and (b) infeasible initial point $(0.5, 0.5, -0.1, -0.4, 0.9, -0.5)^T$
v. Depict the trajectories of the above dynamical system.

7. References

N. S. Kambo, Mathematical Programming Techniques, EWP LTD, New Delhi-Madras, 1984.
M.S. Bazaraa, C.M. Shetty, Nonlinear Programming, Theory and Algorithms, John Wiley and Sons, Inc., New York (1990).
L.Z. Liao, Q. Houduo, Q. Liqun, Solving non-linear complementarity problems with neural networks: a reformulation method approach, 1999.
M. Yashtini, A. Malek, Solving complementarity and variational inequalities problems using neural networks. Applied Mathematic and Computation 190 (2007) 216-230.
Xing-Bao Gao, Li-Zhi Liao and Liqun Qi, A Novel Neural Network for Variational Inequalities with Linear and Nonlinear Constraints, IEEE Trans. Neural networks 16 (6) (2005) 1305-1317.

M. Fukushima, Equivalent differentiable optimization problems and desent method for asymmetric variational inequality problems, Math program. 53 (1992) 99-110.

D. Kinderlehrer, G. Stampcchia, An Introduction to Variational Inequalities and Their Applications, New York: Academic, 1980.

M. Avriel, Nonlinear Programming: Analysis and Methods. Englewood Cliffs, NJ: Prentice-Hall, 1976.

P.T. Harker, J.S. Pang, Finite-dimensional variational inequality and nonlinear Complementarity problems: A survey of theory, algorithms, and applications, Math. Program. 48B (1990) 161-220.

B.S. He, L.-Z. Liao, Improvements of some projection methods for monotone nonlinear variational inequalities, J. Optim. Theory Appl. 112 (1) (2002) 111-128.

J.M. Ortega, W.C. Rheinboidt, Iterative Solution of Nonlinear Equation in Several Variables. New York: Academic, 1970.

B.S. He, Solution and applications of a class of general linear variational inequalities, Science China, ser. A, 39 (1996) 395-404.

B.S. He, A class of projection and contraction methods for monotone variational inequalities, Appl. Math. Optim., Ser. A, 35 (1997) 69-76.

Y.S. Xia, A new neural network for solving linear and quadratic programming problems, IEEE Transactions on Neural Networks 7(1996) 1544-1547.

Y.S. Xia , J. Wang, A general methodology for designing globally convergent optimization neural networks. IEEE Transactions on Neural Networks 9(1998) 1331-1343.

A. Malek and A. Yari, Primal-dual solution for the linear programming problems using neural networks, Applied Mathematics and Computation (2004) In press.

Q. Tao, J.D. Cao, M.S. Xue, H.Qiao. A high performance neural network for solving nonlinear programming problems with hybrid constraints, Phys. Lett. A, 288 (2) (2001) 88-94.

J. Wang, Q. Hu, D. Jiang, A lagrangian neural network for kinematics control of redundant robot manipulators, IEEE Trans. Neural Netw. 10 (5) (1999) 1123-1132.

A. Malek , R. Shekari Beidokhti, Numerical solution for high order differential equations using a hybrid neural network—Optimization method, Applied Mathematic and Computation 183 (2006) 260-271.

F.Pineda, "Generalization of backpropagation to recurrent neural networks," Physical Review Letters, vol. 18,pp 2229-2232, 1988.

T. Khanna, Foundations of Neural Networks. Addison-Wesley, Reading, Massachusets, 1990.

X. Y. Wu, Y. S. Xia, J. Li and W. K. Chen, A High Performance Neural Network for Solving Linear and Quadratic Programming Problems, IEEE Transactions on Neural Networks 7 (3) (1996) 643-651.

Y.S. Xia, H. Leung, J. Wang, A projection neural network and its application to constrained optimization problems, IEEE Trans. Circuits Syst. I, Fundam. Theory Appl. 49 (4) (2002) 447-458.

Y.S. Xia, J. Wang, A recurrent neural network for solving linear projection equations, Neural Network. 13 (2000)337-350.

Y.S. Xia, J. Wang, A dual neural network for kinemics control of redundant robot manipulators. IEEE Transactions on System, 3(2001) 147-154.

A. Malek, H.G. Oskoei, Primal-dual solution for the linear programming problems using neural networks, Appl. Math. Comput. 169 (2005) 451-471.

H. Ghasabi-Oskoei, A. Malek, A. Ahmadi, Novel artificial neural network with simulation aspects for solving linear and quadratic programming problems, Computer& Mathematics with Application, Computers and Mathematics with Applications 53 (2007) 1439-1454.

Rumelhart, D. E., Hinton, G. E. And Williams, R. J. (1986) learning internal representations by error propagation in Parallel Distributed processing vol. 1 & 2, MIT Press.

J. Hopfield, "Neurons with graded response have collective computational properties like those of two state neurons," Proceeding of the National Academy of Science, vol. 81, pp. 3088-3092, 1984.

Grossberg, Studies of Mind and Brain: Neural Principles of Learning, Perception, Development, Cognition, and Motor Control. Reidel, Boston, Masachussets, 1988.

D. W. Tank and J. J. Hopfield, Simple Neural Optimization Networks: An A/D Converter, Signal Decision Network, and Linear Programming Circuit, IEEE Transactions on Circuits and Systems CAS 33, (1986) 533-541.

L. O. Chua and G. N. Lin, Nonlinear Programming without Computation, IEEE Transaction On Circuits and Systems CAS 31 (1984) 182-188.

G. Wilson, Quadratic programming analogs, IEEE Trans. Circuits Syst. CAS 33 (9) (1986) 907-911 1986.

J. B. Dennis, Mathematical programming and electrical networks, Chapman and Hall, London, 1959.

M. V. Rybashov, The gradient method for solving convex programming problems on electronic analog computers, Automation and Remote Contr. 26 (11) (1965) 1886-1898.

M. V. Rybashov, Gradient method for solving convex programming problems on electronic analog computers, Automation and Remote Contr. 26 (12) (1965) 2079-2089.

N. N. Karpinskaya, Method of "Penalty" functions and the foundations of Pyne's method, Automation and Remote Contr. 28, (1) (1967) 124-129.

M.P. Kennedy, L.-O. Chua, Unifying the Tank and Hopfield Linear Programming Ne- twork and the Canonical Nonlinear Programming Circuit of Chua and Lin, IEEE Tra- nsactions On Circuits and Systems CAS, 34: 210-214 (1997).

A. Rodriguez-Vazquez, Dominguer-Castro, A. Rueda, J. L. Huertas, and E. Sanchez-Sinenico, Nonlinear switched-capacitor "neural" networks for optimization problems, IEEE Trans. Circ. Syst. 37 (1990) 384-397.

Y.-H. Chen, S.-C. Fang, Solving Convex Programming Problems with Equality Const-raints By Neural Networks, Vol. 36, No. 7: 41-68 (1998).

A. Malek, M. Alipour, Numerical solution for linear and quadratic programming problems using a recurrent neural network, Appl. Math. Comput. 192 (2007) 27-39.

D.P. Bertsekas, J.N. Tsitsiklis, Parallel and Distributed Computation: Numerical Methods. Englewood Cliffs, NJ: Prentice-Hall, 1989.

D.G. Luenberger, Introduction To Linear Nonlinear Programming, Addison-Wesley, MA, (1973).

R.K. Miller, A.N. Michel, Ordinary Diffrential Equations, Newral Network: Academic, 1982.

J.P. LaSalle, The Stability of Dynamical Systems, Siam ,Philadelphia , PA, (1976).

A. Malek, M. Yashtini, Novel neural network for solving nonlinear programming problem with nonlinear constraints, submitted.

G. D. Smith, Numerical solution of partial differential equations: finite difference methods. 3rd ed. Oxford Applied Mathematics and Computing Science series, 1986.

U. Von Matt, Large constraned Quadratic problems, Verlagder Fuchrereine, Zurich.

A. Savitzky and M. J. E. Golay, Smoothing and Differentiation of Data by simplified Least Squares Procedures, Analytical Chemistry, 36 (1964), pp. 1627-1639.

J. Steinier, Y. Termonia and J. Deltour, Comments on Smoothing and Differentiation of Data by Simplified least Square Procedure, Analytical Chemistry, 44 (1972), pp. 1906-1909.

Initial point: primal variables	Initial point: dual variables	Primal (feasible or infeasible)	Dual (feasible or infeasible)	Optimal values for primal problem (x^*)	Optimal values for dual problem (y^*)	Primal		Dual	
						Optimal objective function value	Absolute error	Optimal objective function value	Absolute error
-5	0	Feasible	Feasibl	5.000	-1.478 e-011				
-5	-12			4.999	-5.999	-224.999		-225.000	
0	24			5.833	1.822 e-012				
35	-35			1.644 e-034	-9.000		9.288 e-011		1.313 e-010
0				10.000					
10				1.804 e-034					
-5	0	Feasible	Infeasible	4.999	-4.203 e-011				
-5	-1			5.000	-5.999	-225.000		-225.000	
0	0			5.833	1.937 e-011				
35	-2			1.569 e-033	-9.000		3.317 e-010		8.378 e-011
0				9.999					
10				2.110 e-034					
3	0	Infeasible	Feasible	5.000	-3.137 e-011				
0	-12			4.999	-5.999	-224.999		-225.000	
7	24			5.833	6.945 e-012				
5	-35			9.028 e-035	-9.000		6.778 e-011		2.160 e-010
-4				10.000					
1				1.804 e-035					
3	0	Infeasible	Infeasible	4.999	-2.430 e-011				
0	-1			5.000	-5.999	-225.000		-225.000	
7	0			5.833	9.282 e-012				
5	-2			9.313 e-035	-9.000		1.111 e-010		8.773 e-011
-4				10.000					
1				1.804 e-035					

Table 1. Numerical results for primal and dual quadratic problems in section 5 using four different initial points (feasible and infeasible) using model (23), proposed by Malek & Alipour, Applied Mathematics and Computation 192 (2007) 27-39

Initial point: primal variables	Initial point: dual variables	Primal (feasible or infeasible)	Dual (feasible or infeasible)	Optimal values for primal problem (x^*)	Optimal values for primal problem (y^*)	Primal		Dual	
						Optimal objective function value	Absolute error	Optimal objective function value	Absolute error
-5	0	Feasible	Feasibl	4.999	-6.145 e-011				
-5	-12			5.000	-5.999	-225.000		-225.000	
0	24			5.833	2.631 e-011				
35	-35			1.302 e-012	-9.000		1.109 e-010		1.213 e-010
0				10.000					
10				-2.014 e-011					
-5	0	Feasible	Infeasible	4.999	-1.186 e-011				
-5	-1			5.000	-5.999	-225.000		-225.000	
0	0			5.833	5.998 e-012				
35	-2			3.990 e-012	-8.999		2.232 e-010		1.466 e-011
0				9.999					
10				-1.725 e-011					
3	0	Infeasible	Feasible	4.999	-6.944 e-011				
0	-12			5.000	-5.999	-225.000		-225.000	
7	24			5.833	2.989 e-011				
5	-35			2.152 e-012	-9.000		1.6217 e-010		1.355 e-010
-4				10					
1				-2.519 e-011					
3	0	Infeasible	Infeasible	4.999	-6.934 e-012				
0	-1			5.000	-5.999	-225.000		-225.000	
7	0			5.833	3.347 e-012				
5	-2			1.694 e-012	-8.999		9.606 e011		1.006 e-011
-4				9.999					
1				-7.803 e-012					

Table 2 Numerical results for primal and dual quadratic problems in section 5 using four different initial points (feasible and infeasible) using model (24) , proposed by Malek & Alipour, Applied Mathematics and Computation 192 (2007) 27-39

Neurodynamic Optimization: Towards Nonconvexity

Xiaolin Hu
*Tsinghua National Lab for Information Science and Technology,
State Key Lab of Intelligent Technology and Systems,
Department of Computer Science and Technology,
Tsinghua University, Beijing 100084,
China*

1. Introduction

Optimization is a ubiquitous phenomenon in nature and an important tool in engineering. As the counterparts of biological neural systems, properly designed artificial neural networks can serve as goal-seeking computational models for solving various optimization problems in many applications. In many engineering applications such as optimal control and signal processing, obtaining real-time locally optimal solutions is more important than taking time to search for globally optimal solutions. In such applications, recurrent neural networks are usually more competent than numerical optimization methods because of the inherent parallel nature.

Since the seminal work of Tank and Hopfield in 1980s (Hopfield & Tank, 1986; Tank & Hopfield, 1986), recurrent neural networks for solving optimization problems have attracted much attention. In the past twenty years, many models have been developed for solving convex optimization problems, from the earlier proposals including the penalty method based neural network (Kennedy & Chua, 1988), the switched-capacitor neural network (Rodriǵuez-Vázquez et al., 1990) and the deterministic annealing neural network (Wang, 1994), to the latest development including (Xia, 2004; Gao, 2004; Gao et al., 2005; Hu & Wang, 2007b; Hu & Wang, 2007c; Hu & Wang, 2008). These latest models have a common characteristic: they were all formulated based on optimality conditions of the problems and therefore their equilibria correspond exactly to the solutions of the problems. In addition, for ensuring this correspondence, in contrast to many earlier proposals such as the penalty-based neural network (Kennedy & Chua, 1988), there is no need to let any parameter go infinity. More importantly, if these neural networks are applied to solve nonconvex optimization problems, this nice property will be retained in the sense of critical points instead of global optima, e.g., Karush-Kuhn-Tucker (KKT) points (i.e., the equilibria will correspond no longer to the global optima but to these critical points). Naturally, one will ask if these models are suitable for searching for critical points, especially local optima, of general nonconvex optimization problems.

Unfortunately, there is no guarantee that these optimality-conditions-based neural networks can be directly adopted to solve nonconvex optimization problems. In designing recurrent neural networks for optimization, letting the equilibria correspond to solutions is just one

issue. The other issue that cannot be neglected is to ensure the stability of the networks at these equilibria. In fact, if the above mentioned neural networks are directly applied to nonconvex problems, their dynamic behaviors could change drastically and become unpredictable. This is not like the circumstance of extending penalty-based neural networks for constrained convex optimization to solve constrained nonconvex problems. In that case, the performances of the networks for solving nonconvex problems can be predicted easily based on their performances in solving convex counterparts, e.g., if a network is previously globally convergent to some points, then it is locally convergent to these points now.
So far, no much achievement in this direction has been obtained yet. In the chapter, I will review some recent progress made by us along this route. Our primary aim is to design locally or globally convergent recurrent neural networks (1) for solving special nonconvex optimization problems whose local minima are also global, and (2) for seeking Karush-Kuhn-Tucker points of general nonconvex optimization problems. The two issues are presented in Section 3 and Section 4, respectively, after a brief introduction of some preliminaries in Section 2. Section 5 summarizes the findings and discusses several possible future directions related to this topic.

2. Preliminaries

Throughout the chapter, without specifications, the following notations are adopted. $\Re^n$ denotes the n dimensional real space and $\Re^n_+$ denotes its nonnegative quadrant. If a function $g : \Re^n \rightarrow \Re$, then $\nabla g \in \Re^n$ stands for its gradient and $\nabla^2 g \in \Re^{n\times n}$ stands for its Hessian matrix. If $g(x, y) : \Re^n \times \Re^m \rightarrow \Re$, then $\nabla_x g(x,y) \in \Re^n$ and $\nabla_{xx} g(x,y) \in \Re^{n\times n}$ are viewed as respectively the gradient and Hessian matrix of g with respect to x. If a function $G : \Re^n \rightarrow \Re^m$, $\nabla G \in \Re^{m\times n}$ stands for its Jacobian matrix. The transpose of a real matrix A is denoted by A^T. A square matrix A is said to be positive definite (positive semidefinite), denoted by $A > 0$ ($A \geq 0$), if $x^TAx > 0$ ($x^TAx \geq 0$) $\forall x \neq 0$. $\|x\| = \sqrt{\sum_{i=1}^n x_i^2}$ denotes the L_2 norm of a vector x.
In many recurrent neural networks, the following projection operator is used as their activation functions

$$P_\Omega(x) = \arg\min_{y\in\Omega} \|x - y\|, \tag{1}$$

where Ω is a closed convex set and "arg" stands for the solution of the minimization problem adhering to it. In general, computing the projection of a point onto a convex set Ω is itself an optimization problem (see (Hu & Wang, 2008) for a neurodynamic solution to such a problem). But if Ω is a box set or a sphere set, the calculation is straight forward. For instance, if $\Omega = \{x \in \Re^n | l_i \leq x_i \leq u_i, \forall i = 1, ..., n\}$, then $P_\Omega(x) = (P_\Omega(x_1),\ldots, P_\Omega(x_n))^T$ and

$$P_\Omega(x_i) = \begin{cases} l_i, & x_i < l_i, \\ x_i, & l_i \leq x_i \leq u_i, \\ u_i, & x_i > u_i. \end{cases} \tag{2}$$

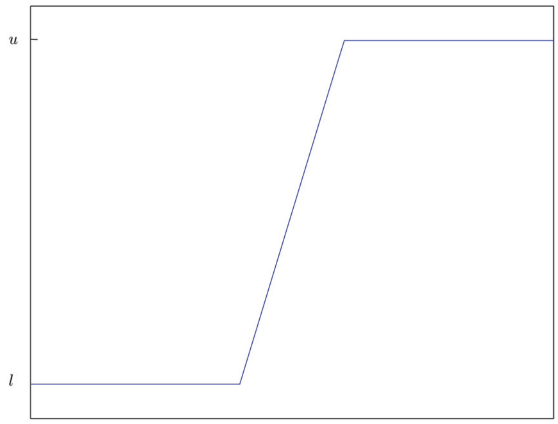


Figure 1. Projection operator in one dimensional case. Reprint of Fig. 1.3 in (Hu, 2007).

Note u_i might be $+\infty$ and l_i might be $-\infty$. Fig. 1 illustrates this operator in one dimensional case, which is somewhat similar in shape to the sigmoid activation function in the Hopfield neural network (cf. Fig. 3(A) in (Hopfield & Tank, 1986)). In particular, if $l = 0$ and $u = \infty$, the operator becomes $P_{\Re^n_+}(x)$. To simplify the notation, in this case it is written as x^+. And the definition can be simplified as $x^+ = (x_1^+, \ldots, x_n^+)^T$ with $x_i^+ = \max(x_i, 0)$.

For another instance, if $\Omega = \{x \in \Re^n | \|x - c\| \le r, r > 0\}$ where $c \in \Re^n$ and $r \in \Re$ are two constants. Then

$$P_\Omega(x) = \begin{cases} x, & \|x - c\| \le r, \\ c + \frac{x-c}{\|x-c\|}, & \|x - c\| > r. \end{cases}$$

Definition 1 (Lipschitz Continuity) *A function F: $\Re^n \to \Re^n$ is said to be Lipschitz continuous with constant L on a set D if, for every pair of points $x, y \in D$,*

$$\|F(x) - F(y)\| \le L\|x - y\|.$$

F is said to be locally Lipschitz continuous on D if each point of D has a neighborhood $D_0 \subset D$ such that the above inequality holds for every pair of points $x, y \in D_0$.

Definition 2 (Convexity) *A function $f : \Re^n \to \Re$ is convex over a convex set D if for all $x, y \in D$, and $0 < \alpha < 1$*

$$f(\alpha x + (1 - \alpha)y) \le \alpha f(x) + (1 - \alpha)f(y).$$

f(x) is strictly convex on D if above strict inequality holds whenever $x \ne y$.

Lemma 1 *A differentiable function $f : \Re^n \to \Re$ is convex on a convex set D if and only if for every pair of distinct points $x, y \in D$,*

$$f(y) \ge f(x) + \nabla f(x)^T (y - x).$$

f(x) is strictly convex if and only if above strict inequality holds whenever $x \ne y$.

3. Solving pseudoconvex optimization problems

In this section we consider solving the following problem

$$\min\ f(x) \quad \text{s.t. } x \in \Omega \tag{3}$$

where $f : \Re^n \to \Re$ is a differentiable nonconvex function and $\Omega \subseteq \Re^n$ is a box set or sphere set defined in Section 2.
To pave the way for discussion, some additional definitions are needed.
Definition 3 (Pseudoconvexity) *A differentiable function* $f : \Re^n \to \Re$ *is pseudoconvex on a convex set D if for every pair of distinct points x, y ∈D,*

$$\nabla f(x)^T(y-x) \geq 0 \Longrightarrow f(y) \geq f(x).$$

f is strictly pseudoconvex on D if for every pair of distinct points x, y ∈D,

$$\nabla f(x)^T(y-x) \geq 0 \Longrightarrow f(y) > f(x),$$

and strongly pseudoconvex on D if there exists a constant $\beta > 0$ *such that for every pair of points x, y ∈D,*

$$\nabla f(x)^T(y-x) \geq 0 \Longrightarrow f(y) \geq f(x) + \beta\|y-x\|^2.$$

Definition 4 (Pseudomonotonicity) *A function F:* $\Re^n \to \Re^n$ *is pseudomonotone on a convex set D if, for every pair of distinct points x, y ∈D,*

$$F(x)^T(y-x) \geq 0 \Longrightarrow F(y)^T(y-x) \geq 0.$$

F is strictly pseudomonotone on D if, for every pair of distinct points x, y ∈D,

$$F(x)^T(y-x) \geq 0 \Longrightarrow F(y)^T(y-x) > 0,$$

and strongly pseudomonotone on D if there exists a constant $\gamma > 0$ *such that for every pair of points x, y ∈D,*

$$F(x)^T(y-x) \geq 0 \Longrightarrow F(y)^T(y-x) \geq \gamma\|x-y\|^2.$$

It is shown in (Karamardian & Schaible, 1990) that a differentiable function is pseudoconvex and strictly pseudoconvex if and only if its gradient is a pseudomonotone and strictly pseudomonotone mapping, respectively. Moreover, if its gradient is strongly pseudomonotone, the function is strongly pseudoconvex; but the converse is not true (Hadjisavvas & Schaible, 1993).
Lemma 2 *Suppose a differentiable function* $f : \Re^n \to \Re$ *is pseudoconvex on* $\Omega \subset \Re^n$*. Then a point* $x^* \in \Omega$ *satisfies*

$$\nabla f(x^*)^T(x-x^*) \geq 0 \quad \forall x \in \Omega$$

if and only if x is a minimum of f(x) in Ω.*
One of the important classes of pseudoconvex optimization problems are the quadratic fractional problems in the following form:

$$\begin{aligned} \min \quad & f(x) = \frac{x^T Q x + a^T x + a_0}{b^T x + b_0} \\ \text{s.t.} \quad & x \in \Re^n \in \mathcal{X} = \{x | b^T x + b_0 > 0\}, \end{aligned} \tag{4}$$

where Q is an $n \times n$ symmetric matrix, a, $b \in \Re^n$, a_0, $b_0 \in \Re$. It is well known (e.g., Avriel et al., 1988) that f is pseudoconvex on $\mathcal{X}$ when $Q \geq 0$. Conditions for f being pseudoconvex on $\mathcal{X}$ when Q is not positive semidefinite are discussed in (Cambini et al., 2002). Specially, when $b = 0$, problem (4) reduces to the classic quadratic programming problem, and when $Q = 0$ it reduces to the so-called *linear fractional problem*, which is of course pseudoconvex on $\mathcal{X}$ (Bazaraa et al., 1993).

Throughout this section, $f(x)$ in (3) is assumed to be pseudoconvex on Ω and ∇f is assumed to be Lipschitz continuous on Ω. Note that if f is twice continuously differentiable in an open set containing Ω, then the latter assumption is satisfied automatically.

3.1 Two-layer projection neural network

Consider a recurrent neural network for solving (3) whose dynamics is governed by

$$\frac{dx}{dt} = \lambda\{-x + P_\Omega(x - \alpha F(x)) + \alpha F(x) - \alpha F[P_\Omega(x - \alpha F(x))]\}, \tag{5}$$

where $\lambda > 0$ and $\alpha > 0$ are two scaling factors, $P_\Omega : \Re^n \to \Omega$ is the projection operator defined in section 2, and $F(x)$ stands for $\nabla f(x)$. The architecture of the network is illustrated in Fig. 2. In contrast to the projection neural network, which has a one-layer structure and will be discussed in next subsection, for convenience, the above network is termed *two-layer projection neural network* or TLPNN for short in the chapter.

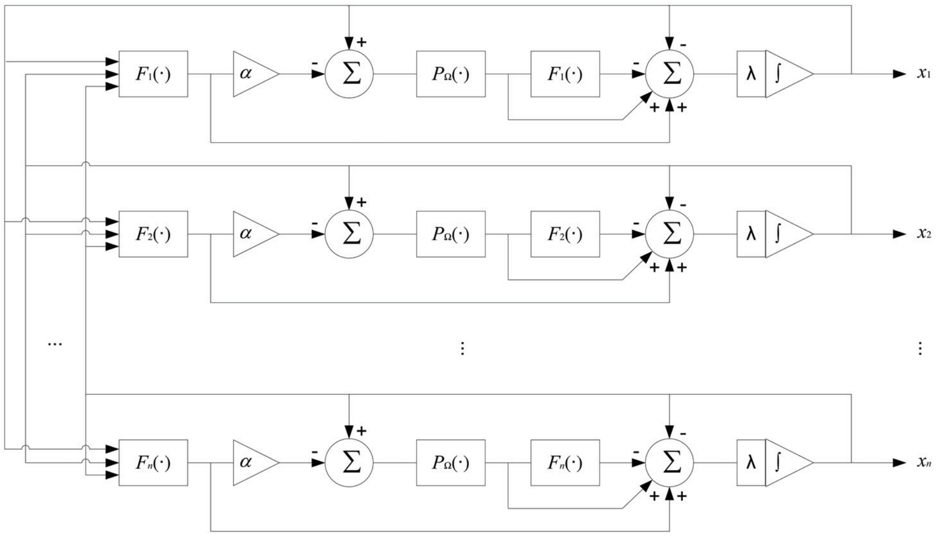


Figure 2. Architecture of the TLPNN (5). Reprint of Fig. 2.1 in (Hu, 2007).

It is proved in (Xia & Wang, 1998) that $x^* \in \Omega$ is a solution of (3) if and only if it is an equilibrium point of the neural network (5). The dynamic behavior of the system was first discussed in (Xia & Wang, 1998), and later in (Xia & Wang, 2000) with different convexity assumptions. In (Hu & Wang, 2006a) we have shown that the corresponding results are still valid when the neural network is employed to solve pseudoconvex optimization problems in the form of (3) (of course with some additional assumptions). The results are contained in the following theorem, which is a restatement of Theorems 2 and 3 in (Hu & Wang, 2006a).

Theorem 1 *Assume that $\nabla f(x)$ is Lipschitz continuous in $\Re^n$ with a constant L.*

- *The TLPNN is globally convergent to a solution of* (3) *with $\alpha < 1/L$. In particular, if* (3) *has a unique solution, the neural network is globally asymptotically stable.*
- *If $\nabla f(x)$ is strongly pseudomonotone on Ω with constant γ, where $\gamma > 4L$, then the TLPNN is globally exponentially stable with $\alpha < (\gamma - 4L)/\gamma L$.*

Remark 1 *Note that the Lipschitz continuity of $\nabla f(x)$ in $\Re^n$ is a stronger condition than the Lipschitz continuity in Ω.*

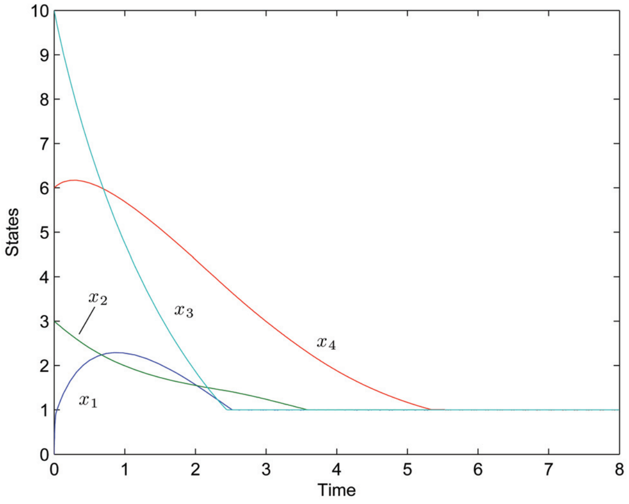


Figure 3. Transient behavior of the TLPNN (5) in Example 1.

Example 1 *We now use the TLPNN to solve a quadratic fractional programming problem* (4) *with*

$$Q = \begin{pmatrix} 5 & -1 & 2 & 0 \\ -1 & 5 & -1 & 3 \\ 2 & -1 & 3 & 0 \\ 0 & 3 & 0 & 5 \end{pmatrix}, \quad a = \begin{pmatrix} 1 \\ -2 \\ -2 \\ 1 \end{pmatrix}, \quad b = \begin{pmatrix} 2 \\ 1 \\ 1 \\ 0 \end{pmatrix},$$
$$a_0 = -2, \quad b_0 = 4.$$

It is easily verified that Q is symmetric and positive definite in $\Re^4$, and consequently f is pseudoconvex on $\mathcal{X} = \{x \in \Re^4 \mid b^T x + b_0 > 0\}$. We minimize f over $\Omega = \{x \in \Re^4 \mid 1 \le x_i \le 10, i = 1,\ldots, 4\} \subset \mathcal{X}$ by using the TLPNN with

$$F(x) = \frac{(b^T x + b_0)(2Qx + a) - b(x^T Qx + a^T x + a_0)}{(b^T x + b_0)^2}.$$

This problem has a unique solution $x^* = (1, 1, 1, 1)^T$ *in* Ω. *Simulations show that the TLPNN* (5) *is globally asymptotically stable at* x^* *with any initial point if* α *is appropriately selected. For instance, Fig. 3 shows that the trajectories of the neural network with* $\lambda = 100$, $\alpha = 0.01$ *and the initial point* $x_0 = (0, 3, 6, 10)^T$ *converge to* x^*.

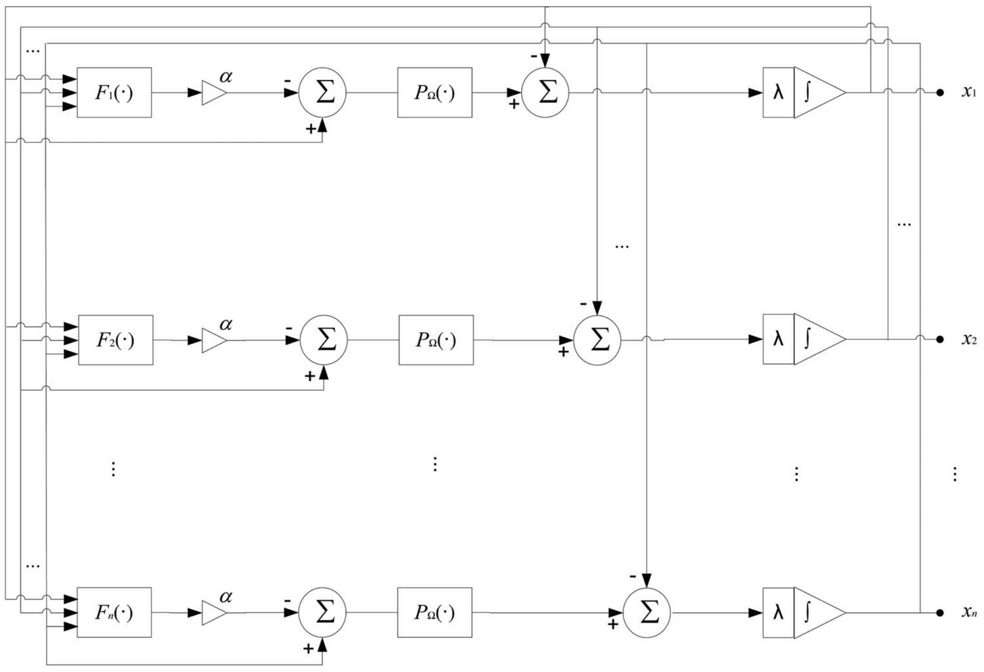


Figure 4. Architecture of the PNN (6). Reprint of Fig. 3.1 in (Hu, 2007).

3.2 One-layer projection neural network

Consider a simpler neural network, called the *projection neural network* or PNN, for solving problem (3) whose dynamic behavior is governed by the following equation

$$\frac{dx}{dt} = \lambda\{-x + P_\Omega(x - \alpha F(x))\}, \tag{6}$$

where the notations are the same as in (5). According to (Kinderlehrer & Stampcchia, 1980), x^* is a solution of (3) if and only if it is an equilibrium point of the PNN. One of the merits of this neural network is its simplicity compared with the TLPNN. The architecture of the network is illustrated in Fig. 4. Its stability results were presented in (Hu & Wang, 2006b, Corollary 3) which is restated as follows.

Theorem 2 *Assume that* $f(x)$ *is twice continuously differentiable on an open set containing* Ω. *Then the PNN* (6) *is stable in the sense of Lyapunov and globally convergent to a solution of* (3). *Moreover,*

- *If rf is strongly pseudomonotone on Ω and there exists $\delta > 0$ such that $f(x) \leq \delta\|x - x^*\|^2$, where x^* is the unique solution of (3), then the neural network is globally exponentially stable;*
- *If ∇f is strongly pseudomonotone on Ω and $\|\nabla f(x)\|$ has an upper bound on Ω, then the neural network is globally asymptotically stable at the unique solution of* (3), *while the convergence rate is upper bounded by*

$$\|x(t) - x^*\| \leq \sqrt{\frac{1}{a + b(t - t_0)}} \quad \forall t \geq t_0,$$

where a, b are two positive constants.

Example 2 *We now use the PNN to solve the pseudoconvex optimization problem in Example 1. Simulations show that the PNN* (6) *is globally asymptotically stable at x^* with any α, λ and any initial point. For instance, Fig.* 5 *shows that the trajectories of the neural network with* λ = α = 1 *and the initial point* $x_0 = (0, 3, 6, 10)^T$ *converge to x^*.*

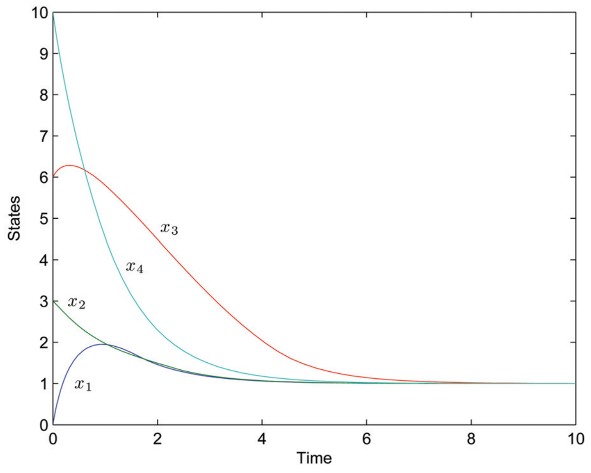


Figure 5. Transient behavior of the PNN (6) in Example 2. Reprint of Fig. 6 in (Hu & Wang, 2006b).

4. Solving general nonconvex optimization problems

Pseudoconvex optimization problems in the form of (3) represent a very special case in the family of nonconvex optimization problems. In this section let's consider solving the following generally constrained nonconvex optimization problem:

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g(x) \leq 0, \quad x \in \mathcal{X} \end{aligned} \tag{7}$$

where $f : \Re^n \rightarrow \Re$, $g(x) = [g_1(x),\ldots,g_m(x)]^T$ is an m-dimensional vector-valued function of n variables, and $\mathcal{X}$ is a box set or a sphere set defined in Section 2. In what follows, the functions f, $g_1(x),\ldots,g_m(x)$ are assumed to be twice continuously differentiable. If all functions $f(x)$ and $g_j(x)$ are convex over $\Re^n$, the problem is called a *convex optimization problem*; otherwise, it is called a *nonconvex optimization problem*, which is what we are interested in here. Equation (7) represents a wide variety of optimization problems. For example, it is well known that if a problem has equality constraints $h(x) = 0$, then this constraint can be expressed as $h(x) \leq 0$ and $- h(x) \leq 0$.

For solving general nonconvex optimization problems (including pseudoconvex optimization problems (3) where Ω is a general convex set instead of box set or sphere set), no much progress has been made in the neural network community. This is mainly due to the difficulty in characterizing global optimality of nonconvex optimization problems by means of explicit equations. From the optimization context, it is known that under fairly mild conditions an optimum of the problem must be a Karush-Kuhn-Tucker (KKT) point, while the KKT points are easier to characterize. In terms of developing neural networks for global optimization, it is very hard to find global optima at the very beginning; and a more attainable goal at present is to design neural networks for seeking local optima first with the aid of KKT conditions.

To pave the way for discussion, some additional notations and definitions are needed in this section. In what follows, let $I = \{1, \ldots ,n\}$, $J = \{1, \ldots ,m\}$. If $u \in \Re^n$, then $u^p = (u_1^p, \ldots , u_n^p)^T$ where p is an integer; $\Gamma(u) = \mathrm{diag}(u_1,\ldots, u_n)$. int$\mathcal{S}$ denotes the interior of a set $\mathcal{S}$.

Definition 5 *A solution x satisfying the constraints in* (7) *is called a feasible solution. A feasible solution x is said to be a regular point if the gradients of* $g_j(x)$, $\nabla g_j(x)$, $\forall j \in \{ j \in J \,|\, g_j(x) = 0\}$, *are linearly independent.*

Definition 6 *A point* x^* *is said to be a strict minimum of the problem in* (7) *if* $f(x^*) < f(x)$, $\forall x \in K(x^*) \cap S$, *where* $K(x^*)$ *is a neighborhood of* x^* *and S is the feasible region of the problem.*

According to (Kinderlehrer & Stampcchia, 1980), the Karush-Kuhn-Tucker (KKT) condition (Bazaraa et al., 1993) for problem (7) can be expressed as

$$\begin{cases} P_{\mathcal{X}}(x - \alpha(\nabla f(x) + \nabla g(x)y)) = x, \\ (y + \alpha g(x))^+ = y, \end{cases} \tag{8}$$

where $\alpha > 0$, $y \in \Re^m$ and $\nabla g(x) = (\nabla g_1(x), \ldots, \nabla g_m(x))$.

The classical Lagrangian function associated with problem (7) is defined as

$$L(x, y) = f(x) + \sum_{j=1}^{m} y_j g_j(x). \tag{9}$$

Note that the Hessian of the Lagrangian function is calculated as

$$\nabla_{xx} L(x, y) = \nabla_{xx} f(x) + \sum_{j=1}^{m} y_j \nabla_{xx} g_j(x). \tag{10}$$

Lemma 3 (Second-order sufficiency conditions (Bazaraa et al., 1993)) *Suppose that* x^* *is a feasible point to problem (7) and* $x^* \in \mathrm{int}\, \mathcal{X}$. *If there exists* $y^* \in \Re^m$, *such that* (x^*, y^*) *is a KKT point pair and the Hessian matrix* $\nabla_{xx}L(x^*, y^*)$ *in* (10) *is positive definite on the tangent subspace:*

$$M(x^*) = \{d \in \Re^n | d^T \nabla g_j(x^*) = 0, \forall j \in J(x^*)\},$$

where $J(x^*)$ is defined by

$$J(x^*) = \{j \in J | y_j^* > 0\}, \tag{11}$$

then $x^*$is a strict minimum point of problem (7).

In what follows, let $\Omega = \mathcal{X} \times \Re_+^n$ and Ω^* denote the KKT point set of (7) or the solution set of (8).

4.1 Local convergence of the extended projection neural network

In a series of papers (Xia & Wang, 2004; Xia, 2004; Xia & Feng, 2005; Xia et al., 2007), a recurrent neural network, termed extended projection neural network (or EPNN for short), was developed for solving the convex optimization problems in the form of (7) with the following dynamical equation:

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} P_{\mathcal{X}}(x - \alpha(\nabla f(x) + \nabla g(x)y)) - x \\ (y + \alpha g(x))^+ - y \end{pmatrix}, \tag{12}$$

where $\alpha > 0$. According to the projection formulation (8), the equilibria of the above EPNN correspond to the KKT points of problem (7) exactly. If problem (7) is convex, then the KKT points correspond to the global optima, and the EPNN solves the problem. One wonders what will happen if (12) is used to solve a nonconvex program in the form of (7). Contrary to our expectation, in the nonconvex case, the EPNN can not be guaranteed to converge to any KKT point (which may not correspond to a global optimum), even locally, as will be shown by numerical examples later on. It is thus demanded to find some necessary and/or sufficient conditions that guarantee the local convergence of the neural network. The following theorem provides such a set of sufficient conditions, which is an improved version of Theorem 9.4 in (Hu, 2007).

Theorem 3 *Let x^* be a feasible and regular point of problem* (7), *and $u^* = ((x^*)^T, (y^*)^T)^T$ be the corresponding KKT point of the problem. If the Hessian matrix $\nabla_{xx}L(x^*; y^*)$ in* (10) *is positive definite, then the EPNN* (12) *is asymptotically stable at u^*, and x^* is a strict local minimum of the problem.*

Remark 2 In *Theorem 9.4 of (Hu, 2007) there is an additional requirement on u^*: it should satisfy the second-order sufficiency conditions in Lemma 3. This requirement is actually unnecessary as it can be covered by the positive definiteness of $\nabla_{xx}L(x^*; y^*)$.*

4.2 Augmented Lagrange networks

Theorem 3 reveals that if the Hessian matrix of the Lagrangian function is positive definite at a local minimum solution, the EPNN (12) may be locally convergent to that local optimum. But in many cases, this condition fails to exist. Fortunately, there exist ways to generate this condition, and one popular technique is to utilize the augmented Lagrangian functions (Li & Sun, 2000).

In 1992, Zhang and Constantinides proposed a neural network based on the augmented Lagrangian function for seeking local minima of the following equality constrained optimization problem (Zhang & Constantinides, 1992):

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & h(x) = 0, \end{aligned}$$

where $f : \Re^n \rightarrow \Re$, $h : \Re^n \rightarrow \Re^m$ and both f and h are assumed twice continuously differentiable. The dynamic equation of the network is as follows

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -(\nabla f(x) + \nabla h(x)y + c\nabla h(x)h(x)) \\ h(x) \end{pmatrix},$$

where $c > 0$ is a control parameter. Under the second-order sufficiency conditions, the neural network can be shown convergent to local minima with appropriate choice of c. The disadvantage of the neural network lies in that it handles equality constraints only. Though in theory inequality constraints can be converted to equality constraints by introducing slack variables, the dimension of the neural network will inevitably increase, which is usually not deemed a good strategy in terms of model complexity.

An alternative extension of the neural network in (Zhang & Constantinides, 1992) for handling inequality constraints in (7) directly can be found in (Huang, 2005) and its dynamic system is as follows (the bound constraint $x \in \mathcal{X}$ is not considered explicitly in that paper):

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -(\nabla f(x) + \nabla g(x)y^2 + c\nabla g(x)\Gamma(y^2)g(x)) \\ 2\Gamma(y)g(x) \end{pmatrix}.$$

The local convergence of the neural network to its equilibrium set, denoted by $\hat{\Omega}^e$, was proved by using the linearization techniques, and moreover, $\Omega^* \subset \hat{\Omega}^e$. However, it is clear that $\hat{\Omega}^e \neq \Omega^*$. For example, any critical point x of the objective function, which makes $\nabla f(x) = 0$, and $y = 0$ constitute an equilibrium point of the neural network, but in rare cases such an equilibrium corresponds to a KKT point.

Other augmented Lagrangians associated with problem (7) could be tested from the viewpoint of recurrent neural networks. But whether a particular Lagrangian is suited for the design of recurrent neural networks does not have a straightforward answer. For example, the *essentially quadratic augmented Lagrangian* discussed in (Sun et al., 2005) might be a candidate, but its Hessian matrix is not continuous which lays difficulties for analyzing the convergence of the resulting neural networks. On the other hand, the *exponential-type augmented Lagrangian* does have a continuous Hessian matrix, but as the reformulation raises the constraints to exponents of some exponential functions which causes numerical difficulties, that method rarely works in practice. In what follows, we discuss about two promising augmented Lagrangians without these difficulties. For convenience, the resulting neural networks are termed *Augmented Lagrange Networks*.

4.2.1 Partial *p*-power augmented Lagrangian

Problem (7) can be written as

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & \hat{g}(x) \leq b, \quad x \in \mathcal{X} \end{aligned} \tag{13}$$

where $\hat{g}(x) = g(x) + b$. Consider the partial p-power transformation of (13):

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & [\hat{g}_j(x)]^p \le [b_j]^p, \quad j \in J, \\ & x \in \mathcal{X} \end{aligned} \tag{14}$$

with $p \geq 1$. If we assume that $b_1,\ldots,b_m$ are positive constants and $g_1(x),\ldots,g_m(x)$ are nonnegative over $\mathcal{X}$, then problem (13) is equivalent to (14). This assumption does not impose a strict restriction on problem (13) as we can always apply some suitable equivalent transformation on the problem if necessary. Correspondingly, the standard Lagrangian function of problem (14) is defined as

$$L_p(x,y) = f(x) + \sum_{j=1}^{m} y_j([\hat{g}_j(x)]^p - [b_j]^p),$$

where $y_j \geq 0$; $j \in J$, which can be regarded as an augmented Lagrangian function associated with the original problem (7). Then, from (12), the neural network for solving (14) becomes

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} P_{\mathcal{X}}(x - \alpha(\nabla f(x) + \nabla \tilde{g}(x) y)) - x \\ (y + \alpha \tilde{g}(x))^+ - y \end{pmatrix}, \tag{15}$$

where $\tilde{g}(x) = [\hat{g}(x)]^p - [b]^p$.It is easy to calculate

$$\nabla \tilde{g}(x) = p\left([\hat{g}_1(x)]^{p-1}\nabla \hat{g}_1(x), \ldots, [\hat{g}_m(x)]^{p-1}\nabla \hat{g}_m(x)\right).$$

Problem (14) is termed *partial p-power transformation* of the problem (13) (Li & Sun, 2000). The following lemma reveals one of the advantages of the transformation.

Lemma 4 (Li & Sun, 2000) *Let x^* be a local optimal solution of* (13) *and $x^* \in int\mathcal{X}$. Assume that x^* is a regular point and satisfies the second-order sufficiency conditions. If* $J(x^*) \neq \varnothing$in (11), then *there exists a $q > 0$ such that the Hessian of the partial p-power Lagrangian function, $\nabla_{xx}L_p(x^*, y^*)$, is positive definite when $p > q$.*

Hence we have the following stability results about neural network (15), which follows from Theorem 3 and Lemma 4.

Theorem 4 *Let x^* be a feasible and regular point of problem* (13), *and $u^* = ((x^*)^T; (y^*)^T)^T$ be the corresponding KKT point of the problem satisfying the second-order sufficiency conditions in Lemma 3. Then there exists $p > 0$ such that the neural network* (15) *is asymptotically stable at u^*, and x^* is a strict local minimum of the problem.*

Example 3 *Consider the following nonconvex programming problem in (Li & Sun, 2000)*

$$\begin{aligned} min \quad & f(x) = 1 - x_1 x_2, \\ s.t. \quad & g(x) = x_1 + 4x_2 \le 1, \quad x \in \Re^2_+. \end{aligned}$$

This problem has only one local solution $x^*=$ (0.5, 0.125)^T, *thus also the global solution. The solution is located on the boundary of the feasible region (see Fig. 6). It can be verified that*

$$y^* = 0.125 \quad and \quad M(x^*) = \{d \in \Re^2 | d_1 + 4d_2 = 0\}.$$

The Hessian of the Lagrangian function is

$$\nabla_{xx} L(x^*, y^*) = \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix},$$

which is indefinite.

Now consider the partial p-power formulation (14) *of the problem. When* $p = 3$*, a direct calculation yields the new optimal Lagrangian multiplier* $y^* = 0.0417$ *and the Hessian of the new Lagrangian*

$$\nabla_{xx} L_p(x^*, y^*) = \begin{pmatrix} 0.25 & 0 \\ 0 & 4 \end{pmatrix},$$

which is a positive definite matrix. We simulate the neural network (15) *to solve the problem. Fig. 7 shows the transient behavior of the neural network* (15) *with the initial point* $u(t_0) = (0.5, 0.2, 0.125)^T$ *that is very close to* u^**. When* $p = 1$*, the neural network is identical to* (12) *and it does not converge to* u^**. But when* $p \geq 1.5$*, the neural network converges. When* $p = 3$*, Fig. 8 displays the transient behavior of* $x(t)$ *with several initial points* $u(t_0) = (x(t_0), y(t_0))$ *chosen as follows:* $y(t_0)$ *is random chosen and* $x(t_0)$ *is chosen as* $P_1(0.8, 0.1)$*,* $P_2(0.3, 0.5)$*,* $P_3(0, 0.2)$*,* $P_4(0.4,-0.3)$*. From Fig. 8, it is observed that all four trajectories converges to* x^* *eventually, although the trajectory started from* P_2 *exhibits obvious instability at its earlier evolving stage.*

Moreover, all simulations show that the neural network does not converge to the other KKT points corresponding to the maximum solution $\bar{x}^* = (0, 0)^T$*, even after the partial p-power transformation. This is because*

$$\nabla^2 f(\bar{x}^*) = \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}$$

is not positive semidefinite.

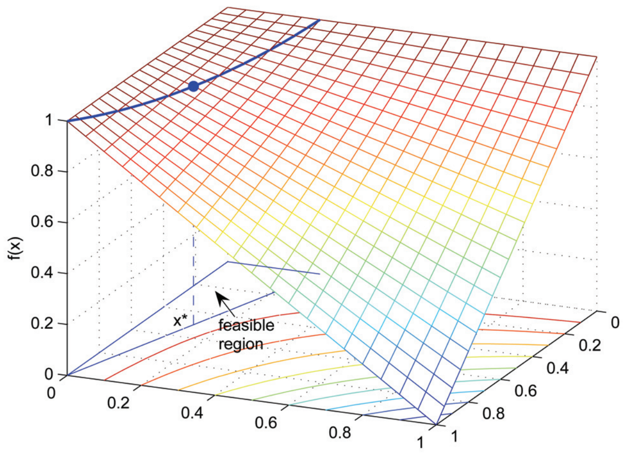


Figure 6. Isometric view of the objective function and constraints in Example 3. Reprint of Fig. 9.1 in (Hu, 2007).

4.2.2 A new augmented Lagrangian

Consider the following augmented Lagrangian function associated with problem (7) slightly differing from that in (Huang, 2005):

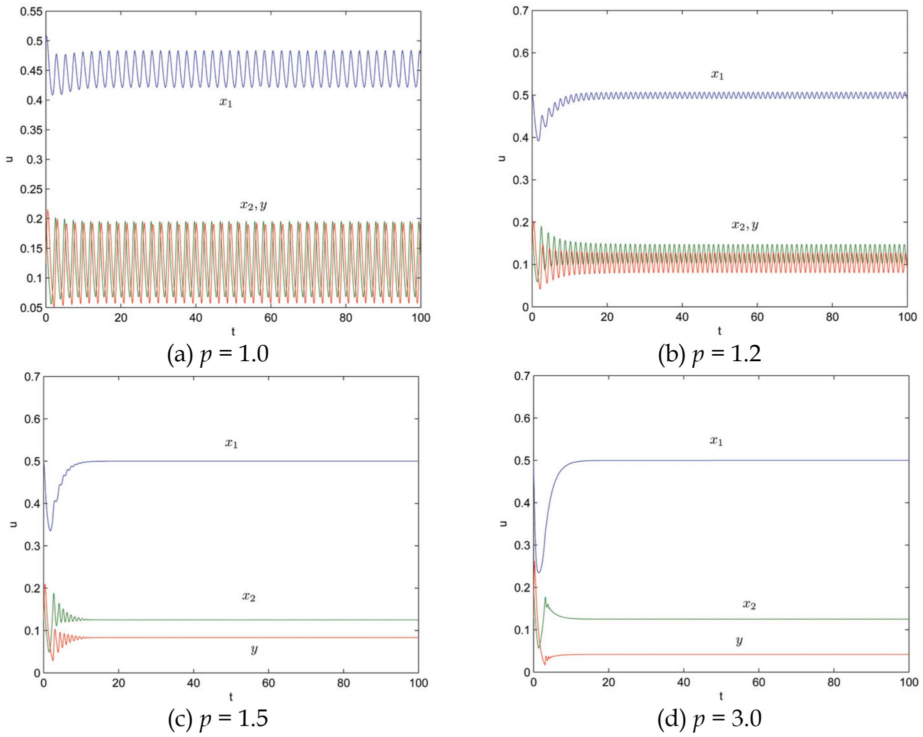


Figure 7. Transient behavior of the neural network (15) with $u(t_0) = (0.5, 0.2, 0.125)^T$ and different values of p in Example 3. Reprint of Fig. 9.2 in (Hu, 2007).

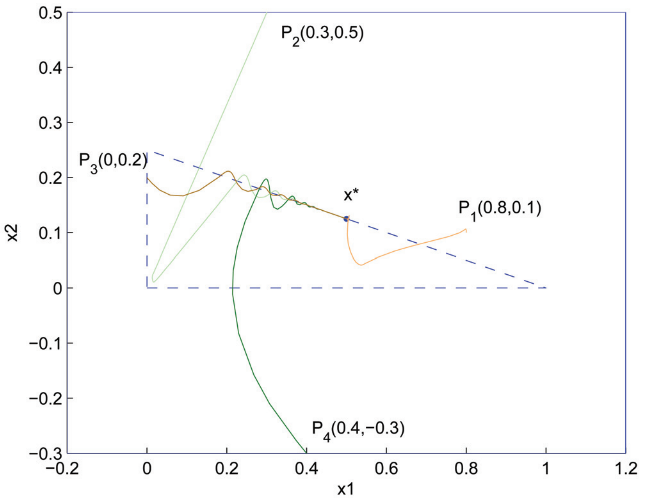


Figure 8: State trajectories ($x_1(t)$, $x_2(t)$) of the neural network (15) with p = 3 and four initial points in Example 3. Reprint of Fig. 9.3 in (Hu, 2007).

$$L_c(x,y) = L(x,y) + \frac{c}{2}\sum_{j=1}^{m}(y_j g_j(x))^2,$$

where $L(x, y)$ is the regular Lagrangian function defined in (9) and $c > 0$ is a scalar. Let Ω^e denote the solution set of the following equations

$$\begin{cases} P_{\mathcal{X}}(x - \nabla_x L_c(x,y)) = x, \\ (y + \alpha g(x))^+ = y, \end{cases}$$

where $\alpha > 0$. We have the following theorem.

Theorem 5 $\Omega^* = \Omega^e$.

Consider a recurrent neural network with its dynamic behavior governed by

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} P_{\mathcal{X}}(x - \alpha(\nabla f(x) + \nabla g(x)y + c\nabla g(x)\Gamma(y^2)g(x))) - x \\ -y + (y + \alpha g(x))^+ \end{pmatrix}, \tag{16}$$

where $\alpha > 0$, $c > 0$ are two contents. Note that the term $\nabla f(x) + \nabla g(x)y + c\nabla g(x)\Gamma(y^2)g(x)$ on the right-hand-side is the expansion of $\nabla_x L_c(x, y)$. Therefore the equilibrium set of the neural network is actually Ω^e, which is equal to Ω^* as claimed in Theorem 5.

Theorem 6 *Let x^* be a feasible and regular point of problem* (7), *and $u^* = ((x^*)^T, (y^*)^T)^T$ be the corresponding KKT point of the problem satisfying the second-order sufficiency conditions in Lemma 3. Then there exists $c > 0$ such that the neural network* (16) *is asymptotically stable at u^*, and x^* is a strict local minimum of the problem.*

The proofs of Theorems 5 and 6 can be found in (Hu & Wang, 2007a) and (Hu, 2007).

Example 4 *Consider the problem in Example 3 again. This time we use the new augmented Lagrange network* (16) *to solve it. Fig. 9 shows the transient behavior of the neural network* (16) *with the initial point $u(t_0) = (0.5, 0.2, 0.125)^T$ (same as in Example 3). When $c \le 1.5$, the neural network does not converge, and when $c \ge 2$ the neural network converges to u^*. When $c = 5$, Fig. 10 displays the transient behavior of $x(t)$ with four initial points chosen in a similar way as in Example* 3. *It is observed that all four trajectories converges to x^* eventually.*

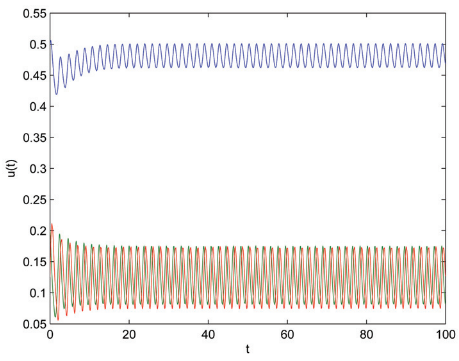
Example 5 *Consider the following problem*

$$\begin{aligned} min \quad & f(x) = 5 - (x_1^2 + x_2^2)/2, \\ s.t. \quad & g_1(x) = -x_1^2 + x_2 - 1 \le 0, \\ & g_2(x) = x_1^4 - x_2 \le 0. \end{aligned}$$

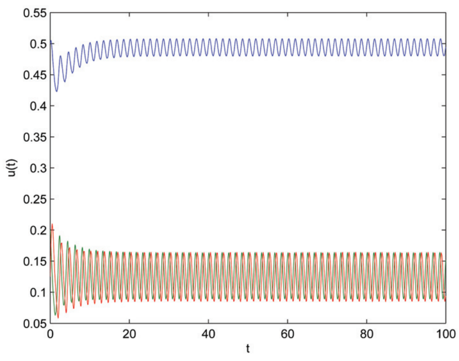
As both $f(x)$ and $g_1(x)$ are concave, the problem is a nonconvex optimization problem. Fig. 11 *shows the contour of the objective function and the solutions to $g_1(x) = 0$ and $g_2(x) = 0$ on the x_1-x_2 plane. The feasible region is the nonconvex area enclosed by the bold curves. Simple calculations yield*

$$\nabla_{xx}L(x,y) = \begin{pmatrix} -2y_1 + 12x_1^2 y_2 - 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

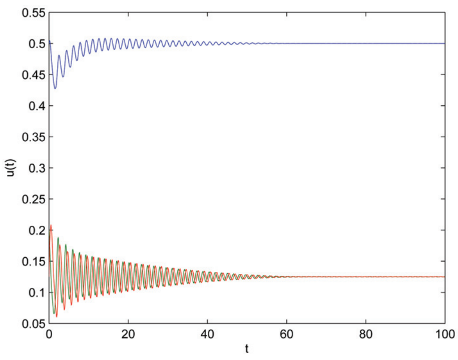
Evidently, $\nabla_{xx}L(x, y)$ *is not positive definite over the entire real space, and the neural network* (12) *can not be applied to solve the problem. Now we check if the neural network (16) can be used to search for the KKT points. There are four KKT points associated with the problem:* $u_1^* = (-1.272, 2.618, 4.013, 1.395)^T$, $u_2^* = (1.272, 2.618, 4.013, 1.395)^T$, $u_3^* = (0, 0, 0, 0)^T$, $u_4^* = (0, 1, 1, 0)^T$, *but only the first two correspond to local minima. Moreover, it is verified that at either* u_1^* or u_2^*, $J(x^*)$ *defined in Lemma 3 is equal to* {1, 2}, *and* $\nabla g_1(x^*)$, $\nabla g_2(x^*)$ *are linearly independent, which indicates* $M(x^*) = 0$. *So the second-order sufficiency conditions holds trivially at either point. According to Theorem* 6, *the neural network* (16) *can be made asymptotically stable at* u_1^* *and* u_2^* *by choosing appropriate* $c > 0$. *Fig. 12 displays the state trajectories of the neural network with different values of c started from the same initial point* $(-2, 3, 0, 0)^T$. *When* $c = 0$, *the neural network reduces to the neural network* (12). *It is seen from Fig.* 12(a) *that some trajectories diverge to infinity. When* $c = 0.1$, *the neural network is not convergent, either, as shown in Fig.* 12(b). *However, when* $c \geq 0.2$, *in Figs.* 12(c) *and* 12(d) *we observe that the neural network converges to* u_1^* *asymptotically.*



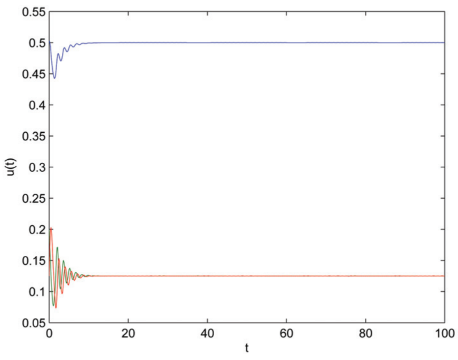
(a) $c = 1.0$



(b) $c = 1.2$



(c) $p = 1.5$



(d) $p = 3.0$

Figure 10. State trajectories $(x_1(t), x_2(t))$ of the neural network (16) with $c = 5$ and four initial points in Example 4.

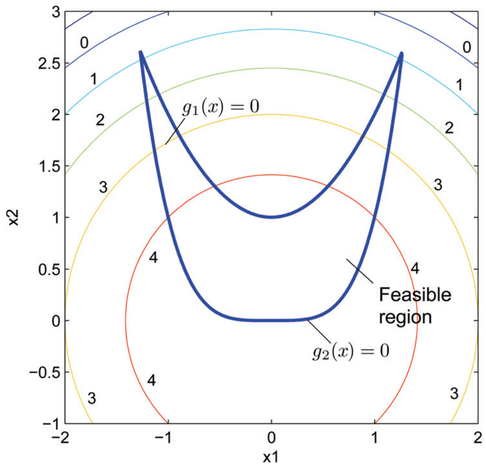


Figure 11. Contour of the objective function and the feasible region in Example 5. Reprint of Fig. 1 in (Hu & Wang, 2007a).

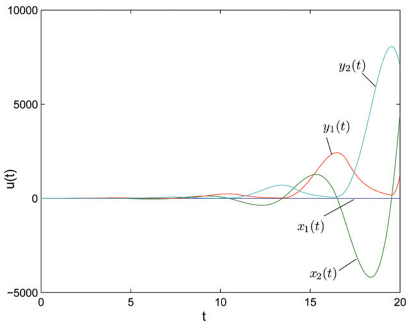
5. Concluding remarks

5.1 Summary of contents

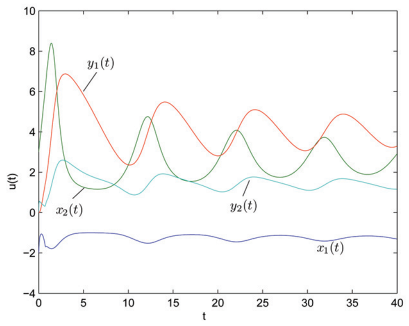
This chapter summarizes our recent work in designing recurrent neural networks for solving nonconvex optimization problems. It is required that the designed neural networks should converge, either locally or globally, to exact local or global solutions of the problems, which is different from the principle of simple penalty-based methods. (Here, the words "locally" and "globally" characterize the convergence behavior of recurrent neural networks while the words "local" and "global" characterize the inherent property of a solution to the problem; they are in general uncorrelated with each other.) First, a special class of nonconvex optimization problems, pseudoconvex optimization problems, were considered. Because any local solution of such a problem is global as well, it is possible to design neural networks which can globally converge to the global solutions. We have revealed that two existing neural networks, called TLPNN and PNN, are capable of accomplishing this task with appropriate conditions.

Second, general nonconvex optimization problems were discussed from the viewpoint of designing neural networks to search for their Karush-Kuhn-Tucker (KKT) points especially the corresponding local solutions. The extended projection neural network (EPNN), originated from solving convex optimization problems in the literature, was studied in this context. The local convergence of the EPNN to KKT points was studied and a set of sufficient conditions was given. Since in many cases these conditions fail to exist, an effective method, augmented Lagrangian techniques were proposed to conquer this difficulty. Two augmented Lagrangian function methods were investigated: one is the partial p-power Lagrangian function existing in the literature and the other is new. Two prominent augmented Lagrange networks were then obtained. For both neural networks, a nice property is that their equilibria are in exact correspondence with the KKT points. Another nice property lies in that by choosing an appropriate control parameter each neural network can be made asymptotically stable at those KKT points associated with local optima under some standard assumptions in the optimization context, although locally. This can be regarded as a meaningful progress for designing neural networks for completely solving nonconvex optimization problems.

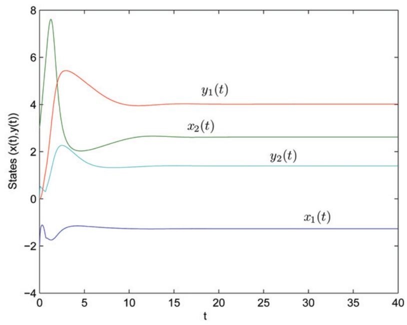
During discussion, numerical examples were provided to illustrate as well as validate the theoretical results.



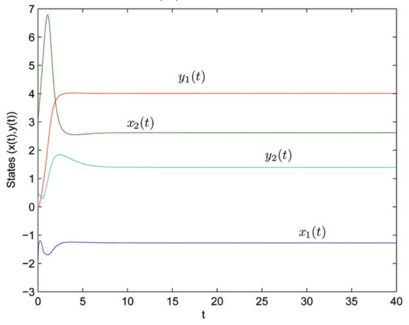
(a) $c = 0$



(b) $c = 0.1$



(c) $c = 0.2$



(d) $c = 0.5$

Figure 12. Transient behavior the neural network (16) with different values of c in Example 5. Reprint of Fig. 2 in (Hu & Wang, 2007a).

5.2 Future directions

If we classify the nonconvex optimization problems into two categories *Type-I* and *Type-II*, referring to those whose local optima are also global optima and those otherwise, respectively, our primary goal at current stage is to devise some neural networks that can converge globally to the solutions of Type-I problems and can converge globally to local optima sets of Type-II problems. Towards this goal, there is still a long way to walk. Related to the contents of this chapter, some meaningful future directions are as follows. Notice that in Section 4.1 it was shown that the EPNN is locally asymptotically stable at a KKT point (x^*, y^*) (corresponding to a local solution) of the Type-II problem provided that the Hessian of the Lagrange function $\nabla L_{xx}L(x, y) > 0$ at this point. The main idea of the proof of this result (see Hu, 2007, Chapter 9.2) is to construct a neighborhood $\Omega_c(x^*, y^*)$ around this KKT point in which $\nabla L_{xx}L(x, y) > 0$. Then the trajectory originated in it will converge to the KKT point. Hence, for the size of the neighborhood, the larger the better. This condition is actually somewhat too strong. For ensuring the local convergence, it is required $\nabla L_{xx}L(x, y) > 0$ on the trajectory of the network in $\Omega_c(x^*, y^*)$ only, not necessarily on the entire $\Omega_c(x^*, y^*)$. This new condition can be utilized to state global convergence of the EPNN to a KKT point, while the original one cannot. The reason is that it is impossible for a nonconvex optimization problem that $\nabla L_{xx}L(x, y) > 0$ over the entire space, but it is possible that this inequality holds over a particular trajectory. This is one of the main ideas of a most recent article (Xia et al., 2007). Obviously, this idea can be also applied to the two augmented Lagrange networks discussed in the chapter.

For solving optimization problems with general constraints, the EPNN and its variants play the dominant roles in the community. Recently, a notable progress has been made in (Xia & Feng, 2007) where a much different model was proposed for solving convex optimization problems. It deserves further investigation from the viewpoint of nonconvex optimization.

6. Acknowledgement

The work was supported by the National Natural Science Foundation of China under Grant 60621062 and by the National Key Foundation R&D Project under Grants 2003CB317007, 2004CB318108 and 2007CB311003. It has been partially presented in (Hu, 2007; Hu & Wang, 2006b; Hu & Wang, 2006a; Hu & Wang, 2007a).

7. References

Avriel, M., Diewert, W. E., Schaible, S. and Zang, I. (1988). *Generalized concav ity*, Mathematical Concepts and Methods in Science and Engineering, Plenum Publishing Corporation, New York.

Bazaraa, M. S., Sherali, H. D. and Shetty, C. M. (1993). *Nonlinear Programming: Theory and Algorithms*, 2nd edn, Wiley, New York.

Cambini, A., Crouzeix, J. P. and Martein, L. (2002). On the pseudoconvexity of a quadratic fractional function, *Optimization* 51(4): 677-687.

Gao, X. (2004). A novel neural network for nonlinear convex programming, *IEEE Trans. Neural Netw.* 15(3): 613-621.

Gao, X., Liao, L. and Qi, L. (2005). A novel neural network for variational inequalities with linear and nonlinear constraints, *IEEE Trans. Neural Netw.* 16(6): 1305- 1317.

Hadjisavvas, N. and Schaible, S. (1993). On strong pseudomonotonicity and (semi)strict quasimonotonicity, *Journal of Optimization Theory and Applications* 79(1): 139-155.

Hopfield, J. J. and Tank, D. W. (1986). Computing with neural circuits: a model, *Scienc* 233(4764): 625-633.

Hu, X. (2007). *Solving Variational Inequalities and Related Problems Using Recurrent Neural Networks*, PhD thesis, The Chinese University of Hong Kong, Hong Kong, China.

Hu, X. andWang, J. (2006a). Global stability of a recurrent neural network for solving pseudomonotone variational inequalities, *Proc. IEEE International Symposium on Circuits and Systems*, Island of Kos, Greece, pp. 755-758.

Hu, X. and Wang, J. (2006b). Solving pseudomonotone variational inequalities and pseudoconvex optimization problems using the projection neural network, *IEEE Trans. Neural Netw.* 17(6): 1487-1499.

Hu, X. and Wang, J. (2007a). Convergence of a recurrent neural network for nonconvex optimization based on an augmented lagrangian function, *Proc. 4th International Symposium on Neural Networks*, Vol. 4493 of *Lecture Notes in Computer Science*, Nanjing, China, pp. 194-203.

Hu, X. and Wang, J. (2007b). Design of general projection neural networks for solving monotone linear variational inequalities and linear and quadratic optimization problems, *IEEE Trans. Syst., Man, Cybern. B* 37(5): 1414-1421.

Hu, X. and Wang, J. (2007c). Solving generally constrained generalized linear variational inequalities using the general projection neural networks, *IEEE Trans. Neural Netw.* 18(6): 1697-1708.

Hu, X. and Wang, J. (2008). An improved dual neural network for solving a class of quadratic programming problems and its k-winners-take-all application, IEEE Trans. Neural Netw. accepted.

Huang, Y. (2005). Lagrange-type neural networks for nonlinear programming problems with inequality constraints, *Proc. 44th IEEE Conference on Decision and Control and the European Control Conference*, Seville, Spain, pp. 4129-4133.

Karamardian, S. and Schaible, S. (1990). Seven kinds of monotone maps, *Journal of Optimization Theory and Applications* 66(1): 37-46.

Kennedy, M. P. and Chua, L. O. (1988). Neural networks for nonlinear programming, *IEEE Trans. Circuits Syst.* 35: 554-562.

Kinderlehrer, D. and Stampcchia, G. (1980). *An Introduction to Variational Inequalities and Their Applications*, Academic, New York.

Li, D. and Sun, X. L. (2000). Local convexification of the lagrangian function in nonconvex optimization, *Journal of Optimization Theory and Applications* 104(1): 109-120.

Rodríguez-Vázquez, A., Domínguez-Castro, R., Rueda, A., Huertas, J. L. and Sánchez-Sinencio, E. (1990). Nonlinear switched-capacitor neural networks for optimization problems, *IEEE Trans. Circuits Syst.* 37: 384-397.

Sun, X. L., Li, D. and Mckinnon, K. I. M. (2005). On saddle points of augmented lagrangians for constrained nonconvex optimization, *SIAM J. Optim.* 15(4): 1128- 1146.

Tank, D. W. and Hopfield, J. J. (1986). Simple neural optimization networks: an A/D converter, signal decision circuit, and a linear programming circuit, *IEEE Trans. Circuits Syst.* 33(5): 533-541.

Wang, J. (1994). A deterministic annealing neural network for convex programming, *Neural Networks* 7(4): 629-641.

Xia, Y. (2004). An extended projection neural network for constrained optimization, *Neural Computation* 16(4): 863-883.

Xia, Y. and Feng, G. (2005). On convergence conditions of an extended projection neural network, *Neural Computation* 17(3): 515-525.

Xia, Y. and Feng, G. (2007). A new neural network for solving nonlinear projection equations, *Neural Networks* 20: 577-589.

Xia, Y., Feng, G. and Kamel, M. (2007). Development and analysis of a neural dynamical approach to nonlinear programming problems, *IEEE Trans. Automatic Control* 52(11): 2154-2159.

Xia, Y. and Wang, J. (1998). A general methodology for designing globally convergent optimization neural networks, *IEEE Trans. Neural Netw.* 9(6): 1331-1343.

Xia, Y. and Wang, J. (2000). Global exponential stability of recurrent neural networks for solving optimization and related problems, *IEEE Trans. Neural Netw.* 11(4): 1017-1022.

Xia, Y. and Wang, J. (2004). A recurrent neural network for nonlinear convex optimization subject to nonlinear inequality constraints, *IEEE Trans. Circuits Syst. I* 51(7): 1385-1394.

Zhang, S. and Constantinides, A. G. (1992). Lagrange programming neural networks, *IEEE Trans. Circuits Syst. II* 39: 441-452.

An Improved Extremum Seeking Algorithm Based on the Chaotic Annealing Recurrent Neural Network and Its Application

Yun-an Hu, Bin Zuo and Jing Li
Department of Control Engineering, Naval Aeronautical and Astronautical University
P. R. China

1. Introduction

Extremum seeking problem deals with the problem of minimizing or maximizing a plant over a set of decision variables[1]. Extremum seeking problems represent a class of widespread optimization problems arising in diverse design and planning contexts. Many large-scale and real-time applications, such as traffic routing and bioreactor systems, require solving large-scale extremum seeking problem in real time. In order to solve this class of extremum seeking problems, a novel extremum seeking algorithm was proposed in the 1950's. Early work on performance improvement by extremum seeking can be found in Tsien. In the 1950s and the 1960s, Extremum seeking algorithm was considered as an adaptive control method[2]. Until 1990s sliding mode control for extremum seeking has not been utilized successfully[3]. Subsequently, a method of adding compensator dynamics in ESA was proposed by Krstic, which improved the stability of the controlled extremum control system[4]. Although those methods improved tremendously the performance of ESA, the "chatter" problem of the output and the switching of the control law and incapability of escaping from the local minima limit the application of ESA.
In order to solve those problems in the conventional ESA and improve the capability of global searching, an improved chaotic annealing recurrent neural network (CARNN) is proposed in the paper. The method of introducing a chaotic annealing recurrent neural network into ESA is proposed in the paper. First, an extremum seeking problem is converted into the process of seeking the global extreme point of the plant where the slope of cost function is zero. Second, an improved CARNN is constructed; and then we can apply the CARNN to finding the global extreme point and stabilizing the plant at that point. The CARNN proposed in the paper doesn't make use of search signals such as sinusoidal periodic signals, so the method can solve the "chatter" problem of the output and the switching of the control law in the general ESA and improve the dynamic performance of extremum seeking system. At the same time, CARNN utilizes the randomicity and the property of global searching of chaos system to improve the capability of global searching of the system[5-6], During the process of optimization, chaotic annealing is realized by decaying the amplitude of the chaos noise and the accepting probability continuously. Adjusting the probability of acceptance could influence the rate of convergence. The process of optimization was divided into two phases: the coarse search based on chaos and the

elaborate search based on RNN. At last, CARNN will stabilize the system to the global extreme point, which is validated by simulating a simplified UAV tight formation flight model and a typical Schaffer Function. At the same time, the stability analysis of ESA can be simplified by the proposed method.

2. Annealing recurrent neural network description

2.1 Problem formulation

Consider a general nonlinear system:

$$\begin{aligned} \dot{x} &= f\left(x(t),u(t)\right) \\ y &= F\left(x(t)\right) \end{aligned} \tag{1}$$

Where $x \in R^n, u \in R^m$ and $y \in R$ are the states, the system inputs and the system output, respectively. $F(x)$ is also defined as the cost function of the system (1). $f(x,u)$ and $F(x)$ are smooth functions. If the nonlinear system (1) is an extremum seeking system, then it must satisfy the following assumptions.

Assumption 1: There is a smooth control law[7]:

$$u(t) = \alpha\left(x(t),\theta\right) \tag{2}$$

to stabilize the nonlinear system(1), where $\theta = \left[\theta_1,\theta_2,\cdots,\theta_i,\cdots,\theta_p\right]^T \left(i \in [1,2,\cdots,p]\right)$ is a parameter vector of p dimension which determines a unique equilibrium vector.

With the control law (2), the closed-loop system of the nonlinear system (1) can be written as:

$$\dot{x} = f\left(x,\alpha(x,\theta)\right)$$

Assumption 2: There is a smooth function $x_e : R^p \rightarrow R^n$ such that:

$$f\left(x,\alpha(x,\theta)\right) = 0 \leftrightarrow x = x_e(\theta)$$

Assumption 3: The static performance map at the equilibrium point $x_e(\theta)$ from θ to y represented by:

$$y = F\left(x_e(\theta)\right) = F(\theta) \tag{3}$$

is smooth and has a unique global maximum or minimum vector $\theta^* \in R^p, \theta^* = \left[\theta_1^*,\theta_2^*,\cdots,\theta_p^*\right]^T$ such that:

$$\frac{\partial F\left(\theta^*\right)}{\partial \theta_i} = 0,\left(i = 1,2,\cdots,p\right)$$

at the same time $\dfrac{\partial^2 F\left(\theta^*\right)}{\partial \theta_i^2} < 0$ or $\dfrac{\partial^2 F\left(\theta^*\right)}{\partial \theta_i^2} > 0$

Differentiating (3) with respect to time yields the relation between $\dot{\theta}$ and $\dot{y}(t)$.

$$\partial(\theta(t))\dot{\theta}(t)=\dot{y}(t) \tag{4}$$

Where $\partial(\theta(t))=\left[\frac{\partial F(\theta)}{\partial\theta_1},\frac{\partial F(\theta)}{\partial\theta_2},\cdots,\frac{\partial F(\theta)}{\partial\theta_p}\right]^T$ and $\dot{\theta}(t)=\left[\dot{\theta}_1,\dot{\theta}_2,\cdots,\dot{\theta}_p\right]^T$.

Based on Assumption 3, once the seeking vector θ of the extremum seeking system (1) converges to the global extreme vector θ^*, then $\left|\partial(\theta)\right|=\left[\left|\frac{\partial F(\theta)}{\partial\theta_1}\right|,\left|\frac{\partial F(\theta)}{\partial\theta_2}\right|,\cdots,\left|\frac{\partial F(\theta)}{\partial\theta_p}\right|\right]^T$ must also converge to zero. A CARNN is introduced into ESA in order to minimize $\left|\partial(\theta)\right|$ in finite time. Certainly the system (1) is subjected to (4).
Then, the extremum seeking problem can be written as follows.

$$\theta^*=arg\,min_{\theta\in\hat{\theta}}\left|\partial(\theta)\right|$$

$$\text{Subject to: } \partial^T(\theta(t))\dot{\theta}(t)=\dot{y}(t) \tag{5}$$

The optimization (5) is then equivalent to

$$\text{Minimize: } f_1(\upsilon)=c^T\upsilon$$

$$\text{Subject to: } p_1(\upsilon)=A\upsilon-b=0 \tag{6}$$

Where, $\partial^T(\theta)$ denotes the transpose of $\partial(\theta)$. $\upsilon=\left[\partial(\theta)\quad \left|\partial(\theta)\right|\quad \dot{\theta}(t)\right]^T$, $c=\left[0_{1\times p}\quad 1_{1\times p}\quad 0_{1\times p}\right]^T$, $A=\begin{bmatrix}1_{1\times p} & -sign\left(\partial^T(\theta)\right) & 0_{1\times p}\\ \dot{\theta}^T(t) & 0_{1\times p} & 0_{1\times p}\\ 0_{1\times p} & 0_{1\times p} & \partial^T(\theta)\end{bmatrix}$, $b=\begin{bmatrix}0\\ \dot{y}(t)\\ \dot{y}(t)\end{bmatrix}$, and $\text{sign}(x)=\begin{cases}1 & x>0\\ 0 & x=0\\ -1 & x<0\end{cases}$.

By the dual theory, the dual program corresponding to the program (6) is

$$\text{Maximize: } f_2(\omega)=b^T\omega$$

$$\text{Subject to: } p_2(\omega)=A^T\omega-c=0 \tag{7}$$

Where, ω denotes the dual vector of υ, $\omega^T=\left[\omega_1\quad \omega_2\quad \omega_3\right]_{1\times 3}$.
Therefore, an extremum seeking problem is converted into the programs defined in (6) and (7).

2.2 Annealing Recurrent Neural Network (ARNN) design

In view of the primal and dual programs (6) and (7), define the following energy function:

$$E(\upsilon,\omega)=\frac{1}{2}T(t)\left(f_1(\upsilon)-f_2(\omega)\right)^2+\frac{1}{2}\left\|p_1(\upsilon)\right\|^2+\frac{1}{2}\left\|p_2(\omega)\right\|^2 \tag{8}$$

Clearly, the energy function (8) is convex and continuously differentiable. The first term in (8) is the squared of the difference between two objective functions of the programs (6) and (7), respectively. The second and the third terms are for the equality constraints of (6) and (7). $T(t)$ denotes a time-varying annealing parameter.

With the energy function defined in (8), the dynamics for CARNN solving (6) and (7) can be defined by the negative gradient of the energy function as follows.

$$\frac{d\sigma}{dt} = -\mu \nabla E(\sigma) \tag{9}$$

Where, $\sigma = (\upsilon, \omega)^T$, $\nabla E(\sigma)$ is the gradient of the energy function $E(\sigma)$ defined in (9), and μ is a positive scalar constant, which is used to scale the convergence rate of annealing recurrent neural network.

The dynamical equation (9) of annealing recurrent neural network can be expressed as:

$$\frac{du_1}{dt} = -\mu \frac{\partial E(\upsilon,\omega)}{\partial \upsilon} = -\mu \left[T(t) \cdot \frac{\partial f_1(\upsilon)}{\partial \upsilon} \cdot \left(f_1(\upsilon) - f_2(\omega)\right) + \frac{\partial p_1(\upsilon)}{\partial \upsilon} \cdot p_1(\upsilon) \right]$$

$$= -\mu \left[T(t) c \left(c^T \upsilon - b^T \omega \right) + A^T (A\upsilon - b) \right] \tag{10}$$

$$\frac{du_2}{dt} = -\mu \frac{\partial E(\upsilon,\omega)}{\partial \omega} = -\mu \left[-T(t) \cdot \frac{\partial f_2(\omega)}{\partial \omega} \cdot \left(f_1(\upsilon) - f_2(\omega)\right) + \frac{\partial p_2(\omega)}{\partial \omega} \cdot p_2(\omega) \right]$$

$$= -\mu \left[-T(t) b \left(c^T \upsilon - b^T \omega \right) + A\left(A^T \omega - c \right) \right] \tag{11}$$

$$\upsilon = q(u_1) \tag{12}$$

$$\omega = q(u_2) \tag{13}$$

Where, $q(\ )$ is a sigmoid activation function, $\upsilon = q(u_1) = \dfrac{b_1 - a_1}{1 + e^{-u_1/\varepsilon_1}} + a_1$ and $\omega = q(u_2) = \dfrac{b_2 - a_2}{1 + e^{-u_2/\varepsilon_2}} + a_2$. a_1 and b_1 denote the upper bound and the below bound of υ . a_2 and b_2 denote the upper bound and the below bound of ω . $\varepsilon_1 > 0$ and $\varepsilon_2 > 0$.

The annealing recurrent neural network is described as the equations (10) ~ (13), which are determined by the number of decision variables such as (υ, ω) , (u_1, u_2) is the column vector of instantaneous net inputs to neurons, (υ, ω) is the column output vector of neurons. The lateral connection weight matrix is defined as $\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} = \begin{bmatrix} -\mu\left(T(t)cc^T + A^T A\right) & \mu T(t) cb^T \\ \mu T(t) bc^T & -\mu\left(T(t)bb^T + AA^T\right) \end{bmatrix}$, the biasing threshold vector of the neurons is defined as $\begin{bmatrix} \vartheta_1 \\ \vartheta_2 \end{bmatrix} = \begin{bmatrix} \mu A^T b \\ \mu A c \end{bmatrix}$.

3. Convergence analysis

In this section, analytical results on the stability of the proposed annealing recurrent neural network and feasibility and optimality of the steady-state solutions to the programs described in (6) and (7) are presented.

3.1 Solution feasibility

Theorem 1. Assume that the Jacobian matrices $J\left[q\left(u_1\right)\right]$ and $J\left[q\left(u_2\right)\right]$ exist and are positive semidefinite. If the temperature parameter $T\left(t\right)$ is nonnegative, strictly monotone decreasing for $t \geq 0$, and approaches zero as time approaches infinity, then the annealing recurrent neural network (10) ~ (13) is asymptotically stable.

Proof: Consider the following Lyapunov function.

$$L = E\left(\upsilon,\omega\right) = \frac{1}{2}T\left(t\right)\left(f_1\left(\upsilon\right) - f_2\left(\omega\right)\right)^2 + \frac{1}{2}\left\|p_1\left(\upsilon\right)\right\|^2 + \frac{1}{2}\left\|p_2\left(\omega\right)\right\|^2 \tag{14}$$

Apparently, $L\left(t\right) > 0$. The difference of L along time trajectory of (14) is as follows:

$$\begin{aligned}\frac{dL}{dt} &= T\left(t\right)\left[\frac{\partial f_1\left(\upsilon\right)}{\partial \upsilon}\cdot\frac{d\upsilon}{dt} - \frac{\partial f_2\left(\omega\right)}{\partial \omega}\cdot\frac{d\omega}{dt}\right]\left(f_1\left(\upsilon\right) - f_2\left(\omega\right)\right) + \frac{\partial p_1\left(\upsilon\right)}{\partial \upsilon}\cdot\frac{d\upsilon}{dt}\cdot p_1\left(\upsilon\right) \\ &\quad + \frac{\partial p_2\left(\omega\right)}{\partial \omega}\cdot\frac{d\omega}{dt}\cdot p_2\left(\omega\right) + \frac{1}{2}\frac{dT\left(t\right)}{dt}\left(f_1\left(\upsilon\right) - f_2\left(\omega\right)\right)^2 \\ &= \left[T\left(t\right)\cdot\frac{\partial f_1\left(\upsilon\right)}{\partial \upsilon}\cdot\left(f_1\left(\upsilon\right) - f_2\left(\omega\right)\right) + \frac{\partial p_1\left(\upsilon\right)}{\partial \upsilon}\cdot p_1\left(\upsilon\right)\right]\cdot\frac{d\upsilon}{dt} \\ &\quad + \left[-T\left(t\right)\cdot\frac{\partial f_2\left(\omega\right)}{\partial \omega}\cdot\left(f_1\left(\upsilon\right) - f_2\left(\omega\right)\right) + \frac{\partial p_2\left(\omega\right)}{\partial \omega}\cdot p_2\left(\omega\right)\right]\cdot\frac{d\omega}{dt} \\ &\quad + \frac{1}{2}\frac{dT\left(t\right)}{dt}\left(f_1\left(\upsilon\right) - f_2\left(\omega\right)\right)^2\end{aligned} \tag{15}$$

According to the equations (10) and (11), and the following equations

$$\frac{d\upsilon}{dt} = J\left[q\left(u_1\right)\right]\cdot\frac{du_1}{dt} \tag{16}$$

$$\frac{d\omega}{dt} = J\left[q\left(u_2\right)\right]\cdot\frac{du_2}{dt} \tag{17}$$

We can have:

$$\frac{dL}{dt} = -\frac{1}{\mu}\cdot\frac{du_1}{dt}\cdot J\left[q\left(u_1\right)\right]\cdot\frac{du_1}{dt} - \frac{1}{\mu}\cdot\frac{du_2}{dt}\cdot J\left[q\left(u_2\right)\right]\cdot\frac{du_2}{dt} + \frac{1}{2}\frac{dT\left(t\right)}{dt}\left(f_1\left(\upsilon\right) - f_2\left(\omega\right)\right)^2 \tag{18}$$

We know that the Jacobian matrices of $J\left[q(u_1)\right]$ and $J\left[q(u_2)\right]$ both exist and are positive semidefinite and μ is a positive scalar constant. If the time-varying annealing parameter $T(t)$ is nonnegative, strictly monotone decreasing for $t \geq 0$, and approaches zero as time approaches infinity, then dL/dt is negative definite. Because $T(t)$ represents the annealing effect, the simple examples of $T(t)$ can described as follows.

$$T(t) = \beta\alpha^{-\eta t} \tag{19}$$

$$T(t) = \beta(1+t)^{-\eta} \tag{20}$$

Where $\alpha > 1$, $\beta > 0$ and $\eta > 0$ are constant parameters. Parameters β and η can be used to scale the annealing parameter.

Because $L(t)$ is positive definite and radially unbounded, and dL/dt is negative definite. According to the Lyapunov's theorem, the designed annealing recurrent neural network is asymptotically stable.

Theorem 2. Assume that the Jacobian matrices $J\left[q(u_1)\right]$ and $J\left[q(u_2)\right]$ exist and are positive semidefinite. If $T(t) \geq 0$, $dT(t)/dt < 0$ and $\lim_{t\to\infty} T(t) = 0$, then the steady state of the annealing neural network represents a feasible solution to the programs described in equations (6) and (7).

Proof: The proof of Theorem 1 shows that the energy function $E(\upsilon,\omega)$ is positive definite and strictly monotone decreasing with respect to time t, which implies $\lim_{t\to\infty} E(\upsilon,\omega,T(t)) = 0$. Because $\lim_{t\to\infty} T(t) = 0$, then we have

$$\lim_{t\to\infty} E(\upsilon,\omega,T(t)) = \lim_{t\to\infty}\left(\frac{1}{2}T(t)\left(f_1(\upsilon)-f_2(\omega)\right)^2 + \frac{1}{2}\left\|p_1(\upsilon)\right\|^2 + \frac{1}{2}\left\|p_2(\omega)\right\|^2\right) \tag{21}$$

$$= \lim_{t\to\infty}\left(\frac{1}{2}\left\|p_1(\upsilon(t))\right\|^2 + \frac{1}{2}\left\|p_2(\omega(t))\right\|^2\right) = 0 \tag{22}$$

Because $p_1(\upsilon(t))$ and $p_2(\omega(t))$ are continuous, $\lim_{t\to\infty}\left(\frac{1}{2}\left\|p_1(\upsilon(t))\right\|^2 + \frac{1}{2}\left\|p_2(\omega(t))\right\|^2\right) = \frac{1}{2}\left\|p_1\left(\lim_{t\to\infty}\upsilon(t)\right)\right\|^2$ $+\frac{1}{2}\left\|p_2\left(\lim_{t\to\infty}\omega(t)\right)\right\|^2 = \frac{1}{2}\left\|p_1(\bar{\upsilon})\right\|^2 + \frac{1}{2}\left\|p_2(\bar{\omega})\right\|^2 = 0$, so we have $p_1(\bar{\upsilon}) = 0$ and $p_2(\bar{\omega}) = 0$, where $\bar{\upsilon}$ and $\bar{\omega}$ are the stable solutions of υ and ω.

3.2 Solution optimality

Firstly, Let $F_1(\upsilon) = \begin{bmatrix}(f_1(\upsilon))\\(f_1(\upsilon))\\(f_1(\upsilon))\end{bmatrix} = I_{3\times1}\cdot(f_1(\upsilon))$ and $F_2(\omega) = \begin{bmatrix}(f_2(\omega))\\(f_2(\omega))\\(f_2(\omega))\end{bmatrix} = I_{3\times1}\cdot(f_2(\omega))$ be the augmented vector.

Theorem 3. Assume that the Jacobian matrices $J\left[q\left(u_1\right)\right]\neq 0$ and $J\left[q\left(u_2\right)\right]\neq 0$ and are positive semidefinite, $\forall t\geq 0$, and $\nabla\left(f_1(\upsilon)\right)\neq 0$ and $\nabla\left(f_2(\omega)\right)\neq 0$. If $dT(t)/dt<0$, $\lim_{t\to\infty}T(t)=0$ and

$$T(t)\geq\max\left\{0,\frac{\left(\nabla p_1\left[\upsilon(t)\right]^T J\left[q(u_1)\right]\frac{\partial p_1(\upsilon)}{\partial\upsilon}p_1(\upsilon)-\nabla F_1\left[\upsilon(t)\right]^T J\left[q(u_1)\right]\frac{\partial p_1(\upsilon)}{\partial\upsilon}p_1(\upsilon)\right)}{\left(\nabla F_1\left[\upsilon(t)\right]^T J\left[q(u_1)\right]\frac{\partial f_1(\upsilon)}{\partial\upsilon}\left(f_1(\upsilon)-f_2(\omega)\right)-\nabla p_1\left[\upsilon(t)\right]^T J\left[q(u_1)\right]\frac{\partial f_1(\upsilon)}{\partial\upsilon}\left(f_1(\upsilon)-f_2(\omega)\right)\right)},\right.$$

$$\left.\frac{\left(\nabla p_2\left[\omega(t)\right]^T J\left[q(u_2)\right]\frac{\partial p_2(\omega)}{\partial\omega}p_2(\omega)-\nabla F_2\left[\omega(t)\right]^T J\left[q(u_2)\right]\frac{\partial p_2(\omega)}{\partial\omega}p_2(\omega)\right)}{\left(\nabla F_2\left[\omega(t)\right]^T J\left[q(u_2)\right]\frac{\partial f_2(\omega)}{\partial\omega}\left(f_1(\upsilon)-f_2(\omega)\right)-\nabla p_2\left[\omega(t)\right]^T J\left[q(u_2)\right]\frac{\partial f_2(\omega)}{\partial\omega}\left(f_1(\upsilon)-f_2(\omega)\right)\right)}\right\}\quad(23)$$

then the steady states $\bar{\upsilon}$ and $\bar{\omega}$ of the annealing neural network represents the optimal solutions υ^* and ω^* to the programs described in equations (6) and (7).

Proof: According to the equation (23), we know

$$T(t)\geq\frac{\left(\nabla p_1\left[\upsilon(t)\right]^T J\left[q(u_1)\right]\frac{\partial p_1(\upsilon)}{\partial\upsilon}p_1(\upsilon)-\nabla F_1\left[\upsilon(t)\right]^T J\left[q(u_1)\right]\frac{\partial p_1(\upsilon)}{\partial\upsilon}p_1(\upsilon)\right)}{\left(\nabla F_1\left[\upsilon(t)\right]^T J\left[q(u_1)\right]\frac{\partial f_1(\upsilon)}{\partial\upsilon}\left(f_1(\upsilon)-f_2(\omega)\right)-\nabla p_1\left[\upsilon(t)\right]^T J\left[q(u_1)\right]\frac{\partial f_1(\upsilon)}{\partial\upsilon}\left(f_1(\upsilon)-f_2(\omega)\right)\right)}$$

The above equation implies

$$\mu T(t)\left[\nabla F_1\left[\upsilon(t)\right]^T J\left[q(u_1)\right]-\nabla p_1\left[\upsilon(t)\right]^T J\left[q(u_1)\right]\right]\frac{\partial f_1(\upsilon)}{\partial\upsilon}\left(f_1(\upsilon)-f_2(\omega)\right)$$

$$\geq\mu\nabla p_1\left[\upsilon(t)\right]^T J\left[q(u_1)\right]\frac{\partial p_1(\upsilon)}{\partial\upsilon}p_1(\upsilon)-\mu\nabla F_1\left[\upsilon(t)\right]^T J\left[q(u_1)\right]\frac{\partial p_1(\upsilon)}{\partial\upsilon}p_1(\upsilon)$$

Rearranging the above inequality, we have

$$\mu T(t)\nabla F_1\left[\upsilon(t)\right]^T J\left[q(u_1)\right]\frac{\partial f_1(\upsilon)}{\partial\upsilon}\left(f_1(\upsilon)-f_2(\omega)\right)+\mu\nabla F_1\left[\upsilon(t)\right]^T J\left[q(u_1)\right]\frac{\partial p_1(\upsilon)}{\partial\upsilon}p_1(\upsilon)$$

$$-\mu T(t)\nabla p_1\left[\upsilon(t)\right]^T J\left[q(u_1)\right]\frac{\partial f_1(\upsilon)}{\partial\upsilon}\left(f_1(\upsilon)-f_2(\omega)\right)-\mu\nabla p_1\left[\upsilon(t)\right]^T J\left[q(u_1)\right]\frac{\partial p_1(\upsilon)}{\partial\upsilon}p_1(\upsilon)\geq 0$$

That is also

$$\mu\nabla F_1\left[\upsilon(t)\right]^T\cdot J\left[q(u_1)\right]\cdot\left[\left(T(t)\cdot\frac{\partial f_1(\upsilon)}{\partial\upsilon}\cdot\left(f_1(\upsilon)-f_2(\omega)\right)+\frac{\partial p_1(\upsilon)}{\partial\upsilon}\cdot p_1(\upsilon)\right)\right]$$

$$-\mu\nabla p_1\left[\upsilon(t)\right]^T\cdot J\left[q(u_1)\right]\cdot\left[\left(T(t)\cdot\frac{\partial f_1(\upsilon)}{\partial\upsilon}\cdot\left(f_1(\upsilon)-f_2(\omega)\right)+\frac{\partial p_1(\upsilon)}{\partial\upsilon}\cdot p_1(\upsilon)\right)\right]\geq 0$$

Substituting equations (10), (11), (16) and (17) into the above inequality, we have

$$\nabla F_1\left[\upsilon(t)\right]^T\cdot J\left[q(u_1)\right]\cdot\frac{du_1(t)}{dt}-\nabla p_1\left[\upsilon(t)\right]^T\cdot J\left[q(u_1)\right]\cdot\frac{du_1(t)}{dt}=$$

$$\nabla F_1\left[\upsilon(t)\right]^T\cdot\frac{d\upsilon(t)}{dt}-\nabla p_1\left[\upsilon(t)\right]^T\cdot\frac{d\upsilon(t)}{dt}=\frac{dF_1\left(\upsilon(t)\right)}{dt}-\frac{dp_1\left(\upsilon(t)\right)}{dt}\leq 0$$

Therefore, we have $dF_1\left(\upsilon(t)\right)/dt\leq dp_1\left(\upsilon(t)\right)/dt$, which implies $F_1\left(\upsilon(t'')\right)-F_1\left(\upsilon(t')\right)\leq p_1\left(\upsilon(t'')\right)-p_1\left(\upsilon(t')\right)$ for any $t'\leq t''$. Let t^* be the time associated with an optimal solution υ^*. We have $F_1\left(\upsilon(\infty)\right)-F_1\left(\upsilon(t^*)\right)\leq p_1\left(\upsilon(\infty)\right)-p_1\left(\upsilon(t^*)\right)$; that is $F_1(\bar{\upsilon})-F_1\left(\upsilon^*\right)\leq p_1(\bar{\upsilon})-p_1\left(\upsilon^*\right)$. Because $p_1(\bar{\upsilon})=p_1\left(\upsilon^*\right)=0$, $F_1(\bar{\upsilon})\leq F_1\left(\upsilon^*\right)$. At last, we have $f_1(\bar{\upsilon})\leq f_1\left(\upsilon^*\right)$. Also, because $\upsilon^*=\arg\min_{\upsilon\in\hat{V}}f_1(\upsilon)$, $f_1(\bar{\upsilon})\geq f_1\left(\upsilon^*\right)$ by definition of υ^*. Consequently, $f_1(\bar{\upsilon})=f_1\left(\upsilon^*\right)=\min_{\upsilon\in\hat{V}}f_1(\upsilon)$, where $\hat{V}$ denotes the feasible region of the optimal solution υ^*.

Next, according to the equation (23), we also know

$$T(t)\geq\frac{\left(\nabla p_2\left[\omega(t)\right]^T J\left[q(u_2)\right]\frac{\partial p_2(\omega)}{\partial\omega}p_2(\omega)-\nabla F_2\left[\omega(t)\right]^T J\left[q(u_2)\right]\frac{\partial p_2(\omega)}{\partial\omega}p_2(\omega)\right)}{\left(\nabla F_2\left[\omega(t)\right]^T J\left[q(u_2)\right]\frac{\partial f_2(\omega)}{\partial\omega}\left(f_1(\upsilon)-f_2(\omega)\right)-\nabla p_2\left[\omega(t)\right]^T J\left[q(u_2)\right]\frac{\partial f_2(\omega)}{\partial\omega}\left(f_1(\upsilon)-f_2(\omega)\right)\right)}$$

By the same reasoning, we know $\dfrac{dF_2\left(\omega(t)\right)}{dt}\geq\dfrac{dp_2\left(\omega(t)\right)}{dt}$, which implies $F_2\left(\omega(t'')\right)-F_2\left(\omega(t')\right)\geq p_2\left(\omega(t'')\right)-p_2\left(\omega(t')\right)$ for any $t'\leq t''$. Let t^* be the time associated with an optimal solution ω^*. We have $F_2\left(\omega(\infty)\right)-F_2\left(\omega(t^*)\right)\geq p_2\left(\omega(\infty)\right)-p_2\left(\omega(t^*)\right)$; that is $F_2(\bar{\omega})-F_2\left(\omega^*\right)\geq p_2(\bar{\omega})-p_2\left(\omega^*\right)$. Because $p_2(\bar{\omega})=p_2\left(\omega^*\right)=0$, $F_2(\bar{\omega})\geq F_2\left(\omega^*\right)$. At last, we have $f_2(\bar{\omega})\geq f_2\left(\omega^*\right)$. Also, because $\omega^*=\arg\max_{\omega\in\hat{U}}f_2(\omega)$, $f_2\left(\omega^*\right)\geq f_2(\bar{\omega})$ by definition of ω^*. Consequently, $f_2(\bar{\omega})=f_2\left(\omega^*\right)=\max_{\omega\in\hat{U}}f_2(\omega)$, where $\hat{U}$ denotes the feasible region of the optimal solution ω^*.

4. A chaotic annealing recurrent neural network description

In order to improve the global searching performance of the designed annealing recurrent neural network, we introduce chaotic factors into the designed neural network. Therefore, the structure of a chaotic annealing recurrent neural network is described as follows.

$$\frac{du_1}{dt}=\begin{cases}-\mu\left[T(t)\cdot\frac{\partial f_1(\upsilon)}{\partial\upsilon}\cdot\left(f_1(\upsilon)-f_2(\omega)\right)+\frac{\partial p_1(\upsilon)}{\partial\upsilon}\cdot p_1(\upsilon)\right]+\eta_1\left(\chi_1(b_1-a_1)+a_1\right) & random<P_1(t)\\ -\mu\left[T(t)\cdot\frac{\partial f_1(\upsilon)}{\partial\upsilon}\cdot\left(f_1(\upsilon)-f_2(\omega)\right)+\frac{\partial p_1(\upsilon)}{\partial\upsilon}\cdot p_1(\upsilon)\right] & \text{otherwise}\end{cases}\tag{24}$$

$$\frac{du_2}{dt}=\begin{cases}-\mu\left[-T(t)\cdot\frac{\partial f_2(\omega)}{\partial\omega}\cdot\left(f_1(\upsilon)-f_2(\omega)\right)+\frac{\partial p_2(\omega)}{\partial\omega}\cdot p_2(\omega)\right]+\eta_2\left(\chi_2(b_2-a_2)+a_2\right) & random<P_2(t)\\ -\mu\left[-T(t)\cdot\frac{\partial f_2(\omega)}{\partial\omega}\cdot\left(f_1(\upsilon)-f_2(\omega)\right)+\frac{\partial p_2(\omega)}{\partial\omega}\cdot p_2(\omega)\right] & \text{otherwise}\end{cases}\tag{25}$$

$$\upsilon=q(u_1)=\frac{b_1-a_1}{1+e^{-u_1/\varepsilon_1}}+a_1\tag{26}$$

$$\omega=q(u_2)=\frac{b_2-a_2}{1+e^{-u_2/\varepsilon_2}}+a_2\tag{27}$$

$$\eta_i(t+1)=(1-\kappa)\eta_i(t)\quad i=1,2\tag{28}$$

$$P_i(t+1)=\begin{cases}P_i(t)-\delta & P_i(t)>0\\ 0 & \text{otherwise}\end{cases}\tag{29}$$

$$\chi_i(t+1)=\gamma\chi_i(t)\left(1-\chi_i(t)\right)\tag{30}$$

Where $\gamma=4$, $P_i(0)>0$, $0<\kappa<1$, $0<\delta<1$, $\eta_i(0)>0$, $\varepsilon_1>0$ and $\varepsilon_2>0$. We know that equation (30) is a Logistic map, when $\gamma=4$, the chaos phenomenon will happen in the system.

As time approaches infinity, the chaotic annealing recurrent neural network will evolve into the annealing recurrent neural network (10) ~ (13). Therefore, we must not repeatedly analyze the stability and solution feasibility and solution optimality of the chaotic annealing recurrent neural network (24) ~ (30).

5. Simulation analysis

5.1 A simplified tight formation flight model simulation

Consider a simplified tight formation flight model consisting of two Unmanned Aerial Vehicles[8].

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\end{bmatrix}=\begin{bmatrix}0 & 1 & 0 & 0\\ -20 & -9 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & -35 & -15\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\\ x_4\end{bmatrix}+\begin{bmatrix}0 & 0\\ 1 & 0\\ 0 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix}\tag{31}$$

with a cost function given by

$$y(t) = -10(x_1(t)+0)^2 - 5(x_3(t)+9)^2 + 590 \tag{32}$$

Where x_1 is the vertical separation of two Unmanned Aerial Vehicles, x_2 is the differential of x_1, x_3 is the lateral separation of two Unmanned Aerial Vehicles, x_4 is the differential of x_3 and y is the upwash force acting on the wingman. It is clear that the global maximum point is $x_1^* = 0$ and $x_3^* = -9$, where the cost function $y(t)$ reaches its maximum $y^* = 590$.
A control law based on sliding mode theory is given by:

$$\begin{cases} \sigma_1 = s_1 x_1 + x_2 \\ u_1 = 20x_1 + (9 - s_1)x_2 - k_1 sign(\sigma_1 - s_1\theta_1) \\ \sigma_2 = s_2 x_3 + x_4 \\ u_2 = 35x_3 + (15 - s_2)x_4 - k_2 sign(\sigma_2 - s_2\theta_2) \end{cases} \tag{33}$$

Where σ_1, σ_2 are two sliding mode surfaces, s_1, k_1, s_2, k_2 are positive scalar constants, $\theta = [\theta_1, \theta_2]$ are an extremum seeking vector.
Remark: The control law is given in (33), which is based on sliding mode theory. We choose $sign(\sigma_i - s_i\theta_i)$, $(i = 1,2)$ so that x_1 and x_3 entirely traces θ_1 and θ_2 in the sliding mode surfaces respectively, and the system will be stable at θ_1^* and θ_2^* finally.
The initial conditions of the system (31) are given as $x_1(0) = -2$, $x_2(0) = 0$, $x_3(0) = -4$, $x_4(0) = 0$, $\theta_1(0) = -2$, $\theta_2(0) = -4$. Choose $T(t) = \beta\alpha^{-\eta t}$, where $\beta = 0.01, \alpha = e, \eta = 5$. Applying CARNN to system (31), the parameters are given as: $\mu = 23.5$, $\gamma = 4$, $P_1(0) = P_2(0) = 1$, $\kappa = 0.01, \delta = 0.01$, $\varepsilon_1 = 10$, $\varepsilon_2 = 10$, $\chi_1(0) = 0.912$, $\chi_2(0) = 0.551$, $\eta_1(0) = [-10 \quad -1 \quad 5]^T$, $\eta_2(0) = [3 \quad 10 \quad 5]^T$, $b_1 = b_2 = 0.5$, $a_1 = a_2 = -0.5$. The simulation results are shown from figure 1 to figure 3.

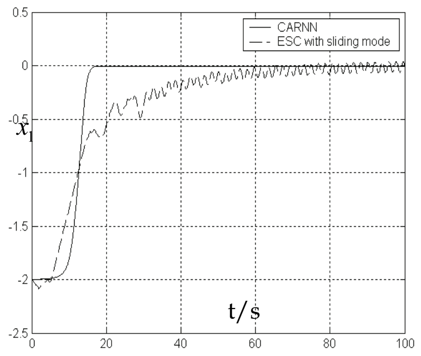


Fig. 1. The simulation result of the state x_1

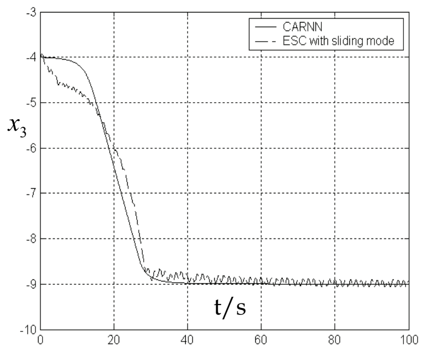


Fig. 2. The simulation result of the state x_3

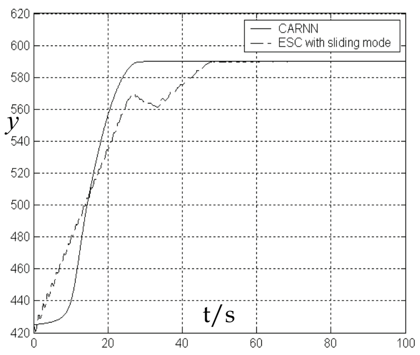


Fig. 3. The simulation result of the output y

Certainly, μ is a main factor of scaling the convergence rate of CARNN, if it is too big, the error of the output will be introduced. On the contrary, if it is too small, the convergence rate of the system will be slow. In conclusion, the values of those parameters should be verified by the system simulation.

In those simulation results, solid lines are the results applying CARNN to ESA; dash lines are the results applying ESA with sliding mode[9]. Comparing those simulation results, we know the dynamic performance of the method proposed in the paper is superior to that of ESA with sliding mode. The "chatter" of the CARNN's output doesn't exist in figure 1 and 2, which is very harmful in practice. Moreover the convergence rate of ESA with CARNN can be scaled by adjusting the chaotic annealing parameter $T(t)$.

5.2 Schaffer function simulation

In order to exhibit the capability of global searching of the proposed CARNN, the typical Schaffer function (34) is defined as the testing function[10].

$$f(x_1,x_2)=\frac{\sin^2\sqrt{x_1^2+x_2^2}-0.5}{\left(1+0.001\left(x_1^2+x_2^2\right)\right)^2}-0.5, |x_i|\le 10, i=1,2 \tag{34}$$

When $x_1=x_2=0$, the schaffer function $f(x_1,x_2)$ will obtain the global minimum $f(0,0)=-1$. There are numerous local minimums and maximums among the range of 3.14 away from the global minimum.

Now, we define $\theta_1=x_1$ and $\theta_2=x_2$. Choose $T(t)=\beta\alpha^{-\eta t}$, where $\beta=0.01$, $\alpha=e$, $\eta=3$, and apply the CARNN to search the global minimum of the function (34). The neural network parameters are given as: $\mu=35$, $\gamma=4$, $P_1(0)=P_2(0)=1$, $\kappa=0.01$, $\delta=0.001$, $\varepsilon_1=10$, $\varepsilon_2=10$, $\chi_1(0)=0.912$, $\chi_2(0)=0.551$, $\eta_1(0)=[-200 \quad -20 \quad 50]^T$, $\eta_2(0)=[100 \quad 300 \quad 50]^T$, $b_1=b_2=0.5$, $a_1=a_2=-0.5$. When the initial conditions of the function (34) are given as $x_1=-2$ and $x_2=3.5$, the simulation results are shown from figure 4 to figure 6.

When the initial conditions of the function (34) are given as $x_1(0) = -1$ and $x_2 = 9.59$, the simulation results are shown as from figure 7 to figure 9.

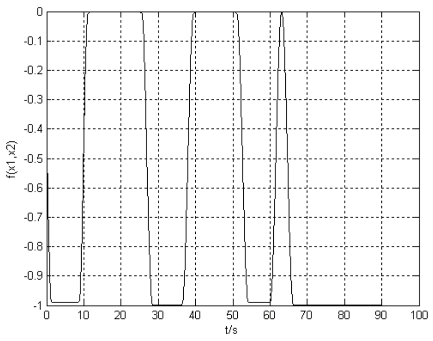


Fig. 4. The simulation result of $f(x_1,x_2)$

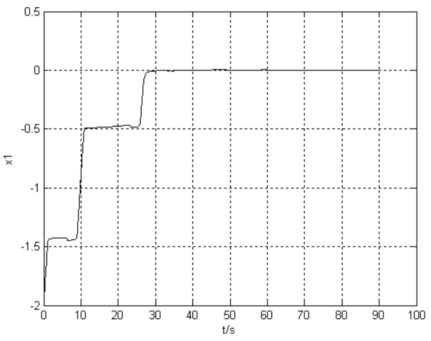


Fig. 5. The simulation result of x_1

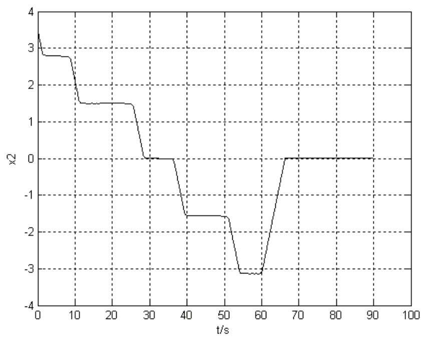


Fig. 6. The simulation result of x_2

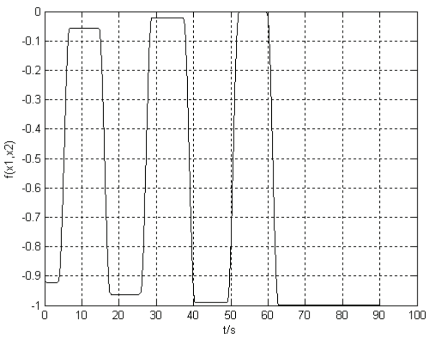


Fig. 7. The simulation result of $f(x_1,x_2)$

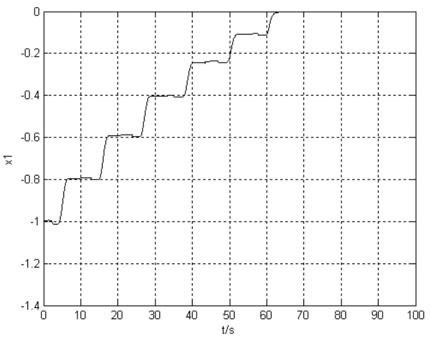


Fig. 8. The simulation result of x_1

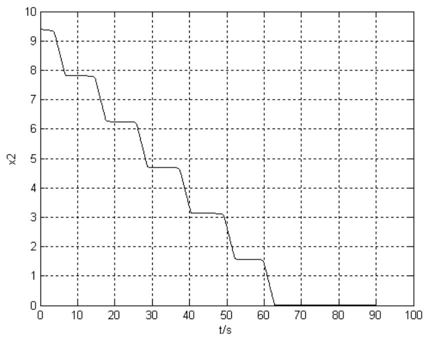


Fig. 9. The simulation result of x_2

We have accomplished a great deal of computer simulations in different initial conditions. The ESA based on the chaotic annealing recurrent neural network can find the global minimum of Schaffer function under different conditions of the simulation.

6. Referring

The method of introducing CARNN into ESA greatly improves the dynamic performance and the global searching capability of the system. Two phases of the coarse search based on chaos and the elaborate search based on ARNN guarantee that the system could fully carry out the chaos searching and find the global extremum point and accordingly converge to that point. At the same time, the disappearance of the "chatter" of the system output and the switching of the control law are beneficial to engineering applications.

7. Acknowledgements

This research was supported by the Natural Science Foundation of P.R.China (No. 60674090).

8. References

Natalia I. M. (2003). Applications of the Adaptive Extremum Seeking Control Techniques to Bioreactor Systems. A dissertation for the degree of Master of Science. Ontario: Queen's University.

Pan, Y., Ozguner, U., and Acarman, T. (2003). Stability and Performance Improvement of Extremum Seeking Control with Sliding Mode. *International Journal of Control,* Vol. 76, pp. 968-985. ISSN 0020-7179.

Drakunov, S., Ozguner, U., Dix, P., and Ashrafi, B. (1995). ABS Control Using Optimum Search via Sliding Mode., *IEEE Transactions on Control Systems Technology,* Vol. 3, No. 1, pp. 79-85. ISSN 1063-6536.

Krstic, M. (1999). Toward Faster Adaptation in Extremum Seeking Control. *Proceeding of the 38th IEEE Conference on Decision and Control,* pp. 4766-4771, ISBN 0-7803-5250-5, Phoenix, USA, Decemeber 1999.

Ying Tan, Baoyun Wang, Zhenya He. (1998). Neural Networks with Transient Chaos and Time-variant gain and Its Application to Optimization Computations. *ACTA ELECTRONICA SINICA,* Vol. 26, No. 7, pp. 123-127. ISBN 0372-2112.

Wang Ling, Zheng Dazhong. (2000). A Kind of Chaotic Neural Network Optimization Algorithm Based on Annealing Strategy. *Control Theory & Applications,* Vol. 17, No. 1, pp. 139-142. ISSN 1000-8152.

Y A Hu, B Zuo. (2005). An Annealing Recurrent Neural Network for Extremum Seeking Control. *International Journal of Information Technology,* Vol. 11, No. 6, pp. 45-52. ISSN 1305-2403.

B. Zuo, and Y. A. Hu. (2004). Optimizing UAV Close Formation Flight via Extremum Seeking. *Proceedings of WCICA2004,* Vol. 4, pp. 3302-3305. ISBN 0-7803-8273-0, Hangzhou, China, June, 2004.

Yu, H., and Ozguner, U. (2002). Extremum-Seeking Control via Sliding Mode with Periodic Search Signals. *Proceeding of the 41st IEEE Conference on Decision and Control,* pp. 323-328. ISBN 0-7803-7516-5, Las Vegas, USA, December 2002.

Wang Ling. (2004). *Intelligent Optimization Algorithms with Application,* Tsinghua University Press and Springer, ISBN 7-302-04499-6, Beijing, China.

15

Stability Results for Uncertain Stochastic High-Order Hopfield Neural Networks with Time Varying Delays[1]

P. Balasubramaniam and R. Rakkiyappan
Department of Mathematics, Gandhigram Rural University
Tamilnadu,
India

1. Introduction

Neural networks have been widely applied in image processing, pattern recognition, optimization solvers, fixed-point computation and other engineering areas. It has been known that these applications heavily depend on the dynamic behaviors of neural networks. The stability of neural networks has been extensively studied over the past years because it is one of the most important behaviors of neural networks. On the other hand, time delays are frequently encountered in neural networks due to the finite switching speed of amplifiers and the inherent communication time of neurons. Since the existence of time delay is often a source of instability for neural networks, the stability study for delayed neural networks is of both theoretical and practical importance.

Hopfield [9, 10] has proposed Hopfield neural networks (HNNs) which have found applications in a broad range of disciplines where the targeted problems can reduce to optimization problems. In recent years, HNNs and their various generalizations have attracted the great attention of many scientists including mathematicians, physicists, computer scientists due to their potential for the tasks of classification, associative memory, parallel computation and their ability to solve difficult optimization problems, see for example [4, 10, 13]. HNNs characterized by first-order interactions, [1, 14] presented their intrinsic limitations. Recently, the study of high-order neural networks has received much attention due to that they have stronger approximation property, faster convergence rate, greater storage capacity and higher fault tolerance than lower-order neural networks [17]. In [3, 5, 6, 8, 11, 12, 15, 16, 18, 19, 22], the authors have been studied the stability analysis of high-order neural networks with constant time delays or time varying delays. In this paper, we are concerned with the global stability for a class of uncertain stochastic high-order neural networks with time varying delays. The structure of the stochastic neural networks under consideration is more general than some previous ones existed in the literature. Based on the Lyapunov stability theory, new global asymptotic stability criteria are presented in

[1] The work of the authors was supported by UGC, New Delhi under SAP(DRS) sanctioned No. F510/6/DRS/2004 (SAP-1).

terms of LMIs . Finally, we also provide a numerical example to demonstrate the effectiveness of the proposed stability results.

2. Problem description and preliminaries

Throughout this chapter we will use the notation $A > 0$ (or $A < 0$) to denote that the matrix A is a symmetric and positive definite (or negative definite) matrix. The notation A^T and A^{-1} mean the transpose of A and the inverse of a square matrix. If A,B are symmetric matrices $A > B$ ($A \geq B$) means that $A - B$ is positive definite (positive semi-definite).
Consider the following high-order Hopfield neural networks with time varying delays described by

$$\begin{aligned}\frac{dx_i(t)}{dt} &= -c_i x_i(t) + \sum_{j=1}^{n} a_{ij} f_j(x_j(t)) + \sum_{j=1}^{n} b_{ij} f_j(x_j(t-\tau_j(t))) \\ &\quad + \sum_{j=1}^{n}\sum_{l=1}^{n} T_{ijl} f_j(x_j(t-\tau_j(t))) f_l(y_l(t-\tau_l(t))) + J_i, \qquad i = 1,2,...,n.\end{aligned} \tag{1}$$

where $i \in \{1, 2, , ..., n\}$, $t \geq t_0$, $x_i(t)$ is the neuron state; c_i is positive constant, it denotes the rate with which the cell resets its potential to the resting state; a_{ij}, b_{ij} are the first-order synaptic weights of the neural networks; T_{ijl} is the second-order synaptic weights of the neural networks; $\tau_j(t)$ ($j = 1, 2, ..., n$) is the transmission delay of the jth neuron such that $0 < \tau_j(t) \leq \tau_j^*$ and $\dot{\tau}_j(t) \leq \eta_j < 1$, where τ_j^*, η_j are constants; the activation function f_j is continuous on $[t_0 - \tau^*, +\infty)$; J_i is the external input.
Assume that
(H1) In the neuron activation function $f(y) = (f_1(y_1), f_2(y_2), \ldots, f_n(y_n))^T$, each function f_i is continuously differentiable with $f_i(0) = 0$ and there exists a positive scalars L_i and $\mathcal{X}_i$ such that for any $\alpha_i, \beta_i \in \mathbb{R}$,

$$|f_i(\alpha_i)| \leq \chi_i \qquad 0 \leq [f_i(\alpha_i) - f_i(\beta_i)](\alpha_i - \beta_i) \leq L_i(\alpha_i - \beta_i)^2.$$

Due to the boundedness of the activation function f_i, by employing the well known Brouwer's fixed point theorem, we can easily obtain that there exists an equilibrium point of the system (1). The uniqueness of the equilibrium point can be deduced from the asymptotic stability which will be proved subsequently.
Let x^* be an equilibrium point of (1) and $y(t) = x(t) - x^*$. Set $g_j(y_j(t)) = f_j(x_j(t)) - f_j(x^*_j)$, $g_j(y_j(t - \tau_j(t))) = f_j(x_j(t - \tau_j(t))) - f_j(x^*_j)$. Apparently, for each $i = 1, 2, ..., n$, we have

$$|g_j(z)| \leq L_j|z|, \qquad \forall z \in \mathbb{R}$$

Consider the following high-order HNNs with time varying delay is given by

$$\tfrac{dy(t)}{dt} = -Cy(t) + Ag(y(t)) + (B + \Gamma^T T_H)g(y(t-\tau(t))) \tag{2}$$

where

$$\begin{aligned}
C &= diag(c_1, c_2, ..., c_n),\\
A &= (a_{ij})_{n\times n}, \quad B = (b_{ij})_{n\times n},\\
T_i &= (T_{ijl})_{n\times n},\\
T_H &= (T_1 + T_1^T, T_2 + T_2^T, ..., T_n + T_n^T)^T,\\
y(t-\tau(t)) &= \Big(y_1(t-\tau_1(t)), y_2(t-\tau_2(t)), ..., y_n(t-\tau_n(t))\Big)^T,\\
g(y(t)) &= \Big(g_1(y_1(t)), g_2(y_2(t)), ..., g_n(y_n(t))\Big)^T,\\
g(y(t-\tau(t))) &= \Big(g_1(y_1(t-\tau_1(t))), g_2(y_2(t-\tau_2(t))), ..., g_n(y_n(t-\tau_n(t)))\Big)^T,\\
\zeta &= (\zeta_1, \zeta_2, ..., \zeta_n)^T,\\
\Gamma &= diag(\zeta, \zeta, ..., \zeta).
\end{aligned}$$

In this paper the following high-order HNN with parameter uncertainties and stochastic perturbations is considered

$$\begin{aligned}
dy(t) &= \Big[-(C+\Delta C(t))y(t) + (A+\Delta A(t))g(y(t)) + ((B+\Delta B(t)) + \Gamma^T T_H)g(y(t-\tau(t)))\Big]\\
&\quad + \sigma(t, y(t), y(t-\tau(t)))dw(t), \qquad (3)
\end{aligned}$$

where $w(t) = (w_1(t), w_2(t), ..., w_m(t))^T$ is an m-dimensional Brownian motion defined on a complete probability space $(\Omega, \mathcal{F}, P)$ with a natural filtration $\{\mathcal{F}_t\}_{t\geq 0}$. Let $\sigma(t, x, y) : \mathbb{R}_+\times\mathbb{R}^n\times\mathbb{R}^n \rightarrow \mathbb{R}^{n\times m}$ is locally Lipschitz continuous and satisfies the linear growth condition. The uncertainties $\Delta C(t)$, $\Delta A(t)$, $\Delta B(t)$ are defined by

$$\Delta C(t) = MF(t)N_C, \quad \Delta A(t) = MF(t)N_A, \quad \Delta B(t) = MF(t)N_B,$$

where $\Delta C(t)$ is a diagonal matrix and M, N_C, N_A and N_B are known real constant matrices with appropriate dimensions, which characterize how the deterministic uncertain parameter in $F(t)$ enters the nominal matrices C, A and B. The matrix $F(t)$, which is time varying unknown and satisfies

$$F(t)^T F(t) \leq I.$$

Let $x(t;\xi)$ denote the state trajectory of the neural network (3) from the initial data $x(\theta) = \xi(\theta)$ on $-\tau^* \leq \theta \leq 0$ in $L^2_{\mathcal{F}_0}([-\tau^*, 0], \mathbb{R}^n)$. It can be easily seen that the system (3) admits a trivial solution $x(t; 0) \equiv 0$ corresponding to the initial data $\xi = 0$, see [2, 7].

3. Main results

Let $C_{2,1}(\mathbb{R}^n \times \mathbb{R}_+ : \mathbb{R}_+)$ denote the family of all non-negative functions $V(y, t)$ on $\mathbb{R}^n \times \mathbb{R}_+$ which are continuously twice differentiable in x and once differentiable in t. For each $V \in C_{2,1}([-\tau^*, \infty] \times \mathbb{R}^n, \mathbb{R}_+)$, define an operator $\mathrm{L}V(y(t), t)$ associated with stochastic high order neural networks (3) from $\mathbb{R}_+ \times C([-\tau^*, 0]; \mathbb{R}^n)$ to $\mathbb{R}$ by

$$\begin{aligned}\mathcal{L}V(y(t),t) &= V_t(y,t) + V_y(y,t)\Big[-(C+\Delta C(t))y(t) + (A+\Delta A(t))g(y(t)) + ((B+\Delta B(t)) \\ &\quad + \Gamma^T T_H)g(y(t-\tau(t)))\Big] + \frac{1}{2}trace\Big[\sigma^T V_{yy}(y,t)\sigma\Big]\end{aligned}$$

where

$$V_t(y,t) = \frac{\partial V(y,t)}{\partial t}, V_y(y,t) = \Big(\frac{\partial V(y,t)}{\partial y_1}, \frac{\partial V(y,t)}{\partial y_2}, ..., \frac{\partial V(y,t)}{\partial y_n}\Big),$$

and

$$V_{yy}(y,t) = \Big(\frac{\partial^2 V(y,t)}{\partial y_i \partial y_j}\Big)_{n\times n}$$

where $i, j = 1, 2, ..., n$. In order to prove our results, we need to state the following definitions and Lemma.

Lemma 3.1. *Given any real matrices* Σ_1, Σ_2, Σ_3 *of appropriate dimensions and a scalar* $\epsilon > 0$ *such that* $0 < \Sigma_3 = \Sigma_3^T$. *Then, the following inequality holds:*

$$\Sigma_1^T \Sigma_2 + \Sigma_2^T \Sigma_1 \leq \epsilon \Sigma_1^T \Sigma_3 \Sigma_1 + \epsilon^{-1} \Sigma_2^T \Sigma_3^{-1} \Sigma_2.$$

We also recall some basic facts about norms of vectors and matrices. Let $y = (y_1, y_2, ..., y_n)^T \in \mathbb{R}^n$. Three commonly used vector norms are given as $\|y\|_1 = \sum_{i=1}^n |y_i|$, $\|y\|_2 = (\sum_{i=1}^n y_i^2)^{1/2}$ and $\|y\|_\infty = \max_{1\leq i\leq n} |y_i|$. It is also known that $\|y\|_\infty \leq \|y\|_1$. The vector $|y|$ will denote $|y| = (|y_1|, |y_2|, ..., |y_n|)^T$. For any matrix $V = (v_{ij})_{n\times n}$, $\lambda_m(V)$ and $\lambda_M(V)$ will denote respectively the minimum and maximum eigenvalues of V. For the matrix V, $\|V\|_2^2 = \lambda_M(V^T V)$.

Now we will prove the following theorem on global asymptotic stability in the mean square for equation (3).

Theorem 3.2. *Assume that there exist matrices* $P > 0$, $D_0 \geq 0$ *and* $D_1 \geq 0$ *such that*

$$trace\Big[\sigma^T\Big(t, y(t), y(t-\tau(t))\Big)P\sigma\Big(t, y(t), y(t-\tau(t))\Big)\Big] \leq y^T(t) D_0 y(t) + y^T(t-\tau(t)) D_1 y(t-\tau(t)).$$

System (3) is globally asymptotically stable in the mean square, if there exist positive definite matrices Σ_1, Σ_2 *and the scalars* $\in_k > 0$ $(k = 1, 2)$ *such that*

$$\Pi_1 = \begin{bmatrix} \psi_1 & PM & PA & PM & PB & PM & \epsilon_1 N_C & \epsilon_2 N_A^T L & L\Sigma_2 N_B \\ * & -\epsilon_1 I & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ * & * & -\epsilon_2 I & 0 & 0 & 0 & 0 & 0 & 0 \\ * & * & * & -\epsilon_3 I & 0 & 0 & 0 & 0 & 0 \\ * & * & * & * & -\Sigma_1 & 0 & 0 & 0 & 0 \\ * & * & * & * & * & -(1-\eta)\Sigma_2 & 0 & 0 & 0 \\ * & * & * & * & * & * & -\epsilon_1 I & 0 & 0 \\ * & * & * & * & * & * & * & -\epsilon_2 I & 0 \\ * & * & * & * & * & * & * & * & -\Sigma_2 \end{bmatrix} < 0 \qquad (4)$$

where $L = diag(L_i), i = 1, 2, ..., n,\ \chi = (\chi_1, \chi_2, ..., \chi_n)^T$ *and* $\psi_1 = -PC - C^T P + \epsilon_2 L^2 + D_1 + LT_H T_H^T L + \frac{1}{1-\eta}\|\chi\|^2 P^2$.

Proof: We use the following Lyapunov functional to derive the stability result

$$V(y,t) \quad = \quad y^T(t)Py(t) + \sum_{i=1}^{n}\sum_{j=1}^{n}\int_{t-\tau_j(t)}^{t} q_{ij} g_j^2(y(s))ds.$$

By Ito's formula, we can calculate $\mathcal{L}V_1(y(t),t), \mathcal{L}V_2(y(t),t)$ along the trajectories of the system (3), then we have

$$\mathcal{L}V_1(y(t),t) \quad = \quad 2y^T(t)P[-(C+\Delta C(t))y(t) + (A+\Delta A(t))g(y(t)) + ((B+\Delta B(t)) + \Gamma^T T_H)g(y(t-\tau(t)))] + trace[\sigma^T(t,y(t),y(t-\tau(t)))P\sigma(t,y(t),y(t-\tau(t)))] \qquad (5)$$

$$\mathcal{L}V_2(y(t),t) \quad = \quad g^T(y(t))Qg(y(t)) - (1-\eta)g^T(y(t-\tau(t)))Qg(y(t-\tau(t))). \qquad (6)$$

From (5)-(6), we get

$$\begin{aligned} \mathcal{L}V(y(t),t) &= \mathcal{L}V_1(y(t),t) + \mathcal{L}V_2(y(t),t) \\ &\le 2y^T(t)P[-(C+\Delta C(t))y(t) + (A+\Delta A(t))g(y(t)) + ((B+\Delta B(t)) \\ &\quad + \Gamma^T T_H)g(y(t-\tau(t)))] + trace[\sigma^T(t,y(t),y(t-\tau(t)))P\sigma(t,y(t),y(t-\tau(t)))] \\ &\quad + g^T(y(t))Qg(y(t)) - (1-\eta)g^T(y(t-\tau(t)))Qg(y(t-\tau(t))). \end{aligned} \qquad (7)$$

By Lemma 3.1 we get,

$$-2y^T(t)P(\Delta C(t))y(t) \le \epsilon_1^{-1}y^T(t)PMM^TPy(t) + \epsilon_1 y^T(t)N_CN_Cy(t) \tag{8}$$

$$2y^T(t)PAg(y(t)) \le \epsilon_2^{-1}y^T(t)PAA^TPy(t) + \epsilon_2 y^T(t)L^2y(t) \tag{9}$$

$$2y^T(t)P(\Delta A(t))g(y(t)) \le \epsilon_3^{-1}y^T(t)PMM^TPy(t) + \epsilon_3 y^T(t)LN_A^TN_ALy(t) \tag{10}$$

$$\begin{aligned} 2y^T(t)PBg(y(t-\tau(t))) &\le \frac{1}{1-\eta}y^T(t)PB\Sigma_1^{-1}B^TPy(t) \\ &+(1-\eta)g^T(y(t-\tau(t)))\Sigma_1 g(y(t-\tau(t))) \end{aligned} \tag{11}$$

$$\begin{aligned} 2y^T(t)P(\Delta B(t))g(y(t-\tau(t))) &\le \frac{1}{1-\eta}y^T(t)PM\Sigma_2^{-1}M^TPy(t) \\ &+(1-\eta)g^T(y(t-\tau(t)))N_B^T\Sigma_2 N_B g(y(t-\tau(t))) \end{aligned} \tag{12}$$

$$\begin{aligned} 2y^T(t)P\Gamma^T T_H g(y(t-\tau(t))) &\le \frac{1}{1-\eta}y^T(t)P\Gamma^T\Gamma Py(t) \\ &+(1-\eta)g^T(y(t-\tau(t)))T_H^T T_H g(y(t-\tau(t))). \end{aligned} \tag{13}$$

Since $\Gamma^T\Gamma = \|\zeta\|^2 I$ and $\|\zeta\| \le \|\chi\|$, it is clear that

$$y^T(t)P\Gamma^T\Gamma Py(t) \le \|\chi\|^2 y^T(t)P^2y(t).$$

Since $Q = L^{-1}D_1L^{-1} + N_B^T\Sigma_2N_B + T_H^TT_H$, and from (7)-(13), it follows that

$$\begin{aligned} \mathcal{L}V(y(t),t) \le\ & y^T(t)(-PC - C^TP)y(t) + y^T(t)(\epsilon_1^{-1}PMM^TP + \epsilon_1 N_C^TN_C)y(t) \\ &+y^T(t)(\epsilon_2^{-1}PAA^TP + \epsilon_2L^2)y(t) + y^T(t)(\epsilon_3^{-1}PMM^TP + \epsilon_3 LN_A^TN_AL)y(t) \\ &+\frac{1}{1-\eta}y^T(t)PB\Sigma_1^{-1}B^TPy(t) + (1-\eta)g^T(y(t-\tau(t)))\Sigma_1 g(y(t-\tau(t))) \\ &+\frac{1}{1-\eta}y^T(t)PM\Sigma_2^{-1}M^TPy(t) + (1-\eta)g^T(y(t-\tau(t)))N_B^T\Sigma_2N_Bg(y(t-\tau(t))) \\ &+\frac{1}{1-\eta}y^T(t)P\Gamma^T\Gamma Py(t) + (1-\eta)g^T(y(t-\tau(t)))T_H^TT_Hg(y(t-\tau(t))) \\ &+g^T(y(t))Qg(y(t)) - (1-\eta)g^T(y(t-\tau(t)))Qg(y(t-\tau(t))) \end{aligned}$$

$$+trace[\sigma^T(t,y(t),y(t-\tau(t)))P\sigma(t,y(t),y(t-\tau(t)))],$$

Then we have $\mathcal{L}V\,(y(t),\,t) < 0$ when $\Pi_1 < 0$, that is the inequality (4) holds, which completes the proof of the theorem.

By constructing another Lyapunov functional, we can obtain the following result.

Theorem 3.3. *Assume that there exist matrices* $D_0 \geq 0$ *and* $D_1 \geq 0$ *such that*

$$trace[\sigma^T(t,y(t),y(t-\tau(t))) \quad D \quad diag\Big\{\dot{g}_1(y_1(t)),\dot{g}_2(y_2(t)),\cdots,\dot{g}_n(y_n(t))\Big\}\sigma(t,y(t),y(t-\tau(t)))]$$

$$\leq \quad y^T(t)D_0y(t)+y^T(t-\tau(t))D_1y(t-\tau(t)).$$

System (3)is globally asymptotically stable in the mean square, if there exist positive definite matrices Σ_1 *and the scalars* $\epsilon_k > 0$ $(k = 1, 2, 3)$ *such that*

$$\Pi_2 = \begin{bmatrix} \psi_2 & DM & \epsilon_1 N_C L^{-1} & DM & \epsilon_2 N_A & DM & DB & \epsilon_3 N_B^T \\ * & -\epsilon_1 I & 0 & 0 & 0 & 0 & 0 & 0 \\ * & * & -\epsilon_1 I & 0 & 0 & 0 & 0 & 0 \\ * & * & * & -\epsilon_2 I & 0 & 0 & 0 & 0 \\ * & * & * & * & -\epsilon_2 I & 0 & 0 & 0 \\ * & * & * & * & * & -\epsilon_3 I & 0 & 0 \\ * & * & * & * & * & * & \Sigma_1 & 0 \\ * & * & * & * & * & * & * & -\epsilon_3(1-\eta)I \end{bmatrix} < 0, \quad (14)$$

where $L = diag(L_i), i = 1,2,...,n,\ \chi = (\chi_1,\chi_2,...,\chi_n)^T,\ D = diag[d_1,d_2,...,d_n] > 0\ i = 1,2,...,n$ *and* $\psi_2 = -DCL^{-1} - L^{-1}CD + DA + A^TD + \|\chi\|^2D^2 + \frac{1}{1-\eta}\Sigma_1 + \frac{1}{1-\eta}T_H^TT_H + D_1 + L^{-1}D_0L^{-1}.$

Proof: We use the following positive definite Lyapunov functional to derive the stability result,

$$V(y,t) \quad = \quad 2\sum_{i=1}^{n} d_i \int_0^{y_i} g_i(s)ds + \frac{1}{1-\eta}\sum_{i=1}^{n}\sum_{j=1}^{n}\int_{t-\tau_j(t)}^{t} q_{ij}g_j^2(y(s))ds$$

where $Q = (q_{ij})_{n\times n} = N_B^TN_B + T_H^TT_H + D_1 + \Sigma_1,\ d_i > 0,\ i = 1,2,...,n.$ Define

$$G(y) = \min\Big\{\min\Big\{\int_0^{y_i} g_i(\theta)d\theta, \int_0^{-y_i} g_i(\theta)d\theta\Big\}, i = 1,2,...,n\Big\},$$

which satisfies

$$G(r) > 0, \quad r > 0, \qquad G(r) \to +\infty, \quad r \to +\infty,$$

and $G(0)=0, G(y)=G(|y|)$, for $x\in\mathbb{R}^n_+$. We have

$$V(y(t),t)\geq 2\sum_{i=1}^{n} d_i\int_0^{y_i} g_i(s)ds \geq 2\lambda_m(D)G(|y|),$$

which gives a lower by a positive radially unbounded function.
It is to verify that

$$2\lambda_m(D)G(|y|)\leq V(y(t),t)\leq \Big[2q\lambda_M(DL)+\frac{\tau(t)}{1-\eta}q\lambda_M(LQL)\Big]\|y(t)\|^2, \quad q>1.$$

By Ito's formula, we can calculate $\mathcal{L}V_1(y(t),\ t)$,$\mathcal{L}V_2(y(t),\ t)$ along the trajectories of the system (3), then we have

$$\begin{aligned}\mathcal{L}V_1(y(t),t) &= 2y^T(t)D[-(C+\Delta C(t))y(t)+(A+\Delta A(t))g(y(t))+((B+\Delta B(t))\\ &\quad +\Gamma^T T_H)g(y(t-\tau(t)))]+trace[\sigma^T(t,y(t),y(t-\tau(t)))D\sigma(t,y(t),y(t-\tau(t)))]\\ &\quad diag\Big\{\dot{g}_1(y_1(t)),\dot{g}_2(y_2(t)),\cdots,\dot{g}_n(y_n(t))\Big\}, \qquad (15)\end{aligned}$$

$$\mathcal{L}V_2(y(t),t) = \frac{1}{1-\eta}g^T(y(t))Qg(y(t))-g^T(y(t-\tau(t)))Qg(y(t-\tau(t))). \qquad (16)$$

Then it follows from Lemma 3.1 that

$$\begin{aligned}-2g^T(y(t))D(\Delta C(t))y(t) &\leq \epsilon_1^{-1}g^T(y(t))DMM^TDy(t)\\ &\quad +\epsilon_1 g^T(y(t))L^{-1}N_C^TN_CL^{-1}g(y(t)) \qquad (17)\end{aligned}$$

$$\begin{aligned}2g^T(y(t))D(\Delta A(t))g(y(t)) &\leq \epsilon_2^{-1}y^T(t)DMM^TDy(t)\\ &\quad +\epsilon_2 g^T(y(t))LN_A^TN_ALg(y(t)) \qquad (18)\end{aligned}$$

$$\begin{aligned}2g^T(y(t))D(\Delta B(t))g(y(t-\tau(t))) &\leq \epsilon_3^{-1}g^T(y(t))DMM^TDg(y(t))\\ &\quad +\epsilon_3 g^T(y(t-\tau(t)))N_B^TN_Bg(y(t-\tau(t))) \qquad (19)\end{aligned}$$

$$\begin{aligned}g^T(y(t))(t)DBg(y(t-\tau(t))) + g^T(y(t-\tau(t)))B^TDg(y(t)) &\leq g^T(y(t))DB\Sigma_1^{-1}B^TDg(y(t))\\ &\quad +g^T(y(t-\tau(t)))\Sigma_1 g(y(t-\tau(t))) \qquad (20)\end{aligned}$$

$$\begin{aligned} g^T(y(t))(t)D\Gamma^T T_H g(y(t-\tau(t))) \quad + \quad & g^T(y(t-\tau(t)))T_H^T\Gamma g(y(t)) \leq g^T(y(t))D\Gamma^T\Gamma Dg(y(t)) \\ & + g^T(y(t-\tau(t)))T_H^T T_H g(y(t-\tau(t))) \\ \leq \quad & \|\chi\|^2 g^T(y(t))D^2 g(y(t)) \\ & + g^T(y(t-\tau(t)))T_H^T T_H g(y(t-\tau(t))). \end{aligned} \tag{21}$$

Since $Q = N_B^T N_B + T_H^T T_H + D_1 + \Sigma_1$, and from (15)-(21) it follows that

$$\mathcal{L}V(y(t),t) \quad \leq \quad g^T(y(t))\Pi g(y(t)).$$

Then we have $\mathcal{L}V(y(t), t) < 0$ when $\Pi_2 < 0$, that is the inequality (14) holds, which completes the proof of the theorem.

Theorem 3.4. *Assume that there exist matrices* $C > 0$, $D_0 \geq 0$ *and* $D_1 \geq 0$ *such that*

$$trace[\sigma^T(t,y(t),y(t-\tau(t)))(C+DL)\sigma(t,y(t),y(t-\tau(t)))] \leq y^T(t)D_0 y(t) + y^T(t-\tau(t))D_1 y(t-\tau(t)).$$

System (3)is globally asymptotically stable in the mean square, if the condition (H_1) is satisfied and there exists positive constants β, ϵ_i, i = 4, 5, 6 such that

$$\Omega = \begin{bmatrix} \psi_3 & D & DM & DM & DM & \epsilon_4 L^{-1}N_C & \epsilon_5 N_A^T & \epsilon_6 N_B^T \\ * & -\beta P & 0 & 0 & 0 & 0 & 0 & 0 \\ * & * & -\epsilon_4 I & 0 & 0 & 0 & 0 & 0 \\ * & * & * & -\epsilon_5 I & 0 & 0 & 0 & 0 \\ * & * & * & * & -\epsilon_6 I & 0 & 0 & 0 \\ * & * & * & * & * & -\epsilon_4 I & 0 & 0 \\ * & * & * & * & * & * & -\epsilon_5 I & 0 \\ * & * & * & * & * & * & * & -\epsilon_6 I \end{bmatrix} < 0, \tag{22}$$

where $L = diag(L_i), i = 1,2,...,n,\ \chi = (\chi_1,\chi_2,...,\chi_n)^T,\ D = diag[d_1,d_2,...,d_n] > 0$ *where* $i = 1,2,...,n$ *and* $\psi_3 = -2DCL^{-1} + DA + A^T D + \frac{1+\|\chi\|^2}{1-\eta}\beta\lambda_M(P)(B^T B + T_H^T T_H)$.

Proof: We use the following positive definite Lyapunov functional to derive the stability result,

$$\begin{aligned} V(y,t) \quad = \quad & y^T(t)Cy(t) + 2\sum_{i=1}^{n} d_i \int_0^{y_i} g_i(s)ds + \frac{1}{1-\eta}\sum_{i=1}^{n}\sum_{j=1}^{n}\int_{t-\tau_j(t)}^{t} q_{ij}g_j^2(y(s))ds \\ & + \alpha\beta\frac{1}{1-\eta}\int_{t-\tau(t)}^{t} g^T(y(s))W^T PWg(y(s))ds + \frac{1}{1-\eta}\int_{t-\tau(t)}^{t} g^T(y(s))W^T Wg(y(s))ds, \end{aligned}$$

where $W = B + \Gamma^T T_H$ and $Q = (q_{ij})_{n\times n} = (\epsilon_3^{-1} + \alpha \epsilon_6^{-1}) N_B^T N_B + L^{-1} D_1 L^{-1}$. By Ito's formula, we can calculate $\mathcal{L}V^1, \mathcal{L}V^2, \mathcal{L}V^3, \mathcal{L}V^4$ and $\mathcal{L}V^5$ along the trajectories of the system (3), then we have

$$\begin{aligned}\mathcal{L}V_1(y(t),t) \quad \leq \quad & -y^T(t)C^2y(t) - y^T(t)C^2y(t) - 2y^T(t)C(\Delta C(t))y(t) + 2y^T(t)CAy(t) \\ & +2y^T(t)C(\Delta A(t))y(t) + 2y^T(t)CWg(y(t-\tau(t))) + 2y^T(t)C(\Delta B(t))g(y(t-\tau(t))) \\ & +trace[\sigma^T(t,y(t),y(t-\tau))C\sigma(t,y(t),y(t-\tau))]. \qquad (23)\end{aligned}$$

Using the inequality technique, we have

$$\begin{aligned}-y^T(t)C^2y(t) + 2y^T(t)CAy(t) \quad = \quad & -(Cy(t) - Ag(y(t)))^T(Cy(t) - Ag(y(t))) \\ & +g^T(y(t))A^TAg(y(t)), \qquad (24)\end{aligned}$$

$$\begin{aligned}-y^T(t)C^2y(t) + 2y^T(t)CWg(y(t-\tau(t))) \quad = \quad & -(Cy(t) - Wg(y(t-\tau(t))))^T(Cy(t) - Wg(y(t-\tau(t)))) \\ & +g^T(y(t-\tau(t)))W^TWg(y(t-\tau(t))). \qquad (25)\end{aligned}$$

From Lemma 3.1, it follows that

$$\begin{aligned}2y^T(t)C(\Delta C(t))y(t) \quad \leq \quad & \epsilon_1^{-1}g^T(y(t))L^{-1}CMM^TCL^{-1}g(y(t)) \\ & +\epsilon_1 g^T(y(t))L^{-1}N_C^TN_CL^{-1}g(y(t)) \qquad (26)\end{aligned}$$

$$\begin{aligned}2y^T(t)C(\Delta A(t))y(t) \quad \leq \quad & \epsilon_2^{-1}g^T(y(t))L^{-1}CMM^TCL^{-1}g(y(t)) \\ & +\epsilon_2 g^T(y(t))N_A^TN_Ag(y(t)) \qquad (27)\end{aligned}$$

$$\begin{aligned}2y^T(t)C(\Delta B(t))g(y(t-\tau(t))) \quad \leq \quad & \epsilon_3 g^T(y(t))L^{-1}CMM^TCL^{-1}g(y(t)) \\ & +\epsilon_3^{-1}g^T(y(t-\tau(t)))N_B^TN_Bg(y(t-\tau(t))). \qquad (28)\end{aligned}$$

Since the first term of the equations (24) and (25) are non-positive, we can write the following inequalities:

$$-y^T(t)C^2y(t) + 2y^T(t)CAy(t) \quad \leq \quad g^T(y(t))A^TAg(y(t)) \qquad (29)$$

$$-y^T(t)C^2y(t) + 2y^T(t)CWg(y(t-\tau(t))) \quad \leq \quad g^T(y(t-\tau(t)))W^TWg(y(t-\tau(t))). \qquad (30)$$

Substitute (26)-(30) in (23), we get

$$\begin{aligned}\mathcal{L}V_1(y(t),t) \;\leq\;& g^T(y(t))A^TAg(y(t)) + g^T(y(t-\tau(t)))W^TWg(y(t-\tau(t)))\\ &+\epsilon_1^{-1}g^T(y(t))L^{-1}CMM^TCL^{-1}g(y(t)) + \epsilon_1 g^T(y(t))L^{-1}N_C^TN_CL^{-1}g(y(t))\\ &+\epsilon_2^{-1}g^T(y(t))L^{-1}CMM^TCL^{-1}g(y(t)) + \epsilon_2 g^T(y(t))N_A^TN_Ag(y(t))\\ &+\epsilon_3^{-1}g^T(y(t))L^{-1}CMM^TCL^{-1}g(y(t)) + \epsilon_3 g^T(y(t-\tau(t)))N_B^TN_Bg(y(t-\tau(t)))\\ &+trace[\sigma^T(t,y(t),y(t-\tau(t)))C\sigma(t,y(t),y(t-\tau(t)))]. \qquad (31)\end{aligned}$$

Also,

$$\begin{aligned}\mathcal{L}V_2(y(t),t) \;=\;& 2\alpha g^T(y(t))D\Big[-(C+\Delta C(t))y(t) + (A+\Delta A(t))g(y(t)) + ((B+\Delta B(t))\\ &+\Gamma^TT_H)g(y(t-\tau(t)))\Big] + trace[\sigma^T(t,y(t),y(t-\tau(t)))DL\sigma(t,y(t),y(t-\tau(t)))].\end{aligned}$$

Adding and subtracting $\alpha\beta^{-1}g^T(y(t))DP^{-1}Dg(y(t))$ in the above equation, then we have

$$\begin{aligned}\mathcal{L}V_2(y(t),t) \;\leq\;& -2\alpha g^T(y(t))DCy(t) - 2\alpha g^T(y(t))D(\Delta C(t))y(t) + 2\alpha g^T(y(t))DAg(y(t))\\ &+2\alpha g^T(y(t))D(\Delta A(t))g(y(t)) + 2\alpha g^T(y(t))DWg(y(t-\tau(t)))\\ &+2\alpha g^T(y(t))D(\Delta B(t))g(y(t-\tau(t))) + \alpha\beta^{-1}g^T(y(t))DP^{-1}Dg(y(t))\\ &-\alpha\beta^{-1}g^T(y(t))DP^{-1}Dg(y(t)). \qquad (32)\end{aligned}$$

From Lemma 3.1, it follows that

$$\begin{aligned}-2\alpha g^T(y(t))D(\Delta C(t))y(t) \;\leq\;& \alpha\epsilon_4^{-1}g^T(y(t))DMM^TDg(y(t))\\ &+\alpha\epsilon_4 g^T(y(t))L^{-1}N_C^TN_CL^{-1}g(y(t)) \qquad (33)\end{aligned}$$

$$\begin{aligned}2\alpha g^T(y(t))D(\Delta A(t))g(y(t)) \;\leq\;& \alpha\epsilon_5^{-1}g^T(y(t))DMM^TDg(y(t))\\ &+\alpha\epsilon_5 g^T(y(t))N_A^TN_Ag(y(t)) \qquad (34)\end{aligned}$$

$$\begin{aligned}2\alpha g^T(y(t))D(\Delta B(t))g(y(t-\tau(t))) \;\leq\;& \alpha\epsilon_6^{-1}g^T(y(t))DMM^TDg(y(t))\\ &+\alpha\epsilon_6 g^T(y(t-\tau(t)))N_B^TN_Bg(y(t-\tau(t))) \qquad (35)\end{aligned}$$

and

$$-2\alpha g^T(y(t))DCy(t) \leq -2\alpha g^T(y(t))DCL^{-1}g(y(t)). \tag{36}$$

Using the inequality technique, we have

$$\begin{aligned}
-\alpha\beta^{-1}g^T(y(t))DP^{-1}Dg(y(t)) \quad &+ \quad 2\alpha g^T(y(t))DWg(y(t-\tau(t))) \\
&= \quad -\alpha[\beta^{\frac{1}{2}}DP^{\frac{1}{2}}g(y(t)) - \beta^{\frac{1}{2}}P^{\frac{1}{2}}Wg(y(t-\tau(t)))]^T \\
&\quad \times[\beta^{\frac{1}{2}}DP^{\frac{1}{2}}g(y(t)) - \beta^{\frac{1}{2}}P^{\frac{1}{2}}Wg(y(t-\tau(t)))] \\
&\quad +\alpha\beta g^T(y(t-\tau(t)))PWg(y(t-\tau(t))).
\end{aligned}$$

Since the first term of the above equation is non-positive, we can write the following inequality

$$\begin{aligned}
-\alpha\beta^{-1}g^T(y(t))DP^{-1}Dg(y(t)) \quad &+ \quad 2\alpha g^T(y(t))DWg(y(t-\tau(t))) \\
&\leq \quad \alpha\beta g^T(y(t-\tau(t)))PWg(y(t-\tau(t))).
\end{aligned} \tag{37}$$

Substitute (33)-(37) in (32), we get

$$\begin{aligned}
\mathcal{L}V_2(y(t),t) \quad \leq \quad & -2\alpha g^T(y(t))DCL^{-1}g(y(t)) + 2\alpha g^T(y(t))DAg(y(t)) + \alpha\epsilon_4^{-1}g^T(y(t))DMM^TDg(y(t)) \\
& +\alpha\epsilon_4 g^T(y(t))L^{-1}N_C^TN_Cg(y(t)) + \alpha\epsilon_5^{-1}g^T(y(t))DMM^TDg(y(t)) \\
& +\alpha\epsilon_5 g^T(y(t))N_A^TN_Ag(y(t)) + \alpha\epsilon_6^{-1}g^T(y(t))DMM^TDg(y(t)) \\
& +\alpha\epsilon_6 g^T(y(t-\tau(t)))N_B^TN_Bg(y(t-\tau(t))) + \alpha\beta^{-1}g^T(y(t))DP^{-1}Dg(y(t)) \\
& +\alpha\beta g^T(y(t-\tau(t)))W^TPWg(y(t-\tau(t))) \\
& +trace[\sigma^T(t,y(t),y(t-\tau(t)))DL\sigma(t,y(t),y(t-\tau(t)))],
\end{aligned} \tag{38}$$

$$\mathcal{L}V_3(y(t),t) \quad \leq \quad \frac{1}{1-\eta}g^T(y(t))Qg(y(t)) - g^T(y(t-\tau(t)))Qg(y(t-\tau(t))), \tag{39}$$

$$\mathcal{L}V_4(y(t),t) \quad \leq \quad \alpha\beta\frac{1}{1-\eta}g^T(y(t))W^TPWg(y(t)) - \alpha\beta g^T(y(t-\tau(t)))W^TPWg(y(t-\tau(t))), \tag{40}$$

$$\mathcal{L}V_5(y(t),t) \quad \leq \quad \frac{1}{1-\eta}g^T(y(t))W^TWg(y(t)) - g^T(y(t-\tau(t)))W^TWg(y(t-\tau(t))). \tag{41}$$

From (31) and (38)-(41), it follows that

$$\begin{aligned}
\mathcal{L}V(y(t),t) &= \mathcal{L}V_1(y(t),t) + \mathcal{L}V_2(y(t),t) + \mathcal{L}V_3(y(t),t) + \mathcal{L}V_4(y(t),t) + \mathcal{L}V_5(y(t),t) \\
\leq\ & g^T(y(t))A^TAg(y(t)) + g^T(y(t-\tau(t)))W^TWg(y(t-\tau(t))) \\
& +(\epsilon_1^{-1}+\epsilon_2^{-1}+\epsilon_3^{-1})g^T(y(t))L^{-1}CMM^TCL^{-1}g(y(t)) + \epsilon_1 g^T(y(t))L^{-1}N_C^TN_CL^{-1}g(y(t)) \\
& +\epsilon_2 g^T(y(t))N_A^TN_Ag(y(t)) + \epsilon_3 g^T(y(t-\tau(t)))N_B^TN_Bg(y(t-\tau(t))) \\
& -2\alpha g^T(y(t))DCL^{-1}g(y(t)) + 2\alpha g^T(y(t))DAg(y(t)) \\
& +\alpha(\epsilon_4^{-1}+\epsilon_5^{-1}+\epsilon_6^{-1})g^T(y(t))DMM^TDg(y(t)) + \alpha\epsilon_4 g^T(y(t))L^{-1}N_C^TN_CL^{-1}g(y(t)) \\
& +\alpha\epsilon_5 g^T(y(t))N_A^TN_Ag(y(t)) + \alpha\epsilon_6 g^T(y(t-\tau(t)))N_B^TN_Bg(y(t-\tau(t))) \\
& +\alpha\beta^{-1}g^T(y(t))DP^{-1}Dg(y(t)) + \alpha\beta g^T(y(t-\tau(t)))W^TPWg(y(t-\tau(t))) \\
& +g^T(y(t))L^{-1}D_0L^{-1}g(y(t)) + g^T(y(t-\tau(t)))L^{-1}D_1L^{-1}g(y(t-\tau(t))) \\
& +\frac{1}{1-\eta}g^T(y(t))Qg(y(t)) - g^T(y(t-\tau(t)))Qg(y(t-\tau(t))) \\
& +\alpha\beta\frac{1}{1-\eta}g^T(y(t))W^TPWg(y(t)) - \alpha\beta g^T(y(t-\tau(t)))W^TPWg(y(t-\tau(t))) \\
& +\frac{1}{1-\eta}g^T(y(t))W^TWg(y(t)) - g^T(y(t-\tau(t)))W^TWg(y(t-\tau(t))).
\end{aligned}$$

Since

$$\begin{aligned}
W^TW &= (B+\Gamma^TT_H)^T(B+\Gamma^TT_H) = B^TB + B^T\Gamma^TT_H + T_H^T\Gamma B + T_H^T\Gamma\Gamma^TT_H \\
&\leq (1+\|\chi\|^2)B^TB + (1+\|\chi\|^2)T_H^TT_H = (1+\|\chi\|^2)(B^TB+T_H^TT_H).
\end{aligned}$$

Therefore,

$$\begin{aligned}
\mathcal{L}V(y(t),t) \leq\ & g^T(y(t))\Big[A^TA + (\epsilon_1^{-1}+\epsilon_2^{-1}+\epsilon_3^{-1})L^{-1}CMM^TCL^{-1} + \epsilon_1 L^{-1}N_C^TN_CL^{-1} + \epsilon_2 N_A^TN_A \\
& +L^{-1}D_0L^{-1} + \epsilon_3 N_B^TN_B + L^{-1}D_1L^{-1}\Big]g(y(t)) \\
& +\alpha g^T(y(t))\Big[-2DCL^{-1} + DA + A^TD + \beta^{-1}DP^{-1}D + (\epsilon_4^{-1}+\epsilon_5^{-1}+\epsilon_6^{-1})DMM^TD
\end{aligned}$$

$$+\epsilon_4 L^{-1}N_C^T N_C L^{-1} + \epsilon_5 N_A^T N_A + \epsilon_6 N_B^T N_B + \frac{1+\|\chi\|^2}{1-\eta}\beta\lambda_M(P)(B^TB + T_H^T T_H)\Big]g(y(t))$$

$$\begin{aligned}
&\leq \lambda_M\Big[A^TA + (\epsilon_1^{-1}+\epsilon_2^{-1}+\epsilon_3^{-1})L^{-1}CMM^TCL^{-1} + \epsilon_1 L^{-1}N_C^TN_CL^{-1} + \epsilon_2 N_A^TN_A \\
&\quad + L^{-1}D_0L^{-1} + \epsilon_3 N_B^TN_B + L^{-1}D_1L^{-1}\Big]g^T(y(t))g(y(t)) - \alpha\lambda_m(-\Omega)g^T(y(t))g(y(t)) \\
&= \Big[\lambda_M\Big[A^TA + (\epsilon_1^{-1}+\epsilon_2^{-1}+\epsilon_3^{-1})L^{-1}CMM^TCL^{-1} + \epsilon_1 L^{-1}N_C^TN_CL^{-1} + \epsilon_2 N_A^TN_A \\
&\quad + L^{-1}D_0L^{-1} + \epsilon_3 N_B^TN_B + L^{-1}D_1L^{-1}\Big] - \alpha\lambda_m(-\Omega)\Big]\|g(y(t))\|_2^2.
\end{aligned}$$

The choice

$$\begin{aligned}
\alpha &> \frac{\lambda_M\Big[A^TA + (\epsilon_1+\epsilon_2+\epsilon_3)L^{-1}CMM^TCL^{-1} + \epsilon_1^{-1}L^{-1}N_C^TN_CL^{-1} + \epsilon_2^{-1}N_A^TN_A\Big]}{\lambda_m(-\Omega)} \\
&\quad + \frac{\lambda_M\Big[L^{-1}D_0L^{-1} + \epsilon_3^{-1}N_B^TN_B + L^{-1}D_1L^{-1}\Big]}{\lambda_m(-\Omega)} > 0
\end{aligned}$$

ensures that $\mathcal{L}V(y(t), t) < 0$, for all $g(y(t)) \neq 0$. Thus, for ensuring negativity of $\mathcal{L}V(y(t), t)$ for any possible state, it suffices to require Ω be a negative definite matrix. This implies that the equilibrium point of system (3) is globally asymptotically stable in the mean square. The proof is completed.

Theorem 3.5. *Assume that there exist matrices $D_0 \geq 0$ and $D_1 \geq 0$ such that*

$$trace[\sigma^T(t, y(t), y(t-\tau(t)))(I+L)\sigma(t, y(t), y(t-\tau(t)))] \leq y^T(t)D_0y(t) + y^T(t-\tau(t))D_1y(t-\tau(t)).$$

System (3)is globally asymptotically stable in the mean square, if the condition (H_1) is satisfied and if the following condition hold:

$$\Omega_1 = 2r^2 - 2r\|A\|_2 - \|W\|_2^2 - \frac{r^2}{1-\eta} - r(\epsilon_4+\epsilon_5+\epsilon_6)\|M\|_2^2 + \epsilon_4^{-1}r\|N_C\|_2^2 + \epsilon_5^{-1}r\|N_A\|_2^2 + \frac{\epsilon_6^{-1}r}{1-\eta}\|N_B\|_2^2 > 0.$$

Proof: We use the following positive definite Lyapunov functional to derive the stability result,

$$\begin{aligned}
V(y,t) &= y^T(t)y(t) + 2\alpha r\sum_{i=1}^n\int_0^{y_i} g_i(s)ds + \alpha r^2\frac{1}{1-\eta}\sum_{i=1}^n\int_{t-\tau_i(t)}^t g_i^T(y_i(s))ds \\
&\quad + \beta\frac{1}{1-\eta}\sum_{i=1}^n\int_{t-\tau_i(t)}^t g_i^T(y_i(s))ds,
\end{aligned}$$

where α and β are some positive constants to be determined later. Let $W = B + \Gamma^T T_H$, by Ito's formula, we can calculate $\mathcal{L}V_1(y(t), t)$,$\mathcal{L}V_2(y(t), t)$,$\mathcal{L}V_3(y(t), t)$ and $\mathcal{L}V_4(y(t), t)$ along the trajectories of the system (3), then we have

$$\begin{aligned}\mathcal{L}V_1(y(t),t) &= -2y^T(t)Cy(t) - 2y^T(t)(\Delta C(t))y(t) + 2y^T(t)Ay(t)\\ &\quad +2y^T(t)(\Delta A(t))y(t) + 2y^T(t)Wg(y(t-\tau(t))) + 2y^T(t)(\Delta B(t))g(y(t-\tau(t)))\\ &\quad +trace[\sigma^T(t,y(t),y(t-\tau(t)))I\sigma(t,y(t),y(t-\tau(t)))]. \end{aligned} \tag{42}$$

Using the inequality technique, we have

$$\begin{aligned}-y^T(t)Cy(t) + 2y^T(t)Ag(y(t)) &= -[C^{\frac{1}{2}}y(t) - C^{-\frac{1}{2}}Ag(y(t))]^T[C^{\frac{1}{2}}y(t) - C^{-\frac{1}{2}}Ag(y(t))]\\ &\quad +g^T(y(t))A^TC^{-1}Ag(y(t)) \end{aligned} \tag{43}$$

$$\begin{aligned}-y^T(t)Cy(t) + 2y^T(t)Wg(y(t-\tau(t))) &= -[C^{\frac{1}{2}}y(t) - C^{-\frac{1}{2}}Wg(y(t-\tau(t)))]^T\\ &\quad [C^{\frac{1}{2}}y(t) - C^{-\frac{1}{2}}Wg(y(t-\tau(t)))]\\ &\quad +g^T(y(t-\tau(t)))W^TC^{-1}Wg(y(t-\tau(t))). \end{aligned} \tag{44}$$

Since the first terms of the equations (43) and (44) are non-positive, we can write the following inequalities

$$-y^T(t)Cy(t) + 2y^T(t)Ag(y(t)) \leq g^T(y(t))A^TC^{-1}Ag(y(t)) \tag{45}$$

$$-y^T(t)Cy(t) + 2y^T(t)Wg(y(t-\tau(t))) \leq g^T(y(t-\tau(t)))W^TC^{-1}Wg(y(t-\tau(t))). \tag{46}$$

From Lemma 3.1, it follows that

$$\begin{aligned}-2y^T(t)(\Delta C(t))y(t) &\leq \epsilon_1^{-1}g^T(y(t))L^{-1}M^TML^{-1}g(y(t))\\ &\quad +\epsilon_1 g^T(y(t))L^{-1}N_C^TN_CL^{-1}g(y(t)) \end{aligned} \tag{47}$$

$$2y^T(t)(\Delta A(t))g(y(t)) \leq \epsilon_2^{-1}g^T(y(t))L^{-1}M^TML^{-1}g(y(t)) + \epsilon_2 g^T(y(t))N_A^TN_Ag(y(t)) \tag{48}$$

$$\begin{aligned}2y^T(t)(\Delta B(t))g(y(t-\tau(t))) &\leq \epsilon_3^{-1}g^T(y(t))L^{-1}M^TML^{-1}g(y(t))\\ &\quad +\epsilon_3 g^T(y(t-\tau(t)))N_B^TN_Bg(y(t-\tau(t))). \end{aligned} \tag{49}$$

From (45)-(49), we get

$$\begin{aligned}\mathcal{L}V_1(y(t),t) &\leq g^T(y(t))A^TC^{-1}Ag(y(t)) + g^T(y(t-\tau(t)))W^TC^{-1}Wg(y(t-\tau(t)))\\ &\quad +\epsilon_1^{-1}g^T(y(t))L^{-1}M^TML^{-1}g(y(t)) + \epsilon_1 g^T(y(t))L^{-1}N_C^TN_CL^{-1}g(y(t))\\ &\quad +\epsilon_2^{-1}g^T(y(t))L^{-1}M^TML^{-1}g(y(t)) + \epsilon_2 g^T(y(t))N_A^TN_Ag(y(t))\\ &\quad +\epsilon_3^{-1}g^T(y(t-\tau(t)))L^{-1}M^TML^{-1}g(y(t)) + \epsilon_3 g^T(y(t-\tau(t)))N_B^TN_Bg(y(t-\tau(t)))\end{aligned}$$

$$+trace[\sigma^T(t,y(t),y(t-\tau(t)))I\sigma(t,y(t),y(t-\tau(t)))]. \quad (50)$$

$$\begin{aligned}\mathcal{L}V_2(y(t),t) &= -2\alpha rg^T(y(t))Cy(t)-2\alpha rg^T(y(t))(\Delta C(t))y(t)+2\alpha rg^T(y(t))Ag(y(t))\\ &\quad +2\alpha rg^T(y(t))(\Delta A(t))g(y(t))+2\alpha rg^T(y(t))Wg(y(t-\tau(t)))\\ &\quad +2\alpha rg^T(y(t))(\Delta B(t))g(y(t-\tau(t)))\\ &\quad +trace[\sigma^T(t,y(t),y(t-\tau(t)))I\sigma(t,y(t),y(t-\tau(t)))]. \quad (51)\end{aligned}$$

From Lemma 3.1, it follows that

$$\begin{aligned}-2\alpha rg^T(y(t))(\Delta C(t))y(t) &\le \epsilon_4^{-1}\alpha rg^T(y(t))M^TMg(y(t))\\ &\quad +\epsilon_4\alpha rg^T(y(t))L^{-1}N_C^TN_CL^{-1}g(y(t)) \quad (52)\end{aligned}$$

$$\begin{aligned}2\alpha rg^T(y(t))(\Delta A(t))g(y(t)) &\le \epsilon_5^{-1}\alpha rg^T(y(t))M^TMg(y(t))\\ &\quad +\epsilon_5\alpha rg^T(y(t))N_A^TN_Ag(y(t)) \quad (53)\end{aligned}$$

$$\begin{aligned}2\alpha rg^T(y(t))(\Delta B(t))g(y(t-\tau(t))) &\le \epsilon_6^{-1}\alpha rg^T(y(t))M^TMg(y(t))\\ &\quad +\epsilon_6\alpha rg^T(y(t-\tau(t)))N_B^TN_Bg(y(t-\tau(t))) \quad (54)\end{aligned}$$

$$\mathcal{L}V_3(y(t),t) \le \frac{\alpha r^2}{1-\eta}g^T(y(t)g(y(t))-\alpha r^2g^T(y(t-\tau(t)))g(y(t-\tau(t))) \quad (55)$$

$$\mathcal{L}V_4(y(t),t) \le \frac{\beta}{1-\eta}g^T(y(t)g(y(t))-\beta g^T(y(t-\tau(t)))g(y(t-\tau(t))). \quad (56)$$

Using the inequality technique, we have

$$\begin{aligned}2\alpha rg^T(y(t))Wg(y(t-\tau(t))) &- \alpha r^2g^T(y(t-\tau(t)))g(y(t-\tau(t)))\\ &= -\alpha[rg(y(t-\tau(t)))-Wg(y(t))]^T[rg(y(t-\tau(t)))-Wg(y(t))]\\ &\quad +\alpha g^T(y(t)W^TWg(y(t)).\end{aligned}$$

Since the first term of the above equation is non-positive, we can write the following inequality,

$$2\alpha rg^T(y(t))Wg(y(t-\tau(t)))-\alpha r^2g^T(y(t-\tau(t)))g(y(t-\tau(t)))\le\alpha g^T(y(t)W^TWg(y(t)). \quad (57)$$

From (42)-(57), it follows that

$$\mathcal{L}V(y(t),t) \le g^T(y(t))A^TC^{-1}Ag(y(t))+g^T(y(t-\tau(t)))W^TC^{-1}Wg(y(t-\tau(t)))$$

$$
\begin{aligned}
& +(\epsilon_1^{-1}+\epsilon_2^{-1}+\epsilon_3^{-1})g^T(y(t))L^{-1}M^TML^{-1}g(y(t))+\epsilon_1 g^T(y(t))L^{-1}N_C^TN_CL^{-1}g(y(t)) \\
& +\epsilon_2 g^T(y(t))N_A^TN_Ag(y(t))+\epsilon_3 g^T(y(t-\tau(t)))N_B^TN_Bg(y(t-\tau(t))) \\
& +y^T(t)D_0y(t)+y^T(t-\tau(t))D_1y(t-\tau(t)) \\
& -2\alpha r g^T(y(t))Cy(t)+2\alpha r g^T(y(t))Ag(y(t))+\alpha g^T(y(t)W^TWg(y(t)) \\
& +\alpha r(\epsilon_4+\epsilon_5+\epsilon_6)g^T(y(t))M^TMg(y(t))+\epsilon_4^{-1}\alpha r g^T(y(t))L^{-1}N_C^TN_CL^{-1}g(y(t)) \\
& +\epsilon_5^{-1}\alpha r g^T(y(t))N_A^TN_Ag(y(t))+\epsilon_6^{-1}\alpha r g^T(y(t-\tau(t)))N_B^TN_Bg(y(t-\tau(t))) \\
& +\frac{\alpha r^2}{1-\eta}g^T(y(t)g(y(t))-\alpha r^2 g^T(y(t-\tau(t)))g(y(t-\tau(t))) \\
& +\frac{\beta}{1-\eta}g^T(y(t)g(y(t))-\beta g^T(y(t-\tau(t)))g(y(t-\tau(t))), \\
\leq\ & -2\alpha r\sum_{i=1}^{n}\frac{c_i}{L_i}g_i^2(y_i(t))+\|A\|_2^2\|C^{-1}\|_2\|g(y(t))\|_2^2+2\alpha r\|A\|_2\|g(y(t))\|_2^2 \\
& +\|W\|_2^2\|C^{-1}\|_2\|g(y(t-\tau(t)))\|_2^2+(\epsilon_1+\epsilon_2+\epsilon_3)\|M\|_2^2\|g(y(t))\|_2^2+\epsilon_1^{-1}\|N_C\|_2^2\|g(y(t))\|_2^2 \\
& +\epsilon_2^{-1}\|N_A\|_2^2\|g(y(t))\|_2^2+\epsilon_3^{-1}\|N_B\|_2^2\|g(y(t-\tau(t)))\|_2^2+\|L^{-1}\|_2^2\|D_0\|_2\|g(y(t))\|_2^2 \\
& +\|L^{-1}\|_2^2\|D_1\|_2\|g(y(t-\tau(t)))\|_2^2+\alpha\|W\|_2^2\|g(y(t))\|_2^2+\alpha r(\epsilon_4+\epsilon_5+\epsilon_6)\|M\|_2^2\|g(y(t))\|_2^2 \\
& +\epsilon_4^{-1}\alpha r\|N_C\|_2^2\|g(y(t))\|_2^2+\epsilon_5^{-1}\alpha r\|N_A\|_2^2\|g(y(t))\|_2^2+\epsilon_6^{-1}\alpha r\|N_B\|_2^2\|g(y(t-\tau(t)))\|_2^2 \\
& +\frac{\alpha r^2}{1-\eta}\|g(y(t))\|_2^2+\frac{\beta}{1-\eta}\|g(y(t))\|_2^2-\beta\|g(y(t-\tau(t)))\|_2^2.
\end{aligned}
$$

Since $r=\min_{1\le i\le n}\left(\frac{c_i}{L_i}\right)$, we have

$$
-2\alpha r\sum_{i=1}^{n}\frac{c_i}{L_i}g_i^2(y_i(t))\leq -2\alpha r^2\sum_{i=1}^{n}g_i^2(y_i(t))=-2\alpha r^2\|g(y(t))\|_2^2.
$$

Let $\beta=\|W\|_2^2\|C^{-1}\|_2+\epsilon_3^{-1}\|N_B\|_2^2+\alpha r\epsilon_6^{-1}\|N_B+\|L^{-1}\|_2^2\|D_1\|_2$.Thus, in the light of the above inequality, $\mathcal{L}V$ can now be written as

$$
\mathcal{L}V(y(t),t)\ \ \leq\ \ -\alpha\Big(2r^2-2r\|A\|_2-\|W\|_2^2-\frac{r^2}{1-\eta}-r(\epsilon_4+\epsilon_5+\epsilon_6)\|M\|_2^2+\epsilon_4^{-1}r\|N_C\|_2^2
$$

$$+\epsilon_5^{-1}r\|N_A\|_2^2+\frac{\epsilon_6^{-1}r}{1-\eta}\|N_B\|_2^2\Big)\|g(y(t))\|_2^2$$

$$+\Big(\|A\|_2^2\|C^{-1}\|_2+\frac{1}{1-\eta}\|W\|_2^2\|C^{-1}\|_2+(\epsilon_1+\epsilon_2+\epsilon_3)\|M\|_2^2+\epsilon_1^{-1}\|N_C\|_2^2$$

$$+\epsilon_2^{-1}\|N_A\|_2^2+\frac{\epsilon_3^{-1}}{1-\eta}\|N_B\|_2^2+\|L^{-1}\|_2^2\|D_0\|_2+\|L^{-1}\|_2^2\|D_1\|_2\Big)\|g(y(t))\|_2^2.$$

Since

$$2r^2-2r\|A\|_2-\|W\|_2^2-\frac{r^2}{1-\eta}-r(\epsilon_4+\epsilon_5+\epsilon_6)\|M\|_2^2+\epsilon_4^{-1}r\|N_C\|_2^2+\epsilon_5^{-1}r\|N_A\|_2^2+\frac{\epsilon_6^{-1}r}{1-\eta}\|N_B\|_2^2>0,$$

the choice

$$\alpha>\frac{N_r}{D_r}>0,$$

ensures that $\mathcal{L}V(y(t),t)<0$, for all $g(y(t))\neq 0$, where

$$\begin{aligned}N_r &= \|A\|_2^2\|C^{-1}\|_2+\frac{1}{1-\eta}\|W\|_2^2\|C^{-1}\|_2+(\epsilon_1+\epsilon_2+\epsilon_3)\|M\|_2^2+\epsilon_1^{-1}\|N_C\|_2^2\\ &\quad+\epsilon_2^{-1}\|N_A\|_2^2+\frac{\epsilon_3^{-1}}{1-\eta}\|N_B\|_2^2+\|L^{-1}\|_2^2\|D_0\|_2+\|L^{-1}\|_2^2\|D_1\|_2\end{aligned}$$

and

$$\begin{aligned}D_r &= 2r^2-2r\|A\|_2-\|W\|_2^2-\frac{r^2}{1-\eta}-r(\epsilon_4+\epsilon_5+\epsilon_6)\|M\|_2^2+\epsilon_4^{-1}r\|N_C\|_2^2\\ &\quad+\epsilon_5^{-1}r\|N_A\|_2^2+\frac{\epsilon_6^{-1}r}{1-\eta}\|N_B\|_2^2.\end{aligned}$$

Thus, for ensuring negativity of $\mathcal{L}V(y(t),t)$ for any possible state, it suffices to require Ω_1 be a positive definite matrix. This implies that the equilibrium point of system (3) is globally asymptotically stable in the mean square. The proof is completed.

Remark 3.6. *In* [12], *stability of equilibrium point of High-order Hopfield neural networks with time varying delays has been considered by means of Lyapunov functional and LMI techniques. We extend this technique to study the stochastic high-order neural networks with time-varying uncertain parameters. In view of this, our results in this chapter extend the results in* [12].

Remark 3.7. *In* [20], *the authors studied the global stability of stochastic high-order neural networks with discrete and distributed delays. Similarly in [21], the authors studied stability results of stochastic high-order Markovian jumping neural networks with mixed time delays. It should be noted that the uncertain stochastic neural network studied in this chapter is time-varying delays. Therefore, our results and those established in* [20, 21] *are complementary each other.*

4. An illustrative example.

The effectiveness of the theories will be demonstrated through the following example. Consider the following high-order stochastic Hopfield neural network with time varying delays

$$dy_i(t) = [-c_i y_i(t) + \sum_{j=1}^{2} a_{ij} g_j(y_j(t)) + \sum_{j=1}^{2} b_{ij} g_j(y_j(t-\tau_j(t))) + \sum_{j=1}^{2}\sum_{j=1}^{2} T_{ijl} g_j(y_j(t-\tau_j(t))) + J_i]dt$$

$$+\sigma(t, y(t), y(t-\tau(t)))dw(t) \tag{58}$$

where $g_1(y_1) = tanh(0.95y_1)$, $g_2(y_2) = tanh(y_2)$,

$$\sigma(t, y(t), y(t-\tau(t))) = [0.5y(t) + 0.5y(t-\tau(t)), 0.4y(t) + 0.4y(t-\tau(t))], \quad \eta = 0.4,\ J_1 = 1.5,\ J_2 = 2,$$

$$C = \begin{bmatrix} 4.5 & 0 \\ 0 & 4.5 \end{bmatrix}, A = \begin{bmatrix} 0.05 & 0.14 \\ 0.20 & 0.31 \end{bmatrix}, B = \begin{bmatrix} 0.09 & 0.25 \\ 0.21 & 0.45 \end{bmatrix}, T_1 = \begin{bmatrix} 0.05 & 0.14 \\ -0.06 & 0.05 \end{bmatrix},$$

$$T_2 = \begin{bmatrix} 0.29 & 0.10 \\ 0.23 & -0.14 \end{bmatrix}, T_3 = \begin{bmatrix} -0.23 & 0.07 \\ 0.09 & -0.02 \end{bmatrix}, M = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.03 \end{bmatrix}, N_C = \begin{bmatrix} 0.06 & 0 \\ 0 & 0.06 \end{bmatrix},$$

$$N_A = \begin{bmatrix} 0.03 & 0 \\ 0 & 0.03 \end{bmatrix}, N_B = \begin{bmatrix} 0.04 & 0 \\ 0 & 0.04 \end{bmatrix}, D_0 = D_1 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.32 \end{bmatrix}.$$

Thus we have $\mathcal{L} = I$, $\|\mathcal{X}\|_2 = 1$. Now, solving the LMI in Theorem 3.2, using Matlab LMI Control toolbox, we get the following feasible solution

$$S_1 = 10^3 \times \begin{bmatrix} 2.0438 & -0.0233 \\ -0.0233 & 1.9622 \end{bmatrix} \qquad S_2 = \begin{bmatrix} 132.3299 & -191.3967 \\ -191.3967 & 494.7130 \end{bmatrix},$$

$\epsilon_1 = 5.5014$, $\epsilon_2 = 0.2838$, $\epsilon_3 = 21.7583$

It follows from Theorem 3.2 that the equilibrium point of the system (58) is globally asymptotically stable in the mean square.

Now, solving the LMI in Theorem 3.3, using Matlab LMI Control toolbox, we get the following feasible solution

$$S_1 = \begin{bmatrix} 0.5141 & -0.1631 \\ -0.1631 & 0.5591 \end{bmatrix}, \qquad \epsilon_1 = 1.0550,$$

$$\epsilon_2 = 4.3317, \quad \epsilon_3 = 4.0591.$$

Therefore, from Theorem 3.3 that the equilibrium point of the system (58) is globally asymptotically stable in the mean square.

Now we let $D_0 = D_1 = \begin{bmatrix} 1.0 & 0 \\ 0 & 0.64 \end{bmatrix}$. Again solving the LMI in Theorem 3.4, using Matlab LMI Control toolbox, we get the following feasible solution

$$\beta = 0.5690, \quad \epsilon_4 = 2.0315, \quad \epsilon_5 = 8.7405, \quad \epsilon_6 = 5.0297.$$

Therefore, from Theorem 3.4 that the equilibrium point of the system (58) is globally asymptotically stable in the mean square.

5. References

P. Baldi, Neural networks, orientations of the hypercube, and algebric threshold functions, *IEEE Trans. Inform. Theory*, 34 (1988) 523–530.

T. Burton, *Stability and periodic solution of ordinary differential equation and functional differential equations*, Orlanda, FL: Academic; 1985.

Z. Chen, D. Zhao, J. Ruan, Dynamicanalysis of high-order Cohen-Grossberg neural networks with time delays, *Chaos, Solitons, Fractals*, 32 (2007) 1538–1546.

L.O. Chua, L. Yang, Cellular neural networks: theory, *IEEE Trans. Circuits. Syst.*, 35 (1988) 1257–1272.

H. Gu, H. Jiang, Z. Teng, Stability and periodicity in high-order neural networks with impulsive effects, *Nonlin. Anal:TMA*,68 (2008) 3186–3200.

Z.H. Guan, D.B. Sun, J.J. Shen, Qualitative analysis of high-order Hopfield neural networks, *Acta Electron. Sin.*, 28 (2000) 77–80.

J.K. Hale, *Theory of functional differential equations*, New York: Springer-Verlag; 1977.

D.W.C. Ho, J. Liang, J. Lam, Global exponential stability of impulsive high-order BAM neural networks with time-varying delays, *Neural Networks*, 19 (2006) 1581–1590.

J. J. Hopfield, Neural networks and physical systems with emergement collective computational abilities, *Proc. Natl. Acad. Sci.*, 79 (1982) 2554–2558.

J. J. Hopfield, Neurons with graded response have collective computational properties like those of two-state of neurons, *Proc. Natl. Acad. Sci.*, 81 (1984) 3088–3092.

X. Liu, K.L. Teo, B. Xu, Exponential stability of impulsive high-oreder Hopfield type neural networks with time varying delays, *IEEE Trans. Neural Networks*, 16 (2005) 1329–1339.

X. Lou, B. Cui, Novel global stability criteria for high-order Hopfield-type neural networks with time-varying delays, *J. Math. Anal. Appl.*, 330 (2007) 144–158.

C.M. Marcus, R.M. Westervelt, Stability of analog neural networks with delay, *Phys. Rev. A*, 39 (1989) 347–359.

R. McEliece, E. Posner, E. Rodemich, S. Venkatesh, The capacity of the Hopfield associative memory, *IEEE Trans. Inform. Theory*, 33 (1987) 461–482.

F. Qiu, B.T. Cui, W. Wu, Global exponential stability of high order recurrent neural network with time varying delays, *Appl. Math. Model.*, in press.

B. Xu, X. Liu, X. Liao, Global exponential stability of high-order Hopfield-type neural networks, *Appl. Math. Comput.*, 174 (2006) 98–116.

B. Xu, X. Liu, X. Liao, Global asymptoticstabilit y of high-order Hopfield type neural networks with time delays, *Comput. Math. Appl.*, 45 (2003) 1729–1737.

B. Xu, Q. Wang, X. Liao, stability analysis of high-order Hopfield neural networks with uncertainty, *Neurocomputing*, 71 (2008) 508–512.

Y. Wang, Global exponential stability analysis of bidirectional associative memory neural networks with timevarying delays, *Nonlin. Anal:RWA*, in press.

Z. Wang, J. Fang,X. Liu, Global stability of stochastic high-order neural networks with discrete and distributed delays, *Chaos, Solitons, Fractals*, 36 (2008) 388–396.

Y. Liu, Z. Wang, X. Liu, An LMI approach to stability analysis of stochastic high-order Markovian jumping neural networks with mixed time delays, *Nonlin. Anal: Hybrid Systems*,2 (2008) 110–120.

B. Zhang, S. Xu, Y. Li, Y. Chu, On global exponential stability of high-order neural networks with time-varying delays, *Phys. lett. A*, 366 (2007) 69-78.

Dynamics of Two-Dimensional Discrete-Time Delayed Hopfield Neural Networks

Eva Kaslik and Ştefan Balint
Dept. of Mathematics and Computer Science
West University of Timişoara
Romania

1. Introduction

This chapter is devoted to the analysis of the complex dynamics exhibited by two-dimensional discrete-time delayed Hopfield-type neural networks.
Since the pioneering work of (Hopfield, 1982; Tank & Hopfield, 1986), the dynamics of continuous-time Hopfield neural networks have been thoroughly analyzed. In implementing the continuous-time neural networks for practical problems such as image processing, pattern recognition and computer simulation, it is essential to formulate a discrete-time system which is a version of the continuous-time neural network. However, discrete-time counterparts of continuous-time neural networks have only been in the spotlight since 2000.
One of the first problems that needed to be clarified, concerned the discretization technique which should be applied in order to obtain a discrete-time system which preserves certain dynamic characteristics of the continuous-time system. In (Mohamad & Gopalsamy, 2000) a semi-discretization technique has been presented for continuous-time Hopfield neural networks, which leads to discrete-time neural networks which faithfully preserve some characteristics of the continuous-time network, such as the steady states and their stability properties.
In recent years, the theory of discrete-time dynamic systems has assumed a greater importance as a well deserved discipline. In spite of this tendency of independence, there is a striking similarity or even duality between the theories of continuous and discrete dynamic systems. Many results in the theory of difference equations have been obtained as natural discrete analogs of corresponding results from the theory of differential equations. Nevertheless, the theory of difference equations is a lot richer than the corresponding theory of differential equations. For example, a simple difference equation resulting from a first order differential equation may exhibit chaotic behavior which can only happen for higher order differential equations. This is the reason why, when studying discrete-time counterparts of continuous neural networks, important differences and more complicated behavior may also be revealed.
The analysis of the dynamics of neural networks focuses on three directions: discovering equilibrium states and periodic or quasi-periodic solutions (of fundamental importance in biological and artificial systems, as they are associated with central pattern generators (Pasemann et al., 2003)), establishing stability properties and bifurcations (leading to the

discovery of periodic solutions), and identifying chaotic behavior (with valuable applications to practical problems such as optimization (Chen & Aihara, 1995, 1997, 2001; Chen & Shih, 2002), associative memory (Adachi & Aihara, 1997) and cryptography (Yu & Cao, 2006)).
We refer to (Guo & Huang, 2004; Guo et al., 2004) for the study of the existence of periodic solutions of discrete-time Hopfield neural networks with delays and the investigation of exponential stability properties.
In (Yuan et al., 2004, 2005) and in the most general case, in (He & Cao, 2007), a bifurcation analysis of two dimensional discrete neural networks without delays has been undertaken. In (Zhang & Zheng, 2005, 2007), the bifurcation phenomena have been studied, for the case of two- and n-dimensional discrete neural network models with multi-delays obtained by applying the Euler method to a continuous-time Hopfield neural network with no self-connections. In (Kaslik & Balint, 2007a-b), a bifurcation analysis for discrete-time Hopfield neural networks of two neurons with self-connections has been presented, in the case of a single delay and of two delays. In (Guo et al., 2007), a generalization of these results was attempted, considering three delays; however, only two delays were considered independent (the third one is a linear combination of the first two) and the analysis can be reduced to the one presented in (Kaslik & Balint, 2007a).
The latest results concerning chaotic dynamics in discrete-time delayed neural networks can be found in (Huang & Zou, 2005) and (Kaslik & Balint, 2007c).
A general discrete-time Hopfield-type neural network of two neurons with finite delays is defined by:

$$\begin{cases} x_{n+1} = a_1 x_n + T_{11} g_1(x_{n-k_{11}}) + T_{12} g_2(y_{n-k_{12}}) \\ y_{n+1} = a_2 y_n + T_{21} g_1(x_{n-k_{21}}) + T_{22} g_2(y_{n-k_{22}}) \end{cases} \quad \forall n \geq \max(k_{11}, k_{12}, k_{21}, k_{22}) \tag{1}$$

In this system $a_i \in (0,1)$ are the internal decays of the neurons, $T = (T_{ij})_{2\times 2}$ is the interconnection matrix, $g_i : \mathbf{R} \to \mathbf{R}$ represent the neuron input-output activations and $k_{ij} \in \mathbf{N}$ represent the delays. The reason for incorporating delays into the model equations of the network is that, in practice, due to the finite speeds of the switching and transmission of signals in a network, time delays unavoidably exist in a working network.
In order to insure that delays are present, we consider $\max(k_{11}, k_{12}, k_{21}, k_{22}) > 0$. The non-delayed case was extensively studied in (He & Cao, 2007). In the followings, we will denote $k_1 = \max(k_{11}, k_{21})$ and $k_2 = \max(k_{12}, k_{22})$.

We will suppose that the activation functions g_i are of class C^3 in a neighborhood of 0 and that $g_i(0) = 0$. In the followings, let $g : \mathbf{R}^2 \to \mathbf{R}^2$ be the function given by $g(x,y) = (g_1(x), g_2(y))^T$ and

$$B = TDg(0) = \begin{pmatrix} T_{11} g_1'(0) & T_{12} g_2'(0) \\ T_{21} g_1'(0) & T_{22} g_2'(0) \end{pmatrix} = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

We use the notations $2\beta = b_{11} + b_{22} = \mathrm{tr}(B)$ and $\delta = b_{11}b_{22} - b_{12}b_{21} = \det(B)$.
The aim of this chapter is to present a complete stability and bifurcation analysis in a neighborhood of the null solution of (1), choosing the characteristic parameters (β, δ) for the

system. Considering equal internal decays $a_1 = a_2 = a$ and delays satisfying $k_{11} + k_{22} = k_{12} + k_{21}$, two complementary situations are discussed:

- $k_{11} = k_{22}$
- $k_{11} \neq k_{22}$ (with the supplementary hypothesis $b_{11} = b_{22}$)

To the best of our knowledge, these are generalizations of all cases considered so far in the existing literature. This analysis allows the description of the stability domain of the null solution and the types of bifurcation occurring at its boundary, in terms of the characteristic parameters. By applying the center manifold theorem and the normal form theory, the Neimark-Sacker bifurcations are analyzed. A numerical example is presented to substantiate the theoretical findings. Moreover, the numerical example shows that the dynamics become more and more complex as the characteristic parameters leave the stability domain, eventually leading to the installation of chaotic behavior. The route from stability towards chaos passes through several stages of strange attractors and periodic solutions.

2. Preliminary results

We will start by giving two results that have particular importance for the bifurcation analysis to follow, namely for the study of the distribution of the roots of the characteristic polynomial associated to sysem (1) with respect to the unit circle.
The first result concerns the distribution of the roots of a polynomial function with respect to the unit circle, and can be proved using Rouché's theorem.
Proposition 1. (see (Zhang & Zheng, 2005, 2007)) Suppose that $S \subset \mathbf{R}$ is a compact and connected set, and the polynomial $P(\lambda,\alpha) = \lambda^m + p_1(\alpha)\lambda^{n-1} + p_2(\alpha)\lambda^{n-2} + ... + p_m(\alpha)$ is continuous on $\mathbf{C} \times S$. Then, as the parameter α varies, the sum of the order of the zeros of $P(\lambda,\alpha)$ out of the unit circle, i.e. $\text{card}(\{\lambda \in \mathbf{C} : P(\lambda,\alpha) = 0, |\lambda| > 1\})$, can change only if a zero appears on or crossed the unit circle. ■
The second result concerns the existence of the roots of a special equation which plays an important role in the analysis of the characteristic polynomial associated to system (1).
Proposition 2. (see (Kaslik & Balint, 2007b)) Let be $m \geq 0$, $[m]$ the integer part of m and $a \in (0,1)$. The equation

$$\sin(m+1)\phi - a \sin m\phi = 0 \tag{2}$$

has exactly $[m]+2$ solutions in the interval $[0,\pi]$. More precisely:

- $\phi_0 = 0$ is a solution;
- if $m \geq 1$, there is one solution ϕ_j in every interval $\left(\frac{(2j-1)\pi}{2m+1}, \frac{j\pi}{m+1}\right) \subset \left(\frac{(j-1)\pi}{m}, \frac{j\pi}{m}\right)$, $j \in \{1,2,...,[m]\}$;
- if $m \in \mathbf{N}$ then $\phi_{[m]+1} = \pi$ is a solution and if $m \notin \mathbf{N}$ then there is one solution $\phi_{[m]+1} \in \left(\frac{[m]\pi}{m}, \pi\right)$. ■

3. Stability and bifurcation analysis

We transform system (1) into the following system of $k_1 + k_2 + 2$ equations without delays:

$$\begin{cases} x_{n+1}^{(0)} = a_1 x_n^{(0)} + T_{11} g_1(x_n^{(k_{11})}) + T_{12} g_2(y_n^{(k_{12})}) \\ x_{n+1}^{(j)} = x_n^{(j-1)} \qquad \forall j = \overline{1,k_1} \\ y_{n+1}^{(0)} = a_2 y_n^{(0)} + T_{21} g_1(x_n^{(k_{21})}) + T_{22} g_2(y_n^{(k_{22})}) \\ y_{n+1}^{(j)} = y_n^{(j-1)} \qquad \forall j = \overline{1,k_2} \end{cases} \qquad \forall n \in \mathbb{N} \tag{3}$$

where $x^{(j)} \in \mathbb{R}$, $j = \overline{0,k_1}$ and $y^{(j)} \in \mathbb{R}$, $j = \overline{0,k_2}$.

Let be the function $F : \mathbb{R}^{k_1+k_2+2} \to \mathbb{R}^{k_1+k_2+2}$ given by the right hand side of system (3). The jacobian matrix of system (3) at the fixed point $\overline{0} \in \mathbb{R}^{k_1+k_2+2}$ is $\hat{A} = DF(\overline{0})$.

The following characteristic equation is obtained:

$$(z - a_1 - b_{11} z^{-k_{11}})(z - a_2 - b_{22} z^{-k_{22}}) - b_{12} b_{21} z^{-(k_{12}+k_{21})} = 0 \tag{4}$$

Studying the stability and bifurcations occurring at the origin in system (1) reduces to the analysis of the distribution of the roots of the characteristic equation (4) with respect to the unit circle. The difficulty of this analysis is due to the large number of parameters appearing in the characteristic equation.

In the followings, considering equal internal decays $a_1 = a_2 = a$ and delays satisfying $k_{11} + k_{22} = k_{12} + k_{21}$, we will analyze the roots of equation (4) in two particular situations, depicting information about the stability and bifurcations occurring at the origin in system (1).

3.1 Situation 1: $k_{11} = k_{22}$

We will denote $k_{11} = k_{22} = k$ and therefore, we have $k_{12} + k_{21} = 2k$.

A particular case of this situation is the one studied in (Kaslik & Balint, 2007a), where in addition, it was considered that $k_{12} = k_{21} = k$, that is, all four delays are equal. Another particular case of this situation is the one analyzed in (Guo et al., 2007), considering the supplementary hypothesis $b_{11} = b_{22}$ (but without assuming that all four delays are equal).

In this situation, the characteristic equation (4) can be written as:

$$z^{2k}(z-a)^2 - 2\beta z^k (z-a) + \delta = 0 \tag{5}$$

The distribution of the roots of the characteristic equation (5) has been thoroughly analyzed in (Kaslik & Balint, 2007a). This analysis provides us with the following results concerning the stability and bifurcations occurring at the origin in system (1):

Considering the following notations and associated basic results:

- ϕ_1 the unique solution of the equation $\sin(k+1)\phi - a \sin k\phi = 0$ from the interval $(0, \frac{\pi}{k+1})$;
- the strictly decreasing function $c : [0, \phi_1] \to \mathbb{R}$, $c(\theta) = \cos(k+1)\theta - a \cos k\theta$;
- $c(\phi_1) = -(a^2 + 1 - 2a \cos\phi_1)^{\frac{1}{2}} < 0$;

- the strictly decreasing function $U:[c(\phi_1),1-a]\to(0,\infty)$ defined by $U(\beta)=1+a^2-2a\cos(c^{-1}(\beta))$;
- the function $\lambda_0:\mathrm{R}\to\mathrm{R}$, $\lambda_0(\beta)=2(1-a)\beta-(1-a)^2$;
- the function $\lambda_1:\mathrm{R}\to\mathrm{R}$, $\lambda_1(\beta)=2c(\phi_1)\beta-c(\phi_1)^2$;
- the function $L:[c(\phi_1),1-a]\to\mathrm{R}$, $L(\beta)=\max(\lambda_j(\beta)/j\in\{0,1\})$;
- $\beta_0=\frac{1}{2}[c(\phi_1)+1-a]$;

The following theorem holds:

Theorem 1. The null solution of (1) is asymptotically stable if and only if β and δ satisfy the following inequalities:

$$c(\phi_1)<\beta<1-a \quad \text{and} \quad L(\beta)<\delta<U(\beta). \tag{6}$$

On the boundary of the set $D_S=\{(\beta,\delta)\in\mathrm{R}^2: c(\phi_1)<\beta<1-a \text{ and } L(\beta)<\delta<U(\beta)\}$ the following bifurcation phenomena causing the loss of asymptotical stability of the null solution of (1) take place:

i. Let be $\beta\in(\beta_0,1-a)$. When $\delta=L(\beta)=\lambda_0(\beta)$ system (1) has a Fold bifurcation at the origin.
ii. Let be $\beta\in(c(\phi_1),\beta_0)$. When $\delta=L(\beta)=\lambda_1(\beta)$ a Neimark-Sacker bifurcation occurs in system (1), i.e. a unique closed invariant curve bifurcates from the origin near $\delta=\lambda_1(\beta)$.
iii. Let be $\beta\in(c(\phi_1),1-a)$. When $\delta=U(\beta)$, system (1) has a Neimark-Sacker bifurcation at the origin. That is, system (1) has a unique closed invariant curve bifurcating from the origin near $\delta=U(\beta)$.
iv. For $\beta=\beta_0$ and $\delta=L(\beta_0)=c(\phi_1)(1-a)$ a Fold-Neimark-Sacker bifurcation occurs at the origin in system (1).
v. For $\beta=c(\phi_1)$ and $\delta=c(\phi_1)^2$, the null solution of (1) is a double Neimark-Sacker bifurcation point.
vi. For $\beta=(1-a)$ and $\delta=(1-a)^2$, the system (1) has a strong 1:1 resonant bifurcation at the origin. ■

The set D_S given by Theorem 1 is the stability domain of the null solution of (1) with respect to the characteristic parameters β and δ.

3.2 Situation 2: $k_{11}\neq k_{22}$ and $b_{11}=b_{22}$

A particular case of this situation has been studied in (Kaslik & Balint, 2007b), where in addition, it was considered that $k_{11}=k_{21}$ and $k_{12}=k_{22}$.

In this situation, the characteristic equation (4) can be written as:

$$z^{k_{11}+k_{22}}(z-a)^2-\beta z^{k_{22}}(z-a)-\beta z^{k_{11}}(z-a)+\delta=0 \tag{7}$$

This equation is the same as the one obtained and analyzed in (Kaslik & Balint, 2007b). The conclusions of this analysis will be presented below.

First, a list of notations will be introduced and some mathematical results will be presented, which can be proved using basic mathematical tools:

- $m = \frac{1}{2}(k_{11} + k_{22})$ and $l = \frac{1}{2}|k_{11} - k_{22}|$; remark: $l \ge \frac{1}{2}$, $m > 1$;
- $S_1 = \{\phi_0 = 0, \phi_1, \phi_2, \ldots, \phi_{[m]+1}\}$ the set of all solutions of the equation (2) from the interval $[0,\pi]$;
- $S_2 = \{\psi_j = \frac{(2j-1)\pi}{2l} / j \in \{1,2,\ldots,\left[\frac{2l+1}{2}\right]\}$;
- $\theta_1 = \min(\phi_1, \psi_1)$;
- the function $c:[0,\pi] \to R$, $c(\theta) = \cos(m+1)\theta - a\cos m\theta$;
- the function $s:[0,\pi] \to R$, $s(\theta) = \sin(m+1)\theta - a\sin m\theta$;
- the strictly decreasing function $h:[0,\theta_1) \to R$, $h(\theta) = c(\theta)\sec(l\theta)$;
- $\alpha = \lim\limits_{\theta \to \theta_1} h(\theta) = \begin{cases} c(\phi_1)\sec(l\phi_1) < 0 & \text{if } \phi_1 < \psi_1 \\ -\infty & \text{if } \phi_1 \ge \psi_1 \end{cases}$
- $h^{-1}:(\alpha, 1-a] \to [0,\theta_1)$ the inverse of the function h;
- the strictly decreasing function $U:(\alpha, 1-a] \to (0,\infty)$, $U(\beta) = 1 + a^2 - 2a\cos(h^{-1}(\beta))$;
- the functions $\lambda_j : R \to R$, $\lambda_j(\beta) = 2c(\phi_j)\cos(l\phi_j)\beta - c(\phi_j)^2$;
- the function $L:(\alpha, 1-a] \to R$, $L(\beta) = \max\left(\lambda_j(\beta) / j \in \{0,1,\ldots,[m]+1\}\right)$;
- β_{ij} the solution of the equation $\lambda_i(\beta) = \lambda_j(\beta)$, $i \ne j$;
- $\beta_0 = \max(\beta_{0j} / j \in \{1,2,\ldots,[m]+1\}, \beta_{0j} < 0)$;
- remark: $L(\beta) = \lambda_0(\beta) = 2(1-a)\beta - (1-a)^2$ for any $\beta \in [\beta_0, 1-a]$;
- if the equation $U(\beta) = L(\beta)$ has some roots in the interval (α, β_0), then β_1 is the largest of these roots; otherwise, $\beta_1 = \alpha$.

We will consider the following two cases:
(c1) At least one of the delays k_{11} or k_{22} is odd.
(c2) Both delays k_{11} and k_{22} are even.

Theorem 2. The null solution of (1) is asymptotically stable if β and δ satisfy the following inequalities:

$$\beta_1 < \beta < 1-a \quad \text{and} \quad L(\beta) < \delta < U(\beta). \tag{8}$$

On the boundary of the set $D_S = \{(\beta,\delta) \in R^2 : \beta_1 < \beta < 1-a \text{ and } L(\beta) < \delta < U(\beta)\}$ the following bifurcation phenomena causing the loss of asymptotical stability of the null solution of (1) take place:

i. Let be $\beta \in (\beta_1, 1-a)$. When $\delta = U(\beta)$, system (1) has a Neimark-Sacker bifurcation at the origin. That is, system (1) has a unique closed invariant curve bifurcating from the origin near $\delta = U(\beta)$.
ii. Let be $\beta \in (\beta_1, \beta_0)$ such that the function L is differentiable at β. When $\delta = L(\beta)$:

(c1) system (1) has a Neimark-Sacker bifurcation at the origin.
(c2) system (1) has a Flip or a Neimark-Sacker bifurcation at the origin.

iii. Let be $\beta \in (\beta_0, 1-a)$. When $\delta = L(\beta) = 2(1-a)\beta - (1-a)^2$ system (1) has a Fold bifurcation at the origin.

iv. For $\beta = (1-a)$ and $\delta = (1-a)^2$, system (1) has a strong $1:1$ resonant bifurcation at the origin.

v. For $\beta = \beta_0$ and $\delta = L(\beta_0) = 2(1-a)\beta_0 - (1-a)^2$, system (1) has a Fold-Neimark-Sacker bifurcation at the origin.

vi. For $\beta = \beta_1$ and $\delta = U(\beta_1)$:
(c1) system (1) has a double Neimark-Sacker bifurcation at the origin.
(c2) system (1) has a double Neimark-Sacker or a Flip-Neimark-Sacker bifurcation at the origin.

vii. If there exists $\beta^* \in (\beta_1, \beta_0)$ such that the function L is not differentiable at β^*, then for $\beta = \beta^*$ and $\delta = L(\beta^*)$:
(c1) system (1) has a double Neimark-Sacker bifurcation at the origin.
(c2) system (1) has a double Neimark-Sacker or a Flip-Neimark-Sacker bifurcation at the origin. ■

We underline that Theorems 1 and 2 completely characterize the stability domain (in the (β, δ)-plane) of the null solution of (1) and the bifurcations occurring at its boundary, in the considered situations.

4. Direction and stability of Neimark-Sacker bifurcations

Let be the function $F : \mathbb{R}^{k_1+k_2+2} \to \mathbb{R}^{k_1+k_2+2}$ given by the right hand side of system (3). Let be the operators $\hat{A} = DF(\overline{0})$, $\hat{B} = D^2F(\overline{0})$ and $\hat{C} = D^3F(\overline{0})$.

In the cases ii. and iii. of Theorem 1 and i. and ii. of Theorem 2, Neimark-Sacker bifurcations occur at the origin in system (1). That is, matrix $\hat{A}$ has a simple pair $(z, \overline{z})$ of eigenvalues on the unit circle, such that z is not a root of order $1,2,3,4$ of the unity.

The restriction of system (3) to its two dimensional center manifold at the critical parameter values can be transformed into the normal form written in complex coordinates (see (Kuznetsov, 2004)):

$$w \mapsto zw(1 + \frac{1}{2} d |w|^2) + O(|w|^4), \qquad w \in \mathbb{C} \tag{9}$$

with

$$d = \overline{z}\langle p, \hat{C}(q,q,\overline{q}) + 2\hat{B}(q,(I-\hat{A})^{-1}\hat{B}(q,\overline{q})) + \hat{B}(\overline{q},(z^2I-\hat{A})^{-1}\hat{B}(q,q))\rangle$$

where $\hat{A}q = zq$, $\hat{A}^T p = \overline{z}p$ and $\langle p,q\rangle = 1$ (with $\langle p,q\rangle = \overline{p}^T q$)

Direct computations provide the following result:

Proposition 3. Suppose that $k_{11}+k_{22}=k_{12}+k_{21}$ and $a_1=a_2=a$. Consider $P(z)=[z^{k_{11}}(z-a)-b_{11}][z^{k_{22}}(z-a)-b_{22}]$. The vectors q and p of $\mathbb{C}^{k_1+k_2+2}$ which verify

$$\hat{A}q = zq \quad ; \quad \hat{A}^T p = \bar{z}p \quad ; \quad \langle p,q\rangle = 1$$

are given by:

$$q = (z^{k_1}q_1, z^{k_1-1}q_1, \dots, zq_1, q_1, z^{k_2}q_2, z^{k_2-1}q_2, \dots, zq_2, q_2)^T$$

$$p = (p_1, (\bar{z}-a)p_1, \bar{z}(\bar{z}-a)p_1, \dots, \bar{z}^{k_1-1}(\bar{z}-a)p_1, p_2, (\bar{z}-a)p_2, \bar{z}(\bar{z}-a)p_2, \dots, \bar{z}^{k_2-1}(\bar{z}-a)p_2)^T$$

where $q_1 = z^{k_{22}}(z-a)-\beta$; $q_2 = b_{21}$; $\bar{p}_1 = \frac{1}{P'(z)}$; $\bar{p}_2 = \frac{z^{k_{11}}(z-a)-\beta}{b_{21}P'(z)}$.■

The following result gives us information about the direction and stability of Neimark-Sacker bifurcations.

Proposition 4. (see (Kuznetsov, 2004)) The direction and stability of the Neimark-Sacker bifurcation is determined by the sign of $\mathrm{Re}(d)$. If $\mathrm{Re}(d)<0$ then the bifurcation is supercritical, i.e. the closed invariant curve bifurcating from the origin is asymptotically stable. If $\mathrm{Re}(d)>0$, the bifurcation is subcritical, i.e. the closed invariant curve bifurcating from the origin is unstable. ■

5. Example

In the following example, we will consider the delays $k_{11}=1$, $k_{22}=5$, $k_{12}=4$ and $k_{21}=2$. We will also choose $a=0.5$ and $b_{11}=b_{22}=\beta$. In this case, using Mathematica, we compute:

- $S_1 = \{0, 0.667561, 1.44928, 2.28703, \pi\}$ (rad), $S_2 = \{\frac{\pi}{4}, \frac{3\pi}{4}\}$;
- $\theta_1 = \phi_1 = 0.667561$ (rad), $\alpha = -2.91934$, $\beta_1 = -0.723816$, $\beta_0 = -0.162831$;
- $\beta^* = -0.380779$.

The bifurcations occurring at the boundary of D_S (provided by Theorem 2) are:

- For $\beta \in (\beta_1, \beta_0)$ and $\delta = U(\beta)$ a Neimark-Sacker bifurcation occurs, with the multipliers $e^{\pm ih^{-1}(\beta)}$;
- For $\beta \in (\beta_0, 1-a)$ and $\delta = \lambda_0(\beta) = 2(1-a)\beta-(1-a)^2$ a Fold bifurcation occurs;
- For $\beta \in (\beta_1, \beta^*)$ and $\delta = L(\beta) = \lambda_2(\beta)$ a Neimark-Sacker bifurcation occurs, with the multipliers $e^{\pm i\phi_2}$;
- For $\beta \in (\beta^*, \beta_0)$ and $\delta = L(\beta) = \lambda_1(\beta)$ a Neimark-Sacker bifurcation occurs, with the multipliers $e^{\pm i\phi_1}$;
- For $\beta = 1-a$ and $\delta = (1-a)^2$ a 1:1 resonant bifurcation occurs;
- For $\beta = \beta_1$ and $\delta = U(\beta_1)$ a double Neimark-Sacker bifurcation occurs;

- For $\beta = \beta_0$ and $\delta = L(\beta_0) = \lambda_0(\beta_0)$ a Fold-Neimark-Sacker bifurcation occurs.
- For $\beta = \beta^*$ and $\delta = L(\beta^*)$ a double Neimark-Sacker bifurcation occurs.

The stability domain in the (β,δ)-plane for this network is the one presented in Figure 1.

More precisely, we consider the delayed discrete-time Hopfield neural network:

$$\begin{cases} x_{n+1} = 0.5x_n + \beta\tanh(x_{n-1}) - \sin(y_{n-4}) \\ y_{n+1} = 0.5y_n + (\delta - \beta^2)\tanh(x_{n-2}) + \beta\sin(y_{n-5}) \end{cases} \quad \forall n \geq 5 \tag{10}$$

Choosing $\beta = -0.25$, we obtain that the origin is asymptotically stable if $\delta \in (-0.385082, 0.324255)$ and supercritical Neimark-Sacker bifurcations occur at $\delta = L(\beta) = -0.385082$ and $\delta = U(\beta) = 0.324255$ respectively (see Figures 4-5). The bifurcation diagram for $\delta \in (-2.5, 2.5)$ is presented in Figure 2 and the values of the Largest Lyapunov Characteristic Exponent are presented in Figure 3. It can be seen that as δ leaves the stability domain D_S, the dynamics in a neighborhood of the origin become more and more complex, eventually leading to the occurrence of chaotic behavior. The phase portraits presented in Figures 6-7 sillustrate the changes which appear on the route from stable dynamics to chaotic dynamics, in a neighborhood of the origin, as $|\delta|$ increases from 0 to 2.5.

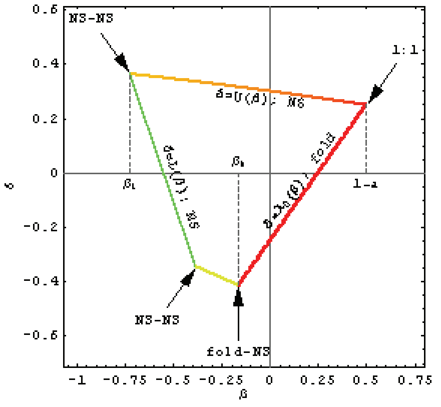


Fig. 1. Stability domain for the null solution when $k_{11} = 1$, $k_{22} = 5$, $k_{12} = 4$, $k_{21} = 2$

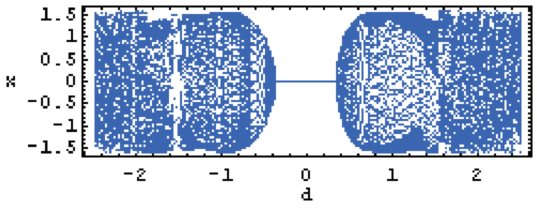


Fig. 2. Bifurcation diagram for system (10) with $\beta = -0.25$, in the (δ,x)-plane, for $\delta \in (-2.5,2.5)$ (with the step size of 0.02 for δ). For this bifurcation diagram, for each δ value, the initial conditions were reset to $(x_0,y_0)=(0.01,0.01)$ and 10^5 time steps were iterated before plotting the data (which consists of 10^2 points per δ value).

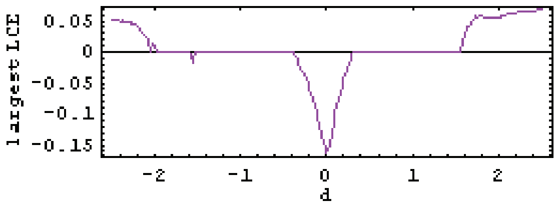


Fig. 3. Largest Lyapunov Characteristic Exponent for system (10) with $\beta = -0.25$. For the computation of the Lyapunov spectrum, for each δ value (step size 0.02 for δ), the initial conditions were reset and 10^5 time-steps were iterated before calculating the LCEs (which were computed over the next 10^5 time steps). The Lyapunov spectrum was computed using the Householder QR based (HQRB) method presented in (Bremen et al., 1997).

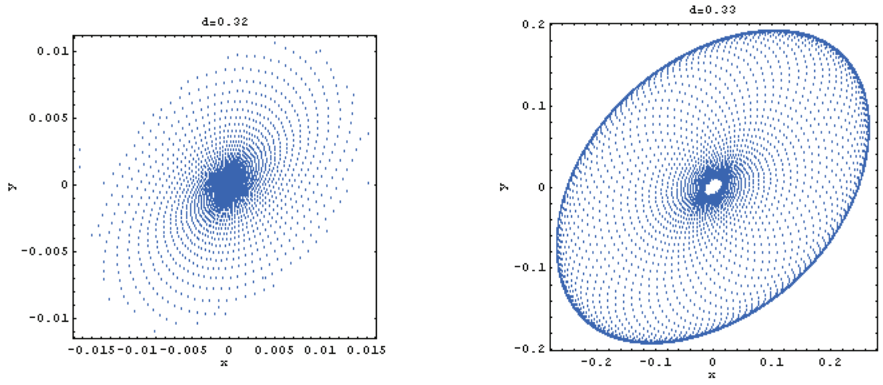


Fig. 4. Supercritical Neimark-Sacker bifurcation at $\delta = 0.324255$. For $\delta = 0.32$, the null solution is asymptotically stable, and the trajectory converges to the origin. For $b = 0.33$, an asymptotically stable cycle (1-torus) is present, and the trajectory converges to this cycle.

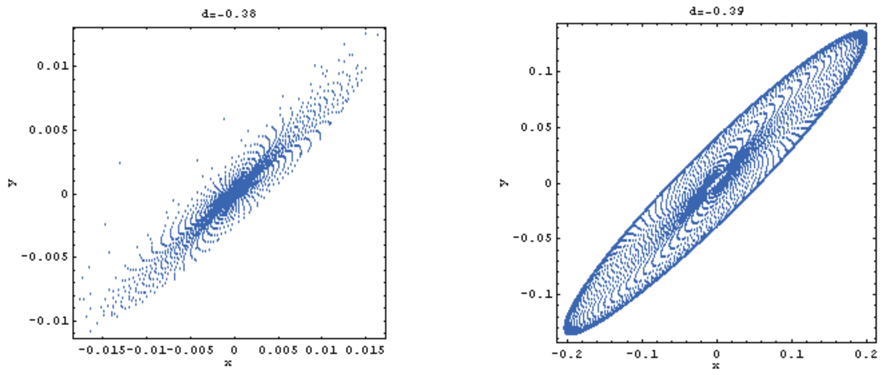


Fig. 5. Supercritical Neimark-Sacker bifurcation at $\delta = -0.385082$. For $\delta = -0.38$, the null solution is asymptotically stable, and the trajectory converges to the origin. For $b = -0.39$, an asymptotically stable cycle (1-torus) is present, and the trajectory converges to this cycle.

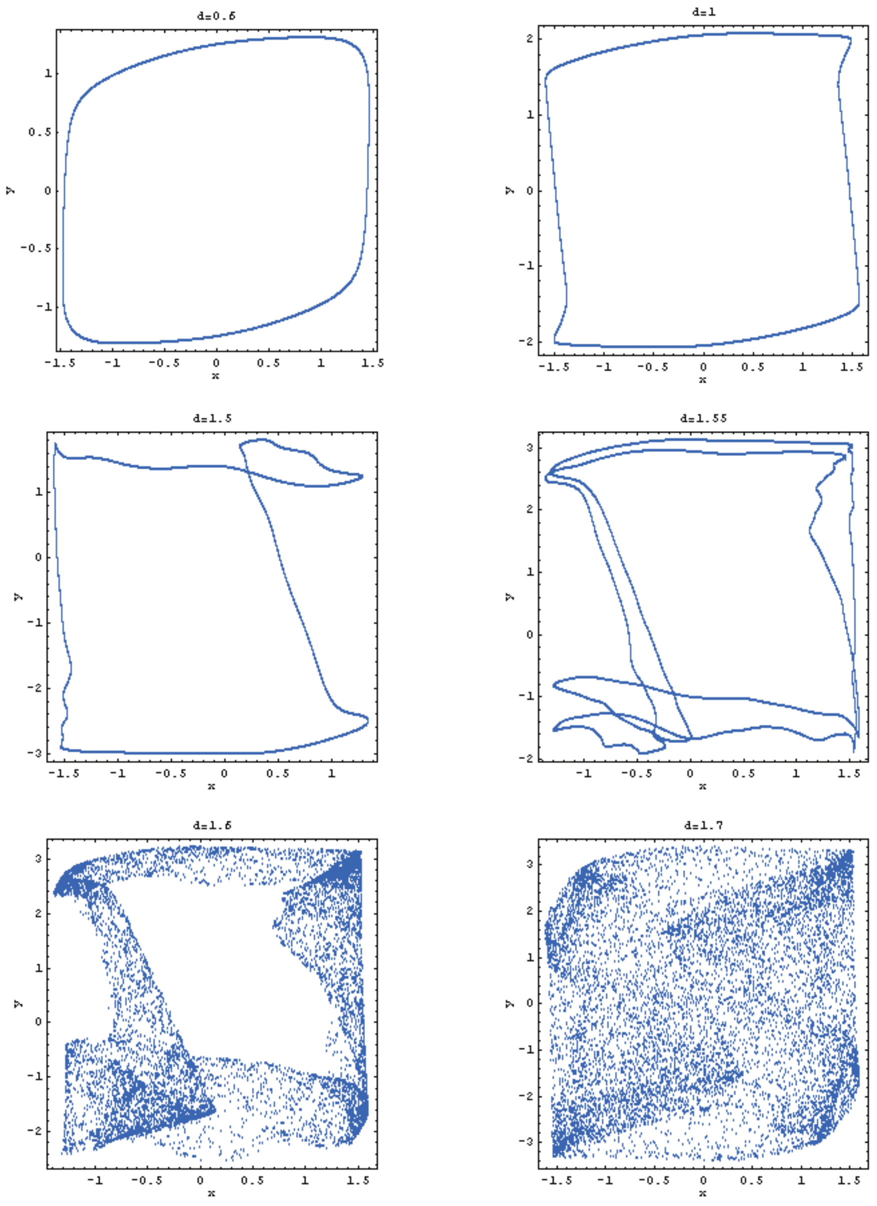


Fig. 6. Phase portraits for various values of $\delta \in (0,2.5)$, at the first step towards chaos. The route towards chaos passes through several stages: $\delta = 0.6$, $\delta = 1$, $\delta = 1.5$: 1-toruses ($\text{LLCE} = 0$); $\delta = 1.55$: 2-torus ($\text{LLCE} = 0$); $\delta = 1.6$: strange attractor ($\text{LLCE} \approx 0$); $\delta = 1.7$: chaos (LLCE>0). For each plot, considering the initial conditions $(x_0,y_0)=(0.01,0.01)$, the first 10^6 iterations of system (10) have been dropped, and the next 10^4 iterations have been plotted.

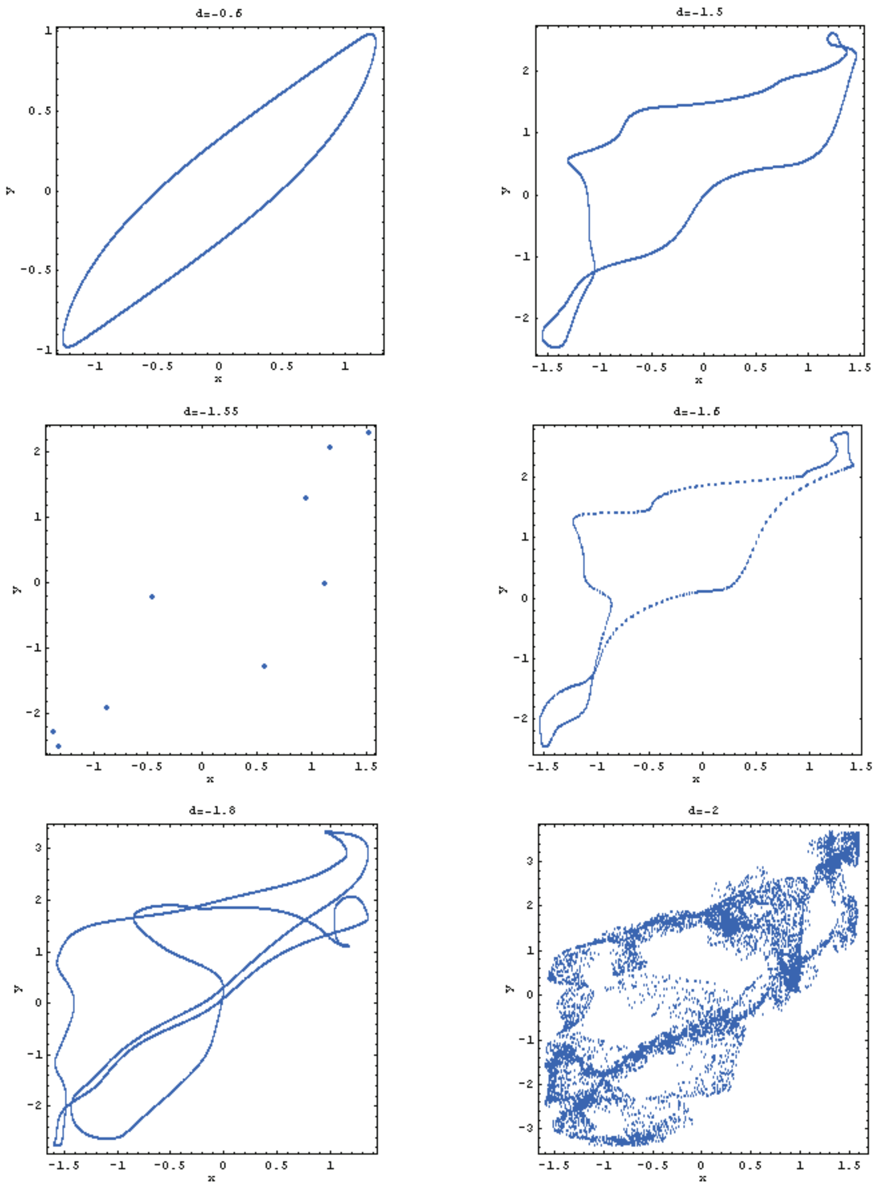


Fig. 7: Phase portraits for various values of $\delta \in (-2.5,0)$, at the first step towards chaos. The route towards chaos passes through several stages: $\delta = -0.6$, $\delta = -1.5$: 1-toruses ($\mathrm{LLCE} = 0$); $\delta = -1.55$: stable period-9 orbit ($\mathrm{LLCE} < 0$); $\delta = -1.6$: 1-torus ($\mathrm{LLCE} = 0$); $\delta = -1.8$: 2-torus ($\mathrm{LLCE} = 0$), $\delta = -2$: strange attractor ($\mathrm{LLCE} \approx 0$). For each plot, considering the initial conditions $(x_0, y_0) = (0.01, 0.01)$, the first 10^6 iterations of system (10) have been dropped, and the next 10^4 iterations have been plotted.

6. Conclusions

A complete bifurcation analysis has been presented for a discrete-time Hopfield-type neural network of two neurons with several delays, uncovering the structure of the stability domain of the null solution, as well as the types of bifurcations occurring at its boundary. The numerical example illustrated the theoretical results and suggested some routes towards chaos as the characteristic parameters of the system leave the stability domain.
A generalization of these results to more complicated networks of two or more neurons may constitute a direction for future research.

7. Acknowledgements

This work has been supported by the Romanian National Authority for Research under the contract PN-II-11028/14.09.2007 (NatComp - New Natural Computing Models in the Study of Complexity).

8. References

Adachi, M. & Aihara, K. (1997). Associative dynamics in a chaotic neural network. *Neural Networks*, 10(1):83-98.

Bremen, H. F.; Udwadia, F.E. & Proskurowski, W. (1997). An efficient QR based method for the computation of Lyapunov exponents. *Physica D: Nonlinear Phenomena*, 101(1-2):1-16.

Chen, L. & Aihara, K. (1995). Chaotic simulated annealing by a neural network model with transient chaos. *Neural Networks*, 8:915--930.

Chen, L. & Aihara, K. (1997). Chaos and asymptotical stability in discrete-time neural networks. *Physica D: Nonlinear Phenomena*, 104(3-4):286-325.

Chen, L. & Aihara K. (2001). Chaotic dynamics of neural networks ans its application to combinatorial optimization. *Journal of Dynamical Systems and Differential Equations*, 9(3):139-168.

Chen, S.S. & Shih, C.W. (2002). Transversal homoclinic orbits in a transiently chaotic neural network. *Chaos*, 12:654-671.

Guo, S. & Huang, L. (2004). Periodic oscillation for discrete-time Hopfield neural networks. *Physics Letters A*, 329(3):199-206.

Guo, S.; Huang, L. & Wang, L. (2004). Exponential stability of discrete-time Hopfield neural networks. *Computers and Mathematics with Applications*, 47:1249-1256.

Guo, S.; Tang, X. & Huang, L. (2007). Stability and bifurcation in a discrete system of two neurons with delays. *Nonlinear Analysis: Real World Applications*, DOI: 10.1016/j.nonrwa.2007.03.002, in press.

He, W. & Cao, J. (2007). Stability and bifurcation of a class of discrete-time neural networks. *Applied Mathematical Modelling*, 31(10):2111-2122.

Hopfield, J.J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proc. Nat. Acad. Sci.*, 79:2554-2558.

Huang, Y. & Zou, X. (2005). Co-Existence of Chaos and Stable Periodic Orbits in a Simple Discrete Neural Network. *Journal of Nonlinear Science*, 15:291-303.

Kaslik, E. & Balint, St. (2007a). Bifurcation analysis for a two-dimensional delayed discrete-time Hopfield neural network. *Chaos, Solitons and Fractals*, 34(4):1245-1253, 2007.

Kaslik, E. & Balint, St. (2007b). Bifurcation analysis for a discrete-time Hopfield neural network of two neurons with two delays and self-connections. *Chaos, Solitons and Fractals*, DOI 10.1016/j.chaos.2007.01.126, in press.

Kaslik, E. & Balint, St. (2007c). Chaotic dynamics of a delayed discrete-time Hopfield network of two nonidentical neurons with no self-connections. *Journal of Nonlinear Science*, DOI 10.1007/s00332-007-9015-5, in press.

Kuznetsov, Yu. A. (2004). *Elements of applied bifurcation theory*. Springer-Verlag, ISBN 0-387-21906-4, New York.

Mohamad, S. & Gopalsamy, K. (2000). Dynamics of a class of discrete-time neural networks and their continuous-time counterparts. *Mathematics and Computers in Simulation*, 53(1-2):1-39.

Pasemann, F.; Hild, M. & Zahedi, K. (2003). SO(2)-Networks as Neural Oscillators. *Lecture Notes in Computer Science: Computational Methods in Neural Modeling*, pp. 1042-1050, ISBN 978-3-540-40210-7, Springer Berlin/Heidelberg.

Tank, D.W. & Hopfield, J.J. (1986). Simple neural optimization networks: an A/D converter, signal decision circuit and a linear programming circuit. *IEEE Transactions on Circuits and Systems*, 33:533-541.

Yu, W. & Cao, J. (2006) Cryptography based on delayed chaotic neural networks. *Physics Letters A*, 356(4-5):333-338.

Yuan, Z.; Hu, D. & Huang, L. (2004). Stability and bifurcation analysis on a discrete-time system of two neurons. *Applied Mathematical Letters*, 17:1239-1245.

Yuan, Z.; Hu, D. & Huang, L. (2005). Stability and bifurcation analysis on a discrete-time neural network. *Journal of Computational and Applied Mathematics*, 177:89-100.

Zhang, C. & Zheng, B. (2005). Hopf bifurcation in numerical approximation of a n-dimension neural network model with multi-delays. *Chaos, Solitons & Fractals*, 25(1):129-146.

Zhang, C. & Zheng, B. (2007). Stability and bifurcation of a two-dimension discrete neural network model with multi-delays. *Chaos, Solitons & Fractals*, 31(5):1232-1242.

Case Studies for Applications of Elman Recurrent Neural Networks

Elif Derya Übeyli[1] and Mustafa Übeyli[2]
[1]*Department of Electrical and Electronics Engineering, Faculty of Engineering, TOBB Ekonomi ve Teknoloji Üniversitesi, Ankara,*
[2]*Department of Mechanical Engineering, Faculty of Engineering, TOBB Ekonomi ve Teknoloji Üniversitesi, 06530 Söğütözü, Ankara, Turkey*

1. Introduction

Artificial neural networks (ANNs) are computational modeling tools that have recently emerged and found extensive acceptance in many disciplines for modeling complex real-world problems. ANN-based models are empirical in nature, however they can provide practically accurate solutions for precisely or imprecisely formulated problems and for phenomena that are only understood through experimental data and field observations. ANNs produce complicated nonlinear models relating the inputs (the independent variables of a system) to the outputs (the dependent predictive variables). ANNs have been widely used for various tasks, such as pattern classification, time series prediction, nonlinear control, and function approximation. ANNs are desirable because (i) nonlinearity allows better fit to the data, (ii) noise-insensitivity provides accurate prediction in the presence of uncertain data and measurement errors, (iii) high parallelism implies fast processing and hardware failure-tolerance, (iv) learning and adaptivity allow the system to modify its internal structure in response to changing environment, and (v) generalization enables application of the model to unlearned data (Fausett, 1994; Haykin, 1994; Hassoun, 1995).
The idea of using ANNs for pattern classification purposes has encountered, for a long time, the favour of many researchers (Miller et al., 1992; Wright et al., 1997; Wright & Gough, 1999; Saxena et al., 2002; Übeyli, 2007a; 2007b; 2008a; 2008b; 2008c). Feedforward neural networks are a basic type of neural networks capable of approximating generic classes of functions, including continuous and integrable ones. One of the most frequently used feedforward neural network for pattern classification is the multilayer perceptron neural network (MLPNN) which is trained to produce a spatial output pattern in response to an input spatial pattern (Fausett, 1994; Haykin, 1994; Hassoun, 1995). The mapping performed is static, therefore, the network is inherently not suitable for processing temporal patterns. Attempts have been made to use the MLPNN to classify temporal patterns by transforming the temporal domain into a spatial domain.
An alternate neural network approach is to use recurrent neural networks (RNNs) which have memory to encode past history. Several forms of RNNs have been proposed and they may be classified as partially recurrent or fully recurrent networks (Saad et al., 1998; Gupta

& McAvoy, 2000; Gupta et al., 2000; Übeyli & Übeyli, 2007; Übeyli, 2008a; 2008c). RNNs can perform highly non-linear dynamic mappings and thus have temporally extended applications, whereas multilayer feedforward networks are confined to performing static mappings. RNNs have been used in a number of interesting applications including associative memories, spatiotemporal pattern classification, control, optimization, forecasting and generalization of pattern sequences (Saad et al., 1998; Gupta & McAvoy, 2000; Gupta et al., 2000; Übeyli & Übeyli, 2007; Übeyli, 2008a; 2008c). In partially recurrent networks, partial recurrence is created by feeding back delayed hidden unit outputs or the outputs of the network as additional input units. The partially recurrent networks, whose connections are mainly feedforward were used, but they include a carefully chosen set of feedback connections. One example of such a network is an Elman RNN which in principle is set up as a regular feedforward network (Elman, 1990). Architecture of Elman RNNs, case studies for biomedical engineering, case study for nuclear engineering are presented in the subtitles of this chapter. The results of the case studies for biomedical engineering and nuclear engineering are presented. These conclusions will assist to the readers in gaining intuition about the performance of the Elman RNNs used in biomedical engineering and nuclear engineering problems.

2. Architecture of Elman recurrent neural networks

RNNs have been used in pattern classification, control, optimization, forecasting and generalization of pattern sequences (Petrosian et al., 2000; Petrosian et al., 2001; Shieh et al., 2004; Übeyli & Übeyli, 2007; Übeyli, 2008a; 2008c). Fully recurrent networks use unconstrained fully interconnected architectures and learning algorithms that can deal with time-varying input and/or output in non-trivial ways. In spite of several modifications of learning algorithms to reduce the computational expense, fully recurrent networks are still complicated when dealing with complex problems. Therefore, the partially recurrent networks, whose connections are mainly feedforward, were used but they include a carefully chosen set of feedback connections. The recurrence allows the network to remember cues from the past without complicating the learning excessively. The structure proposed by Elman (1990) is an illustration of this kind of architecture. Elman RNNs were used in these applications and therefore in the following the Elman RNN is presented.

An Elman RNN is a network which in principle is set up as a regular feedforward network. This means that all neurons in one layer are connected with all neurons in the next layer. An exception is the so-called context layer which is a special case of a hidden layer. Figure 1 shows the architecture of an Elman RNN. The neurons in the context layer (context neurons) hold a copy of the output of the hidden neurons. The output of each hidden neuron is copied into a specific neuron in the context layer. The value of the context neuron is used as an extra input signal for all the neurons in the hidden layer one time step later. Therefore, the Elman network has an explicit memory of one time lag (Elman, 1990).

Similar to a regular feedforward neural network, the strength of all connections between neurons are indicated with a weight. Initially, all weight values are chosen randomly and are optimized during the stage of training. In an Elman network, the weights from the hidden layer to the context layer are set to one and are fixed because the values of the context neurons have to be copied exactly. Furthermore, the initial output weights of the context neurons are equal to half the output range of the other neurons in the network. The Elman network can be trained with gradient descent backpropagation and optimization methods, similar to regular feedforward neural networks (Pineda, 1987). The backpropagation has some problems for many applications. The algorithm is not guaranteed to find the global minimum of the error function since gradient descent may get stuck in local minima, where it may remain indefinitely. In addition to this, long training sessions are often required in order to find an acceptable weight solution because of the well known difficulties inherent in gradient descent optimization (Haykin, 1994; Chaudhuri & Bhattacharya, 2000). Therefore, a lot of variations to improve the convergence of the backpropagation were proposed. Optimization methods such as second-order methods (conjugate gradient, quasi-Newton, Levenberg-Marquardt) have also been used for neural networks training in recent years. The Levenberg-Marquardt algorithm combines the best features of the Gauss-Newton technique and the steepest-descent algorithm, but avoids many of their limitations. In particular, it generally does not suffer from the problem of slow convergence (Battiti, 1992; Hagan & Menhaj, 1994) and can yield a good cost function compared with the other training algorithms.

2.1. Levenberg-Marquardt algorithm

Essentially, the Levenberg-Marquardt algorithm is a least-squares estimation algorithm based on the maximum neighborhood idea. Let $E(\mathbf{w})$ be an objective error function made up of m individual error terms $e_i^2(\mathbf{w})$ as follows:

$$E(\mathbf{w}) = \sum_{i=1}^{m} e_i^2(\mathbf{w}) = \left\| f(\mathbf{w}) \right\|^2, \tag{1}$$

where $e_i^2(\mathbf{w}) = \left(\mathrm{y}_{di} - \mathrm{y}_i\right)^2$ and y_{di} is the desired value of output neuron i, y_i is the actual output of that neuron.

It is assumed that function $f(\cdot)$ and its Jacobian J are known at point $\mathbf{w}$. The aim of the Levenberg-Marquardt algorithm is to compute the weight vector $\mathbf{w}$ such that $E(\mathbf{w})$ is minimum. Using the Levenberg-Marquardt algorithm, a new weight vector $\mathbf{w}_{k+1}$ can be obtained from the previous weight vector $\mathbf{w}_k$ as follows:

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \delta\mathbf{w}_k, \tag{2}$$

where $\delta\mathbf{w}_k$ is defined as

$$\delta\mathbf{w}_k = -(J_k{}^T f(\mathbf{w}_k))(J_k{}^T J_k + \lambda\mathbf{I})^{-1}. \tag{3}$$

In equation (3), J_k is the Jacobian of f evaluated at $\mathbf{w}_k$, λ is the Marquardt parameter, $\mathbf{I}$ is the identity matrix (Battiti, 1992; Hagan & Menhaj, 1994). The Levenberg-Marquardt algorithm may be summarized as follows:

i. compute $E(\mathbf{w}_k)$,

ii. start with a small value of λ ($\lambda = 0.01$),

iii. solve equation (3) for $\delta\mathbf{w}_k$ and compute $E(\mathbf{w}_k + \delta\mathbf{w}_k)$,

iv. if $E(\mathbf{w}_k + \delta\mathbf{w}_k) \geq E(\mathbf{w}_k)$, increase λ by a factor of 10 and go to (iii),

v. if $E(\mathbf{w}_k + \delta\mathbf{w}_k) < E(\mathbf{w}_k)$, decrease λ by a factor of 10, update $\mathbf{w}_k : \mathbf{w}_k \leftarrow \mathbf{w}_k + \delta\mathbf{w}_k$ and go to (iii).

3. Case studies for biomedical engineering

Automated biomedical signals classification algorithms can be divided into three steps: pre-processing, feature extraction/selection, and classification. The techniques developed for automated biomedical signals classification transform the mostly qualitative diagnostic criteria into a more objective quantitative signal feature classification problem (Miller et al., 1992; Wright et al., 1997; Wright & Gough, 1999; Saxena et al., 2002; Übeyli, 2007a; 2007b; 2008a; 2008b; 2008c). For pattern processing problems to be tractable requires the conversion of patterns to features, which are condensed representations of patterns, ideally containing only salient information. Selection of the neural network inputs has two meanings: 1) which components of a pattern, or 2) which set of inputs best represent a given pattern. Different diverse feature vectors can be extracted from the biomedical signals under study by using different spectral analysis methods. The features are then used in representation and/or discrimination of the biomedical signals, i.e., wavelet coefficients and Lyapunov exponents (Miller et al., 1992; Wright et al., 1997; Wright & Gough, 1999; Saxena et al., 2002; Übeyli, 2007a; 2007b; 2008a; 2008b; 2008c). Therefore, the RNNs employing single feature vector or composite features can be implemented for automated classification of biomedical signals.

3.1 Elman recurrent neural networks for analysis of Doppler ultrasound signals

The implementation of Elman RNNs with the Lyapunov exponents for Doppler ultrasound signals classification is presented. This study is based on the consideration that Doppler ultrasound signals are chaotic signals. This consideration was tested successfully using the nonlinear dynamics tools, like the computation of Lyapunov exponents. Decision making was performed in two stages: computation of Lyapunov exponents as representative features of the Doppler ultrasound signals and classification using the RNNs trained on the extracted features (Übeyli, 2008a).

Doppler ultrasound is widely used as a noninvasive method for the assessment of blood flow in both the central and peripheral circulation. It may be used to estimate blood flow, to image regions of blood flow and to locate sites of arterial disease as well as flow characteristics and resistance of ophthalmic and internal carotid arteries (Evans et al., 1989). Doppler systems are based on the principle that ultrasound, emitted by an ultrasonic transducer, is returned partially towards the transducer by the moving targets, thereby inducing a shift in frequency proportional to the emitted frequency and the velocity along the ultrasound beam. Studies in the literature have shown that Doppler ultrasound evaluation can give reliable information on both systolic and diastolic blood velocities of arteries and is useful in screening certain hemodynamic alterations in arteries (Evans et al., 1989; Wright et al., 1997; Wright & Gough, 1999; Übeyli, 2008a).

The objective of the present study in the field of automated diagnosis of arterial diseases is to extract the representative features of the ophthalmic arterial (OA) and internal carotid arterial (ICA) Doppler ultrasound signals and to present the accurate classification model. As in traditional pattern recognition systems, the model consists of three main modules: a feature extractor that generates a feature vector from the raw Doppler ultrasound signals, feature selection that composes composite features (Lyapunov exponents), and a feature classifier that outputs the class based on the composite features (recurrent neural networks – RNNs). A significant contribution of the present work was the composition of composite features which were used to train novel classifier (RNNs trained on computed Lyapunov exponents) for the OA and ICA Doppler ultrasound signals. To evaluate performance of the RNNs trained with the Levenberg-Marquardt algorithm, the classification accuracies and the central processing unit (CPU) times of training were considered.

The technique used in the computation of Lyapunov exponents was related with the Jacobi-based algorithms. For each OA and ICA Doppler segment (256 discrete data), 128 Lyapunov exponents were computed. The computed Lyapunov exponents samples of OA and ICA Doppler signals are shown in Figures 2 and 3. High-dimension of feature vectors increased computational complexity and therefore, in order to reduce the dimensionality of the extracted feature vectors (feature selection), statistics over the set of the Lyapunov exponents were used. The following statistical features were used in reducing the dimensionality of the extracted feature vectors representing the signals under study:

1. Maximum of the Lyapunov exponents of each Doppler ultrasound signal segment.
2. Minimum of the Lyapunov exponents of each Doppler ultrasound signal segment.
3. Mean of the Lyapunov exponents of each Doppler ultrasound signal segment.
4. Standard deviation of the Lyapunov exponents of each Doppler ultrasound signal segment.

The feature vectors were computed by the usage of the MATLAB software package. The RNNs proposed for classification of the Doppler ultrasound signals were implemented by using the MATLAB software package (MATLAB version 7.0 with neural networks toolbox). The key design decisions for the neural networks used in classification are the architecture and the training process. Different network architectures were experimented and the results of the architecture studies confirmed that for the OA Doppler signals, networks with one hidden layer consisting of 20 recurrent neurons results in higher classification accuracy. The RNNs with one hidden layer were superior to models with two hidden layers for the ICA Doppler signals. The most suitable network configuration found was 15 recurrent neurons for the hidden layer.

Classification results of the classifiers were displayed by a confusion matrix. In a confusion matrix, each cell contains the raw number of exemplars classified for the corresponding combination of desired and actual network outputs. The confusion matrices showing the classification results of the classifiers used for classification of the OA and ICA Doppler signals are given in Tables 1 and 2. From these matrices one can tell the frequency with which a Doppler signal is misclassified as another. As it is seen from Table 1, healthy subjects are most often confused with subjects suffering from OA stenosis, likewise subjects suffering from ocular Behcet disease with subjects suffering from OA stenosis. From Table 2, one can see that healthy subjects are most often confused with subjects suffering from ICA stenosis, likewise subjects suffering from ICA stenosis with subjects suffering from ICA occlusion.

The test performance of the classifiers can be determined by the computation of specificity, sensitivity and total classification accuracy. The specificity, sensitivity and total classification accuracy are defined as:

Specificity: number of true negative decisions / number of actually negative cases
Sensitivity: number of true positive decisions / number of actually positive cases
Total classification accuracy: number of correct decisions / total number of cases
A true negative decision occurs when both the classifier and the physician suggested the absence of a positive detection. A true positive decision occurs when the positive detection of the classifier coincided with a positive detection of the physician.
In order to demonstrate performance of the classifiers used for classification of the OA and ICA Doppler signals, the classification accuracies (specificity, sensitivity, total classification accuracy) on the test sets and the CPU times of training (for Pentium 4, 3.00 GHz) of the RNNs are presented in Table 3. The present research demonstrated that the Lyapunov exponents are the features which well represent the Doppler ultrasound signals and the RNNs trained on these features achieved high classification accuracies (Übeyli, 2008a).

3.2 Elman recurrent neural networks for detection of electrocardiographic changes in partial epileptic patients

The aim of this study is to evaluate the diagnostic accuracy of the RNNs with composite features (wavelet coefficients and Lyapunov exponents) on the electrocardiogram (ECG) signals. Two types of ECG beats (normal and partial epilepsy) were obtained from the MIT-BIH database (Al-Aweel et al., 1999). Decision making was performed in two stages: computing composite features which were then input into the classifiers and classification using the classifiers trained on the extracted features (Übeyli, 2008c).
Epileptic seizures are associated with several changes in autonomic functions, which may lead to cardiovascular, respiratory, gastrointestinal, cutaneous, and urinary manifestations (Leutmezer et al., 2003; Rocamora et al., 2003).Cardiovascular changes have received the most attention, because of their possible contribution to sudden unexplained death. Studies have reported the importance of monitoring the ECG signal during epileptic seizures, since the seizures can trigger high risk cardiac arrhythmias. Since seizures can occur at any time in an epileptic patient, the ECG may need to be recorded for several hours or days at a time, leading to an enormous quantity of data to be studied by physicians. To reduce the time and possibility of errors, automatic computer-based algorithms have been proposed to support or replace the diagnosis and analysis performed by the physician (Miller et al., 1992; Saxena et al., 2002; Übeyli, 2007a; 2007b; 2008c). From the hours of ECG data, these algorithms can flag the periods when the patient is having a seizure and, eventually, determine from these periods if any cardiac arrhythmias occured. This study provides a highly accurate algorithm for classifying non-arrhythmic ECG waveforms as normal or partial epileptic.
The evaluation of the classification capabilities of the Elman RNNs trained with Levenberg-Marquardt algorithm was performed on the ECG signals (normal and partial epilepsy ECG beats) from the MIT-BIH database (Al-Aweel et al., 1999). As in traditional pattern recognition systems, the model consists of three main modules: a feature extractor that generates a feature vector from the ECG signals, feature selection that composes composite features (wavelet coefficients and Lyapunov exponents), and a feature classifier that outputs the class based on the composite features. A significant contribution of the work was the composition of composite features which were used to train novel classifier (RNN trained on composite feature) for the ECG signals. To evaluate performance of the classifiers, the classification accuracies, the CPU times of training and the receiver operating characteristic (ROC) curves of the classifiers were examined (Übeyli, 2008c).

The detail wavelet coefficients at the first decomposition level of the two types of ECG beats are presented in Figures 4(a) and (b), respectively. From these figures it is obvious that the detail wavelet coefficients of the two types of ECG beats are different from each other and therefore they can serve as useful parameters in discriminating the ECG signals. A smaller number of parameters called wavelet coefficients are obtained by the wavelet transform (WT). These coefficients represent the ECG signals and therefore, they are particularly important for recognition and diagnostic purposes. The Lyapunov exponents of the two types of ECG beats are shown in Figures 5(a) and (b), respectively. One can see that the Lyapunov exponents of the two types of ECG beats differ significantly from each other so they can be used for representing the ECG signals. As it is seen from Figures 5(a) and (b), there are positive Lyapunov exponents, which confirm the chaotic nature of the ECG signals. Lyapunov exponents are a quantitative measure for distinguishing among the various types of orbits based upon their sensitive dependence on the initial conditions, and are used to determine the stability of any steady-state behavior, including chaotic solutions. The reason why chaotic systems show aperiodic dynamics is that phase space trajectories that have nearly identical initial states will separate from each other at an exponentially increasing rate captured by the so-called Lyapunov exponent.

The following statistical features were used in reducing the dimensionality of the extracted diverse feature vectors representing the ECG signals:

1. Maximum of the wavelet coefficients in each subband, maximum of the Lyapunov exponents in each beat.
2. Minimum of the wavelet coefficients in each subband, minimum of the Lyapunov exponents in each beat.
3. Mean of the wavelet coefficients in each subband, mean of the Lyapunov exponents in each beat.
4. Standard deviation of the wavelet coefficients in each subband, standard deviation of the Lyapunov exponents in each beat.

Different network architectures were tested and the architecture studies confirmed that for the ECG signals, RNN with one hidden layer consisting of 20 recurrent neurons trained on a composite feature vector results in higher classification accuracy. In order to compare performance of the different classifiers, for the same classification problem the MLPNN, which is the most commonly used feedforward neural networks was also implemented. The single hidden layered (25 hidden neurons) MLPNN was used to classify the ECG signals based on a composite feature vector.

The values of the statistical parameters (specificity, sensitivity and total classification accuracy) and the CPU times of training (for Pentium 4, 3.00 GHz) of the two classifiers are presented in Table 4. ROC plots provide a view of the whole spectrum of sensitivities and specificities because all possible sensitivity/specificity pairs for a particular test are graphed. The performance of a test can be evaluated by plotting a ROC curve for the test and therefore, ROC curves were used to describe the performance of the classifiers. A good test is one for which sensitivity rises rapidly and 1-specificity hardly increases at all until sensitivity becomes high. ROC curves which are shown in Figure 6 demonstrate the performances of the classifiers on the test files. The classification results presented in Table 4 and Figure 6 (classification accuracies, CPU times of training, ROC curves) denote that the RNN trained on composite feature vectors obtains higher accuracy than that of the MLPNN (Übeyli, 2008c).

4. Case study for nuclear engineering

Considerable interest has been developed to modeling of dynamic systems with ANNs in recent years. The basic motivation is the ability of neural networks to create data driven representations of the underlying dynamics with less reliance on accurate mathematical or physical modeling. There exist many problems for which such data-driven representations offer more advantages over more traditional modeling techniques, such as availability of fast hardware implementations, ability to cope with noisy or incomplete data and ability to very fast data generation by using ordinary digital computers (Narendra & Parthasarathy, 1990; Boroushaki et al., 2002; Choi et al., 2004; Übeyli & Übeyli, 2007).

Recently, data processing algorithms based on artificial intelligence gained popularity in nuclear technology. In particular, ANNs found their application in a wide range of problems (Uhrig & Tsoukalas, 1999), such as diagnostics (Bartlett & Uhrig, 1992; Kim et al., 1992), signal validation (Fantoni & Mazzola, 1996a; 1996b), anomalies detection (Ogha & Seki, 1991; Kozma & Nabeshima, 1995; Reifman, 1997) and core monitoring (Kozma et al., 1995). ANNs allow modeling of complex systems without requiring an explicit knowledge or formulation of the relationship existing among the variables, and they can constitute a valuable alternative to structured models or empirical correlations (Thibault & Grandjean, 1991).

4.1. Elman recurrent neural networks for neutronic parameters of a thorium fusion breeder

RNNs are capable to represent arbitrary nonlinear dynamical systems (Narendra & Parthasarathy, 1990; Boroushaki et al., 2002). Learning and generalization ability, fast real time operation and ease of implementation features have made RNNs popular in the last decade. Recent works by nuclear engineering researchers demonstrated the ability of RNNs in identification of complex nonlinear plants like nuclear reactor cores (Boroushaki et al., 2002; Adalı et al., 1997; Boroushaki et al., 2003; Şeker et al., 2003; Ortiz & Requena, 2004). Übeyli & Übeyli (2007) used the Elman RNNs for the estimation of the neutronic parameters of a thorium fusion breeder.

The inputs of the implemented nine RNNs (for three types of coolant and three outputs) are atomic densities of the components used in the investigated reactor (Übeyli & Übeyli, 2007). The outputs of the computations are the main neutronic parameters; tritium breeding ratio, energy multiplication factor and net ^{233}U production. Figure 7 shows the RNNs model used in neural computation of the main neutronic parameters.

In calculations by Scale4.3, atomic densities of the blanket zone components, thicknesses and materials of the zones and reaction cross section types are entered to the prepared inputs. Then, outputs are generated by running these inputs in a personal computer. In the outputs, the reaction cross sections with respect to neutron energy groups required to compute neutronic parameters of the reactor are derived from the library. After that, these outputs are processed with a computer code to get neutronic parameters of the reactor for an operation period of 48 months (Übeyli & Acır, 2007).

The nine RNNs proposed for computation of the main neutronic parameters (tritium breeding ratio computation, energy multiplication factor and net ^{233}U production) were implemented by using the MATLAB software package (MATLAB version 7.0 with neural networks toolbox). Different network architectures were experimented and the results of the architecture studies confirmed that, networks with one hidden layer results in higher computation accuracy. The Scale 4.3 was used to generate data (Übeyli & Acır, 2007). For

neural computation of the tritium breeding ratio, energy multiplication factor and net ^{233}U production 49 data, consisting of input parameters and the corresponding computed values of the tritium breeding ratio, energy multiplication factor and net ^{233}U production, were generated for each RNN.
The test results of the RNNs implemented for three types of coolant are compared with the results of Scale 4.3 in Figures 8-10 for the tritium breeding ratio (TBR) computation, the energy multiplication factor (M) and the net ^{233}U production, respectively. It can be clearly seen from these Figures that the results of the RNNs presented in this study are in very good agreement with the results of Scale 4.3. The difference between the output of the network and the desired output (computed using Scale 4.3) is referred to as the error and can be measured in different ways. In this study, mean square error (MSE), mean absolute error (MAE), and correlation coefficient (r) were used for the measuring error of the RNNs during test process. The correlation coefficient is limited with the range [-1,1]. When $r = 1$ there is a perfect positive linear correlation between network output and desired output, which means that they vary by the same amount. When $r = -1$ there is a perfectly linear negative correlation between network output and desired output, that means they vary in opposite ways. When $r = 0$ there is no correlation between network output and desired output. Intermediate values describe partial correlations. In Table 5, performance evaluation parameters of the RNNs implemented for three types of coolant are given for the tritium breeding ratio computation, the energy multiplication factor and the net ^{233}U production during test process. The values of performance evaluation parameters and the very good agreement between the results of the RNNs and the results of Scale 4.3 support the validity of the RNNs trained with the Levenberg-Marquardt algorithm presented in this study (Übeyli & Übeyli, 2007).

5. Conclusions

ANNs may offer a potentially superior method of biomedical signal analysis to the spectral analysis methods. In contrast to the conventional spectral analysis methods, ANNs not only model the signal, but also make a decision as to the class of signal. Another advantage of ANN analysis over existing methods of biomedical signal analysis is that, after an ANN has trained satisfactorily and the values of the weights and biases have been stored, testing and subsequent implementation is rapid. The proposed combined Lyapunov exponents/RNN approach can be evaluated in discrimination of other Doppler ultrasound signals or time-varying biomedical signals. Preprocessing, feature extraction methods and ANN architectures are the main modules of an automated diagnostic systems and therefore they play important roles in determining the classification accuracies. Thus, further work can be performed for improving the classification accuracies by the usage of different preprocessing (different filtering methods), feature extraction methods (different spectral analysis methods) and ANN architectures (self-organizing map, radial basis function, mixture of experts, etc.) (Übeyli, 2008a).
The research demonstrated that the wavelet coefficients and the Lyapunov exponents are the features which well represent the ECG signals and the RNN trained on these features achieved high classification accuracies. The overall results of the RNN were better when they were trained on the computed composite features for each ECG beat. The results demonstrated that significant improvement can be achieved in accuracy by using the RNNs compared to the feedforward neural network models (MLPNNs). This may be attributed to several factors including the training algorithms, estimation of the network parameters and the scattered and mixed nature of the features. The results of the present study

demonstrated that the RNN can be used in classification of the ECG beats by taking into consideration the misclassification rates (Übeyli, 2008c).

ANNs have recently been introduced to the nuclear engineering applications as a fast and flexible vehicle to modeling, simulation and optimization. A new approach based on RNNs was presented for the neutronic parameters of a thorium fusion breeder. The results of the RNNs implemented for the tritium breeding ratio computation, energy multiplication factor and net ^{233}U production in a thorium fusion breeder and the results available in the literature obtained by using Scale 4.3 were compared. The drawn conclusions confirmed that the proposed RNNs could provide an accurate computation of the tritium breeding ratio computation, the energy multiplication factor and the net ^{233}U production of the thorium fusion breeder (Übeyli & Übeyli, 2007).

6. References

Adalı, T., Bakal, B., Sönmez, M.K., Fakory, R. & Tsaoi, C.O. (1997). Modeling nuclear reactor core dynamics with recurrent neural networks. *Neurocomputing*, Vol. 15, 363-381.

Al-Aweel, I.C., Krishnamurthy, K.B., Hausdorff, J.M., Mietus, J.E., Ives, J.R., Blum, A.S., Schomer, D.L. & Goldberger, A.L. (1999). Postictal heart rate oscillations in partial epilepsy. *Neurology*, Vol. 53, No. 7, 1590-1592.

Bartlett, E.B. & Uhrig, R.E. (1992). Nuclear power plant status diagnostics using artificial neural networks. *Nuclear Technology*, Vol. 97, 272-281.

Battiti, R. (1992). First- and second-order methods for learning: between steepest descent and Newton's method. *Neural Computation*, Vol. 4, 141-166.

Boroushaki, M., Ghofrani, M.B. & Lucas, C. (2002). Identification of a nuclear reactor core (VVER) using recurrent neural networks. *Annals of Nuclear Energy*, Vol. 29, 1225-1240.

Boroushaki, M., Ghofrani, M.B., Lucas, C. & Yazdanpanah, M.J. (2003). An intelligent nuclear reactor core controller for load following operations, using recurrent neural networks and fuzzy systems. *Annals of Nuclear Energy*, Vol. 30, 63-80.

Chaudhuri, B.B. & Bhattacharya, U. (2000). Efficient training and improved performance of multilayer perceptron in pattern classification. *Neurocomputing*, Vol. 34, 11-27.

Choi, Y.J., Kim, H.K., Baek, W.P., Chang, S.H. (2004). Hybrid accident simulation methodology using artificial neural networks for nuclear power plants. *Information Sciences*, Vol. 160, 207-224.

Elman, J.L. (1990). Finding structure in time. *Cognitive Science*, Vol. 14, No. 2, 179-211.

Evans, D.H., McDicken, W.N., Skidmore, R. & Woodcock, J.P. (1989). *Doppler Ultrasound: Physics, Instrumentation and Clinical Applications*, Wiley, Chichester.

Fantoni, P.F. & Mazzola, A. (1996a). Multiple-failure signal validation in nuclear power plants using artificial neural networks. *Nuclear Technology*, Vol. 113, 368-374.

Fantoni, P.F. & Mazzola, A. (1996b). A pattern recognition-artificial neural networks based model for signal validation in nuclear power plants. *Annals of Nuclear Energy*, Vol. 23, No. 13, 1069-1076.

Fausett, L. (1994). *Fundamentals of Neural Networks Architectures, Algorithms, and Applications*, Prentice Hall, Inc., Englewood Cliffs, NJ.

Gupta, L. & McAvoy, M. (2000). Investigating the prediction capabilities of the simple recurrent neural network on real temporal sequences. *Pattern Recognition*, Vol. 33, No. 12, 2075-2081.

Gupta, L., McAvoy, M. & Phegley, J. (2000). Classification of temporal sequences via prediction using the simple recurrent neural network. *Pattern Recognition*, Vol. 33, No. 10, 1759-1770.

Hagan, M.T. & Menhaj, M.B. (1994). Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks*, Vol. 5, No. 6, 989-993.

Hassoun, M.H. (1995). *Fundamentals of Artificial Neural Networks*, Massachusetts Institute of Technology Press, London.

Haykin, S. (1994). *Neural networks: A Comprehensive Foundation*, Macmillan, New York.

Kim, K., Aljundi, T.L. & Bartlett, E.B. (1992). Confirmation of artificial neural networks: nuclear power plant fault diagnostics. *Transactions of the American Nuclear Society*, Vol. 66, 112.

Kozma, R. & Nabeshima, K. (1995). Studies on the detection of incipient coolant boiling in nuclear reactors using artificial neural networks. *Annals of Nuclear Energy*, Vol. 22, No. 7, 483-496.

Kozma, R., Sato, S., Sakuma, M., Kitamura, M. & Sugiyama, T. (1995). Generalization of knowledge acquired by a reactor core monitoring system based on a neuro-fuzzy algorithm. *Progress in Nuclear Energy*, Vol. 29, No. 3-4, 203-214.

Leutmezer, F., Schernthaner, C., Lurger, S., Pötzelberger, K. & Baumgartner, C. (2003). Electrocardiographic changes at the onset of epileptic seizures. *Epilepsia*, Vol. 44, No. 3, 348-354.

Miller, A.S., Blott, B.H., & Hames, T.K. (1992). Review of neural network applications in medical imaging and signal processing. *Medical & Biological Engineering & Computing*, Vol. 30, 449-464.

Narendra, K.S. & Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, Vol. 1, No. 1, 4-27.

Ogha, Y. & Seki, H. (1991). Using a neural network for abnormal event identification in BWRs. *Transactions of the American Nuclear Society*, Vol. 63, 110-111.

Ortiz, J.J. & Requena, I. (2004). Using a multi-state recurrent neural network to optimize loading patterns in BWRs. *Annals of Nuclear Energy*, Vol. 31, 789-803.

Petrosian, A., Prokhorov, D., Homan, R., Dasheiff, R. & Wunsch II, D. (2000). Recurrent neural network based prediction of epileptic seizures in intra- and extracranial EEG. *Neurocomputing*, Vol. 30, 201-218.

Petrosian, A.A., Prokhorov, D.V., Lajara-Nanson, W. & Schiffer, R.B. (2001). Recurrent neural network-based approach for early recognition of Alzheimer's disease in EEG. *Clinical Neurophysiology*, Vol. 112, No. 8, 1378-1387.

Pineda, F.J. (1987). Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, Vol. 59, No. 19, 2229-2232.

Reifman, J. (1997). Survey of artificial intelligence methods for detection and identification of component faults in nuclear power plants. *Nuclear Technology*, Vol. 119, 76-97.

Rocamora, R., Kurthen, M., Lickfett, L., von Oertzen, J. & Elger, C.E. (2003). Cardiac asystole in epilepsy: Clinical and neurophysiologic features. *Epilepsia*, Vol. 44, No. 2, 179-185.

Saad, E.W., Prokhorov, D.V. & Wunsch II, D.C. (1998). Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks. *IEEE Transactions on Neural Networks*, Vol. 9, No. 6, 1456-1470.

Saxena, S.C., Kumar, V. & Hamde, S.T. (2002). Feature extraction from ECG signals using wavelet transforms for disease diagnostics. *International Journal of Systems Science*, Vol. 33, No. 13, 1073-1085.

Shieh, J-S., Chou, C-F., Huang, S-J. & Kao, M-C. (2004). Intracranial pressure model in intensive care unit using a simple recurrent neural network through time. *Neurocomputing*, Vol. 57, 239-256.

Şeker, S., Ayaz, E. & Türkcan, E. (2003). Elman's recurrent neural network applications to condition monitoring in nuclear power plant and rotating machinery. *Engineering Applications of Artificial Intelligence*, Vol. 16, 647-656.

Thibault, J. & Grandjean, B.P.A. (1991). A neural network methodology for heat transfer data analysis. *International Journal of Heat Mass Transfer*, Vol. 34, No. 8, 2063-2070.

Uhrig, R.E. & Tsoukalas, L.H. (1999). Soft computing technologies in nuclear engineering applications. *Progress in Nuclear Energy*, Vol. 34, No. 1, 13-75.

Übeyli, M. & Acır, A. (2007). Utilization of thorium in a high power density hybrid reactor with innovative coolants. *Energy Conversion and Management*, Vol. 48, 576-582.

Übeyli, E.D. & Übeyli, M. (2007). Investigating neutronic parameters of a Thorium fusion breeder with recurrent neural networks. *Journal of Fusion Energy*, Vol. 26, No. 4, 323-330.

Übeyli, E.D. (2007a). Comparison of different classification algorithms in clinical decision-making. *Expert Systems*, Vol. 24, No. 1, 17-31.

Übeyli, E.D. (2007b). ECG beats classification using multiclass support vector machines with error correcting output codes. *Digital Signal Processing*, Vol.17, No. 3, 675-684.

Übeyli, E.D. (2008a). Recurrent neural networks employing Lyapunov exponents for analysis of Doppler ultrasound signals. *Expert Systems with Applications*, Vol. 34, No. 4, 2538-2544.

Übeyli, E.D. (2008b). Wavelet/mixture of experts network structure for EEG signals classification. *Expert Systems with Applications*, Vol. 34, No. 3, 1954-1962.

Übeyli, E.D. (2008c). Recurrent neural networks with composite features for detection of electrocardiographic changes in partial epileptic patients. *Computers in Biology and Medicine*, Vol. 38, No. 3, 401-410.

Wright, I.A., Gough, N.A.J., Rakebrandt, F., Wahab, M. & Woodcock, J.P. (1997). Neural network analysis of Doppler ultrasound blood flow signals: A pilot study. *Ultrasound in Medicine & Biology*, Vol. 23, 683-690.

Wright, I.A., & Gough, N.A.J. (1999). Artificial neural network analysis of common femoral artery Doppler shift signals: Classification of proximal disease. *Ultrasound in Medicine & Biology*, Vol. 24, 735-743.

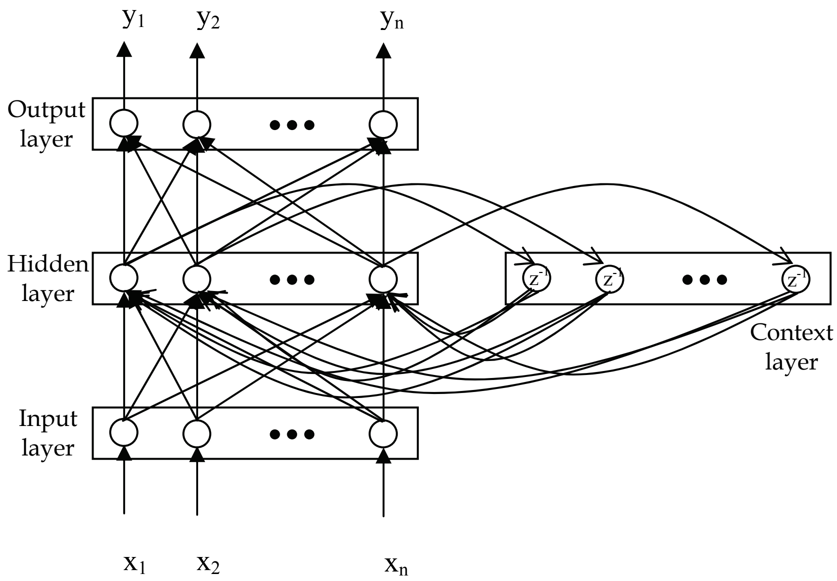


Figure 1. A schematic representation of an Elman recurrent neural network. z^{-1} represents a one time step delay unit.

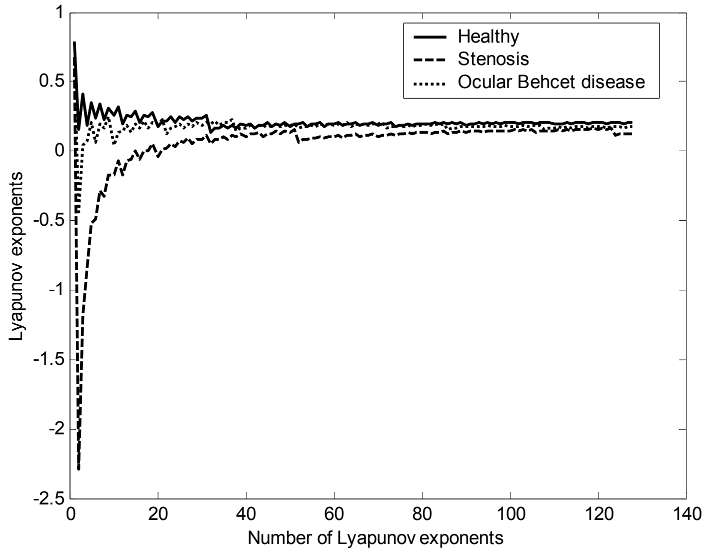


Figure 2. Lyapunov exponents of the OA Doppler signals: healthy subject (subject no: 12); subject suffering from OA stenosis (subject no: 27); subject suffering from ocular Behcet disease (subject no: 38)

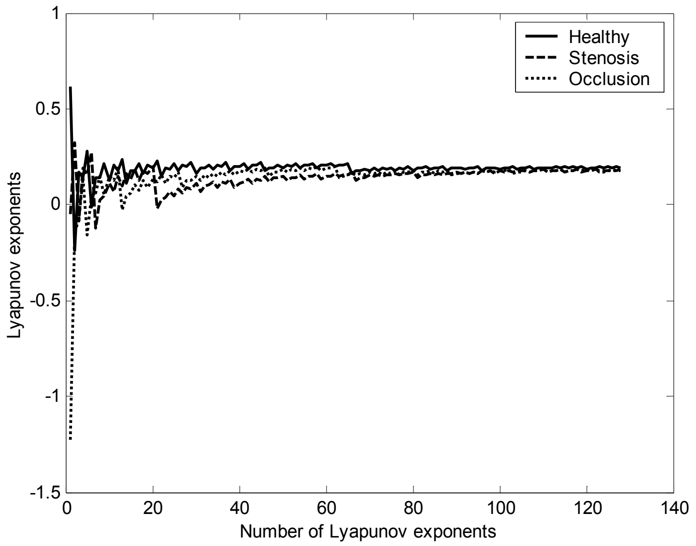
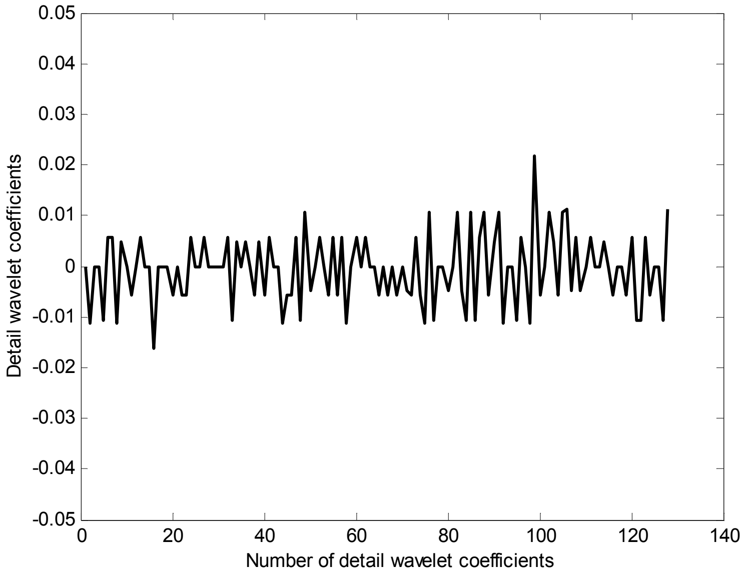
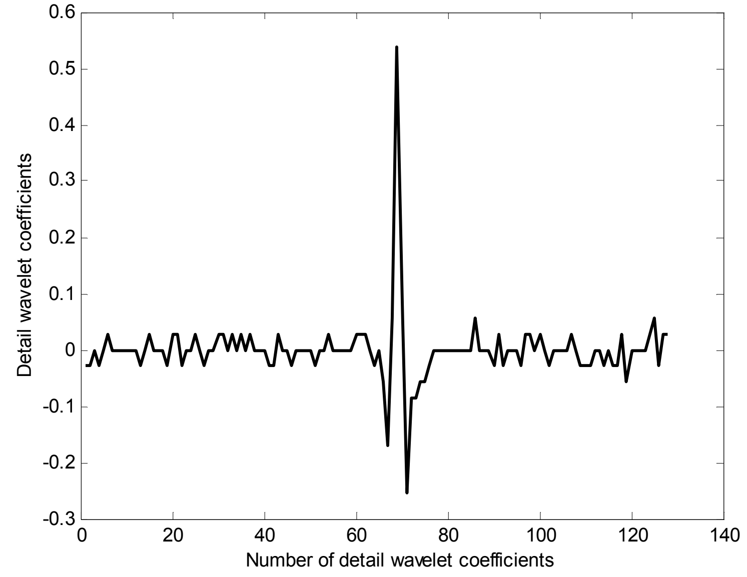


Figure 3. Lyapunov exponents of the ICA Doppler signals obtained from: a healthy subject (subject no: 15); a subject suffering from ICA stenosis (subject no: 32); a subject suffering from ICA occlusion (subject no: 43)

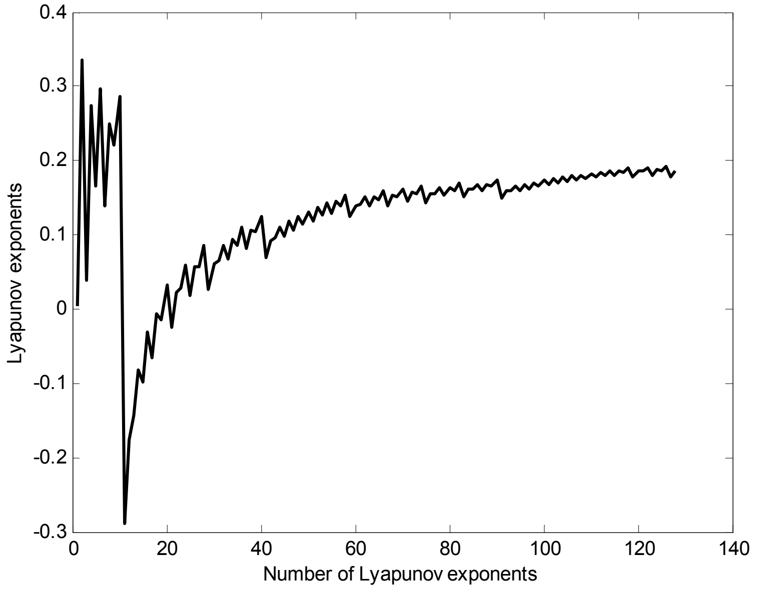


(a)

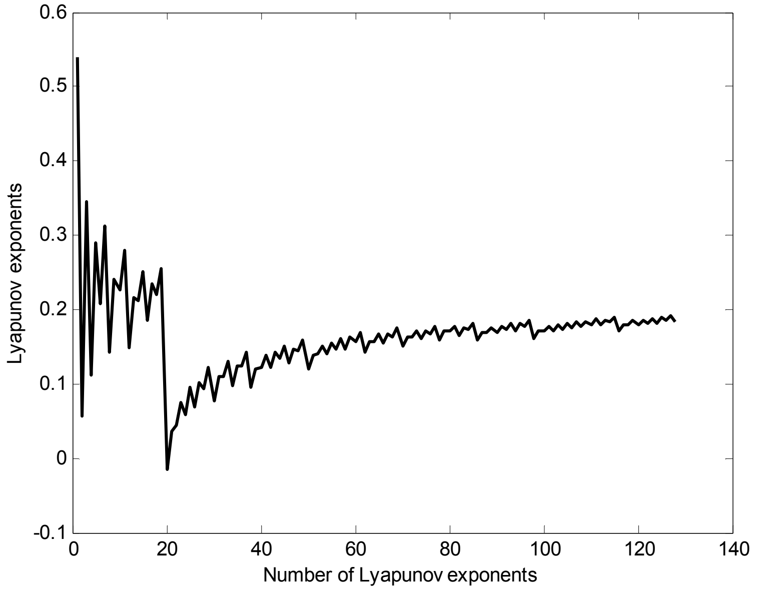


(b)

Figure 4. The detail wavelet coefficients at the first decomposition level of the ECG beats: (a) normal beat, (b) partial epilepsy beat



(a)



(b)

Figure 5. Lyapunov exponents of the ECG beats: (a) normal beat, (b) partial epilepsy beat

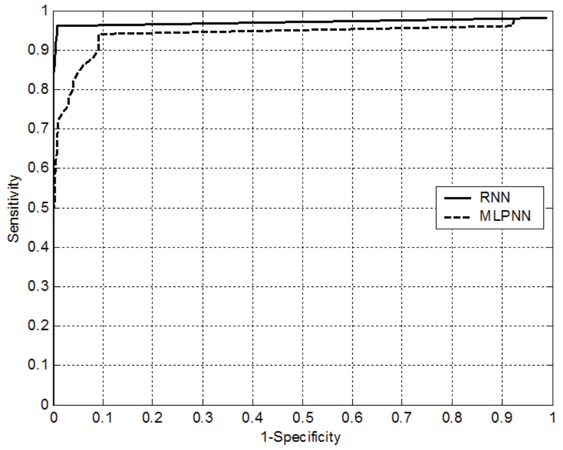


Figure 6. ROC curves of the classifiers used for classification of the ECG beats

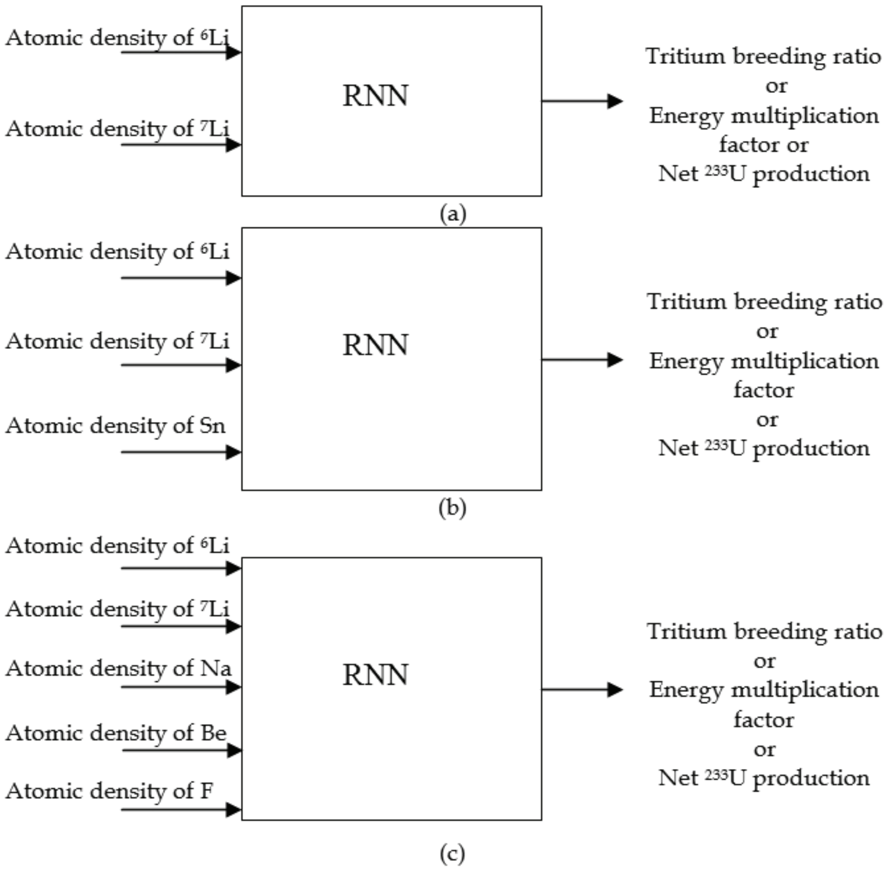

Atomic density of ^{6}Li
Atomic density of ^{7}Li
RNN
Tritium breeding ratio or Energy multiplication factor or Net ^{233}U production
(a)
Atomic density of ^{6}Li
Atomic density of ^{7}Li
Atomic density of Sn
RNN
Tritium breeding ratio or Energy multiplication factor or Net ^{233}U production
(b)
Atomic density of ^{6}Li
Atomic density of ^{7}Li
Atomic density of Na
Atomic density of Be
Atomic density of F
RNN
Tritium breeding ratio or Energy multiplication factor or Net ^{233}U production
(c)

Figure 7. Implemented RNNs for various coolant type (a) Natural Lithium (b) $Li_{20}S_{80}$ (c) Flinabe

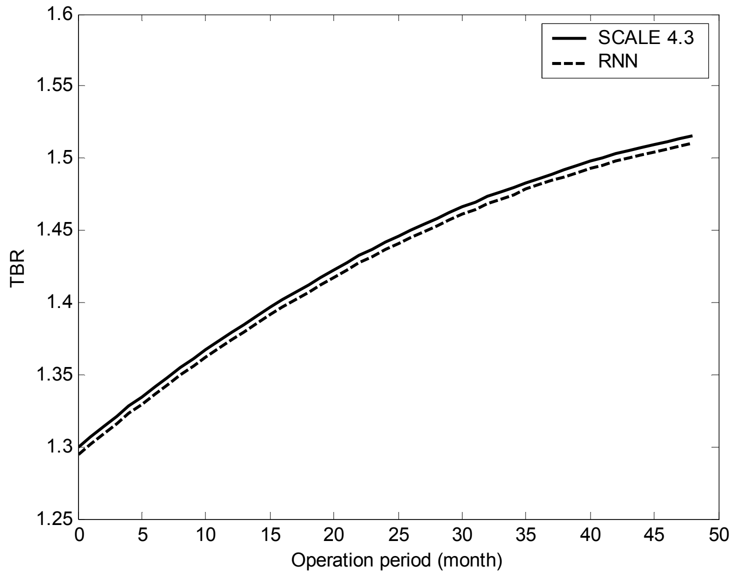


Figure 8. TBR variation in the blanket cooled with natural Li obtained by Scale 4.3 (Übeyli & Acır, 2007) and RNN

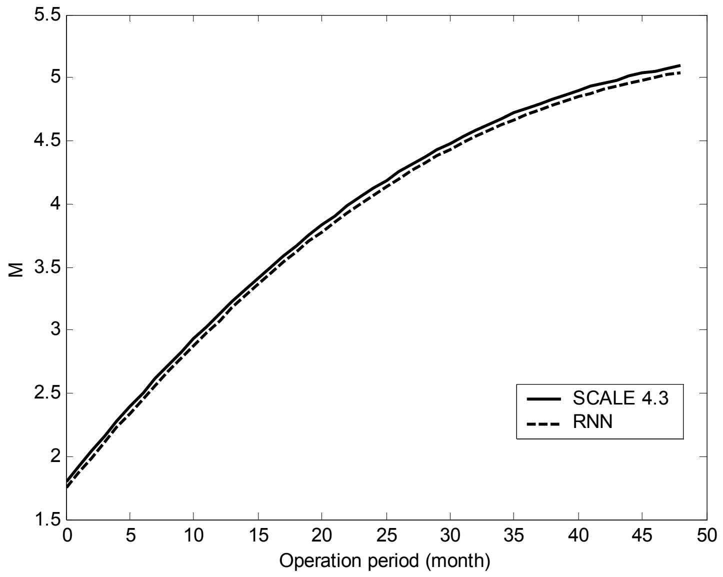


Figure 9. Change in M with respect to time in the blanket using $Li_{20}Sn_{80}$ obtained by Scale 4.3 (Übeyli & Acır, 2007) and RNN

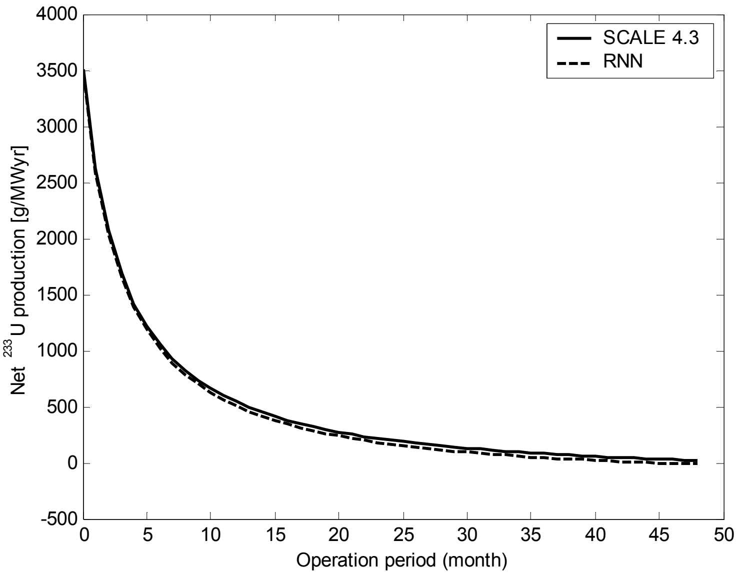


Figure 10. Net ^{233}U production in the blanket using Flinabe obtained by Scale 4.3 (Übeyli & Acır, 2007) and RNN

Desired Result	Output Result		
	Healthy	OA stenosis	Ocular Behcet disease
Healthy	41	0	0
OA stenosis	2	32	1
Ocular Behcet disease	0	0	33

Table 1. Confusion matrix of the RNN used for classification of the OA Doppler signals

Desired Result	Output Result		
	Healthy	ICA stenosis	ICA occlusion
Healthy	31	0	0
ICA stenosis	1	40	0
ICA occlusion	0	2	36

Table 2. Confusion matrix of the RNN used for classification of the ICA Doppler signals

Classifiers	CPU time (min:s)	Classification Accuracies	Values (%)
RNN implemented for OA Doppler signals	8:23	Specificity	95.35
		Sensitivity (OA stenosis)	100.00
		Sensitivity (Ocular Behcet disease)	97.06
		Total classification accuracy	97.25
RNN implemented for ICA Doppler signals	7:41	Specificity	96.88
		Sensitivity (ICA stenosis)	95.24
		Sensitivity (ICA occlusion)	100.00
		Total classification accuracy	97.27

Table 3. The classification accuracies and the CPU times of training of the classifiers used for classification of the OA and ICA Doppler signals

Classifiers	Statistical Parameters (%)			CPU time (min:s)
	Specificity	Sensitivity	Total classification accuracy	
RNN	98.89	97.78	98.33	12:34
MLPNN	92.22	93.33	92.78	17:05

Table 4. The values of the statistical parameters and the CPU times of training of the classifiers used for classification of the ECG beats

Coolant type	Performance	RNNs for tritium breeding ratio	RNNs for energy multiplication factor	RNNs for net ^{233}U production
Natural Lithium	MSE	0.009	0.005	0.008
	MAE	0.008	0.004	0.008
	r	0.892	0.921	0.945
$Li_{20}S_{80}$	MSE	0.089	0.006	0.007
	MAE	0.008	0.005	0.008
	r	0.899	0.934	0.905
Flinabe	MSE	0.009	0.005	0.006
	MAE	0.088	0.005	0.007
	r	0.895	0.924	0.936

Table 5. Performance evaluation parameters of the RNNs implemented for the estimation of the neutronic parameters of a thorium fusion breeder

Partially Connected Locally Recurrent Probabilistic Neural Networks

Todor D. Ganchev, Konstantinos E. Parsopoulos, Michael N. Vrahatis, and Nikos D. Fakotakis
University of Patras
Greece

1. Introduction

In this chapter, we review existing locally recurrent neural networks and introduce a novel artificial neural network architecture that merges the locally recurrent probabilistic neural networks (LRPNN) with swarm intelligence algorithms and concepts.
In particular, we develop an enhanced LRPNN model, referred to as *Partially Connected LRPNN* (PC-LRPNN). In contrast to LRPNN, where the recurrent layer consists of a set of fully connected neurons, the proposed new architecture assumes a *swarm* of neurons in the recurrent layer. Each neuron of the swarm presumes a neighbourhood of neurons with which it communicates through interconnections. The locality that determines the neighbourhoods is defined based on existing neighbourhood and communication schemes proposed in the swarm intelligence literature. Obviously, the PC-LRPNN offers a more general scheme, in which the fully connected LRPNN can be considered as a particular case, where all links in the recurrent layer are implemented.
The neighbourhood topology of the new, swarm-based recurrent layer can be either static or dynamic. Dynamic neighbourhoods have been studied extensively in the field of swarm intelligence, since swarms with dynamic communication schemes among individuals have been shown to achieve remarkably better results than swarms with static communication schemes in the field of optimization. Also, the plasticity of the neighbourhoods can be useful in cases where better fit to unknown data is required. In the present chapter we will limit our exposition to the static neighbourhoods, which are defined once during training, and remain unchanged during the operation of the PC-LRPNN. However, the concepts that we introduce here can be extended further to the dynamic counterparts.
The aforementioned local neighbourhoods and communications schemes facilitate the optimization of the recurrent layer linkage, which leads to much faster operation of the neural network, when compared to the fully linked structure. Furthermore, it significantly reduces the computational load for the overall training of the recurrent layer, which is performed at each case using the Particle Swarm Optimization (PSO) algorithm. Equipping the PC-LRPNN with PSO, results in an efficient hybrid scheme that takes advantage of the virtues of the probabilistic neural networks (PNN), recurrent neural networks (RNN), swarm intelligence concept, and that can tackle successfully real-life classification problems that assume temporal or spatial correlations among subsequent events.

2. Locally recurrent neural networks

A large number of recurrent and locally recurrent neural networks (LRNNs) have been studied in the literature. All they posses the valuable virtue to learn temporal dependences among the training data, which allows for context awareness, and thus, for improved recognition capabilities when compared to their non-recurrent counterparts. This advantage has proved useful in numerous applications of the LRNNs on real-life problems, which among others include: nonlinear system identification (Back & Tsoi, 1992; Lin et al., 1998); grammatical inference (Lin et al., 1998); weather prediction (Aussem et al., 1995); speech recognition (Kasper et al., (1995, 1996)); protection of power systems (Cannas et al., 1998); speaker verification (Ganchev et al., (2003, 2004, 2007)); wind speed prediction (Barbounis & Theocharis, (2007a, 2007b)), etc.

The locally recurrent global feedforward architecture was originally proposed by Back and Tsoi (Back & Tsoi, 1991), who considered an extension of the Multilayer Perceptron (MLP) neural network to exploit contextual information. In their work, they introduced the Infinite Impulse Response (IIR) and Finite Impulse Response (FIR) synapses, able to utilize temporal dependencies in the input data. The FIR synapse has connections to its own, current and delayed, inputs, while the IIR synapse has also connections to its past outputs.

Ku and Lee (Ku & Lee, 1995) proposed Diagonal Recurrent Neural Networks (DRNN) for the task of system identification in real-time control applications. Their approach is based on the assumption that a single feedback from the neuron's own output is sufficient. Thus, they simplify the fully connected neural network to render training easier.

A comprehensive study of several MLP-based Locally Recurrent Neural Networks is available in (Campolucci et al., 1999). They introduced a unifying framework for the gradient calculation techniques, called Causal Recursive Back-Propagation. All aforementioned approaches consider gradient-based training techniques for neural networks, which, as it is well known, require differentiable transfer functions.

From the abundance of LRNN, in the present work, we will consider primary architectures originating from the family of the Probabilistic Neural Network (PNN). Specifically in the present section we will briefly outline the Locally Recurrent Probabilistic Neural Network (LRPNN), which was introduced (Ganchev et al., 2003) as an extension of the feed-forward Probabilistic Neural Network (PNN) architecture (Specht, (1988, 1990)). This structure is used as basis for the novel partially connected LRPNN (PC-LRPNN), which we will discuss in the next sections.

In brief, the LRPNN was derived from the original PNN by incorporating an additional hidden layer, referred to as recurrent layer, between the summation layer and the output competitive layer of the PNN structure. The recurrent layer consists of neurons possessing feedbacks from all other neurons in that layer. Due to this recurrent layer, the LRPNN, in contrast to the original PNN, is sensitive to the context in which the individual input data appear, and thus, it is capable to learn temporal regularities and the sequence of occurrence of events. Specifically, in the frame of speech processing this new capability of the LRPNN enables detecting and exploiting the abundance of correlations among speech features vectors estimated for successive speech frames. Exploiting these correlations was found important for improving the classification accuracy in the speaker verification task (Ganchev et al., (2003, 2004, 2007)).

As presented in earlier studies (Ganchev et al., (2003, 2004)) in the LRPNN architecture each neuron in the recurrent layer receives as input not only current values of its inputs, but also the N previous outputs of all neurons in that layer. Broadly speaking, the input, acting on a recurrent neuron located in the recurrent hidden layer of an LRPNN, is a sum of two differences: The first difference is between the weighted probability of the given class and the sum of weighted probabilities computed for all other classes. These probabilities are computed at the output of the summation layer of the LRPNN. The second difference is between the weighted past output values of the given unit and the sum of the weighted past output values of all other neurons in this layer. Thus, in the proposed architecture, the probability of belonging to a specific class is combined with the probabilities computed for the other classes, and more importantly with the past values of the outputs of the recurrent units for all classes. This incorporation of previous information enables the LRPNN network to take advantage of the temporal context, which results in producing smoother in the time output scores, improved confidence levels, and consequently more accurate final decisions.

In the present chapter, we elaborate further on the LRPNN architecture by studying ways to optimize the recurrent layer linkage. In contrast to LRPNN, where the recurrent layer consists of a set of fully connected neurons, the introduced here new PC-LRPNN architecture assumes a *swarm* of neurons in the recurrent layer. Each neuron of the swarm presumes a neighbourhood of neurons with which it communicates through interconnections. The locality that determines the neighbourhoods is defined based on existing neighbourhood and communication schemes proposed in the swarm intelligence literature. When compared to the original LRPNN architecture, the PC-LRPNN has a greater capacity to adapt (its recurrent layer linkage) to the training dataset. This is due to the additional degree of freedom provided by the recurrent layer linkage selection that can be controlled for a fine-tuning of the neural network to the problem at hand. Obviously, the fully connected LRPNN architecture can be regarded as a particular case of the PC-LRPNN, which implements the full linkage in the recurrent layer.

3. Particle swarms and particle swarm optimization

The *particle swarm* is a community of individual performers, known as *particles*, which communicate/share information and collaborate on finding optimal regions in the search space. In the literature, the particle swarm is synonym to Particle Swarm Optimization (PSO) algorithm, which has become an attractive alternative to other optimization techniques (Clerc and Kennedy, 2002).

In brief, PSO is a stochastic optimization, population-based algorithm. It was introduced in 1995 by Kennedy and Eberhart (Kennedy & Eberhart, 1995), inspired by social behaviour simulation models. Features such as information exchange and neighbour alignment are inherent in such models, allowing the emergence of intelligent behaviour in swarms of simple agents with limited field of action. Similarly to evolutionary algorithms, PSO exploits a population, called a *swarm*, of potential solutions, called *particles*, which adapt their position stochastically at each iteration of the algorithm.

In contrast to standard evolutionary approaches, PSO promotes cooperativeness rather than competition among the solutions. More specifically, instead of using explicit mutation and selection operators in order to modify the population and favour the best performing individuals, PSO uses an adaptable position shift, called *velocity*, to move each particle to a new position at each iteration of the algorithm. The particles are moving towards promising regions of the search space by exploiting information springing from their own experience during the search as well as from the experience of other particles. For this purpose, a memory of the best position ever visited by each particle in the search space is retained.

In the context of single-objective optimization, the PSO can be outlined formally as follows: Let S be an n-dimensional search space, $f : S \rightarrow \mathbb{R}$ be the objective function, and N be the number of particles that comprise the swarm,

$$\mathbb{S} = \{x_1, x_2, \ldots, x_N\}. \tag{1}$$

Then, the ith particle is a point in the search space,

$$x_i = (x_{i1}, x_{i2}, \ldots, x_{in}) \in S, \tag{2}$$

as well as its best position,

$$p_i = (p_{i1}, p_{i2}, \ldots, p_{in}) \in S, \tag{3}$$

which is the best position ever visited by x_i during the search. The velocity of x_i is also an n-dimensional vector,

$$v_i = (v_{i1}, v_{i2}, \ldots, v_{in}). \tag{4}$$

In order to avoid biasing the swarm in specific parts of the search space, the particles as well as their velocities are randomly initialized in the search space.

Let $NG_i \subseteq \mathbb{S}$ be a set of particles that exchange information with x_i. This set is called the *neighbourhood* of x_i and it will be discussed later. Let also, g, be the index of the best particle in NG_i, i.e.,

$$f(p_g) \leq f(p_l), \qquad \text{for all } l \text{ with } x_l \in NG_i, \tag{5}$$

and t denote the iteration counter. Then, the swarm is manipulated according to the equations (Eberhart & Shi, 2000),

$$v_{ij}(t+1) = w\, v_{ij}(t) + c_1\, r_1\, (p_{ij}(t) - x_{ij}(t)) + c_2\, r_2\, (p_{gj}(t) - x_{ij}(t)), \tag{6}$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1), \tag{7}$$

where $i = 1, 2, \ldots, N$; $j = 1, 2, \ldots, n$; w is a positive parameter called *inertia weight*; c_1 and c_2 are two positive constants called *cognitive* and *social* parameter, respectively; and r_1, r_2, are realizations of two independent random variables that assume the uniform distribution in the range [0, 1]. The best position of each particle is updated at each iteration by setting

$$p_i(t+1) = x_i(t+1), \qquad \text{if } f(x_i) < f(p_i), \tag{8}$$

otherwise it remains unchanged. Obviously, an update of the index g is also required at each iteration.

The inertia weight was not used in early PSO versions. However, experiments showed that the lack of mechanism for controlling the velocities could result in *swarm explosion*, i.e., an unbounded increase in the magnitude of the velocities, which resulted in swarm divergence. For this purpose, a boundary, $v_{\max}$, was imposed on the absolute value of the velocities, such that, if $v_{ij} > v_{\max}$ then $v_{ij} = v_{\max}$, and if $v_{ij} < -v_{\max}$ then $v_{ij} = -v_{\max}$. In later, more sophisticated versions, the new parameter was incorporated in the velocity update equation, in order to control the impact of the previous velocity on the current one, although the use of $v_{\max}$ was not abandoned.

Intelligent search algorithms, such as PSO, must demonstrate an ability to combine *exploration*, i.e., visiting new regions of the search space, and *exploitation*, i.e., performing more refined local search, in a balanced way in order to solve problems effectively (Parsopoulos & Vrahatis, (2002, 2004, 2007)). Since larger values of w promote exploration, while smaller values promote exploitation, it was proposed and experimentally verified that declining values of the inertia weight can provide better results than fixed values. Thus, an initial value of w around 1.0 and a gradually decline towards 0.0 are considered a good choice. On the other hand, the parameters c_1 and c_2 are usually set to fixed and equal values such that the particle is equally influenced by its own best position, p_i, as well as the best position of its neighbourhood, p_g, unless the problem at hand implies a different setting.
An alternative velocity update equation was proposed by Clerc & Kennedy, (2002),

$$v_{ij}(t+1) = \chi\,[v_{ij}(t) + c_1 r_1 (p_{ij}(t) - x_{ij}(t)) + c_2 r_2 (p_{gj}(t) - x_{ij}(t))], \tag{9}$$

where χ is a parameter called *constriction factor*. This version is algebraically equivalent with the inertia weight version of (6). However, the parameter selection in this case is based on the stability analysis due to Clerc and Kennedy (2002), which expresses χ as a function of c_1 and c_2. Different promising models were derived through the analysis of the algorithm, with the setting $\chi = 0.729$, $c_1 = c_2 = 2.05$, providing the most promising results and robust behaviour, rendering it the default PSO parameter setting.
Regardless of the PSO version used, it is clear that its performance is heavily dependent on the information provided by the best positions, p_i and p_g, since they determine the region of the search space that will be visited by the particle. Therefore, their selection, especially for p_g, which is related to information exchange, plays a central role in the development of effective and efficient PSO variants. Moreover, the concept of neighbourhood mentioned earlier in this section, raises efficiency issues. A neighbourhood has been already defined as a subset of the swarm. The most straightforward choice would be to consider as neighbours of the particle x_i, all particles enclosed in a sphere with centre x_i and a user-defined radius in the search space. Despite its simplicity, this approach increases significantly the computational burden of the algorithm, since it requires the computation of all distances among particles at each iteration. This deficiency has been addressed by defining neighbourhoods in the space of particles' indices instead of the actual search space.
Thus, the neighbours of x_i are determined based solely on the indices of the particles, assuming different *neighbourhood topologies*, i.e., orderings of the particles' indices. The most common neighbourhood is the *ring topology*, depicted in Fig. 1 (left), where the particles are arranged on a ring, with x_{i-1} and x_{i+1} being the immediate neighbours of x_i, and x_1 following immediately after x_N. Based on this topology, a neighbourhood of radius r of x_i is defined as

$$NG_i(r) = \{x_{i-r}, x_{i-r+1}, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_{i+r-1}, x_{i+r}\}, \tag{10}$$

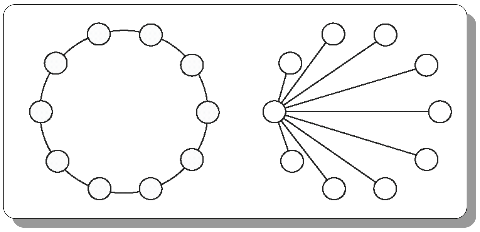


Fig. 1. The ring (left) and star (right) neighbourhood topologies of PSO

and the search is influenced by the particle's own best position, p_i, as well as the best position of its neighbourhood. This topology promotes exploration, since the information carried by the best positions is communicated slowly through the neighbours of each particle. A different topology is the *star topology*, depicted in Fig. 1 (right) where all particles communicate only with a single particle, which is the overall best position, p_g, of the swarm, i.e., $NG_i \equiv \mathbb{S}$. This topology promotes exploitation, since all particles share the same information. This is also called the *global variant* of PSO, denoted as *gbest* in the relative literature, while all other topologies with $NG_i \subset \mathbb{S}$, define *local variants*, usually denoted as *lbest*. Different topologies have also been investigated with promising results (Kennedy, 1999; Janson & Middendorf, 2005).

4. The partially connected locally recurrent probabilistic neural network

The LRPNN was derived (Ganchev et al., 2003) from the original PNN (Specht, 1988) by incorporating an additional hidden layer, referred to as *recurrent layer*, between the summation layer and the output competitive layer of the PNN structure. This recurrent layer consists of neurons possessing feedbacks with all other neurons in that layer. Elaborating on the LRPNN, here, we introduce the *Partially Connected LRPNN* (PC-LRPNN) architecture. Fig. 2 presents the simplified structure of a PC-LRPNN for classification in K classes. In contrast to the fully connected LRPNN, where each neuron in the recurrent layer communicates with all other neurons in that layer (i.e. *global communication* is enabled), in the PC-LRPNN the recurrent layer linkage is implemented only partially, depending on the problem at hand and the actual training data. This is illustrated in Fig. 2, where the

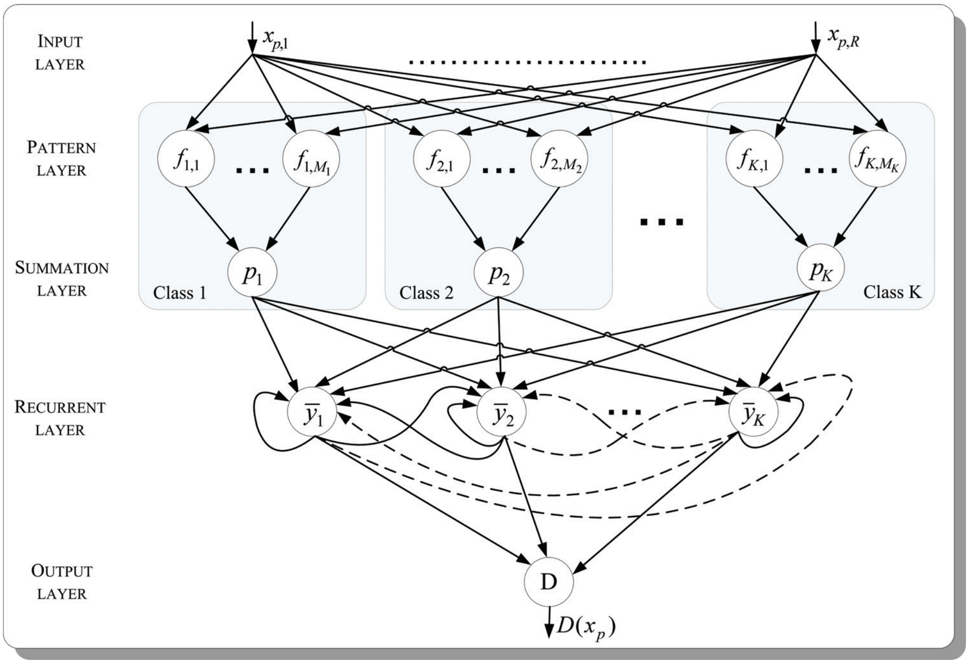


Fig. 2. Structure of the Partially Connected Locally Recurrent Probabilistic Neural Network

dashed line indicates that the linkage between neurons $\bar{y}_1$, $\bar{y}_2$ and $\bar{y}_K$ might not be implemented. In general, the concept of partially connected recurrent layer can be regarded as defining local neighbourhoods for each of the recurrent layer neurons. This can be viewed as establishing a *swarm of neurons* which cooperate (i.e. exchange information) in order to categorize more precisely a given unknown input. However, in contrast to the classic particle swarms that are utilized in the PSO schemes, here the local neural neighbours are not defined by the specific values of the neurons' indexes but the swarm members are selected during training, on a competitive basis, and in data-dependent manner, with respect to certain predefined criterion. In practice, the size of neighbourhood and the recurrence depth (i.e. the depth of memory) in the recurrent layer are specified depending on *a priori* knowledge about the specific problem at hand, or are identified heuristically after some experimentation with a representative dataset.

However, before describing any specific strategy for implementing the (partial) linkage of the recurrent layer, for comprehensiveness of exposition we briefly outline the PC-LRPNN architecture. In brief, the first two hidden layers the PC-LRPNNs, as their predecessor — the PNNs, implement the Parzen window estimator (Parzen, 1962) by using a mixture of Gaussian basis functions. If a PC-LRPNN for classification in K classes is considered, the class conditional probability density function $p_i(\mathbf{x}_p \mid k_i)$ is defined as:

$$p_i(\mathbf{x}_p \mid k_i) = \mathrm{f}_i(\mathbf{x}_p) = \frac{1}{(2\pi)^{d/2}\sigma_i^d} \cdot \frac{1}{M_i}\sum_{j=1}^{M_i} \exp\left(-\frac{1}{2\sigma_i^2}(\mathbf{x}_p - \mathbf{x}_{ij})^T(\mathbf{x}_p - \mathbf{x}_{ij})\right),\ i = 1,2,\dots,K, \quad (11)$$

where for simplicity of further notations $p_i(\mathbf{x}_p \mid k_i)$ is replaced by $\mathrm{f}_i(\mathbf{x}_p)$. Here $\mathbf{x}_{ij}$ is the j th training vector from class κ_i, $\mathbf{x}_p$ belonging to the set $\mathbf{X} = \{\mathbf{x}_p\}$, with $p = 1,2,\dots,P$, is the p th input vector, d is the dimension of the input vectors, and M_i is the number of training patterns in class κ_i. Each training vector $\mathbf{x}_{ij}$ is assumed a centre of a kernel function, and consequently the number of pattern units in the first hidden layer of the neural network is given by the sum of the pattern units for all the classes. The standard deviation σ_i acts as a smoothing factor, which softens the surface defined by the multiple Gaussian functions. Instead of the simple covariance matrix, $\{\sigma_i^2 I\}$, where I represents the identity matrix, the full covariance matrix can be computed using the Expectation Maximization algorithm, as proposed in (Yang & Chen, 1998; Mak & Kung, 2000) and elsewhere. Since the computation of the covariance matrix, or the optimization of the smoothing factor σ_i, does not interfere with the development of the PNN we discuss, for simplicity of exposition, we consider here the simple case, where the value of the standard deviation is identical for all pattern units belonging to a specific class. Moreover, σ_i can be the same for all pattern units, irrespective of their class belonging, as it was originally proposed (Specht, 1990).

Next, the class conditional probability density functions $\mathrm{f}_i(\mathbf{x}_p)$ for each class κ_i, estimated through (11), act as inputs for the recurrent layer. In general, the recurrent layer can be considered as a form of Infinite Impulse Response (IIR) filter that smoothes the probabilities generated for each class, by incorporating information about the probabilities computed for all other classes, and more importantly, by exploiting one or more past values of the outputs for all classes.

The recurrent layer is composed of recurrent neurons, which in addition to the inputs coming from the summation layer also possess feedbacks from their own past outputs and from current and past outputs of the neurons of the other classes. Fig. 3 illustrates the linkage of a single neuron belonging to the hidden recurrent layer. As shown in the figure, beside the PDFs from all classes, $\mathrm{f}_i(\mathbf{x}_p)$, $i=1,2,...,K$, this neuron also receives feedbacks from its past outputs, $\overline{\mathrm{y}}_i(\mathbf{x}_{p-t})$, $t=1,2,...,N$, with i denoting the current neuron number, as well as from current $\mathrm{y}'_{j\neq i}(\mathbf{x}_p)$, $j=1,2,...,K$ and past $\overline{\mathrm{y}}_{j\neq i}(\mathbf{x}_{p-t})$, $j=1,2,...,K$, $t=1,2,...,N$, outputs from all other neurons belonging to that layer. Here, the subscript p stands for the serial number of the input vector $\mathbf{x}_p$. On its own side, the current neuron provides to the other neurons of the recurrent layer its current $\mathrm{y}_i(\mathbf{x}_p)$ and past $\overline{\mathrm{y}}_i(\mathbf{x}_{p-t})$, $t=1,2,...,N$ outputs, again with p standing for the specific input vector.

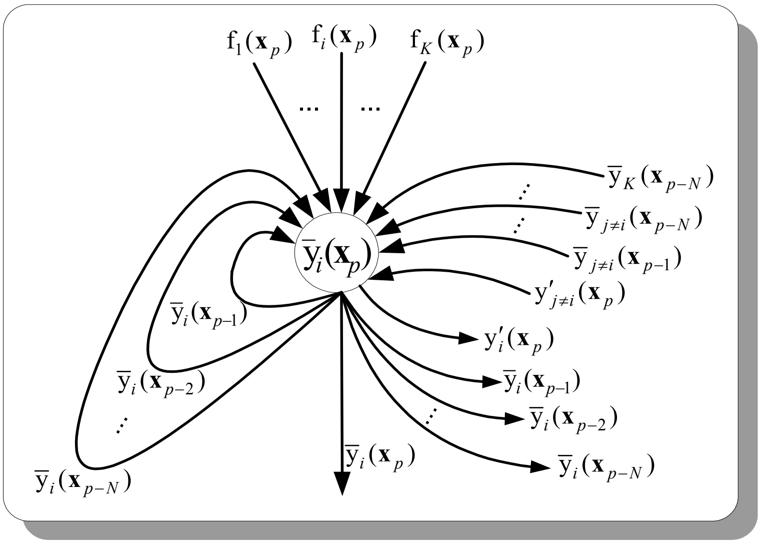

$\mathrm{f}_1(\mathbf{x}_p)$
$\mathrm{f}_i(\mathbf{x}_p)$
$\mathrm{f}_K(\mathbf{x}_p)$
$\overline{\mathrm{y}}_K(\mathbf{x}_{p-N})$
$\overline{\mathrm{y}}_{j\neq i}(\mathbf{x}_{p-N})$
$\overline{\mathrm{y}}_{j\neq i}(\mathbf{x}_{p-1})$
$\mathrm{y}'_{j\neq i}(\mathbf{x}_p)$
$\overline{\mathrm{y}}_i(\mathbf{x}_p)$
$\overline{\mathrm{y}}_i(\mathbf{x}_{p-1})$
$\overline{\mathrm{y}}_i(\mathbf{x}_{p-2})$
$\overline{\mathrm{y}}_i(\mathbf{x}_{p-N})$
$\mathrm{y}'_i(\mathbf{x}_p)$
$\overline{\mathrm{y}}_i(\mathbf{x}_{p-1})$
$\overline{\mathrm{y}}_i(\mathbf{x}_{p-2})$
$\overline{\mathrm{y}}_i(\mathbf{x}_p)$
$\overline{\mathrm{y}}_i(\mathbf{x}_{p-N})$

Fig. 3. Linkage of a neuron that belongs to the recurrent layer

A detailed structure of the recurrent neurons is provided in Fig. 4. As the figure presents, the inputs $\mathrm{f}_i(\mathbf{x}_p)$, $i=1,2,...,K$, denoting the class conditional PDFs, are weighted by the coefficients $b_{i,j}$. The two indexes of the weights of $b_{i,j}$ with $i=1,2,...,K$ and $j=1,2,...,K$ stand for the current recurrent neuron and for the class to which the corresponding input belongs. The first two indexes of the weights $a_{i,j,t}$ have the same meaning as for the weights $b_{i,j}$, and the third index $t=1,2,...,N$ shows the time delay of the specific output before it appear as an input.

All feedbacks $\overline{\mathrm{y}}_i(\mathbf{x}_{p-t})$, $t=1,2,...,N$ that originate from the present neuron i, and the links $\overline{\mathrm{y}}_{j\neq i}(\mathbf{x}_{p-t})$, $j=1,2,...,K$, $t=1,2,...,N$ coming from the other neurons $j\neq i$ of the recurrent layer are weighted by the coefficients $a_{i,i,t}$, $t=1,2,...,N$ and $a_{i,j\neq i,t}$, $j=1,2,...,K$, $t=1,2,...,N$, respectively.

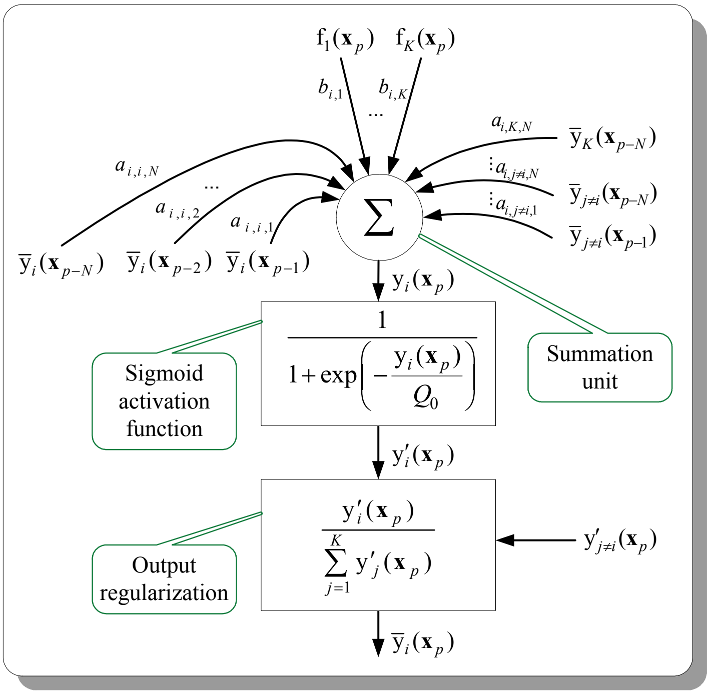

$f_1(\mathbf{x}_p)$
$f_K(\mathbf{x}_p)$
$b_{i,1}$
$b_{i,K}$
$a_{i,i,N}$
$a_{i,i,2}$
$a_{i,i,1}$
$a_{i,K,N}$
$a_{i,j\neq i,N}$
$a_{i,j\neq i,1}$
$\bar{y}_K(\mathbf{x}_{p-N})$
$\bar{y}_{j\neq i}(\mathbf{x}_{p-N})$
$\bar{y}_{j\neq i}(\mathbf{x}_{p-1})$
$\bar{y}_i(\mathbf{x}_{p-N})$
$\bar{y}_i(\mathbf{x}_{p-2})$
$\bar{y}_i(\mathbf{x}_{p-1})$
Σ
$y_i(\mathbf{x}_p)$
$\frac{1}{1+\exp\left(-\frac{y_i(\mathbf{x}_p)}{Q_0}\right)}$
Sigmoid activation function
Summation unit
$y'_i(\mathbf{x}_p)$
$\frac{y'_i(\mathbf{x}_p)}{\sum_{j=1}^{K} y'_j(\mathbf{x}_p)}$
$y'_{j\neq i}(\mathbf{x}_p)$
Output regularization
$\bar{y}_i(\mathbf{x}_p)$

Fig. 4. Internal structure of the *i*th neuron from the recurrent layer of the PC-LRPNN

The summation units' output $y_i(\mathbf{x}_p)$ of the locally recurrent layer is computed by:

$$y_i(\mathbf{x}_p) = \left[b_{i,i}\, f_i(\mathbf{x}_p) - \sum_{\substack{k=1 \\ i\neq k}}^{K} b_{i,k}\, f_k(\mathbf{x}_p) \right] + \sum_{t=1}^{N} \left[a_{i,i,t}\, \bar{y}_i(\mathbf{x}_{p-t}) - \sum_{\substack{k=1 \\ i\neq k}}^{K} a_{i,k,t}\, \bar{y}_k(\mathbf{x}_{p-t}) \right],\ i = 1,2,\ldots,K, \qquad (12)$$

where $f_i(\mathbf{x}_p)$ is the probability density function of each class κ_i, $\mathbf{x}_p$ is the p th input vector, K is the number of classes, N is the recurrence depth, $\bar{y}_i(\mathbf{x}_{p-t})$ is the normalized past output for class κ_i that has been delayed on t time steps, and $a_{i,j,t}$ and $b_{i,j}$ are weight coefficients. The output $y_i(\mathbf{x}_p)$ of each summation unit from the recurrent layer is subject to the regularization transformation:

$$\bar{y}_i(\mathbf{x}_p) = \frac{\operatorname{sgm}\left(y_i(\mathbf{x}_p)\right)}{\sum_{j=1}^{K} \operatorname{sgm}\left(y_j(\mathbf{x}_p)\right)},\ i = 1,2,\ldots,K, \qquad (13)$$

which retains the probabilistic interpretation of the output of the recurrent layer. Here, the designation sgm refers to the sigmoid activation function.

Subsequently, in the output layer, often referred as *competitive layer*, the Bayesian decision rule (14) is applied to distinguish class κ_i, to which the input vector $\mathbf{x}_p$ is categorized:

$$\mathrm{D}(\mathbf{x}_p) = \underset{i}{\operatorname{argmax}}\left\{h_i c_i \,\overline{\mathrm{y}}_i(\mathbf{x}_p)\right\},\ i = 1,2,\ldots,K, \tag{14}$$

where h_i is *a priori* probability of occurrence of a pattern from class κ_i, and c_i is the cost function associated with the misclassification of a vector belonging to class κ_i.
Finally, provided that all classes are mutually exclusive and exhaustive, we can compute the Bayesian confidence for every decision $\mathrm{D}(\mathbf{x}_p)$ by applying the Bayes' theorem:

$$P(k_i \,|\, \mathbf{x}_p) = \frac{h_i \,\overline{\mathrm{y}}_i(\mathbf{x}_p)}{\sum_{j=1}^{K} h_j \,\overline{\mathrm{y}}_j(\mathbf{x}_p)},\ i = 1,2,\ldots,K. \tag{15}$$

The posterior probability $P(k_i \,|\, \mathbf{x}_p)$ for the pth input vector belonging to class κ_i is computed by relying on the *a priori* probabilities h_i and the temporally smoothed PDFs $\overline{\mathrm{y}}_i(\mathbf{x}_p)$.
The decision $\mathrm{D}(\mathbf{x}_p)$, and the confidence for every decision $P(k_i \,|\, \mathbf{x}_p)$, are computed for every input vector. However, in many practical applications (such as speaker verification, speaker identification, emotion detection, etc) every test trial (usually a speech utterance) consists of multiple feature vectors. Therefore, the probability $P(k_i \,|\, \mathbf{X})$ all test vectors originating from a given test trial $\mathbf{X} = \{\mathbf{x}_p\},\ p = 1,2,\ldots,P$ to belong to class κ_i, can be computed by:

$$P(k_i \,|\, \mathbf{X}) = \frac{\sum_{p=1}^{P}\left[\mathrm{D}(\mathbf{x}_p) = k_i\right]}{\sum_{j=1}^{K}\sum_{p=1}^{P}\left[\mathrm{D}(\mathbf{x}_p) = k_j\right]},\ \ i = 1,2,\ldots,K, \tag{16}$$

where $\sum_{p=1}^{P}\left[\mathrm{D}(\mathbf{x}_p) = k_i\right]$ is the number of vectors $\mathbf{x}_p$ classified by the Bayesian decision rule (14) as belonging to class κ_i. In applications that assume an exhaustive taxonomy any of the inputs $\mathbf{x}_p$ falls in one of the classes κ_i, and therefore the equality:

$$P = \sum_{j=1}^{K}\sum_{p=1}^{P}\left[\mathrm{D}(\mathbf{x}_p) = k_j\right], \tag{17}$$

where P is the number of test vectors in the given trial $\mathbf{X}$, is always preserved.
However, in many real-world applications computing the probability $P(k_i \,|\, \mathbf{X})$ is not sufficient as a final outcome from the PC-LRPNN. In such cases, a final decision is made by applying the Bayesian decision rule:

$$D(\mathbf{X}) = \underset{i}{\operatorname{argmax}}\left\{P(k_i \,|\, \mathbf{X})\right\},\ \ i = 1,2,\ldots,K, \tag{18}$$

or alternatively, the outcome of (16) is assessed with respect to a predefined threshold θ:

$$P(k_i \mid \mathbf{X}) \begin{cases} > \theta & \text{decision \#1} \\ \leq \theta & \text{decision \#2} \end{cases} . \tag{19}$$

Most often, the threshold θ is computed on a data set, referred to as development or validation data, which is independent from the training and testing data. A necessary requirement for obtaining a reasonable estimate of θ is the development data to be representative, i.e., they have to bear a resemblance to the real-world data on which the PC-LRPNN will operate within the corresponding application.

5. Training the PC-LRPNN

In general, the training of the PC-LRPNNs is similar to the three-step training procedure of the original fully connected LRPNNs (Ganchev et al, 2004) except for one extra step that is PC-LRPNN specific. Specifically, in the LRPNN, the first two steps implement the usual strategy for training PNNs, while the third step adjusts the weights in the recurrent layer. In the PC-LRPNN the third training step is preceded by procedure which selects the actual linkage that will be implemented in the recurrent layer, i.e. the PC-LRPNN are trained in four steps. In the following we provide a concise description of the entire training process of the PC-LRPNN.

Step 1: In brief, by analogy to the original PNN, the first training step creates the actual topology of the network. In the first hidden layer, a pattern unit for each training vector is created by setting its weight vector equal to the corresponding training vector. In order to reduce the amount of neurons, i.e. the computational load during operation, the training data can be compressed by performing some sort of clustering (for instance, k-means) as pre-processing of the training dataset. An alternative approach could be to employ pruning and discard redundant neurons, or to build the first layer in multistep manner by adding a new neuron only when there is compelling need this to be done.
The outputs of the pattern units associated with the class κ_i are then connected to one of the second hidden layer summation units. The number of summation units is equal to the number of target classes K. The outputs of the summation units can be fed to some or all neurons of the recurrent layer, depending on the implemented linkage.

Step 2: The second training step is the computation of the smoothing parameter σ_i for each class. To this end, various approaches (Meisel, 1972; Cain, 1990; Specht, 1992; Musavi et al., 1992; Specht & Romsdahl, 1994; Masters, 1993; Georgiou et al., (2006, 2008), etc) have been proposed. Although other methods can be employed, here we will mention only the one (Cain, 1990) due to its simplicity. According to that approach, any σ_i is proportional to the mean value of the minimum distances among the training vectors in class κ_i:

$$\sigma_i = \lambda \frac{1}{M_i} \sum_{j=1}^{M_i} \min \left\{ \left\| \mathbf{x}_{i,j} - \mathbf{x}_{i,j \neq i} \right\|_2^2 \right\} \tag{20}$$

where $\mathbf{x}_{i,j}$ is the j th pattern unit (located in the pattern layer) for class κ_i; $\| \, . \, \|_2$ corresponds to the 2-norm on $\mathbb{R}^d$ (reminding that $\mathbf{x}_{i,j}$ are the stored training data, and therefore, $\mathbf{x}_{i,j} \in \mathbb{R}^d$); d is the dimensionality of the input data; the expression

$$\min\left\{\left\|\mathbf{x}_{i,j}-\mathbf{x}_{i,j\neq i}\right\|_2^2\right\} \tag{21}$$

represents the smallest Euclidean distance computed between j th pattern unit of class κ_i and all other pattern units from the same class; and M_i is the number of training patterns in class κ_i. The constant λ, which controls the degree of overlapping among the individual Gaussian functions, is usually selected in the range $\lambda \in [1.1, 1.4]$. If the smoothing parameter is common for all classes, either it is chosen empirically, or it is computed by applying (20) on the entire training data set.

Step 3: For the PC-LRPNNs, the third training step selects the recurrent layer linkage to be implemented. This linkage could be static, i.e. defined once during training, or dynamic, i.e. changing during operation of the PC-LRPNN, depending on the input sequences. Furthermore, it could be expected that many of the recurrent layer neurons will participate in multiple class-specific neighbourhoods, which are then combined to assemble the recurrent layer linkage, but there could be neurons that do not participate in any swarms and are left detached from their neighbours. Usually, the linkage selection is performed in a data-dependent manner but it could be also based on the indexes of the individual neurons, if there is such necessity they to be pre-specified or bounded.

In fact, the linkage selection consists in identifying a sufficient subset of connections which typically is much smaller than the size of the full linkage. An assortment of strategies can be applied for identifying the optimal subsets of interacting neurons, i.e. the scope of swarm, and the neighbourhood for each target class. For instance, examples could be strategies based on identifying the Top-C competitor classes for a given input sequence, and implementing the linkage only for the recurrent neurons corresponding to these classes. The linkage to the less-promising competitors, which are not members of the Top-C club, is not implemented. An alternative strategy could be to perform pruning of the connections, starting from the fully connected LRPNN and iteratively identifying and discarding links which are not contributing for maximizing the overall performance. Yet, another strategy could be to start from the simplest reasonable topology and continue adding connections until the performance of the PC-LRPNN increases, or predefined limits are reached. Other strategies might involve optimization of the linkage of each particular recurrent neuron or the amount of memory it possesses, and then organize teams of super-neurons, etc. Obviously, the most successful strategies should exploit any *a priori* knowledge about the problem at hand and be able to interpret properly the information available in the training dataset.

At this point, we need to remember that in the PC-LRPNNs we deal with classification scheme of the type winner-takes-all, and that the scores acting on the input of the recurrent layer are in fact the probabilities computed by the summation units in the previous layer. These probabilities compete for distinguishing the winning class, and in non-trivial multi-class problems there exist more than one probability bigger than zero. For this type of classification scheme, we can consider a straightforward but efficient and effective strategy that builds the recurrent layer linkage by identifying a neighbourhood for a given recurrent neuron in terms of its closest competitors for the prise. In such a strategy, we follow a two stage procedure:

1. Firstly, we identify the Top-C competitors for each target class, by feeding the original non-compressed training data for that class at the input of the already trained pattern layer. At the output of the class-specific summation units (residing in the summation layer), the outcome will be a set of $M_i \text{x} K$ probabilities, with M_i indicating the number of feature vectors in the training dataset for class κ_i and K the total number of target classes. Having computed the matrix $M_i \text{x} K$ for a specific class κ_i, we can identify the Top-C competitors by computing the average score per class, and sorting these values.
2. Subsequently, we implement symmetric connections only among these Top-C recurrent neurons. Here, symmetric stands for the case where each neuron that receives information form another neuron also supplies back to this neuron the equivalent information about its own class. Thus, the relationship between the two neurons is symmetric in terms of linkage. However, in the general case symmetry might not be reasonable or desirable and should not be imposed unless the properties of the underlying training data indicate such necessity, or there exists some a priori knowledge about the problem at hand.

Eventually, the recurrent layer linkage is formed as union of all class-specific neighbourhoods. This can be expressed as follows: Let L_0 be the $K \text{x} K$ matrix which represents the connections originating form the output of the summation layer to the inputs of the recurrent layer neurons, and $l_0(i,j)=1$ indicates that the specific connection from the summation unit corresponding to class κ_i is connected to the recurrent neuron for class κ_j. Alternatively, the value $l_0(i,j)=0$ would indicate that the specific connection was not implemented. The individual elements of the L_0 matrix, i.e. $l_0(i,j)$, can be referred to as the mask which determines if the specific coefficients $b_{i,j}$ (refer to (12)) will be present or not. Obviously, it is mandatory for the diagonal elements of L_0 to have non-zero values, i.e. $l_0(i,i)=1$, for any $i=1,2,...,K,$ so a connection between the summation and recurrent layer in class κ_j is always guaranteed. As explained earlier, the rest of the linkage $l_0(i,j)\big|_{i\neq j}$ can be identified in a data-dependent manner (for instance, by following the Top-C strategy), or by utilizing *a priori* knowledge.

By analogy, let the $K \text{x} K$ matrixes L_n, with $n=1,2,...,N,$ stand for the links that originate from the past outputs of the recurrent layer neurons to the inputs of neurons in the same layer. Here n is the index of delay, and the elements of L_n serve as a mask, which determines if the coefficients $a_{i,j,n}$ (refer to (12)) will exist, or not. Again, let the elements of L_1, $l_1(i,j)=1$, indicate that there exists a connection between the past output at time $t-1$ of the recurrent neuron for class κ_i and the input of the recurrent neuron for class κ_j, and $l_1(i,j)=0$ indicate for lack of connection. The same logic applies for the other matrixes L_n, but in contrast with L_0 there are no restrictions about the values of their elements, $l_n(i,j)$, i.e. there could be a case where all $l_n(i,j)=0$. In such a case, all recurrent feedbacks from past states as well as the connections between the recurrent neurons are dismissed, which is equivalent to recurrence depth $N=0$. When this is combined with strategy Top-1 (and all coefficients $b_{i,i}=1$), the PC-LRPNN becomes functionally identical to the original PNN. On

the other hand, when all $l_n(i,j)=1$ and $l_0(i,j)=1$ the structure of the PC-LRPNN coincides with the one of the fully connected LRPNN.
Eventually, the overall linkage of the recurrent layer is the composite matrix

$$L = \left[L_0 \vdots L_1 \vdots \dots L_n \dots \vdots L_N\right],\ n = 1,2,\dots,N, \tag{22}$$

with dimensionality $(N+1)\text{x}K\text{x}K$, where K is the total number of target classes, and N is the recurrence depth.
In general, the linkage defined by the matrixes L_n and L_0 can be identified using different strategies, or yet the same Top-C strategy. Furthermore, in the simplest scenario, the matrixes L_n could be duplicates of L_0, so L to have a repeating structure, however, this is not a requisite by any means. Once the proper linkage L is identified the weight of each connection needs to be estimated.
STEP 4: Finally, the forth training step consists in computation of the recurrent layer weights, using the uncompressed training data exploited at step three. In previous work (Ganchev et al, (2003, 2004)), we studied training strategies that aim at adjusting the weights in the recurrent layer in a manner that maximizes the classification accuracy on the training data set. Here we rely on another more successful strategy that was developed recently (Ganchev, in-press-2008). In brief, this new training strategy does not rely on a quantitative measure accounting for the classification performance on the training dataset, but merely aims at maximizing the probability for the target class and simultaneously minimizing the probabilities computed for the non-target classes over the training dataset. This leads to a simplification of the error function and reduction in the number of steps necessary for evaluating the goodness of the recurrent layer weights at each iteration.
Specifically, the new error function that is subject to minimization here involves the complementary to one value of the probability $P(\mathbf{X}_{k_i} \mid k_i)$, and the compound probability for $\mathbf{X}_{k_i}$ belonging to any other class:

$$\text{E}(\mathbf{w}) = \sum_{i=1}^{K} m_i \left(1 - P(\mathbf{X}_{k_i} \mid k_i)\right) P(k_i) + \frac{1}{K-1} \sum_{i=1}^{K} \sum_{\substack{j=1 \\ j \neq i}}^{K} m_j P(\mathbf{X}_{k_i} \mid k_j) P(k_j)\,. \tag{23}$$

Here $\mathbf{X}_{k_i}$ are the training data for class κ_i, and $P(k_i)$ are the *a priori* probability of class κ_i. Finally, the constants m_i and m_j determine the relative importance of (or alternatively the significance of misclassification of an input belonging to) the corresponding class κ_i or $\kappa_{j \neq i}$, respectively.
The first term in equation (23) estimates the distance between the probability $P(\mathbf{X}_{k_i} \mid k_i)$ and one, i.e. the error with respect to the probability computed for a perfect match to the model. This term causes the output for class κ_i of the trained recurrent layer to strive towards value one for input vectors that resemble the training dataset for that class. The second term in equation (23) is the cumulative error of $\mathbf{X}_{k_i}$ being acknowledged as belonging to any of the competitive classes $\kappa_{j \neq i}$. This second term contributes towards restraining the output values produced by the competitive classes for input data that belong to class κ_i.

The minimization of total error $E(\mathbf{w})$ is performed by employing a PSO algorithm Type 1 (Clerc & Kennedy, 2002), which was found more successful and/or much faster than other PSO implementations, such as the basic PSO (Eberhart & Shi, 2000), the local PSO as in (Liang et al., 2006), and the UPSO (Parsopoulos & Vrahatis, 2005).

6. Numerical evaluation

The experimentations reported in the present section aim at illustrating the operation of the PC-LRPNNs, but also serve as a scene for discussing the advantages and disadvantages of the PC-LRPNN, when compared to the original PNN and the fully connected LRPNN. Specifically, for the purpose of experimentations, we selected two interesting problems of different difficulty. Both of these problems are important for the development of human-friendly spoken dialogue applications, and by that reason they currently enjoy significant attention by the speech processing community. The first one is the text-independent speaker identification task, which is of moderate difficulty, and the second one is the speaker-independent emotion recognition task, which is well-known as an extremely challenging problem. In the following paragraphs we offer a brief outline of these tasks:

Task 1: Text-independent speaker identification

Speaker identification is multiple-class decision problem where the identity of a given speaker is judged based on a comparison of a sample of her\his voice against multiple pre-defined models. The outcome of this process is either a decision about the identity of the speaker or a notice that the present input cannot be categorized as any of the known speakers. In the closed set speaker identification that we consider here, the input speech utterances always belong to someone of the known speakers. Here, text-independence referrers to the specific aspect that no explicit modelling of the linguistic contents of the input utterance is performed. Thus, the outcome of the identification process is not dependent on the exact linguistic contents of the phrase, but only on the degree of proximity between the input speech signal and the predefined speaker models.

Task 2: Speaker-independent emotion recognition

The emotion classification task is a multiple-class decision problem, where the emotional state of a given speaker is judged based on comparison of an input (typically a speech utterance) against multiple pre-defined models for the emotional states of interest. Here, the notion for speaker-independency refers to the fact that the models for the emotional states of interest are general for a large population of people, and were built utilizing the speech of people who do not present in the test datasets. Emotion recognition from speech is a very challenging task mainly due to the inherent speaker-dependency of emotion expression but also due to the well-known multi-functionality of speech (Batliner & Huber, 2007).

6.1 Experimental protocol

Common training and testing protocols were followed in all experiments. All classifiers considered in the present evaluation (GMM, PNN, LRPNN, PC-LRPNN) were trained with common task-specific train datasets, and trials were performed with common task-specific test datasets.

For the purpose of the speaker identification task, ten female speakers, extracted from the PolyCost v1.0 telephone-speech speaker recognition corpus (Hennebert et al., 2000) were modelled as authorized users. The training data, comprised of ten utterances, containing both numbers and sentences, obtained from the first session of each speaker. In average, about 17 seconds of voiced speech per speaker were available for training each user model. The test dataset consisted of 450 target trials, including 45 utterances per speaker. Each test trial involved approximately 3 seconds of speech. All speech recordings are of telephone bandwidth, sampled at 8 kHz, and A-law compressed.

For the purpose of the experimentations with the emotion classification task, we utilized the recordings of all eight speakers available in the Emotional Prosody Speech and Transcripts database (LDC, 2002). All recordings were split in utterances, with respect to the provided annotations, and then were down-sampled to 8 kHz and band-limited to telephone quality bandwidth.

In the specific experimental setup considered here, we carry out recognition of three emotional states: neutral, anger, and panic. This combination is of particular interests for practical applications, but also has proved as a very challenging set, since the members of the pairs: hot anger – panic, and cold anger – neutral, share a number of common prosodic characteristics.

The training dataset consist of the available recordings for the seven speakers and the recordings of the remaining speaker were used as the test dataset. Since one of the speakers had only neutral recordings, the training data for the anger and panic models were built from the recordings of six speakers. The amount of available data for training the speaker-independent models for the three emotional categories of interest was much different: approximately 1650, 380 and 180 seconds of speech for neutral, anger and panic, respectively. For the purpose of fair training of the emotion models, we performed k-means clustering as a pre-processing of the training dataset. The resultant codebooks, one per speaker, one per emotion category, were of size 256 feature vectors. Subsequently, these codebooks were used to train the neural network-based classifiers.

On the other hand, the GMM-based classifier was trained directly from the uncompressed dataset, for achieving a higher precision of the emotion models. The diagonal covariance GMM emotion models were trained via a standard version of the Expectation Maximization algorithm (McLachlan & Krishnan, 1997) with a maximum of 200 iterations. Training termination criterion was applied, and training process was interrupted if there was no error reduction among subsequent iterations.

The amount of target trials per category was 115, 29 and 18 utterances for the neutral, anger and panic, respectively. Each test trial consisted of approximately 3 seconds of speech.

In both tasks, only the voiced parts of the speech signal was parameterized to Mel-frequency cepstral coefficients (MFCC) with a rate of 100 feature vectors per second. We utilized the MFCC implementation of Slaney (Slaney, 1998), but adapted for sampling frequency of 8 kHz. This resulted in a filter-bank of thirty-two filters, which cover the frequency range [133, 3954] Hz, from which we computed 29 cepstral coefficients. In all experiments, we excluded the first cepstral coefficient (i.e. the one with index zero) from the feature vector, to avoid dependence on the recording setup (distance to the microphone, communication channel and handset mismatch, etc). Finally, in all experiments we considered a common feature vector consisting of the MFCC parameters {*MFCC(1)* ,…,*MFCC(28)*}. All parameters of the feature vector were normalized to fit in a common dynamic range.

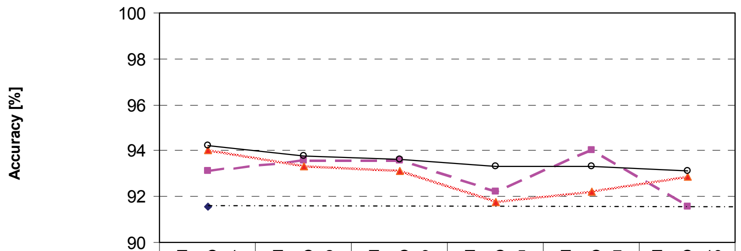
6.2 Experimental results

In this section, we study the performance of the PC-LRPNN for different connectivity range of the neighbourhood, Top-C, and different recurrence depth, N, of the recurrent layer. Comparisons with the PNN, GMM and the fully connected LRPNN are provided as follows:

The PC-LRPNN vs. the PNN and LRPNN, in the speaker identification task

Since in this task we consider identification of 10 different voices, i.e. we have 10 classes, we can note that in the case Top-C=10, the PC-LRPNN is equivalent to the fully connected LRPNN. On the other hand, in the case of $N = 0$, Top-C=1, the PC-LRPNN has the same number of weights in the recurrent layer as the number of connections between the summation and competitive layers in the PNN. However, there is no equivalence between these two structures, mainly because the weights of the connections between the summation and recurrent layers in the PC-LRPNN are adjusted during training, while in the PNN they are all equal. This gives to the PC-LRPNN the capability to model better the training data.

In Fig. 5, we present the performance of the PC-LRPNN in the speaker identification task, and in Table 1, we show the number of recurrent layer weights for different values of the recurrence depth, N, and different values of the neighbourhood, Top-C. As the figure presents, for the PNN we obtained recognition accuracy of 91.6%, which we consider as the



	TopC=1	TopC=2	TopC=3	TopC=5	TopC=7	TopC=10
■ pcLRPNN(N=0)	93.1	93.6	93.6	92.2	94.0	91.6
▲ pcLRPNN(N=1)	94.0	93.3	93.1	91.8	92.2	92.9
○ pcLRPNN(N=2)	94.2	93.8	93.6	93.3	93.3	93.1
◆ PNN	91.6					

Fig. 5. Performance of the PC-LRPNN classifier on the speaker identification task, for different values of the recurrence depth, N, and different size of the recurrent layer neighbourhoods, Top-C.

Number of recurrent layer weights	*N=0*	*N=1*	*N=2*
Top-C=1	10	20	30
Top-C=2	30	60	90
Top-C=3	46	92	138
Top-C=5	68	136	204
Top-C=7	90	180	270
Top-C=10	100	200	300

Table 1. The number of recurrent layer weights to be trained, for different recurrence depth, N, and different size of the recurrent layer neighbourhood, Top-C

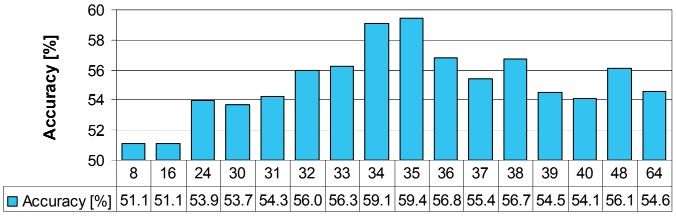
baseline. It is interesting to note that the performance of the PC-LRPNN for $N = 0$, Top-C=1, i.e. when there are no recurrent feedbacks in the recurrent layer and the connection between the summation layer and recurrent layer neurons is implemented only for the top scoring candidates, is higher than the baseline, PNN, although the number of weights is equal. As explained above, this advantage of the PC-LRPNN comes from the fact that the values of these weights are trained in a data-dependent manner, while in the original PNN, these weight are equal. Furthermore, for recurrence depth $N = 0$, the PC-LRPNN with neighbourhood Top-C=7 demonstrated the highest recognition accuracy (94.0%), which is higher than the recognition accuracy for the case of Top-C=10, i.e. the equivalent to the fully connected LRPNN. The last can be explained with the smaller number of weights to be adjusted for the case of Top-C=7, and the limited amount of training data.

Next, as Fig. 5 presents, the highest recognition accuracy among all (94.2%) was achieved for the PC-LRPNN with $N = 2$, and Top-C=1. The top performance here illustrates both the importance of the recurrence depth (i.e. the memory about past states) and the capability of the PC-LRPNN to implement partial linkage in the recurrent layer. As the figure presents, the second best performance is shared between the PC-LRPNN with $N = 1$, and Top-C=1, and the already discussed $N = 0$, and Top-C=7. It is interesting to note that the recurrent PC-LRPNN ($N = 1$) achieves this performance with only 20 weights, while the non-recurrent PC-LRPNN ($N = 0$) needs 90 (please refer to Table 1.)

Speaking generally, we can conclude that the presented example on the speaker identification task illustrates undoubtedly that the PC-LRPNNs provide higher recognition accuracy than the baseline PNN. This advantage is mainly due to the exploitation of information from the competitive classes, which the recurrent layer neurons utilize for proper selection of the class belonging of a given input sequence. Furthermore, we observed that the PC-LRPNNs show performance even better than the one of the fully connected LRPNN. This superiority is due mainly to the additional degree of freedom that the PC-LRPNNs possess, i.e. the better flexibility to adjust the implementation of the recurrent layer linkage to the available training data. Finally, it is worth mentioning that due to the reduced number of weights, the PC-LRPNN are trained and operate much faster than the fully connected LRPNN.

The PC-LRPNN vs. the PNN, LRPNN and GMM, in the emotion recognition task

In the emotion recognition task, we firstly experimented with a state-of-the-art GMM-based classifier to identify the maximum performance that can be obtained (for a context-blind classifier) in our experimental setup. In Fig. 6, we present the identification accuracy obtained for different number of components in a Gaussian mixture. As the figure presents,

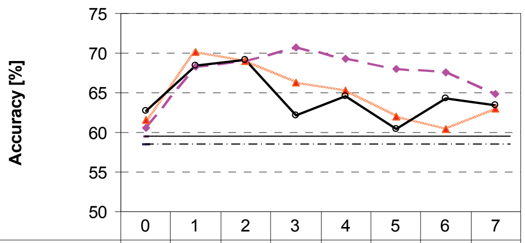


	8	16	24	30	31	32	33	34	35	36	37	38	39	40	48	64
🟦 Accuracy [%]	51.1	51.1	53.9	53.7	54.3	56.0	56.3	59.1	59.4	56.8	55.4	56.7	54.5	54.1	56.1	54.6

Fig. 6. Performance of the GMM classifier for different number of components on the speaker-independent emotion recognition task

the highest recognition accuracy (59.4%) was observed for the case of Gaussian mixture with 35 components. This performance will be considered as the baseline.

In Fig. 7, we present the recognition accuracy obtained for the PNN, PC-LRPNN and the LRPNN for different values of the recurrence depth, N, and different size of the recurrent layer neighbourhoods, Top-C. Table 2 presents the number of weights in the recurrent layer that need to be trained for the PC-LRPNN and LRPNN. Since in the present experimental setup we have three emotional categories, the PC-LRPNN with Top-C=3 is equivalent to the fully connected LRPNN.



	0	1	2	3	4	5	6	7
— GMM	59.4							
-·-·- PNN	58.5							
◆ pcLRPNN(Top-1)	60.5	68.2	69.0	70.7	69.3	68.0	67.6	64.9
▲ pcLRPNN(Top-2)	61.6	70.1	69.0	66.3	65.3	62.0	60.5	63.0
○ pcLRPNN(Top-3)	62.7	68.4	69.2	62.1	64.6	60.4	64.2	63.5

Fig. 7. Performance of the PC-LRPNN classifier on the emotion recognition task, for different values of the recurrence depth, N, and different size of the recurrent layer neighbourhoods, Top-C.

As the figure presents, the recognition accuracy obtained for the PNN is inferior to the one for the GMM classifier. The difference in performance of approximately 1% can be explained by the fact that here we employ the original homoscedastic PNN, which utilizes uniform smoothing factor σ_i for all classes, while the diagonal GMM employed here adjusts the variance for each class and thus is able to adapt better to the underlying distribution of the training data.

Number of recurrent layer weights	*N=0*	*N=1*	*N=2*	*N=3*	*N=4*	*N=5*	*N=6*	*N=7*
Top-C=1	3	6	9	12	15	18	21	24
Top-C=2	7	14	21	28	35	42	49	56
Top-C=3	9	18	27	36	45	54	63	72

Table 2. The number of recurrent layer weights to be trained, for different recurrence depth, N, and different size of the recurrent layer neighbourhood, Top-C

As Fig. 7 presents, the recognition accuracy observed for the PC-LRPNN and the fully connected LRPNN is superior to the one observed for the GMM and the PNN. Inspecting the recognition accuracy presented on the figure and the number of weights in the recurrent layer, presented in Table 2, we can notice that there are some relations among the performance results for similar number of coefficients. Furthermore, the general trend of the plots for different neighbourhood size, Top-C, seems to agree with respect to the increase of the recurrence depth, N. On the present experimental setup, the PC-LRPNN outperforms entirely the fully connected LRPNN, due to its better capacity to adapt to the training data. Exception here is the case for recurrence depth $N = 6$, where the LRPNN outperforms significantly the partially connected counterpart.

Finally, the significant advantage of the PC-LRPNN and LRPNN over the PNN and GMM can be summarized as follows:

1. LRPNNs and PC-LRPNNs process the information coming from the competitive classes (for Top-C > 1) and the target class;
2. the recurrent structures are capable to capture temporal dependences among subsequent feature vectors, and thus, are capable to exploit the context in which a given input appears;
3. the recurrent layer is trained in a constructive manner to maximize the probability generated for the target class and to minimize to probabilities generated by the competitive classes, which favours resolving ambiguous situations.

The smoothing factor σ_i, the PC-LRPNN vs. the PNN

Utilizing the experimental setup of the emotion recognition task, and the best performing locally recurrent neural network, i.e. PC-LRPNN (Top-C=1, N =3), we would like to discuss an interesting phenomenon concerning the optimal value of the smoothing factor, σ_i.

Extensive experimentations with the PNN, PC-LRPNNs and fully connected LRPNNs, demonstrated that the estimation of the smoothing factor σ_i on the training dataset does not lead to optimal performance on the test dataset. This was especially topical in the emotion recognition task. To illustrate this phenomenon, in Fig. 8, we plot the performance of the PNN and the best performing PC-LRPNN for different values of the smoothing factor, σ_i. For comprehensiveness of exposition we computed the recognition accuracy obtained on the training dataset, presented in the figure with dashed line, and on the test datasets, presented with solid line.

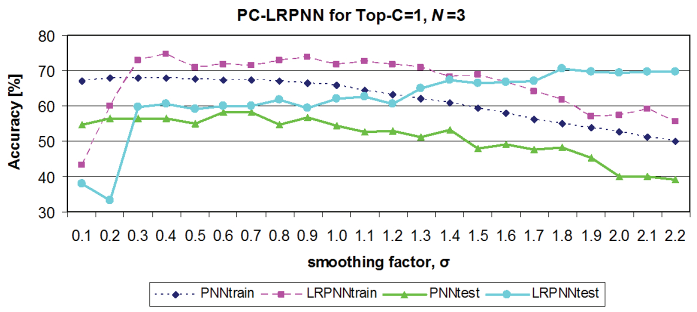


Fig. 8. Performance of the PNN and PC-LRPNN (Top-C=1, N =3), for different values of the smoothing factor σ_i

As the figure presents, for both the PNN and PC-LRPNN there was a significant gap between the recognition accuracy obtained on the training dataset and on the test data. Looking at the plots for the PNN, we can see that the trend of this difference in performance is a relatively smooth monotonically decreasing function. However, for the PC-LRPNN, the initial difference, for small values of σ_i, tends to decrease when σ_i increases. Furthermore, the best performance for the PNN was obtained for σ_i =0.6, and the best performance for the various PC-LRPNN and LRPNN in the different experiment was for σ_i in the range [1.2, 2.0], even though the recognition performance for the training dataset was significantly lower, when compared to the optimal σ_i computed through (Cain, 1990), or any other method. Although the degree of learning for the training dataset varied greatly in the experiments with different PC-LRPNNs, and mostly ranged between 75% and 100%, the best performance on the test dataset was observed always for significantly higher values of σ_i, when compared to the best one for the training dataset. The last indicates that in challenging problems, for which it is known that there is significant mismatch between the training and operational conditions, the computation of the value of σ_i should be performed on another independent dataset, referred to as development or validation data. The development data are independent from the training and test datasets and serve for fine-tuning of the overall performance.

7. Conclusion and future research directions

Although the research on locally recurrent neural networks has a long record of history, the potential of development has not been exhausted. Moreover, in the last few years, there is a resumption of interest to the field, and recently some new paradigms appeared. These new architectures are in anticipation of further in depth studies, and further improvements and elaboration.

Speaking specifically for the family of LRPNNs, there is a compelling need for further studies that will investigate comprehensively how the recurrent layer linkage can be optimized for specific problem on specific dataset. Perhaps, new strategies for automatic selection of neighbourhood size and the specific neighbours of each recurrent neuron that arise directly from the training data will appear. It will be particularly interesting to study new algorithms for developing dynamically varying neighbourhoods, which depend on the input during operation of the neural network, and which go beyond the predefined during training look-up tables.

Finally, despite the progress made during the past decades, we deem that the locally recurrent neural networks still await for their golden time, when they will have significantly better biological plausibility. The human brain is still a source of inspiration, and we are looking forward to see how the development of the neuroscience will contribute further for the progress in the field of recurrent neural networks.

8. References

Aussem, A.; Murtagh, F. & Sarazin, M. (1995). Dynamical recurrent neural networks — towards environmental time series prediction, *International Journal of Neural Systems*, No.6, June 1995, pp.145–170.

Back, A.D. & Tsoi, A.C. (1991). FIR and IIR Synapses, a new neural network architecture for time series modelling, *Neural Computation*, Vol.3, 1991, pp.375–385.

Back, A.D. & Tsoi, A.C. (1992). Nonlinear system identification using multilayer perceptrons with locally recurrent synaptic structure, *Proceedings of 1992 IEEE-SP Workshop on Neural Networks for Signal Processing II*, 1992, pp.444–453.

Barbounis, T.G. & Theocharis, J.B. (2007a). A locally recurrent fuzzy neural network with application to the wind speed prediction using spatial correlation, *Neurocomputing*, Vol.70, No. 7–9, 2007, pp.1525–1542.

Barbounis, T.G. & Theocharis, J.B. (2007b). Locally recurrent neural networks for wind speed prediction using spatial correlation, *Information Sciences*, Vol. 177, No. 24, December 2007, pp.5775–5797.

Batliner, A. & Huber, R. (2007). Speaker characteristics and emotion classification, In: *Speaker Classification I, LNAI 4343*, C. Műller (Ed.), Springer, 2007, pp.138–151.

Cain, B.J. (1990). Improved probabilistic neural network and its performance relative to the other models, *Proceedings of the SPIE, Applications of Artificial Neural Networks*, Vol.1294, 1990, pp.354–365.

Cannas, B.; Celli, G.; Marchesi, M. & Pilo F. (1998). Neural networks for power system condition monitoring and protection, *Neurocomputing*, Vol.23, 1998, pp.111–123.

Clerc, M. & Kennedy, J. (2002). The particle swarm - explosion, stability, and convergence in a multidimensional complex space, *IEEE Transactions on Evolutionary Computation*, 2002, Vol.6, pp.58–73.

Eberhart, R.C. & Shi, Y. (2000). Comparing inertia weights and constriction factors in particle swarm optimization, *Proceedings of the Congress on Evolutionary Computing*, 2000, pp.84–88.

Ganchev, T.; Tasoulis, D.K.; Vrahatis, M.N. & Fakotakis, N. (2003). Locally recurrent probabilistic neural network for text-independent speaker verification, *Proceedings of 8th European Conference on Speech Communication and Technology, EUROSPEECH 2003*, September 1-4, 2003, Vol. 3, pp.1673–1676.

Ganchev, T.; Tasoulis, D.K.; Vrahatis, M.N. & Fakotakis, N. (2004). Locally recurrent probabilistic neural networks with application to speaker verification, *GESTS International Transaction on Speech Science and Engineering*, December 2004, Vol.1, No.2, pp.1–13.

Ganchev, T.; Tasoulis, D.K.; Vrahatis, M.N. & Fakotakis, N. (2007). Generalized locally recurrent probabilistic neural networks with application to text-independent speaker verification, *Neurocomputing*, Vol.70, No.7–9, 2007, pp. 1424–1438.

Ganchev, T. (in-press-2008). Enhanced training for the locally recurrent probabilistic neural networks, to apear in *International Journal on Artificial Intelligence Tools*, 2008.

Georgiou, V.L.; Pavlidis, N.G.; Parsopoulos, K.E.; Alevizos, Ph.D. & Vrahatis, M.N. (2006). New self-adaptive probabilistic neural networks in bioinformatic and medical tasks, *International Journal of Artificial Intelligence Tools*, 2006, Vol. 15, No. 3, pp. 371–396.

Georgiou, V.L.; Alevizos, Ph.D. & Vrahatis, M.N. (2008). Novel approaches to probabilistic neural networks through bagging and evolutionary estimating of prior probabilities, *Neural Processing Letters*, Vol. 27, No. 2, pp. 153–162.

Janson, S. & Middendorf, M. (2005). A hierarchical particle swarm optimizer and its adaptive variant, *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, Vol.35, No.6, December 2005, pp.1272–1282.

Hennebert, J.; Melin, H.; Petrovska, D. & Genoud, D. (2000). POLYCOST: A telephone-speech database for speaker recognition, *Speech Communication*, Vol.31, No.2-3, pp. 265–270.

Kasper, K.; Reininger, H.; Wolf, D. & Wust, H. (1995). A speech recognizer based on locally recurrent neural networks, *Proceedings of the International Conference on Artificial Neural Networks*, 1995, Vol. 2, pp.15–20.

Kasper, K.; Reininger, H. & Wust, H. (1996). Strategies for reducing the complexity of a RNN-based speech recognizer, *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing, ICASSP 1996*, Vol.6, pp.3354–3357.

Kennedy, J. (1999). Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance, *Proceedings of the 1999 Congress on Evolutionary Computation, CEC'99*, Vol.3, pp.1931–1938.

Kennedy, J. & Eberhart, R.C. (1995). Particle swarm optimization, *Proceedings of the IEEE International Conference on Neural Networks, ICNN 1995*, Vol.4, pp.1942–1948.

LDC (2002). University of Pennsylvania, Linguistic Data Consortium, Emotional prosody speech and transcripts (LDC2002S28), Available at: www.ldc.uppen.edu/Catalog/CatalogEntry.jsp?cataloId=LDC2002S28

Liang, J.J.; Qin, A.K.; Suganthan, P.N. & Baskir, S. (2006). Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, *IEEE Transactions on Evolutionary Computation*, 2006, Vol.10, No.3, pp.281–295.

Lin, T.; Horne, B.G. & Giles, C.L. (1998). How embedded memory in recurrent neural network architectures helps learning long-term temporal dependencies, *Neural Networks*, 1998, Vol.11, pp.861–868.

Mak, M.W. & Kung, S.Y. (2000). Estimation of elliptical basis function parameters by the EM algorithm with application to speaker verification, *IEEE Transactions on Neural Networks*, Vol.11, No.4, 2000, pp. 961–969.

McLachlan, G.J. & Krishnan, T. (1997). *The EM algorithm and extensions*. Wiley Series in Probability and Statistics. New York: Wiley, 1997.

Meisel, W. (1972). *Computer-oriented approaches to pattern recognition*. Academic Press, New York, 1972.

Parsopoulos, K.E. & Vrahatis, M.N. (2002). Recent approaches to global optimization problems through particle swarm optimization, *Natural Computing*, Vol.1, No.2-3, pp.235-306.

Parsopoulos, K.E. & Vrahatis, M.N. (2004). On the computation of all global minimizers through particle swarm optimization, *IEEE Transactions on Evolutionary Computation*, Vol.8, No. 3, pp.211–224.

Parsopoulos, K.E. & Vrahatis, M.N. (2005). Unified particle swarm optimization for tackling operations research problems, *Proceedings of the IEEE Swarm Intelligence Symposium, SIS 2005*, June 2005, pp.53–59.

Parsopoulos, K.E. & Vrahatis, M.N. (2007). Parameter selection and adaptation in unified particle swarm optimization, *Mathematical and Computer Modelling*, Vol.46, No.1-2, pp.198–213.

Slaney, M. (1998). Auditory toolbox. Version 2. Technical Report #1998-010, Interval Research Corporation.

Specht, D.F. (1988). Probabilistic neural networks for classification, mapping, or associative memory, *Proceedings of the IEEE Conference on Neural Networks*, 1988, Vol.1, pp.525–532.

Specht, D.F. (1990). Probabilistic neural networks, *Neural Networks*, 1990, Vol.3, No.1, pp.109–118.

Specht, D.F. (1992). Enhancements to probabilistic neural networks, *Proceedings of the IEEE International Joint Conference on Neural Networks*, *IJCNN 1992*, Vol.1, pp. 761–768.

Specht, D.F. & Romsdahl, H. (1994). Experience with adaptive PNN and adaptive GRNN, *Proceedings of the IEEE International Conference on Neural Networks*, *ICNN 1994*, Vol.2, pp.1203–1208.

Yang, Z.R. & Chen, S. (1998). Robust maximum likelihood training of heteroscedastic probabilistic neural networks, *Neural Networks*, 1998, Vol.11, No.4, pp.739–748.