1-1-2004

A generalised feedforward neural network architecture and its applications to classification and regression

Ganesh Arulampalam
Edith Cowan University

Edith Cowan University

Copyright Warning

A Generalised Feedforward Neural Network Architecture and its Applications to Classification and Regression

by

Ganesh Arulampalam
B.Eng. (Hons.)

Thesis submitted for the degree of
Doctor of Philosophy (Engineering)

School of Engineering and Mathematics
Faculty of Computing, Health and Science
Edith Cowan University
Perth
WESTERN AUSTRALIA

Supervisor: Associate Professor Abdesselam Bouzerdoum
Associate Supervisor: Dr. Ganesh Kothapalli

In Memory

of

My Father

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

Abstract

Shunting inhibition is a powerful computational mechanism that plays an important role in sensory neural information processing systems. It has been extensively used to model some important visual and cognitive functions. It equips neurons with a gain control mechanism that allows them to operate as adaptive non-linear filters. *Shunting Inhibitory Artificial Neural Networks* (SIANNs) are biologically inspired networks where the basic synaptic computations are based on shunting inhibition. SIANNs were designed to solve difficult machine learning problems by exploiting the inherent non-linearity mediated by shunting inhibition. The aim was to develop powerful, trainable networks, with non-linear decision surfaces, for classification and non-linear regression tasks.

This work enhances and extends the original SIANN architecture to a more general form called the *Generalised Feedforward Neural Network* (GFNN) architecture, which contains as subsets both SIANN and the conventional *Multilayer Perceptron* (MLP) architectures.

The original SIANN structure has the number of shunting neurons in the hidden layers equal to the number of inputs, due to the neuron model that is used having a single direct excitatory input. This was found to be too restrictive, often resulting in inadequately small or inordinately large network structures.

Enhancements to SIANNs have been developed in this thesis that allow the number of shunting neurons to be varied arbitrarily. Experimental results showed that adding more shunting neurons generally improves performance, whereas reducing the number of shunting neurons often results in a degradation of performance. Furthermore, when the number of shunting neurons is reduced, it is not clear what subset of inputs should be used as direct excitatory inputs.

To overcome this limitation, an excitatory signal is derived from the weighted sum of all input signals and used as the direct input to the shunting neuron. The result is a new neuron model, where all inputs are used to derive the excitatory and inhibitory signals, named the *Generalised Shunting Neuron* (GSN). The GSN has the ability to generate complex decision boundaries by simply varying the synaptic weights. Consequently, a single GSN is able to solve complex machine learning problems much more readily; for example, a single neuron achieves perfect classification on some benchmark problems, like the 3-bit parity and Wisconsin Breast Cancer problems.

Furthermore, a new *Generalised Feedforward Neural Network* (GFNN) architecture has been developed and presented here, based on the GSN neuron. This GFNN architecture is more flexible and includes the original SIANN and the *multilayer perceptron* as special cases.

A number of different types of supervised training algorithms have been developed for SIANNs and GFNNs. These include several first- and second-order algorithms based on backpropagation, stochastic algorithms, and a hybrid algorithm combining direct solution using least-squares minimisation with gradient descent.

Additionally, a novel second order training algorithm, called the *Quadratic Neural Network* (QNN) algorithm, has been developed based on a recurrent neural network for bound-constraint quadratic minimisation.

These training algorithms have been successfully used to train SIANNs and GFNNs, and MLPs for comparison, on a number of standard benchmark classification and regression problems. Extensive experiments have been conducted, which show that the GFNNs achieve accuracy levels that are comparable or better than results reported elsewhere in the literature, using smaller networks in most cases.

Declaration

I certify that this thesis does not, to the best of my knowledge and belief:

(i) incorporate without acknowledgement any material previously submitted for a degree or diploma in any institution of higher education;

(ii) contain any material previously published or written by another person except where due reference is made in the text; or

(iii) contain any defamatory material.

Signature Date 30/3/2005

Acknowledgements

I would like to thank my supervisor, Professor Abdesselam (Salim) Bouzerdoum. I am deeply grateful for his guidance, encouragement and support throughout these years of research. His insights have inspired most of the ideas expounded in this thesis. He has been a patient guide and friend, offering advice and tangible assistance as needed, helping me overcome the many hurdles and challenges faced.

To Dr. Ganesh Kothapalli, I am grateful for the encouragement and support shown during the course of research.

My thanks to Dr. R. Chandrasekhar for his invaluable 'Guide to Writing a Thesis', and his insightful and encouraging review comments.

My gratitude goes to the School of Engineering and Mathematics, Edith Cowan University, for providing the equipment and resources to carry out this research. My thanks also to Greg Yu, Computer Systems Administrator, for his endeavours in keeping the computing resources functioning properly, and his friendly assistance when they did not.

I would like to thank all my friends and family who have contributed, directly and indirectly, to the success of this endeavour.

I am deeply indebted to my brother-in-law, Dr. Shunmugam, and sister, Malar, for their unreserved financial and moral support. I would not have been able to complete my research without their backing.

To my parents, I am eternally grateful for their support and encouragement, and the values they have instilled in me. I have dedicated this work to my father, recently departed. His work ethic, attention to detail, and love of learning, have provided me with a role model that has stood me in good stead in completing this work.

Heartfelt thanks to my wife Urmilla, for her unwavering and loving support through these years of study, despite all the sacrifices that my student life has required of her. I am also grateful to my daughters, for their laughter has helped me maintain my perspective on life during the many stressful periods these past years.

Finally, I would like to express my deepest gratitude to my Godfather, my spiritual guide and mentor, at whose suggestion I embarked on this journey of discovery. He has inspired and sustained me in all ways throughout this adventure.

GANESH ARULAMPALAM

Perth, Western Australia

October 2004

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ANN	Artificial neural network
ARV	Average relative variance
BP	Backpropagation algorithm
CG	Conjugate Gradient algorithms
DS	Direct solution
DS-GDM	Direct-solution - Gradient descent with momentum hybrid algorithm
DS-GDX	Direct-solution - Gradient descent with adaptive learning rate and momentum hybrid algorithm
exp	Exponential activation function
GDM	Gradient descent with momentum algorithm
GDX	Gradient descent with adaptive learning rate and momentum algorithm
GF-init	GFNN default initialisation scheme
GFNN	Generalised Feedforward Neural Network
G-MLP	GFNN-code based MLP
GSN	Generalised Shunting Neuron
lgs, *logsig*	Logistic sigmoid activation function
lin	Linear activation function
LM	Levenberg-Marquardt algorithm
LMAM	Levenberg-Marquardt with Adaptive Momentum algorithm
MLP	Multilayer Perceptron
MSE	Mean squared error
MT-MLP	MATLAB Toolbox MLP
NW-init	Nguyen-Widow initialisation scheme

OLMAM	Optimised Levenberg-Marquardt with Adaptive Momentum algorithm
QNN	Quadratic Neural Network algorithm
RBF	Radial Basis Function
ROM	Random Optimisation Method algorithm
SIANN	Shunting Inhibitory Artificial Neural Network
SICNN	Shunting Inhibitory Cellular Neural Network
SSE	Sum of squares error
SVM	Support Vector Machine
tnh, *tanh*, *tansig*	Hyperbolic tangent sigmoid activation function
WTA	Winner-takes-all

Chapter 1

Introduction and Overview

1.1 Background

Artificial neural networks (ANNs) are inspired by the massively parallel processing capability of the biological brain. The biological neural network, or brain, is an intricate web of billions of interconnected cells, called *neurons*. These simple computing units interact through tiny electrical impulses via a massive number of interconnection points called *synapses*. The brain learns and stores its sensory information in the patterns formed by these interconnections and the 'strength' of these connections. A vivid memory is indeed more deeply 'etched' in your brain. The distributed nature of stored information aids in the linking of various experiences, as well as providing robustness and fault tolerance. This means you won't forget your name by losing a couple of neurons!

The computational paradigm of the brain is massive parallelism; it is the concurrent operation of large numbers of interconnected neurons that enables it to perform the complex information processing tasks involved in human behaviour. This biological computing mechanism is the physical controller of all human activity, be it a 'simple' everyday action like catching a ball or picking your mother out of her high school class photograph, or acknowledged challenging tasks like formulating the theory of relativity or writing a sonnet.

Artificial neural networks are based on models of this biological 'supercomputer', with the aim of creating artificial computing structures that can perform a wide variety of tasks. They are abstractions that aim to reproduce some of the functionality

of biological networks – at the moment – at a very much simpler and smaller scale. ANNs have been applied to a large number of diverse problems, from medical diagnosis to the prediction of sunspot activity, data mining and clustering to facial recognition. These networks can learn from a human expert in a supervised manner, in areas like medical diagnosis, or in an unsupervised manner, forming patterns from the very data presented, in applications such as data mining.

The power of parallel computing is evident from the fact that the world's most powerful supercomputers are comprised of thousands of processors operating in parallel, with massive interconnections (Pulleyblank, 2004). ANNs use the same concept of large numbers of computing units working together to form powerful tools for a variety of problems. The difference lies in the model of the 'node' in the parallel structure. The supercomputers of today use powerful processors, essentially sequential machines in their own right, as the basic 'unit'. ANNs take the opposite end of the spectrum, using extremely simple computing units. The form and function of these computation units, or *neuron models*, may vary widely depending on the particular biological behaviour it is modelled on, or the practical function that it is trying to implement.

It is this idea of proposing and developing a neuron model, and subsequently applying and testing networks based on this neuron model, that forms the thrust and contribution of this thesis. In this investigation, we have taken the biological phenomenon of *shunting inhibition* as the function that we wish to incorporate into the neuron model.

Shunting inhibition is a powerful computational mechanism that plays an important role in sensory information processing systems. It was proposed as a plausible physiological model in the early 1960's (Furman, 1965; Lettvin, 1962), and shunting inhibition has since been extensively used to model some important visual and cognitive functions. Shunting inhibitory networks have primarily been part of adaptive (self-organising) systems that use competitive learning. They have been widely used in modelling psychophysical, neurophysiological and cognitive phenomena. To the best of our knowledge, shunting inhibitory networks have not been used in supervised pattern classification or function approximation, other than in the neocognitron (Fukushima et al., 1983) and ART networks (Carpenter & Grossberg, 1988), until recently (see next section).

The application of shunting inhibition to supervised feedfoward neural networks in particular has been emphasised, in order to keep the scope of work manageable. The reasearch has focussed in depth within this scope, in the anticipation of breaking new ground that will open up new areas of research.

1.2 Research Objectives

Recently Bouzerdoum (Bouzerdoum, 1999, 2000) proposed an artificial neural network architecture, based on shunting inhibition, that can be trained to perform pattern classification or function approximation; he named it *shunting inhibitory artificial neural network* (SIANN). SIANNs are feedforward networks that operate using the steady-state solution of the set of ordinary differential equations that govern the dynamics of the shunting networks, thereby avoiding the need to obtain a numerical solution for these differential equations. This allows the network to operate in a static mode, like most artificial neural networks.

The main thrust of this research is to investigate the ability of shunting inhibition-based feedforward networks, particularly SIANNS, when applied to practical problems.

The initial hypothesis is that shunting inhibitory feedforward neural networks are able to form a new class of powerful networks for classification and non-linear regression tasks. The idea is to exploit the inherent non-linearity of shunting inhibition to develop powerful, trainable networks, with non-linear decision surfaces.

The thrust of the research can therefore be broken down into two main objectives:

- To develop efficient training algorithms for the class of shunting inhibitory artificial neural networks, and test the developed algorithms on some benchmark problems in machine learning and pattern recognition.
- To enhance the structure of shunting inhibitory artificial neural networks, and develop a generalised framework for pattern classification and regression using feedforward artificial neural networks.

1.3 Major Contributions

The main contributions to the body of knowledge made in this thesis are listed below.

1. Training algorithms have been developed for the standard SIANN, and tested on a number of benchmark problems. The results of the tests prove that SIANNs are a viable class of trainable neural networks that can be applied to classification and non-linear regression problems.
2. The standard SIANN structure has been enhanced to a more flexible architecture.
3. A *Generalised Shunting Neuron* (GSN) model has been formulated, which allows multiple excitatory and inhibitory inputs, and encompasses both the

standard shunting inhibitory neuron and the perceptron neuron as special cases. The GSN is capable of producing complex non-linear decision boundaries, with a *single neuron* able to solve real world classification problems.

4. A new neural network architecture based on the GSN has been defined, called the *Generalised Feedforward Neural Network* (GFNN) architecture. This architecture provides a broad framework that also contains SIANNs and MLPs as subsets.
5. GFNNs have been applied to a variety of tasks and demonstrated to be a useful and powerful class of neural network, capable of performing well using networks with a very small number of neurons.
6. A variety of training algorithms have been developed for the shunting inhibitory networks, implemented in a manner that allows SIANNs, GFNNs and MLPs to be trained by this common set of algorithms.
7. A novel neural network training algorithm based on bound-constrained quadratic optimisation has been developed, called the *Quadratic Neural Network* (QNN) algorithm, along with a number of its variants.

1.4 Outline of the Thesis

Following is a chapter-by-chapter outline of the thesis that provides a general overview of the structure and content of this thesis.

Chapter 2 is a review of artificial neural networks that aims to explain the relevant terms and concepts. It describes ANNs in general, covering the biological neuron models as well as artificial neural network structures. It also introduces briefly the various learning paradigms, training algorithms, and the types of problems that can be solved using neural networks.

Chapter 3 presents the development of the *Shunting Inhibitory Artificial Neural Network* (SIANN), from its biological roots to the development of the feedforward shunting inhibitory neuron model and the SIANN architecture. The derivation of the differential equation governing the shunting inhibition dynamics is also presented.

Chapter 4 describes the development and testing of a number of gradient-based, direct solution and stochastic training algorithms for SIANNs. It describes the details of the various algorithms and relevant update equations. The chapter then describes the experimental methods and procedures employed throughout the thesis, for assessing the performance of the networks under investigation. They include network structures, initialisation methods, training and testing parameters and criteria. A set

of five benchmark problems, consisting of two synthetic and three real-world problems, are also described. The benchmarks were selected to incorporate a variety of problems, including time series prediction and multi-class classification. The performance of SIANNs on these benchmark problems is tested and analysed here.

Chapter 5 presents the development of a novel training algorithm, called the *Quadratic Neural Network* (QNN) algorithm, and a number of its variants. The algorithm, based on bound-constrained optimisation using recurrent neural networks, is readily able to incorporate constraints on synaptic weights during the weight update phase. Implementation issues such as the practical application to neural networks and adaptive determination of parameters are also addressed. SIANNs have been trained on the benchmark problems using the QNN algorithm and its variants. A quantitative analysis of the performance of these algorithms is presented along with the results.

Chapter 6 presents enhancements to the standard SIANN structure. The original standard structure has the number of neurons determined by the number of input data attributes and class labels. This sometimes results in structures that are too small, or inordinately large, for the particular problem. In this chapter, enhancements are proposed and developed that allow the size of the shunting layer to be reduced or expanded as required. The enhanced SIANN structures have been trained on the benchmark problems. The performance of these enhanced structures is compared to that of the standard SIANN.

The results obtained in Chapter 6 highlight a certain restriction imposed on the shunting neuron model used in the standard SIANN, namely that it can only have a single excitatory input. In Chapter 7, the shunting neuron model is expanded to cater for *multiple excitatory inputs*. The result is a new neuron model named the *generalised shunting neuron* (GSN). The GSN includes the previous shunting neuron model and the traditional perceptron model as special cases. This 'generalised' shunting neuron is used in a new feedforward architecture, called the *Generalised Feedforward Neural Network* (GFNN). Training algorithms have been extended to the GFNN architecture, which includes both SIANNs and MLPs as subsets. The developed GFNN networks have been tested on benchmark problems, and their performance is compared to that of SIANNs.

Chapter 8 compares the performance of shunting inhibition-based networks with the Multi-layer Perceptron (MLP), as well as results from other methods found in the literature. For each benchmark problem, an MLP structure with approximately the same number of synaptic weights as one of the tested GFNN was trained and tested; the obtained results are compared with those of the GFNN and SIANN. Wherever possible, comparisons are also made with other results presented in the literature. The efficiency of the code developed for this thesis is evaluated by comparing it with MLPs generated, initialised and trained using standard MATLAB Neural Network

Toolbox. The chapter ends with a discussion of the overall results obtained across all benchmark problems and network architectures.

Chapter 9 recapitulates the work presented in the earlier chapters and summarises the results of the research, including a discussion on the full scope of the proposed generalised feedforward neural network architecture. It ends with suggestions for future work based on outcomes of the research presented in this thesis.

1.5 Related Publications

1.5.1 Refereed Journal Papers

1. G. Arulampalam and A. Bouzerdoum, "Training Shunting Inhibitory Artificial Neural Networks as Classifiers," *Neural Network World*, vol. 10, pp. 333-350, 2000.
2. G. Arulampalam and A. Bouzerdoum, "Recurrent Neural Network-based Quadratic Optimisation Training Algorithm for Feedforward Neural Networks," *International Journal of Computers, Systems and Signals*, vol. 3, pp. 65-75, 2002.
3. G. Arulampalam and A. Bouzerdoum, "A generalized feedfoward neural network architecture for classification and regression," *Neural Networks*, vol. 16, pp. 561-568, 2003.

1.5.2 Refereed Conference Papers

4. G. Arulampalam and A. Bouzerdoum, "Novel Training Algorithm Based on Quadratic Optimisation Using Neural Networks," in *Bio-Inspired Applications of Connectionism*, vol. 1, J. Mira and A. Prieto, Eds. Berlin: Springer-Verlag, 2001, pp. 410-417.
5. G. Arulampalam and A. Bouzerdoum, "Application of Shunting Inhibitory Artificial Neural Networks to Medical Diagnosis," in *Proc. 7th Australian and New Zealand Intelligent Information Systems Conference (ANZIIS 2001)*, pp. 89-94, 2001.
6. G. Arulampalam and A. Bouzerdoum, "Expanding the Structure of Shunting Inhibitory Artificial Neural Network Classifiers," in *Proc. Intern. Joint Conf. on Neural Networks (IJCNN '02)*, pp. 2855-2860, 2002.
7. G. Arulampalam and A. Bouzerdoum, "A Generalized Feedforward Neural Network Classifier," in *Proc. Intern. Joint Conf. on Neural Networks (IJCNN 2003)*, pp. 1429-1434, 2003.

Chapter 2

Artificial Neural Networks - A Review

2.1 Introduction

Artificial neural networks employ massive interconnection of simple computing cells, called *neurons*, to perform complex information processing tasks. They are inspired by the massively parallel processing capability of the biological brain.

The biological brain consists of billions of biological neurons, each having thousands of connections to other neurons, forming an intricate web. The connection points between the neural pathways are known as *synapses*. Sensory information causes tiny electrical impulses to be generated and transmitted through the neural pathways, via the synaptic junctions, resulting in patterns of activity in the brain. The pattern of the neuronal connections determines the meaning of the electrical signals (Nicholls et al., 1992). The brain learns and stores its sensory information varying the 'strength' of the synaptic connections, thereby changing the patterns formed.

"A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

1. *Knowledge is acquired by the network through a learning process (learning algorithm).*
2. *Interneuron connection strengths known as synaptic weights are used to store the knowledge."* (Haykin, 1999)

The definition above outlines the method of operation of an artificial neural network. From its 'observations' of the environment, the network learns about the environment it is 'placed in'. The experiential knowledge is stored in a distributed manner within its very structure. Subsequently, when a set of inputs is received, the network is able to produce a response consistent with the environment it has 'observed'.

In practical applications, 'placing a network in an environment' involves presenting the network with sufficient examples related to the required task. The network is then trained to produce the desired outcomes, even when presented with previously unseen examples.

Neural networks have been used in a wide variety of applications, such as financial prediction (Bowen & Bowen, 1990; Giles et al., 1997), control of nuclear power plants (Boroushaki et al., 2003; Na et al., 2004), medical diagnosis (Arulampalam & Bouzerdoum, 2001a; Dickhaus, 2001; Kordylewski et al., 2001; Meesad & Yen, 2001), face recognition (Er et al., 2002; Tivive & Bouzerdoum, 2003), signal classification (Arulampalam et al., 1999; McConaghy et al., 2003) and even the classification of odour levels in a piggery (Hanumantharaya et al., 1999)! They offer improved performance over conventional technologies in many areas, including robust pattern detection, signal filtering, data segmentation, data compression, database mining, adaptive control, optimisation and scheduling, and complex mapping.

A comprehensive treatment of the vast field of artificial neural networks is beyond the scope of this chapter. The aim of the chapter is to explain the relevant terms and concepts, described within the context of the general body of knowledge about artificial neural networks. It describes artificial neural networks in general, starting with the biological neuron model and finishing with various artificial neural network structures. It also introduces briefly the various learning paradigms, training algorithms, and the types of problems that can be solved using neural networks.

The next section discusses the biological neuron and biological neural networks. This is followed by two sections devoted to the general concepts of artificial neural networks, including the different classes of neural networks and their structures, and learning paradigms and algorithms. The kinds of problems being tackled is then presented in Section 2.5. Section 2.6 introduces the popular feedforward neural network architecture, namely *multilayer perceptrons* (MLPs), and the error backpropagation algorithm, while Section 2.7 describes *radial basis functions* (RBF) and *support vector machines* (SVMs). This is followed by a section on common training algorithms for feedforward neural networks. The chapter ends with an overview of adaptive network structures followed by the conclusion.

2.2 The Biological Neuron and Brain

Neurons, or nerve cells, are the building blocks of the biological brain. The human brain contains approximately 10^{11} neurons (Stevens, 1994). The brain is a massive, interconnected biological neural network where interconnected neurons 'communicate' with each other to perform various tasks. The neurons signal each other and perform complex tasks via the transmission of electrical impulses, or *action potentials*. These electrical impulses are actually changes in the potential difference across the cell membrane, due to differences in ionic concentrations on either side of the semi-permeable membrane.

Most neurons can be broken into three functional parts: the cell body, the dendrites, and the axon. The cell body, or *soma*, contains the nucleus of the neuron and the biochemical machinery for synthesising molecules and other enzymes. The cell body also generates an action potential if the total received signal strength exceeds a certain threshold level. The *dendrites* are tiny tube like extensions that tend to form a bushy tree around the cell body, sometimes referred to as the dendritic tree. They provide the main physical surface on which the neuron receives incoming signals. The *axon* extends away from the cell body and provides the pathway over which neural signals travel from the cell body for long distances to other parts of the brain and the nervous system. A diagrammatic representation of the structure of a neuron is shown in Fig. 2.1.

An alternative functional view of the neuron has the dendrites as the 'input device' that collects the signals from the other neurons and transmits them to the soma. The soma is the 'central processing unit' that performs an important non-linear processing step: if the total input exceeds a certain threshold, then an output signal is generated. The axon is the 'output device' that delivers the signal to the other neurons (Gerstner & Kistler, 2002). While this 'picture' gives a simple overview of the process, and indeed formed the basis of many early models of the neuron, the actual process is more complex and has many variations and exceptions. Some of the relevant details are described below.

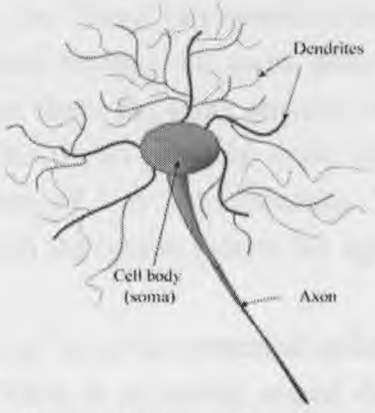


Fig. 2.1: Generic biological neuron structure

2.2.1 Synapses

Information is passed from one neuron to another via a specialised junction point called a *synapse*. A typical neuron may have between 1,000 and 10,000 synapses. *Plasticity* is the ability to adapt the network to its surrounding environment (Haykin, 1999). This is achieved by creating new synaptic connections, varying the strength of existing connections, and removing (pruning) unnecessary connections, and is key to the brain's ability to learn and to retain memories.

The most common type of synapse is a chemical synapse (Gerstner & Kistler, 2002). At synapses, the axon usually enlarges to form a terminal button, which is the information delivering part of the junction. The terminal button contains tiny structures, called synaptic vesicles, which hold chemical neurotransmitters. At this point the axon is very close to the postsynaptic neuron, leaving a tiny gap between the pre- and post-synaptic cell membrane, called the *synaptic cleft*. Nerve impulses (action potentials) at the synapse cause neurotransmitters to be released into the synaptic cleft. When the neurotransmitter molecules reach the postsynaptic membrane, they are detected by specialised chemical receptors that cause an electrical response at the postsynaptic membrane, called the *postsynaptic potential*. If the potential change is positive, it helps to generate nerve impulses; thus, it is known as the *excitatory postsynaptic potential* (EPSP). If the potential change is negative, it opposes the production of nerve impulses; thus, it is termed the *inhibitory postsynaptic potential* (IPSP) (Stevens, 1994). Accordingly, synapses are classified as excitatory or inhibitory, depending on the type of postsynaptic potential generated.

One type of inhibitory synapses works by increasing the conductivity of the cell membrane, thereby 'shunting' the effect of other input potentials and 'clamping' the cell potential to its resting potential. This effect, known as *shunting inhibition*, forms the biological basis for the work presented in this thesis; it is described in greater detail in Chapter 3.

2.2.2 Action potential and spike trains

The basic process describing the 'firing' of a neuron is that if the sum of postsynaptic potentials exceeds a threshold voltage, the soma generates an action potential, a voltage spike, that propagates down the axon, sending the signal to all neurons with synapses connected to it. This action potential spike typically has an amplitude of about 100 mV and a duration of 1 to 2 milliseconds. The spike is followed by a refractory period during which the neuron cannot fire again. Fig. 2.2 shows a typical action potential spike.

Given that the amplitude of an action potential spike of a neuron is always the same, the question arises "How is a 'strong' signal differentiated from a 'weak' signal?" The answer is that a neuron will normally generate a number of action

potentials, called a *spike train*. The strength of the signal is encoded in the number and frequency of the spikes.

The postsynaptic potentials due to the various synapses impact on the soma, that effectively carries out a spatio-temporal summing of signals from all the neurons in contact with it. The temporal summation occurs at the synapse, where chemical reactions result in a potential change whose average amplitude reflects the frequency of the incoming nerve impulses. The spatial summation is effected by the integration of all the postsynaptic potentials 'gathered' by the dendritic tree. The grand sum of the voltage changes, provided it is larger than the threshold voltage, will cause the neuron to fire. This is followed by a period where the cell potential goes below its resting potential, called spike-afterpotential, and it gradually returns to the resting potential. A large input potential will be able to drive the cell potential past the firing threshold faster than a small input potential, thereby causing the neuron to fire faster. This effectively 'encodes' the magnitude in the spike train frequency for transmission to other cells "downstream" in the network.

It should be noted that this level of detail is still a simplification of the complex processes at work in a biological neural system.

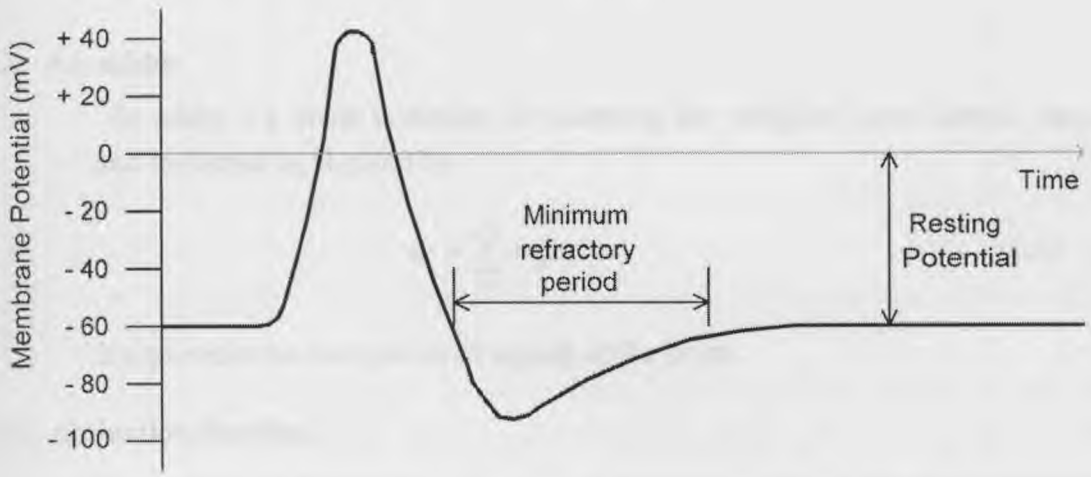


Fig. 2.2: Typical action potential spike

2.3 Artificial Neural Networks

Artificial neural networks (ANNs) mimic the function of biological neural networks. The workings of the biological neuron are modelled mathematically – to varying degrees of complexity – and then simulated, either in software or hardware. These 'artificial' neurons are then combined to form artificial neural networks. This section describes one of the most popular neuron. Each synapse has associated with it a *weight* or strength, *w*. The input neuron models, and outlines the ways in which neurons are combined together to form artificial neural networks.

2.3.1 The Artificial Neuron Model

The most common artificial neuron model is presented in Fig. 2.3. This model has three basic elements to reflect the functions of the biological neuron presented in the previous section:

1) **Synapses** or connecting links

   These correspond to the synapses of the biological neuron. The signal x_j at the input of synapse j connected to neuron k is *multiplied* by the *synaptic weight* w_{kj}.

2) An **Adder**

   The adder is a linear combiner for *summing* the weighted input signals, $w_{kj}x_j$, and its output u_k is given by

$$u_k = \sum_{j=1}^{p} w_{kj} x_j \qquad (2.1)$$

   It represents the integration of signals at the soma.

3) **Activation function**

   The 'firing' of the biological neuron to produce an output signal is modelled by an activation function. The activation function, $\varphi(x)$, is the relationship between the adder output and final output of the neuron. It is often a *non-linear function*, thereby limiting the amplitude of the neuron output. Non-linearity also helps in feature extraction. Normally a constant threshold or bias value (θ) is also added, resulting in the following equation:

$$y_k = \varphi(u_k - \theta_k) \qquad (2.2)$$

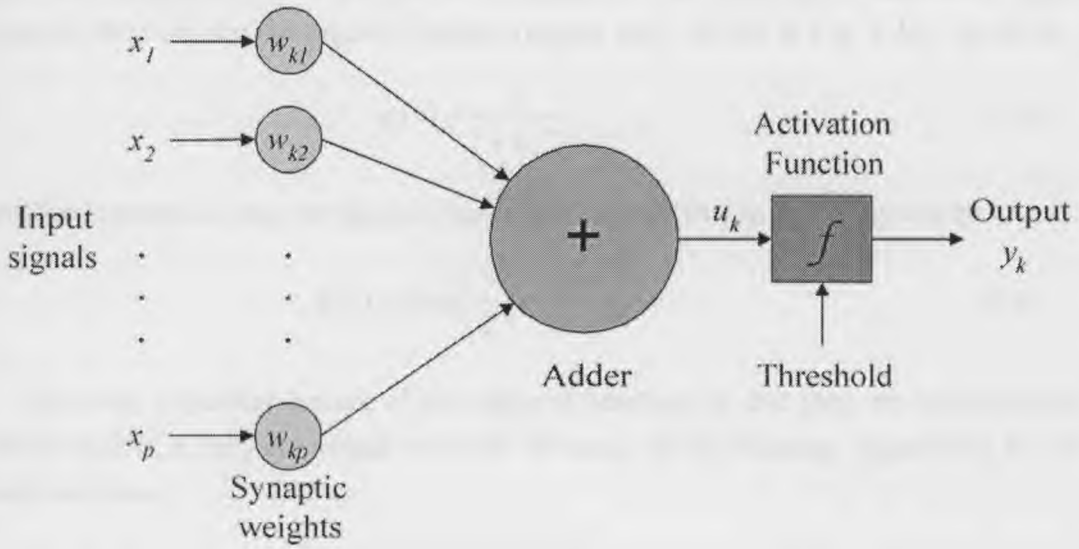


Fig. 2.3: A basic artificial neuron model

2.3.2 Types of activation function

A number of basic activation functions types can be identified (Haykin, 1999, p12). The first is the *threshold function*, shown in Fig. 2.4(a), described by

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases} \tag{2.3}$$

A neuron with this activation function does not produce an output until the total input exceeds a certain threshold level. Once the threshold level is crossed, the full output is produced by the neuron. This type of neuron is called the McCulloch-Pitts model, in honour of the pioneering work done by McCulloch and Pitts (McCulloch & Pitts, 1943).

The second type of activation function is the *piecewise-linear function*, shown in Fig. 2.4(b), where the output is proportional to the input for a given linear range, with the output saturated at the upper or lower bounds outside this range. An example is

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 1 \\ v & \text{if } -1 < v < 1 \\ -1 & \text{if } v \leq -1 \end{cases} \tag{2.4}$$

A special case of this would be the pure linear function, where the linear operation is maintained throughout, without running into saturation.

The third type of activation function is the *sigmoid function*, which is the most popular form of activation function. It is defined as a strictly increasing function that

exhibits smoothness and asymptotic properties. The two most commonly used sigmoid function are the *logistic sigmoid* (*logsig, lgs*), shown in Fig. 2.4(c), given by

$$\varphi(v) = \frac{1}{1+e^{-av}} \tag{2.5}$$

and the *hyperbolic tangent sigmoid* (*tanh, tnh*), shown in Fig. 2.4(d), given by

$$\varphi(v) = \tanh\left(\frac{v}{2}\right) = \frac{1-e^{-v}}{1+e^{-v}} \tag{2.6}$$

The most important feature of the sigmoid functions is that they are continuously differentiable, a very important criterion for most of the training algorithms, as we shall see later.

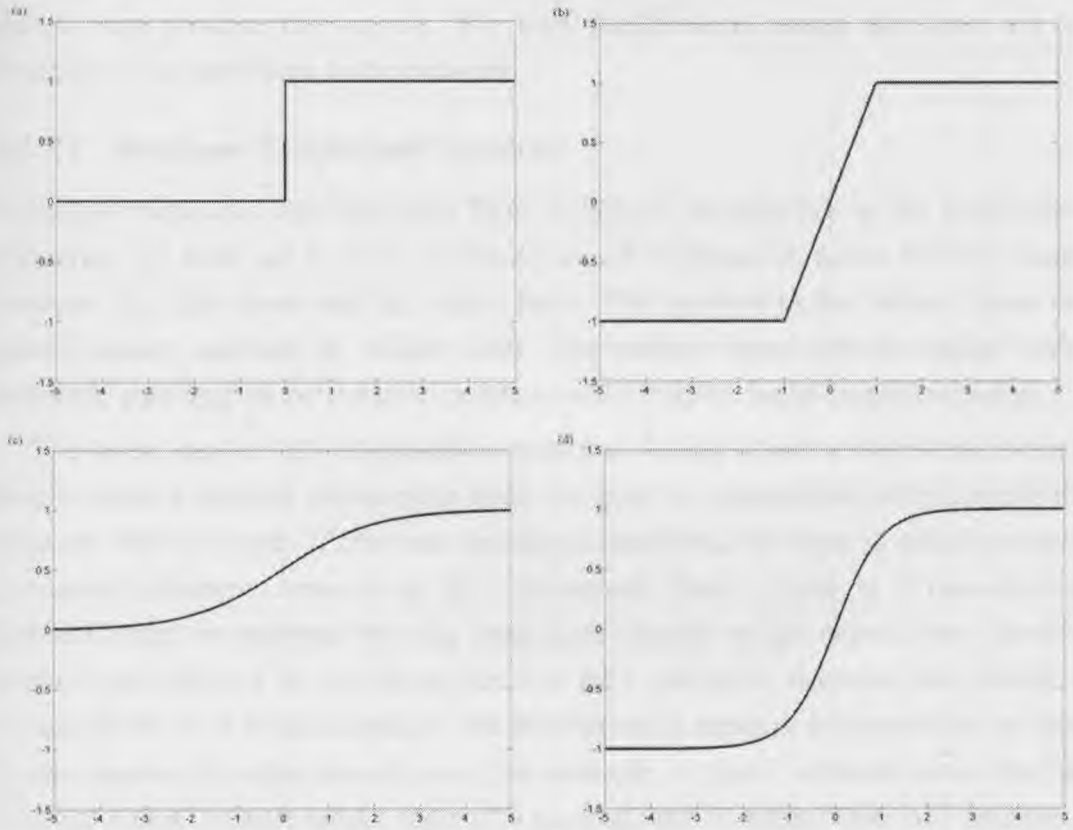


Fig. 2.4: Types of activation functions: (a) threshold, (b) piecewise-linear, (c) logistic sigmoid and (d) hyperbolic tangent

2.3.3 Network Architectures

As the name suggests, neural networks consist of collections of neurons linked together to form a network. The manner in which the neurons are structured in a network is closely linked to the learning algorithm used to train the network. There are three general classes of network architectures: single-layer feedforward networks, multilayer feedfoward networks and recurrent networks (Haykin, 1999, pp21-23). A brief description of each, along with some of the standard terminology that will be used from now on, is given below.

2.3.3.1 Single-layer Feedforward Networks

These networks have neurons (computation nodes) organised in the form of a single layer that form the *output layer* of the network. The *input layer* is simply a set of input sources linked by synaptic connections to the computation nodes. All signals propagate in one direction only, from the inputs to the computation layer neurons that in turn produce the outputs. The term *feedforward* means that there are no feedback loops anywhere in the network.

2.3.3.2 Multilayer Feedforward Networks

Multilayer networks have the same form of layered architecture as the single-layer networks, but with one or more *hidden layers* of computation nodes that are placed between the input layer and the output layer. The neurons in the hidden layers are called *hidden neurons* or *hidden units*. The hidden layers extract higher order statistics, enabling the networks to produce more complex input-output mappings.

The layers can be fully or partially connected. A *fully connected* network is taken here to mean a network where every node in a layer is connected to every node in the *adjacent* forward layer. If there are missing connections, the layer is called *partially connected. Shortcut connections* are connections from a node to a non-adjacent forward layer, for example from the input layer directly to the output layer. Shortcut connections shall not be considered part of a fully connected structure here, though it is considered so in some literature. The structure of a network is represented in short by the number of nodes in each layer. For example, a 10-4-3 network is one that has 10 input nodes, a single hidden layer of 4 neurons, and an output layer of 3 neurons.

2.3.3.3 Recurrent Networks

Recurrent networks differ from feedforward networks in that they have at least one feedback loop. They may be with or without hidden neurons. *Self-feedback* refers to the case where the output of a neuron is fed back as an input to itself. These networks normally have unit delay elements in the feedback loops, resulting in nonlinear dynamical behaviour.

2.4 Knowledge and Learning Process

"Knowledge refers to stored information or models used by a person or machine to interpret, predict or approximately respond to the outside world" (Fischler & Firschein, 1987).

The above is a generic definition of knowledge by Fischler and Firschein. Haykin gives the following definition of learning in the context of neural networks (Haykin, 1999, p50):

"Learning is a process by which the free parameters of a neural network are adapted through a continuing process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place"

The two definitions reinforce the definition of a neural network given in the introduction, which says that neural networks acquire knowledge through a learning process. That definition also says that knowledge is stored in the form of synaptic weights, which is why the learning process is defined above as one of adapting these free parameters. The other point to note is that the definitions refer to 'the outside world' and 'the environment'. Neural networks function by adapting themselves to some external stimulus in order to learn some pattern or trend that can then be used at some other point in time as required.

The *learning process* for neural networks can thus be laid out as follows (Haykin, 1999, p50):

1. The neural network is *stimulated* by the environment.
2. The neural network *undergoes changes* in its free parameters as a result of stimulation.
3. The neural network *responds in a new way* to the environment because of the changes that have occurred in its internal structure.

The changes made to the network are in terms of changes to the synaptic weights in the form:

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n) \qquad (2.7)$$

The calculation of Δw_{kj} is obtained from the *learning rule* used, which is a set of rules for adapting the weights to solve the problem at hand. The *learning paradigm* refers to the manner in which the neural network (learning machine) relates to its environment.

2.4.1 Learning Paradigms

The *learning paradigm* refers to a model of the environment in which the neural network operates. There are three learning paradigms for the training of neural networks: supervised learning, reinforcement learning and unsupervised learning.

2.4.1.1 Supervised learning

In supervised learning, there exists an external 'teacher' with knowledge of the environment, in the form of input-output examples. The difference between the desired output and the actual system output is the error signal that is used to modify the system in order to make the system emulate the teacher. Examples of supervised learning are the least-mean square (LMS) algorithm (Widrow & Hoff, 1960) and back-propagation (BP) algorithm (Rumelhart et al., 1986).

2.4.1.2 Reinforcement learning

In this paradigm, the system receives a reinforcement signal (scalar) based on the actions taken. If positive reinforcement is received, then probability of same action being taken is strengthened or reinforced. Otherwise, the tendency to produce that action is weakened (Sutton et al., 1991). However, the scalar value doesn't indicate if further improvement is possible, or how behaviour should be changed. There is also conflict between the use of existing information and the desire to explore new avenues for improvement. A critic may be used to generate the reinforcement signal.

2.4.1.3 Unsupervised learning

In unsupervised or self-organised systems, there are no external teachers or examples to be learned. Instead, the system learns from the input data presented to it and organises itself accordingly (Becker, 1991). A competitive learning rule or clustering procedure is normally used. It becomes very useful when the size/depth of the network grows large and pure supervised learning becomes unacceptably slow (Jacobs & Jordan, 1991).

2.4.2 Learning Rules

The *learning rule* describes how network 'learns' from its environment i.e. the rule by which the weights of the network are adapted. The term *training algorithm*, on the other hand, is used here to mean the specific set of steps used to update the synaptic weights, and this will fall under the umbrella of one of the learning rules. The learning rules covered here are error correction learning, Hebbian learning, competitive learning, stochastic learning, evolutionary learning and information-theoretic learning. Training algorithms are covered in Section 2.8.

2.4.2.1 Error Correction Learning/ Delta Rule

The *error correction learning rule*, also known as the *delta rule*, assumes that during the learning process the network is presented with a set of examplars from which to learn. These examplars consist of a set of inputs along with the corresponding set of desired outputs. The 'error' is the difference between the actual output of the network and the desired output. This difference is used to work out the changes that need to be made to the weights in order to produce the desired outputs.

For a given input stimulus $\mathbf{x}(n)$, the error signal of neuron k at the n^{th} step, $e_k(n)$, is the difference between the desired response, $d_k(n)$, and the actual response, $y_k(n)$:

$$e_k(n) = d_k(n) - y_k(n) \tag{2.8}$$

The idea is to minimise some *cost function* based on $e_k(n)$, with respect to the synaptic weights of the network. The error term is sometimes denoted as δ (Wasserman, 1989, p. 41), hence the name delta rule. According to the error-correction learning rule (or delta rule), the weight adjustment $\Delta w_{kj}(n)$ is given by (Widrow & Hoff, 1960)

$$\Delta w_{kj}(n) = \eta\, e_k(n)\, x_j(n) = \eta\, \delta_k(n)\, x_j(n) \tag{2.9}$$

where η is the rate of learning and $\delta_k = e_k$.

The choice of η is very important to ensure stability because it acts as a feedback term. For small η, the learning process is smooth but takes a long time, whereas for large η, learning is faster but process may diverge and becomes unstable.

The plot of cost function vs. synaptic weights consists of a multi-dimensional surface called the *error surface*. For a linear neural network, the error surface is a quadratic function of weights, i.e. bowl-shaped with a unique minimum. For a network with non-linear neurons, surface has one or more *global minima* as well as *local minima*. The objective is to start anywhere on the error surface and end up at the global minimum without getting trapped in local minima.

The work presented in this thesis uses error-correction learning almost exclusively. The other methods are only relevant to other referenced works.

2.4.2.2 Hebbian Learning

Hebb's postulate of learning (Hebb, 1949) can be re-presented as follows:

1. *If 2 neurons on either side of a synapse (connection) are activated simultaneously (synchronously), then the strength of that synapse is selectively increased.*
2. *If 2 neurons on either side of a synapse are activated asynchronously, then the synapse is selectively weakened or eliminated.*

In *Hebbian learning*, also known as *correlation learning*, the change in weight is a function of pre- and post-synaptic activities (x_j and y_k) (Kohonen, 1988)

$$\Delta w_{kj}(n) = \eta\, y_k(n)\, x_j(n) - \alpha\, y_k(n)\, w_{kj}(n) \qquad (2.10)$$

where α is a positive constant forgetting factor

The second "forgetting" term is to avoid exponential growth and saturation of $w_{kj}(n)$. $\Delta w_{kj}(n)$ can also be seen statistically as a function of the covariance of pre- and post-synaptic activities.

2.4.2.3 Competitive Learning

In competitive learning only one of the output neurons of the network is allowed to be active. The output neurons compete among themselves for being the one to be active (fired) – the *winner-takes-all neuron*. The network may have lateral connections that perform *lateral inhibition* for the competition to work. Only the winning neuron has its weights adjusted, according to input pattern that made it win.

The basic weight update will be of the form

$$\Delta w_{ji} = \begin{cases} \eta\,(x_i - w_{ji}) & \text{if neuron } j \text{ wins} \\ 0 & \text{if neuron } j \text{ loses} \end{cases} \qquad (2.11)$$

where x_i is *i*th component of input pattern, from input node *i*. The overall effect is to move the weight vector $\mathbf{w}_j$ of winning neuron *j* towards input pattern $\mathbf{x}$. Individual neurons learn to specialize on sets of similar patterns and thereby become feature detectors.

Competitive learning plays an important part in self-organising systems. It is used in Grossberg's Adaptive Pattern Classification (Grossberg, 1973, 1976) and ART networks (Carpenter & Grossberg, 1987, 1988) and Kohonen's Self-Organising Maps (SOM) (Kohonen, 1982).

2.4.2.4 Stochastic and Evolutionary Learning

Stochastic learning rules contain elements that use probabilistic or 'random' events as part of their formulation. Evolutionary algorithms are a separate class, but incorporate some element of randomness in their operation. These non-deterministic methods tend to take longer than deterministic methods, but allow greater coverage of the 'solution space'. The inherent randomness occasionally results in the 'inspired step' that leads to better results.

2.4.2.4.1 Boltzmann learning

The Boltzmann learning rule is a stochastic algorithm derived from information-theoretic and thermodynamic considerations (Hinton & Sejnowski, 1983). In a Boltzmann machine, neurons are in a recurrent structure and operate in a binary fashion: +1 for 'on' state and −1 for 'off' state; none of the neurons has self-feedback. The Boltzmann machine has two modes of operation:

- *Clamped condition*, in which all the visible neurons are clamped to specific states determined by the environment
- *Free running condition*, in which all the neurons (visible and hidden) are allowed to operate freely

The learning algorithm works by randomly flipping the state of one of the neurons. The probability of flipping is based on the states and weights of all neurons, and a pseudo-'temperature'. The weight update according to the Boltzmann learning rule is given by

$$\Delta w_{ji} = \eta(\rho_{ji}^{+} - \rho_{ji}^{-}) \tag{2.12}$$

where ρ_{ji}^{+} is the correlation between states of neurons i and j, conditional on the network being in its clamped condition and ρ_{ji}^{-} is the unconditional correlation between states of neurons i and j (i.e. network in free-running condition).

2.4.2.4.2 Stochastic optimisation

Stochastic optimisation methods update the weight vector of the network, w, using

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \xi(n) \tag{2.13}$$

where $\xi(n)$ is a randomly generated perturbation. The error function $E(\mathbf{w}(n+1))$ is compared with $E(\mathbf{w}(n))$ in order to determine if the new direction in weight space is to be explored (Schalkoff, 1997, p. 294).

2.4.2.4.3 Evolutionary Computation

Evolutionary computation has been widely used to evolve neural network architectures and weights. Evolutionary computing can be divided into three broad categories: *genetic algorithms*, *evolutionary programming* and *evolutionary strategies* (Back, 1997). The first two are commonly used with neural networks.

Genetic algorithms (GA) are defined as algorithms that transform populations of mathematical *objects* (Schalkoff, 1997). The objects codify the real objects to be manipulated (*phenome*) in a manner independent of the problem, usually a string of bits (*genome*). A *fitness function* has to be defined that can give an evaluation score to the object.

The first step is to randomly generate an *initial population* of individuals and evaluate the fitness of each object. The algorithm *selects* probabilistically a subpopulation from the current population, based on the fitness scores. Then some of the individuals are paired up to create a new generation. Here parts of the 'genetic code' from the parents are exchanged using the *crossover* operator to produce the offspring. Some of the individuals in the new generation have a random part of their 'code' inverted as part of the *mutation* process. The fitness of each individual in the new population is then evaluated. The process is repeated until at least one of the individuals in the population has a fitness that exceeds the fitness threshold level, or the number of generations reaches a maximum.

Note that the selection, crossover and mutation processes are non-deterministic. Genetic algorithms, just like the stochastic updates, do not tend to become trapped in local minima. GAs, however, are slow when used for weight adaptation (Schalkoff, 1997, p. 212) and only viable for small structures of less than 50 neurons (Schiffmann et al., 1992b). They show more promise when used for structure adaptation (Schalkoff, 1997; Yao & Liu, 1997), as discussed in Section 2.9.

Evolutionary programming (EP) (D. B. Fogel, 1992; L. J. Fogel et al., 1966), on the other hand, uses a 'natural' representation of the problem, and once chosen mutation operators specific to the scheme are defined. It avoids the need to encode the object in an abstract genomic representation. The other difference between EP and GA is that the mutation operation, the primary operation, changes aspects of the solution according to a statistical distribution that makes minor variations highly probable and substantial variations increasingly unlikely. EP uses stochastic selection via a tournament. Each trial solution competes against a fixed number of opponents, and those with the least 'wins' are eliminated. EP does not explicitly use a crossover operator, though it is argued that this is a matter of philosophy (Back, 1997). EP is apparently the most suited paradigm of evolutionary computing for evolving artificial neural networks (Garcia-Pedrajas et al., 2003), better than GA (Yao & Liu, 1997).

2.4.2.5 Information-theoretic learning

In the last decade, there has been an explosion of information theoretic approaches in neural networks and machine learning (Principe et al., 2004). Descriptors used to quantify information, such as entropy and divergence (or its special case mutual information), are replacing the mean-squared error criteria. The process of learning or adaptation with these new cost functions is named *information theoretic learning*.

The foundations of information theory lie in the work of Shannon (Shannon, 1948). It attempts to quantify the amount of information obtained from the occurrence of any event or message.

The amount of information gained after observing a discrete event $x = x_k$ that has probability p_k is given by

$$I(x_k) = \log\left(1/p_k\right) = -\log p_k \tag{2.14}$$

From this, $I(x_k) = 0$ if $p_k = 1$; that is, if it is known for certain that some event is going to happen, the occurrence of that event doesn't add any information to what is already known. However $I(x_k)$ can never be less than zero, so information cannot be lost through the occurrence of some event (Haykin, 1999).

The entropy $H(x)$ of a discrete random variable x, given by

$$H(x) = E[I(x_k)] = \sum_{k=-K}^{K} p_k I(x_k) = -\sum_{k=-K}^{K} p_k \log p_k \tag{2.15}$$

is a measure of the *average amount of information conveyed per message*. It is also a measure of the prior uncertainty about x. If x is the input of a system with output y, the uncertainty resolved by observing the output, otherwise known as the *average mutual information* between x and y, is given by

$$I(x,y) = H(x) - H(x \mid y) \tag{2.16}$$

where $H(x \mid y)$ is the conditional entropy.

In Linsker's *principle of maximum information preservation*, self-organised learning is achieved by maximising the mutual information between the input-output vectors of the model (Linsker, 1988). This principle, also known as *infomax*, can be used to produce topologically ordered input-output mappings like the SOFM. The idea of maximising mutual information in the unsupervised processing of the image of a natural scene has been used in (Becker & Hinton, 1992).

Renyi proposed a generalised definition of entropy, or information content, that includes Shannon's entropy as a special case (Renyi, 1970). Renyi's entropy has been used as the basis for alternative optimality criteria for supervised neural network training (Erdogmus & Principe, 2000, 2001, 2002; Morejon & Principe, 2004), as well as for clustering of data (Jenssen et al., 2003). A stochastic entropy estimator has also been proposed (Erdogmus et al., 2003).

2.4.3 General methodology for neural network learning

The main task of a neural network is to learn a model of its environment and store that information. The objective is that, for any given set of inputs, the network is able to produce a set of outputs consistent with the environment it is modelling.

The knowledge of the 'world' may be divided into two kinds (Haykin, 1999, p24):

- *Prior information*, facts about the known state of the world
- *Observations* (measurements). These observations of the world are inherently noisy. They form the pool of information from which examplars are selected to train the network

The general methodology by which a neural network is applied to a given problem can be given as follows:

1. A neural network architecture is selected (can be based on prior information of system).
2. A subset of examplars is used to train network by means of a suitable *training algorithm*, depending on the architecture.
3. The network is tested with input data not presented to the network before and the output compared to the actual environment or 'world state'. This is a test of the *generalisation* ability of the network, which is an important capability when a network is applied to a problem.

2.5 Classification and Regression

Neural networks can be applied to a variety of problems, the majority of which fall under the category of classification or regression. This section presents definitions of classification and regression tasks and related terms, including a brief introduction to Bayesian classification.

2.5.1 Classification

Classification is the task of classifying input samples (patterns) into one of a discrete set of possible categories (Mitchell, 1997). The input patterns with d inputs can be represented as points in a d-dimensional Euclidean space E^d, called the *input space*. A pattern classifier is a device that maps the points of E^d into the category numbers, effectively dividing the input space into a number of mutually exclusive subspaces representing the various categories. All input sample points that lie in a particular subspace, or point set, are said to belong to that category. The various subspaces are separated by decision boundaries or *decision surfaces* (Nilsson, 1990). Patterns are said to be *linearly separable* if the classes they represent can be separated by a hyperplane, or a set of hyperplanes (Duda et al., 2001).

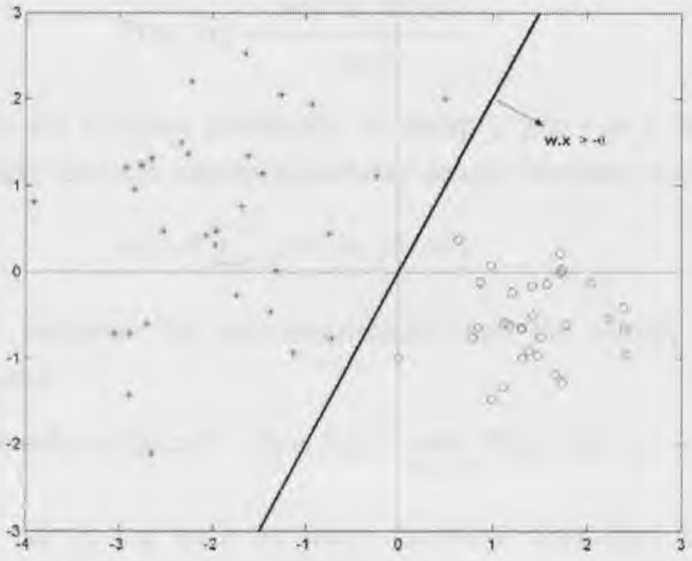


Fig. 2.5: An example of a 2-dimensional hyperplane separating two classes

Nilsson has shown that 'linear machines' can form a set of *linear discriminant functions* of the form

$$g(\mathbf{x}) = \sum_{j=1}^{p} w_j x_j + w_0 \qquad (2.17)$$

A 'machine' with m linear discriminant function units can correctly classify a set of patterns of m distinct classes, provided the patterns are linearly separable (Nilsson, 1990).

Henery extends the above definition of classification, taking it as one of two possible meanings of the term. The first definition above is taken as the 'supervised learning' case, otherwise referred to as *pattern recognition* or *discrimination*. The 'unsupervised learning' meaning of classification refers to *clustering*, where classification is the establishing of the existence of classes or clusters in a given set of data (Henery, 1994).

The term classification used in this thesis shall refer to the first definition above, given by Mitchell, in a supervised learning scenario. Many real-world problems can be treated as classification problems. These include medical diagnosis, character recognition, credit rating, speech recognition, etc.

2.5.1.1 Bayesian Classification Theory

Bayesian decision theory is the basis of statistical classification methods (Duda & Hart, 1973; Duda et al., 2001). Consider an M-group classification problem in which each object has an associated attribute vector $\boldsymbol{x}$. Let ω denote the membership variable that takes the value of ω_j if an object belongs to group j. According to Bayes rule, the posterior probability of group j is given by

$$P(\omega_j \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega_j)P(\omega_j)}{p(\mathbf{x})} \tag{2.18}$$

where $P(\omega_j)$ is the *a priori* probability of group j, $p(\mathbf{x} \mid \omega_j)$ is its conditional probability density function, and the probability density function $p(\mathbf{x})$ is given by

$$p(\mathbf{x}) = \sum_{j=1}^{M} p(\mathbf{x} \mid \omega_j)P(\omega_j) \tag{2.19}$$

In order to minimise the misclassification rate, the widely used Bayesian classification rule is

$$\text{Decide } \omega_k \text{ for } \mathbf{x} \text{ if } \quad P(w_k \mid \mathbf{x}) = \max_{i=1,2,\ldots,M} P(w_i \mid \mathbf{x}) \tag{2.20}$$

This simple rule is the basis for many statistical classifiers. One problem in applying the simple Bayes rule in (2.20) is that, in most practical situations, the density functions are not known or cannot be assumed to be normal, hence the posterior probabilities cannot be determined directly.

2.5.1.2 Advantages of Neural Networks for Classification

Neural networks offer a number of advantages when applied to classification (Zhang, 2000). Firstly, neural networks are data-driven self-adaptive methods, able to adjust to the data without needing an explicit specification of the underlying model. Second, they are universal function approximators and therefore able to map any functional relationships (Hornik et al., 1989). Third, neural networks are non-linear models, making them capable of modelling complex real world relationships. The fourth advantage is that neural networks are able to estimate the posterior probabilities, which provides the basis for establishing classification rules and performing statistical analysis (Richard & Lippmann, 1991).

2.5.1.3 Other Types of Classifiers

Some popular types of non-neural network classifiers referred to are *linear discriminant functions*, *decision trees* and *k-nearest neighbour* classifiers. Fisher's method of linear discrimination (Fisher, 1936) is one of the oldest classification procedures. The idea is to divide the sample space by a series of lines in two dimensions, planes in 3-D and, generally hyperplanes in many dimensions. Decision trees classify instances by sorting them down the tree from the root to the leaf node, which provides the classification for the instance (Mitchell, 1997). The *k*-Nearest Neighbour (*k*-NN) method is a non-parametric method that simply remembers all the training examples and classifies a new observation as the most frequent class of the *k* nearest stored examples. A more detailed coverage of these and other classification techniques can be found in (Duda et al., 2001; Ripley, 1996)

2.5.2 Function Approximation and Regression

Regression analysis concerns the study of relationships between variables, based on random observations (Vapnik, 1998). The estimated relationship can then be used to predict one variable from another (Johnson & Bhattacharya, 1996). Common statistical methods of regression include linear and polynomial regression.

Nonparametric regression addresses the problem of trying to fit a model for a variable Y on a set of possible explanatory variables $\mathbf{X}_1,\ldots,\mathbf{X}_p$, where the relationship between $\mathbf{X}$ and Y may be more complicated than a simple linear relationship. Neural network regression is a special case of nonparametric regression (H. K. H. Lee, 2000). The idea of nonparametric regression is to use models of the form

$$Y_i = f(\mathbf{X}_i) + \varepsilon_i \tag{2.21}$$

where $f \in F$, some class of regression functions, and ε is *i.i.d.* (independent identically distributed) additive error with mean zero and constant variance. Sometimes normality of ε is assumed. The main distinction between the competing nonparametric methods is the class of functions, F, to which f is assumed to belong. In all cases, F is taken to be some class rich enough to be able to sufficiently approximate a very large set of possible regression functions. In other words, nonparametric regression is simply a function approximation task, with added noise. Neural networks are well suited for non-linear regression, recalling that neural networks can be universal function approximators (Hornik et al., 1989).

If the variable or variables to be estimated relate output variables to input variables, then the regression function can be used to *model* the process of the system. If the variables to be estimated are future values then the function is a *predictor* (Specht, 1991). For time-series prediction tasks, temporal information can be presented spatially to the network by a time-lagged vector, also called a tapped delay line (Gershenfeld & Weigend, 1993; Schalkoff, 1997). An alternative is to use recurrent neural networks, since their feedback loops make them well suited to handle such tasks. Recurrent networks have been shown to perform better than feedforward networks on time series prediction tasks (Connor et al., 1994). Neural networks have also been successfully used to track time-varying regression functions (Rutkowski, 2004).

Various types of neural networks have been used for regression tasks, from MLPs (Lawrence et al., 1996; Park et al., 1996; Yao & Liu, 1997) and SVMs (Gunn, 1998; Musicant & Feinberg, 2004) to network ensembles (Islam et al., 2003; Naftaly et al., 1997) and even networks with special types of neurons (Nikolaev & Iba, 2003; Rutkowski, 2004).

2.6 Multilayer Perceptrons

2.6.1 The Perceptron

The perceptron consists of a single neuron with adjustable synaptic weights and a threshold. First introduced by Rosenblatt (Rosenblatt, 1958), it is the simplest form of neural network used for classification of *linearly separable* patterns. The neuron uses a hard-limiter activation function (McCulloch-Pitts model, refer Section 2.3.2). The input to the hard-limiter, u, is:

$$u = \sum_{j=1}^{p} w_j x_j - \theta \tag{2.22}$$

where p is the number of inputs.

The perceptron is therefore able to define two decision regions separated by the hyperplane

$$\sum_{j=1}^{p} w_j x_j - \theta = 0 \tag{2.23}$$

The perceptron inspired Widrow's *Adaline* (Adaptive Linear Element), used for adaptive switching circuits and trained using the LMS algorithm (Widrow & Hoff, 1960). This was followed later by the *Madaline* (multiple adaline), which used a layer of perceptrons (Widrow, 1962).

Minsky and Papert showed that perceptron training is guaranteed to converge provided the examples are linearly separable (Minsky & Papert, 1969). However, they also highlighted the limitations of the perceptrons in handling linearly non-separable problems, dampening research in this area for more than a decade.

It has been shown that even if the activation function is changed from a hard-limiter to another non-linearity such as a sigmoid function, the single-layer perceptron can only properly classify linearly separable patterns (Shynk, 1990; Shynk & Bershad, 1991, 1992).

2.6.2 The Multilayer Perceptron

Although the perceptron may have a nonlinear activation function, the decision surface it represents is still a hyperplane, which is inadequate in most practical situations. This is the weakness of the perceptron. The solution is to use many neurons, arranged in layers, to represent complex nonlinear decision surfaces, i.e. a *Multilayer Perceptron* (MLP). The MLP is a multilayer feedforward network (see Section 2.3.3), where the output signals from one layer are directly fed in as inputs to

the next layer. It is probably the most widely used neural network architecture and has been applied successfully to many diverse and difficult problems. An MLP typically consists of a set of sensory nodes that make up the *input layer*, one or more *hidden layers* of processing units and an *output layer*, which makes the final decision. Fig. 2.6 shows a typical two-hidden layer MLP structure.

The input 'layer' is a set of non-computational nodes that serve as 'distribution points' for the network inputs. The output of the k^{th} neuron in the l^{th} layer (hidden or output layer) is given by

$$y_k^l = \varphi^l\left(\sum_{j=1}^{p} w_{kj}^l y_j^{l-1} + b_k^l\right) \tag{2.24}$$

where φ^l is the activation function for the l^{th} layer, y_j^{l-1} is the output of j^{th} node in previous layer (j^{th} input), w_{kj}^l is the synaptic weigh linking the two nodes, p is the number of inputs, and b_k^l is the bias for the neuron.

In most practical implementations, the bias input to the neurons, b_k^l, is treated as a trainable weight linked to a fixed input of 1. The neuron output given in (2.24) then becomes

$$y_k^l = \varphi^l\left(\sum_{j=0}^{p} w_{kj}^l y_k^{l-1}\right) \tag{2.25}$$

where w_{k0}^l is the bias and y_0^{l-1} has a fixed value of 1. This enables the bias values to be trained in the same manner as the other synaptic weights.

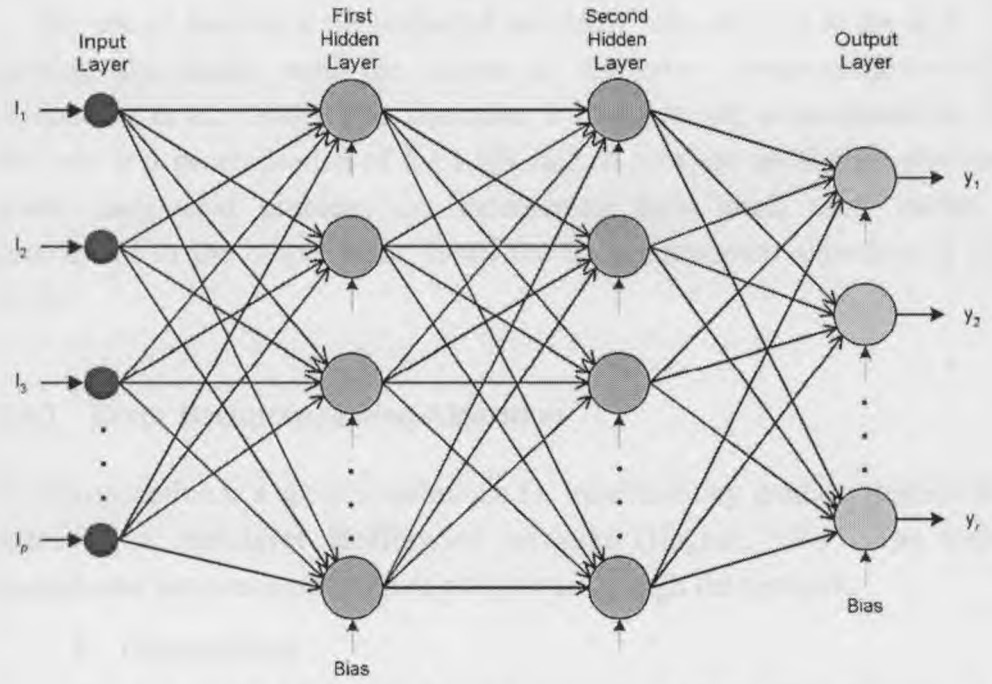


Fig. 2.6: The Multi-Layer Perceptron (MLP) architecture

In classification problems, the function of the hidden layers is to nonlinearly map the input patterns into linearly separable features in the hidden unit space, or *feature space* (Duda et al., 2001, pp. 299-301). The practical goal of training the network is to adapt the synaptic weights so as to transform a linearly non-separable problem in input space into a linearly separable one in feature space.

The capacity of the hidden layer to map the input patterns into a linearly separable form is dependant on the number of hidden units. Increasing the number of hidden neurons increases the dimensionality of the feature space. According to *Cover's theorem* on the separability of patterns, a complex pattern-classification problem is more likely to be linearly separable if nonlinearly cast in high-dimensional space (Cover, 1965). The number of hidden layer neurons is not limited by the problem definition, so it would seem the number of neurons can be increased *ad infinitum* until perfect classification is obtained. In practise, however, there is a limit to number of hidden-layer neurons that can be used, as increasing the number of weights will eventually lead to *overfitting* (refer Section 2.6.6).

It has been shown that an MLP with a single hidden layer can function as a universal approximator, i.e., it can approximate any arbitrary continuous function (Hornik et al., 1989). This is a theoretical analysis, but may not be practical to implement for all functions, as the number of hidden layer neurons required may be too large. On the other hand a two-hidden layer network is able to perform this in a more manageable two-stage fashion (Funahashi, 1989). The first hidden layer extracts local features, whereas the second hidden layer extracts global features from the outputs of the first hidden layer.

The use of multilayer networks did not really take off, due to the lack of proper training algorithms, until the advent of the *error backpropagation* algorithm (Rumelhart et al., 1986). This algorithm is based on the *error-correction learning* rule and is a generalisation of the LMS rule. It provides an elegant solution to the *credit assignment* problem, i.e. determining how much each hidden neuron contributed to the output error. Next, the backpropagation algorithm is explained further.

2.6.3 Error Backpropagation Algorithm

Backpropagation is a specific technique for implementing gradient descent in weight space for a multilayer feedforward networks (Haykin, 1999). The error back-propagation process is made up of two passes through the network:

1. Forward pass

The input signal is applied to the network, and its effect (*function signal*) is propagated through the network, layer by layer

2. Backward pass

The difference between the desired and actual response (*error signal*) is calculated and propagated backward through the neural network. The synaptic weights are adjusted to make the actual response move closer to the desired response using the delta learning rule.

The update for weight w_{ji} connected to neuron j at iteration n, $\Delta w_{ji}(n)$, is given by

$$\Delta w_{ji}(n) = \eta\ \delta_j(n)\ y_j(n) \tag{2.26}$$

where η is the learning rate parameter, $\delta_j(n)$ the error sensitivity and $y_j(n)$ the output signal of neuron j

The sensitivity, $\delta_j(n)$, depends on whether the neuron is an output or hidden node. For the case where neuron j is an output node, the error sensitivity is given by

$$\delta_j(n) = \varphi_j'\left(v_j(n)\right) e_j(n) \tag{2.27}$$

where $\varphi(.)$ is the activation function of the neuron and $e_j(n)$ is the error signal given by the difference between the desired and actual outputs

$$e_j(n) = d_j(n) - y_j(n) \tag{2.28}$$

For the case where neuron j is a hidden node, the sensitivity is given by

$$\delta_j(n) = \varphi_j'\left(v_j(n)\right) \sum_k \delta_k(n)\, w_{kj}(n) \tag{2.29}$$

where δ_k is the δ term from the forward layer neuron k, which is then weighted by the synaptic weight w_{kj} between neuron j and k. In other words, the error term for a hidden neuron is the weighted sum of the error terms of all the neurons it is connected to in the forward layer.

The net effect is that the error signals (δ) propagate backwards, weighted by the synaptic weights, hence the name of backpropagation algorithm. An important point to note here is that the calculation of the δ term, as given in Eq. (2.27) and (2.29), requires the calculation of the differential of activation function (φ'). This means that the activation function φ needs to be differentiable everywhere, like the sigmoidal function. Conversely, the hard-limiter or threshold function cannot be used because of the step (discontinuity) in the function. Further details of this algorithm and improvements to it are covered along with other training algorithms in Section 2.8.

2.6.4 Initialisation

Initialisation refers to the setting of the starting weights and biases before training starts. The weights are normally initialised to a set of random values within a small range. This is so that the networks start from different 'points' in the weight space, increasing the chance of finding the global minimum. The weights are kept small initially, as large weights may result in the neuron outputs going into their saturation regions early in the training phase. This phenomenon, called *premature saturation*, can lead to longer training times (Y. Lee et al., 1991).

A common practise is to have the set of random values uniformly distributed in the range $[-r, r]$. The value of r may be fixed globally or varied from neuron to neuron depending on factors such as the number of inputs to the neuron. Other distributions of weights, such as the Gaussian distribution, can also be used. A good review of initialisation methods for MLPs, including experimental results, is given in (Thimm & Fiesler, 1997).

2.6.5 Training modes: pattern mode vs. batch mode

The error back-propagation algorithm is an error correction algorithm, falling under the supervised learning paradigm. A set of examplars, called the *training set*, is presented to the network. One complete presentation of the whole training set is called an *epoch*. The process of presenting the training examples and updating the synaptic weights is repeated until the mean error over the whole training set falls below a particular value, or some other stopping criterion is met. The frequency of the weight update depends on the mode of training, *pattern mode* or *batch mode*.

In pattern mode, the synaptic weights are updated after each training example is presented. This mode is also referred to as *training by sample* or *online training*. A degree of randomness can be added to the weight updates by randomly changing the order in which the examples are presented at each epoch. This makes it less likely for the training algorithm to be trapped in a local minimum.

In batch mode training, the weight update is performed once an epoch, after all training examples in the set are presented. This is an accumulated correction that represents a smoothing of the weight correction, and avoids mutual interference of weight updates from different examples (Battiti, 1992).

2.6.6 Generalisation and validation

Generalisation is the ability of the neural network to correctly compute the input-output relationship for data not seen during training. As mentioned earlier, the ability to generalise well is crucial in practical applications. *Overfitting* happens when the

network is given "too much" information either in the form of too many neurons for the given problem or excessive training of the network. It tends to represent the input-output relationship for the training examples almost exactly, but doesn't interpolate or extrapolate well. This reduces the generalisation performance of the network. The generalisation performance of the network is tested using a set of sample data not used at any point in the training process, called the *test set*.

In order to improve the final generalisation ability of the network, a third set of sample data is brought into play, called the *validation set*. The validation set is normally a small subset of the training data, but is *not* used in determining the weight updates, i.e., not part of the actual training set. At the end of every epoch of training, the validation data is presented to the network and the error across the whole set worked out. This provides an estimate of the generalisation ability of the network. If the validation error indicates that the network is over-fitting, the training is stopped. This is called *early stopping*. The criteria for stopping can be that the validation error continues to grow for a certain number of epochs, or that it exceeds a certain level above the minimum validation error achieved up to that point. In some implementations the network state that produces the minimum validation error is saved and used as the final network, if training is stopped early.

Determining the 'optimum' network structure for a given application is not an easy task. A structure that is too large will tend to overfit, whereas a structure that is too small may not be able to represent the input-output relationship accurately. Prior knowledge is used where applicable, otherwise trial-and-error is commonly used. Construction and pruning algorithms that modify the network structure as part of the training process are discussed in Section 2.9.

2.6.7 Error surface and local minima

The error value used to determine the performance of the network is a function of the weights of the network. For a fixed structure network, these error values can be visualised as forming an *error surface* in multi-dimensional space. The objective is to modify the network weights until the global minimum of this error surface is reached. For a multilayer network this error surface can be quite complex, and may contain multiple *local minima*. The training algorithms need to be able to avoid getting stuck in the local minima, in order to be able to reach the global minimum – if it can find it. This task is easier said than done. It can be likened to a blind man searching a mountainous landscape for the lowest point, with nothing more than what he can feel around him – in this case gradient information – and where he has been, provided he doesn't forget! Methods of getting out of local minima and improving the speed of training are covered in Section 2.8.

2.7 RBF networks and Support Vector Machines

Radial Basis Function (RBF) networks and Support Vector Machines (SVMs) are two other classes of neural networks that use the concept of non-linear transformations that attempt to convert the input patterns into linearly separable classes, as discussed in the preceding section. While this process is implicit in MLPs, it is explicit in these networks as they are designed with this process in mind. In this section, the basic concepts and modes of operation RBF networks and SVMs are presented, with comparisons to MLPs where appropriate.

2.7.1 Radial-Basis Function Networks

Radial-Basis Function (RBF) networks use the viewpoint that learning is equivalent to finding a surface in multi-dimensional space that provides a "best fit" to the training data. A radial-basis function network in its most basic form consists of three different layers:

1. Input layer of sensory nodes
2. Hidden layer of non-linear nodes of high enough dimension.
3. Output layer that is linear.

The purpose of the hidden layer nodes is to nonlinearly transform the input space to a higher dimensional feature space, for reasons described in the previous section. The hidden units provide a set of "functions" that constitute an arbitrary "basis" for the input vectors when they are expanded into hidden-unit space, called *radial-basis functions* (Powell, 1987). The output of an RBF network can be described by

$$y_k(\mathbf{x}) = \sum_{j=1}^{P} w_{kj}\varphi(\mathbf{x}) + w_0 \tag{2.30}$$

where $\varphi(\mathbf{x})$ is the basis function. This is similar in form to the linear discriminant function in (2.17).

Radial functions are a special class of function. Their characteristic feature is that their response decreases monotonically with distance from a central point. A typical radial function is the Gaussian, which is given by

$$\varphi(\mathbf{x}) = \exp\left(\frac{-\|\mathbf{x}-\mathbf{c}\|^2}{2\sigma^2}\right) \tag{2.31}$$

where $\mathbf{c}$ is the centre of the distribution function and σ is the spread (radius).

The argument of the activation function is the *Euclidean norm (distance)* between the input vector and the centre of that unit. The closer the input vector is to the centre

of the function, the larger the output of the function is, with the maximum being when the two are identical.

The RBF network produces *local* approximations to non-linear input-output mapping. This results in faster learning, and a reduced sensitivity to the order of presentation of training data. However, to represent the mapping smoothly, the number of RBFs required to span the input space may be large. This contrasts with the MLP, which does global approximations and is therefore able to generalise in regions of input space with little or no training data.

In an RBF network, the hidden layer activation functions evolve slowly in accordance with a nonlinear optimisation strategy, whereas the output layer weights change rapidly following a linear optimisation strategy. The layers perform different tasks, so it is reasonable for them to have different optimisation techniques working on different time scales. Different learning strategies may be followed, depending on how the centres of the RBFs are specified. There can be randomly generated fixed centres, supervised selection of centres where the centre positions are trained with the other parameters, or a hybrid learning process where the centres are self-organising. In all cases, however, the linear output weights are trained using a supervised training rule. In summary, radial basis function networks provide a global approximation to the target function, represented by a linear combination of local kernel functions (Mitchell, 1997).

2.7.2 Support Vector Machines

Support Vector Machines (SVM) are a relatively new technique for solving pattern recognition problems, based on statistical learning theory, that contain polynomial classifiers and RBF networks as special cases (Scholkopf et al., 1997). Traditional techniques for pattern recognition are based on minimising the empirical risk (such as the mean squared error), which optimises performance on the training set. SVMs on the other hand, attempt to minimise the structural risk, that is the possibility of misclassifying yet-to-be-seen patterns for a fixed but unknown probability distribution of data (Pontil & Verri, 1998).

The key idea of SVMs can be explained as follows (Vapnik, 1998). Given a training set S that contains points of either of two classes, an SVM separates the classes through a hyperplane determined by certain points of S, termed *support vectors*. In separable cases, the hyperplane maximises the *margin*, or twice the minimum distance of either class from the hyperplane, and all support vectors lie at the same minimum distance from the hyperplane (thus termed margin vectors). In cases where the classes are not separable, both the hyperplane and support vectors are obtained by solving a constrained optimisation problem where the solution is a trade-off between the largest margin and the lowest number of errors. To improve the

separability of the input patterns, they are mapped nonlinearly to a higher-dimensional space by use of kernel functions (such as a Gaussian function), similar to RBF networks.

SVMs are attractive because of their ability to condense the information in the training set and their use of families of decision surfaces of relatively low VC dimension. The *Vapnik-Chervonenkis (VC) dimension* (Vapnik, 1998; Vapnik & Chervonenkis, 1971) is used in statistical learning theory (a.k.a. *VC Theory*) as a measure of complexity (capacity) of a set of approximating functions. For binary classification, the VC dimension is the maximum number of points, h, that can be 'shattered' (classified in all 2^h ways) by the family of dichotomies (binary classification functions or decision rules).

To allow for more general nonlinear decision surfaces, the set of input vectors is nonlinearly mapped into a high-dimensional space by a suitable kernel function K before linear separation is performed. This leads to a decision function of the form (Vapnik, 1998)

$$f(\mathbf{x}) = \text{sign}\left[\sum_{\text{support vectors}} y_i \alpha_i . K(\mathbf{x}, \mathbf{x}_i) + b\right] \tag{2.32}$$

Once again the decision function is similar in form to the linear discriminant function given in (2.17).

Examples of kernel functions are

$$K(\mathbf{x}, \mathbf{x}_i) = \exp\left(\frac{-\|\mathbf{x} - \mathbf{x}_i\|^2}{c}\right) \qquad \text{Gaussian (RBF)}$$

$$K(\mathbf{x}, \mathbf{x}_i) = \left[(\mathbf{x} \cdot \mathbf{x}_i) + 1\right]^d \qquad \text{Polynomial}$$

The performance of SVMs with Gaussian kernels has been compared to classical RBF classifiers and shown to have lower error rates (Scholkopf et al., 1997). The main reason for this is that the classical RBF method of centre selection is based on the concept of clustering of training data, as opposed to SVMs that attempt to minimise the structural risk thereby resulting in better generalisation. The support vector method has also been modified to train MLPs (Suykens & Vandewalle, 1999).

2.8 Training algorithms for Feedforward Networks

The objective of neural network training, using error correction learning, is to minimise some predefined error function such as the sum squared error (SSE). The error function is viewed as an optimisation or minimisation problem in v-dimensional weight space ($\mathbb{R}^v$), where v is the number of free parameters (weights) to be determined from training (van der Smagt, 1994). The state of the network can then be visualised as taking a "walk" through this weight space until some optimal point is reached where the error function is at a minimum. Ideally the minimum reached is a global minimum, not just a local minimum. Various training algorithms have been developed for training feedforward networks based on different approximations and assumptions regarding the error function. The primary consideration has been to determine the direction and size of the "step" to be taken at each iteration of the training. A general description of some common types of training algorithms follows.

2.8.1 First Order Methods

In first-order methods only the first two (constant and linear) terms of the Taylor-series expansion of the error term are considered. These methods, where the local gradient alone determines the direction of minimisation $\mathbf{u}$, are known as *steepest descent* or *gradient descent* methods. For feedforward network training, it is known as error backpropagation (BP), as described in section 2.6.3.

2.8.1.1 Steepest Descent / Standard Backpropagation

When the network is in a state with weight vector $\mathbf{w}(n)$, the gradient of the error function E with respect to w is computed as

$$\mathbf{g}(n) = \frac{\partial E}{\partial \mathbf{w}(n)} \tag{2.33}$$

A minimisation step in the direction $\mathbf{u}(n) = -\mathbf{g}(n)$ is performed.

In normal steepest descent minimisation techniques, a one-directional minimisation in the direction $\mathbf{u}(n)$ is performed such that a point $\mathbf{w}(n)$ is reached where the new gradient $\mathbf{g}(n+1)$ is perpendicular to $\mathbf{u}(n)$. The learning rule is then

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n)\,\mathbf{u}(n) \tag{2.34}$$

and the new search direction is

$$\mathbf{u}(n+1) = -\mathbf{g}(n+1) \tag{2.35}$$

However in standard BP, the line minimisation is replaced by a fixed step-size (learning rate) η in order to reduce the number of function evaluations.

2.8.1.2 Backpropagation with momentum

The BP search direction is often augmented with a momentum term (Rumelhart et al., 1986)

$$\mathbf{u}(n+1) = -\mathrm{g}(n+1) + \alpha\,\mathrm{u}(n) \tag{2.36}$$

A fraction of the previous update is included in the current update, keeping the update going in the same general 'direction'. This extra term is generally interpreted as avoiding oscillations as well as preventing the algorithm from getting stuck in local minima.

2.8.1.3 Backpropagation with variable learning rate

If the learning rate, η, is too small, the number of iterations to arrive at a solution may be very large. On the other hand having η too large may result in the weights oscillating during iterations. A dynamic learning rate, η_i, that varies at each iteration can overcome the need for trial-and-error methods for selecting the learning rate.

One method of varying the learning rate is to use the direction cosine of the error derivative vector to obtain information on error surface curvature (Hsin et al., 1992). The change in the weight vector, Δw, between two successive iterations follows the steepest descent direction for minimising the error function. If the direction is almost the same as the previous direction, this implies the local shape of the error function is relatively unchanged; therefore, a large value of $\eta(n)$ may be used to speed up the process of minimisation. If the current direction is quite different from the previous direction, it implies that the local shape is rather complex and that a smaller η_i value should be applied to avoid overshooting.

A simple method is to use only current and previous direction cosines (Franzini, 1987). An alternative is to use a weighted average of a number of previous directions since they also contain some information about the local error surface (Hsin et al., 1992). In this method, the modified dynamic learning rate, $\eta(n)$, is a weighted average of L+1 successive weight vectors and is given by

$$\eta(n) = \alpha_0 \frac{\Delta\mathrm{w}(n).\Delta\mathrm{w}(n-1)}{\|\Delta\mathrm{w}(n)\|\|\Delta\mathrm{w}(n-1)\|} + \cdots + \alpha_L \frac{\Delta\mathrm{w}(n-L).\Delta\mathrm{w}(n-L-1)}{\|\Delta\mathrm{w}(n-L)\|\|\Delta\mathrm{w}(n-L-1)\|} \tag{2.37}$$

where $\alpha_0 + \alpha_1 + \alpha_2 + \ldots + \alpha_L = 1$

and $\alpha_0 \geq \alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_L$

Another 'quick and dirty' method is the "Bold Driving" method (Battiti, 1989). The method increases the learning rate at successive iterations as long as the error decreases. If the error increases the learning rate is reduced.

$$\eta(n) = \begin{cases} \rho\,\eta(n-1) & \text{if } E(n) < E(n-1) \\ \sigma\,\eta(n-1) & \text{if } E(n) \geq E(n-1) \end{cases} \tag{2.38}$$

where typical values of the constants are $\rho = 1.1$ and $\sigma = 0.5$.

The inefficiency of steepest descent is due to the fact that the minimisation direction and step size are often poorly chosen; unless the first step leads directly to the minimum, steepest descent will zig-zag with many small steps. While backpropagation of error gradients has proven useful, the convergence tends to be slow, particularly when the number of weights in the network are large (Johansson et al., 1992; van der Smagt, 1994).

2.8.2 Second Order Methods

Other numerical methods make use of the second derivative of the function. In this case the quadratic term of the Taylor expansion is also taken into account.

This error equation has the form

$$\Delta E(\mathbf{w}) = E(\mathbf{w} + \Delta\mathbf{w}) - E(\mathbf{w}) \approx \mathbf{g}^T \Delta\mathbf{w} + \tfrac{1}{2}\Delta\mathbf{w}^T \mathbf{H} \Delta\mathbf{w} \tag{2.39}$$

where

$\mathbf{g} = \dfrac{\partial E}{\partial \mathbf{w}}\Big|_{\mathbf{w}_0}$ is the gradient vector and

$\mathbf{H} = \dfrac{\partial^2 E}{\partial \mathbf{w}^2}\Big|_{\mathbf{w}_0}$ is the Hessian matrix

Minima are located at points where the gradient to equation (2.39) is 0, i.e. $\mathbf{H}\Delta\mathbf{w} + \mathbf{g} = 0$.

Therefore the optimal change in the weight matrix, $\Delta\mathbf{w}_{opt} = -\mathbf{H}^{-1}\mathbf{g}$. However, the calculation of the Hessian $\mathbf{H}$ and its inverse is computationally prohibitive, thereby leading to approximation methods being investigated. The above is the basis of *Newton's method* and its variants.

2.8.2.1 Quasi-Newton methods

Newton's method is one of the more successful algorithms for optimisation and, if it converges, has at least a quadratic order of convergence. However, for a general nonlinear objective function, convergence to a solution cannot be guaranteed from an arbitrary initial point. The aim of quasi-Newton (secant) methods such as the BFGS (Broyden-Fletcher-Goldfarb-Shanno) and DFP (Davidon-Fletcher-Powell) methods (Chong & Zak, 1996, pp 147-165 ; van der Smagt, 1994) is to avoid the computation of the inverse matrix $\mathbf{H}^{-1}$ by iteratively computing the matrices $\mathbf{Q}(n)$ such that

$$\lim_{n\to\infty}\mathbf{Q}(n)=\mathbf{H}^{-1} \tag{2.40}$$

The term *quasi-Newton* applies if

$$\mathbf{Q}(n+1)(\mathbf{g}(n+1)-\mathbf{g}(n))=\mathbf{w}(n-1)-\mathbf{w}(n) \tag{2.41}$$

is satisfied. The resulting $\mathbf{Q}(n)$ can be used to find

$$\mathbf{w}(n+1)=\mathbf{w}(n)-\mathbf{Q}(n)\mathbf{g}(n) \tag{2.42}$$

until a minimum is reached.

The disadvantage of these methods is that the storage requirements of $\mathbf{Q}(n)$ is proportional to the square of the number of weights being trained (Hagan & Menhaj, 1994; Johansson et al., 1992; van der Smagt, 1994).

2.8.2.2 *Conjugate Gradient Methods*

In conjugate gradient (CG) optimisation, the direction of the minimisation is always chosen such that the minimisation steps in all previous directions are not spoiled. When a direction $\mathbf{u}(n)$ is chosen and line minimisation is performed in this direction leading to a point $\mathbf{w}(n+1)$, the gradient $\mathbf{g}(n+1)$ at $\mathbf{w}(n+1)$ must be orthogonal to $\mathbf{g}(n)$, $\mathbf{g}(n-1)$, ... $\mathbf{g}_0$, hence the name. The weights and direction updates are given by

$$\mathbf{w}(n+1)=\mathbf{w}(n)+\alpha(n)\mathbf{u}(n) \tag{2.43}$$

$$\mathbf{u}(n+1)=-\mathbf{g}(n+1)+\beta(n)\mathbf{u}(n) \tag{2.44}$$

The algorithm requires the Hessian of the function to evaluate two constants, $\alpha(n)$ and $\beta(n)$. In order to avoid the computation of this matrix, a line search is used to evaluate $\alpha(n)$, whereas for $\beta(n)$ there are a number of formulas that compute it from the gradient and direction vectors such as the *Hestenes-Steifel formula*, *Polak-Ribiere formula* and *Fletcher-Reeves formula* (Chong & Zak, 1996 , pp 132-145).

Hestenes-Steifel formula: $$\beta(n)=\frac{\mathbf{g}^T(n+1)[\mathbf{g}(n+1)-\mathbf{g}(n)]}{\mathbf{u}^T(n)[\mathbf{g}(n+1)-\mathbf{g}(n)]} \tag{2.45}$$

Polak-Ribiere formula: $$\beta(n)=\frac{\mathbf{g}^T(n+1)[\mathbf{g}(n+1)-\mathbf{g}(n)]}{\mathbf{g}^T(n)\mathbf{g}(n)} \tag{2.46}$$

Fletcher-Reeves formula: $$\beta(n)=\frac{\mathbf{g}^T(n+1)\mathbf{g}(n+1)}{\mathbf{g}^T(n)\mathbf{g}(n)} \tag{2.47}$$

For quadratic functions with v degrees of freedom, only v iterations are required to arrive at a solution. However, since the error functions are not exactly quadratic, as well as a result of round-off errors, this does not normally happen in neural network

training. The common practise is to reinitialise the direction vector to the negative of the gradient vector every v steps. An improvement to this is the *Powell restart* procedure, which uses the second order information in resetting the direction vector (Powell, 1977).

The CG algorithm is well suited for large-scale problems due to the simplicity of the computations involved and the extremely moderate storage requirements. Unfortunately the CG algorithm is only applicable to functions with positive definite Hessians; it is highly unstable when applied to functions with Hessians that are not positive definite (Madyastha & Aazhang, 1994).

2.8.3 Hybrid Methods

Second order methods are far superior in terms of learning time when compared to standard backpropagation, but they are more likely to get stuck in local minima. Hybrid methods such as *trust-region methods* try to combine both these approaches in a single algorithm. A trust-region is a region within which we can "trust" the quadratic approximation to the objective function.

2.8.3.1 Levenberg-Marquadt Algorithm

The Levenberg-Marquadt algorithm is an approximation to Newton's method (Hagan & Menhaj, 1994). Suppose we have a function $E(\mathbf{w})$ which we want to minimise with respect to the vector $\mathbf{w}$, then Newton's method gives a weight update

$$\Delta \mathbf{w} = -\mathbf{H}^{-1}\mathbf{g} \tag{2.48}$$

where $\mathbf{H}$ is the Hessian matrix and $\mathbf{g}$ is the gradient.

If we assume that $E(\mathbf{w})$ is the sum of squares function, then it can be shown that

$$\nabla E(\mathbf{w}) = \mathbf{J}^T(\mathbf{w})\,\mathbf{e}(\mathbf{w}) \tag{2.49}$$

$$\nabla^2 E(\mathbf{w}) = \mathbf{J}^T(\mathbf{w})\,\mathbf{J}(\mathbf{w}) + \mathbf{S}(\mathbf{w}) \tag{2.50}$$

where $\mathbf{J}(\mathbf{w})$ is the Jacobian matrix

$$\mathbf{J}(\mathbf{w}) = \begin{bmatrix} \frac{\partial e_1(\mathbf{w})}{\partial w_1} & \frac{\partial e_1(\mathbf{w})}{\partial w_2} & \cdots & \frac{\partial e_1(\mathbf{w})}{\partial w_v} \\ \frac{\partial e_2(\mathbf{w})}{\partial w_1} & \frac{\partial e_2(\mathbf{w})}{\partial w_2} & \cdots & \frac{\partial e_2(\mathbf{w})}{\partial w_v} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_N(\mathbf{w})}{\partial w_1} & \frac{\partial e_N(\mathbf{w})}{\partial w_2} & \cdots & \frac{\partial e_N(\mathbf{w})}{\partial w_v} \end{bmatrix} \tag{2.51}$$

and

$$\mathbf{S}(\mathbf{w}) = \sum_{i=1}^{N} \mathbf{e}_i(\mathbf{w}) \nabla^2 \mathbf{e}_i(\mathbf{w}) \tag{2.52}$$

For the *Gauss-Newton method*, it is assumed that $\mathbf{S}(\mathbf{w}) \approx 0$, and the update (2.48) becomes

$$\Delta \mathbf{w} = \left[\mathbf{J}^T(\mathbf{w})\mathbf{J}(\mathbf{w})\right]^{-1} \mathbf{J}^T(\mathbf{w})\mathbf{e}(\mathbf{w}) \tag{2.53}$$

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

$$\Delta \mathbf{w} = \left[\mathbf{J}^T(\mathbf{w})\mathbf{J}(\mathbf{w}) + \mu \mathbf{I}\right]^{-1} \mathbf{J}^T(\mathbf{w})\mathbf{e}(\mathbf{w}) \tag{2.54}$$

where $\mathbf{I}$ is the v x v identity matrix and μ is a variable parameter.

The parameter μ is multiplied by some factor (β) whenever a step would result in an increased $E(\mathbf{w})$. When a step reduces $E(\mathbf{w})$, μ is divided by β. When μ is large the algorithm becomes steepest descent (with step $1/\mu$), while for small μ the algorithm reduces to the Gauss-Newton (second order) update in (2.53). The Levenberg-Marquadt algorithm can be considered a trust-region modification to Gauss-Newton (Hagan & Menhaj, 1994; van der Smagt, 1994). For neural networks, the terms in the Jacobian matrix can be computed by a simple modification to the BP algorithm.

Summary of Levenberg-Marquadt algorithm

1. Present all inputs to the network and compute the corresponding network outputs and errors. Compute the sum of squares of errors ($E(\mathbf{w})$).
2. Compute the Jacobian matrix.
3. Solve (2.54) to obtain $\Delta \mathbf{w}$.
4. Recompute the sum of squares of errors using $\mathbf{w} + \Delta \mathbf{w}$.
   If this sum of squares is smaller that computed in step 1, then reduce μ by β, let $\mathbf{w} = \mathbf{w} + \Delta \mathbf{w}$, and go back to step 1. If the sum of squares is not reduced, then increase μ by β, and go back to step 3.
5. The algorithm is assumed to have converged when the norm of the gradient is less than some predetermined value, or when the sum of squares has been reduced to some error goal.

2.8.4 Direct Methods

Another approach is based on a direct determination of the matrices of weights, by solving in a classical or in the least-squares sense a set of systems of linear equations. The main advantages of this approach are that there is no risk of getting trapped in local minima during training and that the weights adapt quickly. The fundamental idea of these "direct methods" lies in an extension and a generalisation of the singular-value decomposition (SVD) algorithm for the "one-shot" evaluation of the matrix of weights.

Examples of direct methods are FBFBK (Barmann & Biegler-Konig, 1992), named after the authors, iterative conjugate gradient singular-value decomposition (ICGSVD) (Di Martino et al., 1993) and least-squares backpropagation (LSB) (Barmann & Biegler-Konig, 1993).

Analysis of these various methods has shown that for middle-size networks (several hundred neurons) these methods are competitive in terms of computation time with the best BP methods for MLP networks. For larger networks these methods are generally too expensive (Di Martino et al., 1996).

Verma uses hybrid algorithms that combine direct solution with other methods, where the output layer weights are directly solved using the modified Gram-Schmidt algorithm (Verma, 1997). He proposes three different ways of training the hidden layer weights, including using standard backpropagation (BP). The direct solution of the outputs was able to speed up training considerably and avoid getting stuck in local minima, even with BP training of the hidden weights.

2.8.5 Stochastic Methods

As mentioned previously, stochastic methods update the weight vector with a random vector as given in Eq. (2.13). An example is the stochastic Random Optimisation Method (ROM) algorithm given in (Schalkoff, 1997, p.208). The weight update is given by

$$\mathbf{w}(n+1) = \begin{cases} \mathbf{w}(n)+\xi & \text{if } E(\mathbf{w}(n)+\xi) < E(\mathbf{w}(n)) \\ \mathbf{w}(n)-\xi & \text{if } E(\mathbf{w}(n)+\xi) \geq E(\mathbf{w}(n)) \text{ and } E(\mathbf{w}(n)-\xi) < E(\mathbf{w}(n)) \\ \mathbf{w}(n) & \text{otherwise} \end{cases} \quad (2.55)$$

2.9 Adaptive Structures

Up to this point, it has been assumed that the neural network being trained has a fixed structure. The question then arises "What is the optimum size of the network?". As mentioned in Section 2.6.6, having too many neurons can result in overfitting, whereas too few may not allow the network to reach the desired performance level. Designers have to use prior knowledge on the problem, their experience, or just trial-and-error, in order to find a structure that performs well, let alone an 'optimal' structure. "The exhaustive search over the space of network architectures is computationally infeasible even for networks of modest size" and hence "the use of heuristic strategies that dramatically reduce the search complexity" (Karampiperis et al., 2002).

There are two opposing approaches for adaptive strategies: *Constructive* and *destructive*. Constructive methods start with a minimal network, even a single neuron, then "grow" the network as needed by adding new connections, nodes or layers. Destructive methods, also known as *pruning algorithms*, on the other hand, start with a complex structure and remove unnecessary connections, nodes and layers during training (Mitchell, 1997; Yao & Liu, 1997). A frequently used constructive algorithm is the Cascade-Correlation algorithm (Fahlman & Lebiere, 1990), while well known pruning algorithms include Optimal Brain Damage (OBD)(LeCun et al., 1990) and Optimal Brain Surgeon (OBS) (Hassibi & Stork, 1993). A partial review of constructive algorithms is given in (Fiesler, 1994), while one of pruning algorithms is given in (Reed, 1993).

Investigating methods for adaptively determining network structures is an active area of research, with researchers combining constructive and pruning algorithms or creating new variations (Islam et al., 2000; Islam et al., 2003; Karampiperis et al., 2002; Rivals & Personnaz, 2003; Thivierge et al., 2003; Tsai & Lee, 2004). Evolutionary computation, which includes genetic algorithms and evolutionary programming, has also been commonly used for this purpose (Garcia-Pedrajas et al., 2003; Leung et al., 2003; Nikolaev & Iba, 2003; Yao & Liu, 1997).

Adaptive structures are not used in this thesis, but the work presented here provides scope for future work in this area. The concept has therefore been introduced here with a brief overview. Relevant references have been included for the interested reader.

2.10 Conclusion

An overview of artificial neural networks has been presented in this chapter, starting from the biological roots to various artificial neural network structures, applications and training algorithms.

The information presented here shows that the massive parallel processing demonstrated by biological brains has inspired the creation of a versatile and powerful computational tool. The field of artificial neural networks is a diverse one, both in terms of the kinds of networks and algorithms as well as their numerous applications. In such a large and growing field, there is still room for much work in exploring new models and paradigms, and such work is ongoing. The work presented in this thesis represents one such exploration, and it is hoped that it will not only yield interesting results and discoveries, but also open new areas of continued research.

In order to maintain focus, the emphasis has been on feedforward networks, supervised training techniques and other topics relevant to the subsequent chapters. Brief descriptions and overviews of cognate areas have been presented where appropriate.

The biophysical mechanism of shunting inhibition, in biological neurons, has been introduced briefly, but the use of shunting inhibition in artificial neural networks has been deferred to the next chapter, where it will be developed in greater detail.

Chapter 3

Shunting Inhibitory Artificial Neural Networks

3.1 Introduction

Shunting inhibition is a powerful computational mechanism that plays an important role in sensory information processing systems. Since it was proposed as a plausible physiological model in the early 1960's (Furman, 1965; Lettvin, 1962), shunting inhibition has been extensively used to model some important visual and cognitive functions. For example, Grossberg used it to model long-term and short-term memory mechanisms, feature detection, and other cognitive functions (Grossberg, 1973, 1976, 1988). Fukushima employed it for local feature detection as part of the *neocognitron* (Fukushima et al., 1983). Pinter used it to model the adaptation phenomena in receptive field organization and modulation transfer function (Pinter, 1983, 1984, 1985). Bouzerdoum and Pinter proposed a model of motion detection in insects based on shunting inhibition (Bouzerdoum, 1993; Bouzerdoum & Pinter, 1989, 1992). They also introduced a *shunting inhibitory cellular neural network* and used it to model receptive field profiles of neurons in the early parts of the visual system (Bouzerdoum & Pinter, 1993). Other researchers have also used shunting inhibition, including some of its VLSI implementations (Darling & Dietze, 1993; Moini et al., 1997; Nabet, 1992; Nilson et al., 1994; Wolpert & Micheli-Tzanakou, 1993).

Despite their widespread use in modelling psychophysical, neurophysiological and cognitive phenomena, to our knowledge, shunting inhibitory networks have not been used in supervised pattern classification or function approximation, other than

in the neocognitron (Fukushima et al., 1983) and ART networks (Carpenter & Grossberg, 1988). Other than these, shunting networks have primarily been part of adaptive (self-organising) systems that use competitive learning. Cellular neural networks based on shunting inhibition have shown great promise as information processors in vision and image processing tasks (Beare & Bouzerdoum, 1999; Cheung et al., 1999; Pontecorvo & Bouzerdoum, 1995, 1997), but they have not been used for classification and regression tasks before 1999. One of the main reasons for this has been the lack of proper training algorithms. The expert knowledge of the designer has had to be used to choose the connection weights based on the task at hand. This does not allow complex pattern recognition problems to be handled, resulting in limited applications. Another reason is that the operation of a shunting inhibitory cellular neural network (SICNN) is governed by a system of nonlinear differential equations, which must be solved in order to obtain the output of the network for a given input pattern.

It is only recently that Bouzerdoum proposed an artificial neural network architecture, based on shunting inhibition, that can be trained to perform pattern classification or function approximation; he named it *shunting inhibitory artificial neural network* (SIANN) (Bouzerdoum, 1999, 2000). Derived from SICNNs, SIANNs are feedforward networks that operate using the steady-state solution of the set of ordinary differential equations governing the dynamics of the shunting networks, thereby avoiding the need to obtain a numerical solution for the differential equations. This allows the network to operate in a static mode like *multilayer perceptrons* (MLPs). The idea was to exploit the inherent nonlinearity of shunting inhibition to develop powerful, trainable networks, with nonlinear decision surfaces, for classification, nonlinear regression and pattern association.

This chapter presents the development of SIANNs from its biological roots to the shunting neuron model and SIANN architecture. The next section explains the biological roots of shunting inhibition. The third section presents the electrical circuit approximation of a patch of dendritic membrane and the derivation of the differential equations that govern the shunting inhibition model. The fourth section describes the precursor to SIANNs, Shunting Inhibitory Cellular Neural Networks. This is followed by the development of the feedforward shunting inhibitory neuron model and the SIANN architecture in Sections 3.5 and 3.6, respectively. Section 3.7 illustrates the non-linear decision boundaries of the shunting inhibitory neuron model. This is followed by the conclusion.

3.2 Shunting Inhibition in Biological Systems

In a biological neuron, the cell at rest has a potential difference across the cell membrane due to the difference in ionic concentrations on either side of the membrane. The cell membrane consists of a thin, semi-permeable bilayer of lipids and is a near perfect electrical insulator. At equilibrium, the concentration of sodium (Na^+) ions is higher in the extracellular fluid compared to within the cell and this difference in concentration causes a Nernst potential (or reversal potential) E_{Na} of about +50 mV (Gerstner & Kistler, 2002). The concentration of potassium (K^+) ions, on the other hand, is higher inside the cell than outside with a reversal potential E_K of approximately –77 mV. Both these and other ion types are simultaneously present and contribute to the resting potential across the membrane, V_r, of approximately -65 mV. Since $E_K < V_r < E_{Na}$, at the resting potential potassium ions flow out of the cell and sodium ions flow into it. Active ion pumps in the cell membrane pump these ions in the reverse direction in order to maintain a dynamic equilibrium.

An input at an excitatory synapse reduces the negative polarisation of the membrane, also called a depolarising potential. Conversely, an input at an inhibitory synapse increases the negative polarisation of the membrane, called hyperpolarizing potential, caused by positive ions (usually potassium) moving out of the cell (Stevens, 1994). If the sum of postsynaptic potentials causes the membrane voltage to cross a threshold value, the cell body produces an action potential that propagates down the axon of the neuron. The inhibitory mechanism described here therefore is additive (subtractive) inhibition.

Inhibition can be mediated by both pre- and post-synaptic contacts. Post-synaptic inhibition functions to reduce the excitability of the target cell by increasing the permeability of the post-synaptic membrane to chloride (Cl^-) and potassium (K^+) ions, thereby increasing the ionic conductance of their respective channels (Nicholls et al., 1992). In shunting inhibition, where the synaptic activity opens mostly Cl^- channels, the reversal potential of the inhibitory synapses is equal or very close to the membrane resting potential (Faber & Korn, 1982). These inhibitory inputs therefore have hardly any effect on the membrane potential if the neuron is at rest. The effect of the shunting inhibitory inputs is to increase the local conductivity of the cell, allowing the ions to flow in or out of the cell, depending on the state of the neuron. If the neuron is depolarised, then the inhibitory inputs result in inhibitory postsynaptic potentials. If the neuron is hyperpolarised, however, the inhibitory input results in a depolarising potential. The net effect is to 'clamp' the membrane potential to the resting potential by 'shunting' the effects of other synaptic potentials. Inhibitory synapses are often located on the soma or the shaft of the dendritic tree. This strategic positioning allows a few inhibitory spikes to 'shunt' the whole input gathered by the dendritic tree from all the synaptic inputs (Gerstner & Kistler, 2002).

3.3 Development of the Shunting Inhibitory Model

Shunting inhibitory neural networks are based on a neuron model that is inspired by human and animal visual systems. The equivalent circuit is derived from a lumped parametric approximation of a uniform patch of dendritic membrane as shown in Fig. 3.1 (Bouzerdoum & Pinter, 1993). The circuit consists of the ordinary or nonsynaptic membrane in parallel with the excitatory and inhibitory pathways. The resting potentials and conductances of the different ionic channels are lumped together in the resting potential V_r and the resting conductance g_r. These two, in parallel with the membrane capacitance C_m, represent the nonsynaptic membrane. Each synaptic pathway, excitatory or shunting inhibitory, consists of a synaptic potential (battery), V_e and V_s, in series with a synaptic conductance, g_e and g_s, respectively. V_m is the total membrane potential; i_s, i_e, i_r and i_c represent the ionic currents, and R is the receptor region feeding the excitatory synaptic inputs.

The conductances of the excitatory and inhibitory ionic channels are zero at rest under this representation. The excitatory input synapses control the conductance g_e with a reversal potential $V_e > V_r$. On the other hand, the inhibitory input synapses are assumed to be of the shunting type. As decribed in previous section, in shunting inhibition the synaptic activity opens mostly Cl^- channels whose reversal potential is equal or very close to the membrane resting potential; the role of shunting inhibition being to "clamp" the cell to its resting potential. Here it is assumed that shunting inhibition is mediated by modulating the conductance g_s, with equilibrium potential equal to the resting potential, i.e. $V_s = V_r$.

The node equation of the equivalent circuit shown in Fig. 3.1 is

$$C_m \frac{dV_m}{dt} + g_e(V_e + V_m) - g_r(V_r - V_m) - g_s(V_s - V_m) = 0 \tag{3.1}$$

Rearranging the terms, this can be written as

$$\frac{dV_m}{dt} = -\frac{g_e}{C_m}(V_e + V_m) + \frac{g_r}{C_m}(V_r - V_m) + \frac{g_s}{C_m}(V_s - V_m) \tag{3.2}$$

If the deviation of the membrane voltage V_m from the resting potential V_r is designated ΔV (ie. $\Delta V = V_r - V_m$) then, by using the fact that $V_s = V_r$, the equation describing the change in membrane potential V_m relative to the resting potential V_r is given by

$$\frac{d\Delta V}{dt} = \frac{g_e}{C_m}(V_e + V_m) - \frac{g_r}{C_m}(\Delta V) - \frac{g_s}{C_m}(\Delta V) \tag{3.3}$$

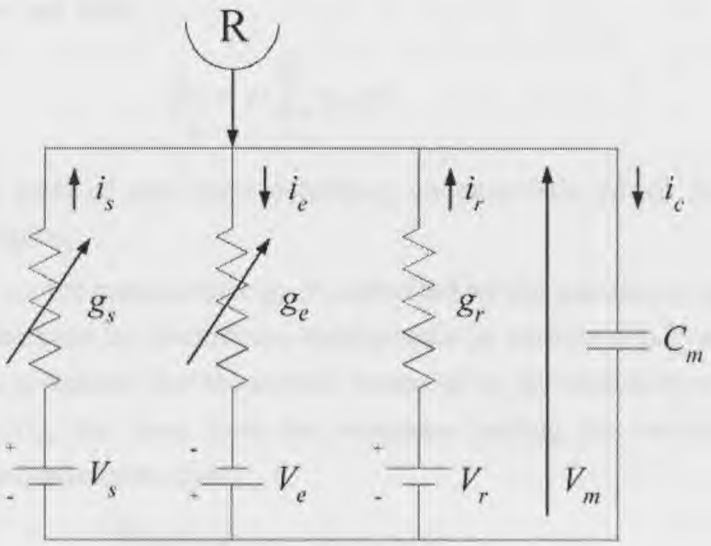


Fig. 3.1: Electrical equivalent circuit of a cell

Depolarisation diminishes the membrane voltage, thereby increasing V_m; therefore, the quantity ΔV is nonnegative definite ($\Delta V \geq 0$), provided the equilibrium potential of the inhibitory channels is equal to the resting potential ($V_s = V_r$).

Shunting inhibition attempts to 'clamp' the membrane potential to its resting potential by changing the conductance of specific ionic currents. Therefore, the influence of inhibitory synapses in Fig. 3.1 is to increase the conductance g_s, thereby shunting the excitation-induced current. To see this, let I_{ex} be the sum of the leakage current and the current induced by the excitatory influences ($I_{ex} = i_c + i_e$). It can be shown that

$$i_r = \frac{I_{ex}}{1 + g_s/g_r}, \qquad \text{and} \qquad i_s = \frac{I_{ex}}{1 + g_r/g_s}$$

Therefore, i_r and i_s are, respectively, monotonically decreasing and increasing functions of g_s. An increase in g_s will divert part of the current from i_r to i_s. This is what gives rise to the name *"shunting inhibition"*.

The basic assumption is that the biophysical mechanism underlying shunting inhibition is the control of conductance g_s by the voltage of neighbouring cells (or cell subunits). In the compartmental approach, each compartment represents an electrically independent cell (or cell subunit) equivalent to the circuit of Fig. 3.1. Let x_j be the voltage deviation from the resting potential, ΔV, in the jth unit or compartment. Since each compartment has independent interactions with its neighbouring cells, the voltage-controlled conductance is merely the resultant of the various synaptic interactions, $c_{ji}x_i$. Assuming the synaptic junctions are close to each other, hence ignoring the resistive path between adjacent synapses, the shunting conductance is the parallel combination of individual synaptic conductances.

Therefore, we can write

$$\frac{g_s}{C_m} = f(\sum_{i=1}^{n} c_{ji}x_i) \tag{3.4}$$

where f is some kind of non-linear saturating characteristic which limits the total shunting conductance.

In contrast to g_s, the conductance g_e is controlled by the excitatory input synapses which work to increase the membrane conductance to sodium (Na^+) and potassium (K^+) ions. If it is postulated that the current produced in the excitatory channels, i_e, is proportional to $I_j(t)$, the input from the receptors feeding the excitatory synaptic inputs in the jth compartment, then

$$\frac{g_e}{C_m}(V_e+V_m) = \frac{i_e}{C_m} = I_j(t) \tag{3.5}$$

Furthermore, identifying the remaining constant term in the right hand side of (3.3) as

$$\frac{g_r}{C_m} = a_j \tag{3.6}$$

then Equation (3.3) becomes

$$\frac{dx_j}{dt} = I_j(t) - a_j x_j - f(\sum_i c_{ji}x_i)x_j, \qquad i = 1,2,...,n \tag{3.7}$$

The system of coupled nonlinear differential equations given by (3.7) describes the activity of recurrent neural network.

3.4 Shunting Inhibitory Cellular Neural Networks

In *shunting inhibitory cellular neural networks* (SICNNs), the neurons (or cells) are arranged in a two-dimensional grid as shown in Fig. 3.2. Each neuron has a single external excitatory input, which is not shown in the figure but can be assumed to be coming perpendicular to the page. The weighted outputs of the neurons in a predefined neighbourhood are fed back as shunting inhibitory inputs and passed through the nonlinear activation function. Let C_{ij} represent the cell (neuron) at position (i,j) in the lattice. The activity of a cell is governed by the non-linear differential equation:

$$\frac{dx_{ij}}{dt} = I_{ij}(t) - a_{ij}x_{ij} - f(\sum_{C_{kl}\in N_r(i,j)} c_{ij}^{kl}x_{kl})x_{ij} + b_{ij} \tag{3.8}$$

where x_{ij} is the activity of cell C_{ij} , I_{ij} is the external input to cell C_{ij}, a_{ij} is the passive decay rate of the cell activity (positive constant), c_{ij}^{kl} is the connection weight from cell C_{ij} to cell C_{kl}, $N_r(i,j)$ is the neighbourhood of cell C_{ij}, b_{ij} is a constant bias, and f is a positive non-decreasing activation function.

Equation (3.8) describes the activity of a recurrent (or feedback) network, where the activity of a neuron is dependent on the outputs of its neighbouring neurons, and, in turn, its output is used as an input by the neighbouring neurons. Equation (3.8) is a two-dimensional version of (3.7); it only differs in that a bias term, b_{ij}, has also been added. In the 2-dimensional structure shown in Fig. 3.2, the neighbourhood covers the eight immediately adjacent neurons. The inhibitory feedback weights depend on the relative positions of the neighbourhood neurons. More detailed descriptions of SICNNs, including stability analysis, can be found in articles by Bouzerdoum and Pinter (Bouzerdoum, 1992; Bouzerdoum & Pinter, 1993).

SICNNs have been successfully applied to vision and image processing tasks (Beare & Bouzerdoum, 1999; Cheung et al., 1999; Pontecorvo & Bouzerdoum, 1995, 1997), but they have not been used for classification or regression tasks. One of the main reasons for this has been the lack of proper training algorithms. The applications have generally used fixed weight matrices, defined using prior knowledge of the task at hand. This will not suffice for complex pattern recognition problems. Development of training algorithms for these networks is fairly complex as they are recurrent networks and the activity of the neurons is governed by a system of coupled non-linear differential equations.

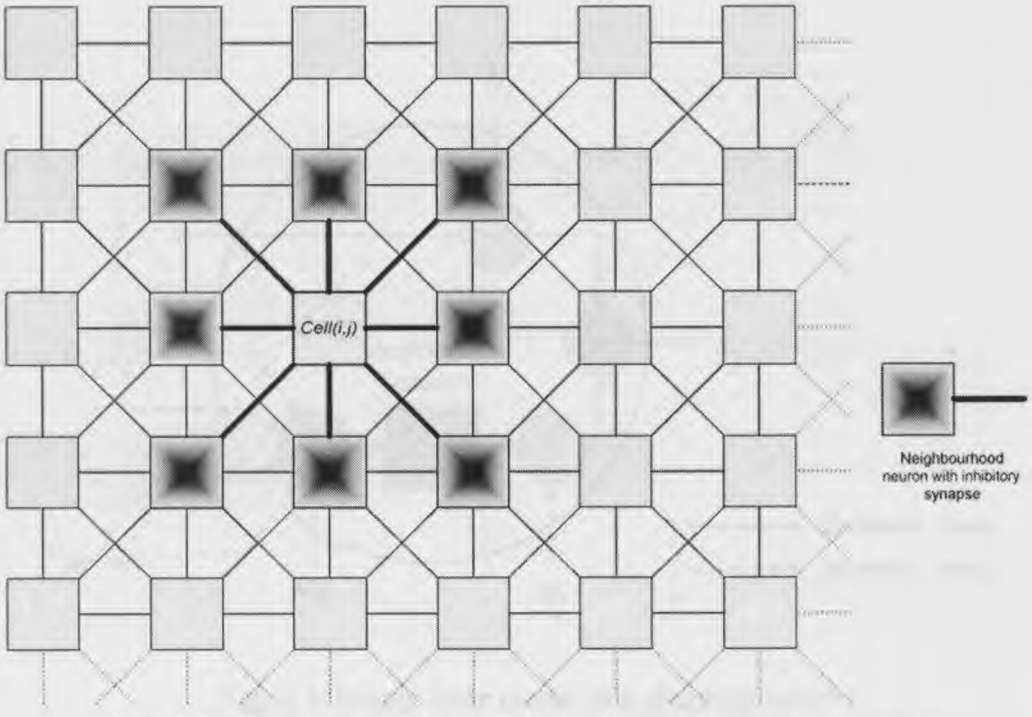


Fig. 3.2: Shunting Inhibitory Cellular Neural Network structure

3.5 Feedforward Shunting Inhibitory Neuron Model

In order to apply shunting inhibitory networks to classification and regression problems, Bouzerdoum proposed modifying the neuron model so that it operates in a feedforward mode (Bouzerdoum, 1999, 2000). To operate the shunting inhibitory neurons in feedforward mode, the feedback loops are removed, and the inhibitory inputs are the weighted *external inputs* (I_i) of all the neurons. The neurons are now arranged in a layer (or layers) instead of in a grid pattern.

Based on these modifications, the differential equation in (3.7), which describes the activity of jth neuron in a recurrent network, becomes

$$\frac{dx_j}{dt} = I_j - a_j x_j - f(\sum_i c_{ji} I_i) x_j + b_j \tag{3.9}$$

In order to study the behaviour of this network in a static mode, the behaviour of the neuron is modelled by its steady state response. Using the steady state solution, rather than attempting to solve a set of differential equations, simplifies the analysis and implementation of the network.

The steady-state solution of (3.9) is given by

$$x_j = \frac{I_j + b_j}{a_j + f\left(\sum_i c_{ji} I_i\right)} \tag{3.10}$$

This is shown diagrammatically in Fig. 3.3.

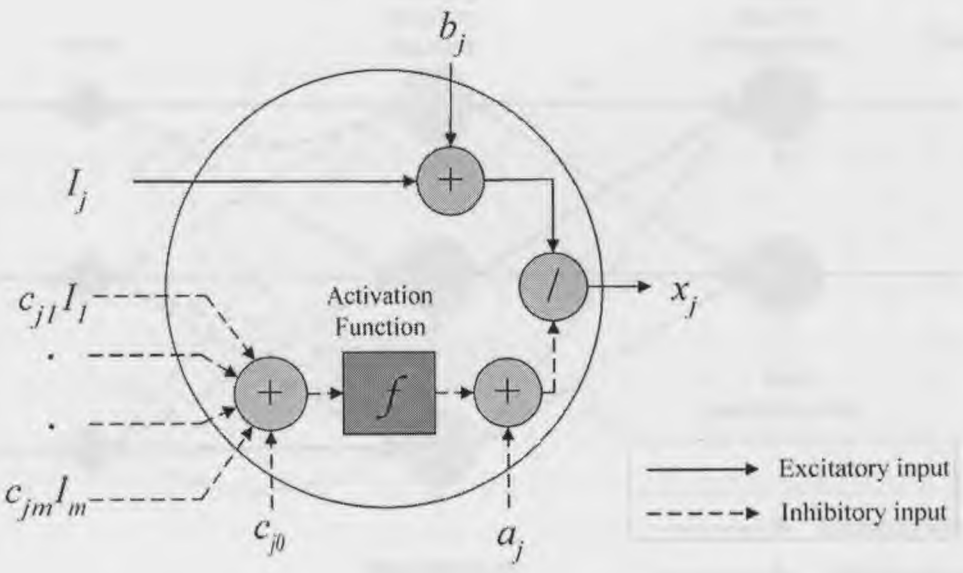


Fig. 3.3: Steady-state model of a shunting neuron.

We define the denominator in (3.10) as the shunting term for the jth neuron, s_j

$$s_j \equiv a_j + f\left(\sum_i c_{ji} I_i\right) \tag{3.11}$$

The original definition, given in (3.8), places constraints that the term a_j be a positive constant and that the activation function f be a positive non-decreasing function. These constraints are now relaxed, and replaced by the constraint that s_j has to be positive so as not to encounter a divide by zero error, i.e. $s_j > 0$.

3.6 Feedforward Network Structure

The shunting inhibitory neuron described in the previous section has a single unweighted excitatory input and is part of a layered network. The feedforward shunting network was originally proposed as a fully connected structure (Arulampalam & Bouzerdoum, 2000; Bouzerdoum, 1999); that is, each input is fed directly into one shunting neuron as an excitatory input, whereas all the inputs are weighted and fed in as inhibitory inputs to all neurons in the layer (refer Fig. 3.4). Therefore, *the fully connected SIANN structure has as many shunting neurons as there are inputs*. Subsequent layers would also need to be of the same size, for full excitatory propagation of the signals.

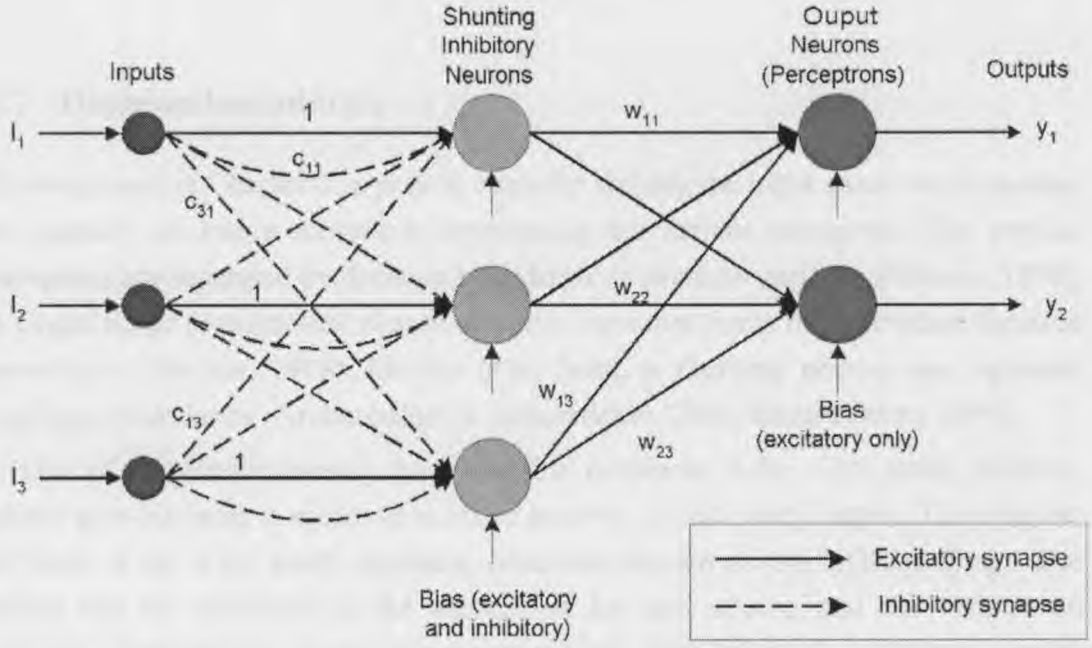


Fig. 3.4 : Feedforward Shunting Inhibitory Artificial Neural Network structure

In most problems, however, the number of outputs is different from the number of inputs. In order to create a structure that would result in the correct number of outputs, the output layer was set to consist of the required number of linear or sigmoidal (perceptron-type) neurons. The output neurons are able to sum the outputs of the shunting neurons to produce the final output of the network.

This network structure of a layer (or layers) of shunting inhibitory neurons with a layer of output neurons is called a *Shunting Inhibitory Artificial Neural Network* (SIANN). Fig. 3.4 shows a SIANN with a single layer of 3 shunting inhibitory neurons connected to 2 output neurons.

The output of the kth output neuron is given by

$$y_k = g(\sum_{j=0}^{m} w_{kj} x_j) = g(v_k) \tag{3.12}$$

where g is the output layer activation function; w_{kj} is the connection weight from jth shunting neuron to the kth output neuron; w_{k0} is the bias of the output neuron connected to a fixed 'input', $x_0 = 1$, and v_k is the net input to the activation function g:

$$v_k = \sum_{j=0}^{m} w_{kj} x_j \tag{3.13}$$

The output layer activation function can be a simple linear function that just sums the inputs, or a sigmoid function. This structure can now be applied to problems with any combination of number of inputs and required outputs.

3.7 Decision boundaries

As mentioned in Chapter 2, a pattern classifier divides the input space into a number of mutually exclusive subspaces representing the various categories. The various subspaces are separated by decision boundaries or *decision surfaces* (Nilsson, 1990). A single linear or sigmoidal neuron can only represent linear or hyperplane decision boundaries (Haykin, 1999). On the other hand, a shunting neuron can represent nonlinear boundaries (Arulampalam & Bouzerdoum, 2000; Bouzerdoum, 1999).

One of the classic linearly non-separable problems is the n-bit parity problem, where an n-bit input is meant to produce an even or odd parity output. The simplest of these is the 2-bit parity problem, otherwise known as the XOR problem. The inputs can be visualised as the vertices of the unit square, and the vertices on opposite sides of the square belong to the same class. A single perceptron cannot

solve this linearly non-separable problem (Haykin, 1999), but a single shunting neuron can (Arulampalam & Bouzerdoum, 2001a, 2002a), as shown in Fig. 3.5.

Fig. 3.6 shows an example of non-linear decision boundary obtained by training a single shunting neuron on the synthetic two-class problem obtained from (Ripley, 1996) along with the 250 data points used for training. The standard shunting neuron by definition can only have a single excitatory input; thus, for both these problems only one input was fed in as excitatory, and both inputs are fed in as inhibitory.

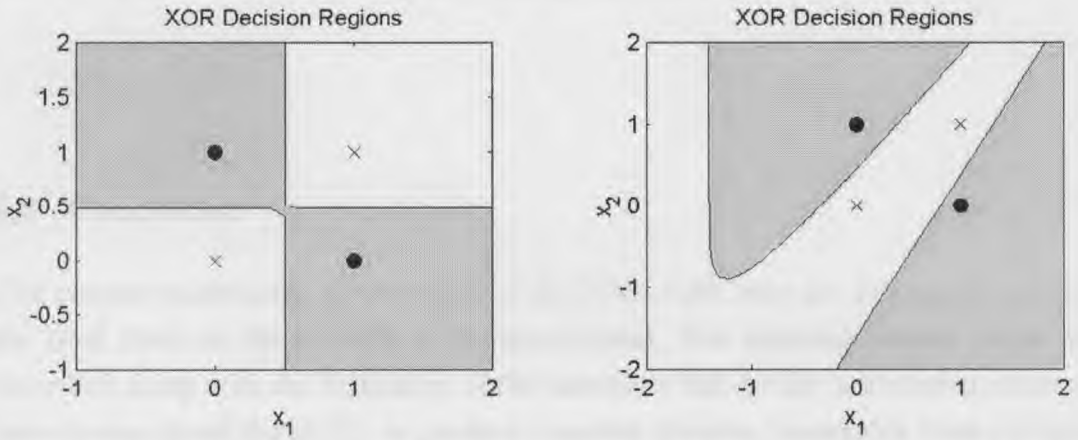


Fig. 3.5: Examples of decision boundaries formed by single shunting inhibitory neuron solving XOR problem

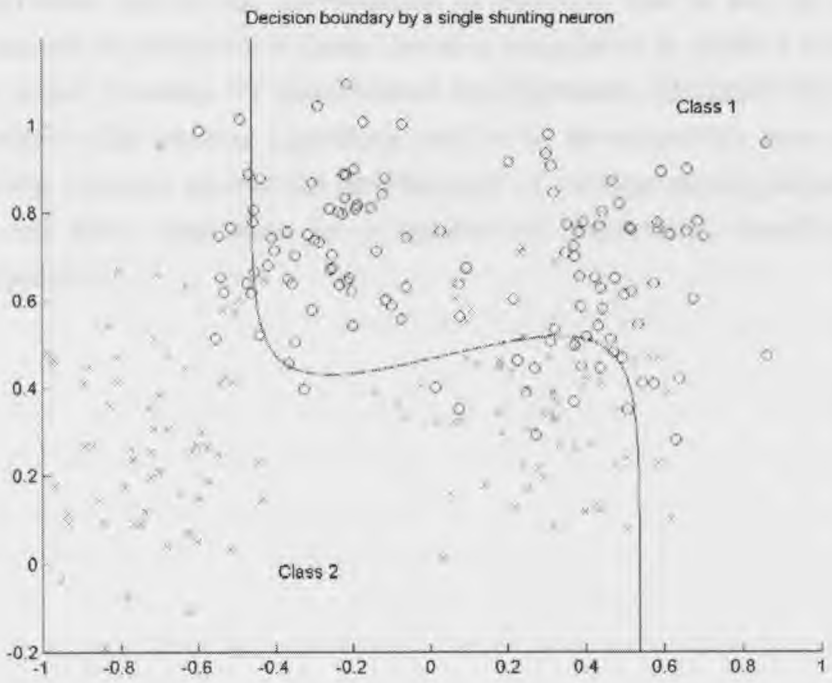


Fig. 3.6: Decision boundary formed by single shunting inhibitory neuron trained on Ripley's synthetic 2-class problem

Despite the fact that a perceptron is only able to represent hyperplane decision boundaries, MLPs with a single hidden layer can approximate any given continuous function on any compact subset to any degree of accuracy, provided that a sufficient number of hidden layer neurons are used (Hornik et al., 1989). As explained in Chapter 2, this is because of the non-linear transformations performed by the hidden layer neurons. SIANNs should therefore be able to represent complex nonlinear decision surfaces more efficiently than MLP networks, by leveraging the inherent non-linear capability of shunting neurons demonstrated here. This is the major motivating factor for introducing the shunting inhibitory neuron.

3.8 Conclusion

This chapter outlines the development of SIANNs, right from the biological roots to the final form of the network to be investigated. The shunting neuron model is described along with the derivation of the equations that define it. Shunting neurons have demonstrated the ability to produce complex decision boundaries from a single neuron. This compares favourably with the perceptron, which can only produce linear decision boundaries. This in turn indicates that SIANNs should be able to represent complex nonlinear decision surfaces more efficiently than MLP networks.

The motivation behind the investigation of SIANNs was to use the ability of shunting neurons to produce non-linear decision boundaries to create a new class of high-order neural networks for classification and regression (Bouzerdoum, 1999). In order to achieve this, training algorithms need to be developed for these networks. The following chapters present the development of various training algorithms for SIANNs, and their application to a number of benchmark classification and regression problems.

Chapter 4

Development of Training Algorithms

4.1 Introduction

The previous chapter outlines the motivation and development of the SIANN architecture. As mentioned in the previous chapter, one of the limitations faced by the cellular form, SICNN, was the lack of training algorithms. In order to apply SIANNs to classification and regression problems, training algorithms needed to be developed. This chapter describes the development of a number of training algorithms for SIANNs.

The training algorithms developed are broadly divided into *gradient-based, direct solution* and *stochastic* methods. The gradient-based algorithms are described in the next section. The third and fourth sections describe the direct solution and stochastic algorithms, respectively. Section 4.5 describes the experimental methods used to test the performance of networks trained using the developed algorithms, covering network structures, initialisation methods, and evaluation criteria. It also describes the various benchmark problems on which SIANNs are trained and tested. This is followed by experimental results, presented in Section 4.6. This section contains an investigation into the effect the shunting term has on training performance, as well as the actual test results. The final section contains the conclusion. The derivation of the training equations for the gradient-based algorithms is shown in Appendix A and selected tables of results are presented in Appendix B.

4.2 Gradient-based Algorithms

This section describes the various training algorithms developed for SIANNs that use the gradient of the objective function to update the weights. All the algorithms developed in this thesis are based on supervised learning using the error-correction learning rule (c.f. Chapter 2).

The network is presented with a set of exemplars in the form of pairs ($\mathbf{I}(q)$, $\mathbf{d}(q)$) where $\mathbf{I}(q)$ is the input vector and $\mathbf{d}(q)$ is the corresponding vector of desired values. The difference between the desired and the actual output of the network is the error vector, given by

$$\mathbf{e}(q) = \mathbf{y}(q) - \mathbf{d}(q) \tag{4.1}$$

where $\mathbf{y}(q)$ is the output vector due to the input $\mathbf{I}(q)$.

The algorithms developed operate in batch mode, where the whole set of exemplars is presented to the network before the weights are updated. The training algorithm seeks to minimise an objective function, E, which may be the sum squared error (SSE)

$$E = \tfrac{1}{2}\sum_{N} \mathbf{e}(q)^T \mathbf{e}(q) \tag{4.2}$$

or the mean squared error (MSE)

$$E = \frac{1}{2N}\sum_{N} \mathbf{e}(q)^T \mathbf{e}(q) \tag{4.3}$$

where N is the number of exemplars in the training set.

The gradient-based training algorithms developed here can be divided into two categories: the first-order gradient descent algorithm and its variants; and the Levenberg-Marquardt algorithm and its variants.

The Conjugate Gradient algorithm described in Chapter 2 was not implemented. The reason is that the shunting neuron decay parameter a has a lower bound imposed on it during training, in order to avoid division by zero (see equation (4.6)). The conjugate gradient algorithm requires the weight updates to be performed such that the current gradient update direction is always orthogonal to the previous gradients. Adjustments to the weight update of parameter a by the lower bound may violate this requirement, hence this algorithm was not implemented.

The following sub-sections describe the training algorithms implemented in more detail.

4.2.1 Gradient Descent

All the gradient descent-based algorithms implemented for SIANNs are based on the error-backpropagation (BP) algorithm (Rumelhart et al., 1986), described in Section 2.6.3. The standard gradient descent (GD) algorithm is a first-order algorithm that uses a fixed learning rate as in standard BP (refer Section 2.8.1). At the n^{th} training step the weight update is given by

$$\Delta \mathbf{w}(n) = -\eta \mathbf{g}(n) \tag{4.4}$$

where η is the learning rate and $\mathbf{g}(n)$ is the gradient given by

$$\mathbf{g}(n) = \frac{\partial E}{\partial \mathbf{w}(n)} \tag{4.5}$$

The backpropagation algorithm requires the partial derivatives of the objective error function, E, with respect to each of the parameters (weights) being updated to calculate the gradient.

The 'standard' SIANN is a feedforward neural network with a hidden layer of shunting neurons and an output layer of linear or sigmoid neurons. For the sake of clarity, the equations describing the operation of the SIANN, defined in Chapter 3, are presented again in Eqs. (4.6) to (4.8) below.

The output of the j^{th} shunting neuron, x_j, is given by

$$x_j = \frac{I_j + b_j}{a_j + f\left(\sum_{i=0}^{m} c_{ji} I_i\right)} \tag{4.6}$$

where I_j is the j^{th} input; a_j is the 'decay term'; b_j is the bias; c_{ji} is the synaptic weight connecting the j^{th} neuron to the i^{th} input; c_{j0} is the bias for the shunting activation function connected to a fixed 'input', $I_0 = 1$; and f is a non-decreasing activation function.

The output of the kth output neuron is given by

$$y_k = g\left(\sum_{j=0}^{m} w_{kj} x_j\right) \tag{4.7}$$

where g is the output layer activation function; w_{kj} is the connection weight from j^{th} shunting neuron to the k^{th} output neuron and w_{k0} is the bias of the output neuron connected to a fixed 'input', $x_0 = 1$.

The denominator in (4.6) is defined as the shunting term for the jth neuron, s_j

$$s_j \equiv a_j + f\left(\sum_{i=0}^{m} c_{ji} I_i\right) \tag{4.8}$$

This shunting term is constrained to be always positive, achieved by imposing a lower bound on the parameter a_j during the initialization and training phases.

The parameters to be trained in a standard SIANN, therefore, are the weights and biases of the output neurons (w_{kj}), the decay and bias terms of the shunting neurons (a_j and b_j) and the inhibitory weights of the input signals and shunting bias (c_{ji}). The partial derivatives of the error function with respect to these SIANN parameters are given in Eqs. (4.9) to (4.14) below (Refer to Appendix A for the full derivation of these equations).

The partial derivative of the error function, E, with respect to the synaptic weight connecting the k^{th} output neuron to the j^{th} shunting neuron, w_{kj}, is given by

$$\frac{\partial E}{\partial w_{kj}} = \delta_{ok} x_j \tag{4.9}$$

where δ_{ok}, known as the un-normalized error sensitivity, is given by

$$\delta_{ok} = e_k(q)\, g'(v_k) \tag{4.10}$$

and $e_k(q)$ is the output error for the qth training point, g is the output layer activation function and v_k is the net input to the activation function. For the bias term, w_{k0}, the input, x_0, is assumed fixed at 1.

The partial derivative with respect to the decay term of the j^{th} shunting neuron, a_j, is given by

$$\frac{\partial E}{\partial a_j} = -\delta_j \frac{\left(I_j + b_j\right)}{s_j^{\,2}} = -\delta_j \frac{x_j}{s_j} \tag{4.11}$$

with δ_j, the backpropagated error sensitivity for the jth shunting neuron, defined as

$$\delta_j = \sum_{k=1}^{n} \delta_{ok} w_{kj} \tag{4.12}$$

and s_j the shunting layer denominator as defined in equation (4.8).

The partial derivative with respect to the bias of the j^{th} shunting neuron, b_j, is

$$\frac{\partial E}{\partial b_j} = \frac{\delta_j}{s_j} \tag{4.13}$$

The partial derivative with respect to the shunting synaptic weight from the i^{th} input to the j^{th} shunting neuron, c_{ji}, is given by

$$\frac{\partial E}{\partial c_{ji}} = -\delta_j \frac{x_j}{s_j} f'(v_j) I_i \tag{4.14}$$

The shunting activation function bias, c_{j0}, is assumed to have a constant 'input' of 1.

4.2.2 Gradient Descent with Momentum

The gradient descent with momentum (GDM) is the GD algorithm with an additional momentum term, as described in Section 2.8.1. The weight update, $\Delta w(n)$, is given by

$$\Delta w(n) = -\eta g(n) + \alpha \Delta w(n-1) \tag{4.15}$$

where α is the momentum constant.

4.2.3 Gradient Descent with Adaptive Learning Rate and Momentum

The speed of convergence and success rate of the gradient-descent based algorithms have previously been shown to depend heavily on the learning rate (Magoulas et al., 1999). To avoid the trial-and-error method of determining the optimal learning rate, an adaptive learning rate strategy was developed, called *Gradient Descent with Adaptive learning rate* (GDA). The method used increases the learning rate at successive iterations unless the error grows beyond a certain ratio to previous step, an adaptation of the "Bold Driving" method (Battiti, 1989; Demuth & Beale, 1992) described in Section 2.8.1. The next step was to incorporate a momentum term, resulting in the *Gradient Descent with Adaptive Learning Rate and Momentum* (GDX) algorithm (Demuth & Beale, 1992). The only difference in the algorithms is that the GDA weight update uses (4.4), whereas GDX uses (4.15).

Summary of the GDA/GDX algorithm

1. Determine initial squared error, E_0
2. Select initial learning rate, η_0, and calculate a new weights using (4.4) / (4.15)
3. Calculate the new squared error, E_{new}
   a. If $E_{new} / E_{old} \leq \delta E_{max}$
      (δE_{max} is usually set slightly above 1 (e.g. 1.04) to allow training to get out of shallow local minima)
      i. If $E_{new} / E_{old} < 1$, set $\eta_{new} = \beta \eta_{old}$ where $\beta > 1$ (typically 1.05)
      ii. Calculate the weight change using (4.4) / (4.15), and update
   b. If $E_{new} / E_{old} > \delta E_{max}$,
      i. Set $\eta_{new} = \gamma \eta_{old}$, where γ less than 1 (typically 0.7)
      ii. Set the weight change, $\Delta w = -\eta_{new} g$ and update weights
4. Go back to 3.

The weight update in Step 3(b)(ii) is the same for both algorithms, meaning that if the error increases more than the limit, the algorithm discards all momentum information and updates the weights using only the gradient at that point.

4.2.4 Levenberg-Marquardt (LM) algorithm

The Levenberg-Marquadt (LM) algorithm is a second-order trust-region algorithm, described in Section 2.8.3. In the standard LM algorithm, at the n^{th} step the gradient, $\mathbf{g}(n)$, and Hessian matrix, $\mathbf{H}(n)$, are approximated from the Jacobian $\mathbf{J}(n)$

$$\mathbf{g}(n) = \nabla E(\mathbf{w}) \approx \mathbf{J}^T(\mathbf{w}(n))\mathbf{e}(\mathbf{w}(n)) \tag{4.16}$$

$$\mathbf{H}(n) = \nabla^2 E(\mathbf{w}) \approx \mathbf{J}^T(\mathbf{w}(n))\mathbf{J}(\mathbf{w}(n)) + \mu\mathbf{I} \tag{4.17}$$

where $\mathbf{I}$ is the identity matrix and μ is a variable parameter.

The standard LM weight update is then given by

$$\Delta\mathbf{w} = \left[\mathbf{J}^T(\mathbf{w}(n))\mathbf{J}(\mathbf{w}(n)) + \mu\mathbf{I}\right]^{-1}\mathbf{J}^T(\mathbf{w}(n))\mathbf{e}(\mathbf{w}(n)) \tag{4.18}$$

The parameter μ is multiplied by some factor (β) whenever a step results in an increased error $E(\mathbf{w}(n))$. When a step reduces $E(\mathbf{w}(n))$, μ is divided by β. Typically, $\beta = 10$. When μ is large the algorithm becomes steepest descent (with step $1/\mu$), while for small μ the algorithm becomes Gauss-Newton (second order).

4.2.5 Levenberg-Marquardt with Adaptive Momentum (LMAM)

The LM algorithm is acknowledged as one of the fastest training algorithms with quadratic rate of convergence as it approaches a solution. One disadvantage of the LM algorithm is that if it converges to a local minimum there is no way to escape it, resulting in a suboptimal solution.

The Levenberg-Marquadt with Adaptive Momentum (LMAM) provides a momentum term that can help overshoot a local minimiser. It is based on the algorithm for MLPs presented in (Ampazis & Perantonis, 2000, 2002). This particular algorithm has two free parameters that have to be determined at the start of training, δP and ξ. The first parameter, δP, defines the trust region in weight space around the current state of the network within which the new optimum point will be restricted. The second parameter, ξ, determines the contribution of the momentum term to the weight update. A large ξ indicates the update is closer to the standard LM step, whereas a small ξ indicates a greater contribution by the momentum term.

As in the standard LM algorithm, at step n the gradient, $\mathbf{g}(n)$, and Hessian matrix, $\mathbf{H}(n)$, are approximated from the Jacobian, $\mathbf{J}(n)$, as given in (4.16) and (4.17). The standard LM weight update given in (4.18), is denoted in this algorithm as $\Delta\mathbf{w}_{LM}$.

In the LM with Adaptive Momentum algorithm, the weight update is restricted to a trust region defined by δP. To solve this constrained optimisation problem, two Lagrange multipliers, λ_1 and λ_2, are introduced, given respectively by

$$\lambda_1 = -2\frac{(\lambda_2 dQ + I_{GF})}{I_{GG}} \tag{4.19}$$

$$\lambda_2 = \frac{1}{2}\left[\frac{I_{GG} r_i^2 (1-\xi^2)}{I_{FF} I_{GG} - I_{GF}^2}\right]^{-1/2} \tag{4.20}$$

where

$$I_{FF} = \mathbf{g}^T(n)\,\mathbf{H}(n)\,\mathbf{g}(n) \tag{4.21}$$

$$I_{GF} = \mathbf{g}^T(n)\,\Delta\mathbf{w}(n-1) \tag{4.22}$$

$$I_{GG} = \mathbf{g}^T(n)\,\Delta\mathbf{w}_{LM} \tag{4.23}$$

$$dQ = -\xi.\delta P\sqrt{I_{GG}} \tag{4.24}$$

The final weight update is then given by

$$\Delta\mathbf{w}(n) = -\frac{\lambda_1}{2\lambda_2}\Delta\mathbf{w}_{LM} + \frac{1}{2\lambda_2}\Delta\mathbf{w}(n-1) \tag{4.25}$$

The form of the weight update is similar to the update for gradient descent with momentum. The first term contains the standard LM weight update, $\Delta\mathbf{w}_{LM}$, and the second contains the previous weight update, akin to the momentum term. It should be noted that $\Delta\mathbf{w}_{LM}$ is also used implicitly in calculating the multipliers λ_1 and λ_2.

4.2.6 Optimised Levenberg-Marquardt with Adaptive Momentum (OLMAM)

The LMAM algorithm described in the previous section has two free parameters, δP and ξ, that need to be externally determined. The Optimised Levenberg-Marquardt with Adaptive Momentum (OLMAM) algorithm is a modification of the LMAM algorithm, proposed in (Ampazis & Perantonis, 2002), to achieve independence from externally provided parameter values. The optimal values for these parameters are determined adaptively at each epoch:

$$\xi = \sqrt{1 - \frac{I_{GF}^2}{I_{GG} I_{FF}}} \tag{4.26}$$

$$\delta P = \sqrt{I_{GG}} \tag{4.27}$$

Ampazis and Perantonis have also used $\sqrt{I_{GG}}/64 < \delta P < \sqrt{I_{GG}}/8$ in their experiments, achieving similar performance. Initial tests with SIANNs indicate that better accuracy and speed is achieved using the 'optimal' value as defined in (4.27), and this has been used in subsequent experiments.

4.3 Direct Solution Algorithms

The initial attempts to implement Direct Solution (DS) methods for SIANNs were based on the FBFBK and LSB algorithms developed in (Barmann & Biegler-Konig, 1992, 1993), as described in Section 2.8.4. These attempts were not successful. The algorithms were unstable, probably due to the complexity of the shunting layer. The shunting inhibitory neuron equation does not lend itself easily to a direct solution.

For MLPs, the algorithm takes the desired output of the neuron, works out the desired input to the activation function by using the inverse of the function, then works out the new synaptic weights by directly solving for them from the given inputs in a least-squares sense. In the case of SIANNs the process is much more complicated because the activation function is just one term in the denominator, with the a and b terms to be solved for as well. Additionally, the a term is constrained by the limit placed on the denominator.

In order to overcome this problem, an alternative hybrid approach was used, similar to that described in (Verma, 1997). Direct solution for the output perceptron layer is combined with Gradient Descent with Momentum (GDM) for the shunting layer. At each epoch, the optimal output layer weights and biases are "solved" directly; the target (optimal) values for the outputs of the shunting neurons are calculated; and these then become the target values for the GDM-based update of the shunting layer parameters. This hybrid scheme, named DS-GDM, was implemented successfully.

The natural progression was then to combine the Direct Solution method with the GDX algorithm for the Shunting Inhibitory layer. The resulting algorithm (DS-GDX) performed better than DS-GDM.

Summary of the DS-GDM and DS-GDX algorithms

1. Calculate the outputs of shunting layer neurons ($\mathbf{x}$) from the inputs.
2. Calculate desired inputs to the output layer activation function ($\mathbf{v}_{target}$) by passing target values through inverse of output activation function.
3. Directly solve for the output layer weights and biases using $\mathbf{x}$ and $\mathbf{v}_{target}$ in a least squares sense, from the set of equations $\mathbf{w}.\mathbf{x} = \mathbf{v}_{target}$.
4. Calculate 'new' target values for the shunting layer ($\mathbf{x}_{target}$) by backpropagating $\mathbf{v}_{target}$ through the updated output layer weights.
5. Use backpropagation based algorithms (GDM or GDX) to update shunting layer weights, using the difference between $\mathbf{x}_{target}$ and $\mathbf{x}$ as the error vector.
6. Go back to step 1.

4.4 Stochastic algorithms

Stochastic algorithms involve a search for weights using random techniques. The motivation behind stochastic algorithms is to find solutions that may not otherwise be found. While a lot of effort may be wasted in "blind alleys", the computational simplicity may compensate for the apparent inefficiency of the search. This concept is used by most initialisation schemes. By randomly initialising the networks, each network starts at a different point in the weight space, thereby covering a greater portion of the search space. The following algorithms use a random update to search the weight space.

4.4.1 Random Optimisation Method (ROM)

The Random Optimisation Method (ROM) is based on the stochastic algorithm given in (Schalkoff, 1997, p.208). The error, $E(\mathbf{w})$, is defined as the objective function and X as the region over which to search for the value of w that minimises $E(\mathbf{w})$. The basic formulation of the random optimisation method is as follows:

1. Select $\mathbf{w} \in X$; set $n = 0$.
   Let M be the total number of steps or iterations allowed.
2. Generate a Gaussian random vector $\xi(n)$.
   If $\mathbf{w}(n) + \xi(n) \in X$, go to step 3.
   Otherwise go to step 4.
3. If $E(\mathbf{w}(n) + \xi(n)) < E(\mathbf{w}(n))$,

   then $\mathbf{w}(n+1) = \mathbf{w}(n) + \xi(n)$.

   Else, check the 'reverse' side:

   - If $E(\mathbf{w}(n) - \xi(n)) < E(\mathbf{w}(n))$,

     then $\mathbf{w}(n+1) = \mathbf{w}(n) - \xi(n)$

   - Otherwise $\mathbf{w}(n+1) = \mathbf{w}(n)$.
4. If $n = M$, stop (limit on number of iterations has been reached). Otherwise, let $n \rightarrow n+1$ and go to step 2.

The ROM algorithm implements "reverse side checking". The idea is that if a step takes E 'uphill', then the reverse step is likely to take it 'downhill'. If E does not decrease then continue with a new random vector.

4.4.2 Extension to the Random Optimisation Method (ROM2)

An extension to ROM was implemented based on (Solis & Wets, 1981), as given in (Schalkoff, 1997, p.208). The extension incorporates a statistical bias into the weight adjustment procedure by allowing the mean of ξ to be non-zero. The mean of ξ at iteration n is denoted by $\mathbf{b}(n)$. The only modification involves step 3:

3a. If $E(\mathbf{w}(n)+\xi(n)) < E(\mathbf{w}(n))$, then

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \xi(n)$$

$$\mathbf{b}(n+1) = k_1\xi(n) + k_2\mathbf{b}(n).$$ (Typical values are $k_1 = 0.4$ and $k_2 = 0.2$.)

Otherwise, check the 'reverse' side:

i) If $E(\mathbf{w}(n)-\xi(n)) < E(\mathbf{w}(n))$, then

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \xi(n)$$

$$\mathbf{b}(n+1) = \mathbf{b}(n) - k_3\xi(n).$$ (Typical value is $k_3 = 0.4$.)

ii) Otherwise,

$$\mathbf{w}(n+1) = \mathbf{w}(n)$$

$$\mathbf{b}(n+1) = k_4\mathbf{b}(n).$$ (Typical value is $k_4 = 0.5$.)

Note: $\mathbf{b}_0 = 0$

The adjustment of the mean of $\xi(n+1)$, namely $\mathbf{b}(n+1)$, is updated using the values of $\xi(n)$ and $\mathbf{b}(n)$ that have been successful in reducing E. This could be viewed as a form of momentum, in a statistical sense. When the error does not decrease, the mean $\mathbf{b}(n)$ decays toward 0.

4.5 Experimental Methods

This section describes the network structures, training and testing procedures, evaluation metrics, and benchmark test problems, used to train and evaluate the performance of SIANNs.

4.5.1 Network Structures

An m-dimensional input vector is presented to the network and is used to produce an n-dimensional output vector, the values of m and n being determined by the particular problem. By definition, the standard SIANN structure therefore consists of an m-neuron shunting layer and n-neuron output layer (refer Section 3.6).

For binary classification problems only one output neuron is required to give the classification result. The mid-point of the neurons output range is taken as the threshold value. Any output above the threshold is taken as a one class, and values below taken as the other. For multi-class problems, the number of output neurons is set to be equal to the number of classes, where each output neuron corresponds to one class. The 'winning' class is the neuron with the highest output, otherwise known as winner-takes-all (WTA) configuration. This is in accordance with the 'benchmarking rules' laid out in (Prechelt, 1994). For time-series prediction, the number of output neurons will be equal to the number of predicted variables.

4.5.2 Weight Initialisation

In order eliminate any bias due to initial conditions, as well as to increase the coverage of the input space, fifty networks with randomly generated initial weights were tested for each problem. The weights $\mathbf{c}$ and $\mathbf{w}$ were initialised using a random number generator that generates uniformly distributed values in the range $[-r, r]$.

Thimm and Fiesler have compared initialisation schemes for perceptrons and found that schemes of this form perform well over a variety of problems (Thimm & Fiesler, 1997). Initial tests on SIANNs used $r = 1$, but subsequent results indicated that the scheme used in (Smieja, 1991) performed well over the different range of problems. In this scheme, the range r is defined by

$$r = \frac{2}{\sqrt{N}} \tag{4.28}$$

where N is the number of inputs of the particular neuron (fan-in). This initialisation scheme ensures that the sigmoid activation functions start in their linear regions and not in saturation, thereby improving training performance (Y. Lee et al., 1991).

The shunting neuron bias b was initialised with $r = 1$ as it does not affect the activation functions. The decay parameter a was initialised to a random value between 0 and 1, then offset with a constant. The offset constant, a_{lim}, is set so that the constant added with the lower bound of the denominator activation function will not be smaller than the predefined limit value for the denominator, s_{lim} (refer (4.31)). For example, the hyperbolic tangent (*tansig*) activation function has a lower bound of -1; therefore, for $s_{lim} = 0.1$, the constant offset would be 1.1. This lower limit for a and its effects on training performance are discussed in greater detail in Section 4.6.1.

4.5.3 Input pre-processing

The input attributes to a learning problem can have magnitudes and distributions that vary widely. There are some common methods to represent these attributes when applying such problems to neural networks (Prechelt, 1994). The real- and integer-valued inputs have generally been scaled and offset to the range $[-1, 1]$ in the experiments. One exception is with time series prediction, such as the Sunspots problem, where the inputs in some exemplars are the output targets in others. In order to enable the sigmoid output activation functions to produce the required output values, the data has been scaled to the range $(0, 1)$.

4.5.4 Data partitioning

Each dataset was partitioned into training, validation and test sets; unless otherwise stated, the general strategy is to partition the dataset into 50% as training set, 25% as validation set, and the remainder 25% as a test set. A well-trained neural network should be able to correctly classify previously unseen inputs (good generalisation). The networks were trained using the training set data and their performance measured using the test set, which generally contains data not seen during training. The validation set is used for early stopping so that the networks are not overtrained and are able to generalise well. All the results presented in this chapter are based on the test set, except where the algorithm training performance is evaluated.

4.5.5 Activation functions

Three different activation functions were used with the shunting neurons: the *hyperbolic tangent* (*tansig*, *tnh*), *logistic sigmoid* (*logsig*, *lgs*) and the *exponential* (*exp*) functions. For the output neurons, the tansig, logsig and linear (*lin*) activation functions were tested. All possible combinations were tested to observe their effect on performance.

4.5.6 Training Termination Criteria

The training is stopped if the target objective function value or 'error goal' is achieved. It is quite possible that the neural network being trained cannot achieve the error goal. Therefore, the maximum number of training epochs is set to 1000 in these tests. Initial investigations revealed that, in most cases, this is sufficient and it was only non-converging networks that trained beyond this limit, consuming processing time with no significant improvement in performance. This limit also allowed training times to be kept within reasonable limits.

In order to achieve good generalisation, a validation set is normally used for early stopping so that the networks are not over-trained. If a validation set is used during training, the network weights that result in the minimum validation set error are saved. If the validation set error is not reduced for 50 consecutive epochs, the training is stopped and the final network weights used for testing are those that resulted in the minimum validation set error.

4.5.7 Test Performance Metrics

In order to compare the performance of the trained neural networks, performance metrics have to be used. The performance measure used during training is the mean squared error (MSE). The test set performance can similarly be evaluated using the MSE. The MSE, however, can vary depending on the problem and the way it is implemented, particularly the magnitude of the target and actual output values. It is not clear from the MSE whether it represents a 'good' or 'poor' performance. Intuitively appealing metrics should not only be relatively independent of the implementation of the problem, but also ideally should easily differentiate 'good' from 'bad' performance.

For the classification problems, the test classification error rate is used, where the error rate is simply the percentage of the test set that is misclassified. The tests are carried out with a batch of randomly initialised networks trained on the same problem. The general performance for the set of networks is represented by the mean and median of the test error rates for the whole batch.

The 95% confidence interval (CI) on the mean error rate is also calculated and presented. As each batch consists of 50 networks, the number of samples in the batch is large enough to assume normal distribution using the Central Limit theorem (Johnson & Bhattacharya, 1996; Walpole et al., 1998). The 95% CI gives the range of values within which the true mean error lies with 95% probability. It is calculated from the sample mean and sample standard deviation, μ and s, according to

$$\mu \pm 1.96\frac{s}{\sqrt{n}} \tag{4.29}$$

where n is the number of samples (Mitchell, 1997).

The best case error rate is also presented to give an indication of the performance level that can be achieved by a single network. The performance of a batch of networks is also given by the percentage of networks in the batch achieving a particular performance target such as the error goal, 0% classification error (perfect test classification) or less than 20% classification errors.

For time series predication, the actual network output values need to be analysed. The performance metrics used are the MSE and the average relative variance (ARV) measure (Nikolaev & Iba, 2003; Weigend et al., 1990), given by

$$ARV = \frac{\sum_{i=1}^{N}(y_i - P(\mathbf{x}_i))^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2} \tag{4.30}$$

where y_i is the true outcome of the i^{th} example, $P(\mathbf{x}_i)$ is the estimated outcome with the i^{th} input vector $\mathbf{x}_i$ in the same example, and $\bar{y}$ is the mean of the true outcomes.

The ARV is essentially the MSE divided by the variance of the target values. This scales the error value down if the series is highly variable, so the network is not unduly punished.

In order to compare the computational power and time required to train the neural networks, the mean CPU time, in seconds, required to train one network was measured and recorded. This training time was measured using an internal Matlab function and was set up to measure only the time spent on training the network and not the time spent on other tasks such as setting up the data and saving the results. The simulations were run on MATLAB v6.5 on Sun workstations. The Sun Blade 1000 was used as the 'standard' for measuring the training time, and any measurements made on other systems were scaled based on comparative test measurements. Despite these precautions, it should be noted that there could be some variations in measurement due to varying load factors on these multi-user systems.

4.5.8 Benchmark Tests

A number of benchmark problems were used to test the learning capabilities of SIANNs. The benchmarks used were the 3-bit parity problem, Wisconsin Breast Cancer dataset, the Pima Indians Diabetes dataset, an artificial multi-class problem and the Sunspot time series. They form the standard set of benchmark problems used throughout the rest of this thesis. These benchmarks consist of four classification problems, including one multi-class problem, and one time-series prediction problem. The parity and multi-class problems are synthetic, while the remaining three are real-word problems. The benchmark problems are described below.

4.5.8.1 The 3-bit parity problem

The 3-bit parity problem is a popular artificial classification problem where the network has to generate the appropriate binary output for a 3-bit binary input so that there is always an even (or odd) number of ones. This 3-dimensional problem is not linearly separable and is one level of complexity higher than the 2-dimensional XOR problem. The 8 input combinations can be visualised as the vertices of a unit cube, where no two adjacent vertices are of the same class. The problem can also be described as a 3-input modulo-2 addition. For this problem, since there are only 8 possible input patterns, all 8 were used for both training and test sets.

4.5.8.2 The Wisconsin Breast Cancer problem

The Wisconsin Breast Cancer dataset is a real-world medical diagnosis dataset obtained from the UCI Machine Learning Repository (Blake & Merz, 1998). The breast cancer dataset has 699 samples with 9 integer inputs and two output classes (benign and malignant). The data has missing values that were replaced by zeros before scaling. Obviously, this is not the best approach for estimating the missing values, but was chosen for the sake of simplicity (Hathaway & Bezdek, 2001).

4.5.8.3 The Pima Indians Diabetes problem

The Pima Indians Diabetes dataset is a real-world medical diagnosis dataset obtained from the UCI Machine Learning Repository (Blake & Merz, 1998). The dataset has 768 samples with 8 real-valued inputs and two output classes. The diabetes diagnosis problem is supposed to be a lot harder for the neural networks compared to the breast cancer problem. In previously reported results for this problem (Prechelt, 1994; Sherrah, 1998; Waschulzik et al., 2000), the best case error rates were around 20%.

The diabetes dataset is said not to have any missing values, but there are a number of zero-value entries that appear to have simply been inserted to replace missing values. The effect of these zero-value entries as well as the effect of removing the two inputs with large numbers of zero values, has been investigated using both MLPs

and SIANNs in (Arulampalam & Bouzerdoum, 2001a). It was found that removing these inputs doesn't affect the performance drastically. However, these entries and inputs are retained here in order to enable fair comparison with other published data.

4.5.8.4 Artificial multi-class problem

The previous benchmark problems have only 2 output classes; therefore, the networks only require a single output neuron. In order to test the training algorithms and the networks on a problem with multiple outputs, i.e., a problem requiring more than one output neuron, the Multi-class classification problem was artificially created. This is an artificially created 2-dimensional, 3-class problem that is not linearly separable. It has a total of 600 samples evenly distributed over the 3 classes. A small amount of overlap between classes was also built-in to make the problem more difficult. This overlap will not allow perfect classification results. The distribution of the classes is shown diagrammatically in Fig. 4.1.

4.5.8.5 The Sunspot time series

A time series problem was chosen as the final benchmark, to provide a different kind of problem that would round off the initial set of tests. The Sunspot time series is a real-world dataset that contains the number of sunspots detected from 1700 onwards. The data obtained is an updated series that goes up to 2002 (Cugnon & team, 2003), but for comparison with previous work (Nikolaev & Iba, 2003; Park et al., 1996; Weigend et al., 1990) only a subset of the data has been used. The problem is to predict a 'future' value of the number of sunspots given some previous recorded values. The Sunspots time series is shown in Fig. 4.2.

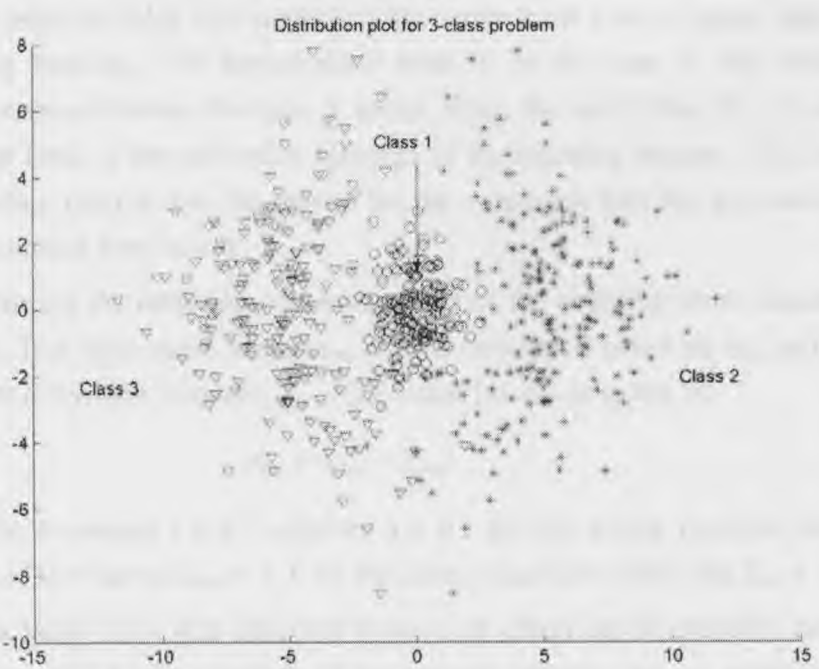


Fig. 4.1: Distribution plot for the Multi-class problem

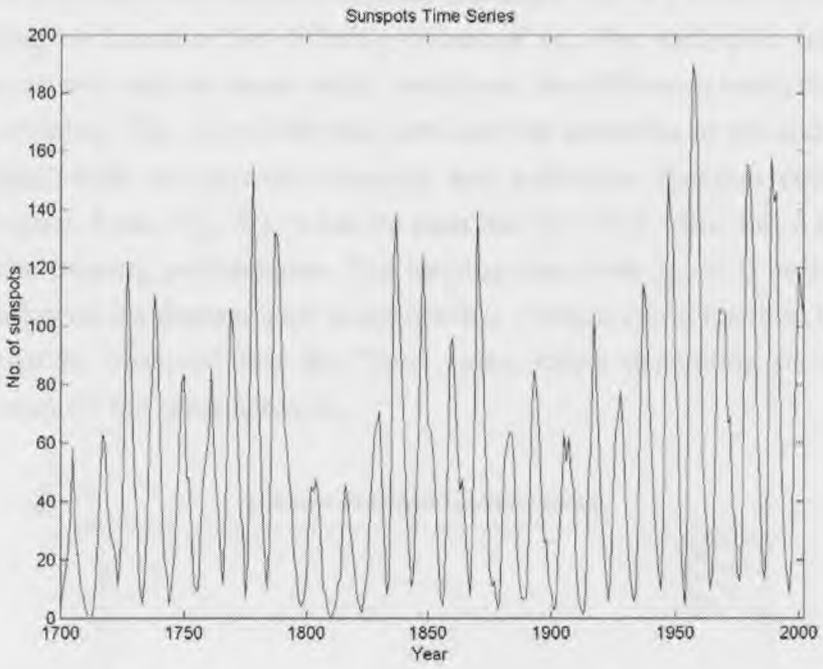


Fig. 4.2. Plot of Sunspots activity for the years 1700 - 2002

4.6 Experimental Test Results

4.6.1 Effect of limiting the decay term *a* during training

The denominator, s, of the shunting neuron, as given in (4.8), is constrained to be positive. In order to fulfil this condition, the decay term a has a lower limit imposed on it during training. The denominator term, s, is the sum of the output of the shunting neuron activation function, f, and a. Thus, the limit value for a is dependent on the lower limit of the activation function of the shunting neuron. This lower limit on the shunting term is also the reason for the constraint that the activation function has to be bounded from below.

When training the networks, the lower limit of the shunting term, denoted as $s_{\lim}$, is specified. The limit value for a, $a_{\lim}$, is then calculated based on $s_{\lim}$ and the lower bound of the activation function, $f_{\min}$. The value for $a_{\lim}$ is given by

$$a_{\lim} = s_{\lim} - f_{\min} \tag{4.31}$$

For example, to ensure $s \geq 0.1$ requires $a \geq 0.1$ for the *logsig* function which has a lower limit of 0, whereas $a_{\lim} = 1.1$ for the *tansig* function which has $f_{\min} = -1.0$.

The limit value for s was observed to have an effect on the stability and duration of training as well as the accuracy of the trained network. As $s_{\lim}$ approaches 0, the

likelihood of instability and non-convergence increases. Fig. 4.3 shows two examples of the training performance for differing values of s_{lim}. For each plot, the network training was started with the same initial conditions, the difference being the value of a_{lim} during training. The only difference between the networks in (a) and (b) is the initial weights, with the network structure and activation function combinations exactly the same. From Fig. 4.3, it can be seen that the limit value has a significant impact on the training performance. The training runs with $s_{\text{lim}} = 0$ performed the worst as it allowed the denominator to approach 0 – which could result in instability. It should also be observed that the 'best' value varies depending on the initial conditions even for the same network.

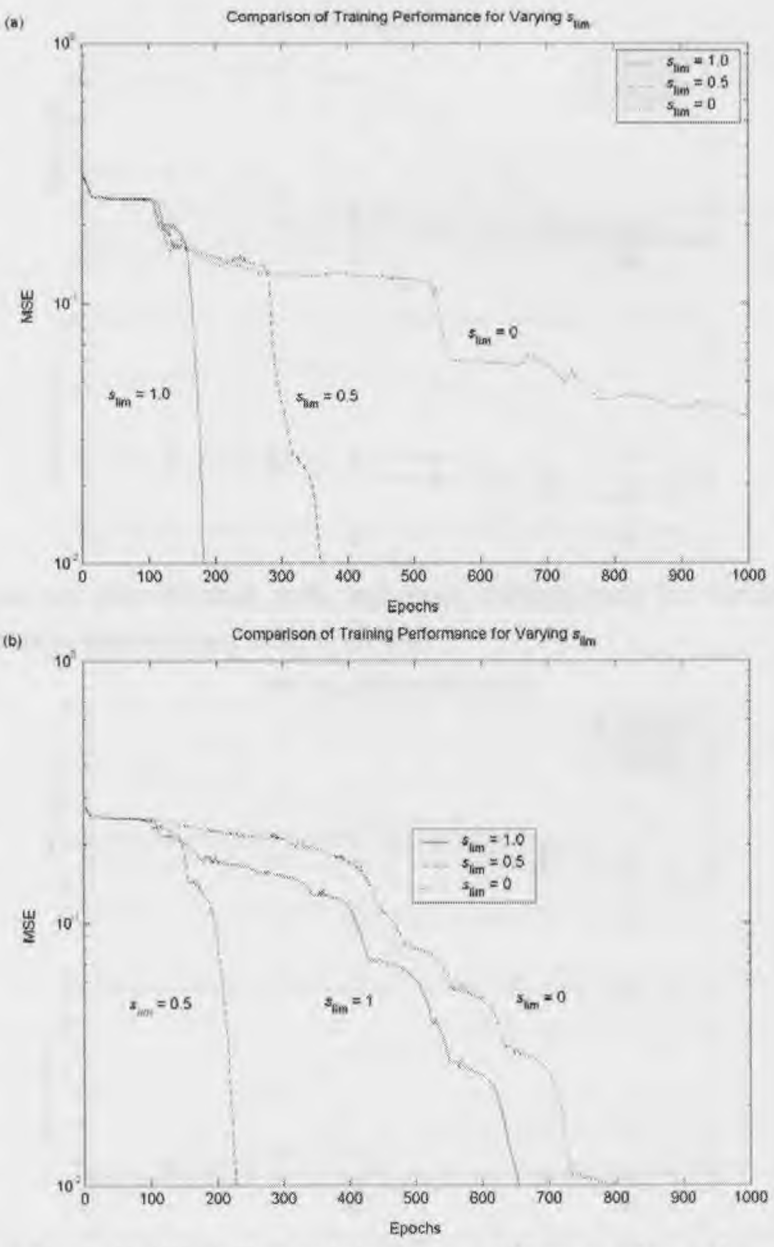


Fig. 4.3: Evolution of MSE during training using different values of s_{lim}. Plots in (a) and (b) use different initial values.

In order to find an (approximate) optimum value for s_{lim} to be used in subsequent experiments, tests were carried out using s_{lim} values of 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1.0 and 2.0 on some of the benchmark problems. The networks were trained using both the Levenberg-Marquadt (LM) and gradient-descent with adaptive learning rate and momentum (GDX) algorithms. All the different combinations of shunting and output layer activation functions were tested for s_{lim} = 0.1 and 1. Only the best performing networks for each case were then tested for the other values of s_{lim}. If the same network performed best for both values, the second best combination was also tested. The variation in performance of these networks for different s_{lim} are shown in Figs. 4.4 to 4.7.

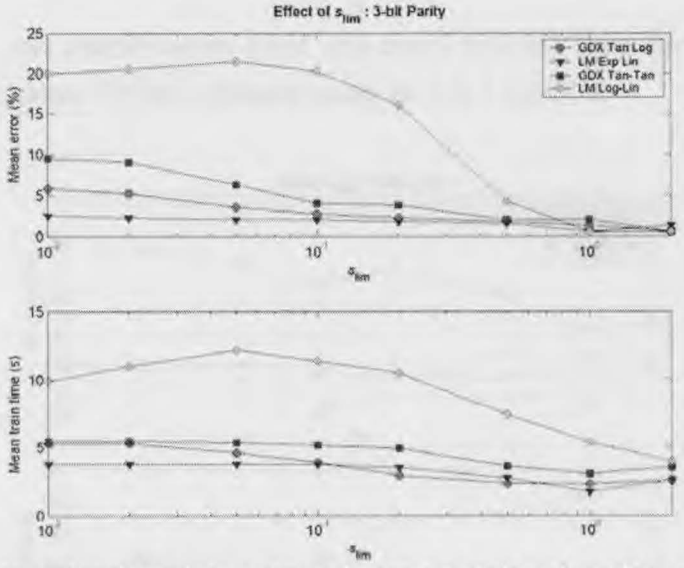


Fig. 4.4: Mean test classification error and mean training time for various s_{lim} for 3-bit parity dataset using a 3-3-1 SIANN.

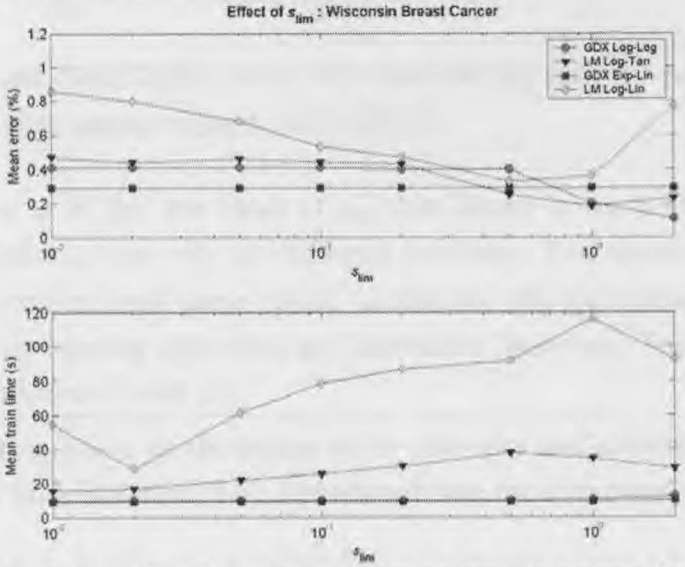


Fig. 4.5: Mean test classification error and mean training time for various s_{lim} for Wisconsin Breast Cancer dataset using a 9-9-1 SIANN.

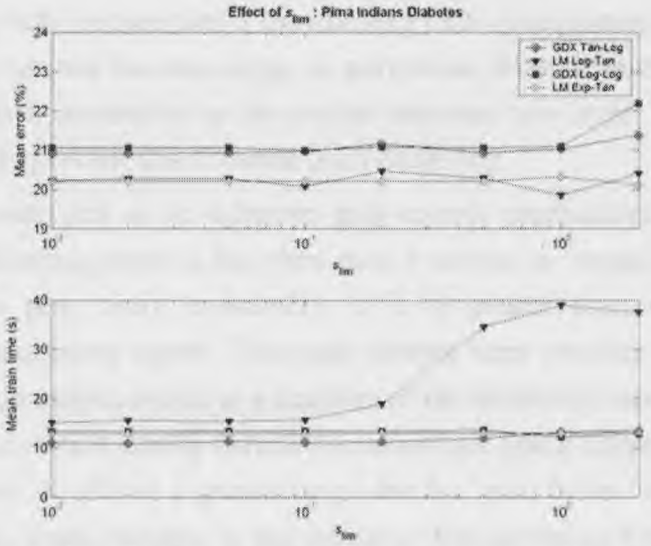


Fig. 4.6: Mean test classification error and mean training time for various s_{lim} for Pima Indians Diabetes dataset using an 8-8-1 SIANN.

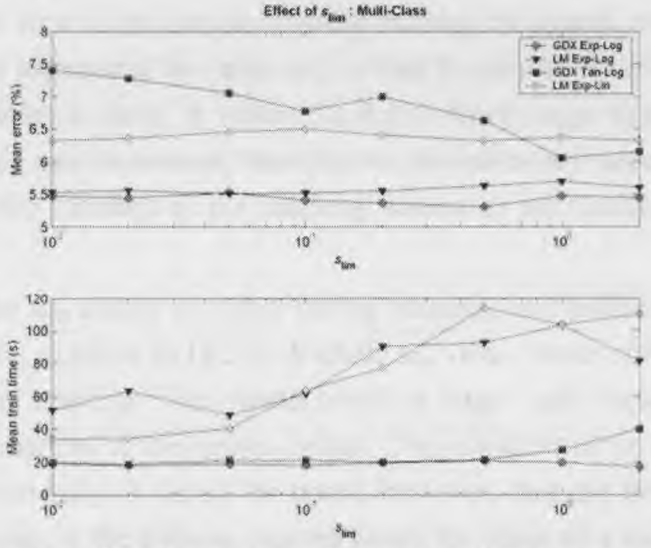


Fig. 4.7: Mean test classification error and mean training time for various s_{lim} for the Multi-class dataset using a 2-2-3 SIANN.

The objective is to find the value of s_{lim} that results in the best combination of accuracy and training time over the different problems. The results show that even for the same problem and same initial conditions, the performance trends vary depending on the training algorithm and activation functions. The optimum value appears to be between 0.5 and 1.0.

The effect of s_{lim} and a on the output of the networks and stability during training can be deduced from Equation (4.6), reproduced here for convenience:

$$x_j = \frac{I_j + b_j}{a_j + f\left(\sum_{i=0}^{m} c_{ji} I_i\right)} \tag{4.32}$$

From (4.32), it can be seen that if the shunting term (denominator) becomes small, the output of the neuron becomes large. In particular, if the denominator approaches zero, then we have instability as the output becomes too large. The limit on the shunting term is to prevent this situation from occurring.

The denominator acts as an adaptive gain-control mechanism for the shunting neuron. If this shunting term is less than one, it serves to 'amplify' the numerator (excitatory input plus bias); conversely, if it is greater than one, it serves to "attenuate" the excitatory signal. This gain control term consists of two parts: the activation function output, which is a function of the inhibitory inputs, and the decay term a, which is constant during normal operation (not being trained). A small value for the decay term, a, allows a greater range for the 'gain factor' determined by the inhibitory inputs. Small changes in the output of the activation function, due to the inhibitory inputs, can then result in large changes in the neuron output. On the other hand, a large value for a reduces the variation in the shunting term, also reducing the range of the gain. In other words, the decay parameter a has a 'dampening' effect on the shunting gain control mechanism, thereby making the neuron output more stable. The drawback of increasing the value of a is that it reduces the effectiveness of the neuron and its ability to learn. A value of a that is much larger than the range of the activation function can 'drown out' the effect of the inhibitory inputs. In the extreme, this will reduce the function of the shunting neuron to just scaling and biasing the excitatory input.

The limit value s_{lim} comes into play during training, as it defines the lower bound on the decay term, as given in (4.31). A small s_{lim} value means that a could become very small during training. This would result in large 'gain factor', and therefore possibly large variations in the neuron output. This could result in instability, as was observed experimentally. It should be noted, however, that the limit value may not even come into play, if the training process keeps the value of a away from its lower limit.

From this discussion, it becomes apparent that the selected value of s_{lim} should reduce the risk of instability, but at the same time should not overly restrict the function of the shunting neuron. The value chosen should attempt to balance these conflicting requirements. Setting the value of s_{lim} to 0.5 allows the shunting inputs to 'amplify' the output to a reasonable level, up to a maximum factor of 2. On the other hand, setting the value of s_{lim} to 1 restricts the effect of the shunting term to a 'dampening' or 'attenuating' effect. This effect would appear to be more in line with the name 'shunting inhibition'. The 'amplification' effect, if any, will depend on the synaptic weights connecting the shunting neurons to the output layer.

Given the experimental results and the above discussion, it was decided that for the remaining experiments the value of s_{lim} is 1, unless otherwise stated.

4.6.2 Benchmark Test Results

In this sub-section, the results for the five different benchmark tests are presented. The results shown here are for the best performing activation function combination for each of the training algorithms used. The full set of mean test error rates and mean test ARV for the five benchmark tests are given in Tables B.1 to B.5 in Appendix B.

4.6.2.1 Results for the 3-bit parity problem

A set of 50 randomly generated 3-input, 1-output SIANNs, as described in Section 4.5.1, was trained on the 3-bit Parity problem. The 3-bit parity problem has only 8 binary input combinations and all these input patterns were used for training and testing, with no validation. The networks were trained for a maximum of 1,000 epochs with an error goal of 0.01. All nine activation function combinations, described in Section 4.5.5, were trained using eight different training algorithms.

The results for the best performing activation function for each of the algorithms are shown in Table 4.1. The first column of the table shows all the algorithms used to train the networks and the second and third columns show the best performing activation function configuration for the respective algorithms. The rest of the table shows the performance metrics for the given set of networks. Columns 4 to 6 show the percentage of the networks achieving the following: the objective function (mean squared error) goal; zero classification errors for the test set; and less than 20% classification errors. Columns 7 to 9 give the average number of training epochs for the following: all networks; networks that achieved the objective function goal; and networks that achieve all correct test set classification. This is followed in Columns 10 to 14 by the test set classification error: best case; mean; 95% confidence interval (CI) on the mean; and median. The last column gives the mean training time per network based on CPU time usage in seconds. The mean and median test error and mean training time for each case are shown graphically in Fig. 4.8. Note that since the median is 0 in most cases, it is not visible on the plot.

The results prove that SIANNs are able to correctly solve the 3-bit parity problem consistently. Most of the algorithms had a median error of 0% and over 85% of the networks achieve 100% correct classification, except for GDM and the stochastic algorithms (ROM and ROM2). Even these three algorithms were able to achieve 100% correct classification for more than one third of the networks. These algorithms are actually able to achieve better accuracy if run for longer, e.g. 10,000 epochs. However, for the sake of consistency and to be able to make fair comparisons between algorithms, the training was restricted to 1,000 epochs for all algorithms.

Table 4.1 Best results for 3-bit Parity problem using 3-3-1 SIANN

Training Algorithm	Activation functions		Performance (% of runs)			Avg. Epochs			Test Set Classification Error				Mean Training CPU time (s)
	Shunt	Out	→ goal	0% err	< 20% err	All runs	→ goal	0% Errors	Best (%)	Mean (%)	95% CI	Median (%)	
GDM	Tnh	Tnh	36	44	50	880	667	727	0.00	22.50	± 6.26	18.75	5.5
GDX	Tnh	Lgs	94	94	96	352	311	311	0.00	1.25	± 1.44	0.00	2.3
LM	Exp	Lin	96	96	100	84	46	46	0.00	0.50	± 0.69	0.00	1.7
LMAM	Exp	Lin	86	86	88	169	33	33	0.00	4.25	± 3.18	0.00	3.3
OLMAM	Exp	Lin	92	92	96	138	63	63	0.00	1.50	± 1.51	0.00	2.8
DSGDX	Tnh	Lin	94	94	100	169	116	116	0.00	0.75	± 0.83	0.00	1.3
ROM	Exp	Lgs	16	34	74	942	636	829	0.00	12.25	± 3.17	12.50	7.4
ROM2	Tnh	Lgs	2	42	78	987	359	969	0.00	10.50	± 3.00	12.50	8.1

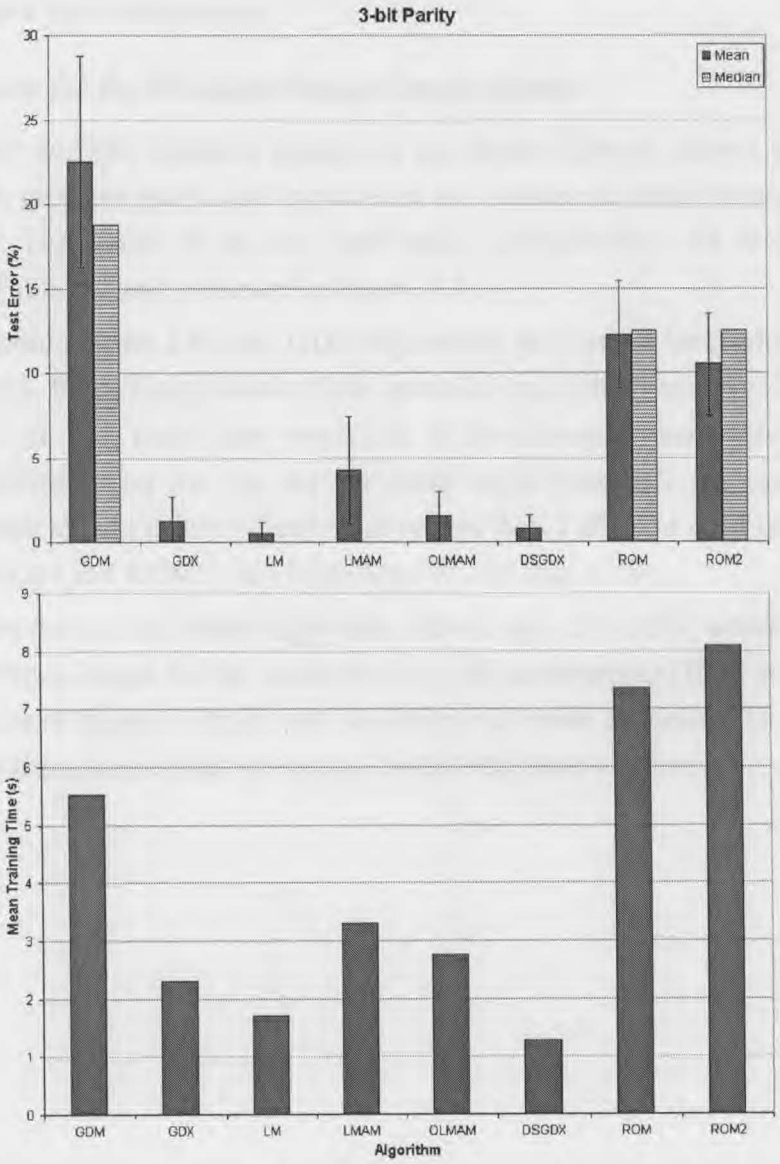


Fig. 4.8: Mean and median test classification error and mean training time for 3-bit parity problem using 3-3-1 SIANN with various training algorithms.

The LM algorithm was the most accurate with 96% of networks achieving perfect classification and 0.5% mean error, and also required the fewest epochs to train the networks. However, it was not the fastest in terms of time as it is a second order algorithm that requires more computation per epoch. The fastest algorithm in terms of computation time was the DS-GDX algorithm, 5 times faster than LM. DS-GDX was also the second best in terms of accuracy with 94% of networks producing perfect classification and mean error of 0.75%. Next, in terms of accuracy, is the GDX algorithm followed by the LM variants, LMAM and OLMAM, that had lower accuracy and longer training times than the standard LM. The longest to train were the GDM and ROM algorithms.

There does not appear to be single 'optimal' selection of activation function as the different algorithms have different 'best' combinations, though LM and its variants all had the same best combination.

4.6.2.2 Results for the Wisconsin Breast Cancer dataset

The 'standard' SIANN structure trained on the Breast Cancer dataset was a 9-9-1 SIANN as the problem has 9 input parameters and requires a single binary output for classification. The results of the best performing configurations for each algorithm are given in Table 4.2 and presented in Figure 4.9.

For this problem, the LM and GDX algorithms performed best with mean test errors of 0.20%. Even though none of the networks was able to achieve the objective function goal of 10^{-4}, more than two-thirds of the networks were able to achieve perfect classification on the test set for these algorithms. All the non-stochastic algorithms were able to achieve mean error of less than 1.0% and even the stochastic algorithms (ROM and ROM2) had mean error of less than 1.5%.

GDX was also the second fastest algorithm, behind only DS-GDX, whereas LM took more than 3 times longer for the same accuracy. By comparison GDM and OLMAM took 6 to 7 times longer to train only to achieve a lower accuracy. The ROM and ROM2 algorithms trained fast but had comparatively poor accuracy.

Table 4.2 Best results for Wisconsin Breast Cancer dataset using 9-9-1 SIANNs

Training Algorithm	Activation functions		Performance (% of runs)			Avg Epochs			Test Error				Mean Training CPU time (s)
	Shunt	Out	→ goal	0% err	< 20%	All runs	→ goal	0% Errors	Best (%)	Mean (%)	95% CI	Median (%)	
GDM	Tnh	Lgs	0	42	100	978	*	967	0.00	0.36	± 0.09	0.56	61.8
GDX	Lgs	Lgs	0	66	100	161	*	160	0.00	0.20	± 0.08	0.00	10.3
LM	Lgs	Tnh	0	74	100	181	*	119	0.00	0.20	± 0.10	0.00	34.8
LMAM	Lgs	Lin	0	28	100	98	*	97	0.00	0.76	± 0.19	0.56	24.1
OLMAM	Lgs	Tnh	0	56	100	297	*	263	0.00	0.37	± 0.14	0.00	71.0
DSGDX	Tnh	Lgs	0	36	100	96	*	100	0.00	0.54	± 0.14	0.56	6.4
ROM	Lgs	Lgs	0	6	100	1000	*	1000	0.00	1.38	± 0.27	1.13	14.7
ROM2	Lgs	Lgs	0	10	100	1000	*	1000	0.00	1.30	± 0.22	1.13	14.7

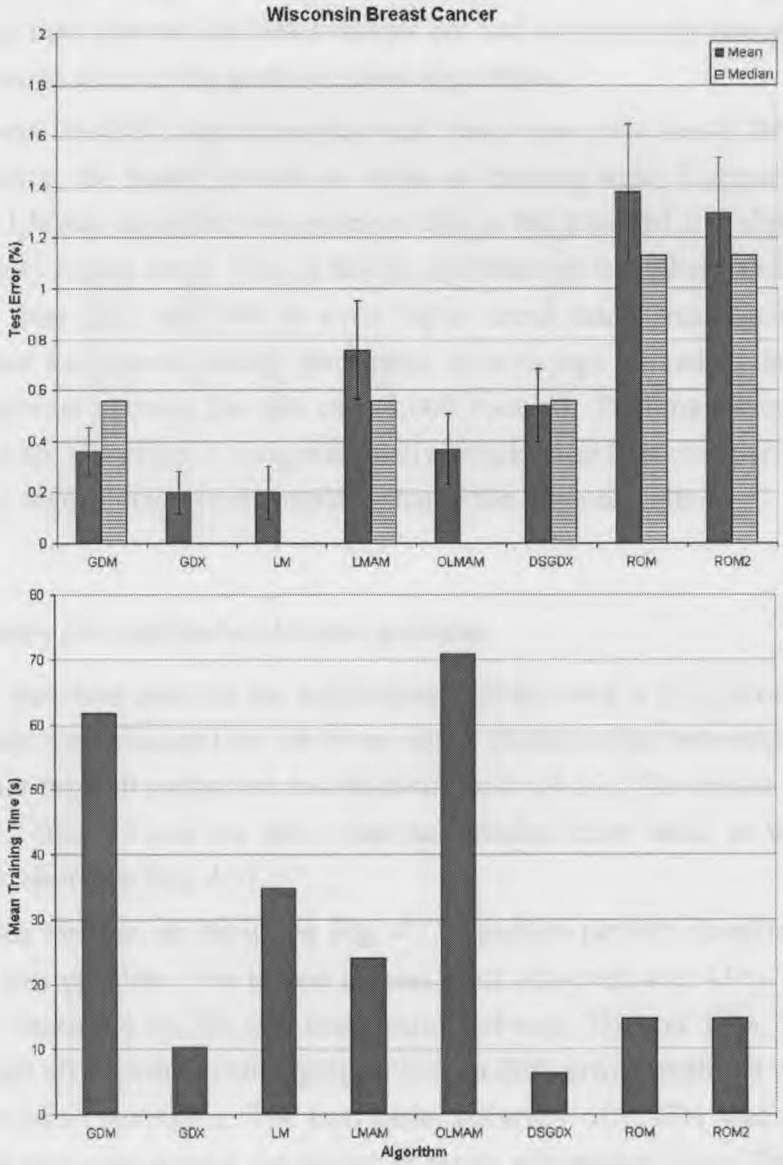


Fig. 4.9: Mean and median classification error for test set and mean training time for Breast Cancer dataset using 9-9-1 SIANN with various training algorithms.

4.6.2.3 Results for Pima Indians Diabetes dataset

The 'standard' 8-8-1 SIANNs were used for the Diabetes dataset and the results are shown in Table 4.3 and Figure 4.10. As mentioned in sub-section 4.5.8, it does not appear possible to achieve perfect classification for this dataset; error rates below 20% are considered 'good'. The results obtained conform to these expectations, with none achieving perfect classification and the average ranging from 19% to 22%. The best case results have test error rates of around 18%. Surprisingly, the best performing algorithm was the first-order GDM algorithm that had the lowest mean error of 19.05% and had 94% of networks achieving below 20% error. The GDM algorithm achieved this despite being 'only' a first-order algorithm. The time taken to train, however, was one of the highest as the number of epochs required was high. In contrast, the GDX algorithm, which is GDM with variable learning rate, had a mean training time almost four times shorter but had a mean error rate of 21%. This is the worst mean error of the gradient-based algorithms.

The LM and DS-GDX algorithms also had mean error rates below 20%, with DS-GDX also being the fastest overall in terms of training time. Comparing the LM variants, the LMAM algorithm was twice as fast as the standard LM algorithm, with only marginally higher error. The OLMAM algorithm on the other hand took nearly 50% longer than LM, and had an even higher error rate. Once again the ROM algorithms had the highest average error rates, even though trained for the maximum number of epochs allowed (in this case 1,000 epochs). Training times were short though, since the algorithm is computationally simple. The best case performance for the stochastic algorithms is comparable to that of the other algorithms.

4.6.2.4 Results for artificial multi-class problem

The SIANN structure used for the multi-class problem was a 2-2-3 structure. Three output neurons were required for the three output classes as the networks were tested using a winner-take-all method as described in Section 4.5.1. The results obtained are presented in Table 4.4 and the best, mean and median error rates, as well as mean training time, shown in Fig. 4.11.

The classes overlap, as shown in Fig. 4.1, therefore perfect classification is not possible for this problem. The lowest test set error achieved was 4.0% with median error in the range 5% to 7% and mean error between 5% and 10%. Most of the algorithms had all networks converging (less than 20% error), with the worst having 90% of networks converging. The best mean accuracy of 5.47% was achieved by GDX, which was also among the fastest in terms of training time. The algorithms that were faster than GDX, namely DS-GDX, ROM and ROM2, all had much higher error rates, with mean error more than 7.5%.

Table 4.3 Best results for Pima Indians Diabetes dataset using 8-8-1 SIANNs

Training Algorithm	Activation functions		Performance (% of runs)			Avg Epochs			Test Error				Mean Training CPU time (s)
	Shunt	Out	→ goal	0% err	20% <	All runs	→ goal	0% Errors	Best (%)	Mean (%)	95% CI	Median (%)	
GDM	Lgs	Tnh	0	0	94	710	*	*	17.71	19.05	± 0.20	19.27	48.1
GDX	Tnh	Lgs	0	0	14	195	*	*	17.71	21.03	± 0.29	21.35	13.2
LM	Lgs	Tnh	0	0	58	182	*	*	17.71	19.88	± 0.32	19.79	38.9
LMAM	Lgs	Tnh	0	0	54	80	*	*	17.71	20.22	± 0.36	19.79	19.9
OLMAM	Lgs	Tnh	0	0	44	238	*	*	18.23	20.34	± 0.34	20.31	58.0
DSGDX	Lgs	Lgs	0	0	68	94	*	*	18.75	19.82	± 0.29	19.79	6.4
ROM	Lgs	Lgs	0	0	12	1000	*	*	17.71	21.50	± 0.43	21.35	14.7
ROM2	Lgs	Lgs	0	0	22	1000	*	*	18.75	21.69	± 0.53	21.61	14.7

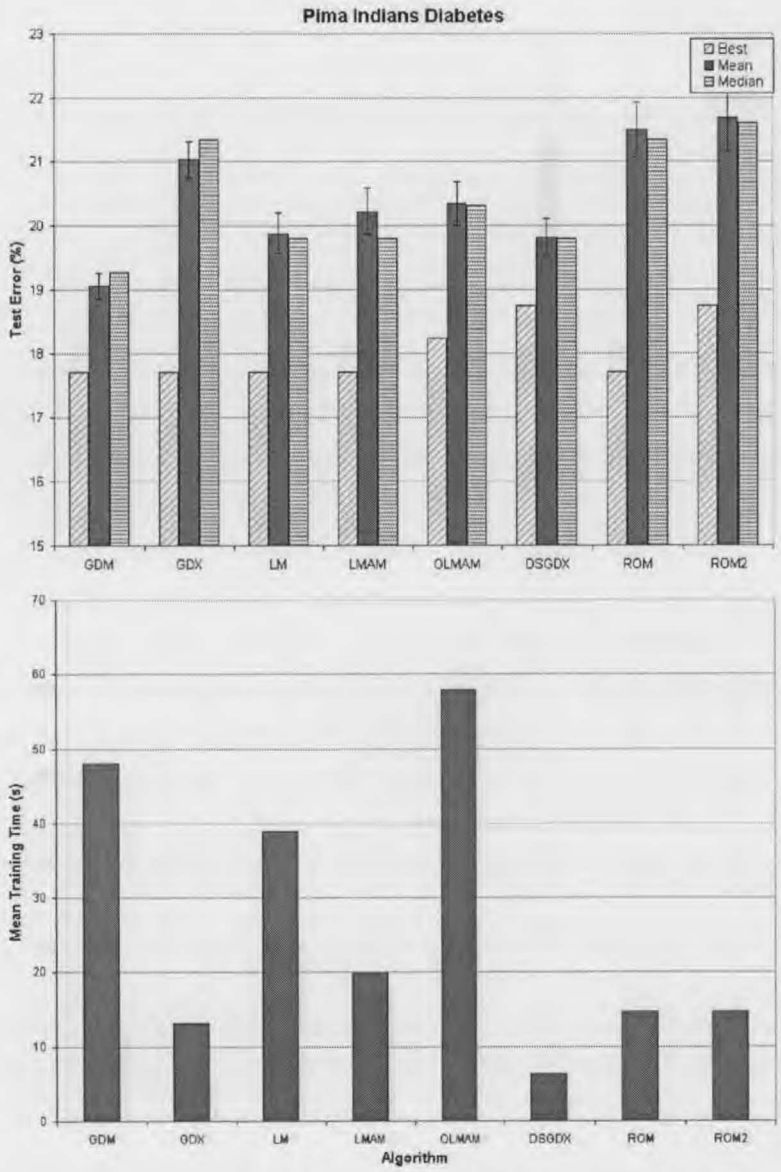


Fig. 4.10: Best case, mean and median classification error for test set and mean training time for Diabetes dataset using 8-8-1 SIANN with various training algorithms.

Table 4.4 Best results for Multi-Class dataset using 2-2-3 SIANN

Training Algorithm	Activation functions		Performance (% of runs)			Avg Epochs			Test Error				Mean Training CPU time (s)
	Sh	Out	→ goal	0% err	20% <	All runs	→ goal	0% Errors	Best (%)	Mean (%)	95% CI	Median (%)	
GDM	Exp	Lgs	0	0	100	999	*	*	4.67	5.73	± 0.23	5.67	51.2
GDX	Exp	Lgs	0	0	100	377	*	*	4.00	5.47	± 0.16	5.33	19.4
LM	Exp	Lgs	0	0	100	228	*	*	4.00	5.69	± 0.26	5.33	103.0
LMAM	Exp	Lgs	0	0	100	163	*	*	4.67	6.13	± 0.28	6.00	74.7
OLMAM	Lgs	Lgs	0	0	100	560	*	*	4.00	5.81	± 0.20	6.00	251.6
DSGDX	Exp	Lin	0	0	90	206	*	*	5.33	9.39	± 2.24	6.67	10.9
ROM	Exp	Lgs	0	0	98	1000	*	*	4.67	7.49	± 1.23	6.67	11.1
ROM2	Exp	Lgs	0	0	96	1000	*	*	4.00	8.33	± 1.68	6.33	11.3

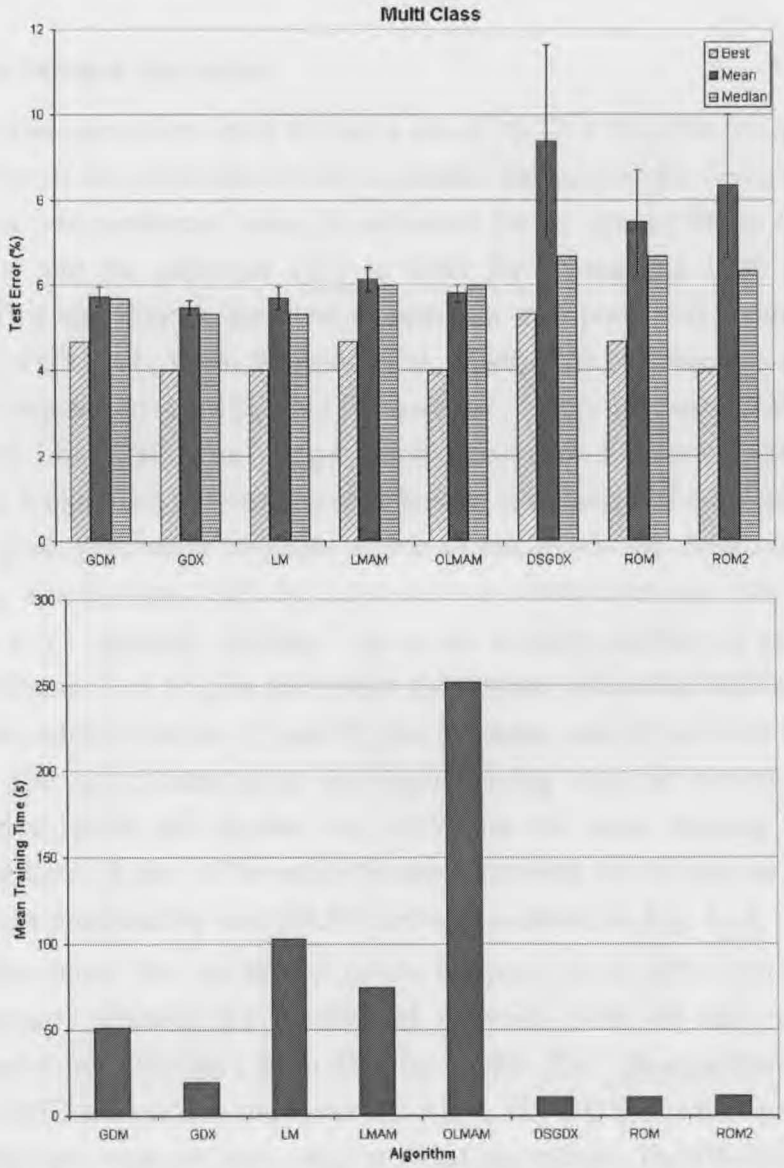


Fig. 4.11: Best case, mean and median test classification error and mean training time for Multi-class dataset using 2-2-3 SIANN with various training algorithms.

From Fig. 4.11 it can be seen that there is not much difference between the accuracy of the GDX and the GDM, LM, LMAM and OLMAM algorithms, though the training times are between 2.5 and 13 times longer than GDX for the latter algorithms. GDX and GDM had the best overall performance when factoring the training time on top of the accuracy.

For the Multi-class problem there appears to be a trend in the activation functions achieving the best results, as 7 out of 8 had the exponential function as the shunting layer activation function. Similarly, 7 out of 8 algorithms had a logistic sigmoid output layer activation function. The exceptions were the OLMAM and DSGDX algorithms that had log sigmoid shunting and linear output activation functions, respectively.

4.6.2.5 The Sunspot time series

The Sunspot time series was used to train a set of 10-10-1 SIANNs, using the scaled sunspot counts of 10 consecutive years to predict the number for the next year. The Sunspots data was partitioned using the subseries for the years 1700 to 1920 to train the networks, and the subseries 1921 to 1965 for testing and 1966 to 1989 for validation. This was done to facilitate comparison with published results (Nikolaev & Iba, 2003; Park et al., 1996; Weigend et al., 1990). The performance metrics used are the mean square error (MSE) and the *average relative variance* (ARV), defined in Section 4.5.7 by (4.30). The best performing activation function results are shown in Table 4.5. Columns 4 to 6 in this case are the percentage of networks achieving the training goal; percentage networks where all test results are within tolerance and networks for which at least 80% of test points are within tolerance. The tolerance in this case is ± 0.1 (scaled). Column 7 gives the average number of epochs for all networks. Columns 8 to 11 give the lowest and median values for MSE and ARV for the test points while columns 12 and 13 give the mean and 95% CI for the ARV for test points. The last column gives the mean training time per network. Fig. 4.12 shows the best, mean and median test ARV and the mean training time for the various algorithms. A plot of the actual Sunspot numbers for the test set range, along with the values predicted by one SIANN network is shown in Fig. 4.13.

The results show that practically all the networks have 80% of the test points within tolerance, whereas the number of networks with all test points within tolerance varies from 8% for LM to 42% for GDM. The LM algorithm got the best median test MSE and median and mean test ARV. The GDX was the fastest of all the algorithms but the worst accuracy, bar the ROM algorithms. The DS-GDX algorithm and the LM algorithm with its variants, LMAM and OLMAM, all achieved similar test ARV.

Table 4.5 Best results for Sunspots dataset using 10-10-1 SIANN

Training Algorithm	Act-fns		Performance (% of runs)			Avg Epochs	Test MSE		Test ARV				Mean Train CPU time (s)
	Shunt	Out	→ goal	all in tol	80% tol		Best	Median	Best	Median	Mean	95% CI	
GDM	Tnh	Lin	0	42	94	886	0.0094	0.0117	0.113	0.140	0.161	± 0.025	37.3
GDX	Tnh	Lin	0	14	98	147	0.0085	0.0134	0.102	0.161	0.174	± 0.020	6.8
LM	Lgs	Lin	0	8	100	54	0.0075	0.0093	0.090	0.111	0.112	± 0.003	9.8
LMAM	Lgs	Lin	0	26	98	207	0.0075	0.0101	0.090	0.121	0.125	± 0.006	36.2
OLMAM	Lgs	Lin	0	22	100	89	0.0072	0.0095	0.086	0.114	0.117	± 0.005	15.6
DSGDX	Lgs	Lin	0	40	100	161	0.0077	0.0097	0.096	0.121	0.119	± 0.002	7.6
ROM	Lgs	Lgs	0	18	98	1000	0.0105	0.0208	0.126	0.250	0.286	± 0.034	12.8
ROM2	Lgs	Lgs	0	28	100	1000	0.0105	0.0214	0.126	0.256	0.271	± 0.027	12.8

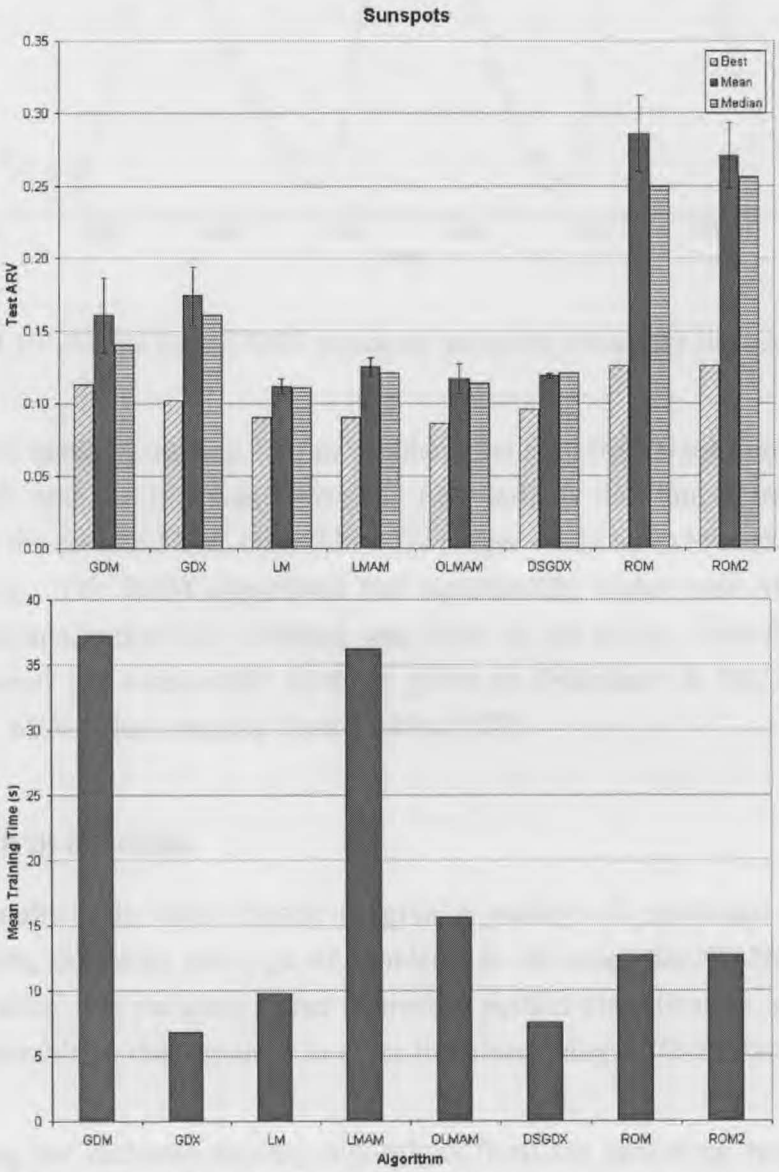


Fig. 4.12: Best case, mean and median test ARV and mean training time for various training algorithms for Sunspots dataset.

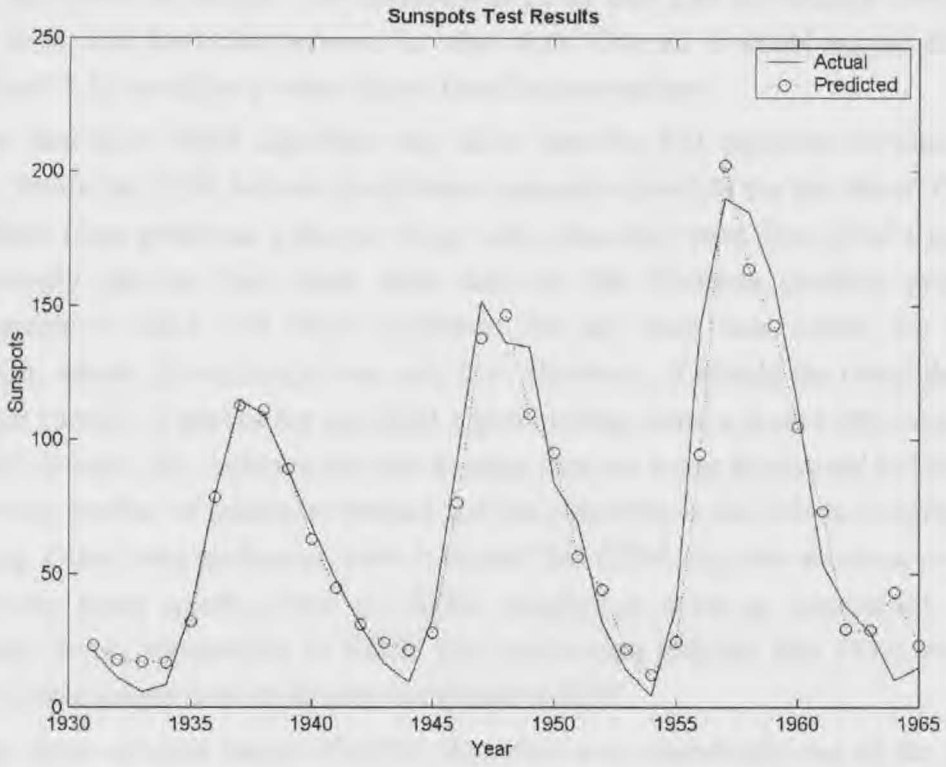


Fig. 4.13: Actual and SIANN predicted sunspots values for the test set.

In terms of speed of training for this problem, the DS-GDX algorithm was second best to GDX, and the LM was third. The LM variants had longer training times compared to the standard LM, OLMAM 50% longer while LMAM took almost four times as long. The ROM algorithms had significantly higher test ARV, but the number of points within the tolerance was close to the others. Overall, the ARV figures obtained are comparable to those given in (Nikolaev & Iba, 2003), who reported test ARV values ranging from 0.086 to 0.229.

4.6.3 Analysis of results

The benchmarks tests were chosen to give a variety of problems in terms of dimensionality, difficulty and type of problem. In all cases the SIANNs could be trained to 'solve' the problem; either achieving perfect classification, or achieving results comparable to that reported in other literature using different types of neural networks.

Comparing the different training algorithms from the preceding results, certain general trends appear. The LM-trained networks are consistently among the most accurate. The time taken to train with the LM algorithm tends to be average to high, though it was never the longest. The LM variants, LMAM and OLMAM, had comparable or worse accuracy than the 'standard' LM algorithm. In terms of training

time, the results are mixed. The LMAM was better than LM and OLMAM worst in some tests, and the order reversed in other tests. Overall it would appear that the 'standard' LM would be a better choice than the two variants.

The first-order GDX algorithm was faster than the LM algorithm in almost all cases. While the GDX had similar or better accuracy than LM for the Breast Cancer and Multi-class problems, it did not do as well in the other tests. The GDM algorithm surprisingly got the best mean error rate for the Diabetes problem and was comparable to GDX and other algorithms for the other tests except the Parity problem, where the accuracy was very low. However, it should be noted that the average number of epochs for the GDM algorithm was always close to the maximum of 1000 epochs. This indicates that the training runs are being terminated because the maximum number of epochs is reached and the algorithm is not able to complete the training. Other tests performed have indicated that GDM requires about an order of magnitude more epochs (limit of 10,000 epochs) in order to consistently reach accuracy levels comparable to GDX. The results also indicate that GDM requires significantly longer time to training compared to GDX.

The direct-solution based DS-GDX algorithm was consistently one of the fastest algorithms in terms of computation time. In terms of accuracy, it was comparable to the LM algorithm, except for the Breast Cancer and Multi-class problems. The stochastic algorithms ROM and ROM2 had the worst error rates in the majority of tests. The exceptions are the 3-bit Parity test, where GDM come out worst, and the Multi-Class problem, where DS-GDX had a higher error rate. The ROM algorithms trained for the full 1000 epochs allowed in all cases, except for a few networks that reached the error goal with the 3-bit Parity problem. Further tests showed that these algorithms would go on for 10,000 epochs, if allowed, with no significant improvement in error rates, except for the 3-bit Parity problem.

In terms of training time, the ROM and ROM2 training time varies from the longest for the 3-bit parity to almost the fastest for the Sunspots problem. There appears to be a trend that the comparative training time improves as the number of training examples increases. This is probably due to the fact that, for the stochastic algorithms, only the objective function is calculated from the training examples, not the actual weight update. This would give the stochastic algorithms an advantage in terms of training time as the number of training examples increases, though not in terms of accuracy.

From the 'best-case' results shown, there do not appear to be any clear trends in the choice of activation function across the various test problems and algorithms, except in the case of the Multi-Class problem. Tables B.1 to B.5 in Appendix B show the mean test classification error (or mean test ARV for Sunspots) for all combinations of activation functions and training algorithm. These results show that there are large differences in terms of accuracy, even for the same problem and

training algorithm using different activation functions. This means that though the 'best-performing' activation function combinations may be achieving similar results, these combinations need to be determined experimentally.

The previous analysis of training algorithm performance was also based on these 'best-performing' combinations. The average error rates across all activation functions for a given training algorithm and problem, shown in Appendix B, also indicate that there are differences in the performance of various training algorithms for a given problem.

The question then is whether these differences can be considered significant, and how to compare performance across the various tests. In order to determine if there are statistically significant differences in performance across the activation functions and training algorithms over all the tests, statistical analysis was performed on the full set of results obtained.

The statistical method for showing that significant differences do exist across a number of samples is to test the null hypothesis H_0 that the k independent samples have equal means (or are from identical populations). The alternative hypothesis H_1 is that they have different means or are from different populations. The k independent samples in this case would be the nine different activation function combinations tested or, alternatively, the eight different training algorithms. The statistical test chosen was the **Kruskal-Wallis H test** (Walpole et al., 1998). This test is a non-parametric procedure for testing the equality of means while avoiding the assumption that the samples were selected from normal populations. The distribution of means across the various functions, algorithms and benchmarks may not be normal, hence the selection of a test that avoids that assumption.

The procedure for applying the test is as follows. For each of the benchmarks tests, the mean errors as shown in Appendix B were ranked from 1 to 72 in ascending order. For cases where there is more than one sample with the same value, the rank will be the average of the rank positions. For example if there are two samples in equal 5th position, they will both be ranked 5.5 and 3 samples in equal 13th position will be ranked 14 (average of 13, 14 and 15).

The h statistic for the particular benchmark is calculated using the formula

$$h = \frac{12}{n(n+1)} \sum_{i=1}^{k} \frac{r_i^2}{n_i} - 3(n+1) \qquad (4.33)$$

where

n_i is the number of observations in the ith sample ($i = 1,2,\ldots,k$)

r_i is the sum of the ranks of the n_i observations in the ith sample

$n = n_1 + n_2 + .. + n_k$ is the total number of observations.

The statistic h is approximated very well by a *chi*-squared distribution with k-1 degrees of freedom when H_0 is true and if each sample consists of at least 5 observations (Walpole et al., 1998). The null hypothesis H_0 is therefore rejected with 95% confidence if the calculated value of h is greater than the value for $\chi^2_{0.05}$ with degrees of freedom ν = 8 when comparing the nine different activation function combinations used.

The tables of rankings for the various benchmarks are shown in Tables B.6 to B.10 in Appendix B, along with a ranking of the activation function combinations based on the sum of rankings across the rows. The calculated h values for each benchmark are given in Table 4.6, along with the critical χ^2 value and an overall h statistic. The h values that are larger than the critical value are shown in bold. The 'overall' h value was obtained by summing the rankings across all the benchmarks (shown in Table B.11), then ranking the sums from 1 to 72 (as shown in Table B.12 in Appendix B) and finally calculating h.

From Table 4.6, it can be seen that the null hypothesis quite clearly holds true for all the benchmarks except for the Multi-Class problem. In the case of the Multi-Class problem, H_0 is rejected with greater than 95% confidence, indicating that there is a significant difference between the means. This bears out the observation that the combination of exponential shunting and log sigmoid output activation functions gives the best performance with most of the algorithms for the Multi-Class problem. For all the other benchmark tests there is no statistically significant difference in performance across the various activation function combinations.

The next step was to use Tables B.12, and sum down along the columns to compare the means for the various training algorithms using the same procedure. This time the critical chi-squared value is $\chi^2_{0.05}$ with ν = 7 since there are eight algorithms being compared. Table 4.6 shows that the null hypothesis H_0 is rejected with greater than 95% confidence for the overall ranking as well as for all the individual benchmark tests. These results confirm the conclusion, obtained by visual observation of the graphs and tables, that there are significant differences in the accuracy achieved by the various training algorithms.

Table 4.6 The h values calculated for all benchmark tests

Benchmark Test	Comparison across activation functions		Comparison across training algorithms	
	h Calculated	Critical value	h Calculated	Critical value
3-bit Parity	8.572	15.507	**31.716**	14.067
Breast Cancer	7.922		**40.670**	
Diabetes	2.235		**30.099**	
Multi-Class	**28.857**		**27.689**	
Sunspots	6.669		**54.102**	
OVERALL	2.218		**51.023**	

Table 4.7 Overall ranking of training algorithms for SIANNs

Rank	Training Algorithms	Composite Sum of ranks
1	LM	864.5
2	OLMAM	1039
3	LMAM	1412.5
4	DS-GDX	1578.5
5	GDX	1543
6	GDM	1732
7	ROM2	2474
8	ROM	2496.5

Table 4.8 Overall ranking of activation functions for SIANNs

Rank	Activation Functions		Composite Sum of ranks
	Shunting	Output	
1	Exp	Lgs	1275.5
2	Lgs	Lgs	1293
3	Tnh	Lgs	1302
4	Lgs	Lin	1496
5	Exp	Lin	1474.5
6	Tnh	Lin	1493.5
7	Exp	Tnh	1538
8	Tnh	Tnh	1628
9	Lgs	Tnh	1639.5

Tables 4.7 and 4.8 shows the 'overall' rankings for the training algorithms and the activation functions obtained by summing the ranks across all benchmarks and then ranking the sums, as given in Table B.12. The composite sum of ranks across all benchmarks tests, as shown in Table B.11, is also presented, to give an indication of how 'far apart' the rankings are.

It should be noted that the rankings for the activation functions are given as an indication only, as the preceding tests show that the overall differences in performance due to the activation functions are not statistically significant. The composite sum of rankings bears this out, as the sums are quite close to each other, with the biggest gap being between the third and fourth ranked combinations. Also, the fourth ranked Lgs-Lin combination has a larger sum than the sixth ranked Tnh-Lin combination. This means that the ranking as done using Table B.12 gives a different order of ranking than the ranking given in Table B.11 that was based purely on the sum of the individual benchmark ranks. This is due to the fact that the results are actually too close to clearly differentiate and rank them. An interesting point to note, however, is that there appears to be a trend with the output activation functions. The combinations with the logistic sigmoid (*lgs*) output function perform the best followed by the linear output combinations, and the hyperbolic tangent sigmoid (*tnh*) combinations coming in last. There is no such trend apparent when looking at the 'best-performing' combinations though, with a number of combinations the tan sigmoid output coming out best for different algorithms and problems.

For the training algorithms, the LM algorithm is the highest ranked algorithm followed by its variants the OLMAM and LMAM algorithms. The direct solution DS-GDX algorithm comes in next followed by the first order GDX and GDM algorithms, with the adaptive learning rate GDX algorithm ahead of the simpler

GDM algorithm. Not surprisingly the stochastic ROM and ROM2 algorithms are ranked the lowest. This ranking follows the same pattern as the general trends observed using the 'best performing' activation functions. Looking at the sum of ranks, there is a distinct difference between the algorithms, except between DS-GDX and GDX and between ROM and ROM2. In fact GDX has a lower sum than DS-GDX, but, as with the activation functions, when there is no significant difference the various ranking mechanisms sometimes produce different orderings.

The rankings here are based on the accuracy of the networks on the test set, but other factors such as the time required for training and computational complexity also need to be considered when selecting a training algorithm for a given problem.

4.7 Conclusion

This chapter describes the development of a number of training algorithms for SIANNs, including the derivation of the appropriate equations. The test methodology has been presented and a number of benchmarks tests described. SIANNs have been applied to these benchmark problems, trained using the algorithms derived, and the experimental results presented. The results obtained are comparable to those obtained by other types of neural networks, showing that the SIANNs can be trained successfully on a variety of problems.

The effect of the limit on the shunting neuron denominator during training has been investigated and analysed. Inferences have also been drawn, from these results, on the effect of the combination of activation functions and performance of training algorithms. It can be concluded that the choice of activation functions has a significant effect on the accuracy of the trained network, but there is no single combination that works best across all the training algorithms and problems. The optimum combination for a particular problem and training algorithm therefore has to be determined experimentally. The differences in training algorithm performance, on the other hand, are statistically significant, and an overall ranking based on the accuracy has been produced. It should be noted, however, that there is often a trade-off between accuracy and training time, and that the relative performance of the algorithms is still problem dependent.

In general, the experimental results show that SIANNS can be applied successfully to classification and prediction problems using the training algorithms developed. This means that SIANNs are a viable class of neural networks that can be applied to various types of problems.

Chapter 5

The Quadratic Neural Network Algorithm

5.1 Introduction

The training of feedforward neural networks is based on the minimisation of an objective function related to the output error. The general strategy for supervised learning is based on combining a quickly convergent local method with a globally convergent one (Battiti, 1992). The local methods are based on local models of the generally complex error surface. Most algorithms are based on a linear (first order) model or quadratic (second order) model. Quadratic methods tend to have faster convergence, though they occasionally get trapped in local minima.

Second order methods rely on minimising a quadratic approximation to the error function, $E(\mathbf{w})$, that uses the first three terms of the Taylor-series expansion about the current point, $\mathbf{w}_o$, given by

$$E(\mathbf{w}+\Delta\mathbf{w}) \approx E(\mathbf{w}) + \mathbf{g}^T\Delta\mathbf{w} + \tfrac{1}{2}\Delta\mathbf{w}^T\mathbf{H}\Delta\mathbf{w} \qquad (5.1)$$

where $\Delta\mathbf{w}$ is the weight change, $\mathbf{g}$ is the gradient vector and $\mathbf{H}$ is the Hessian matrix.

Solving this equation yields the optimal change in the weight matrix, given by $\Delta\mathbf{w}_{opt} = \mathbf{H}^{-1}\mathbf{g}$. However, the calculation of the Hessian $\mathbf{H}$ and its inverse is computationally prohibitive, thereby leading to approximation methods being investigated. There are also problems where the Hessian is not positive definite, is singular, or ill-conditioned (Battiti, 1992). The matter is further complicated if constraints are imposed on the solution, as is the case for Shunting Inhibitory Artificial Neural Networks (SIANNs), where certain weights need to be constrained.

The *Quadratic Neural Network* (QNN) algorithm is a novel second order method that uses a recurrent "neural network" to determine the minimum point of the objective function to be minimised. It is based on work using recurrent neural networks for bound constrained quadratic minimisation proposed by Bouzerdoum and Pattison (Bouzerdoum & Pattison, 1993a, 1993b).

This chapter presents the development of the Quadratic Neural Network training algorithm and a number of variants, and their implementation in training SIANNs on a number of benchmark problems. The following section outlines the development of the algorithm and its implementation for training feedforward neural networks. The third section covers the adaptive determination of parameters, followed by the section on constraining the QNN update. Section 5.5 presents experimental results comparing the performance of the various QNN-based algorithms with other algorithms and analysis of the results obtained. Conclusions are provided in Section 5.6.

5.2 Development of the QNN Algorithm

This section firstly outlines the development of the method of using recurrent neural networks for bound constrained quadratic minimisation upon which the QNN algorithm is based (Bouzerdoum & Pattison, 1993b). This is followed by sub-sections on the recursive equations used to model the recurrent neural network, and the method of applying this to the practical training of neural networks in general, and SIANNs in particular.

5.2.1 Algorithm Formulation

Bouzerdoum and Pattison's method (Bouzerdoum & Pattison, 1993a, 1993b) uses a recurrent neural network to solve the bound constrained quadratic optimisation problem

$$\min_{\Delta \mathbf{w}} \{E(\mathbf{w}_o + \Delta \mathbf{w}) : \mu \le \Delta \mathbf{w} \le \upsilon\} \tag{5.2}$$

with $\mu, \upsilon \in \mathbf{R}^s$ for $\mathbf{w} \in \mathbf{R}^s$.

In order to ensure the constraints are always satisfied, let

$$\Delta \mathbf{w} = \mathbf{p}(\mathbf{u}) = \mathbf{B}\mathbf{f}(\mathbf{u}) \tag{5.3}$$

where $\mathbf{f} : \mathbf{R}^s \to \mathbf{R}^s$ is a piecewise-linear function defined as

$$f_i(\mathbf{u}) \equiv f_i(u_i) = \begin{cases} \varsigma_i & u_i < \varsigma_i \\ u_i & u_i \in [\varsigma_i, \xi_i] \\ \xi_i & u_i > \xi_i \end{cases} \tag{5.4}$$

The n-dimensional vector $\mathbf{u}$ is permitted to vary without constraint, $\mathbf{B}$ is an n-by-n positive diagonal matrix that serves as a *preconditioner*, and $\varsigma, \xi \in \mathbf{R}^n$ are the constraints μ, υ on $\Delta\mathbf{w}$ mapped onto corresponding values of $\mathbf{u}$ such that $\varsigma = \mathbf{B}^{-1}\mu$ and $\xi = \mathbf{B}^{-1}\upsilon$. By identifying $\mathbf{p}(\mathbf{u})$ such that $\Delta\mathbf{w}$ is confined to the constraint region, the problem now becomes an *unconstrained* minimisation of the objective function $M(\mathbf{u})$ over $\mathbf{u}$ where

$$M(\mathbf{u}) = \mathbf{g}^T\mathbf{p}(\mathbf{u}) + \tfrac{1}{2}\mathbf{p}(\mathbf{u})^T\mathbf{H}\mathbf{p}(\mathbf{u}) \tag{5.5}$$

Consider now the single-layered recurrent neural network whose state vector is defined by the differential equation

$$\frac{d\mathbf{u}}{dt} = -\mathbf{g} - \mathbf{A}\mathbf{u} - \mathbf{C}\mathbf{f}(\mathbf{u}) \tag{5.6}$$

where $\mathbf{g}$ is the external input, $\mathbf{f}(\mathbf{u})$ is the network output, $\mathbf{C}$ is the lateral feedback matrix with zero diagonal entries, and $\mathbf{A}$ is a positive diagonal matrix representing the passive decay rate of the state vector.

To map the constrained quadratic problem onto the neural network, we set

$$\mathbf{A} = \mathrm{diag}(\mathbf{HB}) \tag{5.7}$$

$$\mathbf{C} = \mathbf{HB} - \mathbf{A} \tag{5.8}$$

where diag(.) selects the diagonal elements of its matrix argument.

The desired output $\Delta\mathbf{w} = \mathbf{B}\mathbf{f}(\mathbf{u})$ is obtained from the network output $\mathbf{f}(\mathbf{u})$ through multiplication with the diagonal preconditioner $\mathbf{B}$. Bouzerdoum & Pattison (Bouzerdoum & Pattison, 1993a) showed that, provided the matrix $\mathbf{H}$ is positive definite, the neural network defined by (5.6) has a unique equilibrium point $\mathbf{u}^*$ which is mapped by $\mathbf{p}$ onto $\Delta\mathbf{w}^*$, the optimal constrained weight update to the minimum of $E(\mathbf{w})$. They have also shown that the network is globally convergent to this equilibrium point.

The *spectral condition number* of a matrix is defined as the ratio of the maximum to the minimum singular values of the matrix. If the state-feedback matrix has a large condition number, then numerical computations are susceptible to round-off errors and errors in the weights of the state-feedback matrix. Preconditioning is used to keep the condition number small. It has also been shown that preconditioning speeds up convergence (Bouzerdoum & Pattison, 1993b).

For the system in question, a simple choice for the preconditioner matrix **B** is

$$b_{ii} = \frac{\alpha}{h_{ii}} \qquad \alpha > 0 \tag{5.9}$$

where b_{ii} and h_{ii} are the diagonal elements of **B** and **H** respectively. The choice of preconditioning has the added advantage of simplifying the expression for matrix **A**, which then simply becomes

$$\mathbf{A} = \alpha \mathbf{I} \tag{5.10}$$

where **I** is the identity matrix.

The matrix $\mathbf{C} = \mathbf{HB} - \mathbf{A}$ can then be defined by

$$c_{ij} = \begin{cases} \alpha \dfrac{h_{ij}}{h_{jj}} & i \neq j \\ 0 & i = j \end{cases} \qquad i, j \in 1,\ldots,n \tag{5.11}$$

5.2.2 Simulating the Recurrent Neural Network

In this training algorithm, the operation of the recurrent network for the quadratic minimisation is approximated by a discrete time recursive equation. At each training epoch, a recurrent neural network is 'constructed' with constraints based on the state of the network being trained at that point. The "recurrent network" modelled by the recursive equation will return the optimal weight update for that epoch and the network being trained will have its parameters updated using (5.3).

The differential equation (5.6) can be approximated by

$$\frac{\mathbf{u}(k+1) - \mathbf{u}(k)}{d} = -\mathbf{g} - \mathbf{A}\mathbf{u}(k) - \mathbf{C}\mathbf{f}(\mathbf{u}(k)) \tag{5.12}$$

$$\therefore \mathbf{u}(k+1) = \mathbf{u}(k) - d\left(\mathbf{g} + \mathbf{A}\mathbf{u}(k) + \mathbf{C}\mathbf{f}(\mathbf{u}(k))\right) \tag{5.13}$$

where d is the discrete 'time-step'.

The recursive equation (5.13) is iterated a finite number of times to obtain an approximate optimal value of **u**, then the update of the weights, $\Delta \mathbf{w}$, is calculated from (5.3). The recursive equation can be iterated a fixed number of times, because even if the weight update is sub-optimal, the overall effect of any error is not critical since the process will be repeated for a number of epochs.

5.2.3 Applying the QNN Algorithm to neural network training

The recursive equations as they stand require the evaluation of the Hessian matrix to determine **C**, as given in (5.8). In practical implementations, the computational cost of calculating the Hessian matrix is too high, so approximations of the Hessian are used. The QNN algorithm has been implemented for MLPs (Arulampalam & Bouzerdoum, 2001b, 2002b) based on the Levenberg-Marquardt (LM) approximation (Hagan & Menhaj, 1994); that is, the same approximations for the gradient and Hessian based on the Jacobian as in LM have been used:

$$\mathbf{g} \approx \mathbf{J}^T(\mathbf{w})\mathbf{e}(\mathbf{w}) \tag{5.14}$$

$$\mathbf{H} \approx \mathbf{J}^T(\mathbf{w})\mathbf{J}(\mathbf{w}) + \mu\mathbf{I} \tag{5.15}$$

where $\mathbf{J}(\mathbf{w})$ is the Jacobian matrix, $\mathbf{e}(\mathbf{w})$ is the vector of residuals (errors) for the training set, $\mathbf{I}$ is the identity matrix and μ is a variable parameter that determines the trust region.

This approximation of the Hessian has been used instead of the Gauss-Newton approximation ($\mathbf{H} \approx \mathbf{J}^T(\mathbf{w})\mathbf{J}(\mathbf{w})$) to overcome the problems of rank deficiency, since neural network training problems are intrinsically ill-conditioned (Haykin, 1999, p. 235), as well as the requirement of the QNN algorithm that the Hessian be positive definite. Tests with the Gauss-Newton approximation for the QNN algorithm resulted in non-convergence due to the above-mentioned problems.

The only difference between the implementation of the LM and QNN algorithms is that the step where the change in weights is calculated with the matrix inversion in LM (refer to Section 2.8.3) has been replaced with the "recurrent neural network", i.e. the recursive equation given in (5.13).

5.2.4 Determining 'optimum' parameters for the QNN algorithm

The parameters that affect the QNN algorithm are the constant for the preconditioner matrix, α, the discrete time-step, d, and i, the number of iterations to update the recursive equation. One important observation that was made during these experiments was that the product αd could be taken as one parameter for the algorithm since the parameters α and d had inverse effects, For example, setting $\alpha = 1$ and $d = 0.1$ produces exactly the same results as $\alpha = 2$ and $d = 0.05$. As such, the term α was fixed at 1 and only the d and i parameters were varied.

In order to find an approximate 'optimum' value for these parameters, SIANNs were trained on the Wisconsin Breast Cancer and Pima Indians Diabetes datasets using the QNN algorithm with varying d and i values. Fig. 5.1 shows the mean error and mean training time as the time-step, d, is increased from 0.01 to 2.0. Fig. 5.2

shows the same as the number of iterations, i, is increased from 5 to 100. The minimum error achieved is also shown for the Diabetes problem, as it is non-zero. Details of the results are given in Tables B.18 and B.19 in Appendix B.

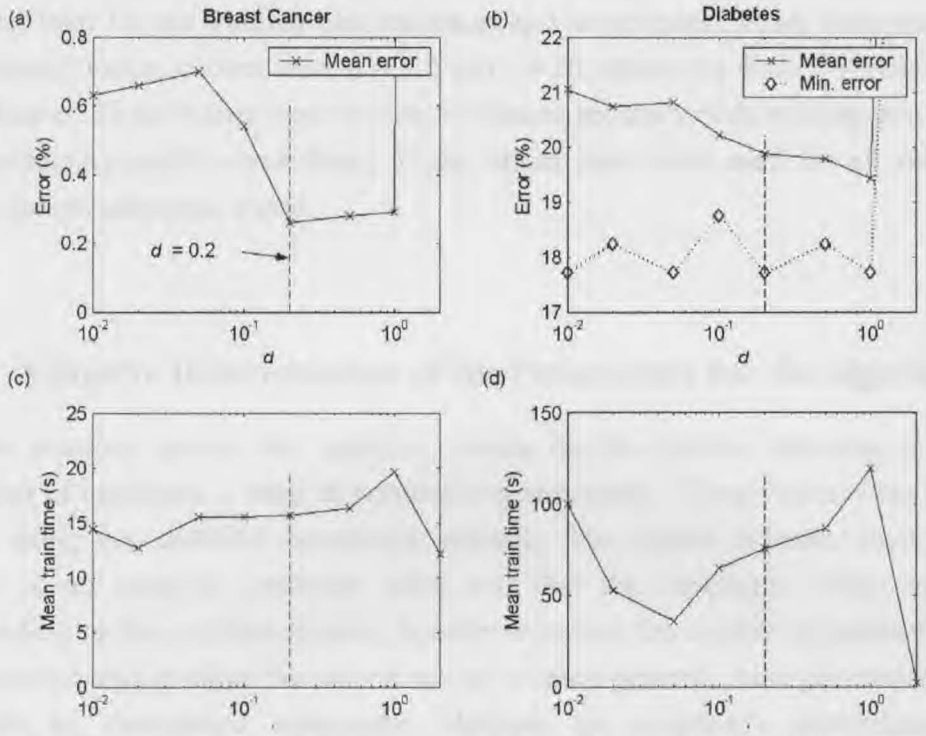


Fig. 5.1: Percentage error and average training time vs. discrete time-step d for SIANNs trained on Breast Cancer (a,c) and Diabetes (b,d) datasets

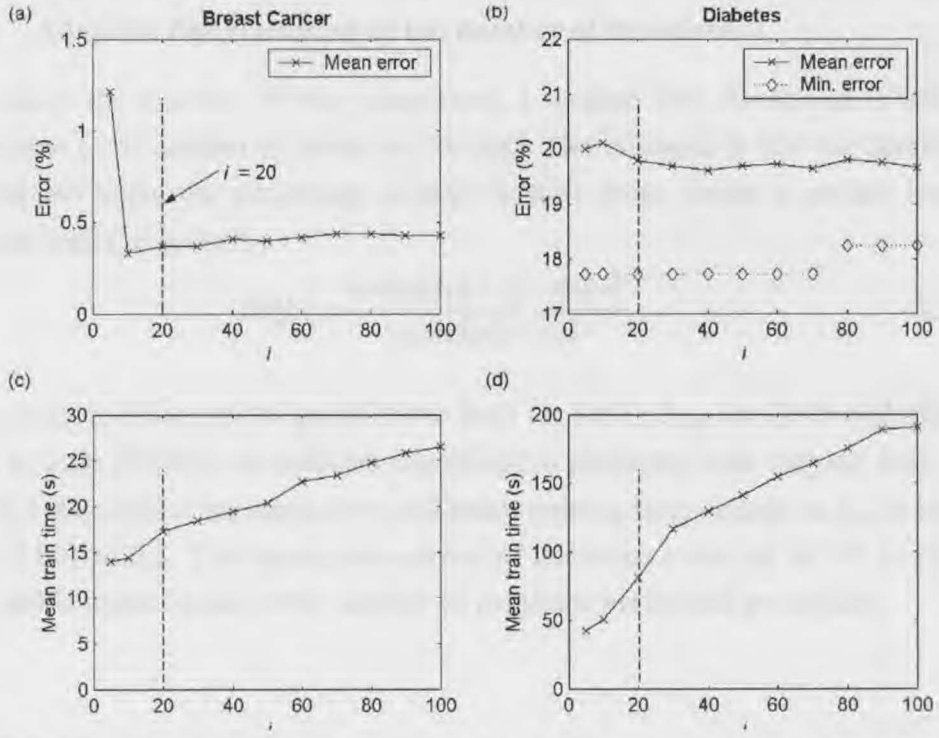


Fig. 5.2: Percentage error and average training time vs. iterations for SIANNs trained on Breast Cancer (a,c) and Diabetes (b,d) datasets

Fig. 5.1 shows that increasing the parameter d above 1.0 results in the algorithm 'blowing-up', with mean error values over 40% for $d = 2.0$. These error values were allowed to go off the graph in order to clearly show the variations for the other values. Fig. 5.2 shows that there is no significant improvement in accuracy for i greater than 10, but training time increases as i is increased. From these results, the 'optimum' values chosen were $d = 0.2$ and $i = 20$, shown by dashed vertical lines in the figures. These values were chosen to balance accuracy with training time, as well as avoiding possible instabilities. These values have been used for all subsequent tests, unless otherwise stated.

5.3 Adaptive Determination of the Parameters for the Algorithm

In the previous section the 'optimum' values for the discrete time-step, d, and the number of iterations, i, were determined experimentally. These values were based on tests using two different benchmark datasets. The results indicate, however, that there is no clear-cut optimum value and that the 'optimum' value may vary depending on the problem at hand. In order to reduce the number of parameters to be determined and to allow the algorithms to be more general, these parameters should ideally be determined adaptively. Methods for adaptively determining these parameters are presented in the following sub-sections.

5.3.1 Adaptive determination of the number of iterations, i

To reduce the number of free parameters, a method was developed to adaptively determine i, the number of iterations for $\mathbf{u}(k)$. The rationale is that the iterations can be stopped when the percentage change in $\mathbf{u}(k)$ drops below a certain limit. The change, $\delta\mathbf{u}(k)$, is given by

$$\delta\mathbf{u}(k) = \frac{\text{norm}\left(\mathbf{u}(k+1) - \mathbf{u}(k)\right)}{\text{norm}\left(\mathbf{u}(k+1)\right)} \tag{5.16}$$

In order to determine a 'good' lower limit for $\delta\mathbf{u}(k)$, δ_{lim}, the QNN algorithm was used to train SIANNs on selected classification problems with varying limit values. Fig. 5.3 shows how the mean error and mean training time change as δ_{lim} is increased from 0.001 to 0.5. The maximum number of iterations I was set at 100 to provide a reasonable upper bound to the number of iterations performed per update.

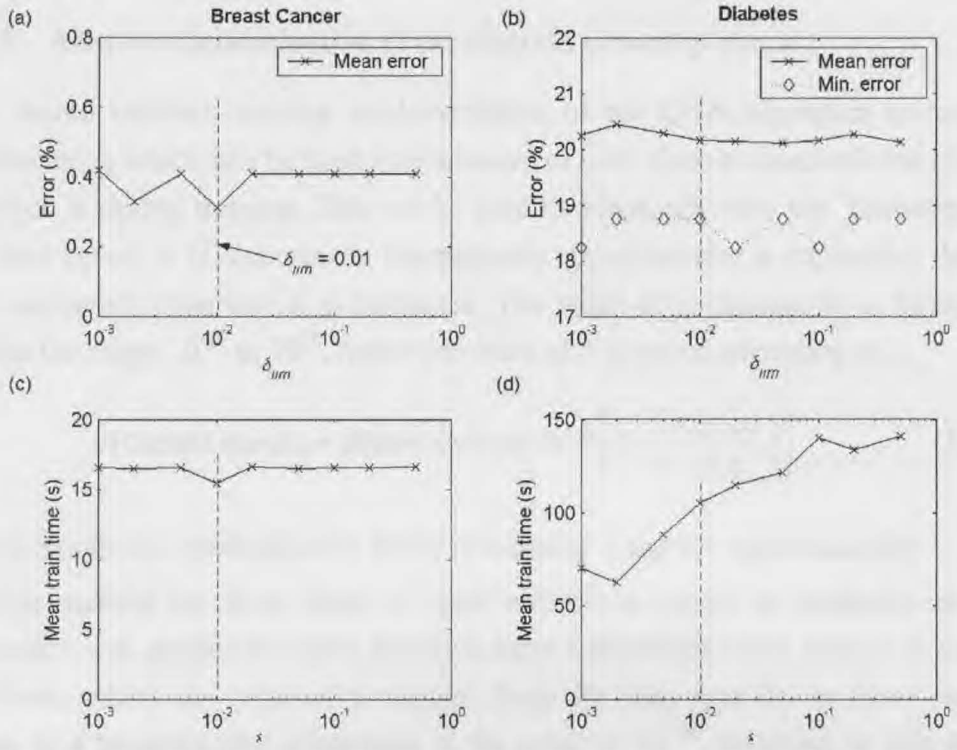


Fig. 5.3 : Percentage error and average training time vs. δ_{lim} , the lower limit for $\delta u(k)$, for SIANNs trained on Breast Cancer (a,c) and Diabetes (b,d) datasets

The results indicate that there is no significant variation in accuracy as δ_{lim} is varied. However, there is a decrease in the training time for the Diabetes problem as the limit is lowered until δ_{lim} = 0.002, after which it increases again. From these results, the chosen lower limit is δ_{lim} = 0.01 (1%), a conservative limit that balances training time with accuracy, while avoiding a very small limit that could potentially lead to excessive iterations with different problems.

This stopping criterion was incorporated into the algorithm primarily as a method of reducing the training time by stopping the iteration of the recursive equation if $\mathbf{u}(k)$ was not changing significantly. The maximum number of iterations was set to 20, which is the 'optimum' value determined in the previous section, and the iterations stopped earlier if $\delta\mathbf{u}(k) < 0.01$, i.e. if the norm of $\mathbf{u}(k)$ changed less than 1%. This would reduce the training time without significantly impacting the weight update. This method has been incorporated into all subsequent tests performed.

5.3.2 Adaptive determination of the discrete time-step size, d

The neural network training implementation of the QNN algorithm contains the parameter μ, which can be used as a measure of how close to quadratic the objective function is during training. This can be used to adaptively vary the 'time-step' term for each epoch, d. If μ decreases, the quadratic approximation is improving, therefore d is increased, otherwise it is decreased. The value of μ changes by a factor of 10 within the range 10^{-10} to 10^{10}, hence the value of d is varied according to

$$d(\text{current epoch}) = d(\text{previous epoch}) * \left(1 - \left(\frac{\log_{10}\mu}{10.1}\right)\right) \qquad (5.17)$$

which results in a multiplicative factor of between 2 and 0.1 approximately.

This method has been found to work well for a variety of problems when the algorithm was applied to MLPs, but does have a drawback when applied to complex problems where the value of μ remains large for long periods. In these cases the value of d becomes tiny, sometimes in the order of 10^{-100}, resulting in long training times without any significant improvement in performance (Arulampalam & Bouzerdoum, 2001b, 2002b).

An alternative method formulated was thus to vary d only when the final value of μ changes compared to previous epoch. The step size would be increased by a constant factor when μ decreases and vice versa, for example

$$d(\text{current epoch}) = \begin{cases} d(\text{previous epoch}) * 1.1 & \mu(current) < \mu(previous) \\ d(\text{previous epoch}) & \mu(current) = \mu(previous) \\ d(\text{previous epoch}) * 0.9 & \mu(current) > \mu(previous) \end{cases} \qquad (5.18)$$

Another alternative is to increase the frequency of the d value update from once every epoch to every time μ is updated. The value of μ is increased by a factor of 10 until the objective function is equal to or lower than previous epoch and then it is decreased by a factor of 10 (refer Section 2.8.3 on LM algorithm).

Other variants include combining the different update times (both once an epoch and every μ update) and varying the update factors. The various combinations of update frequency and update formulae tested are summarised in Table 5.1.

The various methods of varying d were tested on the Wisconsin Breast Cancer and Pima Indians Diabetes benchmark problems. The main objective of these algorithm variants is to minimise the dependence of the results on the initial value of d chosen. To test this, the same sets of networks were trained using the different algorithms with the initial value d_0 set to 0.05, 0.2 and 1. The values were chosen around 0.2 because the 'optimum' value for d as determined in the previous section was 0.2.

Table 5:1 Summary of d update methods for QNN algorithm variants

Algorithm	Update frequency	Update formula
QNN		d fixed
QNN2	Every epoch	$d = d^*\left(1-\left(\frac{\log_{10}\mu}{10.1}\right)\right)$
QNN3	Every μ update	$d = \begin{cases} d^*1.1 & \mu(current) < \mu(previous) \\ d & \mu(current) = \mu(previous) \\ d/1.1 & \mu(current) > \mu(previous) \end{cases}$
QNN5	i) Every epoch ii) Every μ update	i) QNN2 update ii) QNN3 update
QNN6	i) Every epoch ii) Every μ update	i) $d = d^*1.1$ ii) $d = \begin{cases} d^*1.1 & \mu(current) < \mu(previous) \\ d & \mu(current) = \mu(previous) \\ d^*0.7 & \mu(current) > \mu(previous) \end{cases}$
QNN7	i) Every epoch ii) Every μ update	i) $d = d^*1.1$ ii) $d = \begin{cases} d & \mu(current) \le \mu(previous) \\ d^*0.9 & \mu(current) > \mu(previous) \end{cases}$
QNN8	Every epoch	$d = \begin{cases} d^*1.1 & \mu(current) \le \mu(previous) \\ d^*0.8 & \mu(current) > \mu(previous) \end{cases}$
QNN9	Every epoch	$d = \begin{cases} d^*1.1 & \mu(current) < \mu(previous) \\ d & \mu(current) = \mu(previous) \\ d^*0.9 & \mu(current) > \mu(previous) \end{cases}$

The results obtained are shown in Tables 5.2 and 5.3 and in Figs. 5.4 and 5.5. The tables show each variant in the first column, the best performing activation combination for that algorithm, followed by the mean classification error for the three d_0 values in columns 4 to 6. Columns 7 and 8 show the percentage variation of the means for d_0 =0.05 and d_0 = 1.0 from that of the 'basic' d_0 of 0.2. The last three columns show the mean training time for each of the starting d_0 values.

Table 5.2 Results for QNN variant comparison using Breast Cancer dataset

Training Algorithm	Activation functions		Mean Classification Error (%)			Change in error from $d = 0.2$ (%)		Avg. CPU time (s)		
	Shunt	Out	$d_0 = 0.05$	$d_0 = 0.2$	$d_0 = 1.0$	$d_0 = 0.05$	$d_0 = 1.0$	$d_0 = 0.05$	$d_0 = 0.2$	$d_0 = 1.0$
QNN	Exp	Lgs	0.70	0.26	0.29	169.2%	11.5%	15.5	15.7	19.6
QNN2	Tnh	Lgs	0.49	0.43	10.09	14.0%	2246.5%	13.1	10.4	17.4
QNN3	Tnh	Lgs	0.58	0.33	0.41	75.8%	24.2%	27.5	17.4	23.0
QNN5	Lgs	Lgs	0.36	0.24	0.42	50.0%	75.0%	43.1	49.0	253.6
QNN6	Lgs	Lgs	0.37	0.36	0.34	2.8%	-5.6%	24.4	20.0	20.3
QNN7	Exp	Lgs	0.67	0.35	0.32	91.4%	-8.6%	15.3	15.6	21.2
QNN8	Tnh	Lgs	0.66	0.42	0.85	57.1%	102.4%	24.9	15.9	26.2
QNN9	Lgs	Lgs	0.53	0.32	0.17	65.6%	-46.9%	25.6	19.4	20.4

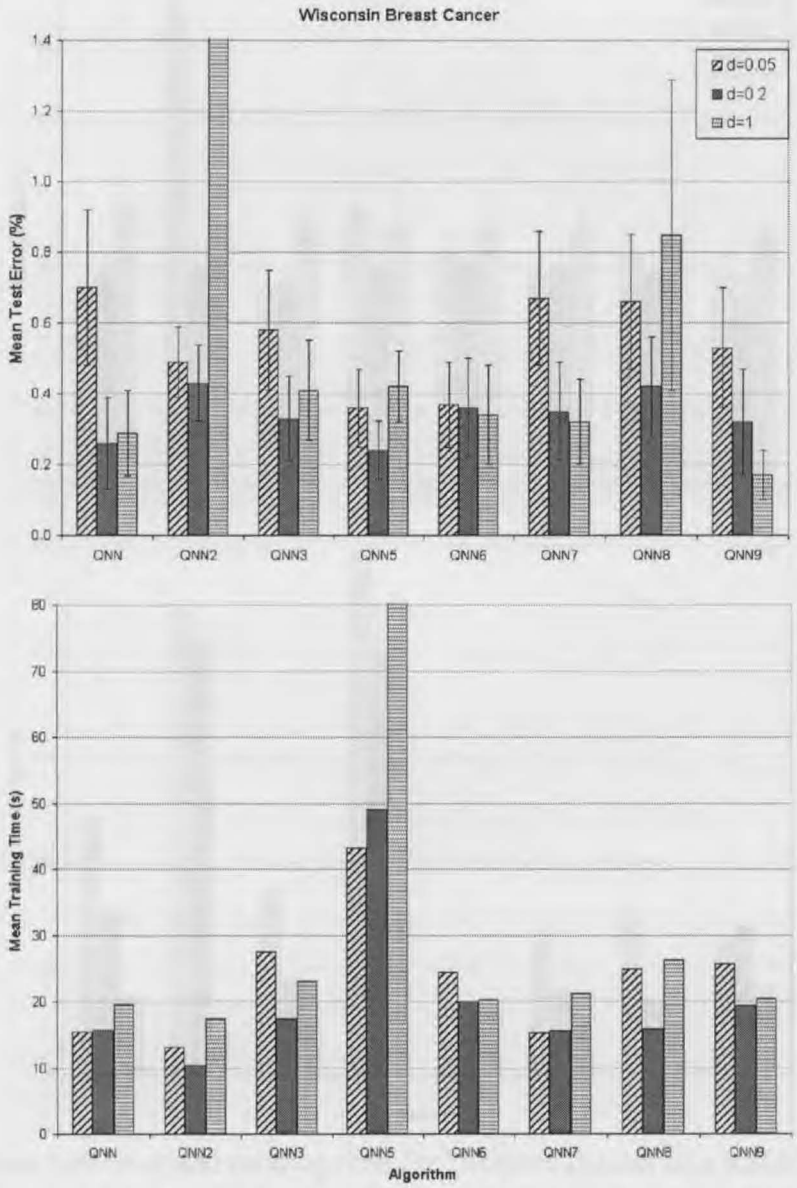


Fig. 5.4: Mean test error and training time for Breast Cancer dataset using SIANNs trained with QNN algorithm variants.

Table 5.3 Results for QNN variant comparison using Diabetes dataset

Training Algorithm	Activation functions		Mean Classification Error (%)			Change in error from $d = 0.2$ (%)		Avg. CPU time (s)		
	Shunt	Out	$d_0 = 0.05$	$d_0 = 0.2$	$d_0 = 1.0$	$d_0 = 0.05$	$d_0 = 1.0$	$d_0 = 0.05$	$d_0 = 0.2$	$d_0 = 1.0$
QNN	Lgs	Tnh	20.82	19.88	19.43	-2.3%	4.7%	121.7	76.7	36.3
QNN2	Exp	Lin	20.36	19.95	23.52	17.9%	2.1%	219.1	203.8	18.5
QNN3	Exp	Tnh	20.66	19.80	19.67	-0.7%	4.3%	86.8	44.6	25.3
QNN5	Lgs	Lin	20.54	19.96	20.76	4.0%	2.9%	245.5	198.6	75.7
QNN6	Lgs	Tnh	20.57	20.02	20.17	0.7%	2.7%	28.9	23.3	28.0
QNN7	Exp	Tnh	20.50	20.05	19.90	-0.7%	2.2%	66.1	40.0	23.3
QNN8	Exp	Tnh	20.42	19.91	20.14	1.2%	2.6%	76.6	34.2	24.4
QNN9	Lgs	Tnh	20.42	19.57	19.85	1.4%	4.3%	63.0	69.4	41.2

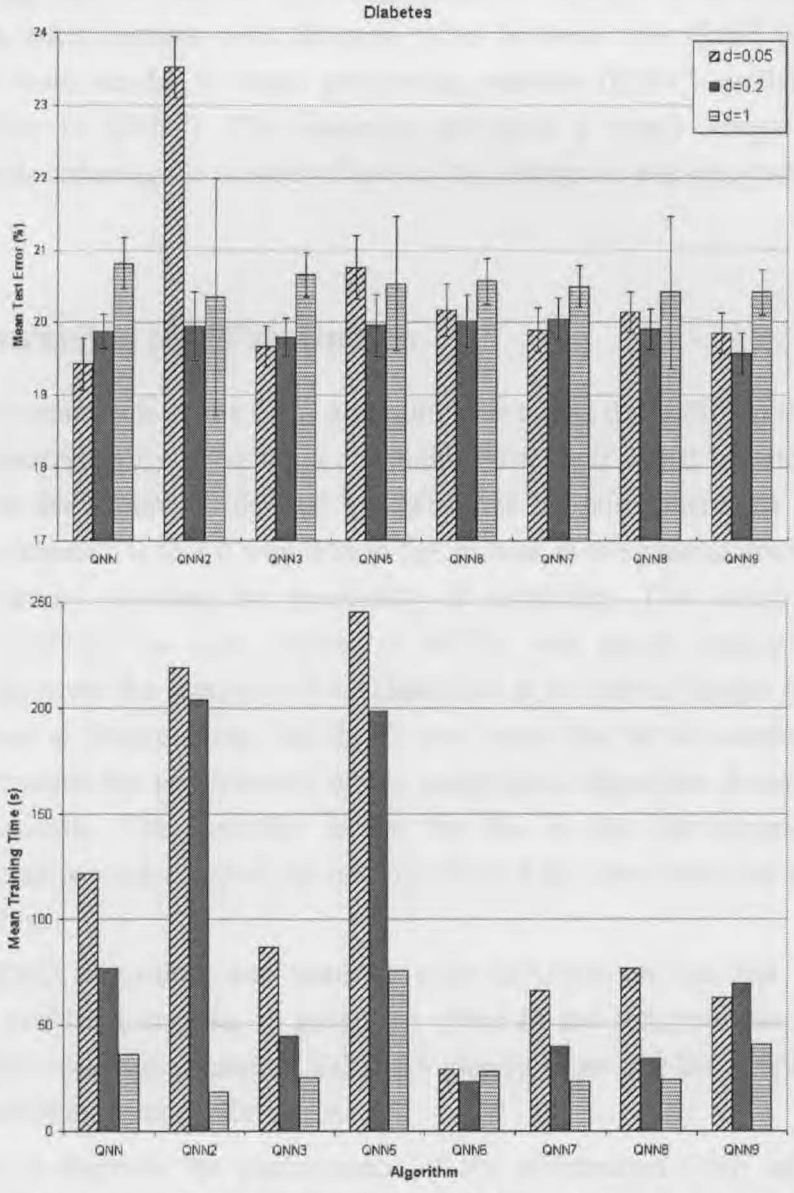


Fig. 5.5: Mean test error and training time for Diabetes dataset using SIANNs trained with QNN algorithm variants.

The percentage variation, in Columns 7 and 8 of the tables above, show that the QNN2 algorithm performance is still highly dependent on the starting value of d. There is a large variation in the means as d_0 is changed, even larger than for the 'standard' QNN with fixed d. This defeats the purpose of varying the d value in the first place. QNN8 also showed a fairly significant variation for the Breast Cancer dataset, as did QNN7. The QNN5 algorithm had long training times and moderate variation in mean error, but had low error rates for $d_0 = 0.2$. The QNN6 variation appears to have the most stable performance across the various d values with no more than 6% variation.

The decision was taken to evaluate a subset of these variants in the subsequent sections along with the standard QNN algorithm, namely QNN3, QNN5, QNN6 and QNN9. The other variants were dropped either because they didn't perform well (QNN2) or were similar to better performing variants (QNN7 similar to QNN6, QNN8 similar to QNN9). This selection maintains a broad comparison of the methods while reducing the number of tests to be performed and reported.

5.4 Constraining the QNN Update

One enhancement made to the QNN algorithm was to use the ability of the algorithm to handle constraints by imposing a constraint on $\mathbf{u}$ such that it is bounded by the function $\mathbf{f}$ to the hypercube defined by 100 times the components of the gradient vector. The rationale is that it would keep the updates in the general gradient descent quadrant, thereby reducing the possibility of instability. This constrained QNN algorithm (QNN-C) has been applied to MLPs, with results indicating that the constraint improves the accuracy of the classifiers at the cost of longer training time (Arulampalam & Bouzerdoum, 2002b). It was found that as the complexity of the problem increases the performance of the constrained algorithm drops, sometimes quite dramatically. The probable reason for this is that the simple constraint condition actually works against the minimisation of the error when the error surface is too complex.

The QNN-C algorithm was used to train SIANNs on the full set of five benchmark problems in order to gauge the effect of the constraint on a variety of problems. The results presented in Table 5.4 clearly show that the simple constraint results in extremely poor performance.

In order to improve the performance of the constrained QNN algorithm, an alternative constraint function was proposed. The alternative update constraint consists of merging the original hypercube, formed by the components of the gradient multiplied by 50, with a smaller hypercube centred on the origin with

boundary value calculated from the norm of the previous update (norm(Δw)) and the value of μ as follows

$$\text{boundary value} = \text{norm}\,(\Delta w(\text{previous epoch}))^{*}\left(1-\left(\frac{\log_{10}\mu}{10.1}\right)\right) \qquad (5.19)$$

The smaller hypercube serves to free the weight update to 'move' in directions other than that of the gradient. The size of the 'freeing' hypercube is determined by the previous step size as well as the quadracity of the update (μ). If the approximation is more linear (large μ) the size of the second hypercube is smaller resulting in the constraint to be closer to the gradient. If the update is closer to quadratic, the size of the hypercube is expanded allowing update in other directions. The algorithm variant using this second hypercube in the constraint is referred to as QNN-C2.

The effects of this second constraint can be seen in the results that follow, where both the constrained and various unconstrained versions of the QNN algorithm are compared.

Table 5.4 Best case results for SIANNs trained using original QNN-C algorithm

Benchmark Test	Percentage error		
	Best	Mean	95% CI
3-bit parity	25.0	48.5	± 1.93
Breast Cancer	0.56	20.4	± 3.55
Diabetes	30.7	39.5	± 2.11
Multi-class	26.0	67.6	± 5.32
	AVR		
Sunspots	0.233	1.480	± 0.294

5.5 Benchmark Test Results and Analysis

The QNN algorithm and selected variants of it were tested on the same set of benchmarks problems, with the same training and test conditions as the other algorithms tested in Chapter 4. The results obtained are presented in this section. The previously presented results obtained for the Gradient Descent with momentum and adaptive learning rate (GDX) and Levenberg-Marquardt (LM) algorithms are also presented for comparison, representing the best first- and second-order algorithms previously tested.

5.5.1 Results for the Wisconsin Breast Cancer dataset

The results obtained by testing the 9-9-1 SIANNs trained using the various QNN algorithms are presented in Table 5.5 and Fig. 5.6. Note that the median error in all cases was zero and so is not visible in the figure. The results indicate that there is no significant difference in the accuracy of the classifiers trained using the different algorithms. The mean error rates range between 0.24% to 0.36% and the percentage of networks achieving 100% accuracy between 56% and 68%, for the QNN algorithm and its variants. This is fairly close to the results of the GDX and LM algorithms: 66% networks with no error achieved by GDX, 74% with LM, and 0.20% mean error achieved by both. In fact, comparing the QNN results with the other algorithms tested in Chapter 4 , it can be seen that QNN outperforms the rest of them in terms of accuracy.

The training times, on the other hand, show more variation. Most of the QNN variants tested took a similar amount of time to train the set of networks. The exception was QNN5, which took a much longer time to train in the preliminary tests. QNN5 was retained for this section for two reasons: firstly, it was capable of producing good results; and second, to highlight the effect on training time the 'wrong' selection of the d update method could have. It fulfilled both requirements, achieving a mean error of 0.24%, the lowest mean error among all the QNN variants, and having the longest mean training time. The QNN5 training time was more than double all the other algorithms except LM. The other QNN variants had training times one and a half to two times longer than GDX, but about half the time of LM.

Overall, the standard QNN algorithm appears to be the 'best' of the QNN variants for this test, with the second best accuracy and fastest training time. There does not appear to be any significant differences in performance between the variants, except for the training time of QNN5.

5.5.2 Results for Pima Indians Diabetes dataset

The results for this test, presented in Table 5.6 and Fig. 5.7, show a similar trend to the results of the Breast Cancer. All the QNN variants achieved good accuracy with mean errors at 20% or below, which is a good result for this problem. The error rate achieved is between 19.6 and 20.0 %, similar to or better than the 19.88% reached using LM and better than GDX, which averaged 21.03%. Two of the variants, QNN9 and the constrained QNN (QNN-C2), were able to average close to 19.6% with the upper 95% Confidence limit below 20%. All of the QNN algorithms were able to get more than 50% of networks with error rates less than 20%, the best being QNN-C2 with 70%. The QNN6 variant was able to produce the best performing network with an error rate of only 15.63%, which is significantly lower than the best case performance of 17.71% achieved by most of the other variants.

Table 5.5 Best results for Wisconsin Breast Cancer dataset using 9-9-1 SIANNs trained with QNN algorithm variants.

Training Algorithm	Activation functions		Performance (% of runs)			Avg Epochs			Test Error				Mean Train time (s)
	Sh	Out	→ goal	0% err	20% <	All runs	→ goal	0% Errors	Best (%)	Mean (%)	95% CI	Median (%)	
QNN	Exp	Lgs	0	68	100	70	*	64	0.00	0.26	± 0.13	0.00	15.7
QNN3	Tnh	Lgs	0	56	100	78	*	70	0.00	0.33	± 0.12	0.00	17.4
QNN5	Lgs	Lgs	0	60	100	201	*	201	0.00	0.24	± 0.08	0.00	49.0
QNN6	Lgs	Lgs	0	60	100	109	*	86	0.00	0.36	± 0.14	0.00	20.0
QNN9	Lgs	Lgs	0	64	100	87	*	75	0.00	0.32	± 0.15	0.00	19.4
QNN-C2	Tnh	Lgs	0	62	100	77	*	74	0.00	0.29	± 0.12	0.00	17.1
GDX	Lgs	Lgs	0	66	100	161	*	160	0.00	0.20	± 0.08	0.00	10.3
LM	Lgs	Tnh	0	74	100	181	*	119	0.00	0.20	± 0.10	0.00	34.8

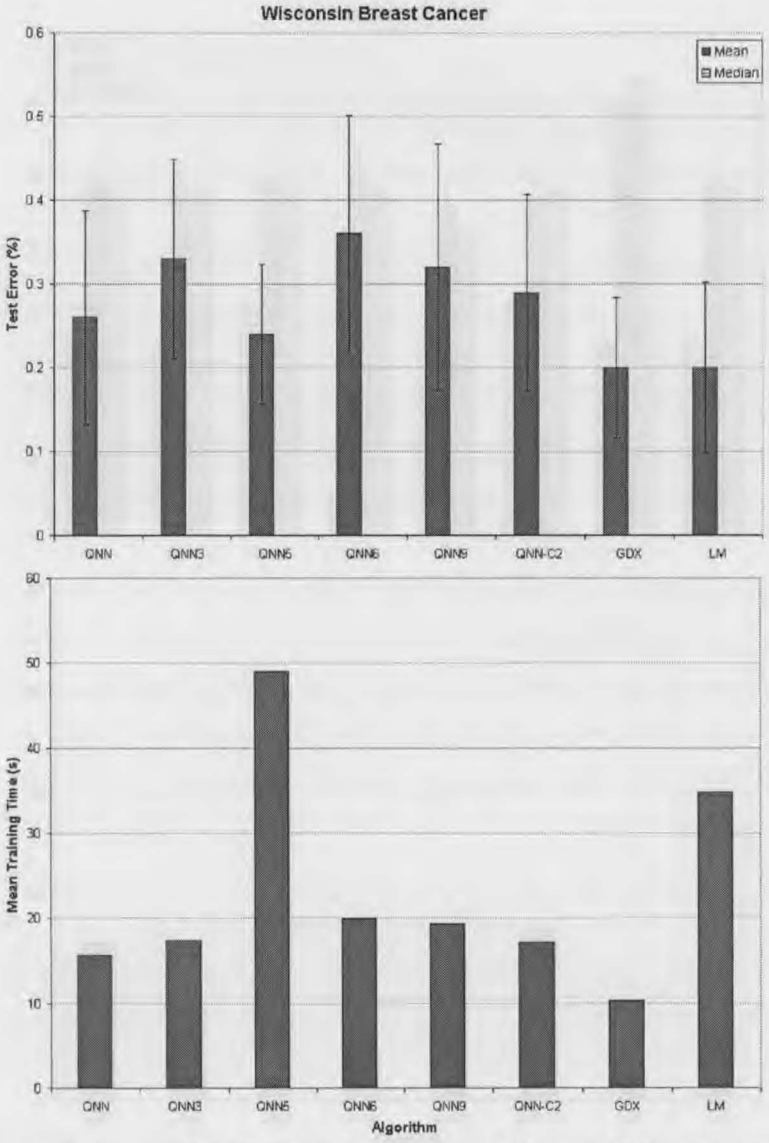


Fig. 5.6: Mean test error and training time for Breast Cancer dataset using SIANNs trained with QNN algorithm variants.

Table 5.6 Best results for Pima Indians Diabetes dataset using 8-8-1 SIANNs trained with QNN algorithm variants

Training Algorithm	Activation functions		Performance (% of runs)			Avg Epochs			Test Error				Mean Train time (s)
	Shunt	Out	→ goal	0% err	20% <	All runs	→ goal	0% Errors	Best (%)	Mean (%)	95% CI	Median (%)	
QNN	Lgs	Tnh	0	0	52	311	*	*	17.71	19.88	± 0.24	19.79	76.7
QNN3	Exp	Tnh	0	0	60	182	*	*	17.71	19.80	± 0.27	19.79	44.6
QNN5	Lgs	Lin	0	0	68	765	*	*	18.23	19.96	± 0.42	19.79	198.6
QNN6	Lgs	Tnh	0	0	56	117	*	*	15.63	20.02	± 0.37	19.79	23.3
QNN9	Lgs	Tnh	0	0	66	281	*	*	17.71	19.57	± 0.28	19.27	69.4
QNN-C2	Lgs	Tnh	0	0	70	363	*	*	18.23	19.69	± 0.22	19.79	83.2
GDX	Tnh	Lgs	0	0	14	195	*	*	17.71	21.03	± 0.29	21.35	13.2
LM	Lgs	Tnh	0	0	58	182	*	*	17.71	19.88	± 0.32	19.79	38.9

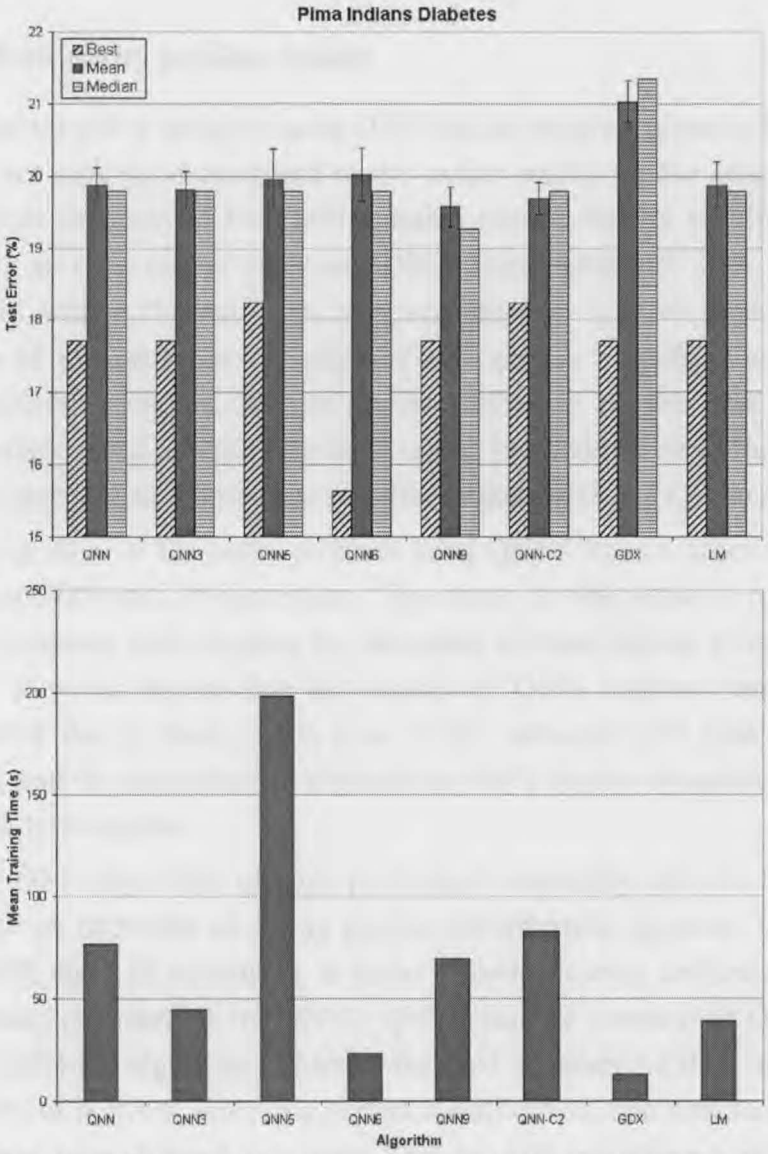


Fig. 5.7: Best case and mean test error and mean training time for Diabetes dataset using SIANNs trained with QNN algorithm variants

The training time required once again shows significant variation between the variants of the QNN algorithm. As in the previous test, the QNN5 variant required more than double the time required by any of the others, but for this problem the mean error rate was not the best, 'only' just under 20%. The QNN9 and QNN-C2 variants required around the same time as the standard QNN algorithm but achieved the best mean error rates of 19.6% and 19.7%, respectively. However, these algorithms took one- and-a-half times to twice as long to train compared to the LM algorithm. The QNN6 variant took about two-thirds the time of LM and about one-third the time of the standard QNN. It was still slower than the first order GDX algorithm, but achieved much better accuracy. Overall, the QNN6 would be the best QNN variant for this problem, with good mean accuracy, short training time and the best individual network performance by far.

5.5.3 The 3-bit parity problem results

The results for the parity problem using QNN and its variants, given in Table 5.7 and Fig. 5.8, do not look good compared to the earlier results for the other algorithms. The mean error for most of the QNN variants was in the 3% to 6% range, with QNN6 having an error rate of 16% and QNN5 even worse with 31%, compared to GDX and LM with 1.3% and 0.5% error respectively. A closer look reveals that around 80% of the networks actually achieved perfect classification (0% error), QNN5 and QNN6 excepted. The mean was driven up by the runs that did not converge, as they ended up with very large errors. It should be noted that the median classification error for all the variants was 0%, again QNN5 and QNN6 excepted.

The training time for the parity problem using QNN5 was an order of magnitude larger than the GDX and LM algorithms. The reason for this is that in most cases the runs did not terminate until reaching the maximum allowed number of epochs (in this case 1,000). It would appear that this variant of QNN requires more than 1000 epochs to solve this problem, since none of the networks was able to reach the training goal, and the networks that did achieve 100% correct classification required the maximum 1000 epochs.

The other QNN algorithm variants performed reasonably well on this test with about 80% of the networks achieving perfect classification accuracy, but could not match the GDX and LM algorithms, in terms of both accuracy and training time. Of these algorithms (standard QNN, QNN3, QNN9 and the constrained QNN-C2), the constrained QNN-C2 algorithm achieved the best accuracy, with a mean error of 3.5% and 92% of network achieving perfect classification, and also had the shortest average training time of about 7 seconds. This was still more than double both error rate and training time of the GDX algorithm.

Table 5.7 Best results for 3-bit Parity problem using 3-3-1 SIANNs trained with QNN algorithm variants

Training Algorithm	Activation functions		Performance (% of runs)			Avg. Epochs			Test Set Classification Error				Mean Train time (s)
	Shunt	Out	→ goal	0% err	< 20% err	All runs	→ goal	0% Errors	Best (%)	Mean (%)	95% CI	Median (%)	
QNN	Lgs	Lin	72	80	84	413	184	266	0.00	5.00	± 2.97	0.00	8.0
QNN3	Lgs	Lgs	72	80	88	436	216	295	0.00	4.75	± 3.04	0.00	9.4
QNN5	Tnh	Lin	0	14	30	981	*	1000	0.00	31.00	± 5.30	31.25	34.1
QNN6	Exp	Lin	14	24	62	270	156	263	0.00	15.75	± 3.42	12.50	5.4
QNN9	Lgs	Lgs	74	78	84	370	189	231	0.00	5.75	± 3.37	0.00	9.1
QNN-C2	Lgs	Lgs	78	82	92	376	200	239	0.00	3.50	± 2.33	0.00	7.0
GDX	Tnh	Lgs	94	94	96	352	311	311	0.00	1.25	± 1.44	0.00	2.3
LM	Exp	Lin	96	96	100	84	46	46	0.00	0.50	± 0.69	0.00	1.7

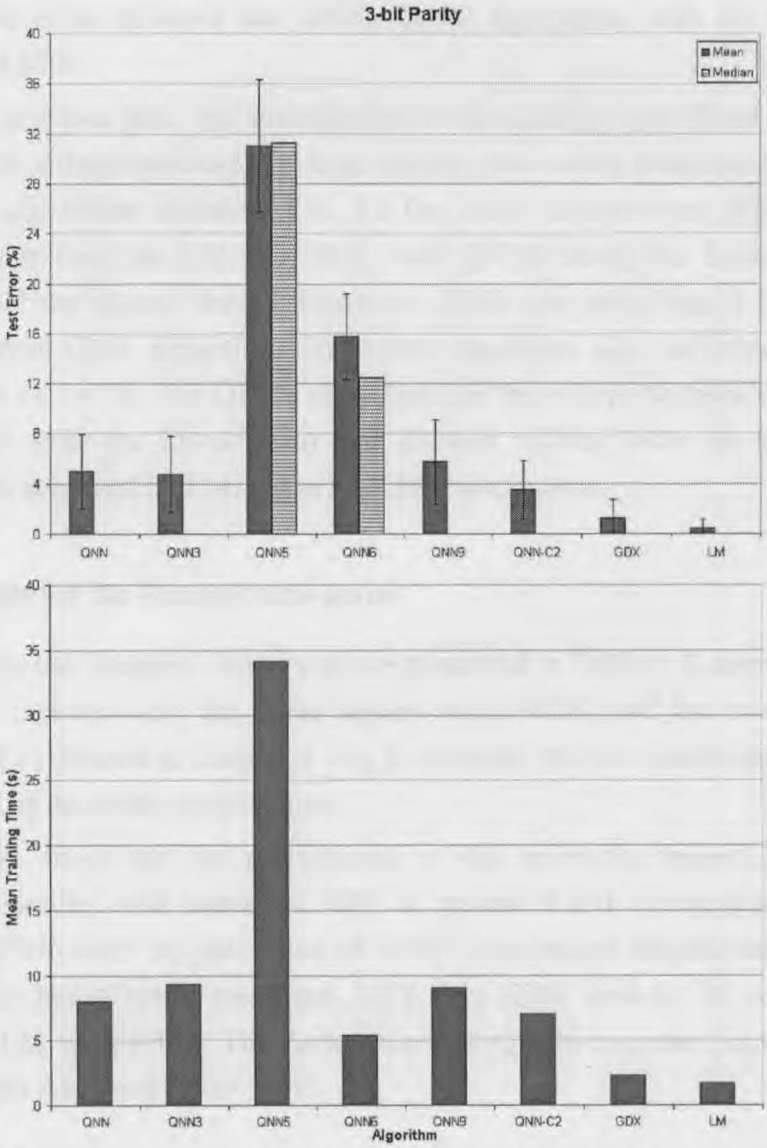


Fig. 5.8: Mean test error and training time for 3-bit parity dataset using SIANNs trained with QNN algorithm variants.

It was observed during training that, for some of the networks, the QNN6 algorithm was terminating prematurely because the value of μ was reaching the maximum allowed during training, hence the low training time and large average error rate.

5.5.4 Results for artificial multi-class problem

The results for the Multi-class benchmark problem using SIANNs trained with QNN-based algorithms are presented in Table 5.8 and Fig. 5.9. The results show that most of the QNN variants achieved mean classification error rates in the 5.7% to 6.1% range, which is similar to that achieved by GDX and LM. The exception is QNN5 algorithm with a mean error of 7.05%. The standard QNN algorithm achieved 6.05% error, with the other variants getting better results, QNN6 being the best with 5.72%. The best case error achieved was 4.00% for all algorithms, with the exception of QNN5 with 4.67%.

As in the previous tests, the variation lies in the training time. Once again QNN5 sticks out with a disproportionately large training time, more than double that of any of the other algorithms including LM. All the other variants were able to train the networks faster than the LM algorithm, with QNN6 being the fastest of all and standard QNN the second fastest. However, GDX was more than 2.5 times faster than the fastest QNN algorithm. The GDX algorithm also achieved the lowest average error of 5.47%. The QNN6 algorithm had the best performance of the QNN variants, with both the lowest error and shortest training time. Its accuracy was similar to that achieved by LM, but in half the training time.

5.5.5 Results for the Sunspot time series

The results for the Sunspots time series are presented in Table 5.9, showing both the performance metrics used: the mean square error (MSE) and the average relative variance (ARV) defined in Chapter 4. Fig. 5.10 shows the best, mean and median test ARV as well as the mean training time.

The results show that the performance of the networks trained by the QNN variants are similar, with mean test ARV of around 0.130, except for QNN5 with 0.202 and QNN6 with the best mean of 0.100. This means that the majority of the QNN variants had a better mean test ARV than GDX with 0.174, and not much higher than LM with 0.112. The performance of QNN6 is better than that of LM, with lower test ARV and lower MSE.

Table 5.8 Best results for Multi-Class dataset using 2-2-3 SIANNs trained with QNN algorithm variants

Training Algorithm	Activation functions		Performance (% of runs)			Avg Epochs			Test Error				Mean Train time (s)
	Sh	Out	→ goal	0% err	20% <	All runs	→ goal	0% Errors	Best (%)	Mean (%)	95% CI	Median (%)	
QNN	Exp	Lgs	0	0	100	239	*	*	4.00	6.05	± 0.27	6.00	61.2
QNN3	Exp	Lgs	0	0	100	166	*	*	4.00	5.79	± 0.23	6.00	74.7
QNN5	Lgs	Lgs	0	0	98	472	*	*	4.67	7.05	± 1.22	6.33	220.8
QNN6	Exp	Lgs	0	0	100	158	*	*	4.00	5.72	± 0.21	5.33	55.4
QNN9	Exp	Lgs	0	0	100	207	*	*	4.00	5.83	± 0.26	5.33	95.6
QNN-C2	Exp	Lgs	0	0	100	269	*	*	4.00	5.83	± 0.19	6.00	72.5
GDX	Exp	Lgs	0	0	100	377	*	*	4.00	5.47	± 0.16	5.33	19.4
LM	Exp	Lgs	0	0	100	228	*	*	4.00	5.69	± 0.26	5.33	103.0

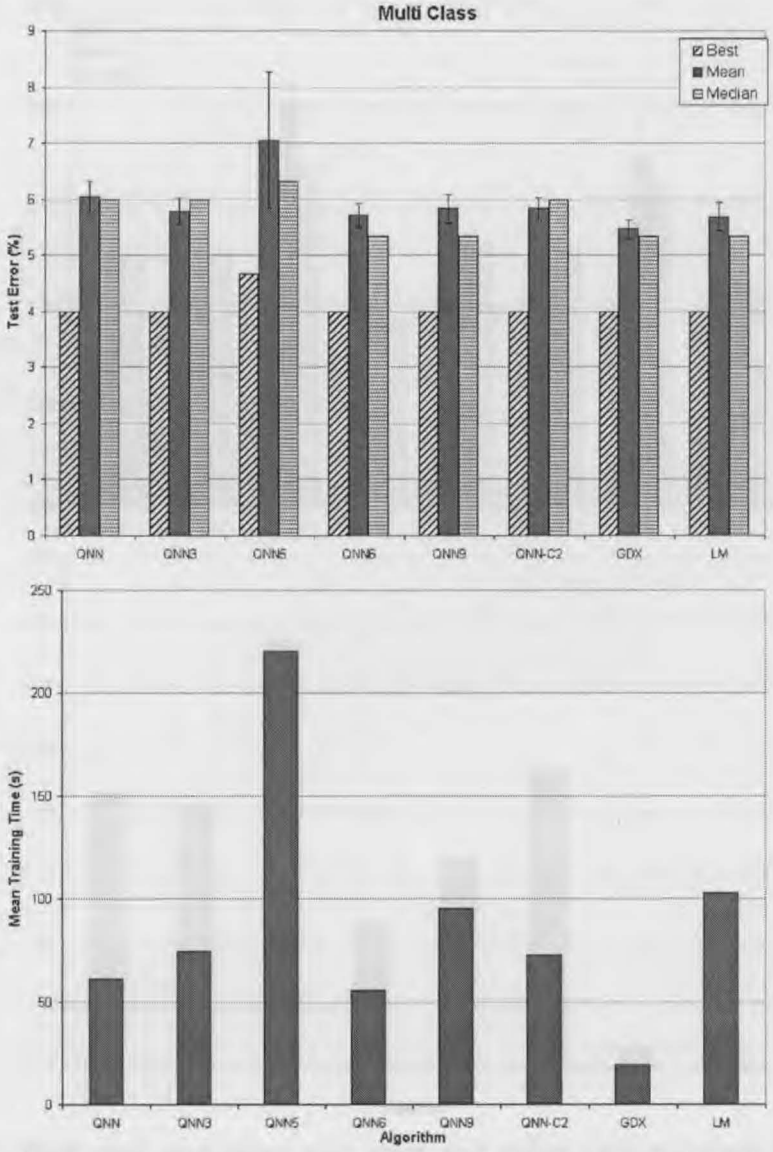


Fig. 5.9: Best case and mean test error and mean training time for Multi-class dataset using SIANNs trained with QNN algorithm variants.

Table 5.9 Best results for Sunspots dataset using 10-10-1 SIANNs trained with QNN algorithm variants

Training Algorithm	Act-fns		Performance (% of runs)			Avg Epochs	Test MSE		Test ARV				Mean Train time (s)
	Sh	Out	→ goal	all in tol	80% tol		Best	Median	Best	Median	Mean	95% CI	
QNN	Lgs	Lgs	0	70	100	495	0.0077	0.0106	0.092	0.127	0.127	± 0.003	86.5
QNN3	Lgs	Lgs	0	60	100	472	0.0070	0.0107	0.084	0.128	0.126	± 0.004	82.6
QNN5	Tnh	Lin	0	30	96	694	0.0101	0.0142	0.121	0.170	0.202	± 0.027	134.1
QNN6	Lgs	Lgs	0	80	100	337	0.0054	0.0077	0.065	0.093	0.100	± 0.009	45.8
QNN9	Lgs	Lgs	0	76	100	373	0.0093	0.0111	0.111	0.133	0.133	± 0.003	65.5
QNN-C2	Lgs	Lgs	0	72	100	535	0.0073	0.0107	0.088	0.128	0.127	± 0.003	93.7
GDX	Tnh	Lin	0	14	98	147	0.0085	0.0134	0.102	0.161	0.174	± 0.020	6.8
LM	Lgs	Lin	0	8	100	54	0.0075	0.0093	0.090	0.111	0.112	± 0.003	9.8

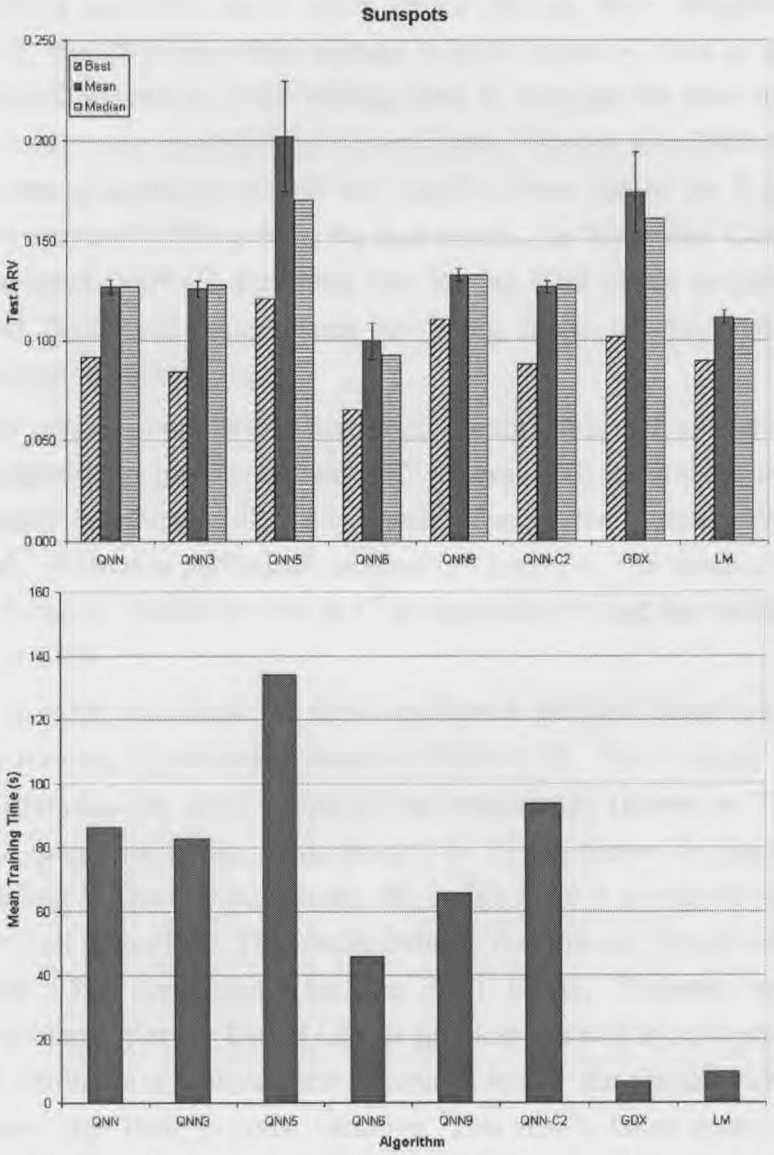


Fig. 5.10: Best case and mean test error and mean training time for Sunspots dataset using SIANNs trained with QNN algorithm variants.

In terms of training time, the QNN5 algorithm takes the longest by far, as in the earlier tests. This time, however, the other QNN variants take a lot longer to train than LM, mostly 8 to 10 times longer. The best is the QNN6 algorithm that still takes 5 times longer than LM (45.8 vs. 9.8 seconds). GDX was even faster than LM, but this was offset by the significantly lower accuracy. The best performance was by the QNN6 algorithm, with the best accuracy overall, better than even LM, and the fastest among all the QNN variants.

5.5.6 Analysis and Discussion

A visual analysis of the 'best case' results, as shown in the previous sections, does not show any definite trends in accuracy across all the algorithms. The clearest trend is that the QNN5 algorithm has a much longer training time compared to all the others. Overall, the QNN algorithm appears to have accuracy close to that achieved by LM and GDX algorithms, with training time in between the two or even worse than LM, but there are exceptions in every case. Among the QNN variants, the QNN6 algorithm achieved the overall best result in three out of the five benchmark tests, with the standard QNN getting the best results for the Breast Cancer problem and the constrained QNN-C2 algorithm best for the 3-bit Parity problem. It should also be noted that these comparisons are being made on the best performing activation function combinations.

In order to perform comparison across all combinations of activation functions and training algorithms, mean error and ARV values of all cases were compiled, and ranked for each benchmark. Statistical testing using the Kruskal-Wallis H test (Walpole et al., 1998) was performed, as done in Chapter 4. The mean error values of all tests are shown in Tables B.13 to B.17 in Appendix B, and the rankings given in Tables B.20 to B.24.

The final h value calculated for each benchmark problem based on comparison across all the training algorithms is shown in Table 5.10. The 'overall' h value was obtained by summing the ranks across all the benchmarks (shown in Table B.25 in Appendix B), then ranking the sums from 1 to 72 (as shown in Table B.26) and finally calculating h. The null hypothesis, H_0, is that there is no significant difference in the means of all algorithms. The results indicate that the null hypothesis is strongly rejected (with 95% confidence) for the 3-bit Parity, Diabetes and Sunspots benchmark problems. For the Breast Cancer problem the null hypothesis is accepted, while for the Multi-class problem, the statistic is below the critical value, but large enough to show that there is some variation. This result, taken across all possible activation function combinations, is similar to the 'best case' results as shown in the previous section. The overall comparison also indicates that there are significant differences in the means.

In order to get an idea of the differences in performance and to get an approximate ranking of the various algorithms, the final 'overall' rank of all activation function combinations for each algorithm was summed, and the algorithms ranked according to the column sum (refer Table B.26). This algorithm ranking along with the sum of ranks for each algorithm (as given in Table B.25) is given in Table 5.11.

Table 5.10 The h values calculated for all benchmark tests using QNN algorithm

Benchmark Test	Comparison across training algorithms	
	h Calculated	Critical value
3-bit Parity	**39.552**	14.067
Breast Cancer	3.057	
Diabetes	**22.511**	
Multi-Class	11.889	
Sunspots	**44.029**	
OVERALL	**36.919**	

Table 5.11 Overall ranking of QNN training algorithm variants

Rank	Training Algorithm	Sum of Ranks	Avg. Training time (s)	Time Ranking
1	QNN3	1362.5	45.7	4
2	QNN9	1397	51.8	6
3	QNN	1405	49.6	5
4	LM	1406	44.1	3
5	QNN-C2	1397.5	54.7	7
6	QNN6	1683	30.0	2
7	GDX	2188.5	10.3	1
8	QNN5	2300.5	127.3	8

The results in Table 5.11 show that the QNN3 algorithm came out on top. The second to fifth ranked QNN9, QNN, LM and QNN-C2 algorithms have scores that are very close to each other, and also close to QNN3. In fact the QNN-C2 algorithm was ranked fifth but has a sum of ranks lower than the QNN and LM algorithms, and only half a 'point' behind QNN9. The difference in the ranking shown, which is obtained from Table B.26 which ranks the sums obtained in Table B.25, and the Sum of ranks obtained directly from Table B.25, is due to the fact that the difference in performance is not significant. These four algorithms can in fact be considered to have equal ranking. The remaining algorithms had much higher (worse), and significantly different, scores. The GDX algorithm is second from bottom, with only the QNN5 algorithm coming out worse.

These overall rankings are quite different from the 'best case' results, where neither QNN3 nor QNN9 performed spectacularly well. Also, the QNN6 algorithm,

which appeared to be the 'best' in three out of the five tests, was only ranked sixth. However, these rankings give a better overall picture as they take into account performance across all possible activation function combinations.

The last two columns of Table 5.11 consist of the average training time of the best case networks and a ranking of the algorithms based on this average. The average time here is just to give an indication of the relative speed of the algorithms. It does not take into account differences in complexity of the problems and therefore the weighting given to the problem in working out the average. The time based ranking shows the GDX to be the fastest, as expected from a first-order algorithm. The QNN6 algorithm is ranked second, and is the only QNN variant faster than LM. All other times given are fairly close to that of LM, except for QNN5, which, not surprisingly, has the worst time performance. The GDX and QNN6 algorithms appear to compensate for their poor accuracy ranking by being fast to train. The QNN5 algorithm, however, has no such saving grace, coming out worst on both counts.

The results obtained lead to a number of conclusions. Firstly, the standard QNN has been shown to have performance comparable to the LM algorithm. One of the motivations in formulating the QNN algorithm was the hypothesis that using the recurrent network to replace the Hessian matrix inversion would result in shorter training times. However, a time saving was only seen in some cases, and not others.

The second conclusion is that the QNN3 has the best method to adaptively determine the step size, d. The method used in QNN9, however, comes in a close second. Looking back at the actual methods used to modify the step size for these two variants, it can be seen that the methods are almost identical, except for the fact that in QNN3 the value of d is updated every time μ is updated, whereas in QNN9 the step-size d is updated every epoch.

The third conclusion is that the constraint used in the QNN-C2 algorithm works and can improve performance in some cases. The QNN-C2 algorithm has performance comparable to that of the standard QNN. Even though it has an overall rank lower than QNN, the sum of ranks indicates that there is no significant difference in performance. The QNN-C2 is also able to work well when some of the other QNN variants had difficulty, such as with the Parity benchmark test.

Overall, the results show that the QNN algorithm and its variants tested are capable of achieving training performance similar to the second-order LM algorithm, except for QNN5. They are also able to train networks to achieve better accuracy than the first-order GDX algorithm, but have longer training times. This means that the QNN algorithm is a viable training algorithm for SIANNs.

5.6 Conclusion

In this chapter we have shown how the idea of using a recurrent neural network for bound constrained quadratic optimisation can be developed into a training algorithm for feedforward neural networks. The Quadratic Neural Network (QNN) algorithm is a second order algorithm that avoids the need to invert the Hessian matrix by using a recursive equation that simulates a recurrent neural network.

The QNN algorithm has been successfully applied to train SIANNs on a number of standard benchmark problems, and the results show that this algorithm is able to train the networks to achieve results comparable to or better than the LM and GDX algorithms.

The QNN algorithm has been shown to be a viable training algorithm that is capable of producing good results. It has the added advantage of being able to readily incorporate constraints that may need to be imposed during training. A number of variants have also been formulated and tested. These variants were formulated to reduce the number of free parameters that need to be set, and to incorporate different constraints on the weight updates. Two variants that adaptively modify the step-size, QNN3 and QNN9, are able to achieve better performance than the standard QNN. The QNN-C2 constrained version has also been shown to improve performance in some cases, and has overall performance comparable to standard QNN.

Chapter 6

Further Development of Shunting Inhibitory Artificial Neural Networks

6.1 Motivation

Originally, the SIANN was proposed as a fully connected structure (Bouzerdoum, 1999); that is, each input is fed directly into one shunting neuron as its excitatory input, whereas all inputs are weighted and fed in as inhibitory inputs (refer Section 3.6). Therefore, the fully connected 'standard' SIANN structure has as many shunting neurons as there are inputs. Using this basic SIANN structure, the size of the network is actually determined by the dataset. The number of neurons in the shunting layer(s) is determined by the number of data attributes, whereas the number of neurons in the output layer is determined by the number of class labels. While this architecture removes the need for finding an optimal network structure, it was found to be too restrictive in some problems. In particular, when the data has a large number of inputs and outputs, the resulting network structure is inordinately large. This leads to increased computational complexity and training time.

In this chapter, enhancements to the SIANN structure are proposed that would remove the restrictions on the size of the network, in particular the number of shunting neurons. The enhancements allow the size of the network to be reduced for problems that have a large number of inputs and outputs, resulting in reduced computational complexity and better generalisation. Conversely, if the number of inputs is small, then the networks structure can be expanded to have more shunting neurons than inputs; this provides for additional computational capacity, if required.

The following section outlines the development of enhancements to the standard SIANN structure. The third section presents experimental results, comparing the performance of the enhanced structures with the standard SIANN structure. Finally, conclusions are given in the fourth section.

6.2 The Enhanced SIANN structure

This section describes the development of the enhanced SIANN structures. The motivation and implementation of the reduced SIANN structure are presented next. This is followed by the expanded SIANN structure. The third subsection presents the *Enhanced* SIANN structure; it combines the previous two, seemingly contradictory, structures (expanded and reduced structures) into one generic structure.

6.2.1 Reduced SIANN structure

The first enhancement is to reduce the complexity of the SIANN structure when there is a large number of inputs. One reason for doing this is that smaller networks are less likely to over-fit the data and therefore are more likely to generalise well. The smaller number of weights would also help reduce the computational complexity and memory requirements during training, thereby reducing the time to train the networks.

The 'reduced' SIANN structure has less shunting inhibitory neurons than there are inputs, while the number of output neurons remains equal to the number of outputs required. All the inputs are fed into the network as inhibitory inputs, whereas only the first m inputs can be fed in as excitatory inputs, where m is the number of shunting neurons in the reduced structure. The restriction on the number of excitatory inputs is due to the fact that each shunting neuron can only have one unweighted excitatory input. The Reduced SIANN structure is shown in Fig. 6.1.

6.2.2 Expanding the SIANN structure

The second case considered is when the problem on which the network is being trained is too complex for the standard SIANN structure. In this case the network has insufficient neurons, and therefore insufficient weights, to be able to map the input-output relationship as required by the problem. This could happen when the number of inputs is a small, resulting in a small number of neurons in the shunting layer. The solution would be to provide the required extra 'processing power' in the form of additional neurons.

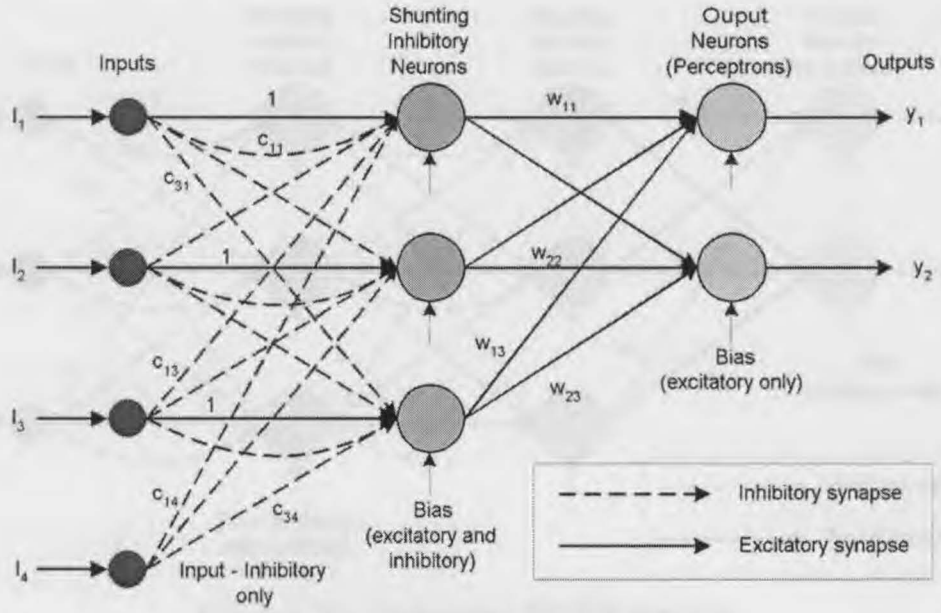


Fig. 6.1: The 'Reduced' SIANN structure.

Two ways of adding neurons to the SIANN structure have been considered. The first method is to add extra neurons to the shunting layer. These additional neurons only have a bias term as the solitary excitatory input, but all network inputs are fed in as inhibitory inputs (refer Fig. 6.2). These additional neurons have been dubbed 'interneurons' based on biological parallels. This method allows neurons to be added incrementally to provide additional computational capacity as required, without changing the fundamental methods of operation and training.

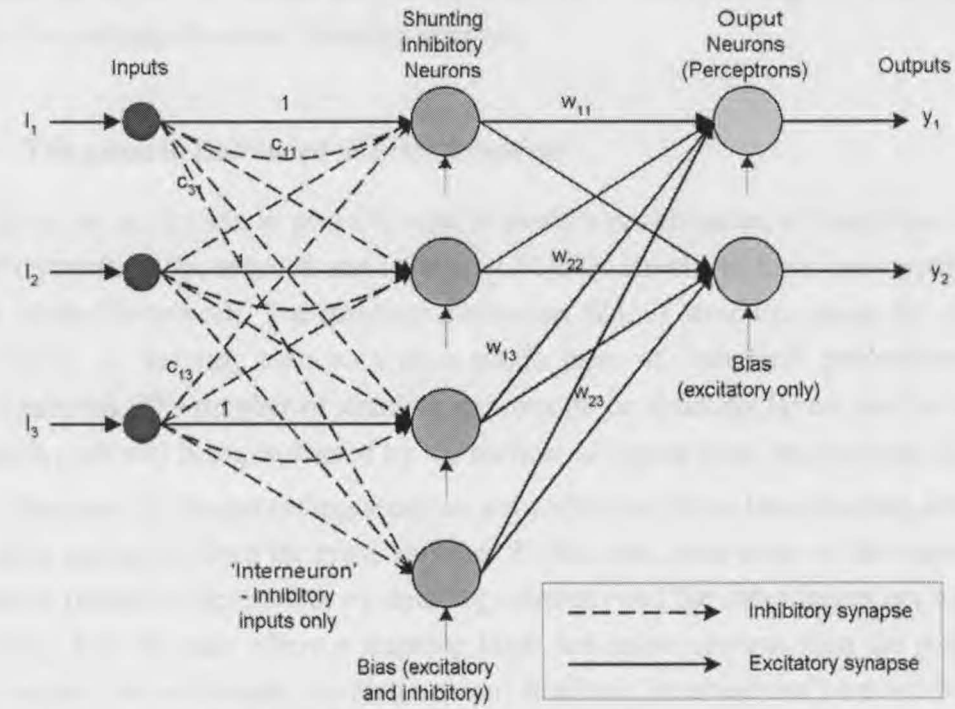


Fig. 6.2: The 'Expanded' SIANN structure.

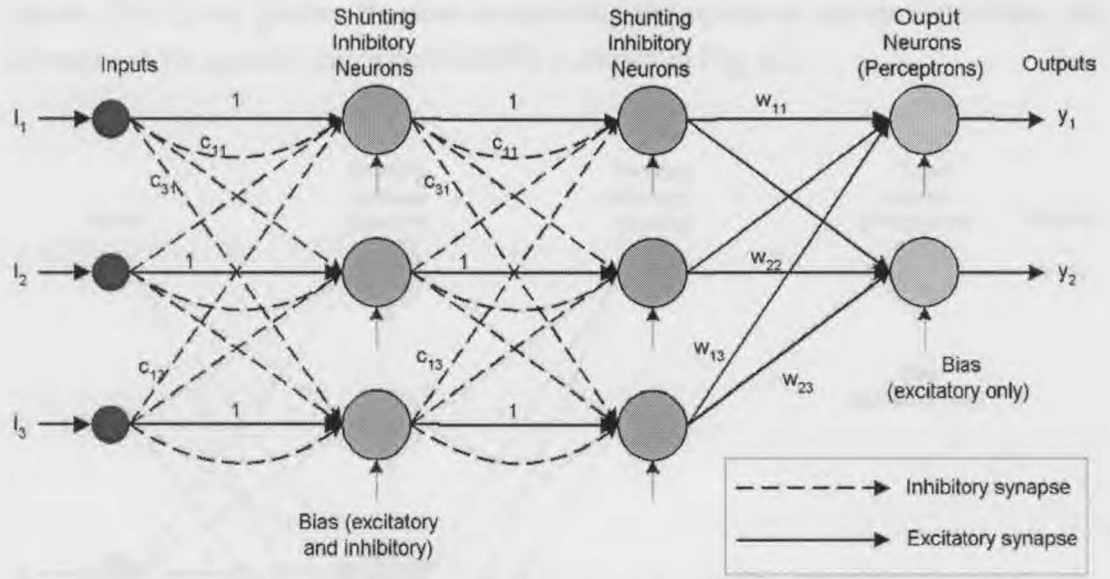


Fig. 6.3: The Multi-layer SIANN structure.

The other method of increasing the processing power of the network is to add additional layers of shunting neurons. The result is a Multi-layer SIANN structure, as shown in Fig. 6.3. This is analogous to MLPs, where the number of hidden layers can be increased for complex problems. In its simplest form, the number of neurons in each shunting layer would be the same. Adding layers not only adds additional weights through the extra neurons, but also allows for more complex input–output characteristics to be formed due to the multiple layers. This gain comes with an associated cost in the form of increased computational complexity during training. The training algorithms would need to be modified or enhanced slightly to be able to handle the multiple layers of shunting neurons.

6.2.3 The generic Enhanced SIANN structure

In order to be as flexible as possible, and to avoid a proliferation of variations to the SIANN structure, the reduced and expanded SIANN structures have been combined into a single framework. The resulting *Enhanced SIANN* structure caters for one or more layers of shunting neurons with a single layer of 'standard' perceptron-type output neurons. The number of shunting neurons in the shunting layers can be varied arbitrarily, without being restricted by the number of inputs from the previous layer.

As discussed in the preceding sections, networks may have less shunting neurons than there are inputs from the previous layer. In this case, only some of the inputs are excitatory (equal to the number of shunting neurons) and the other inputs act only as inhibitory. For the case where a shunting layer has more neurons than the previous layer outputs, the additional shunting neurons (dubbed 'interneurons') are fed in with a constant bias as the only excitatory input, but with the normal variable inhibitory

inputs. This gives greater freedom in selecting the optimum network structure. An example of the generic Enhanced SIANN is shown in Fig. 6.4.

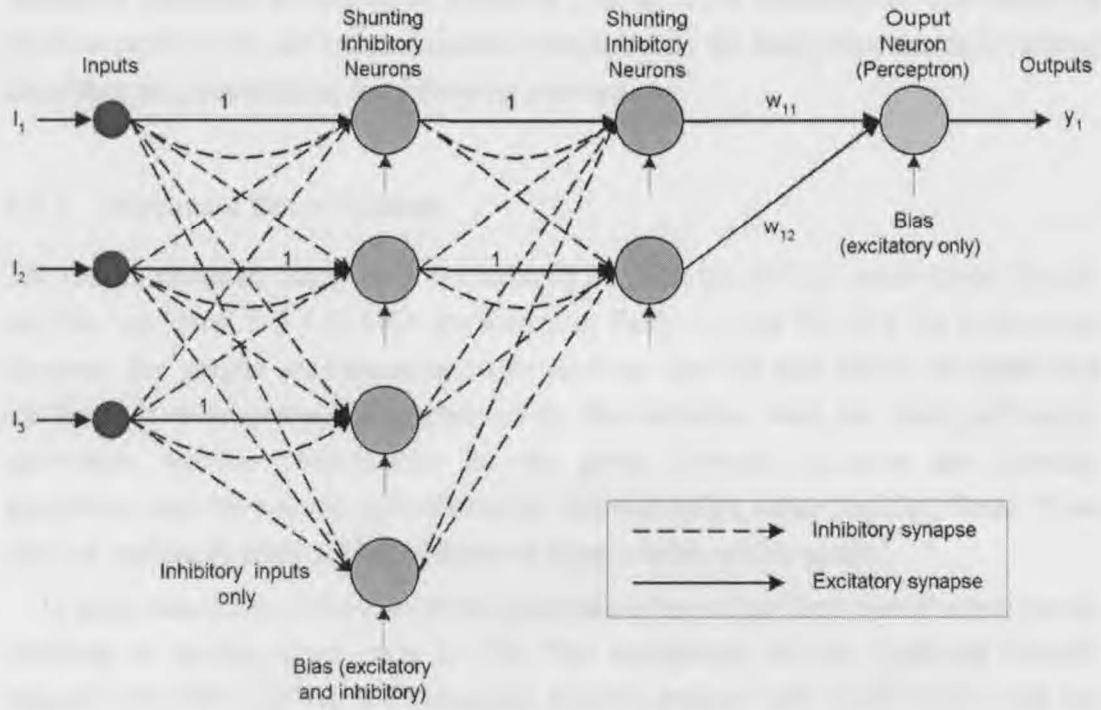


Fig. 6.4: The generic Enhanced SIANN structure.

6.3 Benchmark Test Results and Analysis

The Enhanced SIANN structures were tested on the same set of benchmark problems as the in the previous two chapters, and the results obtained are presented in this section. For each benchmark problem, a 'Reduced' SIANN structure and an 'Expanded' structure (either multi-layer or single SIANN layer with additional neurons) was tested and compared to the performance of the 'standard' SIANN. For problems with a small number of inputs, such as the 3-bit Parity and Multi-Class problems, the 'Expanded' structure chosen was a single-layer SIANN with additional neurons. For the other problems, with a relatively large number of inputs, the expanded structures used were Multi-layer SIANNs that had the same number of shunting neurons in the first layer as the 'Reduced' SIANN for that problem, and a smaller number of neurons in the second shunting layer. In all cases, there are fewer shunting neurons and synaptic weights in the multi-layer SIANN structure than in the 'standard' SIANN.

As in the previous chapters, 50 networks were generated for each structure. These networks were trained using the Gradient Descent with adaptive learning rate and momentum (GDX), Levenberg-Marquardt (LM) and Direct Solution combined with GDX (DS-GDX) algorithms. The initialisation and training parameters used are the

same as given in Chapter 4, for consistency. All possible combinations of activation functions were tested for each structure. For each benchmark problem, the mean error values of all combinations of network structures, training algorithms and activation functions are shown in Tables B.27 to B. 31 in Appendix B. The results of the best performing activation function combinations for each structure and training algorithm are presented in the following sections.

6.3.1 Wisconsin Breast Cancer

The results obtained using the 9-4-1 reduced SIANN, the 9-4-2-1 multi-layer SIANN and the 'standard' 9-9-1 SIANN are shown in Table 6.1 and Fig. 6.5. As in previous chapters, the graphs are broken into two sections: the top part shows the mean and median test error percentages achieved by the networks with the best performing activation function combination for the given network structure and training algorithm, and the second part shows the corresponding mean training times. Note that the median is often zero, and hence it is not visible on the graph.

In most cases more than half of the networks achieved perfect classification result, resulting in median error rates of 0%. The exceptions are the Reduced SIANN trained with GDX (16%), the Expanded SIANN trained with GDX (42%) and the Standard SIANN trained with DS-GDX (36%). The best appears to be the Standard SIANN trained with LM, with 74% of the networks achieving 0% classification error. The mean error rates range from 0.20% to 0.55%.

Comparing the performance across the different structures, the trends are different for the different algorithms. The GDX algorithm achieved best results with the standard SIANN structure, achieving the lowest mean error (0.20%) jointly with the LM algorithm; all other structures have mean errors in excess of 0.5%. The Reduced SIANN had the second best accuracy, with the Multi-layer SIANN having the worst classification accuracy. The training time required for GDX was short, generally the shortest of the three algorithms. The Reduced SIANN had the shortest training time for GDX, and the Multi-layer SIANN the longest.

The LM algorithm also achieved the best mean error rate of 0.20% with the standard SIANN, but the reduced SIANN had only a slightly higher error (0.23%). However, the time taken to train was about 20% less for the reduced SIANN. The multi-layer SIANN had the highest error rate (0.31%), among all the networks trained with LM, and a training time between those of the other two structures.

Using the DS-GDX algorithm, on the other hand, the best performance was obtained with the reduced SIANN (0.26%), the multi-layer SIANN next and the standard SIANN having the worst error (0.54%). In terms of training time, however, the standard SIANN was fastest, followed by the reduced SIANN, the Multi-layer SIANN being the slowest.

Table 6.1 Best results for Wisconsin Breast Cancer dataset using Enhanced SIANNs

SIANN Structure	Training Algor.	Activation functions		Performance (% of runs)			Avg. Epochs			Test Error				Avg. CPU time (s)
		Sh	Out	→ goal	0% err	20% <	All runs	→ goal	0% Error	Best (%)	Mean (%)	95% CI	Med. (%)	
Reduced 9-4-1 (53 weights)	GDX	Exp	Lin	0	16	100	145	*	141	0.00	0.55	± 0.09	0.56	9.0
	LM	Exp	Lin	0	68	100	147	*	136	0.00	0.23	± 0.10	0.00	28.6
	DSGDX	Exp	Lin	0	68	100	193	*	184	0.00	0.26	± 0.13	0.00	11.7
Expanded 9-4-2-1 (65 weights)	GDX	Exp	Lin	0	42	100	187	*	203	0.00	0.51	± 0.17	0.56	14.7
	LM	Exp	Lin	0	56	100	105	*	85	0.00	0.31	± 0.11	0.00	30.8
	DSGDX	Exp	Lin	0	54	100	196	*	185	0.00	0.47	± 0.23	0.00	17.7
Standard 9-9-1 (118 weights)	GDX	Lgs	Lgs	0	66	100	161	*	160	0.00	0.20	± 0.08	0.00	10.3
	LM	Lgs	Tnh	0	74	100	181	*	119	0.00	0.20	± 0.10	0.00	34.8
	DSGDX	Tnh	Lgs	0	36	100	96	*	100	0.00	0.54	± 0.14	0.56	6.4

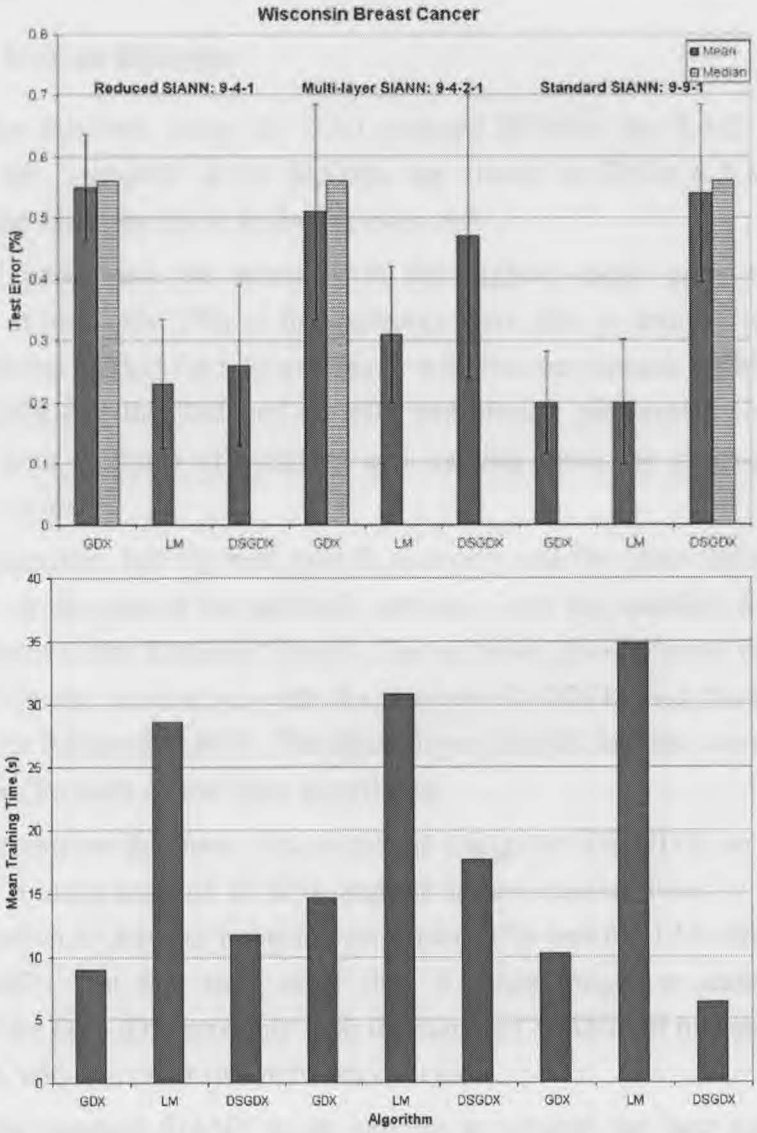


Fig. 6.5: Mean and median test error and mean training time for the Wisconsin Breast Cancer dataset using Enhanced SIANNs.

Overall, the standard SIANN performed best, the exception being with the DS-GDX algorithm. This is not surprising given that this is the largest structure tested, both in terms of number of neurons as well as number of synaptic weights. Additionally, the standard SIANN has all inputs serving as excitatory and inhibitory, while the other networks have only the first few inputs fed as excitatory, with the remaining inputs serving as inhibitory only. The drawback of longer training times due to more complex computation was only seen with the second-order LM algorithm. The reduced SIANN had the benefit of reducing the training time required by the LM algorithm, without significantly changing the accuracy; it also got the best results with the DS-GDX algorithm. The Multi-layer SIANN, though having more neurons than the reduced SIANN, had lower overall accuracy when compared to the Reduced SIANN. It also had the longest training times for the GDX and DS-GDX algorithms, thereby ending up as the 'worst' overall.

6.3.2 Pima Indians Diabetes

The results for this test, using the 8-3-1 reduced SIANN, the 8-3-2-1 multi-layer SIANN and the 'standard' 8-8-1 SIANN, are shown in Table 6.2 and Fig. 6.6, presented in the same format as in the previous test.

The GDX performed the worst, with the highest mean error rates overall. Additionally, at best only 20% of the networks were able to achieve less than 20% error, which is the marker for a 'good' result with this benchmark problem. Both the standard SIANN and the Reduced SIANN had similar performance when trained using GDX, both in terms of accuracy and training time; the multi-layer SIANN performed the worst.

The LM algorithm had the best overall accuracy and the same trend of reducing training time as the size of the network reduced, from the standard SIANN to the Multi-layer down to the Reduced SIANN. The accuracy also followed the same trend as the Breast Cancer benchmark, with the standard SIANN having the best accuracy followed by the Reduced SIANN. The Multi-layer SIANN had the worst mean error, as was the case for each of the three algorithms.

The best mean performance was obtained using the DS-GDX on the standard SIANN, with a mean error of 19.82%, and the fastest training time by far. The only other combination to achieve a mean error under 20% was the LM algorithm on the standard SIANN, but that took more than 6 times longer to train. The speed advantage of the DS-GDX was only with the standard SIANN; it had similar training times to GDX with the other two network structures.

Overall, the standard SIANN again appears to achieve the best performance in terms of accuracy, but the Reduced SIANN helps to shorten the training time for the LM and GDX algorithms. The probable reasons for why the standard SIANN

achieves the best accuracy are the same as for the Breast Cancer problem. Firstly, it is the largest structure in terms of neurons and synaptic weights. Second, all eight inputs are fed in as both excitatory and inhibitory, whereas for the other structures only three inputs serve as excitatory inputs, with the rest being inhibitory only.

6.3.3 The 3-bit parity problem

The results for the parity problem, presented in Table 6.3 and Fig. 6.7, are for a 3-2-1 Reduced SIANN, a 3-4-1 Expanded SIANN, and 'the standard' 3-3-1 SIANN. Note that in all cases the median is zero, and so is the mean in two cases, hence these graphs are not visible on the plot.

The results show significant variations in performance, as the number of neurons is decreased or increased. Reducing the size of the network, by just removing one shunting neuron, results in significantly higher error rates. The mean error rate jumps from between 0.50% and 1.25% to between 8.50% and 11.50%, and the percentage of networks achieving 0% error drops from around 95% to between 48% and 60%. Conversely, just adding one additional shunting neuron (without an excitatory input) yields 'perfect' results when trained with the LM and DS-GDX algorithms, i.e., 100% of networks achieving 100% correct classification.

In terms of training time, the Reduced SIANN took longer to train: between 2 and 8 times longer than the standard SIANN. The Expanded SIANN training times, on the other hand, were shorter. The Expanded SIANN is able to increase accuracy as well as reduce training time.

The sensitivity to the size of the network can probably be attributed to the fact that there is only a small number of inputs (three) and small number of training examples (eight). Reducing the size of the structure resulted in only 2 out of 3 inputs being excitatory as well as cutting the number of weights by a third, for what is essentially a fairly complex problem for neural networks. The addition of a spare shunting neuron (with only inhibitory inputs) would seem to provide the extra computational power to easily solve this problem. These results show the advantage of expanding the standard SIANN structure on some problems, particularly when the number of inputs is small, resulting in a small number of shunting neurons in the standard structure.

Table 6.2 Results for Pima Indians Diabetes dataset using Enhanced SIANNs

SIANN Structure	Training Algor.	Activation functions		Performance (% of runs)			Avg Epochs			Test Error				Avg. CPU time (s)
		Sh	Out	→ goal	0% err	20% <	All runs	→ goal	0% Error	Best (%)	Mean (%)	95% CI	Med. (%)	
Reduced 8-3-1 (37 weights)	GDX	Tnh	Tnh	0	0	20	192	*	*	18.23	20.96	± 0.31	21.35	11.9
	LM	Exp	Tnh	0	0	56	58	*	*	16.67	20.18	± 0.36	19.79	11.9
	DSGDX	Exp	Lin	0	0	42	189	*	*	18.23	20.62	± 0.38	20.57	11.8
Expanded 8-3-2-1 (48)	GDX	Tnh	Tnh	0	0	14	178	*	*	18.75	21.55	± 0.42	21.35	16.9
	LM	Exp	Tnh	0	0	46	80	*	*	17.71	20.43	± 0.45	20.31	24.8
	DSGDX	Tnh	Lin	0	0	30	178	*	*	17.19	20.59	± 0.38	20.83	17.0
Standard 8-8-1 (97)	GDX	Tnh	Lgs	0	0	14	195	*	*	17.71	21.03	± 0.29	21.35	13.2
	LM	Lgs	Tnh	0	0	58	182	*	*	17.71	19.88	± 0.32	19.79	38.9
	DSGDX	Lgs	Lgs	0	0	68	94	*	*	18.75	19.82	± 0.29	19.79	6.4

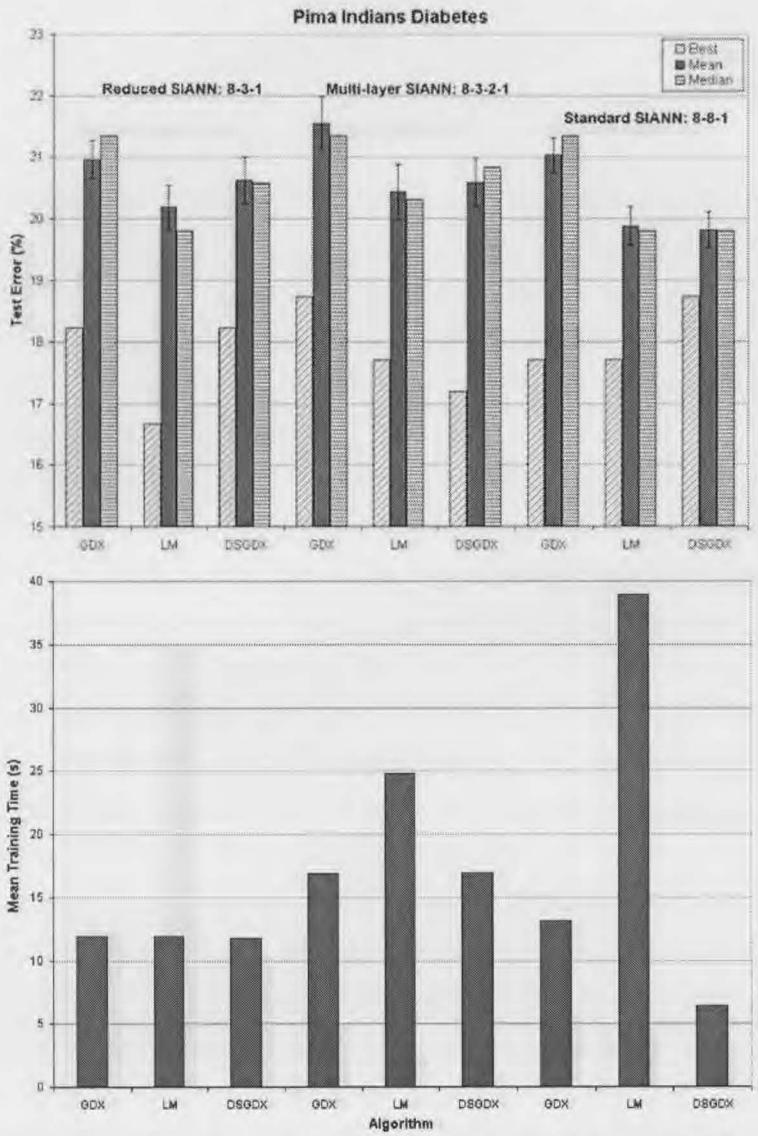


Fig. 6.6: Best, mean and median test error and mean training time for the Diabetes dataset using Enhanced SIANNs

Table 6.3 Best results for the 3-bit Parity problem using Enhanced SIANNs

SIANN Structure and no. of weights	Training Algor.	Activation functions		Performance (% of runs)			Avg. Epochs			Test Error				Avg. CPU time (s)
		Sh	Out	→ goal	0% err	20% <	All runs	→ goal	0% Error	Best (%)	Mean (%)	95% CI	Med. (%)	
Reduced 3-2-1 (15 weights)	GDX	Tnh	Tnh	46	48	88	756	512	579	0.00	8.50	± 3.16	0.00	4.9
	LM	Lgs	Tnh	50	60	76	700	399	500	0.00	8.75	± 3.38	0.00	13.8
	DSGDX	Tnh	Lin	52	52	68	596	223	223	0.00	11.50	± 3.82	0.00	4.1
Expanded 3-4-1 (29)	GDX	Tnh	Lgs	96	96	100	301	271	271	0.00	0.50	± 0.69	0.00	2.0
	LM	Exp	Tnh	100	100	100	48	48	48	0.00	0.00	± 0.00	0.00	1.1
	DSGDX	Tnh	Lin	100	100	100	48	48	48	0.00	0.00	± 0.00	0.00	0.5
Standard 3-3-1 (22)	GDX	Tnh	Lgs	94	94	96	352	311	311	0.00	1.25	± 1.44	0.00	2.3
	LM	Exp	Lin	96	96	100	84	46	46	0.00	0.50	± 0.69	0.00	1.7
	DSGDX	Tnh	Lin	94	94	100	169	116	116	0.00	0.75	± 0.83	0.00	1.3

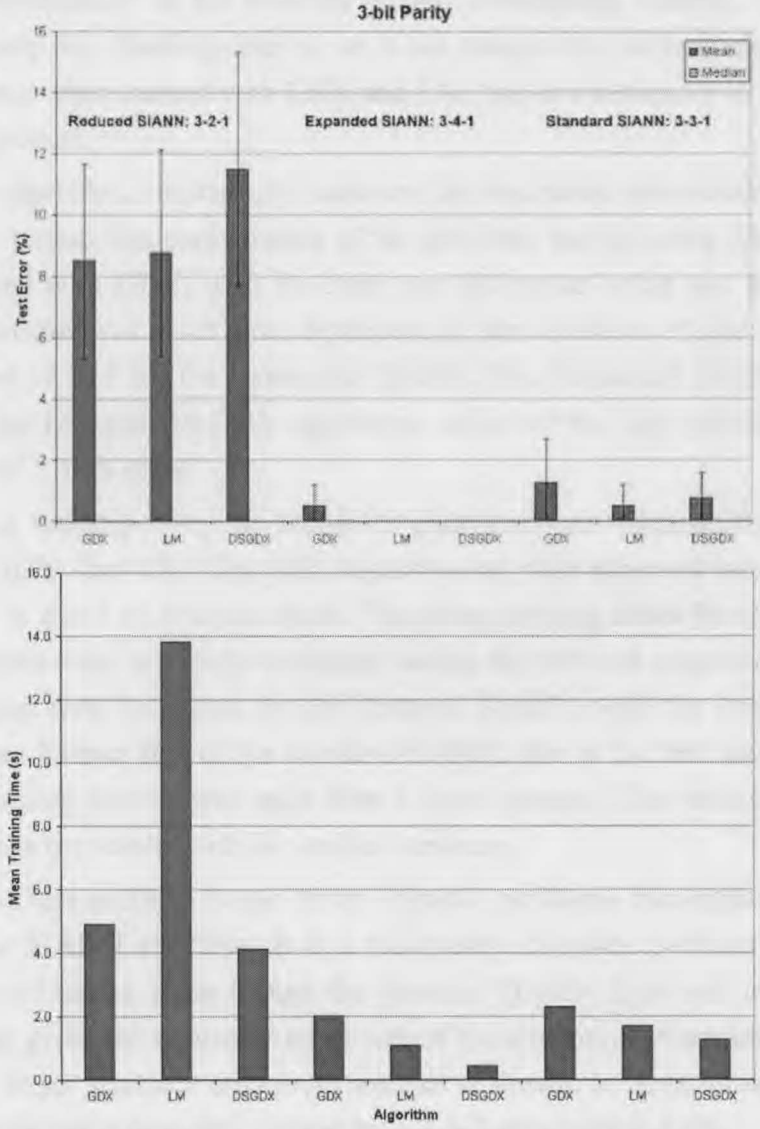


Fig. 6.7: Mean and median test error and mean training time for 3-bit parity dataset using Enhanced SIANNs

6.3.4 Artificial Multi-Class Problem

The networks used for the multi-class problem were the Reduced SIANN (2-1-3), the Expanded SIANN (2-3-3) and the standard SIANN (2-2-3). The results are shown in Table 6.4 and Fig. 6.8. Note that the error rates of the Reduced SIANN trained with DS-GDX are around 34%, which exceeds the range of the ordinate in Fig. 6.8; this was done deliberately so as to make the other variations clearer.

The results show that the mean error rate tends to decrease as the size of the network is increased, with the most pronounced change being for the DS-GDX algorithm (from 34.76% down to 5.57%). For all algorithms, the Reduced SIANN gave the lowest classification accuracy, whereas the Expanded SIANN gave the highest accuracy. This dependency on size is not surprising given the small network sizes (19 weights for the standard SIANN and only 11 for the Reduced SIANN) and the relative complexity of the problem having overlapping classes. The Reduced SIANN has only one shunting neuron, yet it has comparable performance to those of larger structures when trained with GDX and LM; this is a testimony to the power of the shunting neuron.

The GDX algorithm, surprisingly, achieved the best mean error results for each of the structures tested. The performance of the networks trained using LM were close to those trained with GDX, with the error rate difference being less than 1%. The DS-GDX algorithm was much more dependent on the variations of size, but achieved the error level of LM for the Expanded SIANN. The Expanded SIANN structure, trained with the LM and DS-GDX algorithms, achieved the best individual network performance of 3.33% error.

In terms of training time, the DS-GDX algorithm was consistently the fastest, followed by GDX, then LM. The GDX algorithm not only achieved lower error rates than LM, but it was 5 to 13 times faster. The mean training times for GDX and DS-GDX algorithms were relatively consistent across the network structures. However, the LM training time *increased* for the Reduced SIANN, with the average training time more than 3 times that of the standard SIANN; due to the fact that the average number of training epochs was more than 3 times greater. This indicates a greater effort to achieve the results with the smaller structure.

The Multi-Class problem is one of the 'classic' problems that highlight the need to expand the SIANN structure. It is a moderately complex problem with a very small number of inputs. Even though the standard SIANN does well to achieve the results it does, given the relatively small size of the network, the advantage of being able to use a larger structure with more weights is shown, in terms of accuracy. Fig. 6.9 shows the decision boundary formed by a 2-3-3 enhanced SIANN.

Table 6.4 Best results for Multi-Class dataset using Enhanced SIANNs

SIANN Structure	Training Algor.	Activation functions		Performance (% of runs)			Avg Epochs			Test Error				Avg. CPU time (s)
		Sh	Out	→ goal	0% err	20% <	All runs	→ goal	0% Error	Best (%)	Mean (%)	95% CI	Med. (%)	
Reduced 2-1-3 (11 weights)	GDX	Exp	Lgs	0	0	100	552	*	*	4.67	5.89	± 0.25	6.00	26.6
	LM	Exp	Lgs	0	0	96	776	*	*	4.00	6.69	± 1.70	5.33	341.7
	DSGDX	Exp	Lgs	0	0	0	173	*	*	34.67	34.76	± 0.06	34.67	8.9
Expanded 2-3-3 (27)	GDX	Exp	Tnh	0	0	100	552	*	*	4.00	5.00	± 0.18	5.33	29.2
	LM	Lgs	Tnh	0	0	100	201	*	*	3.33	5.60	± 0.23	6.00	149.0
	DSGDX	Exp	Lin	0	0	100	189	*	*	3.33	5.57	± 0.19	6.00	10.1
Standard 2-2-3 (19)	GDX	Exp	Lgs	0	0	100	377	*	*	4.00	5.47	± 0.16	5.33	19.4
	LM	Exp	Lgs	0	0	100	228	*	*	4.00	5.69	± 0.26	5.33	103.0
	DSGDX	Exp	Lin	0	0	90	206	*	*	5.33	9.39	± 2.24	6.67	10.9

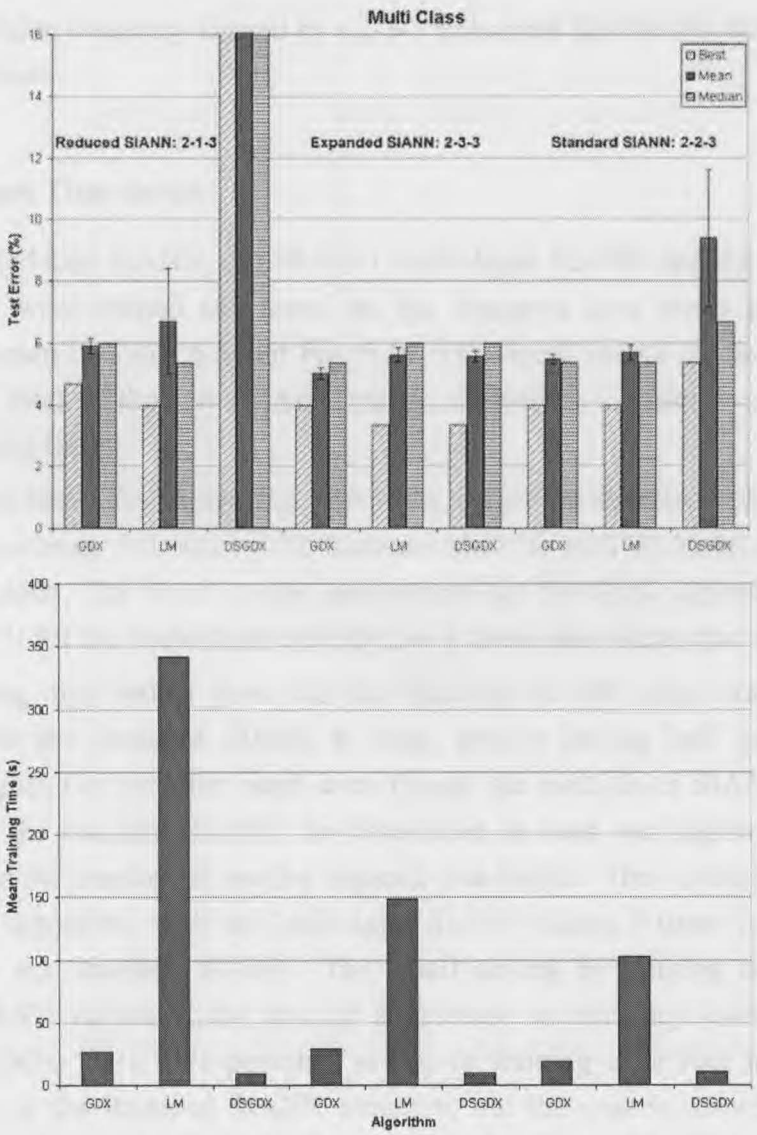


Fig. 6.8: Best, mean, median test error and mean training time for Multi-class dataset using Enhanced SIANNs.

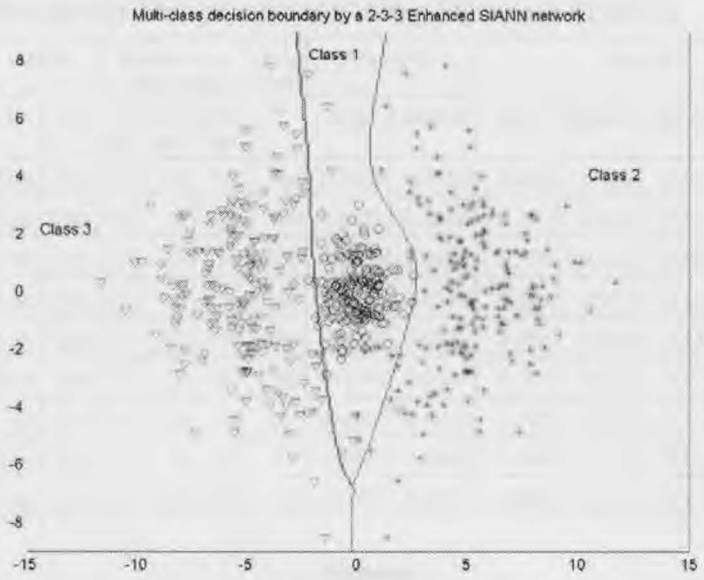


Fig. 6.9: Decision boundary formed by a 2-3-3 Enhanced SIANN for Multi-class problem.

6.3.5 Sunspot Time Series

The 10-5-1 Reduced SIANN, the 10-5-3-1 Multi-layer SIANN and the standard 10-10-1 SIANN were trained and tested on the Sunspots time series problem. The results are shown in Table 6.5 and Fig. 6.10. The figure shows the best, mean and median error rates of the test set ARV metric, defined in Chapter 4, as well as the average training times.

The general trend, for all training algorithms, is that the standard SIANN structure has the best accuracy followed by the Reduced SIANN, with the Multi-layer SIANN coming out worst. The trend is most pronounced for the GDX algorithm, with the mean test ARV for the Multi-layer SIANN 2 to 3 times that of the other structures.

The training time results show that the Reduced SIANN takes only marginally less time than the Standard SIANN to train, despite having half the number of synaptic weights. On the other hand, even though the multi-layer SIANN has fewer weights than the standard SIANN, the time taken to train was higher for all three algorithms, as the number of epochs required was higher. This is most pronounced with the LM algorithm, with the Multi-layer SIANN taking 9 times longer to train compared to the standard SIANN. The small saving in training time with the Reduced SIANN comes at the cost of a decrease in accuracy compared to the standard SIANN. It is this potential saving in training time that motivated the development of the Reduced SIANN structure, but the cost in accuracy does not always justify the saving.

The Multi-layer SIANN did not perform well, coming out worst both in terms of accuracy and training time, with all three training algorithms.

Table 6.5 Results for Sunspots dataset using Enhanced SIANNs

SIANN Struct.	Training Algor.	Act-fns		Performance (% of runs)			Avg Epochs	Test MSE		Test ARV				Avg. CPU time (s)
		Sh	Out	→ gl	all in tol	80% tol		Best	Median	Best	Median	Mean	95% CI	
Reduc. 10-5-1 (71 wts)	GDX	Tnh	Tnh	0	30	98	164	0.0089	0.0158	0.106	0.189	0.213	± 0.028	6.7
	LM	Lgs	Tnh	0	2	100	60	0.0073	0.0104	0.087	0.125	0.127	± 0.004	8.4
	DSGDX	Tnh	Lin	0	16	100	163	0.0082	0.0101	0.098	0.122	0.123	± 0.005	6.9
Expand 10-5-3-1 (93 w.)	GDX	Tnh	Tnh	0	4	68	165	0.0122	0.0311	0.147	0.373	0.515	± 0.097	10.2
	LM	Lgs	Lgs	0	62	98	504	0.0080	0.0107	0.096	0.128	0.133	± 0.006	90.6
	DSGDX	Exp	Lin	0	32	100	169	0.0084	0.0101	0.100	0.121	0.133	± 0.016	10.5
Stand. 10-10-1 (141 w)	GDX	Tnh	Lin	0	14	98	147	0.0085	0.0134	0.102	0.161	0.174	± 0.020	6.8
	LM	Lgs	Lin	0	8	100	54	0.0075	0.0093	0.090	0.111	0.112	± 0.003	9.8
	DSGDX	Lgs	Lin	0	40	100	161	0.0077	0.0097	0.096	0.121	0.119	± 0.002	7.6

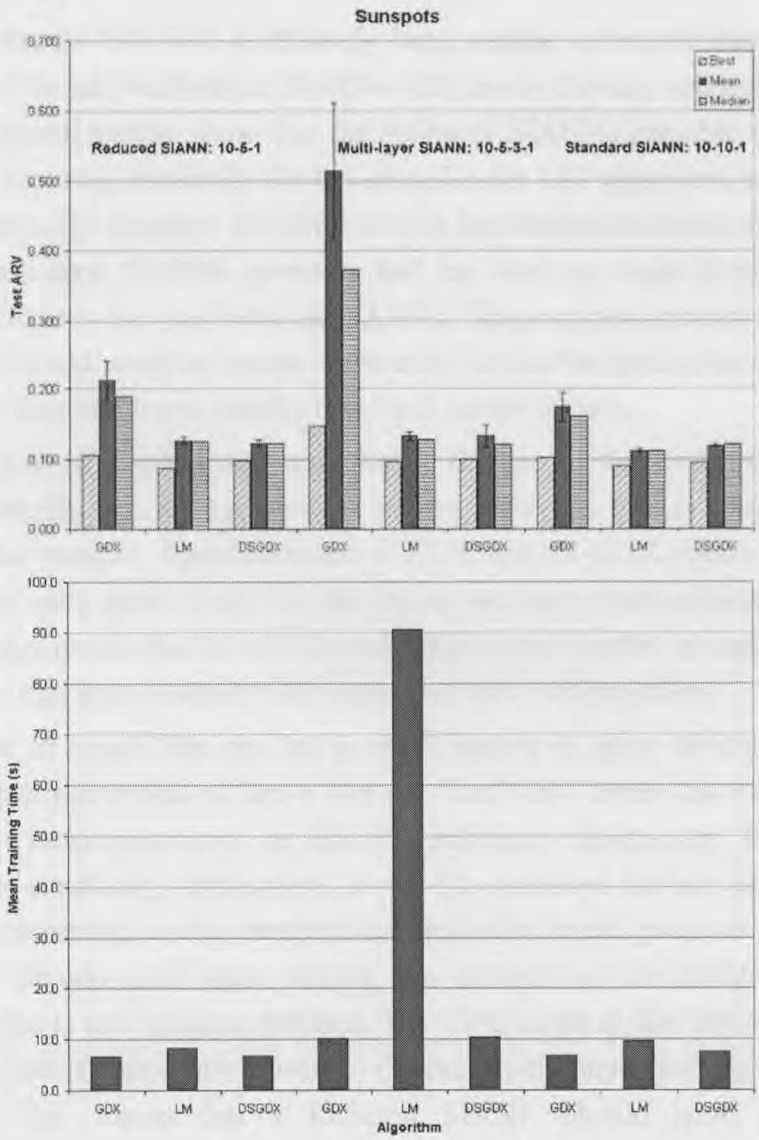


Fig. 6.10: Best, mean and median test error and mean training time for Sunspots data using Enhanced SIANNs.

6.3.6 Analysis of Results

From the results presented in the preceding subsections, the cases where the number of inputs is small clearly highlight the advantage of being able to expand the SIANN structure. The Expanded SIANNs have higher accuracy and most of them also have shorter training times. Conversely, reducing the number of shunting neurons results in lower performance, sometimes dramatically lower, as the network becomes too small to handle the problem. Either the error rate or the training time goes up significantly, or both. However, the reduced SIANN structure can still produce good results when the number of inputs is large; for example, the 8-3-1 reduced SIANN structure achieved the lowest "best-case" error of 16.67% on the Diabetes problem. In any case, these results justify the use of the 'interneurons' in the single-layer Expanded SIANNs.

The benchmarks tests with a relatively large number of inputs were tested with Reduced SIANNs and Multi-layer SIANNs with fewer shunting neurons than system inputs. The overall results show that the Reduced SIANNs are able to reduce the training time required, especially for the second-order LM algorithm, but with some loss in accuracy. The standard SIANN networks had the best accuracy in most cases, while the Multi-layer SIANNs generally had the worst accuracy even though they had larger structures than the Reduced SIANNs. The poor performance of the Multi-layer SIANNs could possibly be due to the error surface becoming too complex with the additional shunting layer, thereby making it harder to train.

The reduction in training time may justify the use of the Reduced SIANNs in some cases, but the time saving does not always justify the loss in accuracy. This is more so for the 'simpler' algorithms such as GDX and DS-GDX, where the time cost savings are not very great, if any, but the loss in accuracy tends to be high. The loss of accuracy may not be due to only the reduction in the number of weights, but also due to the fact that only a subset of the inputs are fed in as excitatory.

The choice of inputs that are fed in as excitatory is quite arbitrary. In all the benchmark tests performed so far, it was the 'first' few inputs that were fed in as excitatory; the remaining inputs are fed in as inhibitory inputs only. This was done for the sake of simplicity; furthermore, in real-life situations the role of each input is not known beforehand, unless the problem definition itself gives an indication of which inputs should carry more weight. An example of the problem definition providing a clue is the Sunspots problem. The 'first' input is the 'latest' observation i.e. the point that is temporally closest to the predicted output, and inputs are sorted accordingly. This means that a Reduced SIANN should have the 'closest' observation inputs used as excitatory and the earlier values as inhibitory only. For other problems, it may not be possible to arrive at such an 'ordering' of the inputs without some pre-processing and analysis, if at all.

6.3.7 Results obtained by re-ordering inputs

In order to get a feel of the effect of changing the excitatory inputs, the best performing Reduced SIANNs for the Breast Cancer and Diabetes problems were tested with different input permutations so that different inputs could serve as excitatory. For both datasets, the inputs used previously as excitatory were changed to inhibitory only and the succeeding inputs were fed as both excitatory and inhibitory. Tables 6.6 and 6.7 compare the results before and after input-reordering.

For the Breast Cancer dataset, the change in excitatory inputs resulted in an obvious degradation in performance of all algorithms. It is especially clear for the LM and DS-GDX algorithms, with the mean error rate up from 0.2% to 0.7% and the percentage of networks achieving perfect results down from 68% to 30% or less. For the Diabetes problem, the change shows a similar trend, with the mean error rate increasing between 0.1% and 1.3% and the number of networks achieving under 20% error dropping by one-third or more.

These differences were the result of what amounts to arbitrary changes in the selection of excitatory inputs. Without analysis of the data or experimental results, it is generally not possible to decide which inputs should be used as excitatory. There is also the possibility of errors or missing values resulting in some inputs being redundant or causing spurious connections, as is the case with the Pima Indians Diabetes dataset (Arulampalam & Bouzerdoum, 2002a; Waschulzik et al., 2000).

To find the 'optimal' network would require not only the selection of a network structure, but also testing all possible combinations of excitatory inputs for that structure in order to find the optimal combination; this is not a practical option. Ideally, there should be a method that allows all the inputs to the network to serve both as excitatory and inhibitory. A generalised feedforward architecture that caters for this is presented in the next chapter.

Table 6.6 Results for Wisconsin Breast Cancer dataset using the 9-4-1 Reduced SIANN, with re-ordered inputs

SIANN Structure	Training Algor.	Activation functions		Performance (% of runs)			Avg Epochs			Test Error				CPU time (s)
		Sh	Out	→ goal	0% err	20% <	All runs	→ goal	0% Error	Best (%)	Mean (%)	95% CI	Med. (%)	
Reduced 9-4-1	GDX	Exp	Lin	0	16	100	145	*	141	0	0.55	± 0.09	0.56	9.0
	LM	Exp	Lin	0	68	100	147	*	136	0	0.23	± 0.10	0	28.6
	DSGDX	Exp	Lin	0	68	100	193	*	184	0	0.26	± 0.13	0	11.7
Reduced 9-4-1 (re-ordered inputs)	GDX	Exp	Lin	0	10	100	191	*	132	0	0.72	± 0.12	0.56	11.6
	LM	Exp	Lin	0	30	100	131	*	131	0	0.79	± 0.20	0.56	26.0
	DSGDX	Exp	Lin	0	22	100	195	*	187	0	0.72	± 0.16	0.56	11.6

Table 6.7 Results for Pima Indians Diabetes dataset using the 8-3-1 Reduced SIANN, with re-ordered inputs

SIANN Structure	Training Algor.	Activation functions		Performance (% of runs)			Avg Epochs			Test Error				CPU time (s)
		Sh	Out	→ goal	0% err	20% <	All runs	→ goal	0% Error	Best (%)	Mean (%)	95% CI	Med. (%)	
Reduced 8-3-1	GDX	Tnh	Tnh	0	0	20	192	*	*	18.23	20.96	± 0.31	21.35	11.9
	LM	Exp	Tnh	0	0	56	58	*	*	16.67	20.18	± 0.36	19.79	11.9
	DSGDX	Exp	Lin	0	0	42	189	*	*	18.23	20.62	± 0.38	20.57	11.8
Reduced 8-3-1 (re-ordered inputs)	GDX	Tnh	Tnh	0	0	10	184	*	*	19.27	22.27	± 0.50	21.88	11.8
	LM	Exp	Tnh	0	0	28	60	*	*	17.19	20.74	± 0.39	20.57	12.6
	DSGDX	Exp	Lin	0	0	30	183	*	*	17.71	20.76	± 0.41	20.57	11.9

6.4 Conclusion

The motivation behind the enhancement of the standard SIANN network structure has been outlined in this chapter. The proposed enhancements allow the network size to be expanded or reduced as required. Details of the structures and the experimental results obtained using such structures on benchmarks problems have also been presented along with those of the standard SIANN

The experimental results show that expanding the SIANN structure, by adding additional shunting neurons, improves the results when tackling complex problems with a small number of inputs; it helps improve accuracy and reduce the time required to train the network.

The reduced SIANN structure was obtained by reducing the number of shunting neurons to less than the number of inputs when working on problems with large number of inputs; this requires some inputs to be used as inhibitory only. The results show that the Reduced SIANN structure is able to shorten the training time in some cases, particularly when using the Levenberg-Marquardt (LM) algorithm. This reduction generally also results in a reduction in accuracy, as there are fewer weights to be trained and only a subset of the inputs can be used as excitatory. The selection of the excitatory inputs was arbitrary and it was shown to have an impact on the results.

In conclusion, the expanded form of the SIANN has been shown to improve performance where applicable, but the reduced form has limitations because not all inputs serve as excitatory. The solution would be to find some method that would enable the SIANN structure to be reduced without having to make a choice of which inputs should serve as excitatory and which should serve as inhibitory only; instead, all inputs should serve as excitatory and inhibitory simultaneously. The next chapter expands the shunting neuron structure to explore this option.

Chapter 7

A Generalised Feedforward Neural Network Architecture

7.1 Introduction

In the preceding chapter, we saw that reducing the standard SIANN structure so that there are fewer shunting neurons than inputs can lead to savings in terms of training time. In some cases, this reduced structure can perform as well as the standard SIANN, but more often it is less accurate. This is in part due to the fact that only a subset of the inputs, equal to the number of shunting neurons, can be used as excitatory input; the other inputs can only exert inhibitory influences on the activity of the network. Furthermore, there is no simple way of using prior knowledge to determine which inputs should serve as excitatory and which should not. This limitation arose from the fact that the shunting neuron used was allowed to have one excitatory input only. It was concluded in the previous chapter that one way to solve this dilemma would be to modify the structure of the shunting neuron to allow more than one excitatory input.

In this chapter, the shunting neuron model used in SIANNs is expanded to allow greater flexibility in the network structure. The result is a new neuron model that combines the shunting neuron model with the traditional perceptron model. We use this 'generalised' shunting neuron model in a feedforward architecture, which henceforth is referred to as the *generalised feedforward neural network* (GFNN). The next section describes the *generalised shunting neuron* (GSN) model and the structure of the GFNN. The third section presents experimental results obtained by

applying GFNNs to the selected benchmark problems. Finally, the discussion and the conclusion are presented in Sections 7.4 and 7.5, respectively.

7.2 Development of the Generalised Feedforward Neural Network

In this section, the *generalised feedforward neural network* (GFNN) architecture is developed as an extension of SIANNs. The basic computing element of the SIANN architecture, the static shunting neuron, is recapped next, followed by the development of the generalised shunting neuron model, then the GFNN architecture.

7.2.1 The Static Shunting Neuron and SIANNs

The starting point of the development of the new generalised shunting neuron model is the static shunting neuron model presented in Chapter 3. This model is shown here again for convenience. The static shunting neuron is defined by the equation

$$x_j = \frac{I_j + b_j}{a_j + f\left(\sum_{i=0}^{m} c_{ji} I_i\right)} \tag{7.1}$$

where x_j represents the activity (output) of the jth neuron; I_j is the input to the jth neuron; a_j is the passive decay rate of the neuron (positive constant); b_j represents the bias for the neuron; c_{ji} is the connection weight from the ith input to the jth neuron, with c_{j0} being the bias for the activation function; and f is an activation function bounded from below.

We define the denominator in (7.1) as the shunting term for the jth neuron, s_j, given by

$$s_j \equiv a_j + f\left(\sum_{i=0}^{m} c_{ji} I_i\right) \tag{7.2}$$

The term s_j is constrained to be positive definite so as not encounter a divide by zero error (i.e. $s_j > 0$). This is achieved by imposing a lower bound on the parameter a_j during the initialization and training phases (refer Chapter 4).

The static shunting neuron model is shown diagrammatically in Fig. 7.1. All inhibitory (shunting) inputs are weighted and fed into an activation function. However, as mentioned previously, the shunting neuron has only one unweighted excitatory input, which is the limitation to be addressed. To alleviate this inherent limitation, a *generalised shunting neuron model* is proposed next.

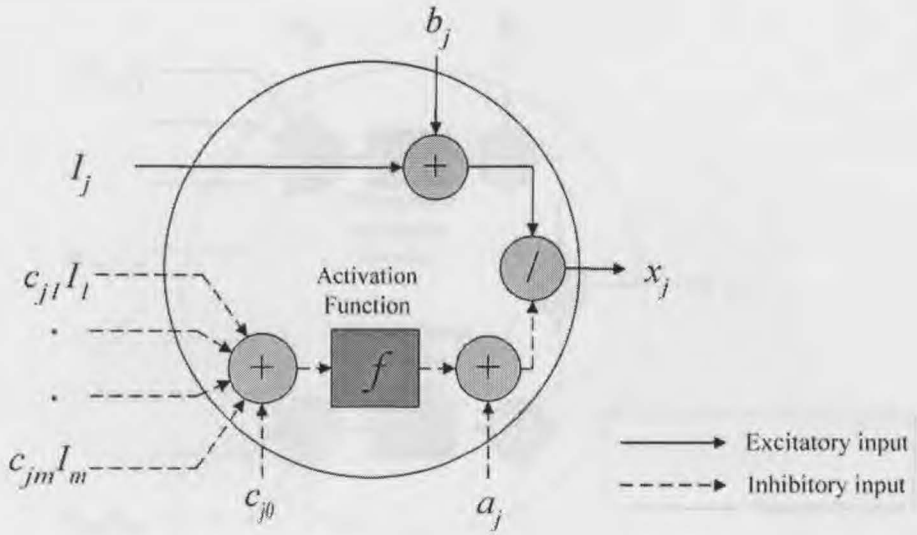


Fig. 7.1: The structure of the static shunting neuron model.

7.2.2 The Generalised Shunting Neuron Model

One of the shortcomings of the shunting neuron model described above was the fact that each neuron can only have one unweighted excitatory input. This means that either the network needs to have at least as many neurons as there are inputs, or only a subset of inputs can serve as excitatory. One way out of this is to have multiple excitatory inputs weighted, summed and passed through an activation function, as done with the perceptron neuron. In fact, the proposed new neuron model combines the perceptron neuron model with the shunting neuron model. The output of this "generalised" shunting neuron can be described by

$$x_j = \frac{g\left(\sum_{i=0}^{m} w_{ji} I_i\right) + b_j}{f\left(\sum_{i=0}^{m} c_{ji} I_i\right) + a_j} \tag{7.3}$$

It should be noted that both the perceptron neuron and the shunting neuron are special cases of this new model. The perceptron neuron is a special case of the generalised shunting neuron where the denominator weights c_{ji} are fixed at 0 and a is set to a constant that makes the denominator equal to 1, depending on the activation function f. Furthermore, (7.3) reduces to (7.1), which models the normal shunting neuron, when $w_{jj} = 1$, all other weights w_{ji} are 0, and g is the linear activation function. We have therefore named this new model the *Generalised Shunting Neuron* (GSN) model. The Generalised Shunting Neuron model is shown diagrammatically in Fig. 7.2.

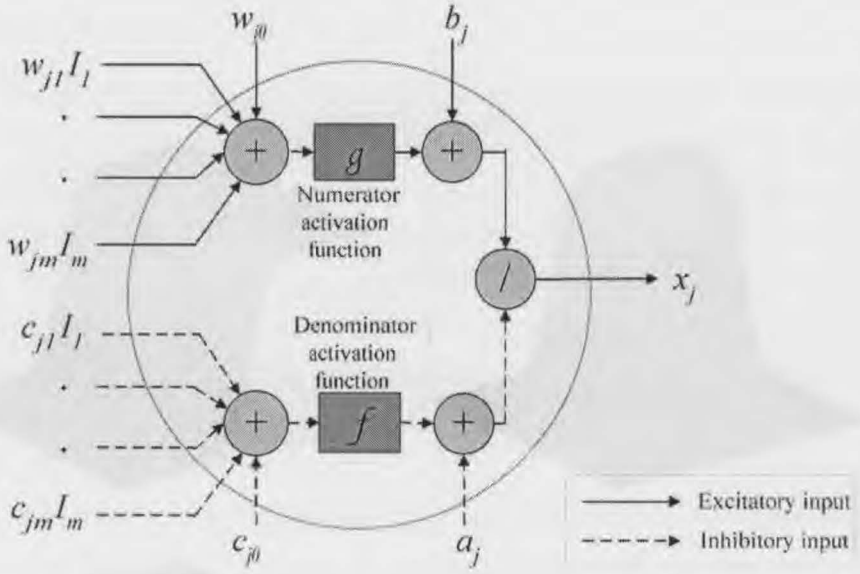


Fig. 7.2: The Generalised Shunting Neuron model.

More importantly, the input-output transfer characteristic of a generalised shunting neuron is adaptive; that is, even when the activation functions f and g, in (7.3), are fixed, the type of input-output transfer characteristic each computing element can have varies depending on its synaptic weights. Fig. 7.3 shows some input-output transfer characteristics of a generalised shunting neuron, having the logarithmic sigmoid activation function for both f and g; these transfer characteristics are obtained by simply changing the synaptic weights. More complex characteristics can be obtained by mixing different activation functions together. This is in contrast to traditional artificial neural models, such as the RBF (radial basis function) and the perceptron neurons, which have input-output transfer characteristics of fixed type, bell-shaped or sigmoid-shaped. This we believe places an artificial constraint on the type of decision surfaces a particular neuron can produce.

Jankowski & Duch have investigated the role of activation functions in neural network performance, and have used a number of transfer characteristic, such as bicentric and extended conic functions, that can produce complex decision boundaries, thus allowing the number of adaptive units in the network to be reduced (Duch & Jankowski, 2001; Jankowski, 1999; Jankowski & Duch, 2001). Neurons with these activation functions have been used in *ontogenic* and *heterogenous* neural networks. The GSN neuron is able to produce similar transfer characteristics, with the added advantage that it requires only $2N+4$ parameters per neuron instead of $3N$ or $4N$ parameters as given by Jankowski & Duch.

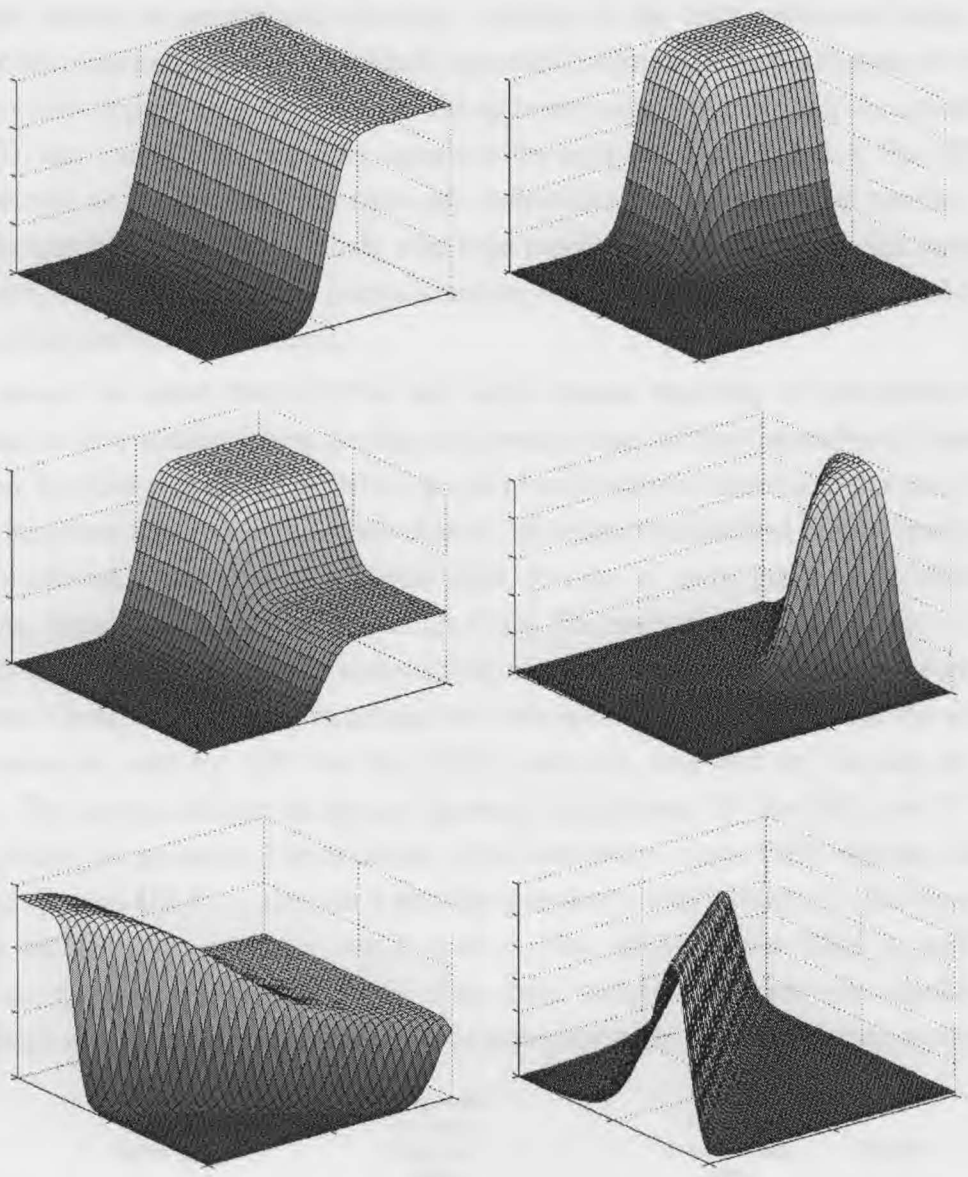


Fig. 7.3: Input-output transfer characteristics of a 2-input generalised shunting neuron obtained with the same f and g functions, but different $\mathbf{w}$ and $\mathbf{c}$ weight vectors.

7.2.3 The GFNN Architecture

The GFNN is a multilayer feedforward neural network architecture consisting of one or more layers of generalised shunting neurons. In the tests performed here, the output layer may consist of generalised, sigmoidal or linear neurons. Neurons in each layer receive inputs only from the preceding layer, calculate their outputs according to (7.3), and transmit the resulting signals to the next layer, see Fig. 7.4. The GFNN architecture as defined does not have any restrictions on the number of neurons per layer or number of layers. The only effect the problem definition has on the network structure is on the number of output neurons, which corresponds to the number of outputs required by the problem.

It should be noted that GFNNs may also contain shunting or perceptron-type neurons in their hidden layers, as they are special cases of the Generalised Shunting Neuron. In other words, the GFNN is a hybrid architecture combining shunting-type and perceptron-type neurons. In this chapter, only the two simplest GFNN structures are considered. The first is a single layer of one or more generalised shunting neurons, the simplest of which is a single GSN. The second structure consists of one hidden-layer, containing one or more GSNs, and an output layer of linear or sigmoid neurons. These two network structures are denoted by the prefix 'G' for the single layer network, and by 'GP' for the 2-layer network, followed by the size of the layers. The letters indicate the type of neuron in each layer; 'G' for GSN, and 'P' for perceptron-type neurons. For example, G 9-1 denotes a single GSN neuron with 9 inputs, whereas GP 8-2-1 denotes a two-layer network with 8 inputs, 2 GSN neurons in the hidden layer, and one output neuron. The ability of the GSN to produce complex decision boundaries means that these simple structures are capable of handling most problems, as shall be shown experimentally in the following section.

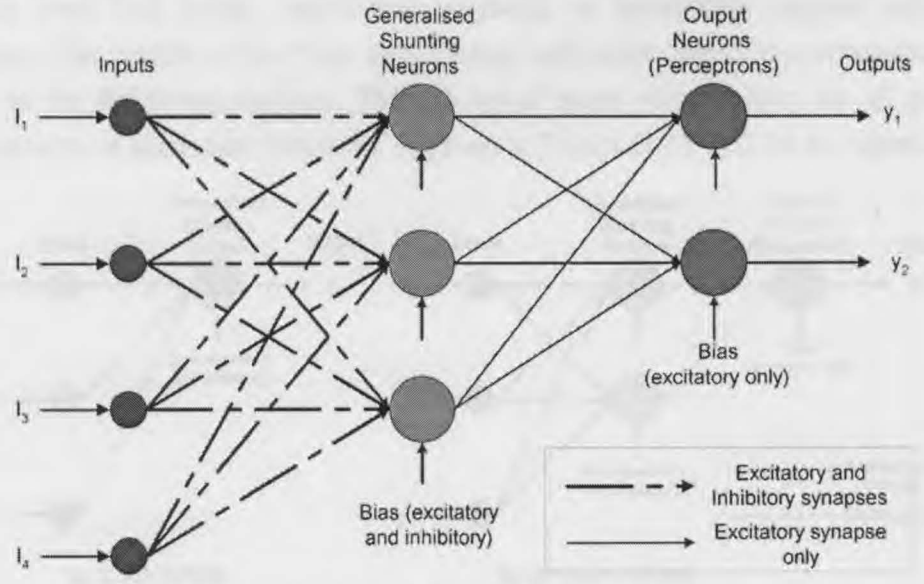


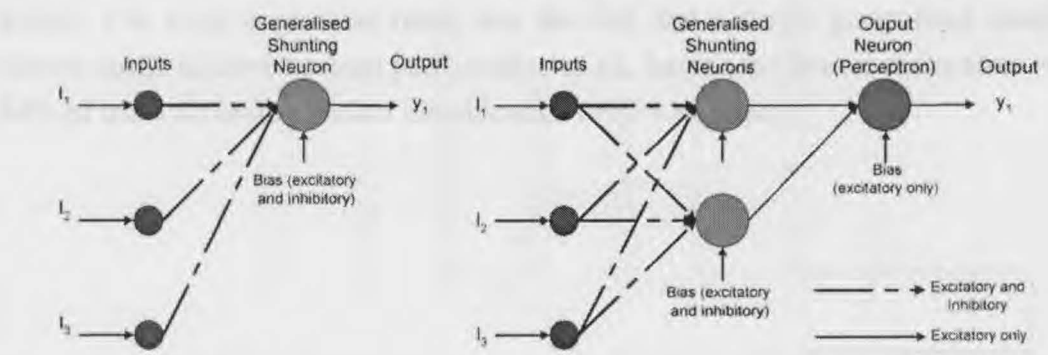
Fig. 7.4: The Generalised Feedforward Neural Network architecture.

7.3 Benchmark Test Results and Analysis

The GFNNs were tested and compared to SIANNs on the same set of benchmarks problems used in the previous chapters and the obtained results are presented in this section. For each benchmark problem, two types of GFNNs were tested. The first is a single layer of GSNs, with the number of neurons equal to the number of outputs required. For most of the problems this means a single GS neuron, with the exception of the Multi-class problem that has 3 GSNs. The second GFNN structure tested is the simplest two-layer GFNN structure - a GP *n*-2-1 structure (2 GSNs and a perceptron output), or in the case of the Multi-class problem a GP 2-2-3. Examples of these network structures are shown in Fig. 7.5.

These GFNN structures were trained and tested and their performance compared to the 'standard' SIANN. As in the previous chapters, 50 networks were generated for each structure, and these were trained using the Gradient Descent with momentum and adaptive learning rate (GDX) and the Levenberg-Marquardt (LM) algorithms. The multi-layer GFNNs and SIANNs were also trained using the Direct Solution-GDX (DS-GDX) algorithm. The single-layer GSN networks could not be trained using the DS-GDX algorithm as the algorithm requires an output layer of linear or sigmoid neurons to work. The initialisation and training parameters used are the same as described in Chapter 4.

The GSN neurons were tested with various combinations of activation functions in the numerator and denominator. Linear (*lin*), hyperbolic tangent sigmoid (*tnh*), logarithmic sigmoid (*lgs*) and exponential (*exp*) activation functions were used for the numerator. The constraint on the shunting term given in (7.3) requires the denominator activation function to have a lower bound. Therefore the linear activation function could not be used in the denominator. The output perceptron neurons used had linear, logarithmic sigmoid or hyperbolic tangent activation functions. The results of the 'best performing' activation function combinations are shown in the following sections. The full set of mean error values, for all possible combinations of activation functions, is given in Tables B.32 to B.36 in Appendix B.



a) G 3-1 GFNN b) GP 3-2-1 GFNN

Fig. 7.5: Examples of GFNN structures: (a) G 3-1 network and (b) GP 3-2-1 network

7.3.1 Wisconsin Breast Cancer dataset results

The results obtained using a single GSN (G 9-1), a GP 9-2-1 GFNN and the 'standard' 9-9-1 SIANN trained on the Wisconsin Breast Cancer problem are presented in Table 7.1 and Fig. 7.6. As in previous chapters, the graphs are broken into two sections: the top part shows the mean and median test error percentages for the best performing activation function combination for the given network structure and training algorithm, and the second part shows the corresponding mean training times. Note that the median is often zero, and hence it is not visible on the graph.

The G 9-1 network, consisting of a single generalised shunting neuron, trained using the GDX algorithm, had the lowest average error (0.16%) with 84% of networks (neurons) able to achieve perfect classification, i.e. 0% error. This simple structure was also the second fastest to train, next to the SIANN trained with DS-GDX. The single neurons trained using the LM algorithm did not work that well, with an average error of 0.44% and 'only' 22% of them achieving perfect classification.

The GP 9-2-1 GFNN trained with GDX did not do as well as the single neuron. It had the second best average error, but also the second longest training time. The GP 9-2-1 GFNN trained with LM had one of the highest mean error rates (0.49%), but still had almost half the networks achieving perfect classification. It could also be trained fast, more than twice as fast as a SIANN trained with LM. When trained with DS-GDX, the GP 9-2-1 GFNN did better than the corresponding SIANN networks; the average error rate was cut by almost half, and two thirds of the networks achieved perfect classification (compared to 36% for the SIANNs). However the average training time was twice as long despite the fact that the GFNN network had less than half the number of weights to train compared to SIANN.

Overall the GFNNs did better than the SIANNs when trained using GDX and DS-GDX algorithms. With the LM algorithm, the accuracy was not as good for the GFNNs, but there were significant reductions in training times. In reality, the average error rates achieved were not vastly different, with all achieving 99.5% accuracy or higher. The most impressive result was the fact that a single generalised shunting neuron could achieve the best performance of all, having the lowest mean error, with 84% of trials achieving perfect classification (100% accuracy).

Table 7.1 Best results for Wisconsin Breast Cancer dataset using GFNNs

Network Structure	Training Algor.	Activation functions			Performance (% of runs)			Avg Epochs			Test Error				CPU time (s)
		Shunting		Out	→ goal	0% err	20% <	All runs	→ goal	0% Error	Best (%)	Mean (%)	95% CI	Med. (%)	
		Nu	Den												
Single GSN 9-1 (22 weights)	GDX	Lin	Lgs		0	84	100	134	*	141	0	0.16	± 0.11	0.00	7.0
	LM	Lin	Lgs		0	22	100	56	*	54	0	0.44	± 0.07	0.56	9.4
GFNN GP 9-2-1 (47 weights)	GDX	Lin	Exp	Tnh	0	70	100	279	*	304	0	0.19	± 0.09	0.00	22.6
	LM	Exp	Exp	Tnh	0	48	100	57	*	57	0	0.49	± 0.16	0.56	15.7
	DSGDX	Lgs	Exp	Lin	0	62	100	162	*	145	0	0.28	± 0.13	0.00	13.4
SIANN 9-9-1 (118 weights)	GDX		Lgs	Lgs	0	66	100	161	*	160	0	0.20	± 0.08	0.00	10.3
	LM		Lgs	Tnh	0	74	100	181	*	119	0	0.20	± 0.10	0.00	34.8
	DSGDX		Tnh	Lgs	0	36	100	96	*	100	0	0.54	± 0.14	0.56	6.4

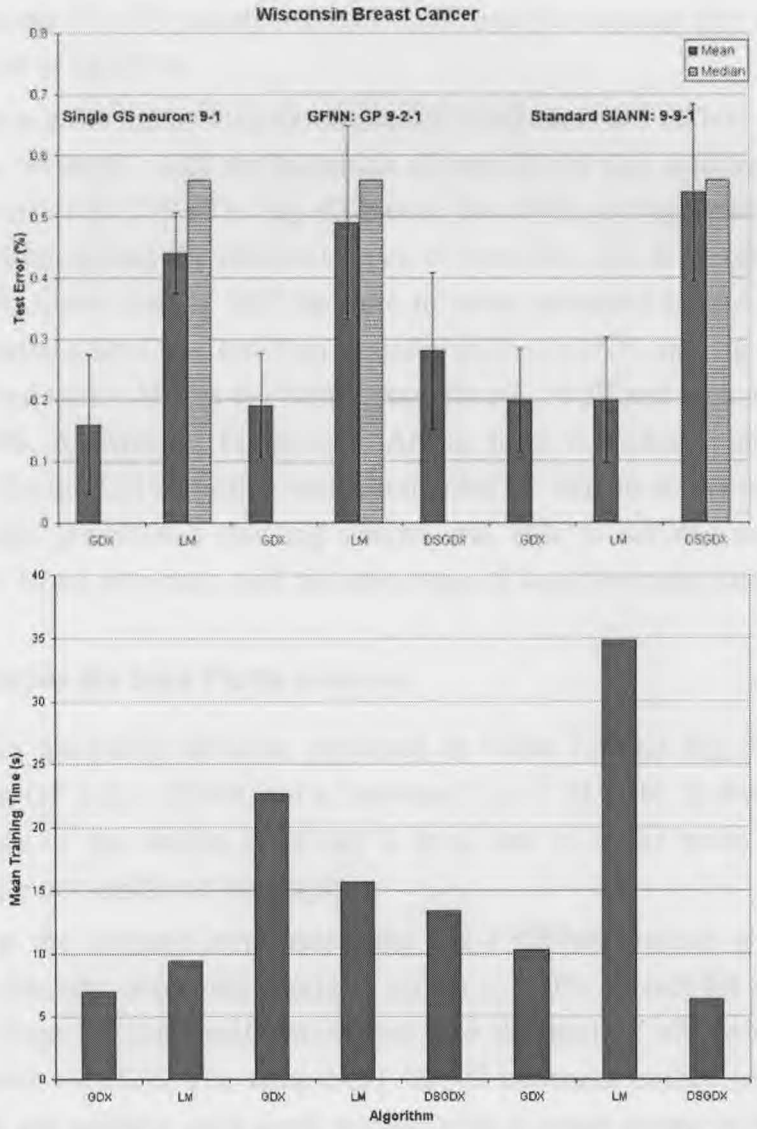


Fig. 7.6: Mean and median test error and mean training time for the Wisconsin Breast Cancer dataset using GFNNs.

7.3.2 Pima Indians Diabetes dataset results

The results obtained for this dataset, using a single GSN (G 8-1), a GP 8-2-1 GFNN and the 'standard' 8-8-1 SIANN, are shown in Table 7.2 and Fig. 7.7. The figure here shows the lowest error achieved by a single network for each case, in addition to the mean and median test error, since the 'best case' error is not zero for this problem.

Both the G 8-1 and the GP 8-2-1 GFNNs trained with GDX were able to achieve better results than the SIANN trained with the same algorithm, with a mean error of 20.6% as opposed to 21.0% for SIANNs and more than double the number of networks having error below 20%.

The 8-2-1 GFNN trained with DS-GDX was able to achieve a mean error rate below 20%, with 56% of networks achieving rates below 20%. However, this was not as good as the SIANN trained with DS-GDX, and the training time required was also double that of SIANNs.

The accuracy achieved by both the single GSN and the 8-2-1 GFNN when trained with LM was 'average', with the exception of one GFNN that achieved the lowest 'best case' error of 16.15%. The big difference lies in the average training time for the LM algorithm across the different types of network. The 8-2-1 GFNN trained with LM took approximately half the time to train compared to SIANN, and the single GSN training time was less than a quarter of the SIANN training time. In fact, the GSN trained with LM was the fastest combination of all and took an average of only 22 epochs. As with the Enhanced SIANNs, there is a clear trend linking the training time for the LM algorithm with the number of weights in the network. Here again the single generalised shunting neuron was able to achieve accuracy rates comparable to larger networks, with the advantage of faster training times.

7.3.3 Results for the 3-bit Parity problem

The results for the parity problem, presented in Table 7.3 and Fig. 7.8, are for a single GSN, a GP 3-2-1 GFNN and a 'standard' 3-3-1 SIANN. It should be noted that most cases of the median error rate is zero, and so is the mean in one case; hence, these are not visible on the graph.

Looking at the average error rates, the 3-2-1 GFNN trained with LM was undoubtedly the best, achieving 'perfect' results – 100% correct for all networks. The training time for this combination was also the best of all, twice as fast as SIANNs trained with LM. The same 3-2-1 GFNN networks trained with GDX and DS-GDX did not achieve such good results, with average errors in the region of 2.5%, but still managed to get 80% and 90% of networks achieving perfect classification respectively.

Table 7.2 Best results for Pima Indians Diabetes dataset using GFNNs

Network Structure	Training Algor.	Activation functions			Performance (% of runs)			Avg Epochs			Test Error				CPU time (s)
		Shunting		Out	→ goal	0% err	20% <	All runs	→ goal	0% Error	Best (%)	Mean (%)	95% CI	Med. (%)	
		Nu	Den												
Single GSN 8-1 (20 wt.)	GDX	Tnh	Exp		0	0	38	173	*	*	18.23	20.58	± 0.26	20.83	9.7
	LM	Tnh	Lgs		0	0	34	22	*	*	18.75	20.56	± 0.27	20.31	5.2
GFNN GP 8-2-1 (43 weights)	GDX	Tnh	Tnh	Tnh	0	0	34	175	*	*	18.23	20.58	± 0.34	20.57	15.7
	LM	Exp	Lgs	Lgs	0	0	42	58	*	*	16.15	20.36	± 0.33	20.31	17.8
	DSGDX	Lin	Lgs	Lin	0	0	56	150	*	*	17.71	19.94	± 0.27	19.79	13.5
SIANN 8-8-1 (97 weights)	GDX		Tnh	Lgs	0	0	14	195	*	*	17.71	21.03	± 0.29	21.35	13.2
	LM		Lgs	Tnh	0	0	58	182	*	*	17.71	19.88	± 0.32	19.79	38.9
	DSGDX		Lgs	Lgs	0	0	68	94	*	*	18.75	19.82	± 0.29	19.79	6.4

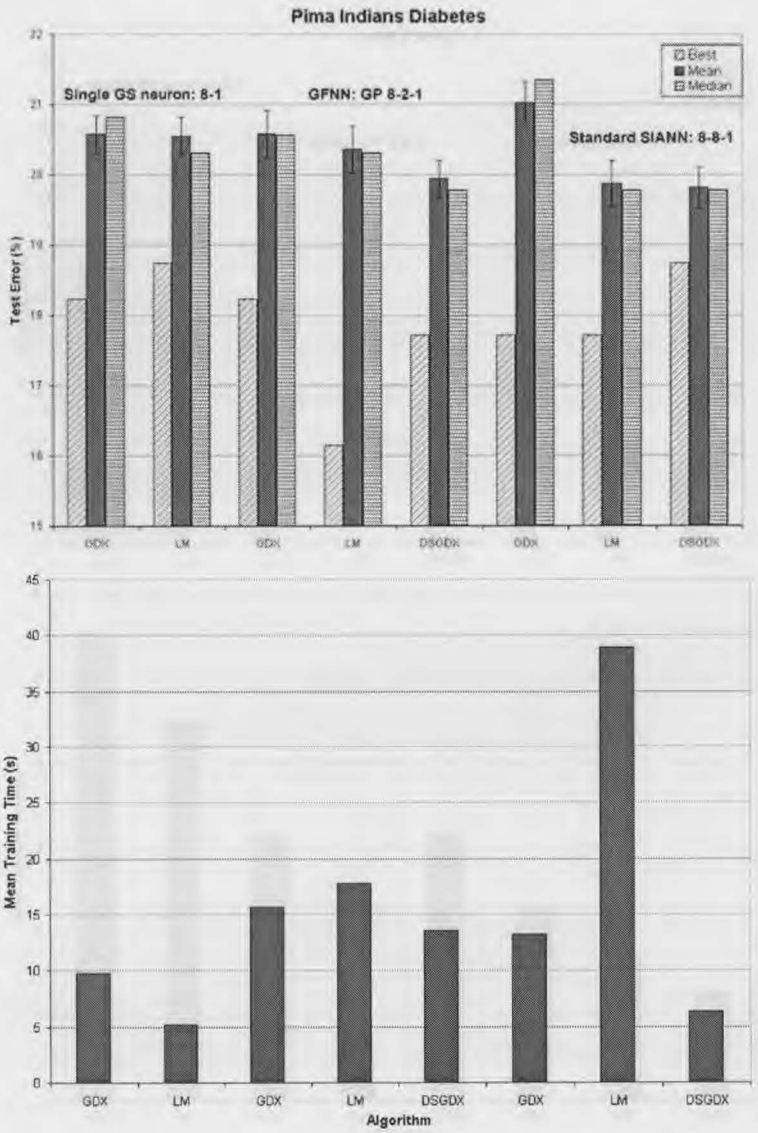


Fig. 7.7: Best, mean and median test error and mean training time for the Diabetes dataset using GFNNs.

Table 7.3 Best results for 3-bit Parity dataset using GFNNs

Network Structure	Training Algor.	Activation functions			Performance (% of runs)			Avg Epochs			Test Error				CPU time (s)
		Shunting		Out	→ goal	0% err	20% <	All runs	→ goal	0% Error	Best (%)	Mean (%)	95% CI	Med. (%)	
		Nu	Den												
Single GSN 3-1 (10 wt.)	GDX	Lin	Tnh		0	40	48	1000	*	1000	0.00	16.00	± 4.26	25.00	5.5
	LM	Exp	Lgs		64	66	72	343	153	178	0.00	11.00	± 5.03	0.00	4.5
GFNN GP 3-2-1 (23 weights)	GDX	Lin	Exp	Lgs	80	80	100	454	318	318	0.00	2.50	± 1.40	0.00	3.2
	LM	Lin	Lgs	Lin	98	100	100	30	22	30	0.00	0.00	± 0.00	0.00	0.8
	DSGDX	Lin	Lgs	Lin	76	90	92	467	344	408	0.00	2.75	± 2.45	0.00	3.2
SIANN 3-3-1 (22 weights)	GDX		Tnh	Lgs	94	94	96	352	311	311	0.00	1.25	± 1.44	0.00	2.3
	LM		Exp	Lin	96	96	100	84	46	46	0.00	0.50	± 0.69	0.00	1.7
	DSGDX		Tnh	Lin	94	94	100	169	116	116	0.00	0.75	± 0.83	0.00	1.3

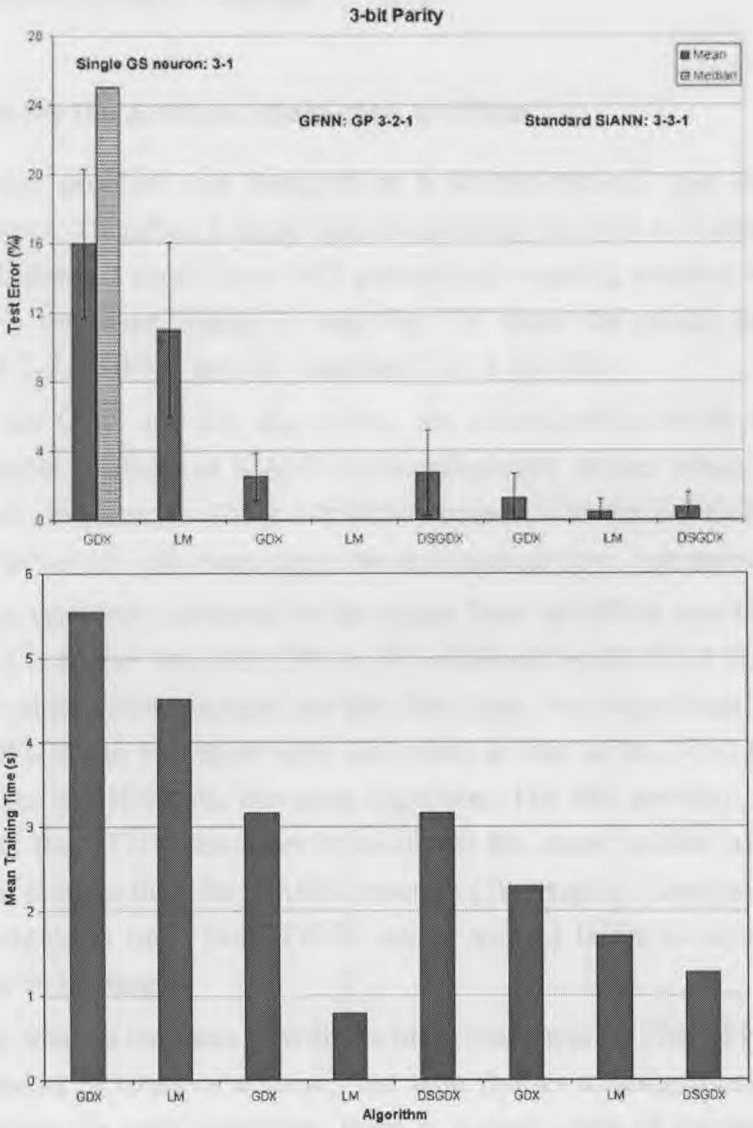


Fig. 7.8: Mean and median test error and mean training time for 3-bit parity dataset using GFNNs

The performance of the single generalised shunting neuron doesn't look good, with average error rates above 10% and no real advantage in training time. It should be noted, however, that that a single neuron is still able to solve correctly the 3-bit parity problem, in 40% and 66% of the cases when trained with the GDX and LM algorithms, respectively. This is quite an achievement when compared to a single perceptron neuron, which cannot solve problems that are not linearly separable.

Overall, the GFNNs were able to perform as well as SIANNs with a simpler structure, with the added advantage of savings in training time for the LM algorithm. For this problem, having simpler structures does not always mean less synaptic weights because of the small number of inputs. The GP 3-2-1 GFNN has 3 neurons and 23 weights compared to the SIANN with 4 neurons but only 22 weights. The complexity of the GSN in terms of number of weights, in this case, offsets the savings in terms of number of neurons.

7.3.4 Results for the Artificial Multi-class problem

The test for this problem was designed as a winner-take-all type output with 3 possible outcomes. Therefore a single neuron could not be used, as 3 separate outputs are required. Instead, a single layer of 3 generalised shunting neurons with 2 inputs (G 2-3 GFNN) was used. Table 7.4 and Fig. 7.9 show the results for the G 2-3 GFNN, the GP 2-2-3 GFNN and the 'standard' 2-2-3 SIANN.

With both the GDX and LM algorithms, the classification results for the 2-2-3 GFNN are similar to those of SIANN (with marginally higher error), but training times are about 30% shorter. The 2-2-3 GFNN trained with the DS-GDX algorithm, achieved a reduction in both mean error rate and training time, compared to SIANN.

The average error rate achieved by the single layer of GSNs was between 6.5% and 7%, which is higher than the 5.5% to 6% achieved by the 2-2-3 GFNN and the SIANN. The training time required, on the other hand, was significantly lower. The single-layer GFNN can be trained with LM twice as fast as the 2-2-3 GFNN and 3 times as fast the SIANN with the same algorithm. For this problem, it should be noted that both the GFNN structures have almost the same number of weights (24 and 25), which is more than the SIANN structure (19 weights). Despite the fact they have more weights to train, the GFNNs can be trained faster as they require less epochs to achieve the target.

Overall, the trend is the same as with the other benchmarks. The GFNNs achieved comparable results in terms of accuracy but with shorter training times. Looking at the training times for each algorithm, there is a clear trend of increasing training times, as one goes from the G 2-3, to the GP 2-2-3 GFNN and finally to the SIANN. This is most pronounced for the LM algorithm. An example of the decision boundary formed by the GFNN is shown in Fig. 7.10.

Table 7.4 Best results for Multi-Class dataset using GFNNs

Network Structure	Training Algor.	Activation functions			Performance (% of runs)			Avg Epochs			Test Error				CPU time (s)
		Shunting		Out	→ goal	0% err	20% <	All runs	→ goal	0% Error	Best (%)	Mean (%)	95% CI	Med. (%)	
		Nu	Den												
GFNN G 2-3 (24 wt.)	GDX	Lgs	Exp		0	0	100	173	*	*	4.67	6.81	± 0.43	6.67	9.0
	LM	Lin	Exp		0	0	98	58	*	*	5.33	6.65	± 1.19	6.00	28.1
GFNN GP 2-2-3 (25 weights)	GDX	Lin	Lgs	Lgs	0	0	100	180	*	*	4.00	5.61	± 0.17	5.33	13.2
	LM	Lin	Lgs	Tnh	0	0	100	99	*	*	4.00	5.79	± 0.17	6.00	65.4
	DSGDX	Lgs	Exp	Lgs	0	0	100	110	*	*	4.67	7.21	± 0.77	6.00	8.5
SIANN 2-2-3 (19 weights)	GDX		Exp	Lgs	0	0	100	377	*	*	4.00	5.47	± 0.16	5.33	19.4
	LM		Exp	Lgs	0	0	100	228	*	*	4.00	5.69	± 0.26	5.33	103.0
	DSGDX		Exp	Lin	0	0	90	206	*	*	5.33	9.39	± 2.24	6.67	10.9

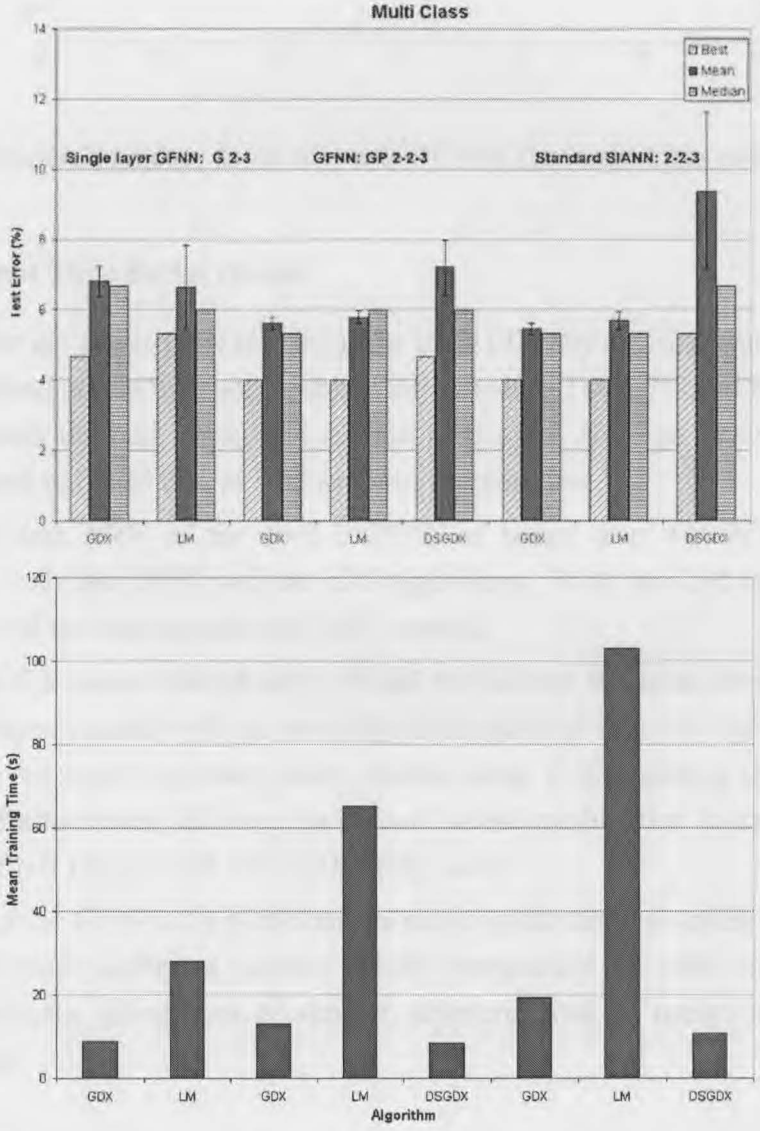


Fig. 7.9: Best, mean, median test error and mean training time for Multi-class dataset using GFNNs.

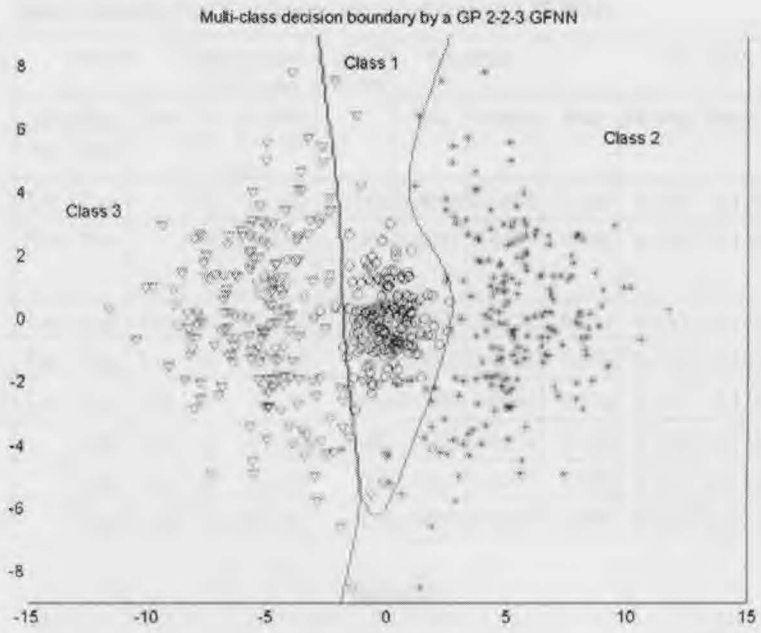


Fig. 7.10: Decision boundary formed by a GFNN for the Multi-class problem.

7.3.5 Sunspot Time Series results

The results for the single GSN (G 10-1), the 10-2-1 GFNN and the 'standard' 10-10-1 SIANN trained on the Sunspots problem are shown in Table 7.5 and Fig. 7.11. The figure illustrates the best, mean and median of the test Average Relative Variance (ARV), defined in Eq. (4.30), as well as mean training time.

The mean test ARV of the 10-2-1 GFNN is better than that of the standard SIANN, for both the GDX and the LM algorithms. With the LM algorithm, the GFNN achieved the lowest mean test ARV overall.

The single GS neuron trained with LM did not achieve the same level of accuracy as the other types trained with the same algorithm but was faster to train. The single neuron achieved better accuracy when trained using GDX, getting the best result using GDX. Furthermore, this was the second fastest combination to train, with only the 10-2-1 GFNN trained with DS-GDX being faster.

Once again the GFNNs, in particular the single generalised shunting neuron, have demonstrated their ability to achieve results comparable to other more complex networks, with the advantages of simpler structures and, in many cases, reduced training times.

Table 7.5 Best results for Sunspots dataset using GFNNs

Network Struct.	Training Algor.	Act-fns			Performance (% of runs)			Avg Epochs	Test MSE		Test ARV				CPU time
		Shunting		Out	→ gl	all in tol	80% tol		Best	Median	Best	Median	Mean	95% CI	
		Nu	Den												
Single GSN 10-1 (24 wt.)	GDX	Lin	Lgs		0	0	100	159	0.0088	0.0099	0.105	0.119	0.120	± 0.002	5.5
	LM	Exp	Tnh		0	10	100	64	0.0071	0.0107	0.085	0.129	0.140	± 0.019	7.3
GFNN GP 10-2-1 (51 wt.)	GDX	Lin	Lgs	Lin	0	6	100	146	0.0095	0.0122	0.113	0.147	0.146	± 0.004	8.0
	LM	Lin	Lgs	Lin	0	8	100	61	0.0065	0.0083	0.078	0.100	0.100	± 0.004	10.9
	DSGDX	Lin	Lgs	Lin	0	0	100	106	0.0083	0.0095	0.118	0.135	0.138	± 0.004	4.4
SIANN 10-10-1 (141 weights)	GDX		Tnh	Lin	0	14	98	147	0.0085	0.0134	0.102	0.161	0.174	± 0.020	6.8
	LM		Lgs	Lin	0	8	100	54	0.0075	0.0093	0.090	0.111	0.112	± 0.003	9.8
	DSGDX		Lgs	Lin	0	40	100	161	0.0077	0.0097	0.096	0.121	0.119	± 0.002	7.6

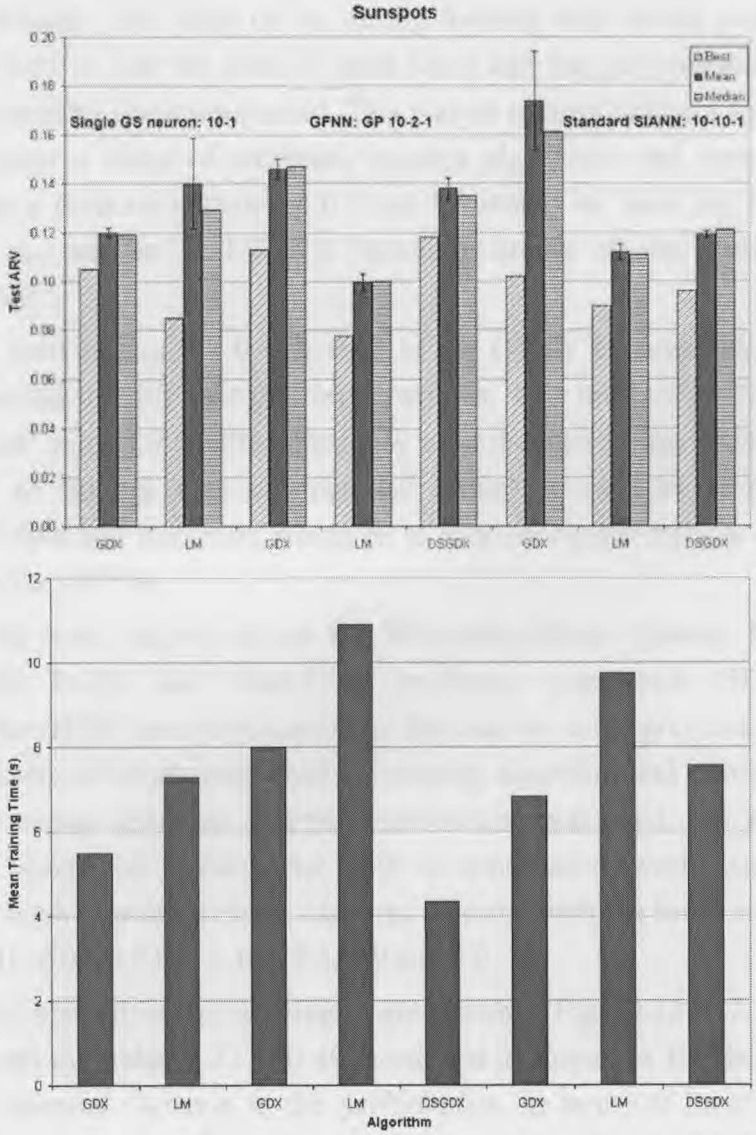


Fig. 7.11: Best, mean and median test error and mean training time for Sunspots data using GFNNs.

7.3.6 The 'Optimal' Lower-Bound of s

As discussed in Chapter 4, the denominator of the shunting neuron (or shunting term) s, given in (7.2), is constrained to be positive definite so as to avoid a divide by zero condition. By definition, the GSN contains the same shunting term, with the same constraints. During the training of the GFNNs, as with SIANNs, a lower limit for the s term, s_{lim}, is set. The limit s_{lim} correspondingly determines the lower bound for the parameter a during training, depending on the lower bound of the denominator activation function.

Previously in Chapter 4, it has been shown that changing the value of s_{lim} used during training affects the stability and duration of training for SIANNs, as well as the accuracy of the trained network. Tests were conducted on SIANNS using a number of combinations of problems, training algorithms, network structures and activation functions. The value of s_{lim} during training was varied over a range of values (from 0.01 to 2.0) for each of these cases and the performance in terms of accuracy and training times were noted. This was an attempt to find a limit value that worked best over a range of problems, training algorithms and networks. It was concluded that a limit value between 0.5 and 1.0 would be 'best' for SIANNs, and subsequently s_{lim} was set to 1.0 as a 'standard' across all the benchmarks tests conducted so far.

The same limit of $s_{lim} = 1.0$ was used in the GFNN experiments, to maintain consistency across the different network structures. This limit value, however, may not be the 'best' value for GFNNs since they have their own transfer characteristics. One method of finding such an 'optimal' value for s_{lim}, within the practical constraints of time and resources, would be to conduct similar tests as were done in Chapter 4, for the GFNNs.

Experiments were carried out on the Wisconsin Breast Cancer, Pima Indians Diabetes, 3-bit Parity, and Multi-Class problems, using both GDX and LM algorithms. The GFNN structures used were the same as in the previous sections. For each combination of benchmark problem, training algorithm and network structure, the best performing activation function combination was used. For each network structure and benchmark problem, the same 50 initialised networks used previously were trained. Each network training case was repeated with the lower limit s_{lim} set to the values 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1.0 and 2.0.

The results obtained, as s_{lim} is changed, are shown in Figs. 7.12 to 7.15. Details of results are given in Tables B.37 to B.40, presented in Appendix B. The results show that there is minimal variation in the performance, in terms of both classification accuracy and training time. There is a consistent and significant effect on the results only when s_{lim} becomes greater than 1. The results show a drop in performance, marked by an increase in the error rate and sometimes also an increase in training

time, in the cases that are affected. The pattern that emerges is that only the GDX trained networks appear to be affected by changing s_{lim}. All the networks trained with LM have remarkably constant performance across the range of values of s_{lim} tested.

For the Breast Cancer and Diabetes problems, the networks trained using GDX show an increase in the error rate for s_{lim} greater than 1. The two single-layer GSN networks, G 3-1 trained on the 3-bit parity problem and G 2-3 trained on the Multi-Class problem, are the only networks that buck this trend. Their performances vary significantly as the value of s_{lim} is varied; actually improving as s_{lim} increases.

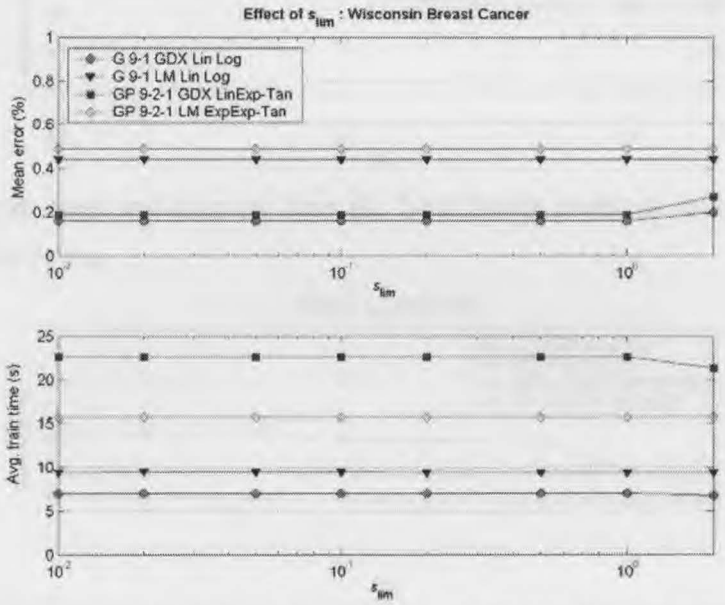


Fig. 7.12: Mean error and training time for Breast Cancer dataset using GFNNs with various s_{lim}.

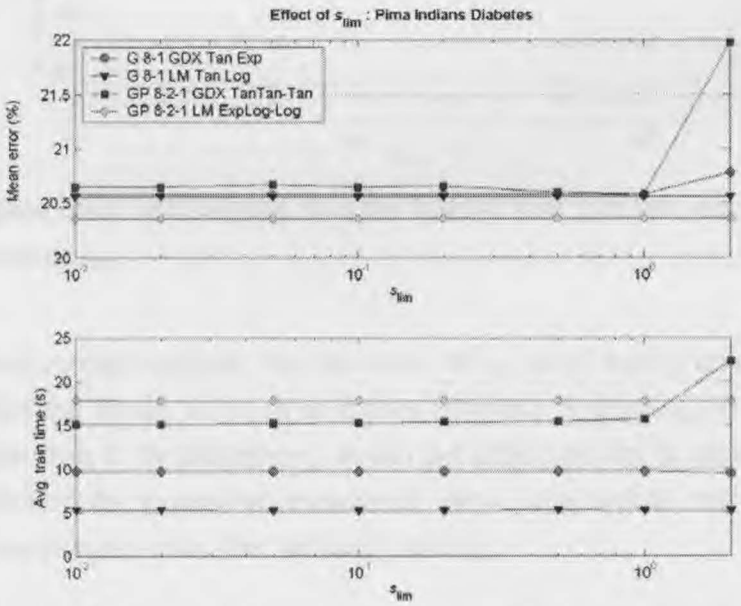


Fig. 7.13: Mean error and training time for Diabetes dataset using GFNNs with various s_{lim}.

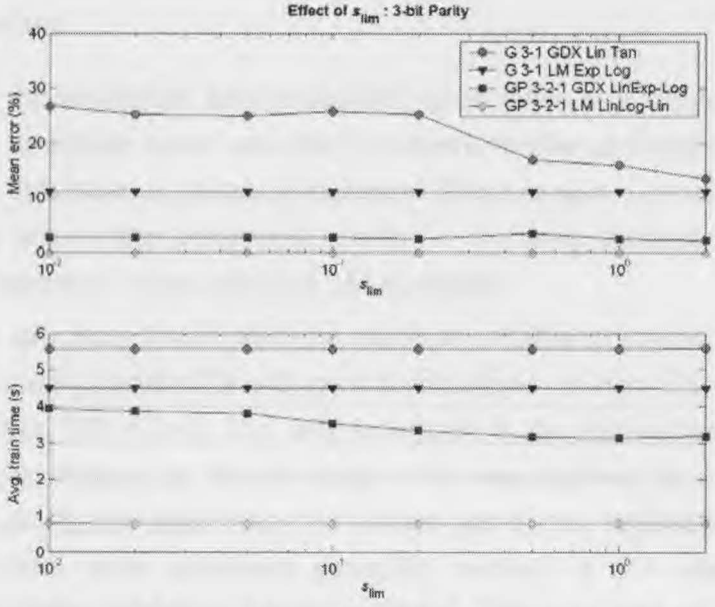


Fig. 7.14: Mean error and training time for 3-bit Parity problem using GFNNs with various s_{lim}.

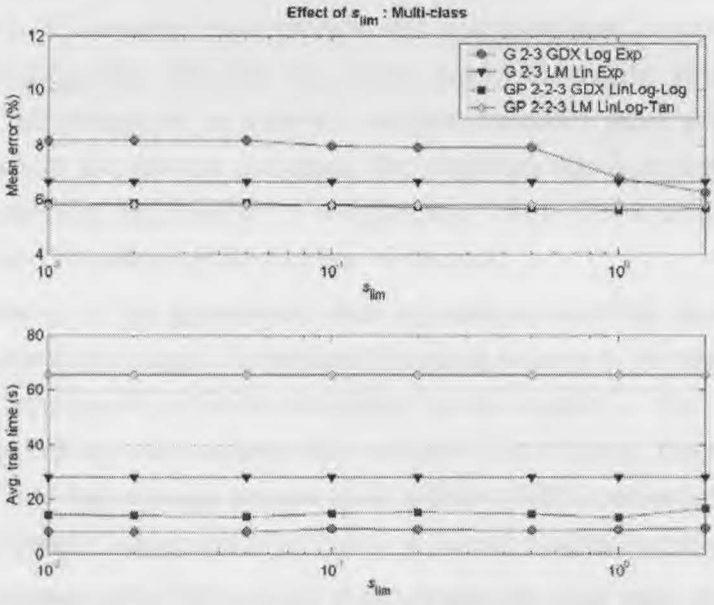


Fig. 7.15: Mean error and training time for Multi-Class problem using GFNNs with various s_{lim}.

These results would indicate that the value of s_{lim} used during training of GFNNs does not affect the results as much as it does SIANNs. Setting s_{lim} to a value of 1 or less (but more than 0, by definition), would not affect results in most cases. Taking into consideration the exceptions mentioned above, this would make s_{lim} = 1.0, as used in the benchmarks tests, the 'optimal' setting.

7.4 Discussion

The results of the benchmark tests conducted, when taken as a whole, show that the simple GFNN structures tested are able to achieve similar or better results than the larger SIANN structures in terms of accuracy. There is also a consistent pattern of faster training times when comparing results for the same training algorithm. This saving is most apparent when using the LM algorithm.

The GFNN structures tested were the simplest possible structures: either a single generalised shunting neuron (G n-1) or a hidden layer of two GS neurons with a perceptron output (GP n-2-1). The only exception is the Multi-Class problem that requires multiple outputs; for this the single GSN was replaced by a single layer of GS neurons, and in the other case the output perceptron replaced by a layer of perceptrons. These small structures generally resulted in less synaptic weights, provided the number of inputs is greater than 2. This was one of the motivating factors in enhancing the SIANN structure that ultimately led to the formulation of the GFNN. For large problems, it was reasoned that smaller structures would lead to saving in terms of memory requirements and computational complexity, hence a reduction in training time. This has been borne out by the results obtained. It should be noted that this reduction in synaptic weights becomes more prominent as the number of inputs to the network increases. For problems that have a small number of inputs, the complexity and number of weights in a single GS neuron tends to offset the gains obtained by reducing the number of neurons.

The 'complexity' of the generalised shunting neuron, however, is what gives it its power. It has enabled a single Generalised shunting neuron to be used to solve four out of the five benchmark problems considered in this chapter — the exception being the Multi-class problem that requires three outputs. Out of these, three are real world problems. The fact that a single neuron could achieve 100% correct classification for the Wisconsin Breast Cancer and 3-bit Parity problems is a noteworthy point.

Another advantage of GFNNs is that it can reduce the time taken to find the 'best' network structure for the problem. A network that is too small may not be able to 'solve' the problem, whereas too large a network will result in overfitting. With the GFNNs, the search starts with just a single neuron, and it appears that the size of the network required will normally be small, thereby reducing the number of network structures that need to be tested. This search for an optimal structure is a hidden time and effort — a cost that does not show up on training time results.

The results obtained when attempting to find the optimal value for the lower bound of the shunting term, s_{lim}, show that the limit hardly has any impact on the results, particularly if it is kept at or below 1. This means that the shunting term (or denominator) would not be going below the limit value of one, in most cases. Looking at this from another perspective, the effect of the shunting term, which is

controlled by the inhibitory synapses, would therefore be to reduce the magnitude of the output of the numerator that is driven by the excitatory synapses. This is intuitively appealing, as the function of inhibition is to lower the output of the neuron or network, and the opposite for excitation. This would not be the case if the shunting term were to go below one, as it then would be amplifying the output of the numerator. Put simply, the GSNs operate such that the excitatory and inhibitory inputs affect the output in the expected manner.

While the results presented here show good performance by the GFNNs, Arulampalam and Bouzerdoum have obtained somewhat better results using some different initialisation and training conditions (Arulampalam & Bouzerdoum, 2003a, 2003b). The first difference is that the value of s_{lim} used was 0.1, instead of 1.0. Secondly, the network initial weights were generated using a normal distribution, instead of a uniform distribution.

With these alternate conditions, the single generalised shunting neuron (G 9-1) trained with GDX and LM algorithms on the Breast Cancer was able to achieved 94% and 90%, respectively, of networks having 0% error. This is an improvement on the 84% and 22% achieved using the standard conditions used in this thesis. For the 3-bit Parity problem, the single shunting neuron was able to get 94% of neurons achieving 0% error when trained with LM, compared to 66% with standard conditions. The mean error also dropped from 11.0% to 1.5%. It has been shown that the value of s_{lim} has minimal impact on GFNNs, particularly when trained with LM. This would indicate that this performance improvement is due to the initialisation scheme. The improvements are problem dependent, however, as the results for the Diabetes problem were worse using these alternate conditions.

An alternative method tried out was not to use the bias term, b, in the numerator, by setting b to 0 and not varying it during training. A single shunting neuron trained with GDX in this manner was able to achieve 'perfect' results on the Wisconsin Breast Cancer dataset – all 50 'networks' achieving 100% correct classification. These results reinforce the conclusion that GFNNs are a powerful class of networks, able to achieve good classification results, even with a single neuron.

7.5 Conclusion

In this chapter we have presented the motivation behind the development of the Generalised Feedforward Neural Network (GFNN) architecture, initiated by the need to overcome some of the limitations of SIANNs. The development of the Generalised Feedforward Neural Network (GFNN) architecture and structure of its basic building block, the Generalised Shunting Neuron (GSN) model, have been presented.

The Generalised Shunting Neuron model presented here includes both the shunting inhibitory neuron and the perceptron neuron as special cases. The ability of the GSN depends on the combination of numerator and denominator activation functions used. It has been shown that a particular combination of activation functions can produce various types of transfer characteristics by simply varying the synaptic weights.

Details of experimental results obtained using GFNNs on benchmarks problems have been presented. Investigations were also carried out to determine the 'best' lower bound for the shunting term, s_{lim}. The results obtained show that the GFNNs are able to achieve comparable or better results than SIANNs for the benchmarks problems, using smaller, simpler network structures. There is also generally a saving in terms of training time, especially when using the LM algorithm. The most striking fact was that a single neuron could actually be used as a viable 'network' for these problems.

The Generalised Shunting Neuron is capable of producing complex decision boundaries, and hence it is able to solve some real word classification and regression problems. This is exemplified by the perfect solutions of the 3-bit parity and Breast Cancer problems using a single GSN. Furthermore, using the GSN avoids the problem of having to choose an arbitrary subset of excitatory inputs, a problem faced when reducing the size of SIANN structure. This was one of the prime motivating factors in the development of the GSN model.

In conclusion, GFNNs show the ability to form the basis of a class of powerful new classifiers. Further investigations needed to compare their performance with other types of networks, particularly on more complex problems. The next step would therefore be to compare the performance of GFNNs to that of MLPs and SIANNs for a variety of problems.

Chapter 8

Extended Benchmark Tests

8.1 Introduction

In the previous chapters, the performances of 'standard' SIANNs, 'enhanced' SIANNs and GFNNs were compared on a set of benchmarks problems. This comparison between various shunting inhibition based networks begs the question "How do shunting inhibition based networks compare with other types of networks?" In this chapter, the issue is addressed by comparing the shunting inhibition-based networks with what is arguably the most commonly used artificial neural network for these types of problems, the Multilayer Perceptron (MLP). Moreover, whenever possible, comparisons are also made with other results presented in the literature.

The GFNN architecture contains MLPs as a subset. The code used for the simulations has been written in a manner that allows the same code to be used for generating and training SIANNs, GFNNs and MLPs. For each of the five benchmark problems used in previous chapters, MLP structures with similar number of weights as the GFNN structures were generated, trained and tested. The objective was to investigate differences due to the types of network architectures; therefore, the same code was used for initialisation and training, with all parameters being the same. An additional benchmark problem has been considered, namely the Thyroid disease classification problem, in order to provide an insight into the capabilities of shunting networks with larger problems.

It should be noted that the MATLAB code developed so far for training SIANNs, GFNNs and MLP has not been optimised for speed of execution. Code optimisation would have significant impact on the time taken to train the networks. In order to provide a reference for comparison with other experimental results, 'MATLAB Toolbox MLPs' (MT-MLPs), generated and trained using the MATLAB Neural Network Toolbox, have also been trained on the same set of benchmark problems. These 'MT-MLPs' differ from the earlier MLPs, which we hereafter refer to as G-MLPs, only in the method of initialisation and the efficiency of the training algorithms, including the method of representing and storing information in memory.

The next section presents the results obtained by training MLPs on the benchmark problems and compares them to earlier results obtained using GFNNs and SIANNs, as well as making comparisons with other published results. The third section presents a comparison of the MLP results obtained using G-MLPs with those obtained using the MATLAB Neural Network Toolbox code. A discussion is presented in Section 8.4, followed by the conclusion in Section 8.5.

8.2 Test Results and Comparison

The next five sub-sections present results of MLPs trained and tested on the five benchmark problems used in the previous chapters and compares them to results obtained with SIANNs and GFNNs. For each benchmark problem, one of the two GFNN networks presented in Chapter 7 was chosen for comparison with an MLP having similar number of weights. As in the previous chapters, 50 networks were generated, and these were trained using GDX (Gradient Descent with momentum and adaptive learning rate) and LM (Levenberg-Marquardt) algorithms. Additionally, both the GFNN and MLP were trained on the QNN variant that achieved the best performance using SIANNs for the particular benchmark problem. The DS-GDX algorithm was not compared, as it can't be used for single layer networks. The initialisation and training parameters used are the same as described in Chapter 4.

As mentioned in the introduction, the MLPs were generated, initialised, trained and tested using the same MATLAB code used to train SIANNs and GFNNs. The objective is to vary only the type of neuron used, so that a fair comparison can be made on the relative effectiveness of the two architectures. This is also the reason why the tested MLP structure was not one with the same number of neurons as the GFNN structure, but instead one that had similar number of synaptic weights. This would give the MLP network the same 'capacity' to learn as the GFNN network, making the comparison fairer. It would also make the sizes of the gradient vectors and Hessian matrices comparable, making the time comparisons fairer as well. The results for the 'Standard' SIANNs are presented for comparison.

An additional 'real world' problem, the Thyroid disease classification problem, has been added to the set of benchmark tests to compare the different network types. Appropriate GFNN, SIANN and MLP networks were trained and tested on this problem. The description of this dataset and the test results are presented in Subsection 8.2.6.

Comparisons with results for the benchmark tests from the literature have also been carried out where available and appropriate.

8.2.1 Wisconsin Breast Cancer Dataset

The results obtained using a 9-2-1 MLP, a single generalised shunting neuron (GSN) and the 'standard' 9-9-1 SIANN, trained on the Wisconsin Breast Cancer problem, are presented in Table 8.1 and Fig. 8.1. As in previous chapters, the figure contains two graphs: the top graph shows the mean and median test error rates for the best performing activation function combination, and the bottom graph illustrates the corresponding mean training times. Note that the median test error is often zero, and hence is not visible on the graph.

The 9-2-1 MLP network trained using the GDX algorithm had the lowest average error (0.08%) with 86% of networks able to achieve perfect classification, i.e., 0.00% error. This network structure was also the second fastest to train, next to the GFNN trained with GDX. The MLP trained using the LM algorithm, on the other hand, had the highest error rate, with a mean error of 0.51%. When trained with the QNN3 algorithm, the MLP achieved an 'average' error of 0.24%.

The MLP took marginally longer to train than the GFNN for both the GDX and LM algorithms. The GFNN was the slowest to train with QNN. The mean training times for the LM and GDX algorithms are quite similar, with the exception of the SIANN trained with LM.

Overall, the first-order GDX algorithm surprisingly produced the best set of results, both in terms of accuracy and speed of training. The LM algorithm had the highest average error rates, except for the SIANN architecture. The MLP results were the extremes – the best of the best (GDX), worst of the worst (LM), and middle with the QNN algorithm!

Here it has to be highlighted once again that, in reality, the average error rates achieved were not very different. All cases achieved error rates of less than 0.5%, in other words, mean accuracy of 99.5% or better, and over half the networks achieved perfect classification.

Table 8.1 Results for Wisconsin Breast Cancer dataset using MLP, GFNNs and SIANNs

Network Structure	Training Algor.	Activation functions			Performance (% of runs)			Avg Epochs			Test Error				Mean time to train (s)
		Shunting		Out	→ goal	0% err	20% <	All runs	→ goal	0% Error	Best (%)	Mean (%)	95% CI	Med. (%)	
		Nu	Den												
MLP 9-2-1 (23 weights)	GDX	Lgs		Lin	0	86	100	158	*	159	0.00	0.08	± 0.05	0.00	8.6
	LM	Lgs		Tnh	0	44	100	57	*	56	0.00	0.51	± 0.15	0.56	9.9
	QNN3	Lgs		Tnh	0	70	100	80	*	67	0.00	0.24	± 0.12	0.00	13.2
Single GSN 9-1 (22 weights)	GDX	Lin	Lgs		0	84	100	134	*	141	0.00	0.16	± 0.11	0.00	7.0
	LM	Lin	Lgs		0	22	100	56	*	54	0.00	0.44	± 0.07	0.56	9.4
	QNN3	Lin	Lgs		0	68	100	194	*	204	0.00	0.19	± 0.08	0.00	30.9
SIANN 9-9-1 (118 weights)	GDX		Lgs	Lgs	0	66	100	161	*	160	0.00	0.20	± 0.08	0.00	10.3
	LM		Lgs	Tnh	0	74	100	181	*	119	0.00	0.20	± 0.10	0.00	34.8
	QNN3		Tnh	Lgs	0	56	100	78	*	70	0.00	0.33	± 0.12	0.00	17.4

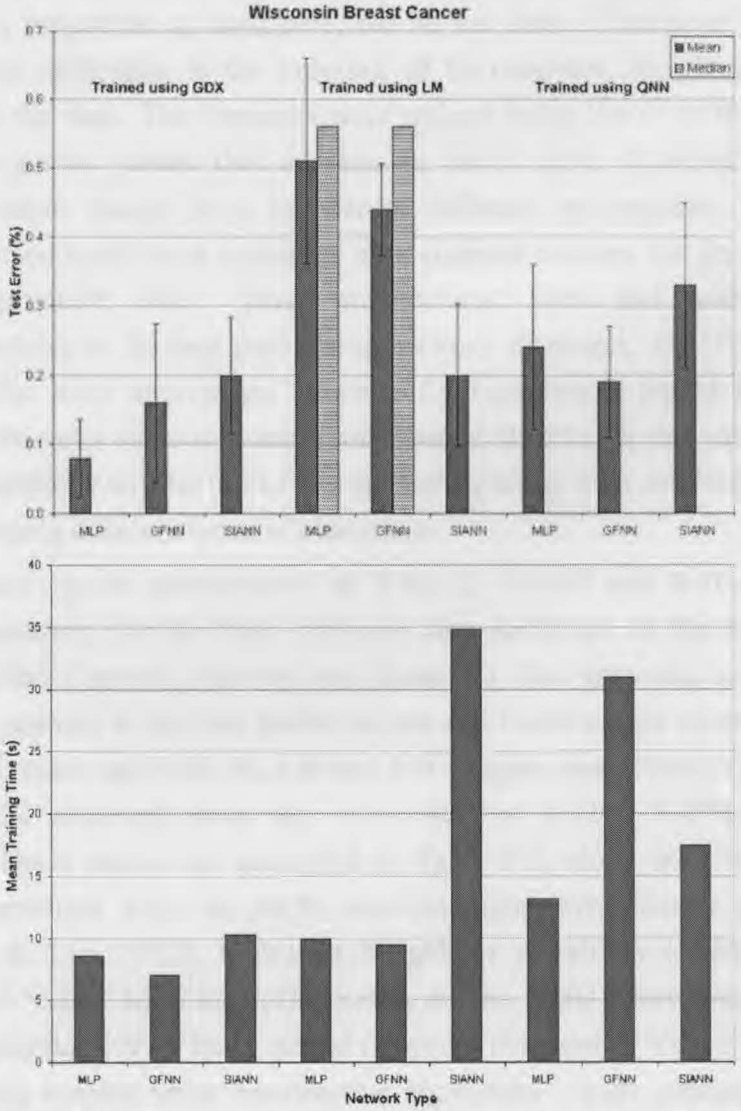


Fig. 8.1: Mean and median test error and mean training time for the Wisconsin Breast Cancer dataset using MLPs, GFNNs and SIANNs.

To get an idea of what these benchmark test results mean in the broader context of pattern classification, comparisons have to be made with other results published in the literature. The results obtained by Prechelt, with the *Proben1* set of benchmark problems (Prechelt, 1994), provide a good reference for comparison for a number of reasons. Firstly, the results are well documented with good descriptions of the datasets, architectures and training parameters, as this study attempts to set the standard for benchmark testing and reporting. Secondly, the *Proben1* set contains three out of the six datasets used here, namely the Wisconsin Breast Cancer, Pima Indians Diabetes and Thyroid datasets. The third reason is that the guidelines and 'standards' laid out by Prechelt have been followed fairly closely in the benchmark tests carried out in this investigation, which allows meaningful comparisons to be made. Finally, the *Proben1* datasets and the results given by Prechelt are referenced fairly frequently, forming a common reference point for comparison.

Prechelt divides the datasets into training, validation and test sets using the same 50%-25%-25% proportion as used here, but he has three versions of each dataset, where the only difference is the ordering of the samples, resulting in different partitioning of the data. The networks were trained using the RPROP algorithm, a fast backpropagation variant that operates in batch mode (Riedmiller & Braun, 1993). He presents results for a number of different architectures: purely linear networks; selected multi-layer structures with sigmoid neurons for finding the 'best performing' structure; and 'pivot architectures' with and without shortcut connections (relates to the best performing network structures, see (Prechelt, 1994) for details). The most appropriate structure for comparison would be the 'pivot architecture' networks without shortcut connections. Results for this structure trained on all three partitions will be used for comparison, along with selected results from other sources using different types of classifiers.

Prechelt used 'pivot architectures' of 9-4-2-2, 9-8-4-2 and 9-16-8-2, with no shortcut connections, for the three different data partitions of the Breast Cancer problem (labelled *Cancer1*, *Cancer2* and *Cancer3*). The networks used are MLPs with sigmoid neurons in the two hidden layers and linear output neurons, with one output for each class, and have 56, 126 and 314 weights, respectively (no shortcuts). These networks achieved mean test error rates of 1.32%, 3.47% and 2.60%, respectively. These results are presented in Table 8.2, along with results from a variety of classifiers such as MLPs evolved using evolutionary programming (*EPNet*)(Yao & Liu, 1997), k-Nearest Neighbour classifiers (*kNN*) (Jankowski, 2003), Support Vector Machines (*SVM*)(Shin & Cho, 2003), test feature classifiers (Lashkia & Aleshin, 2001), fuzzy neural networks (Meesad & Yen, 2001), cascade neural networks created using constructive algorithms - with pruning (*CNNDA – Case I*) and without pruning (*CNNDA – Case II*) (Islam et al., 2000) and neural network committees (*Committee*) (Verikas et al., 2002).

Table 8.2 shows the results from the other literature, with mean error rates ranging from 1.16% to 6.70%. It is fairly obvious that GFNNs and SIANNS, with mean error rates between 0.16% and 0.44%, outperform these other classifiers by a significant margin. What makes it more notable is the fact that the shunting network results have been obtained with very small structures, including a single neuron GFNN.

Table 8.2 Comparison of mean test error for Wisconsin Breast Cancer dataset with results from other literature.

Instance	Source	Mean Test Error (%)
GFNN - GDX	Chap. 7 / 8	0.16
GFNN - LM	Chap. 7 / 8	0.44
GFNN – QNN3	Chap. 7 / 8	0.19
SIANN - GDX	Chap. 4 / 8	0.20
SIANN - LM	Chap. 4 / 8	0.20
SIANN – QNN3	Chap. 4 / 8	0.33
MLP - GDX	Chap. 8	0.08
MLP - LM	Chap. 8	0.51
MLP – QNN3	Chap. 8	0.24

Instance	Source	Mean Test Error (%)
Cancer1	Proben1	1.32
Cancer2	Proben1	3.47
Cancer3	Proben1	2.60
EPNet	Yao	1.38
kNN	Jankowski	2.95
SVM	Shin	6.70
Test feature	Laskia	4.00
Fuzzy NN	Meesad	1.75
CNNDA – Case I	Islam	1.27
CNNDA – Case II	Islam	1.16
Committee – All	Verikas	3.10
Committee – 2 Features	Verikas	2.35

8.2.2 Pima Indians Diabetes Dataset

The results obtained for the Diabetes dataset, using an 8-2-1 MLP, a single generalised shunting neuron (G 8-1) and the 'standard' SIANN (8-8-1), are presented in Table 8.3 and Fig. 8.2. Since the 'best case' error is not zero for this problem, here Fig. 8.2 shows the lowest error rate achieved by a single network in each case, in addition to the mean and median test error rates.

The MLP was able to achieve a mean error rate between 20.45% and 20.75%, which is comparable to the results obtained using the GFNN. Compared on the basis of training algorithm, the MLP achieved the best result for the GDX algorithm, followed by the GFNN and SIANN, whereas for the LM and QNN algorithms the order was reversed, with SIANN doing best and MLP worst. The SIANN was the only network structure here able to achieve average error rates below 20%, (19.88% with LM and 19.80% with QNN3 algorithm). An error rate below 20.0% is considered very good for the Diabetes dataset, as most test results tend to be above this level (Michie et al., 1994; Prechelt, 1994). It should be noted that the GP 8-2-1 GFNN trained with DS-GDX was able to achieve 19.94%, see Table 7.2. The best average error of all was with a SIANN trained using the first-order GDM algorithm – a remarkable 19.05% (see Table 4.3).

Table 8.3 Results for Pima Indians Diabetes dataset using MLP, GFNNs and SIANNs

Network Structure	Training Algor.	Activation functions			Performance (% of runs)			Avg Epochs			Test Error				Mean time to train (s)
		Shunting		Out	→ goal	0% err	20% <	All runs	→ goal	0% Error	Best (%)	Mean (%)	95% CI	Med. (%)	
		Nu	Den												
MLP 8-2-1 (21 wt.)	GDX	Tnh		Tnh	0	0	30	188	*	*	18.75	20.45	± 0.22	20.31	11.1
	LM	Lgs		Tnh	0	0	26	58	*	*	18.75	20.75	± 0.31	20.83	10.8
	QNN3	Lgs		Lin	0	0	34	144	*	*	18.23	20.48	± 0.28	20.31	26.2
Single GSN 8-1 (20 wt.)	GDX	Tnh	Exp		0	0	38	173	*	*	18.23	20.58	± 0.26	20.83	9.7
	LM	Tnh	Lgs		0	0	34	22	*	*	18.75	20.56	± 0.27	20.31	5.2
	QNN3	Tnh	Lgs		0	0	46	94	*	*	18.23	20.05	± 0.24	20.31	17.0
SIANN 8-8-1 (97 weights)	GDX		Tnh	Lgs	0	0	14	195	*	*	17.71	21.03	± 0.29	21.35	13.2
	LM		Lgs	Tnh	0	0	58	182	*	*	17.71	19.88	± 0.32	19.79	38.9
	QNN3		Exp	Tnh	0	0	60	182	*	*	17.71	19.80	± 0.27	19.79	44.6

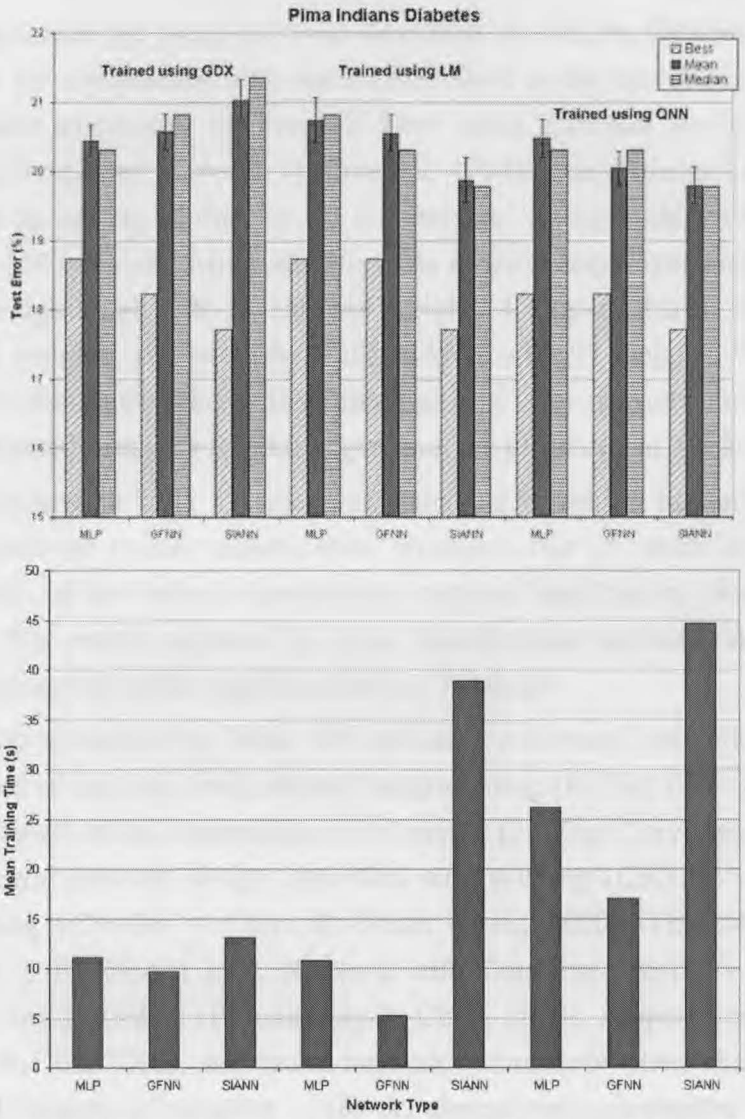


Fig. 8.2: Best, mean and median test error and mean training time for the Diabetes dataset using MLPs, GFNNs and SIANNs.

The SIANN has the advantage of more synaptic weights, 97 versus 21 and 20 weights for the MLP and GFNN structures, respectively, hence providing it with a larger 'learning capacity'. On the other hand, the increased network size has the disadvantage of more complex computation, as reflected by the longest training times for SIANNs, particularly for the second-order LM and QNN algorithms. One notable fact is that the single GSN was the fastest to train for all three algorithms. Furthermore, it achieved better results than the MLP when trained using the second-order algorithms, LM and QNN. For the first-order GDX algorithm, the MLP achieved the best average error rate, but its best case error is still worse than that of the single GSN. Finally, it should be point out that the lowest error achieved by a single GFNN was 16.15%, a GP 8-2-1 network trained using LM, while the best single network overall was the standard 8-8-1 SIANN trained using the QNN6 algorithm which achieved an outstanding 15.63% (see Tables 7.2 and 5.6 respectively).

Table 8.4 presents the mean test error rates from the MLPs, GFNNs and SIANNs obtained here for comparison with results published in the literature. The *Proben1* benchmark tests conducted by Prechelt have three different partitions (labelled *diabetes1*, *diabetes2* and *diabetes 3*) (Prechelt, 1994). The *diabetes1* and *diabetes3* partitions end up having as their 'pivot architecture' a single hidden-layer network (8-32-2, with 354 weights), while *diabetes2* has a two hidden-layer network (8-16-8-2, with 298 weights), all with no shortcut weights. Compare this to the GSN used, which has 20 weights, and even the 'full' SIANN with 97 weights. The mean test error of the *Proben1* networks, with sigmoid hidden layer neurons and linear output neurons and trained using the RPROP algorithm, are presented in Table 8.4.

The *Statlog* project tests a variety of statistical, machine learning and neural network methods on twenty classification problems, one of which is the Diabetes problem; details of the various classification methods used can be found in (Michie et al., 1994). The results achieved by these classification methods on the Diabetes problem are presented on the right hand side of Table 8.4.

Other results presented in Table 8.4 include feedforward networks constructed using a number of methods: evolutionary programming (*EPNet*) (Yao & Liu, 1997); cascade correlation-based construction with weight pruning (Thivierge et al., 2003); correlation neural network design algorithm with pruning (*CNNDA – Case I*) and without pruning (*CNNDA – Case II*) (Islam et al., 2000). The other compared classifiers are a Functional Link Network with Gaussian functions trained using Genetic Learning (*GLFLN*) (Bhumireddy & Chen, 2003), support vector machines (*SVM*) (Shin & Cho, 2003), and neural network committees where the members are trained on all features (*Committee – All*) or, alternatively, committee members are trained on selected features (*Committee – 2 features*) (Verikas et al., 2002).

The GFNN, SIANN and MLP network results presented here range from 19.1% to 21.0% error, which are better than all the results reported in *Statlog* and *Proben1*, and most of the others as well. The only instances from other sources with mean error rates below 21% are *CNNDA Case II*, *GLFLN* and *Committee – 2 features*. The *CNNDA Case II* is the only instance with an average error below 20%; it has an average error rate (19.9%), which is comparable to that of SIANN trained with LM (19.88%) and QNN3 (19.80%), but not as good as the SIANN trained with QNN9 (19.57%) and GDM (19.05%). This means that the results achieved by the shunting inhibition based networks are better than most of the other surveyed classification methods, including many that use networks that are far larger in terms of number of neurons and weights.

Table 8.4 Comparison of mean test error for Pima Indians Diabetes dataset with results from other literature

Instance	Source	Mean Test Error (%)	Instance	Source	Mean Test Error (%)
GFNN - GDX	Chap. 7 / 8	20.6	Logdisc	Statlog	22.3
GFNN - LM	Chap. 7 / 8	20.6	DIPOL92	Statlog	22.4
GFNN – QNN3	Chap. 7 / 8	20.1	Discrim	Statlog	22.5
SIANN - GDX	Chap. 4 / 8	21.0	SMART	Statlog	23.2
SIANN - LM	Chap. 4 / 8	19.9	RBF	Statlog	24.3
SIANN – QNN3	Chap. 4 / 8	19.8	ITrule	Statlog	24.5
MLP - GDX	Chap. 8	20.5	Backprop	Statlog	24.8
MLP - LM	Chap. 8	20.8	Cal5	Statlog	25.0
MLP – QNN3	Chap. 8	20.5	CART	Statlog	25.5
GFNN GP 8-2-1 – DS-GDX	Chap. 7	19.9	CASTLE	Statlog	25.8
SIANN – QNN9	Chap. 5	19.6	Quadisc	Statlog	26.2
SIANN - GDM	Chap. 4	19.1	NaiveBay	Statlog	26.2
Diabetes1	Proben1	24.1	C4.5	Statlog	27.0
Diabetes2	Proben1	26.4	IndCART	Statlog	27.1
Diabetes3	Proben1	22.6	Baytree	Statlog	27.1
EPNet	Yao	22.4	LVQ	Statlog	27.2
Cascade Correlation	Thivierge	21.3	Kohonen	Statlog	27.3
CNNDA – Case I	Islam	22.3	AC^2	Statlog	27.6
CNNDA – Case II	Islam	19.9	NewID	Statlog	28.9
GLFLN	Bhumireddy	20.3	CN2	Statlog	28.9
SVM	Shin	29.9	ALLOC80	Statlog	30.1
Committee – All	Verikas	21.7	k-NN	Statlog	32.4
Committee – 2 Features	Verikas	20.8			

8.2.3 The 3-bit Parity Problem

The results for the parity problem, presented in Table 8.5 and Fig. 8.3, are obtained with a 3-3-1 MLP, a GP 3-2-1 GFNN and the 3-3-1 'standard' SIANN. It should be noted that in all cases the median error rate is zero, and so is the mean in two cases; these are, therefore, not visible on the graph.

The 3-3-1 MLP trained with LM was able to achieve 'perfect' results – 100% correct for all networks – just like the GFNN network, but in a shorter time. The MLP achieved the best result obtained using the GDX algorithm, with a mean error rate of only 0.5%. However when trained using QNN, the MLP did not perform as well, with the second highest overall error rate of 3.0%. Overall, the MLP always outperformed the SIANN in terms of accuracy, but had mixed results compared to the GFNN. The MLP was, however, the fastest to train for all the three algorithms tested.

This is a fairly simple problem for the neural networks, with all the network types achieving 100% correct classification with more than three quarters of the trained networks. In this case, the simplicity of the MLP neuron structure has resulted in faster training times while still achieving similar accuracy compared to the other network types.

Comparison with other literature has not been made for this problem for a couple of reasons. Firstly, most of the literature where parity-type problems have been used refer to the simple XOR, or 2-bit parity, problem, while others jump to the more complex 5-bit or higher parity cases. Secondly, even in cases where the 3-bit parity problem has been used, the results are generally not in a form that allows any meaningful comparisons to be made. For example, some results are in the form of the number of epochs or number of operations required to achieve a particular error goal, and in most cases the error goal is different to that used in the tests performed in this work.

8.2.4 Artificial Multi-class Problem

Since this problem has two input features and three classes, all trained networks had two inputs and three output neurons: the GFNN was a 2-2-3 structure, with two GSNs in the hidden layer and three perceptron-type output neurons; the MLP was a 2-4-3 structure, with four hidden neurons; and the 'standard' SIANN had a 2-2-3 structure. Among these neuron network structures, the MLP has the most synaptic weights with 27, the SIANN has the least number of weights for once with 19, and the GFNN has 25 weights.

Table 8.5 Best results for 3-bit Parity dataset using MLP, GFNNs and SIANNs

Network Structure	Training Algor.	Activation functions			Performance (% of runs)			Avg Epochs			Test Error				Mean time to train (s)
		Shunting		Out	→ goal	0% err	20% <	All runs	→ goal	0% Error	Best (%)	Mean (%)	95% CI	Med. (%)	
		Nu	Den												
GMLP 3-3-1 (16 wt.)	GDX	Lgs		Lgs	96	96	100	224	192	192	0.00	0.50	± 0.69	0.00	1.0
	LM	Lgs		Lin	100	100	100	11	11	11	0.00	0.00	± 0.00	0.00	0.2
	QNN3	Lgs		Lgs	78	78	98	333	145	145	0.00	3.00	± 1.65	0.00	5.4
GFNN GP 3-2-1 (23 weights)	GDX	Lin	Exp	Lgs	80	80	100	454	318	318	0.00	2.50	± 1.40	0.00	3.2
	LM	Lin	Lgs	Lin	98	100	100	30	22	30	0.00	0.00	± 0.00	0.00	0.8
	QNN3	Exp	Tnh	Lin	84	84	100	283	160	160	0.00	2.00	± 1.28	0.00	7.6
SIANN 3-3-1 (22 weights)	GDX		Tnh	Lgs	94	94	96	352	311	311	0.00	1.25	± 1.44	0.00	2.3
	LM		Exp	Lin	96	96	100	84	46	46	0.00	0.50	± 0.69	0.00	1.7
	QNN3		Lgs	Lgs	72	80	88	436	216	295	0.00	4.75	± 3.04	0.00	9.4

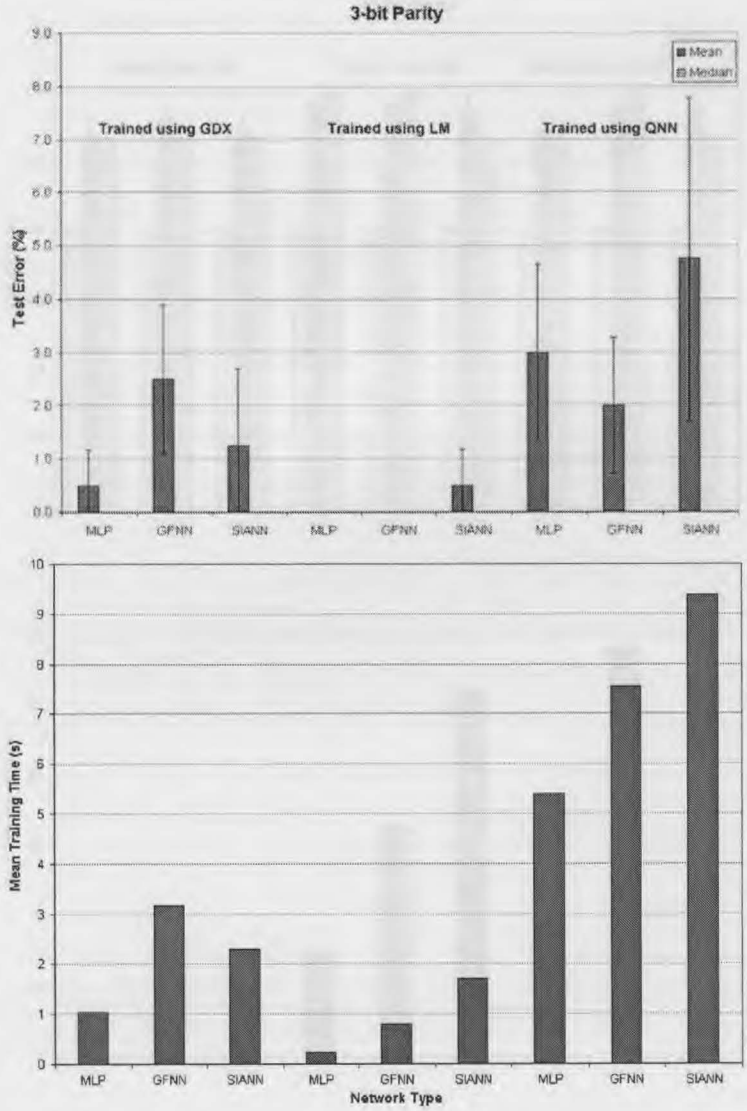


Fig. 8.3: Mean and median test error and mean training time for 3-bit parity dataset using MLPs, GFNNs and SIANNs.

Table 8.6 Results for Multi-Class dataset using MLP, GFNNs and SIANNs

Network Structure	Training Algor.	Activation functions			Performance (% of runs)			Avg Epochs			Test Error				Mean time to train (s)
		Shunting		Out	→ goal	0% err	20% <	All runs	→ goal	0% Error	Best (%)	Mean (%)	95% CI	Med. (%)	
		Nu	Den												
MLP 2-4-3 (27 weights)	GDX	Lgs		Lgs	0	0	100	225	*	*	4.00	5.37	± 0.22	5.33	11.5
	LM	Tnh		Tnh	0	0	100	67	*	*	4.00	5.83	± 0.16	6.00	29.9
	QNN6	Lgs		Lgs	0	0	100	119	*	*	4.00	5.43	± 0.22	5.33	39.5
GFNN GP 2-2-3 (25 weights)	GDX	Lin	Lgs	Lgs	0	0	100	180	*	*	4.00	5.61	± 0.17	5.33	13.2
	LM	Lin	Lgs	Tnh	0	0	100	99	*	*	4.00	5.79	± 0.17	6.00	65.4
	QNN6	Lin	Tnh	Lgs	0	0	100	232	*	*	4.00	5.76	± 0.24	6.00	116.5
SIANN 2-2-3 (19 weights)	GDX		Exp	Lgs	0	0	100	377	*	*	4.00	5.47	± 0.16	5.33	19.4
	LM		Exp	Lgs	0	0	100	228	*	*	4.00	5.69	± 0.26	5.33	103.0
	QNN6		Exp	Lgs	0	0	100	158	*	*	4.00	5.72	± 0.21	5.33	55.4

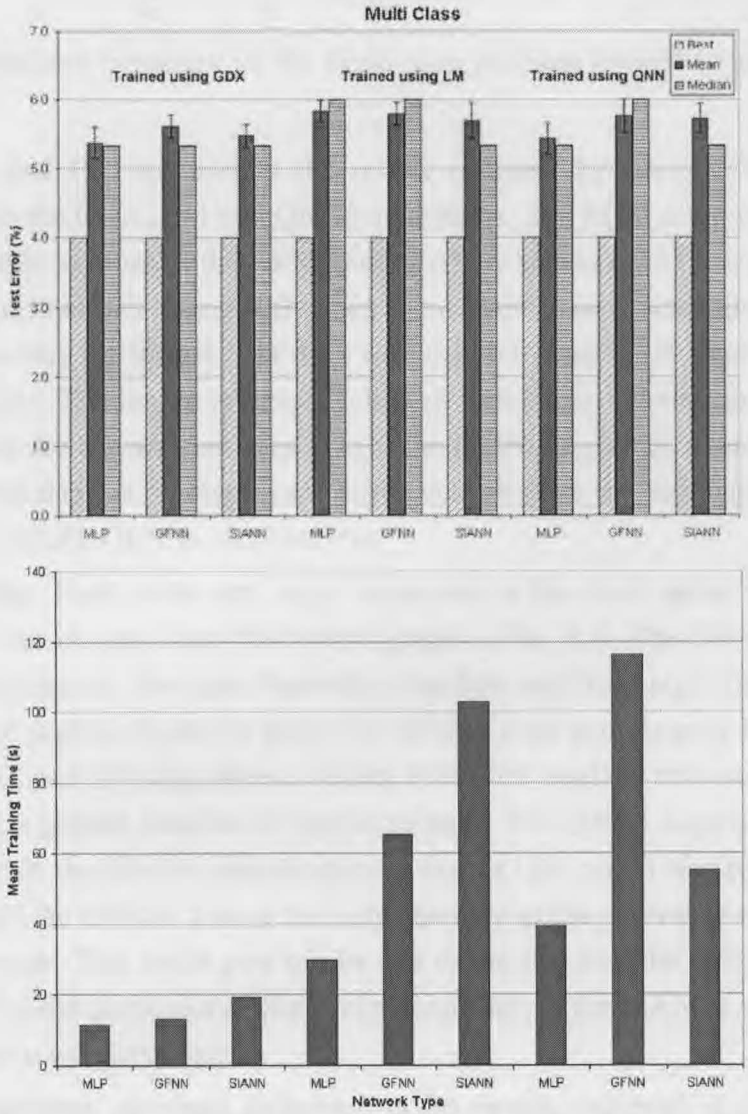


Fig. 8.4: Best, mean, median test error and mean training time for Multi-class data using MLPs, GFNNs and SIANNs.

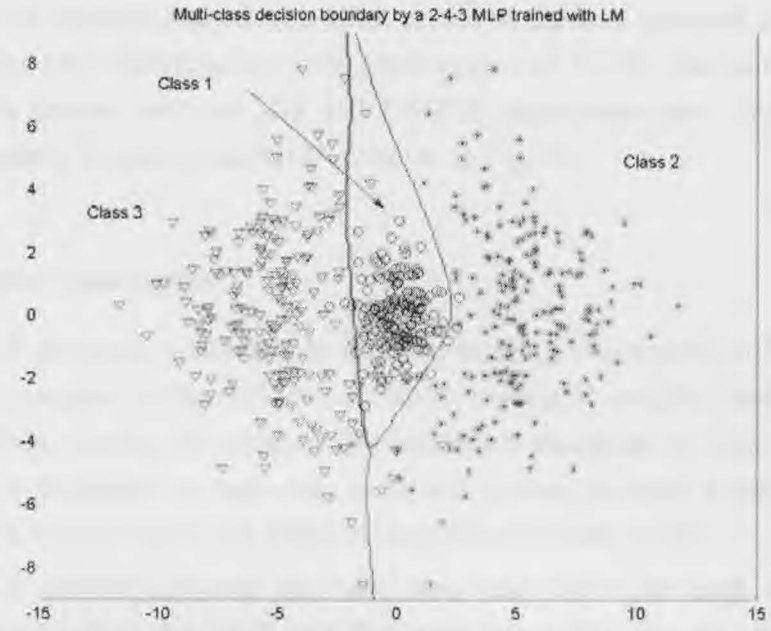


Fig. 8.5: Decision boundary for the Multi-class problem formed by an MLP.

Table 8.6 and Fig. 8.4 present the results obtained by training these network structures with the GDX, LM and QNN6 algorithms. The MLP achieved mean error rates comparable to those of the GFNN and SIANN, between 5.37% and 5.83%. For any given algorithm, the largest difference in mean error rates was only 0.33%, with the MLP achieving the lowest error rates with the GDX and QNN algorithms and the highest with LM. The lowest error rate achieved by a single network in all cases was 4.00%, and the median was between 5.33% and 6.00% across the board. Essentially, this means that there is no significant difference between the accuracy of the three architectures: MLPs, GFNNs and SIANNs.

On the other hand, there are large variations in the time taken to train these networks, as can be seen from the bottom graph in Fig. 8.4. The GDX algorithm is approximately three to five times faster than the LM and QNN algorithms, and in all cases the MLP was the fastest to train. The SIANN took significantly longer to train using the GDX and LM algorithms, despite being the smallest network in this case, as it required a greater number of epochs to train. The QNN6 algorithm, however, was able to train the SIANN almost twice as fast as LM, and it was twice as fast as when it trained the GFNN. This is the only anomaly in the otherwise regular pattern in the time graph. This could possibly be due to the fact that the QNN algorithm is able to factor in the constraint on the decay parameter, a, for SIANNs while working out the optimum weight update.

For this problem, the main difference in the results achieved is in the training time. The MLP has the simplest neuron structure, and hence it is the simplest to train, giving it the edge in performance. There are no results in other literature available for comparison for this artificially generated dataset. It should be noted that the

Expanded 2-3-3 SIANN trained with GDX achieved the best reported mean error of 5.00%, and the best individual network performance of 3.33% was achieved by the same network trained with the LM and DSGDX algorithms (see Table 6.4). The decision boundary formed by an MLP is shown in Fig. 8.5.

8.2.5 Sunspot Time Series

A 10-4-1 MLP structure, which has 49 synaptic weights, was trained on the Sunspots problem and compared to the GP 10-2-1 GFNN, having 51 weights, and the 10-10-1 standard SIANN, having 141 weights. The results are presented in Table 8.7 and Fig. 8.6. Figure 8.6 illustrates the best-case, mean and median Average Relative Variance (ARV), Eq. (4.30), as well as the mean training time (bottom graph).

The 10-2-1 GFNN achieved the best mean test ARV for each of the three algorithms used, while the MLP had the worst test ARV for the LM and QNN algorithms and second best for GDX. The difference between the mean test ARV of the MLP and GFNN ranges from approximately 10% (when trained with GDX) to 50% (when trained with QNN). This shows that the GFNN is able to perform significantly better compared to the MLP, despite the fact that the MLP has similar number of weights and more neurons. The test ARV achieved by the standard SIANN was also better than that of the MLP when trained with the LM and QNN algorithms, but worse when using the GDX algorithm. In terms of training time, the MLP trained the fastest with GDX and LM, SIANN next and GFNN the slowest; with the QNN algorithm, GFNN trained the fastest and SIANN the slowest.

Comparing the performances of the training algorithms, the GDX had the worst accuracy, with QNN best, slightly better than LM. The training time trend was the other way around, with GDX fastest, LM slightly slower and QNN taking 2 to 6 times longer. This is to be expected, as the 'price' for the improved accuracy is the longer training time – the 'no free lunch' concept. The amount of additional time required by the QNN algorithm for the marginal improvement in accuracy, however, makes it seem 'expensive', though that is a subjective conclusion.

Overall the GFNN trained with QNN was the most accurate, both in terms of test ARV and test MSE. The GFNN trained with LM was only slightly less accurate, but trained in less than half the time. The GP 10-2-1 GFNN clearly outperformed the MLP in terms of accuracy. Referring back to Chapter 7, we see that a single generalised shunting neuron, with half the number of weights, is able to match the performance of the MLP network. The single GSN achieved mean test ARV of 0.120 and 0.140 with the GDX and LM algorithms, respectively. The best performing neuron has a test ARV of only 0.085 (see Table 7.5). This reinforces the fact that GFNNs are able to achieve good results with extremely simple structures.

Table 8.7 Results for Sunspots dataset using MLP, GFNNs and SIANNs

Network Struct.	Training Algor.	Act-fns			Performance (% of runs)			Avg Epochs	Test MSE		Test ARV				Mean time to train (s)
		Shunting		Out	→ gl	all in tol	80% tol		Best	Median	Best	Median	Mean	95% CI	
		Nu	Den												
MLP 10-4-1 (49 wt.)	GDX	Tnh		Lin	0	14	98	151	0.0094	0.0129	0.113	0.155	0.162	± 0.024	5.8
	LM	Lgs		Lin	0	22	100	57	0.0079	0.0113	0.095	0.136	0.138	± 0.006	7.0
	QNN6	Tnh		Lin	0	24	100	279	0.0071	0.0115	0.085	0.138	0.139	± 0.007	25.3
GFNN GP 10-2-1 (51 wt.)	GDX	Lin	Lgs	Lin	0	6	100	146	0.0095	0.0122	0.113	0.147	0.146	± 0.004	8.0
	LM	Lin	Lgs	Lin	0	8	100	61	0.0065	0.0083	0.078	0.100	0.100	± 0.004	10.9
	QNN6	Lin	Lgs	Lgs	0	48	100	170	0.0063	0.0076	0.075	0.091	0.092	± 0.003	23.4
SIANN 10-10-1 (141 weights)	GDX		Tnh	Lin	0	14	98	147	0.0085	0.0134	0.102	0.161	0.174	± 0.020	6.8
	LM		Lgs	Lin	0	8	100	54	0.0075	0.0093	0.090	0.111	0.112	± 0.003	9.8
	QNN6		Lgs	Lgs	0	80	100	337	0.0054	0.0077	0.065	0.093	0.100	± 0.009	45.8

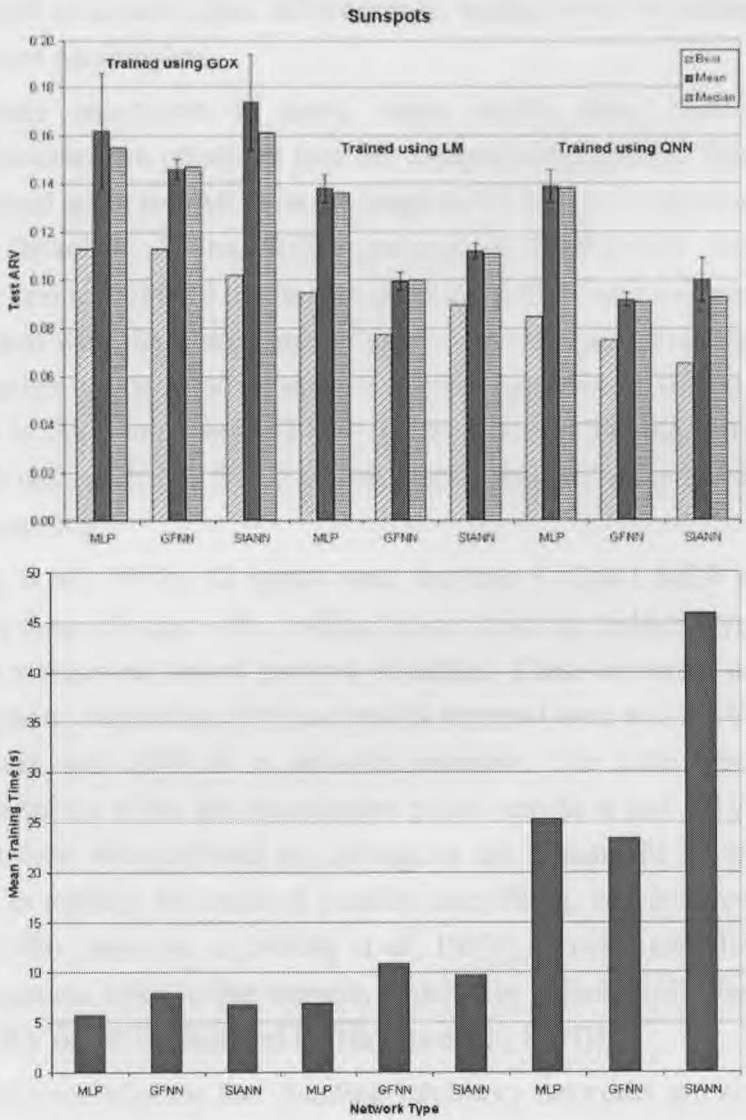


Fig. 8.6: Best, mean and median test ARV and mean training time for Sunspots data using MLPs, GFNNs and SIANNs.

The experiments on the sunspots time series prediction problem reported in the literature have been carried out using various parameters and measures. For example, some use a different number of inputs to what has been used here (Lawrence et al., 1996; Naftaly et al., 1997; Park et al., 1996; Weigend et al., 1990). Some do not use a complete set of consecutive previous time samples as inputs, but instead use a selected subset of non-consecutive samples points, based on previous analysis of the data (Naftaly et al., 1997). This makes any comparison questionable as the networks are being given different information on which to make the prediction.

The task is made even more difficult by the fact that the performance measures differ, unlike for classification tasks where the test error rate or success rate is used in most cases. For time-series prediction, criteria other than the test ARV are often used, such as the mean squared error (MSE) (Park et al., 1996). However, the MSE is not a normalised parameter; thus, differences in scaling prior to training can render this measurement meaningless.

Bearing these constraints in mind, some results using test ARV as the performance measure are presented here for comparison purposes. The GFNNs and SIANNs achieved mean test ARVs in the range 0.092 to 0.174, with best case as low as 0.065. In (Nikolaev & Iba, 2003), polynomial feedforward neural networks (PFNNs) were trained with 10 inputs, the same number as used to train the SIANNs and GFNNs, and with the same range of points for training, validation and testing. The generalization or test ARV reported by them ranged from 0.077 to 0.442, which is comparable to the results obtained here. It should noted that the better performing networks in (Nikolaev & Iba, 2003) all had their 'optimum' structure determined by genetic programming.

In (Naftaly et al., 1997), 12 inputs were fed into a 12-4-1 MLP structure. The networks were then enlarged with feedback loops from the hidden layer to the input layer to form a recurrent neural network structure. These recurrent networks were tested singly and as ensembles. The best results reported were test ARVs of 0.073 for a single network and 0.070 for a network ensemble. The same type of networks trained using a subset of six non-consecutive points results in test ARV of 0.070 and 0.067, respectively. Weigand and his colleagues use a standard 12-8-1 MLP, with weight decay to address the issue of possible overfitting, and achieved a best case test ARV of 0.086 (reported in (Naftaly et al., 1997)). Nowlan and Hinton impose a mixture of Gaussians prior on the weights, which they called "Soft Weight Sharing", to get a test ARV of 0.072 (reported in (Naftaly et al., 1997)).

The general conclusion is that shunting inhibitory networks are able to achieve performance levels comparable to the other results reported here, with the best shunting network achieving the lowest test ARV of 0.065.

8.2.6 Thyroid Disease Dataset

In order to provide an insight into the capabilities of shunting networks with larger problems, an additional benchmark test has also been included in this chapter. The thyroid problem chosen has more than seven thousand samples with 21 input parameters and three output classes. The thyroid dataset was chosen as it is a "hard practical classification task" that could provide a good test for the algorithms and networks being evaluated (Schiffmann et al., 1992a).

8.2.6.1 Description of the Thyroid dataset

The Thyroid disease dataset is another real-world medical diagnosis dataset obtained from the UCI Machine Learning Repository (Blake & Merz, 1998). The repository has a number of datasets pertaining to the Thyroid disease and the dataset chosen is the "ANN" version, deemed the most amenable to artificial neural networks. It is in a form that can be used for neural networks without need for pre-processing and has been used fairly commonly in the literature (Abe et al., 1999; Jankowski, 2003; Koshiba & Abe, 2003; Prechelt, 1994; Schiffmann et al., 1992a, 1992b, 1993; Tsujinishi & Abe, 2003; Yao & Liu, 1997). The thyroid dataset has 21 attributes, of which 15 are binary and 6 continuous real-valued inputs, and three output classes. The problem is to determine the patient's thyroid function based on the input attributes, with the three output classes being normal, hyper-functional, and subnormal. The class probabilities for the test set are 92.6%, 5.1% and 2.3% respectively. The normal patients make up the vast majority of cases, therefore a good classifier needs to have success rate much higher than 92.6% (Schiffmann et al., 1992a), i.e., an error rate significantly lower than 7.4%.

The dataset is divided into a training set containing 3772 samples, and test data with 3428 samples. While the whole training set was used to train the networks, the test data was divided in two subsets: one half used as a validation set for early stopping of training, and the other half used as a test set. This is in line with the 50%-25%-25% division of the dataset used for the other problems.

8.2.6.2 Results for the Thyroid problem

The Thyroid problem results presented in Table 8.8 and Fig. 8.7 are for one SIANN, one GFNN and an MLP network that were trained with GDX, LM and QNN6 algorithms. The GFNN structure consists of two generalised shunting neurons and three sigmoid output neurons (GP 21-2-3). We should note that the chosen GFNN structure is small, compared to most structures that have been reported in the literature as 'optimum'. The MLP structure used was a single hidden-layer 21-4-3 MLP, selected because it has almost the same number of weights as the GFNN network, 103 compared to 101 for the GFNN. The SIANN structure used was the

'standard' one, with the number of shunting neurons in the hidden layer being equal to the number of input attributes, i.e., a 21-21-3 structure with a total of 570 weights.

The results presented in Table 8.8 and Fig. 8.7 show that all three types of networks trained with the first-order GDX algorithm have mean error rates of around 6%, with the best result achieved by a single network of 5.60%. This result is not practically useful as the 'default' error rate is 7.4%. However, the results are much better for networks trained with the LM and QNN algorithms.

When trained with the LM algorithm, the MLP achieved a mean error rate of 1.72%, the lowest mean error rate obtained here; the best single MLP network achieved an error rate of 1.17%. The GFNN performance was not as good, with a mean test error of 3.34% and best error rate of 1.98%. The LM-trained SIANN overall performance was poor, with a mean error of 5.40%, but the best case performance was an acceptable 2.16%.

The MLPs trained with the QNN6 algorithm did not perform as well as the LM-trained ones, having a mean test error rate of only 2.18% and best error rate of 1.69%. The GFNN performance was only slightly worse with a mean error rate of 2.62%, but better than when trained with LM. The best single QNN-trained GFNN network achieved a good 1.81% error rate, better than the best SIANN network with 1.87%. The SIANNs trained with QNN had a better mean error rate though, 2.19%, almost the same as that of the MLPs.

An overall comparison of accuracy by algorithm would have QNN better than LM, with GDX the worst by far. This is not surprising as both LM and QNN are second-order algorithms while GDX is first order. From a network 'type' perspective, there is a marked difference when trained with LM: MLP best, GFNN next and SIANN worst. The results are close for all three types trained using the GDX and QNN algorithms, with SIANN slightly worse than the other two for GDX, and GFNN slightly worse for QNN.

The trend for mean training time is that MLP is fastest, GFNN next and SIANN takes the longest to train. The GDX algorithm was the fastest, an order of magnitude faster in most cases, as it is the simplest algorithm. This is negated by the fact that it is unable to produce any useful results with this dataset. The variation in training time across the different types of network was only about 10% with GDX. The LM algorithm took much longer than GDX, as expected, and the QNN even longer in most cases. MLPs took more than three times longer to train with LM compared to GDX, and more than twice as long again with QNN. The GFNNs took 2 to 3 times longer to train than MLPs for these second order algorithms. The SIANNs took the longest to train by far, more than 3 times longer to train than the GFNNs with QNN, and almost 8 times longer than GFNNs with the LM algorithm.

Table 8.8 Results for the Thyroid disease classification dataset using MLPs, GFNNs and SIANNs

Network Structure	Training Algor.	Activation functions			Performance (% of runs)			Avg Epochs			Test Error				Mean CPU time (s)
		Shunting		Out	→ goal	0% err	20% <	All runs	→ goal	0% Error	Best (%)	Mean (%)	95% CI	Med. (%)	
		Nu	Den												
MLP 21-4-3 (103 weights)	GDX	Tnh		Lgs	0	0	100	419	*	*	5.60	5.96	± 0.04	6.01	254.0
	LM	Lgs		Lgs	0	0	100	119	*	*	1.17	1.72	± 0.08	1.69	882.2
	QNN6	Lgs		Lgs	0	0	100	384	*	*	1.69	2.18	± 0.06	2.16	2162.6
GFNN GP 21-2-3 (101 weights)	GDX	Lin	Lgs	Lgs	0	0	100	348	*	*	5.83	6.00	± 0.02	6.01	275.4
	LM	Lin	Tnh	Tnh	0	0	100	300	*	*	1.98	3.34	± 0.27	3.73	2799.7
	QNN6	Lin	Tnh	Lgs	0	0	100	582	*	*	1.81	2.62	± 0.33	2.33	4585.9
SIANN 21-21-3 (570 weights)	GDX		Exp	Lgs	0	0	100	373	*	*	5.66	6.32	± 0.15	6.01	292.5
	LM		Exp	Lgs	0	0	100	703	*	*	2.16	5.40	± 0.30	5.83	36403.2
	QNN6		Tnh	Lin	0	0	100	559	*	*	1.87	2.19	± 0.06	2.16	15844.3

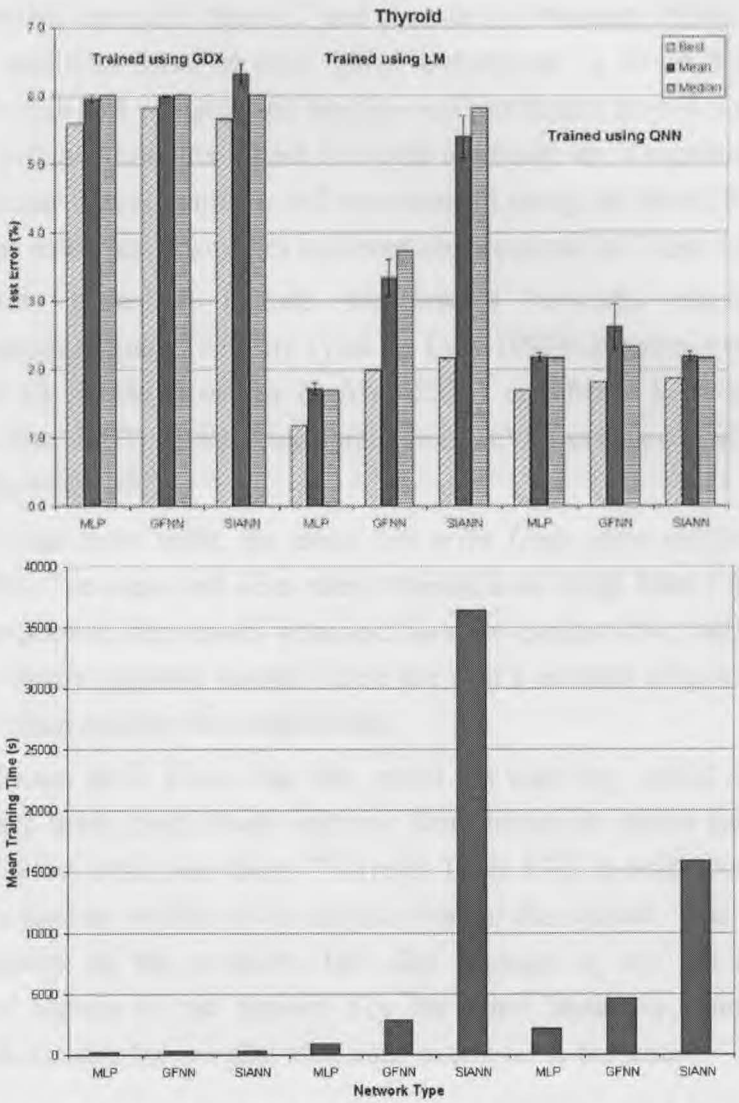


Fig. 8.7: Best, mean and median test error and mean training time for Thyroid dataset using MLPs, GFNNs and SIANNs.

Overall, the LM-trained MLP was best with the lowest error rate, and the shortest training time among the 'useful' networks. The SIANN with LM on the other hand was the worst, with high error rate and the longest training time by far.

Table 8.9 presents the mean test error rate from the MLPs, GFNNs and SIANNs obtained here, along with results from other literature for comparison. The documentation for this dataset, provided by the UCI Repository, refers to work by Schiffmann et al., where the thyroid dataset has been used to evaluate the performance of the backpropagation algorithm and a number of improvements to it (Schiffmann et al., 1992a, 1993), as well as evaluating 'optimal' MLP structures determined by genetic algorithms (Schiffmann et al., 1992b). They use a fully interconnected 21-10-3 MLP, trained by a number of different algorithms. The mean test error rates achieved, given in (Schiffmann et al., 1993), are presented in the right hand side of Table 8.9.

As with the previous benchmark problems, the *Proben1* dataset has three different partitions (labelled *thyroid1*, *thyroid2* and *thyroid 3*) (Prechelt, 1994). The *thyroid1* and *thyroid3* partitions have as their 'pivot architecture' a 21-16-8-3 two-hidden-layer network, with 515 weights, and the *thyroid3* partition a 21-8-4-3 structure, with 227 weights, with no shortcuts. These networks are made up of sigmoid hidden layer neurons and linear output neurons, and were trained using the RPROP algorithm, as previously. The mean test error rates achieved are presented in Table 8.9.

Other results presented include feedforward networks constructed using evolutionary programming (*EPNet*) (Yao & Liu, 1997); support vector machines using L1 and L2 SVMs (Koshiba & Abe, 2003) and fuzzy least squares SVMs (Tsujinishi & Abe, 2003), and *k* nearest neighbour (*kNN*) and weighted kNN (*IWkNN*) methods (Jankowski, 2003).

As can be seen from table, the mean test error from other results ranges from 1.44% to 7.29%. The mean test error rates obtained here range from 1.72% to 6.32%. From that perspective, the results obtained here are comparable, falling within the spread of previously reported results. There are also a number of points to take into consideration when making the comparisons.

Firstly, Schiffmann et al. found that they could not train any useful MLP networks for this problem using batch mode updates. Both instances where batch mode was used resulted in test error rates above 7% (refer Table 8.9), in other words the trained networks were useless, as this is the default error for the dataset. This is not just due to the complexity of the problem, but also because of the extremely uneven distribution of classes in the dataset. For the other instances, they used online training, which updates the weights after each exemplar is presented.

Table 8.9 Comparison of mean test error for Thyroid dataset with other results from literature

Instance	Source	Mean Test Error (%)
GFNN - GDX		6.00
GFNN - LM		3.34
GFNN - QNN		2.62
SIANN - GDX		6.32
SIANN - LM		5.40
SIANN - QNN		2.19
MLP - GDX		5.96
MLP - LM		1.72
MLP - QNN		2.18
Thyroid1	Proben1	2.38
Thyroid2	Proben1	1.91
Thyroid3	Proben1	2.27
EPNet	Yao	2.12
L1 SVM	Koshiba	2.51
L2 SVM	Koshiba	2.65
Fuzzy LS-SVM	Tsujinishi	2.42
kNN	Jankowski	3.70
WkNN	Jankowski	1.44

Instance	Source	Mean Test Error (%)
Backprop	Schiffmann	2.42
Backprop (batch mode)	Schiffmann	7.15
Backprop (batch mode) + Eaton and Oliver	Schiffmann	7.29
Backprop + Darken and Moody	Schiffmann	2.10
J. Schmidhuber	Schiffmann	2.77
R. Salomon	Schiffmann	5.86
Chan and Fallside	Schiffmann	5.83
Polak-Ribiere + line search	Schiffmann	5.83
Conj. gradient + line search	Schiffmann	6.16
Silva and Almeida	Schiffmann	1.55
SuperSAB	Schiffmann	1.58
Delta-Bar-Delta	Schiffmann	1.63
RPROP	Schiffmann	1.98
Quickprop	Schiffmann	1.75
Cascade correlation 10 units	Schiffmann	1.58
Cascade correlation 20 units	Schiffmann	1.52

The results obtained here, on the other hand, are all using batch mode updates. The GDX algorithm is comparable to the batch-mode Backprop algorithms and its results are around 6% error, a little better than that achieved by Schiffmann, but just as useless. The LM and QNN algorithms however achieved much better results despite using batch mode updates, just as Prechelt did with his batch-mode RPROP algorithm in the Proben1 tests.

The other point to note is that in most cases the network structures used in the literature were much larger than the MLP and GFNN structures used here. Only, the SIANN structure is of comparable size to the others. Despite their size, the GFNN (101 weights) and MLP (103 weights) networks were able to achieve comparable results to the much larger networks. Our 21-4-3 MLP trained with the LM algorithm achieved a mean error rate below 2%. Schiffmann et al managed to get a number of instances where the error was under 2%, but using a much larger, fully interconnected 21-10-3 MLP. Only two of the other reported instances managed to reach this level of accuracy, Prechelt reported 1.91% with the Proben1 *thyroid2* dataset using a 21-16-8-3 two-hidden-layer network and Jankowski achieved 1.44% with a weighted k-nearest neighbour classifier with $k = 3$ and 3-fold cross-validation.

The third point is the maximum number of training epochs. Schiffmann trained the networks for a maximum of 5000 epochs, Prechelt set the maximum at 3000 epochs, while here it was set to 1000 epochs for consistency with all the other

benchmark problems. In all three investigations, a validation set was used for early stopping. Even though early stopping was used, a number of runs trained for the maximum number of epochs, particularly for the GDX and QNN algorithm. However, it is debatable whether increasing the maximum number of epochs would actually result in an improvement in performance.

These three points indicate areas that can be investigated in the future, particularly for this problem; that is, training these networks using on-line mode training, using larger structures, and training for a greater number of epochs.

In the final analysis, what has been demonstrated is that the networks under investigation here were able to achieve results comparable to the previously reported results by other investigators, and, in the case of GFNNs and MLPs, they were able to do this with much simpler network structures.

8.3 Performance Comparison with MATLAB Toolbox MLPs

8.3.1 Benchmark tests using MT-MLPs

The results shown in the previous sections were obtained using the same MATLAB code for GFNNs, SIANNs and MLPs. The code was written to take advantage of the fact that SIANNs and MLPs are subsets of GFNNs. The majority of the code and data structures used are generic, with relevant sections of code branching out to cater for the differences in neuron type. While the code has been based loosely on the standard MATLAB Neural Network Toolbox code, it has 'evolved' as the research progressed, with emphasis on achieving results rather than speed of execution. The code has not been optimised for memory or computational efficiency, and therefore the time taken to train the networks would probably be much longer than it should.

In this section, the performance of the 'Generalised' MLPs, or *G-MLPs*, used in the previous section is compared to that of MLPs using the MATLAB Neural Netwok toolbox code, dubbed "MATLAB Toolbox MLPs" or *MT-MLPs*. The purpose of this comparison is to quantify the 'inefficiency' of the 'generalised' code, at least approximately, so that valid comparisons of the training time could be made with other results in the literature. Additionally, the comparisons would give an idea of differences in performance between the G-MLPs and the 'off-the-shelf' MT-MLPs.

For each of the benchmark tests, the same MLP network structures used previously were generated and trained using the default MATLAB data structures and code. The networks were trained using the MATLAB Toolbox implementation of the GDX and LM algorithms. All the training parameters were kept the same, the only difference being the weight initialisation scheme. The MT-MLPs were

initialised using MATLAB's default weight initialisation scheme, the Nguyen-Widrow initialisation scheme (Demuth & Beale, 1992; Nguyen & Widrow, 1990), instead of the scaled uniform weight distribution scheme described in Section 4.5.2.

The mean test error and test ARV for the six benchmarks tests are presented in Tables B.41 to B.46 in Appendix B. The results obtained showed definite difference in performance between the G-MLPs and the MT-MLPs, with the G-MLPs generally achieving better performance. The difference in accuracy can be attributed to one or two possible sources: the difference in initialisation schemes and the difference in the implementations of the training algorithms. In order to remove the differences due to initialisation from the equation, tests were conducted with MT-MLPs with initial weights set to *exactly* the same weights as the G-MLPs. The MT-MLPs had their initial weights copied across from the G-MLPs, and activation functions set the same as the best performing G-MLPs for each algorithm.

Tables 8.10 and 8.11 present the results obtained using these three sets of MLPs and Figs. 8.8 to 8.13 show a comparison of the mean, median and minimum error and mean training times. The results for the standard structures, initialisation and training algorithms, as given in Chapter 4, are denoted as G-MLP. The standard MATLAB Neural Network Toolbox networks and training algorithms are denoted MT-MLP, with two sets of results based on initialisation scheme. The MT-MLPs initialised with the default Nguyen-Widrow initialisation scheme are labelled 'NW-init' while those with initial weights copied across from the G-MLPs are labelled 'GF-init'. The GF-init MT-MLP results are for the same activation function combination as the corresponding G-MLP, so that differences in results can only be due to differences in the implementation of the algorithms.

Comparison of the results for the Wisconsin Breast Cancer dataset shows that the NW-init MT-MLPs had a higher error rate than the G-MLPs for both training algorithms, though the actual difference between the means is less than 0.3%. The NW-init MT-MLPs had 54% and 28% of networks achieving perfect classification with the GDX and LM algorithms, respectively—compared to 86% and 44% for the G-MLPs. The GF initialised MT-MLPs results indicate that the reasons for these differences are not the same for both training algorithms. When trained with GDX, there is hardly any difference in accuracy between the G-MLPs and the GF-init MT-MLPs, indicating that the initialisation scheme is the cause for the difference in performance. The LM-trained networks, on the other hand, have both MT-MLPs producing similar results and G-MLPs with better accuracy. This would indicate that the difference in results between G-MLPs and MT-MLPs in this case is not due to the initial weights, but due to differences in the implementation of the algorithm.

Table 8.10 Comparison of G-MLPs and MT-MLPs trained on (a) Breast Cancer (b) Diabetes, (c) 3-bit Parity, (d) Multi-class and (e) Thyroid datasets.

Network Structure	Train Algor.	Activation functions		Performance (% of runs)			Avg Epochs			Test Error				Mean train time (s)
		Hidden	Out	→ goal	0% err	20% <	All runs	→ goal	0% Err	Best (%)	Mean (%)	95% CI	Med. (%)	
a) Wisconsin Breast Cancer														
G-MLP	GDX	Lgs	Lin	0	86	100	158	*	159	0.00	0.08	± 0.05	0.00	8.6
	LM	Lgs	Tnh	0	44	100	57	*	56	0.00	0.51	± 0.15	0.56	9.9
MT-MLP (GF-init)	GDX	Lgs	Lin	0	88	100	191	*	192	0.00	0.07	± 0.05	0.00	2.2
	LM	Lgs	Tnh	0	28	100	60	*	58	0.00	0.75	± 0.18	0.56	1.4
MT-MLP (NW-init)	GDX	Lgs	Lin	0	54	100	232	*	255	0.00	0.37	± 0.14	0.00	2.8
	LM	Lgs	Lin	0	28	100	63	*	61	0.00	0.72	± 0.22	0.56	1.4
b) Pima Indians Diabetes														
G-MLP	GDX	Tnh	Tnh	0	0	30	188	*	*	18.75	20.45	± 0.22	20.31	11.1
	LM	Lgs	Tnh	0	0	26	58	*	*	18.75	20.75	± 0.31	20.83	10.8
MT-MLP (GF-init)	GDX	Tnh	Tnh	0	0	16	188	*	*	19.79	20.95	± 0.64	20.57	2.3
	LM	Lgs	Tnh	0	0	34	61	*	*	17.71	20.88	± 0.52	20.83	1.3
MT-MLP (NW-init)	GDX	Tnh	Lgs	0	0	12	191	*	*	19.27	21.48	± 0.88	20.83	2.4
	LM	Lgs	Lin	0	0	30	61	*	*	18.23	21.11	± 0.41	21.35	1.3
c) 3-bit Parity														
G-MLP	GDX	Lgs	Lgs	96	96	100	224	192	192	0.00	0.50	± 0.69	0.00	1.0
	LM	Lgs	Lin	100	100	100	11	11	11	0.00	0.00	± 0.00	0.00	0.2
MT-MLP (GF-init)	GDX	Lgs	Lgs	96	96	100	266	235	235	0.00	0.50	± 0.69	0.00	2.1
	LM	Lgs	Lin	96	96	98	28	12	12	0.00	1.50	± 2.49	0.00	0.4
MT-MLP (NW-init)	GDX	Lgs	Lgs	70	72	94	500	286	306	0.00	4.25	± 2.05	0.00	4.1
	LM	Lgs	Lgs	92	92	96	27	22	22	0.00	2.00	± 2.25	0.00	0.4
d) Multi-Class														
G-MLP	GDX	Lgs	Lgs	0	0	100	225	*	*	4.00	5.37	± 0.22	5.33	11.5
	LM	Tnh	Tnh	0	0	100	67	*	*	4.00	5.83	± 0.16	6.00	29.9
MT-MLP (GF-init)	GDX	Lgs	Lgs	0	0	100	278	*	*	4.00	6.01	± 0.21	6.00	3.5
	LM	Tnh	Tnh	0	0	100	94	*	*	6.00	12.84	± 5.04	7.33	2.5
MT-MLP (NW-init)	GDX	Lgs	Lgs	0	0	100	294	*	*	3.33	5.53	± 0.27	5.33	4.0
	LM	Lgs	Lin	0	0	100	81	*	*	5.33	5.99	± 0.14	6.00	2.4
e) Thyroid														
G-MLP	GDX	Lgs	Lgs	0	0	100	419	*	*	5.60	5.96	± 0.04	6.01	254.0
	LM	Lgs	Lgs	0	0	100	119	*	*	1.17	1.72	± 0.08	1.69	882.2
MT-MLP (GF-init)	GDX	Lgs	Lgs	0	0	100	457	*	*	5.83	6.04	± 0.02	6.07	20.9
	LM	Lgs	Lgs	0	0	100	102	*	*	1.23	1.97	± 0.24	1.87	180.1
MT-MLP (NW-init)	GDX	Lgs	Lin	0	0	100	396	*	*	5.89	6.59	± 0.17	6.65	18.4
	LM	Lgs	Lin	0	0	100	189	*	*	1.81	2.49	± 0.18	2.28	185.8

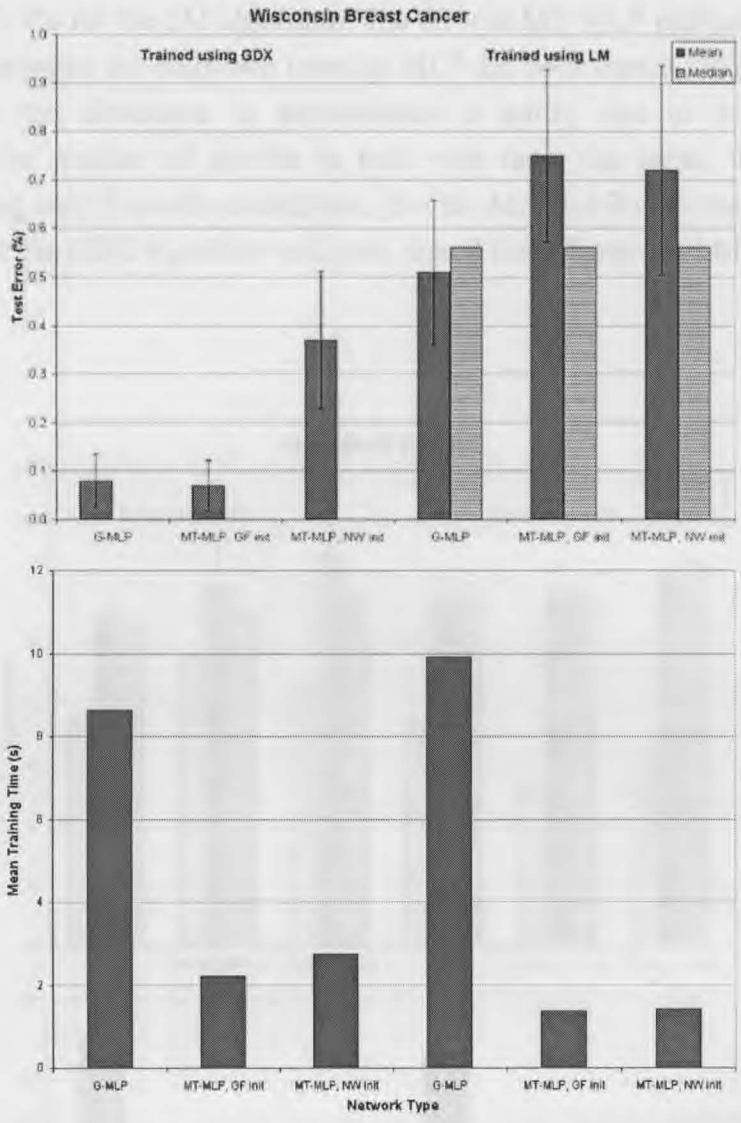


Fig. 8.8: Comparison of mean and median error and mean training time for the Breast Cancer dataset using 'generalised' and MATLAB Neural Network Toolbox MLPs.

The big difference in performance lies in the training time. The MT-MLPs actually needed more epochs to train, but were still 3 times faster in actual execution time for the GDX and 7 times faster for the LM algorithm. This means that the MATLAB Toolbox code has been optimised to a point where it can train the networks in a fraction of the time. The MATLAB Toolbox MLP implementation is in the order of 4 to 8 times faster than the GFNN implementation, taking into account the additional epochs.

The results for the Diabetes dataset show similar trends, with the NW-init MT-MLPs having higher mean and median error rates for both algorithms; the average error rates for both algorithms were more than 21%, compared to under 21% for the G-

MLPs. The actual difference between means is about 1.0% for the GDX algorithm and less than 0.4% for the LM algorithm. The GF-init MT-MLP performance, in this case, was in between the other two types of MLP for both training algorithms. This indicates that the difference in performance is partly due to implementation differences. The number of epochs to train was most the same, the maximum difference being only 3 epochs in each case, but the MATLAB code was more than 4 times faster for the GDX algorithm and more than 8 times faster for LM.

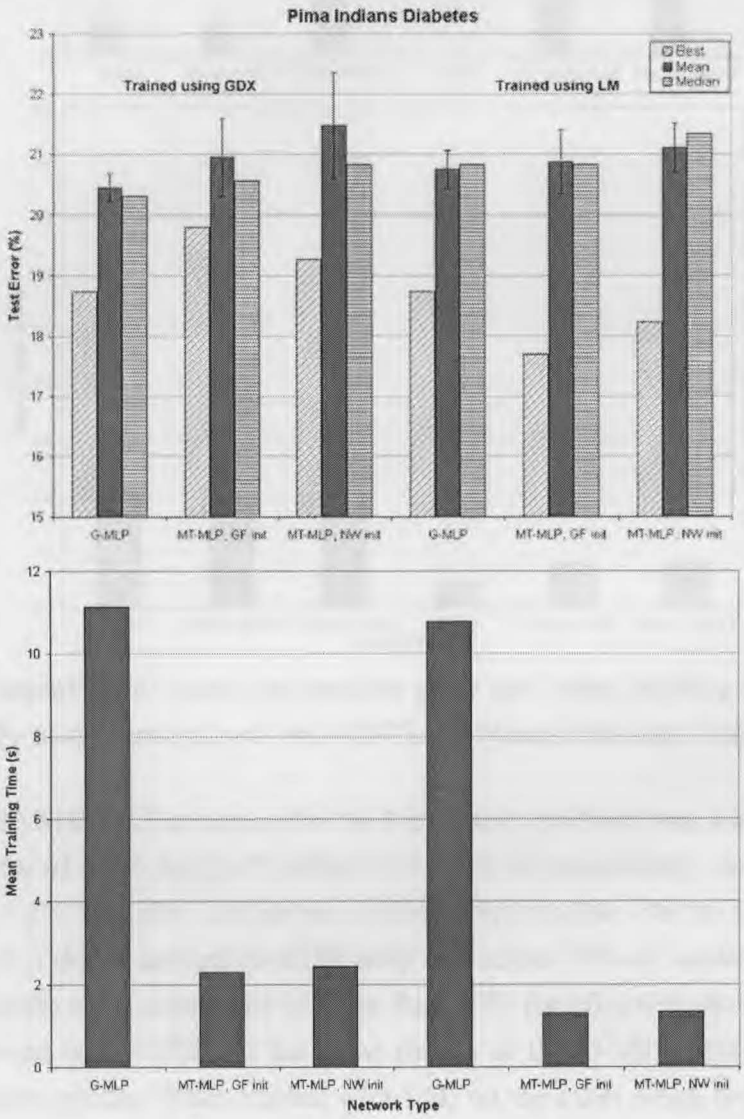


Fig. 8.9: Comparison of best, mean and median error and mean training time for the Diabetes dataset using 'generalised' and MATLAB Neural Network Toolbox MLPs.

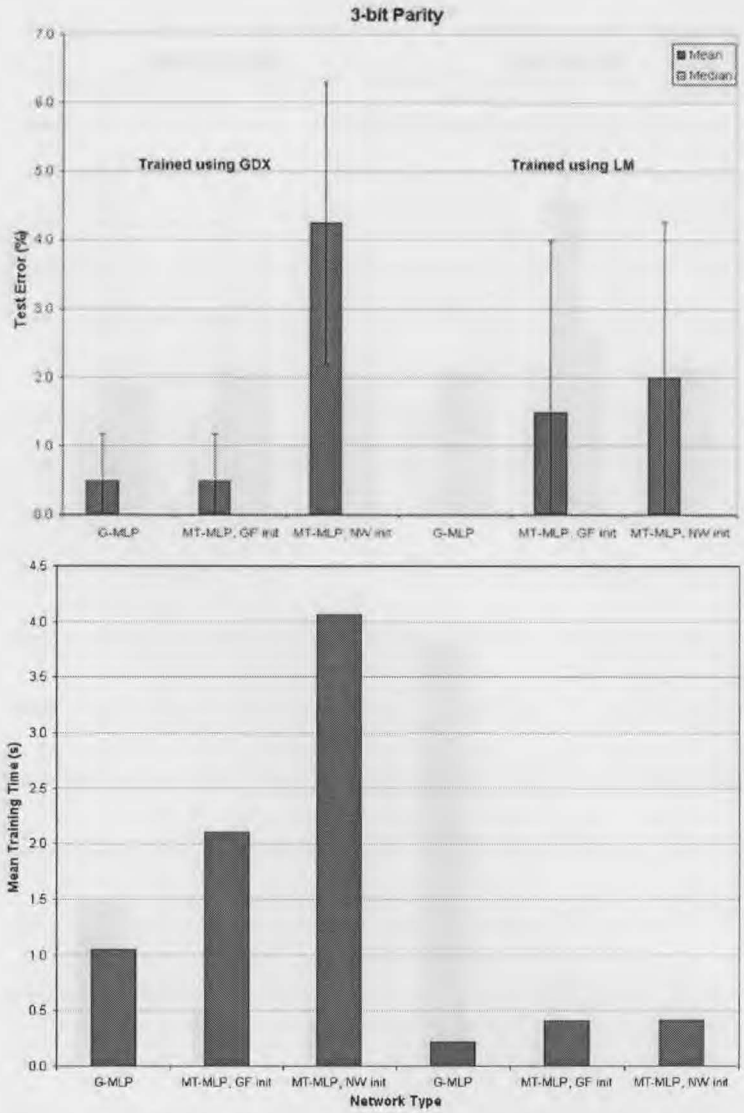


Fig. 8.10: Comparison of mean and median error and mean training time for 3-bit Parity using 'generalised' and MATLAB Neural Network Toolbox MLPs.

The NW-init MT-MLP accuracy for the 3-bit parity problem was a lot worse, with mean error rates of 4.2% and 2.0% with GDX and LM respectively, compared to the G-MLP with 0.5% for GDX and perfect (0.0% error) results with the LM algorithm. The NW-init MT-MLP trained on GDX only had about 70% of networks achieving all correct classification, compared to more than 90% for all other cases. The GF-init MT-MLP trained with GDX had the same results as the G-MLP, except for taking about 20% more epochs. When trained with LM, on the other hand, the GF-init MT-MLP performance was in between the others two. Again, the initialisation seems to have a bigger effect with the GDX algorithm. The Parity problem is the only one where the G-MLPs were faster to train than the MT-MLPs, twice as fast for LM and up to 4 times faster for GDX. After factoring in the greater number of epochs required, the training speed was almost equal for the LM algorithm, but the GDX was still twice as fast as the MATLAB Toolbox implementation.

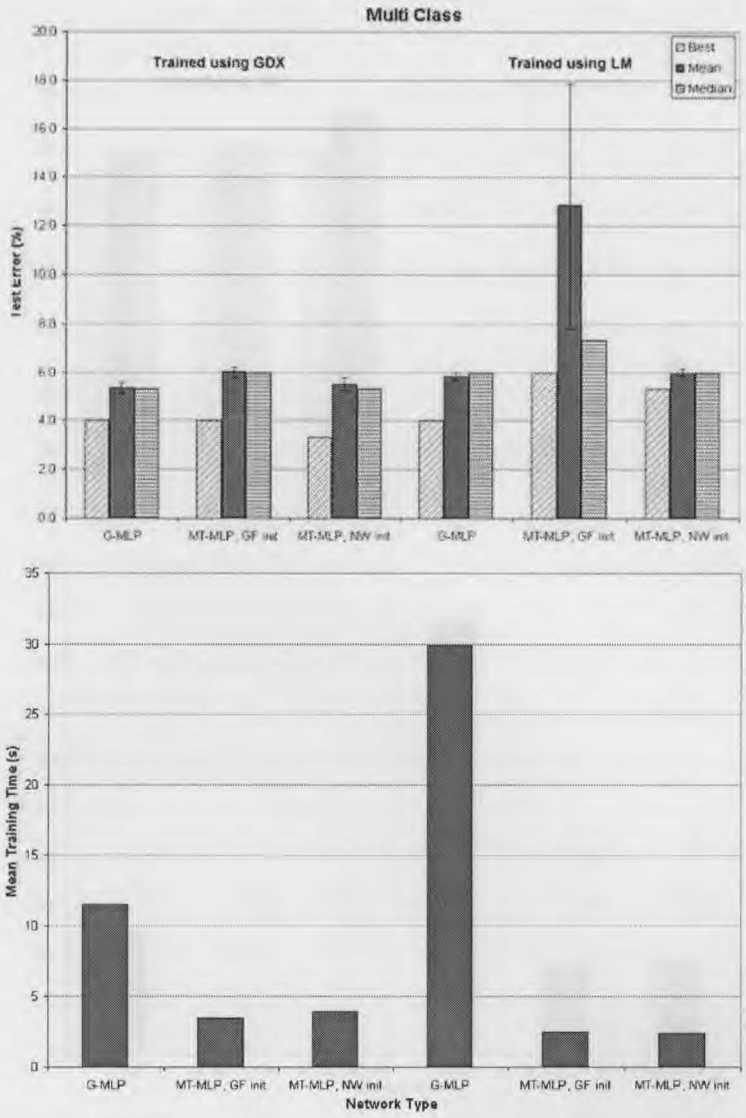


Fig. 8.11: Comparison of best, mean and median error and mean training time for the Multi-Class problem using G-MLPs and MT-MLPs.

For the Multi-Class problem, the NW-init MT-MLPs had error rates exactly 0.16% higher than the G-MLPs for each of the training algorithms, and took 20% to 30% more epochs to train. The GF-init MT-MLPs had higher error rates than both the other types of MLP for this case. When trained with LM, the GF-init MT-MLP had an error rate more than double the other two. The most probable explanation is that this is due to the activation function. Looking at the overall results in Appendix B, it can be observed that the MATLAB implementation does poorly when the output activation is the hyperbolic tangent function. It is especially bad when trained with LM, the trend observed across all the benchmark tests. However, it is not clear why this is so. In terms of training time, the trend is similar to most of the other tests. The MT-MLPs are about 3 times faster to train using the GDX algorithm and 12 times faster using the LM algorithm, in terms of actual computation time, despite requiring more epochs.

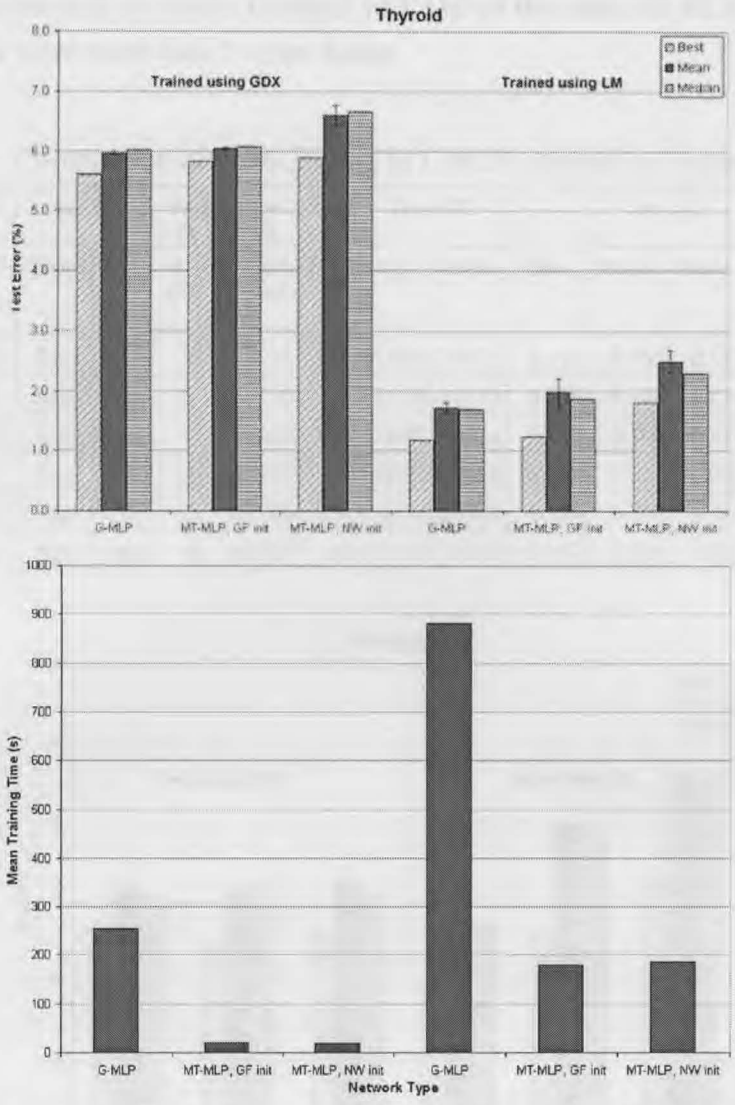


Fig. 8.12: Comparison of best, mean and median error and mean training time for the Thyroid problem using G-MLPs and MT-MLPs.

The results on the Thyroid dataset show the same trends as most of the earlier classification problems. The mean error rates of the NW-init MT-MLPs were slightly higher than that of the G-MLPs, with a difference in mean error rates of 0.65% and 0.77% for GDX and LM, respectively. The GF-init MLP error rate was within 0.1% of the G-MLP when trained with GDX. When trained with LM, it achieved error rates midway between the other two. The G-MLPs took more than 12 times longer to train with GDX, and more than 4 times longer with LM.

The Sunspots time-series prediction results show the mean test ARV achieved by all three types of MLPs, when trained with GDX, to be almost exactly the same. When trained with LM, the 'usual' trend is observed with G-MLP being best and the NW-init MT-MLP worst, and the GF-init MT-MLP in between. The NW-init MT-MLPs' required almost 3 times as many epochs as the G-MLPs when trained with the GDX algorithm, but still took about 15% less time. When trained with the LM

algorithm, the number of epochs required was almost the same for all three types, but the MT-MLPs were more than 5 times faster.

Table 8.11 Comparison of G-MLPs and MT-MLPs trained on Sunspots dataset.

Network Struct.	Train-ing Algor-ithm	Act-fns		Performance (% of runs)			Avg Epoc hs	Test MSE		Test ARV				Mean time to train (s)
		Hidden	Out	→ gl	all in tol	80% tol		Best	Median	Best	Median	Mean	95% CI	
G-MLP	GDX	Tnh	Lin	0	14	98	151	0.0094	0.0129	0.113	0.155	0.162	± 0.024	5.8
	LM	Lgs	Lin	0	22	100	57	0.0079	0.0113	0.095	0.136	0.138	± 0.006	7.0
MT-MLP (GF-init)	GDX	Tnh	Lin	0	0	100	255	0.0087	0.0112	0.123	0.160	0.158	± 0.005	2.8
	LM	Lgs	Lin	0	6	100	60	0.0091	0.0138	0.129	0.196	0.200	± 0.012	1.2
MT-MLP (NW-init)	GDX	Lgs	Lin	0	6	100	434	0.0091	0.0109	0.130	0.155	0.163	± 0.008	4.9
	LM	Lgs	Lgs	0	8	100	59	0.0114	0.0161	0.162	0.229	0.231	± 0.012	1.3

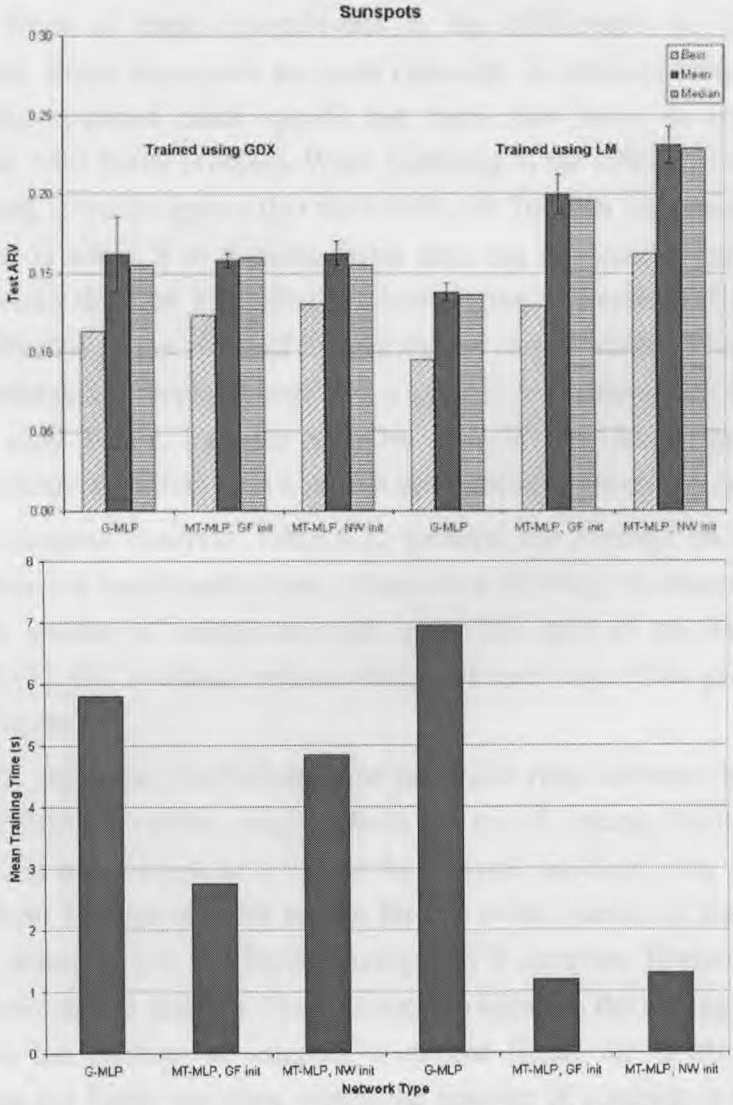


Fig. 8.13: Comparison of best, mean and median test ARV and mean training time for Sunspots prediction problem using G-MLPs and MT-MLPs.

8.3.2 Analysis of efficiency test results

The overall trend was that the NW-init MT-MLPs generally had lower accuracy than the G-MLPs. This is reinforced by the results obtained across all the different activation functions. Looking at the Tables B.41 to B.46 in Appendix B, it can be seen that the average error across all the activation functions is always lower for the G-MLPs, with the exception of GDX with the Sunpots problem. The results for the GF-init MT-MLPs indicate that the difference in accuracy, for networks trained using the GDX algorithm, is mainly due to the initialisation scheme. For the LM-trained networks, on the other hand, only about half of the difference can be attributed to initialisation, the other half arising from implementation of the algorithms. These results, though problem-dependent, can be viewed as an endorsement of the scaled uniform weight distribution initialisation scheme used with G-MLPs (GF-init). This, however, is only an interesting aside.

The main focus of these comparisons is the differences in 'efficiency' of implementations, where the results are more clear-cut. In terms of computation time, MT-MLPs often required more epochs but were still faster to train, with the exception of the 3-bit Parity problem. When factoring in the difference in the number of epochs trained, it would appear that the MATLAB Toolbox implementation of the GDX algorithm is about 3 to 4 times faster than the implementation used in our experiments, while the LM algorithm implementation is between 5 and 16 times faster. The difference in the speed of the algorithms can be attributed to two factors. Firstly, the 'generalised' implementation is a generic implementation that is written to handle not only MLPs, but also SIANNs and GFNNs. More importantly, the actual code for these algorithms has not been optimised for execution performance.

For a more detailed analysis, Table 8.12 presents the average training time per epoch for the various benchmarks tests, obtained by dividing the mean training time by the average number of epochs for each case. The ratio of this average for the GFNN and MATLAB Toolbox implementation of each algorithm gives an idea of the difference in speed.

For the GDX algorithm, the training time per epoch ratio between the G-MLP and MT-MLPs is relatively similar, ranging from 3.5 to 4.9, except for the 3-bit Parity where the ratio is much lower at 0.59 and the Thyroid problem with a much higher ratio of more than 13. One possible reason for this is the number of training samples used for each problem, with the Parity having only 8 samples, Thyroid having 3772 and the rest between 220 and 384. The relationship between the average training time per epoch and the number of samples is almost linear for G-MLPs. The only exception being the Parity problem, where the number of samples is very small. In this case, the time taken for 'overhead' activities, that could normally be ignored as negligible compared to the actual training time, would come into play.

Table 8.12 Comparison of average training time per epoch for G-MLPs and MT-MLPs for all datasets

Benchmark Test	No. of Training Samples	Validation	No. of outputs	Average Training time per epoch (s)					
				GDX			LM		
				G-MLP	MT-MLP	Ratio	G-MLP	MT-MLP	Ratio
Breast Cancer	348	Yes	1	0.055	0.012	4.72	0.174	0.023	7.64
Diabetes	384	Yes	1	0.059	0.012	4.86	0.186	0.021	8.77
3-bit Parity	8	No	1	0.005	0.008	0.59	0.020	0.015	1.37
Multi-Class	300	Yes	3	0.051	0.013	4.08	0.446	0.027	16.81
Sunspots	221	Yes	1	0.038	0.011	3.55	0.122	0.020	6.10
Thyroid	3772	Yes	3	0.606	0.046	13.26	7.413	1.766	4.20
	MEAN					*5.18*			*7.48*

The larger the number of training samples, the larger the ratio, indicating that the MATLAB Toolbox implementation is able to process large numbers of samples more efficiently. The average time per epoch does not vary that much for the MATLAB implementation of GDX, with the longest time about 6 times longer than the shortest. The GFNN implementation, on the other hand, has the longest, more than 100 times longer than the shortest. The MATLAB code has been optimised for large array computation, thereby making the training time per epoch much less sensitive to the size of the training set.

The trend for the LM algorithm is similar to the GDX algorithm. The Parity problem has a ratio of 1.37, indicating that the GFNN implementation is only slightly less efficient than the MATLAB Toolbox implementation for the small number of samples. The 'medium'-sized datasets have ratios in the region of 6.1 to 8.8, except for the Multi-class problem, which has a ratio of 16.8. This is most likely due to the use of three outputs neurons in this problem, which increases the size of the Hessian matrix used in the LM algorithm, and hence increasing the computation time. The average training time per epoch for the GFNN implementation is in fact proportional to the product of the sample size and the number of outputs. The exception is again the Parity problem because the sample size is extremely small.

The increased Hessian size affects both implementations of the LM algorithm, but the MATLAB Toolbox is not impacted as much. The only anomaly is thyroid problem, with a ratio of only 4.2. For all the other problems, the average time per epoch for the MATLAB Toolbox implementation is between 0.015 to 0.030 seconds, but this jumps to 1.77 seconds for the thyroid problem. The GFNN implementation of the algorithm shows its sensitivity to such factors as dataset size and number of outputs, ranging from 0.02 to 7.41 seconds.

On average, the MATLAB Toolbox is around 5 times faster than the GFNN implementation for the GDX algorithm, and more than 7 times faster for the LM algorithm.

While on the topic of efficiency, the QNN algorithm and its variants have not been optimised for speed either. Unfortunately, there is no implementation of the QNN algorithm in the MATLAB Toolbox to compare with! There will definitely be room for fine tuning and improving the efficiency of the code, as it is based on the same structures and principles as the implementations of the GDX and LM algorithms for GFNNs.

The implementation of the QNN algorithm for 'pure' MLP networks can in fact be improved quite simply by using the fact that the weights of an MLP are unconstrained. Removing all constraints simplifies the 'recurrent network' equation considerably. This simplification has been implemented successfully, and details of the changes to the equations along with some benchmark results can be found in (Arulampalam & Bouzerdoum, 2001b, 2002b). This unconstrained version has not been used in these tests, however. The same generic code that can handle MLPs, SIANNs and GFNNs has been used so that it is clear that the differences in results are due to the different network structures and not due to changes to the algorithm.

8.4 Discussion

In order to link the various results obtained in this chapter, the general trends across all the benchmark tests are now discussed. This will provide an overview of the relative merits of the networks and algorithms used.

8.4.1 Trends in Training Algorithm Performance

In terms of accuracy, the QNN algorithm variants were able to train the various network types to achieve good accuracy, particularly with the SIANNs and GFNNs that have constraints on some weights. An interesting point to note is that the QNN algorithm appears to come out best for the 'harder' problems, such as the Diabetes, Sunspots and Thyroid datasets. The best results for the GFNNs and SIANNs trained on these three problems were achieved with the QNN algorithm, as well as the best overall results for the Diabetes and Sunspots problems. This is probably due to the fact that the QNN algorithm is able to incorporate the constraint on the decay parameter a while working out the 'optimal' weight update. The other algorithms update the weights, then impose the constraint on the parameter. This may result in a sub-optimal weight update if the constraint changes the weights.

The only disadvantage of the QNN algorithm is the long training time required, hence the need to improve the efficiency of this algorithm. Additionally, it should be remembered that the QNN variants used were selected based on tests on SIANNs

only. This could have skewed the choice of variants, and possibly even their formulation, in favour of SIANNs.

The second-order LM algorithm results in good accuracy as expected, coming out as the best by far in some problems, such as the 3-bit Parity problem. In most cases, the LM worked best with the MLPs, and resulted in slightly higher error rates compared to the QNN algorithm for the SIANNs and GFNNs. However, the LM algorithm was generally much faster than the QNN, with a couple of exceptions. It has lived up to its reputation of being one of the most powerful neural network algorithms, but the disadvantage of the LM algorithm has always been the requirement to calculate and invert the Hessian matrix. The resultant memory and computation requirements tend to offset the fact that the LM algorithm generally requires fewer epochs to train the networks compared to other algorithms. As discussed in Section 8.3, the problem becomes more apparent as the number of samples in the training set and the number of outputs increase. As can be seen from Table 8.12, the MATLAB implementation of the LM algorithm also gets affected by these increases, but not as badly as the GFNN implementation since the MATLAB code is more efficient.

The GDX algorithm, being a first-order algorithm, is generally the fastest algorithm in terms of actual computation time because of the relative simplicity of the algorithm. This simplicity, however, means that the GDX algorithm generally does not perform as well as the second-order algorithms in terms of the performance of the trained networks, particularly for the so-called 'harder' problems.

8.4.2 Trends in Network Performance

This brings us to the topic of accuracy of various types of networks over the benchmark problems. Overall, the results obtained here compare well with results reported in the literature. What is noteworthy is that these results were obtained using much smaller networks in most cases. The GFNN structures used have only one, or a maximum of two, generalised shunting neurons plus one or three linear or sigmoid output neurons, depending on the number of outputs required. In some cases, such as the Breast Cancer and Diabetes problems, a single GS neuron has been used as the 'network'. Amazingly, this single-neuron network was able to achieve 100% correct classification for the Breast Cancer problem for the majority of the test runs. Even the MLP structures used here were smaller than in the majority of those reported in the literature; the MLP structures were chosen to have approximately the same number of weights as the GFNN networks to which they were being compared.

The 'best' network type tends to vary from problem to problem. The best average and individual network performance for the Diabetes problem was obtained by SIANN. For the Parity and Sunspots problems, on the other hand, the GFNN had the

best mean error, though the SIANN had the best individual network performance for Sunspots. For the other three problems, namely the Breast Cancer, Multi-class and Thyroid problems, the MLP produced the best results. While this is not the resounding endorsement of the shunting inhibition-based networks hoped for, it is not altogether surprising, indicating why MLPs have been the most popular type of neural network used for these kinds of problems over the last couple of decades.

It should be noted that the three types of artificial neural networks compared here are not really three different types of networks, but all actually fall within the umbrella of GFNNs. As presented in Chapter 7, the generalised shunting neuron (GSN) has the 'plain' static shunting neuron and perceptron-type sigmoid and linear neurons as special cases, therefore SIANNs and MLPs are just subsets of GFNNs. From this point of view, GFNNs were the best performing networks in all cases!

8.5 Conclusion

The performance of SIANNs, GFNNs and MLPs, tested across a number of benchmark problems, has been evaluated and compared. The performance of the training algorithms developed for them, has also been investigated, including comparisons of efficiency of code with commercially available implementations. The results obtained here have also been compared to work done by other researchers, putting this work in perspective of the general body of knowledge in this area.

The results are promising. The shunting inhibition networks are able to perform well with very small network structures. The GFNN networks used in the benchmark tests had only one or two GSNs, plus an output layer of linear or sigmoid neurons where needed. Two of the six benchmark tests used only a single GS neuron, the simplest possible network structure. The overall results are comparable to or better than other reported results. This is despite the fact that, in most cases, the networks used in the other literature are much larger.

From a training algorithm perspective, the first-order GDX algorithm has proven to be a fast and effective training algorithm, though sometimes not able to achieve the desired accuracy levels with the more complex problems. The second-order LM algorithm was able to achieve better accuracy, though taking longer due to its relative complexity. The QNN algorithm was also able to achieve good results, quite often even better than LM, but at the cost of longer training time.

Comparisons with the MATLAB Toolbox code show that the training algorithms implemented for GFNNs could be optimised to improve efficiency and reduce computation time. These tests also showed that the initialisation scheme used with

the GFNNs tends to produce better results than the MATLAB default initialisation scheme.

The question posed at the beginning of this chapter, "How do shunting inhibition based networks compare with other types of networks?", can now be answered. The answer is that shunting inhibitory networks compare well. They are capable of achieving accuracy levels comparable to or better than other types of networks, and they are able to do so with simple structures.

Chapter 9

Conclusion

9.1 The Journey of Discovery

This chapter brings together the various threads of the research conducted thus far. We can think of the work presented here as a 'journey of discovery', one result leading into the next exploration, with detours along the way to investigate some interesting prospects. The structure of this thesis reflects this journey, forming the 'travelogue'. A 'map' of this journey is provided in Fig. 9.1, showing the path travelled and the 'discoveries' made.

The starting point was the investigation of SIANNs, motivated by the ability of shunting neurons to produce non-linear decision boundaries. The objective was to create shunting inhibition-based feedforward neural networks that could be trained for classification and regression. Applying SIANNs to problems of this kind required training algorithms to be developed. A number of different types of training algorithms have been developed, from the basic gradient descent to hybrid and novel algorithms. An interesting detour has been the development of a novel algorithm: the Quadratic Neural Network (QNN) algorithm; it uses a recurrence equation to simulate a recurrent neural network performing bound-constrained quadratic optimisation.

SIANNs have been successfully applied to a number of problems, but the standard SIANN network structure is restricted in terms of size of the layers. This sometimes results in structures that are too small, or inordinately large, for the particular problem at hand. Consequently, enhancements have been made to allow the network

size to be expanded or reduced as required. Problems faced when reducing the SIANN layer size highlighted one major deficiency: since the shunting neuron is allowed only one excitatory input, it is not clear what subset of inputs can be used as excitatory inputs. The solution was to create a shunting neuron model that allows multiple excitatory, as well as inhibitory, inputs, resulting in the Generalised Shunting Neuron model. This then led to the creation of the Generalised Feedforward Neural Network (GFNN) architecture.

In order to prove the worth of the neural networks developed, SIANNs, GFNNs and MLPs have been tested across a number of benchmark problems, and their performance evaluated and compared, including comparisons with results reported by other authors in the literature.

Now that the end of this particular journey has been reached, it is time to reminisce, savour the highlights, and look to the journeys ahead. The next section is a summary of the results that form the highlights and link the various strands of the work done so far. The final section discusses future research directions that have emerged from the research presented here.

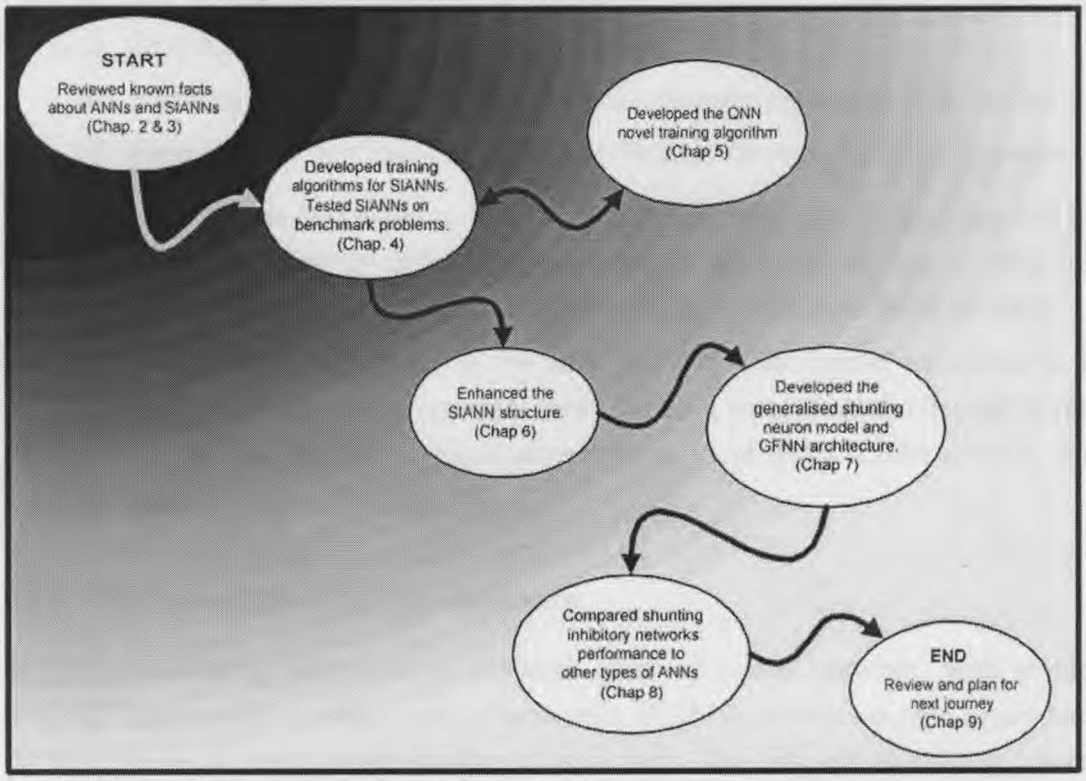


Fig. 9.1: A map of the 'Journey of Discovery'

9.2 Summary of Research Outcomes

The initial thrust of this research was to investigate the suitability of shunting inhibition-based feedforward networks, particularly SIANNS, for classification and non-linear regression tasks. The aim was to create powerful, trainable networks, with non-linear decision surfaces. The contribution of this thesis can be divided into two main parts:

a) Development of training algorithms for SIANNs.
b) Enhancement of the SIANN architecture to improve performance.

9.2.1 Development of training algorithms

The training algorithm part of the research has resulted in the development and implementation of a number of algorithms for shunting inhibitory networks. The algorithms can be divided into *five main types*, with a number of variants for each:

a) Gradient descent (4 variants). The Gradient Descent with adaptive learning rate and momentum (GDX) has been the main variant used as it has the best performance among the gradient descent algorithms.
b) Levenberg-Marquardt (LM) (3 variants)
c) Direct Solution-GDX hybrid (DS-GDX)
d) Random Optimisation Method (ROM) stochastic algorithm (2 variants)
e) Novel algorithms based on Quadratic Neural Network (QNN) (9 variants).

The 'bonus' in this part was the development of the novel QNN algorithm and its variants. This algorithm is able to produce good results, particularly with the shunting networks that require certain parameters to be constrained while training.

Overall, the ROM algorithm was the only one that didn't meet expectations. It was fast to run, but the trained networks were not able to achieve the desired levels of accuracy. All the other algorithms were able to yield good results overall, and some excellent results in particular tests.

9.2.2 Enhancing the SIANN architecture

SIANNs have been shown to be a viable class of neural network, with results obtained comparable to other types of networks. SIANNs were even able to produce the best results in some of the final benchmark tests. The original SIANN structure, however, had the size of its layers determined by the number of inputs and outputs of the problem. The enhanced structure, described in Chapter 6, enabled greater flexibility in the size of layers. Adding extra shunting neurons for problems that had a small number of inputs generally resulted in improved accuracy. Problems with

large number of inputs tend to end up with inordinately large SIANN structures. However, reducing the number of shunting layer neurons normally resulted in reduced accuracy, as only a subset of the inputs could be used as excitatory inputs. This is due to the restriction imposed on the shunting neuron model used, allowing it to have only a single excitatory input.

Addressing this restriction resulted in the creation of the Generalised Shunting Neuron (GSN) model. A GSN can have multiple, weighted excitatory and inhibitory inputs, with a transfer function for each type of input. It has been shown that a GSN can produce various types of transfer characteristics by simply varying the synaptic weights. The GSN has the static shunting neuron and perceptron-type sigmoid and linear neurons as special cases, where certain weights have been removed or fixed to constant values. This has been a key 'discovery' of this work. It has led to the definition of the Generalised Feedforward Neural Network (GFNN) architecture.

The broad definition of the GFNN architecture encompasses a variety of structures, including SIANNs, MLPs, and 'plain' GFNNs as investigated in this work. The term 'plain' GFNNs has been used for the networks with a single layer of generalised shunting neurons (denoted G) and networks with a hidden layer of GSNs and an output layer of perceptron-type neurons (denoted GP).

Fig. 9.2 illustrates the point diagrammatically. It shows MLPs, SIANNs and 'plain' GFNNs as subsets within the GFNN architecture, with points to highlight the differences between the three. It also has a brief description of the types of networks that are outside these three subsets, but still fall within the broad definition of GFNNs.

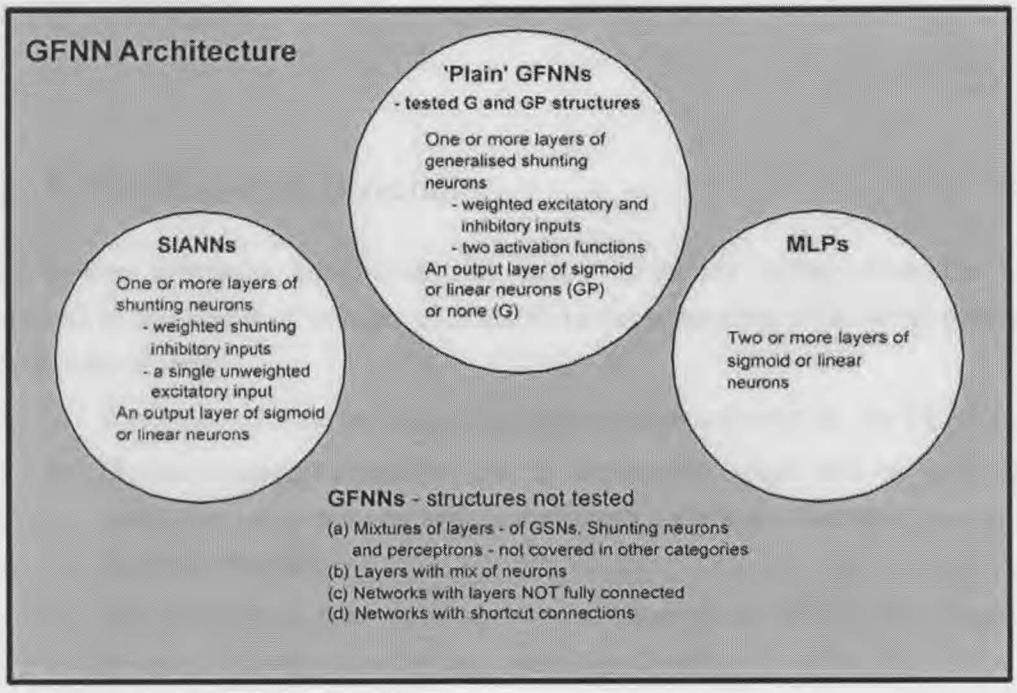


Fig. 9.2: The GFNN architecture superset with SIANN and MLP subsets.

9.2.3 Overview of Results

The performance of MLP, SIANN and 'plain' GFNN networks has been evaluated across a number of benchmark problems. The results have been compared to each other, as well as with results using a wide variety of network types and algorithms obtained from the literature.

"The proof of the pudding is in the eating" goes the saying. The proof of this work is in the application to various problems and the results obtained. And the proof appears quite positive, for the shunting inhibitory networks were able to achieve good results across a variety of problems. The networks using the generalised shunting neuron had the added advantage of being able to perform well with very small network structures. The GFNN networks used in the benchmark tests all had a maximum of two generalised shunting neurons, some only one, plus an output layer of linear or sigmoid neurons, where needed. A single-neuron was able to achieve 100% correct classification for the Breast Cancer and 3-bit Parity problems for the majority of the test runs. In the final comparison tests, a single GS neuron has been used as the 'network' for the Breast Cancer and Diabetes problems. This is the ultimate in structural simplicity. The overall results obtained compare well with other reported results, in many cases better than those achieved by much more complex networks.

The initial hypothesis was that shunting inhibition allows neurons to produce non-linear decision boundaries, therefore shunting inhibition-based feedforward neural networks can form a new class of powerful networks for classification and regression. From the evidence presented in this thesis, it can be concluded that this hypothesis holds true.

9.3 Future Research Directions

This section discusses possibilities for future directions arising from the work presented in this thesis. A number of research issues pertaining to the work presented remain unexplored:

a) Work can be done on comparing *initialisation schemes* for the GFNNs.

b) *On-line training algorithms* can be developed, which will be particularly useful for large datasets with unbalanced population distributions like the Thyroid problem.

c) The *efficiency of the code* used to implement the GFNN structures and training algorithms can be improved significantly.

Another area of promising future investigation is the *network structure*. From Fig. 9.2, it can be seen that the network 'types' investigated here are distinct subsets of the broader definition of a 'Generalised Feedforward Neural Network'. The grey areas in-between, both literally and figuratively, represent the largely unexplored area of GFNN structures not tested here. It includes several categories of networks:

a) Networks containing mixtures of layers of neurons (GSN, standard shunting neuron, perceptron) not previously tested. For example, networks with both GSN and static shunting neuron layers, or with perceptron layers in between shunting layers.

b) Networks with heterogeneous layers. Layers can contain more than one type of neuron, unlike current implementations where it is assumed that a layer contains only one type of neuron.

c) Networks with layers not fully connected. Some of the inter-layer synaptic weights are removed (fixed at 0), as would happen when using pruning algorithms.

d) Networks with shortcut connections, where there are synaptic connections between non-adjacent layers. Current implementations assume connections only exist between adjacent layers.

The categories listed above are *not* mutually exclusive, but are listed to give a clear picture of the variety of possibilities that can be explored in future work. At the time of writing, the current implementation of GFNNs and their training algorithms is only able to handle networks of type (a).

In the work presented here, the training of networks has been based on adjusting the weights of a fixed neural network structure. In a partial attempt to find 'good' structures, a few structures have been trained, with various combinations of activation functions. As mentioned in Chapter 2, investigating heuristic methods of architecture selection is an active area of research, with researchers combining constructive and pruning algorithms, or using evolutionary computation, which includes genetic algorithms and evolutionary programming.

The research done here has been able to 'broaden the horizons' of shunting inhibition-based neural networks. The expanded framework offered by the GFNN structure would allow for many more possibilities in the dynamic modification of network structures, resulting in networks of the types listed above. A single generalised shunting neuron has been shown to be a viable 'network' in solving problems, thereby providing a good starting point. Alternatively, it is possible to start from a purely excitatory 'MLP-type' network, then go to a shunting inhibition-based network, or back, seamlessly, as GFNNs have both excitatory and inhibitory synapses.

The scope of constructive and pruning algorithms and other 'evolutionary'-type algorithms that aim to find an optimal neural network structure now literally have a whole new dimension opened up. Employing such methods would lead to a myriad of possibilities in terms of network structures that could be used for classification and regression problems.

Appendix A

Derivation of Training Equations for SIANNs

A.1 Introduction

The backpropagation algorithm requires the partial derivatives of the objective (error) function with respect to each of the trainable parameters (synaptic weights) being updated to calculate the gradient. This appendix shows the derivation of the partial differential equations and error sensitivity functions used in the gradient-based training algorithms, as presented in Chapter 4. The next section recaps the SIANN equations and parameter definitions, followed by the definition of the error function. The final section presents the actual derivation of the training equations.

A.2 SIANN Equations and Parameters

The 'standard' SIANN is a feedforward neural network with a hidden layer of shunting neurons and an output layer of linear or sigmoid neurons. For the sake of clarity, the equations describing the operation of the SIANN, defined in Chapter 3, are presented again in Eqs. (4.6) to (4.8) below.

The output of the j^{th} shunting neuron, x_j, is given by

$$x_j = \frac{I_j + b_j}{a_j + f\left(\sum_{i=0}^{m} c_{ji} I_i\right)} \tag{9.4}$$

where I_j is the j^{th} input; a_j is the 'decay term'; b_j is the bias; c_{ji} is the synaptic weight connecting the j^{th} neuron to the i^{th} input; c_{j0} is the bias for the shunting activation function connected to a fixed 'input', $I_0 = 1$; and f is a non-decreasing activation function.

The output of the kth output neuron is given by

$$y_k = g(\sum_{j=0}^{m} w_{kj} x_j) \tag{9.5}$$

where g is the output layer activation function; w_{kj} is the connection weight from j^{th} shunting neuron to the k^{th} output neuron and w_{k0} is the bias of the output neuron connected to a fixed 'input', $x_0 = 1$.

The denominator in (4.6) is defined as the shunting term for the jth neuron, s_j

$$s_j = a_j + f\left(\sum_{i=0}^{m} c_{ji} I_i\right) \tag{9.6}$$

This shunting term is constrained to be always positive, achieved by imposing a lower bound on the parameter a_j during the initialization and training phases.

The parameters to be trained in a standard SIANN, therefore, are the weights and biases of the output neurons (w_{kj}), the decay and bias terms of the shunting neurons (a_j and b_j) and the inhibitory weights of the input signals and shunting bias (c_{ji}). The following sections derive the training equations for these parameters.

A.3 Error Function

The gradient-based training algorithms developed are based on the standard backpropagation algorithm. The network is trained with training pairs ($\mathbf{I}(q)$, $\mathbf{d}(q)$) where $\mathbf{I}(q)$is the input vector and $\mathbf{d}(q)$ is the corresponding desired target value. (Note: Since the network may have multiple output neurons, $\mathbf{d}(q)$is a vector). The difference between the desired and actual output of the network is the error, given by

$$\mathbf{e}(q) = \mathbf{y}(q) - \mathbf{d}(q) \tag{9.7}$$

where $\mathbf{y}(q)$ is output vector for input $\mathbf{I}(q)$.

The training algorithm seeks to minimise the objective function, which is the sum of squares of the error term:

$$E = \tfrac{1}{2}\sum \mathbf{e}(q)^T \mathbf{e}(q) \tag{9.8}$$

To get avoid having to consider the summation, consider the simple case where the parameter updates are performed on a pattern-by-pattern basis. The objective function can then be given by $\hat{E} = \tfrac{1}{2}\mathbf{e}(q)^T\mathbf{e}(q)$ (Haykin, 1999, pp144-147).

A.4 Training Equations

This section gives the actual derivation of the partial differential equations and error sensitivity functions used in the gradient-based training algorithms. These equations were presented in Chapter 4, as equations (4.9) to (4.14), corresponding to the boxed equations below. The change made to any parameter is always in the direction of the negative gradient, in order to minimise the objective function. For the activation function bias terms, w_{k0} and c_{j0}, the corresponding 'inputs', x_0 and I_0, are assumed fixed at 1.

A.4.1 Equation for weight of the k^{th} output neuron, w_{kj} and error sensitivity function, δ_{ok}

The update to the weight, Δw_{kj}, is proportional to the gradient $\partial E/\partial w_{kj}$. Differentiating (9.8) with respect to w_{kj}, and using (9.7), (4.7) and (3.13),

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial e_k(q)} \frac{\partial e_k(q)}{\partial y_k} \frac{\partial y_k}{\partial v_k} \frac{\partial v_k}{\partial w_{kj}} \qquad \text{(Chain Rule)}$$

$$= e_k(q).1.g'(v_k)\, x_j$$

$$\therefore \quad \frac{\partial E}{\partial w_{kj}} = \delta_{ok}\, x_j \qquad (9.9)$$

where

$$\delta_{ok} = e_k(q) g'(v_k) \qquad (9.10)$$

The term δ_{ok} is defined as the error sensitivity function for the k^{th} output.

A.4.2 The error sensitivity function for the jth shunting neuron, δ_j.

The change in the objective function E with respect to the output of each individual shunting neuron x_j, $\partial E/\partial x_j$ is given by

$$\frac{\partial E}{\partial x_j} = \sum_{k=1}^{n} \frac{\partial E}{\partial e_k(q)} \cdot \frac{\partial e_k(q)}{\partial y_k} \cdot \frac{\partial y_k}{\partial v_k} \cdot \frac{\partial v_k}{\partial x_j}$$

$$= \sum_{k=1}^{n} e_k(q).1.g'(v_k).w_{kj}$$

$$\frac{\partial E}{\partial x_j} = \sum_{k=1}^{n} \delta_{ok}.w_{kj}$$

$$\therefore \frac{\partial E}{\partial x_j} = \delta_j \tag{9.11}$$

where we define

$$\delta_j = \sum_{k=1}^{n} \delta_{ok} . w_{kj} \tag{9.12}$$

The term δ_j is the backpropagated error sensitivity function for the jth shunting neuron.

A.4.3 Equation for the decay parameter of the j^{th} shunting neuron, a_j

The same procedure used in the derivation of $\partial E/\partial w_{kj}$ is applied for the gradient $\partial E/\partial a_j$. Differentiating (4.6) with respect to a_j, we get

$$\frac{\partial x_j}{\partial a_j} = \frac{-\left(I_j + b_j\right)}{\left[a_j + f\left(\sum_i c_{ji} I_i\right)\right]^2} \tag{9.13}$$

The denominator in (9.13) contains the shunting term for the jth neuron, s_j, as given in (4.8), therefore

$$\frac{\partial x_j}{\partial a_j} = \frac{-\left(I_j + b_j\right)}{s_j^{\,2}} \tag{9.14}$$

or alternatively, substituting (4.6) and (4.8) into (9.13), we get

$$\frac{\partial x_j}{\partial a_j} = \frac{-x_j}{s_j} \tag{9.15}$$

Using the Chain rule and equations (9.12) and (9.14/9.15),

$$\frac{\partial E}{\partial a_j} = \frac{\partial E}{\partial x_j}\frac{\partial x_j}{\partial a_j}$$

$$= \delta_j \frac{-\left(I_j + b_j\right)}{s_j^{\,2}}$$

$$\therefore \frac{\partial E}{\partial a_j} = -\delta_j \frac{\left(I_j + b_j\right)}{s_j^{\,2}} = -\delta_j \frac{x_j}{s_j} \tag{9.16}$$

A.4.4 Equation for the bias parameter of the j^{th} shunting neuron, b_j

The same procedure is applied for the gradient $\partial E/\partial b_j$. Differentiating (4.6) with respect to b_j, we get

$$\frac{\partial x_j}{\partial b_j} = \frac{1}{a_j + f\left(\sum_i c_{ji} I_i\right)} = \frac{1}{s_j} \tag{9.17}$$

Using the Chain rule and equations (9.12) and (9.17),

$$\frac{\partial E}{\partial b_j} = \frac{\partial E}{\partial x_j} \cdot \frac{\partial x_j}{\partial b_j}$$

$$\therefore \frac{\partial E}{\partial b_j} = \frac{\delta_j}{s_j} \tag{9.18}$$

A.4.5 Equation for the connection weight between input I_i and the j^{th} shunting neuron, c_{ji}

The same procedure used in the derivation of $\partial E/\partial a_j$ is applied for the gradient $\partial E/\partial c_{ji}$.

Let

$$v_j = \sum_i c_{ji} I_i \tag{9.19}$$

and

$$p_j = f\left(\sum_i c_{ji} I_i\right) = f(v_j) \tag{9.20}$$

where f is the activation function of the neuron.

The output of the shunting neuron, x_j, can then be re-written as

$$x_j = \frac{I_j + b_j}{a_j + p_j} \tag{9.21}$$

Differentiating (9.21) with respect to c_{ji} using the Chain rule, we get

$$\frac{\partial x_j}{\partial c_{ji}} = \frac{\partial x_j}{\partial p_j} \frac{\partial p_j}{\partial v_j} \frac{\partial v_j}{\partial c_{ji}}$$

$$= -\frac{I_j + b_j}{\left[a_j + p_j\right]^2} f'(v_j) I_i$$

$$\frac{\partial x_j}{\partial c_{ji}} = -\frac{x_j}{s_j} f'(v_j) I_i \tag{9.22}$$

Using the Chain rule and eqs (9.12) and (9.22),

$$\frac{\partial E}{\partial c_{ji}} = \frac{\partial E}{\partial x_j} \frac{\partial x_j}{\partial c_{ji}}$$

$$= \delta_j \frac{-x_j}{s_j} f'(v_j) I_i$$

$$\therefore \frac{\partial E}{\partial c_{ji}} = -\delta_j \frac{x_j}{s_j} f'(v_j) I_i \tag{9.23}$$

Appendix B

Details of Experimental Results

B.1 Experimental Results for Chapter 4

This section presents tables containing the details of the experimental results obtained in Chapter 4, 'Development of Training Algorithms'.

B.1.1 Mean Error Results

This sub-section presents the mean test error (or test ARV) for all combinations of activation functions and training algorithms obtained using SIANNs, for each of the benchmark tests. The average for each activation function combination (row) and training algorithm (column) are also presented.

Table B.1 Mean test classification error for 3-bit Parity dataset using 3-3-1 SIANNs

Activation Functions		Training Algorithms								Average
Shunt	Out	GDM	GDX	LM	LMAM	OLMAM	DSGDX	ROM	ROM2	
Tnh	Lin	27.50	24.75	8.00	9.00	10.50	0.75	29.50	24.00	*16.75*
Tnh	Lgs	28.75	1.25	6.75	7.00	10.75	6.25	15.50	10.50	*10.84*
Tnh	Tnh	22.50	2.00	8.25	9.00	9.75	43.25	22.25	22.25	*17.41*
Lgs	Lin	53.00	49.75	0.50	21.75	10.25	4.00	40.50	43.25	*27.88*
Lgs	Lgs	49.50	29.00	1.00	21.75	3.25	49.75	22.25	21.25	*24.72*
Lgs	Tnh	51.25	48.25	5.75	30.00	7.75	45.75	39.00	38.00	*33.22*
Exp	Lin	38.00	38.00	0.50	4.25	1.50	1.50	30.50	30.25	*18.06*
Exp	Lgs	48.50	1.25	2.00	4.50	3.25	28.25	12.25	11.00	*13.88*
Exp	Tnh	38.00	3.25	3.00	8.75	4.00	45.75	23.75	23.75	*18.78*
Average		*39.67*	*21.94*	*3.97*	*12.89*	*6.78*	*25.03*	*26.17*	*24.92*	

Table B.2 Mean test classification error for Breast Cancer dataset using 9-9-1 SIANN

Activation Functions		Training Algorithms								Average
Shunt	Out	GDM	GDX	LM	LMAM	OLMAM	DSGDX	ROM	ROM2	
Tnh	Lin	0.53	0.36	0.71	1.18	0.75	0.66	5.71	5.51	*1.93*
Tnh	Lgs	0.36	0.60	1.22	1.29	0.86	0.54	1.40	1.54	*0.98*
Tnh	Tnh	1.69	0.82	0.85	1.14	0.98	2.69	3.33	3.53	*1.88*
Lgs	Lin	0.79	0.98	0.44	0.76	0.49	0.73	5.15	5.75	*1.89*
Lgs	Lgs	0.36	0.20	0.66	1.24	0.62	0.69	1.38	1.30	*0.81*
Lgs	Tnh	1.31	0.84	0.20	0.76	0.37	3.28	3.11	3.18	*1.63*
Exp	Lin	0.52	0.29	0.88	1.95	1.29	0.98	6.29	5.75	*2.24*
Exp	Lgs	0.43	0.45	1.21	1.63	0.76	0.54	1.46	1.55	*1.00*
Exp	Tnh	1.23	0.54	0.79	1.86	1.36	3.25	2.70	3.63	*1.92*
Average		*0.80*	*0.56*	*0.77*	*1.31*	*0.83*	*1.48*	*3.39*	*3.53*	

Table B.3 Mean test classification error for Diabetes dataset using 8-8-1 SIANNs

Activation Functions		Training Algorithms								Average
Shunt	Out	GDM	GDX	LM	LMAM	OLMAM	DSGDX	ROM	ROM2	
Tnh	Lin	20.03	22.14	21.30	22.55	21.34	20.23	26.62	27.68	*22.74*
Tnh	Lgs	22.01	21.03	20.96	21.96	21.10	20.17	22.81	22.83	*21.61*
Tnh	Tnh	19.70	21.16	21.09	21.65	20.52	40.64	27.76	26.81	*24.92*
Lgs	Lin	19.48	22.27	20.61	20.98	20.96	20.02	26.52	26.07	*22.11*
Lgs	Lgs	25.80	21.09	20.88	21.06	20.73	19.82	21.50	21.69	*21.57*
Lgs	Tnh	19.05	22.03	19.88	20.22	20.34	41.24	25.71	24.35	*24.10*
Exp	Lin	19.47	22.07	20.60	24.39	22.41	20.20	29.62	28.83	*23.45*
Exp	Lgs	27.20	21.76	20.58	21.40	21.00	20.27	22.67	22.05	*22.12*
Exp	Tnh	19.28	21.15	20.31	22.37	20.56	39.72	26.09	27.17	*24.58*
Average		*21.34*	*21.63*	*20.69*	*21.84*	*21.00*	*26.92*	*25.48*	*25.28*	

Table B.4 Mean test classification error for Multi-class dataset using 2-2-3 SIANN

Activation Functions		Training Algorithms								Average
Shunt	Out	GDM	GDX	LM	LMAM	OLMAM	DSGDX	ROM	ROM2	
Tnh	Lin	32.48	23.79	9.60	10.91	12.84	23.67	44.87	43.03	*25.15*
Tnh	Lgs	6.72	6.05	7.45	6.76	9.91	33.59	10.47	11.83	*11.60*
Tnh	Tnh	29.51	21.05	9.12	10.08	18.13	32.20	42.07	41.29	*25.43*
Lgs	Lin	33.08	32.25	16.55	15.03	17.88	23.57	46.08	42.59	*28.38*
Lgs	Lgs	8.11	7.05	6.83	6.15	5.81	32.65	13.93	12.99	*11.69*
Lgs	Tnh	33.23	32.76	17.80	15.27	21.37	31.00	41.33	39.27	*29.00*
Exp	Lin	10.23	10.53	6.42	7.87	6.31	9.39	43.79	44.95	*17.44*
Exp	Lgs	5.73	5.47	5.69	6.13	6.79	24.91	7.49	8.33	*8.82*
Exp	Tnh	11.80	9.84	9.27	7.39	7.29	19.80	35.25	38.76	*17.43*
Average		*18.99*	*16.53*	*9.86*	*9.51*	*11.81*	*25.64*	*31.70*	*31.45*	

Table B.5 Mean test ARV for Sunspots dataset using 10-10-1 SIANNs

Activation Functions		Training Algorithms								Average
Shunt	Out	GDM	GDX	LM	LMAM	OLMAM	DSGDX	ROM	ROM2	
Tnh	Lin	0.161	0.174	0.131	0.140	0.128	0.130	1.223	0.877	*0.371*
Tnh	Lgs	1.008	0.987	0.143	0.151	0.132	0.190	0.333	0.334	*0.410*
Tnh	Tnh	0.226	0.201	0.129	0.147	0.129	0.137	0.790	0.742	*0.313*
Lgs	Lin	0.223	0.188	0.112	0.125	0.117	0.119	0.614	0.670	*0.271*
Lgs	Lgs	1.007	1.006	0.143	0.152	0.130	0.194	0.286	0.271	*0.399*
Lgs	Tnh	0.274	0.212	0.126	0.135	0.122	0.131	0.622	0.551	*0.272*
Exp	Lin	0.274	0.191	0.134	0.167	0.143	0.135	0.773	0.912	*0.341*
Exp	Lgs	1.009	1.008	0.146	0.150	0.176	0.195	0.324	0.361	*0.421*
Exp	Tnh	0.314	0.213	0.133	0.151	0.134	0.135	0.762	0.743	*0.323*
Average		*0.500*	*0.464*	*0.133*	*0.146*	*0.135*	*0.152*	*0.636*	*0.607*	

B.1.2 Rankings of Experimental Results

This sub-section presents the Kruskal-Wallis rankings of the mean test errors for all combinations of activation functions and training algorithms using SIANNs, for each of the benchmark tests. Tables B.6 to B.10 contain the ranking of the means presented in the corresponding table in sub-section B.1.1. The sum of rankings for each activation function combination (row) and training algorithm (column) is also presented, along with a relative ranking of the functions and training algorithms based on this total. Table B.11 presents the sum of all ranks over the five different benchmark tests. Table B.12 ranks the sums presented in Table B.11 from 1 to 72, and is used to calculate the overall h statistic, as well as the 'overall' ranking.

Table B.6 Rankings for 3-bit Parity dataset results using SIANNs

Activation Functions		Training Algorithms								Total	Rank
Shunt	Out	GDM	GDX	LM	LMAM	OLMAM	DSGDX	ROM	ROM2		
Tnh	Lin	48	47	24	27.5	31.5	3	52	46	*279*	6
Tnh	Lgs	50	5.5	21	22	33	20	36	31.5	*219*	1
Tnh	Tnh	43	9.5	25	27.5	29	62.5	41	41	*278.5*	5
Lgs	Lin	72	69.5	1.5	38.5	30	15.5	61	62.5	*350.5*	8
Lgs	Lgs	68	51	4	38.5	13	69.5	41	37	*322*	7
Lgs	Tnh	71	66	19	53	23	64.5	60	57.5	*414*	9
Exp	Lin	57.5	57.5	1.5	17	7.5	7.5	55	54	*257.5*	3
Exp	Lgs	67	5.5	9.5	18	13	49	35	34	*231*	2
Exp	Tnh	57.5	13	11	26	15.5	64.5	44.5	44.5	*276.5*	4
Total		*534*	*324.5*	*116.5*	*268*	*195.5*	*356*	*425.5*	*408*		
Rank		8	4	1	3	2	5	7	6		

Table B.7 Rankings for Breast Cancer dataset results using SIANNs

Activation Functions		Training Algorithms								Total	Rank
Shunt	Out	GDM	GDX	LM	LMAM	OLMAM	DSGDX	ROM	ROM2		
Tnh	Lin	13	5	22	39	24	19.5	69	68	*259.5*	4
Tnh	Lgs	5	17	41	44.5	33	15	50	52	*257.5*	3
Tnh	Tnh	55	30	32	38	36	58	64	65	*378*	9
Lgs	Lin	28.5	36	9	26	11	23	67	70.5	*271*	5
Lgs	Lgs	5	1.5	19.5	43	18	21	49	46	*203*	1
Lgs	Tnh	47	31	1.5	26	7	63	60	61	*296.5*	6
Exp	Lin	12	3	34	57	44.5	36	72	70.5	*329*	7
Exp	Lgs	8	10	40	54	26	15	51	53	*257*	2
Exp	Tnh	42	15	28.5	56	48	62	59	66	*376.5*	8
Total		*215.5*	*148.5*	*227.5*	*383.5*	*247.5*	*312.5*	*541*	*552*		
Rank		2	1	3	6	4	5	7	8		

Table B.8 Rankings for Diabetes dataset results using SIANNs

Activation Functions		Training Algorithms								Total	Rank
Shunt	Out	GDM	GDX	LM	LMAM	OLMAM	DSGDX	ROM	ROM2		
Tnh	Lin	9	47	35	51	36	13	62	66	*319*	7
Tnh	Lgs	43	28	24.5	42	32	10	53	54	*286.5*	4
Tnh	Tnh	5	34	30.5	39	17	71	67	63	*326.5*	9
Lgs	Lin	4	48	21	26	24.5	8	61	59	*251.5*	2
Lgs	Lgs	58	30.5	23	29	22	6	38	40	*246.5*	1
Lgs	Tnh	1	44	7	12	16	72	57	55	*264*	3
Exp	Lin	3	46	20	56	50	11	69	68	*323*	8
Exp	Lgs	65	41	19	37	27	14	52	45	*300*	5
Exp	Tnh	2	33	15	49	18	70	60	64	*311*	6
Total		*190*	*351.5*	*195*	*341*	*242.5*	*275*	*519*	*514*		
Rank		1	6	2	5	3	4	8	7		

Table B.9 Rankings for Multi-class dataset results using SIANNs

Activation Functions		Training Algorithms								Total	Rank
Shunt	Out	GDM	GDX	LM	LMAM	OLMAM	DSGDX	ROM	ROM2		
Tnh	Lin	55	49	25	32	35	48	70	68	*382*	7
Tnh	Lgs	10	5	17	11	27	60	30	34	*194*	3
Tnh	Tnh	51	45	22	28	43	53	66	64	*372*	6
Lgs	Lin	58	54	40	38	42	47	72	67	*418*	8
Lgs	Lgs	20	14	13	7	4	56	37	36	*187*	2
Lgs	Tnh	59	57	41	39	46	52	65	63	*422*	9
Exp	Lin	29	31	9	19	8	24	69	71	*260*	4
Exp	Lgs	3	1	2	6	12	50	18	21	*113*	1
Exp	Tnh	33	26	23	16	15	44	61	62	*280*	5
Total		*318*	*282*	*192*	*196*	*232*	*434*	*488*	*486*		
Rank		5	4	1	2	3	6	8	7		

Table B.10 Rankings for Sunspots dataset results using SIANNs

Activation Functions		Training Algorithms								Total	Rank
Shunt	Out	GDM	GDX	LM	LMAM	OLMAM	DSGDX	ROM	ROM2		
Tnh	Lin	32	34	12.5	22	7	10.5	72	64	*254*	3
Tnh	Lgs	69.5	66	24	29.5	14	37	52	53	*345*	8
Tnh	Tnh	45	41	8.5	27	8.5	21	63	59	*273*	4
Lgs	Lin	44	36	1	5	2	3	56	58	*205*	1
Lgs	Lgs	68	67	24	31	10.5	39	49	46	*334.5*	7
Lgs	Tnh	47.5	42	6	19	4	12.5	57	55	*243*	2
Exp	Lin	47.5	38	16.5	33	24	19	62	65	*305*	6
Exp	Lgs	71	69.5	26	28	35	40	51	54	*374.5*	9
Exp	Tnh	50	43	15	29.5	16.5	19	61	60	*294*	5
Total		*474.5*	*436.5*	*133.5*	*224*	*121.5*	*201*	*523*	*514*		
Rank		6	5	2	4	1	3	8	7		

Table B.11 Sum of ranks across all five benchmarks datasets

Activation Functions		Training Algorithms								Total	Rank
Shunt	Out	GDM	GDX	LM	LMAM	OLMAM	DSGDX	ROM	ROM2		
Tnh	Lin	157	182	118.5	171.5	133.5	94	325	312	*1493.5*	5
Tnh	Lgs	177.5	121.5	127.5	149	139	142	221	224.5	*1302*	3
Tnh	Tnh	199	159.5	118	159.5	133.5	265.5	301	292	*1628*	8
Lgs	Lin	206.5	243.5	72.5	133.5	109.5	96.5	317	317	*1496*	6
Lgs	Lgs	219	164	83.5	148.5	67.5	191.5	214	205	*1293*	2
Lgs	Tnh	225.5	240	74.5	149	96	264	299	291.5	*1639.5*	9
Exp	Lin	149	175.5	81	182	134	97.5	327	328.5	*1474.5*	4
Exp	Lgs	214	127	96.5	143	113	168	207	207	*1275.5*	1
Exp	Tnh	184.5	130	92.5	176.5	113	259.5	285.5	296.5	*1538*	7
Total		*1732*	*1543*	*864.5*	*1412.5*	*1039*	*1578.5*	*2496.5*	*2474*		
Rank		6	4	1	3	2	5	8	7		

Table B.12 Rankings for Overall performance across all datasets

Activation Functions		Training Algorithms								Total	Rank
Shunt	Out	GDM	GDX	LM	LMAM	OLMAM	DSGDX	ROM	ROM2		
Tnh	Lin	32	41.5	16	37	22	7	70	67	*292.5*	6
Tnh	Lgs	40	17	19	30	25	26	53	54	*264*	3
Tnh	Tnh	45	33.5	15	33.5	22	60	66	63	*338*	8
Lgs	Lin	47	57	2	22	12	9.5	68.5	68.5	*286.5*	4
Lgs	Lgs	52	35	5	28	1	44	50.5	46	*261.5*	2
Lgs	Tnh	55	56	3	30	8	59	65	62	*338*	8
Exp	Lin	30	38	4	41.5	24	11	71	72	*291.5*	5
Exp	Lgs	50.5	18	9.5	27	13.5	36	48.5	48.5	*251.5*	1
Exp	Tnh	43	20	6	39	13.5	58	61	64	*304.5*	7
Total		*394.5*	*316*	*79.5*	*288*	*141*	*310.5*	*553.5*	*545*		
Rank		6	5	1	3	2	4	8	7		

B.2 Experimental Results for Chapter 5

This section presents tables containing the details of the experimental results obtained in Chapter 5, 'The Quadratic Neural Network Algorithm'.

B.2.1 Mean Error Results

This sub-section presents details of experimental results obtained. Tables B.13 to B.17 show the mean test error for all combinations of activation functions and training algorithms, for each of the benchmark tests, using SIANNs trained on QNN algorithm variants. The average for each activation function combination (row) and training algorithm (column) are also presented. Tables B.18 and B.19 show results obtained when varying the step-size, *d*, for the standard QNN algorithm.

Table B.13 Mean test classification error for Breast Cancer dataset using SIANNs trained with QNN algorithm variants

Activation Functions		Training Algorithms								Average
Shunt	Out	QNN	QNN3	QNN5	QNN6	QNN9	QNN-C2	GDX	LM	
Tnh	Lin	0.46	0.54	0.64	0.56	0.50	0.45	0.36	0.71	*0.53*
Tnh	Lgs	0.36	0.33	0.31	0.41	0.41	0.29	0.60	1.22	*0.49*
Tnh	Tnh	3.51	3.55	4.08	3.73	3.45	3.49	0.82	0.85	*2.94*
Lgs	Lin	0.55	0.60	0.86	0.55	0.34	0.54	0.98	0.44	*0.61*
Lgs	Lgs	0.40	0.34	0.24	0.36	0.32	0.35	0.20	0.66	*0.36*
Lgs	Tnh	0.47	0.50	0.78	0.62	0.70	0.46	0.84	0.20	*0.57*
Exp	Lin	0.66	0.63	0.51	0.63	0.56	0.66	0.29	0.88	*0.60*
Exp	Lgs	0.26	0.35	0.27	0.56	0.40	0.31	0.45	1.21	*0.48*
Exp	Tnh	1.90	1.86	2.31	1.86	0.82	2.06	0.54	0.79	*1.52*
Average		*0.95*	*0.97*	*1.11*	*1.03*	*0.83*	*0.96*	*0.56*	*0.77*	

Table B.14 Mean test classification error for Diabetes dataset using SIANNs trained with QNN algorithm variants

Activation Functions		Training Algorithms								Average
Shunt	Out	QNN	QNN3	QNN5	QNN6	QNN9	QNN-C2	GDX	LM	
Tnh	Lin	20.72	20.71	20.10	21.24	20.51	20.54	22.14	21.30	*20.91*
Tnh	Lgs	20.65	20.56	23.48	20.80	20.69	20.67	21.03	20.96	*21.11*
Tnh	Tnh	20.08	20.01	20.15	20.66	20.17	20.18	21.16	21.09	*20.44*
Lgs	Lin	20.09	20.30	19.96	20.35	20.11	20.16	22.27	20.61	*20.48*
Lgs	Lgs	20.74	20.73	25.21	20.66	20.61	20.60	21.09	20.88	*21.32*
Lgs	Tnh	19.88	19.95	19.91	20.02	19.57	19.69	22.03	19.88	*20.12*
Exp	Lin	20.79	20.62	20.08	20.88	20.59	20.79	22.07	20.60	*20.80*
Exp	Lgs	20.69	20.77	23.68	21.09	20.54	20.79	21.76	20.58	*21.24*
Exp	Tnh	19.92	19.80	20.23	20.22	19.96	20.05	21.15	20.31	*20.21*
Average		*20.40*	*20.38*	*21.42*	*20.66*	*20.31*	*20.39*	*21.63*	*20.69*	

Table B.15 Mean test classification error for 3-bit Parity dataset using SIANNs trained with QNN algorithm variants

Activation Functions		Training Algorithms								Average
Shunt	Out	QNN	QNN3	QNN5	QNN6	QNN9	QNN-C2	GDX	LM	
Tnh	Lin	11.50	9.50	31.00	21.25	10.25	11.75	24.75	8.00	*16.00*
Tnh	Lgs	8.50	9.00	36.75	20.25	12.50	10.25	1.25	6.75	*13.16*
Tnh	Tnh	10.00	12.50	33.00	19.00	10.50	10.25	2.00	8.25	*13.19*
Lgs	Lin	5.00	6.00	47.00	28.50	6.00	4.25	49.75	0.50	*18.38*
Lgs	Lgs	5.25	4.75	47.00	30.50	5.75	3.50	29.00	1.00	*15.84*
Lgs	Tnh	7.25	6.50	46.50	30.25	8.75	10.25	48.25	5.75	*20.44*
Exp	Lin	9.00	7.75	36.50	15.75	6.75	9.50	38.00	0.50	*15.47*
Exp	Lgs	9.25	8.00	32.00	20.00	7.00	9.25	1.25	2.00	*11.09*
Exp	Tnh	9.00	9.50	33.75	15.75	8.50	9.00	3.25	3.00	*11.47*
Average		*8.31*	*8.17*	*38.17*	*22.36*	*8.44*	*8.67*	*21.94*	*3.97*	

Table B.16 Mean test classification error for Multi-class dataset using SIANNs trained with QNN algorithm variants

Activation Functions		Training Algorithms								Average
Shunt	Out	QNN	QNN3	QNN5	QNN6	QNN9	QNN-C2	GDX	LM	
Tnh	Lin	8.87	8.36	23.55	8.79	8.85	8.72	23.79	9.60	*12.57*
Tnh	Lgs	6.11	6.83	8.03	6.24	6.88	6.80	6.05	7.45	*6.80*
Tnh	Tnh	8.96	8.80	25.95	9.41	10.68	8.60	21.05	9.12	*12.82*
Lgs	Lin	9.57	9.33	30.88	19.29	11.85	10.32	32.25	16.55	*17.51*
Lgs	Lgs	6.60	6.51	7.05	6.28	6.71	6.49	7.05	6.83	*6.69*
Lgs	Tnh	11.57	11.17	32.09	16.21	12.32	9.40	32.76	17.80	*17.92*
Exp	Lin	7.40	7.00	14.57	7.05	7.89	7.47	10.53	6.42	*8.54*
Exp	Lgs	6.05	5.79	7.87	5.72	5.83	5.83	5.47	5.69	*6.03*
Exp	Tnh	7.03	6.47	15.29	7.00	6.73	6.91	9.84	9.27	*8.57*
Average		*8.02*	*7.81*	*18.36*	*9.55*	*8.64*	*7.84*	*16.53*	*9.86*	

Table B.17 Mean test ARV for Sunspots dataset using SIANNs trained with QNN algorithm variants

Activation Functions		Training Algorithms								Average
Shunt	Out	QNN	QNN3	QNN5	QNN6	QNN9	QNN-C2	GDX	LM	
Tnh	Lin	0.132	0.133	0.202	0.124	0.133	0.134	0.174	0.131	*0.145*
Tnh	Lgs	0.138	0.136	0.506	0.116	0.135	0.138	0.987	0.143	*0.287*
Tnh	Tnh	0.144	0.147	0.269	0.136	0.149	0.144	0.201	0.129	*0.165*
Lgs	Lin	0.133	0.131	0.315	0.105	0.134	0.131	0.188	0.112	*0.156*
Lgs	Lgs	0.127	0.126	0.604	0.100	0.133	0.127	1.006	0.143	*0.296*
Lgs	Tnh	0.149	0.148	0.336	0.134	0.149	0.147	0.212	0.126	*0.175*
Exp	Lin	0.134	0.135	0.317	0.127	0.135	0.134	0.191	0.134	*0.163*
Exp	Lgs	0.145	0.144	0.589	0.150	0.139	0.143	1.008	0.146	*0.308*
Exp	Tnh	0.148	0.148	0.320	0.138	0.147	0.147	0.213	0.133	*0.174*
Average		*0.139*	*0.139*	*0.384*	*0.126*	*0.139*	*0.138*	*0.464*	*0.133*	

Table B.18 Results for QNN algorithm with different d values applied to Wisconsin Breast Cancer dataset

Discrete step-size, d	Activation functions		Performance (% of runs)			Avg Epochs			Test Error				Mean Train time (s)
	Sh	Out	→ goal	0% err	20% <	All runs	→ goal	0% Errors	Best (%)	Mean (%)	95% CI	Median (%)	
0.01	Exp	Lgs	0	16	50	108	*	100	0.00	0.63	± 0.16	0.56	14.4
0.02	Exp	Lgs	0	17	50	82	*	63	0.00	0.66	± 0.18	0.56	12.7
0.05	Exp	Lgs	0	20	50	71	*	60	0.00	0.70	± 0.22	0.56	15.5
0.1	Exp	Lgs	0	24	50	69	*	60	0.00	0.54	± 0.18	0.56	15.5
0.2	Exp	Lgs	0	34	50	70	*	64	0.00	0.26	± 0.13	0.00	15.7
0.5	Exp	Lgs	0	35	50	72	*	72	0.00	0.28	± 0.11	0.00	16.3
1.0	Exp	Lgs	0	30	50	86	*	87	0.00	0.29	± 0.12	0.00	19.6
2.0	Exp	Lgs	0	0	9	42	*	0	6.78	52.01	± 8.77	64.69	12.1

Table B.19 Results for QNN algorithm with different d values applied Pima Indians Diabetes dataset

Discrete step-size, d	Activation functions		Performance (% of runs)			Avg Epochs			Test Error				Mean Train time (s)
	Sh	Out	→ goal	0% err	20% <	All runs	→ goal	0% Errors	Best (%)	Mean (%)	95% CI	Median (%)	
0.01	Lgs	Tnh	0	0	11	531	*	*	17.71	21.05	± 0.41	20.83	100.5
0.02	Lgs	Tnh	0	0	12	292	*	*	18.23	20.73	± 0.35	20.83	52.8
0.05	Lgs	Tnh	0	0	14	153	*	*	17.71	20.82	± 0.35	20.83	35.9
0.1	Lgs	Tnh	0	0	23	267	*	*	18.75	20.22	± 0.24	20.31	65.0
0.2	Lgs	Tnh	0	0	26	311	*	*	17.71	19.88	± 0.24	19.79	75.8
0.5	Lgs	Tnh	0	0	39	353	*	*	18.23	19.79	± 0.34	19.27	86.6
1.0	Lgs	Tnh	0	0	39	477	*	*	17.71	19.43	± 0.35	19.27	119.7
2.0	Lgs	Tnh	0	0	0	1	*	*	33.33	41.37	± 2.91	36.46	2.0

B.2.2 Rankings of Experimental Results

This subsection presents the Kruskal-Wallis rankings of the mean test errors for all combinations of activation functions and training algorithms, for each of the benchmark tests using SIANNs trained on QNN algorithm variants. Tables B.20 to B.24 contain the ranking of the means presented in the corresponding table in B.2.1. The sum of rankings for each activation function combination (row) and training algorithm (column) is also presented, along with a relative ranking of the functions and training algorithms based on this total. Table B.25 presents the sum of all ranks over the five different benchmark tests. Table B.26 ranks the sums presented in Table B.25 from 1 to 72, and is used to calculate the overall h statistic, as well as the 'overall' ranking.

Table B.20 Rankings for Breast Cancer dataset results using QNN algorithm

Activation Functions		Training Algorithms								Total	Rank
Shunt	Out	QNN	QNN3	QNN5	QNN6	QNN9	QNN-C2	GDX	LM		
Tnh	Lin	26.5	33	45	38	29.5	24.5	17	50	*263.5*	4
Tnh	Lgs	17	11	8.5	21.5	21.5	6.5	40.5	61	*187.5*	3
Tnh	Tnh	69	70	72	71	67	68	53.5	56	*526.5*	9
Lgs	Lin	35.5	40.5	57	35.5	12.5	33	59	23	*296*	6
Lgs	Lgs	19.5	12.5	3	17	10	14.5	1.5	47	*125*	1
Lgs	Tnh	28	29.5	51	42	49	26.5	55	1.5	*282.5*	5
Exp	Lin	47	43.5	31	43.5	38	47	6.5	58	*314.5*	7
Exp	Lgs	4	14.5	5	38	19.5	8.5	24.5	60	*174*	2
Exp	Tnh	64	62.5	66	62.5	53.5	65	33	52	*458.5*	8
Total		*310.5*	*317*	*338.5*	*369*	*300.5*	*293.5*	*290.5*	*408.5*		
Rank		4	5	6	7	3	2	1	8		

Table B.21 Rankings for Diabetes dataset results using QNN algorithm

Activation Functions		Training Algorithms								Total	Rank
Shunt	Out	QNN	QNN3	QNN5	QNN6	QNN9	QNN-C2	GDX	LM		
Tnh	Lin	46	45	17	63	28	29.5	68	64	*360.5*	6
Tnh	Lgs	39	31	70	53	43.5	42	57	56	*391.5*	7
Tnh	Tnh	14.5	11	19	40.5	21	22	62	59	*249*	4
Lgs	Lin	16	25	9.5	27	18	20	69	36.5	*221*	3
Lgs	Lgs	48	47	72	40.5	36.5	34.5	59	54.5	*392*	8
Lgs	Tnh	4.5	8	6	12	1	2	66	4.5	*104*	1
Exp	Lin	51	38	14.5	54.5	33	51	67	34.5	*343.5*	5
Exp	Lgs	43.5	49	71	59	29.5	51	65	32	*400*	9
Exp	Tnh	7	3	24	23	9.5	13	61	26	*166.5*	2
Total		*269.5*	*257*	*303*	*372.5*	*220*	*265*	*574*	*367*		
Rank		4	2	5	7	1	3	8	6		

Table B.22 Rankings for 3-bit Parity dataset results using QNN algorithm

Activation Functions		Training Algorithms								Total	Rank
Shunt	Out	QNN	QNN3	QNN5	QNN6	QNN9	QNN-C2	GDX	LM		
Tnh	Lin	46	38	61	55	42.5	47	56	25.5	*371*	9
Tnh	Lgs	28.5	32.5	66	54	48.5	42.5	4.5	20.5	*297*	5
Tnh	Tnh	40	48.5	63	52	45	42.5	6.5	27	*324.5*	7
Lgs	Lin	13	17.5	69.5	57	17.5	11	72	1.5	*259*	3
Lgs	Lgs	14	12	69.5	60	15.5	10	68	3	*242*	1
Lgs	Tnh	23	19	68	59	30	42.5	71	15.5	*328*	8
Exp	Lin	32.5	24	65	50.5	20.5	38	67	1.5	*299*	6
Exp	Lgs	35.5	25.5	62	53	22	35.5	4.5	6.5	*244.5*	2
Exp	Tnh	32.5	38	64	55	28.5	32.5	9	8	*263*	4
Total		*265*	*255*	*588*	*491*	*270*	*301.5*	*348.5*	*109*		
Rank		3	2	8	7	4	5	6	1		

Table B.23 Rankings for Multi-class dataset results using QNN algorithm

Activation Functions		Training Algorithms								Total	Rank
Shunt	Out	QNN	QNN3	QNN5	QNN6	QNN9	QNN-C2	GDX	LM		
Tnh	Lin	42	36	66	39	41	38	67	50	*379*	6
Tnh	Lgs	9	20.5	35	10	22	19	7.5	31	*154*	3
Tnh	Tnh	43	40	68	48	54	37	65	44	*399*	7
Lgs	Lin	49	46	69	64	57	52	71	62	*470*	8
Lgs	Lgs	16	15	28	11	17	14	28	20.5	*149.5*	2
Lgs	Tnh	56	55	70	61	58	47	72	63	*482*	9
Exp	Lin	30	24.5	59	28	34	32	53	12	*272.5*	5
Exp	Lgs	7.5	4	33	3	5.5	5.5	1	2	*61.5*	1
Exp	Tnh	26	13	60	24.5	18	23	51	45	*260.5*	4
Total		*278.5*	*254*	*488*	*288.5*	*306.5*	*267.5*	*415.5*	*329.5*		
Rank		3	1	8	4	5	2	7	6		

Table B.24 Rankings for Sunspots dataset results using QNN algorithm

Activation Functions		Training Algorithms								Total	Rank
Shunt	Out	QNN	QNN3	QNN5	QNN6	QNN9	QNN-C2	GDX	LM		
Tnh	Lin	15	18	59	5	18	23.5	55	13	*206.5*	2
Tnh	Lgs	33	30.5	67	4	28	33	70	37	*302.5*	5
Tnh	Tnh	40	45.5	62	30.5	52	40	58	11	*339*	6
Lgs	Lin	18	13	63	2	23.5	13	56	3	*191.5*	1
Lgs	Lgs	9	6.5	69	1	18	9	71	37	*220.5*	3
Lgs	Tnh	52	49	66	23.5	52	45.5	60	6.5	*354.5*	7
Exp	Lin	23.5	28	64	9	28	23.5	57	23.5	*256.5*	4
Exp	Lgs	42	40	68	54	35	37	72	43	*391*	9
Exp	Tnh	49	49	65	33	45.5	45.5	61	18	*366*	8
Total		*281.5*	*279.5*	*583*	*162*	*300*	*270*	*560*	*192*		
Rank		5	4	8	1	6	3	7	2		

Table B.25 Sum of ranks across all five benchmarks datasets using QNN algorithm

Activation Functions		Training Algorithms								Total	Rank
Shunt	Out	QNN	QNN3	QNN5	QNN6	QNN9	QNN-C2	GDX	LM		
Tnh	Lin	175.5	170	248	200	159	162.5	263	202.5	*1580.5*	8
Tnh	Lgs	126.5	125.5	246.5	142.5	163.5	143	179.5	205.5	*1332.5*	3
Tnh	Tnh	206.5	215	284	242	239	209.5	245	197	*1838*	9
Lgs	Lin	131.5	142	268	185.5	128.5	129	327	126	*1437.5*	4
Lgs	Lgs	106.5	93	241.5	129.5	97	82	217.5	162	*1129*	1
Lgs	Tnh	163.5	160.5	261	197.5	190	163.5	324	91	*1551*	7
Exp	Lin	184	158	233.5	185.5	153.5	191.5	250.5	129.5	*1486*	5
Exp	Lgs	132.5	133	239	207	111.5	137.5	167	143.5	*1271*	2
Exp	Tnh	178.5	165.5	279	193.5	155	179	215	149	*1514.5*	6
Total		*1405*	*1362.5*	*2300.5*	*1683*	*1397*	*1397.5*	*2188.5*	*1406*		
Rank		4	1	8	6	2	3	7	5		

Table B.26 Rankings for Overall performance across all datasets results using QNN algorithm

Activation Functions		Training Algorithms								Total	Rank
Shunt	Out	QNN	QNN3	QNN5	QNN6	QNN9	QNN-C2	GDX	LM		
Tnh	Lin	36	35	64	48	26	29	67	49	*354*	8
Tnh	Lgs	9	7	63	19	31	20	39	50	*238*	3
Tnh	Tnh	51	54.5	70	61	58.5	53	62	46	*456*	9
Lgs	Lin	14	18	68	41.5	10	11	72	8	*242.5*	4
Lgs	Lgs	5	3	60	12.5	4	1	56	28	*169.5*	1
Lgs	Tnh	31	27	66	47	43	31	71	2	*318*	6
Exp	Lin	40	25	57	41.5	23	44	65	12.5	*308*	5
Exp	Lgs	15	16	58.5	52	6	17	34	21	*219.5*	2
Exp	Tnh	37	33	69	45	24	38	54.5	22	*322.5*	7
Total		*238*	*218.5*	*575.5*	*367.5*	*225.5*	*244*	*520.5*	*238.5*		
Rank		3	1	8	6	2	5	7	4		

B.3 Experimental Results for Chapter 6

This section presents the details of the experimental results obtained in Chapter 6, 'Further Development of Shunting Inhibitory Artificial Neural Networks'. The mean test error for each of the benchmark tests, obtained using Enhanced SIANNs, for all combinations of activation functions and training algorithms, is presented in Tables B.27 to B.31. The average for each activation function combination (row) and training algorithm (column) are also presented.

Table B.27 Mean test classification error for Wisconsin Breast Cancer dataset using Enhanced SIANNs

Activation Functions		Training Algorithms / Structure									Avg.
		Reduced 9-4-1			Expanded 9-4-2-1			Standard 9-9-1			
Sh	Out	GDX	LM	DSGDX	GDX	LM	DSGDX	GDX	LM	DSGDX	
Tnh	Lin	0.75	0.76	0.52	0.90	0.69	0.63	0.36	0.71	0.66	*0.66*
Tnh	Lgs	0.88	0.90	1.05	1.14	1.21	1.10	0.60	1.22	0.54	*0.96*
Tnh	Tnh	0.86	0.35	5.81	1.08	0.71	5.92	0.82	0.85	2.69	*2.12*
Lgs	Lin	0.98	0.52	0.47	1.11	0.58	0.62	0.98	0.44	0.73	*0.71*
Lgs	Lgs	0.94	0.84	0.58	1.53	0.77	1.22	0.20	0.66	0.69	*0.83*
Lgs	Tnh	1.22	0.50	6.70	2.15	0.29	9.22	0.84	0.20	3.28	*2.71*
Exp	Lin	0.55	0.23	0.26	0.51	0.31	0.47	0.29	0.88	0.98	*0.50*
Exp	Lgs	0.81	0.79	0.84	2.27	0.90	0.96	0.45	1.21	0.54	*0.97*
Exp	Tnh	0.70	0.61	7.62	1.01	0.41	12.44	0.54	0.79	3.25	*3.04*
Average		*0.85*	*0.61*	*2.65*	*1.30*	*0.65*	*3.62*	*0.56*	*0.77*	*1.48*	

Table B.28 Mean test classification error for Pima Indians Diabetes dataset using Enhanced SIANNs

Activation Functions		Training Algorithms / Structure									Avg.
		Reduced 8-3-1			Expanded 8-3-2-1			Standard 8-8-1			
Sh	Out	GDX	LM	DSGDX	GDX	LM	DSGDX	GDX	LM	DSGDX	
Tnh	Lin	21.77	21.30	20.89	22.26	21.36	20.59	22.14	21.30	20.23	*21.32*
Tnh	Lgs	21.01	20.64	21.23	22.86	21.14	21.50	21.03	20.96	20.17	*21.17*
Tnh	Tnh	20.96	20.53	43.48	21.55	20.81	45.66	21.16	21.09	40.64	*28.43*
Lgs	Lin	22.84	21.22	20.96	23.57	21.15	20.58	22.27	20.61	20.02	*21.47*
Lgs	Lgs	22.05	20.76	21.13	25.55	20.67	21.54	21.09	20.88	19.82	*21.50*
Lgs	Tnh	21.94	20.35	41.96	22.47	20.76	45.31	22.03	19.88	41.24	*28.44*
Exp	Lin	21.78	21.24	20.62	25.56	20.95	20.72	22.07	20.60	20.20	*21.53*
Exp	Lgs	21.38	20.52	21.23	27.61	20.90	21.46	21.76	20.58	20.27	*21.75*
Exp	Tnh	21.34	20.18	42.26	25.95	20.43	44.34	21.15	20.31	39.72	*28.41*
Average		*21.67*	*20.75*	*28.20*	*24.15*	*20.91*	*29.08*	*21.63*	*20.69*	*26.92*	

Table B.29 Mean test error for 3-bit Parity dataset using Enhanced SIANNs

Activation Functions		Training Algorithms / Structure									Avg.
		Reduced 3-2-1			Expanded 3-4-1			Standard 3-3-1			
Sh	Out	GDX	LM	DSGDX	GDX	LM	DSGDX	GDX	LM	DSGDX	
Tnh	Lin	19.25	19.75	11.50	17.00	2.00	0.00	24.75	8.00	0.75	*11.44*
Tnh	Lgs	12.50	17.75	30.00	0.50	1.00	3.75	1.25	6.75	6.25	*8.86*
Tnh	Tnh	8.50	23.75	49.00	0.75	3.50	35.25	2.00	8.25	43.25	*19.36*
Lgs	Lin	50.00	9.75	14.25	40.00	0.00	0.00	49.75	0.50	4.00	*18.69*
Lgs	Lgs	39.50	10.25	49.25	21.25	0.25	1.50	29.00	1.00	49.75	*22.42*
Lgs	Tnh	49.50	8.75	50.00	26.50	4.50	28.50	48.25	5.75	45.75	*29.72*
Exp	Lin	43.50	15.50	19.00	42.25	0.25	0.00	38.00	0.50	1.50	*17.83*
Exp	Lgs	12.50	17.00	41.25	0.50	0.25	2.75	1.25	2.00	28.25	*11.75*
Exp	Tnh	13.75	11.75	49.75	1.75	0.00	40.00	3.25	3.00	45.75	*18.78*
Average		*27.67*	*14.92*	*34.89*	*16.72*	*1.31*	*12.42*	*21.94*	*3.97*	*25.03*	

Table B.30 Mean test error for Multi-Class dataset using Enhanced SIANNs

Activation Functions		Training Algorithms / Structure									Avg.
		Reduced 2-1-3			Expanded 2-3-3			Standard 2-2-3			
Sh	Out	GDX	LM	DSGDX	GDX	LM	DSGDX	GDX	LM	DSGDX	
Tnh	Lin	34.64	34.99	34.67	7.45	6.64	8.39	23.79	9.60	23.67	*20.43*
Tnh	Lgs	8.21	7.13	34.67	5.51	9.69	7.80	6.05	7.45	33.59	*13.34*
Tnh	Tnh	34.96	35.41	34.67	6.32	6.33	9.24	21.05	9.12	32.20	*21.03*
Lgs	Lin	34.33	34.93	34.67	11.04	6.11	5.96	32.25	16.55	23.57	*22.16*
Lgs	Lgs	6.71	7.77	34.67	5.64	7.72	9.09	7.05	6.83	32.65	*13.13*
Lgs	Tnh	34.51	35.24	34.67	9.49	5.60	9.53	32.76	17.80	31.00	*23.40*
Exp	Lin	34.36	35.36	34.67	5.52	5.79	5.57	10.53	6.42	9.39	*16.40*
Exp	Lgs	5.89	6.69	34.67	5.39	6.24	6.51	5.47	5.69	24.91	*11.27*
Exp	Tnh	35.27	35.93	35.33	5.00	8.15	6.56	9.84	9.27	19.80	*18.35*
Average		*25.43*	*25.94*	*34.74*	*6.82*	*6.92*	*7.63*	*16.53*	*9.86*	*25.64*	

Table B.31 Mean test ARV for Sunspot dataset using Enhanced SIANNs

Activation Functions		Training Algorithms / Structure									Avg.
		Reduced 10-5-1			Expanded 10-5-2-1			Standard 10-10-1			
Sh	Out	GDX	LM	DSGDX	GDX	LM	DSGDX	GDX	LM	DSGDX	
Tnh	Lin	0.220	0.134	0.123	0.536	0.132	0.148	0.161	0.119	0.126	*0.189*
Tnh	Lgs	1.030	0.256	1.380	1.020	0.160	1.340	0.970	0.139	1.310	*0.845*
Tnh	Tnh	0.213	0.195	0.151	0.515	0.137	0.223	0.185	0.117	0.133	*0.208*
Lgs	Lin	0.306	0.147	0.124	0.858	0.127	0.144	0.169	0.108	0.121	*0.234*
Lgs	Lgs	1.020	0.167	1.350	1.020	0.133	1.340	0.976	0.116	1.330	*0.828*
Lgs	Tnh	0.317	0.127	0.153	0.888	0.147	0.322	0.200	0.116	0.129	*0.267*
Exp	Lin	0.271	0.142	0.133	1.000	0.134	0.133	0.175	0.135	0.131	*0.250*
Exp	Lgs	1.020	0.161	1.350	1.020	0.155	1.350	0.992	0.140	1.350	*0.838*
Exp	Tnh	0.303	0.132	0.147	1.030	0.134	0.211	0.194	0.133	0.133	*0.269*
Average		*0.522*	*0.162*	*0.546*	*0.876*	*0.140*	*0.579*	*0.447*	*0.125*	*0.529*	

B.4 Experimental Results for Chapter 7

This section presents the experimental results obtained in Chapter 7, 'A Generalised Feedforward Neural Network Architecture'. For each benchmark test, the mean test error, obtained using GFNNs, for all combinations of activation functions and training algorithms are presented in Tables B.32 to B.36. Tables B.37 to B.40 give the detailed results for the various values of s_{lim} with the different benchmarks tests.

Table B.32 Mean test error for Wisconsin Breast Cancer dataset using GFNNs.

Activation Functions		Training Algorithms / Structure											Avg.
		G 9-1		GP 9-2-1									
Output		None		Linear			Log sigmoid (Lgs)			Tan sigmoid (Tnh)			
GSN		GDX	LM	GDX	LM	DS-GDX	GDX	LM	DS-GDX	GDX	LM	DS-GDX	
Num	Den												
Lin	Lgs	0.16	0.44	0.37	0.46	0.34	0.32	0.75	0.73	0.82	0.62	4.61	*0.87*
Lgs	Lgs	0.53	0.54	1.50	2.25	0.82	5.67	3.55	1.29	1.91	3.51	7.67	*2.66*
Tnh	Lgs	0.35	1.36	0.96	0.89	1.06	1.19	1.06	0.63	0.29	0.58	3.15	*1.05*
Exp	Lgs	4.61	0.66	2.18	0.58	0.61	0.87	1.03	3.11	0.38	0.79	18.05	*2.99*
Lin	Tnh	0.47	0.47	0.40	0.49	0.46	1.08	1.28	0.86	0.81	1.12	4.35	*1.07*
Lgs	Tnh	1.05	1.14	1.41	1.41	0.53	1.80	3.47	1.01	1.51	2.09	10.55	*2.36*
Tnh	Tnh	0.60	1.25	0.36	0.73	0.62	1.98	1.21	0.86	0.46	1.03	2.75	*1.08*
Exp	Tnh	3.80	0.55	1.68	0.60	0.63	0.73	1.63	3.20	0.38	0.66	14.50	*2.58*
Lin	Exp	0.50	0.58	0.31	0.87	0.36	0.37	1.12	0.88	0.19	0.77	2.93	*0.81*
Lgs	Exp	0.15	1.82	0.61	1.51	0.28	4.15	1.05	1.16	0.71	1.49	8.29	*1.93*
Tnh	Exp	0.56	0.73	0.75	*0.45*	0.68	0.96	1.15	0.79	0.27	0.70	2.40	*0.86*
Exp	Exp	3.03	0.66	0.92	0.56	0.45	0.85	1.27	3.15	0.59	0.49	15.62	*2.51*
Average		*1.32*	*0.85*	*0.95*	*0.98*	*0.57*	*1.66*	*1.55*	*1.47*	*0.69*	*1.15*	*7.91*	

Table B.33 Mean test error for Pima Indians Diabetes dataset using GFNNs

Activation Functions		Training Algorithms / Structure											Avg.
		G 8-1		GP 8-2-1									
Output		None		Linear			Log sigmoid (Lgs)			Tan sigmoid (Tnh)			
GSN		GDX	LM	GDX	LM	DS-GDX	GDX	LM	DS-GDX	GDX	LM	DS-GDX	
Num	Den												
Lin	Lgs	21.43	23.20	22.01	21.53	19.94	21.04	20.82	24.05	22.53	20.68	42.73	*23.63*
Lgs	Lgs	20.70	23.30	29.72	22.83	21.26	30.87	23.70	27.17	30.23	22.41	44.03	*26.93*
Tnh	Lgs	20.76	20.56	21.05	20.61	20.21	21.45	21.35	26.78	20.66	20.47	43.68	*23.42*
Exp	Lgs	23.25	21.50	23.53	21.20	21.05	21.08	20.36	26.95	22.75	20.88	46.99	*24.50*
Lin	Tnh	22.05	22.31	22.05	21.72	20.60	20.99	20.89	25.18	21.93	21.07	41.59	*23.67*
Lgs	Tnh	21.55	23.76	23.36	21.57	21.27	26.54	23.40	27.65	23.32	21.59	45.28	*25.39*
Tnh	Tnh	20.83	21.53	20.79	21.32	20.05	20.69	21.82	27.83	20.58	21.06	43.26	*23.61*
Exp	Tnh	23.13	21.52	23.29	21.66	20.95	20.85	20.86	27.42	22.72	20.51	47.52	*24.58*
Lin	Exp	22.15	22.46	21.70	21.40	20.54	21.23	21.33	26.05	21.27	20.72	42.17	*23.73*
Lgs	Exp	21.03	21.92	26.78	21.35	20.73	27.66	20.94	26.83	27.84	20.52	43.66	*25.39*
Tnh	Exp	20.58	21.45	20.79	21.24	20.00	21.49	21.98	26.89	20.67	20.55	43.12	*23.52*
Exp	Exp	22.81	21.49	23.15	21.16	21.02	21.06	20.85	26.27	22.47	21.11	45.92	*24.30*
Average		*21.69*	*22.08*	*23.19*	*21.47*	*20.64*	*22.91*	*21.53*	*26.59*	*23.08*	*20.96*	*44.16*	

Table B.34 Mean test classification error for 3-bit Parity dataset using GFNNs

Activation Functions		Training Algorithms / Structure											Avg.
		G 3-1		GP 3-2-1									
Output		None		Linear			Log sigmoid (Lgs)			Tan sigmoid (Tnh)			
GSN		GDX	LM	GDX	LM	DS-GDX	GDX	LM	DS-GDX	GDX	LM	DS-GDX	
Num	Den												
Lin	Lgs	16.00	11.25	50.00	0.00	2.75	22.50	5.50	41.75	30.50	3.00	48.75	*21.09*
Lgs	Lgs	19.25	14.25	35.75	22.25	16.75	20.00	26.25	27.25	22.50	22.00	45.00	*24.66*
Tnh	Lgs	31.25	7.50	28.25	29.25	21.25	24.75	27.00	22.00	25.50	23.50	46.25	*27.95*
Exp	Lgs	15.75	11.00	39.00	3.00	6.75	7.75	7.75	15.25	15.50	5.50	44.50	*15.61*
Lin	Tnh	16.00	17.50	43.50	0.75	4.00	3.75	10.00	12.25	4.75	3.75	47.25	*14.86*
Lgs	Tnh	26.50	16.75	14.25	20.25	5.50	20.00	25.00	12.50	13.75	23.50	44.00	*20.18*
Tnh	Tnh	34.75	25.25	12.50	21.25	6.25	12.50	26.25	22.00	8.50	25.00	43.00	*21.57*
Exp	Tnh	26.00	17.75	24.50	0.00	3.50	5.25	8.50	13.75	5.50	5.25	42.50	*13.86*
Lin	Exp	22.25	16.00	46.25	0.50	3.25	2.50	2.75	21.75	2.75	2.25	45.25	*15.05*
Lgs	Exp	29.25	26.50	38.50	14.75	6.25	6.75	17.50	21.50	8.25	16.00	45.00	*20.93*
Tnh	Exp	32.50	34.00	18.00	19.50	7.75	7.00	17.00	17.25	9.75	19.75	45.50	*20.73*
Exp	Exp	15.00	13.50	38.00	1.00	7.75	8.00	3.50	16.25	12.25	1.75	44.00	*14.64*
Average		*23.71*	*19.35*	*32.38*	*11.04*	*7.65*	*11.73*	*14.75*	*20.29*	*13.29*	*12.60*	*45.08*	

Table B.35 Mean test classification error for Multi-Class dataset using GFNNs

Activation Functions		Training Algorithms / Structure											Avg.
		G 2-3		GP 2-2-3									
Output		None		Linear			Log sigmoid (Lgs)			Tan sigmoid (Tnh)			
GSN		GDX	LM	GDX	LM	DS-GDX	GDX	LM	DS-GDX	GDX	LM	DS-GDX	
Num	Den												
Lin	Lgs	11.95	7.11	36.73	5.83	9.61	5.61	5.97	13.85	11.71	5.79	16.44	*11.87*
Lgs	Lgs	9.57	12.95	20.40	15.23	14.47	18.36	19.85	16.52	17.20	21.95	18.19	*16.79*
Tnh	Lgs	31.05	12.37	23.08	11.33	17.31	13.19	14.72	8.67	22.16	17.97	22.85	*17.70*
Exp	Lgs	63.35	17.56	66.17	8.76	65.87	16.24	11.76	47.45	15.27	9.45	46.85	*33.52*
Lin	Tnh	7.56	8.53	49.35	5.92	8.76	6.52	6.63	8.69	8.23	6.61	15.69	*12.04*
Lgs	Tnh	7.29	12.59	12.45	12.92	11.53	6.83	14.92	15.47	11.32	18.49	14.99	*12.62*
Tnh	Tnh	19.67	10.88	11.01	13.19	14.93	7.99	13.27	11.00	12.37	18.57	21.43	*14.03*
Exp	Tnh	62.61	16.20	65.83	10.27	65.93	17.24	9.27	47.87	19.33	8.63	44.88	*33.46*
Lin	Exp	7.69	6.65	11.71	6.23	8.37	6.47	5.92	8.28	11.17	7.19	10.21	*8.17*
Lgs	Exp	6.81	8.89	7.71	9.73	8.25	5.95	11.32	7.21	6.11	11.08	12.33	*8.67*
Tnh	Exp	7.95	7.25	10.29	7.11	10.79	6.27	7.48	9.61	9.29	8.79	16.61	*9.22*
Exp	Exp	62.51	13.11	64.33	7.09	64.11	32.15	7.17	47.20	41.48	6.33	50.59	*36.01*
Average		*24.83*	*11.17*	*31.59*	*9.47*	*24.99*	*11.90*	*10.69*	*20.15*	*15.47*	*11.74*	*24.26*	

Table B.36 Mean test ARV for Sunspot dataset using GFNNs

Activation Functions		Training Algorithms / Structure											Avg.
		G 10-1		GP 10-2-1									
Output		None		Linear			Log sigmoid (Lgs)			Tan sigmoid (Tnh)			
GSN		GDX	LM	GDX	LM	DS-GDX	GDX	LM	DS-GDX	GDX	LM	DS-GDX	
Num	Den												
Lin	Lgs	0.120	0.158	0.146	0.100	0.138	1.010	0.115	0.220	0.158	0.106	0.153	*0.220*
Lgs	Lgs	0.827	0.253	0.987	0.257	0.204	1.020	0.316	0.439	0.982	0.280	0.277	*0.531*
Tnh	Lgs	0.544	1.430	0.306	0.172	0.158	1.010	0.249	0.317	0.357	0.212	0.177	*0.448*
Exp	Lgs	0.367	0.145	0.275	0.121	0.163	0.830	0.161	0.264	0.226	0.125	0.171	*0.259*
Lin	Tnh	0.141	0.144	0.173	0.106	0.148	0.921	0.127	0.229	0.166	0.137	0.164	*0.223*
Lgs	Tnh	0.604	0.322	0.843	0.173	0.173	0.999	0.294	0.486	0.799	0.207	0.239	*0.467*
Tnh	Tnh	0.298	0.229	0.248	0.138	0.165	1.020	0.201	0.300	0.253	0.157	0.186	*0.290*
Exp	Tnh	0.316	0.140	0.256	0.139	0.173	0.811	0.168	0.290	0.233	0.130	0.185	*0.258*
Lin	Exp	0.124	0.221	0.162	0.138	0.153	1.010	0.161	0.238	0.176	0.139	0.167	*0.244*
Lgs	Exp	0.506	0.201	0.908	0.201	0.182	1.020	0.167	0.385	0.933	0.192	0.225	*0.447*
Tnh	Exp	0.164	0.162	0.366	0.148	0.171	1.020	0.193	0.320	0.433	0.166	0.204	*0.304*
Exp	Exp	0.271	0.151	0.271	0.138	0.167	0.844	0.162	0.270	0.226	0.127	0.168	*0.254*
Average		*0.357*	*0.296*	*0.412*	*0.153*	*0.166*	*0.960*	*0.193*	*0.313*	*0.412*	*0.165*	*0.193*	

Table B.37 Results for Breast Cancer dataset using GFNNs with various s_{lim}

Network	G 1 LinLgs Trained using GDX		G 1 LinLgs Trained using LM		GP 2-1 LinExp-Tnh Trained using GDX		GP 2-1 ExpExp-Tnh Trained using LM	
s limit	Mean Error (%)	CPU Time (s)	Mean Error (%)	CPU Time (s)	Mean Error (%)	CPU Time (s)	Mean Error (%)	CPU Time (s)
0.01	0.16	6.96	0.44	9.39	0.19	22.60	0.49	15.69
0.02	0.16	6.96	0.44	9.45	0.19	22.62	0.49	15.69
0.05	0.16	6.96	0.44	9.44	0.19	22.63	0.49	15.69
0.1	0.16	6.96	0.44	9.44	0.19	22.61	0.49	15.70
0.2	0.16	6.96	0.44	9.42	0.19	22.62	0.49	15.69
0.5	0.16	6.96	0.44	9.42	0.19	22.64	0.49	15.70
1.0	0.16	6.96	0.44	9.41	0.19	22.63	0.49	15.69
2.0	0.20	6.71	0.44	9.43	0.27	21.30	0.49	15.70

Table B.38 Results for Diabetes dataset using GFNNs with various s_{lim}

Network	G 1 TnhExp Trained using GDX		G 1 TnhLgs Trained using LM		GP 2-1 TnhTnh-Tnh Trained using GDX		GP 2-1 ExpLgs-Lgs Trained using LM	
s limit	Mean Error (%)	CPU Time (s)	Mean Error (%)	CPU Time (s)	Mean Error (%)	CPU Time (s)	Mean Error (%)	CPU Time (s)
0.01	20.58	9.72	20.56	5.25	20.65	15.10	20.36	17.83
0.02	20.58	9.73	20.56	5.26	20.65	15.10	20.36	17.82
0.05	20.58	9.73	20.56	5.24	20.67	15.18	20.36	17.81
0.1	20.58	9.73	20.56	5.24	20.65	15.26	20.36	17.81
0.2	20.58	9.72	20.56	5.24	20.66	15.35	20.36	17.81
0.5	20.58	9.73	20.56	5.26	20.60	15.44	20.36	17.81
1.0	20.58	9.73	20.56	5.21	20.58	15.67	20.36	17.82
2.0	20.78	9.46	20.56	5.19	21.98	22.40	20.36	17.81

Table B.39 Results for 3-bit Parity using GFNNs with various s_{lim}

Network	G 3-1 LinTnh Trained using GDX		G 3-1 ExpLgs Trained using LM		GP 3-2-1 LinExp-Lgs Trained using GDX		GP 3-2-1 LinLgs-Lin Trained using LM	
s limit	Mean Error (%)	CPU Time (s)	Mean Error (%)	CPU Time (s)	Mean Error (%)	CPU Time (s)	Mean Error (%)	CPU Time (s)
0.01	26.75	5.56	11.00	4.50	2.75	3.94	0.00	0.80
0.02	25.25	5.56	11.00	4.50	2.75	3.87	0.00	0.79
0.05	25.00	5.56	11.00	4.50	2.75	3.80	0.00	0.79
0.1	25.75	5.56	11.00	4.50	2.75	3.53	0.00	0.79
0.2	25.25	5.56	11.00	4.50	2.50	3.34	0.00	0.79
0.5	17.00	5.56	11.00	4.50	3.50	3.17	0.00	0.79
1.0	16.00	5.56	11.00	4.50	2.50	3.14	0.00	0.79
2.0	13.50	5.59	11.00	4.48	2.25	3.94	0.00	0.79

Table B.40 Results for Multi-Class problem using GFNNs with various s_{lim}

Network	G 2-3 LgsExp Trained using GDX		G 2-3 LinExp Trained using LM		GP 2-2-3 LinLgs-Lgs Trained using GDX		GP 2-2-3 LinLgs-Tnh Trained using LM	
s limit	Mean Error (%)	CPU Time (s)	Mean Error (%)	CPU Time (s)	Mean Error (%)	CPU Time (s)	Mean Error (%)	CPU Time (s)
0.01	8.16	8.15	6.65	28.10	5.88	14.26	5.79	65.40
0.02	8.16	8.15	6.65	28.10	5.85	13.97	5.79	65.40
0.05	8.16	8.20	6.65	28.10	5.87	13.49	5.79	65.40
0.1	7.96	9.10	6.65	28.10	5.77	14.80	5.79	65.40
0.2	7.92	8.95	6.65	28.10	5.72	15.47	5.79	65.40
0.5	7.92	8.65	6.65	28.10	5.67	14.60	5.79	65.40
1.0	6.81	9.00	6.65	28.10	5.61	13.20	5.79	65.40
2.0	6.27	9.50	6.65	28.10	5.67	16.63	5.79	65.40

B.5 Experimental Results for Chapter 8

This section presents the experimental results obtained in Chapter 8, "A Generalised Feedforward Neural Network Architecture". For each benchmark test, the mean test error, obtained using MLPs, for all combinations of activation functions and training algorithms are presented in Tables B.41 to B.46. The average for each activation function combination (row) and training algorithm (column) are also shown.

Table B.41 Mean test classification error for Breast Cancer dataset using MLPs

Activation Functions		Network type / Training Algorithms					Average
		G- MLP			MT- MLP, NW init		
Hidden	Output	GDX	LM	QNN	GDX	LM	
Lgs	Lin	0.08	0.61	0.36	0.37	0.72	*0.43*
Tnh	Lin	0.09	0.60	0.24	0.73	0.72	*0.48*
Lgs	Lgs	0.33	0.77	0.95	0.46	0.92	*0.69*
Tnh	Lgs	0.56	1.07	1.14	0.86	1.36	*1.00*
Lgs	Tnh	0.12	0.51	0.24	1.74	23.95	*5.31*
Tnh	Tnh	0.32	0.73	0.51	1.28	31.02	*6.77*
Average		*0.25*	*0.72*	*0.57*	*0.91*	*9.78*	

Table B.42 Mean test classification error for Diabetes dataset using MLPs

Activation Functions		Network type / Training Algorithms					Average
		G- MLP			MT- MLP, NW init		
Hidden	Output	GDX	LM	QNN	GDX	LM	
Lgs	Lin	22.85	21.38	20.48	22.45	22.45	*21.92*
Tnh	Lin	21.42	21.42	20.95	25.14	25.14	*22.81*
Lgs	Lgs	25.32	21.70	21.57	21.66	21.66	*22.38*
Tnh	Lgs	20.88	21.71	21.55	21.48	21.48	*21.42*
Lgs	Tnh	22.14	20.75	21.35	21.68	21.68	*21.52*
Tnh	Tnh	20.45	21.09	20.77	29.24	29.24	*24.16*
Average		*22.18*	*21.34*	*21.11*	*23.61*	*23.61*	

Table B.43 Mean test classification error for 3-bit Parity dataset using MLPs

Activation Functions		Network type / Training Algorithms					Average
		G- MLP			MT- MLP, NW init		
Hidden	Output	GDX	LM	QNN	GDX	LM	
Lgs	Lin	2.00	0.00	3.00	6.25	3.50	*2.95*
Tnh	Lin	2.50	1.25	10.25	13.75	4.25	*6.40*
Lgs	Lgs	0.50	0.00	7.75	4.25	2.00	*2.90*
Tnh	Lgs	2.00	1.25	13.75	11.75	6.75	*7.10*
Lgs	Tnh	0.50	1.00	7.00	4.75	13.50	*5.35*
Tnh	Tnh	2.00	3.50	11.25	11.25	19.00	*9.40*
Average		*1.58*	*1.17*	*8.83*	*8.67*	*8.17*	

Table B.44 Mean test classification error for Multi-Class dataset using MLPs

Activation Functions		Network type / Training Algorithms					Average
		G- MLP			MT- MLP, NW init		
Hidden	Output	GDX	LM	QNN	GDX	LM	
Lgs	Lin	6.07	5.95	5.73	6.09	5.99	*5.97*
Tnh	Lin	5.77	5.96	6.04	6.43	6.12	*6.06*
Lgs	Lgs	5.37	6.52	5.43	5.53	12.24	*7.02*
Tnh	Lgs	5.48	6.56	5.69	10.39	7.37	*7.10*
Lgs	Tnh	5.85	12.64	7.36	30.21	28.80	*16.97*
Tnh	Tnh	5.76	5.83	5.69	22.65	13.59	*10.70*
Average		*5.65*	*7.50*	*6.04*	*15.04*	*13.62*	

Table B.45 Mean test classification error for Thyroid dataset using MLPs

Activation Functions		Network type / Training Algorithms					Average
		G- MLP			MT- MLP, NW init		
Hidden	Output	GDX	LM	QNN	GDX	LM	
Lgs	Lin	6.75	2.43	2.23	6.59	2.49	*4.10*
Tnh	Lin	6.47	2.52	2.69	6.7	2.69	*4.21*
Lgs	Lgs	5.97	1.72	2.18	6.59	7.89	*4.87*
Tnh	Lgs	5.96	1.83	3.68	13.85	7.45	*6.55*
Lgs	Tnh	7.17	2.02	2.22	21.14	40.66	*14.64*
Tnh	Tnh	6.25	2.07	2.47	28.36	29.17	*13.66*
Average		*6.36*	*2.03*	*2.65*	*15.33*	*17.57*	

Table B.46 Mean test ARV for Sunspot dataset using MLPs

Activation Functions		Network type / Training Algorithms					Average
		G- MLP			MT- MLP, NW init		
Hidden	Output	GDX	LM	QNN	GDX	LM	
Lgs	Lin	0.363	0.138	0.164	0.163	0.234	*0.212*
Tnh	Lin	0.162	0.144	0.139	0.222	0.326	*0.199*
Lgs	Lgs	1.023	0.157	0.218	0.181	0.231	*0.362*
Tnh	Lgs	0.496	0.151	0.420	0.195	0.235	*0.299*
Lgs	Tnh	0.406	0.139	0.183	0.186	1.124	*0.408*
Tnh	Tnh	0.177	0.142	0.157	0.214	1.243	*0.387*
Average		*0.453*	*0.147*	*0.223*	*0.200*	*0.632*	

Bibliography

Abe, S., Thawonmas, R., & Kayama, M. (1999). A Fuzzy Classifier with ellipsoidal regions for diagnosis problems. *IEEE Transactions on Systems, Man and Cybernetics: Part C: Application and Reviews, 29*(1), 140-149.

Ampazis, N., & Perantonis, S. J., 2000. Levenberg-Marquadt Algorithm with Adaptive Momentum for the Efficient Training of Feedfoward Networks, in *Proc. Intern. Joint Conf. on Neural Networks (IJCNN 2000)*, pp. 126-131.

Ampazis, N., & Perantonis, S. J. (2002). Two Highly Efficient Second-Order Algotithms for Training Feedfoward Networks. *IEEE Trans. on Neural Networks, 13*(5), 1064-1074.

Arulampalam, G., & Bouzerdoum, A. (2000). Training Shunting Inhibitory Artificial Neural Networks as Classifiers. *Neural Network World, 10*(3), 333-350.

Arulampalam, G., & Bouzerdoum, A., 2001a. Application of Shunting Inhibitory Artificial Neural Networks to Medical Diagnosis, in *Proc. 7th Australian and New Zealand Intelligent Information Systems Conference (ANZIIS 2001)*, pp. 89-94.

Arulampalam, G., & Bouzerdoum, A. (2001b). Novel Training Algorithm Based on Quadratic Optimisation Using Neural Networks. In J. Mira & A. Prieto (Eds.), *Bio-Inspired Applications of Connectionism* (Vol. 1, pp. 410-417). Berlin: Springer-Verlag.

Arulampalam, G., & Bouzerdoum, A., 2002a. Expanding the Structure of Shunting Inhibitory Artificial Neural Network Classifiers, in *Proc. Intern. Joint Conf. on Neural Networks (IJCNN '02)*, pp. 2855-2860.

Arulampalam, G., & Bouzerdoum, A. (2002b). Recurrent Neural Network-based Quadratic Optimisation Training Algorithm for Feedforward Neural Networks. *International Journal of Computers, Systems and Signals (IJCSS), 3*(2), 65-75.

Arulampalam, G., & Bouzerdoum, A. (2003a). A Generalized Feedforward Neural Network architecture for classification and regression. *Neural Networks, 16*, 561-568.

Arulampalam, G., & Bouzerdoum, A., 2003b. A Generalized Feedfoward Neural Network Classifier, in *Proc. Intern. Joint Conf. on Neural Networks (IJCNN 2003)*, pp. 1429-1434.

Arulampalam, G., Ramakonar, V., Bouzerdoum, A., & Habibi, D., 1999. Classification of Digital Modulation Schemes using Neural Networks, in *Proc. 5th International Symposium on Signal Processing and its Applications*, pp. 649-652.

Back, T. (1997). Evolutionary Computation: Comments on the History and Current State. *IEEE Trans. on Evolutionary Computation, 1*(1), 3-17.

Barmann, F., & Biegler-Konig, F. (1992). On a class of efficient learning algorithms for neural networks. *Neural Networks, 5*, 139-144.

Barmann, F., & Biegler-Konig, F. (1993). A learning algorithm for multilayered neural networks based on linear least squares problems. *Neural Networks, 6*, 127-131.

Battiti, R. (1989). Accelerated backpropagation learning: Two optimization methods. *Complex Systems, 3*, 331-342.

Battiti, R. (1992). First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method. *Neural Computation, 4*(2), 141-166.

Beare, R., & Bouzerdoum, A. (1999). Biologically inspired local motion detector architecture. *J. Opt. Soc. Am. A, 16*(9), 2059-2068.

Becker, S. (1991). Unsupervised learning procedures for neural networks. *International Journal of Neural Systems, 2*, 17-33.

Becker, S., & Hinton, G. E. (1992). A self-organising neural network that discovers surfaces in random-dot stereograms. *Nature, 355*(161-163).

Bhumireddy, C., & Chen, C. L. P., 2003. Genetic learning of functional link networks, in *Proc. Intern. Joint Conf. on Neural Networks (IJCNN 2003)*, pp. 432-437.

Blake, C. L., & Merz, C. J. (1998). *UCI Repository of machine learning databases*, from http://www.ics.uci.edu/~mlearn/MLrepository.html

Boroushaki, M., Ghofrani, M. B., Lucas, C., & Yazdanpanah, M. J. (2003). Identification and control of a nuclear reactor core (VVER) using recurrent neural networks and fuzzy systems. *Nuclear Science, IEEE Transactions on, 50*(1), 159-174.

Bouzerdoum, A. (1992). Convergence of symmetric shunting competitive neural networks. In D. Green & T. Bossomaier (Eds.), *Complex Systems: From Biology to Computation* (pp. 301-312). Amsterdam: IOS Press.

Bouzerdoum, A. (1993). The elementary movement detection mechanism in insect vision. *Phil. Trans. R. Soc. Lond, B-339*, 375-384.

Bouzerdoum, A., 1999. A new class of high-order neural networks with nonlinear decision boundaries, in *Proc. Int. Conference on Neural Information Processing (ICONIP '99)*, pp. 1004-1009.

Bouzerdoum, A., 2000. Classification and function approximation using feed-forward shunting inhibitory artificial neural networks, in *Proc. Intern. Joint Conf. on Neural Networks (IJCNN 2000)*, pp. 613-618.

Bouzerdoum, A., & Pattison, T. R., 1993a. Constrained Quadratic Optimisation using Neural Networks, in *Proc. 4th ACNN*, pp. 10-13.

Bouzerdoum, A., & Pattison, T. R. (1993b). Neural Network for Quadratic Optimization with Bound Constraints. *IEEE Transactions on Neural Networks, 4*(2), 293-304.

Bouzerdoum, A., & Pinter, R. B., 1989. Image motion processing in biological and computer vision systems, in *Proc. Proc. of SPIE*, pp. 1229-1240.

Bouzerdoum, A., & Pinter, R. B. (1992). Nonlinear lateral inhibition applied to motion detection in the fly visual system. In R. B. Pinter & B. Nabet (Eds.), *Nonlinear Vision* (pp. 423-450). Boca Raton: CRC Press.

Bouzerdoum, A., & Pinter, R. B. (1993). Shunting Inhibitory Cellular Neural Networks: Derivation and Stability Analysis. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, 40*(3), 215 - 221.

Bowen, J. E., & Bowen, W. E., 1990. Neural nets vs. expert systems: predicting in the financial field, in *Proc. Artificial Intelligence for Applications, 1990., Sixth Conference on*, pp. 72-77 vol.71.

Carpenter, G. A., & Grossberg, S. (1987). A massively parallel architecture for a self-organising neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing, 37*, 54-115.

Carpenter, G. A., & Grossberg, S. (1988). The ART of Adaptive Pattern Recognition by a Self-Organising Neural Network. *IEEE Computer, 21*(3).

Cheung, H. N., Bouzerdoum, A., & Newland, W., 1999. Properties of Shunting Inhibitory Cellular Neural Networks for Colour Image Enhancement, in *Proc. 6th Int. Conf. on Neural Info. Processing (ICONIP '99)*, pp. 1219-1223.

Chong, E. K. P., & Zak, S. H. (1996). *An Introduction to Optimization*. New York: Wiley-Interscience.

Connor, J. T., Martin, R. D., & Atlas, L. E. (1994). Recurrent Neural Networks and Robust Time Series Prediction. *IEEE Trans. on Neural Networks, 5*(2), 240-254.

Cover, T. M. (1965). Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers, 14*, 326-334.

Cugnon, P., & team, S. (2003). *Online catalogue for the sunspot index*, 2003, from http://sidc.oma.be/html/sunspot.html

Darling, R. B., & Dietze, W. T. (1993). Implementation of multiplicative lateral inhibition in a GaAs sensory neural network photodetector array. *IEEE J. Quantum Electronics, 29*(2), 645-654.

Demuth, H., & Beale, M. (1992). *Neural Network Toolbox User's Guide* (Version 3 ed.): The MathWorks Inc.

Di Martino, M., Fanelli, S., & Protasi, M., 1993. An efficient algorithm for the binary classification of patterns using MLP networks, in *Proc. 2nd IEEE Int. Conf. on Neural Networks*, pp. 936 - 943.

Di Martino, M., Fanelli, S., & Protasi, M. (1996). Exploring and Comparing the Best "Direct Methods" for the Efficient Training of MLP-Networks. *IEEE Transactions on Neural Networks, 7*(6), 1497-1502.

Dickhaus, H., 2001. Wavelet neural networks for clinical diagnosis, in *Proc. Engineering in Medicine and Biology Society, 2001. Proceedings of the 23rd Annual International Conference of the IEEE*, pp. 4095 vol.4094.

Duch, W., & Jankowski, N., 2001. Transfer functions: hidden possibilities for better neural networks, in *Proc. 9th European Symposium on Artificial Neural Networks (ESANN)*, pp. 81-94.

Duda, P. O., & Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. New York: Wiley.

Duda, P. O., Hart, P. E., & Stork, D. G. (2001). *Pattern Classification* (2nd ed.). New York: Wiley.

Er, M. J., Wu, S., Lu, J., & Toh, H. L. (2002). Face recognition with radial basis function (RBF) neural networks. *Neural Networks, IEEE Transactions on, 13*(3), 697-710.

Erdogmus, D., K.E. Hild, I., & Principe, J. C. (2003). Online Entropy Manipulation: Stochastic Information Gradient. *IEEE Signal Processing Letters, 10*(8), 242-245.

Erdogmus, D., & Principe, J. C., 2000. Comparison of Entropy and Mean Square Error Criteria in Adaptive System Training Using Higher Order Statistics, in *Proc. Independent Component Analysis 2000.*

Erdogmus, D., & Principe, J. C., 2001. Entropy minimization algorithm for multilayer perceptrons, in *Proc. International Joint Conference on Neural Networks (IJCNN 2001)*, pp. 3003-3008.

Erdogmus, D., & Principe, J. C. (2002). Generalized Information Potential Criterion for Adaptive System Training. *IEEE Trans. on Neural Networks, 13*(5), 1035-1043.

Faber, D. S., & Korn, H. (1982). Transmission at a Central Inhibitory Synapse. I. Magnitude of unitary postsynaptic conductance change and kinetics of channel activation. *Journal of Neurophysiology, 48*(3), 654-678.

Fahlman, S. E., & Lebiere, C. (1990). *The Cascade-Correlation Learning Architecture* (No. CMU-CS-90-100). Pittsburgh, PA.: School of Computer Science, Carnegie Mellon University.

Fiesler, E., 1994. Comparative Bibliography of Ontogenic Neural Networks, in *Proc. International Conference on Artificial Neural Networks (ICANN 94)*, pp. 793-796.

Fischler, M. A., & Firschein, O. (1987). *Intelligence: The Eye, the Brain, and the Computer.* Reading, MA: Addison-Wesley.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics, 7*, 179-188.

Fogel, D. B. (1992). *Evolving artificial intelligence.* Unpublished Ph.D., Univ. of California, San Diego, CA.

Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). *Artificial Intelligence Through Simulated Evolution.* New York: Wiley.

Franzini, M. A., 1987. Speech Recognition with Back Propagation", in *Proc. IEEE Ninth Annual Conference on Engineering in Medicine and Biology*, pp. 1702-1703.

Fukushima, K., Miyake, S., & Ito, T. (1983). Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Trans. Systems, Man & Cybernetics, 13*, 826-834.

Funahashi, K. (1989). On the approximate realization of continuous mappings by neural networks. *Neural Networks, 2*, 183-192.

Furman, G. G. (1965). Comparison of models for subtractive and shunting lateral-inhibition in receptor-neuron fields. *Kybernetik, 2*, 257-274.

Garcia-Pedrajas, N., Hervas-Martinez, C., & Munoz-Perez, J. (2003). COVNET: A Cooperative Coevolutionary Model for Evolving Artificial Neural Networks. *IEEE Trans. on Neural Networks, 14*(3), 575-596.

Gershenfeld, N. A., & Weigend, A. S. (1993). The Future of Time Series: Learning and Understanding. In A. S. Weigend & N. A. Gershenfeld (Eds.), *Time Series Prediction: Forecasting the future and understanding the past.* Reading, Mass.: Addison-Wesley.

Gerstner, W., & Kistler, W. M. (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*: Cambridge University Press.

Giles, C. L., Lawrence, S., & Tsoi, A. C., 1997. Rule inference for financial prediction using recurrent neural networks, in *Proc. Computational Intelligence for Financial Engineering (CIFEr), 1997., Proceedings of the IEEE/IAFE 1997*, pp. 253-259.

Grossberg, S. (1973). Contour enhancement, short term memory, and constancies in reverberating neural networks. *Studies in Applied Mathematics, 52*(3), 213-257.

Grossberg, S. (1976). Adaptive Pattern Classification and Universal Recoding: I. Parallel Development and Coding of Neural Feature Detectors. *Biol. Cybernetics, 23*, 121-134.

Grossberg, S. (Ed.). (1988). *Neural Networks and Natural Intelligence.* Cambridge, Mass.: MIT Press.

Gunn, S. (1998). *Support Vector Machines for Classification and Regression* (No. Technical ReportISIS-1-98): Image Speech & Intelligent Systems Group, University of Southampton.

Hagan, M. T., & Menhaj, M. B. (1994). Training Feedforward Networks with the Marquadt Algorithm. *IEEE Transactions on Neural Networks, 5*(6), 989 - 993.

Hanumantharaya, U., Leis, J., & Hancock, N., 1999. Quantitative Odour Modelling using Electronic Nose Information, in *Proc. 5th International Symposium on Signal Processing and its Applications*, pp. 163-166.

Hassibi, B., & Stork, D. G. (1993). Second Order Derivatives for Network Pruning: Optimal Brain Surgeon. In S. J. Hanson, J. D. Cowan & C. L. Giles (Eds.), *Advances in Neural Information Processing Systems* (Vol. 5, pp. 164-171). San Mateo, CA: Morgan Kaufmann.

Hathaway, R., & Bezdek, J. C. (2001). Fuzzy c-Means Clustering of Incomplete Data. *IEEE Trans. on Systems, Man and Cybernetics, Part B: Cybernetics, 31*(5), 735-744.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation* (2nd ed.). New York: Prentice-Hall.

Hebb, D. O. (1949). *The Organization of Behaviour: A Neuropsychological Theory*. New York: Wiley.

Henery, R. J. (1994). Classification. In D. Michie, D. J. Spiegelhalter & C. C. Taylor (Eds.), *Machine Learning, Neural and Statistical Classification*. London, U.K.: Ellis Horwood.

Hinton, G. E., & Sejnowski, T. J., 1983. Optimal Perceptual Inference, in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 448-453.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward neural networks are universal approximators. *Neural Networks, 2*, 359-366.

Hsin, H.-C., Li, C.-C., Sun, M., & Sclabassi, R. J. (1992). An Adaptive Training Algorithm for Back-Propagation Neural Networks. 1049 - 1052.

Islam, M. M., Shahjahan, M., & Murase, K., 2000. An Algorithm for Automatic Design of Two Hidden Layered Artificial Neural Networks, in *Proc. Intern. Joint Conf. on Neural Networks (IJCNN 2000)*, pp. 467 - 472.

Islam, M. M., Yao, X., & Murase, K. (2003). A Constructive Algorithm for Training Neural Network Ensembles. *IEEE Trans. on Neural Networks, 14*(4), 820-834.

Jacobs, R. A., & Jordan, M. I. (1991). A competitive modular connectionist architecture. In R. P. Lippmann, J. E. Moody & D. S. Touretzky (Eds.), *Advances in Neural Information Processing Systems 3* (pp. 767-773). San Mateo, CA: Morgan Kauffman.

Jankowski, N. (1999). *Flexible transfer functions with ontogenic neural networks*. Torun, Poland: Computational Intelligence Lab, DCM NCU.

Jankowski, N., 2003. Discrete feature weighting & selection algorithm, in *Proc. Intern. Joint Conf. on Neural Networks (IJCNN 2003)*, pp. 636-641.

Jankowski, N., & Duch, W., 2001. Optimal transfer function neural networks, in *Proc. 9th European Symposium on Artificial Neural Networks (ESANN)*, pp. 101-106.

Jenssen, R., K.E. Hild, I., Erdogmus, D., Principe, J. C., & Eltoft, T., 2003. Clustering using Renyi's Entropy, in *Proc. Intern. Joint Conf. on Neural Networks (IJCNN 2003)*, pp. 523-528.

Johansson, E. M., Dowla, F. U., & Goodman, D. M. (1992). Backpropagation Learning for Multilayer Feed-forward Neural Networks Using the Conjugate Gradient Method. *International Journal of Neural Systems, 2*(4), 291-301.

Johnson, R. A., & Bhattacharya, G. K. (1996). *Statistics: principles and methods* (3rd ed.). New York: John Wiley & Sons.

Karampiperis, P., Manouselis, N., & Trafalis, T. B., 2002. Architecture selection for neural networks, in *Proc. Intern. Joint Conf. on Neural Networks (IJCNN 2002)*, pp. 1115-1119.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics, 43*, 59-69.

Kohonen, T. (1988). *Self-Organization and Associative Memory* (3rd ed.). New York: Springer-Verlag.

Kordylewski, H., Graupe, D., & Liu, K. (2001). A novel large-memory neural network as an aid in medical diagnosis applications. *Information Technology in Biomedicine, IEEE Transactions on, 5*(3), 202-209.

Koshiba, Y., & Abe, S., 2003. Comparison of L1 and L2 Support Vector Machines, in *Proc. Intern. Joint Conf. on Neural Networks (IJCNN 2003)*, pp. 2054-2059.

Lashkia, V., & Aleshin, S. (2001). Test Feature Classifiers: Performance and Applications. *IEEE Transactions on Systems, Man and Cybernetics: Part B: Cybernetics, 31*(4), 643-650.

Lawrence, S., Tsoi, A. C., & Back, A. D., 1996. Function approximation with neural networks and local methods: bias, variance and smoothness, in *Proc. Australian Conf. on Neural Networks (ACNN '96)*, pp. 16-21.

LeCun, Y., Denker, J., Solla, S., Howard, R. E., & Jackel, L. D. (1990). Optimal Brain Damage. In D. S. Touretzky (Ed.), *Advances in Neural Information Processing Systems II* (pp. 598-605). San Mateo, CA: Morgan Kauffman.

Lee, H. K. H., 2000. A Framework for Nonparametric Regression Using Neural Networks, in *Proc. Pacific Rim International Conference on Artificial Intelligence*, pp. 617-626.

Lee, Y., Oh, S., & Kim, M., 1991. The effect of initial weights on premature saturation in back-propagation learning, in *Proc. International Joint Conference on Neural Networks*, pp. 765-770.

Lettvin, J. Y. (1962). *Form-Function Relations in Neurons* (Res. Quart. Prog. Report): MIT.

Leung, F. H. F., Lam, H. K., Ling, S. H., & K.S., T. P. (2003). Tuning of the Structure and Parameters of a Neural Network Using an Improved Genetic Algorithm. *IEEE Trans. on Neural Networks, 14*(1), 79-88.

Linsker, R. (1988). Self-organization in a perceptual network. *Computer, 21*, 105-117.

Madyastha, R. K., & Aazhang, B. (1994). An Algorithm for Training Multilayer Perceptrons for Data Classification and Function Interpolation. *IEEE Transactions on Circuits and Systems-I:Fundamental Theories and Applications, 41*(12), 866-875.

Magoulas, G. D., Vrahatis, M. N., & Androulakis, G. S. (1999). Improving the Convergence of the Backpropagation Algorithm Using Learning Rate Adaptation Methods. *Neural Computation, 11*(7), 1769-1796.

McConaghy, T., Leung, H., Bosse, E., & Varadan, V. (2003). Classification of audio radar signals using radial basis function neural networks. *Instrumentation and Measurement, IEEE Transactions on, 52*(6), 1771-1779.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics, 5*, 115-133.

Meesad, P., & Yen, G. G., 2001. A Hybrid Intelligent System for medical diagnosis, in *Proc. Intern. Joint Conf. on Neural Networks (IJCNN 2001)*, pp. 2558-2563.

Michie, D., Spiegelhalter, D. J., & Taylor, C. C. (Eds.). (1994). *Machine Learning, Neural and Statistical Classification*. London, U.K.: Ellis Horwood.

Minsky, M. L., & Papert, S. A. (1969). *Perceptrons*. Cambridge MA: MIT Press.

Mitchell, T. M. (1997). *Machine Learning*. New York: McGraw-Hill.

Moini, A., Bouzerdoum, A., & Eshragian, K., 1997. A current mode implementation of Shunting Inhibition, in *Proc. International Symposium on Circuits and Systems*, pp. 557-560.

Morejon, R. A., & Principe, J. C. (2004). Advanced Search Algorithms for Information-Theoretic Learning With Kernel-Based Estimators. *IEEE Trans. on Neural Networks, 15*(4), 874-884.

Musicant, D. R., & Feinberg, A. (2004). Active Set Support Vector Regression. *IEEE Trans. on Neural Networks, 15*(2), 268-275.

Na, M. G., Shin, S. H., Lee, S. M., Jung, D. W., Kim, S. P., Jeong, J. H., et al. (2004). Prediction of major transient scenarios for severe accidents of nuclear power plants. *Nuclear Science, IEEE Transactions on, 51*(2), 313-321.

Nabet, B. (1992). Electronic hardware for vision modelling. In R. B. Pinter & B. Nabet (Eds.), *Nonlinear Vision* (pp. 463-474). Boca Raton: CRC Press.

Naftaly, U., Intrator, N., & Horn, D. (1997). Optimal Ensemble Averaging of Neural Networks. *Network: Comput. Neural Syst., 8*(3), 283-296.

Nguyen, D., & Widrow, B., 1990. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights, in *Proc. Intern. Joint Conf. on Neural Networks*, pp. 21-26.

Nicholls, J. G., Martin, A. R., & Wallace, B. G. (1992). *From Neuron to Brain: A Cellular Approach to the Function of the Nervous System* (3rd ed.). Sunderland, Massachussetts: Sinauer Associates Inc.

Nikolaev, N. Y., & Iba, H. (2003). Learning polynomial feedforward neural networks by genetic programming and backpropagation. *IEEE Trans. on Neural Networks, 14*(2), 337-350.

Nilson, C. D., Darling, R. B., & Pinter, R. B. (1994). Shunting neural network photodetector arrays in analog CMOS. *IEEE J. Solid State Electronics, 29*(10), 1291-1296.

Nilsson, N. J. (1990). *The Mathematical Foundations of Learning Machines*. San Mateo, CA: Morgan Kaufmann Publishers.

Park, Y. R., Murray, T. J., & Chen, C. (1996). Predicting sun spots using a layered perceptron neural network. *IEEE Trans. on Neural Networks, 7*(2), 501-505.

Pinter, R. B. (1983). Product term nonlinear lateral inhibition enhances visual selectivity for small objects or edges. *J. Theor. Biol., 100*, 525-531.

Pinter, R. B. (1984). Adaptation of receptive field organization by multiplicative lateral inhibition. *J. Theor. Biol., 110*, 435-444.

Pinter, R. B. (1985). Adaptation of spatial modulation transfer function via nonlinear lateral inhibition. *Biological Cybernetics, 51*, 285-291.

Pontecorvo, C., & Bouzerdoum, A., 1995. Edge detection using a cellular neural network, in *Proc. 3rd Conference on Digital Image Computing Techniques and Applications (DICT'95)*, pp. 637-642.

Pontecorvo, C., & Bouzerdoum, A., 1997. Edge detection in multiplicative noise using the shunting inhibitory cellular neural network, in *Proc. Engineering Applications of Neural Networks (EANN'97)*, pp. 281-285.

Pontil, M., & Verri, A. (1998). Properties of Support Vector Machines. *Neural Computation, 10*, 955-974.

Powell, M. J. D. (1977). Restart procedures for conjugate gradient method. *Mathematical Programming, 12*, 241-254.

Powell, M. J. D. (1987). Radial basis functions for multivariable interpolation : a review. In J. C. Mason & M. G. Cox (Eds.), *Algorithms for Approximation of Functions and Data* (pp. 143-167). Oxford: Clarendon Press.

Prechelt, L. (1994). *PROBEN 1 - A Set of Neural Network Benchmark Problems and Benchmarking Rules* (No. Tech. Rep. 21/94). Karlsruhe, Germany: Fakultat fur Informatik, Universitat Karlsruhe.

Principe, J. C., Oja, E., Xu, L., Cichocki, A., & Erdogmus, D. (2004). Guest Editorial : Special Issue on Information Theoretic Learning. *IEEE Trans. on Neural Networks, 15*(4), 789.

Pulleyblank, W. (2004). How to build a supercomputer. *IEE Review, 50*(1), 48-52.

Reed, R. (1993). Pruning Algorithms - A Survey. *IEEE Trans. on Neural Networks, 4*, 740-747.

Renyi, A. (1970). *Probability Theory*. Amsterdam: North-Holland.

Richard, M. D., & Lippmann, R. P. (1991). Neural network classifiers estimate Bayesian *a posteriori* probabilities. *Neural Computation, 3*, 461-483.

Riedmiller, M., & Braun, H., 1993. A direct adaptive method for faster backpropagation learning: the RPROP algorithm, in *Proc. IEEE International Conference on Neural Networks*, pp. 586 - 591.

Ripley, B. D. (1996). *Pattern Recognition and Neural Networks*. Cambridge: Cambridge University Press.

Rivals, I., & Personnaz, L. (2003). Neural-Network Construction and Selection in Nonlinear Modeling. *IEEE Trans. on Neural Networks, 14*(4), 804-819.

Rosenblatt, F. (1958). The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review, 65*, 386-408.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*(323), 533-536.

Rutkowski, L. (2004). Generalized Regression Neural Networks in Time-Varying Environment. *IEEE Trans. on Neural Networks, 15*(3), 576-596.

Schalkoff, R. J. (1997). *Artificial Neural Networks*. New York: McGraw-Hill.

Schiffmann, W., Joost, M., & Werner, R. (1992a). *Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons*. Koblenz, Germany: University of Koblenz.

Schiffmann, W., Joost, M., & Werner, R. (1992b). *Synthesis and Performance Analysis of Multilayer Neural Network Architectures.* Koblenz, Germany: University of Koblenz.

Schiffmann, W., Joost, M., & Werner, R., 1993. Comparison of optimized backpropagation algorithms, in *Proc. European Symposium on Artificial Neural Networks, ESANN '93*, pp. 97-104.

Scholkopf, B., Sung, K.-K., Burges, C. J. C., Girosi, F., Niyogi, P., Poggio, T., et al. (1997). Comparing Support Vector Machines with Gaussian Kernels to Radial Basis Function Classifiers. *IEEE Trans. on Signal Processing, 45*(11), 2758-2765.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal, 27*, pp 379-423, 623-656.

Sherrah, J. (1998). *Automatic Feature Extraction for Pattern Recognition.* Unpublished PhD, University of Adelaide, Adelaide.

Shin, H., & Cho, S., 2003. How many neighbours to consider in pattern pre-selection for Support Vector classifiers?, in *Proc. Intern. Joint Conf. on Neural Networks (IJCNN 2003)*, pp. 565-570.

Shynk, J. J. (1990). Performance surfaces of a single-layer perceptron. *IEEE Transactions on Neural Networks, 1*, 268-274.

Shynk, J. J., & Bershad, N. J. (1991). Steady-state analysis of a single-layer perceptron based on a system identification model with bias terms. *IEEE Transactions on Circuits and Systems, 38*, 1030-1042.

Shynk, J. J., & Bershad, N. J., 1992. Stationary points and performance surfaces of a perceptron learning model for a nonstationary data model, in *Proc. International Joint Confererence on Neural Networks*, pp. 133-139.

Smieja, F. J. (1991). *Hyperplane 'spin' dynamics, network plasticity and backpropagation learning.* St. Augustin, Germany: GMD.

Solis, F. J., & Wets, J. B. (1981). Minimization by random search techniques. *Mathematics of Operations Research, 9*, 19-30.

Specht, D. F. (1991). A General Regression Neural Network. *IEEE Trans. on Neural Networks, 2*(6), 568-576.

Stevens, C. F. (1994). The Neuron. In M. M. Gupta & D. H. Rao (Eds.), *Neuro-Control Systems: Theory & Applications* (pp. 101-111): IEEE Press.

Sutton, R. S., Barto, A. G., & Williams, R. J., 1991. Reinforcement Learning is Direct Adaptive Optimal Control, in *Proc. 1991 American Control Conference*, pp. 2143-2146.

Suykens, J. A. K., & Vandewalle, J. (1999). Training Multilayer Perceptron Classifiers Based on Modified Support Vector Method. *IEEE Trans. on Neural Networks, 10*(4), 907-911.

Thimm, G., & Fiesler, E. (1997). High-Order and Multilayer Perceptron Initialization. *IEEE Trans. on Neural Networks, 8*(2), 349-359.

Thivierge, J. P., Rivest, F., & Shultz, T. R., 2003. A dual-phase technique for pruning constructive networks, in *Proc. Intern. Joint Conf. on Neural Networks (IJCNN 2003)*, pp. 559-564.

Tivive, F. H. C., & Bouzerdoum, A., 2003. A new class of convolutional neural networks (SICoNNets) and their application of face detection, in *Proc. Neural Networks, 2003. Proceedings of the International Joint Conference on*, pp. 2157-2162 vol.2153.

Tsai, H.-L., & Lee, S.-J. (2004). Entropy-Based Generation of Supervised Neural Networks for Classification of Structured Patterns. *IEEE Trans. on Neural Networks, 15*(2), 283-297.

Tsujinishi, D., & Abe, S., 2003. Fuzzy least squares Support Vector Machines, in *Proc. Intern. Joint Conf. on Neural Networks (IJCNN 2003)*, pp. 1599-1604.

van der Smagt, P. P. (1994). Minimisation Methods for Training Feedforward Neural Networks. *Neural Networks, 7*(1), 1-11.

Vapnik, V. N. (1998). *Statistical Learning Theory*. New York: Wiley-Interscience.

Vapnik, V. N., & Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theoretical Probability and Its Applications, 17*, 264-280.

Verikas, A., Bacauskiene, M., & Malmqvist, K., 2002. Selecting features for neural network committees, in *Proc. Intern. Joint Conf. on Neural Networks (IJCNN '02)*, pp. 215-220.

Verma, B. (1997). Fast Training of Multilayer Perceptrons. *IEEE Trans. on Neural Networks, 8*(6), 1314-1320.

Walpole, R. E., Myers, R. H., & Myers, S. L. (1998). *Probability and Statistics for Engineers and Scientists* (6th ed.). New Jersey: Prentice Hall.

Waschulzik, T., Brauer, W., Castedello, T., & Henery, B., 2000. Quality Assured Efficient Engineering of Feedforward Neural Networks with Supervised Learning (QUEEN) Evaluated with "Pima Indians Diabetes Database", in *Proc. Intern. Joint Conf. on Neural Networks (IJCNN 2000)*, pp. 97-102.

Wasserman, P. D. (1989). *Neural Computing: Theory and Practice*. New York: Van Nostrand Reinhold.

Weigend, A. S., Huberman, B. A., & Rumelhart. D. E. (1990). Predicting the future: A connectionist approach. *International Journal of Neural Systems, 1*(3), 193-209.

Widrow, B. (1962). Generalization and information storage in networks of adaline 'neurons'. In M. C. Yovitts, G. T. Jacobi & G. D. Goldstein (Eds.), *Self-Organizing Systems* (pp. 435-461). Washington DC: Sparta.

Widrow, B., & Hoff, M. E. (1960). Adaptive switching circuits. *IRE WESCON Convention Record*, 96-104.

Wolpert, S., & Micheli-Tzanakou, E. (1993). Silicon models of lateral inhibition. *IEEE Trans. on Neural Networks, 4*(6), 955-961.

Yao, X., & Liu, Y. (1997). A New Evolutionary System for Evolving Artificial Neural Networks. *IEEE Trans. on Neural Networks, 8*(3), 694-713.

Zhang, G. P. (2000). Neural Networks for Classification: A Survey. *IEEE Transactions on Systems, Man and Cybernetics: Part C: Application and Reviews, 30*(4), 451-462.