Improving Time Efficiency of Feedforward Neural Network Learning

by

Batsukh Batbayar

Bachelor of Engineering, National University of Mongolia

Master of Physics, National University of Mongolia

A thesis submitted in fulfilment of the requirement for the Degree of

Doctor of Philosophy

School of Electrical and Computer Engineering

RMIT University

June 2008
Declaration

I hereby declare that this thesis is my own work and effort and that it has not been submitted anywhere for any award. Where other sources of information have been used, they have been acknowledged.

________________________________________
Batsukh Batbayar

16 June 2008
Acknowledgment

I would like to express my true appreciation to the School of Electrical and Computer Engineering of RMIT University. I will always owe major gratitude to my Thesis Advisor Professor Xinghuo Yu, for his patience and dedication in teaching me to become a better researcher.

I acknowledge the financial support by the State Training Fund of the Mongolian Government.

Finally, I would like to express my deepest gratitude for the constant support, understanding and love that I received from my wife Khulan, son Geser and my parents during the past years.
Table of Contents

Chapter 1
Introduction

1.1 Introduction
1.2 Motivation
1.3 Objectives and Contributions
1.4 Overview and Structure of the Thesis
1.5 Summary

Chapter 2
Review of the Neural Networks

2.1 Introduction
2.2 The Basic Elements
  2.2.1 The Neuron
  2.2.2 The Perceptron: McCulloch-Pitts Model
  2.2.3 The ADALINE
2.3 Activation Functions
2.4 Neural Networks Architecture
  2.4.1 Overview
  2.4.2 Basic Parameters
  2.4.3 Topologies
  2.4.4 Modelling of NNs
2.5 The Multi Layered Perceptron and Feedforward Neural Networks
  2.5.1 Multi Layered Perceptron
  2.5.2 Feedforward Neural Networks
2.6 Learning and Training
  2.6.1 Supervised Learning
    2.6.1.1 Gradient Descent Rule
    2.6.1.2 The delta rule
2.6.2 Unsupervised Learning ................................................................. 33
2.6.3 Reinforcement Learning .............................................................. 34
2.6.4 Competitive Learning ................................................................. 34
2.6.5 Hebbian Learning ...................................................................... 35
2.7 Neural Networks Classifiers ........................................................... 36
  2.7.1 Introduction ............................................................................. 36
  2.7.2 Linear Optimisation ................................................................. 37
  2.7.3 Nonlinear Optimisation ............................................................ 38
  2.7.4 Hybrid Optimisation ............................................................... 47
2.8 Backpropagation Learning Algorithms ........................................... 49
  2.8.1 Basic concept ........................................................................ 49
  2.8.2 GDBP Algorithm ................................................................... 51
  2.8.3 An Learning Example ............................................................. 52
  2.8.4 Assessment of Latest BP Algorithms ...................................... 54
2.9 Practical Applications ..................................................................... 62
2.10 Summary ..................................................................................... 65

Chapter 3
Terminal Attractor Based Backpropagation Algorithms ............................... 66

3.1 Introduction .................................................................................. 66
3.2 About Terminal Attractor Idea ...................................................... 68
3.3 Terminal Attractor Based Backpropagation Algorithms .................... 70
  3.3.1 An Overview ........................................................................ 70
  3.3.2 TABP Learning in the Neighbourhood of the Global Minimum .... 72
3.4 Fast Terminal Sliding Mode Concept .............................................. 74
3.5 The Fast Terminal Attractor Based Backpropagation Algorithm .......... 77
3.6 Terminal Attractors based Levenberg-Marquardt Backpropagation Algorithm 79
3.7 Simulation Results ....................................................................... 81
  3.7.1 Simulation Results Using TABP Learning Algorithm ................. 81
  3.7.2 Simulation Results Using FTABP Algorithm .............................. 84
  3.7.3 Simulation Using Terminal Attractors Based Levenberg-Marquardt Algorithm .... 93
3.8 Summary ...................................................................................... 96
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1.</td>
<td>Overview of the each corresponding chapter</td>
<td>7</td>
</tr>
<tr>
<td>Figure 2.1.</td>
<td>Basic neuron model</td>
<td>12</td>
</tr>
<tr>
<td>Figure 2.2.</td>
<td>The ADALINE</td>
<td>15</td>
</tr>
<tr>
<td>Figure 2.3.</td>
<td>Neural networks structure</td>
<td>22</td>
</tr>
<tr>
<td>Figure 2.4.</td>
<td>Perceptron misclassification 1-0</td>
<td>23</td>
</tr>
<tr>
<td>Figure 2.5.</td>
<td>Supervised learning</td>
<td>27</td>
</tr>
<tr>
<td>Figure 2.6.</td>
<td>Function minimisation</td>
<td>30</td>
</tr>
<tr>
<td>Figure 2.7.</td>
<td>Slope of $u(a)$</td>
<td>31</td>
</tr>
<tr>
<td>Figure 2.8.</td>
<td>Small changes used in computing the slope of $u(a)$</td>
<td>31</td>
</tr>
<tr>
<td>Figure 2.9.</td>
<td>Gradient descent for a network</td>
<td>34</td>
</tr>
<tr>
<td>Figure 2.10.</td>
<td>Delta learning rule</td>
<td>35</td>
</tr>
<tr>
<td>Figure 2.11.</td>
<td>BP training algorithm</td>
<td>53</td>
</tr>
<tr>
<td>Figure 2.12.</td>
<td>Influence of momentum upon weight change</td>
<td>57</td>
</tr>
<tr>
<td>Figure 3.1.</td>
<td>Convergence to regular attractor</td>
<td>68</td>
</tr>
<tr>
<td>Figure 3.2.</td>
<td>Convergence to terminal attractor</td>
<td>69</td>
</tr>
<tr>
<td>Figure 3.3.</td>
<td>The function to be approximated</td>
<td>82</td>
</tr>
<tr>
<td>Figure 3.4.</td>
<td>Learning performance comparison</td>
<td>82</td>
</tr>
<tr>
<td>Figure 3.5.</td>
<td>Learning performance comparison with best learning rate</td>
<td>83</td>
</tr>
<tr>
<td>Figure 3.6.</td>
<td>Learning performance of the TABP</td>
<td>83</td>
</tr>
<tr>
<td>Figure 3.7.</td>
<td>FTABP output after 30,000 iterations</td>
<td>85</td>
</tr>
<tr>
<td>Figure 3.8.</td>
<td>TABP output after 100,000 iterations</td>
<td>86</td>
</tr>
<tr>
<td>Figure 3.9.</td>
<td>GDBP output after 200,000 iterations</td>
<td>86</td>
</tr>
<tr>
<td>Figure 3.10.</td>
<td>Performance comparison of FTABP, TABP and GDBP</td>
<td>86</td>
</tr>
<tr>
<td>Figure 3.11.</td>
<td>Function to be approximated (Sensor 22)</td>
<td>88</td>
</tr>
<tr>
<td>Figure 3.12.</td>
<td>NNs outputs with GDBP, TABP and FTABP</td>
<td>88</td>
</tr>
<tr>
<td>Figure 3.13.</td>
<td>NNs outputs with two other algorithms</td>
<td>88</td>
</tr>
<tr>
<td>Figure 3.14.</td>
<td>Learning performance of sample algorithms</td>
<td>89</td>
</tr>
<tr>
<td>Figure 3.15.</td>
<td>Learning over performance</td>
<td>91</td>
</tr>
<tr>
<td>Figure 3.16.</td>
<td>Function to be approximated</td>
<td>93</td>
</tr>
<tr>
<td>Figure 3.17.</td>
<td>Improved LMBP output after 4952 iterations</td>
<td>94</td>
</tr>
<tr>
<td>Figure 3.18.</td>
<td>LMBP output after 10,000 iterations</td>
<td>94</td>
</tr>
<tr>
<td>Figure 3.19.</td>
<td>Performance comparison of the LMBP and improved LMBP</td>
<td>95</td>
</tr>
</tbody>
</table>
Figure 3.20. Performance comparison (enlargement) ................................................................. 95
Figure 4.1. Methodology tree for forecast .................................................................................. 99
Figure 4.2. Neural network forecasting diagram ....................................................................... 101
Figure 4.3. NNs structure with short term prediction ................................................................. 103
Figure 4.4. Simple autocorrelation correlogram ....................................................................... 106
Figure 4.5. Partial autocorrelation correlogram ......................................................................... 106
Figure 4.6. Partial autocorrelation coefficient ............................................................................ 107
Figure 4.7. NNs output with original index .................................................................................. 108
Figure 4.8. Performance of FTABP learning algorithm in logarithmic axes ............................. 109
Figure 4.9. Performance of NN forecasters for short term prediction ........................................ 109
Figure 4.10. NNs structure with long term prediction ............................................................... 111
Figure 4.11. Function to be approximated with long term prediction ........................................ 112
Figure 4.12. Performance of FTABP learning algorithm in logarithmic axes ............................. 113
Figure 4.13. Performance for long term forecasting ................................................................. 113
Figure 4.14. Image to data preprocessing .................................................................................... 116
Figure 4.15. The input and output data for ............................................................................... 117
Figure 4.16. Performance of FTABP for with 480 inputs in logarithmic axes ............................ 117
Figure 4.17. Performance of FTABP for with 1080 inputs in logarithmic axes ......................... 118
Figure 4.18. Interpolations resize and resort ........................................................................... 120
Figure 4.19. Process of generating training set ........................................................................ 121
Figure 4.20. Process of enlargement system with ANNs ......................................................... 122
Figure 4.21. Image used NN training process ........................................................................... 123
Figure 4.22. Performance of FTABP learning algorithm in logarithmic axes ............................. 124
Figure 4.23. Original image .................................................................................................... 124
Figure 4.24. Interpolated image .................................................................................................. 125
List of Tables

Table 3.1. Performance comparison of TABP with different $\rho$s ........................................ 84
Table 3.2. $J$ with different learning rates ............................................................................. 87
Table 3.3. Result of all algorithms ..................................................................................... 90
Table 3.4. Result of all algorithms ..................................................................................... 92
Table 4.1. Performance of neural network forecasters ....................................................... 110
Table 4.2. Result of all algorithms ..................................................................................... 111
Table 4.3. Result of all algorithms ..................................................................................... 118
Table 4.4. Result of all algorithms ..................................................................................... 119
Table 4.5. The first simulation results of interpolation ....................................................... 123
Table 4.6. Simulation results ............................................................................................ 125
Abbreviations

ADALINE. Adaptive Linear Neuron.
ANNs. Artificial Neural Networks.
ASCII. American Standard Code for Information Interchange.
BFGS. Broyden, Fletcher Goldfard and Shanno.
BNN. Biological Neural Networks.
BP. Back-Propagation.
BPALM. Backpropagation with Adaptive Learning rate and Momentum term.
CGAs. Conjugate Gradient Algorithms.
CPU. Central Processing Unit.
CRFs. Conditional Random Fields.
DWA. Dynamic Weighted Aggregation.
DWM. Deterministic Weight Modification.
EBP. Error Backpropagation.
EGD. Exponentiated Backpropagation Gradient Descent.
EIV. Errors In Variables.
EQUAL. Eigenvector-based QUAdratic Learning.
ER. Emergency Room.
ES. Error Saturation.
ESM. Error Saturation Method.
FFNM. FNN Model.
FNNs. Feedforward Neural Networks.
FPGA. Field Programmable Gate Array.
FTABP. Fast Terminal Attractor based Backpropagation.
GA. Genetic Algorithm.
GDBP. Gradient Descent Backpropagation.
GO. Global Optimisation.
ICNC. International Conference on Natural Computation.
ISSNIP. International Conference on Intelligent Sensors, Sensor Networks and Information Processing.
IEEE. Institute of Electrical and Electronics Engineers Inc.
LF. Lyapunov Function.
LF I. Lyapunov Function-based learning algorithm.
LF II. Modified version of Lyapunov Function-based learning algorithm.
LM. Levenberg-Marquardt.
LMBP. Levenberg-Marquardt Backpropagation.
MBP. Gradient Descent with Momentum.
MGFPROS. Magnified Gradient Function Learning Algorithm.
MIMO. Multi-Input Multi-Output.
MLP. Multi Layer Perceptron.
MR. Magnetic Resonance.
NNs. Neural Networks.
NSGA. Non-Dominated Sorting Genetic Algorithm.
OA. Orthogonal Array.
OBS. Optimal Brain Surgeon.
OCR. Optical Character Recognition.
OM. Optimal Memory.
PA. Power Amplifier.
PC. Personal Computer.
PCA. Principle Component Analysis.
PSO. Particle Swarm Optimisation.
RBF. Radial Bases Function.
SATA. Step Acceleration based Training Algorithm.
SBP. Standard Backpropagation.
SCGA. Scaled Conjugate Gradient Algorithms.
SLFNs. Single Hidden Layer Feedforward Neural Networks.
STSA. Systematic Trajectory Search Algorithm.
TABP. Terminal Attractor based Backpropagation.
TLS. Total Least Squares.
TRMD. Twin Rotor Multi-input-Multi output Systems.
TSMs. Terminal Sliding Modes.
US. United States.
VM. Variable Memory.
VSS. Variable Structure System.
WCDMA. Wideband Code Division Multiple Access.
2D. Two Dimensional.
Summary

Feedforward neural networks have been widely used for solving various problems in science and engineering. The training of this class of networks is mainly undertaken using the backpropagation based learning algorithms. It is currently the most widely used learning algorithm in the neural network learning. Due to the slow convergence speed of this group of algorithms, development of the neural networks can not forward quickly. In order to improve the training speed of backpropagation learning algorithms, many researchers have developed different improvements and enhancements. However, the slow convergence problem has not been fully addressed.

This thesis makes several contributions in improving time efficiency of feedforward neural network learning. Firstly, it presents a thorough review of the state of the art of learning algorithms for neural networks. A terminal attractor based backpropagation algorithm is proposed, which improves significantly the convergence speed near the ideal weights. A necessary condition has been derived to avoid the singularity problem. Secondly, a fast terminal sliding mode concept is adopted to develop a fast terminal attractor based BP algorithm. Thirdly, an improved Levenberg-Marquardt algorithm based on the terminal attractor is developed. Lastly, several typical applications are undertaken using the new learning algorithms to show their effectiveness.
Chapter 1

Introduction

1.1 Introduction

Discovering how the brain works has been an ongoing effort that began more than 2000 years ago with Aristotle and Heraclitus and has continued with the work of Ramon Cajal, Colgi, Hebb and others. The better we can understand the brain, the better we can emulate it and eventually build artificial machines that will assist with the ‘repair’ of biological damage. As information about the function of the brain was accumulated, a new technology emerged and the quest for Artificial Neural Networks (ANNs) or simply Neural Networks (NNs), began.

The human brain has the ability to process vast amounts of information in parallel. It is this parallelism and interconnectivity between neurons in the brain that allow complex operations such as pattern recognition to take place within the span of a few hundred milliseconds.
Neural networks are derived from the observed structure of the biological neural processing system. These are a class of computational architectures that are composed of interconnected simple processing nodes with weighted interconnections. It can be considered as one of the soft computation tools.

1.2 Motivation

Neural networks have drawn considerable attention in recent years because of their interesting abilities including nonlinear maps, memory and self-learning, etc. In fact, the advances in neural networks stimulated much interest in co-called universal approximants of the last decades.

One of the most popular and important types of NN architectures is Feedfoward Neural Networks (FNNs). A FNN is usually a static network with a well-defined direction of signal flow and no feedforward loops. FNNs have been widely used for various tasks such as signal processing, pattern recognition, function approximation, dynamical modelling, data mining, time series forecasting, control system, etc. The training of FNNs is mainly undertaken using the Back-Propagation (BP) based learning algorithms.

However, there are several drawbacks in the BP learning algorithms; for example, the main, basic defect is the convergence of BP algorithms which are generally slow. Many researchers have researched on the improvement of learning efficiency of BP algorithms. Some of the popular methods are Error Backpropagation [Rumelhart, Hilton and Williams 1986], Levenberg-Marquardt algorithms [Hagan and Menhaj 1994], Modified BP [Kathirvalavakumar and Tangavel 2006], Terminal Attractor based
algorithm [Zak 1986], Gradient Descent with Momentum [Hertz et al 1991], Conjugate Gradient BP with Powell Beale restarts [Powell 1977], BFGS quasi-Newton [Dennis and Schnabel 1983], Bayesian regularisation [MacKay 1992], One Step Secant [Battiti 1992], and Fletcher Reeves Conjugate Gradient [Fletcher et al 1964]. This thesis will propose new time efficient training algorithms for FNNs.

1.3 Objectives and Contributions

This thesis explores the possibility for the improvement of the FNN learning efficiency and based on this study, more novel and rapid learning algorithms will be composed. In order to achieve this we will first study the basics of NNs as well as understanding the architecture and multilayered perceptrons of NNs. This investigation is fundamental to the main research described here and will assist the explanation of the following objectives.

Firstly, following a description of the objectives, an indepth investigation will illustrate well-known, innovative and rapid learning algorithms for FNNs. The investigation will strive to examine various major fast-learning algorithms which will be used in the simulation section.

Secondly, the Gradient Descent Backpropagation (GDBP) algorithm will be an important part of this study because of its simple and reliable characteristics. An investigation of the latest, fast learning algorithms enables a description of what kind of learning algorithms have the ability to improve learning speed and which methodology of convergence we can use for proposing newer and faster learning algorithms. In
addition, a theoretical investigation will provide a comprehensive understanding of the selected learning algorithms.

Thirdly, one of the fast learning algorithms – Terminal Attractor Based Backpropagation (TABP) algorithm, which makes use of the terminal-attractor concept to enable a finite-time convergence, will be studied entirely. A common dynamic characteristic of GDBP and other algorithms is the fixed-learning rate it incurs as an asymptotic convergence, which means that the closer to the desired weights, the slower the convergence speed - making fast convergence with higher precision tolerance more difficult. The TABP learning algorithm can solve this problem because it is able to converge faster when close to the desired weights.

The terminal-attractor concept has been successfully used in control systems to improve the transient control process [Yu and Man 2002]. Based on this investigation we will compose a new, fast learning algorithm for FNNs.

To demonstrate learning efficiency of the new algorithms, comparison investigation will be done. Major fast-learning algorithms will be used in this part with MATLAB functions to be especially selected.

Finally, some large-sized practical applications of NNs will be considered. Image processing and interpolation, NN based prediction for finance will be explored in this part. In terms of using the latest practical applications, this research will not only
examine the theoretical investigations but it will also demonstrate that it has practical applications.

The work presented in this thesis makes original contributions in several areas, including:

1. The finding in this thesis contributes to a better understanding of the learning capabilities of the NNs via developing fast learning algorithms for FNNs.

2. A new learning algorithm, Fast Terminal Attractor Based Backpropagation (FTABP) which combines the advantages of both the GDBP and TABP learning algorithms is proposed, which is based on the fast terminal sliding mode concept and utilizes the conventional gradient descent concept and terminal attractor backpropagation weight update law.

3. The general NN learning algorithm [Yu, Efe and Kaynak 2002] is incorporated with the fast terminal attractor concept, giving rise to a specific type of FNN learning algorithm.

4. An improved Levenberg-Marquardt algorithm based on the terminal attractor concept is proposed, which combines the advantages of both the Levenberg-Marquardt Backpropagation (LMBP) and TABP learning algorithms.

5. The new FTABP algorithm is used in several large-sized practical applications,
including time series forecasting, high accuracy optical character recognition, and an image interpolation problem.

1.4 Overview and Structure of the Thesis

The research presented in this thesis demonstrates fast learning efficiency of the proposed learning algorithms through comparative studies. Some advanced learning algorithms and benchmark datasets are employed in these experiments. The overview of the theoretical survey, proposed new learning algorithms, and use of such in the practical applications in this thesis are shown in Figure 1.1

This thesis is comprised of four sections: A review of NNs (Chapter 2), the TABP learning algorithm, the FTABP learning algorithm and the improved LMBP algorithm (Chapter 3), and practical applications (Chapter 4). These chapters are supported by NN Fundamentals and Preliminaries (Section 2.2-2.6), Literature Survey (Section 2.7) and concluded by Conclusions and Future Works (Chapter 5). The chapters are described in more detail as follows:
Figure 1.1 An overview of the structure of this thesis
Chapter 2 describes NN fundamentals and preliminaries. This chapter begins with the introduction of NNs and will also provide a description of the neuron, which is the basic element of NNs. The background information of NN learning is described next as well as a brief discussion of the NN architectures. Section 2.5 describes multilayered perceptron and weight update laws of FNNs. Section 2.7 mainly investigates the NN optimization techniques, especially gradient based algorithms. Section 2.8 reviews the BP learning algorithms. Subsection 2.8.1 presents the preambles, GDBP and learning examples. Subsection 2.8.2 provides assessments for the latest fast algorithms. The final section of this chapter presents a summary.

Chapter 3 deals with TABP algorithms. Section 3.1 presents the introduction. Section 3.2 introduces the terminal attractor as the basis for the fundamental idea of TABP learning algorithms. In the next section, the TABP algorithm is described extensively. Also, an understanding is gained of terminal attractor based learning in the neighbourhood of the global minimum. The simulation result of TABP algorithm is illustrated at the end of this chapter. Based on the previous investigations, the FTABP algorithm is then proposed. Firstly, the fast terminal sliding mode is evaluated in section 3.4. Next, the fundamental root, theoretical conclusion and proof of the FTABP algorithm are presented. Another improved version using the terminal attractor is the terminal attractor based LMBP algorithm which is illustrated in the next section and finally we review the simulation results using above algorithms at the end of this chapter.
In Chapter 4 practical applications are undertaken using the FNN with the FTABP algorithm. In section 4.2, the latest, main practical applications for FNNs are presented. Next, time series forecasting with a FNN is described and section 4.4 exhibits the simulation results for high accuracy optical character recognition using the FTABP algorithm. Image interpolation with a FNN with the FTABP algorithm is also demonstrated in the next section.

Chapter 5 summarizes the work presented in this thesis and presents the conclusion that has been drawn from this research. This chapter also suggests future research directions. Section 5.4 is a conclusion.

1.5 Summary

As we have mentioned in the introduction, the slow convergence is a major problem of NN Learning, which will be the focus of this thesis. An overview of the thesis has been provided and the structure of this study has also been outlined, paving the way for the following studies in this thesis.
Chapter 2

Review of the Neural Networks

2.1 Introduction

The basic attributes of NNs may be divided into two areas: architecture and functional properties. Architecture defines the network structure, that is, the number of artificial neurons in the network and their interconnectivity. Neural networks consist of many interconnected neurons, or processing elements, with familiar characteristics such as inputs, synaptic strengths, activation, outputs and biases. The functional properties of NNs define their properties which are how the NN learns, recalls, associates and continuously compares new information with existing knowledge. It is also how it classifies new information and how it develops new classifications if necessary.

In this chapter both of these attributes are described extensively. Presented first, are the fundamentals of the NNs. These principles emanate from the biological world and mathematics attempts to closely describe the biological behavior of neurons and their networks. Pictures and formulas provide additional and clear information regarding
their principles. Moreover, learning, training, optimization classifications, BP learning algorithms and the latest popular algorithms are also reviewed.

This chapter is organized in the following manner: Section 2 defines the neuron, which is an abstraction of biological neurons and the basic unit of the NNs. Section 3 illustrates the activation function of NNs. Section 4 discusses the NN architecture. Multilayered perceptron and FNNs are classified in Section 5. Section 6 presents learning and training. Classification of optimization techniques of NNs is presented in Section 7. Section 8 outlines various BP learning algorithms and Section 9 of this chapter gives an overview of the latest main practical applications. Section 10 concludes this chapter.

2.2 The Basic Elements

2.2.1 The Neuron

Attention will now be paid to the fundamental unit or building block of the NNs: the neuron. The neuron is also called ‘processing element’. This term, however, is used here with the understanding that it in no way describes the biological neuron. The neuron has two modes of operation: the training mode and the using mode [Kevin 1997]. In the training mode, the neuron can be trained to fire (or not), for particular input patterns. In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output.
The basic artificial neuron has a set of \( n \) inputs where \( x \) indicates the source of the input signal. Each input \( x \) is weighted before reaching the main body of the neuron by the connection strength or the weight factor \( w \). Moreover, it has a bias term \( w_0 \), a threshold value \( \Theta \) (that has to be reached or exceeded for the neuron to produce a signal) nonlinearity function \( F \) that acts on the produced signal (or activation) and an output \( y \) after the nonlinearity function: \( y \) constitutes input to other neurons. Inputs, weights, activation functions, outputs, thresholds, and nonlinear functions are written as \( x_i, w_i, R, y, \Theta \) and \( F \) respectively. The basic model of a neuron is illustrated in Figure 2.1.

\[
y_i = F\left(\sum_{i=1}^{n} w_i x_i\right), \tag{2.1}
\]

where \( x_0 = 1 \). The neuron’s firing condition is

\[
\sum_{k=1}^{n} w_k x_k \geq \Theta_i, \tag{2.2}
\]

where the index \( i \) represents the index of neuron.
The purpose of the nonlinearity function is to ensure that the neuron’s response is bounded. That is, the actual response of the neuron is conditioned or damped, as a result of large or small activating stimuli and is thus controllable. In mathematics, a nonlinear system is a system which is not linear and does not satisfy the superposition principle. The most popular nonlinear neurons are sigmoid, logsig and tansig functions. In the biological world, conditioning of stimuli is continuously done by all sensory inputs. For example, it is well known that to perceive a sound to be twice as loud, an actual increase in sound amplitude of about 10 times must take place; hence, the almost logarithmic response of the ear. Biological neurons condition their output response in a similar manner, so this concept is consistent with the biological neuron. But the nonlinearity function used in many paradigms is not necessarily a close replica of the biological one; often it is merely used, depending on the paradigm and the algorithm used.

### 2.2.2 The Perceptron: McCulloch-Pitts model

The first mathematical model of a single idealized biological neuron was introduced by neurobiologist Warren McCulloch and statistician Walter Pitts in 1943. Known as the McCulloch-Pitts model (also known as linear threshold gate), it is quite simple, with no learning or adaptation [Minsky 1969]. In addition, it has been the basic building block that inspired and stimulated subsequent work in developing paradigms. McCulloch and Pitt’s seminal paper was followed by others expanding the ideas of pattern recognition. However, this model is so simplistic; it only generates a binary output whilst the weight and threshold values are fixed.
In this model each input receives a stimulus, $x_j$ which is weighed (multiplied) by some value, $w_i$ which represents synaptic strength. All weighed inputs are summed, and if the combined input reaches a certain threshold level, a response is generated which is further modulated by a nonlinear transfer function $f$. The output is then expressed by:

$$y = f\left(\sum_{i=1}^{N} (w_i x_i) + w_0\right),$$

(2.3)

where $w_0 = 1$, $x_i$ is the incoming signal or stimulus on the $i$th neuron, $f$ is the nonlinearity, and $y_i$ is the output response of the $i$th neuron. In this model the bias term $\Theta$ and the weights $w_i$ are assumed to have reached steady state. Constant weights imply that all learning or adaptation has been completed.

The McCulloch-Pitts model is a simple open-loop system; its purpose is to model a single neuron. No mechanism exists here to compare the actual expected output response, and therefore, no weight adjustment or learning can take place. The need for feedback was quickly recognized, particularly at a time when feedback control theory was moving in the direction of adaptive control. Moreover, it attracted the interest of many researchers, who built on the McCulloch-Pitts model and developed the next-generation paradigms with learning and adaptation. Such a model is the Perceptron by Frank Rosenblatt. Rosenblatt’s key contribution was the introduction of a learning rule for training perceptron networks to solve pattern recognition problems in Cornell Aeronautical Laboratory [Rosenblatt 1958]. He proved that his learning rule will always converge to the correct network weights, if weights exist that can solve the problem. Learning was simple and automatic. Examples of proper behavior were presented to the
network, which then learned from its mistakes. The perceptron could even learn when initialized with random values for its weights and biases.

Today the perceptron is still viewed as an important network. It remains a fast and reliable network for the class of problems that it is able to solve. Moreover, an understanding of operations of the perceptron provides a good basis for understanding more complex networks.

2.2.3 The ADALINE

ADALINE (Adaptive Linear Neuron or later Adaptive Linear Element) is a single layer neural network. It was developed at Stanford University in 1960 by Bernard Widrow and his graduate student, Ted Hoff and it is based on the McCulloch-Pitts neuron. The structure of ADALINE is shown in Figure 2.2. Notice that it has the same basic structure as the perceptron. The only difference is that it has a linear transfer function.

![Figure 2.2 The ADALINE](image-url)
The output of the network is given by:

\[ y = \sum x_i w_i + x_0 \]  

(2.4)

### 2.3 Activation Functions

The behaviour of NNs depends on both the weights and the activation specified for the units. This function typically falls into one of three categories:

- linear (or ramp)
- threshold
- sigmoid

For linear units, the output activity is proportional to the total weighted output. For threshold unit(s), the output is set at one of two levels, depending on whether the total input is greater than, or less than some threshold value. For sigmoid units, the output varies continuously - but not linearly as the input changes. Sigmoid units bear a greater resemblance to real neurones than do linear or threshold units, but all three must be considered rough approximations. Typically, most researchers use linear, sigmoid and tansig functions.
2.4 Neural Network Architecture

2.4.1 Overview

In the previous sections the basic elements of NNs are described. Without an effective understanding of the network paradigm knowledge of NNs is then unattainable. Characteristics of NNs, inspired from the biological world, have been developed and are known as paradigms. The search for the best representative paradigms - that truly emulate the Biological Neural Networks (BNNs) - is still underway. Here in, the terms ANNs, NNs, and paradigms will be used synonymously. A typical paradigm is structured in layers of neurons. Some have one layer – Single-Layer Neural Networks (SLNN) - and some have a greater number of multilayer neural networks.

Based on the topologies of NNs, the layer of application for input patterns is the input layer. The layer from which the output response is obtained is the output layer. Intermediate layers are called hidden layers because their outputs are not readily observable. This hierarchical organization of layers can also be seen in biological neural networks.

In general, the NNs may be thought of as a sophisticated signal processor [Peretto 1992]. The processing ability of the networks, however, does not depend on serial algorithms executed by sequential von Neumann machines. In a NN the program is distributed across the network and stored at the synaptic point of each neuron. During the learning phase, synaptic weights and threshold values are adjusted until they yield the desired outputs. In the general case, the strength of each synapse (i.e. the synaptic
weight) and the threshold value of each neuron at steady state constitute the network’s program. Thus, every neuron takes part in a massive parallel program execution. In today’s paradigms the synaptic weight is adjusted automatically by a supervisory computer or by a direct feedback mechanism during learning. Interestingly, early paradigms used manual potentiometers to set the synaptic values and thresholds. Although tedious, they proved the point and stimulated enormous interest.

2.4.2 Basic Parameters

The NN’s performance is described by the figure of merit, which expresses the number of recalled patterns when complete, partially complete or noisy input patterns are applied. A 100% performance in recalled patterns means that for every trained input stimulus signal, the network always produces the desired, or target, output pattern. When designing a NN, one should be concerned with the following [Kartalopoulos 1996]:

- **Bias terms**
- **Boundaries of the synaptic weights**
- **Choice of the nonlinearity function**
- **Degree of adaptability of the NNs**
- **Learning algorithm to be adopted**
- **Network capacity**
- **Network noise immunity**
- **Network performance**
- **Network plasticity**
- **Network topology**
- Number of classifications per iteration
- Number of iterations per pattern during training
- Number of layers in the networks
- Number of neurons per layer
- Speed to recall a pattern
- Steady-state or final values of the synaptic
- Threshold terms

2.4.3 Topologies

Neural networks consist of many neurons interconnected in a particular way as to cast them into identifiable topologies. The major topologies are [Kartalopoulos 1996]:

- Multilayer cooperative/competitive
- Bilayer feedforward/backward
- Monolayer heterofeedback

Typically, the layer where the input patterns are applied is the input layer, the layer where the output is obtained is the output layer, and the layers between the input and output layers are the hidden layers. There may be one or more hidden layers, which are so named because their outputs are not directly observable.

2.4.4 Modeling of NNs

The modeling of NNs is a human attempt to understand and evaluate nature. Modeling means to develop a set of mathematical expressions that, to some extent, faithfully describes the neuron and the network. Mathematical analysis of a NN can tell us the following about a network:
• **Complexity** - how large the NN can be in order to execute a task
• **Capacity** - how much information can be stored in the NNs
• **Paradigms choice** - which NN implementation is more suitable
• **Performance** - which NN performs best
• **Learning efficiency** - how fast a NN learns
• **Response** - how fast a NN provides an output from the time a stimulus is present
• **Reliability** - whether the NN can reach the same desired solution for the same stimulus
• **Noise sensitivity** - how accurately an ANNs provides the desired output at the presence of noise
• **Fault tolerance** - how accurately an ANNs associates if it partially fails

2.5 Multilayered Perceptron and Feedforward Neural Networks

2.5.1 Multi Layered Perceptron

This class of networks consists of multiple layers of computational units, usually interconnected in a feedforward way. Each neuron in one layer, has directed connections to the neurons of the subsequent layer. In many applications the units of these networks apply a sigmoid function as an activation function.

The *universal approximation theorem* for NNs that every continuous function that maps intervals of real numbers to some output interval of real numbers can be approximated arbitrarily and closely by a multi-layer perceptron with just one hidden layer [Haykin
This result holds only for restricted classes of activation functions, e.g. for the sigmoid functions.

Multi-layer networks use a variety of learning techniques, the most popular being BP based algorithms [Hagan and Menhaj 1994]. Here the output values are compared with the correct answer to compute the value of some predefined error-function. The error is then fed back through the network through the use of various techniques. Using this information, the algorithm adjusts the weights of each connection in order to reduce the value of the error function by some small amount. After repeating this process for a sufficiently large number of training cycles the network will usually converge to some state where the error of the calculations is small. In this case one says that the network has learned a particular target function. To properly adjust weights, one applies a general method for non-linear optimization that is called ‘gradient descent’. For this, the derivative of the error function with respect to the network weights is calculated, and the weights are then changed such that the error decreases (thus going downhill on the surface of the error function). For this reason, BP algorithms can only be applied on networks with differentiable activation functions.
In general, the problem of teaching a NN to perform well, even on samples that were not used as training samples, is a quite subtle issue that requires additional techniques. This is especially important for cases where only very limited numbers of training samples are available. The danger here is that the network overfits the training data and fails to capture the true statistical process generating the data. Computational learning theory is concerned with training classifiers on a limited amount of data [Hertz et al 1991]. In the context of NNs a simple heuristic, called *early stopping*, often ensures that the network will generalize well to examples not in the training set.
Other typical problems of the BP algorithms are the speed of convergence and the possibility of finishing in a local minimum of the error function.

### 2.5.2 Feedforward Neural Networks

We now suppose to train single perceptron with augmented weight vector \( w \) using the training set consisting of pairs like \( x, y_d \) [Kartalopoulos 1996]. Suppose we present an input vector \( x \) to the training unit with a desired response \( y_d = 1 \) and, with the current weight vector, it produces an output \( y = 0 \). The perceptron has misclassified and we must make some adjustment to the weights. To produce ‘0’ the activation must have been negative when it should have been positive. Assume that there are two vectors, \( w \), and \( v \) in the state space. The dot product \( w^*v \) was negative and the two vectors were pointing away from each other as shown on the left hand side of Figure 2.4.

![Figure 2.4 Perceptron misclassification 1-0.](image)

In order to correct the situation we need to rotate \( w \) so that it points more in the direction of \( v \). At the same time, drastic change is not what is needed as this may upset
previous learning. Both goals can be achieved by adding a fraction of \( v \) to \( w \) to produce a new weight vector \( w' \), that is

\[
w' = w + \gamma v
\]  

(2.5)

where \( 0 < \gamma < 1 \), which is shown schematically on the right hand side of the figure.

Now we need to rotate \( w \) away from \( v \), which may be effected by subtracting a fraction of \( v \) from \( w \), that is

\[
w' = w - \gamma v
\]  

(2.6)

Both (2.5) and (2.6) may be combined as a single rule in the following way

\[
w' = w + \gamma(y) v
\]  

(2.7)

This may be written in terms of the change in the weight vector \( \Delta w = w' - w \) as follows

\[
\Delta w = \gamma(y_d - y) v
\]  

(2.8)

or in terms of the components

\[
\Delta w_i = \gamma(y_d - y) v_i
\]  

(2.9)

**2.6 Learning and Training**

Neural networks consist of a set of inputs and outputs, but they are not initially programmed to solve a problem; they must learn the best solution to the problem. NN learning is highly important and is undergoing intense research in both biological and artificial networks. The fundamental questions that behaviorists try to answer are: how do we learn? What is the most efficient process for learning? How much and how fast can we learn? What are the roadblocks in learning?
Research in learning has been conducted on animals of varying intelligence, on humans of different age and intelligence levels, on marine life and on more primitive life. Learning is not a unique process; there are a variety of learning processes, each suitable to different species. Not all learning processes are equally efficient. Behaviorists will need to determine which learning process is the most suitable and most efficient for a particular species. In NNs, the concepts of learning processes have been borrowed from the behaviorist’s lab and ported in actual electronic circuitry. The NN engineer selects the most efficient type of learning and integrates it with the most suitable NNs.

At its most basic level, learning can be described as the process by which a NN adjusts its weights in response to its inputs in order to produce the desired output. In other words, it has ‘acquired knowledge’. A network is said to have learned a process or acquired knowledge when its response is within acceptable neighborhood of the actual response. Learning is also the process by which the NN adapts itself to a stimulus, and eventually (after making the proper parameter adjustments to itself) produces a desired response. Paradigms observe learning rules described by mathematical expressions called ‘learning equations’. Learning equations describe the learning process for the paradigm, which in actuality is the process for self-adjusting its synaptic weights. As different learning methodologies suit different applications, so too different learning techniques suit different kinds of NNs.

Training is the procedure by which the NN learns. During training the NN, through an iterative process, each of the connection weights is adjusted and the most appropriate set of weights are found to represent the model. Initially, each of the weights is
randomly chosen and nonzero. To attain this best fit, the model goes through a set of iterations known as epochs, using a predefined learning algorithm. It sometimes takes thousands of epochs to train a NN adequately; like regression, this level is frequently measured with the mean square error, with stronger models approaching zero.

2.6.1 Supervised Learning

For supervised learning, a set of inputs is applied to the system. The network's output is compared with a priori-desired output signal, also known as the target response [Tsoukalas and Uhrig 1997]. If the deviation between the target response and the network's output is not within acceptable limits, an error signal or energy function is generated, that is used to recalculate the synaptic weights - so that the actual output matches the target output. In other words, the error is minimized, possibly to zero. The supervised learning is illustrated in Figure 2.5. The error minimization process requires a special circuit known as a target or teacher, hence the name supervised learning. The notion of a teacher signal comes from biological observations. With NNs the amount of calculation required to minimize the error depends on the algorithm used; clearly, this is purely a mathematical tool derived from optimization techniques.

![Figure 2.5 Supervised learning](image-url)
2.6.1.1 Gradient Descent Rule

Finding the minimum
Consider a quantity $u$ that depends on one variable $a$, that is $u = u(a)$ [Zurada 1992]. Suppose now that we wish to find that value $a_0$ for which $u$ is a minimum as shown in Figure 2.6. Let $a^*$ be the current best estimate for $a_0$; then one sensible thing to do in order to obtain a better estimate is to change $a$ so as to follow the function ‘downhill’ as it were. Thus, if increasing $a$ (starting at $a^*$) implies a decrease in $u$ then a small positive change is made, $\Delta a > 0$, to our estimate $a^*$. On the other hand, if decreasing $a$ results in decreasing $u$ then a negative change is made, $\Delta a < 0$. The knowledge used to make these decisions is contained in the slope of the function at $a^*$. If increasing $a$ increases $u$, the slope is positive, otherwise it is negative.

![Figure 2.6 Function minimisation](image)

The extension to general functions of a single variable is straightforward, as shown in Figure 2.7. The slope at any point $a$ is just the slope of a straight line, the tangent, which just grazes the curve at that point. There are two ways to find the slope. Firstly, we may draw the function on graph paper, draw the tangent at the required point, complete the triangle as shown in the figure and measure the sizes $\Delta a$ and $\Delta u$. It is possible, however,
to calculate the slope from \( u(a) \) using a branch of mathematics known as the differential calculus. It is not part of our brief, to demonstrate or use any of the techniques of the calculus but it is possible to understand what is being computed and where some of its notation comes from.

Figure 2.8 shows a close-up of the region around point \( P \) in Figure 2.7. The slope at \( P \) has been constructed in the usual way but this time, the change \( \Delta a \) used to construct the base of the triangle is supported to be very small. If \( \delta u \) is the change in the value of the function \( y \) due to \( \Delta a \) then, if the changes are small enough, \( \delta u \) is approximately equal to \( \Delta u \). We write this symbolically as \( \delta u \approx \Delta u \).
Now, dividing $\Delta u$ by $\Delta a$ and then multiplying by $\Delta a$ leaves $\Delta u$ unchanged. Thus we have

$$\Delta u = \frac{\Delta u}{\Delta a} \Delta a$$

(2.10)

Since $\delta u \approx \Delta u$ we can now write

$$\delta u \approx \text{slope} \times \Delta a$$

(2.11)

Alternatively, we can have

$$\delta u = \frac{du}{da} \Delta a$$

(2.12)

To evaluate the slope or derivative of $u$ and put

$$\Delta a = -\gamma \frac{du}{da}$$

(2.13)

where $\gamma > 0$ and is small enough to ensure $\delta u \approx \Delta u$, then, substituting this in (2.12),

$$\delta u = -\gamma \left(\frac{du}{da}\right)^2$$

(2.14)

Since taking the square of anything gives a positive value the $-\gamma$ term on the right hand side of (2.14) ensures that it is always negative and so $\delta u < 0$; that is, we have ‘travelled down’ the curve towards the minimal point as required [Zurada 1992]. If we keep
repeating a step like (2.14) iteratively, then we should approach the value $a_0$ associated with the function minimum. This technique is called, not surprisingly, gradient descent and its effectiveness hinges, of course, on the ability to calculate or make estimates of the quantities like $du/da$.

For a multivariable function, that is, $y$ is a function of more than one variable, say $u = u(a_1, a_2,.. a_n)$, we can still talk about the slope of the function with respect to each of these variables independently [Zurada 1992]. However, the slope in the other direction (The slope or derivative of function $y$ with respect to the variable $a_i$ is written $\partial u/\partial a_i$ and is known as the partial derivative. Just as for the ordinary derivatives like $du/da$, these should be read as a single symbolic entity standing for something like ‘slope of $s$ when $a_i$ alone is varied’. The equivalent of (2.13) is then

$$\Delta a = -\gamma \frac{\partial u}{\partial a_i} \tag{2.15}$$

There is an equation like this for each variable and all of them must be used to ensure that $\partial u<0$ and that there is gradient descent.

**Gradient descent on the error**

Consider, for the sake of simplicity, a ‘network’ consisting of a single perceptron. We assume a supervised regime so that, for every input pattern, $p$ in the training set, there is a corresponding target $y_d$. The behaviour of the network is completely characterized by the augmented weight vector $w$, so that any function $J$, which expresses the discrepancy between desired and actual network output, may be considered a function of the weights, $J = J(w_1, w_2,.. w_{n+1})$. The optimal weight vector is then found by minimising
this function by gradient descent as shown schematically in Figure 2.9 [Zurada 1992].

By applying (2.15) in this case we obtain

$$\Delta w_i = -\gamma \frac{\partial J}{\partial w_i}$$

(2.16)

It remains now to define a suitable error $J$. One way to proceed is to assign equal importance to the error for each pattern so that, if $e^p$ is the error for training pattern $p$, the total error $J$ is just the average or mean over all patterns

$$J = \frac{1}{N} \sum_{p=1}^{N} e^p$$

(2.17)

where there are $N$ patterns in training set. Clearly, just as for $J$, any $e^p$ will also be completely determined by the weights. As a first attempt to define $e^p$ we might simply use of difference, $e^p = y_d^p - y^p$, where $y^p$ is the perceptron output in response to $p$. This definition falls within the general remit since $y^p$, and hence $e^p$, may be written as a function of the weights.

![Figure 2.9 Gradient descent for a network](image)

The problem here, however, is that the error is then smaller for the combination $y_d^p = 0$, $y^p = 1$, than is it for $y_d^p = 1$, $y^p = 0$, whereas both are as ‘wrong’ as each other. The way
around this is to work with the purely positive quantity obtained by squaring the difference, so the improvement is

$$e^p = (y_d^p - y_p^p)^2 \quad (2.18)$$

### 2.6.1.2 The Delta Rule

The delta rule is a gradient descent learning rule for updating the weights of the artificial neurons in a single-layer perceptron. It is based on the idea of continuous adjustments of the value of the weights such that the difference of error (*delta*) between the desired (or target) output value and the actual output value of a processing element is reduced. This is also known as the Widrow-Hoff learning rule or as the Least Mean Square (because it minimises the mean square error). The delta rule was first introduced in NN training [McClelland and Rumelhart 1986]. It is only valid for continuous activation functions in the supervised training mode. The learning signal for this rule is called delta and it is defined as follows

$$delta = \left[ y_d^p - f(net) \right] f'(w^p x) \quad (2.19)$$

The term $f'(w^p x)$ is the derivative of the activation function $f(net)$ computed for $net=w^p x$. The explanation of the delta learning rule is shown in Figure 2.10. This learning rule can be readily derived from the condition of least squared error between $y^p$ and $y_p$. 

![Diagram of the delta rule](image)
We obtain the error gradient vector value

\[ \nabla J = -(y^p_d - y^p) f'(w^p) x \]  

(2.20)

The components of the gradient vector are

\[ \frac{\partial J}{\partial w_i} = \nabla J \]  

(2.21)

We then obtain from equation (2.16) and (2.20)

\[ \Delta w_i = \gamma (y^p_d - y^p) f'(net) x_i \]  

(2.22)

Note that the weight adjustment as in (2.22) is computed based on minimisation of the squared error.

**2.6.2 Unsupervised Learning**

During unsupervised learning, there is no desired output and there is no ‘teacher’ [Tsoukalas and Uhrig 1997]. The purpose of this type of learning is to organize information into categories. After training, when inputs are applied to the system, the output of the network is the class to which the inputs belong. If no category is found, another is created. An analogy would be presenting a person with different sets of objects and then asking him or her to separate them into groups. When another
object is presented to the person, it can be categorized into one of the groups. If the object does not fall into any of the groups, another is created.

### 2.6.3 Reinforcement Learning

As suggested by the name, the ‘teacher’ only informs the network if its output is acceptable [Tsoukalas and Uhrig 1997]. The difference between this type of learning and supervised learning is that the learning algorithm gives no indication of how close the outputs are to the desired results. A binary ‘pass/fail’ is the only feedback. If the teacher’s indication is ‘bad’, the network readjusts its parameters and tries again and again until it achieves its correct output response. During this process there is no indication to whether the output response is moving in the right direction or how close it is to the correct response. Hence, the process of correcting synaptic weights follows a different strategy than the supervised learning process.

Some parameters to watch are the following: the time per iteration and the number of iterations per pattern to reach the desired output during the training session; whether the neural network reaches a global or local minimum, and when in a local pattern, is it able to escape if it becomes trapped. When reinforced learning is used as a training technique, certain boundaries should be established so that the trainee does not continue trying to illicit the correct response ad infinitum.

### 2.6.4 Competitive Learning

Competitive learning is another form of supervised learning that is distinctive because of its characteristic operation and architecture. In this scheme, several neurons are at the
output layer. When an input stimulus is applied, each output neuron competes with the others to produce the closest output signal to the target. This output then becomes the dominant one and the other output ceases producing an output signal for that stimulus [Kartalopoulos 1996]. For another stimulus, another output neuron becomes the dominant one and so on. Thus each output neuron is trained to respond to a different input stimulus. Competitive learning can also be viewed as a random specialization process. When the NNs with competitive learning are part of a greater NNs system, then, due to connectivity issues, this type of random specialization may not always be desirable. In this case, one might try reinforced learning.

Competitive learning is frequently encountered in groups of people where each member of the group is selected and trained to perform specific tasks based on the principle of the right person at the right time at the right place.

2.6.5 Hebbian Learning

The Hebbian rule was one of the first NN learning laws. It was proposed by Donald Hebb in 1949 as a possible mechanism for synaptic modification in the brain and has been used to train NNs. Hebb’s postulate:

*When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes place in firing it, some growth process or metabolic change takes place in one or both cells such that A’s efficiency, as one of the cells firing B, is increased.*

Thus, the synaptic strength (known as weight $w$) between cell A and cell B is modified according to the degree of corrected activity between input and output. This type of
learning is called Hebbian learning, a term encountered frequently in NNs. Anti-Hebbian learning refers to NNs where the synaptic contacts are inhibitory only.

2.7 Neural Networks Classifiers

2.7.1 Introduction

The background information concerning NN learning and preliminaries were described in the previous sections. Examples of such understandings include the perceptron, multilayered perceptron, learning samples, weight update law, and FNNs. Based on that knowledge we can move forward around the ‘NN atmosphere’. There are many research projects dealing with NN classification. The most general perspective NN system identification as a training task is to view it as an optimization problem in which a performance index is minimized with respect to the parameters being identified [McLoone 1998]. While these diverse interpretations result in differing terminologies and representations, the algorithms involved are essentially equivalent. From this perspective neural modelling structures can be classified as linear, nonlinear and hybrid.

In this section the use of McLoone’s classification method is presented. Latest optimization techniques between 1998 and now will be investigated in this section.

2.7.2 Linear Optimisation

Linear models with nonlinear parameters (pre-selected) lead to linear optimization problems with a least-squared formulation. Linear optimization is the oldest
optimization technique and such problems have been extensively studied. A variety of optimization techniques have been developed to solve them. Baldi and Hornik (1996) conducted a notable investigation dealing with linear neural networks. They treated the case of supervised learning with BP and the corresponding autoassociative as a special case. Moreover the researchers studied the problems of validation, generalisation, and over fitting in simple one-layer network trained to learn the identity map. Since mid 1990, some researchers have been studying and composing new algorithm belonging to this group, because of the simple condition of calculation.

One simple design is piecewise linear NN [Chandrasekaran and Manry 1999]. Piecewise linear NN, which assigns each input vector to the appropriate clustering using a distance measure, and the input vector is then multiplied by a matrix to generate an output vector. The next fast learning algorithm is Step Acceleration based Training Algorithm (SATA) presented by Li, Wang and Zhang (2002), which does not need to calculate the gradient of the target function. In 2004, a new fast learning algorithm called Extreme Learning Machine (ELM) was proposed [Huang, Zhu and Siew 2004] for Single Hidden Layer Feedforward Neural Networks (SLFNs) which randomly choose the input weights and analytically determines the output weights of FNNs. As the input weights and biases of hidden layer are fixed, the single SLFN can be considered as a linear system.

In the new initialisation method for neural networks, sensitivity analysis was used [Berdinas and Romeo 2006]. First, random values are assigned to the outputs of the first layer, second, these initial values are updated based on sensitivity formulas, and finally
the weights are calculated using a linear system of equations. This method presented the advantage of achieving a good solution in just one epoch using few computational times.

The latest work named as local linear NN, based on the principal component analysis [Ramrath, Munchhof and Iserman 2006], was used to develop an estimator of parameters of a linear process model. The developed estimator showed Total Least Squares (TLS) properties and is therefore most suitable in Errors In Variables (EIV) scenarios where both input and output measurements are corrupted by noise.

2.7.3 Nonlinear Optimisation

The system identification task with a nonlinear model where all parameters are adapted is much more difficult. Problems are highly nonlinear and ill-conditioned cost functions with multiple local minima and plateaux. The first major breakthrough in NN identification came with the development of the BP routine by Rumelhart et al (1986), which is a computationally efficient method for computing the gradient of cost function with respect to hidden layer weights of neural networks. The development of more powerful NN training algorithms has been the subject of much research over the last decades. These developments have included extensions to BP such as SBP (Standard BP) with generalized learning [Fine 1999].

In 1999, as a result of using the Error Saturation (ES) prevention method, Lee, Huang and Chen (1999) improved the learning speed of the BP algorithm. Error Saturation Method (ESM) is simple and intuitive to prevent ES condition during the learning
process. A new method for faster NN learning introducing functions of synaptic weights [Obayashi and Kobayashi 1999] was also introduced in the same year. For applying this method researchers calculated sensitivity of the criterion functions with respect to the synaptic weights. The main characteristic of this method is the introduction of the functions of synaptic weights in order to improve the sensitivity of the criterion functions with respect to the synaptic weights. At the same time, Magoulus, Plagianakos and Vrahatis (1999) proposed to use different learning rates for each weight. Their proposal was successfully completed due to Newton’s selection method and Lipschitz’s constant for different learning rates. Bhaumik, Banerjee and Sil proposed a new supervised training algorithm for FNNs [Bhaumik et al 1999]. Instead of applying single valued input-output information at time, multivalued information in the form is applied to each node of the input-output layer. The algorithm was tested successfully for a wide range of data, mainly, when values of output are low compared to input values. Another simple learning algorithm was presented that achieves the learning of the derivatives of a function while learning the function [Basson et al 1999]. Duch proposed the use of the Global Optimization (GO) method in learning the gradient descent algorithm [Duch 1999]. Duch’s GO has two methods: firstly an initialization method and secondly an ensemble method.

In 2000, a new method for BP algorithm to avoid local minimum was proposed by means of gradually adding training data and hidden units [Wei et al 2000]. The new algorithm was not only quite simple to calculate, but also very effective. Tadic’s new normalized stochastic gradient algorithm is based on new methodology which is close to the approach based on the martingale convergence argument [Tadic et al 2000]. Oh
and Lee proposed an adaptive learning rate to markedly reduce the sensitivity of Multi Layer Perceptron (MLP) performance to the order of $n$th - cross entropy error function [Oh et al 2000]. But this learning rate has a weakness that it may have to take a very large value during a process of learning. Hunter produced a new algorithm which is based upon a novel iterative estimation of the principal eigen-subspace of the Hessian, together with a quadratic step estimation procedure [Hunter 2000]. This algorithm (call EQUAL, for Eigenvector-based QUAdratic Learning) had superior performance in the early and middle stages of convergence in comparison with the conjugate gradient descent algorithm, but inferior performance during the terminal stage. Also the algorithm was efficient only for a fairly small subspace.

Abid, Fnaiech and Najim proposed a modified BP algorithm [Abid et al 2001] based on a criterion taking into account the linear and nonlinear signal errors and new term was added to SBP algorithm. The convergence of the new algorithm requires less iteration than SBP in all fields. Due to that the LMBP algorithm demands huge amount of memory, Wilamowski et al proposed two modifications to the LMBP algorithm [Wilamowski et al 2001]. One modification is made on performance index, while the other one is on calculating gradient information. The modified algorithm gives a better convergence rate compared to the standard LMBP algorithm, and it is also less computationally intensive and requires less memory.

In 2001, many different algorithms (based on different methods for FNNs) were proposed. One of them is Yu and Liu’s simple procedure in BP training [Yu and Liu 2001]. This method is based on the BP algorithm which employs an adaptive learning
rate and momentum factor to reduce the training time. For achieving this, the researchers proposed two rules. Firstly, each weight has its own learning rate and momentum factor. Secondly, both learning rate and momentum factors are adaptively adjusted at the each iteration. Several new gradient-based learning algorithms were also proposed. Wang and Huang’s approximation method was effective but only for solving discrete regular equations [Wang et al 2001]. Another algorithm, using localised learning was introduced by Weaver, Baird and Polycarpou (2001). The new gradient based NNs for solving linear and quadratic programming problems was proved to be efficient [Leung et al 2001]. The new algorithm was constructed on the basis of duality theory, optimisation theory, convex analysis theory, Lyapunov stability theory, and LaSalle invariance principle to solve a linear and quadratic program problem. The complete proof of the stability and convergence of the proposed network was given.

In 2002, several nonlinear optimisation training algorithms were proposed for FNNs: Exponentiated Backpropagation Gradient Descent (EGD) for multilayer FNNs algorithms [Williamson et al 2002] and its applications in various tasks [Srinivasan et al 2002], BP with Adaptive Learning rate and Momentum term (BPALM) [Yu and Liu 2002]. Using two novel error function families algorithm [Jiang et al 2002], the natural gradient learning algorithm for second order nonstationary source separation [Choi et al 2002] and scaled version of conjugate gradient method [Sotiropulos, Kostopoulos and Grapsa 2002] were interesting algorithms for nonlinear optimisation..

Another attractive solution was a new paradigm of single layer FNNs that included dendritic processes [Ritter and Ianchu 2003]. Dendritic structure function as many
functional subunits, each unit being capable of realising logical operations. Based on the two theorems for lattice algebra, researchers developed training algorithms which always drew closed regions around pattern clusters. Shaheed (2004) completed an analysis of different types of conjugate gradient algorithms in the nonlinear dynamic modelling of Twin Rotor Multi-input-Multi-output Systems (TRMD). Four different types of Conjugate Gradient Algorithms (CGAs) were used in that investigation for supervised learning and their performance was compared in terms of input-output mapping and speed of convergence. The performance of the Scaled Conjugate Gradient Algorithm (SCGA) which was designed to avoid a time-consuming line search by combining the model-trust region approach used by the LMBP algorithm with the conjugate gradient approach, was ultimately better than that of the others. He also pointed out that a model with a minimum number of neurons in different layers of the network made the process faster.

The GO algorithm for FNNs supervised learning is another way to improve learning performance [Jordanov 2004]. The difference with this algorithm is that it utilised a stochastic technique based on the use of low discrepancy sequences. The Particle Swarm Optimisation (PSO) is one of the basements for NN learning technique. The two most commonly used methods were illustrated as gbest model and lbest model in PSO [Liu et al 2004]. The gbest model converges quickly on problem solutions but has the weakness of becoming trapped in local optima, while the lbest model is also to ‘flow around’ local optima, as the individuals explore different regions. Furthermore, Guerra and Coelho proposed radial basis NN learning based on PSO [Guerra and Coelho 2005].
This latter proposal used clustering methods to tune the centres of the Gaussian functions used in the hidden layer of a radial basis function based NN.

Due to the fact that PSO combined algorithms are computationally expensive, a new simple and easy to implement algorithm based on a Lyapunov stability theory was developed [Behara et al 2004]. This algorithm was used for network inversion as well as controller weight adaptation [Behara et al 2006]. The memory restricted BFGS (Broyden, Fletcher, Goldfard and Shanno) training algorithms became another development of fast learning algorithms for fixed size MLP networks [Asirvadam and Irwin 2004]. Two memory efficient training algorithms (Variable Memory (VM) and Optimal Memory (OM) BFGS) which bridge the gap between matrix and vector based NNs training algorithms showed superior performance. Overall OM BFGS showed best results with less memory and computational requirement in comparison to VM BFGS and provided a variable option. BP Magnitude Gradient Function (MGFPROP) and Deterministic Weight Modification (DWM) were also proposed to speed up convergence rate and improve the global convergence capability of the BP algorithms [Ng et al 2004]. Pattern search method based BP algorithm was one of the interesting attempts of new, simple learning algorithms [Wang et al 2004]. The learning model had two phases: a pattern search phase, and a local minimum escaping phase.

Another simple solution for FNN learning is a novel link structure [Wu 2004]. In order to increase the learning speed, Wu expanded the link from hidden layer to output layer. The learning algorithm was derived on the basis of the new structure. Kathirvalavakumar and Tangavel (2006) proposed another new simple, efficient
learning algorithm for single hidden layer NNs. The idea of this algorithm is to train the output layer and hidden layer separately. Zhang, Wu and Zheng (2006) developed a gradient based method whose momentum is to prove convergence of the resulting gradient method with momentum, whereas the error function was not assumed to be quadratic. On the other hand, other researchers found more effective changing traditional gradient methods [Wu et al 2005]. Vishwanathan and Schraudolph united Conditional Random Fields (CRFs) and stochastic gradient methods for Stochastic Meta-Descent (SMD) [Vishwanathan and Schraudolph 2006]. The CRF have gained popularity in the machine-learning community [Lafferty et al 2001, Sha and Pereira 2003, Kumar and Hebert 2004]. Stochastic gradient methods are online and scale sub-linearly with the amount of training data, making them very attractive for large data sets. On the other hand, some researchers have improved the Bayesian method for multioutput FNNs topology [Rossi & Vila 2006]. They used a specific recourse to the Bayesian conjugate prior theory and to the so-called empirical Bayes approach.

Another significant approach is the Lyapunov stability theory. Use of Lyapunov stability theory in NN learning is well known. Yu et al (2002) derived a generalized weight update law using a Lyapunov function that guarantees global convergence. In another work, Yu et al (2001) used the Lyapunov stability theory to derive a stable learning law for multilayer dynamic NNs. They showed that their learning algorithm is similar to the BP algorithm for MLP with an additional term which ensures the stability of the identification error. Moreover, Lyapunov stability theory-based learning algorithms were proposed for FNNs [Behera, Kumar and Patriak 2006]. Lyapunov function-based learning algorithm (LF I) and its modified version (LF II) have exact

Second order BP computes the exact Hessian matrix of a given objective function. Mizutani et al (2005) described two algorithms for FNN learning with emphasis on how to organise Hessian elements into a so-called stagewise-partitioned block-arrow matrix form; (1) stagewise BP, an extension of the discrete-time optimal-control stagewise and (2) nodewise BP, based on direct implementation of the chain rule for differentiation. The FNN with Gaussian function is one of the new, interesting approaches for NN learning improvement [Han and Hou 2006]. Firstly, researchers made a new and quantitative proof of the fact that a single layer NNs with \( i+1 \) hidden neurons can learn \( i+1 \) distinct sample with zero error. Then they gave approximate interpolants. A new adaptive BP algorithm based on Lyapunov stability theory for NNs was another promising algorithm [Man et al 2006].

Another important feature of Lyapunov’s adaptive BP algorithm is that although NNs may have bounded input disturbances, the effects of the disturbances can be eliminated through adaptively updating the weights according to Lyapunov stability theory. The excellent robustness property with respect to large bounded input disturbances and fast error convergence was shown. A new learning algorithm for function approximation by
incorporating *a priori* information into FNNs was introduced [Han and Ling 2007]. The algorithm incorporated two kinds of constraints into single hidden layered FNNs, which are architectural constraints and connection weight constraints.

At the same time, the Systematic Trajectory Search Algorithm (STSA) for FNNs training was presented by Tseng and Chen (2007). The STSA utilizes the Orthogonal Array (OA) to uniformly generate the initial population in order to globally explore the solution space, and then it applies a novel trajectory search method that can exploit the promising area thoroughly. Also Zhang, Wu and Zheng proved the convergence of the resulting gradient method with momentum, whereas they the error function was not assumed to be quadratic [Zhang *et al* 2007]. The learning rate was set to be a constant and the momentum factor an adaptive variable. Both the strong and weak convergence results were proved.
2.7.4 Hybrid Optimisation

The third part of hybrid optimisation is a new class of training algorithms which allows the nonlinear and other methods (such as genetic algorithm, unsupervised learning) to be estimated simultaneously using nonlinear optimization techniques. In 1999, Efe and Kaynak presented a hybrid training algorithm for FNNs which combines the gradient descent technique with Variable Structure Systems (VSS). The objective of this research was to develop a training procedure for FNNs, which would enforce the adjustable weights and biases to settle down to a steady state solution while minimising an appropriate cost function.

Their next work presented stability of FNN training by utilizing the gradient information [Efe and Kaynak 2000]. The method proposed to construct a dynamic model of the conventional update mechanism and derive the stabilising values of the learning. This was achieved by the old technique with the integration of EBP and VSS. Other researchers were trying to combine the standard BP algorithm and improvement over the Fletcher-Reeves method [Jiang et al 2000, 2004]. This hybrid algorithm of global optimisation of the dynamic learning rate for FNNs provided its merit of its speed and simplicity. Also another attempt was proposed which combined BP and GA (Genetic Algorithm) [Lu and Chen 2000]. BP and GA are among the most effective and simple algorithms of NNs. As a deterministic GD algorithm and stochastic optimising algorithm respectively, there exists great compensability between their advantages and disadvantages. The proposed hybrid GD-GA learning method for multilayer FNNs blends the merit of both BP and GA. Simulation results also showed effectiveness of
this new method. One of the interesting solutions for hybrid NNs is a combination of RBF (Radial Bases Function) and MLP networks [Ciocoiu 2002].

Another interesting work in this field is a study on relations between FNNs and wavelet transformation [Liu et al 2002]. Wavelet transformation is one of the most efficient and important tools on the signal processing. It is known that the particular advantages of NNs are embodied in the model identification, function approximations, as well as the detail description and features extraction of wavelet analysis to the non-stationary random signal. A lot of research was done. The notion of ‘wavelet network’ was proposed by Zhang and Benveniste in 1992. Zhang etc (1995) concluded that wavelet network was more suitable for function approximants, Pati and Krishnaprasad (1993) presented the analysis and synthesis of FNNs using discrete affine wavelet transformation.

Multiwavelet NNs and its approximation [Jiao et al 2001], the relationship of FNNs and wavelet transformation [Liu et al 2002], the integration of magnified gradient function and weight evolution algorithms [Ng, Leung and Luk 2002], Magnified Gradient Function Learning Algorithm (MGFPROS) [Ng et al 2000], the robust algorithm for NNs [Manic & Wilamowski 2002] were another examples of hybrid algorithms.

Liu proposed a hybrid neural networks learning system [Liu 2004]. This research studied negative correlation learning which was based on both supervised and unsupervised learning. Negative correlation learning combines supervised learning by allowing each NN to learn the target output with unsupervised learning by minimising
the correlations among all the NNs in an ensemble. Jin, Okabe and Sendhoff proposed a combined NN learning which was named as a NN regularisation and ensembling using multi-objective evolutionary algorithms [Jin et al 2004]. This approach exhibited two advantages over traditional regularization techniques. Firstly, a number of NNs of a spectrum of model complexity instead of one single neural network can be obtained in one optimization run. Secondly, a new and more direct regularizer can be used. These ideas were successfully demonstrated on a test problem using the Dynamic Weighted Aggregation (DWA) and Non-Dominated Sorting Genetic Algorithm (NSGA-II) algorithms.

One of the latest hybrid algorithms is a combined GA and orthogonal transformation for designing FNNs [Xu and Ho 2007]. Only the connections and parameters between the input layer and hidden layer were tuned using GA, while the connections and parameters between hidden layer and output layer was tuned using the orthogonal transformation. In simulations, single hidden layer FNNs were used for sunspot number forecasting.

### 2.8 Backpropagation Learning Algorithms

#### 2.8.1 Basic Concept

GDBP algorithms have been widely used, well investigated and one of the most popular learning algorithm class for FNNs. Their effectiveness has been well documented. For example, an online GDBP learning algorithm for time-varying inputs [Zhao 1996], an adaptive learning algorithm with reduced complexity [Zhou and Si 1998], a fast
learning algorithm based on the gradient descent of neuron space [Parisi et al 1995], the LMBP algorithm [Hagan and Menhaj 1994], and a general BP learning algorithm for FNN [Yu, Efe and Kaynak 2002]. A common dynamic characteristic in this class of algorithm is that because of the fixed learning rate, an asymptotic convergence is incurred, the result being that the closer to the desired weights, the slower the convergence speed - making it difficult for fast convergence with higher precision tolerance.

For the subsequent derivations, we introduce some standard notions and terminologies.

Denote the inputs, weights, desired outputs and actual outputs of the FNNs as

\[ x(t) = (x_1, x_2, \cdots, x_n)^T \in \mathbb{R}^n \]  \hspace{1cm} (2.28)

\[ w(t) = (w_1, w_2, \cdots, w_l)^T \in \mathbb{R}^l \]  \hspace{1cm} (2.29)

\[ y_d(t) = (y_{d1}, y_{d2}, \cdots, y_{dm})^T \in \mathbb{R}^m \]  \hspace{1cm} (2.30)

\[ y(t) = (y_1, y_2, \cdots, y_m)^T \in \mathbb{R}^m \]  \hspace{1cm} (2.31)

where \( x(t) \) is the input vector, \( w(t) \) the weight vector (including weights for input, output and hidden layers), \( y_d(t) \) the desired output vector and \( y(t) \) the output vector of the FNNs. The error at any instant is represented as

\[ e(t) = \frac{1}{2} (y(t) - y_d(t))^T (y(t) - y_d(t)) = \frac{1}{2} \| y(t) - y_d(t) \|^2 \]  \hspace{1cm} (2.32)

where the symbol ‘\( T \)’ represents the transpose, \( \| \cdot \| \) - Euclidean norm. Note that here the input \( x(t) \) is of a general type, and it can be discrete, continuous and time varying. The weight vector \( w(t) \) represents weights for perceptrons (single layer FNNs) as well as multi-layer FNNs. The criterion for evaluating the performance of the FNNs learning is
now developed. Since the inputs can be time-varying, a time window should be used to evaluate the training efficiency, that is

\[ J = \frac{1}{\tau} \int_{t-\tau}^{t} e(\theta) d\theta \]  

(2.33)

where \( \tau \) is the length of the time window. The formula (2.33) is particularly useful for on line continuous-time learning as it considers evolution of learning in an average sense within a prescribed time-window. However, for discrete data sets and since the evaluation of errors can only be done at ‘discrete moments’, (2.33) can be rewritten as

\[ J_k = \lim_{\tau \to 0} \frac{1}{\tau} \int_{t_k-\tau}^{t_k} e(\theta) d\theta = e(k) \]  

(2.34)

which becomes the usual form for training FNNs with discrete data sets.

### 2.8.2 GDBP Algorithms

Most GDBP learning algorithms for FNNs can be considered as finding zeros of \( \frac{\partial J}{\partial w} \), which corresponds to their local as well as global minima. The convergence properties of the GDBP learning algorithms can be shown below using the Lyapunov theory by treating \( J \) as the Lyapunov function. The learning of the weight vector \( w \) can be considered as a ‘control’ to be determined to minimize the Lyapunov function \( J \). Taking time-derivative of \( J \) which leads to

\[ \frac{dJ}{dt} = (\nabla_w J)^T \frac{dw}{dt} \]  

(2.35)

where \( (\nabla_w J)^T = (\frac{\partial J}{\partial w_1}, \cdots, \frac{\partial J}{\partial w_i}) \). The continuous-time form of the GDBP learning algorithms can be written as
\[
\frac{dw}{dt} = -\gamma \nabla_{w} J
\]  
(2.36)

where \( \gamma > 0 \) is a constant learning rate, such that Equation (2.35) becomes

\[
\frac{dJ}{dt} = (\nabla_{w} J)^{T} \frac{dw}{dt} = -\gamma (\nabla_{w} J)^{T} \nabla_{w} J = -\gamma \| \nabla_{w} J \|^{2}
\]  
(2.37)

which result in the weight vector \( w \) settling in the equilibrium satisfying \( \nabla_{w} J = 0 \). The common problem is that GDBP learning algorithms based on (2.36) often result in asymptotical convergence because of the constant learning rate \( \gamma > 0 \). This fact can be observed from (2.37). When \( \nabla_{w} J \to 0 \), the convergence speed of \( \frac{dJ}{dt} \) slows down asymptotically.

### 2.8.3 An Learning Example

Figure 2.11 illustrates the flowchart of the GDBP training algorithm for a basic two-layer network. The learning begins with the feedforward recall phase (Step 2). After a single pattern vector \( x \) is submitted at the input, the layers responses \( y \) and \( o \) are computed in this phase. Then, the error signal computation phase (Step 4) follows. Note that the error signal vector must be determined in the output layer first, and then it is propagated toward the network input nodes. The output weights are subsequently adjusted in Step 5. Finally, hidden weights are adjusted in Step 6.

Note that the cumulative cycle error of input to output mapping is computed in Step 3 as a sum over all continuous output errors in the entire training set. The final error value for the entire training cycle is calculated after each completed pass through the training.
set \( \{z_1, z_2, \ldots, z_p\} \). The learning procedure stops when the final error value below the upper bound, \( J_{\text{max}} \), is obtained as shown in Step 8.

Figure 2.11 BP Algorithm Framework
2.8.4 Assessment of Latest BP Algorithms

LMBP Algorithm

Since the BP learning algorithm was first popularized, there has been considerable research on methods to accelerate their convergence. This research falls roughly into two categories [Medsker 1994]. The first category involves the development of ad hoc techniques. These techniques include such ideas as varying the learning rate, using momentum and rescaling variables. Another category of research has focused on standard numerical optimisation techniques. The most popular approaches from the second category have used conjugate gradient or quasi-Newton methods [Dennis et al 1983]. The quasi-Newton methods are considered to be efficient too, but their storage and computational requirements increase as the square of the size of the network. There have been some limited memory quasi-Newton algorithms that speed up convergence while limiting memory requirements. If exact line searches are used, the one step secant methods produce conjugate directions. Another area of numerical optimisation that has been applied to NNs is nonlinear least squares. The most general optimisation methods were designed to work effectively on all sufficiently smooth objective functions.

The LMBP algorithm is an approximation of Newton’s method [Zurada 1992]. If $J(w)$ is the performance index which we want to minimise with respect to the weight vector $w$, then Newton’s method would be

$$
\Delta w = -\left[\nabla^2_w J(w)\right]^{-1}\nabla_w J(w)
$$

(2.38)
where $\nabla_w^2 J(w)$ is the Hessian matrix and $\nabla_w J(w)$ is the gradient. If the sum of square error is $J$ defined then it can be shown that

$$\nabla_w J(x) = M^T(w)e(w) \quad (2.39)$$

$$\nabla_w^2 J(w) = M^T(w)M(w) + S(w) \quad (2.40)$$

where $M(w)$ is the Jacobian matrix

$$M(x) = \begin{bmatrix} \frac{\partial e_1(w)}{\partial w_1} & \frac{\partial e_1(w)}{\partial w_2} & \ldots & \frac{\partial e_1(w)}{\partial w_n} \\ \frac{\partial e_2(w)}{\partial w_1} & \frac{\partial e_2(w)}{\partial w_2} & \ldots & \frac{\partial e_2(w)}{\partial w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_N(w)}{\partial w_1} & \frac{\partial e_N(w)}{\partial w_2} & \ldots & \frac{\partial e_N(w)}{\partial w_n} \end{bmatrix} \quad (2.41)$$

and

$$S(w) = \sum_{i=1}^{N} e_i(w) \nabla^2 e_i(w) \quad (2.42)$$

For the Gauss-Newton method it is assumed that $S(w) \approx 0$, and the update (2.38) becomes

$$\Delta w = \left[ M^T(w)M(w) \right]^{-1} M^T(w)e(w) \quad (2.43)$$

The Levenberg-Marquardt modification to the Gauss-Newton method is

$$\Delta w = \left[ M^T(w)M(w) + \mu I \right]^{-1} M^T(w)e(w) \quad (2.44)$$

where $I$ – identity matrix. The parameter $\mu$ is multiplied by some factor whenever a step would result in an increased $J$. 

55
**Modified BP Algorithm**

A novel learning algorithm was proposed for single hidden layer FNNs (Kathirvalavakumar and Thangavel 2006). In this algorithm, the hidden and output layers were trained separately. The output layer trained using the modified standard BP algorithm is:

$$\Delta w_{\text{output}} = \gamma e_1 y(i) + y'(x) e_2 y(i) \tag{2.45}$$

where $\lambda$ is the learning coefficient, $e_1$ is a linear and $e_2$ is a nonlinear errors.

Fictitious teacher signals for the output of hidden layer neurons are determined algebraically as described by Yamamoto and Nikiforuk (2000). Then the hidden layer is trained using the proposed optimisation criterion. The weight update rule for the hidden layer shall be derived as follows:

$$\Delta w_{\text{hidden}} = \gamma y(i-1)e_1 + yf'(i)e_2 + \gamma e_3 y \tag{2.46}$$

where $e_3$ is a linear error using temporal teacher signal, $\tau$ is a constant parameter usually 0.0005.

**Gradient Descent with Momentum**

Another technique to reduce training time is the use of momentum because it enhances the stability of the training process. Momentum is used to keep the training process going in the same general direction analogous to the way that momentum of a moving object behaves. This involves adding a term to the weight adjustment that is proportional to the amount of the previous weight change. In effect, the previous adjustment is ‘remembered’ and used to modify the next change in weight. Hence, equation of GDBP (2.36) now becomes
\[ \Delta w(i+1) = -\gamma \nabla_w J + \mu \Delta w(i) \]  

(2.47)

where \( \mu \) is the momentum coefficient (typically about 0.8).

Relationship between the old weights and the updated ones are shown in Figure 2.12. The new value of the weight then becomes equal to the previous value of the weight plus weight change of equation, which includes the momentum term.

![Figure 2.12 Influence of momentum upon weight change](image)

This process works well in many problems, but not so well in others. Another way of viewing the purpose of momentum is to overcome the effects of local minima.

**Gradient Descent with Adaptive Learning Rate**

Gradient Descent with adaptive learning rate can train any network as long as its weight, net input and transfer functions have derivative functions. BP is used to calculate derivatives of performance with respect to the weight and bias variables. Each variable is adjusted according to gradient descent:
At each epoch, if performance decreases toward the goal, then the learning rate is increased by the constant parameter. If performance increases by more than the maximum parameter, the learning rate is adjusted by another constant parameter and the change that increased the performance is not made.

**Gradient Descent with Momentum and Adaptive Learning Rate**

This algorithm is used to calculate derivatives of performance with respect to the weight and bias variables. Each variable is adjusted according to adaptive learning rate with momentum,

\[
\Delta w(i+1) = \mu \Delta w(i) + \gamma_{ad} \mu \Delta w(i)
\]  

(2.49)

For each epoch, if performance decreases toward the goal, then the learning rate is increased by the constant parameter. If performance increases by more than the maximum parameter, the learning rate is adjusted by another constant parameter and the change that increased the performance is not made.

**Bayesian Regularization Algorithm**

Bayesian regularization algorithm [MacKay 1992] can train any network as long as its weights, net inputs, and transfer functions have derivative functions. This regularization algorithm minimizes a linear combination of squared errors and weights. It also
modifies the linear combination so that at the end of training the resulting network has good generalization qualities. This Bayesian regularization takes place within the LM algorithm. BP is used to calculate the $M(x)$ of performance with respect to the weight and bias variables $w$. Each variable is adjusted:

\[ jj = M(w)M(w) \] (2.50)
\[ je = M(w) J(w) \] (2.51)
\[ \Delta w(i) = -(jj+I \gamma)/ je \] (2.52)

where $I$ is the identity matrix.

The adaptive value $\gamma$ is increased by $\gamma_{inc}$ until the change shown above results in a reduced performance value. The change is then made to the network, and $\gamma$ is decreased by $\gamma_{dec}$.

**BFGS Quasi-Newton Backpropagation**

Quasi-Newton [Gill and Murray 1981] can train any network as long as its weight, net input and transfer functions have derivative functions. Each variable is adjusted according to the following formula:

\[ \frac{\partial J}{\partial w} = \frac{\partial J}{\partial w} + ad_x \] (2.53)

where $d_x$ is the search direction.

The parameter $a$ is selected to minimize the performance along the search direction. The line search function is used to locate the minimum point. The first search direction is the
negative of the gradient of performance. In succeeding iterations the search direction is computed according to the following formula:

\[ d_s = -H/\nabla_w J \]  

(2.54)

where \( \nabla_w J \) is the gradient and \( H \) is an approximate Hessian matrix.

**Conjugate Gradient BP with Powell-Beale Restarts**

Conjugate Gradient BP with Powell-Beale restarts algorithm [Powell 1977] can train any network as long as its weight, net input and transfer functions have derivative functions. Each variable is adjusted according to the formula (2.53). In succeeding iterations the search direction is computed from the new gradient and the previous search direction according to the formula

\[ d_s(i+1) = -\nabla_w J + d_s(i) Z \]  

(2.55)

The parameter \( Z \) can be computed in several different ways. The Powell-Beale variation of conjugate gradient is distinguished by two features. Firstly, the algorithm uses a test to determine when to reset the search direction to the negative of the gradient. Secondly, the search direction is computed from the negative gradient, the previous search direction and the last search direction before the previous reset.
**Conjugate Gradient BP with Fletcher-Reeves Updates**

CGBP with Fletcher-Reeves [Scales 1985] can train any network as long as its weight, net input and transfer functions have derivative functions. This algorithm is also used to calculate derivatives of performance with respect to the weight and bias variables $w$. The search direction is computed from the new gradient and the previous search direction, according to the formula

$$d_{s(i+1)} = -\nabla_w J + d_s(i)Z_{FR}$$  \hspace{1cm} (2.56)

For the Fletcher-Reeves variation of conjugate gradient it is computed according to

$$Z_{FR} = \frac{\text{normnew}_sqr}{\text{norm}_sqr}$$  \hspace{1cm} (2.57)

where $\text{norm}_sqr$ is the norm square of the previous gradient and $\text{normnew}_sqr$ is the norm square of the current gradient.

**One Step Secant Algorithm**

The one step secant algorithm [Battiti 1992], can train any network as long as its weight, net input, and transfer functions have derivative functions. The first search direction is the negative of the gradient of performance. In succeeding iterations the search direction is computed from the new gradient and the previous steps and gradients according to the following formula:

$$d_{s(i+1)} = -\nabla_w J + Ac \Delta w(i) + Bc \nabla_w J(i+1);$$  \hspace{1cm} (2.58)

where $Ac$ and $Bc$ are constant parameters.
2.9 Practical Applications

NNs have been applied in solving many practical problems. Typical examples include:

- System identification and control, process control, vehicle control
- Classification, including pattern and sequence recognition, novelty detection
  Data processing, including filtering, clustering, blind signal separation and compression
- Facial expression, detecting entire face images using Two Dimensional (2D) discrete transform
- Game-playing and decision making backgammon, chess, racing
- Multimedia, signal processing from MPEG-1, 2, 4
- Medical diagnosis
- Financial application
- Data mining or Knowledge Discovery in Database (KDD)

There have been extensive researches done in control engineering using NNs, for example, Seul and Sung (2008) presented a NN control for a wheel-driven mobile inverted pendulum. Dong and Youdan (2006) presented an NN based adaptive controller for a class of discrete-time multi-input multi-output nonlinear systems.
Classification including pattern recognition, detection and sequential decision making have obtained numerous applications, such as Zernike Moment Based Image Registration [Wu and Xie 2004], Pattern Recognition based Detection and Localization in a Network of Randomly Distributed Sensor Nodes [Hertani and Ilow 2005], a Pyramidal NN for Visual Pattern Recognition [Phung and Bouzerdoum 2007], Novel Detection for a Neural Network-Based Online Adaptive System [Liu et al 2005] and The Autoregressive BP Algorithm for Sequence Recognition [Leighton and Conrath 1991] etc. Most of above works proposed new training algorithms and implemented that with practical applications.

Data processing has also used NNs, for example, in compression and data mining. Rahman (2006) proposed rule mining methods using single layered NNs. Another key contribution was the proposal of rule mining methods using supervised NNs. Three rule mining algorithms using single-layered NNs were proposed for the three major classes of rules on the basis of the proposed theorems.

Face detection is another large area for NN applications, such as fast neural implementation of Principle Component Analysis (PCA) for face detection [El-Bakry and Zhao 2006], NN-Based Face Detection [Rowley et al 2006], and Facial Expression recognition Using Constructive FNNs [Ma and Khorasani 1995] and Fast Modular NNs for Human Face Detection [El-Bakry et al 2000] can represent major work for this area.

NN technologies present a unified solution to a broad spectrum of multimedia applications such as NN application for Multimedia Processing [Bojkovic et al 2000],
Video Watermarking based on NNs [El'arbi et al 2006], a NN based Test Bed for Evaluating the Quality of Video Streams in IP Networks [Frank and Incera 2006] and Face and Eye Rectification in Video Conference Using NNs [Yip 2005] etc.

Medical and financial applications also use NNs intensively. For medical applications, typical works include Improved Neural Network-Based Interpretation of Colonoscopy Images through On-line Learning and Evolution [Magoulas et al 2001], NN based Medical Decision Support Tools for predicting transfusion requirements of emergency room patients [Walczak 2005], Data Truncation Artifact Reduction in Magnetic Resonance Imaging Using a Multilayer NN [Yan and Mao 1993], Uncertainty in the Output of NNs [Jiang et al 2004] and Financial Application Knowledge Discovery for Large Data Sets using NNs [Shobha and Sharma 2005], and a sensitivity analysis for finance applications using NNs [Tsaih 1999].

Other examples include a variety of applications in different areas, such as in pattern recognition for recognising speech and optical patterns, in security systems for fingerprint and/or voice identification to validate and confirm the owners, and in robotics for guiding and controlling movement, recognising optical patterns, and manipulating objects. One of the latest practical fields using NNs is the FNN Implementation in FPGA using Layer Multiplexing for Effective Resource Utilization [Himavathi et al 2007] which proposed a simple architecture to implement a complete NN using minimum resource regardless of the size of the network. The second example is the application of FNN to Wideband Code Division Multiple Access (WCDMA) power amplifier model [Sonbai et al 2005]. This work proposed a FNN model for the
class B Power Amplifier for WCDMA communication systems amplifier nonlinear characteristics.

2.10 Summary

This chapter has reviewed the NNs and their learning processes extensively, which are beneficial for understanding their basic elements, major parameters, learning or training processes and latest learning algorithms and applications. It has also investigated the main classifications of linear, nonlinear and hybrid latest training algorithms for NNs.

As shown in the chapter, slow convergence has been a major problem for all BP learning algorithms. In the following chapters, new fast BP learning algorithms will be developed based on the terminal attractor concept and various simulations and applications will be undertaken to show the effectiveness of the proposed new algorithms.
Chapter 3

Terminal Attractor Based Backpropagation Algorithms

3.1 Introduction

The previous chapter has presented a review of NNs. Since the first NN learning algorithms were created, many new and more efficient algorithms have been proposed. Each algorithm has both advantages and disadvantages. As we mentioned in the previous chapter, NN learning algorithms can be divided into three major groups: linear, nonlinear and hybrid. The biggest and most effective candidates are from are the nonlinear optimisation based algorithms.

This chapter investigates the TABP learning algorithm based on the terminal attractor concept and their performance analysis in the neighbourhood of the global minimum. We will show that the TABP learning algorithm is much faster when weights are close to desirable ones. However, when the weights are far away from the ideal weights, the GDBP algorithm still over-performs the TABP algorithm. It would be desirable to improve the convergence when the initial weights are far away from the ideal weights.
The new Fast Terminal Attractor Based Backpropagation (FTABP) algorithm is then proposed which is able to combine the advantages of both the TABP and GDBP algorithms, based on a faster convergent new terminal attractor. Firstly, a detailed convergence analysis is given and then how this new algorithm enables a faster convergence both at a distance and from range closer to the ideal weight is showed.

This chapter also presents our innovative proposal of an improved LMBP algorithm based on the terminal attractor concept. Various simulation studies are presented to show the effective learning performance of our FTABP and LMBP algorithms.

This chapter is organized as follows: Section 2 outlines the terminal attractor concept. Section 3 presents the TABP algorithm. Section 4 introduces a fast terminal sliding mode concept. Section 5 proposes the FTABP algorithm. Section 6 provides an improved Levenberg-Marquardt algorithm based on the terminal attractor concept. Section 7 presents simulation results using the above algorithms and Section 8 presents a summary of this chapter.
3.2 About Terminal Attractor Idea

Zak (1989) firstly proposed the terminal attractor concept in system theory. The idea of terminal attractor is based upon violation of the Lipschitz condition at a fixed point. Since the Lipschitz condition is violated, the fixed point becomes a singular solution which envelopes the family of the regular solution. Nonlinear dynamic systems satisfying the Lipschitz condition have a unique solution for each initial condition and the trajectory of the solution is able to approach equilibrium points asymptotically, the trajectory can never reach these points but can approach them as closely as possible. If the system is of the ill condition, the time of convergence will be long; especially as it may spend a lot of time at the end of the transition. Unfortunately, NNs are often like an ill conditioned system [Owens and Filkin 1989]. The learning transient processing takes more time at the later stage than the transient phase at the very beginning. Unlike asymptotic convergence, terminal attractors violate the Lipschitz condition at equilibrium points so that the solution can reach these points in finite time. This is the fascinating point of terminal attractors.

![Figure 3.1 Convergence to regular attractor](image-url)
The concept of terminal attractors is illustrated in the following example. Consider the differential equation

\[ \dot{x} = -x^\rho, \quad \rho > 0 \]  

(3.1)

Obviously \( x = 0 \) is the equilibrium point and the Lipschitz condition is

\[ \frac{d\dot{x}}{dx} = -\rho x^{\rho - 1} = \begin{cases} 0 & \rho > 1 \\ -1 & \rho = 1 \\ -\infty & 0 < \rho < 1 \end{cases} \quad \text{as } x \to 0 \]  

(3.2)
Notice that how the Lipschitz condition is violated for $0 < \rho < 1$ at $x = 0$ and the real part of eigenvalue of matrix $\frac{dx}{dx}$ at $x = 0$ is negative (in this case $\frac{dx}{dx}$ is a scalar and its eigenvalue is $-\infty$), so the equilibrium point $x = 0$ is stable in this case. Solving (3.2) directly, we get

$$x^{\rho} = (1 - \rho)(g-t)$$

(3.3)

where $g$ is a constant to be determined by initial conditions. Obviously, if $0 < \rho < 1$ the relaxation time of the solution $x$ will reach the equilibrium point $x = 0$ is finite from any positive initial condition, otherwise the transient will take infinite time. The trajectory will stay at $x = 0$ because the derivative at $x = 0$ is zero.

### 3.3 Terminal Attractor Based Backpropagation Algorithms

#### 3.3.1 An Overview

The terminal attractor has been applied to single layer and multilayer NNs [Zak 1989] by using the Lagrange multiplier technique. These approaches update the connection weights indirectly since the Lagrange multipliers should also be solved simultaneously. The TABP learning algorithm lies in its ‘tailored time varying’ learning rate [Wang and Hsu 1991, Bianchini et al 1997] as:

$$\gamma = \zeta \frac{\Omega(J)}{\|\nabla w J\|^2}, \quad \zeta > 0,$$

(3.4)

where $\Omega(J)$ is a non-negative continuous function of $J, \nabla_w J$ the gradient vector of weights. The dynamics of error function becomes
\[
\frac{dJ}{dt} = (\nabla_w J)^T \frac{dw}{dt} = (\nabla_w J)^T \left( -\frac{\varsigma \Omega(J)}{\|\nabla_w J\|^2} \nabla_w J \right) = -\varsigma \Omega(J) \quad (3.5)
\]

In the TABP algorithm [Wang and Hsu 1991, Bianchini et al 1997], \( \Omega(J) = J^\rho \) \((0 < \rho < 1)\), which leads to

\[
\frac{dJ}{dt} = -\varsigma J^\rho. \quad (3.6)
\]

A simple calculation of (3.6) leads to that \( J \) reaches zero in a finite-time determined by

\[
t_r = \frac{1}{\varsigma (1 - \rho)} J^{1 - \rho} (0). \]

Clearly, this is an attractive characteristic of TAPB learning algorithms as it indicates that the exact time taken to reach zero is finite-time and tuneable (by choosing proper \( \varsigma \) and \( \rho \), as pointed out in [Wang and Hsu 1991]. The error function \( J \) decreases to zero in finite-time.

This characteristic is depicted by (3.1) where \( J \) tends to be a nonzero value, that is a local minimum, then \( \nabla_w J \to 0 \), from

\[
\left| \frac{dw}{dt} \right| = \frac{|\Omega(J)|}{\|\nabla_w J\|} \quad (3.7)
\]

one can see that the variation ratio \( dw/dt \) tends to infinity, resulting in a disruptive change to the weights in \( w \). This change may move the weights out of the neighbourhood of the local minimum region toward the global minimum.
However, this assertion was later proved untrue [Bianchini et al 1997] as such a disruptive change does not necessarily result in the convergence to the global minimum. In [Bianchini et al 1997], a further explanation was given to demonstrate that jumps might occur even in the neighbourhood of the global minimum, with the simplification of $J$ in the form as $J(w) = w^{2m}$, $m>0$. However, apart from the research in [Wang and Hsu 1991, Bianchini et al 1997], there has not been any further research conducted to investigate the dynamical characteristics of this class of learning algorithm, in particular their characteristics near the global minimum.

### 3.3.2 TABP Learning in the Neighbourhood of the Global Minimum

To study the dynamic characteristics of the TABP learning algorithms in the neighbourhood of the global minimum, first, the error function is represented by its Taylor series expansion about $w^*$, which corresponds to the global minimum, as

$$ J(w) = J(w^*) + G(w^*)\Delta w + \frac{1}{2} \Delta w^T H(w^*)\Delta w + \cdots, \quad (3.8) $$

where $\Delta w = w - w^*$, $G(w^*) = \nabla_w J \big|_{w=w^*}$, $H(w^*) = \nabla^2_w J \big|_{w=w^*}$. If $\|\Delta w\|$ is very small, the higher order part of (3.8) can be neglected and the only important terms of interest are the first and second order terms. If $w^*$ is the global minimum, which means $J(w^*) = 0$ and $G(w^*) = 0$, then (3.8) becomes

$$ \Delta J(w) = J(w) - J(w^*) = \frac{1}{2} \Delta w^T H(w^*)\Delta w \quad (3.9) $$

where $\Delta J(w) = J(w) - J(w^*)$. From (3.8) and (3.9), it can be concluded that
\[ \nabla_w J = \frac{1}{2} H(w^*) \Delta w \]  
(3.10)

If \( \Omega(J) = J' \) is set \((0 < \rho < 1)\), substituting (3.10) and (3.9) into (3.7) yields

\[ \left\| \frac{dw}{dt} \right\| = \frac{((1/2)\Delta w^T H(w^*) \Delta w)^\rho}{\| H(w^*) \Delta w \|} \]  
(3.11)

Because \( w^* \) is the global minimum, \( H(w^*) \) must be positive semi-definite [Hagan and Menhaj 1994]. To better understand (3.11), the following transformation is performed for \( H(w^*) \) [Hagan and Menhaj 1994]

\[ H(w^*) = B\Lambda B^T \]  
(3.12)

where matrix \( \Lambda \) is a diagonal matrix with all eigenvalues of \( H(w^*) \) on the diagonal and \( B \) is an orthogonal matrix. Note that since \( B \) is orthogonal, then \( B^{-1} = B^T \). Now let \( \Delta w = Bc \) where \( c \) is a \( n \)-dimensional constant vector. Equation (3.11) can be rewritten as

\[ \left\| \frac{dw}{dt} \right\| = \frac{((1/2)c^T B^T B\Lambda B^T Bc)^\rho}{[(B\Lambda B^T Bc)^T (B\Lambda B^T Bc)]^{1/2}} = \frac{(1/2)^\rho (c^T \Lambda c)^\rho}{(c^T \Lambda^2 c)^{1/2}} = \frac{1}{2^\rho} \left( \sum_{i=1}^n (\sum c_i^2)^{1/2} \right)^\rho \]  
(3.13)

where \( \lambda_i \) \((i = 1, \ldots, n)\), are the eigenvalues of \( H(w^*) \) and \( n \) (which may be less than the rank of matrix \( H(w^*) \)) is the total number of the nonzero eigenvalues of \( H(w^*) \). The following relation is then easily derived from (3.13)

\[ \left\| \frac{dw}{dt} \right\| \leq \frac{1}{2^\rho} \frac{\lambda_{\max}^\rho}{\lambda_{\min}^{1/2}} \left( \sum_{i=1}^n c_i^2 \right)^{1/2} \]  
(3.14)
where $\lambda_{\text{max}}$ is the largest of $\lambda_i (i=1,\ldots,n)$, and $\lambda_{\text{min}}$ is the smallest nonzero $\lambda_i (i=1,\ldots,n)$.

Since
\[
\| c \| \| B^{-1} \Delta w \| \leq \| B \| \| \Delta w \| \| \Delta w \| \tag{3.15}
\]

Inequality (3.14) can be rewritten as
\[
\left\| \frac{dw}{dt} \right\| \leq \frac{1}{2^\rho} \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}} \| \Delta w \|^{\rho - \frac{1}{2}} . \tag{3.16}
\]

It is necessary that for
\[
\rho > \frac{1}{2}
\]

we have
\[
\lim_{w \to w^*} \left\| \frac{dw}{dt} \right\| = 0.
\]

This means that $\left\| \frac{dw}{dt} \right\|$ will decrease to zero asymptotically when $w$ approaches the global minimum $w^*$, hence there will be no explosive behaviour (i.e. singularity) of $\left\| \frac{dw}{dt} \right\|$ occurring in the neighbourhood of the global minimum. Therefore, $w$ will approach $w^*$ asymptotically and $J$ will converge to zero in finite-time, as determined by the property of the terminal-attractor.

### 3.4 Fast Terminal Sliding Mode Concept

Sliding mode control systems exhibit robustness and simplicity in design and applications, and have been widely used. In Control Theory, sliding mode control is a type of variable structure control where the dynamics of a nonlinear system is altered via application of a high-frequency switching control. This is a state feedback control scheme where the feedback is not a continuous function of time. The sliding mode is attained by designing the control laws which drive the system to reach and remain on
the intersection of a set of prescribed switching manifolds commonly selected as asymptotical stable linear switching hyperplanes.

However, for high-precision control, the asymptotical stability may not deliver a fast convergence without imposing strong control force. Nonlinear switching manifolds such as the Terminal Sliding Modes (TSMs), can improve the transient performance substantially. TSM is very popular in control theory but in comparison with the sliding-mode control based on linear switching hyperplanes, the existing TSM control may not deliver the same convergence performance when the system state is far away from the equilibrium, albeit its finite time convergence lies in its exponentially growing convergence rate when the state is near the equilibrium. The TSM concept, which is equivalent to the terminal attractor concept, can be formulated as [Man et al 1994]

\[ s = \dot{z} + \beta z^{q/p} = 0 \]  

(3.17)

where \( z \in R^1 \) is a scalar variable, and \( \beta > 0 \) and \( p, q \) \((p > q)\) are positive integers. Note that the parameters \( p \) and \( q \) must be odd integers and only the real solution is considered so that, for any real number \( z, z^{q/p} \) is always a real number. It can be easily verified that, given any initial state \( z(0) \neq 0 \), the dynamics (3.17) will reach \( z = 0 \) in a finite time determined by

\[ t^* = \left( \frac{p}{\beta(p - q)} \right) \|z(0)\|^{\frac{p-q}{p}} \]  

(3.18)

The equilibrium 0 is a terminal attractor [Zak 1989], i.e. the state \( z = 0 \) can be reached in a finite time and it is stable. The term ‘terminal’ is referred to the equilibrium which can be reached in finite time and is stable. The reaching time \( t^* \) can be tuned by setting parameters \( p, q, \) and \( \beta \). The introduction of the nonlinearity term \( z^{q/p} \) improves the
convergence toward equilibrium. The closer to equilibrium, the faster the convergence rate which then results in finite time convergence. Note that although the terminal dynamics is not Liptchitz, for any nonzero initial condition, the solution in the forward time direction is unique.

It should also be noted that there is a close relationship between the TSM (3.13) and the time optimal control. In fact, the time optimal control for the double integrator system [Yu and Man 2002] can be approximated by a TSM model. When the system state is far away from the equilibrium, the TSM model (3.18) does not prevail over the linear counterpart (setting $p = q$) since the term $z^{q/p}$ tends to reduce the magnitude of the convergence rate at a distance from equilibrium. One immediate solution is to introduce the following so-called Fast Terminal Sliding Mode (FTSM) model:

$$s = \dot{z} + \alpha z + \beta z^p$$  \hspace{1cm} (3.19)

where $\alpha, \beta > 0$, $\rho = q/p$. By doing so, we have $\dot{z} = -\alpha z - \beta z^{q/p}$. For correctly chosen $q$ and $p$, given an initial state $z(0) \neq 0$, the dynamics (3.19) will reach $z = 0$ in finite time. The physical interpretation is: when $z$ is far away from zero, the approximate dynamics become $\dot{z} = -\alpha z$ whose fast convergence when far away from zero is well understood. When close to $z = 0$, the approximate dynamics become $\dot{z} = -\beta z^{q/p}$ which is a terminal attractor [Zak 1989]. More precisely, we can solve the differential equation (3.19) analytically. The exact time to reach zero, $t^*$, is determined by

$$t^* = \frac{p}{\alpha(p-q)} \left( \ln(\alpha z(0)^{(p-q)/p} + \beta) - \ln \beta \right)$$  \hspace{1cm} (3.20)

and the equilibrium $0$ is a terminal attractor.
3.5 The Fast Terminal Attractor Based Backpropagation Algorithm

As mentioned before, the most serious gap of the GDBP algorithm is a very slow convergence and it can be easily trapped into local minimum. To improve the learning performance, the TABP learning algorithm has been proposed [Jiang and Yu 2002], which is

\[ \frac{dw}{dt} = - \frac{\nabla J}{\parallel \nabla J \parallel^2} (\nabla J) \]  

(3.21)

where \( \nabla J \) is the gradient vector for weights, \( J(w) \) the error function, \((0 < \rho < 1)\). It is seen that the effectiveness of the TABP algorithm lies in the introduction of an equivalent time-varying learning rate which increases when the actual weights are close to the ideal weights. However, when the initial weights are far away from the ideal weights, the convergence is shown to be slower than the conventional GDBP algorithm.

From Section 3.4, it is known that when the system state is very far away from the equilibrium, the dynamic equation (3.19) is dominated by the linear term and can be approximate by \( \dot{z} = -\alpha z \). When the system state is close to the equilibrium, the dynamic equation (3.19) is dominated by the conventional terminal attractor (3.22). When (3.18) becomes zero

\[ \dot{z} = -\alpha z - \beta z^\rho \]  

(3.22)

which is the desired fast terminal attractor we wish to have. If we use the idea of (3.22) in the FNN weights learning, then we have
\[
\frac{dw}{dt} = -\gamma \nabla_w J - \frac{\zeta J^\rho}{\| w \nabla J \|^2} (w J),
\] (3.23)

which is called the Fast Terminal Attractor Based Backpropagation (FTABP) learning algorithm. Then dynamics of the error function is described as follows

\[
\frac{dJ}{dt} = (\nabla_w J)^T \frac{dw}{dt} = - (\nabla_w J)^T (\gamma \nabla_w J) - (\nabla_w J)^T \frac{\zeta J^\rho}{\| w \nabla J \|^2} (w J)
\]

\[
= - \gamma (\nabla_w J)^T \nabla_w J - \zeta J^\rho
\] (3.24)

which leads to

\[
\frac{dJ}{dt} = -\gamma \| \nabla_w J \|^2 - \zeta J^\rho
\] (3.25)

Since \( J > 0 \), then we have

\[
\frac{dJ}{dt} = -\gamma \| \nabla_w J \|^2 - \zeta J^\rho \leq -\zeta J^\rho
\] (3.26)

which means the FTABP algorithm enables the weights to converge in a time which should be much less than \( t_r = \frac{1}{\zeta (1-\rho)} J^{(1-\rho)}(0) \). When the weights are far away from the ideal weights, the conventional GDBP algorithm will play an important role in the convergence. When the weights are close to the ideal weights, the TAP algorithm will dominate the weight updating with faster convergence. This combination takes advantage of both the GDBP algorithm and the TAP algorithm for faster learning in FNNs.
3.6 Terminal Attractor Based Levenberg-Marquardt Backpropagation Algorithm

Recently, the general type of BP algorithm was proposed [Yu et al 2002], which is

\[
\frac{dw}{dt} = \left\{ -\left( \mu I + \sigma \frac{\partial^2 J}{\partial w \partial w^T} \right) \right\}^{-1} \left( \mu \frac{\partial J}{\partial t} + \sigma \frac{\partial J}{\partial w} \frac{\partial^2 J}{\partial \partial w^T} + \frac{\partial J}{\partial w} \right) + \eta J \tag{3.27} \]

where the parameters \( \mu, \sigma > 0 \) determine the relative importance of each term. This formula is able to interpret many existing BP algorithms. For example the conventional BP algorithm can be obtained by setting \( \sigma = 0 \) and \( \eta = 0 \), which yields

\[
\dot{w} = -\mu^{-1} \frac{\partial J}{\partial w^T} = -\gamma \frac{\partial J}{\partial w^T} \tag{3.28} \]

The LMBP algorithms can be obtained by setting \( \eta = 0 \), and since \( \frac{\partial J}{\partial t} = 0 \) and \( \frac{\partial^2 J}{\partial t \partial w^T} = 0 \)

\[
\dot{w} = -\left( \mu I + \sigma \frac{\partial^2 J}{\partial w \partial w^T} \right)^{-1} \left( \sigma \frac{\partial J}{\partial w^T} \right) \tag{3.29} \]

If we use the general BP algorithm with the LMBP algorithm (3.29) into the FTSM (3.19), then we have

\[
\dot{w} = -\left( \mu I + \sigma \frac{\partial^2 J}{\partial w \partial w^T} \right)^{-1} \left( \frac{\partial J}{\partial w^T} \right) - \frac{\partial J}{\partial w^T} \tag{3.30} \]

The weight update law for FNNs then becomes

\[
\frac{dw}{dt} = -\gamma (J^\rho)\nabla_w J - \left[ M^T M + \mu I \right]^{-1} \nabla_w J \tag{3.31} \]
where $M$ is the Jacobian matrix, $I$ an identity matrix. The algorithm (3.31) called the Terminal Attractor based LMBP algorithm. To analyse the dynamics of the error function, we have

$$\frac{dJ}{dt} = (\nabla_w J)^T \frac{dw}{dt} = -(\nabla_w J)^T \left( [M^T M + \mu I]^{-1} \nabla_w J \right) - (\nabla_w J)^T \frac{\varphi J^p}{\| \nabla_w J \|^2} (\nabla_w J)$$

$$= -(\nabla_w J)^T [M^T M + \mu I]^{-1} \nabla_w J - \varphi J^p$$  \hspace{1cm} (3.32)

Since $J > 0$, and according to the properties of symmetric matrices [Dickinson 1996], matrix $[M^T M + \mu I]$ is nonnegative definite, hence its inverse is nonnegative definite, therefore we have

$$\frac{dJ}{dt} = -(\nabla_w J)^T [M^T M + \mu I]^{-1} \nabla_w J - \varphi J^p \leq -\varphi J^p$$  \hspace{1cm} (3.33)

From (3.33) it can be seen that the new improved terminator attractor based LMBP algorithm can converge in a time which should be much less than the conventional LMBP algorithm. The LMBP algorithm which is the first part of the new algorithm (3.31) will dominate when the weights are far away from the ideal weights. The second part with terminal attractors will converge much faster than the LMBP algorithm when the weights are close to ideal weights. As a result of this combination, the new improved terminal attractor based LMBP algorithm would be much faster than the conventional LMBP and other BP algorithms for FNNs, because it has advantages of both the conventional LMBP and the FTSM based algorithms.
3.7 Simulation Results

3.7.1 Simulation Results Using TABP Learning Algorithm

In this section, simulation studies are presented which demonstrate the performance of the TABP algorithm in comparison with the GDBP and Gradient Descent with Momentum (MBP) learning algorithms. The weights update law for the GDBP algorithm in discrete-time is

\[ w(i+1) = w(i) - \gamma \nabla_w J \]  

(3.34)

MBP algorithm in discrete-time is

\[ w(i+1) = w(i) - \lambda \Delta w(i-1) - (1-\lambda) \gamma \nabla_w J \]  

(3.35)

where \( \lambda = 0.8 \) and the TABP algorithm in discrete-time is

\[ w(i+1) = w(i) - \gamma \left( \frac{J^\rho}{\| \nabla_w J \|^2} \right) \nabla_w J \]  

(3.36)

where \( \gamma > 0, \ 0 < \rho < 1 \). A single hidden-layer FNN with two inputs, 15 hidden neurons and one output is used. Initial values of weights of input and hidden and output layers are randomly selected between -1 to +1. The target output function to be approximated is:

\[ y_d(x_1, x_2) = 0.5 + 10^8 \left[ \frac{\sin[0.5^*(x_1 + 5)^2 + (x_2 - 5)^2]}{(x_1 + 5)^2 + (x_2 - 5)^2} \right], \quad x_1, x_2 \in [0, 2] \]  

(3.37)

which is meshed with stepsize 0.1 (Figure 3.3)
The FNN was trained using the GDBP, MBP and TABP algorithms respectively with 40,000 iterations. Figure 3.4 shows the simulation results with the same learning rate (for the TABP, \( \rho = 0.6 \), which satisfies the necessary condition (3.16)).

In the next investigation, all algorithms are simulated using best learning rates. After several test simulations, we determined the best learning rates for each of the algorithms as \( \gamma_{GDBP} = 0.01 \), \( \gamma_{MBP} = 0.002 \), \( \gamma_{TABP} = 0.0008 \). This investigation also proved that the TABP had the best performance. (See Figure 3.5)
The next task is to study the TABP algorithm with different $\rho$s. Figure 3.6 compares the learning performances of the TABP algorithm with $\rho=0.6$ and $\rho=0.4$ respectively. It can be seen that if the necessary condition (3.16) is not satisfied, the convergence will suffer as shown by the spikes in Figure 3.6(b).

The simulation is for the specific investigation of the TABP learning algorithms. In this study, different learning rates, and $\rho$ parameter were used. Table 3.1 illustrates the simulation result. From this table the best parameters of TABP algorithms for this task
can be found. The table shows that the network does not perform well with learning rate which is too big or too small. Also the experiments show that the proper value of $p$ is 0.6.

<table>
<thead>
<tr>
<th>Learning Performance</th>
<th>$\gamma=0.0003$</th>
<th>$\gamma=0.0009$</th>
<th>$\gamma=0.003$</th>
<th>$\gamma=0.009$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho=0.5$ Mean</td>
<td>0.22</td>
<td>0.18</td>
<td>0.21</td>
<td>1.06</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.47</td>
<td>0.42</td>
<td>0.46</td>
<td>1.02</td>
</tr>
<tr>
<td>$\rho=0.6$ Mean</td>
<td>0.14</td>
<td>0.13</td>
<td>0.15</td>
<td>0.16</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.37</td>
<td>0.36</td>
<td>0.39</td>
<td>0.4</td>
</tr>
<tr>
<td>$\rho=0.7$ Mean</td>
<td>0.18</td>
<td>0.18</td>
<td>0.17</td>
<td>0.24</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.42</td>
<td>0.42</td>
<td>0.4</td>
<td>0.49</td>
</tr>
</tbody>
</table>

Table 3.1 Performance comparison of TABP algorithms with three different $\rho$s

### 3.7.2 Simulation Results Using FTABP Algorithm

**Simulation 1**

A number of computer simulations have been conducted, in order to compare the performances of the FTABP, TABP and GDBP algorithms for solving a function approximation problem. The weights updating rule for the TABP algorithm is given by

$$w(i+1) = w(i) - \gamma (\frac{J^\rho}{\| \nabla_w J \|^2}) \nabla_w J$$  \hspace{1cm} (3.38)

and for the FTABP algorithm, it is given by

$$w(i+1) = w(i) - \gamma (\frac{J^\rho}{\| \nabla_w J \|^2}) \nabla_w J - \gamma_2 \nabla_w J$$  \hspace{1cm} (3.39)
and the GDBP algorithm is described as

\[
\frac{dw}{dt} = -\gamma \nabla w J \tag{3.40}
\]

The previous function approximation problem is used here again, which is:

\[
y_d(x_1, x_2) = 0.5 + 10^* \left[ \frac{\sin[0.5^* (x_1 + 5)^2 + (x_2 - 5)^2]}{(x_1 + 5)^2 + (x_2 - 5)^2} \right] \tag{3.41}
\]

In this simulation, A FNN with 2 inputs, 15 hidden neurons and one output was used. Initial values of weights of input, hidden and output layers were randomly initiated varying from -1 to +1. The parameter \( \rho \) was selected to be 0.6.

First, the learning efficiencies of the FTABP, GDBP and TABP algorithms are compared. The FNNs were trained to approximate the function in (3.41) with best learning rates. Figure 3.7 shows the FNN output with FTABP algorithm after 30,000 iterations. Figures 3.8 and 3.9 illustrate the FNN outputs using the TABP and GDBP algorithms respectively with different iterations to achieve similar results.

Figure 3.7 FTABP output after 30,000 iterations with \( \gamma = 0.0008 \)
Figure 3.10 illustrates the comparison of learning performances of the FNNs using the FTABP, GDBP and TABP algorithms respectively within the first 30,000 iterations. It can be seen that the FTABP algorithm is faster than others with less learning error.

Figure 3.8 TABP output after 100,000 iterations with $\gamma = 0.003$

Figure 3.9 GDBP output after 200,000 iterations with $\gamma = 0.01$

Figure 3.10 Performance comparison of the FTABP, TAPB and GDBP algorithms

Figure 3.10 illustrates the comparison of learning performances of the FNNs using the FTABP, GDBP and TABP algorithms respectively within the first 30,000 iterations. It can be seen that the FTABP algorithm is faster than others with less learning error.
Next, we consider how the learning rates affect the learning efficiency of the FTABP algorithm. Table 3.2 shows how learning rates affect the learning speed.

<table>
<thead>
<tr>
<th>$\gamma_1$</th>
<th>0.0003</th>
<th>0.0009</th>
<th>0.003</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma_2$</td>
<td>0.681</td>
<td>0.387</td>
<td>0.265</td>
</tr>
<tr>
<td>$\gamma_3$</td>
<td>0.536</td>
<td>0.267</td>
<td>0.178</td>
</tr>
<tr>
<td>$\gamma_4$</td>
<td>0.344</td>
<td>0.14</td>
<td>0.091</td>
</tr>
</tbody>
</table>

Table 3.2 $J$ values for different learning rates

Simulation 2

The second problem is related to sensor networks [Ozdemir et al 2005]. In this case, sensor data values in with 1-5-1 FNNs are simulated. The TABP, FTABP and other popular BP algorithms were trained respectively using the function approximation problem with less data described in Figure 3.11. Note the output images of each learning algorithms in Figures 3.12 and 3.13.
Figure 3.11 Function to be approximated
(output of sensor 22)

Figure 3.12 NN outputs with GDBP, TABP
and FTABP algorithms

Figure 3.13 NN outputs with two other algorithms

Figure 3.14 Learning performances of selected algorithms
Figure 3.14 shows the performances of the selected learning algorithms. Now it can be seen that the FTABP algorithm performs very well for the sensor networks problem. After 12,000 iterations, the FNN output with the FTABP algorithm is almost the same as the target output. This simulation result describes how the new FTABP algorithm can play an important role in sensor network applications.

In the next simulation, most popular BP algorithms are used for comparing with the TABP algorithm. These algorithms come from Matlab’s Neural Network Toolbox. The previous tasks were trained with the same iteration numbers. The statistical t-test method was used. The simulations showed that several popular learning algorithms are faster than the FTABP algorithm but only for small sized training data. The FTBP algorithm performed equally well.
<table>
<thead>
<tr>
<th>Name of Function</th>
<th>Learning Performance</th>
<th>T-Test statistics</th>
<th>General description from MATLAB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Levenberg-Marquardt BP</td>
<td>7.78E-06</td>
<td>0.214</td>
<td>Fastest training algorithm for networks of moderate size. Has memory reduction feature for use when the training set is large.</td>
</tr>
<tr>
<td>2 Conjugate gradient BP with Powell-Beale restarts</td>
<td>4.91E-06</td>
<td>0.345</td>
<td>Generally faster convergence.</td>
</tr>
<tr>
<td>3 BFGS Quasi-Newton BP</td>
<td>5.95E-06</td>
<td>0.612</td>
<td>Requires storage of approximate Hessian matrix and has more computation in each iteration than conjugate gradient algorithms, but usually converges in less iteration.</td>
</tr>
<tr>
<td>4 Bayesian Regularization BP</td>
<td>2.71E-05</td>
<td>0.451</td>
<td>Modification of the Levenberg-Marquardt training algorithm to produce networks that generalizes well. Reduces the difficulty of determining the optimum network architecture.</td>
</tr>
<tr>
<td>5 One Step Secant BP</td>
<td>3.00E-05</td>
<td>0.021</td>
<td>Compromise between conjugate gradient methods and quasi-Newton methods.</td>
</tr>
<tr>
<td>6 Fletcher-Reeves Conjugate Gradient Algorithm</td>
<td>3.56E-05</td>
<td>0.145</td>
<td>Has smallest storage requirements of the conjugate gradient algorithms.</td>
</tr>
<tr>
<td>7 FTABP</td>
<td>4.39E-05</td>
<td>0.246</td>
<td></td>
</tr>
<tr>
<td>8 Scaled Conjugate Gradient Algorithm</td>
<td>6.08E-05</td>
<td>0.154</td>
<td>The only conjugate gradient algorithm that requires no line search. A very good general purpose training algorithm.</td>
</tr>
<tr>
<td>9 Modified BP</td>
<td>2.03E-04</td>
<td>0.652</td>
<td></td>
</tr>
<tr>
<td>10 TABP</td>
<td>0.0002</td>
<td>0.145</td>
<td>With fast convergence and minimal storage requirements.</td>
</tr>
<tr>
<td>11 Resilient BP</td>
<td>0.0004</td>
<td>0.214</td>
<td>Adaptive learning rate. Faster training than GDBP.</td>
</tr>
<tr>
<td>12 BPALM</td>
<td>0.0004</td>
<td>0.41</td>
<td>Gradient descent with momentum. Generally faster than GDBP. Can be used in incremental mode training.</td>
</tr>
<tr>
<td>13 GDBP with Momentum</td>
<td>0.0288</td>
<td>0.2</td>
<td>Basic gradient descent. Slow response; can be used in incremental mode training.</td>
</tr>
<tr>
<td>14 GDBP</td>
<td>0.0302</td>
<td>0.214</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3 Results of all algorithms
Figure 3.15 shows the over training issue of the FTABP learning algorithm. This picture illustrates the over training can decrease the learning efficiency of the training algorithm, which is consistent with the conclusion with other learning algorithms.

![Figure 3.15 Learning over performance](image)

3.7.2.3 Simulation 3

The previous simulations have a small number of input data and desired output data. It is not enough to describe the learning performance of training algorithms. So in this task a larger problem was selected. This simulation used 1,000x150 input data, 150x150 target output data which were selected randomly. The FNN has 1,000 inputs, 50 hidden and 150 output neurons respectively. Simulation results are shown in the next table. Here the stopping criterion is the number of iterations. Unfortunately Table 3.4 shows that some algorithms stopped before reaching maximum iterations.
<table>
<thead>
<tr>
<th>Name of Function</th>
<th>Learning Performance</th>
<th>Short Description or Training Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Levenberg-Marquardt BP</td>
<td>No iteration</td>
<td>Maximum variable size allowed by the program is exceeded.</td>
</tr>
<tr>
<td>2 Conjugate Gradient BP with</td>
<td>0.059</td>
<td>32 min</td>
</tr>
<tr>
<td>Powell-Beale Restarts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 BFGS Quasi-Newton BP</td>
<td>No iteration</td>
<td>Maximum variable size allowed by the program is exceeded.</td>
</tr>
<tr>
<td>4 Bayesian Regularization BP</td>
<td>No iteration</td>
<td>Maximum variable size allowed by the program is exceeded.</td>
</tr>
<tr>
<td>5 One Step Secant BP</td>
<td>0.068</td>
<td>24 min</td>
</tr>
<tr>
<td>6 Fletcher-Reeves Conjugate</td>
<td>0.078</td>
<td>28 min</td>
</tr>
<tr>
<td>Gradient Algorithm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 FTABP</td>
<td>0.071</td>
<td>15 min</td>
</tr>
<tr>
<td>8 Scaled Conjugate Gradient</td>
<td>0.042</td>
<td>26 min</td>
</tr>
<tr>
<td>Algorithm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 Modified BP</td>
<td>0.408</td>
<td>16 min</td>
</tr>
<tr>
<td>10 TABP</td>
<td>0.791</td>
<td>14 min</td>
</tr>
<tr>
<td>11 Resilient BP</td>
<td>0.131</td>
<td>16 min</td>
</tr>
<tr>
<td>12 BPALM</td>
<td>After 3519 iterations</td>
<td>Minimum gradient reached, performance goal was not met.</td>
</tr>
<tr>
<td>program stopped.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 GDBP with Momentum</td>
<td>0.643</td>
<td>14 min</td>
</tr>
<tr>
<td>14 GDBP</td>
<td>0.643</td>
<td>12 min</td>
</tr>
</tbody>
</table>

Table 3.4 Results of all algorithms

The simulation results demonstrate the effectiveness of the new FTABP learning algorithm. It is shown that the most popular, fast learning algorithms either did not work or converged very slowly. The others did work but their convergence speed was not slower than the FTABP algorithm.
3.7.3 Simulation Using the Terminal Attractor Based Levenberg-Marquardt Algorithm

In this section several experiments of learning efficiency for FNNs are carried. A number of simulations have been done to compare the performances of the LMBP algorithm and the terminal attractor based LMBP algorithm for solving a function approximation problem. The LMBP algorithm is

$$w(i+1) = w(i) - \left[ M^T M + \mu I \right]^{-1} \nabla_w J$$  \hspace{1cm} (3.42)

The weights update formula for the terminal attractor based improved LMBP algorithm is

$$w(i+1) = w(i) - \gamma \left( \frac{J}{\| \nabla_w J \|^2} \right) \nabla_w J - \left[ M^T M + \mu I \right]^{-1} \nabla_w J$$  \hspace{1cm} (3.43)

The target output function for this simulation is described in Figure 3.16. The function approximation problem is

$$y_d(x) = 100 \cos(102x) + 102 \sin(100x)$$  \hspace{1cm} (3.44)

A noise is also added to the desired signal. For this function problem, the input is sampled from 0 to 3.14 with stepsize 0.1.

![Figure 3.16 Function to be approximated](image-url)
In this simulation, a $1 \times 15 \times 1$ single hidden layer FNN was used with log sigmoid and linear transfer functions in hidden and output layers respectively. Weights and biases of input, hidden and output layers were initiated randomly varying from -1 to +1. The parameter $\rho$ is selected to be $0.6$, and $\gamma$ for the terminal attractor is $0.002$ for this simulation. The terminate condition for main loop of our program is that the error value is lower than $0.003$.

![Figure 3.17 NN output with the improved LMBP algorithm after 4952 iterations](image1)

**Figure 3.17 NN output with the improved LMBP algorithm after 4952 iterations**

![Figure 3.18 NN output with LMBP algorithm after 10,000 iterations](image2)

**Figure 3.18 NN output with LMBP algorithm after 10,000 iterations**

The learning efficiencies of the LMBP algorithm and our improved LMBP algorithm are compared here. (See results in Figures 3.17, 3.18, 3.19). After 4,952 iterations, the
program terminated due to the error of the improved LMBP algorithm reached 4.9e-4. For the conventional LMBP algorithm, the error was 0.001 after 10,000 iterations. This investigation was repeated a few times with different initial weights and biases; but there is main difference between final training results. The investigation results demonstrate that the new terminal attractor based LMBP algorithm is much more effective than the LMBP and other algorithms. Figure 3.20 shows an enlargement part of the figure to show the performances of the terminal attractor based improved LMBP algorithm and the conventional LMBP algorithm. One can see clearly the advantage of our new algorithm.

Figure 3.19 Performance comparison of LMBP and, the improved LMBP algorithms

Figure 3.20 Performance comparison (enlargement)
3.8 Summary

This chapter has investigated the idea of a fundamental terminal attractor idea as well as a TABP learning algorithm and the dynamic characteristics of the TABP algorithm for FNN learning. The cause of the reported ‘disruptive behaviours’, which is the peculiar singularity problem, has been revealed and a necessary condition has been obtained, in order to avoid the singularity problem.

To speed up the learning performance the FTABP learning algorithm has been proposed for FNNs. It is based on the fast terminal sliding mode concept so that fast transient convergence both at a distance from and at a close range of the equilibrium can be obtained. Based on this, a new FTABP learning algorithm which combined advantages of both GDBP and TABP algorithms has been proposed. This chapter has also proposed an improved LMBP algorithm based on the terminal attractor. The proposed new algorithm has been shown to be faster than conventional LMBP and TABP algorithms. The FTABP training algorithm did not converge faster than LMBP algorithm but the improved LMBP algorithm showed better performance than the LMBP algorithm. Comparative simulation studies have been presented to show the effectiveness of our new algorithms.
Chapter 4

Applications

4.1 Introduction

Neural networks have been successfully applied in solving many practical problems. In this chapter, some practical applications are addressed with our proposed, new and basic, conventional BP algorithms. Firstly, a short description of the latest practical applications is given. Then the problems of stock market prediction, optical character recognition and image interpolation are addressed using the FNN with the proposed learning algorithms.

This chapter is organized as follows. Section 2 proposes a time series forecaster. The next section is about optical character recognition. Section 4 presents the image interpolation. Section 5 is a summary of this chapter.
4.2 Time Series Forecasting with Neural Networks

4.2.1 Overview

Forecasting is the art of predicting the future. Such activities are critical in many areas including macroeconomics, finance, production and facilities planning, sales and inventory control. In general, a forecast is often required whenever a decision is to be made regarding an uncertain future. To handle the increasing variety and complexity of forecasting problems, many new techniques have been developed. Each has its special use, and care must be taken to select the correct technique for a particular application.

The selection of a method depends on many factors including: the context of the forecast, the degree of accuracy desired, the relevance and availability of historical data, the time period to be forecast, and the analyst’s time available for performing the analysis. For example, Figure 4.1 displays a selection tree [Armstrong 2001] for forecasting methods split by data availability and type, depth of knowledge of relationships, and purpose. Since the domain of nonlinear models is extensive, experiments frequently use linear models as a first-order approximation of the more complex, but unknown nonlinear models. Another option is to consider a theoretical, but flexible class of statistical models such as NNs.
Forecasting with NNs have been widely studied and applied to a variety of areas. Although there have been many encouraging reports, Lapedes and Farber (1987) reported that simple NNs can outperform conventional methods, sometimes by orders of magnitude. Their conclusions were based on two specific time series without noise.
Sharda and Patil (1990) conducted a forecasting competition between the NN model and a traditional forecasting technique (namely the Box-Jenkins method) using 75 times series of various nature. Cheng and Titterington (1994) and Ripley (1993) provided reviews of NNs from a statistical perspective. Moody (1995) suggested that NNs were superior to linear and econometric models for macroeconomic forecasting. Swanson and White (1997) compared real time forecasts from nine macroeconomics variables using various adaptive and nonadaptive, linear and nonlinear models. NNs were also applied in financial forecasting. Campbell, Lo and MacKinlay (1997) explained that the econometric methods typically taught and used in practice could be designed to detect liner structure and financial data. They developed tests for asset returns predictability based on weighted combinations of return autocorrelations.

Lin, Yu, Gregor and Irons (1995) proposed a scheme for time series forecasting with a FNN. Their work has three parts: detection of input patterns, determination of the number of neurons in hidden layer, and construction of NN forecaster. In this section, the method in [Lin et al 1995] is simulated with the new terminal attractor based BP learning algorithms. The work has the following phases. The first phase is the detecting phase. In this part the autocorrelation analysis is used to identify input patterns of data. Then the number of hidden neurons is determined using the Baum-Haussler rules. The calculated number of neurons for the hidden layers and the determined input patterns are then used to construct the NN forecaster. The simulation part was done using the traditional GDBP, TABP and the proposed new FTABP learning algorithms. Figure 4.2 shows the short term and long term forecasting NN diagrams.
4.2.2 Detection of Input Patterns

Input pattern detection is done using autocorrelation analysis. The Appendix gives a brief description of autocorrelation analysis. The detection involves two steps:
Step 1
For a given time series, calculate the autocorrelation coefficients. If a trend is detected, then differencing should be used to remove the trend. This step should be repeated until the trend is removed to a reasonable degree. This step is important as autocorrelation coefficients may be used to determine the lags of residuals for short term forecasting.

Step 2
Calculate the partial autocorrelation coefficients. The information will tell how $y_i$ is auto-correlated to $y_{i+k}$. Choose those partial autocorrelation coefficients that are significantly different from the rest of the coefficients. The largest lag between any two of the coefficients is the number of inputs needed for the neural network forecaster.

4.2.3 Determination of the Number of Neurons in the Hidden Layer

The number of neurons in the hidden layer is a concern in the application of NN to time series forecasting. A rule of thumb, known as the Baum-Haussler rule, is used to determine the number of hidden neurons to be used:

$$N_{\text{hidden}} \leq \frac{N_{\text{train}} E_{\text{tolerance}}}{N_{\text{pts}} + N_{\text{output}}}$$ (4.1)

where $N_{\text{hidden}}$ is the number of hidden neurons, $N_{\text{train}}$ is the number of training examples, $E_{\text{tolerance}}$ is the error tolerance, $N_{\text{pts}}$ is the number of data points per training example, and $N_{\text{output}}$ is the number of output neurons. This rule generally ensures that neural networks generalise, rather than memorise. Baum and Haussler (1988) already defined the proper
number of hidden layer in Theorem 1. They proposed theoretical lower and upper bounds on the sample size vs. net size needed.

4.2.4 Construction of the Neural Network Forecaster

Based on the input patterns and the number of neurons in the hidden layer determined, the neural network forecaster can be constructed. There are two cases that should be considered in time series forecasting: short-term forecasting and long-term forecasting. By short-term forecasting we mean that the neural network forecaster is actually a one-step-ahead predictor. With determined input patterns and the number of neurons in hidden layer(s), we propose the neural network forecaster for the short-term forecasting as shown in Figure 4.3 in which $z^j$ represents the delay operator; that is, $z^{-1}y_k = y_{k-1}$.

![Figure 4.3 NN structure with short term prediction](image)
This structure is distinguished from other neural network forecasters in that residuals are considered as inputs as well. This structure is inspired by mechanism of conventional statistical forecasting models such as ARIMA models, which consider forecasting as a decision made based on several previous successive actual observations, and residuals that are the difference between actual observations and their predictions.

Long-term prediction is of importance in determining the future trend of a time series that requires several or a number of steps ahead predictions. Residuals are no longer available, as actual future data is not known. For long-term forecasting, because residuals are not available, the "feedback" loops from output in Figure 4.3 should be removed. Training long-term forecasters does not involve residual terms - this is apparently different from training short-term forecasters.

4.2.5 Autocorrelation Analysis

In time series forecasting using statistical approaches, the autocorrelation function is extremely useful in obtaining a partial description of a time series for forecasting. Autocorrelation coefficients measure the degree of correlation between neighbouring data observations in a time series. Let’s consider some examples from the Internet. The formula for computing autocorrelations is:

\[
R_k = \frac{\sum (x_k - x_{k,b})^* (y_k - y_{k,b})}{\sum (y_k - y_{k,b})^2}
\]  \hspace{1cm} (4.2)
where \( y \) is the value of the object series, \( x \) is a lagged value of it, and \( y_b \) and \( x_b \) are the respective means; there are \( n-k \) terms in the numerator summation, and \( n \) terms in the denominator summation.

### 4.2.6 The Correlogram

A correlogram is a graph of the autocorrelations versus the time series lags. The user can specify some high order of autocorrelation, say \( k=20 \), for which autocorrelation coefficients are desired in a statistical package designed for time series analysis. When the autocorrelation coefficients are computed a simple autocorrelation correlogram can be plotted against the order such as that illustrated in Figure 4.4.

A partial autocorrelation coefficient for order \( k \) measures the strength of correlation among pairs of entries in the time series while accounting for all autocorrelations below order \( k \). For example, the partial autocorrelation coefficient for order \( k=5 \) is computed in such a manner that the effects of the \( k=1, 2, 3, \) and \( 4 \) partial autocorrelations have been excluded. The partial autocorrelation coefficient of any particular order is the same as the autoregression coefficient of the same order. Figure 4.5 illustrates a partial autocorrelation correlogram.
Figure 4.4 Simple autocorrelation correlogram

Figure 4.5 Partial autocorrelation correlogram

4.2.7 The Simulation Results

Short Term Prediction With NNs
An Acer Veriton 3700GX PC was used for all simulation tests. It has Pentium-D CPU, 1G memory. The MATLAB 7 and its NN toolbox program were used for calculations. In this investigation, the proposed new FTABP learning algorithm was mainly used. Because of long training time required, the TABP learning algorithm was dropped here. The single hidden layer FNN was used with nonlinear transfer functions in the hidden and output layers respectively. Autocorrelation analysis was first used to check the autocorrelations between successive observations. SYSTAT 12 Calculation program was used to calculate the autocorrelation coefficients and partial autocorrelation coefficients. The result is shown in Figure 4.6 in which practical autocorrelation coefficient versus lag is showed. $K$ and $K+15$ are totally different from others. Thus, NN forecasting should use 15 hidden neurons.

**Partial Autocorrelation Plot**

![Partial Autocorrelation Plot](image)

To show the effectiveness of the new algorithm, average daily data of US Dow Jones opening index was considered here. The data was selected between 1990 and 2007 from the historical data of yahoo.finance.com. The first 4,000 indexes were used for training the NN forecaster and the next 50 weeks were forecast in order to make comparisons.
Figure 4.3 shows the NN structure for forecasting for short term prediction. Initial values of weights of input, hidden and output layers were initiated varying from -1 to +1 and were selected randomly. The parameter $\rho$ was selected to be 0.6 for the FTABP algorithm. Learning rates of FTABP were $\gamma = 0.0001$ and $\varsigma = 0.0001$. The methodology to calculate the number of input patterns and the number of hidden neurons are described in Sections 4.3.3 and 4.3.4. Figure 4.7 shows the NN forecaster outputs and the original index.
Figure 4.8 shows the learning performances of the FNN using the FTABP algorithm with nearly 500,000 iterations. After nearly 500,000 iterations, error resulted from using the FTABP algorithm is 0.07. Figure 4.9 shows the performance of the NN forecaster after 500,000 iterations.

Figure 4.8 Performance of FTABP algorithm in logarithmic axes.

Figure 4.9 Performance of NN forecasters for short term prediction
Several NN forecasters with different configurations were trained for short term forecasting. Table 4.1 shows their errors.

<table>
<thead>
<tr>
<th>NN structure (Input-hidden-output)</th>
<th>20-20-1</th>
<th>20-15-1</th>
<th>20-10-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>0.07</td>
<td>0.16</td>
<td>0.46</td>
</tr>
</tbody>
</table>

Table 4.1 Performance of NN forecasters

Consequently all popular training algorithms were used for this simulation. Table 4.2 shows the training algorithms result with previous initial conditions. All learning algorithms were trained with same iterations. BFGS, Quasi-Newton, One step Secant, Scaled Conjugate Gradient Algorithms showed better performances than FTABP. But on the other hand, the FTABP learning algorithms spent less time than the above three algorithms. The training processes were simulated several times. Standard deviations show the digression of average error.

<table>
<thead>
<tr>
<th>Name of Function</th>
<th>Learning Performance</th>
<th>Time</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Levenberg-Marquardt BP</td>
<td>1*</td>
<td>Stoped after 2,000 iterations.</td>
</tr>
<tr>
<td>2</td>
<td>Conjugate gradient BP with Powell-Beale Restarts</td>
<td>2*</td>
<td>Stoped after 442 iterations</td>
</tr>
<tr>
<td>3</td>
<td>BFGS Quasi-Newton BP</td>
<td>0.02</td>
<td>16 min 26 sec</td>
</tr>
<tr>
<td>4</td>
<td>Bayesian Regularization BP</td>
<td>86</td>
<td>70 min 30sec</td>
</tr>
<tr>
<td>5</td>
<td>One Step Secant BP</td>
<td>0.03</td>
<td>12 min 12 sec</td>
</tr>
<tr>
<td>6</td>
<td>Fletcher-Reeves Conjugate Gradient Algorithm</td>
<td>2*</td>
<td>Stoped after 650 iterations.</td>
</tr>
<tr>
<td>7</td>
<td>FTABP</td>
<td>0.05</td>
<td>3 min 17 sec</td>
</tr>
<tr>
<td>8</td>
<td>Scaled Conjugate Gradient Algorithm.</td>
<td>0.02</td>
<td>12 min 14 sec</td>
</tr>
<tr>
<td>9</td>
<td>Modified BP</td>
<td>0.17</td>
<td>6 min 2 sec</td>
</tr>
<tr>
<td>10</td>
<td>TABP</td>
<td>4</td>
<td>3 min 22 sec</td>
</tr>
<tr>
<td>11</td>
<td>Resilient BP</td>
<td>0.8</td>
<td>5 min 7 sec</td>
</tr>
<tr>
<td>12</td>
<td>BPALM</td>
<td>0.4</td>
<td>5 min 7 sec</td>
</tr>
</tbody>
</table>

110
Table 4.2 Results of all algorithms

1*. Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. Training time is about 22 minute after around 2,000 iterations.
2*. Minimum step size reached, performance goal was not met.

From the simulations it can be seen that some of the fast learning algorithms did not perform properly because of the large numbers of input data. Our new FTABP algorithm has proved to be the best among them.

*Long Term Prediction With NNs*

Figure 4.11 shows the NN forecaster for long term prediction. Since residuals were no longer available for forecasting more than one step, we removed the feedback loop. The index is illustrated in Figure 4.12. In this section, a 20-20-1 NN was trained and compared its performance for further 60 weeks.

![Figure 4.10 NNs structure with long term prediction](image-url)
Figures 4.12 and 4.13 show the performance of the FTABP algorithm. After 500,000 iterations, the error of FTABP is 0.08. Industrial daily data of Dow Jones high index was considered between 1990 and 2007 from the finance.yahoo.com.

Figure 4.11 Function to be approximated with long term prediction
Figure 4.13 illustrates the efficiency of the proposed FTABP algorithm. Here the NN output with the proposed new algorithm is much closer to the original index data. In this simulation only the FTABP learning algorithm was used in long term prediction.

Figure 4.12 Performance of the FTABP learning algorithm in logarithmic axes.

Figure 4.13 Performance of forecasters for long term forecasting
4.3 High Accuracy Optical Character Recognition with NNs

4.3.1 Overview
Optical character recognition, usually abbreviated to OCR, is the mechanical or electronic translation of images of handwritten, typewritten or printed text (usually captured by a scanner) into machine-editable text. OCR is a field of research in pattern recognition, and artificial intelligence. In 1929, Gustav Tauschek obtained a patent on OCR in Germany, followed by Handel who obtained a US patent on OCR in USA in 1933 (U.S. Patent 1,915,993). In 1935 Tauschek was also granted a US patent on his method (U.S. Patent 2,026,329). The first commercial system was installed at the Readers Digest in 1955, which, many years later, was donated by Readers Digest to the Smithsonian, where it was put on display. The second system was sold to the Standard Oil Company of California for reading credit card imprints for billing purposes, with many more systems sold to other oil companies.

The accurate recognition of Latin-script, typewritten text has been considered largely a solved problem. Typical accuracy rates exceed 99%, although certain applications demanding even higher accuracy require human review for errors. Handwriting recognition, including recognition of hand printing, cursive handwriting, is still the subject of active research, as is recognition of printed text in other scripts (especially those with a very large number of characters).

In this section, the FNN scheme is implemented to recognize multi-font character images with high accuracy. The study consists of two parts. The first part focuses on
single size and single font characters, and the two-layered NN is trained to recognize the full set ASCII character images. The second part trades accuracy with additional font and size capability, and a larger two-layered NN is trained to recognize American Standard Code for Information Interchange (ASCII) character images commonly used fonts. The recognition process works as follows:

Firstly, the two-dimensional pixel array of the input character is preprocessed, normalized and decomposed into a vector. Secondly, the vector is processed by the neural network to yield an output of 94 numbers. Thirdly, the neuron in the output layer with the highest value is declared the winner, identifying the input character image. Fourthly, a simple postprocessing algorithm is used to detect invalid characters and to discriminate between characters whose images become indistinguishable during preprocessing. These include single quotes and commas of certain fonts and the case information of some characters. The latter part of postprocessing applies to multi-font case only.

4.3.2 Training with Single Font Characters

A single hidden-layer NN which has 1080 inputs, 20 hidden neurons and 93 outputs is used in this simulation. Initial values of weights of input, hidden and output layers are randomly selected between -1 to +1. Figure 4.14 shows the image for converting to data. “A” converts to 30x36 or 20x24 sized matrices and it becomes one column of input data. It means one image one column. If 93 optical characters are selected then input data equals 93x1080 or 93x480 which are massive. Figure 4.15 illustrates the input and output data for FNNs.
Figure 4.14 Image to data preprocessing

The input signal is:

93

<table>
<thead>
<tr>
<th>a</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>C</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
First simulation had 480x93 input data. After 50,000 iterations, the error of the FTABP algorithm equalled to 0.0125. In this simulation a 480x20x93 single hidden layer FNN was trained with learning rate 0.002. Figure 4.16 shows the performance of the new algorithm.

Figure 4.15 The input and output data

Figure 4.16 Performance of FTABP for character recognition with 480 inputs in logarithmic axes.
In the next simulation a large set of input data was used. The number of column was 1080 pixels. Figure 4.17 shows the performance of the FTABP learning algorithm.

<table>
<thead>
<tr>
<th>NNs training function</th>
<th>Learning Performance</th>
<th>Description or training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Levenberg Marquardt BP</td>
<td>No iteration</td>
<td>Out of memory.</td>
</tr>
<tr>
<td>2 Conjugate gradient BP with Powell-Beale restarts</td>
<td>After 533 iterations program stopped</td>
<td>Minimum gradient reached, performance goal was not met.</td>
</tr>
<tr>
<td>3 BFGS Quasi-Newton BP</td>
<td>No iteration</td>
<td>Maximum variable size allowed by the program is exceeded.</td>
</tr>
<tr>
<td>4 Bayesian Regularization BP</td>
<td>No iteration</td>
<td>Out of memory.</td>
</tr>
<tr>
<td>5 One Step Secant BP</td>
<td>0.085</td>
<td>is 27 min 30 sec</td>
</tr>
<tr>
<td>6 Fletcher-Reeves Conjugate Gradient Algorithm</td>
<td>0.074</td>
<td>36 min 38 sec</td>
</tr>
<tr>
<td>7 FTABP</td>
<td>0.083</td>
<td>15 min 03 sec</td>
</tr>
<tr>
<td>8 Scaled Conjugate Gradient Algorithm</td>
<td>Minimum gradient reached</td>
<td>After 529 iterations program stops</td>
</tr>
<tr>
<td>9 Modified BP</td>
<td>0.44</td>
<td>18 min 14 sec</td>
</tr>
<tr>
<td>10 TABP</td>
<td>1.33</td>
<td>15 min 07 sec</td>
</tr>
<tr>
<td>11 Resilient BP</td>
<td>Minimum gradient reached</td>
<td>After 5638 iterations program stops</td>
</tr>
<tr>
<td>12 BPALM</td>
<td>0.097</td>
<td>18 min 3 sec</td>
</tr>
<tr>
<td>13 GDBP with Momentum</td>
<td>0.75</td>
<td>18 min 23 sec</td>
</tr>
<tr>
<td>14 GDBP</td>
<td>1.24</td>
<td>18 min 11 sec</td>
</tr>
</tbody>
</table>

Table 4.3 Results of all algorithms for single font characters

In the next simulation a large set of input data was used. The number of column was 1080 pixels. Figure 4.17 shows the performance of the FTABP learning algorithm. Simulation results with all popular learning algorithms are showed in Table 4.3 after 20,000 iterations.

Figure 4.17 Performance of FTABP with 1080 inputs in logarithmic axes.
4.3.3 Training with Multi-Font Characters

A single hidden-layer NN which has 1080 inputs, 50 hidden neurons and 184 outputs was used in this simulation. Initial values of weights of input, hidden and output layers were randomly selected between -1 to +1. Simulation results after 25,000 iterations are shown in Table 4.4. This table illustrates that most fast learning algorithms did not work. On the other hand, the basic, slow algorithms worked well but training time is too long. From this simulation, the effectiveness and simplicity of the new FTABP algorithm was again proven.

<table>
<thead>
<tr>
<th>NNs training function</th>
<th>Learning Performance</th>
<th>Description or training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Levenberg Marquardt BP</td>
<td>No iteration</td>
<td>Maximum variable size allowed by the program is exceeded</td>
</tr>
<tr>
<td>2 Conjugate Gradient BP with Powell-Beale Restarts</td>
<td>After 1597 iterations program stopped</td>
<td>Minimum gradient reached, performance goal was not met.</td>
</tr>
<tr>
<td>3 BFGS Quasi-Newton BP</td>
<td>No iteration</td>
<td>Maximum variable size allowed by the program is exceeded</td>
</tr>
<tr>
<td>4 Bayesian Regularization BP</td>
<td>No iteration</td>
<td>Maximum variable size allowed by the program is exceeded.</td>
</tr>
<tr>
<td>5 One Step Secant BP</td>
<td>0.055</td>
<td>90 min</td>
</tr>
<tr>
<td>6 Fletcher-Reeves Conjugate Gradient Algorithm</td>
<td>0.058</td>
<td>36 min</td>
</tr>
<tr>
<td>7 FTABP</td>
<td>0.057</td>
<td>18 min</td>
</tr>
<tr>
<td>8 Scaled Conjugate Gradient Algorithm</td>
<td>0.056</td>
<td>45 min</td>
</tr>
<tr>
<td>9 Modified BP</td>
<td>0.16</td>
<td>23 min</td>
</tr>
<tr>
<td>10 TABP</td>
<td>0.96</td>
<td>19 min</td>
</tr>
<tr>
<td>11 Resilient BP</td>
<td>0.138</td>
<td>30 min</td>
</tr>
<tr>
<td>12 BPALM</td>
<td>0.519</td>
<td>21 min 8 sec</td>
</tr>
<tr>
<td>13 GDBP with Momentum</td>
<td>0.572</td>
<td>21 min 24 sec</td>
</tr>
<tr>
<td>14 GDBP</td>
<td>0.872</td>
<td>24 min 29 sec</td>
</tr>
</tbody>
</table>

Table 4.4 Results of all algorithms for multi font characters
4.4 Image Interpolations with Neural Networks

4.4.1 Overview

Interpolation is a process of generating pixels derived from sampled pixels. Figure 4.18 shows the simple image interpolation process and results. The quality of the interpolated image highly depends on the interpolation algorithm. With new video technology improving image quality as well as decreasing bitrates for transmission, image interpolation is increasingly required in video signal processing. Traditional methods for example: Bicubic Convolution [Glassner 1995], Using Approximated sinc Function [Shannon 1948] and Subjective Evaluation [Aokage et al 2003] can be noticed here.

This section presents a new image interpolation method [Aokage et al 2005] using the FNN using the new FTBP algorithm.
4.4.2 The NN and Training Data

The FNN with one input layer, a hidden layer, and an output layer was used. To change the area size that affects to interpolated points, the number of neurons of the input layer was set to 16 (4x4). The output layer had four neurons. The number of neurons in the hidden layer was 20. The activation functions of both hidden and output layers were sigmoid functions.

In the starfish method, the transformation ratios for the interpolated pixel are determined by the neighbouring 16 (4x4) sampled points. Thus, it was assumed that the number of network input was 16(4x4) as an example. Figure 4.19 shows the process of determining a training set from an original image when the image is enlarged doubly.

![Figure 4.19 Process of generating training set](image)

The outline of generating the training set is as follows. Figure 4.19 (a) shows the original image for training and white circles depict the pixels. At first, 16 pixels
indicated by grey circles are selected as inputs. As the correct values, which is to be compared with output values, four pixels are selected from the original image (shown as black circles in the Figure 4.19(c)). This set of 16 pixels and four pixels form a pattern of a training set. Then, as shown in the Figure 4.19(d), the next training pattern is generated in the same way as described above. After generating all patterns of a training set, they are saved as M file. The FNN with the FTABP algorithm was used for interpolation. Figure 4.20 shows the flow chart of the image enlargement system using the FNN. As shown in the figure, the system consists of five parts of processes indicated by rectangles, and data files indicated by dotted rectangles.

![Figure 4.20 Process of enlargement system with ANNs](image)

Figure 4.20 Process of enlargement system with ANNs
4.4.3 Simulation Results

In this simulation, the learning efficiency of the FTABP algorithm is investigated. Two simulations were done with the FTABP algorithm. Figure 4.21 illustrates the image used learning process. The black and white JPEG image was used in the only NN training process before interpolation process. Figure 4.22 shows the learning performances of the FNN using the FTABP with 100,000 iterations. Figure 4.23 shows the original image and Table 4.5 shows the first simulation and TTest results with FTABP learning algorithm. From this table, it can be shown that the FTABP algorithm did better performance than BP with adaptive learning rate.

<table>
<thead>
<tr>
<th>Name</th>
<th>Learning Performance</th>
<th>TTest</th>
</tr>
</thead>
<tbody>
<tr>
<td>BP with adaptive learning rate</td>
<td>0.014</td>
<td>0.245</td>
</tr>
<tr>
<td>FTABP</td>
<td>0.003</td>
<td>0.143</td>
</tr>
</tbody>
</table>

Figure 4.21 Image used NN training process

Table 4.5 The first simulation results of interpolation
Consequently all popular training algorithms were used with same initial conditions.

The results are showed in Table 4.6.
Figure 4.24 shows an interpolated image, which is almost the same with original one. But it can be seen that the image is little dull than the part of Figure 4.23.

![Figure 4.24 Interpolated image](image)

<table>
<thead>
<tr>
<th></th>
<th>NNs training function</th>
<th>Learning Performance</th>
<th>Description or Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Levenberg-Marquardt BP</td>
<td>0.0002</td>
<td>56 min</td>
</tr>
<tr>
<td>2</td>
<td>Conjugate Gradient BP with Powell-Beale Restarts</td>
<td>Minimum step size reached, performance goal was not met</td>
<td>The network stopped after 1387 iterations</td>
</tr>
<tr>
<td>3</td>
<td>BFGS Quasi-Newton BP</td>
<td>0.0003</td>
<td>10 min 50 sec</td>
</tr>
<tr>
<td>4</td>
<td>Bayesian Regularization BP</td>
<td>Maximum MU reached</td>
<td>The network stopped after 1247 iterations.</td>
</tr>
<tr>
<td>5</td>
<td>One Step Secant BP</td>
<td>0.0006</td>
<td>2 min 7 sec</td>
</tr>
<tr>
<td>6</td>
<td>Fletcher-Reeves Conjugate Gradient Algorithm</td>
<td>Minimum step size reached, performance goal was not met.</td>
<td>The network stopped after 2969 iterations.</td>
</tr>
<tr>
<td>7</td>
<td>FTABP</td>
<td><strong>0.0008</strong></td>
<td><strong>1 min 15 sec</strong></td>
</tr>
<tr>
<td>8</td>
<td>Scaled Conjugate Gradient Algorithm.</td>
<td>0.0004</td>
<td>2 min 4 sec</td>
</tr>
<tr>
<td>9</td>
<td>Modified BP</td>
<td>0.032</td>
<td>3 min 10 sec</td>
</tr>
<tr>
<td><strong>10</strong></td>
<td>TABP</td>
<td><strong>0.091</strong></td>
<td><strong>1 min 24 sec</strong></td>
</tr>
<tr>
<td>11</td>
<td>Resilient BP</td>
<td>0.001</td>
<td>2 min 14 sec</td>
</tr>
<tr>
<td>12</td>
<td>BPALM</td>
<td>0.010</td>
<td>2 min</td>
</tr>
<tr>
<td>13</td>
<td>GDBP with Momentum</td>
<td>0.059</td>
<td>1 min 42 sec</td>
</tr>
<tr>
<td>14</td>
<td>GDBP</td>
<td>0.059</td>
<td>1 min 40 sec</td>
</tr>
</tbody>
</table>

Table 4.6 Simulation results
4.5 Summary

This chapter has used the new FTABP learning algorithm for solving three large sized practical problems, namely stock market forecasting, high accuracy optical character recognition and image interpolation. It has further demonstrated the effectiveness of the proposed FTABP learning algorithm.
Chapter 5

Conclusions and Future Research

5.1 Introduction

This chapter summarises the outcome of the research carried out in this thesis. It illustrates the purpose of the research, brief descriptions, and explanations of results. Also included in this chapter, based on the results of this thesis, are some suggestions of what future research can be done in the area.

5.2 Summary of Research and Results

The backpropagation is one of the most popular learning algorithms for FNNs. However, it suffers from the slow convergence. This thesis has developed a set of terminal attractor based learning algorithms for FNNs. The major contributions of this thesis include

- A thorough review of the state of the art of learning algorithms for FNNs.
• A terminal attractor based BP algorithm has been proposed, which improves significantly the convergence speed near the ideal weights. A necessary condition has been derived to avoid the singularity problem.

• A fast terminal sliding mode concept has been adopted to develop a fast terminal attractor based BP algorithm.

• An improved LMBP algorithm based on the terminal attractor has also been developed.

• Several typical applications have been undertaken using the proposed FTABP algorithm.

5.3 Future Research Directions

The thesis has used several types of algorithms for developing new, effective terminal attractor based BP learning algorithms. However, there are other possible BP learning algorithms which can be incorporated as well, for example, Conjugate Gradient, Fletcher Reeves Conjugate Gradient, Modified BP, BFGS Quasi-Newton method, Bayesian Regularisation and One Step Secant algorithms. Further work is required to extend the terminal attractor concept into other BP algorithms.

There is another venue where the terminal attractor based LMBP algorithm may be further improved through combining with some other versions of the LMBP algorithms such as the Modified LM algorithm [Wilamowski et al 2001]. The simulation results using large sided problems proved that the standard LMBP algorithm has a difficulty in computation and demands a large amount of memory for storage. On the other hand, the Modified LMBP algorithm [Wilamowski et al 2001] gives a better convergence rate
compared to the standard LMBP algorithm and is less computationally intensive and requires less memory. Further improvement of the LMBP algorithm can be proposed to give rise to another new fast, simple, effective learning algorithm for FNN.
Appendix

Some popular transfer functions

- **Hard Limit Transfer Function**
  \[ y = \text{hardlim}(x) \]
  - \( y = \text{hardlim}(w \times x + b) \)
  - Single Input **hardlim** Neuron

- **Symmetric Hard Limit Transfer Function**
  \[ y = \text{hardlims}(x) \]
  - \( y = \text{hardlims}(w \times x + b) \)
  - Single Input **hardlims** Neuron

- **Log-Sigmoid Transfer Function**
  \[ y = \text{logsig}(x) \]
  - \( y = \text{logsig}(w \times x + b) \)
  - Single Input **logsig** Neuron

- **Linear Transfer Function**
  \[ y = \text{purelin}(x) \]
  - \( y = \text{purelin}(w \times x + b) \)
  - Single Input **purelin** Neuron
\[ y = \text{radbas}(x) \]
Radial Basis Function

\[ y = \text{radbas}(\text{dist}(w,x) \cdot b) \]
Single Input \text{radbas} Neuron

\[ y = \text{satlins}(x) \]
Satlins Transfer Function

\[ y = \text{satlins}(w \cdot x + b) \]
Single Input Satlins Neuron

\[ y = \text{satlin}(w \cdot x + b) \]
Single \text{satlin} Neuron

\[ y = \text{tansig}(x) \]
Tan-Sigmoid Transfer Function

\[ y = \text{tansig}(w \cdot x + b) \]
Single Input Tan-Sigmoid Neuron
NNs outputs of simulation result 2:

Target output:

\[
\begin{array}{cccccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{array}
\]

Actual output (training data for FTABP):

\[
\begin{array}{cccccccccccccc}
1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\
0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\
0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\
0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 \\
0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 \\
0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0000 \\
0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.9940 \\
0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\
0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\
0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\
0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\
0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\
\end{array}
\]
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Actual output (test data for FTABP):

<table>
<thead>
<tr>
<th>0.7457</th>
<th>0.0000</th>
<th>0.0004</th>
<th>0.0000</th>
<th>0.0000</th>
<th>0.0000</th>
<th>0.0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000</td>
<td>0.9786</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0002</td>
</tr>
<tr>
<td>0.0005</td>
<td>0.0000</td>
<td>0.9311</td>
<td>0.0003</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.0001</td>
<td>0.0000</td>
<td>0.0439</td>
<td>0.8615</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.0007</td>
<td>0.0000</td>
<td>0.0001</td>
<td>0.0000</td>
<td>0.9836</td>
<td>0.0054</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0001</td>
<td>0.1258</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.0000</td>
<td>0.0001</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.9509</td>
</tr>
<tr>
<td>0.0002</td>
<td>0.0000</td>
<td>0.0001</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0011</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.0003</td>
<td>0.0000</td>
<td>0.0006</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0001</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0001</td>
<td>0.0012</td>
</tr>
<tr>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0005</td>
<td>0.0018</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0124</td>
</tr>
<tr>
<td>0.0000</td>
<td>0.0001</td>
<td>0.0002</td>
<td>0.0000</td>
<td>0.0002</td>
<td>0.0000</td>
<td>0.0018</td>
</tr>
<tr>
<td>0.0060</td>
<td>0.0000</td>
<td>0.0015</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0001</td>
<td>0.0000</td>
</tr>
</tbody>
</table>
Used characters in simulation 4.3.3. and 4.3.4
**Bibliography**


Diederich, J. (1990), Artificial Neural Networks: Concept Learning, IEEE Computer Society Press, Los Alamitos, California.


Kartalopoulos, V. (1996), *Understanding Neural Networks and Fuzzy Logic: Basic Concepts and Applications*, Institute of Electrical and Electronics Engineers, NY.


propagation algorithm by error saturation prevention method,” Proceedings of
International Joint Conference on Neural Networks (IJCNN), vol.3, pp.1737-
1742.

Proceedings of Seattle International Joint Conference on Neural Networks, vol 2,
pp.369-377.

neural network for solving linear and quadratic programming problems,” IEEE
Transactions on Neural Networks, vol.12, pp.1074–1083.

feedforward neural networks,” Proceedings of 16th International Conference on
Pattern Recognition, vol.2, pp.84-87.

pattern recognition,” IEEE Transactions on Neural Networks, vol.17, pp.1212-
1221.


Sha, F., and Pereira, F “Shallow Parsing with Conditional Random Fields.”

http://www.seas.upenn.edu/~strctlrn/bib/PDF/shallow.pdf


Author’s Publication List

