DETERMINISTIC NONMONOTONE TRAINING OF RECURRENT NEURAL NETWORKS AND APPLICATIONS USING SYMBOLIC SEQUENCE DATA

A THESIS SUMBITTED FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FOR THE UNIVERSITY OF LONDON

BY

CHUN-CHENG PENG

SUPERVISOR: DR. G.D. MAGOULAS

SCHOOL OF COMPUTER SCIENCE AND INFORMATION SYSTEMS

February 2010

Declaration Statement

I, Chun-Cheng PENG, hereby confirm and declare that the thesis entitled by "DETERMINISTIC NONMONOTONE TRAINING OF RECURRENT NEURAL NETWORKS AND APPLICATIONS USING SYMBOLIC SEQUENCE DATA" and submitted for the Degree of Doctor of Philosophy for the University of London, is my own and original work.

Signed: (PENG, Chun-Cheng)

Date: 1. FEB. 2011

Abstract

Recurrent Neural Networks (RNNs) are well known for their power to model temporal dependencies and process sequences for classification, recognition and transduction. Gradient-based methods are a popular choice for training RNNs despite the inherent limitations of the gradient descent method. Typically, these methods require monotonicity of the error values, i.e. they reduce the error function at each iteration. Nevertheless, enforcing monotonicity does not guarantee that a method will efficiently explore the search space in the sense that it may be trapped in a local minimum point early on and never jump out to reach a desired solution under ill conditions.

In this thesis, we propose novel deterministic RNN training algorithms that adopt a nonmonotone approach. This allows learning behaviour to deteriorate in some iterations; nevertheless the overall learning performance is improved over time. The proposed nonmonotone RNN training methods, which take their theoretical basis from the theory of deterministic nonlinear optimisation, aim at better exploring the search space and enhancing the convergence behaviour of gradient-based methods. They generate nonmonotone behaviour by incorporating conditions that employ forcing functions, which are used to measure the sufficiency of error reduction, and an adaptive window, whose size is informed by estimating the morphology of the error surface locally.

The thesis develops nonmonotone first-order and second-order methods and discusses their convergence properties. The proposed algorithms are applied to training RNNs of various sizes and architectures, namely Feed-Forward Time-Delay networks, Elman Networks and Nonlinear Autoregressive Networks with Exogenous Inputs networks, in symbolic sequence processing problems. Numerical results show that the proposed nonmonotone learning algorithms train more effectively RNNs for sequence processing than other gradient-based methods in the literature.

Keywords: Adaptive algorithm, BFGS update, conjugate gradient, deterministic training, learning horizon, Levenberg-Marquardt approach, nonmonotone learning, resilient propagation, recurrent neural networks, symbolic sequence processing

Acknowledgements

First of all, I appreciate my supervisor, Prof. George Magoulas, for all his guidance, advices and helps during the past few years. Without Prof. Magoulas continuing support and deep knowledge of the matter, I wouldn't have been able to achieve this result.

I also want to express my thanks to the Examiners of my oral examination, Prof. D. Palmer-Brown and Dr. N. Nikolaev, for their helpful suggestions and corrections.

Lastly, supports from the Peng's Family and my wife Wen-Yu are the key points to implement my overseas studying.

Contents

List of Abbreviations

A2NM-CG	Advanced Adaptive Non-Monotone Conjugate Gradient method
ANM-CG	Adaptive Non-Monotone Conjugate Gradient method
ANM-JRprop	**A**daptive Non-Monotone Jacobi Resilient Propagation method
ANM-LM	Adaptive Non-Monotone Levenberg-Marquardt methods
ANM-LMAM	Adaptive Non-Monotone LMAM method
ANM-OLMAM	Adaptive Non-Monotone OLMAM method
ASCNM-BFGS	Adaptive Self-sCaling Non-Monotone BFGS method
BFGS	Broyden-Fletcher-Goldfarb-Shanno method
CE	classification error
FFTD	Feed-Forward Time-Delayed Network
LMAM	Levenberg-Marquardt method with adaptive momentum
LRN	Layered Recurrent Network
MSE	mean-squared error
NARX	Nonlinear Auto-Regressive network with eXogenous Inputs
OLMAM	Optimised LMAM
P5	the Parity-5 problem
P10	the Parity-10 problem
RA	the Reading Aloud problem
RNN	Recurrent Neural Networks
SC	the Sequence Classification problem
SL	the Sequence Learning problem

List of Tables

List of Figures

Chapter 1

Introduction

In this Chapter the aim of this research is firstly introduced, followed by definition and discussion of the main target application, i.e. temporal sequence processing. The organisation of this thesis is then presented, and in the last section its contribution is highlighted.

1.1 Introduction

Recurrent networks constitute an elegant way of increasing the capacity of feedforward networks to deal with complex data in the form of sequences of patterns. Recurrent neural networks are well known for their power to model temporal dependencies and process sequences for classification, recognition, and transduction. Modern RNNs architectures are capable of learning to solve many previously unlearnable tasks, even in partially observable environments. Recent directions in RNN research focus on investigating and proposing new ways for better modelling of non-stationarity in sequences, such as sequences produced when modelling speech or handwritten characters, with no temporal independence assumptions.

With regards to architectures, hybrid models based on combinations of Hidden Markov Models and RNNs as well as modular structures are considered promising approaches to solve sequence processing problems that occur in natural language and speech processing. In addition, a number of applications of the so-called Long Short-Term Memory RNN [90] have provided some encouraging results, demonstrating that these recurrent architectures can overcome several of the fundamental problems of traditional RNNs, and efficiently learn to solve many previously unlearnable tasks.

As far as RNN training is concerned, which is the main focus of this PhD, gradient descent approaches, which enforce the monotone decrease of the learning error, remain popular despite the demonstrated potential of some new approaches that are based on evolutionary algorithms [174] or nonmonotone learning strategies [145][146].

In this thesis, we identify some challenges involved in training RNNs and propose algorithmic approaches based on gradient information for nonmonotone training of RNNs for sequence processing. The term *sequence processing* referred here involves several tasks such as clustering, classification, prediction, and transduction of sequential data. In the general case data can be *symbolic*, *non-symbolic* or mixed. Examples of symbolic data patterns occur in modelling natural (human) language, while the prediction of water level of River Thames is an example of processing non-symbolic data. On the other hand, if the content of a sequence will be varying through different time steps, the sequence is called *temporal* or *time-series*: a temporal sequence consists of nominal symbols from a particular alphabet, while a time-series sequence deals with continuous, real-valued elements [13]. Processing

both these sequences mainly consists of applying the current known patterns to produce or predict the future ones, while a major difficulty is that the range of data dependencies is usually unknown. Therefore, an intelligent system or approach with memorising and learning capabilities for previous information is crucial for effective and efficient sequence processing and modelling. In this work, we concentrate on temporal sequence processing problems where nominal symbols are used to generate the sequence.

1.2 Aim of the Research

The purpose of this research is to design novel gradient-based algorithms for effective training of recurrent neural networks (RNNs) and apply them in problems of symbolic sequence processing.

1.3 Objectives

In order to achieve the above stated aim the research is organised in terms of the following objectives:

- Review gradient-based training algorithms for RNNs and relevant RNN architectures;
- Develop and implement adaptive nonmonotone strategies;
- Develop and implement first-order (i.e., resilient propagation and conjugate gradient) and second-order (i.e., BFGS quasi-Newton and

Levenberg-Marquardt) training algorithms, equipped with nonmonotone strategies;

- Test the proposed algorithms on three different RNN architectures using symbolic sequence datasets;

1.4 Methodology

Our methodology is based on theory of linear and nonlinear iterative methods, while the idea of nonmonotone learning is inspired from theories for cognitive development (see e.g. [59][58]) and recent advances in optimisation methods, which showed that nonmonotone methods possess properties, such as global and superlinear convergence, require fewer numbers of line-searches and function evaluations, and are effective for large-scale unconstrained problems.

We start by developing nonmonotone first-order training algorithms, i.e. variants of the resilient backpropagation and conjugate gradient methods, and then progressively move into second-order training algorithms, i.e. the nonmonotone BFGS quasi-Newton and Levenberg-Marquardt methods. In all cases, the convergence of the methods is discussed in the framework of deterministic optimisation. The proposed algorithms are tested and comparatively evaluated in applications from the domain of temporal processing using symbolic sequences.

1.5 Thesis Structure

In the following chapter of the thesis, we review RNN architectures and relevant training algorithms. An architectural classification scheme for RNNs is proposed [148] and three architectures from the RNN literature, i.e. *Feed-Forward Time-Delayed network* (FFTD) [196][197], *Layered Recurrent Network* (LRN) [57][85], *Nonlinear Autoregressive Network with Exogenous Inputs* (NARX) [128][137], are chosen in order to evaluate the behaviours of the novel training algorithms later on.

After dealing with RNNs architectures systematically, the formulation of neural networks' learning in the context of unconstrained optimisation is presented in Chapter 3. By introducing a general form of the objective function, the problem of nonmonotone learning is formulated in terms of unconstrained optimisation and ways to introduce nonmonotone conditions are discussed.

Four nonmonotone training algorithms are then developed and implemented. Two first-order methods, the nonmonotone Jacobi-Resilient backpropagation and the adaptive nonmonotone conjugate gradient methods are presented in Chapters 4 and 5 respectively. Then two second-order methods, the adaptive nonmonotone BFGS quasi-Newton and the adaptive nonmonotone Levenberg-Marquardt methods, are presented in Chapters 6 and 7. All methods use nonmonotone learning strategy with adaptive tuning mechanism for the learning horizon.

In Chapters 4-7, besides the parity-N problem, which is one of the classical model problems to verify the performance for new algorithms, experiments with three real-world symbolic sequences, the *Sequence Learning* problem (SL; [124]), the

Sequence Classification problem (SC; [116]), and the *Reading Aloud* problem (RA; [158]) are also provided and discussed. Our results using three different RNN architectures provide evidence that the new proposed training algorithms have potential to contribute to the research in the area of RNNs for temporal sequence processing. Details of the four classes of simulated applications and relative settings of experiments for this research can be found in Appendix A.1.

In Chapter 8, conclusions of this thesis are drawn and future work and potential benefits are discussed.

1.6 Contribution of the Thesis

In Chapter 2, the thesis reviews RNNs architectures and examines gradient-based training algorithms in order to provide a systematic view of the research area. In Chapter 3, RNN training is examined in the framework of nonlinear optimisation and a formulation of the deterministic nonmonotone learning is provided. The thesis then proposes two first-order (Chapters 4 and 5) and two second-order (Chapters 6 and 7) training algorithms. These methods generate nonmonotone behaviours by incorporating conditions that measure the sufficiency of error reduction, and an adaptive window, whose size is informed by estimating the morphology of the error surface locally.

In Chapter 4, a gradient descent based heuristic scheme, called *nonmonotone Jacobi-Rprop*, that locates an approximation of the subminimiser along each weight direction is introduced. This training scheme belongs to the Rprop class of training

algorithms, which employ sign information, and appears to provide improved convergence in high dimensional non convex functions when conditions are far from a minimiser (a situation common in RNN training), and helps avoiding convergence to local minima in some cases.

In Chapter 5, a class of nonmonotone conjugate gradient methods is proposed. The two proposed algorithms, i.e. *adaptive nonmonotone conjugate gradient* (ANM-CG) and *advanced adaptive nonmonotone conjugate gradient* (A2NM-CG), possess the property of global convergence and behave more efficiently and effectively than their original versions, in terms of fewer training epochs and lower training/testing errors.

In Chapter 6, a quasi-Newton based learning approach is presented, called *adaptive self-scaling nonmonotone BFGS* (ASCNM-BFGS) method. The proposed ASCNM-BFGS method retains the benefit of the latest works on the self scaling properties of the Hessian approximation, and the property of global convergence, and is equipped with the adaptive tuning mechanism of nonmonotone learning horizon. On the other hand, comparing to the original BFGS approach, the proposed nonmonotone revision is more effective by applying fewer number of required hidden nodes.

In Chapter 7, two Levenberg-Marquardt (LM) based nonmonotone methods with adaptive momentum (AM) terms are introduced, i.e. *adaptive nonmonotone LMAM* (ANM-LMAM) and *adaptive nonmonotone optimised LMAM* (ANM-OLMAM). The two algorithms inherit the benefits of two recently proposed Levenberg-Marquardt approaches while the use of nonmonotone strategy alleviates

some of the drawbacks of the original monotone versions.

The proposed nonmonotone algorithms are comparatively evaluated against monotone methods in four applications (see Appendix A.1), using three different RNN architectures. Results of our experiments provide evidence that the four proposed nonmonotone learning algorithms are more effective than monotone approaches outperforming in all cases in terms of convergence.

Chapter 2

Review of Recurrent Neural Networks

A Recurrent Neural Network (RNN) is an artificial neural network in which self-loops and/or backward connections between nodes are allowed [57][83][112] [129]-[131][130][173]. Comparing to feedforward neural networks, RNNs are well-known for their power to memorise time dependencies and model nonlinear systems [128]. RNNs can be trained from examples to map input sequences to output sequences and in principle they can implement any kind of sequential behaviour. They are biologically more plausible and computationally more powerful than other modelling approaches, such as Hidden Markov Models (HMMs), which have non-continuous internal states, feedforward neural networks and Support Vector Machines (SVMs), which do not have internal states at all.

One of the first RNNs was the *avalanche network* developed by Grossberg in 1969 [83] for learning and processing an arbitrary spatiotemporal pattern. Jordan's *sequential network* [96] and the *simple recurrent network* [57] were proposed later. The first RNNs did not work very well in practical applications, and their operation was poorly understood. However, several variants of these models were developed for real-world applications, such as robotics, speech recognition, music composition,

vision, and their potential for solving real-world problems has motivated a lot of research in the area of RNNs. Current research in RNNs has overcome some of the major drawbacks of the first models. Progress has come in the form of new architectures and learning algorithms, and has led in a better understanding of the RNNs' behaviours.

In summary, a RNN consists of two components, i.e., the recurrent architecture, and the learning algorithm. As has been pointed out in [131] "RNN applications have been hindered due to the high computational cost of training". Thus, the drawbacks of RNNs are mainly caused by inefficient training (e.g. when backpropagation errors are vanished) and poor generalization (e.g. when no negative samples can be used at all). In addition, as the architectures of RNNs can be generally adjusted to fit the various applications, how to select a suitable architecture among the various recurrent topologies available are consider important issues as well.

In the following subsections, architectures and learning algorithms of RNNs are reviewed and discussed. The chapter derives a more general and systematic classification scheme than the ones currently proposed and presents a review on the reported weaknesses/advantages of various implemented/applied learning methods [147][148].The chapter concludes with a summary and contribution.

2.1 Architectures

In the literature, several classification schemes have been proposed to organise RNN architectures starting from different principles for the classification, i.e. some consider the loops of nodes in the hidden layers, while others take the types of output into account. For example, they can be organised into *canonical* RNNs and *dynamic MLPs* [193]; *autonomous converging* and *non-autonomous non-converging* [24]; *locally* (receiving feedback(s) from the same or directly connected layer), *output feedback*, and *fully connected* (i.e. all nodes are capable to receive and transfer feedback signals to the other nodes, even within different layers) RNNs [54]; *binary* and *analog* RNNs [142].

From mathematical point of view [103], assuming that y and z are respectively the response of the output layer and the output of the hidden layer, a static feedforward neural network can be formulated as follows:

$$y = \phi\left(\mathbf{W}^{\mathrm{II}} z + b^{\mathrm{II}}\right), \tag{2.1}$$

and

$$z = \phi\left(\mathbf{W}^{\mathrm{I}} x + b^{\mathrm{I}}\right), \tag{2.2}$$

where $\phi(\bullet)$ denotes nonlinear activation function, $\mathbf{W}^{\mathrm{I}}$ and $\mathbf{W}^{\mathrm{II}}$ the weights of the hidden layer and the output layer, x the input vector, and b the biases. This general form could be easily transformed to describe a Feed-Forward Time-Delayed (FFTD, see Figure 2.1, [196][197]) RNN by substituting the following delayed equations

with time index *t*,

$$y(t)=\phi\left(\mathbf{W}^{\mathrm{II}}z(t)+b^{\mathrm{II}}\right), \tag{2.3}$$

$$z(t)=\phi\left(\mathbf{W}^{\mathrm{I}}s(t)+b^{\mathrm{I}}\right), \tag{2.4}$$

and

$$s(t)=\left\{x(t)\oplus x(t-1)\oplus\cdots\oplus x(t-d)\right\}, \tag{2.5}$$

where *s(t)* denotes the state vector at time *t*, $\oplus$ the Cartesian product, *d* the number of delays.

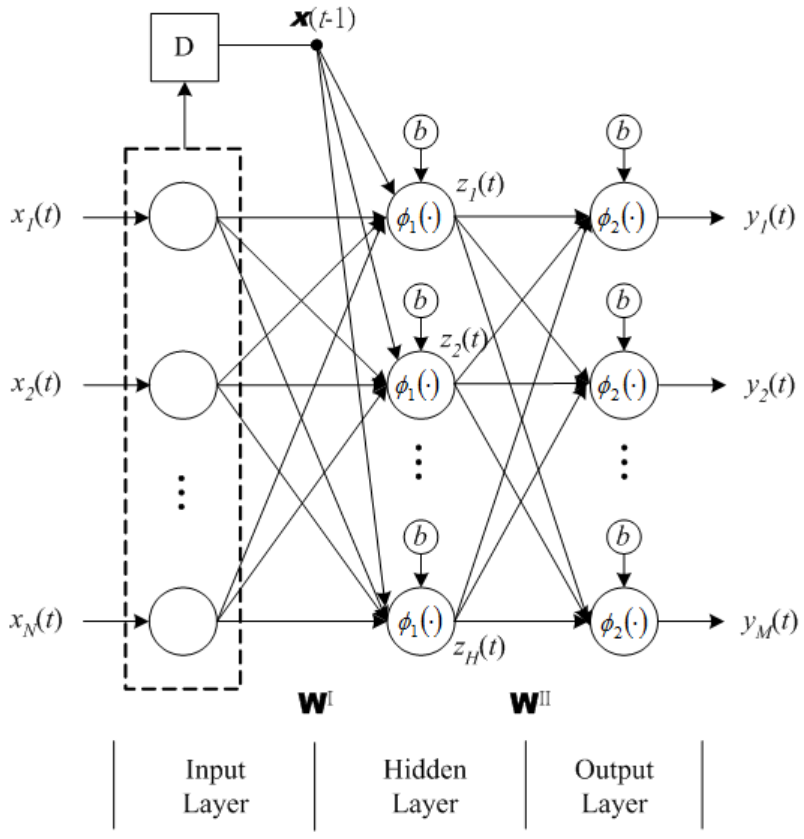


Figure 2.1 A three-layer Feed-Forward Time-Delayed network [196][197], with *N*

input, 1 time-delay, H hidden and M output nodes

By adding a feedback connection from the hidden layer to the delay unit then Eq. (2.4) can be stated as

$$z(t) = \phi\left(\Lambda z(t-1) + \mathbf{W}^{\mathrm{I}} x(t) + b^{\mathrm{I}}\right), \tag{2.6}$$

where Λ is a diagonal matrix, which describes an Elman-type RNN, also called Layered Recurrent Network or Simple Recurrent Network (LRN, see Figure 2.2, [57][85]).

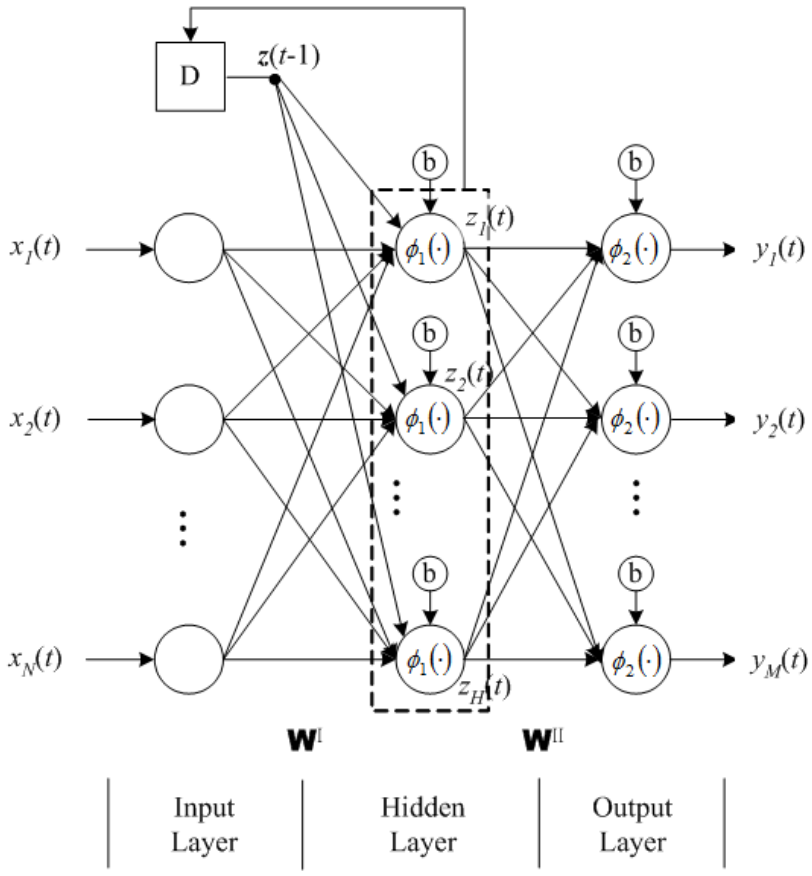


Figure 2.2 A three-layer Layered Recurrent Network [57][85], with *N* input, *H* hidden and *M* output nodes

For the Nonlinear Autoregressive Network with Exogenous Inputs (NARX, see Figure 2.3, [128][137]) the state is described as

$$s(t)=\{x(t)\oplus x(t-1)\oplus\cdots\oplus x(t-d+1)\}\oplus\{y(t-1)\oplus y(t-2)\oplus\cdots\oplus y(t-m)\},\tag{2.7}$$

where *m* is the number of output feedbacks. The formulations of a fully RNN, as shown in Figure 2.4, can also be derived by combining Eqs. (2.3) and (2.7) with the following one:

$$z(t)=\phi\left(\Lambda z(t-1)+\mathbf{W}^{\mathrm{I}}s(t)+\mathbf{W}^{\mathrm{I}}x(t)+b^{\mathrm{I}}\right).\tag{2.8}$$

Table 2.1 provides an overview of the various architectures and of the relevant literature, based on our proposed classification scheme [147][148].

In our previous works [147][148], a general scheme for the architectural classification of RNNs had been proposed. This scheme considers the networking topologies of RNNs' recurrence by revising the classification proposed by dos Santos [54], and is shown in Table 2.1. The first row of Table 2.1 divides RNNs into *local* (receiving feedback(s) from the same or directly connected layer) and *global* by considering their recurrence. It also takes their connectivity into consideration and separates RNNs as *fully* (all nodes are capable to receive and transfer feedback signals to the other nodes, even within different layers) and *partially recurrent*. Under this kind of classification, we can include all types of RNNs by creating combinations of these four classes, such as fully global RNNs and partially local RNNs. One more advantage of this classification scheme is that we can easily and

systematically observe the relationships between different RNN architectures. For example, considering the dynamic architectures with one hidden layer, as shown in Figure 2.5, the topological relationships between the FFTD, LRN, NARX, and fully RNN can be easily perceived. In Figure 2.5 the boxes with dashed line represent nodes of the input layer which receive input data and delayed versions of input, hidden or output signals, depending on the architecture.

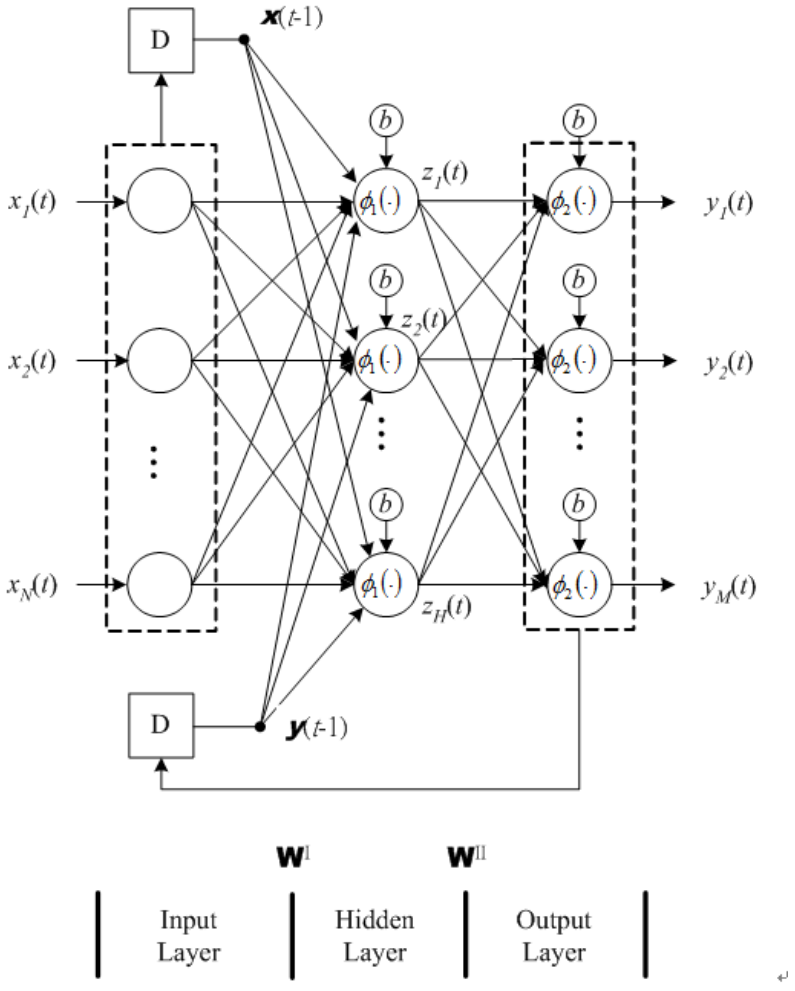


Figure 2.3 A three-layer Nonlinear Autoregressive Network with Exogenous Inputs [128][137], with *N* input, 1 time-delay, *H* hidden and *M* output nodes

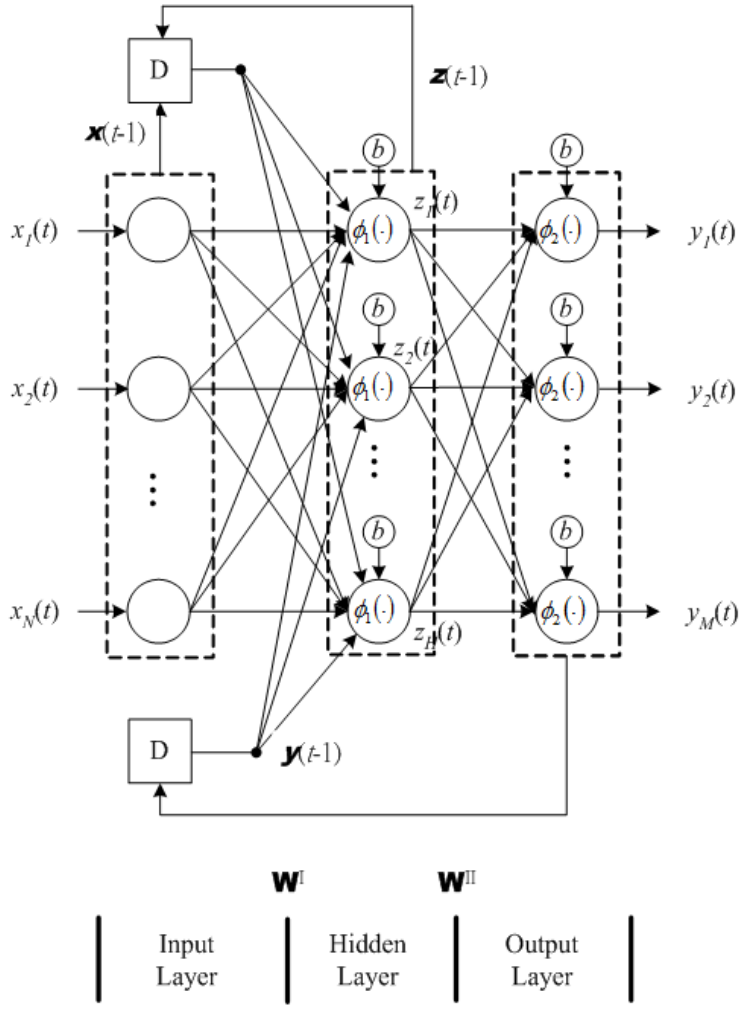


Figure 2.4 A three-layer fully recurrent network

Table 2.1 Classification summary of RNNs [148]

Type of Recurrence	Globally	Locally	Fully	Partially
References	[29][102][164]	[16][27][186][188][190]	[144]	All except [144]
Equations	(2.3)(2.4)(2.7)	(2.3)(2.4)(2.5) or (2.3)(2.5)(2.6)	(2.3)(2.7)(2.8)	(see Globally/Locally)

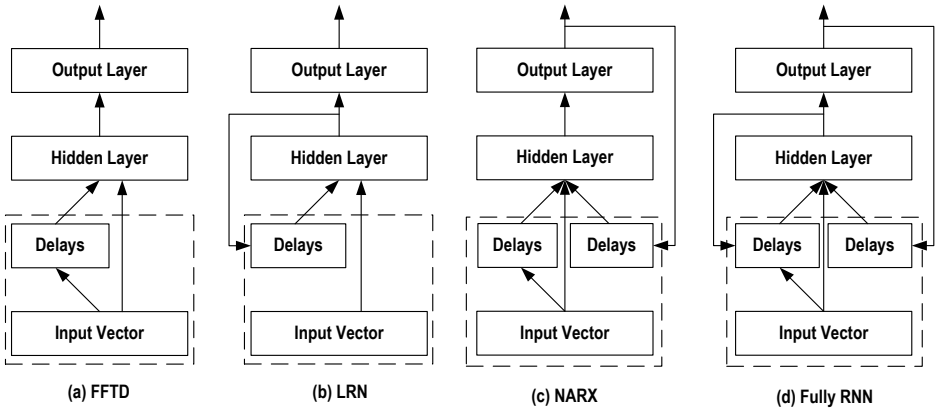


Figure 2.5 Schematics of typical dynamic neural networks architectures [148].

2.2 Training Algorithms

As quoted in [130], there are two main classes for derivative computation in RNNs, i.e., Backpropagation Through Time-BPTT and Real-time Recurrent Learning-RTRL. The former is a method for unfolding a RNN in time to build an equivalent feedforward representation so that the derivatives can be computed via standard BP method, while the latter computes derivatives at each time step by propagating the gradient history forward in time. In other words, BPTT is suitable for batch/offline learning and with *backward* derivative propagation, while RTRL is for sequential/online mode and with *forward* derivative propagation.

Followed by the nature of computing and propagating derivatives, two of the fundamental drawbacks for the BPTT hence are "batch data must be applied and the extensive memory requirements that are dictated by the need to store significant amounts of state information." [31]; while the two major disadvantages for the RTRL are large computational complexity O(N^4) [31][131] and storage requirements O(N^3)

[31], where *N* is the number of neurons. Furthermore, as indicated in [189] that "both BPTT and RTRL methods are failed to learning patterns that extend greater than 10 time-steps," the trade-off between these two classes for RNNs' derivative computation is therefore obvious and depended on applications that requiring batch or online learning.

With regards to training RNNs and storing information in their internal representations, Gradient Descent-based (GD) learning algorithms are the most commonly applied methods, even though it has been claimed that GD has some drawbacks [25]. Firstly, when the delays or recursive connections are very deep, i.e. when long-term memory is required, the backpropagation error may be vanished and the training process could become inefficiently. Secondly, the most common way to apply GD algorithms into RNN, as mentioned above, is to unfold the recursive layers and train the whole network as a feedforward network. Another drawback is that the generalisation is highly affected by the samples in the training dataset - in temporal processing it is difficult to extract or prepare negative samples from a given training dataset and the specific RNN then predicts or classifies new coming samples according to the learned knowledge only.

In [24], besides the Backpropagation Through Time (BPTT [200][201]) method and real-time gradient computation, approaches with space and/or time locality are also reviewed. However, local algorithms can be applied to some specific local feedback RNNs and for short-term memorisation only due to their inherent representation capabilities. The inefficiency of GD in learning long-term dependencies is mainly because previous information is treated initially as noise and gradually is ignored [24][25]. Therefore, two alternative algorithms are revised and discussed in Bengio's

works: the time-weighted pseudo-Newton and the discrete error propagation. The former applies the unfolding method to the pseudo-Newton optimisation and the later considers the limited case of propagation only; it has to be verified whether this would work on other more general situation or not.

Two types of learning algorithms are discussed in [143]: the *fixed point*, and the *non-fixed point*. Well-known algorithms such as the BPTT method and Real-time Recurrent Learning (RTRL) [203] are included in this classification and a way of introducing time constants and time delays is also suggested. The method of extended RTRL (eRTRL) is also discussed and other relevant approaches, such as Elman networks, Jordan networks, the moving targets method, feedforward networks with state, teach forcing in continuous time and Kalman filter are reviewed. Pearlmutter in [143] also compares the complexity both in time and space, and discusses the learning mode, and locality of these algorithms.

For fully connected hidden layer networks and dynamic MLPs, Tsoi [194] has investigated two first-order gradient learning algorithms. This work discusses some drawbacks of these methods, such as slow convergence and generalisation, and derives two 2^{nd}-order approaches to speed up the convergence and to tackle the issue of weight pruning; it also provides a discussion on output sensitivity. The lower sensitivity of output to a specific adjustable parameter, the better performance of the network is. Although the related formulas are well defined in this work [194], there is still a crucial constant which is used to set the level of sensitivity that should be defined by the users. Quasi-2^{nd} order methods, such as conjugate gradient, scaled conjugate gradient, and the Newton approach have been also discussed and suggested as suitable only for batch training, while Kalman filtering and the

extended Kalman filter are classified as 2^{nd}-order GD based learning algorithms, which can be used under online mode, where extended Kalman filter could be used to prune weights from a RNN.

Kremer [103] reviews 14 kinds of memories used in spatiotemporal connectionist networks, capable of computing the state vectors, and provides a general formulation for computing output vectors. The author also summarises 10 different kinds of updating rules, such as full GD, truncated GD, auto-associative GD, and stack learning. It examines three open issues: the temporal credit assignment, the representation capabilities and the knowledge encoding.

From the point of view of time-series modelling, Kolehmainen's work in [99] covers BPTT and RTRL for learning RNNs, while Dietterich [52] suggests BPTT. In the same vein with [18] and [143], fixed point networks are also considered and five relative algorithms, such as BPTT and GD learning of time constants, gains and delays are summarised.

Some attempts have been made to propose second-order learning algorithms, e.g. dos Santos & von Zuben [54] proposed a quasi 2^{nd}-order method. Also, simulated annealing has given some promising results but the training time is relatively higher [25]. Table 2.2 provides an overview of RNNs learning, giving examples of training algorithms for locally and globally RNNs for various applications.

2.3 Summary and Contribution of the Chapter

Artificial neural networks (ANNs) can be classified according to their states into static and dynamic, and if considering their connective topologies only they are non-recurrent (feedforward) and recurrent.

In this Chapter, the architectures and learning algorithms for RNNs, mainly for temporal sequence processing problems, were reviewed and the challenges involved in training RNNs for sequence processing were discussed. As reviewed in Section 2.1, many schemes have been proposed in the literature in order to classify RNNs architectures. In relation to these works, we proposed a more general classification scheme [148], which is based on the ideas of [54] (for classified criteria) and [103] (for mathematical equations). From our proposed scheme [148] the relationships between the four sub-classes, i.e., globally, locally, fully, and partially, can be presented more systematically.

In addition, our review showed that, despite the drawbacks of the GD-based methods on various applications, most RNNs are trained by first-order learning algorithms and that no attempt has been made to apply nonmonotone approaches yet. Part of contents in this Chapter has been published in [148].

Table 2.2 Recurrent neural networks applications and learning algorithms

Recurrent	Applications	Algorithms*	Typical Problems Encountered	Reference
Locally	Time series prediction: sunspots, Mackey-Glass, laser, reservoir inflows	BPTT, RTRL, DEKF	• Limited choice of cost functions • Occasional instability in the convergence of GD[1] • Prior knowledge required about system tuning • Oversized architecture, poor generalisation	[16][27][28][151]
	Symbolic transduction and Prediction: grammatical inference	BPTT, RTRL, DEKF	• High computational complexity • Not suitable for complex sequences[2]	[102][151]
	Classification of structures	BP, eBPTT, eRTRL, rBP	• Prior knowledge required	[29][175][186][206]
	System Identification	Casual rBP	• Slightly higher computational complexity	[36]
	Signal processing	Rprop, COM	• Drawbacks of GD methods	[69][188]
	Moore Machine	SpikeProp-TT	• Discontinuous error surfaces • Memory vanishing • Suitable for simpler patterns	[190]
Globally	Time series generation/prediction	DEKF	• Suitable for short-term and noiseless patterns	[163][164]
	Sequences classification	BP	• Drawbacks of GD methods	[29]

* The notation used here is: BP- backpropagation; BPTT- BP through time; eBPTT- extended BPTT; RTRL- real time recurrent learning; eRTRL- extended RTRL; rBP- recurrent/recursive BP.; COM- combination of gradient descent, truncated BPTT and RTRL; DEKF- Decouple Extended Kalman Filter; Rprop- Resilient BP; SpikeProp TT- Spike Propagation Through Time

[1] This can be due to sensitivity to initial conditions and the multitudes of local minima [2] For example sequences in the area of natural language processing

Chapter 3

Monotone and Nonmonotone Learning in an Unconstrained Optimisation Framework

In this chapter the training of RNNs is treated in the framework of unconstrained minimisation, as the general training goal for RNNs is to achieve a small-enough error which will hopefully lead to good generalisation. Although this approach does not directly address the generalisation problem, it has been very popular in neural networks training [121] because it offers flexibility in the derivation of learning algorithms by exploiting advances in nonlinear optimisation. In this context, conventional learning algorithms so far operate in a monotone way, i.e. the value of the target function f should be constantly reduced at each iteration. In the next subsection, the problem in the framework of unconstrained optimisation is firstly formulated, and then a brief review of classical monotone algorithms for unconstrained optimisation is provided in Section 3.2. As nonmonotone approaches are proved to be more efficient than their monotone versions, Section 3.3 introduces the classical theory of nonmonotone learning and provides a review of recent variations. Section 3.4 concludes this Chapter.

3.1 Neural Networks' Monotone Learning as Unconstrained Minimisation

Let us consider training an RNN as the following unconstrained optimisation problem

$$\min f(w),\ w \in R^n, \tag{3.1}$$

where $f : R^n \to R$ is the learning error function and its gradient, $g = g(w) = \nabla f(w)$, is available through the method of backpropagation through time (BPTT [200]).

Let the current approximation to the solution of the above problem be w_k, and if $g_k = \nabla f(w_k) \neq 0$, then, in some way, an iterative method finds a stepsize α_k along a search direction d_k, and computes the next approximation w_{k+1} as follows:

$$w_{k+1} = w_k + \alpha_k d_k. \tag{3.2}$$

Traditional optimisation strategies for RNNs (and for unconstrained optimisation) are monotone ones, i.e. these strategies compute a step length that reduces the error function value at each iteration:

$$f_{k+1} \leq f_k, \tag{3.3}$$

which is the most straight-forward way to minimise an objective function. Unfortunately, even when an algorithm is proved to be globally convergent, there is no guarantee that the method will efficiently explore the search space in the sense that it may be trapped in a local minimum point early on and never jump out to a global one under ill conditions [71], such as poorly initialised weights.

Several attempts have been made to use algorithms from the field of unconstrained

optimization in training neural networks. In the rest of this section, we look at the issue from the perspective of monotone learning and consider two broader classes of techniques for achieving monotone convergence, i.e. line-search and trust-region.

Considering the error function, from unconstrained optimisation point of view there are two concerns: number of variables (*univariate* vs. *multivariate*) and shape of the objective function (*smooth* vs. *non-smooth*). When derivatives are available the algorithms for unconstrained optimisation can be classified as *first-derivative* (e.g. quasi-Newton and steepest descent), *second-derivative* (e.g. Newton methods), and *non-derivative* methods (e.g. finite difference approximation) [71]. In the context of neural networks, the error function should be generally smooth enough - typical examples are the total sum of squared errors, or the mean squared error – and derivatives are available through backpropagation or BPTT.

Various schemes to organise unconstrained optimisation algorithms have been proposed in the literature:

- Greig's [78]: *line-search*, *general-search*, *gradient*, and *Newton* (including quasi-Newton) methods;
- Scales's [172]: *univariate minimization* (e.g. polynomial interpolation/ extrapolation and hybrid methods), *multivariate minimization* (e.g. steepest descent, Newton, quasi-Newton and conjugate gradient methods) and *non-linear least squares* (e.g. small residue and large residue methods);
- Fletcher's [68]: *Newton-like* (e.g. Newton and quasi-Newton methods), *conjugate direction* (conjugate gradient and direction set methods), *restricted step* (e.g. Levenberg-Marquardt methods), and *sums of squares and nonlinear equations* (e.g. over-/well-determined systems and no-derivative methods);

In terms of the technique used to achieve monotone convergence, nonlinear optimisation methods can be organised into two boarder classes: ***step-length-based*** (or ***line-search***) methods and ***trust-region*** (or ***restricted-step***) methods (pp. 105 and 113, [71]; pp. 21, [68]). The basic structure of the *k*-th iteration for a line-search (LS) method [68] is as follows:

(LS-a) determine a direction of search d_k;

(LS-b) find a step-length α_k to minimise $f\left(w_k + \alpha_k d_k\right)$ with respect to α;

(LS-c) set $w_{k+1} = w_k + \alpha_k d_k$.

Let, at iteration *k*, q_k be a quadratic function, while g_k and G_k respectively are the first and second derivatives of object function *f*. For a trust-region (TR) method it takes the following form [68][180]:

(TR-a) given w_k and h_k, calculate g_k and G_k;

(TR-b) solve δ_k in

$$\min_{\delta} q_k\left(\delta\right) \text{ subject to } \left\|\delta\right\| \le h_k;$$

(TR-c) evaluate $f\left(w_k + \delta_k\right)$ and $r_k = \Delta f_k / \Delta q_k$,

where $\Delta f_k = f_k - f\left(w_k + \delta_k\right)$ and $\Delta q_k = f_k - q_k \delta_k$;

(TR-d) if $r_k < 0.25$, set $h_{k+1} = \left\|\delta_k\right\|/4$;

if $r_k > 0.75$, and $\left\|\delta_k\right\| = h_k$ set $h_{k+1} = 2h_k$;

otherwise, set $h_{k+1} = h_k$;

(TR-e) if $r_k \le 0$, set $w_{k+1} = w_k$; else, set $w_{k+1} = w_k + \delta_k$;

while, as indicted in (p. 96, [68]), the above constants 0.25, 0.75, etc. are arbitrary and the algorithm is quite insensitive to their change.

Different line-search approaches may correspond to different ways of choosing d_k in step (LS-a), while step (LS-b) is the so-called line-search *sub-problem* and is carried out by repeatedly sampling *f*(*w*) and possibly its derivatives for different points $w = w_k + \alpha_k d_k$ along the line. In the ideal case, in step (LS-b) the exact minimising value of α is required (an exact line-search) but this cannot be implemented in practice in a finite number of operations. "Essentially the nonlinear equation $df/d\alpha = 0$ must be solved." (p. 21, [68]) "Also it may be that the minimizing value of α_k might not exist ($\alpha_k = \infty$). Nonetheless the idea is conceptually useful, and occurs in some idealized proofs of convergence. In this respect it is convenient to point out the consequential property that the slope $df/d\alpha$ at α_k must be zero, which gives"

$$g_{k+1}^T d_k = 0. \tag{3.4}$$

On the other hand, both line-search and trust-region methods have the following common features (pp. 114-115, [71]):

- "If the function is well-behaved, both classes of methods are designed to become equivalent to Newton's method as the solution is approached."
- "The search direction is implicitly defined by a scalar that is adjusted according to the degree of agreement between the predicted and actual change in the target function."
- "If the Hessian matrix G_k is indefinite and norm of the gradient $\|g_k\|$ is small or zero, both types of methods must compute a direction of negative curvature from a factorization of the Hessian matrix or a modified version of it."
- "If the Hessian matrix becomes indefinite, both types of methods compute w_{k+1}

based on information from the positive-definite part of G_k."

3.2 Theory of Nonmonotone Learning

As mentioned in the previous sections, minimisation problems in the context of unconstrained optimisation are typically dealt with by trying to reduce the value of objective function *f* at each iteration – an approach called *monotone learning* in this thesis. Although, the thesis examines the problem of training RNN in the context of unconstrained nonlinear optimisation, the proposed approaches can also be considered relevant to the nonmonotone way learning occurs in cognitive development [57]. From optimisation perspective, nonmonotone strategies have been proved to offer several advantages, such as the properties of global and superlinear convergence, fewer numbers of line searches and function evaluations, and have demonstrated effectiveness for large-scale unconstrained optimisation problems [60][78]-[81], which is a valuable property when training RNNs, as this usually involves optimising several hundred free parameters.

The motivation for nonmonotone learning is the need to better explore the search space and accelerate the convergence rate. Nonmonotone strategies, such as the one introduced in [80] for Newton's method, take into consideration the *M* previous *f* values by using a constraint such as the following one

$$f\left(w_k+\alpha_k d_k\right)\le \max_{0\le j\le M}\left\{f\left(w_{k-j}\right)\right\}+\delta\alpha_k g_k^T d_k, \qquad (3.5)$$

where *M* is a nonnegative integer and constant $\delta\in(0,1)$. In the neural networks context, *M* has been named nonmonotone learning horizon [156].

The proposed nonmonotone approach exploits conditions of the form of Eq. (3.5), by varying the search direction depending on the designed algorithm and a nonmonotone learning horizon M that is adapted using local estimations of the Lipschitz constant – an approach originally introduced in [119] to adapt the learning rate. This allows us to avoid using a poorly user-defined nonmonotone learning horizon and exploit the morphology of the error function through a local estimation of the Lipschitz constant to determine the size of M dynamically. This technique for tuning M proved to work well in our previous work in the context of static feedforward backpropagation networks [156]. Further details on the tuning the learning horizon M and our proposed nonmonotone approaches are presented in Section 3.2.2, as well as in the later-presented sections of proposed algorithms in Chapters 4-7 of this thesis.

It is worth mentioning that learning algorithms with momentum terms, such as the well known momentum backpropagation, do not belong by default to the class of nonmonotone algorithms discussed here. Although momentum backpropagation may occasionally exhibit nonmonotone behaviour, it does not formally apply a nonmonotone strategy, such as the one derived in Eq. (3.5). Nevertheless, nonmonotone strategies can be combined with momentum parameters, such as in the work of [156] that proposed *nonmonotone learning with momentum* and compared it with momentum backpropagation.

In the rest of this section, after presenting some well known and recently proposed nonmonotone conditions, the discussion of nonmonotone learning horizon is provided.

3.2.1 Nonmonotone Conditions

The first nonmonotone strategy was proposed in [80], which is a line-search method for Newton methods. Let α_k be a stepsize, d_k the search direction, g the gradient of f, $x_{k+1} = x_k + \alpha_k d_k$, $0 < \delta < \beta < 1$, σ_1, σ_2 two forcing functions, M a nonnegative integer. Then the Grippo-Lampariello-Lucidi's (GLL) nonmonotone Newton method [80] can be stated as shown in Table 3.1.

Table 3.1 the GLL nonmonotone Newton algorithm

Algorithm: Grippo-Lampariello-Lucidi's nonmonotone Newton method

Step 0. Initialise w_0, integer $M \geq 0$, $c_1 > 0$, $c_2 > 0$, $c_3 \in (0,1)$, and $c_4 \in (0,1)$;

Step 1. Set $k = 0$, $m(0) = 0$ and compute $f_0 \triangleq f(w_0)$;

Step 2. If $g_k = 0$, stop;

Else-if H_k is singular, set $d_k = -g_k$, $m(k) = 0$, and go to Step 5;

Step 3. Compute $d_k = -H_k^{-1} g_k$;

If $\left| g_k^T d_k \right| < c_1 \left\| g_k \right\|^2$ or $c_2 \left\| g_k \right\| < \left\| d_k \right\|$, set $d_k = -g_k$, $m(k) = 0$,

and go to Step 5;

Step 4. If $g_k^T d_k > 0$, set $d_k = -d_k$;

Step 5. Set $\alpha = 1$;

Step 6. Compute $f_\alpha = f(w + \alpha d_k)$;

If $f_\alpha \leq \max\limits_{0 \leq j \leq m(k)} \left[f_{k-j} \right] + c_3 \alpha g_k^T d_k$, set $f_{k+1} = f_\alpha$, $w_{k+1} = w_k + \alpha d_k$,

$k = k + 1$, $m(k) \leq \min\left[m(k-1) + 1, M \right]$ and go to Step 2;

Step 7. Set $\alpha = c_4 \alpha$ and go to Step 6

In the following, after presenting the main theoretical result (Theorem 3.1) of [80], other variations of the nonmonotone conditions are reviewed.

□ **Theorem 3.1** [80]. *Let* $\{w_k\}$ *be a sequence defined by* $w_{k+1} = w_k + \alpha_k d_k$, *where* $d_k \neq 0$. *Let* $a > 0$, $\sigma \in (0,1)$, $\gamma \in (0,1)$ *and let M be a nonnegative integer. Assume (**A**) that*

(A1). *The level set* $\Omega_0 \triangleq \{w: f(w) \leq f(w_0)\}$ *is compact;*

(A2). *There exist positive numbers* c_1, c_2 *such that*

$$g_k^T d_k \leq c_1 \|g_k\|^2, \tag{3.6}$$

$$\|d_k\| \leq c_2 \|g_k\|; \tag{3.7}$$

(A3). $\alpha_k = \sigma^{h_k} a$, *where* h_k *is the first nonnegative integer h for which*

$$f(w_k + \sigma^h a d_k) \leq \max_{0 \leq j \leq m(k)} [f(w_{k-j})] + \gamma \sigma^h a g_k^T d_k, \tag{3.8}$$

where $m(0) = 0$ *and*

$$0 \leq m(k) \leq \min[m(k-1)+1, M], \quad \forall k \geq 1. \tag{3.9}$$

*Then (**T**),*

(T1). *The sequence* $\{w_k\}$ *remains in* Ω_0 *and every limit point* $\bar{w}$ *satisfies* $g(\bar{w}) = 0$;

(T2). *No limit point of* $\{w_k\}$ *is a local maximum of f;*

(T3). *If the number of the stationary points of f in* Ω_0 *is finite, the sequence* $\{w_k\}$ *converges.* □

When the integer *M* in Step 6 in Table 3.1 and Eq. (3.8) is set to zero, the GLL nonmonotone steplength selection rule is reduced to the standard Armijo rule, and therefore, it can be viewed as a generalisation of the Armijo's rule, as indicated in [80].

The ***GLL nonmonotone linesearch*** (NM-GLL) can be summarised as below [113][114].

$$\begin{cases} f\left(w_{k+1}\right) \le \max\limits_{0\le j\le m(k)}\left\{ f\left(x_{k-j}\right)\right\} + \varepsilon_3 \alpha_k g_k^T d_k \\ \left|\left\langle g\left(w_{k+1}\right), d_k \right\rangle\right| \le -\varepsilon_4 g_k^T d_k \end{cases}, \tag{3.10}$$

where $0 < \varepsilon_3 < \varepsilon_4 < 1/2$ and $m(k)$ is defined as in Eq. (3.9). Note that in the original work of Grippo et al., the settings of constants for scaling the amount of changes (in Eqs. (3.6) and (3.8)) are more relaxed.

In the work of [87], the referred *GLL linesearch* is as follows.

$$f\left(w_{k+1}\right) \le \max_{0\le j\le M}\left\{ f\left(w_{k-j}\right)\right\} + \varepsilon_1 \alpha_k g_k^T d_k, \tag{3.11}$$

and

$$g\left(w_{k+1}\right) d_k \ge \max\left\{\varepsilon_2, 1-\left(\alpha_k \left\|d_k\right\|\right)^p\right\} g_k^T d_k, \tag{3.12}$$

where $p \in \left(-\infty, 1\right)$, $\varepsilon_1 \in \left(0,1\right)$, and $\varepsilon_2 \in \left(0, 1/2\right)$. Although the term "GLL" seems to represent Grippo-Lampariello-Lucidi, comparing to Step 3 in Table 3.1 and Eq. (3.6), Han and Liu [87] had actually extended the form of $g\left(w_{k+1}\right) d_k$ in Eq. (3.10), by the curvature condition of the Wolfe linesearch. And the Wolfe linesearch is a special case of this GLL linesearch [87]. The work of [207] was based on the approach of [87].

The ***nonmonotone Armijo rule*** (NM-Armijo, see e.g. [46][100][185][187]) is revised from the standard Armijo's rule [14] and can be described as follows. Let $a > 0$, $\gamma \in (0,1)$ and $\beta \in (0,1)$. For each *k*, let *m*(*k*) satisfy $m(0) = 0$ and *m*(*k*) satisfy the definition in Eq. (3.9). Let

$$\lambda_k = \beta^{p(k)} a \tag{3.13}$$

and *p*(*k*) be the smallest nonnegative integer *p* such that

$$f(w_{k+1}) \le \max_{0 \le j \le m(k)} \{ f(w_{k-j}) \} + \gamma \lambda_k g_k^T d_k , \tag{3.14}$$

where $w_{k+1} = w_k + \lambda_k d_k$. Note that the way of tuning steplength of this approach [185][187], i.e. *p*(*k*) in Eq. (3.13), is different to the GLL nonmonotone linesearch, i.e., see assumption (A3) in Theorem 3.1. But in [46], the assumption (A3) in Theorem 3.1 was still applied, as the monotone Armijo's rule. Dai also proposed a revised way of NM-Armijo linesearch as: if NM-Armijo hold, apply the stepsize; else, consider standard monotone Armijo linesearch, i.e. $m(k) \equiv 0$. Another work referring to NM-Armijo rule is [182] but it takes a slightly different way for updating $m(k)$: $m(k) = \min\{m(k-1), M\}, \quad \forall k \ge 1.$

In [187], there is another referred linesearch approach, i.e., the ***nonmonotone Goldstein rule*** (NM-Goldstein), derived from Goldstein's rule [74][73], which can be defined as below:

$$\max_{0 \le j \le m(k)} \{ f(w_{k-j}) \} + \mu_2 \lambda_k g_k^T d_k \le f(w_{k+1}) \le \max_{0 \le j \le m(k)} \{ f(w_{k-j}) \} + \mu_1 \lambda_k g_k^T d_k , \tag{3.15}$$

where $0 < \mu_1 \le \mu_2 < 1$. Sun et al. also proposed the ***nonmonotone F-rule*** (NM-F) as stated in Eq. (3.16). Let *m*(*k*) satisfy the definition in Eq. (3.9) and $\lambda_k \ge 0$ be bounded above and satisfy

$$f\left(w_{k+1}\right) \le \max_{0 \le j \le m(k)} \left\{ f\left(w_{k-j}\right) \right\} - \sigma\left(t_k\right), \tag{3.16}$$

where σ is a forcing function and

$$t_k = -\frac{g_k^T d_k}{\left\| d_k \right\|}. \tag{3.17}$$

It has been also proved [187] that the ***NM-Armijo***, ***NM-Goldstein*** and ***NM-Wolfe*** rules are the special cases of the ***NM-F*** linesearch approach.

Deriving from the standard monotone Wolfe's rule [203][205], the ***nonmonotone Wolfe conditions*** (NM-Wolfe, see e.g., [101][113][114][187]) are defined as:

$$\begin{cases} f\left(w_{k+1}\right) \le \max\limits_{0 \le j \le M} \left\{ f\left(w_{k-j}\right) \right\} + \varepsilon_3 \alpha_k g_k^T d_k \\ g\left(w_{k+1}\right)^T d_k \ge \varepsilon_4 g_k^T d_k \end{cases}, \tag{3.18}$$

but $0 < \varepsilon_3 \le \varepsilon_4 < 1$ and the nonmonotone horizon is a constant, with no iterative upper bound. As in the case of NM-Armijo (and other nonmonotone approaches), when the nonmonotone horizon is set to zero, i.e. considering only one previous function value f_k, NM-Wolfe is reduced to the standard monotone Wolfe rule.

In [61][104] the following nonmonotone linesearch was proposed,

Step_0. set $\alpha = 1, \quad \gamma \in (0,1), 0 < \sigma_1 < \sigma_2 < 1, M > 1;$

Step_1. if

$$f\left(w_{k+1}\right) \le \max_{0\le j\le m(k,M)} \left\{ f\left(w_{k-j}\right)\right\} - \gamma\alpha^2 \left\|d_k\right\|^3, \tag{3.19}$$

set $\alpha_k = \alpha$ and stop.

Step_2. choose $\sigma \in \left[\sigma_1, \sigma_2\right]$, set $\alpha = \sigma\alpha$ and go to Step_1.

while in the approach of [55], the condition for constants further relaxed as $0 < \rho < \sigma_3 < \sigma_4$,

$$\begin{cases} f\left(w_{k+1}\right) \le \max\limits_{0\le j\le m(k)} \left\{ f\left(w_{k-j}\right)\right\} + \rho\alpha_k g_k^T d_k \\ \sigma_3 g_k^T d_k \le g\left(w_{k+1}\right)^T d_k \le \sigma_4 g_k^T d_k \end{cases}, \tag{3.20}$$

where *m*(*k*) satisfies the definition stated in Eq. (3.9).

The work of [181] is as follows. Given $\mu \in \left(0, 0.5\right)$, $\rho \in \left(0,1\right)$, and $c \in \left(0.5,1\right)$,

$$s_k = \frac{1-c}{L_k} \cdot \frac{\left\|g_k\right\|^2}{\left\|d_k\right\|^2} \tag{3.21}$$

and α_k is the largest α in $\left\{s, s\beta, s\beta^2, \cdots\right\}$ such that

$$\begin{cases} f\left(w_{k+1}\right) \le \max\limits_{0\le j\le m(k)} f\left(w_{k-j}\right) + \mu\alpha g_k^T d_k \\ g_{k+1}^T d_{k+1} \le -c\left\|g_{k+1}\right\|^2 \end{cases} \tag{3.22}$$

where *m*(*k*) is defined as in Eq. (3.9), and the new search direction d_{k+1} is defined as,

$$d_{k+1} = -g_{k+1} + \frac{g_{k+1}^T\left(g_{k+1} - g_k\right)}{\left\|g_k\right\|^2} d_k \tag{3.23}$$

and the estimated Lipschitz constant L_k in Eq. (3.21) is defined as,

$$L_k = \max\left(L_{k-1}, \frac{\|y_{k-1}\|}{\|\delta_{k-1}\|} \right), \tag{3.24}$$

or

$$L_k = \max\left(L_{k-1}, \frac{\delta_{k-1}^T y_{k-1}}{\|\delta_{k-1}\|^2} \right), \tag{3.25}$$

with $\delta_{k-1} = w_k - w_{k-1}$ and $y_{k-1} = g_k - g_{k-1}$.

Another work proposed by Shi and Shen [182] can be described as follows. Given $\sigma \in (0, 0.5)$, $\beta \in (0,1)$, $\delta \in [0.5, 2)$, B_k is the approximated Hessian matrix and

$$s_k = -\frac{\delta g_k^T d_k}{d_k^T B_k d_k}. \tag{3.26}$$

Choose α_k to be the largest α in $\{s, s\beta, s\beta^2, \cdots\}$ such that

$$f(w_{k+1}) \le \max_{0 \le j \le m(k)} f(w_{k-j}) + \sigma\alpha\left[g_k^T d_k + \frac{1}{2}\alpha d_k^T B_k d_k \right], \tag{3.27}$$

where $m(k)$ is defined as in Eq. (3.9) and B_k is updated by either BFGS, DFP or other quasi-Newton formulae.

Yin and Du [209] proposed two nonmonotone conditions, the first one is:

$$\begin{cases} f(w_{k+1}) \le \max\limits_{0 \le j \le M} \{ f(w_{k-j}) \} - \delta \min\{ \sigma_1(\mu_k), \sigma_2(\nu_k) \} \\ g(w_{k+1})^T d_k \ge \beta g_k^T d_k \end{cases}, \tag{3.28}$$

where

$$\mu_k = -\frac{g_k^T d_k}{\|d_k\|}, \tag{3.29}$$

note $\mu_k = -t_k$, which is relevant to the term in Eq. (3.17),

and

$$\nu_k = -\alpha_k g_k^T d_k. \tag{3.30}$$

The second nonmonotone condition in [209] is defined as below:

$$\begin{cases} f\left(w_{k+1}\right) \le C_k - \delta \min\left\{\sigma_1\left(\mu_k\right), \sigma_2\left(\nu_k\right)\right\} \\ g\left(w_{k+1}\right)^T d_k \ge \beta g_k^T d_k \end{cases}, \tag{3.31}$$

where μ_k and ν_k are the same terms as in Eqs. (3.29) and (3.30), $C_0 = f\left(w_0\right)$, $Q_0 = 1$ and

$$C_{k+1} = \frac{\eta_k Q_k C_k + f\left(w_{k+1}\right)}{Q_{k+1}}, \tag{3.32}$$

with $\phi_k \in [0,1]$, and

$$Q_{k+1} = \phi_k Q_k + 1. \tag{3.33}$$

One of the latest works on nonmonotone conditions is [210]: let $\lambda \in (0,1]$, $M \ge 1$,

$$m_k = \min\left\{k+1, M\right\}, \tag{3.34}$$

(note that m_k here is different to Eq. (3.9), i.e. considering the iteration counter *k*, not the previous size of learning horizon m_{k-1}), $\lambda_{kr} \geq \lambda,\ \ r = 0,1,2,\ldots,m_k - 1$ and $\hat{m} = m_k - 1,$

$$\sum_{r=0}^{\hat{m}} \lambda_{kr} = 1. \tag{3.35}$$

Given $\rho_2 > \rho_1 > 0,\ \ 1 > \eta_1 > 0,\ \ \eta_2 > 0, 1 > \theta > 0,$

$$\tau_k = \frac{\left| g_k^T d_k \right|}{\left\| d_k \right\|^2}, \tag{3.36}$$

$\Delta_k \in \left[\rho_1 \tau_k, \rho_2 \tau_k \right]$ and

$$\alpha_k = \max\left[\theta^j \Delta_k, j = 1,2,\ldots \right], \tag{3.37}$$

such that

$$f\left(w_{k+1}\right) \leq \max\left[f\left(w_k\right), \sum_{r=0}^{m\_k-1} \lambda_{kr} f\left(w_{k-r}\right) \right] + \eta_1 \alpha_k g_k^T d_k - \eta_2 \alpha_k^T \left\| d_k \right\|^2. \tag{3.38}$$

All algorithms applying the above nonmonotone conditions have been proved to have the property of global convergence. More details can be found in the original works and relative literature.

3.2.2 Nonmonotone Learning Horizon

The key idea of nonmonotone learning is that, for the target function *f*, to consider not only the current one iteration f_k but also the previous *M* function values and

therefore, an increase of function value f is allowed. In the 13 versions of nonmonotone conditions mentioned in previous subsection, the ways of determining the size of the nonmonotone learning horizon M are mainly the variations of Eq. (3.9), while Eqs. (3.11), (3.18) and (3.28) use a fixed value. However, choosing a fixed or non-adaptive value for all applications is not the proper way of determining the nonmonotone learning horizon because the curvature of specific target functions for different applications may have totally different features [156].

As stated in [119][156][195], it is well-known that the Lipschitz constant is closely related to the morphology of a function, i.e. for a function having steep regions, the Lipschitz constant is large and when the function is flat the Lipschitz constant is small. However, in neural networks training practice, neither the morphology of the error surface nor the value of the Lipschitz constant are known in advance [119]. Therefore, the following procedure can provide a dynamical way to adapt the size of the nonmonotone learning horizon at k-th iteration, without additional evaluations of target function or gradient:

$$M_k = \begin{cases} M_{k-1}+1, & \text{if } \Lambda_k < \Lambda_{k-1} < \Lambda_{k-2} \\ M_{k-1}-1, & \text{if } \Lambda_k > \Lambda_{k-1} > \Lambda_{k-2}, \\ M_{k-1}, & \text{otherwise,} \end{cases} \tag{3.39}$$

where Λ_k is the local approximation of the Lipschitz constant and defined by

$$\Lambda_k = \frac{\|g_k - g_{k-1}\|}{\|w_k - w_{k-1}\|}. \tag{3.40}$$

If Λ_k constantly increases during consecutive epochs, i.e. the second condition in Eq. (3.39) is satisfied, it indicates that the sequence of weight vectors w_k approaches a steep region, and the value of M_k should be decreased to "avoid

overshooting a possible minimum point" [156]. On the other hand, when the first condition in Eq. (3.39) is satisfied, the current point may possibly enter a valley in the weight space and M_k should be increased in order to enlarge the learning horizon and fasten the convergent speed.

All proposed nonmonotone learning algorithms in this thesis apply the above adaptive procedure, i.e. Eqs. (3.39)-(3.40), to dynamically determine the size of nonmonotone learning horizon. As mentioned in all relative works, M_k has to be a non-negative integer. Since our experimental results show that the upper bound of M_k ($M^{\max}$) is not critical for different applications and different recurrent architectures, i.e. there is no significant difference between $M^{\max}=100$, $M^{\max}=50$ and $M^{\max}=15$, the boundaries of M_k is set to $3\le M_k\le 15$, for all the simulations in this thesis.

3.3 Summary and Contribution of the Chapter

The unconstrained-optimisation formulation of neural networks learning reveals the possibility of applying a large base of algorithms from the field of unconstrained optimisation into the ANNs training problem. The review of unconstrained optimisation provides a basic and rough sketch of the classical minimisation approaches for this research. As indicated in Chapter 2, the most commonly applied training algorithms for RNNs belong to the class of the gradient-based methods, even it has been claimed may not be the best choice.

This chapter also presented a theory for nonmonotone learning, while the basic steps

of the nonmonotone learning strategy at the *k*-th iteration [156] can be summarised as:

(1). Update the weights $w_{k+1} = w_k + \alpha_k d_k$;

(2). If a nonmonotone condition, such as the one in Eq. (3.8), is satisfied, store w_{k+1}, set $k = k+1$ and go to Step (1); else, go to Step (3);

(3). Use a tuning technique for α_k and return to Step (2).

Different approaches to determine the steplength α_k and search direction d_k will compose different algorithms for unconstrained optimisation. As indicated in [156], the nonmonotone learning strategy can be incorporated in any batch training algorithm as a sub-procedure that secures and accelerates the convergence of a batch training algorithm.

Forcing functions are the terms, such as $+\gamma\sigma^h a g_k^T d_k$ in Eq. (3.8), $-\sigma(t_k)$ in Eq. (3.16), $-\gamma\alpha^2\|d_k\|^3$ in Eq. (3.19), $+\sigma\alpha\left[g_k^T d_k + \frac{1}{2}\alpha d_k^T B_k d_k\right]$ in Eq. (3.27), $-\delta\min\{\sigma_1(\mu_k), \sigma_2(\nu_k)\}$ in Eq. (3.28), and $+\eta_1\alpha_k g_k^T d_k - \eta_2\alpha_k^T\|d_k\|^2$ in Eq. (3.38), used to guarantee a *sufficient* change for the next function value f_{k+1}. From the mentioned examples in Subsection 3.3.1, it is obvious that there is not only one type of forcing functions can be applied. In this thesis, the one in Eq. (3.8) was chosen for the four proposed nonmonotone algorithms.

Lastly, all the nonmonotone learning algorithms that are presented in the rest of the thesis apply the adaptive procedure of Eqs. (3.39)-(3.40), to dynamically determine the size of nonmonotone learning horizon.

Chapter 4

Adaptive Nonmonotone Resilient Propagation Algorithm

As presented later in this chapter, the Resilient Propagation (Rprop, also called Resilient backpropagation [165]-[168]) has been designed to tackle the drawbacks of the traditional BP method by exploiting information from the signs of the gradients of the cost function in the two successive iterations. In this chapter, we propose nonmonotone first-order methods [149] based on the Rprop algorithm. In particular we use the Jacobi-Rprop (JRprop) variant [11]- a recently proposed modification of the Rprop algorithm that combines the sign-based updates of the original Rprop with ideas from the composite nonlinear Jacobi method.

The rest of this chapter is organised as follows. The original Rprop method and the recent modifications [11][93] are reviewed in Section 4.1, while their global convergence is discussed in Section 4.2. The proposed nonmonotone Rprop methods are then described in Section 4.3, followed by experimental results in Section 4.4. Concluding remarks are made in Section 4.5.

4.1 Rprop Methods

Aiming to overcome the disadvantages of the pure gradient descent backpropagation procedure, the first version of *Resilient Propagation* algorithm was proposed in [165], which was based on the so-called "Manhattan Learning" rule, revised below

$$\Delta w_k = \begin{cases} -\Delta_k, & \text{if } g_k > 0 \\ +\Delta_k, & \text{if } g_k < 0, \\ 0, & \text{else} \end{cases} \tag{4.1}$$

where Δw_k is the update amount of weight vector w_k, Δ_k a predefined problem-dependent constant, and g_k the gradient. The second learning rule of the Rprop approach is as follows,

$$\Delta_k = \begin{cases} \Delta_{k-1} \bullet \eta^+, & \text{if } g_k^T \bullet g_{k-1} > 0 \\ \Delta_{k-1} \bullet \eta^-, & \text{if } g_k^T \bullet g_{k-1} < 0, \\ \Delta_{k-1}, & \text{else} \end{cases} \tag{4.2}$$

with $0 < \eta^- < 1 < \eta^+$. Table 4.1 shows the two original versions of the Rprop methods , where "*Rprop+*" represents the original Rprop with weight back-tracking [166] and "*Rprop-*" is the version without back-tracking [167][168]. As observed from Eqs. (4.1) and (4.2), the original intent of Rprop is to increase the steplength when the signs of two joined gradients remain the same, and to decrease the steplength when the signs of two were different.

The "*Improved Rprop*" (*iRprop*) methods were then proposed by Igel & Hüsken [93], as shown in Table 4.2. The revision of Anastasiadis et al. [9], i.e. the Jacobi-Rprop (JRprop) method, combined the sign-based updates of the original Rprop algorithm with the composite nonlinear Jacobi method, as shown in Table 4.3. Experiments in [9]

have shown that the JRprop algorithm is more efficient and superior to both the original Rprop and the iRprop algorithms. Thus, we focus on the JRprop approach to develop a nonmonotone variant of the Rprop method in this chapter.

Table 4.1 Key loops of original Rprop methods: (a) Rprop+ and (b) Rprop-

(a) Loop of Rprop+ [166]

for each w_k do{

if $g_k^T * g_{k-1} > 0$ then{

$\Delta_k = \min\left(\Delta_{k-1} \cdot \eta^+, \Delta_{\max}\right);$

$\Delta w_k = -\text{sign}\left(g_k\right) \cdot \Delta_k;$ }

elseif $g_k^T * g_{k-1} < 0$ then{

$\Delta_k = \max\left(\Delta_{k-1} \cdot \eta^-, \Delta_{\min}\right);$

$\Delta w_k = -\Delta w_{k-1};$

$g_k = 0;$ }

elseif $g_k^T * g_{k-1} = 0$ then{.

$\Delta_k = \Delta_{k-1};$

$\Delta w_k = -\text{sign}\left(g_k\right) \cdot \Delta_k;$ }

$w_{k+1} = w_k + \Delta w_k;$

}

(b) Loop of Rprop- [167][168]

for each w_k do{

if $g_k^T * g_{k-1} > 0$ then{

$\Delta_k = \min\left(\Delta_{k-1} \cdot \eta^+, \Delta_{\max}\right);$ }

elseif $g_k^T * g_{k-1} < 0$ then{

$\Delta_k = \max\left(\Delta_{k-1} \cdot \eta^-, \Delta_{\min}\right);$ }

$w_{k+1} = w_k - \text{sign}\left(g_k\right) \cdot \Delta_k;$

}

Table 4.2 Key loops of iRprop methods [93]: (a) iRprop+ and (b) iRprop-

(a) Loop of iRprop+

for each w_k do{

if $g_k^T * g_{k-1} > 0$ then {

$\Delta_k = \min\left(\Delta_{k-1} \cdot \eta^+, \Delta_{\max}\right);$

$\Delta w_k = -\text{sign}\left(g_k\right) \cdot \Delta_k;$ }

elseif $g_k^T * g_{k-1} < 0$ then{

$\Delta_k = \max\left(\Delta_{k-1} \cdot \eta^-, \Delta_{\min}\right);$

If $E^t > E^{t-1}$ then{

$\Delta w_k = -\Delta w_{k-1};$

$g_k = 0;$ } }

elseif $g_k^T * g_{k-1} = 0$ then{.

$\Delta w_k = -\text{sign}\left(g_k\right) \cdot \Delta_k;$ }

$w_{k+1} = w_k + \Delta w_k;$

}

(b) Loop of iRprop-

for each w_k do{

if $g_k^T * g_{k-1} > 0$ then{

$\Delta_k = \min\left(\Delta_{k-1} \cdot \eta^+, \Delta_{\max}\right);$

}

elseif $g_k^T * g_{k-1} < 0$ then{

$\Delta_k = \max\left(\Delta_{k-1} \cdot \eta^-, \Delta_{\min}\right);$

$g_k = 0;$

}

$w_{k+1} = w_k - \text{sign}\left(g_k\right) \cdot \Delta_k;$

}

Table 4.3 Key loop of the JRprop algorithm

Loop of JRprop [9]

if $E_k \le E_{k-1}$ then{

for each w_k do{

if $g_{k-1}^T \cdot g_k > 0$ then{

$\Delta_k = \min\left(\Delta_{k-1} \cdot \eta^+, \Delta_{\max}\right);$ $\Delta w_k = -\text{sign}\left(g_k\right) \cdot \Delta_k;$

$w_{k+1} = w_k + \Delta w_k;$ $g_{k-1} = g_k;$

$\Delta w_{k-1} = \Delta w_k;$ }

elseif $g_{k-1}^T \cdot g_k < 0$ then{

$\Delta_k = \max\left(\Delta_{k-1} \cdot \eta^-, \Delta_{\min}\right);$ $g_{k-1} = 0;$ }

elseif $g_{k-1}^T \cdot g_k = 0$ then{.

$\Delta w_k = -\text{sign}\left(g_k\right) \cdot \Delta_k;$ $w_{k+1} = w_k + \Delta w_k;$

$g_{k-1} = g_k;$ }

$q = 1;$

} }

elseif $E_k > E_{k-1}$ then{

$w_{k+1} = w_k + \frac{1}{2^q} \Delta w_{k-1};$

$q = q + 1;$

}

4.2 Global Convergence of Rprop Methods

In this section, global convergence of the Rprop class is discussed. Before stating the resulted theorem of global convergence proved in [8], the assumptions (**H4.1**) and (**H4.2**) should be made.

(**H4.1**) For a given point $w_0 \in R^n$ and for every w in some region that contains the initial weight vector w_0, the level set $L_0 = \{w \in R^n \mid f(w) \le f(w_0)\}$ is bounded.

(**H4.2**) In some neighbourhood B of the level set L_0, the error function is continuous differentiable and the gradient g is Lipschitz continuous, i.e. there exists $L > 0$ for every pair w, v such that

$$\|g(w) - g(v)\| \le L\|w - v\|, \forall w, v \in B \tag{4.3}$$

□ **Theorem 4.1** [8]. *Suppose that the assumption (**H4.1**) and (**H4.2**) hold. For any* $w_0 \in R^n$ *and* $\forall k \ge 0$, *any sequence* $\{w_k\}_{k=0}^{\infty}$ *generated by the Rprop scheme*

$$w_{k+1} = w_k - \tau_k \cdot \operatorname{diag}\{\eta_k^1, \ldots, \eta_k^i, \ldots, \eta_k^n\} \cdot \operatorname{sign}(g_k), \tag{4.4}$$

where $\tau_k > 0$ *satisfying Wolfe's conditions,* η_k^m $(m = 1, 2, \ldots, i-1, i+1, \ldots, n)$ *are small positive real numbers generated by Rprop's learning rates procedure and*

$$\eta_k^i = -\frac{\sum_{j=1, j\ne i}^{n} \eta_k^j g_k^j + \delta}{g_k^i}, \tag{4.5}$$

with $0 < \delta \ll \infty$, $g_k^i \ne 0$, *it holds that*

$$\lim_{k\to\infty} g_k = 0. \tag{4.6}$$

As we focus on the JRprop method [9] in this chapter, the revision of globally-convergent Rprop approaches shall be implemented in our future works.

4.3 The Nonmonotone Jacobi Rprop Algorithm

The proposed nonmonotone version of JRprop method is presented in this section. Followed the same way as other versions of Rprop, the key loop of our proposed algorithm, i.e. Adaptive Non-Monotone JRprop (ANM-JRprop), is presented in Table 4.4. ANM-JRprop applies one-step subminimisation by employing an Rprop-based heuristic scheme to locate an approximation of the subminimiser along each weight direction as the original JRprop does. The main difference between the original JRprop [9] and our ANM-JRprop is that the subminimisation procedure of the composite nonlinear Jacobi method operates in a nonmonotone way. When the current training error E_k is not larger than the error of previous training epoch E_{k-1}, the weights (and biases) are updated according to the JRprop's rules; otherwise, the subminimisation is performed under the predefined adaptive nonmonotone condition.

Table 4.4 Key loop of the Adaptive Non-Monotone JRprop algorithm

Loop of ANM-JRprop

If $E_k \le E_{k-1}$ then{

update w_{k+1} by Rprop;

set $q = 1$; }

Else {

if $E_k > \max\limits_{0 \le j \le M_k} \left(E_{k-j} \right) + \delta \alpha_k g_k d_k$ then {

$$w_{k+1} = w_k + \frac{1}{2^q} \Delta w_{k-1};$$

$q = q + 1$; } }

Examples of convergence behaviours for the parity-5 (P5) and the parity-10 (P10) problems are shown in Figures 4.1 and 4.2, respectively, illustrating the evolution of the MSE during training NARX networks, the stepsizes, and the adaptive nonmonotone learning horizon, where the maximum number of hidden nodes from the ones listed in Table A.1 was used, i.e., 7 for the P5 and 10 for the P10.

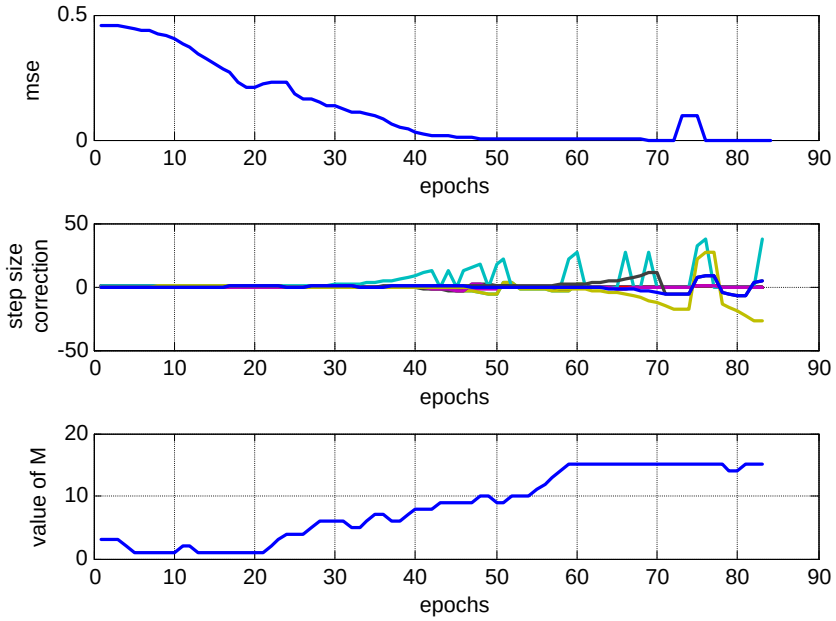


Figure 4.1 Convergence behaviours in P5: NARX networks trained by ANM-JRprop

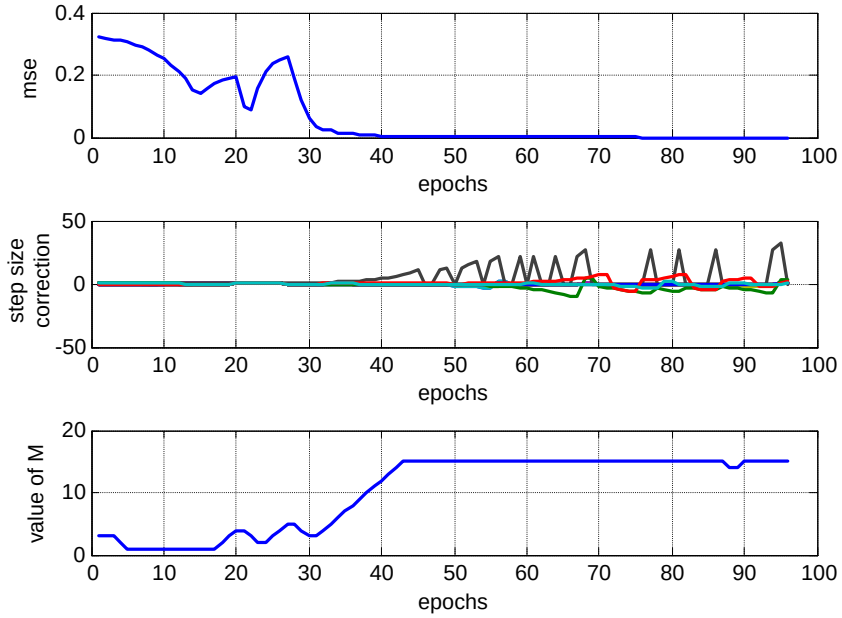


Figure 4.2 Convergence behaviours in P10: NARX networks trained by ANM-JRprop

4.4 Experimental Results

Four problems are tested in this chapter: the parity-N, Sequence Classification (SC, [116]), Sequence Learning (SL, [124]) and Reading Aloud (RA, [158]) problems. More details on these problems, such as description, information about the datasets, numbers of weights and biases, settings of training parameters, and stopping criteria, are provided in Appendix A.1.

In the following subsections, simulation results of these four applications for 3 different RNNs architectures, i.e. FFTD ([196][197]), LRN ([57][85]), and NARX ([128][137]), are presented and examples of learning behaviours are provided. Mathematical models of these RNNs architectures can be found in Chapter 2. All numerical results are averaged from 100 random initialized runs, while in the

following Tables *#hid* is the number of hidden nodes used, *Conv* shows the percentage of runs that reach the predefined training goal, *MSE* and *STD* are respectively the mean-squared-error and corresponding standard deviation in percentage. In addition, the columns under *Epoch* represent the average (*Ave*), minimum (*Min*), maximum (*Max*) and standard deviation (*Std*) of the training epochs for the converged runs, while the definitions of convergence for the four simulated applications are provided in Appendix A.1. A dash (-) indicates no single run converged under the specified termination conditions, and this results to placing a 0 in the *Conv* column.

4.4.1 The N-Bit Parity Problems

Being a typical application for testing new algorithms, the parity-N problem plays an important role for testing the ability of how an algorithm handles nonlinear separable datasets with lots of local minima. Appendix A.1.1 provides for further details on the particular problem, including latest references applying this application, such as [41][84][110][125][136][158][183]. We choose two instances, i.e. parity-5 (P5) and parity-10 (P10) for our experiments.

Tables 4.5-4.7 and Figures 4.3-4.5 show the simulation results and learning behaviours examples of the P5 problems, while Tables 4.8-4.10 and Figures 4.6-4.8 for the P10 problems. From these numerical results, it can be easily observed that for the P5 problem, the ANM-JRprop is better than the JRprop in terms of MSE with improvements ranging between 26% and 45% for FFTD networks and LRNs, where by applying 5 and 7 hidden nodes, our approach can reach the training goal (MSE=0.01): 11 and 37 runs for FFTD, 12 and 37 runs for LRN converged. Although for NARX networks, the MSEs of our approach are not all better than JRprop's, i.e.,

only 0.01% and 0.007% higher for 1 and 7 hidden nodes, respectively, our approach does converge for all the random 100 runs and achieves much smaller number of converged epochs in terms of average, minimum, maximum and standard deviation. For example, as shown in Table 4.7, when using 2 hidden nodes for NARX networks, JRprop has 99 runs converged to $\text{MSE} \le 0.01$, while the averaged epoch is 84, maximum epoch of converged runs is 401, and standard deviation of epochs for all 99 converged runs is 70. Our approach, ANM-JRprop, has 100% convergence rate, with 61.9% smaller average, 87.7% smaller maximum, and much more robust standard deviation (91.4%) of converged epochs.

Note that, in Figures 4.3-4.5, the behaviours of the JRprop method are caused by the condition of $E_k > E_{k-1}$, where, according to the original description of the JRprop algorithm (see Table 4.3), the updates of weights and biases are given by $w_{k+1} = w_k + \frac{1}{2^q}\Delta w_{k-1}$, and $q = q + 1$. Under this situation the JRprop method will consider only the error function of the previous epoch, i.e. E_{k-1}, and as a result, the JRprop method in Figures 4.3-4.5 then behaves in a pseudo-nonmonotone way, just as the ones in Figures 4.6-4.8.

Table 4.5 Average performance for FFTD networks in the P5 problem: class of JRprop

Algorithms	#hid	Conv (%)	MSE (%)	STD (%)	Epoch			
					Ave	Min	Max	Std
JRprop	1	0	49.393	0.573	-	-	-	-
	2	0	49.305	0.601	-	-	-	-
	5	0	48.251	0.495	-	-	-	-
	7	0	47.466	0.454	-	-	-	-
ANM-JRprop	**1**	**0**	**23.668**	2.581	-	-	-	-
	2	**0**	**18.137**	6.047	-	-	-	-
	5	**11**	**6.007**	4.352	**997**	**356**	**1696**	**533**
	7	**37**	**3.288**	5.618	**708**	**227**	**1766**	**431**

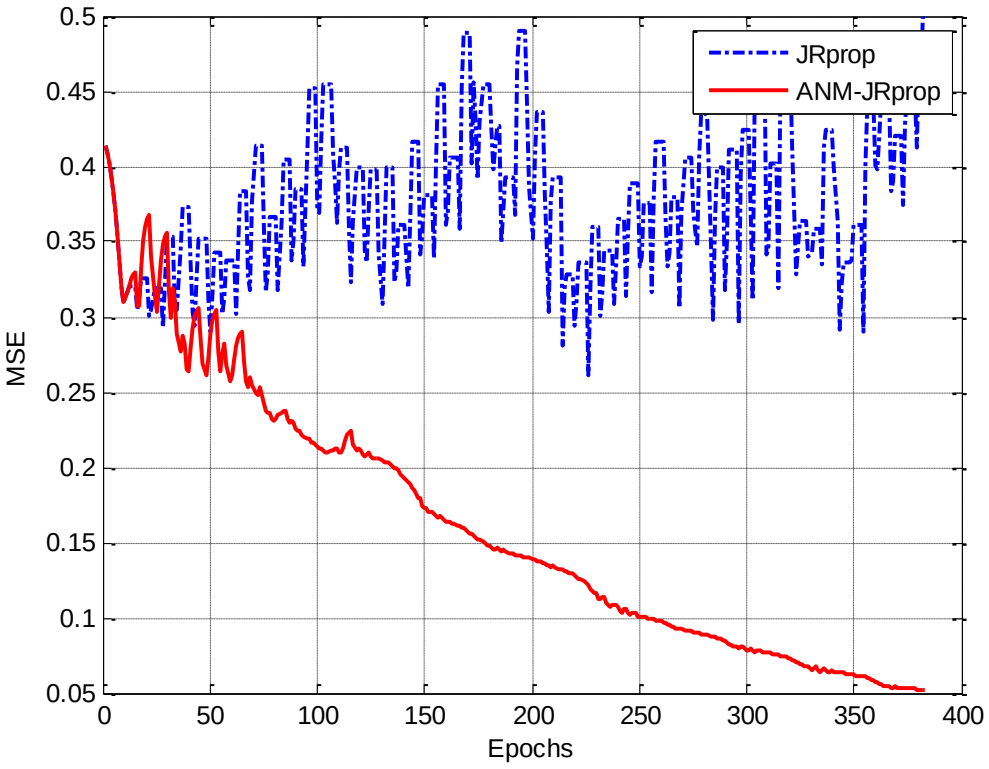


Figure 4.3 Examples of learning behaviours (JRprop vs. ANM-JRprop): P5 problem, FFTD network

Table 4.6 Average performance for LRN networks in the P5 problem: class of JRprop.

Algorithms	#hid	Conv (%)	MSE (%)	STD (%)	Epoch			
					Ave	Min	Max	Std
JRprop	1	0	49.154	0.499	-	-	-	-
	2	0	49.124	0.516	-	-	-	-
	5	0	48.251	0.478	-	-	-	-
	7	0	48.020	0.433	-	-	-	-
ANM-JRprop	**1**	**0**	**23.326**	2.150	-	-	-	-
	2	**0**	**17.698**	4.938	-	-	-	-
	5	**12**	**6.521**	4.503	**747**	**362**	**1793**	**418**
	7	**37**	**3.387**	4.990	**758**	**234**	**1852**	**406**

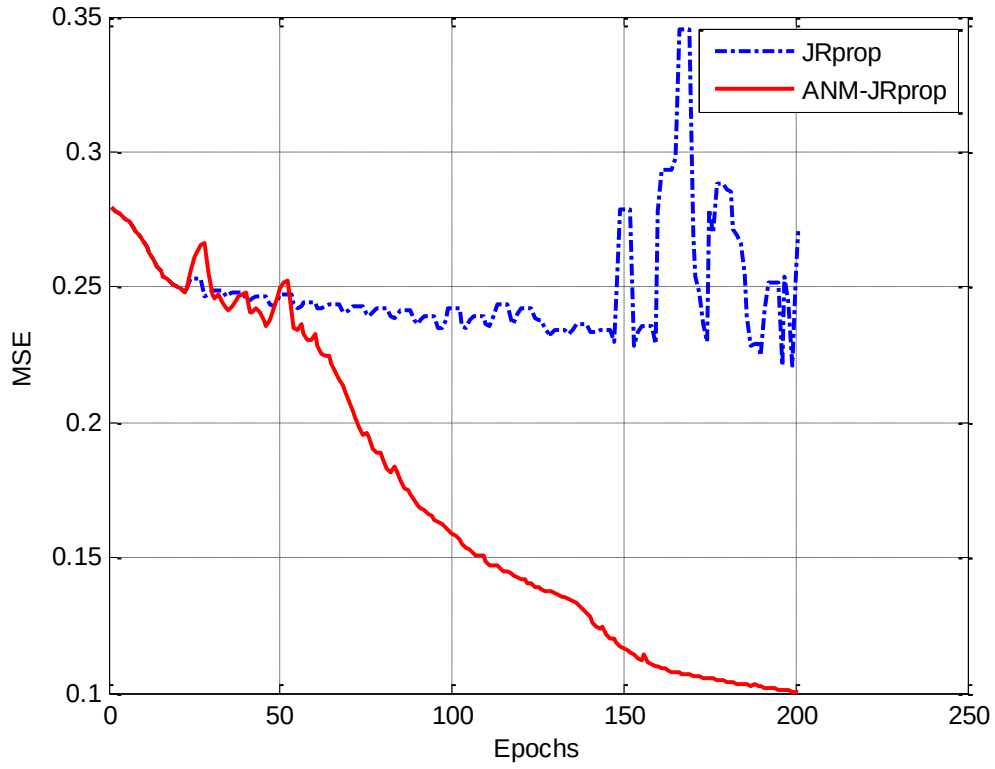


Figure 4.4 Examples of learning behaviours (JRprop vs. ANM-JRprop): P5 problem, LRN network

Table 4.7 Average performance for NARX networks in the P5 problem: class of JRprop.

Algorithms	#hid	Conv (%)	MSE (%)	STD (%)	Epoch			
					Ave	Min	Max	Std
JRprop	1	100	0.791	0.163	99	23	643	88
	2	99	1.223	0.406	84	23	401	70
	5	100	0.693	0.288	55	18	182	47
	7	100	0.691	0.282	44	15	227	36
ANM-JRprop	**1**	**100**	0.818	0.140	**37**	**20**	**64**	**9**
	2	**100**	**0.737**	0.257	**32**	**23**	**49**	**6**
	5	**100**	**0.692**	0.304	**26**	**18**	**46**	**5**
	7	**100**	0.698	0.133	**25**	**15**	**40**	**5**

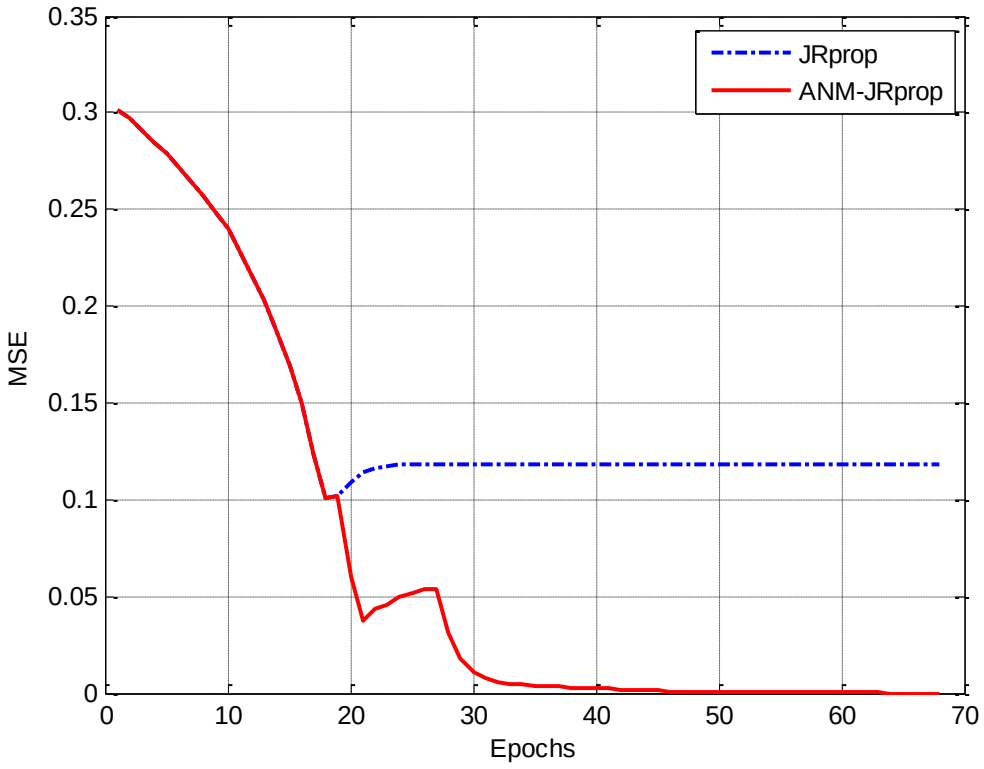


Figure 4.5 Examples of learning behaviours (JRprop vs. ANM-JRprop): P5 problem, NARX network

Comparing to P5, the P10 problem is more difficult, and convergence rates of FFTD and LRN are therefore relatively small. Similar to the improved range of the P5 problem, the differences in terms of MSEs made by our approach are between 25% and 45%. In this problem LRNs training requires extensive computational resources, so we have left simulations with 2, 5, and 7 hidden nodes for our future works.

For NARX networks, as in the case of the P5, the MSEs of the ANM-JRprop are slightly higher than JRprop, where the largest difference is 0.05%, but again our approach does achieve very promising results in the converged runs, in terms of much fewer training epochs, as shown in Table 4.10. For example, applying 10 hidden nodes for NARX networks, both JRprop and ANM-JRprop are 100% converged. But when our approach takes only 27 epochs in average to converge, JRprop needs 104 training epochs, which is about 4 times higher. Furthermore, we have the same best 20-epoch minimum number in the converged run as JRprop, and the maximum number of epochs within the 100 random runs drops from 284 for JRprop to 38 for our ANM-JRprop. This is an improvement of 86.6% in the most difficult case (maybe caused by poor initial weights) for the simulated runs.

As revealed in the P5 problem, our approach has more concrete behaviours as the values of standard deviation are relative smaller than JRprop, and the ratios between standard deviation (*Std*) and average epochs (*Ave*) of our approach are about 11.1% to 22.2%, while the same ratios of JRprop are from 54.8% to 85.0%. Behaviour of the JRprop algorithm in Figure 4.6 can be explained in the same way as in the P5 problem.

Table 4.8 Average performance for FFTD networks in the P10 problem: class of JRprop

Algorithms	#hid	Conv (%)	MSE (%)	STD (%)	Epoch			
					Ave	Min	Max	Std
JRprop	1	0	49.992	0.606	-	-	-	-
	2	0	49.972	0.727	-	-	-	-
	5	0	49.681	0.594	-	-	-	-
	7	0	49.835	0.549	-	-	-	-
	10	0	49.665	0.511	-	-	-	-
ANM-JRprop	**1**	**0**	**24.983**	2.731	-	-	-	-
	2	**0**	**23.660**	3.127	-	-	-	-
	5	**0**	**15.568**	5.535	-	-	-	-
	7	**0**	**9.495**	4.608	-	-	-	-
	10	**2**	**4.554**	6.710	**3696**	**3527**	**3864**	**238**

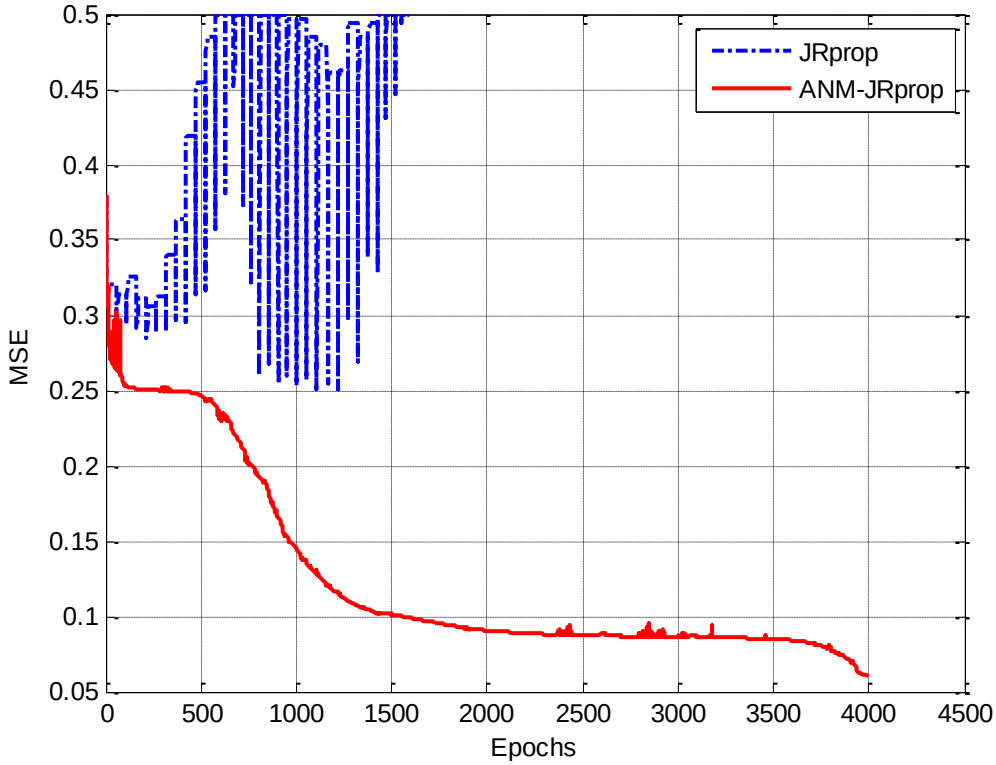


Figure 4.6 Examples of learning behaviours (JRprop vs. ANM-JRprop): P10 problem, FFTD network

Table 4.9 Average performance for LRN networks in the P10 problem: class of JRprop

Algorithms	#hid	Conv (%)	MSE (%)	STD (%)	Epoch			
					Ave	Min	Max	Std
JRprop	1	0	49.989	0.473	-	-	-	-
	10	0	49.674	0.594	-	-	-	-
ANM-JRprop	**1**	**0**	**24.971**	3.842	-	-	-	-
	10	**0**	**4.724**	5.107	-	-	-	-

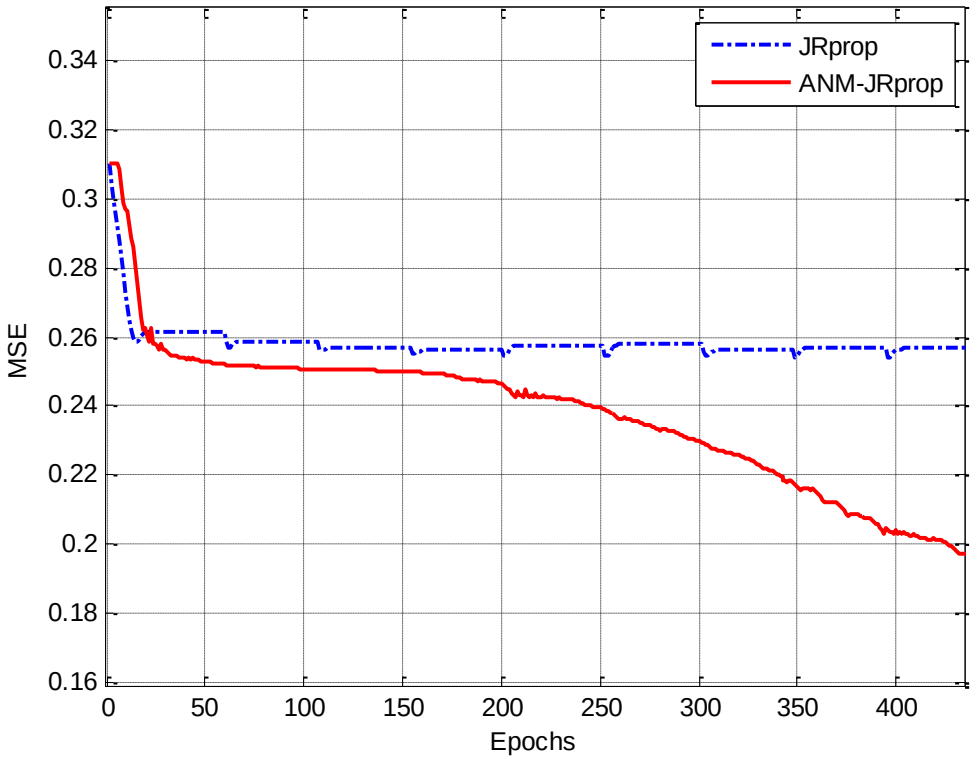


Figure 4.7 Examples of learning behaviours (JRprop vs. ANM-JRprop): P10 problem, LRN network

As illustrated in Figure 4.3, both methods behave exactly the same within the first couple training epochs, since the same weight-updating condition (the original Rprop) was invoked; Figures 4.5 and 4.8 provide more local examples of NARX networks for this type of situations. After that point ANM-JRprop has a generally decreasing trend and converges at 382 epochs (i.e. Figure 4.3), but JRprop seems to be trapped in a local state by considering only the error of the previous epoch. Similar situations can be observed in Figures 4.4 and 4.6.

One more point is worth mentioning is that shown in Figure 4.6. Starting from the same initialisation point, while JRprop has stopped its training at about 1500 epochs because the gradient was smaller than the specified threshold or equal to zero, i.e. the minimum gradient is set to 1e-100 in all simulations of this thesis, our approach ANM-JRprop can be trained till the predefined 4000-epoch. Lastly, Figure 4.7 shows that, at the beginning of training, ANM-JRprop behaves differently to JRprop, and when JRprop is trapped to a local minimum with MSE of about 0.26, our approach exploits the power of non-monotonicity to decrease the MSE effectively.

Table 4.10 Average performance for NARX networks in the P10 problem: class of JRprop

Algorithms	#hid	Conv (%)	MSE (%)	STD (%)	Epoch			
					Ave	Min	Max	Std
JRprop	1	100	0.815	0.148	164	27	402	90
	2	99	1.174	0.394	120	22	619	102
	5	100	0.645	0.273	89	23	278	60
	7	100	0.699	0.282	95	21	287	68
	10	100	0.669	0.195	104	20	284	61
ANM-JRprop	**1**	**100**	0.816	**0.143**	**36**	**25**	**92**	**8**
	2	**100**	**0.715**	**0.214**	**32**	**22**	**51**	**6**
	5	**100**	0.667	**0.258**	**28**	**22**	**44**	**4**
	7	**100**	0.720	**0.221**	**27**	**21**	**36**	**3**
	10	**100**	0.707	**0.147**	**27**	**20**	**38**	**4**

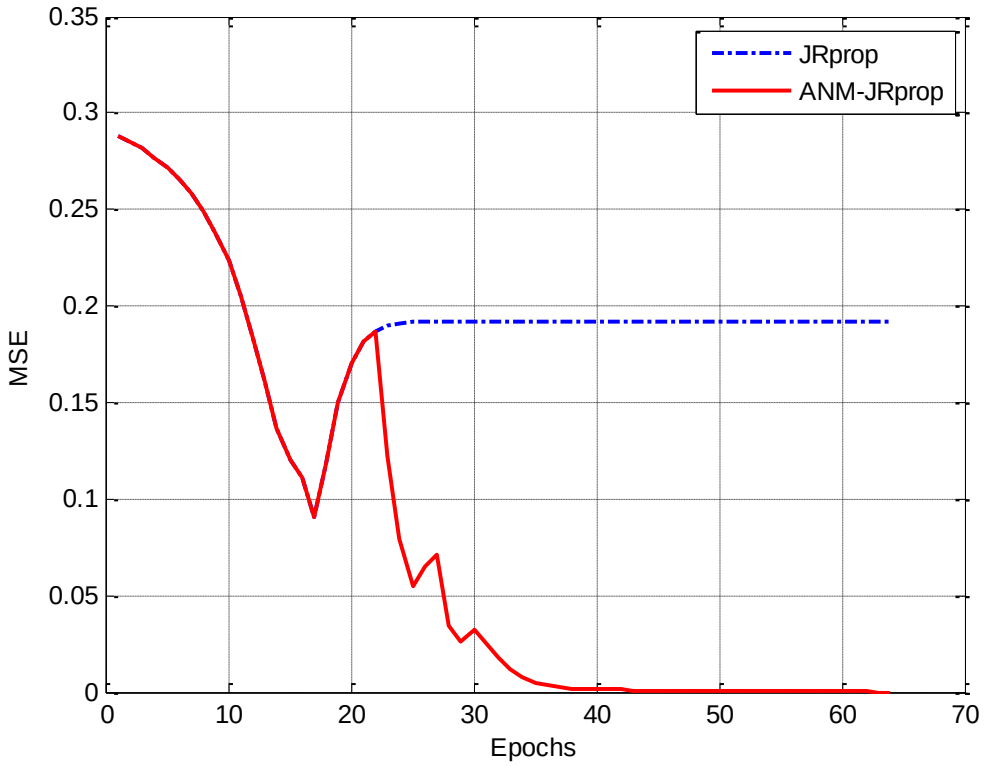


Figure 4.8 Examples of learning behaviours (JRprop vs. ANM-JRprop): P10 problem, NARX network

4.4.2 The Sequence Classification Problem

Simulation results in terms of MSEs and CEs (Classification Errors) for both training and testing datasets of the SC problem are shown in Tables 4.11-4.13, while examples of learning behaviours for the FFTD, LRN and NARX networks are provided in Figures 4.9-4.11. More details on the experimental parameters and the recurrent architectures used in this application are listed in Appendix A.1.2.

Although the largest improvements are made by FFTD networks, i.e. 10.0% (MSE) and 10.7% (CE) for the training dataset, and 30.8% (MSE) and 24.2% (CE) for the testing dataset, the average improvements of our ANM-JRprop are 3.6% (MSE) and 3.7% (CE) for LRN, and, 5.8% (MSE) and 0.8% (CE) for NARX on the training set, while for the testing set performances of ANM-JRprop are 10.4% (MSE) and 2.8% (CE) superior to JRprop for LRN, and, 16.8% (MSE) and 2.6% (CE) better than JRprop for NARX.

Figure 4.9 displays a typical example of why nonmonotone JRprop provides large improvement for FFTD networks. As this figure shows JRprop is generally trapped at local minima with values around an MSE=0.425, while our approach ANM-JRprop exhibits the benefits of nonmonotone learning and achieves improvements which are larger than a half of JRprop's training MSE.

Learning behaviours shown in Figures 4.10 and 4.11 illustrate that since JRprop considers the MSE of previous training epoch only, both RNN architectures suffer from improper weight updates and the MSEs jump to high level at the end of training, while the nonmonotone horizon and adaptive tuning of our ANM-JRprop efficiently prevent this kind of situations.

Table 4.11 Average performance for FFTD networks in the SC problem: class of JRprop

Algorithm	#hid	MSE (%)		CE (%)	
		Train/STD	Test/STD	Train/STD	Test/STD
JRprop	5	29.168/2.693	27.698/2.185	78.828/5.732	55.685/5.003
	10	28.258/2.940	26.451/2.774	82.552/6.033	55.795/5.146
	15	28.592/2.822	27.389/2.901	78.690/5.591	51.068/4.790
ANM-JRprop	**5**	**19.146**/4.373	**16.918**/3.044	**51.724**/6.158	**31.507**/5.998
	10	**19.145**/4.373	**16.917**/3.044	**51.724**/6.158	**31.507**/5.998
	15	**19.145**/4.372	**16.917**/3.043	**51.724**/6.158	**31.507**/5.998

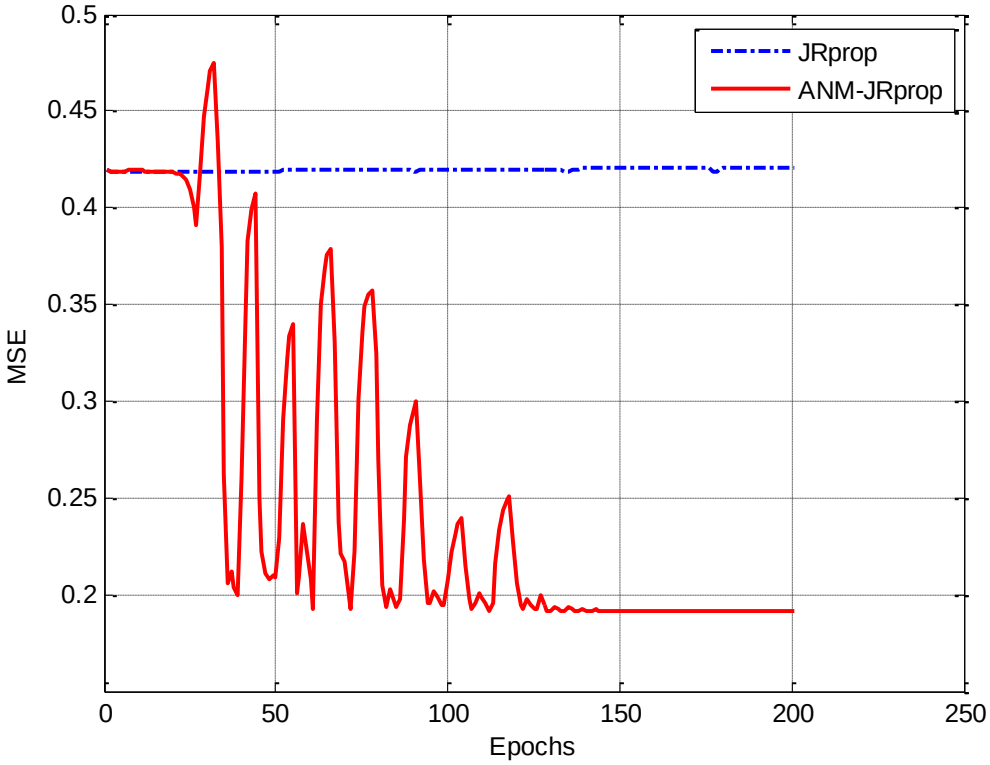


Figure 4.9 Examples of learning behaviours (JRprop vs. ANM-JRprop): SC problem, FFTD network

Table 4.12 Average performance for LRN networks in the SC problem: class of JRprop

Algorithm	#hid	MSE (%)		CE (%)	
		Train/STD	Test/STD	Train/STD	Test/STD
JRprop	5	12.672/2.143	11.828/1.906	42.714/3.729	14.973/2.484
	10	10.058/1.059	8.901/1.663	34.645/2.957	10.178/2.093
	15	10.039/1.007	9.214/1.815	36.330/3.631	8.232/1.732
ANM-JRprop	**5**	**7.013**/3.870	**6.067**/2.713	**26.576**/3.992	**7.657**/2.036
	10	**7.434**/4.073	**6.450**/3.588	**27.887**/4.271	**8.917**/5.673
	15	**7.413**/4.125	**6.249**/3.402	**27.828**/4.200	8.328/5.227

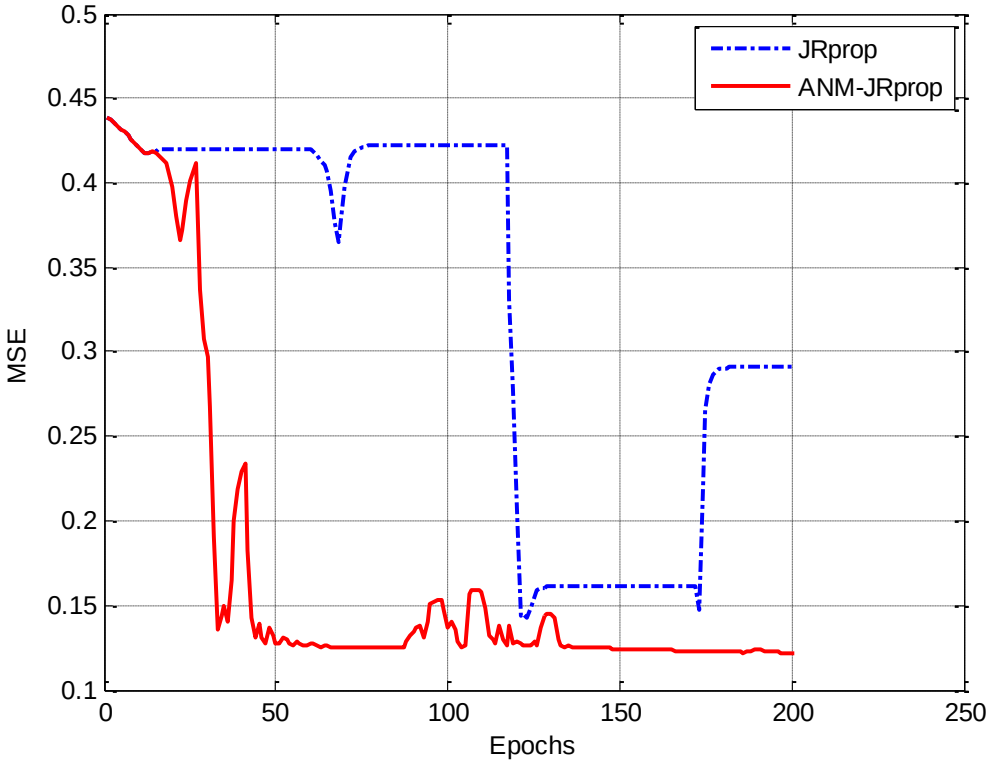


Figure 4.10 Examples of learning behaviours (JRprop vs. ANM-JRprop): SC problem, LRN network

Table 4.13 Average performance for NARX networks in the SC problem: class of JRprop

Algorithm	#hid	MSE (%)		CE (%)	
		Train/STD	Test/STD	Train/STD	Test/STD
JRprop	5	10.701/2.479	19.593/4.263	36.330/3.498	30.712/5.125
	10	7.661/1.952	17.314/5.217	26.813/3.120	27.315/6.012
	15	7.397/1.847	17.576/4.970	26.803/3.754	27.740/5.991
ANM-JRprop	**5**	**2.842/1.423**	**17.091/2.578**	**13.394/2.762**	**26.192/4.903**
	10	**2.633/1.392**	17.326/**2.720**	**13.034/2.693**	**25.781/4.888**
	15	**2.605/1.350**	**17.410/2.954**	**13.069/2.813**	**25.767/4.821**

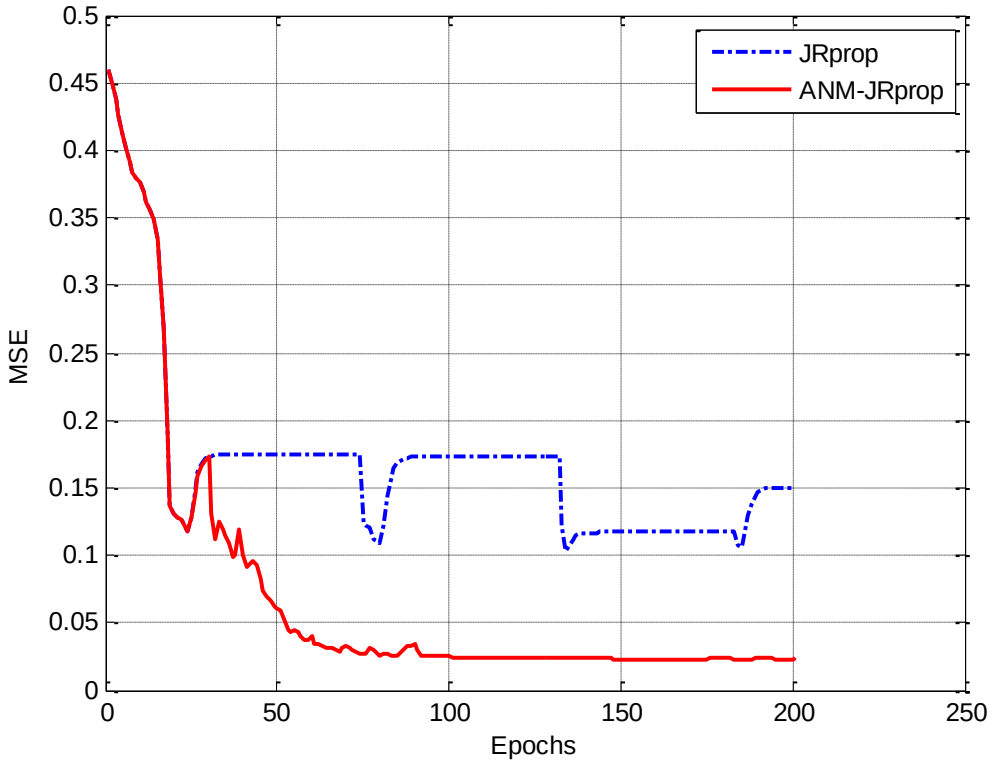


Figure 4.11 Examples of learning behaviours (JRprop vs. ANM-JRprop): SC problem, NARX network

4.4.3 The Sequence Learning Problem

Tables 4.14-4.16 present simulated results of this application using 1, 2, 5 and 10 hidden nodes for the three RNN architectures. For both training and testing datasets our approach ANM-JRprop performs better than JRprop in terms of MSE, while, on average, the testing improvements are 3.2% for FFTD networks, 2.0% for LRNs, and 1.3% for NARX networks.

Figures 4.12-4.14 show examples of learning comparing JRprop and ANM-JRprop. As indicated in previous subsection nonmonotone learning scheme does help our approach behave better, and in general, exhibit a more robust training process than JRprop method.

Table 4.14 Average performance for FFTD networks in the SL problem: class of JRprop

Algorithm	#hid	Training		Testing	
		MSE (%)	STD (%)	MSE (%)	STD (%)
JRprop	1	43.199	22.764	43.179	23.602
	2	32.136	15.817	32.131	16.688
	5	28.356	12.263	28.327	14.271
	10	26.806	14.085	26.740	13.995
ANM-JRprop	**1**	**39.481**	**18.356**	**39.424**	**18.200**
	2	**28.960**	**13.784**	**28.897**	**13.607**
	5	**25.034**	**10.349**	**24.986**	**10.331**
	10	**24.239**	**10.060**	**24.268**	**10.243**

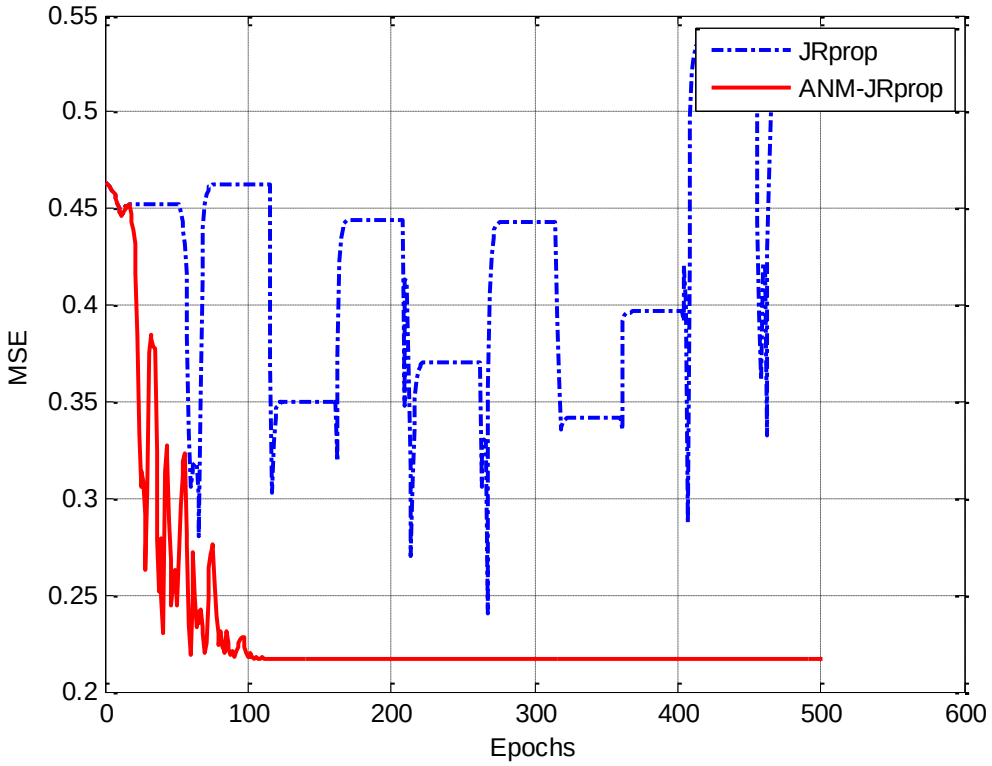


Figure 4.12 Examples of learning behaviours (JRprop vs. ANM-JRprop): SL problem, FFTD network

Table 4.15 Average performance for LRN networks in the SL problem: class of JRprop

Algorithm	#hid	Training		Testing	
		MSE (%)	STD (%)	MSE (%)	STD (%)
JRprop	1	36.630	13.792	37.449	16.074
	2	27.200	11.503	28.291	11.990
	5	21.359	9.005	23.065	10.784
	10	20.388	8.376	21.682	9.132
ANM-JRprop	**1**	**35.487**	**10.324**	**36.379**	**11.021**
	2	**26.308**	**11.097**	**27.428**	12.191
	5	**18.880**	**8.412**	**20.804**	**9.778**
	10	**15.521**	**6.309**	**17.923**	**6.571**

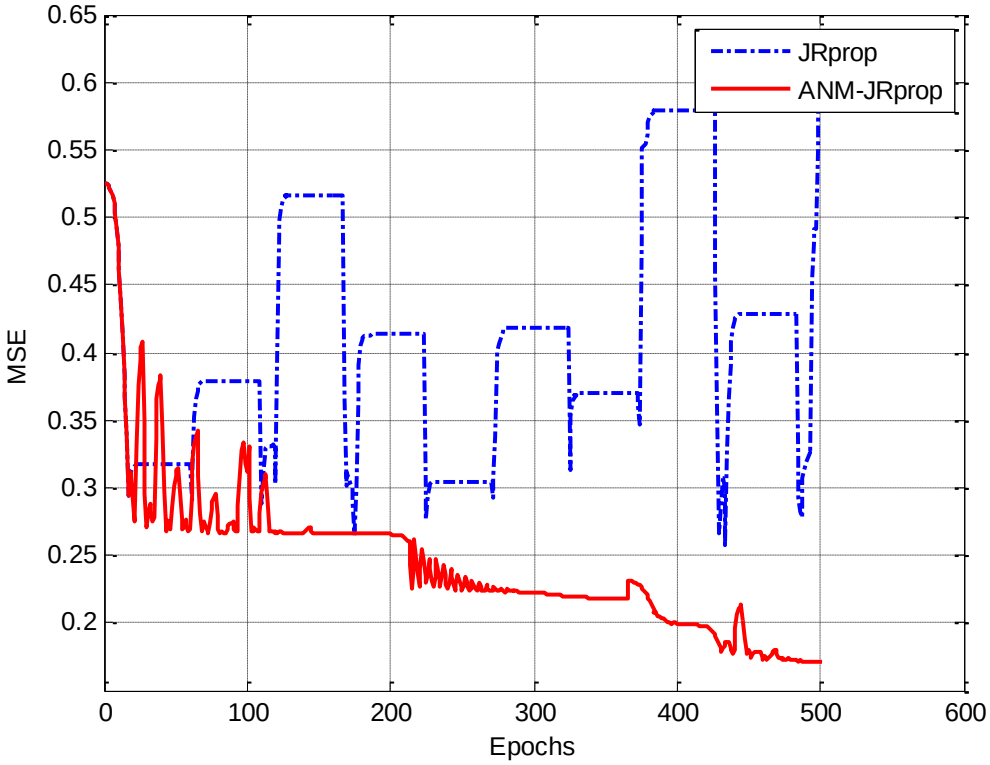


Figure 4.13 Examples of learning behaviours (JRprop vs. ANM-JRprop): SC problem, LRN network

Table 4.16 Average performance for NARX networks in the SL problem: class of JRprop

Algorithm	#hid	Training		Testing	
		MSE (%)	STD (%)	MSE (%)	STD (%)
JRprop	1	35.077	11.792	36.660	12.046
	2	26.814	9.984	28.495	11.737
	5	18.274	10.003	21.897	12.854
	10	14.030	8.314	17.839	11.402
ANM-JRprop	**1**	**33.932**	**9.458**	**35.493**	**10.880**
	2	**25.431**	**10.671**	**27.345**	**11.724**
	5	**16.809**	**5.477**	**20.733**	**7.999**
	10	**12.027**	**4.890**	**16.160**	**8.005**

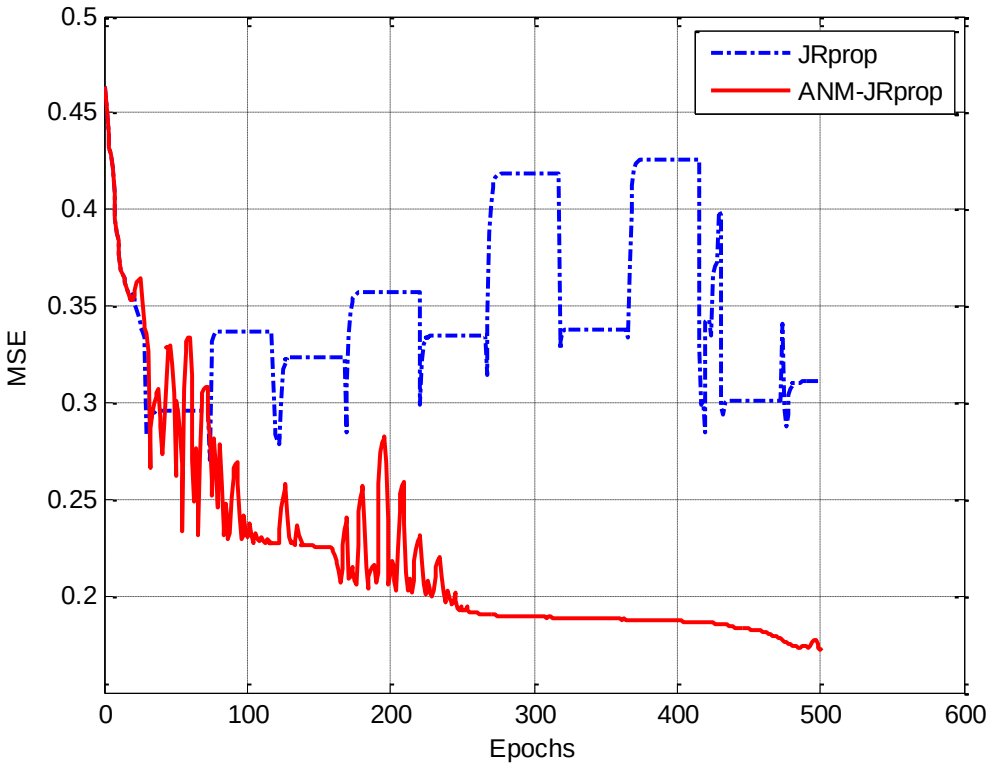


Figure 4.14 Examples of learning behaviours (JRprop vs. ANM-JRprop): SL problem, NARX network

Since these experiments in the thesis were designed with the aim to verify how effective and efficient the proposed nonmonotone algorithms could be, the training processes are extended, i.e., from the 23-epoch simulation of the original work for this problem, [124], to additional 200- and 1000-epoch runs in order to make a more extensive comparison of the two methods. Results are shown in Tables 4.17-4.19. Note that, in the additional simulated runs, the same behaviours (as it has been indicated on page 53 for Figures 4.3-4.5) of the JRprop method appeared again and this caused poor performance for this method for the three RNN architectures.

Table 4.17 Results of additional simulations for FFTD networks in the SL problem: class of JRprop

Algorithms	200-epoch		1000-epoch	
	Train (MSE/CE)	Test (MSE/CE)	Train (MSE/CE)	Test (MSE/CE)
JRprop	*44.6/97.8*	*47.3/99.1*	*29.9/91.2*	*31.0/93.5*
ANM-JRprop	*21.2/74.6*	*22.3/75.5*	*18.4/47.5*	*21.7/73.4*

Table 4.18 Results of additional simulations for LRN networks in the SL problem: class of JRprop

Algorithms	200-epoch		1000-epoch	
	Train (MSE/CE)	Test (MSE/CE)	Train (MSE/CE)	Test (MSE/CE)
JRprop	*33.5/95.1*	*33.1/94.3*	*29.8/90.8*	*31.2/93.2*
ANM-JRprop	*13.3/31.3*	*14.9/38.8*	*11.4/28.6*	*13.2/30.7*

Table 4.19 Results of additional simulations for NARX networks in the SL problem: class of JRprop

Algorithms	200-epoch		1000-epoch	
	Train (MSE/CE)	Test (MSE/CE)	Train (MSE/CE)	Test (MSE/CE)
JRprop	*22.5/78.4*	*24.1/82.3*	*17.6/43.4*	*17.3/42.1*
ANM-JRprop	*12.3/29.2*	*14.0/34.5*	*5.9/17.5*	*8.7/20.8*

4.4.4 The Reading Aloud Problem

This high-dimensional problem consists of 2998 training patterns which require RNNs with 105 input and 61 output nodes. The description of this application, numbers of weights for our simulations, and other experimental parameters are stated in Appendix A1.4. Simulation results are shown in Tables 4.20 and 4.21, while examples of learning behaviours are provided in Figures 4.15 and 4.16.

The training MSE improvements of the proposed ANM-JRprop using 5 and 10 hidden nodes are about 4.9% and 10.4% for FFTD networks, and 8.9% and 10.8% for NARX networks, respectively. For the exclusive 30-word testing dataset, ANM-JRprop is 8.7% and 3.8% better for FFTD, and 6.6% and 4.8% for NARX, in terms of testing MSE.

Figure 4.15 shows an example that when JRprop fails to train this application successfully, our ANM-JRprop behaves relatively better and converges to a smaller MSE. On the other hand, as shown in Figure 4.16, when JRprop is trapped to a local solution with an MSE value of about 0.05 our approach trains the NARX network better, demonstrating a general trend of descending errors.

Table 4.20 Average performance for FFTD networks in the RA problem: class of JRprop

Algorithm	#hid	Training		Testing	
		MSE (%)	STD (%)	MSE (%)	STD (%)
JRprop	5	15.765	6.785	22.540	13.335
	10	10.225	4.301	17.508	8.996
ANM-JRprop	**5**	**5.359**	**2.713**	**13.844**	**7.088**
	10	**5.309**	**2.005**	**13.795**	**6.792**

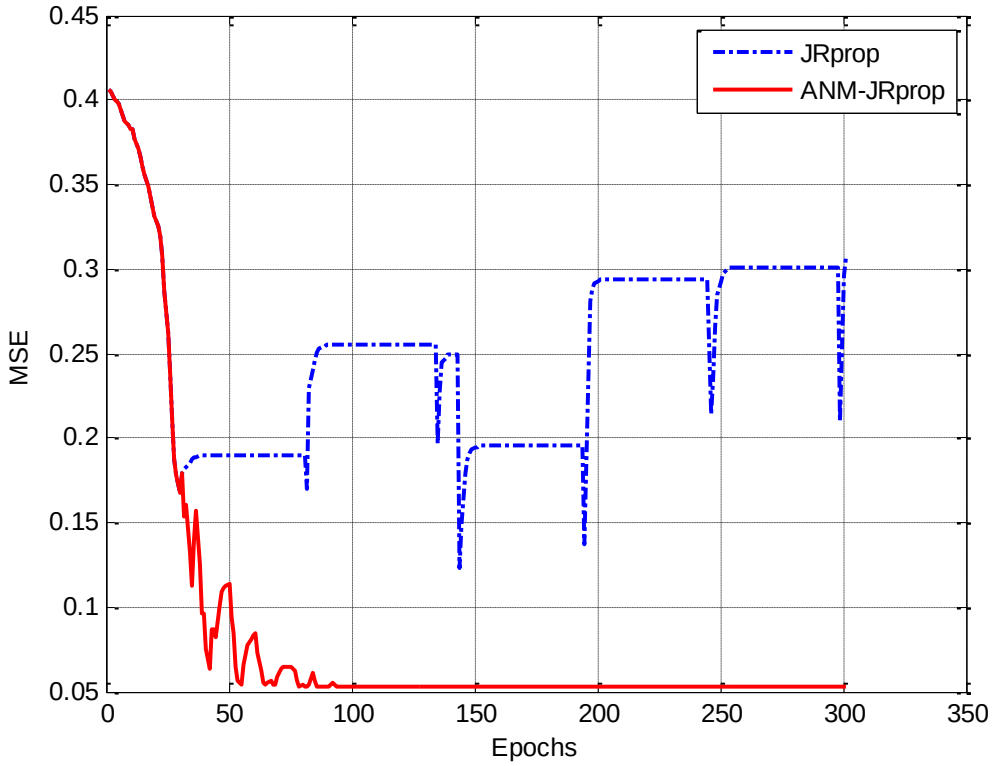


Figure 4.15 Examples of learning behaviours (JRprop vs. ANM-JRprop): RA problem, FFTD network

Table 4.21 Average performance for NARX networks in the RA problem: class of JRprop

Algorithm	#hid	Training		Testing	
		MSE (%)	STD (%)	MSE (%)	STD (%)
JRprop	5	12.608	4.728	24.215	11.288
	10	13.843	5.925	22.800	9.264
ANM-JRprop	**5**	**3.750**	**1.847**	**17.663**	**6.340**
	10	**3.009**	**2.296**	**18.055**	**7.111**

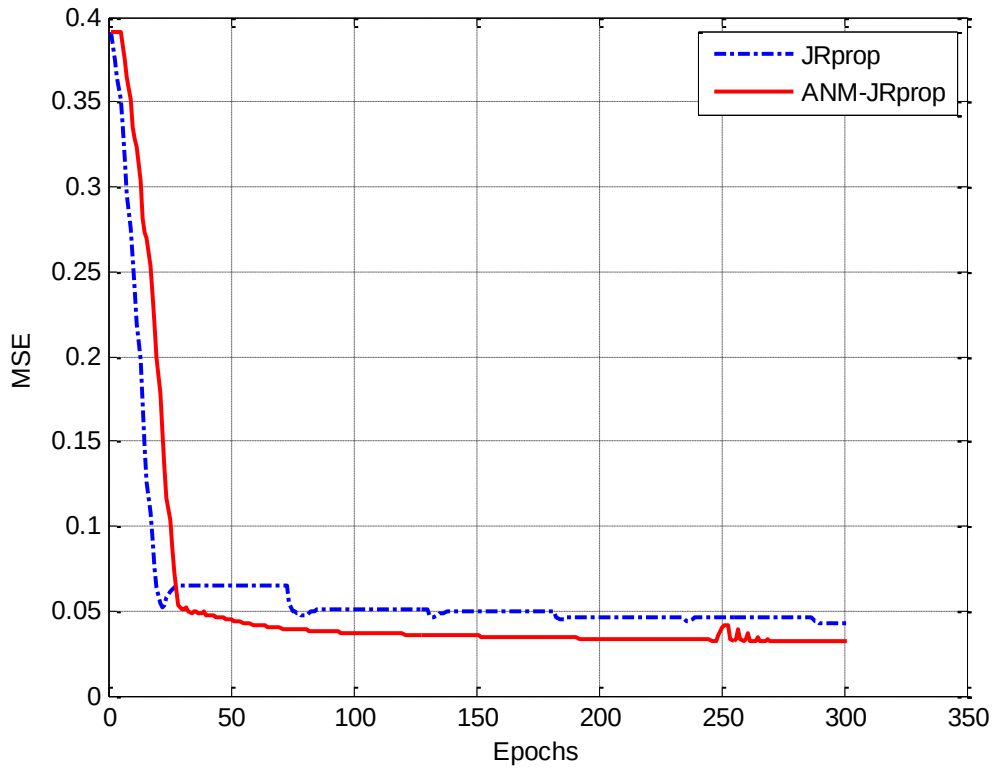


Figure 4.16 Examples of learning behaviours (JRprop vs. ANM-JRprop): RA problem, NARX

4.5 Summary and Contribution of the Chapter

In this chapter, starting from original versions of the Rprop approach, four modifications were discussed in Section 4.1, while the property of global convergence was reviewed in Section 4.2. A nonmonotone version of Rprop algorithm was developed in Section 4.3. This approach exploits function comparisons to produce an Rprop-inspired scheme that can be theoretically considered as a composite nonlinear Jacobi method. The subminimisation process of the nonlinear Jacobi locates subminimisers along each weight direction by employing Rprop steps with a nonmonotone strategy. As shown in Section 4.4 simulation results prove that our modification of Rprop algorithm is superior to the recently proposed JRprop approach, which in previous work demonstrated better performance than other four Rprop methods discussed in Section 4.1. In our tests, the nonmonotone algorithm outperformed the monotone JRprop version in terms of lower training MSEs, higher rates of convergence, fewer epochs in the converged runs, and smaller standard deviations of the converged runs for the P5 and P10 problems. It also exhibited smaller MSEs and CEs in both training and testing for the SC problem, as well as lower MSEs in both training and testing for the SL and RA problems.

Chapter 5

Adaptive Nonmonotone Conjugate Gradient Algorithms

The famous conjugate gradient (CG) method was originally proposed in 1952 [88] for linear functions, while the non-linear version was introduced in 1964 [65]. As stated in Chapter 1 of this thesis, one objective of the research is focusing on the class of CG methods. Therefore, the traditional monotone (such as [3][26][37][32][33][38][49][62] [53][54][61][71][75][133][159][161][166][3][172][76]) and the latest nonmonotone revisions (such as [113][114][185][55][181][182]) are firstly discussed, followed by the proposed adaptive nonmonotone CG approaches and their concrete simulation results on one artificial (i.e. N-bit parity) and three real-world (i.e. SC [116], SL [124] and RA [158]) applications. In the experiments, three different RNN architectures are used (i.e. FFTD [196][197], LRN [57][85] and NARX [128][137], see discussion in Section 2.1 for further details) with different numbers of hidden nodes (precise settings of these RNNs are provided in Appendix A.1, while the summary of free parameter amounts for simulated problems is depicted in Table A.1).

The rest of this chapter is organised as follows. In Section 5.1, the CG methods are firstly reviewed, and then global convergence of the nonmonotone version is

discussed in Section 5.2, followed by our proposed nonmonotone CG algorithms (Section 5.3, [146][147]) and experimental results in Section 5.4. Section 5.5 concludes this Chapter.

5.1 Conjugate Gradient Methods

Conjugate Gradient methods are in principle approaches suitable for large-scale problems [71]. The basic idea of CG methods is to find the stepsize in Eq. (3.2) along a linear combination of the current gradient vector and the previous search direction. The ways of determining search direction *d* can be expressed as follows:

$$d_0 = -g_0 \tag{5.1}$$

$$d_k = -g_k + \beta_{k-1} d_{k-1}, \text{ if } k \geq 1, \tag{5.2}$$

where the well-known choices of the parameter β are: the Hestenes-Stiefel formula [88] $\beta_k^{HS} = \frac{(g_k - g_{k-1})^T g_k}{d_{k-1}^T (g_k - g_{k-1})}$, the Fletcher-Reeves update [65]

$$\beta_k^{FR} = \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}} \tag{5.3}$$

and the Polak-Ribière approach [159]

$$\beta_k^{PR} = \frac{(g_k - g_{k-1})^T g_k}{g_{k-1}^T g_{k-1}}, \tag{5.4}$$

Traditional CG methods [3][161][172][76] are designed for static neural networks (NNs). There are however some CG approaches proposed for dynamic networks, such as [32][33][38]. Using the information of the Hessian matrix and/or proper line search methods, these CG approaches are guaranteed to find a local minimum rapidly. The larger the scale of the problem is, the more obvious the benefits of using CGs [71].

By using two product-calculating techniques [33], the *direct calculation* of the Hessian matrix in the scaled conjugate gradient (SCG) method [133] was prevented, and then two new 2^{nd}-order dynamic SCG-based algorithms were proposed. By recursively determining the CG direction, the SCG requires only O(*N*) memory usage, where *N* is the number of weights [133]. However, instead of directly calculating, the SCG has used an estimation of the Hessian matrix. The statement of [33] was not exactly correct. Furthermore, the simulations in [33] only compared the authors' revised version with RBP (recursive backpropagation, [32][34] and TRBP (truncated RBP, [35]) and no direct comparisons with the original SCG was made. Thus, claims for 'better' performance need to be verified experimentally, as in [53].

Through deriving general formulas, the works in [53][54] applied their improved SCG to dynamic NNs, and simulation results in nonlinear system identification and time series prediction problems showed that the performance is better than the original SCG. The improved SCG [53] uses the hybrid choice of direction parameter and exact multiplication by the Hessian matrix, while the approach of exact multiplication is the method indicated in [33]. Comparing to the original SCG, the performance of the hybrid SCG [53] is better, involving the usage of their output feedback RNN (OFRNN) and fully RNN (FRNN). The cases of equivalences between globally RNN, locally RNN, OFRNN and FRNN are more clearly indicated in [54]. From the discussions in [53][54] it is worth mentioning that when a very large number of

training iterations is involved the numerical precision in the direct calculation of the Hessian matrix is gradually lost so it might make sense to estimate the Hessian instead.

One other advantage in [33] as the authors claim is the adoption of the revised SCG into a dynamic version. Applying the two product-calculation techniques to the RBP and TRBP, the learning algorithms designed for IIR-MLP, the formulas of forward and backward phases are obtained. The ambiguous question here refers to the relationships between RBP/TRBP and SCG. As the name indicating the moving step of SCG is scaled by a formula consisting of the 1^{st}- and 2^{nd}-order information through the recursively decided conjugate direction. If the revised SCGs of [33] only applied the product methods into RBP and TRBP and then became CG-based dynamic algorithms, RBP and TRBP must have some equivalence to SCG, i.e. for example, usage of the Hessian matrix. However, since they are gradient descent methods, only the 1^{st}-order information is used, it seems almost all benefits of the original SCG are gone, even though RBP and TRBP are suitable for dynamic NNs. The question of how to compare the dynamic version of SCG to the static version under very different basis of network complexity should be explored further.

The work in [38] modified the real time recurrent learning (RTRL) with conjugate gradient and proposed a CGRL algorithm. The pros and cons between the BPTT and RTRL were also discussed, while more details about comparison of BPTT and RTRL can be found in [49]. Feng [62] proposed a study of the CG method, while a latest extended review reveals its dynamic behaviour in [169]. Bhaya and Kaszkurewicz [26] built various connections between CG and BP with momentum for modifying the convergence properties. More discussions about gradient descent and CG approaches can be found in [37]. Different CG methods had been compared in [115][178] for both

static and dynamic neural networks. The authors in [135] proposed an adaptive CG for linear minimisation, while González and Dorronsoro [76] introduced natural gradient into CG for training perceptrons. Numerous applications and developments for CG methods can be found in the literature, such as the works in [1][4][17][92][108] [139][153][191][199].

The nonmonotone version of CG methods has been also studied in the context of optimisation [113][114][185][55][181][182], and there is theoretical evidence that these variants are globally convergent for both convex [55][113][181][182] and nonconvex [185] objective functions. Furthermore, nonmonotone methods for general nonconvex functions have been analysed in [46] and conditions for global convergence have also been established [45]. Note that the global convergence property is totally different from global optimisation; it means that starting from almost any initial weight these methods will always reach a minimiser [118] but not necessarily the global minimum.

Considering the objective functions of RNNs' learning are nonconvex, we present below the main theoretical results for global convergence of nonmonotone CG methods which hold in this case. It is worth mentioning that proving global convergence for nonconvex objective functions is a very challenging problem. Here we are based on the work of Liu and Wei [114], and apply the nonmonotone Grippo-Lampariello-Lucidi (GLL) line-search [80], which can be stated as follows

$$E\left(w_k+\alpha_k d_k\right)\le \max_{0\le j\le m(k)}\left\{E\left(w_{k-j}\right)\right\}+\varepsilon_1\alpha_k g_k^T d_k\,, \tag{5.5}$$

$$\left|\left\langle g\left(w_k+\alpha_k d_k\right), d_k\right\rangle\right|\le -\varepsilon_2 g_k^T d_k\,, \tag{5.6}$$

where ε_1 and ε_2 are constants, $0 < \varepsilon_1 < \varepsilon_2 < 1/2$, and $m(k)$ is updated by the following rule:

$$\begin{cases} m(0) = 0 \\ m(k) \le \min\{m(k-1)+1, M\} \end{cases}. \tag{5.7}$$

5.2 Global Convergence of Nonmonotone Conjugate Gradient

Before presenting the main theorems for global convergence, the following Assumption (**H5**) and Property (**P**) are needed.

(**H5.1**) For a given point $w_0 \in R^n$ and for every w in some region that contains the initial weight vector w_0, the level set $L_0 = \{w \in R^n \mid f(w) \le f(w_0)\}$ is bounded.

(**H5.2**) In some neighbourhood B of the level set L_0, the error function is continuous differentiable and the gradient g is Lipschitz continuous, i.e. there exists $L > 0$ for every pair w, v such that

$$\|g(w) - g(v)\| \le L\|w - v\|, \forall w, v \in B \tag{5.8}$$

(**H5.3**) The search direction d_k satisfies the following sufficient decrease condition, i.e., there exists a positive constant c_0 such that

$$\langle -g_k, d_k \rangle \ge c_0 \|g_k\|^2. \tag{5.9}$$

Property (**P**) [70]. Consider a CG method defined by Eqs. (3.2), (5.1) and (5.2), and suppose that

$$0 < \zeta_1 \le \|g_k\| \le \zeta_2 \tag{5.10}$$

for all $k \ge 1$, where ζ_1 and ζ_2 are constants. Under this assumption, the CG method is called to have property (**P**) if there exist constants $\zeta_3 > 1$ and $a > 0$ such that, for all *k*,

$$|\beta_k| \le \zeta_3, \tag{5.11}$$

$$\|w_{k+1} - w_k\| \le a \Rightarrow |\beta_k| \le \frac{1}{2\zeta_3} \tag{5.12}$$

Below we present the theorems by Liu and Wei [114] which need part of Property (**P**) on the sufficient descent condition and Assumptions (**H5**) to establish the global convergence of the nonmonotone CG method.

□ **Theorem 5.1** [114]. *Suppose that Assumptions (**H5**) hold. Let* $\{w_k\}$ *and* $\{d_k\}$ *be generated by the CG rule, Eqs. (3.2), (5.1) and (5.2), using the Fletcher-Reeves, Eq. (5.3), or the Polak-Ribière, Eq. (5.4), update.* $|\beta_k| \le \zeta_3$, *and the stepsize* α_k *satisfies the nonmonotone GLL linesearch, Eqs. (5.5)-(5.7). If Eq. (5.10) holds, then*

$$\sum_{k=0}^{\infty} \frac{1}{\|d_k\|^2} < +\infty. \tag{5.13}$$

Assumption (**H5.1**) holds in RNN training, defined in Chapters 2 and 3, because the error function *f* is bounded below in R^n since $f \ge 0$: for a RNN with a fixed architecture and a finite set of training patterns, if a w^* exists such that $E(w^*) = 0$, then w^* is the global minimum; otherwise the vector *w* with the smallest available value is the "global" minimiser. Assumption (**H5.2**) also holds for RNNs that use

smooth enough activations functions (the derivatives of order *p* are available and continuous), such as the well know hyperbolic tangent and logistic (used in our experiments) activations. Moreover, (**H5.2**) implies that there exists a constant ζ such that $\|g(w)\| \le \zeta, \forall w \in B$. Lastly, Gilbert and Nocedal [70] have shown that the condition of Eq. (5.9) is important to ensure global convergence of CG methods, and suggested that can be guaranteed by incorporating bracketing procedures in the method.

Using the result of Theorem 5.1, the following Theorem can establish the property of global convergence.

□ **Theorem 5.2** [114]. *Suppose that Assumptions (**H5**) hold. Let* $\{w_k\}, \{d_k\}$, *and* $\{\alpha_k\}$ *be generated as in Theorem 5.1, where* $\beta_k \ge 0$ *satisfy Property (**P**). Then*

$$\liminf_{k\to\infty} \|g_k\| = 0. \tag{5.14}$$

A detailed proof is provided in [114], which shows that the limit of Eq. (5.14) is the best type of global convergence result that can be obtained for nonconvex functions.

5.3 The Proposed Nonmonotone Conjugate Gradient Algorithms: ANMCG and A2NMCG

In this section, the two nonmonotone CG-class approaches [146][147] are presented. The first version of our proposed algorithm is the Adaptive Non-Monotone CG (ANMCG, [146]), as stated in Table 5.1, then the improved version of ANMCG is shown in Table 5.2, i.e., the Advanced ANMCG (A2NMCG, [147]).

Table 5.1 Algorithm: Adaptive Non-Monotone CG (ANMCG, [144])

Algorithm: ANMCG

STEP 0. Initialize w_0, k, boundaries of M_k, and $d_0 = -g_0$;

STEP 1. If $g_k = 0$ stop;

STEP 2. Adapt M_k by the following conditions:

$$M_k = \begin{cases} M_{k-1} + 1, & \text{if } \Lambda_k < \Lambda_{k-1} < \Lambda_{k-2} \\ M_{k-1} - 1, & \text{if } \Lambda_k > \Lambda_{k-1} > \Lambda_{k-2} \\ M_{k-1}, & \text{otherwise,} \end{cases},$$

where $\Lambda_k = \dfrac{\|g_k - g_{k-1}\|}{\|w_k - w_{k-1}\|}$;

STEP 3. Find a step length α_k satisfying the following nonmonotone condition:

For $0 < \lambda_1 < \lambda_2$ and $\sigma, \delta \in (0,1)$, at each iteration, one chooses a parameter l_k such that the step length $\alpha_k = \bar{\alpha}_k \cdot \sigma^{l_k}$, where $\bar{\alpha}_k \in (\lambda_1, \lambda_2)$, satisfies ;

$$E\left(w_k + \alpha_k d_k\right) \le \max_{0 \le j \le M_k} \left[E\left(w_{k-j}\right)\right] + \delta \cdot \alpha_k \cdot \nabla f\left(w_k\right)^T \cdot d_k$$

STEP 4. Generate new point by $w_{k+1} = w_k + \alpha_k d_k$;

STEP 5. Update search direction by $d_k = -g_k + \beta_{k-1} d_{k-1}$, where $\beta_k = \beta_k^{PR}$ or β_k^{FR};

STEP 6. STEP 6. Let $k = k + 1$, go to STEP 1.

Table 5.2 Algorithm: Advanced ANM-CG Algorithm (A2NMCG, [145])

Algorithm: A2NMCG

STEP 0. Initialise w_0, $k=0$, $M^0 = 0$, $M^{\max}$ is an upper boundary for M_k, $l_0 = 0$, $a_0, \sigma, \delta \in (0,1)$ and $d_0 = -g_0$;

STEP 1. If $g_k = 0$, then stop;

STEP 2. If $k \geq 1$, calculate a local approximation of the Lipschitz as $\Lambda_k = \dfrac{\|g_k - g_{k-1}\|}{\|w_k - w_{k-1}\|}$, and adapt M_k by the following conditions:

$$M_k = \begin{cases} M_{k-1} + 1, & \text{if } \Lambda_k < \Lambda_{k-1} < \Lambda_{k-2} \\ M_{k-1} - 1, & \text{if } \Lambda_k > \Lambda_{k-1} > \Lambda_{k-2} \\ M_{k-1}, & \text{otherwise,} \end{cases},$$

where $M_k = \min\{M_k, M^{\max}\}$;

STEP 3. For all $k \geq 1$, find a stepsize $\alpha_k = (2\Lambda_k)^{-1} \cdot \sigma^{l_k}$ satisfying the following condition:

$$E(w_k + \alpha_k d_k) \leq \max_{0 \leq j \leq M_k} \left[E(w_{k-j}) \right] + \delta \cdot \alpha_k \cdot g_k^T \cdot d_k,$$

where $l_k = l_k + 1$;

STEP 4. Generate a new point by $w_{k+1} = w_k + \alpha_k d_k$;

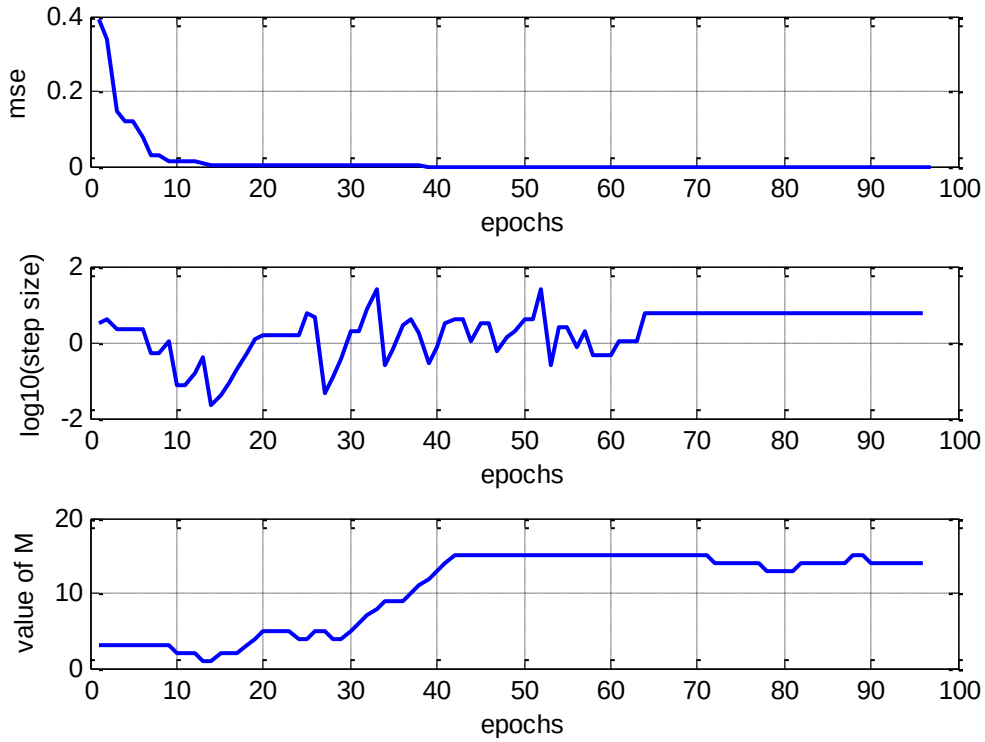
STEP 5. Update search direction $d_k = -g_k + \beta_{k-1} d_{k-1}$, where $\beta_k = \beta_k^{PR}$ or β_k^{FR} and is greater than zero;

STEP 6. Let $k = k+1$, go to STEP 1.

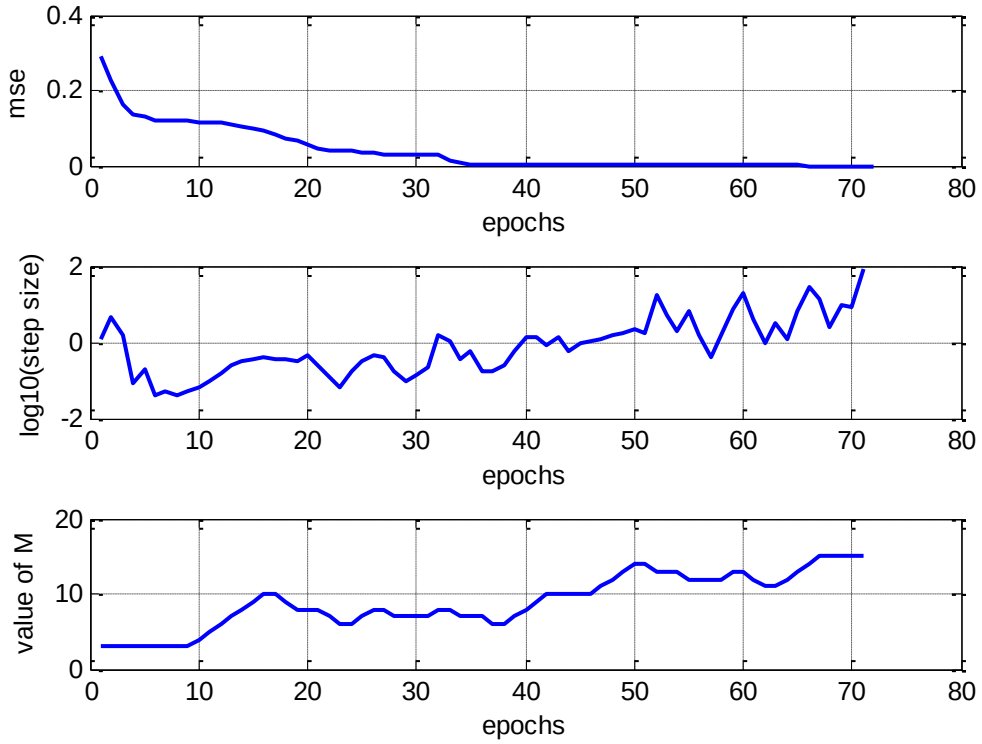
The main difference between the Adaptive Non-Monotone Conjugate Gradient (ANMCG) method and the A2NMCG method lies in the way the stepsize in Step 3 is adapted. In ANMCG we use the steplength of the previous iteration as the new stepsize, while A2NMCG takes a local approximation of the Lipschitz constant at each iteration as the initial trail steplength to satisfy the nonmonotone condition. The

Lipschitz constant is related to the morphology of a function. The local approximation used here provides information regarding the local shape of the error function. Thus, the local approximation gets large values in steep regions and small values in flat regions (see also [119][195] for the usefulness of this estimate).

To illustrate the behaviour of the new algorithm we provide below typical examples of convergence behaviour from learning the partity-5 and parity-10 problems [176] using RNNs that belong to the NARX group of models. Figures 5.1 and 5.2 illustrate the behaviour of the MSE, the stepsize, and the adaptive learning horizon for the P5 and P10 problems using a 2-hidden-node NARX network, which was trained to reach an error goal of 1e-100 within 1000 epochs. In all cases, the use of an adaptive learning horizon *M* did not affect stable behaviours (i.e. faster convergence to training goals) of the methods. Moreover, Figures 5.1 and 5.2 show that the trend for the adaptive *M* is to increase in all cases, despite temporary reductions at some iterations; an observation which is in accordance with theoretical results [113][114].

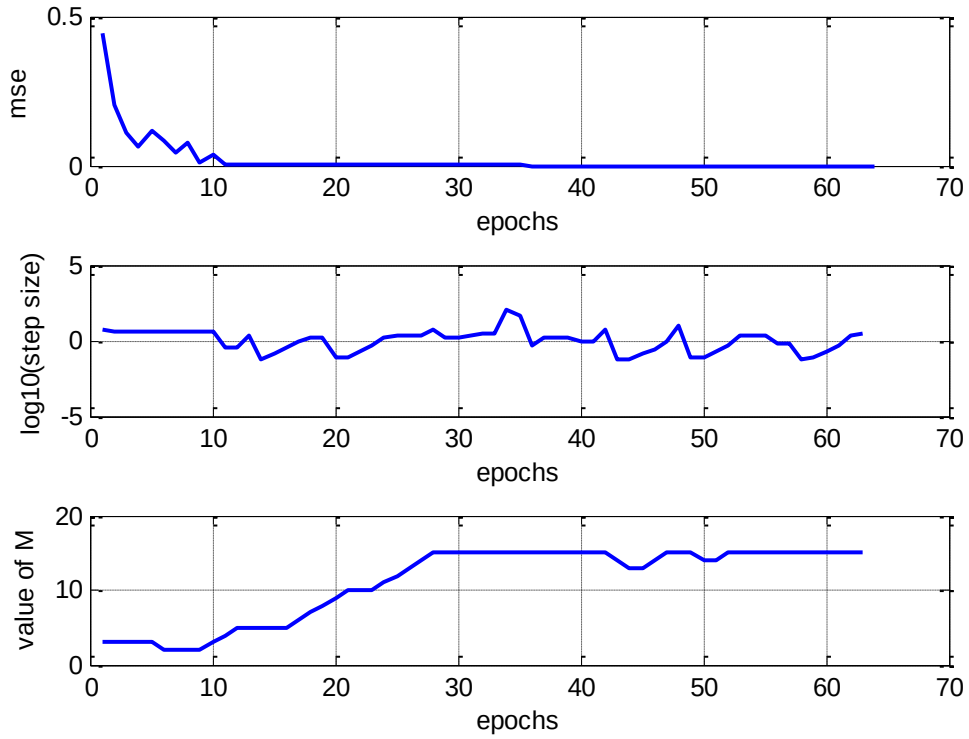


(a) ANMCG

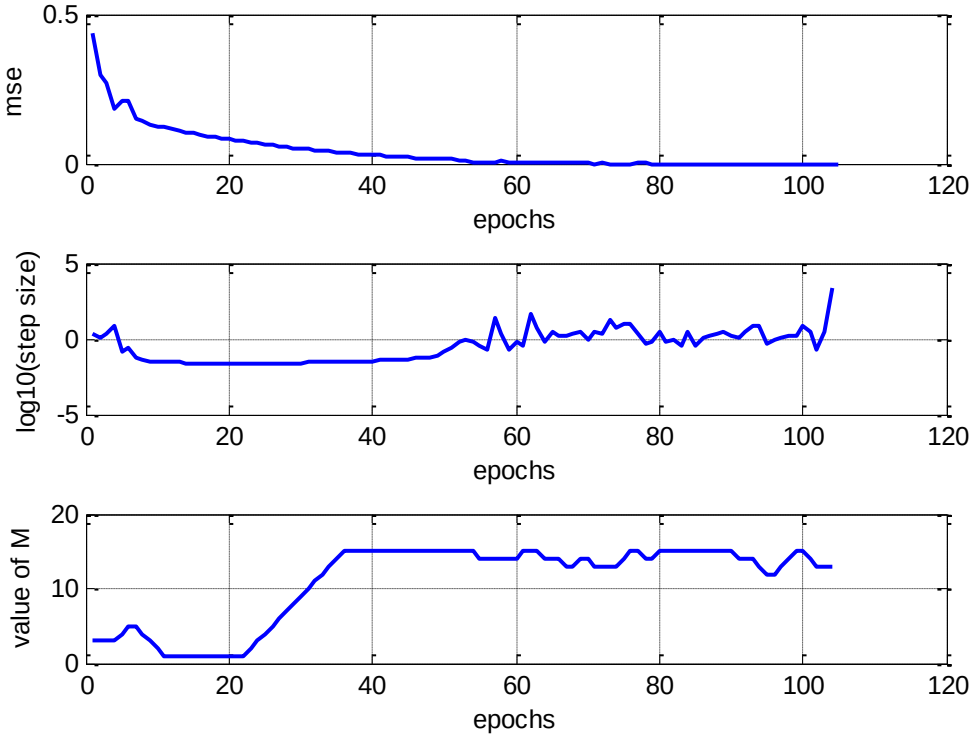


(b) A2NMCG

Figure 5.1 Convergence behaviours of NARX networks trained with the (a) ANMCG and (b) A2NMCG methods in the P5 problem.



(a) ANMCG



(b) A2NMCG

Figure 5.2 Convergence behaviours of NARX networks trained with the ANMCG and A2NMCG methods for the P10 problem.

5.4 Experimental Results

The five simulated problems used here include two instances of the *N-bit parity* (P5 and P10), a *sequence classification* (SC) [116] problem, the *sequence learning* (SL) task [124] and the *reading aloud* (RA) [158] problem, while full descriptions can be found in Appendix A.1 and Table A.1 summarises the numbers of the three RNNs' adjustable parameters used in the experiments of the thesis.

In the tables below, *Algo* denotes the training algorithm, *#hid* stands for the number of hidden nodes, and *Conv* denotes the convergence success in terms of the number of runs that reached the predefined training goals out of 100. *MSE* is the average of the mean-squared-error (in percentage) achieved in the 100 runs (i.e. the average MSE has been calculated over the total number of runs and not only the converged ones to provide an estimate of the overall performance of the methods for practical applications when it is difficult for the user to set precise values for the training goal and the number of epochs), while *STD* is the corresponding standard deviation of the MSE values. *Ave*, *Min*, *Max* and *Std* respectively denote the average, minimum, maximum, and standard deviation of epochs for the converged runs. In all cases, the nonmonotone parameters were set to $M^{\max} = 15,\ \delta = 0.01,$ and $\sigma = 0.1,$ in order to test the robustness and generalisation of the proposed algorithms.

Besides the numerical results in the tables for each simulated problem, the graphical examples of convergence behaviours are provided to illustrate the improvements made by our proposed nonmonotone algorithms.

5.4.1 Parity Problems

We tested RNNs with 1, 2, 5 and 7 hidden nodes for the P5 problem and 1, 2, 5, 7 and 10 hidden nodes for the P10 problem. The training goal was to reach an MSE of 0.01 within 2000 epochs for P5 and 4000 epochs for P10. Tables 5.3-5.5 show the results for P5, while Tables 5.6-5.8 for P10. Figures 5.3-5.6 are examples of typical learning behaviours from these problems, comparing the monotone CG with the ANMCG, and illustrate some of the improvements made by the A2NMCG method.

Table 5.3 Average performance for FFTD networks in the P5 problem: class of CG

Algo	#hid	Conv	MSE	STD	Epoch			
		(%)	(%)	(%)	Ave	Min	Max	Std
CG	1	0	24.981	8.500	-	-	-	-
	2	0	22.315	7.032	-	-	-	-
	5	12	10.546	4.117	406	62	1638	417
	7	41	3.782	2.929	215	22	1097	172
ANMCG	1	**2**	**14.781**	**7.891**	1565	1565	1565	0
	2	**5**	**10.332**	**4.272**	1198	981	1833	695
	5	**23**	**4.459**	**1.950**	842	254	1826	523
	7	**76**	**1.972**	**1.005**	645	167	1828	434
A2NMCG	1	**5**	**13.450**	**5.020**	1097	524	1547	833
	2	**13**	**8.518**	**2.982**	850	494	1789	714
	5	**34**	**2.762**	**1.117**	484	127	1313	457
	7	**81**	**1.404**	**0.705**	357	106	1922	398

Table 5.4 Average performance for LRN networks in the P5 problem: class of CG.

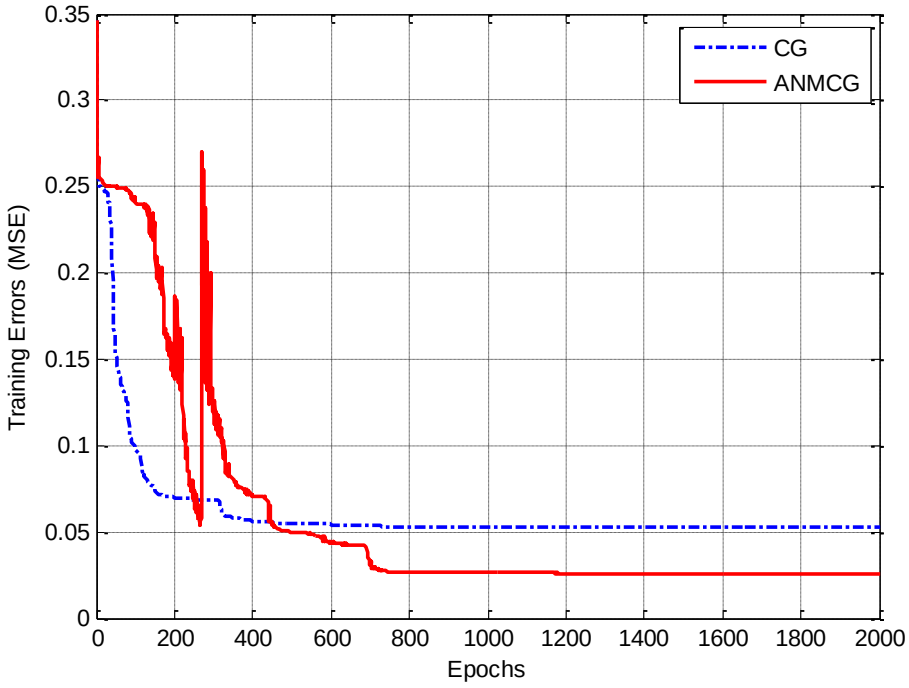
Algo	#hid	Conv (%)	MSE (%)	STD (%)	Epoch			
					Ave	Min	Max	Std
CG	1	0	23.302	8.674	-	-	-	-
	2	0	20.964	7.183	-	-	-	-
	5	14	7.610	3.871	202	46	610	145
	7	37	5.678	3.007	189	39	561	239
ANMCG	1	9	18.776	**6.409**	1452	62	1994	1275
	2	22	11.351	**4.032**	1378	334	1851	977
	5	32	4.699	**2.595**	1135	213	1947	633
	7	84	2.785	**1.730**	798	61	1933	492
A2NMCG	1	13	16.641	**5.875**	981	54	1959	860
	2	27	9.765	**3.711**	1039	231	1937	1084
	5	36	3.561	**2.194**	587	148	1894	571
	7	86	1.866	**1.364**	532	56	1908	445

Table 5.5 Average performance for NARX networks in the P5 problem: class of CG.

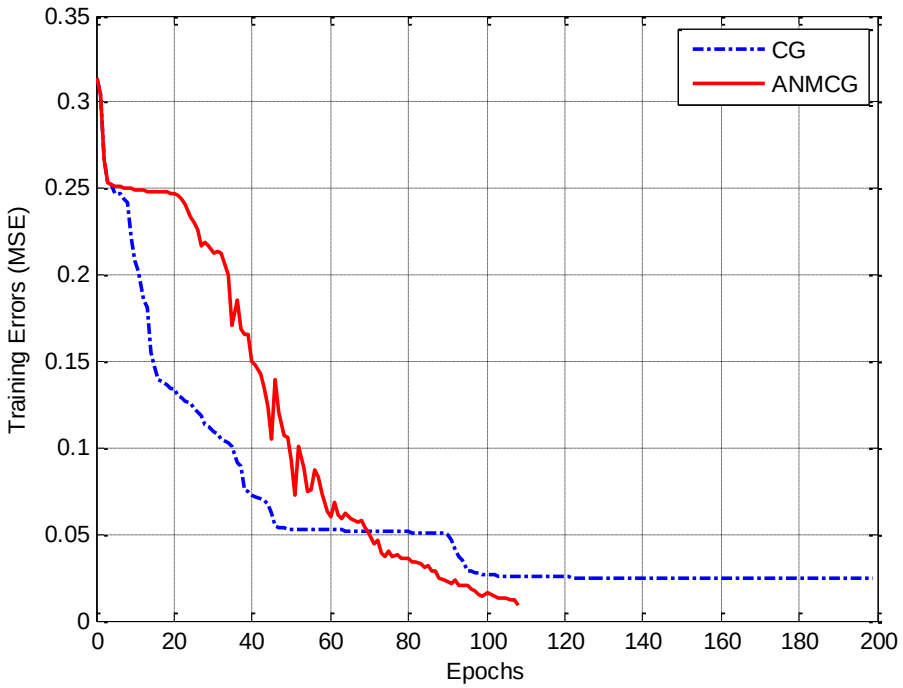
Algo	#hid	Conv (%)	MSE (%)	STD (%)	Epoch			
					Ave	Min	Max	Std
CG	1	24	10.589	4.779	73	21	283	46
	2	30	6.985	3.261	44	13	136	33
	5	76	3.451	1.604	38	9	161	14
	7	95	0.915	0.047	42	6	1199	124
ANMCG	1	97	1.110	**0.951**	32	2	1341	138
	2	98	1.063	**0.730**	20	4	151	23
	5	99	0.833	**0.054**	14	4	36	11
	7	99	0.812	**0.013**	12	4	54	18
A2NMCG	1	96	1.419	1.288	21	2	150	21
	2	99	0.998	**0.684**	18	5	71	15
	5	100	0.729	0.066	13	4	34	8
	7	100	0.638	**0.004**	11	4	23	6

The results show that the A2NMCG algorithm consistently converges to desired solutions, even with fewer adjustable variables. In certain cases the use of the nonmonotone strategy may result in an increase in the number of epochs but this additional cost allows the new method to locate minimisers with smaller error values. Even when the nonmonotone methods do not reach an MSE = 0.01 within 2000 epochs, they always achieve smaller errors than the CG method. For example, in Table 5.3, CG-trained FFTD networks with 2 hidden nodes cannot reach the training goal within 2000 epochs (Conv = 0 in Table 5.3) and on the average generate an MSE=22.315%. Although only 5% of the ANMCG-trained FFTD networks reach the

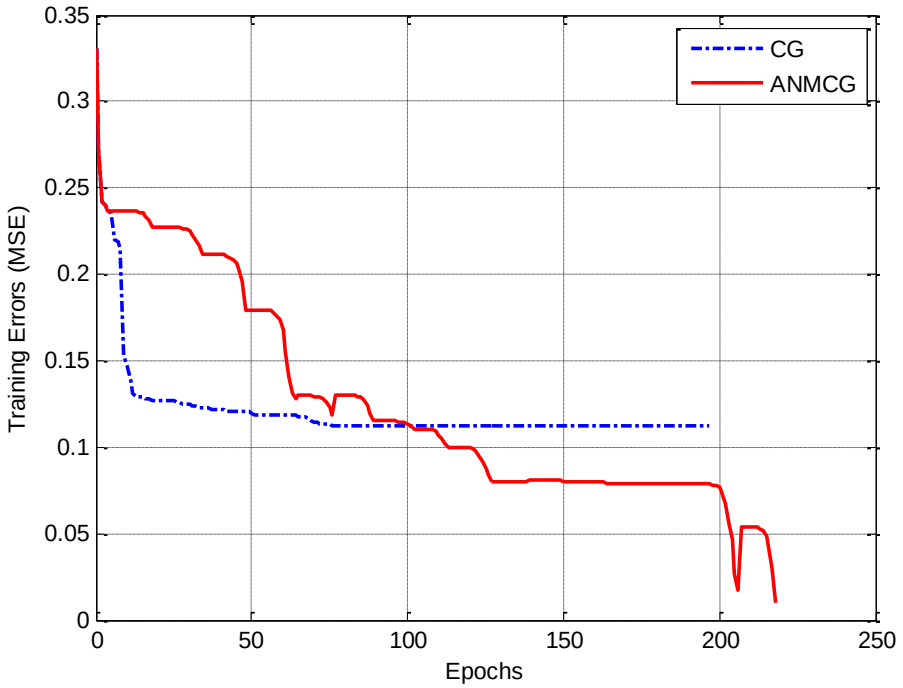
training goal, these networks produce on the average significantly lower error values, i.e. the average MSE over the total number of runs (not only the converged ones) is 10.332%. For A2NMCG-trained FFTD networks, results in Table 5.3 show that 13% reach the training goal within 2000 epochs, and that the average MSE over the total number of runs is 8.518%, which constitutes a significant improvement over the other two methods.



(a) FFTD



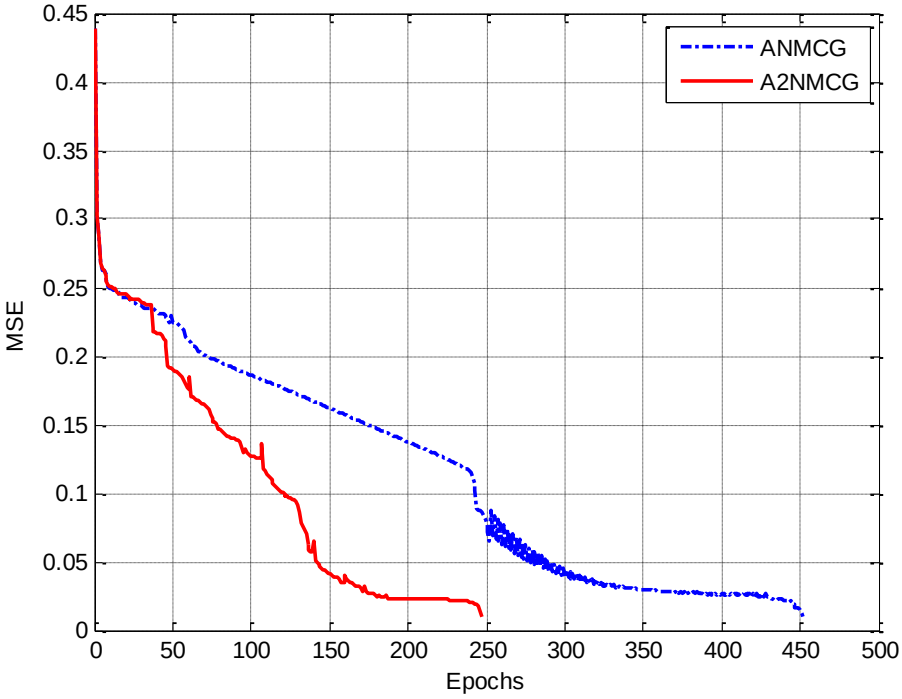
(b) LRN



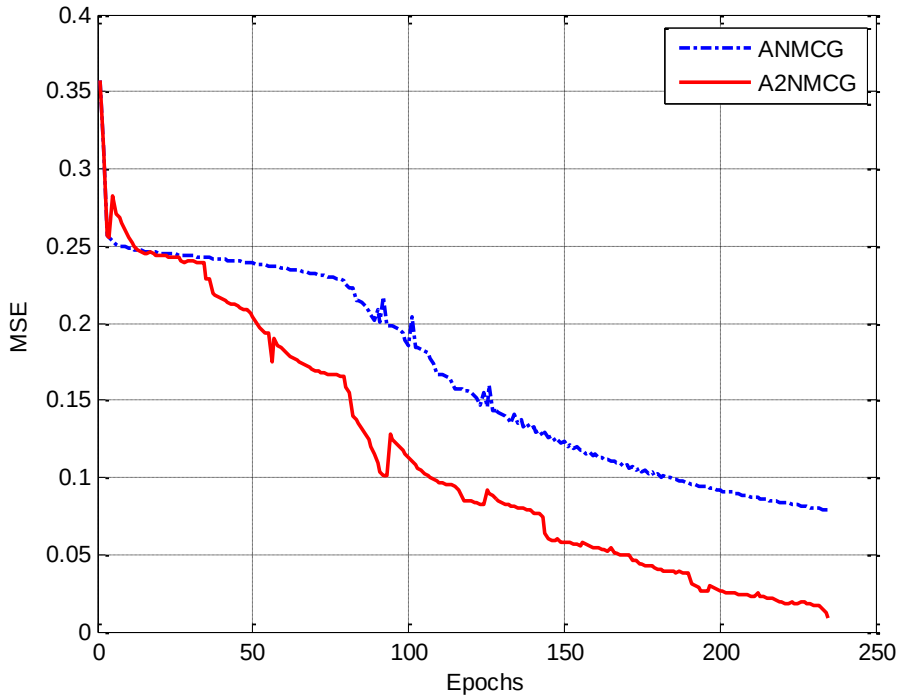
(c) NARX

Figure 5.3 Examples of convergence behaviour for the CG (blue dashed line) and the ANMCG (red solid line) in the P5 problem for three RNNs.

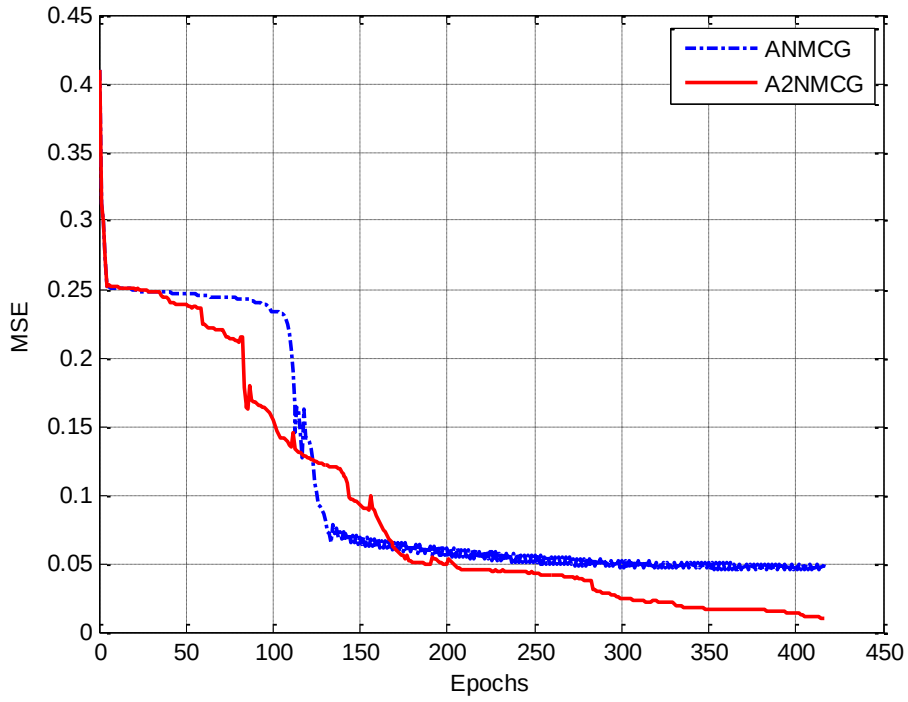
Examples of learning behaviours for the CG and the nonmonotone version in the P5 and P10 problems are illustrated in Figures 5.3-5.4 and 5.5-5.6, respectively. In particular, Figure 5.3a provides an example of the robustness of the approach even when there are large variations in the M previous error function values. Figure 5.4 demonstrates the improved behaviour of the A2NMCG over the ANMGG.



(a) FFTD



(b) LRN



(c) NARX

Figure 5.4 Examples of convergence behaviour for the ANMCG (blue dashed line) and the A2NMCG (red solid line) methods in the P5 problem for three RNNs.

Table 5.6 Average performance for FFTD networks in the P10 problem: class of CG.

Algo	#hid	Conv (%)	MSE (%)	STD (%)	Epoch			
					Ave	Min	Max	Std
CG	1	0	24.997	7.643	-	-	-	-
	2	0	24.841	7.081	-	-	-	-
	5	1	20.119	8.990	775	775	775	0
	7	10	15.597	5.287	966	318	2976	780
	10	28	8.197	4.901	554	207	1001	216
ANM-CG	1	**11**	**18.013**	**7.275**	**1698**	**30**	**2744**	**1707**
	2	**19**	**15.604**	**7.006**	**1451**	**585**	**2315**	**1321**
	5	**23**	**13.531**	**6.804**	1302	**287**	2355	1219
	7	**32**	**9.387**	**5.103**	1394	334	2477	1588
	10	**39**	**6.156**	**2.834**	1164	210	2753	1251
A2NM-CG	1	**15**	**15.635**	**6.101**	**1345**	**30**	3112	**938**
	2	**22**	**13.851**	**6.382**	**1181**	**563**	2320	**994**
	5	**29**	**10.944**	**5.719**	**936**	**216**	**2131**	**1027**
	7	**35**	**7.368**	**3.504**	**878**	**192**	2537	**815**
	10	**45**	**4.581**	**1.933**	**872**	**186**	**2710**	**702**

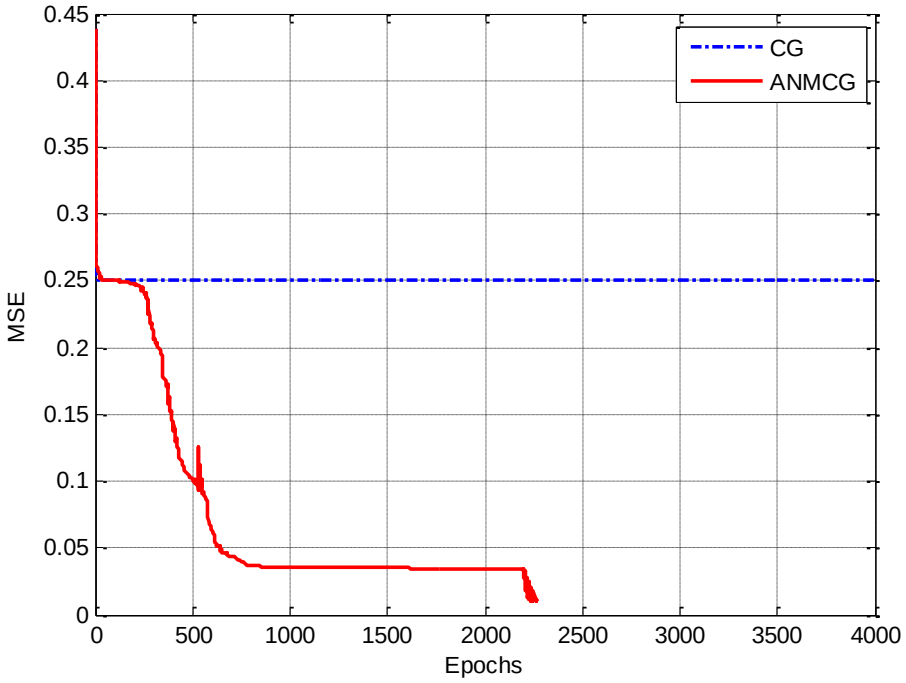
Table 5.7 Average performance for LRN networks in the P10 problem: class of CG.

Algo	#hid	Conv (%)	MSE (%)	STD (%)	Epoch			
					Ave	Min	Max	Std
CG	1	0	24.991	7.433	-	-	-	-
	10	42	8.441	4.827	523	196	2328	264
ANMCG	1	0	25.001	**7.109**	-	-	-	-
	10	**52**	**3.189**	**3.203**	3155	**179**	3852	2561
A2NMCG	1	**9**	**22.752**	**5.410**	**1431**	**80**	**3942**	**1130**
	10	**61**	**1.984**	**2.788**	**2548**	**142**	**3756**	**854**

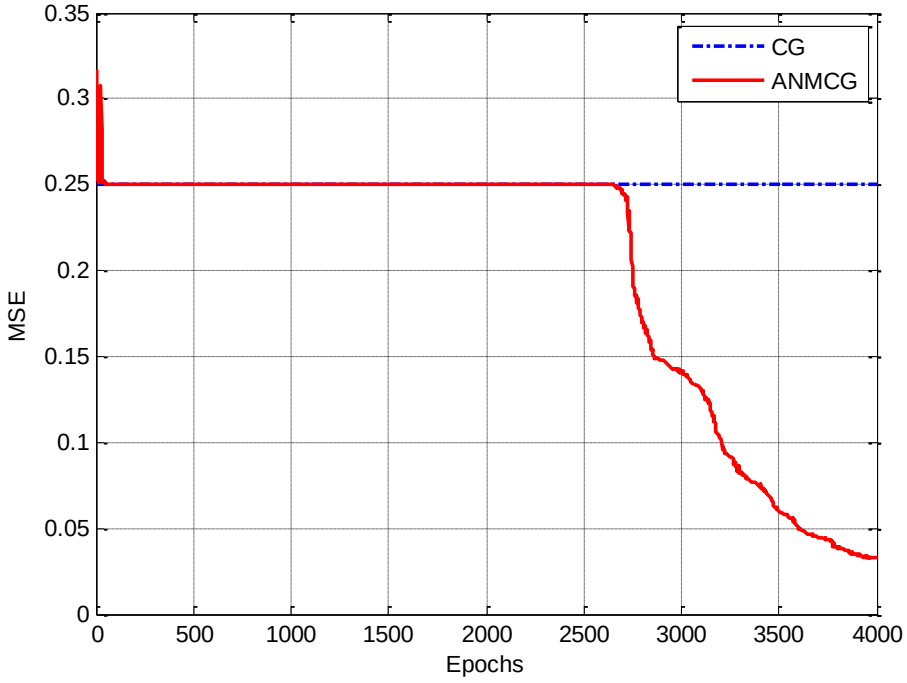
Table 5.8 Average performance for NARX networks in the P10 problem: class of CG.

Algo	#hid	Conv (%)	MSE (%)	STD (%)	Epoch			
					Ave	Min	Max	Std
CG	1	35	7.473	5.832	42	23	119	29
	2	43	6.161	4.991	35	15	99	21
	5	97	1.238	2.489	26	12	82	13
	7	100	0.920	0.073	18	11	65	9
	10	100	0.854	0.058	21	14	47	6
ANMCG	1	**61**	**4.562**	**3.862**	827	42	3849	563
	2	**66**	**3.089**	**1.997**	564	41	3154	471
	5	**100**	**0.931**	**0.084**	383	54	1128	125
	7	99	**0.889**	**0.068**	210	61	1251	124
	10	**100**	0.924	0.077	194	67	325	59
A2NMCG	1	**98**	**1.079**	**0.993**	**12**	**4**	**299**	**30**
	2	**99**	**0.894**	**0.764**	**15**	**5**	**68**	**11**
	5	**100**	**0.636**	**0.082**	**12**	**4**	**34**	**5**
	7	**100**	**0.695**	0.091	**11**	**4**	**28**	**5**
	10	**100**	**0.639**	0.088	**12**	**4**	**235**	**23**

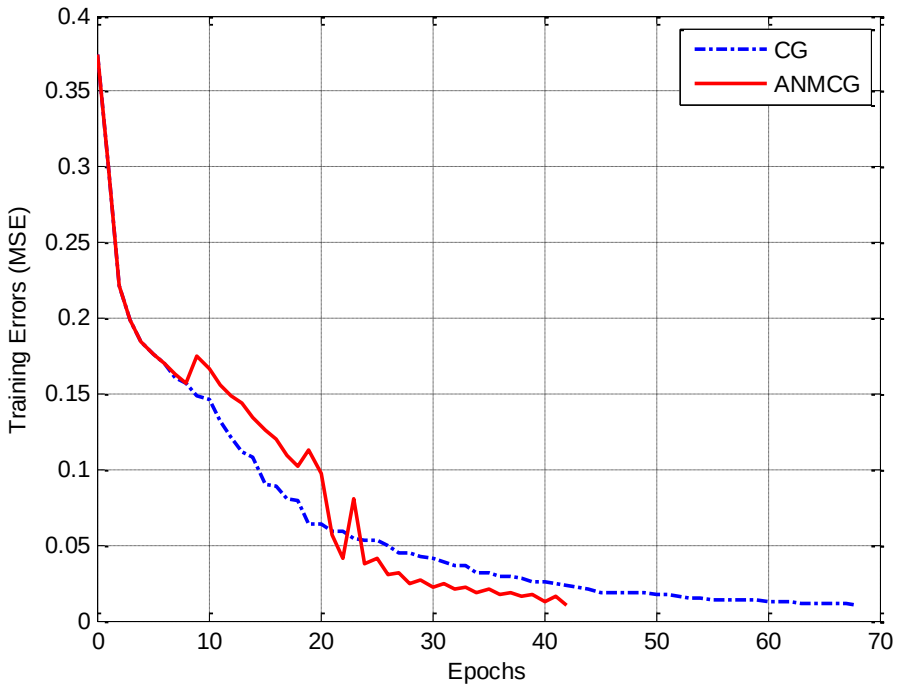
In Figures 5.5a and 5.5b, two cases are illustrated, for a FFTD network and a LRN respectively, where the nonmonotone CG escapes from local minima. Figure 5.6 provides examples of the improvements made by A2NMCG compared to the ANMCG.


CG
ANMCG
MSE
Epochs
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
500
1000
1500
2000
2500
3000
3500
4000

(a) FFTD

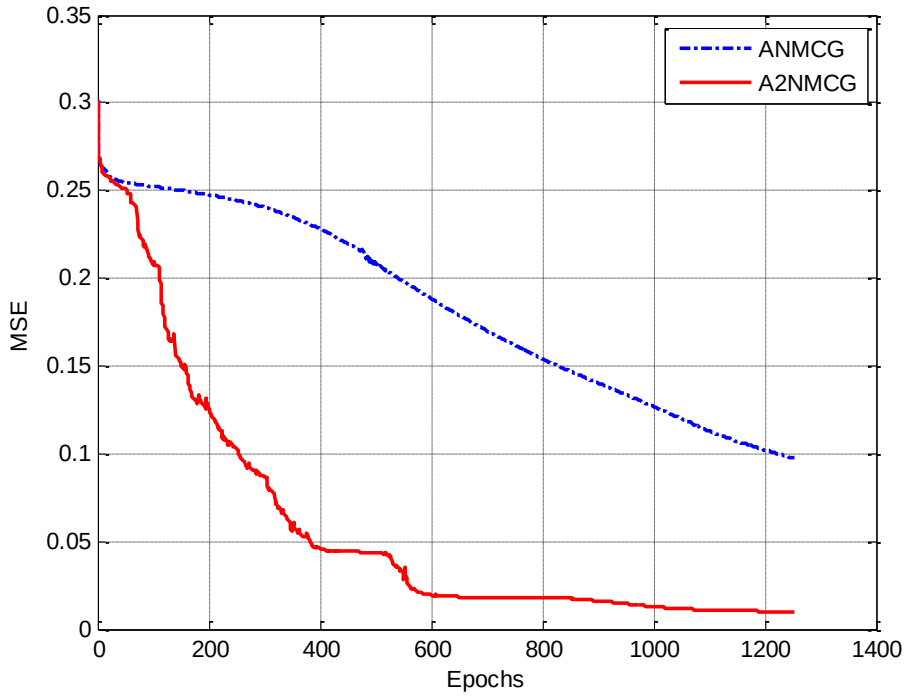

CG
ANMCG
MSE
Epochs
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
500
1000
1500
2000
2500
3000
3500
4000

(b) LRN

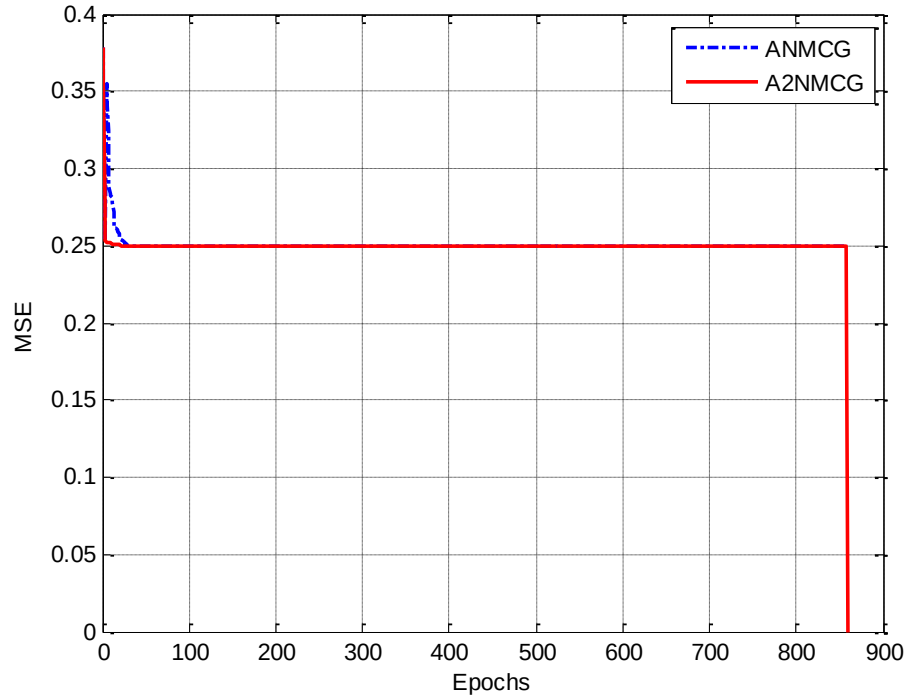


(c) NARX

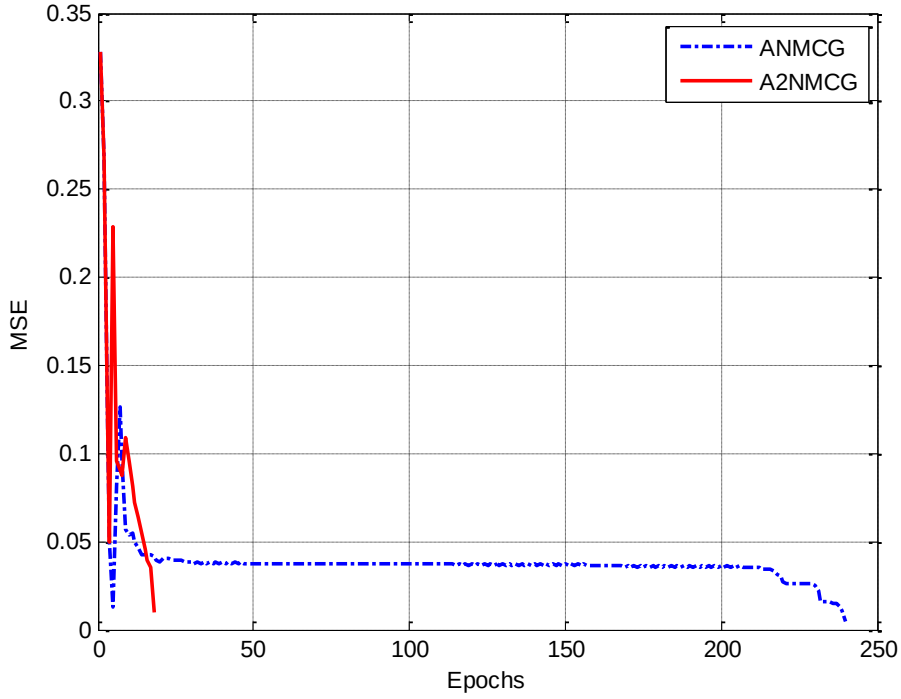
Figure 5.5 Examples of convergence behaviour for the CG (blue dashed line) and the ANMCG (red solid line) in the P10 problem for three RNNs.



(a) FFTD



(b) LRN



(c) NARX

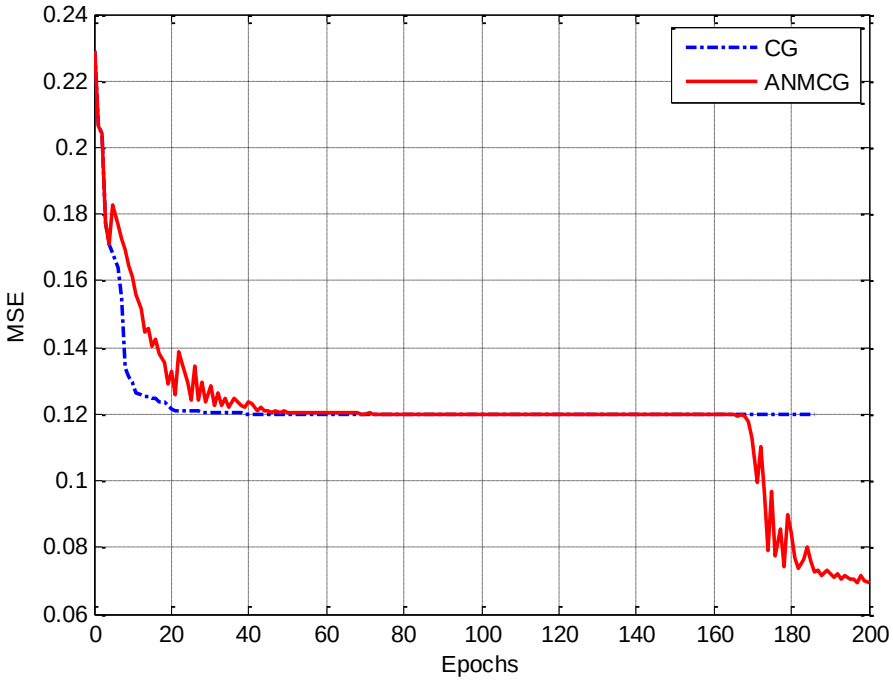
Figure 5.6 Examples of convergence behaviour for the ANMCG (blue dashed line) and the A2NMCG (red solid line) in the P10 problem for three RNNs.

5.4.2 Sequence Classification Problem

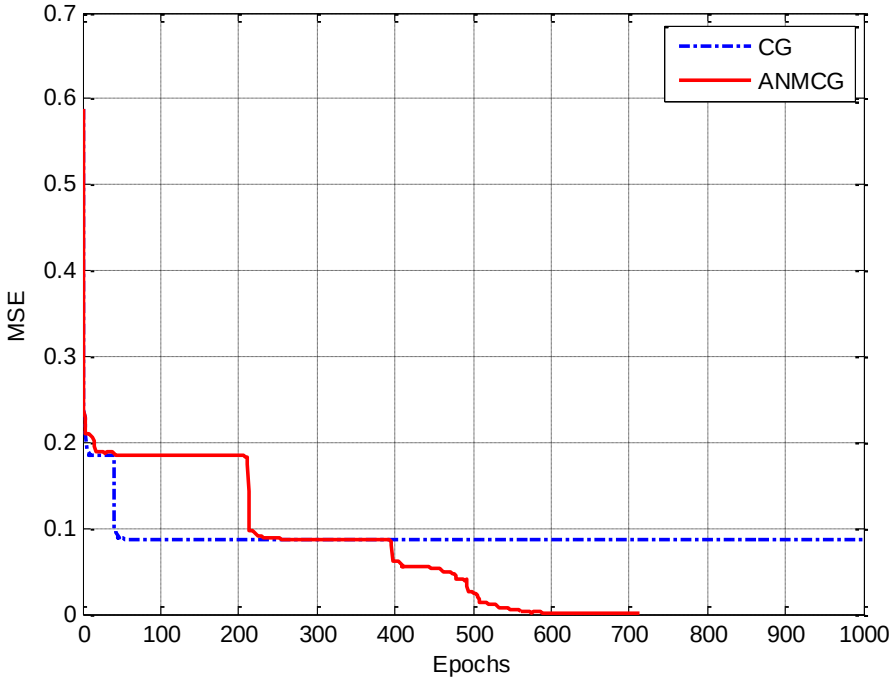
Table 5.9 shows the results; *Epo* is the average number of epochs for the converged runs out of 100 and *CE* represents the classification error, in percentage, while the two columns under *STD* (in percentage) depict the standard deviations of the training MSEs and testing CEs, respectively. The LRN performed better than the other RNN models and the A2NMCG method produced slightly lower errors in testing in all cases (the average MSE values have been calculated over the total number of runs – not only the converged ones). Examples of learning behaviours are in Figure 5.7, showing how the nonmonotone strategy helps locating minimisers with lower error function values, leading in the particular case to lower average classification error during testing (cf. with Table 5.9).

Table 5.9 Results for three RNN architectures in the SC problem: class of CG.

RNN	Algo	Conv (%)	Epo	Training		Testing	
				MSE (%)	STD (%)	CE (%)	STD (%)
FFTD	CG	0	-	22.004	3.085	34.685	7.140
	ANMCG	0	-	22.109	3.770	34.685	**7.038**
	A2NMCG	**80**	**108**	**5.016**	**0.921**	**8.789**	**2.353**
LRN	CG	0	-	14.029	4.851	20.548	3.692
	ANMCG	0	-	**13.553**	**4.730**	**18.178**	4.070
	A2NMCG	0	-	**10.756**	**3.908**	**14.685**	3.947
NARX	CG	58	470	2.816	2.493	38.986	5.924
	ANMCG	**84**	**446**	**1.435**	**1.006**	**37.466**	**4.027**
	A2NMCG	**91**	452	**0.940**	**0.237**	**36.986**	5.300



(a) LRN



(b) NARX

Figure 5.7 Examples of convergence behaviour for the CG and the ANMCG methods in the SC problem for (a) LRN and (b) NARX.

5.4.3 Sequence Learning Problem

Results are shown in Tables 5.10 and 5.11. Table 5.10 compares the performances of the algorithms using the three RNN architectures. The MSEs of the nonmonotone methods are, in all cases, better than the original CG because the nonmonotone versions appear to locate better approximations of the optimal weight set. For comparison, we should mention that the approach of the original work by McLeod et al. in 1998 is based on LRNs trained with the GD method. It uses 10 hidden nodes and produces an MSE of 25% in training and 22% in testing. Thus CG methods, in general, produce better results than GD methods in this problem.

In order to have further comparison we also performed experiments with different numbers of hidden nodes for the NARX architecture. Table 5.11 shows that promising results can be obtained using 2 or 5 hidden nodes, making the application of CG methods to this difficult dataset a promising alternative against GD methods. An example of learning behaviour is shown in Figure 5.8, for NARX networks.

Table 5.10 Results for three RNNs architectures in the SL problem: class of CG.

RNN	Algorithm	Training		Testing	
		MSE (%)	STD (%)	MSE (%)	STD (%)
FFTD	CG	18.743	8.138	18.654	9.072
	ANMCG	**17.856**	**5.854**	**17.673**	**7.110**
	A2NMCG	**15.342**	**2.708**	**15.389**	**3.205**
LRN	CG	17.266	6.877	17.286	6.919
	ANMCG	**16.392**	**4.912**	**16.405**	**4.886**
	A2NMCG	**14.732**	**2.082**	**14.681**	**1.794**
NARX	CG	18.321	7.698	18.146	7.447
	ANMCG	**15.485**	**4.001**	**15.164**	**3.592**
	A2NMCG	**10.859**	**1.171**	**11.819**	**1.341**

Table 5.11 MSEs for NARX networks in the SL problem: class of CG.

Algo	#hid	Training		Testing	
		MSE (%)	STD (%)	MSE (%)	STD (%)
CG	2	16.534	6.132	16.761	6.810
	5	16.921	6.477	17.025	6.956
	10	18.321	7.698	18.146	7.447
ANMCG	2	**16.601**	6.737	**16.462**	7.005
	5	**15.275**	**5.892**	**15.377**	**6.014**
	10	**15.485**	**4.001**	**15.164**	**3.592**
A2NMCG	2	21.951	**9.237**	22.689	10.039
	5	**13.449**	**3.206**	**14.206**	**4.836**
	10	**10.859**	**1.171**	**11.819**	**1.341**

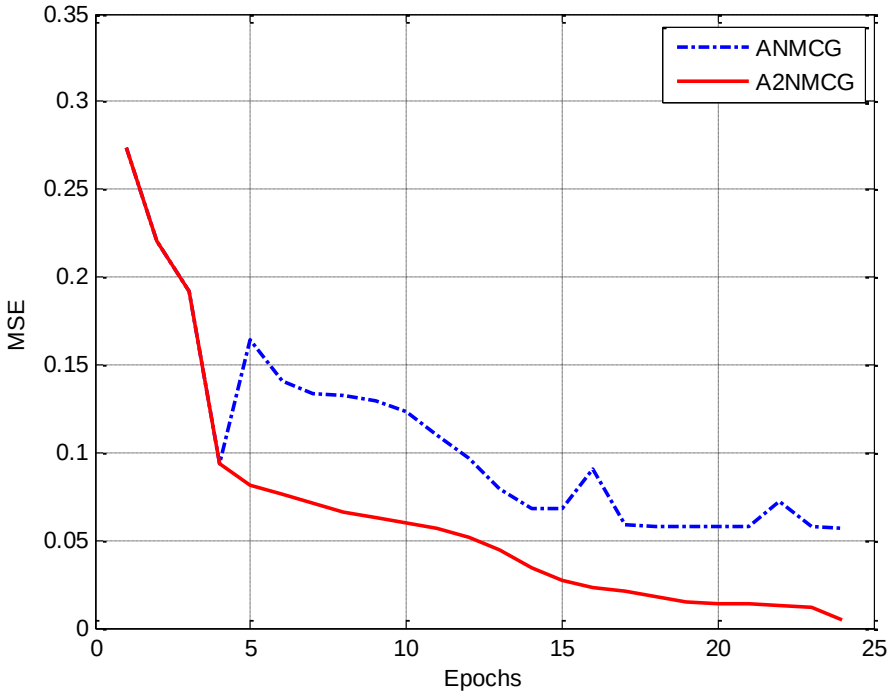


Figure 5.8 Example of convergence behaviour for the ANMCG and the A2NMCG in the SL problem for NARX networks.

In order to fairly exhibit the improved performance of the proposed nonmonotone CG algorithms, the default 23-epoch training process of the SL problem is extended to the 200- and 1000-epoch ones, as presented in Tables 5.12-5.14.

Table 5.12 Results of additional simulations for FFTD networks in the SL problem: class of CG

Algorithms	200-epoch		1000-epoch	
	Train (MSE/CE)	Test (MSE/CE)	Train (MSE/CE)	Test (MSE/CE)
CG	16.1/39.4	16.8/40.6	14.7/38.2	16.4/39.9
ANM-CG	13.9/31.3	14.5/38.2	11.3/24.7	13.9/31.2
A2NM-CG	9.4/22.4	10.6/23.8	8.6/21.4	11.1/25.4

Table 5.13 Results of additional simulations for LRN networks in the SL problem: class of CG

Algorithms	200-epoch		1000-epoch	
	Train (MSE/CE)	Test (MSE/CE)	Train (MSE/CE)	Test (MSE/CE)
CG	16.7/40.6	17.2/42.2	14.9/38.9	16.6/40.3
ANM-CG	14.4/38.2	15.9/39.7	12.6/25.7	14.3/38.0
A2NM-CG	13.5/26.9	15.5/39.1	11.0/25.5	12.9/26.0

Table 5.14 Results of additional simulations for NARX networks in the SL problem: class of CG

Algorithms	200-epoch		1000-epoch	
	Train (MSE/CE)	Test (MSE/CE)	Train (MSE/CE)	Test (MSE/CE)
CG	17.0/41.7	18.3/46.2	15.5/38.9	17.3/42.1
ANM-CG	13.4/26.2	15.8/39.6	9.3/22.6	10.9/23.2
A2NM-CG	9.1/22.0	11.3/24.8	7.6/19.4	9.2/22.6

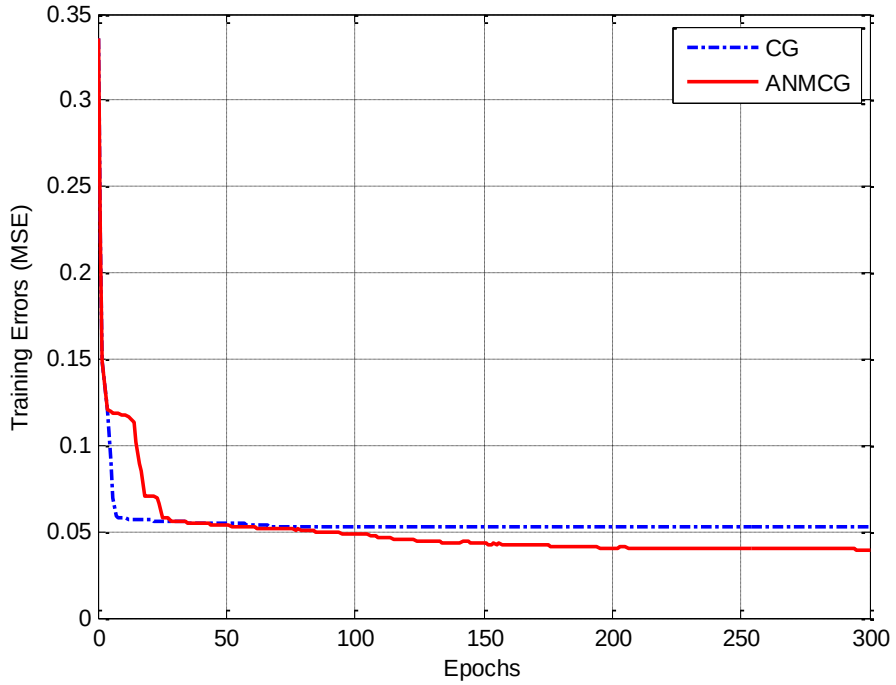
5.4.4 Reading Aloud Problem

In the work of [158] a specially designed RNN architecture with 100 hidden nodes (26582 adjustable parameters) is needed to solve this problem. Here we tried to solve it using FFTD and NARX networks. Architectures with 5 and 10 hidden nodes and 300 training epochs are used for this problem; a FFTD with 5 hidden nodes has 1421 adjustable parameters (weights plus biases), while a NARX network with 5 hidden nodes has 2031 parameters.

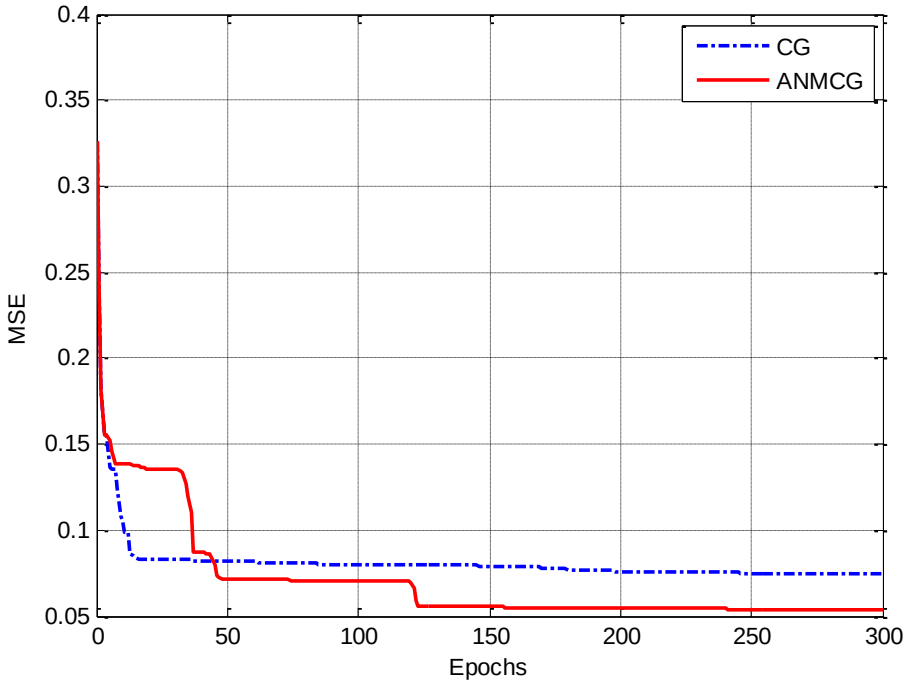
As shown in Table 5.15, our approaches are more effective than the original CG. It is noteworthy that, in the work of [158], the training process takes 1900 epochs to achieve similar results, which is about 6 times more than the results of the CG methods. Figures 5.9 and 5.10 show typical learning behaviours for this problem, and of the improvements achieved by the A2NMCG method against ANMCG.

Table 5.15 Results for two RNN architectures in the RA problem: class of CG.

RNNs	Algo	#hid	Training		Testing	
			MSE (%)	STD (%)	MSE (%)	STD (%)
FFTD	CG	5	10.500	8.892	8.378	6.993
		10	4.499	3.784	6.732	2.196
	ANMCG	5	11.899	**8.015**	**6.523**	**3.509**
		10	7.499	4.223	**3.129**	3.005
	A2NMCG	5	**10.734**	**7.483**	**5.946**	**3.007**
		10	8.197	4.829	**2.657**	**2.221**
NARX	CG	5	8.294	7.725	7.849	6.894
		10	4.796	4.108	5.824	3.003
	ANMCG	5	10.063	**6.342**	**6.057**	**4.777**
		10	6.690	**3.557**	**3.283**	**2.095**
	A2NMCG	5	**8.942**	6.590	**4.610**	**4.051**
		10	7.879	4.178	**2.182**	**1.457**

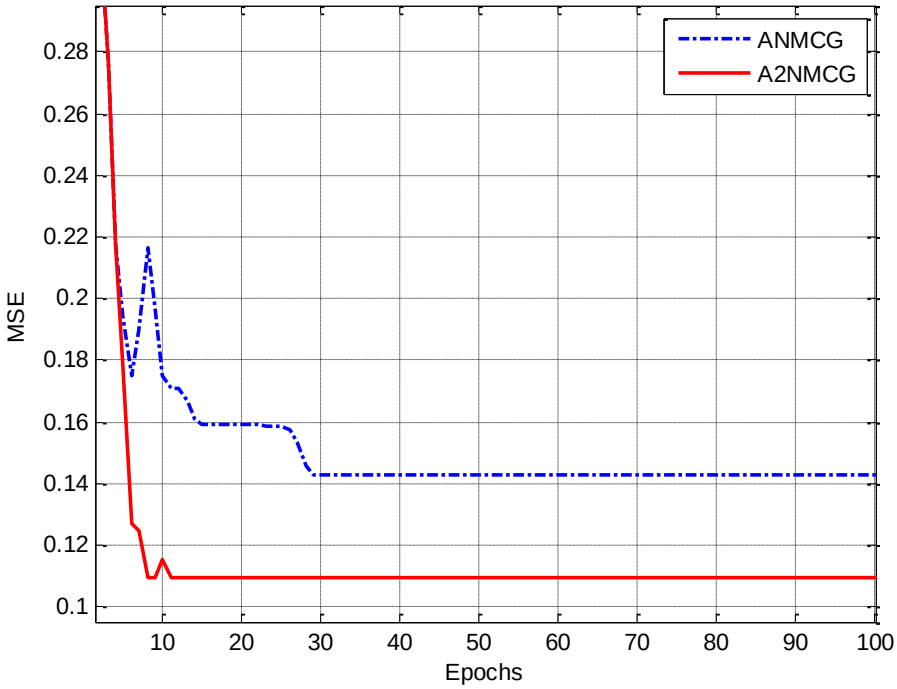


(a) FFTD

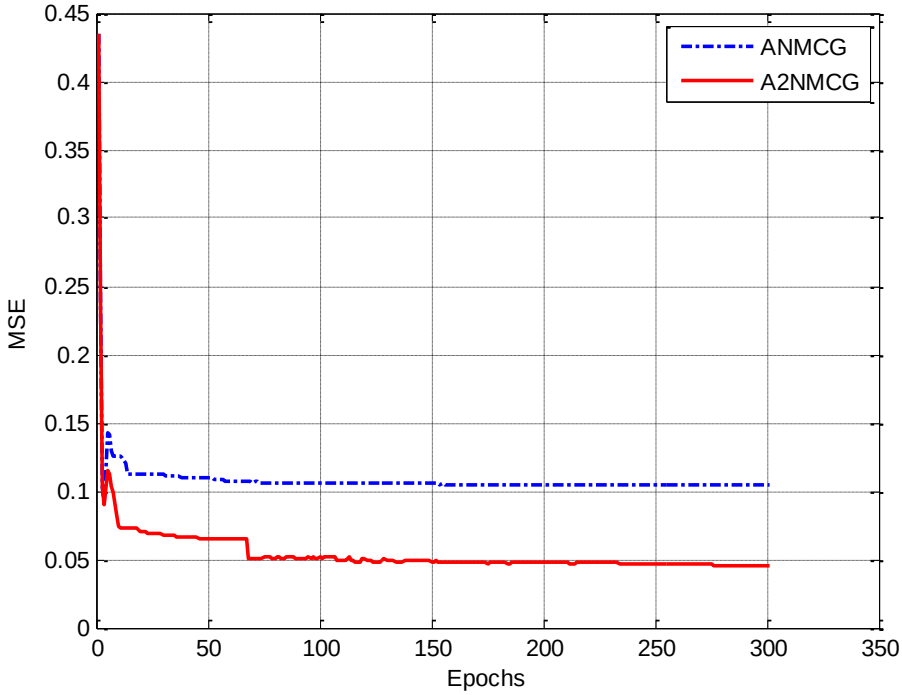


(b) NARX

Figure 5.9 Examples of convergence behaviour for the CG and the ANMCG methods n the RA problem for (a) FFTD and (b) NARX.



(a) FFTD



(b) NARX

Figure 5.10 Examples of convergence behaviour for the ANMCG and the A2NMCG methods in the RA problem for (a) FFTD and (b) NARX.

5.5 Summary and Contribution of the Chapter

In this chapter, the traditional monotone CG methods were firstly introduced and the versions of nonmonotone CG (NMCG) approaches were discussed. After the properties and theorems of global convergence for the NMCG methods in the literature were reviewed, two NMCG algorithms, i.e. ANMCG and A2NMCG (see Tables 5.1 and 5.2) were presented.

Experimental results using RNNs from three different architectures, i.e. FFTD, LRN and NARX, in five applications, i.e. P5, P10, SC, SL, and RA, showed that our proposed algorithms perform well, in terms of higher convergence rates, fewer training epochs for P5 and P10, lower training MSEs for all the five classes of simulations, smaller testing CEs for SC, and better testing MSEs for SL and RA. Graphical examples of learning behaviours for all RNNs in each application reveal the benefits of our nonmonotone algorithms.

Chapter 6

Adaptive Self-Scaling Nonmonotone BFGS Algorithm

In this chapter, starting from the introduction of quasi-Newton (QN) methods [47][48] in Section 6.1, the most well known update formulas [160][30][47] [64][67][72][157][179] of the approximated Hessian matrix and scaling techniques [4][124][138][140][141][209] are reviewed. After a discussion on the traditional monotone QN approaches, Section 6.2 focuses on cases of nonmonotone QN [79][81][86][187][212], and the relative assumptions to build theorems of global convergence. The proposed algorithm of QN class is developed in Section 6.3, while experimental results for the *N*-bit parity, SC [116], SL [124] and RA [158] applications (details are provided in Appendix A.1), using the three different RNN architectures, i.e. FFTD [196][197], LRN [57][85] and NARX [128][137] (structural topologies defined in Chapter 2), are shown in Section 6.4. Conclusions are made in Section 6.5.

6.1 Quasi-Newton Methods

In the context of deterministic unconstrained optimisation, QN methods, sometimes called variable metric methods, are well-known algorithms for finding local minima of functions in the form of Eq. (3.1). The original method was firstly proposed in 1959 by W.C. Davidon [47] and finally published in 1991 [48]. QN methods are based on Newton's method to find the stationary point of a function, where the gradient is zero. Newton's method assumes that the function can be locally approximated by a quadratic function in the region around the optimum, and requires the first and second derivatives [68], i.e. the gradient vector and the Hessian matrix, to find the stationary point. Moreover, the Newton's method and its variants require that the Hessian is positive definite - a condition that is difficult to guarantee in practice.

QN methods exploit the idea of building up curvature information as the iterations of the training method are progressing. This is achieved by using the objective function values and its gradient to estimate the Hessian matrix. Thus, a new approximated Hessian matrix B_{k+1} is required at each iteration to satisfy the QN condition $B_{k+1}s_k = y_k$, where s_k and y_k are the changes in function variable and in gradient, respectively. At the *k*-th iteration, a QN method has the following basic structure:

(1) Set $d_k = -H_k g_k$;

(2) Apply linesearch along d_k giving $w_{k+1} = w_k + \alpha_k d_k$;

(3) Update H_k giving H_{k+1};

where d is the search direction, *H* is the Hessian approximation, *g* denotes the first derivative, and α the stepsize. The initial *H* is any given $n \times n$ symmetric positive definite matrix, and $H_k = B_k^{-1}$.

Some of the most famous approaches for updating B_{k+1} are the Powell-Symmetric-Broyden (PSB) update equation [160]:

$$B_{k+1}^{PSB} = B_{k+1} + \frac{1}{s_k^T s_k}\left(\left(y_k - B_k s_k\right) s_k^T + s_k \left(y_k - B_k s_k\right)^T\right) - \frac{\left(y_k - B_k s_k\right)^T s_k}{\left(s_k^T s_k\right)^2} s_k s_k^T, \quad (6.1)$$

the Davidon-Fletcher-Powell (DFP) formula [47][64]:

$$\begin{aligned} B_{k+1}^{DFP} &= B_{k+1} + \frac{1}{s_k^T B_k s_k} B_k s_k s_k^T B_k + \frac{1}{y_k^T s_k} y_k y_k^T + \left(s_k^T B_k s_k\right) r_k r_k^T, \\ r_k &= \frac{1}{y_k^T s_k} y_k - \frac{1}{s_k^T B_k s_k} B_k s_k \end{aligned} \quad (6.2)$$

the Broyden-Fletcher-Goldfarb-Shanno (BFGS) [30][67][72][179]:

$$B_{k+1}^{BFGS} = B_{k+1} - \frac{1}{s_k^T B_k s_k} B_k s_k s_k^T B_k + \frac{1}{y_k^T s_k} y_k y_k^T, \quad (6.3)$$

and the Broyden's class of methods, which uses a linear combination of the DFP and the BFGS updates:

$$B_{k+1}^{Broyden} = \left(1-\xi\right) B_{k+1}^{BFGS} + \xi B_{k+1}^{DFP}, \xi \in \left[0,1\right], \quad (6.4)$$

while the most commonly used update technique for training neural networks is the BFGS [63].

There are relatively large number of applications, where QN methods have been

used for training static Artificial Neural Networks (ANNs), e.g. [1][6][23][77][80] [94][98][110][111][126][127][156][176][177][183][186][193][209].

As mentioned in [91][176][177] the QN method for training ANNs is very fast to converge but in many cases it converges to a local minimum. Although several efforts have been made to reduce the memory requirement of updating the Hessian approximation [17][95][111][126][170][193], the need of using a monotone line search and the drawback of getting trapped in neighbourhoods of local minimum points limit the application of these methods in real-world applications. Another problem in neural networks applications is that QN methods suffer from large eigenvalues in the approximated Hessian matrices of the objective function as Powell discovered in 1986.

Despite the emergence of the self-scaling approaches for the Hessian approximation in the field of numerical optimisation [209] (the fundamental concept of self-scaling is to accommodate the change of target variables efficiently), self-scaling is rarely introduced when training ANNs [136]. In addition, the literature of RNNs includes only very few attempts to train RNNs using QN methods with limited results [17][21][22][54][91][103].

The self-scaling techniques can resolve problems caused by larger eigenvalues by scaling the Hessian approximation before it is updated at each iteration to keep the eigenvalues of the approximated Hessian matrix within a suitable range [209]. This technique was first proposed in [140][141], and was used to update the approximated Hessian as follows:

$$B_{k+1}^{Oren} = \left(B_k - \frac{B_k y_k y_k^T B_k}{y_k^T B_k y_k} + \theta v v^T \right) \gamma + \frac{p p^T}{p^T y_k}, \tag{6.5}$$

where

$$p = -\alpha B_k g, \tag{6.6}$$

$$v = \left(y_k^T B_k y_k \right)^{\frac{1}{2}} \left(\frac{p}{p^T y_k} - \frac{B_k y_k}{y_k^T B_k y_k} \right), \tag{6.7}$$

$$\gamma = \varphi \frac{g' p}{g' B_k y_k} + (1-\varphi) \frac{p^T y_k}{y_k^T B_k y_k}, \tag{6.8}$$

and $\varphi, \theta \in [0,1]$. After these attempts, more relative works have been developed, such as in [4][138].

In our approach, presented in detail in the following sections, we use the scaling factor ρ_k which was introduced for the BFGS method by [209]. This is defined as

$$B_{k+1}^{SCBFGS} = \rho_k \left[B_k - \frac{B_k s_k s_k B_k^T}{s_k^T B_k s_k} \right] + \frac{y_k y_k^T}{y_k^T s_k}, \tag{6.9}$$

where

$$\rho_k = \frac{y_k^T s_k}{s_k^T B_k s_k}. \tag{6.10}$$

Numerical evidence has shown that methods that apply a scaling factor for B_{k+1} are superior to the original QN methods. Especially in real-world applications the

scaling factor could potentially play an important role: when ρ_k is sufficiently large, the eigenvalues of B_{k+1} are relative small, with strong self-correcting property [126][209]. Despite this looks particularly appealing for training RNNs, to the best of our knowledge it has not been exploited at all in this area to improve the effectiveness of second-order training algorithms. Another useful characteristic of the factor ρ_k, which makes it useful in RNN training, is that it takes only the information of the most current point to scale the Hessian approximation and no user-defined parameters, compared to the factor γ in Eq. (6.8). This is particularly helpful when dealing with high-dimensional search spaces, such as the ones encountered in the applications discussed in this thesis.

6.2 Global Convergence

From a deterministic optimisation perspective, nonmonotonicity can be introduced through conditions, such as those initially proposed by Grippo et. al. [80], for finding a stepsize that occasionally permits an increase in the function value while retaining global convergence of the minimisation method:

$$E\left(w_k+\alpha_k d_k\right) \le \max_{0\le j\le m(k)}\left[E(w_{k-j})\right]+\varepsilon_1\alpha_k g_k^T d_k, \tag{6.11}$$

and

$$g\left(w_k+\alpha_k d_k\right)^T d_k \le -\varepsilon_2 g_k^T d_k, \tag{6.12}$$

where $0<\varepsilon_1<\varepsilon_2<\frac{1}{2}, m(0)=0,$

$$m(k) = \min\{m(k-1)+1, M\}, \tag{6.13}$$

As discussed in Chapter 3, the parameter $m(k)$ plays the role of a memory element, or buffer, and is typically a non-decreasing integer (cf. with Eq. (6.13)), bounded by a nonnegative prefixed integer M. Another approach proposed recently is to replace *max* in Eq. (6.11) by an average of function values [212]. Lastly, Grippo et. al. [81] proposed the use of a slightly different approach that employs the following condition instead of Eq. (6.12)

$$\|d_k\| \le \alpha\beta^h, \tag{6.14}$$

where $\alpha > 0$ and $\beta \in (0,1)$ are user-defined real numbers and h is an integer that increases by one unit whenever the condition is satisfied. If Eq. (6.14) is satisfied $\alpha_k = 1$ and the new point, w_{k+1} is accepted without evaluating the objective function. If Eq. (6.14) does not hold then Eq. (6.11) is used to determine the stepsize α_k.

Grippo et al. also proved the following theorem that describes the convergence properties of quasi-Newton algorithms that adopt a nonmonotone strategy.

Theorem 6.1 [81]. *Let w_k be a sequence produced by an iterative scheme of the form $w_{k+1} = w_k + \alpha_k d_k$, where the stepsize α_k is computed by Eqs. (6.11) and (6.14). Assume that:*

***(H6.1.1)** the level set $\Phi = \{w | E(w) \le E(w_0)\}$ is compact.*

(H6.1.2) *positive numbers* ε, c, p_1 *and* p_2 *exist such that the following conditions hold:*

$$g_k^T d_k \le -\varepsilon \|g_k\|^{p_1},$$

$$\|d_k\|^{p_2} \le c\|g_k\|.$$

Then either the algorithm terminates at some point w_μ *such that* $g(w_\mu) = 0,$ *or it generates an infinite sequence such that:*

(1). *the sequence* $\{w_k\}$ *remains in a compact set and every limit point* w^* *belongs to the level set and satisfies* $g(w^*) = 0;$

(2). *no limit point of* $\{w_k\}$ *is a local maximum of* $E;$

(3). *if the number of stationary points of* E *in* Φ *is finite or there exists a limit point where* H *is nonsingular, the sequence* $\{w_k\}$ *converges.*

Although Grippo et al. [81] do not directly make any assumptions about the convexity of the objective function, they assume that the search direction is computed by minimising a quadratic approximation of the objective function at the current point. Furthermore, they demonstrate that this scheme works well even when the search direction is computed approximately by means of a truncated quasi-Newton algorithm with finite difference approximations of second-order derivatives.

Theorem 6.1 can be specialised to algorithms of the Newton class, such as those employing the updates defined in Eqs. (6.1)-(6.4), by imposing appropriate form in the conditions **H6.1.2**. Thus, when the search direction is defined by

$$d_k = -B_k^{-1} g_k,$$

and $\{B_k\}$ is a sequence of symmetric positive definite matrices with uniformly bounded eigenvalues $\lambda(B_k)$, i.e. there exist λ, Λ such that for all k:

$$0 < \lambda \leq \lambda_i(B_k) \leq \Lambda.$$

Then

$$g_k^T d_k \leq -\Lambda^{-1} \|g_k\|^2,$$

$$\|d_k\| \leq \lambda^{-1} \|g_k\|.$$

Since RNNs' error functions are nonconvex, we present and discuss below the main theoretical results for global convergence of nonmonotone BFGS methods that hold in this case. It is worth mentioning that proving global convergence for nonconvex objective functions is a very challenging problem that has not been explored totally yet. Also it is important to distinguish between the notion of *global convergence* and that of *global optimisation*: a globally convergent algorithm always reaches a minimiser (not necessarily the global minimiser) starting from almost any initial weight [118]. Here we are based on the work of Yin and Du [209] which applies the nonmonotone technique of Han and Liu [86] stated as follows:

$$E\left(w_k+\alpha_k d_k\right)\le \max_{0\le j\le m(k)}\left[E(w_{k-j})\right]-\delta\min\left\{\sigma_1\left(\mu_k\right),\sigma_2\left(\nu_k\right)\right\}, \tag{6.15}$$

and

$$g\left(w_k+\alpha_k d_k\right)^T d_k\ge\beta g_k^T d_k, \tag{6.16}$$

where

$$\mu_k=-\frac{g_k^T d_k}{\left\|d_k\right\|}, \tag{6.17}$$

$\nu_k=-\lambda g_k^T d_k$, $0<\delta<\beta<1$, and σ_1 and σ_2 are two forcing functions, which are used to measure the sufficiency of descent and prove convergence. As shown in [86], Eqs. (6.15) and (6.16) formulate one of the most general types of line search, which has as special cases many monotone and nonmonotone techniques. Furthermore, Sun et al. [187] have shown that the nonmonotone Armijo rule, the nonmonotone Goldstein rule, and the nonmonotone Wolfe rule employ special forms of forcing functions.

Before presenting the main theorem for global convergence, the following Assumptions (**H6.2**) are needed.

(**H6.2.1**) The level set $\Psi=\left\{w\in\Re^n : E\left(w\right)\le E\left(w_0\right)\right\}$ is bounded.

(**H6.2.2**) In some neighbourhood $\aleph\left(\Psi\right)$ of Ψ, the gradient of $E\left(w\right)$, $g\left(w\right)$ is Lipschitz continuous, that is, there exists a constant $L>0$ such that for all w,

$\bar{w} \in \aleph(\Psi)$,

$$\left\| g(w) - g(\bar{w}) \right\| \leq L \left\| w - \bar{w} \right\|. \tag{6.18}$$

Below we present the theorem of Yin and Du that needs Assumptions (**H6.2**). It makes use of the BFGS property to generate positive definite matrices B_k^{BFGS} [68], and exploits their own result that shows the update Equation (6.9) preserves the positive definiteness of the matrices B_k^{SCBFGS}.

Theorem 4.2 [209]. *Suppose that Assumptions (**H6.2**) hold, and let us assume that* w_0 *is any starting point,* B_0 *is any symmetric positive definite matrix, and that the sequence* $\{w_k\}$ *is generated by the iterative scheme, i.e.* $w_{k+1} = w_k + \alpha_k d_k$, *where* $d_k = -\left(B_k^{SCBFGS}\right)^{-1} g_k$, *and the stepsize* α_k *is determined by Eqs. (6.15) and (6.16). If there exists a positive constant* $K \geq 1$ *for which*

$$\left\| \hat{y}_k \right\| \leq (1 - \beta) \left\| g_k \right\|$$

for all $k \geq K$, *then*

$$\liminf_{k \to \infty} \left\| g_k \right\| = 0. \tag{6.19}$$

Assumption (**H6.2.1**) holds for training RNNs of a fixed architecture on a finite set of training patterns because the error function is bounded below in $\Re^n$ since $E \geq 0$: if a w^* exists such that $E(w^*) = 0$ then w^* is the global minimum; otherwise the vector w with the smallest available value is the global minimiser.

Assumption (**H6.2.2**) also holds for RNNs that use smooth enough activation functions (the derivatives of order p are available and continuous), such as the logistic function that is used in our experiments later in the chapter. Moreover, **H6.2.2** implies that there exists a constant c such that $\|g_k\| \leq c \quad \forall w \in \aleph(\Psi)$. A detailed proof is provided in [209], which shows that the limit of Eq. (6.19) is the best type of global convergence result that can achieved for nonconvex functions.

6.3 Our Proposed Algorithm

In this section we present the proposed algorithm, named *Adaptive Self-scaling Non-Monotone BFGS* (ASCNM-BFGS), through the high-level description presented below.

Table 6.1 Adaptive Self-scaling Non-monotone BFGS Algorithm

Algorithm: ASCNM-BFGS

STEP 0. Initialise $w_0, k=0,$ a symmetric positive definite matrix B_0, M_0, $M^{\max}$ (boundary of nonmonotone learning horizon M_k), $\alpha_0 \in (\lambda_1, \lambda_2)$, $0<\lambda_1<\lambda_2$ are positive constants, $d_0 = -g(w_0)$ and $\sigma, \delta \in (0,1)$;

STEP 1. If $g_k = 0,$ stop;

STEP 2. If $k \geq 1,$ calculate a local approximation of the Lipschitz constant $\Lambda_k = \|g_k - g_{k-1}\| / \|w_k - w_{k-1}\|$ and adapt M_k by the following scheme:

$$M_k = \begin{cases} M_{k-1}+1, & \text{if } \Lambda_k < \Lambda_{k-1} < \Lambda_{k-2} \\ M_{k-1}-1, & \text{if } \Lambda_k > \Lambda_{k-1} > \Lambda_{k-2}, \\ M_{k-1}, & \text{otherwise,} \end{cases}$$

where $M_k = \min\{M_k, M^{\max}\}$;

STEP 3. $\forall k \geq 1,$ set $\alpha_k = \max\{\alpha_{k-1}, \bar{\alpha}_k\}$, where $\bar{\alpha}_k = \dfrac{2|E_k - E_{k-1}|}{g^T w_k d_k}$, and check that α_k satisfies the nonmonotone condition

$$E(w_k + \alpha_k d_k) \leq \max_{0 \leq j \leq M_k} [E(w_{k-j})] + \delta \cdot \alpha_k \cdot g_k^T \cdot d_k;$$

otherwise, find stepsize $\alpha_k = \alpha_k \bullet \sigma^{l_q}$ that satisfies the above condition, setting each time $l_q = l_q + 1$;

STEP 4. Generate a new weight vector $w_{k+1} = w_k + \alpha_k d_k$;

STEP 5. Update the search direction $d_k = -B_k^{-1} g_k,$ using the Hessian approximation B_k calculated by the self-scaling BFGS formula

$$B_{k+1} = \rho_k \left[B_k - \frac{B_k s_k s_k B_k^T}{s_k^T B_k s_k} \right] + \frac{y_k y_k^T}{y_k^T s_k},$$

where $s_k = w_{k+1} - w_k$, $y_k = g_{k+1} - g_k$ and $\rho_k = \dfrac{y_k^T s_k}{s_k^T B_k s_k}$;

STEP 6. Let $k = k+1$ and $l_q = 0,$ go to STEP 1.

A feature of the ASCNM-BFGS method is the use of an adaptive memory term M_k, called *nonmonotone learning horizon*, instead of a fixed heuristic value. To this end, it calculates in Step 2 a local estimation of the Lipschitz constant, which could provide helpful information on the morphology of a function, and uses it to

automatically adapt the size of M. The local estimation of the Lipschitz constant gets large values in steep regions of search space and small values in flat areas. At the beginning, i.e. $k<3$, there is not enough information to adapt M through the local estimation of the Lipschitz, and as a result the nonmonotone conditions in Step 3 actually operates as a monotone one comparing the new function value against the previous one.

Also the initial choice of the stepsize merits some attention. At $k=0$ the stepsize is an arbitrary positive real number randomly chosen in the interval (λ_1,λ_2) and the algorithm operates in the direction of the negative of the gradient, $d_0=-g(w_0)$. That quickly changes, $\forall k\geq 1$, as the search direction is updated through the self-scaling BFGS update equation, which tunes the Hessian approximations at every iteration the eigenvalues possess large values; when ρ_k is sufficiently large, then the eigenvalues of B_k^{SCBFGS} are small. The stepsize is then initialised through $\bar{\alpha}$ following a technique suggested by Charalambous [39], and constantly tuned to ensure that, whilst it is not smaller than the stepsize of the previous iteration, it satisfies the nonmonotone condition in Step 3. This condition regulates the sufficient decrease of the error function through the forcing function $+\delta\bullet\alpha_k\bullet g^T(w_k)\bullet d_k$, whilst for $k=1,2$ this condition is reduced to the monotone Armijo rule (cf. with Theorem 2, [119]).

The algorithm also employs some heuristic parameters: an upper bound for M_k to help the algorithm concentrate on the recent past, while, in Step 4, σ regulates the stepsize, i.e. the larger σ the smaller trial stepsize is used, while δ controls the

amount of change. The error function E is calculated through the Mean Squared Error (MSE) formula, while the gradient is calculated using the Backpropagation-through-time (BPTT) formulae [18].

To illustrate the behaviour of the method we provide below some examples of convergence behaviour from learning the parity-5 problem, [176], using RNNs of the three types discussed above, namely the FFTD network, the LRN and the NARX network, where 7 hidden nodes were used as listed in Table A.1. Figures 6.1-6.3 illustrate the behaviour of the MSE, the stepsize, the value of M, and the scaling factor. Despite the nonmonotone behaviour that one can observe in the MSE values, it is clear that there is a trend toward smaller learning errors, whilst in all cases, the use of adaptive M does not affect the convergent behaviours of the method. The scaling factor behaviour indicates the self-correcting property of the method, which results in smaller eigenvalues for B_k^{SCBFGS} for relatively larger ρ values.

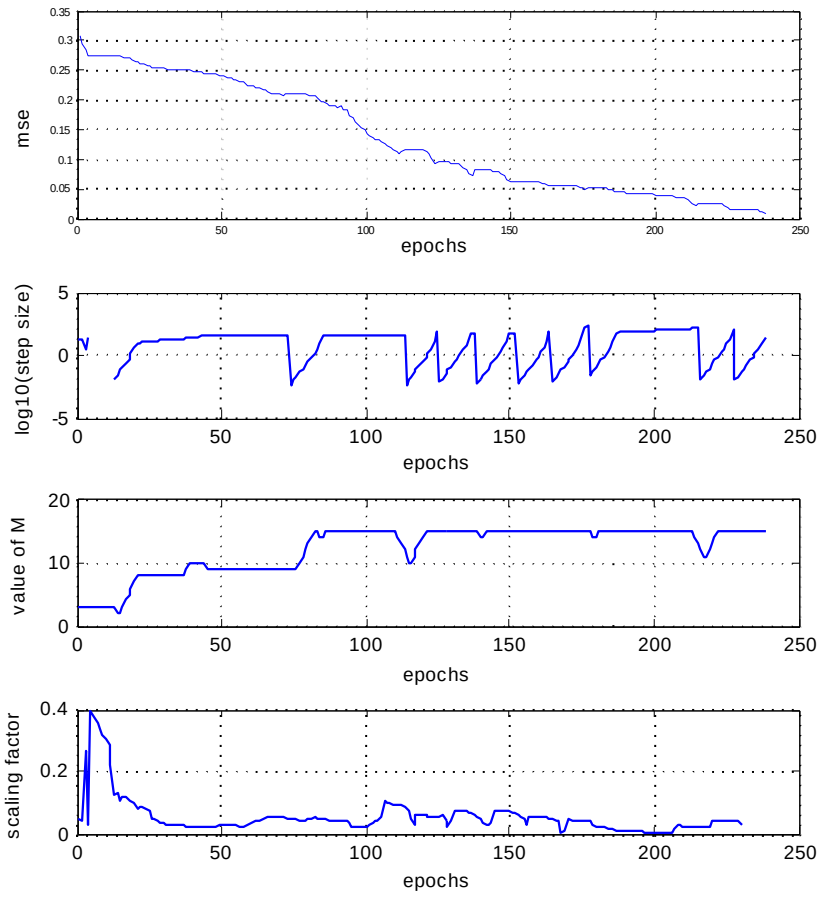


Figure 6.1 Convergence behaviours of P5: FFTD, trained by ASCNM-BFGS

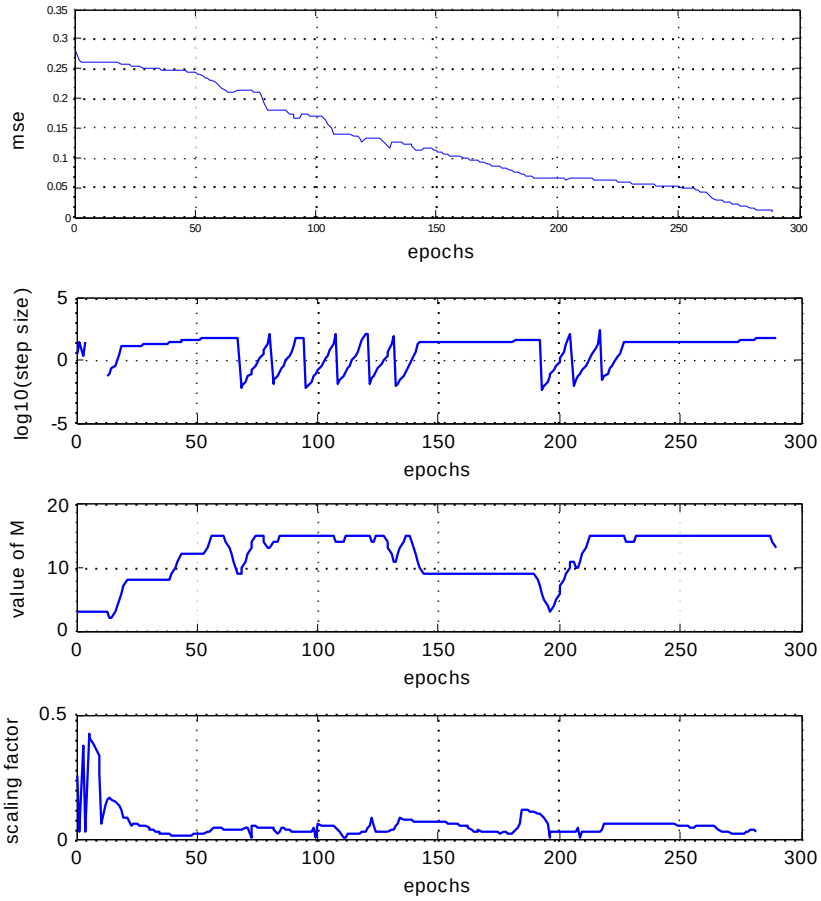


Figure 6.2 Convergence behaviours of P5: LRN, trained by ASCNM-BFGS

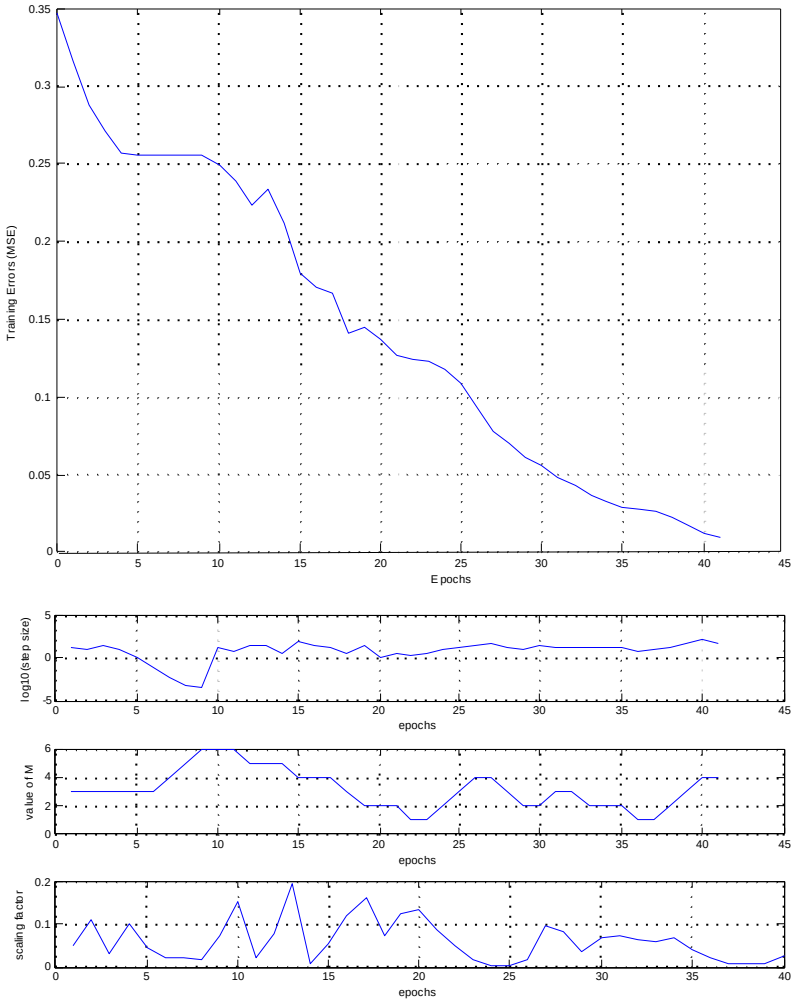


Figure 6.3 Convergence behaviours of P5: NARX, trained by ASCNM-BFGS

6.4 Experimental Results

As mentioned in Chapter 4, all settings of the simulations in the thesis are the same and are provided in Appendix (A.1), e.g. types of RNNs, amounts of hidden nodes, relative delays, boundaries of learning horizon *M*, and the constant δ used for the nonmonotone linesearch. The notations used in the following Tables have been explained in Section 4.4, e.g. a dash indicates that the algorithm did not converge

within the predefined iterations limit.

6.4.1 The N-Bit Parity Problems

The numerical results of the P5 and P10 problems are shown in Tables 6.2-6.4, and 6.5-6.7, respectively, while examples of learning behaviours, where 7 hidden nodes are used for P5 and 10 nodes for P10, are provided in Figures 6.4 and 6.5. More details of experimental parameters are presented in Appendix A.1.

Table 6.2 Average performance for FFTD networks in the P5 problem: class of BFGS

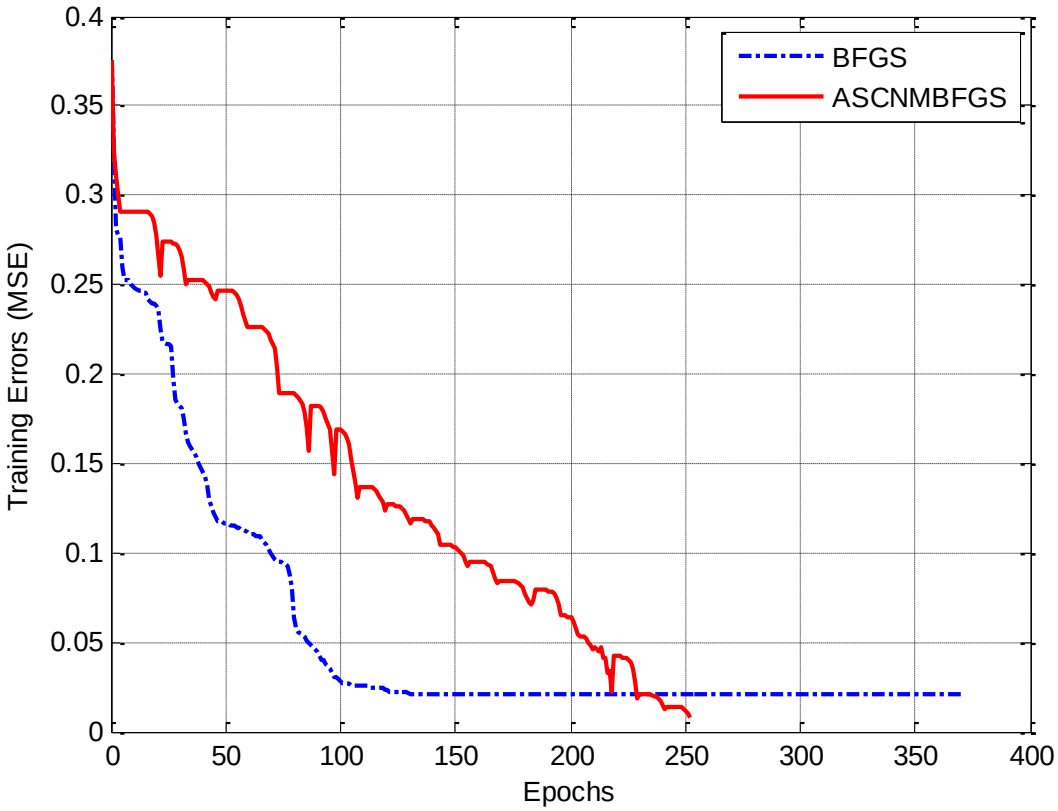
Algorithm	#hid	Conv (%)	MSE (%)	STD (%)	Epochs			
					Ave	Min	Max	Std
BFGS	1	0	23.825	7.647	-	-	-	-
	2	0	20.862	6.130	-	-	-	-
	5	4	9.229	3.583	1933	53	1171	331
	7	19	5.131	1.706	1679	33	1719	682
ASCNM-BFGS	**1**	**0**	**23.794**	**6.833**	**-**	**-**	**-**	**-**
	2	**0**	**17.487**	**5.028**	**-**	**-**	**-**	**-**
	5	**30**	**3.336**	**1.584**	**1616**	**52**	**1974**	**653**
	7	**74**	**1.772**	**0.594**	**803**	**61**	**1983**	**327**

Table 6.3 Average performance for LRN in the P5 problem: class of BFGS

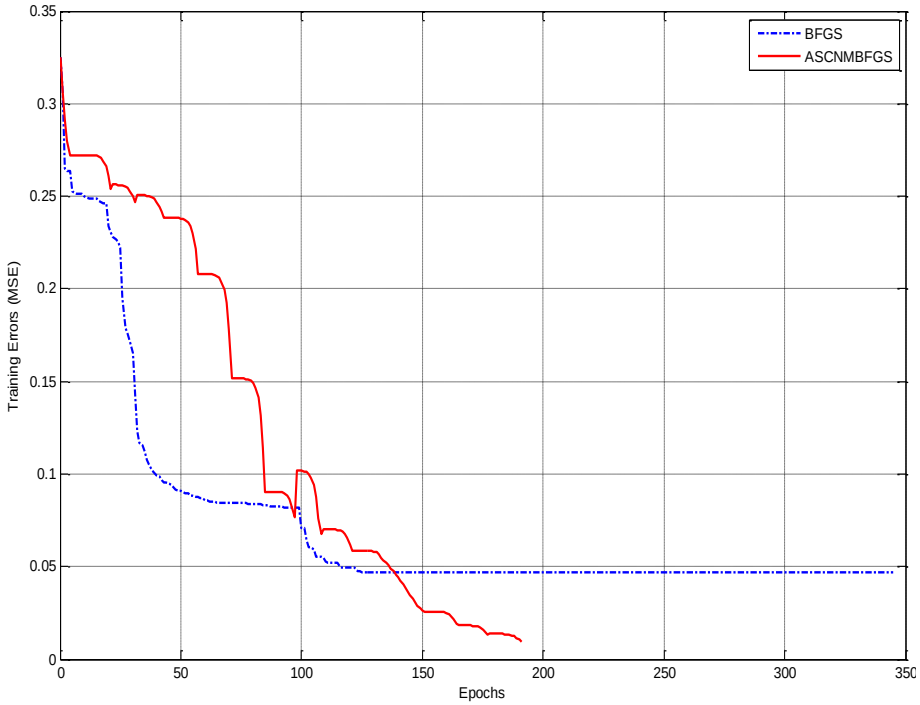
Algorithm	#hid	Conv (%)	MSE (%)	STD (%)	Epochs			
					Ave	Min	Max	Std
BFGS	1	0	23.195	8.327	-	-	-	-
	2	0	19.978	6.720	-	-	-	-
	5	5	9.205	4.579	1928	53	1568	339
	7	15	5.635	2.852	1173	34	1927	573
ASCNM-BFGS	**1**	**0**	**22.849**	**6.558**	**-**	**-**	**-**	**-**
	2	**0**	**17.699**	**5.061**	**-**	**-**	**-**	**-**
	5	**30**	**3.316**	**1.915**	**1617**	**54**	**1978**	**652**
	7	**77**	**2.029**	**1.783**	**758**	**63**	**1951**	**762**

Table 6.4 Average performance for NARX in the P5 problem: class of BFGS

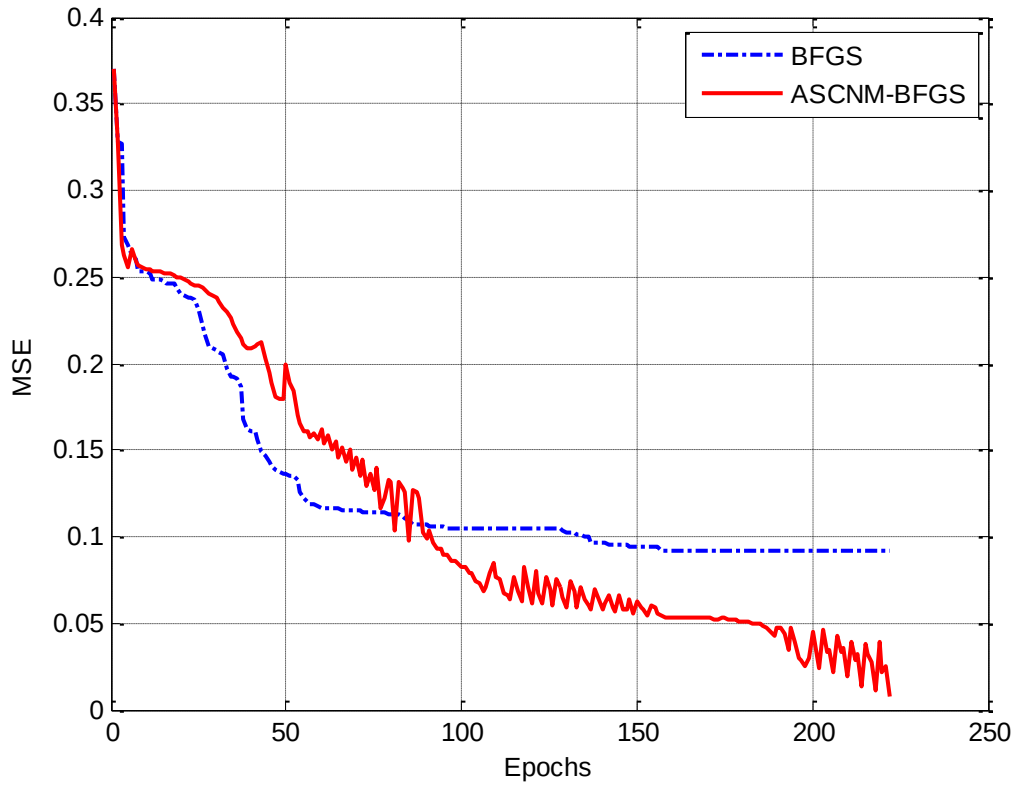
Algorithm	#Hid.	Conv (%)	MSE (%)	STD (%)	Epochs			
					Ave	Min	Max	Std
BFGS	1	15	24.979	6.004	903	254	1983	1003
	2	31	11.370	4.137	688	121	1537	892
	5	59	3.862	1.662	373	72	1244	538
	7	68	1.979	0.721	146	63	893	301
ASCNM-BFGS	**1**	**100**	**0.551**	**0.214**	**14**	**3**	**62**	**10**
	2	**100**	**0.586**	**0.470**	**17**	**5**	**53**	**10**
	5	**100**	**0.543**	**0.333**	**16**	**3**	**39**	**6**
	7	**100**	**0.595**	**0.403**	**15**	**5**	**33**	**5**


BFGS
ASCNMBFGS
Training Errors (MSE)
Epochs
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
50
100
150
200
250
300
350
400

(a) FFTD


BFGS
ASCNMBFGS
Training Errors (MSE)
Epochs
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
50
100
150
200
250
300
350

(b) LRN



(c) NARX

Figure 6.4 Examples of learning behaviours of 3 RNNs for the P5 problem, BFGS vs. ASCNMBFGS: (a) FFTD, (b) LRN and (c) NARX

As shown in Tables 6.2-6.4, the performance of the new method for the P5 problem employing three different neural architectures, i.e. FFTD, LRN and NARX, using 1, 2, 5 or 7 hidden nodes is always better than the original BFGS. For example, BFGS-trained NARX networks using 5 hidden nodes converged in 59 out of 100 runs (see Table 6.4), exhibiting a 100-run average MSE of 0.09229, while the proposed method reaches 100% convergence rate with improvements that are 6 times better in terms of MSE, 22 times smaller in the average and minimum number of training epochs, 30 times smaller in the maximum number of training epochs, and a 88 times smaller in the value of standard deviation for the converged runs.

Table 6.5 Average performance for FFTD in the P10 problem: class of BFGS

Algorithm	#hid	Conv (%)	MSE (%)	STD (%)	Epoch			
					Ave	Min	Max	Std
BFGS	1	0	24.977	7.599	-	-	-	-
	2	0	23.307	6.736	-	-	-	-
	5	0	14.067	4.325	-	-	-	-
	7	4	9.612	5.827	2022	1047	3091	859
	10	12	6.428	3.283	3697	327	3949	912
ASCNM-BFGS	**1**	**0**	**24.925**	**6.382**	**-**	**-**	**-**	**-**
	2	0	23.349	6.931	**-**	**-**	**-**	**-**
	5	**1**	**11.253**	**2.966**	**1520**	**1520**	**1520**	**0**
	7	**15**	**2.938**	**1.991**	**1848**	**711**	**3743**	**856**
	10	**57**	**1.786**	**0.824**	**2798**	**671**	**3983**	**1198**

Table 6.6 Average performance for LRN in the P10 problem: class of BFGS

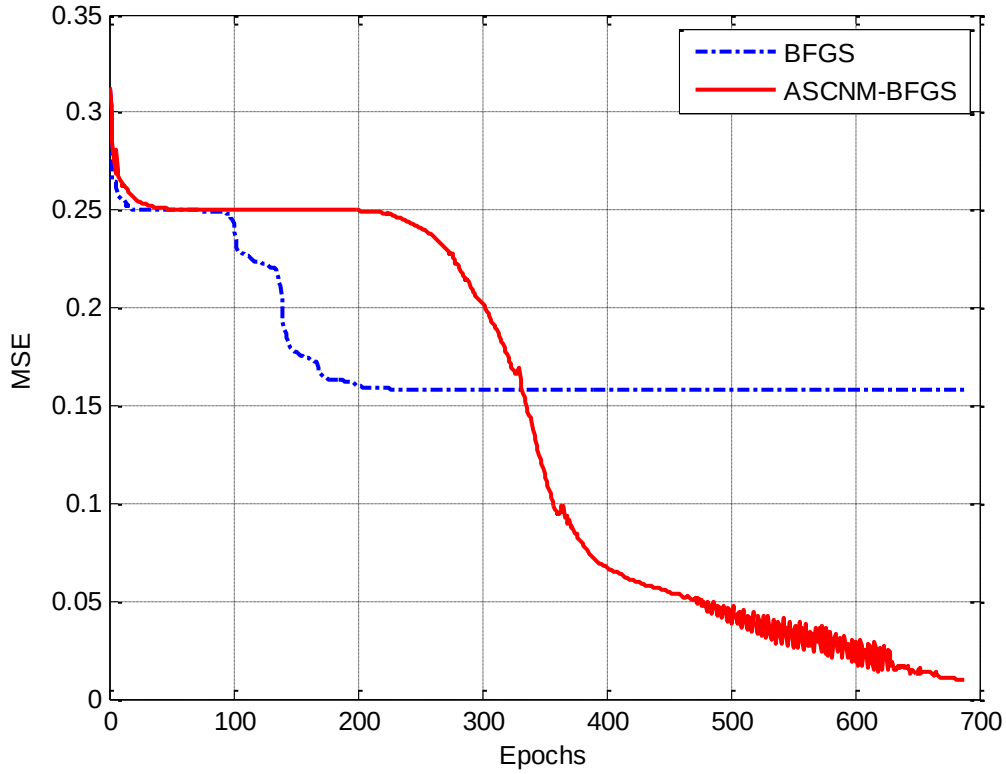
Algorithm	#hid	Conv (%)	MSE (%)	STD (%)	Epoch			
					Ave	Min	Max	Std
BFGS	1	0	24.862	8.126	-	-	-	-
	10	0	5.873	3.463	-	-	-	-
ASCNM-BFGS	**1**	**0**	**24.794**	**6.210**	**-**	**-**	**-**	**-**
	10	**100**	**0.964**	**0.026**	**1736**	**889**	**1970**	**841**

Table 6.7 Average performance for NARX in the P10 problem: class of BFGS

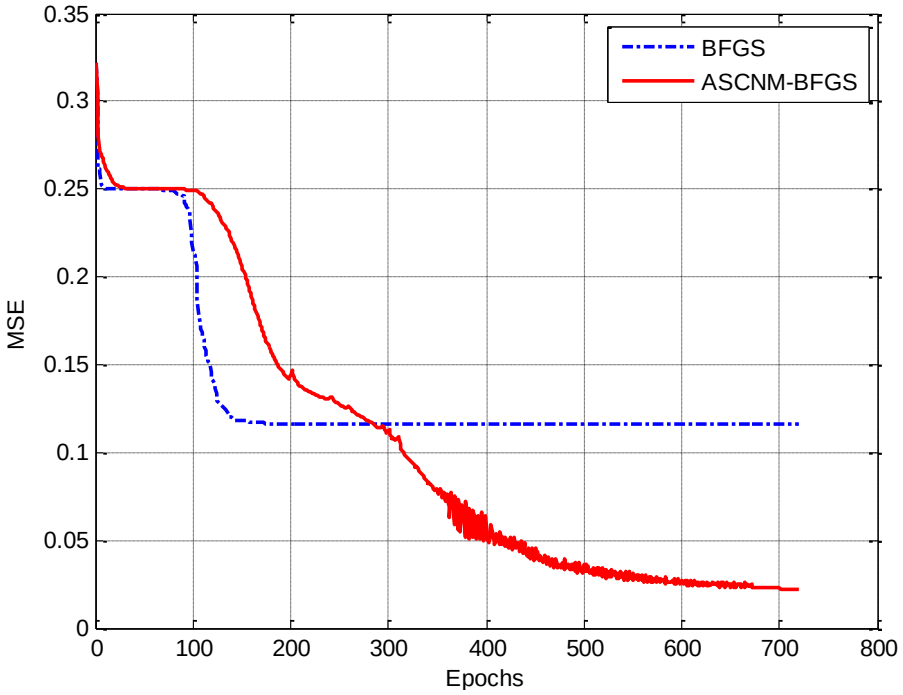
Algorithm	#hid	Conv (%)	MSE (%)	STD (%)	Epochs			
					Ave	Min	Max	Std
BFGS	1	0	24.998	8.037	-	-	-	-
	2	0	17.632	5.440	-	-	-	-
	5	34	7.659	3.131	1352	591	3439	1387
	7	52	4.205	1.984	855	273	2155	601
	10	67	3.837	2.003	492	107	1563	489
ASCNM-BFGS	**1**	**100**	**0.788**	**0.136**	**13**	**4**	**121**	**12**
	2	**100**	**0.739**	**0.147**	**21**	**6**	**91**	**14**
	5	**100**	**0.742**	**0.230**	**18**	**6**	**40**	**6**
	7	**100**	**0.726**	**0.177**	**17**	**5**	**30**	**5**
	10	**100**	**0.730**	**0.206**	**18**	**5**	**28**	**4**

In general, experimental results in Tables 6.2-6.7 provide evidence that the new method is able to locate minimisers with smaller function values than the original method, which is important in certain real-world problems to provide good generalisation. For example, in Table 6.5, 12% of the BFGS-trained FFTD networks reached an MSE=0.01 in a maximum of 3949 epochs, while the average MSE achieved by BFGS in that case was 0.06428. That was caused by the fact that the majority of the BFGS-trained networks did not reach the MSE goal within 4000 training epochs; some of them stuck to minima with higher function values while others failed to converge because of instabilities in the Hessian. When BFGS fails to reach the error goal we only provide the average error obtained. Also a 0%

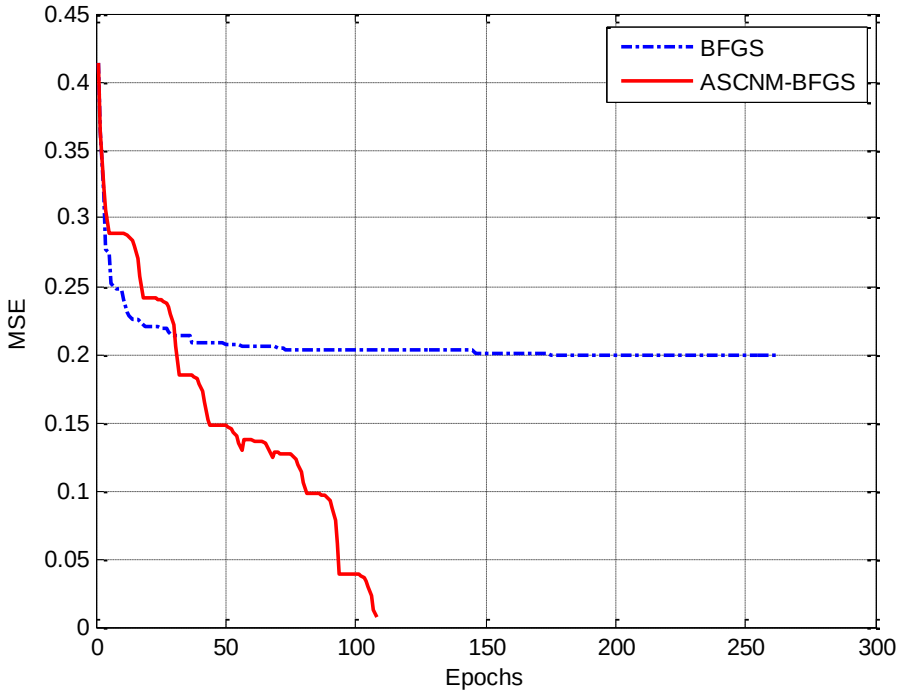
convergence in Tables 6.2-6.3 and 6.5-6.7 indicates that not a single run of the BFGS method converged within the predefined number of epochs, and since only epochs of the converged runs are reported, we enter the symbol "-" in the corresponding cells. We observed that the ASCNM-BFGS method provided consistently a stable behaviour with the use of the scaling factor and a better ability to escape from swallow local minima, which could be attributed to its nonmonotone behaviour; some examples of improved learning behaviour are illustrated in Figures 6.4-6.5.



(a) FFTD



(b) LRN



(c) NARX

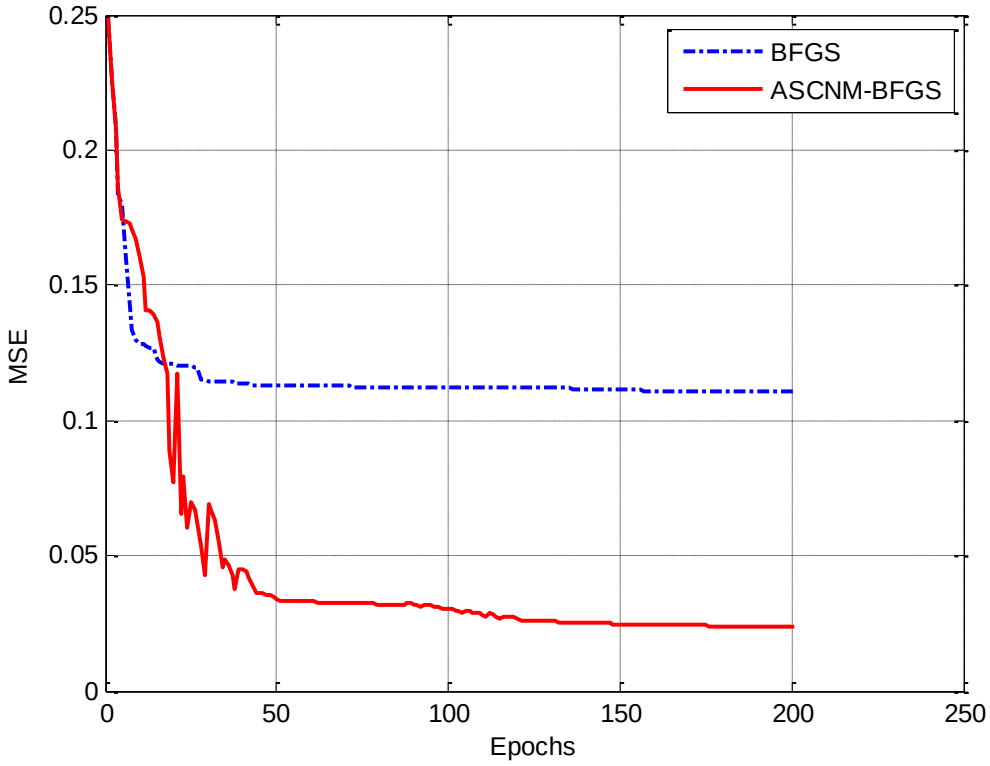
Figure 6.5 Examples of learning behaviours of 3 RNNs for the P10 problem, BFGS vs. ASCNMBFGS: (a) FFTD, (b) LRN and (c) NARX

6.4.2 The Sequence Classification Problem

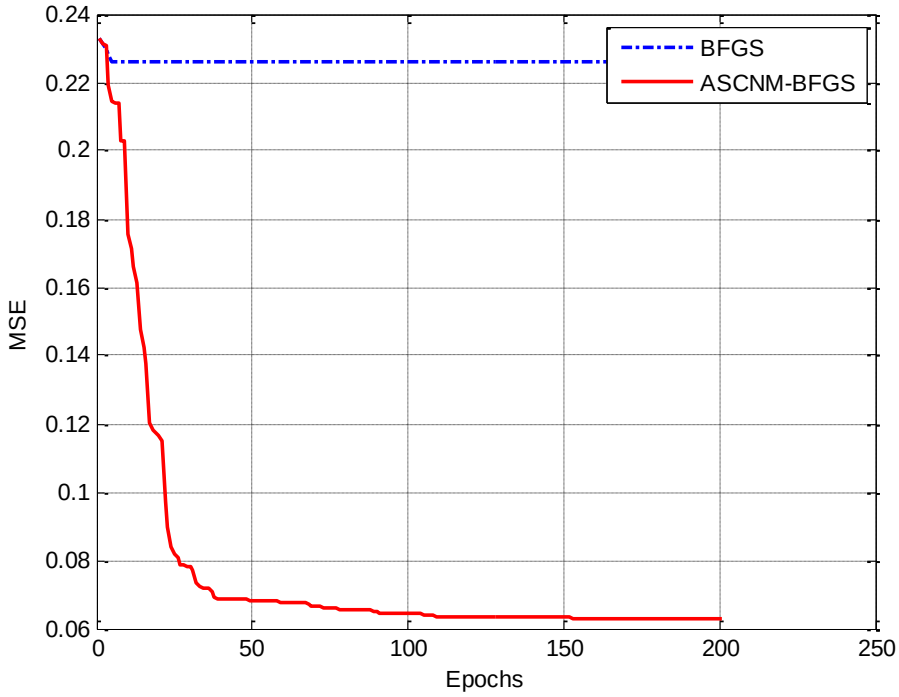
The 100-run averaged results are shown in Table 6.8, while learning examples are in Figure 6.6. Table 6.8 shows the average performance in terms of MSE (%) achieved in training and CE (%) in testing. In all cases, the proposed algorithm achieves better MSE, from 0.1% to 12%, and CE, from 0.2% to about 20%, with LRNs producing better generalisation (i.e. lower CE) than the other RNNs. Examples of learning behaviours are in Figure 6.6, showing how the nonmonotone strategy helps locating minimisers with lower error values, which leads to lower average classification error in testing (cf. with Table 6.8).

Table 6.8 Average performance for 3 RNNs in the SC problem: class of BFGS

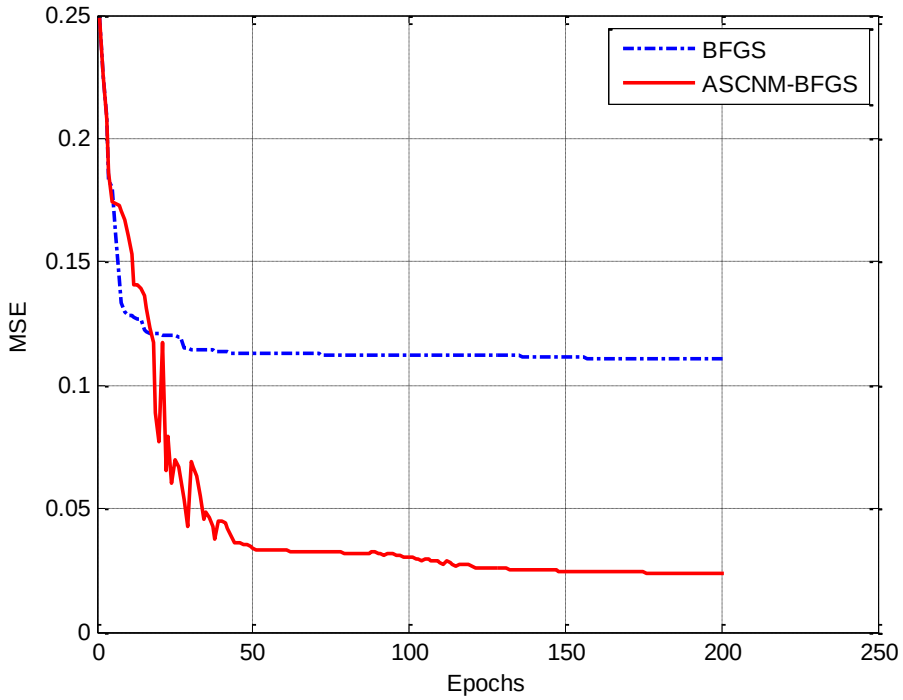
RNN	Algorithm	Training		Testing	
		MSE (%)	STD (%)	CE (%)	STD (%)
FFTD	BFGS	21.484	6.247	33.363	8.772
	ASCNM-BFGS	**20.326**	**4.891**	**32.041**	**7.213**
LRN	BFGS	21.518	5.903	32.301	7.867
	ASCNM-BFGS	**9.175**	**2.213**	**11.534**	**4.566**
NARX	BFGS	7.496	4.259	27.247	7.386
	ASCNM-BFGS	**7.100**	**3.881**	**27.082**	**6.832**


BFGS
ASCNM-BFGS
MSE
Epochs
0.25
0.2
0.15
0.1
0.05
0
0
50
100
150
200
250

(a) FFTD


BFGS
ASCNM-BFGS
MSE
Epochs
0.24
0.22
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0
50
100
150
200
250

(b) LRN



(c) NARX

Figure 6.6 Examples of learning behaviours of 3 RNNs for the SC problem, BFGS vs. ASCNMBFGS: (a) FFTD, (b) LRN and (c) NARX

6.4.3 The Sequence Learning Problem

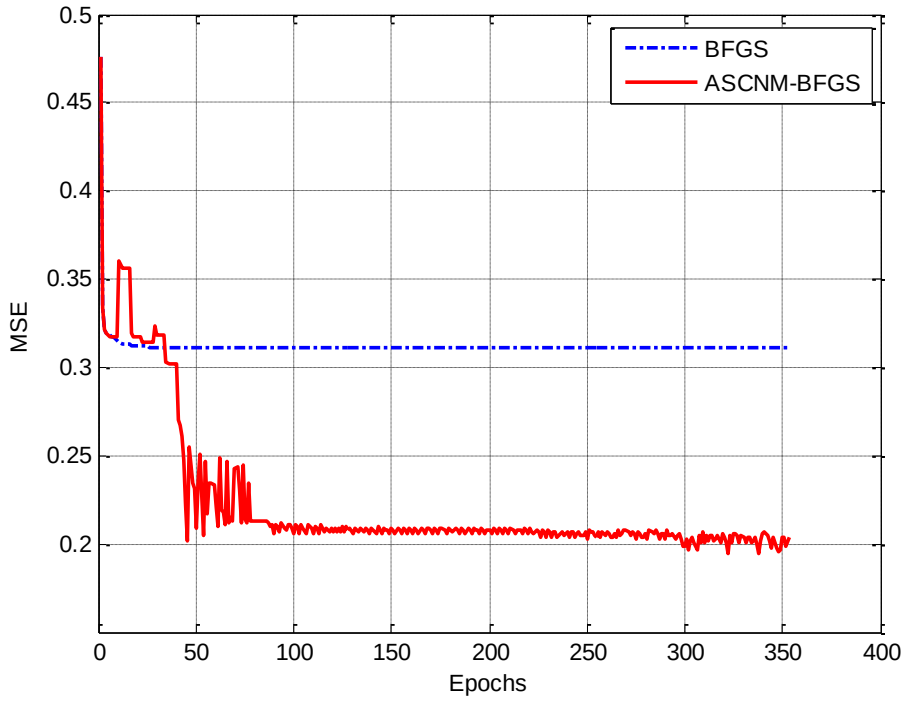
We made 100 runs for each of the three RNN architectures to estimate the average generalisation performance of the two algorithms. It is worth mentioning that the results in Table 6.9 were achieved using RNNs with 10 hidden nodes, while the work in [124] requires 16 hidden nodes to produce an average MSE of 25% using a first-order training method. For the purpose of further comparison another set of simulations were carried out for the NARX networks, as shown in Table 6.10, with 2, 5 and 10 hidden nodes. Typical examples of learning behaviours for the three RNNs are provided in Figure 6.7.

Table 6.9 Average performance for 3 RNNs in the SL problem: class of BFGS.

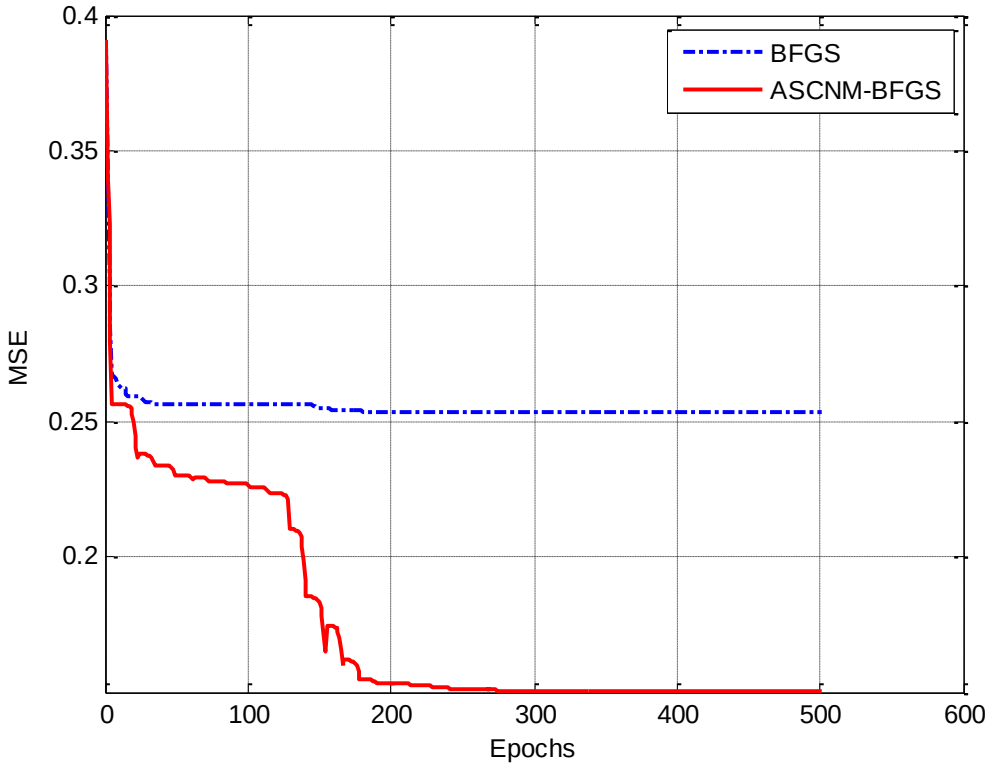
RNN	Algorithm	Training		Testing	
		MSE (%)	STD (%)	MSE (%)	STD (%)
FFTD	BFGS	21.485	9.027	17.437	16.128
	ASCNM-BFGS	**17.275**	**6.537**	**15.457**	**7.117**
LRN	BFGS	23.347	7.820	16.492	14.371
	ASCNM-BFGS	**17.854**	**6.233**	**15.070**	**6.821**
NARX	BFGS	8.991	3.435	9.846	5.823
	ASCNM-BFGS	**7.584**	**2.428**	**8.313**	**3.007**

Table 6.10 Average MSEs values for NARX networks in the SL problem: class of BFGS.

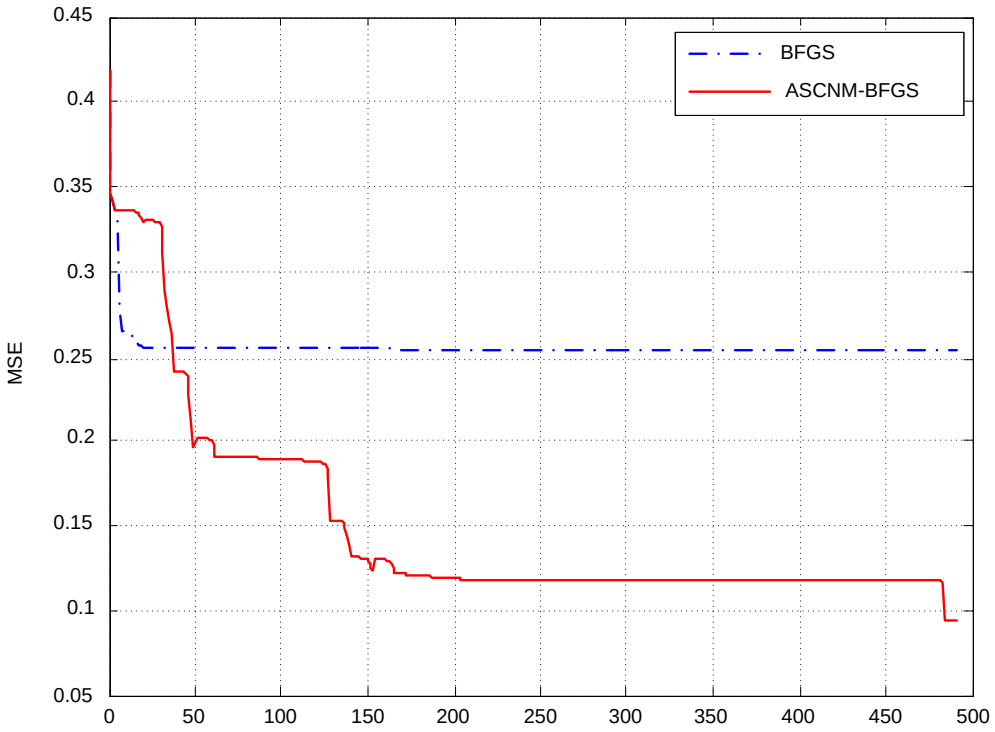
Algorithm	#hid	Training		Testing	
		MSE (%)	STD (%)	MSE (%)	STD (%)
BFGS	2	20.196	10.267	20.824	9.782
	5	11.029	6.641	12.119	6.020
	10	8.991	3.435	9.846	5.823
ASCNM-BFGS	**2**	**19.972**	**8.621**	**20.792**	**8.883**
	5	**9.740**	**5.418**	**10.360**	**5.892**
	10	**7.584**	**2.428**	**8.313**	**3.007**


BFGS
ASCNM-BFGS
MSE
Epochs
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250
300
350
400

(a) FFTD


BFGS
ASCNM-BFGS
MSE
Epochs
0.4
0.35
0.3
0.25
0.2
0
100
200
300
400
500
600

(b) LRN



(c) NARX

Figure 6.7 Examples of learning behaviours of 3 RNNs for the SL problem, BFGS vs. ASCNMBFGS: (a) FFTD, (b) LRN and (c) NARX

Tables 6.11-6.13 exhibit the results of additional simulations for FFTD, LRN and NARX networks in order to explore the generalisation performance when additional training iterations take place, as done in Subsections 4.4.3, 5.4.3.

Table 6.11 Results of additional simulations for FFTD networks in the SL problem: class of BFGS

Algorithms	200-epoch		1000-epoch	
	Train (MSE/CE)	Test (MSE/CE)	Train (MSE/CE)	Test (MSE/CE)
BFGS	20.5/71.9	21.1/73.4	16.6/40.3	17.4/45.1
ANM-BFGS	16.4/40.2	16.9/40.8	14.7/32.5	15.1/38.4

Table 6.12 Results of additional simulations for LRN networks in the SL problem: class of BFGS

Algorithms	200-epoch		1000-epoch	
	Train (MSE/CE)	Test (MSE/CE)	Train (MSE/CE)	Test (MSE/CE)
BFGS	20.6/72.1	16.0/39.6	14.2/31.7	15.0/38.5
ANM-BFGS	15.6/39.3	14.8/37.8	11.0/26.7	13.5/29.9

Table 6.13 Results of additional simulations for NARX networks in the SL problem: class of BFGS

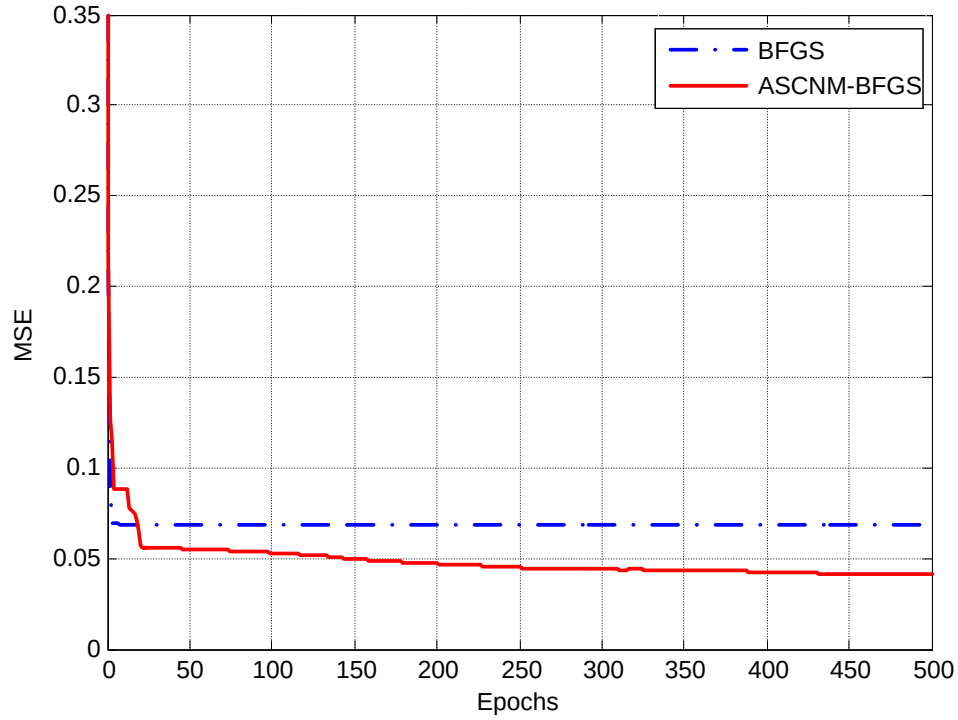
Algorithms	200-epoch		1000-epoch	
	Train (MSE/CE)	Test (MSE/CE)	Train (MSE/CE)	Test (MSE/CE)
BFGS	7.8/20.1	9.4/23.5	7.2/19.4	8.7/21.6
ANM-BFGS	6.0/17.6	7.9/20.3	5.8/17.0	7.4/19.9

6.4.4 The Reading Aloud Problem

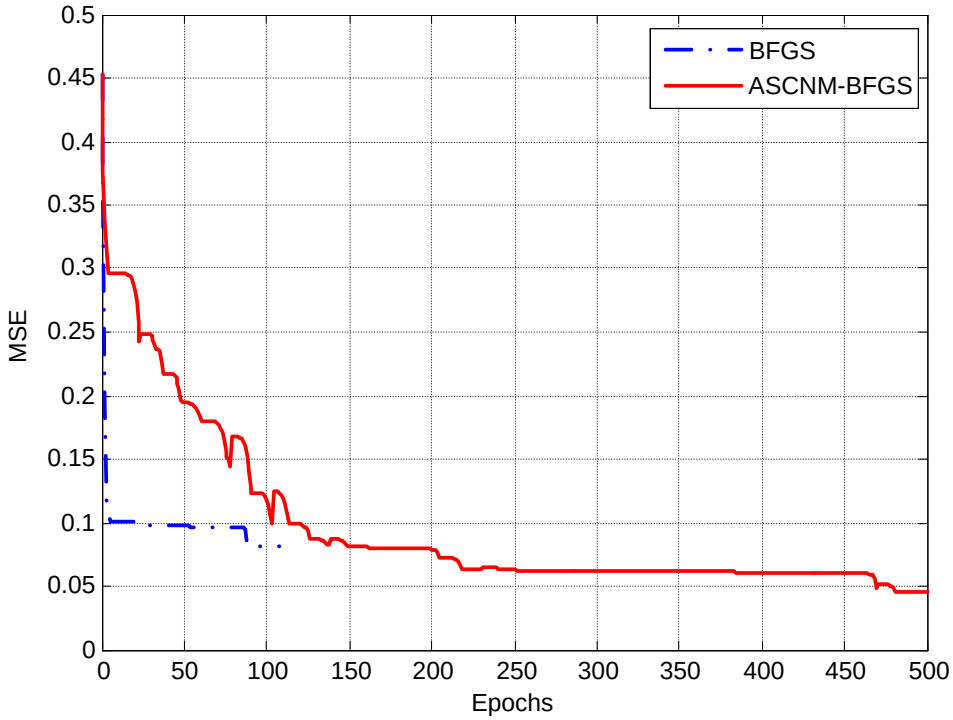
This task requires a special RNN architecture and is computationally expensive in training. As our purpose was to test the new method on standard RNN architectures, we didn't deploy the special architecture proposed by [158] which was trained for 1900 epochs. Instead, we used general type RNNs, i.e. FFTD and NARX networks, with only 5 hidden nodes that produced good results in previous tests, and trained them for 500 epochs. Figure 6.8 presents examples of convergence behaviour for BFGS and the ASCNMBFGS. The computational cost was still very high but we produced good solutions (see Table 6.14) using the ASCNMBFGS and BFGS algorithm. The heuristic parameters were set to the same values as in the parity problems. It is worth mentioning the error reported in [158] was applied 100 hidden nodes and 3 times larger number of training epochs.

Table 6.14 Average performance for two RNN architectures in the RA problem: class of BFGS.

RNNs	Algorithm	#hid	Training		Testing	
			MSE (%)	STD (%)	MSE (%)	STD (%)
FFTD	BFGS	5	10.665	5.298	18.716	7.625
		10	6.982	3.613	15.840	5.834
	ASCNM-BFGS	**5**	**9.558**	**4.779**	**18.143**	**7.032**
		10	**6.081**	**3.043**	**15.652**	**4.990**
NARX	BFGS	5	8.589	4.106	16.625	5.428
		10	6.248	2.733	15.407	3.961
	ASCNM-BFGS	**5**	**7.080**	**2.824**	**16.106**	**4.752**
		10	**5.184**	3.039	**14.547**	4.006



(a) FFTD



(b) NARX

Figure. 6.8 Behaviours of BFGS and our method for training (a) FFTD and (b) NARX networks on the RA problem

6.5 Summary and Contribution of the Chapter

In this chapter we proposed a nonmonotone approach which is based on the BFGS method, a well-known quasi-Newton method. It employs self-scaling of the approximations of the Hessian matrix. Furthermore, it is equipped with an adaptive nonmonotone strategy to better exploit information collected as it searches the space of the adjustable parameters. Comparing to the traditional monotone learning approach, our experiments using various data sequences provide evidence that the self-scaling BFGS with Adaptive Nonmonotone Strategy enhances the convergence behaviour of RNNs and is more effective for training networks of various RNN architectures than the BFGS, even when hidden nodes' numbers are smaller than the ones typically reported in the literature for these problems.

Chapter 7

Adaptive Nonmonotone Levenberg-Marquardt Algorithms

Equipped with a damping factor, the Levenberg-Marquardt (LM, so-called *damped Gauss-Newton*) methods [106][121] are capable of relaxing the difficulties of Hessian-based training, i.e. the ill-conditioning of the Hessian matrix. In addition, when the damping factor is zero, the LM methods become identical to the Gauss-Newton approach; while as the damping factor gets close to infinity, the LM methods are then get equivalent to the steepest descent method. More details are provided within this chapter and can also be found in the latest relevant literature such as [129][130].

In the rest of this chapter, the original LM methods [106][121] are firstly reviewed in Section 7.1 with a discussion of their applications to train neural networks [7][42][60] [84][97][105][134][149][171][191][192][198][208][215] and a formulation of the learning problem [7][85][106][121]. After discussing global convergence for monotone [7] and nonmonotone [213] LM methods in Section 7.2, we present the proposed nonmonotone algorithms in Section 7.3. Experimental results for two

artificial (i.e., *N*-bit parity, P5 and P10) and three real-world (i.e., SC [116], SL [124] and RA [158]) on three different RNNs architectures (i.e. FFTD [196][197], LRN [57][85] and NARX [128][137]) are provided in Section 7.4, while Section 7.5 concludes this chapter. Note that full descriptions of the simulated applications, relative structural settings, and definitions of the applied RNNs can be found in Appendix A.1 and Section 2.1, respectively.

7.1 Levenberg-Marquardt Methods

Since the first attempt [85] to train static neural networks, the Levenberg-Marquardt (LM) method [106][121] has been revised to incorporate adaptive-momentum terms [7] and attracted a lot of attention in training neural networks [43][60][97][105][134] [149][171][191][192][198][208][215]. Among these works only [43][134][171] concern dynamic or recurrent neural networks (RNN) mostly for time-series problems. All these LM-type algorithms are descent methods, i.e. they accept the next weight iterate if its associated error function value is smaller than the value of the current iterate. This property of monotonicity ensures that each successful iteration produces a weight set that is better than any previous one in terms of learning error value.

From the perspective of the Levenberg-Marquardt method, the error function E is a nonlinear least-square problem with zero or small residual:

$$E(w) = (1/2)\|e(w)\|_2^2 = (1/2)\sum_{i=1}^{p} e_i^2(w), \tag{7. 1}$$

where the i^{th} component of the p-dimensional vector $e(w)$ is

$$e_i(w) = \sum_i \left(y_i(w) - \bar{y}_i(w)\right), \tag{7. 2}$$

(7.3)

with the network's output y and desired output $\bar{y}$, and $e_i(w)$ is twice continuously differentiable, $\|e(w)\|$ is termed residual at w.

Assume that the current approximation to the solution of the above problem is w_k and J_k denotes the Jacobian matrix of $e(w)$, if $g_k = J_k e_k \neq 0$, then the LM method is based on a set of linear equations in order to determine the increment β_k, the so-called *optimal step* or *Newton step*

$$(H_k + \alpha_k D_k)\beta_k = -g_k, \tag{7.4}$$

where $H_k = J_k^T J_k$, α_k represents a nonnegative scalar, $D_k = D(w_k)$ is a continuous, positive-definite diagonal matrix.

Besides the most common way that considers

$$D(w) = \gamma I, \tag{7.5}$$

where constant $\gamma > 0$ and I the identity matrix, $D(w)$ can be chosen as

$$D(w) = \text{diag}\left\{H_{11}(w), H_{22}(w), \ldots, H_{nm}(w)\right\} + \sigma I, \tag{7.6}$$

where $H_{ii}(w)$ is the *i*-th diagonal element of $H(w)$ and σ a small positive constant. As with other second-order methods, this formulation assumes that we can have a local quadratic approximation of *E* denoted by $\tilde{E}$,

$$\tilde{E}(w, w_k) = E(w_k) + g_k^T \beta_k + (1/2)\beta_k^T H_k \beta_k, \tag{7.7}$$

which is defined in Eq. (7.7). More details and discussions about the LM method can be found in the literature, such as in [7][85][106][121]. Note that α_k in Eq. (7.4) controls both the magnitude and direction of β_k. When α_k is zero, Eq. (7.4) is identical to the Gauss-Newton method; while as α_k is closed to infinity, Eq. (7.4) is equivalent to the steepest descent method. For easy reference, we recall the pseudo-code of the monotone LM method (MLM) here.

Table 7.1 The monotone Levenberg-Marquardt algorithm

MLM Algorithm [106][121]

STEP 0. Initialize $\alpha_0 > 0$, $\tau > 1$, $\bar{\xi} \in (0,1)$, w_0, D_0, and $k = 0$;

STEP 1. If $g_k = J_k e_k \neq 0$, calculate $H_k = J_k^T J_k$; otherwise, stop;

STEP 2. Compute β_k in Eq. (7.4);

STEP 3. Calculate

$$\xi_k = (E_k - E_{k+1}) \Big/ \left[\tilde{E}(w_k, w_k) - \tilde{E}(w_k + \beta_k, w_k)\right], \tag{7.8}$$

where $\tilde{E}$ is defined by Eq. (7.7);

STEP 4. If $\xi_k < \bar{\xi}$, set $\alpha_k = \tau \alpha_k$ and go to Step 2; Else, $\alpha_k = \alpha_k / \tau$ and $w_{k+1} = w_k + \beta_k$;

STEP 5. Set $k = k + 1$, go to STEP 1.

In the following sections, we explore the possibility of removing the monotonicity restriction by equipping LM methods with strategy that allows the sequence of error values at the weight iterates to be nonmonotone. To this end, we develop nonmonotone versions of two LM algorithms, which were originally proposed for training static neural networks [7], i.e. the LM with adaptive momentum (LMAM) and the optimised LMAM (OLMAM). In the original work, these methods were comparatively evaluated against the original LM, the BFGS method, and conjugate gradient algorithms, and achieved outstanding performances on three applications, i.e. one N-bit parity problem and two non-symbolic classification problems.

7.2 Global Convergence

In this section, we briefly review the theory for global convergence of nonmonotone LM methods [213] in relation to the methods proposed in [7]. For both works, [7] and [213], the following assumptions were firstly made, which are the hypotheses of the Zoutendijk's theorem.

Assumptions [7][213]. The error function E is bounded and continuously differentiable in a neighborhood N of the level set $L(w_0) = \left\{ w \in R^n \,\middle|\, E(w) \le E(w_0) \right\}$, while L is compact.

Similar to the first Wolfe's condition referred in [7], i.e.

$$E\left(w_k + \beta_k\right) \le \max_{0 \le j \le M} \left\{ E\left(w_{k-j}\right) \right\} + \delta \cdot \Phi, \tag{7.9}$$

where Φ is an adaptive term that measures the sufficiency of the error decrease, the so-called forcing function. Moreover, [213] proved there exists constant $\delta > 0$ such that the inequality (7.9) holds, while the second Wolfe's condition was ignored. Then, under the above Assumptions, the following three theorems were proved [213].

□ **Theorem 7.1** [213]. *If the sequence* $\{w_k\}$ *is generated by the NMLM1 & NMLM2 (both algorithms are described in Appendix A.2), then*

$$\liminf_{k\to\infty} \|g_k\| = 0. \tag{7.10}$$

□

□ **Theorem 7.2** [213]. *If sequence* $\{w_k\}$ *is generated by the NMLM1, and if the approximated Hessian matrix* $T(w_k)$ *is uniformly positive definite for sufficiently large k, then*

$$\lim_{k\to\infty} \|g_k\| = 0. \tag{7.11}$$

Furthermore, if the set of stationary points of $E(w)$ *is finite, say* $\{w_1^*, \ldots, w_m^*\}$, *then there is an integer q,* $1 \le q \le m$, *such that*

$$\lim_{k\to\infty} w_k = w_q^*. \tag{7.12}$$

□

□ **Theorem 7.3** [213]. *Suppose that* w^* *is the unique stationary point of* $f(w)$ *on*

$L(w_0)$, $f(w^*)=0$, and $T(w^*)$ is positive definite. If $\{w_k\}$ is generated by NMLM1, then

$$\lim_{k\to\infty} w_k = w^*. \tag{7.13}$$

□

Comparing to [7], only Eq. (7.11) in Theorem 7.2 was proved by applying Zoutendijk's theorem.

7.3 Our Proposed Algorithms

Our proposed revisions of LMAN and OLMAN [7], named Adaptive Non-Monotone LMAM (ANM-LMAM) and Adaptive Non-Monotone OLMAM (ANM-OLMAM), are as follows. Compared to the standard monotone LM algorithm, presented in Table 7.1, STEP 2 of the LMAM method includes two adaptive momentum terms, i.e., ψ_1 and ψ_2., defined in Eqs. (7.15) and (7.16).

Table 7.2 Adaptive nonmonotone LM method with adaptive momentum

Algorithm: ANM-LMAM

STEP 0. Initialize $\tau > 1,$, w_0, D_0, $M^{\max}$, Φ, $\delta \in (0,1)$, $\Theta \in (0,1)$, ε, and $k = 0$;

STEP 1. If $g_k = J_k e_k \neq 0,$ calculate $H_k = J_k^T J_k$; otherwise, stop;

STEP 2. Compute β_k by

$$\beta_k = -\frac{\psi_1}{2\psi_2}\left[\left(H_k + \alpha_k D_k\right)\right]^{-1} g_k + \frac{1}{2\psi_2}\beta_{k-1}, \tag{7.14}$$

where

$$\psi_1 = \frac{-2\psi_2\varepsilon + g_k^T\beta_k}{g_k^T H_k^{-1} g_k}, \tag{7.15}$$

$$\psi_2 = \frac{1}{2}\left[\frac{\beta_{k-1}^T H_k \beta_k \cdot g_k^T H_k^{-1} g_k - \left(g_k^T \beta_{k-1}\right)^2}{g_k^T H_k^{-1} g_k \Theta^2 - \varepsilon^2}\right]^{1/2}, \tag{7.16}$$

and

$$\varepsilon = -c\Theta\left(g_k^T H_k^{-1} g_k\right)^{1/2}; \tag{7.17}$$

STEP 3. If $k \geq 1$, calculate the local Lipschitz approximation Λ^k by

$$\Lambda_k = \frac{\left\|g_k - g_{k-1}\right\|}{\left\|w_k - w_{k-1}\right\|}, \tag{7.18}$$

and update M_k by

$$M_k = \begin{cases} M_{k-1} + 1, & \text{if } \Lambda_k < \Lambda_{k-1} < \Lambda_{k-2} \\ M_{k-1} - 1, & \text{if } \Lambda_k > \Lambda_{k-1} > \Lambda_{k-2}, \\ M_{k-1}, & \text{otherwise,} \end{cases} \tag{7.19}$$

where $M_k = \min\left\{M_k, M^{\max}\right\}$;

STEP 4. If Eq. (7.9) is not satisfied, $\alpha_k = \tau\alpha_k$ and go to Step 2;

Else, $\alpha_k = \alpha_k / \tau$ and $w_{k+1} = w_k + \beta_k$;

STEP 5. $k = k + 1,$ go to Step 1.

To derive the ANM-OLMAM algorithm, following the OLMAM method's description in [7], the two constants Θ and c, which are initialized in Step 0 and used in Step 2, are revised by

$$\Theta = \left(g_k^T H_k^{-1} g_k \right)^{1/2}, \tag{7.20}$$

and

$$c = \left(1 - \frac{\left(g_k^T \beta_{k-1} \right)^2}{g_k^T H_k^{-1} g_k \cdot \beta_{k-1}^T H_k \beta_k} \right)^{1/2}. \tag{7.21}$$

Therefore, following the changes in Step 0 and Step 2, we have the second proposed LM-like algorithm, i.e. ANM-OLMAM. The derivation processes of Eqs. (7.10)-(7.13) and (7.16)-(7.17) can be found in [7].

As reported in [7], the settings of the first Wolfe condition are $\delta = 0.1$ and the forcing function $\Phi = g_k^T \beta_k$; therefore, comparing to LMAM and OLMAM, there is only one extra free parameter within our proposed algorithms, i.e. the upper bound of nonmonotone learning horizon M^k. As already stated in previous chapters, we have found setting $M^{\max}$ to 15 generally provides good performance in all applications [145][146].

Examples of convergence behaviour for ANM-LMAM and ANM-OLMAM trained NARX networks (7 hidden nodes for the P5 and 10 for the P10 were used, as listed in Table A.1) are shown in Figures 7.1 and 7.2 using relatively small training goals. Despite temporary reductions in the size of the sliding window, Figures 7.1 and 7.2 show that there is a trend to enlarge the length of the window and that the methods

produce from time to time large values for elements of β_k, which speeds up the process. It is worth noticing that this behaviour is in accordance with theoretical and empirical results about the behaviour of nonmonotone methods [82][79][146]. Examples of nonmonotone convergence behaviours for the simulated problems are provided in the next section.

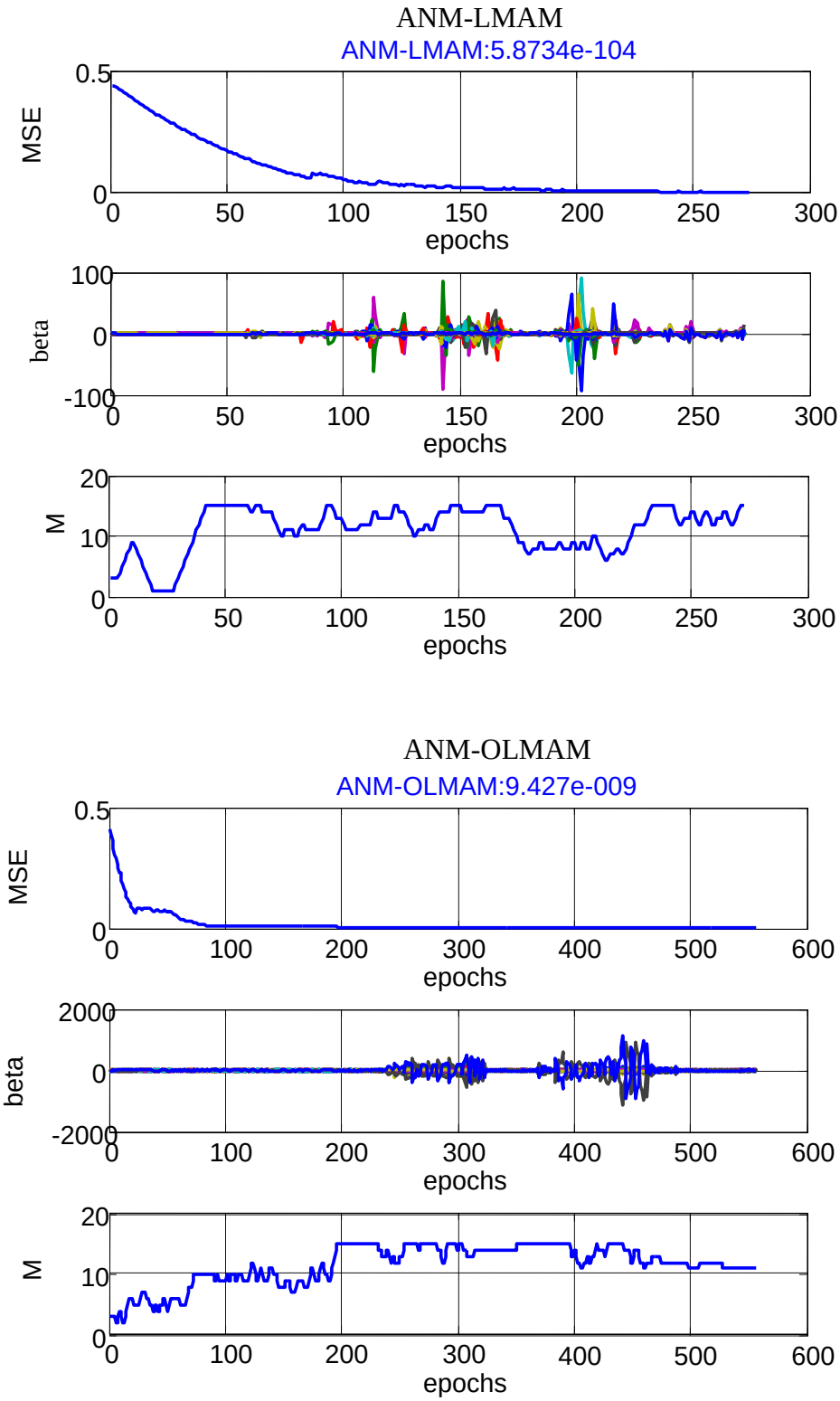


Figure 7.1 Convergence behaviours of ANM-LMAM and ANM-OLMAM in the P5 problem for NARX networks

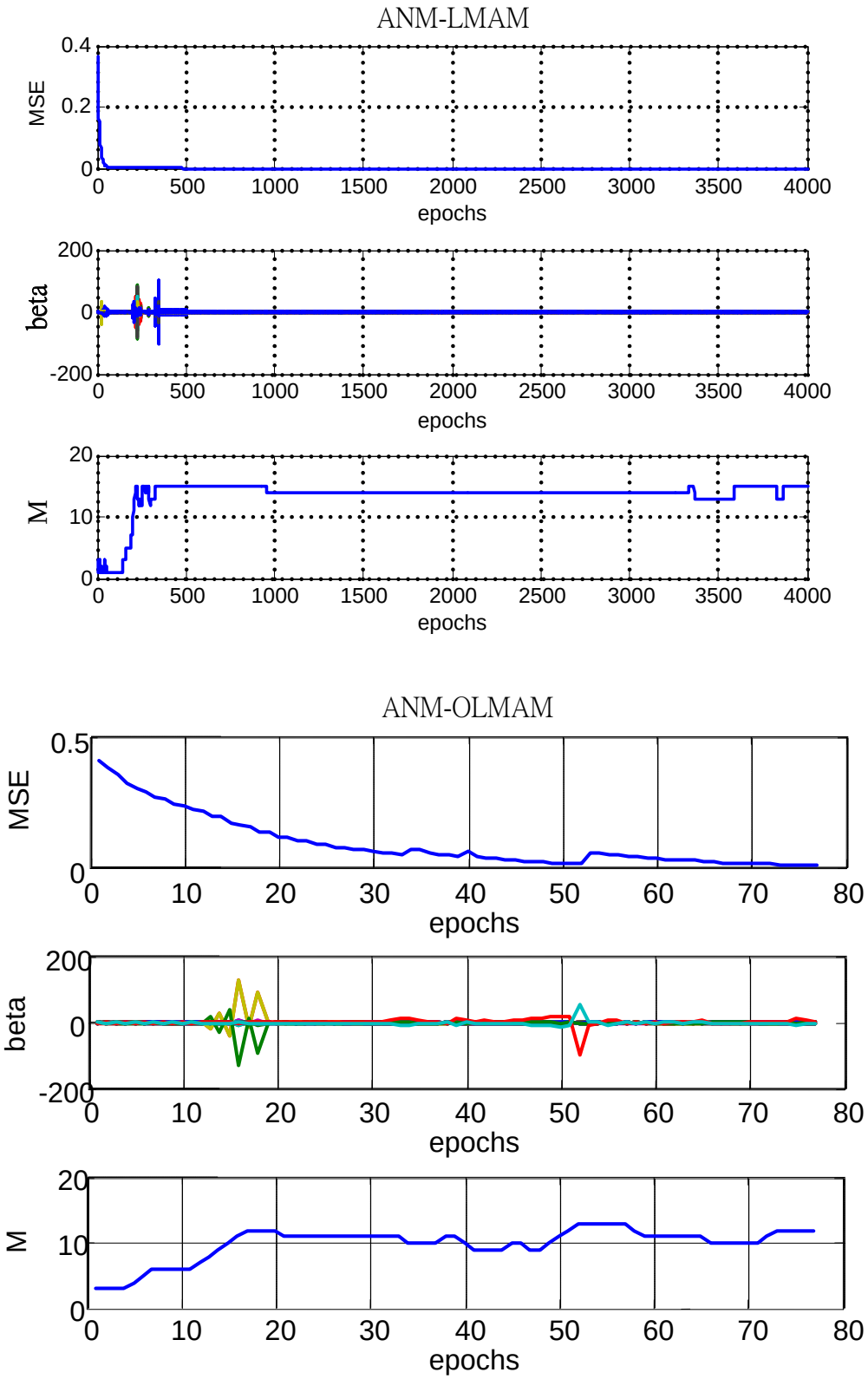


Figure 7.2 Convergence behaviours of (a) ANM-LMAM and (b) ANM-OLMAM in the P10 problem for NARX networks

7.4 Experimental Results

As mentioned in previous chapters, relative references and numerical settings of the simulations in this thesis can be found in Appendix (A.1), i.e. types of RNNs, amounts of hidden nodes, relative delays, boundaries of learning horizon *M*, and the constant δ used for the nonmonotone strategy, while all results shown are the average of 100 runs initialised randomly. MSE and CE values are shown in percentage, while the *STD* column represents the corresponding standard deviation. The forcing function Φ is the same as the one applied in Chapters 4, 5 and 6, while the other parameters are set to $\tau = 2$, $\Theta = 0.03$, $c = 0.95$, and D_0 denotes the identity matrix. Details about the notations used in the following Tables can be found in Section 4.4.

7.4.1 N-bit Parity Problems

Simulation results of the P5 problem are shown in Tables 7.3-7.5, while Tables 7.6-7.8 exhibit results for the P10 problem. From these results it can be easily observed that all the monotone versions of LM methods, i.e. LM, LMAM and OLMAM, failed to converge and were trapped in some local minimum points, while the proposed nonmonotone modifications, i.e., ANM-LMAM and ANM-OLMAM, perform generally much better. Although there are some non-converged cases for ANM-OLMAM trained networks with 1 or 2 hidden nodes, as shown in Table 7.7, the average MSEs for 100-runs when using 1 or 2 hidden nodes are about 13% better than the ones achieved by OLMAM. Examples of learning behaviours for the P5 and P10 problems are shown in Figures 7.3 and 7.4, respectively.

Table 7.3 Average performance for **FFTD** networks in the **P5** problem: class of LM.

Algorithms	#hid	Conv (%)	MSE (%)	STD (%)	Epochs			
					Ave	Min	Max	Std
LM	1	0	37.686	2.317	-	-	-	-
	2	0	38.819	2.592	-	-	-	-
	5	0	36.319	3.214	-	-	-	-
	7	0	35.945	2.763	-	-	-	-
LMAM	1	0	22.453	3.930	-	-	-	-
	2	0	18.328	7.622	-	-	-	-
	5	0	35.816	2.338	-	-	-	-
	7	0	36.225	2.846	-	-	-	-
ANM-LMAM	**1**	**24**	**17.855**	**2.753**	**62**	**46**	**107**	**11**
	2	**50**	**10.934**	**3.105**	**66**	**46**	**151**	**25**
	5	**67**	**4.086**	**1.992**	**164**	**47**	**646**	**111**
	7	**99**	**0.910**	**0.073**	**172**	**63**	**312**	**52**
OLMAM	1	0	36.865	1.995	-	-	-	-
	2	0	37.572	2.658	-	-	-	-
	5	0	36.057	2.472	-	-	-	-
	7	0	35.632	2.208	-	-	-	-
ANM-OLMAM	**1**	**26**	**16.520**	2.335	**214**	**46**	**587**	**172**
	2	**43**	**9.686**	**1.850**	**283**	**46**	**1194**	**234**
	5	**85**	**1.781**	**1.421**	**267**	**46**	**1883**	**356**
	7	**99**	**0.974**	**0.147**	**64**	**37**	**364**	**36**

Table 7.4 Average performance for **LRN** networks in the **P5** problem: class of LM.

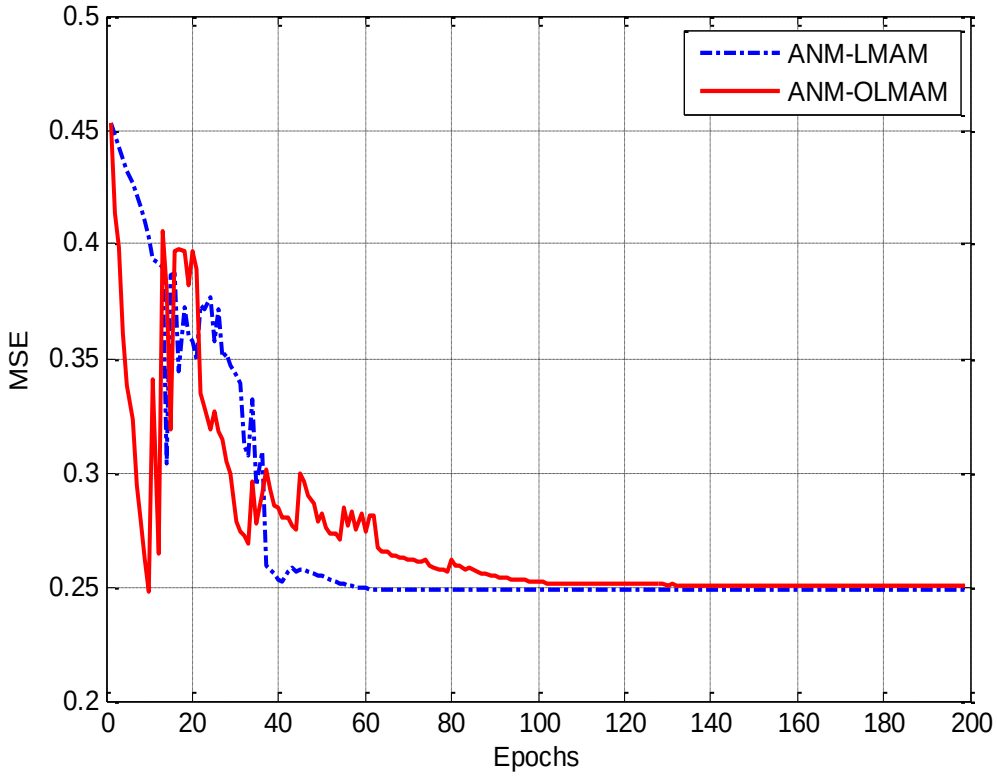
Algorithms	#hid	Conv (%)	MSE (%)	STD (%)	Epochs			
					Ave	Min	Max	Std
LM	1	0	37.520	2.872	-	-	-	-
	2	0	38.462	2.701	-	-	-	-
	5	0	36.319	3.338	-	-	-	-
	7	0	36.649	2.940	-	-	-	-
LMAM	1	0	37.920	2.717	-	-	-	-
	2	0	38.416	3.023	-	-	-	-
	5	0	36.319	3.402	-	-	-	-
	7	0	36.649	2.785	-	-	-	-
ANM-LMAM	**1**	**20**	**15.650**	3.572	**133**	**54**	**1439**	**308**
	2	**37**	**13.714**	**2.856**	**64**	**46**	**199**	**28**
	5	**66**	**4.325**	**1.184**	**165**	**47**	**646**	**111**
	7	**100**	**0.645**	**0.032**	**175**	**60**	**427**	**65**
OLMAM	1	0	37.244	2.692	-	-	-	-
	2	0	37.912	2.836	-	-	-	-
	5	0	36.075	3.307	-	-	-	-
	7	0	36.424	2.655	-	-	-	-
ANM-OLMAM	**1**	**9**	**20.277**	4.256	**267**	**57**	**444**	**135**
	2	**44**	**8.731**	**1.730**	**282**	**46**	**910**	**233**
	5	**85**	**1.800**	**0.954**	**270**	**46**	**1883**	**363**
	7	**96**	**1.089**	**0.147**	**98**	**41**	**1332**	**165**

Table 7.5 Average performance for **NARX** networks in the **P5** problem: class of LM.

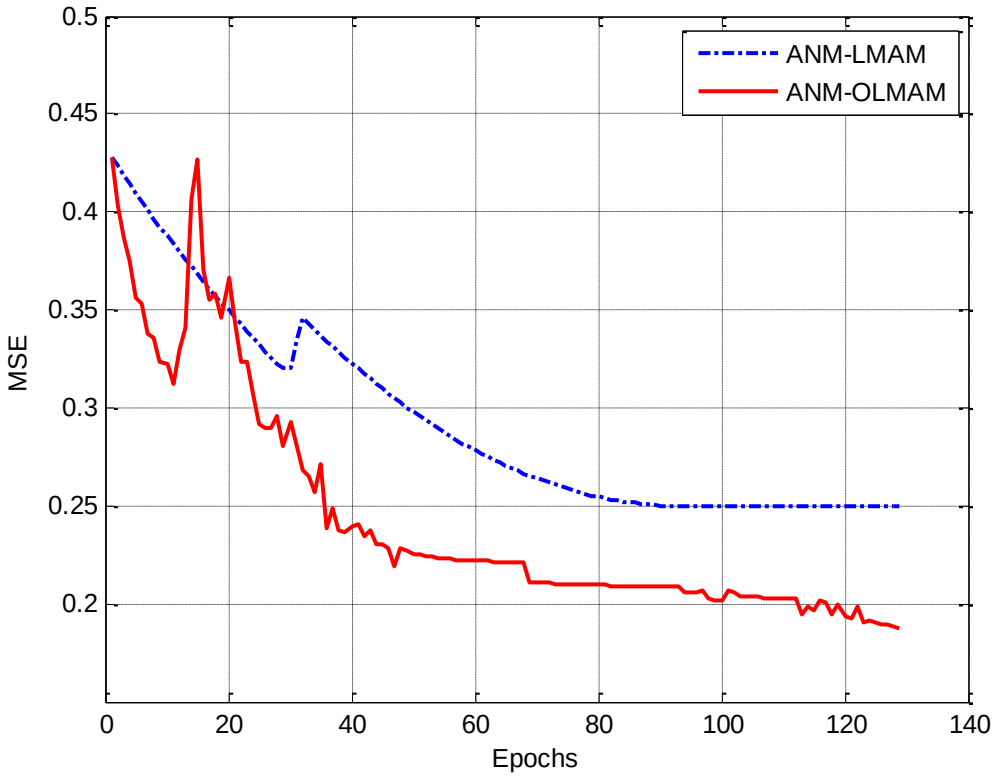
Algorithms	#hid	Conv (%)	MSE (%)	STD (%)	Epochs			
					Ave	Min	Max	Std
LM	1	0	37.953	2.748	-	-	-	-
	2	0	37.851	2.503	-	-	-	-
	5	0	36.205	2.115	-	-	-	-
	7	0	36.391	2.628	-	-	-	-
LMAM	1	0	38.448	3.224	-	-	-	-
	2	0	37.851	2.682	-	-	-	-
	5	0	36.205	2.006	-	-	-	-
	7	0	36.391	2.439	-	-	-	-
ANM-LMAM	**1**	**73**	**3.032**	**1.731**	**108**	**63**	**168**	**21**
	2	**85**	**2.689**	**1.340**	**120**	**86**	**204**	**27**
	5	**99**	**0.955**	**0.003**	**132**	**89**	**225**	**28**
	7	**99**	**0.970**	**0.010**	**155**	**52**	**289**	**40**
OLMAM	1	0	38.386	3.309	-	-	-	-
	2	0	36.205	2.731	-	-	-	-
	5	0	36.093	2.488	-	-	-	-
	7	0	36.247	3.067	-	-	-	-
ANM-OLMAM	**1**	**96**	**0.625**	**1.047**	**72**	**33**	**571**	**86**
	2	**95**	**1.081**	**0.793**	**59**	**37**	**349**	**37**
	5	**100**	**0.945**	**0.008**	**47**	**38**	**77**	**6**
	7	**100**	**0.943**	**0.008**	**47**	**32**	**80**	**7**

More precisely, the results show that considerable improvements in terms of MSE can be achieved by the ANM-LMAM and ANM-OLMAM methods. For example, when using FFTD networks with 7 hidden nodes both ANM-LMAM and ANM-OLMAM in Table 7.3 outperform LMAM and OLMAM by 35.2% and 32.5%, respectively. Similar improvements can be seen in Tables 7.4 and 7.5 for LRNs (35.9% and 34.2%) and NARX (35.4% and 35.2%) networks. The improvements in the MSE for the P10 problem using FFTD, LRN and NARX recurrent networks with 10 hidden nodes are approximately 20.7%, 29.7%, 33.4% for the ANM-LMAM, and 35.4%, 32.0%, 24.5% for the ANM-OLMAM, as shown in Tables 7.6-7.8. It also appears that the ANM-OLMAM converges more times than the ANM-LMAM consistently. Taking the simulations with 10 hidden nodes in Table 7.8 as an example, ANM-OLMAM achieves a 100% convergence, with 10 times less training epochs on average. This result is in line with observations about the behaviour of the original LMAM and OLMAM in training static neural networks [7].

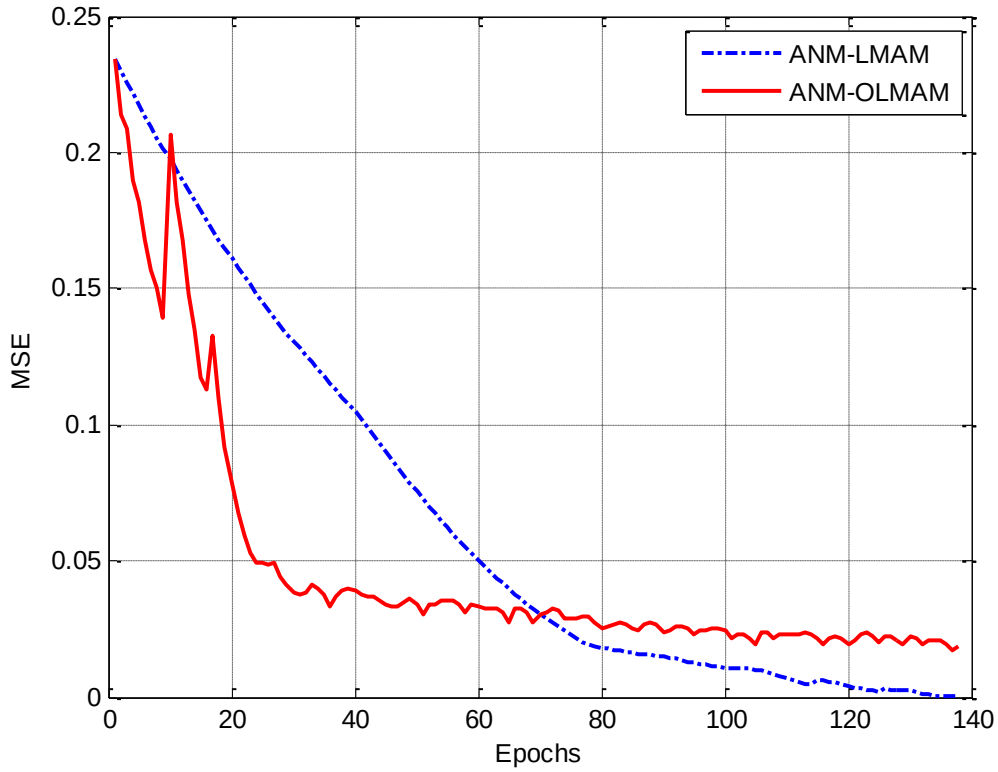
Figures 7.3-7.4 show some examples of nonmonotone convergence behaviour. Figure 7.3 provides an example of how the convergence behaviour of ANM-LMAM and ANM-OLMAM differ in a case where ANM-LMAM has stuck in the neighbourhood of a local minimum whilst ANM-OLMAM still trains the LRN network and continuously reduces the training MSE for P5. Similar behaviour is observed in Figure 7.4 for NARX networks in P10, where ANM-OLMAM appears to be better than ANM-LMAM.


ANM-LMAM
ANM-OLMAM
MSE
Epochs
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
20
40
60
80
100
120
140
160
180
200

(a) FFTD


ANM-LMAM
ANM-OLMAM
MSE
Epochs
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
20
40
60
80
100
120
140

(b) LRN



(c) NARX

Figure 7.3 Examples of convergence behaviours of ANM-LMAM and ANM-OLMAM in the P5 problem for three RNNs

Table 7.6 Average performance for **FFTD** networks in the **P10** problem: class of LM.

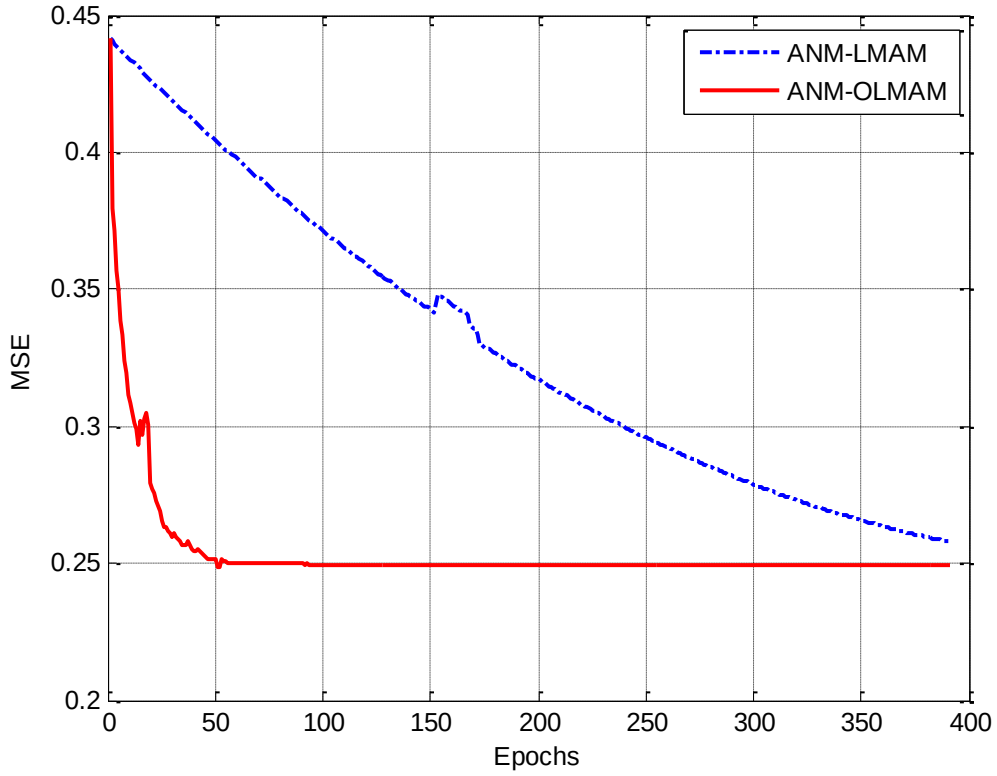
Algorithms	#hid	Conv (%)	MSE (%)	STD (%)	Epochs			
					Ave	Min	Max	Std
LM	1	0	37.871	3.782	-	-	-	-
	2	0	38.623	4.126	-	-	-	-
	5	0	36.707	3.006	-	-	-	-
	10	0	35.543	2.997	-	-	-	-
LMAM	1	0	24.040	3.116	-	-	-	-
	2	0	38.436	3.907	-	-	-	-
	5	0	35.825	2.793	-	-	-	-
	10	0	27.340	3.670	-	-	-	-
ANM-LMAM	**1**	**0**	**24.304**	3.256	**-**	**-**	**-**	**-**
	2	**8**	**19.822**	**2.071**	**699**	**250**	**1071**	**286**
	5	**14**	**6.970**	**1.939**	**869**	**289**	**2065**	**571**
	10	**31**	**6.669**	**1.451**	**968**	**155**	**2588**	**533**
OLMAM	1	0	37.715	4.697	-	-	-	-
	2	0	38.436	3.852	-	-	-	-
	5	0	35.825	2.792	-	-	-	-
	10	0	36.157	3.239	-	-	-	-
ANM-OLMAM	**1**	**0**	**23.931**	**2.894**	**-**	**-**	**-**	**-**
	2	**4**	**20.831**	**2.383**	**1149**	**410**	**1899**	**688**
	5	**14**	**12.975**	3.671	**815**	**146**	**3900**	**1007**
	10	**51**	**0.769**	**1.836**	**438**	**69**	**3760**	**774**

Table 7.7 Average performance for **LRN** networks in the **P10** problem: class of LM.

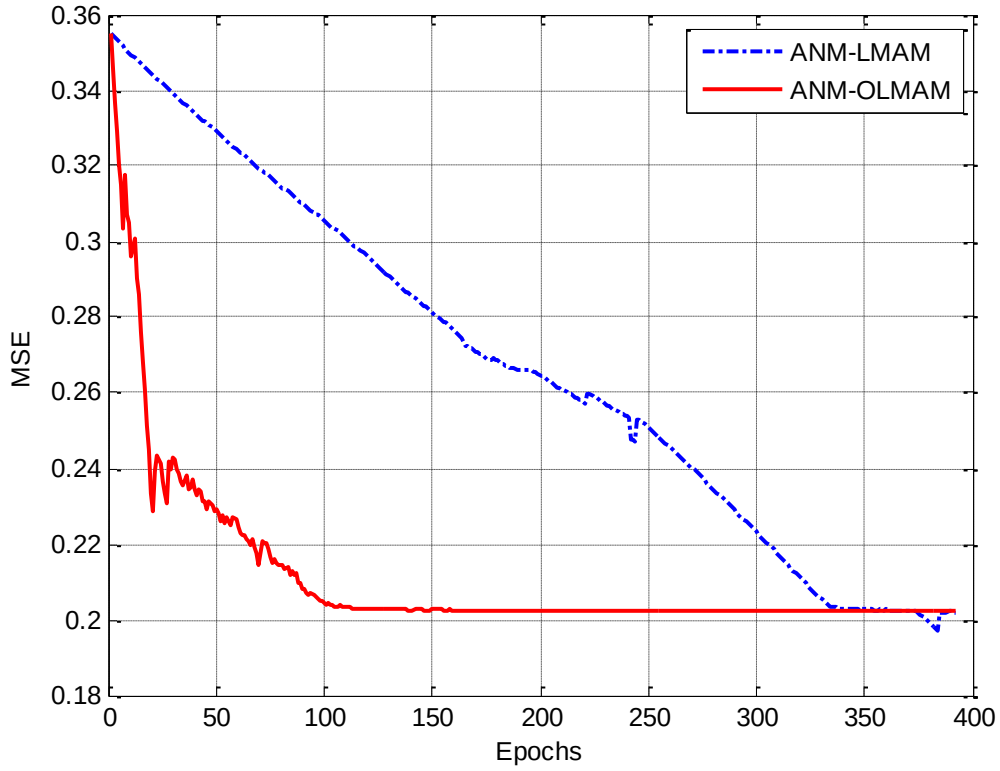
Algorithms	#hid	Conv (%)	MSE (%)	STD (%)	Epochs			
					Ave	Min	Max	Std
LM	1	0	37.739	3.529	-	-	-	-
	2	0	38.444	3.788	-	-	-	-
	5	0	36.801	2.813	-	-	-	-
	10	0	36.043	2.954	-	-	-	-
LMAM	1	0	37.744	3.733	-	-	-	-
	2	0	38.283	4.107	-	-	-	-
	5	0	36.639	3.242	-	-	-	-
	10	0	36.122	2.871	-	-	-	-
ANM-LMAM	**1**	**1**	**23.890**	**2.847**	**1959**	**1959**	**1959**	**0**
	2	**4**	**21.319**	**2.575**	**673**	**599**	**820**	**103**
	5	**9**	**12.689**	**1.813**	**761**	**626**	**925**	**221**
	10	**30**	**6.448**	**1.045**	**1047**	**367**	**3398**	**550**
OLMAM	1	0	37.744	3.733	-	-	-	-
	2	0	38.283	4.107	-	-	-	-
	5	0	36.639	3.242	-	-	-	-
	10	0	36.122	2.871	-	-	-	-
ANM-OLMAM	**1**	**0**	**24.089**	**3.118**	**-**	**-**	**-**	**-**
	2	**0**	**22.098**	**2.707**	**-**	**-**	**-**	**-**
	5	**18**	**11.703**	**1.543**	**1249**	**311**	**2749**	**1330**
	10	**47**	**4.160**	**0.837**	**467**	**68**	**3990**	**860**

Table 7.8 Average performance for **NARX** networks in the **P10** problem: class of LM.

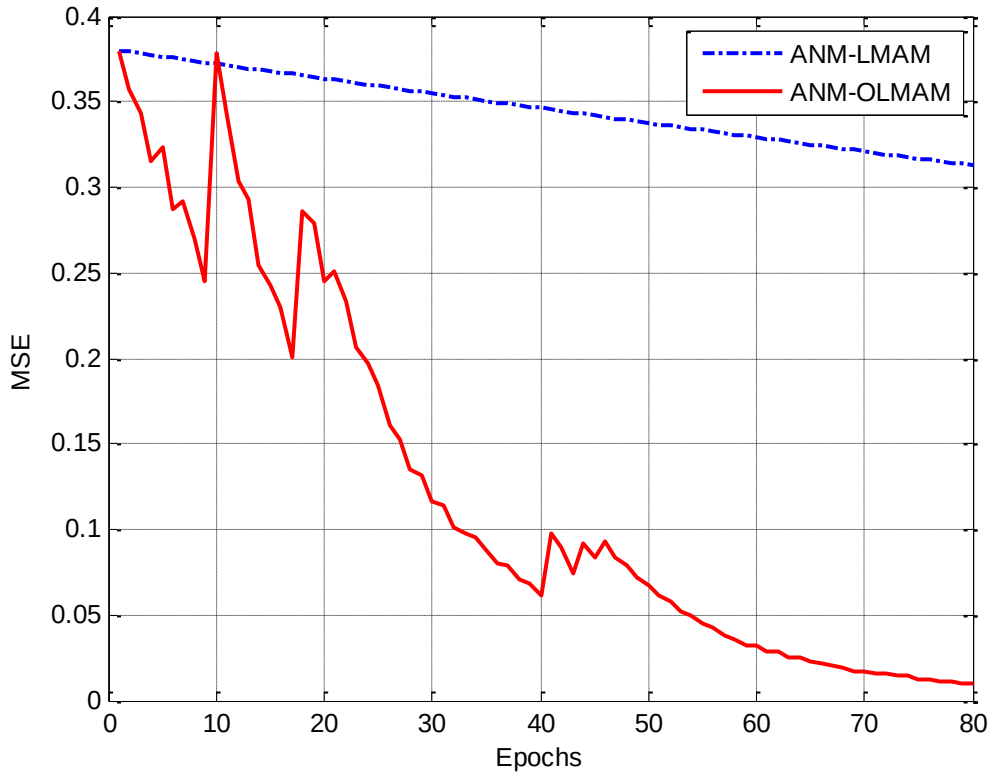
Algorithms	#hid	Conv (%)	MSE (%)	STD (%)	Epochs			
					Ave	Min	Max	Std
LM	1	0	37.536	2.797	-	-	-	-
	2	0	38.210	3.203	-	-	-	-
	5	0	36.283	2.999	-	-	-	-
	10	0	35.275	2.740	-	-	-	-
LMAM	1	0	37.933	2.962	-	-	-	-
	2	0	37.012	2.719	-	-	-	-
	5	0	26.394	4.621	-	-	-	-
	10	0	34.434	2.193	-	-	-	-
ANM-LMAM	**1**	**93**	**1.388**	**1.925**	**651**	**394**	**834**	**89**
	2	**95**	**1.868**	**1.203**	**627**	**413**	**869**	**71**
	5	**85**	**1.823**	**2.337**	**563**	**139**	**1073**	**114**
	10	**70**	**1.002**	**1.306**	**592**	**224**	**964**	**119**
OLMAM	1	0	37.933	2.714	-	-	-	-
	2	0	27.297	6.398	-	-	-	-
	5	0	26.973	5.023	-	-	-	-
	10	0	25.409	2.711	-	-	-	-
ANM-OLMAM	**1**	**98**	**0.960**	**0.046**	**64**	**39**	**293**	**35**
	2	**98**	**0.966**	**0.049**	**68**	**41**	**268**	**35**
	5	**99**	**0.958**	**0.037**	**54**	**35**	**91**	**12**
	10	**100**	**0.950**	**0.033**	**53**	**37**	**159**	**14**


ANM-LMAM
ANM-OLMAM
MSE
Epochs
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250
300
350
400

(a) FFTD


ANM-LMAM
ANM-OLMAM
MSE
Epochs
0.36
0.34
0.32
0.3
0.28
0.26
0.24
0.22
0.2
0.18
0
50
100
150
200
250
300
350
400

(b) LRN



(c) NARX

Figure 7.4 Examples of convergence behaviours of ANM-LMAM and ANM-OLMAM in the P10 problem for three RNNs

7.4.2 Sequence Classification Problem

Tables 7.9-7.11 show the results of FFTD, LRN and NARX networks for 3 monotone LM methods and the proposed nonmonotone approaches, while examples of learning behaviours for the three RNN architectures are provided in Figure 7.5. From the numerical results of Tables 7.9-7.11, the smallest improvements made by our proposed nonmonotone algorithms are about 16.6% in MSE and 20.7% in CE, while 40.5% in MSE and 85.9% in CE for the largest improvements. More details for different RNNs are provided in Table 7.12.

Table 7.9 Average performance for **FFTD** networks in the **SC** problem: class of LM.

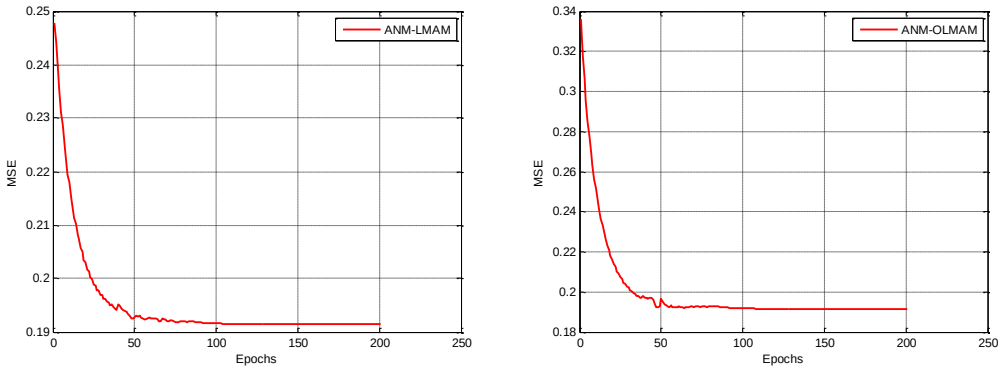
Algorithms	#hid	MSE (%)		CE (%)	
		Train/STD	Test/STD	Train/STD	Test/STD
LM	5	40.259/12.102	40.134/14.488	85.325/13.712	64.205/22.274
	10	42.174/13.374	42.472/12.843	88.862/14.537	69.438/23.780
	15	40.236/11.896	40.715/15.325	87.241/14.661	65.164/23.028
LMAM	5	39.987/12.558	39.289/12.301	86.478/14.070	62.082/21.837.
	10	40.753/12.147	40.693/12.476	89.212/14.525	66.315/23.342
	15	40.224/12.384	40.235/11.706	86.473/14.193	67.288/24.826
ANM-LMAM	**5**	**24.000/8.734**	**22.032/8.993**	**63.837/7.887**	**39.356/14.782**
	10	**24.874/9.383**	**22.740/9.572**	**62.759/7.030**	**36.247/15.013**
	15	**22.146/7.070**	**20.780/8.094**	**65.502/8.127**	**38.411/14.339**
OLMAM	5	39.624/14.307	38.908/16.060	85.596/15.673	61.027/22.947
	10	39.509/13.856	39.489/14.986	84.956/14.918	60.370/21.892
	15	39.687/14.482	39.703/15.344	85.108/15.702	65.712/24.380
ANM-OLMAM	**5**	**19.226/9.004**	**17.001/8.991**	**51.724/6.285**	**31.507/10.807**
	10	**19.561/9.153**	**17.301/9.481**	**52.207/5.432**	**32.041/11.427**
	15	**19.826/9.543**	**17.746/9.720**	**53.936/5.046**	**32.575/11.921**

Table 7.10 Average performance for **LRN** networks in the **SC** problem: class of LM.

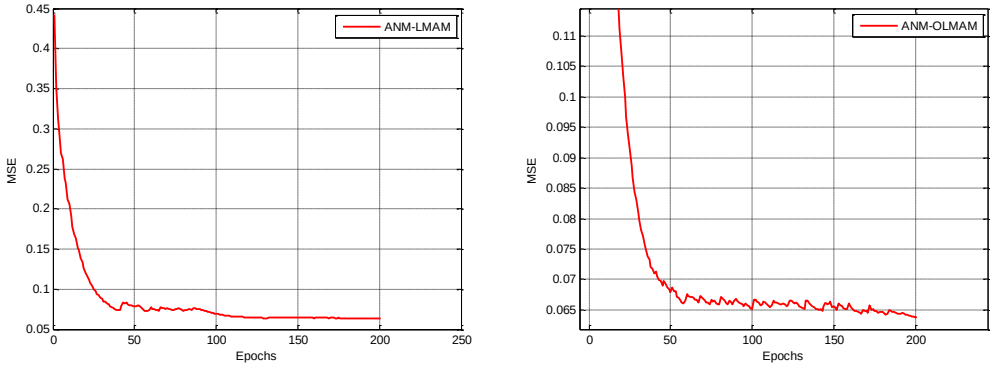
Algorithms	#hid	MSE (%)		CE (%)	
		Train/STD	Test/STD	Train/STD	Test/STD
LM	5	42.250/11.995	42.404/12.676	87.877/14.147	69.466/21.303
	10	41.282/11.036	41.662/11.580	87.030/12.993	63.904/24.677
	15	39.715/9.495	39.531/9.177	89.414/13.738	66.411/22.581
LMAM	5	42.328/12.163	42.781/12.277	88.626/11.945	65.685/23.227
	10	39.633/12.726	39.793/14.971	87.591/15.261	64.849/22.031
	15	40.044/11.839	39.701/11.887	88.789/10.704	66.849/21.364
ANM-LMAM	**5**	**20.817/7.004**	**17.214/7.358**	**66.094/8.397**	**31.973/12.786**
	10	**24.802/8.121**	**22.182/8.778**	**63.212/7.401**	**29.685/11.005**
	15	**25.011/9.730**	**22.592/9.170**	**64.261/7.997**	**29.219/10.792**
OLMAM	5	41.999/11.868	42.459/11.913	88.571/11.077	65.164/22.899
	10	39.546/12.343	39.230/14.577	85.118/14.981	66.644/23.630
	15	41.471/11.395	41.506/12.257	88.153/10.219	64.205/23.388
ANM-OLMAM	**5**	**6.316/1.983**	**5.508/1.307**	**24.631/3.975**	**6.095/1.411**
	10	**6.320/2.017**	**5.441/1.043**	**24.631/3.881**	**5.958/1.103**
	15	**7.133/3.872**	**6.136/1.729**	**26.404/4.585**	**8.095/2.678**

Table 7.11 Average performance for **NARX** networks in the **SC** problem: class of LM.

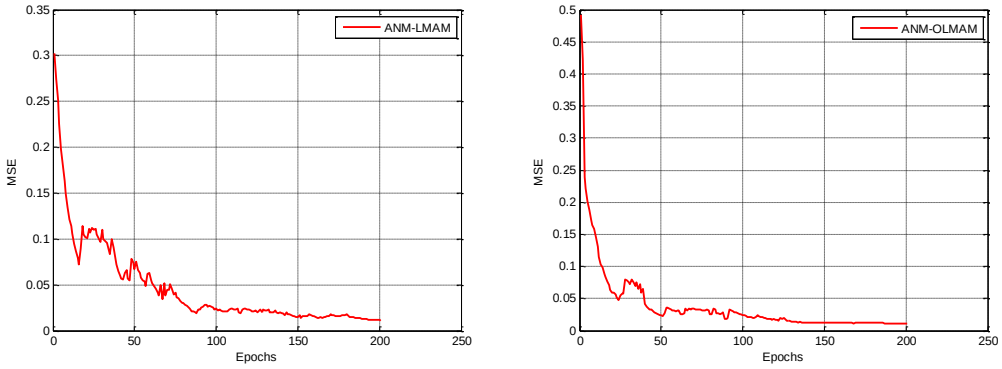
Algorithms	#hid	MSE (%)		CE (%)	
		Train/STD	Test/STD	Train/STD	Test/STD
LM	5	40.161/11.465	39.947/12.602	86.172/14.627	65.082/22.536
	10	41.161/12.361	40.947/14.124	85.443/15.478	67.315/23.674
	15	41.555/11.840	42.561/14.100	88.478/14.183	63.863/23.801
LMAM	5	42.385/11.614	42.101/12.447	88.000/10.320	69.890/22.933
	10	40.192/10.835	39.942/9.997	86.517/9.776	65.589/20.335
	15	40.243/10.361	39.949/9.982	87.227/9.871	64.082/21.632
ANM-LMAM	**5**	**18.921/6.949**	**22.998/8.030**	**69.069/8.825**	**45.397/19.803**
	10	**11.701/4.072**	**13.277/3.718**	**63.833/6.579**	**38.658/14.906**
	15	**8.850/5.100**	**17.938/6.998**	**66.493/6.947**	**42.493/17.281**
OLMAM	5	40.713/10.074	41.600/11.837	88.044/10.733	65.233/21.739
	10	38.586/9.003	38.431/9.238	85.512/8.276	64.466/20.117
	15	40.268/9.989	40.145/11.204	89.537/11.372	66.137/21.472
ANM-OLMAM	**5**	**0.617/0.226**	**12.630/3.782**	**2.004/0.973**	**19.397/4.554**
	10	**0.347/0.183**	**10.954/1.993**	**1.113/1.207**	**15.712/3.102**
	15	**0.180/0.087**	**11.083/3.186**	**0.758/0.367**	**16.247/3.972**



(a) FFTD



(b) LRN



(c) NARX

Figure 7.5 Examples of convergence behaviours of ANM-LMAM and ANM-OLMAM in the SC problem for three RNN architectures

Table 7.12 Average improvement achieved by the nonmonotone LM methods over their monotone counterparts for the SC problem

Algorithms	RNN	MSE (%)		CE (%)	
		Train	Test	Train	Test
ANM-LMAM	FFTD	16.648	18.222	23.355	27.224
	LRN	17.125	20.096	23.813	35.502
	NARX	27.783	22.953	20.783	24.338
ANM-OLMAM	FFTD	20.484	22.419	34.016	32.310
	LRN	33.969	34.956	63.095	58.905
	NARX	40.559	29.108	85.956	49.402

Improvements in terms of both MSE and CE were observed for all RNN architectures when nonmonotone learning methods were used. Taking the average CE in testing as an example, results for FFTD networks in Table 7.9 show that networks trained with the ANM-LMAM and ANM-OLMAM methods exhibit significant reduction in CE compared with the original monotone methods. As shown in Tables 7.10 and 7.11 improvements for LRNs and NARX networks are large as well. The largest improvement in testing CE was achieved for ANM-OLMAM-trained LRNs using 5 hidden nodes: a difference of 59.1% in CE compared to OLMAM-trained LRNs.

7.4.3 Sequence Learning Problem

Simulation results of the SL problem are shown in Tables 7.13-7.15, for FFTD, LRN and NARX networks, respectively, while Figure 7.6 provides examples of learning behaviours. Tables 7.16-7.18 exhibit results when increasing the number of epochs from 23 to 200 and then to 1000 epochs. Table 7.19 summaries the average improvements of our proposed ANMLM methods for the three RNNs.

The results for FFTD networks in Table 7.13 show that improvements in terms of MSE (%) in testing for ANM-LMAM and ANM-OLMAM range from 0.5% to 3.9% and from 15.4% to 16.5%, respectively. Improvements in Table 7.14 appear to be larger for LRN networks, ranging from 0.6% to 14.5% for ANM-LMAM and from 17.2% to 32.1% for ANM-OLMAM. In Table 7.15, ANM-LMAM and ANM-OLMAM trained NARX networks are 3.0% and 31.4% better respectively than networks trained with the original monotone versions. Figure 7.6 provides an example of nonmonotone learning behaviour for NARX showing how ANM-OLMAM achieves a relative smaller MSE than ANM-LMAM.

Table 7.13 Average performance for **FFTD** networks in the **SL** problem: class of LM.

Algorithms	#hid	Training		Testing	
		MSE (%)	STD (%)	MSE (%)	STD (%)
LM	1	44.045	23.873	43.997	24.023
	2	44.388	24.018	44.525	25.129
	5	40.646	21.996	40.669	21.634
	7	41.108	22.726	40.998	21.836
	10	41.063	22.549	41.199	22.037
LMAM	1	46.545	24.518	46.620	24.001
	2	42.970	23.076	42.966	24.336
	5	41.317	22.672	41.280	24.428
	7	41.231	22.901	41.051	22.693
	10	41.559	22.892	41.373	23.152
ANM-LMAM	**1**	**42.513**	**21.373**	**42.662**	**22.857**
	2	**42.045**	**22.167**	**42.062**	**22.263**
	5	**40.792**	**20.637**	**40.756**	**21.439**
	7	**40.708**	**20.552**	**40.530**	**21.097**
	10	**41.034**	**21.348**	**40.848**	**21.652**

Table 7.13 Average performance for **FFTD** networks in the **SL** problem: class of LM (cont'd).

Algorithms	#hid	Training		Testing	
		MSE (%)	STD (%)	MSE (%)	STD (%)
OLMAM	1	46.437	24.052	46.547	23.919
	2	41.527	22.378	41.511	22.027
	5	41.273	23.551	41.294	22.073
	7	40.434	22.753	40.517	22.784
	10	39.753	22.004	39.836	21.712
ANM-OLMAM	**1**	**30.070**	**17.233**	**30.927**	**18.742**
	2	**27.304**	**14.592**	**27.305**	**14.172**
	5	**25.818**	**13.848**	**25.871**	**14.335**
	7	**25.089**	**13.863**	**25.089**	**15.027**
	10	**24.932**	**12.045**	**24.860**	**12.783**

Table 7.14 Average performance for **LRN** networks in the **SL** problem: class of LM.

Algorithms	#hid	Training		Testing	
		MSE (%)	STD (%)	MSE (%)	STD (%)
LM	1	46.447	23.847	46.639	23.299
	2	43.809	22.714	44.011	21.076
	5	42.503	22.378	42.468	20.281
	7	40.437	22.530	40.554	20.949
	10	40.982	21.029	40.828	20.027

Table 7.14 Average performance for **LRN** networks in the **SL** problem: class of LM (cont'd).

Algorithms	#hid	Training		Testing	
		MSE (%)	STD (%)	MSE (%)	STD (%)
LMAM	1	46.583	23.427	46.765	24.176
	2	42.813	23.129	42.885	23.853
	5	41.933	21.612	41.686	22.394
	7	40.424	21.238	40.454	21.227
	10	38.782	20.857	38.703	20.788
ANM-LMAM	**1**	**45.542**	**22.145**	**45.718**	**21.635**
	2	**41.798**	**21.623**	**41.957**	**22.291**
	5	**41.253**	**21.804**	**41.028**	**21.982**
	7	**39.763**	**20.061**	**39.813**	**19.169**
	10	**24.209**	**18.776**	**24.229**	**19.438**
OLMAM	1	46.927	22.712	46.564	23.927
	2	43.358	21.489	43.592	21.367
	5	40.695	19.687	40.711	20.474
	7	40.966	20.323	40.994	20.755
	10	39.995	19.128	39.795	19.762
ANM-OLMAM	**1**	**26.482**	**17.448**	**26.815**	**17.629**
	2	**24.946**	**16.002**	**25.667**	**17.241**
	5	**20.555**	**14.127**	**21.794**	**14.924**
	7	**17.068**	**11.341**	**18.302**	**11.877**
	10	**10.203**	**7.865**	**10.867**	**8.048**

Table 7.15 Average performance for **NARX** networks in the **SL** problem: class of LM.

Algorithms	#hid	Training		Testing	
		MSE (%)	STD (%)	MSE (%)	STD (%)
LM	1	47.467	23.129	47.981	23.730
	2	42.918	22.908	43.233	22.675
	5	40.843	21.437	40.728	21.661
	7	39.012	21..010	39.548	22.476
	10	40.672	22.105	40.602	22.240
LMAM	1	46.665	22.672	46.176	23.237
	2	42.892	21.859	43.249	22.541
	5	41.548	21.178	41.805	21.704
	7	41.713	22.603	41.142	23.813
	10	40.813	21.230	40.723	21.392
ANM-LMAM	**1**	**45.784**	**21.917**	**46.132**	**22.342**
	2	**43.331**	**21.265**	**43.505**	**21.493**
	5	**39.427**	**20.222**	**38.871**	**19.904**
	7	**39.190**	**20.197**	**38.631**	**19.737**
	10	**40.039**	21.989	**39.949**	**20.556**

Table 7.15 Average performance for **NARX** networks in the **SL** problem: class of LM (cont'd).

Algorithms	#hid	Training		Testing	
		MSE (%)	STD (%)	MSE (%)	STD (%)
OLMAM	1	45.729	21.007	45.866	22.571
	2	44.246	20.379	43.918	22.097
	5	41.380	20.827	41.213	20.329
	7	40.071	20.520	40.027	20.125
	10	40.672	21.833	40.602	20.578
ANM-OLMAM	**1**	**21.773**	**12.487**	**22.142**	**13.625**
	2	**14.937**	**9.843**	**15.552**	**11.450**
	5	**11.540**	**7.204**	**11.831**	**7.925**
	7	**12.836**	**8.746**	**12.919**	**9.186**
	10	**9.115**	**5.044**	**9.374**	**6.458**

Referring to the errors for each prediction, this tends to be high when predicting consonants and low when predicting vowels. Given the semi-random nature of the sequence, this behaviour is not unusual: once a network has received a consonant at the input, it learns to predict the identity of the following vowel but at the end of the vowel sequence it has no way to predict what the next consonant will be; thus it produces a high error at these time points. In order to investigate the generalisation performance of the RNNs further we conducted additional tests using RNNs with 10-hidden nodes. These were trained for 1000 epochs and were tested using a randomly generated sequence of length 30 that conformed to the same regularities

that underlie the training sequence. In Tables 7.16-7-18, results are presented in the form of MSE/CE (in percentage) obtained after 200 and 1000 epochs in both training and testing. Note that the notation "--" used in these tables indicates that the monotone LMAM and OLMAM failed to train the RNNs because they got stuck, and as the result, both the training and testing MSE/CE values were identical to the ones reached at the 200th-epoch. In contrast, results for the nonmonotone methods indicate that they have potential to improve error performance as training progresses successfully beyond 200 epochs (up to the termination condition of 1000 epochs). This allows ANM-LMAM and ANM-OLMAM to achieve significant reductions in the MSE/CE (%) values for all RNN architectures in this task.

Table 7.16 Results of additional simulations for FFTD networks in the SL problem: class of LM.

Algorithms	200-epoch		1000-epoch	
	Train (MSE/CE)	Test (MSE/CE)	Train (MSE/CE)	Test (MSE/CE)
LMAM	43.7/91.7	45.0/93.6	--	--
ANM-LMAM	37.3/83.6	39.7/86.8	22.9/67.1	22.8/69.4
OLMAM	41.5/88.9	43.4/91.3	--	--
ANM-OLMAM	33.2/74.1	34.4/78.2	21.7/64.0	22.3/66.8

Table 7.17 Results of additional simulations for LRN networks in the SL problem: class of LM.

Algorithms	200-epoch		1000-epoch	
	Train (MSE/CE)	Test (MSE/CE)	Train (MSE/CE)	Test (MSE/CE)
LMAM	40.6/88.1	39.5/87.4	--	--
ANM-LMAM	22.7/79.9	23.1/81.2	16.3/47.6	18.2/53.2
OLMAM	36.5/84.4	34.9/85.0	--	--
ANM-OLMAM	9.0/45.8	9.4/48.5	5.6/33.3	7.9/42.5

Table 7.18 Results of additional simulations for NARX networks in the SL problem: class of LM.

Algorithms	200-epoch		1000-epoch	
	Train (MSE/CE)	Test (MSE/CE)	Train (MSE/CE)	Test (MSE/CE)
LMAM	28.5/89.1	28.1/72.4	--	--
ANM-LMAM	23.3/81.0	22.6/54.5	9.7/21.3	12.4/25.1
OLMAM	26.7/86.7	26.2/69.8	--	--
ANM-OLMAM	7.3/34.2	7.7/43.1	4.2/12.6	5.8/16.9

Table 7.19 Average improvements achieved by the nonmonotone LM methods over their monotone counterparts for the SL problem

Algorithms	RNN	MSE (%)	
		Train	Test
ANM-LMAM	FFTD	1.306	1.286
	LRN	3.594	3.549
	NARX	1.172	1.201
ANM-OLMAM	FFTD	15.242	15.131
	LRN	22.256	21.410
	NARX	28.379	27.961

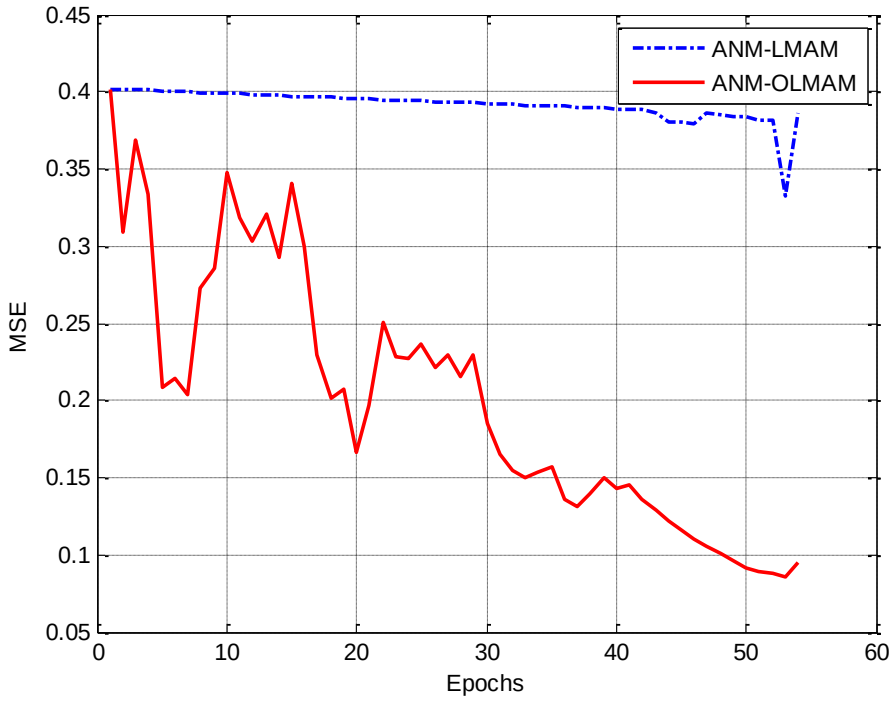


Figure 7.6 Example of convergence behaviours of ANM-LMAM and ANM-OLMAM in the SL problem for NARX network

7.4 Summary and Contribution of the Chapter

The chapter looked at the classical LM methods and explored its application in training ANNs. A discussion on the global convergence of the nonmonotone versions, was provided and two adaptive nonmonotone LM approaches were proposed to investigate the potential of incorporating nonmonotone strategies in LM methods. Experimental results show that our nonmonotone approaches dramatically improve LM methods performance, i.e. not only by increasing the speed of learning in the N-bit parity problems, but also provide more efficient training in two difficult real-world applications.

In this chapter we developed nonmonotone learning algorithms by equipping the Levenberg-Marquardt algorithms proposed in [7] with nonmonotone strategy. In this way we extended the application of the LM with adaptive momentum (LMAM) and the optimized LMAM (OLMAM) methods to training recurrent neural networks and enhanced their performance through the use of an adaptive nonmonotone strategy. We examined the behaviour of these nonmonotone variants in a set of experiments that involved training RNNs of various dimensions belonging to three different architectures, i.e., FFTD, ERN, and NARX, in symbolic sequence processing problems. Results are promising as these algorithmic extensions outperformed the original monotone versions managing to train effectively RNNs of smaller size than the original methods and produced better results in testing using unknown data.

Chapter 8

Conclusions and Future Works

Recurrent networks constitute an elegant way of increasing the processing capabilities of feedforward networks to deal with high-dimensional data in the form of sequences of real vectors and are well known for their power to model temporal dependencies and process sequences for classification, recognition, and transduction. Modern RNNs architectures are capable of learning to solve many previously not-learnable tasks, even in partially observable environments. From the literature reviews of architectures and learning algorithms for RNNs in Chapter 2, a more general classification scheme of recurrent architecture was proposed [148] and highlighted the fact that in the literature very few attempts have been made to train RNNs by second-order approaches.

By providing a general formulation for the neural networks' learning task in Chapter 3, the nature of training neural networks can be considered as a minimisation problem of unconstrained optimisation. An overview of traditional and classical methods in the field of nonlinear unconstrained optimisation was then provided, followed by the introductions of the new class of approaches, i.e. the nonmonotone learning, which is the main feature of our proposed algorithms.

The aim of this research was to design novel algorithms for training RNNs effectively and efficiently, in order to tackle the processing problems of temporal sequences. The new algorithms, i.e. the nonmonotone Rprop, CG, BFGS, and LM were proposed in Chapters 4-7, respectively.

The reported experimental results show that the goal of this research has been successfully achieved by the adaptive nonmonotone approaches, in terms of faster and better rate of convergence, and smaller average training and testing errors.

The proposed algorithms apply an adaptive way to determine the size of the nonmonotone learning horizon, which prevents the manual setting of application-dependent constant, i.e. size of learning horizon, alleviating the need for user defined values. The application of nonmonotone conditions and the adaptive mechanism provide deterministic nonmonotone training that exploits curvature information, and as a result, the influence of application-dependent settings can be eliminated to an extremely small degree.

Despite a variety of available measures to use in these applications, such as the normalized mean squared error (NMSE), the root mean squared error (RMSE), and the mean absolute percentage error (MAPE), we have used the MSE as this was used in the other works in the literature and thus makes comparison of our approach with other approaches quite straightforward.

In the future we are planning to extend our list of applications, e.g. solve the Reading Aloud (RA) problem for three different types of RNNs (FFTD, LRN and NARX) using LM-type of methods (these methods frequently encounter out-of-memory

run-time errors in implementations when high dimensions are required as in RA), and RA for LRN networks using the JRprop method. Simulations of RNNs in general, and for the RA problem in particular, require more computational resources than training static neural networks, and therefore, we are planning to convert our Matlab codes into other programming languages such as C, or, to upgrade our computational resources in order to tackle them.

Another extension of our work is to produce an experimental comparison of nonmonotone conditions, since most of all proposed nonmonotone approaches focus on proving the properties of their global convergence, and rarely make any comparison with each other. In Chapter 3 of this thesis we provided an overview of these approaches, which covers approaches from the first ones proposed up to the latest published work in this area. Since choosing the right recurrent architecture and learning algorithm is application-dependent, the choice of applying nonmonotone conditions is still an open problem in the field of nonlinear optimisation. According to our experimental results, the proposed adaptive approach can alleviate this application-dependent nature of nonmonotone learning and reduce the number of user-defined parameters or constants to a relative small amount.

In our future work we are also planning to extend the applicability of the nonmonotone approach to other methods of the Rprop family, e.g. develop a nonmonotone revision of the GRprop [9] and the GJRprop [10].

As the thesis focused on temporal sequence processing problem, and made simulations on real-world applications of thousands adjustable parameters, as indicated in Appendix A.1, another part of our future agenda is to extend the

application of nonmonotone methods in other domains such as bioinformatics where several important problems can be modelled and processed as symbolic sequences. This will hopefully reveal further information about the ability and limitations of our proposed nonmonotone learning algorithms.

Appendix

A.1 Experimental Applications

The four simulated applications are described in this section, while each of them with relative reference, problem descriptions, and the setting of parameters and stopping criteria of the training process. Table A.1 provides a summary of the number of adjustable parameters for the three RNN architectures with different amounts of hidden nodes, which are uniformly used in this thesis. In addition, all the weights and biases of the RNNs applied in this thesis are randomly initialised in the range of [-1, 1].

A.1.1 The 5-bit and 10-bit Parity Problems

References: [23][40][42][41][51][84][94][98][110][125][136][158] [183]

Problem Description: *The class of N-bit parity problems, which are typically nonlinear separable and possess a multitude of local minima, have been widely used to verify performances of novel training algorithms, such as in the latest works [41][84][134][154][182]. In this problem, a one-bit output string, containing either 0 or 1, is generated from an N-bit long input string that consists of values from the set {0,1}, implementing a parity function. There are 32 and 1024 patterns for the 5-bit (P5) and the 10-bit (P10) problems, respectively.*

P5 patterns are of the form

$$1\ 0\ 1\ 0\ 1 \rightarrow 1$$

while P10 patterns are

$$1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0 \rightarrow 1,$$

where the binary string in the left-side of the arrow denotes an input pattern and the 1s on the right side of the arrow denotes the desired output.

Parameters and Stopping Criteria: Training goals for both P5 and P10 are set to 0.01, within 2000 training epochs for P5 and 4000 for P10. When any of these two conditions, i.e. training goal or epoch, is reached, the training process then stops.

A.1.2 The Sequence Classification Problem

Reference: [116]

Problem Description: This problem concerns labelling the task from a sequence of events presented at the input of the RNNs. In sequence processing problems of this type, a sequence $s = \{s_1, s_2, \ldots s_n\}$ is read and a single value, label or pattern c, taken from a suitable set C, is computed from it. A tracking engine monitors states of interface objects in a personalised system and produces this sequence of events as a user interacts with the software. A task consists of a sequence of events, such as opening a browser window, searching for information, saving information from the search results or storing bookmarks.

This is a challenging problem as individual users may execute the same task in slightly different ways, i.e. they generate slightly different event sequences where, for example, one chain of events generated for a task might be more complex from another chain of events generated for the same task. Moreover, certain events (e.g. a mouse right click event) might occur during execution of more than one task making the sequence more "noisy". A sequence of 203 events was used for training and a sequence of 73 events, which was generated by a different user, was used for testing. Each event is coded by a 36-dimensional binary vector and a task by a 3-dimensional vector. For example, two of the training vectors are shown below, in the form of "36-bit input → 3-bit output":

#202: 0000 0000 0000 0001 0000 0000 0000 0000 0000 → 101

#203: 0000 1000 0000 0000 0000 0000 0000 0000 0000 → 001.

The training sequence is wrapped around so that the first event of the sequence is presented again after the last one.

Parameters and Stopping Criteria: For this problem, all 10-hidden-node RNNs were trained 200 epochs and the training goal is set to MSE=0. The amounts of input/output delays are 5 for FFTD and 1-input-1-output delays for NARX.

A.1.3 The Sequence Learning Problem

Reference: [124]

Problem Description: This problem belongs to the class of sequence continuation or prediction problems, where a processor reads a sequence s_1, s_2, … s_n and produces at the output a possible continuation of this sequence s_{n+1}, s_{n+2}… Interesting applications of this approach can be found in time series prediction problems, where the aim is to predict the future behaviour of a system, and in predictive coding and compression. If the prediction is good enough in terms of a mean-squared-error type criterion, the difference between the predicted continuation of the sequence and its actual continuation may be transmitted using a channel with a lower bandwidth or a lower bit rate such as in applications of speech coding in digital cellular phone systems [183].

In the particular instance of the problem, a symbolic sequence is used [57][124]. It concerns a set of 6 letters, {a,b,d,i,g,u}, where each letter is coded by a 4-dimensional binary vector. The letters formulate the strings {ba}, {dii} and {guuu}, and all possible permutations of these 3 strings are legal. For example, part of a sequence that one can generate using this alphabet is:

babaguuudiiguuu…→

1100 0100 1100 0100 1001 0001 0001 0001 1010 0010 0010 1001 0001 0001 0001…

This type of sequence is semi-random because the consonants occur randomly but the identity and number of vowels is regular.

In terms of sequence processing application, this is a symbolic sequence prediction task which requires predicting successive letters in a semi-random sequence of 1000 words and each word consists of one of the above 3 consonant-vowel combinations. This type of serial patterns are longer in duration than those produced in the parity sequences discussed above, they are of variable length so that a prediction depends upon a variable amount of temporal context.

The training regime involves presenting each one of the 4-bit input vectors in sequence. The task for the RNN is to predict the next input, and thus gradually is learning to predict the vowels from the consonants and also the fact that a consonant follows the vowels, although it might not be possible to predict which one. So the best one could expect from an RNN is to predict that all three consonants are equally likely to occur in word-initial positions but once one of them is received at the input then the identity and number of the following vowels should be predicted with accuracy. A sequence of length 2993 (4×2993 binary serial patterns) is used for training, as in [124]. RNNs are then tested on a shorter sequence of length 30 (4×30 patterns), which conforms to the same regularities that underlie the training sequence but is created from a different initial randomization. The training sequence is wrapped around so that the first pattern is presented again after the last one.

Parameters and Stopping Criteria: All RNNs apply 10 hidden nodes, $\delta = 0.01$, and training goal is 0.01, where training processes 23 epochs only, as in the original work [124]. Amounts of delays are as following. NARX networks apply (5,5) input-output delays and 5 for input-delays for FFTD. Furthermore, 300-epoch simulations were also carried out, for the sake of comparison of behaviours.

A.1.4 The Reading Aloud Problem

Reference: [158]

Problem Description: This task concerns learning the mapping of a set of orthographic representation to their phonological forms. Both subsets of orthography and phonology have 3 different parts, i.e. onset, vowel and coda, with 30, 27 and 48, and 23, 14 and 24 possible characters, respectively. There are 105-dimensional input patterns and 61-dimensional output patterns, while the training dataset has 2998 patterns. Ideally, a specially designed RNN architecture with 100 hidden nodes, which is described in detail in [158], is needed to solve this problem. Although in the original work there is no special testing dataset, we choose 30 words which are not included in the original training set from an online dictionary in order to verify our algorithm's generalisation ability.

Parameters and Stopping Criteria: 10 hidden nodes and 300 training epochs are used for this problem, while δ is set to the same value of the SL problem, i.e., 0.01. 2 delays were applied, i.e., 2 input delays for FFTD, and 2-input-2-output delays for NARX.

Table A.1 summarises the numbers of RNN adjustable parameters used in the experiments of the thesis: *I/O* represents the number of the input and output nodes of the RNNs, *#hid* is the number of hidden nodes for each problem and the total number of RNNs' adjustable parameters (weights plus biases) for each one of the architectures discussed in the thesis (i.e. FFTD, LRN and NARX).

Table A.1 Summary of RNN free parameters for the tested problems

Problem (I/O)	#hid	RNN Architecture		
		FFTD [196][197]	LRN [57][85]	NARX [128][137]
P5 (5/1)	1	13	9	15
	2	25	19	29
	5	61	61	71
	7	85	99	99
P10 (10/1)	1	23	14	25
	2	45	29	49
	5	111	86	121
	7	155	134	169
	10	221	221	241
SC (36/3)	5	383	228	304
	10	763	503	433
	15	1143	828	648
SL (4/4)	1	17	14	25
	2	30	26	46
	5	69	74	109
	7	95	116	151
	10	134	194	214
RA (105/61)	5	1421	921	2031
	10	2781	1831	4001

A.2 The Nonmonotone LM Algorithms

For the sake of discussion of global convergence for nonmonotone LM approaches in section 7.2, the two algorithms proposed in [213], i.e. NMLM1 and NMLM2, are reviewed below.

A.2.1 the NMLM1 Algorithm

0. Choose $\tilde{a}_0 > 0, \tilde{\eta} > 0, \tilde{\gamma} > 1, 1 > \tilde{\mu} > 0,$ an integer $M > 0,$ an initial point $x_0,$, and a continuous, positive-definite and diagonal matrix function $W(x)$;

1. Set $k = 0,$ compute $f(0) = f(x_0)$;

2. Calculate A_k and $g_k = g(x_k)$. If $\|g_k\| = 0,$ set $x^* = x_k$ and stop; otherwise, compute $T_k = A_k^T A_k$;

3. Solve the problem $(T_k + \tilde{\alpha}_k W_k)\tilde{\mu} = -g_k$ to obtain the solution $\tilde{\mu}_k = \tilde{\mu}(\tilde{\alpha}_k, x_k)$; compute $f_{k+1} = f(x_k + \tilde{\mu}_k)$;

4. Compute

$$\hat{\sigma}_{l(k)} = \hat{\sigma}(\tilde{\alpha}_k, x_{l(k)}) = (f_{l(k)} - f_{k+1}) \Big/ \left[\hat{f}(x_k, x_k) - \hat{f}(x_k + \tilde{\mu}_k, x_k)\right],$$

$$\hat{\mu}_k = \begin{cases} \tilde{\mu}, & \text{if } M = 0; \\ \min\left\{\tilde{\mu}, \tilde{\eta}\|g_k\|\|\tilde{\mu}_k\| \Big/ \left[\hat{f}(x_k, x_k) - \hat{f}(x_k + \tilde{\mu}_k, x_k)\right]\right\}, & \text{otherwise;} \end{cases}$$

where $\hat{f}(x, x_k) = f_k + g_k^T(x - x_k) + \frac{1}{2}(x - x_k)T_k(x - x_k)$, and

$f_{l(k)} = f(x_{l(k)}) = \max\{f(x_k), f(x_{k-1}), \ldots, f(x_{k-M})\},$ with the convention that, for any negative integer $j, f(x_j)$ does not exist.

5. If $\hat{\sigma}_{l(k)} < \hat{\mu}_k$, set $\tilde{\alpha}_k = \tilde{\gamma}\tilde{\alpha}_k$ and go to Step 3; otherwise, set $x_{k+1} = x_k + \mu_k$, $\tilde{\alpha}_{k+1} = \tilde{\alpha}_k / \tilde{\gamma}$;

6. Set $k = k+1, k = k+1$; go to Step 2.

A.2.2 the NMLM2 Algorithm

This algorithm is similar to the NMLM1 Algorithm except that:

1. The $\hat{\alpha}_k$ and $\underset{\frown}{\mu}_k$ in the NMLM1 Algorithm are replaced by

$$\hat{\sigma}_{l(k)} = \hat{\sigma}\left(\tilde{\alpha}_k, x_{l(k)}\right) = \left(f_{l(k)} - f_{k+1}\right) \Big/ \left[\breve{f}\left(x_k, x_k\right) - \breve{f}\left(x_k + \tilde{\mu}_k, x_k\right)\right],$$

$$\hat{\mu}_k = \begin{cases} \tilde{\mu}, & \text{if } M = 0; \\ \min\left\{\tilde{\mu}, \tilde{\eta}\|g_k\|\|\tilde{\mu}_k\| \Big/ \left[\breve{f}\left(x_k, x_k\right) - \breve{f}\left(x_k + \tilde{\mu}_k, x_k\right)\right]\right\}, & \text{otherwise;} \end{cases}$$

where $\breve{f}\left(x, x_k\right) = f_k + g_k^T\left(x - x_k\right) + \frac{1}{2}\left(x - x_k\right)^T\left(T_k + \tilde{\alpha}_k W_k\right)\left(x - x_k\right)$;

2. When $\tilde{\sigma}_{l(k)} \geq \tilde{\mu}_k$, the parameter $\tilde{\alpha}_k$ in Step 5 is updated by

$\tilde{\alpha}_{k+1} = \max\left\{\tilde{\alpha}_k / \tilde{\gamma}, \alpha_{\min}\right\}$, where $\alpha_{\min}$ is a positive constant chosen at Step 0.

A.3 Publication List

Journal papers

1. C.-C. Peng and G.D. Magoulas (2008), "Advanced Adaptive Nonmonotone Conjugate Gradient Training Algorithm for Recurrent Neural Networks", *International Journal on Artificial Intelligence Tools (IJAIT)*, 17(5), pp. 963-984.
2. C.-C. Peng and G.D. Magoulas (2011), "Nonmonotone BFGS-trained Recurrent Neural Networks for Temporal Sequence Processing", *Applied Mathematics and Computation*, 217(12), pp. 5421-5441.
3. C.-C. Peng and G.D. Magoulas (forthcoming), "Improved Levenberg-Marquardt Algorithms for Training Recurrent Neural Networks Using Adaptive Nonmonotone Strategy", *Neural Computing and Applications*.
4. C.-C. Peng and G.D. Magoulas (under review), "Adaptive Nonmonotone Resilient Propagation Learning for Recurrent Neural Networks", *Applied Numerical Mathematics*.

Book chapter

5. C.-C. Peng and G.D. Magoulas (2008), "Sequence Processing with Recurrent Neural Networks", *Encyclopedia of Artificial Intelligence*, ISBN: 9781599048499, pp. 1411-1417.

Conference papers

6. C.-C. Peng and G.D. Magoulas (2007), "Effective Modification of BFGS

Method for Training Recurrent Neural Networks", *Proc. Conf. Numerical Analysis (NumAn'07)*, 3-7 September 2007, Kalamata, Greece, pp. 113-117.

7. C.-C. Peng and G.D. Magoulas (2007), "Adaptive Self-Scaling Non-Monotone BFGS Training Algorithm for Recurrent Neural Networks", *Proc. Int'l Conf. Artificial Neural Networks 2007 (ICANN'07)*, 9-13 September 2007, Porto, Portugal, pp. 259-268.
8. C.-C. Peng and G.D. Magoulas (2007), "Adaptive Nonmonotone Conjugate Gradient Training Algorithm for Recurrent Neural Networks", *Proc. 19th IEEE Int'l Conf. Tools with Artificial Intelligence 2007 (ICTAI'07)*, 29-31 October 2007, Patras, Greece, pp. 374-381.
9. C.-C. Peng and G.D. Magoulas (2009), "Nonmonotone Learning of Recurrent Neural Networks in Symbolic Sequence Processing Application", *Proc. 11th Int'l Conf. Engineering Applications of Neural Networks (EANN'09)*, 22-29 August 2009, London, England, pp. 325-335.

Workshop paper

10. C.-C. Peng and G.D. Magoulas (2007), "Second-order Algorithm Based on the BFGS Update Rule for Training Recurrent Neural Architectures", *Proc. 2007 UK Workshop on Computational Intelligence (UKCI)*, 23-24 July, London, 2007.

References

[1] Abraham, A. (2004). Meta learning evolutionary artificial neural networks. *Neurocomputing*, **56**, 1-38.

[2] Adamowski, J. F. (2008). Development of Levenberg-Marquardt, Resilient Back-Propagation, and Conjugate Gradient Powell-Beale Artificial Neural Networks for Peak Urban Water Demand Forecasting in Nicosia, Cyprus, *American Geophysical Union Fall Meeting Abstracts*, C874.

[3] Adeli, H. & Hunc, S.L. (1994). An adaptive conjugate gradient learning algorithm for efficient training of neural networks, *Applied Mathematics and Computation*, **62**(1), 81-102.

[4] Aizenberg, I., Paliy, D.V., Zurada, J.M. & Astola, J.T. (2008). Blur Identification by Multilayer Neural Network Based on Multivalued Neurons, *IEEE Trans. Neural Networks*, **19**(5), 883-898.

[5] Al-Baali, M. (1998). Numerical experience with a class of self-scaling quasi-Newton algorithms, *J. Optimization Theory and Applications*, **96**(3), 533–553.

[6] Alpsan, D., Towsey, M., Ozdamar, O., Tsoi, A.C. & Ghista, D.N. (1995). Efficiency of modified backpropagation and optimisation methods on a real-world medical problem. *Neural Networks*, **8**(6), 945-962.

[7] Ampazis, N. & Perantonis, S.J. (2002). Two highly efficient second-order algorithms for training feedforward networks. *IEEE Trans. Neural Networks*, **13**, 1064-1074.

[8] Anastasiadis, A., Magoulas, G.D. & Vrahatis, M.N. (2005a). New Globally Convergent Training Scheme Based on the Resilient Propagation Algorithm. *Neurocomputing*, **64**, 253-270.

[9] Anastasiadis, A., Magoulas, G.D. & Vrahatis, M.N. (2005b). Sign-based Learning Schemes for Pattern Classification. *Pattern Recognition Letters*, **26**, 1926–1936.

[10] Anastasiadis, A., Magoulas, G.D., & Vrahatis, M.N. (2006). Improved sign-based learning algorithm derived by the composite nonlinear Jacobi process. *Journal of Computational and Applied Mathematics*, **191**, 166-178.

[11] Anastasiadis, A.D., Magoulas, G.D. & Vrahatis, M.N. (2003). An efficient improvement of the Rprop algorithm. In: Gori & Marinai (eds.), *Artificial Neural Networks in Pattern Recognition*, *Proc. 1st Int'l Association of Pattern Recognition-TC3 Workshop*, Florence, Italy, September 2003, Firenze: Stampa Digitale, 197-201.

[12] Anastasiadis, A.D., Magoulas, G.D. & Vrahatis, M.N. (2004). A New Learning Rates Adaptation Strategy for the Resilient Propagation Algorithm. In: *Proc. European Symposium on Neural Networks (ESANN-04)*, Bruges, Belgium, 1-6.

[13] Antunes, C.M. & Oliveira, A.L. (2001). Temporal data mining: an overview. In: *Proc. KDD Workshop on Temporal Data Mining*, San Francisco, CA, 26 August 2001, 1-13.

[14] Armijo, L. (1966). Minimization of function having Lipschitz continuous first partial derivatives. *Pacific J. Mathematics*, **16**, 1-3.

[15] Asirvadam, V.S., McLoone, S.F. & Irwin, G.W. (2004). Memory efficient BFGS neural-network learning algorithms using MLP-network: a survey. In: *Proc. IEEE Int'l Conf. Control Application*. 586-591.

[16] Assaad, M., Boné, R. & Cardot, H. (2005). Study of the behaviour of a new boosting algorithm for recurrent neural networks. In: Duch W. et al. (eds.) *Proc. 15th Int'l Conf. Artificial Neural Networks (ICANN)*, 169-174.

[17] Bajramovic, F., Gruber, C. & Sick, B. (2004). A comparison of first- and second- order training algorithms for dynamic neural networks. In: *Proc. IEEE Int'l Joint Conf. Neural Networks*, 837-842.

[18] Baldi, P. (1993). Gradient descent learning algorithm overview: a general dynamical systems perspective. *IEEE Trans. Neural Networks*, **6**(1), 182-195.

[19] Baldi, P., Brunak, S., Soda, G. & Pollastri, G. (1999). Exploiting the past and the future in protein secondary structure prediction, *Bioinformatics*, **15**, 937-946.

[20] Battiti, R. (1992). First- and second-order methods for learning: between steepest descent and Newton's method. *Neural Computation*, **4**, 141-166.

[21] Becerikli, Y., Konar, A.F. & Samad, T. (2003). Intelligent optimal control with dynamic neural networks. *Neural Networks*, **16**, 251-259.

[22] Becerikli, Y., Oysal, Y. & Konar, A.F. (2004). Trajectory priming with dynamic fuzzy networks in nonlinear optimal control. *IEEE Trans. Neural Networks*, **15**(2), 383-394.

[23] Beigi, H.S.M. (1993). Neural networks learning through optimally conditioned quadratically convergent methods requiring no line search. In: *Proc. 36th Midwest Symposium on Circuits and Systems*, 109-112.

[24] Bengio, Y., Frasconia, P. & Gori, M. (1993). Recurrent neural networks for adaptive temporal processing. In: *Proc. 6th Italian Workshop on Parallel*

Architectures and Neural Networks, 85-117.

[25] Bengio, Y., Simard, P. & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks*, **5**(2), 157-166.

[26] Bhaya, A. & Kaszkurewicz, E. (2004). Steepest descent with momentum for quadratic functions is a version of the conjugate gradient method, *Neural Networks*, **17**, 65-71.

[27] Boné, R. & Cardot, H. (2005). Time delay learning by gradient descent in recurrent neural networks. In: Duch W. et al. (eds.) *Proc. 15th Int'l Conf. Artificial Neural Networks (ICANN)*, 175-180.

[28] Boné, R., Crucianu, M. & de Beauville, J.-P.A. (2002). Learning long-term dependencies by the selective addition of time-delayed connections to recurrent neural networks. *Neurocomputing*, **48**, 251-266.

[29] Brouwer, R.K. (2005). Training of a discrete recurrent neural network for sequence classification by using a helper FNN. *Soft Computing*, **9**, 749-756.

[30] Broyden, C.G. (1970). The convergence of a class of double-rank minimization algorithms. *J. The Institute of Mathematics and Its Applications*, **6**, 76-90.

[31] Budik, D. & Elhanany, I. (2006). TRTRL: a localized resource-efficient learning algorithm for recurrent neural networks. In: *Proc. 2006 IEEE Int'l Midwest Symposium on Circuits & Systems (MWSCAS)*.

[32] Campolucci, P., Piazza, F. & Uncini, A. (1995). On-line learning algorithms for neural networks with IIR synapses, In: *Proc. IEEE Int'l Conf. Neural Networks*, 865-870.

[33] Campolucci, P., Sunibettu, M., Uncini, A. & Piazza, F. (1998). New second-order algorithms for recurrent neural networks based on conjugate

gradient, In: *Proc. 2nd IEEE World Congress on Computational Intelligence*, 384-398.

[34] Campolucci, P., Uncini, A. & Piazza, F. (1997a). A unifying view of gradient calculations and learning for locally recurrent neural networks, In: *Proc. Italian Workshop on Neural Networks (WIRN97)*, Springer-Verlag Ed.

[35] Campolucci, P., Uncini, A. & Piazza, F. (1997b). A new IIR-MLP learning algorithm for on-line signal processing, In: *Proc. Int'l Conf. Acoustic Speech and Signal Processing (ICASSP97).*

[36] Campolucci, P., Uncini, A., Piazza, F. & Rao, B.D. (1999). On-line learning algorithms for locally recurrent neural networks. *IEEE Trans. Neural Networks*, **10**(2), 253-271.

[37] Caruana, R., Lawrence, S. & Giles, L. (2000). Overfitting in neural nets: Backpropagation, conjugate gradient and early stopping, In: *Proc. Neural and Information Processing Systems*, 402-408, MIT Press.

[38] Chang, W.F. & Mak, M.W. (1999). A conjugate gradient learning algorithm for recurrent neural networks, *Neurocomputing*, **24**, 173-189.

[39] Charalambous, C. (1992). A conjugate gradient algorithm for the efficient training of artificial neural networks. In: *IEE Proceedings Part G. 139*, 301 -310.

[40] Chella, A., Gentile, A., Sorbello, F. & Tarantino, A. (1993). Supervised learning for feed-forward neural networks: a new minimax approach for fast convergence. In: *Proc. IEEE Int'l Conf. Neural Networks*, 605-609.

[41] Chen, K., Xu, L. & Chi, H. (1999). Improved learning algorithms for mixture of experts in multiclass classification, *Neural Networks*, **12**, 1229-1252.

[42] Chen, O.T.-C. & Sheu, B.J. (1994). Optimization schemes for neural network

training. In: *Proc. IEEE Int'l Conf. Neural Networks*, 817-822.

[43] Chu, Y.-C. & Huang, J. (1999). A neural-network method for the nonlinear servomechanism problem. *IEEE Trans. Neural Networks*, **10**, 1412-1423.

[44] Clarke, F.H. (1990). *Optimization and Nonsmooth Analysis*, Philadelphia: SIAM.

[45] Dai, Y.H. (2002). A nonmonotone conjugate gradient algorithm for unconstrained optimization, *J. Systems Science and Complexity*, **15**(2), 139-145.

[46] Dai, Y.H. (2002). On the Nonmonotone Line Search, *J. Optimization Theory and Applications*, **112**(2), 315–330.

[47] Davidon, W.C. (1959). Variable metric method for minimization. *A.E.C. Research Develop. Report ANL-5990*, Argonne National Laboratory, Argonne, Illinois.

[48] Davidon, W.C. (1991). Variable metric method for minimization. *SIAM J. Optimization*, **1**(1), 1-17.

[49] De Jesus, O. & Hagan, M.T. (2001). Forward perturbation algorithm for a general class of recurrent network, *Proc. Int'l Joint Conf.*, vol.4, 2626-2631.

[50] Dennis, J.E. & Schnabel, R.B. (1989). A view of unconstrained optimization. In: Nemhauser et al. (eds) *Optimization*, Amsterdam: Elsevier Science, 1-72.

[51] Denton, J.W. & Hung, M.S. (1996). A comparison of nonlinear methods for supervised learning in multilayer feedforward neural nets. *European J. Operational Research*, **93,** 358-368.

[52] Dietterich, T.G. (2002). Machine learning for sequential data: a review. In: *Proc. Joint IAPR Workshop on Structural, Syntactic, and Statistical Pattern Recognition, Lecture Notes in Computer Science*, **2396**, 15-30.

[53] dos Santos, E.P. & Von Zuben, F.J. (1999). Improved second-order training algorithms for globally and partially recurrent neural networks, *Proc. Int'l Joint Conf. Neural Networks (IJCNN99)*, 3, 1501-1506.

[54] dos Santos, E.P. & von Zuben, F.J. (2000). Efficient second-order learning algorithms for discrete-time recurrent neural networks. In: Medsker& Jain (eds.): *Recurrent Neural Networks: Design and Applications*. New York: CRC Press, 47-75.

[55] Du, S.Q. & Chen, Y.Y. (2004). Convergence analysis of a class of nonmonotone conjugate gradient methods without sufficient decrease condition, *Chinese Quart. J. Mathematics*, **19**(2), 142-145.

[56] Duch, W. & Korczak, J. (1998). Optimisation and global minimisation methods suitable for neural networks, *Neural Computing Surveys*, **2**.

[57] Elman, J.L. (1990). Finding structure in time. *Cognitive Science*, **14**, 179-211.

[58] Elman, J.L. (2005). Connectionist models of cognitive development: where next? *Trends in Cognitive Sciences*, **9**(3), 111-117.

[59] Elman, J.L., Bates, E.A., Johnson, M.H., Karmiloff-Smith, A., Parisi D. & Plunkett, K. (1997). The shape of change. In *Rethinking Innateness: A Connectionist Perspective on Development*, Cambridge, MA: MIT Press, ch. 6.

[60] Erdogmus, D., Fontenla-Romero, O., Principe, J., Alonso-Betanzos, A., & Castillo, E. (2005). Linear-least-squares initialization of multilayer perceptrons through backpropagation of the desired response. *IEEE Trans. Neural Networks*, **16**, 325-337.

[61] Fasano, G., Lampariello, F. & Sciandrone, M. (2006). A truncated nonmonotone Gauss-Newton method for large-scale nonlinear least-squares problems. *Computational Optimization and Applications*, **34**, 343-358.

[62] Feng, Y.T. (2006). On the discrete dynamic nature of the conjugate gradient method, *Journal of Computational Physics*, **211**(1), 91-98.

[63] Fischer, M.M. & Staufer, P. (1999). Optimisation in an error backpropagation neural network environment with a performance test on a spectral pattern classification problem. *Geographical Analysis*, **31**(2), 89-108.

[64] Fletcher, R. & Powell, M.J.D. (1963). A rapid convergent descent method for minimization. *Computer Journal*, **6**, 163-168.

[65] Fletcher, R. & Reeves, C.M. (1964). Function minimization by conjugate gradients. *Computer Journal*, **7,** 149-154.

[66] Fletcher, R. (1969). A review of methods for unconstrained optimization. In: Fletcher (ed.) *Optimization*, London: Academic Press, 1-12.

[67] Fletcher, R. (1970). A new approach to variable metric algorithms. *Computer Journal*, **13**, 317-322.

[68] Fletcher, R. (1987). *Practical methods of optimization*. Edn. 2nd, West Sussex: Wiley, reprinted 2006.

[69] Franklin, J.A. & Locke, K.K. (2004). Recurrent neural networks for musical pitch memory and classification. *Int'l J. Artificial Intelligence Tools*, **14**(9), 329-342.

[70] Gilbert, J.C. & Nocedal, J. (1992). Global convergence properties of conjugate gradient methods for optimization, *SIAM J. Optimization*, **2**, 21-42.

[71] Gill, P.E., Murray, W. & Wright, M.H. (1981). *Practical Optimization*, London: Academic Press.

[72] Goldfarb, D. (1970). A family of variable metric updates derived by variational means. *Mathematics of Computation*, **24**, 23-26.

[73] Goldstein, A.A. & Price, J.F. (1967). An effective algorithm for minimization.

Numerical Mathematics, **10**, 184-189.

[74] Goldstein, A.A. (1962). Cauchy's method of minimization. *Numerical Mathematics*, **4**, 146-150.

[75] Goldstein, A.A. (1965). On steepest descent. *SIAM J. Control*, **3**, 147-151.

[76] González, A. & Dorronsoro, J.R. (2008). Natural conjugate gradient training of multilayer perceptrons, *Neurocomputing*, **71**, 2499-2506.

[77] Gordienko, P. (1993). Construction of efficient neural networks: algorithms and tests. In: *Proc. IEEE Int'l Joint Conf. Neural Networks*, 313-316.

[78] Greig, D.M. (1980). *Optimisation*, London: Longman.

[79] Grippo, L. & Sciandrone, M. (2002). Nonmonotone globalization techniques for the Barzilai-Borwein gradient method, *Computational Optimization and Applications*, **23,** 143-169.

[80] Grippo, L., Lampariello, F. & Lucidi, S. (1986). A nonmonotone line search technique for Newton's method. *SIAM J. Numerical Analysis*, **23**, 707-716.

[81] Grippo, L., Lampariello, F. & Lucidi, S. (1990). A quasi-discrete Newton algorithm with a nonmonotone stabilization technique. *J. Optimization Theory and Applications*, **64**(3), 495-510.

[82] Grippo, L., Lampariello, F. & Lucidi, S. (1991). A class of nonmonotone stabilization methods in unconstrained optimization. *Numerische Mathematik*, **59**, 779-805.

[83] Grossberg, S. (1969). Some networks that can learn, remember, and reproduce any number of complicated space-time patterns. *Int'l J. Mathematics and Mechanics*, **19**, 53-91.

[84] Gruber, C. & Sick, B. (2003). Fast and efficient second-order training of the dynamic neural network paradigm. In: *Proc. IEEE Int'l J. Conf. Neural*

Networks, 2482-2487.

[85] Hagan, M.T. & Menhaj, M.B. (1994). Training feedforward networks with the Marquardt algorithm. *IEEE Trans. Neural Networks*, **5**, 989-993.

[86] Han, J.Y. & Liu, G.H. (1995). General form of stepsize selection rule of line search and relevant analysis of global convergence of BFGS algorithm. *Acta Mathematicae Applicatae Sinica*, **18**, 112-122.

[87] Han, J.Y. & Liu, G.H. (1997). Global convergence analysis of a new nonmonotone BFGS algorithm on convex objective functions. *Computational Optimization and Applications*, **7**, 277-289.

[88] Hestenes, M.R. & Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems. *J. Research of the National Bureau of Standards*, **49**(6), 409-436.

[89] Hestenes, M.R. (1980). *Conjugate direction methods in optimization*, New York: Springer-Verlag.

[90] Hochreiter, S. & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, **9**(8), 1735-1780.

[91] Hui, L.C.K., Lam, K.-Y. & Chea, C.W. (1997). Global optimisation in neural network training. *Neural Computing and Applications*, **5**, 58-64.

[92] Huseynov, J.J., Baliga, S.B., Widmer, A. & Boger, Z. (2008). An adaptive method for industrial hydrocarbon flame detection, *Neural Networks*, **21**(2-3), 398-405.

[93] Igel, C. & Hüsken, M. (2003). Empirical evaluation of the improved Rprop learning algorithms. *Neurocomputing*, **50**, 105-123, 2003.

[94] Irwin, G., Lightbody, G. & McLoone, S. (1994). Comparison of gradient based training algorithms for multilayer perceptrons. In: *Proc. IEE Colloquium*

Advances in Neural Networks for Control and Systems, 11/1-11/6.

[95] Ishikawa, T., Tsukui, Y. & Matsunami, M. (1996). Optimization of electromagnetic devices using artificial neural network with quasi-Newton algorithm, *IEEE Trans. Magnetics*, **32**(3), 1226-1229.

[96] Jordan, M.I. (1986). Attractor dynamics and parallelism in a connectionist sequential machine. In: *Proc. 8th Annual Conf. the Cognitive Science Society*, 531-546.

[97] Jordanov, I. & Georgieva, A. (2007). Neural network learning with global heuristic search. *IEEE Trans. Neural Networks*, **18**, 937-942.

[98] Karjala, T.W., Himmelblau, D.M. & Miikkulainen, R. (1992). Data rectification using recurrent (Elman) neural networks. In: *Proc. IEEE Int'l J. Conf. Neural Networks*, 901-906.

[99] Kolehmainen, M. (2003). Lecture notes on time-series modeling using neural networks. University of Kuopio.

[100] Kostopoulos, A., Glotsos, D., Spyridonos, P., Nikiforidis, G., Sotiropoulos, D. & Grapsa, T. (2004a). Comparative evaluation of feedforward and probabilistic neural networks for the automatic classification of brain tumours. In: *Proc. 1st International Conference "From Scientific Computing to Computational Engineering" (IC-SCCE)*, Athens, Greece, 8-10 September.

[101] Kostopoulos, A.E., Sotiropoulos, D.G. & Grapsa, T.N. (2004b). A new efficient variable learning rate for Perry's conjugate gradient training method. In: *Proc. 1st International Conference "From Scientific Computing to Computational Engineering" (IC-SCCE)*, Athens, Greece, 8-10 September.

[102] Kremer, S.C. & Kolen, J.F. (1998). Dynamical Recurrent Networks for Sequential Data Processing. In Wermter & Sun (eds.) *Hybrid Neural Systems,*

Revised Papers From A Workshop (December 04-05, 1998), *Lecture Notes in Computer Science*, **1778**, 107-122.

[103] Kremer, S.C. (2001). Spatiotemporal connectionist networks: a taxonomy and review. *Neural Computation*, **13**, 249-306.

[104] Lampariello, F. & Sciandrone, M. (2003). Use of the minimum-norm search direction in a nonmonotone version of the Gauss-Newton method, *J. Optimization Theory and Applications*, **119**(1), 65-82.

[105] Lera, G. & Pinzolas, M. (2002). Neighborhood based Levenberg-Marquardt algorithm for neural network training. *IEEE Trans. Neural Networks*, **13**, 1200-1203.

[106] Levenberg, K. (1944). A method for the solution of certain problems in least squares. *Quart. Applied Mathematics*, **5**, 164-168.

[107] Li, C.J. & Yan, L. (1995). Mechanical system modelling using recurrent neural networks via quasi-Newton learning methods, *Applied Mathematical Modelling*, **19**, 421-428.

[108] Li, H. & Adali, T. (2008). A Class of Complex ICA Algorithms Based on the Kurtosis Cost Function, *IEEE Trans. Neural Networks*, **19**(3), 408-420.

[109] Lightbody, G. & Irwin, G.W. (1995). A novel neural internal model control structure. In: *Proc. American Control Conference*, 350-354.

[110] Lightbody, G. & Irwin, G.W. (1996). Multi-layer perceptron based modelling of nonlinear systems. *Fuzzy Sets and Systems*, **79**, 93-112.

[111] Likas, A. & Stafylopatis, A. (2000). Training the random neural network using quasi-Newton methods. *European J. Operational Research*, **126**, 331-339.

[112] Lin, C.-T. & Lee, C.S.G. (1996). *Neural Fuzzy Systems: A Neuro-Fuzzy*

Synergism to Intelligent Systems. New Jersey: Prentice Hall.

[113] Liu, G.H., Jing, L.L., Han, L.X. & Han, D. (1999). A class of nonmonotone conjugate gradient methods for unconstrained optimization, *J. Optimization Theory and Applications*, **101**(1), 127-140.

[114] Liu, Y. & Wei, Z. (2002). A class of nonmonotone conjugate gradient methods for nonconvex functions, *Applied Mathematics J. Chinese University Ser. B*, **17**(2), 208-214.

[115] Ma, W.H. (1997). A study of conjugate gradient recurrent network on time series problems, *M.Sc. thesis*, Hong Kong Polytechnic University.

[116] Magoulas, G.D., Chen, S.Y. & Dimakopoulos, D. (2004). A personalised interface for web directories based on cognitive styles. In: *Proc. 8th ERCIM Workshop on User Interfaces for All, Lecture Notes in Computer Science,* **3196**, 159-166.

[117] Magoulas, G.D., Plagianakos, V.P. & Vrahatis, M.N. (2000). Development and convergence analysis of training algorithms with local learning rate adaptation. In: *Proc. INNS-IEEE Int'l J. Conf. Neural Networks*, 24-27 July 2000, Como, Italy, v. 1, 21-26.

[118] Magoulas, G.D., Plagianakos, V.P. & Vrahatis, M.N. (2002). Globally convergent algorithms with local learning rates. *IEEE Trans. Neural Networks*, **13**(3), 774-779.

[119] Magoulas, G.D., Vrahatis, M.N. & Androulakis, G.S. (1997). Effective backpropagation with variable stepsize. *Neural Networks*, **10**, 69-82.

[120] Magoulas, G.D., Vrahatis, M.N. & Androulakis, G.S. (1997). Effective back-propagation training with variable stepsize, *Neural Networks*, **10**, 69-82.

[121] Magoulas G. and Vrahatis M.N. (2006) Adaptive Algorithms for Neural

Network Supervised Learning: A Deterministic Optimization Approach. *International Journal of Bifurcation and Chaos*, **16**(7), 1929–1950.

[122] Marquardt, D. (1963). An algorithm for least squares estimation of nonlinear parameters. *J. Society for Industrial and Applied Mathematics*, **11**(2), 431-441.

[123] Marwala, T. (2001). Scaled conjugate gradient and Bayesian training of neural networks for fault identification in cylinders. *Computers and Structures*, **79**, 2793-2803.

[124] McLeod, P., Plunkett, K. & Rolls, E.T. (1998). *Introduction to connectionist modelling of cognitive processes*, Oxford: Oxford University Press, 148-151.

[125] McLoone, S. & Irwin, G.W. (1997). Fast parallel off-line training of multilayer perceptrons. *IEEE Trans. Neural Networks*, **8**(3), 646-653.

[126] McLoone, S. & Irwin, G.W. (1999). A variable memory quasi-Newton training algorithm. *Neural Processing Letters*, **9**, 77-89.

[127] McLoone, S., Asirvadam, V.S. & Irwin, G.W. (2002). A memory optimal BFGS neural network training algorithm. In: *Proc. IEEE Int'l J. Conf. Neural Networks*, 513-518.

[128] Medsker, L.R. & Jain, L.C. (2000). *Recurrent neural networks: design and applications*, Boca Raton, FL: CRC Press.

[129] Mirikitani, D. & Nikolaev, N. (2007). Recursive Bayesian Levenberg-Marquardt training of recurrent neural networks. In: *Proc. Int'l Joint Conf. Neural Networks (IJCNN'07)*, 282-287.

[130] Mirikitani, D. & Nikolaev, N. (2010). Recursive Bayesian Recurrent Neural Networks for Time Series Modeling. *IEEE Trans. Neural Networks*, **21**(2), 262-274.

[131] Mirikitani, D. and Nikolaev, N. (2010). Efficient Online Recurrent Connectionist Learning with the Ensemble Kalman Filter. *Neurocomputing*, 73(4-6), 1024-1030.

[132] Mizutani, E. (1999). Powell's dogleg trust-region steps with the quasi-Newton augmented Hessian for neural nonlinear least-squares learning. In: *Proc. IEEE Int'l Joint Conf. Neural Networks (IJCNN'99)*, 1239-1244.

[133] Moller, M.F. (1993). A scaled conjugate gradient algorithm for fast supervised learning, *Neural Networks*, **6**(4), 525-533.

[134] Morejon, R. & Principe, J. (2004). Advanced search algorithms for information-theoretic learning with kernel-based estimators. *IEEE Trans. Neural Networks*, **15**, 874-884.

[135] Nagaraja, G. & Jagadeesh Chandra Bose, R.P. (2006). Adaptive conjugate gradient algorithm for perceptron training, *Neurocomputing*, **69**, 368-386.

[136] Nawi, N.M., Ransing, M.R. & Ransing, R.S. (2006). An improved learning algorithm based on the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method for back propagation neural networks. In: *Proc. 6th Int'l Conf. Intelligent Systems Design and Applications*, 152-157.

[137] Nelles, O. (2000). *Nonlinear System Identification*, Berlin: Springer.

[138] Nocedal, J. & Yuan, Y. (1993). Analysis of a self-scaling quasi-Newton method. *Mathematical Program*, **61**, 19–37.

[139] Olivier, C. (2007) Training a Support Vector Machine in the Primal, *Neural Computation*, **19**(5), 1155-1178.

[140] Oren S.S. (1972). Self-scaling variable metric algorithms for unconstrained minimization, *PhD Thesis*, Stanford University, California, USA.

[141] Oren, S.S. & Luenberger, D.G. (1974). Self-scaling variable metric (SSVM) algorithms, Part I: Criteria and sufficient conditions for scaling a class

of algorithms, *Management Science*, **20**, 845–862.

[142] Orponen, P. (2000). An overview of the computational power of recurrent neural networks. In: *Proc. 9th Finnish AI Conference, Vol. **3**: "AI of Tomorrow"*, 89-96.

[143] Pearlmutter, B.A. (1995). Gradient calculations for dynamic recurrent neural networks: a survey. *IEEE Trans. Neural Networks*, **6**(5), 1212-1228.

[144] Pedersen, M.W. (1997). Optimization of recurrent neural networks for time series modelling. *PhD Thesis*. Technical University of Denmark.

[145] Peng C.-C. & Magoulas G.D. (2007a). Adaptive self-scaling nonmonotone BFGS training algorithm for recurrent neural networks. In: *Proc. 17th Int'l Conf. Artificial Neural Networks (ICANN'07),* 9-13 September, Porto, Portugal, 259-268.

[146] Peng, C.-C. & Magoulas, G.D. (2007b). Adaptive Nonmonotone Conjugate Gradient Training Algorithm for Recurrent Neural Networks. In: *Proc. 19th IEEE Int'l Conf. Tools with Artificial Intelligence (ICTAI'07)*, 29-31 October, Patras, Greece, 374-381.

[147] Peng, C.-C. & Magoulas, G.D. (2008a). Advanced Adaptive Nonmonotone Conjugate Gradient Training Algorithm for Recurrent Neural Networks. *Int'l J. Artificial Intelligence Tools (IJAIT)*, **17**(5), 963-984.

[148] Peng, C.-C. & Magoulas, G.D. (2008b). Sequence Processing with Recurrent Neural Networks. *Encyclopedia of Artificial Intelligence*, 1411-1417.

[149] Peng, C.-C. & Magoulas, G.D. (2009), Nonmonotone Learning of Recurrent Neural Networks in Symbolic Sequence Processing Application. In: *Proc. 11th Int'l Conf. Engineering Applications of Neural Networks (EANN'09)*, 22-29 August 2009, London, England, pp. 325-335.

[150] Peng, H., Ozaki, T., Haggan-Ozaki, V. & Toyoda, Y. (2003). A parameter optimization method for radial basis function type models. *IEEE Trans. Neural Networks*, **14**, 432-438.

[151] Pérez-Ortiz, J.A., Calera-Rubio, J. & Forcada, M.L. (2001). Online symbolic-sequence prediction with discrete-time recurrent neural networks. In: Dorffner, Bischof & Hornik (eds.) *Int'l Conf. ANNs, Lecture Notes in Computer Science*, **2130**, 719-724.

[152] Phua, P.K.H. & Ming, D. (2003). Parallel nonlinear optimization techniques for training neural networks. *IEEE Trans. Neural Networks*, **14**(6), 1460-1468.

[153] Phung, S.L. & Bouzerdoum, A. (2007). A pyramidal neural network for visual pattern recognition, *IEEE Trans. Neural Networks*, **18**(2), 329-343.

[154] Plagianakos V.P., Magoulas G.D. & Vrahatis M.N. (2006). Improved learning of neural nets through global search. In: *Global Optimization - Scientific and Engineering Case Studies*, János D. Pintér (ed.), Series: Nonconvex Optimization and Its Applications, vol. 85, NY: Springer-Verlag, 361-388.

[155] Plagianakos, V.P., Magoulas, G.D. & Vrahatis, M.N. (1999). Optimization strategies and backpropagation neural networks. In: *Proc. 7th Hellenic Conference on Informatics*, Ioannina, Greece, 26-29 August, 88-95.

[156] Plagianakos, V.P., Magoulas, G.D. & Vrahatis, M.N. (2002). Deterministic nonmonotone strategies for effective training of multi-layer perceptrons. *IEEE Trans. Neural Networks*, **13**(6), 1268-1284.

[157] Plagianakos, V.P., Sotiropoulos, D.G. & Vrahatis, M.N. (1998). A nonmonotone backpropagation training method for neural networks. *Technical*

Report, TR98-04, University of Patras.

[158] Plaut, D., McClelland, J., Seidenberg, M. & Patterson, K. (1996). Understanding normal and impaired word reading: computational principles in quasi-regular domains, *Psychological Review*, **103**(1), 56-115.

[159] Polak, E. (1971). *Computational methods in optimization: a unified approach.* New York: Academic Press.

[160] Powell, M.J.D. (1970). A new algorithm for unconstrained optimization. In: Rosen et al. (eds.) *Nonlinear Programming*, London: Academic Press, 31-65.

[161] Powell, M.J.D. (1977). Restart procedures for the conjugate gradient method, *Mathematical Programming*, **12**, 241-254.

[162] Powell, M.J.D. (1986). How bad are the BFGS and DPF methods when the objective function is quadratic? *Math. Program*, **34**, 34–47.

[163] Priel, A. & Kanter, I. (2003). Time series generation by recurrent neural networks. *Annals of Mathematics and Artificial Intelligence*, **39**, 315-332.

[164] Puskorius, G.V. & Feldkamp, L.A. (1994). Neurocontrol of nonlinear dynamical systems with Kalman filter trained recurrent networks. *IEEE Trans. Neural Networks*, **5**(2), 279-297.

[165] Riedmiller, M. & Braun, H. (1992). Rprop – a fast adaptive learning algorithm. In: *Proc. Int'l Symposium on Computer and Information Sciences*, Antalya, Turkey, 279-285.

[166] Riedmiller, M. & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: the Rprop algorithm. In: Ruspini E.H. (ed): *Proc. IEEE Int'l Conf. Neural Networks*, San Francisco, 586-591.

[167] Riedmiller, M. (1994a). Rprop – description and implementation details. Technical Report.

[168] Riedmiller, M. (1994b). Advanced supervised learning in multi-layer perceptrons – from backpropagation to adaptive learning algorithms. *Computer Standards and Interfaces*, **16**(5), 265-278.

[169] Sahari, M.L. & Djellit, I. (2009). The complex dynamic of conjugate gradient method, *International Journal of Computer Mathematics*, **86**(3), 407-422.

[170] Saini, L.M. & Soni, M.K. (2002). Artificial neural network based peak load forecasting using Levenberg-Marquardt and quasi-Newton methods. In: *Proc. Generation, Transmission and Distribution*, 578-584.

[171] Savran, A. (2007). Multifeedback-layer neural network. *IEEE Trans. Neural Networks*, **18**, 373-384.

[172] Scales, L.E. (1985). *Introduction to non-linear optimization*, London: McMillan, 56-109.

[173] Schalkoff, R.J. (1997). *Artificial Neural Networks*, New York: McGraw-Hill.

[174] Schmidhuber J., Wierstra D., Gagliolo M., & Gomez F. (2007). Training Recurrent Networks by Evolino. *Neural Computation*, **19**(3), 757-779.

[175] Seow, M.J. & Asari, V.K. (2006). Recurrent neural network as a linear attractor for pattern association. *IEEE Trans. Neural Networks*, **17**(1), 246-250.

[176] Setiono, R. & Hui, L.C.K. (1993). Some n-bit parity problems are solvable by feed-forward networks with less than n hidden units. In: *Proc. IEEE Int'l Joint Conf. Neural Networks*, 305-308.

[177] Setiono, R. & Hui, L.C.K. (1995). Use of a quasi-Newton in a feedforward\d neural network construction algorithm. *IEEE Trans. Neural Networks*, **6**(1), 273-277.

[178] Shaheed, M.H. (2004). Performance analysis of 4 types of conjugate gradient algorithms in the nonlinear dynamic modelling of a TRMS using feedforward neural networks, In: *IEEE Proc. Systems, Man and Cybernetics*, 6, 5985- 5990.

[179] Shanno, D.F. (1970). Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, **24**, 647-655.

[180] Shepherd, A.J. (1997). *Second-Order Methods for Neural Networks: Fast and Reliable Training Methods for Multi-Layer Perceptrons*, *Perspectives in Neural Computing* series, London: Springer-Verlag.

[181] Shi, Z.-J. & Shen, J. (2006a). Convergence of PRP method with new nonmonotone line search. *Applied Mathematics and Computation*, **181,** 423-431.

[182] Shi, Z.J. & Shen, J. (2006b). Convergence of nonmonotone line search method, *J. Computational and Applied Mathematics*, **193**(2), 397-412.

[183] Sluijter, R., Wuppermann, F., Taori, R., Kathmann, E. (1995) State of the art and trends in speech coding, *Philips Journal of Research*, **49**(4), 455-488.

[184] Sorensen, P.H., Norgaard, M., Ravn, O. & Poulsen, N.K. (1999). Implementation of neural network based non-linear predictive control. *Neurocomputing*, **28**, 37-51.

[185] Sotiropoulos, D.G., Kostopoulos, A.E. & Grapsa, T.N. (2002). A spectral version of Perry's conjugate gradient method for neural network training. In: *Proc. 4th GRACM Congress on Computational Mechanics*, 291-298.

[186] Sperduti, A. & Starita, A. (1997). Supervised neural networks for the classification of structures. *IEEE Trans. Neural Networks*, **8**(3), 714-735.

[187] Sun, W., Han, J. & Sun, J. (2002). Global convergence of nonmonotone

descent methods for unconstrained optimization problems. *J. Computational and Applied Mathematics*, **146**, 89–98.

[188] Temurtasm, F., Yumusak, N., Gunturkun, R. Temurtas, H. & Cerezci, O. (2004). Elman's recurrent neural networks using resilient back propagation for harmonic detection. In: Zhang C. et al. (eds.): *Proc. Pacific Rim Int'l Conf. Artificial Intelligence*, 422-428.

[189] Tiflin, C. & Omlin, C.W. (2003). LSTM Recurrent Neural Networks for Signature Verification. In: *Proc. Southern African Telecommunication Networks & Applications Conference (SATNAC 2003)*.

[190] Tiňo, P. & Mills, A. (2005). Learning beyond finite memory in recurrent networks for spiking neurons. In: Wang L. et al. (eds.): *Proc. Int'l Conf. Natural Computation 2005*, *Lecture Notes in Computer Science*, **3611**, 666-675.

[191] Tivive, F.H.C. & Bouzerdoum, A. (2005). Efficient training algorithms for a class of shunting inhibitory convolutional neural networks. *IEEE Trans. Neural Networks*, **16**, 541-556.

[192] Toledo, A., Pinzolas, M., Ibarrola, J. & Lera, G. (2005). Improvement of the neighborhood based Levenberg-Marquardt algorithm by local adaptation of the learning coefficient. *IEEE Trans. Neural Networks*, **16**, 988-992.

[193] Tsoi, A.C. (1998a). Recurrent neural network architectures: an overview. In: Giles C.L. & Gori M. (eds.): *Adaptive processing of sequences and data structures*. Berlin: Springer-Verlag, 1-26.

[194] Tsoi, A.C. (1998b). Gradient based learning algorithms. In: Giles C.L. & Gori M. (eds.): *Adaptive processing of sequences and data structures*. Berlin: Springer-Verlag, 27-62.

[195] Vrahatis, M.N., Andreoulakis, G.S., Lambrinos, J.N. & Magoulas, G.D.

(2000). A class of gradient unconstrained minimisation algorithms with adaptive stepsize, *J. Computational and Applied Mathematics*, **114**, 367-386.

[196] Waibel, A. (1989). Modular construction of time-delay neural networks for speech recognition, *Neural Computation*, **1**(1), 39-46.

[197] Waibel, A., Hanazawa, T., Hilton, G., Shikano, K., Lang, K.J. (1989). Phoneme recognition using time-delay neural networks, *IEEE Trans. Acoustics, Speech, and Signal Processing*, **37**, 328-339.

[198] Wan, S. & Banta, L. (2006). Parameter incremental learning algorithm for neural networks. *IEEE Trans. Neural Networks*, **17**, 1424-1438.

[199] Wei, C., Chong, J.O. & Keerthi, S.S. (2006). An improved conjugate gradient scheme to the solution of least squares SVM, *IEEE Trans. Neural Networks*, **16**(2), 1045-9227.

[200] Werbos, P.J. (1974). Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. *Doctoral Dissertation*, Applied Mathematics, Harvard University, Boston, MA. (BPTT)

[201] Werbos, P.J. (1990). Backpropagation through time: what it does and how to do it. In: *Proc. IEEE*, 78, 1550-1560.

[202] Wilde, D.J. & Beightler, C.S. (1967). *Foundations of Optimization*. Englewood Cliffs: Prentice Hall.

[203] Williams, R. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, **1**(2), 270-280.

[204] Wolfe, P. (1969). Convergence conditions for ascent methods, *SIAM Mathematical Review*, **11**, 226-235.

[205] Wolfe, P. (1971). Convergence conditions for ascent methods II: Some corrections, *SIAM Mathematical Review*, **13**, 185-188.

[206] Xiangrui, W. & Chaudhari, N.S. (2004). Recurrent neural networks for learning mixed k^{th}-order Markov chains. In: Pal N.R et al. (eds.): *Proc. Int'l Conf. Neural Information Processing*, 477-482.

[207] Xu, D.C. (2003). Global convergence of the Broyden's class of quasi-Newton methods with nonmonotone linesearch, *Acta Mathematicatae Applicatae Sinica, English Series*, **19**(1), 19-24.

[208] Yam, J. & Chow, T. (1997). Extended least squares based algorithm for training feedforward networks. *IEEE Trans. Neural Networks*, **8**, 806-810.

[209] Yin, H.X & Du, D.L. (2006). The global convergence of self-scaling BFGS algorithm with nonmonotone line search for unconstrained nonconvex optimization problems. *Acta Math. Sinica*, published online, 11 September 2006.

[210] Yu, Z.S., Zhang, W.G. & Wu, B.F. (2007). Strong global convergence of an adaptive nonmonotone memory gradient method, *Applied Mathematics and Computation*, **185**(1), 681-688.

[211] Zanghirati, G. (2000). Global convergence of nonmonotone strategies in parallel methods for block-bordered nonlinear systems, *Applied Mathematics and Computation*, **107**, 137-168.

[212] Zhang, H. & Hager, W.W. (2004). A nonmonotone line search technique and its application to unconstrained optimization. *SIAM J. Optim.*, **14**, 1043-1056.

[213] Zhang, J.Z. & Cheng, L.H. (1997). Nonmonotone Levenberg-Marquardt algorithms and their convergence analysis. *J. Optimization Theory and Applications*, **92**, 393-418.

[214] Zhang, L.-J., Li, Y.-D. & Chen, H.-M. (1995). A novel global training

algorithm and its convergence theorem for fuzzy neural networks. In: *Proc. IEEE Int'l Conf. Neural Networks*, 1001-1006.

[215] Zhou, G. & Si, J. (1998). Advanced neural-network training algorithm with reduced complexity based on Jacobian deficiency. *IEEE Trans. Neural Networks*, **9**, 448-453.

[216] Zimmer, W., Keats, J.B. & Prairie, R.R. (1998). Characterization of non-monotone hazards. In: *IEEE Proc. Annual Reliability and Maintainability Symposium*, 176-181.