# Active Learning Algorithms for Multilayer Feedforward Neural Networks

By

**Adebola Adebisi Adejumo**

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

**Master of Science**

**in the Faculty of Science**

AT

**University of Pretoria**

PRETORIA

2 December 1999

*To God Almighty who hath been my Tower of Refuge and Strenght*

# Abstract

Backpropagation (BP) has played a vital role in the resurgence of interest in artificial neural networks (ANNs). Eversince, a lot of research effort concentrated on finding ways to improve its performance. Active learning has emerged as an efficient alternative to improve the performance of multilayer feedforward neural networks. The learner is given active control over the information to include in the training set, and in doing so, the generalization accuracy is improved and the computational cost and complexity of the network are reduced compared to training on a fixed set of data.

While many research effort has been invested in designing new learning approaches, an elaborate comparison of active learning approaches is still lacking. The objective of this research study is to compare and critisize active learning approaches and also to propose a new selective learning algorithm.

This thesis presents a comparison of four selected active learning algorithms. The thesis concentrates on one type of application, namely function and time series approximation.

# Opsomming

Terugwaartspropagering neurale netwerke het 'n belangrike rol gespeel in die oplewing van die belangstelling in kunsmatige neurale netwerke. Verskeie navorsingsstudies konsentreer op die verbetering van die prestasie van neurale netwerke. Aktiewe leer het getoon om 'n effektiewe alternatief te wees om die prestasie van multi-vlak vorentoe-voer neurale netwerke te verbeter. Die leerder word aktiewe beheer gegee oor die inligting wat in die leerversameling ingesluit word. Sodoende word veralgemening verbeter, en die berekeningskoste en -kompleksiteit van die netwerk verlaag in vergeleke met afrigting op 'n vaste leerversameling.

Terwyl vele navorsing gedoen is in die ontwikkeling van nuwe leerstrategieë, is daar 'n tekort aan 'n uitgebreide vergelykende studie van aktiewe leer. Die doelwit van hierdie studie is om aktiewe leer strategieë te vergelyk en te kritiseer. 'n Nuwe selektiewe leer algoritme word ook aangebied.

Hierdie tesis bied 'n vergelyking van vier aktiewe leer algoritmes aan. Die tesis konsentreer op die benadering van funksies en tydreekse.

# Acknowledgements

**I came, I saw and I conquered**

I thank God Almighty for my stay and experience in South Africa, and most especially for His wisdom and guidance when I plunged my self into the unknown ocean of Neural Networks.

I deeply express my thanks to the following people:

- My supervisor, A.P Engelbrecht, for his time and advice.
- My sister and her hubby - Dr and Dr Ogunleye for making my postgraduate study a reality.
- My Uncle and his family - Uncle Akin, for his support and encouragement.
- My family at home especially my mother- For your prayers and encouragement.
- My Daddy - Late Prof J.A Adejumo, you inspired me, taught me the values of education, and that with power of knowledge, all men are equal.
- My Husband - Boye Aderogba, for waiting for me and being there when I needed you, thanks.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# INTRODUCTION

The study of neural networks (NN) is one of the most rapidly expanding fields attracting people from a wide variety of disciplines. The study of neural networks is a field which cuts across many disciplines like philosophy, biology, psychology, mathematics, statistics, neuroscience, physics, engineering and even linguistics [Wasserman 1989]. These interwoven disciplines have made the study of neural networks unique. Neural networks bring together various subjects and disciplines in building intelligent systems.

## 1.1 What is a Neural Network?

The term neural network (NN) in this thesis refers to artificial neural network (ANN) which mimics biological neural systems.

There are several definitions as to what a neural network means: Maren defines neural networks as computational systems, either hardware or software, which mimic the computational abilities of biological systems by using simple *interconnected* artificial neurons [Maren *et al* 1990].

Hecht-Nielsen gives a rigorous definition of a neural network as "a *parallel*, distributed information processing structure consisting of processing elements which can possess a local memory and carry out localized information processing operations, *interconnected* together with unidirectional signal channels called connections. Each processing element has a single output connection which branches (fans out) into as many collateral connections as desired (each carrying the same signal - the processing element output signal). The processing element output signal can be of any mathematical type desired. All of the processing that goes on within each processing element must be completely local, i.e. must depend only upon the current values of the input signal arriving at the processing element via impinging connections and upon values stored in the processing element's local memory " [Hecht-Nielsen 1989].

A simpler definition of a neural network, given by Fausett, is that, a NN is an information processing system that has certain performance characteristics, such as adaptive learning, and parallel processing of information, in common with biological neural networks [Fausett 1994].

Haykins defines a neural network as a massively *parallel* distributed processor that has a natural propensity for storing experiential knowledge and making the knowledge available for use [Haykins, 1994].

A neural network can also be defined as a distributed computational system composed of a number of individual processing elements operating largely in *parallel*, *interconnected* according to some specific topology (architecture) and having the capability to self modify connection strengths and processing elements parameters [Rogas 1996].

From Müller and Reinhardt's view, a neural network model is defined as an algorithm for cognitive tasks, such as learning and optimization, which are in a loose sense based on concepts derived from research into the nature of the brain [Müller *et al* 1990].

From all these definitions, it can be deduced that

- A neural network is inspired by studies of the brain. Though, it would be wrong to say that a neural network duplicates brain functions, because the brain is highly complex and the actual *intelligence* exhibited by the most sophisticated neural network is well below the level of intelligence of any animal [Wasserman 1989].
- A neural network is made of several interconnected units similar to the neurons in the brain.
- A NN is an information processing system that operates in parallel.
- Signals are passed between units over connection links and each link has an associated weight.
- Artificial neurons are simple emulations of biological neurons. Artificial neurons receive information from other artificial neurons or the environment, perform a simple operation by applying functions on these input signals and pass the result to other neurons or the environment.
- Each unit applies an activation function (usually nonlinear) to the net input and determines the unit's output signal.

### 1.1.1 Characteristics Of A Neural Network?

A neural network is characterized by

- the architecture of the NN, which refers to the number of layers in the network, the number of neurons in the layer, and how these neurons are interconnected. Neural network types include single layer networks such as the Hopfield NN [Maren *et al* 1990], multilayer feedforward neural networks (MLNNs) such as backpropagation [Wasserman 1989]and recurrent NNs (RNNs) [Simpson 1990].

- The method of adjusting weights for each connection, referred to as the learning algorithm. Learning algorithms are divided into two main categories, namely supervised and unsupervised learning. Learning in supervised mode is done by comparing the network's output to the desired output, which is provided by the system or external teachers [Simpson 1990]. Learning in unsupervised mode, on the other hand, is by self organization. There is no target or desired output and hence no comparison to predetermined responses [Simpson 1990].

- The activation function used which can be linear, discrete functions such as the ramp function or continuous functions such as the sigmoid function.

The advantages of, and reasons for using neural networks rather than conventional methods of optimization, regression, classification and clustering are discussed in section 1.2.

## 1.2 Why Neural Networks?

The interest in neural networks is motivated by the desire to understand the brain, i.e. the principles on which the human brain works, to emulate some of the brain's strength and the wish to build machines that are capable of performing complex tasks for which the sequentially operating programmable computers are not well suited for.

Everyday observation shows that the brains of even animals of lower intelligence can perform tasks that are far beyond the range of even the largest and the fastest modern electronic computers. For example, dogs bark at human beings that are strangers while they are quite relaxed with human friends. Dogs can distinguish between foes and friends. No present day electronic computer has sufficient computational power to match this similar accomplishment. This accomplishment involves some need for the recognition of complex optical or acoustical patterns which are not determined by simple logical rules

[Müller *et al* 1990].

Neural networks are also used when data, on which conclusions are to based, is fuzzy. When the influential or informative patterns are subtle or hidden, a NN has the ability to discover patterns which are not clear, or unknown to the human researcher or standard statistical methods. An example is credit worthiness of loan applicants based on spending and payment history [Masters 1993]. NNs have been applied to data that exhibits significant unpredictable nonlinearity [Fausett 1994]. NNs have been adapted to predict future values not based on strictly defined models, and offer possibilities for solving problems that require pattern recognition, pattern mapping, dealing with noisy data and pattern completion [Masters 1993].

The advantages of NNs are summarized below:

1. A NN has the ability to learn.
2. Neural networks are robust to noise.
3. Neural networks work excellently for nonlinear data.
4. Because NN can learn to discriminate patterns based on examples and training, an elaborate a priori model is not needed neither is the probability function needed to be specified. The statistical distribution of the data used for training is not needed.

Specific areas where NNs have been applied include: pattern recognition and classification, adaptive control applications, financial analysis such as forecasting and credit assessment, database mining, function approximation and clustering [Fausett 1994, Masters 1993, Wasserman 1989, Towell *et al* 1993].

### 1.2.1 Features of Neural Networks

A very important feature of a neural network is an ability to adapt to changing environments, where learning is by examples. That is, the NN learns how to perform certain tasks by undergoing training with illustrative examples. Once trained, a NN can perform tasks without any external help, even if presented with distorted patterns [Beale *et al* 1990]. This feature makes NNs very appealing especially in application problems where little or no understanding of the problem is known, but where training data which reflects the characteristics of the problem is available. Neural networks can learn various things such as distinguishing a straight line from a convex curved line. The NN can discriminate between the lines once trained, even when the lines are shifted up or down, or even if the data is noisy.

Another feature of neural networks is the *parallel* architecture, which allows faster computation of some problems when the network is implemented on parallel digital computers, or when the network simulates parallelism. Electronic computers are designed to carry out one instruction after the other, extremely rapid whereas the brain work with slower units. A computer is a high speed, serial machine compared to the highly parallel nature of the brain. Computers therefore manage tasks such as counting (an essentially serial activity) which suit its design well, making the computer superior to the brain in such tasks. However, for highly parallel tasks such as vision or speech, computers perform badly. The brain is able to operate in parallel easily and thus is much faster than the computer in performing these tasks.

The approach of NNs in various applications is to capture the guiding principle that underlines the way the human brain solve problems and apply these principles to computer systems.

## 1.3 Background to Neural Networks

Neural networks have been motivated right from their inception by the fact that the brain computes in an entirely different way from the conventional von Neumann machines (computers) [Hassoun, 1995]. The brain is a highly complex, nonlinear and parallel information processing system. The brain has the capability of organizing neurons to perform certain tasks such as pattern recognition, speech recognition, pattern classification many times faster than the fastest digital computer in existence today.

The understanding of this neurobiology has allowed researchers to simulate neural behavior. This idea of simulating neural behavior dates back to the early 40's when one of the abstract models of a neuron was introduced by McCulloch and Pitts. They proposed a general theory of information processing based on networks of binary switches called neurons. These neurons were much simpler than their real biological counterparts. McCulloch and Pitts demonstrated that even simple types of neural networks could in principle, compute any arithmetic or logical function [Hecht-Nielsen 1989].

In 1949, Donalds Hebb proposed a learning rule that explained how a network of neurons learned. He used the learning rule to build a qualitative explanation of some experimental results. This bold step served to inspire many other researchers to pursue the same theme, which further laid the ground work for the advent of neural networks [Hecht-Nielsen 1989].

Rosenblatt invented the perceptron and its learning algorithm in 1958. The perceptron in its simplest form consists of two separate layers of neurons representing the input and output layers. An iterative algorithm for constructing synaptic coupling such that a specific input pattern is transformed into the desired output pattern was introduced. However, the perceptron had a serious shortcoming: it was only capable of solving classification problems that are linearly separable at the output layer [Fu 1994]. At the same time, Widrow and Hoff developed an important variation of perceptron learning known as the Widrow-Hoff rule [Fu 1994].

In the late 60's, Minsky and Papert caused research in NNs to be terminated with their results published in their landmark book called Peceptron [Hecht-Nielsen 1989]. Minsky and Papert pointed out the theoretical limitations of single layer neural network models. They proved that the perceptron cannot implement the exclusive or (XOR) logical function. The perceptron also had difficulty in learning other binary predicate functions. The implicit conclusion from their book was that essentially all neural networks suffer the same fatal flaw as the perceptron and they left the impression that neural network research was a dead end [Hecht-Nielsen 1989]. Due to this pessimistic work, research on neural network lapsed into an eclipse (a dark age for neural network research) for nearly two decades [Fu 1994].

Despite this, a few *faithful* researchers still continued their work on NNs and produced meaningful results during this period. For example, Anderson and Grossberg did important work on the psychological models [Hecht-Nielsen 1989]. Kohonen invented the self organising map (SOM), an associative memory model [Fu 1994].

In the early 80s, after two decades of obscurity, there was a renewed enthusiasm in the neural network field. A notable researcher who increased the visibility and respect for NN study is Hopfield. In 1982, Hopfield introduced the idea of energy minimization in physics to neural networks [Hopfield 1982, Fu 1994].

In the mid 80s, Rumelhart, Hinton and Williams developed a learning algorithm for multilayer networks called the backpropagation algorithm (BP) [Wasserman 1989]. This algorithm offered a powerful solution to training a multilayer neural network and hence counters the implicit conclusion of Minsky and Papert. Their development of multilayer feedforward networks was not restricted to linearly separable training sets. Along with a reasonably effective training algorithm for NNs, Rumelhart *et al* demonstrated that neural networks can provide real solutions to practical problems [Rumelhart *et al* 1986. Masters 1993]. Problems such as the XOR and lack of a general method of training a multilayer neural network, which had originally contributed to the demise of neural networks in the 60s, were overcome using the backpropagation algorithm and other techniques which followed

[Wasserman 1989]. It is interesting to note that Werbos had developed the idea of backpropagation in 1974 and also Parker in 1982 independently [Maren *et al* 1990].

A spectacular success of backpropagation is demonstrated by the NETTALK system developed by Sejnowski and Rosenberg in 1987. NETTALK is a system that converts English text into highly intelligible speech [Wasserman 1989]. The backpropagation algorithm is probably the most well known and widely used training algorithm [Maren *et al* 1990]. Much research effort was expended to improve backpropagation. The objective of this study is to further study methods to improve BP. Approaches and specific research to improve the performance of NNs using BP include finding optimal weight initialization [Wessels *et al* 1992], optimal learning rate and momentum [Yu *et al* 1997, Weir 1990], finding optimal architectures [Engelbrecht *et al* 1996. Hassibi *et al* 1994, Le Cun 1990, Karnin 1990, Sietsma *et al* 1988], using second order optimization techniques [Becker *et al* 1988], adaptive activation functions [Fletcher *et al* 1994, Engelbrecht *et al* 1995, Zurada 1992a] and active learning [Röbel 1994a, Zhang 1994, Engelbrecht *et al* 1999a].

A large number of neural networks are trained using the gradient descent optimization method in the supervised mode. In order to train the network successfully, the output of the network is made to approach the desired output by continually reducing the error between the network's output and the desired output. Training a NN is achieved by presenting the network with information to learn, which consists of a fixed set of input attributes and corresponding target outputs. The weights between the layers are then adjusted using an optimization algorithm, usually the gradient descent optimization, the error is computed and backpropagated from one layer to the previous layer. But presenting all the available data to the network can be problematic, especially when there are redundant data in the training set. The computational expense in terms of training time and the complexity can be unnecessarily high if all the data are used for training.

Studies have shown that selecting the most informative data, rather than training on

all the available data, improves, or at least maintains the generalization performance, as well as reduces the training cost, and the data needed for training [Engelbrecht *et al* 1998, Engelbrecht *et al* 1999a, Röbel 1994a, Zhang 1994]. Active learning refers to such selection of a subset of the available training data containing the most informative patterns for training. The concept of active learning is to efficiently select high utility patterns from available data for training the network. There are two approaches to active learning, namely incremental and selective learning.

This thesis focuses on the study of active learning as a method of improving performance of NNs on function approximation and time series problems. Section 1.4 discusses the objectives of this study.

## 1.4 Objective and Justification

The backpropagation learning algorithm played a vital role in the resurgence of interest in neural networks. Eversince, a lot of research effort has been concentrated on finding ways to improve the performance of backpropagation learning. Research has concentrated on finding the *optimal* size of networks, to make *optimal* use of training data, to *optimize* initial weights and learning parameters.

This thesis concentrates on methods to optimize the use of training data, i.e. *active learning*. A new selective algorithm for time series problems is proposed and is used as one of the selected active learning algorithms to be compared. A comparative study is carried out on three additional active learning algorithms. While many research efforts have concentrated on designing new active learning approaches, as well as other learning algorithms, an elaborate comparison of these approaches is still lacking and hence the motivation for this study. The four selected active learning algorithms are compared to each other with reference to their respective performances in terms of accuracy, computational complexity

and convergence characteristics.

Accuracy of a learning algorithm is how well a function is approximated by the network using the algorithm. The mean squared error (MSE) on the training set and the test set are used as the measure of accuracy. The training error is the error computed over all the patterns or data presented to a network for training, while the generalization error is the error computed over a set of patterns not used for training a network i.e. test set. A low generalization and training error is an indication of good approximation of the problem and a good performance of the network. However, a low training error and a large generalization error is an indication that the training set is overfitted. A MSE value close to zero shows a small error between the target and the output function. Computational complexity measures the cost of training the network. The cost is measured by the number of calculations made during training. The number of patterns selected for training is quite important because of the proportional relationship between computational cost and the number of patterns. The more patterns selected for training, the more calculations are made during training and thus, a higher training cost. Based on these criteria, a critique of the different algorithms as well as suggested future work are discussed.

The scope of this thesis is *multilayer feedforward neural networks*, focusing on function approximation and time series problems. Gradient descent is used as optimization method and sigmoid activation functions are used. A three layer neural network with one input layer, one hidden layer, and one output layer is used.

## 1.5 Outline

The rest of this thesis is organized as follows:

- Chapter 2 deals with learning in multilayer neural networks. A general introduction and a background study of multilayer neural networks are given. The architectures, learning algorithms and weight updating methods are discussed. The difficulties of training multilayer neural networks, as well as solutions to these difficulties are discussed.

- Active learning is discussed in chapter 3. The concept and the basis for active learning are examined. Results and simulations of the four selected active learning algorithms are presented.

- Chapter 4 concludes this thesis with observations and suggestions for future research.

# Chapter 2

# MULTILAYER NEURAL NETWORK LEARNING

This chapter discusses learning in multilayer neural networks (MNNs). MNNs are by far the most common applications of artificial neural networks (ANNs). The chapter covers fundamental issues such as the different types of MNNs and available learning algorithms. Performance aspects of the different learning algorithms are discussed, as well as difficulties encountered in the learning process.

## 2.1 Introduction

An artificial neural network (ANN) is a model of the biological neural system of human beings, modeling one of the most important features of the brain - *the ability to learn*. This feature shows parallel to the intellectual development of human beings. As human beings,

we learn how to write, read, understand speech, recognize and distinguish pattern - all by *learning from examples.* In the same way, ANNs are *trained*, rather than programmed. ANNs develop solutions to problems unlike conventional data processing techniques which require complex programming.

An artificial neural network consists of processing units, organized in layers of units (also referred to as artificial neurons). Training of an ANN is done using a training algorithm, which is an adaptive way by which a network of processing units organizes themselves to implement the desired behavior: When a network is presented with information to learn (consisting of input attributes and corresponding desired output values). the connection links in between, referred to as the weights, are adjusted to produce a response consistent to the desired output. This learning algorithm is a closed loop of presentation of patterns or examples and of corrections to the network according to a learning rule. An optimization algorithm such as gradient descent, conjugate gradient or second order derivatives techniques, is used to adjust the weights of the network [Becker *et al* 1988]. There are different classes of training algorithms and different topologies of artificial neural networks.

The rest of this chapter is organized as follows: The parallelism between biological and artificial neural networks is discussed in section 2.2 to show how ANNs were inspired from the biological counterpart. A taxonomy of different neural network training algorithms is given in section 2.3. Section 2.4 discusses the training of multilayer neural networks using gradient descent. The learning equations are derived in this section. Section 2.5 discusses problems of learning by gradient descent.

## 2.2 Biological Neural Networks

The basic building block of biological neural systems is the neuron. A neuron is a cell which communicates information to and from the various parts of the human body. Figure 2.1

shows a simplified representation of a neuron. A neuron consists of a cell body referred to as *soma*, several spline-like extensions of the cell body referred to as *dendrites* and a single nerve fiber referred to as an *axon*. An axon branches out from the soma and connects to many other neurons.

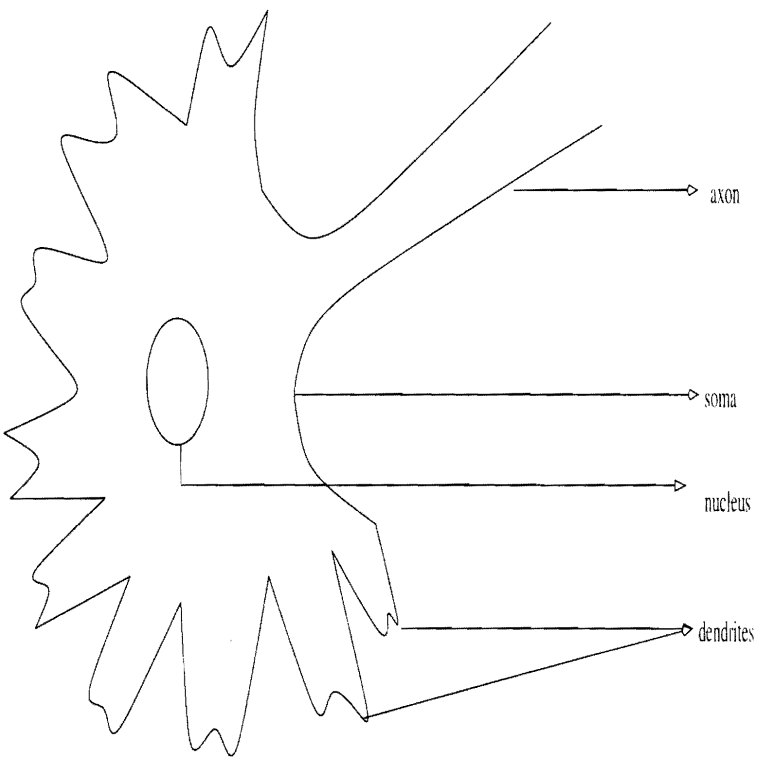



Figure 2.1: A simplified representation of a biological neuron

Dendrites extend from the cell body to other neurons where the dendrites receive signals at a connection point referred to as a *synapse*. These signals serve as inputs which are conducted to the soma (cell body). In the nucleus, these received inputs are summed up. If the cumulative excitation in the nucleus exceeds a threshold, the neuron fires, sending signals down the axon to other neurons. While the biological neural system is extremely complex, an ANN is an attempt at modeling the information processing capabilities of the biological neural system.

An artificial neuron was designed to mimic simple characteristics of the biological neuron. An artificial neuron receives input signals from the environment, or from other artificial neurons. These inputs signals are weighted with a value which models the synaptic strength of the corresponding connection. The weighted sum of the input signals is used to determine the activation level of the neuron. The activation of an artificial neuron is modeled using an activation (or transfer) function. The different activation functions are discussed in section 2.3.2.

Figure 2.2 illustrates a general representation of an artificial neuron. In the rest of this

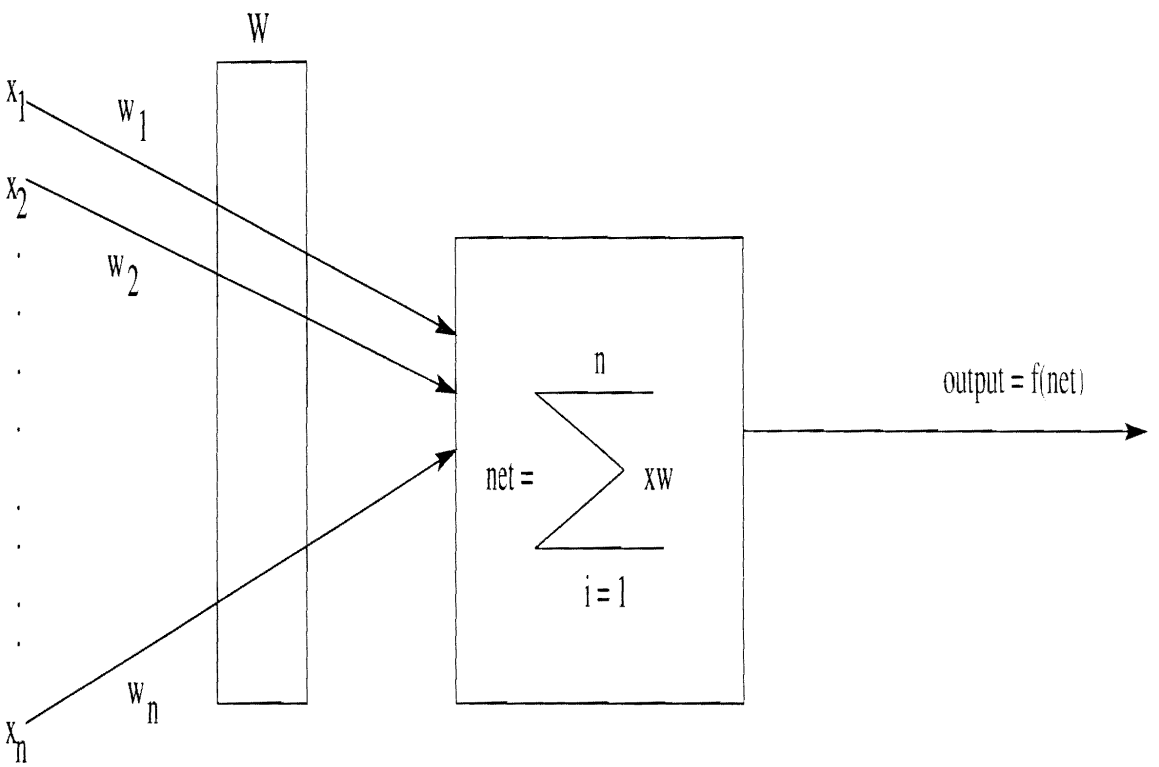



Figure 2.2: An artificial neuron

thesis, the term neural network (NN) is used instead of artificial neural network (ANN).

Several key features of the processing elements of a neural network are suggested by the properties of the biological neuron, namely that,

- a processing unit (neuron) receives many signals from other neurons or the environment;
- these signals may be modified by a weight;
- the processing units sum the weighted inputs which is transformed to an output signal using a squashing function to simulate firing;
- the neuron transmits this single output to other neurons, or to the environment; and
- the output from a particular neuron may be transmitted to many other neurons.

One important characteristic an ANN shares with biological neural systems (BNS) is fault tolerance. A BNS is fault tolerant in two ways: Firstly, human beings are able to recognize many input signals that are somewhat different from any signals they have seen before. Secondly, a BNS can tolerate damage to itself. Human beings are born with as many as 100 billion neurons. Most of these neurons are located in the brain and are not replaced when neurons die [Fausett 1994]. Despite the loss of these neurons, human beings still continue to learn. Even in cases of traumatic neural loss, other neurons can sometimes be trained to take over the function of the damaged cells [Fausett 1994]. In a similar manner, an ANN can be designed to be insensitive to small damage to the network and the network can be retrained in cases of significant damage.

The number of layers, and the way in which neurons are interconnected, resulted in the design of various ANN topologies. Section 2.3.1 surveys different ANN topologies and also discusses the different classes of training available.

## 2.3 A Taxonomy Of Training

One of the interesting features of neural networks is their ability to learn, which implies that the NN has to be trained. *How is this done?*

The objective of training a NN is to produce desired (or at least consistent) output when a set of inputs is applied to the network. A neural network is trained by applying an input vector to neurons while adjusting the weights according to a predetermined procedure in order to bring the NN's learned concept closer to the desired output. During training, weights gradually converge to values such that each set of input patterns produces a close approximation to the desired output patterns. There are two main training paradigms:

1. **Supervised training,** which is perhaps the most frequently used training method. For training purposes, a *training pattern* is required which consists of a vector of input values and a vector of associated target/desired output values. Patterns can be provided by external teachers or by the system which contains the network, in which case the network is self supervised. The network is usually trained by presenting an input vector to the NN, the actual output of the NN is calculated and compared to the corresponding desired (target) output. Training patterns are grouped into a training set. Each pattern in the training set is presented to the network, and the prediction error used to adjust weights. Patterns in the training set are repeatedly presented to the network until an acceptable error is achieved over the entire training set.

   Supervised learning is analogous to a lesson in school where the teacher applies the correct answer for each problem. Different approaches to supervised learning have been developed.

   - *Error correction learning* which adjusts the connection weights between processing units, in proportion to the difference between the desired and computed values of each neuron in the output layer [Simpson 1990].
   - *Reinforcement learning* which is similar to error-correction learning in that weights are reinforced for properly performed actions and punished for poorly performed actions [Simpson 1990].

   The difference between error correction and reinforcement learning, is that, error

correction learning requires an error value for each output unit while reinforcement learning requires only a state to describe the output layer's performance.

2. **Unsupervised training**, also referred to as self-organization learning, requires no target or desired outputs. Hence, no comparison to predetermined responses are needed. Training sets consist solely of input patterns. The task of the NN is to learn to group together patterns that are similar and also to find common threads in a mass of data. The NN is supposed to discover statistically salient features of input patterns and develop its own representation of these patterns. Unsupervised learning is used for tasks such as clustering [Fausett 1994].

For the purpose of this thesis, only supervised training is considered.

### 2.3.1 Topology of Neural Networks

In addition to the classes of neural network training algorithms, another distinguishing characteristic of the different neural networks is topology. Topology refers to the architecture of neurons, including the interconnection scheme within the network.

Neurons are arranged in one or more than one layer. Neurons within the same layer usually have the same activation function, and are fully connected to the neurons in the next layer. A NN can consist of just a single layer of fully interconnected units, or can have an input and an output layer with zero or more hidden units, referred to as a multilayer neural network (MLNN). Figure 2.3 illustrates a MLNN with a hidden layer. The figure has three units in the input layer with a single output unit. The input layer consists of units that receive input signals from the environment and distributes the signals to the other layers in the network. The output layer returns signals to the environment. Hidden layers are those layers in between the input and output layers. The hidden units provide nonlinearities for the network.

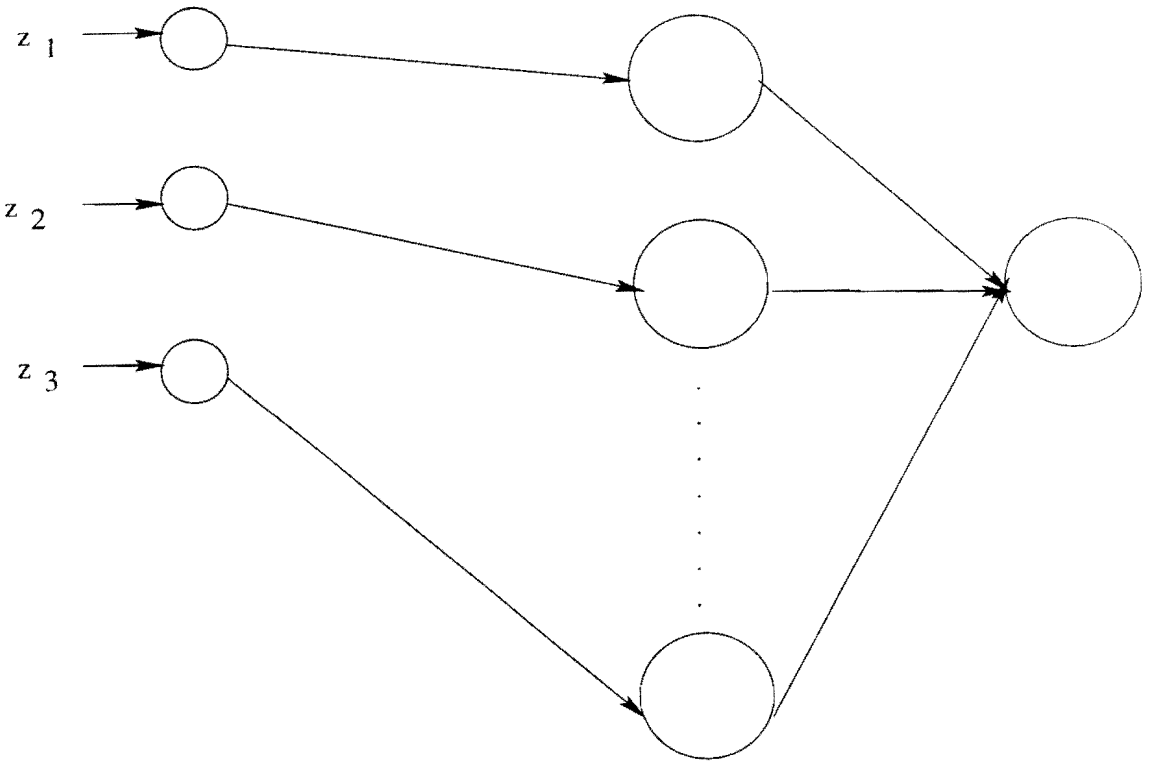



Figure 2.3: A multilayer neural network with a hidden layer

Each neuron produces an activation value (output signal) which usually is a function of the weighted sum of the input signals. The activation value represents the activation level for the neuron. Section 2.3.2 discusses activation functions that can be used in a NN.

### 2.3.2 Activation Functions

The basic operation of an artificial neuron (unit) involves summing the neuron's weighted input signal and to produce an output signal through application of an activation function to the net input signal. Activation functions map a neuron's domain, which is the input, to a prespecified range - the output. Figure 2.2 illustrated the basic building block of a NN. In figure 2.2 *net* is the weighted input signal. The output signal $o$ is calculated as

$$o = f(net) \tag{2.1}$$

Various mathematical functions have been used as activation functions. There are functions that squashes the net input signal into a finite range. These functions can be discrete functions, such as the ramp and step functions, or continuous functions, for example the arctangent, sigmoid, sine or gaussian (radial basis). Linear functions can also be used as activation functions, in which case the input signal is not mapped into a finite range. Figure 2.4 illustrates the different activation functions that can be used.

One of the major reasons why earlier work on NNs came to a halt, was that the learning rule could not be substantially improved for multilayer NNs using the discrete and linear activation functions [Maren *et al* 1990]. Linear and discrete functions could only solve problems that are linearly separable, and being linearly separable limits the NN to problems (classification) in which the sets of points (corresponding to input values) can be separated geometrically. Hence, the network used then (perceptron) could not solve the XOR problem.

A new learning rule (backpropagation) was developed to handle linearly inseparable functions. However, backpropagation requires continuous, monotonic increasing activation functions, since these functions need to be differentiated when the gradient of the error surface is calculated during the weight update process.

The sigmoid function, given in equation (2.2), is widely used as activation function and is a continuous function bounded in the range (0,1). The sigmoid function is expressed mathematically as:

$$f(net) = \frac{1}{1 + e^{-net}} \tag{2.2}$$

The sigmoid function is desirable because of its simple derivative. The sigmoid function has the advantage of providing a form of automatic gain control. That is, for small signals ($net$ near zero), the slope is steep producing high gain in the magnitude of the network's output and as the magnitude of $net$ increases, the gain in the magnitude of the network's output decreases. In this way, large input signals can be accommodated by the network without saturation, while small signals are allowed to pass through without excessive attenuation.

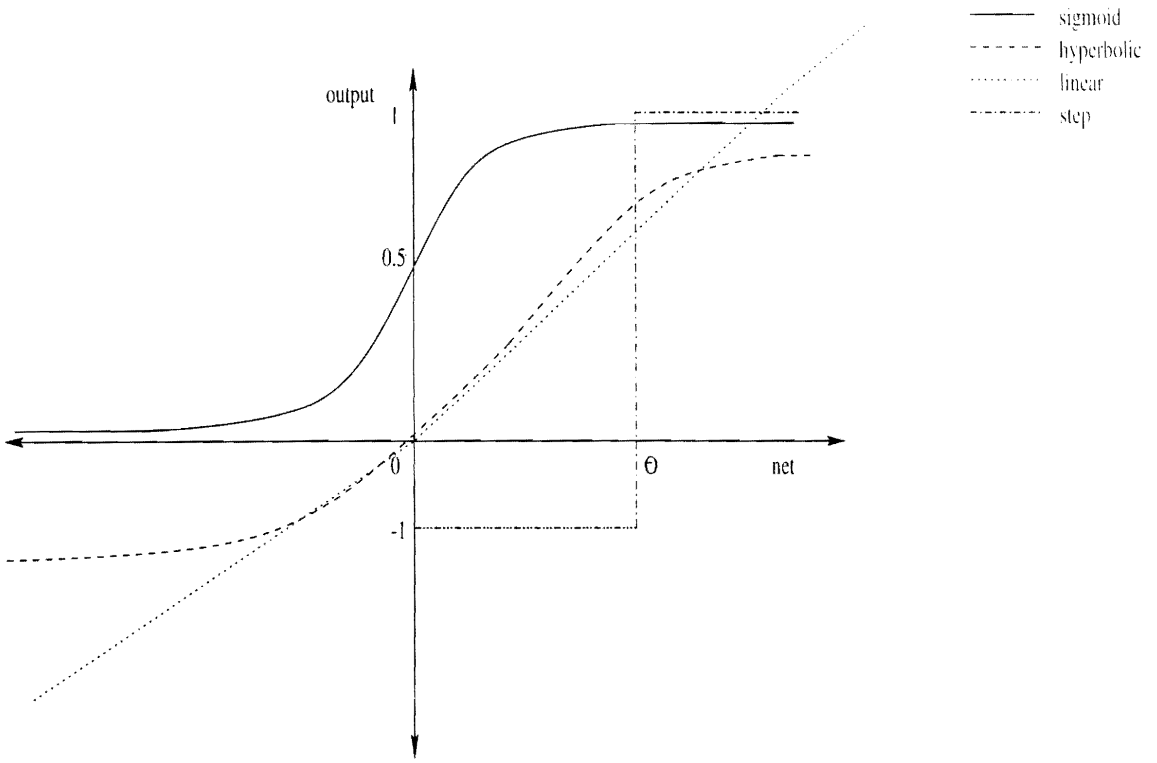


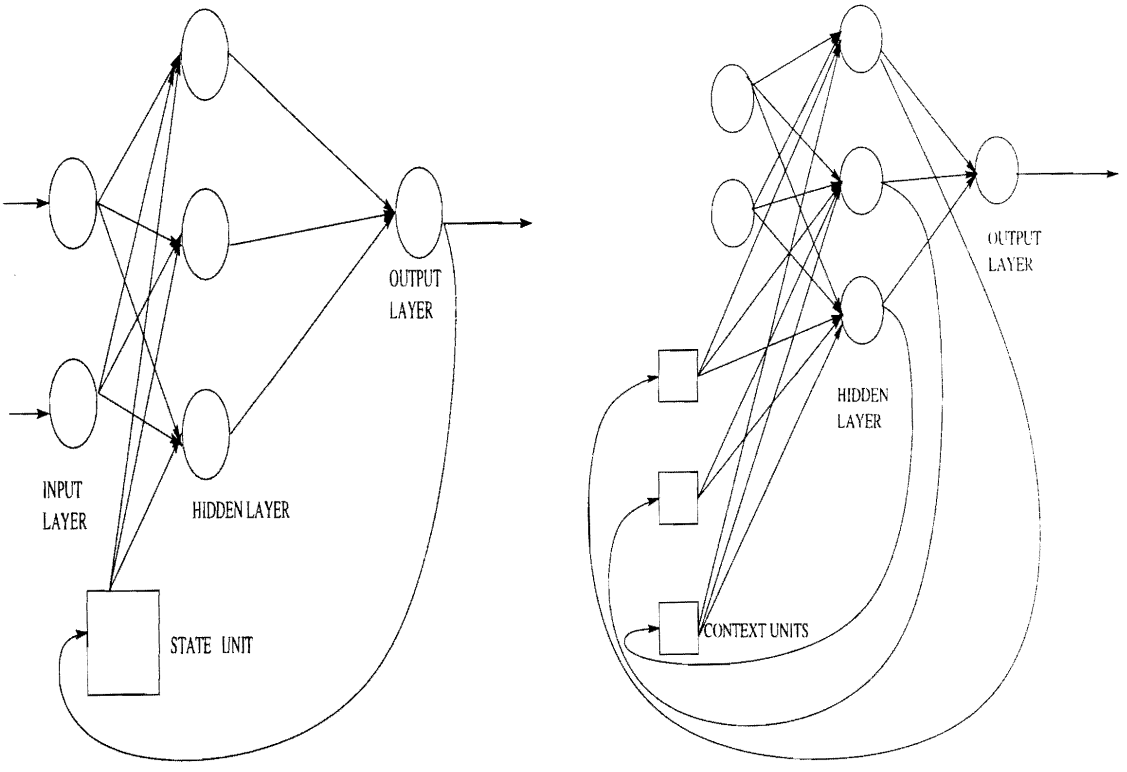

Figure 2.4: Activation functions

### 2.3.3 Neural Network Types

Based on the different network topologies and training approaches, different types of NNs have been developed. A summary of the different NN types are presented below:

1. **Recurrent neural network (RNN)**: A RNN, also referred to as a feedback neural network, employs feedback connections in order to learn temporal characteristics of data presented for learning. The feedback connections thus allow the network to produce complex time varying outputs in response to simple static input [Carling 1992]. RNNs exhibit properties very similar to short term memory in human beings. There are different types of RNNs, e.g. Jordan and Elman RNNs.

   In Jordan RNNs, the state of the output layer is fed back to state units in the input layer (see figure 2.5(a)), while the state of the hidden layer is copied into context




(a) Jordan RNN (b) Elmann RNN

Figure 2.5: Recurrent Neural Networks (RNNs)

units in the input layer for Elman RNNs (see figure 2.5(b)). Hybrid networks can also be built by combining Jordan and Elman networks. Also, any number of previous time steps can be incorporated by simply having additional state units (for Jordan RNN) and context units (for Elman RNN) for each time step [Carling 1992].

2. **Functional link neural network (FLNN)**: In a FLNN, the input layer is expanded to a layer of functional units, which consists of higher order combinations of the input units [Zurada 1992b, Hussain *et al* 1997]. Each functional unit is fully connected to the next layer. The addition of higher order combinations of inputs artificially increases the dimension of the input space. Figure 2.6 shows an illustration of a functional link neural network.

3. **Product unit neural network (PUNN)**: PUNNs allow learning of higher-order

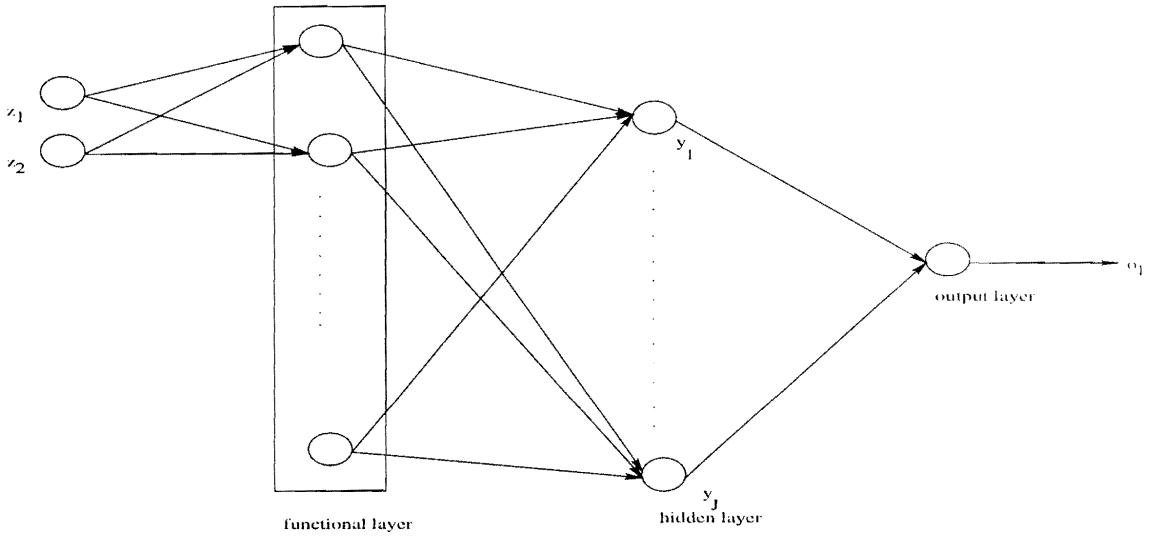



Figure 2.6: A functional link neural network (FLNN)

input terms, by using product units instead of summation units to compute the net signal to a neuron [Durbin *et al* 1989].

A weighted product

$$\prod_{i=1}^{I} z_i^{v_{ji}}$$

is therefore used instead of the usual weighted sum

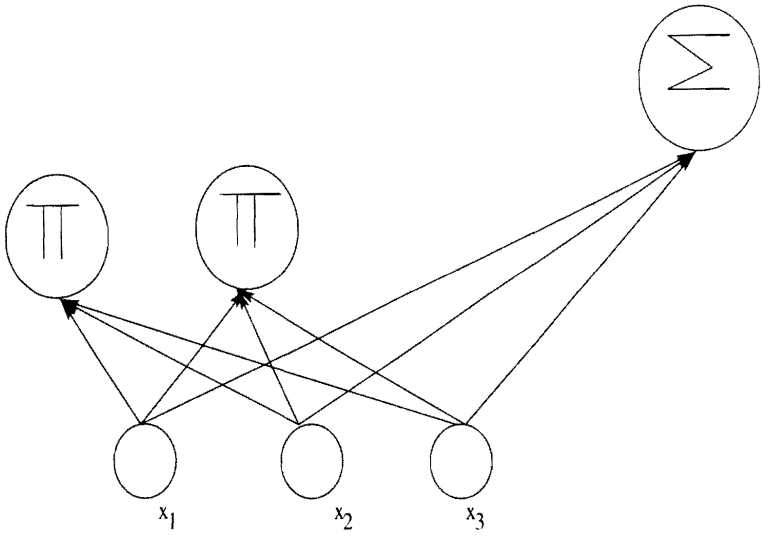
$$\sum_{i=1}^{I} z_i v_{ji}$$

where $z_i$ is the input signal to neuron $j$, $v_{ji}$ is the weight between neuron $i$ in the previous layer and unit $j$. Durbin and Rumelhart proposed two PUNN architectures (refer to figure 2.7):

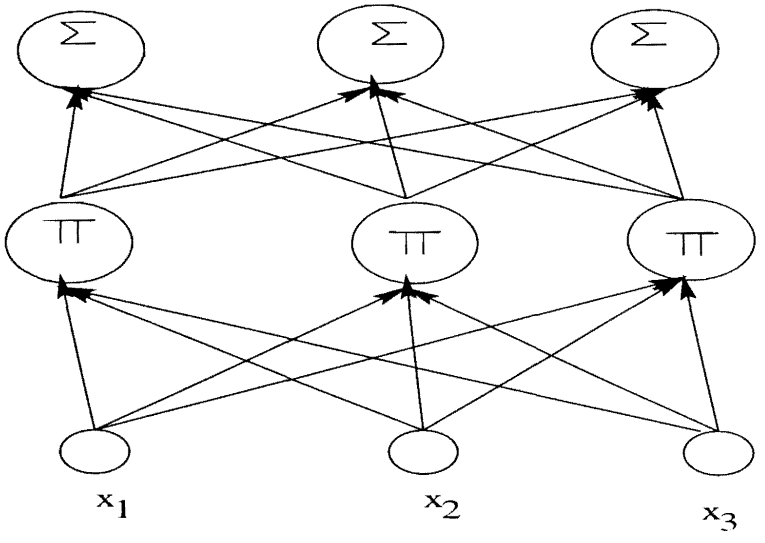
(a) In the first architecture, a set of product units is added to the current summation units in the hidden layer (refer to figure 2.7(a)).

(b) In the second arrangement, layers of product units alternate with layers of summation units (refer to figure 2.7(b))

The main reason for using PUNNs, is to learn to represent generalized polynomial




(a) The first arrangement




(b) The second arrangement

Figure 2.7: Product Unit Neural Networks

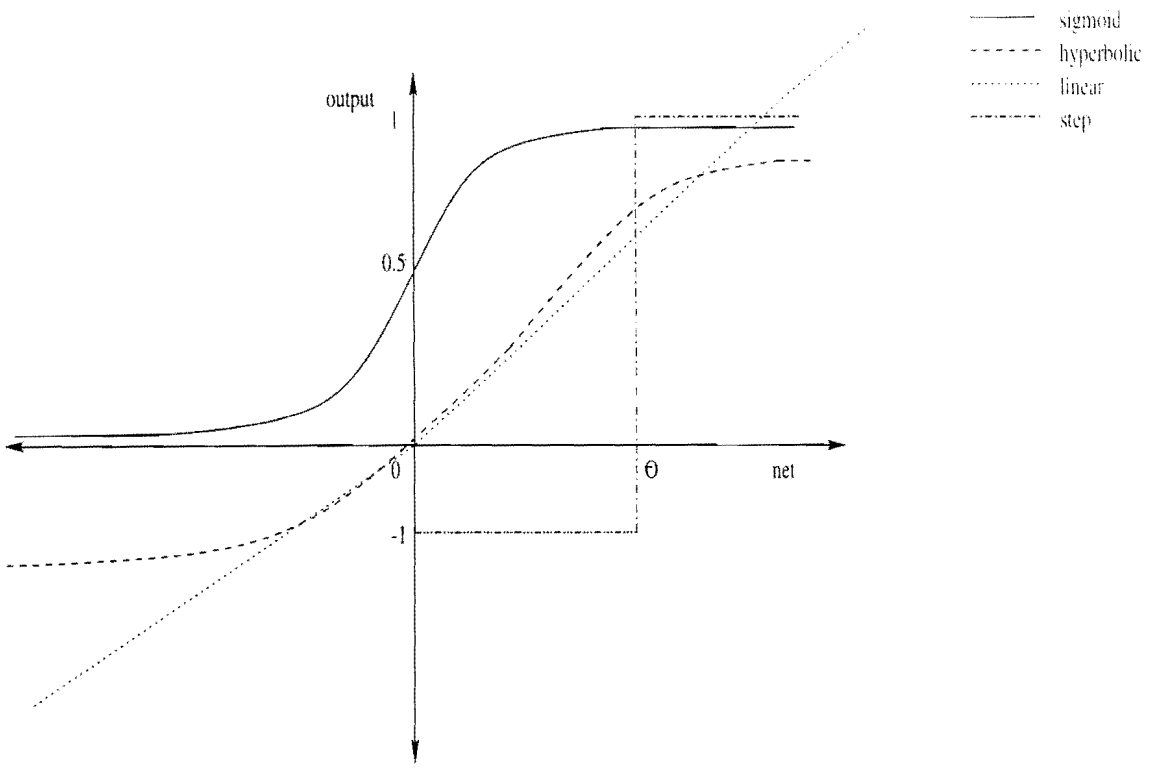



Figure 2.4: Activation functions

### 2.3.3 Neural Network Types

Based on the different network topologies and training approaches, different types of NNs have been developed. A summary of the different NN types are presented below:

1. **Recurrent neural network (RNN)**: A RNN, also referred to as a feedback neural network, employs feedback connections in order to learn temporal characteristics of data presented for learning. The feedback connections thus allow the network to produce complex time varying outputs in response to simple static input [Carling 1992]. RNNs exhibit properties very similar to short term memory in human beings. There are different types of RNNs, e.g. Jordan and Elman RNNs.

   In Jordan RNNs, the state of the output layer is fed back to state units in the input layer (see figure 2.5(a)), while the state of the hidden layer is copied into context

terms in the input and hence a better representation of data in cases where higher order combinations of inputs are significant [Leerink *et al* 1995]. Adjusting the weights is, however, computational expensive since derivatives of these product units are complex due to an exponential term and the occurrence of complex numbers.

4. **Feedforward neural network (FFNN)**: In a FFNN data flows strictly from the input layer to the output layer. A FFNN has no memory and the output is solely determined by the current input and weights values. A feedforward neural network consists of one or more layers of usually non-linear processing units (can use linear activation functions as well). The output of each layer serves as input to the next layer. This thesis concentrates on FFNNs, and studies network learning using FFNNs as well as problems associated with learning in FFNNs.

Apart from the neural network types mentioned above, there are other NN types: for example the single layer Hopfield NN (HNN) [Hopfield 1982, Fausett 1994], and clustering NNs, for example the self organizing map (SOM), which use unsupervised learning [Simpson 1990].

Section 2.3.4 discusses optimization algorithms that can be used to adjust the weights of feedforward neural networks.

### 2.3.4 Optimization Algorithms

Training a neural network involves finding optimal values for the weights of the network through numerical optimization of a nonlinear objective function. The objective function is usually the sum squared error, computed from the actual network output and the desired output of the NN to be trained. Different optimization algorithms can be applied to NN learning. The algorithm chosen is usually based on the characteristics of the problem to be solved.

1. **Gradient descent optimization** is by far the most common technique used for weight optimization. In training the network, a gradient descent is performed on the error function, which is a function of the weights of the neural network. Weights are adjusted to move towards the negative gradient of the objective function [Masters 1993, Becker *et al* 1988]. Gradient optimization is discussed in more details in the next section.

2. **Newton optimization** uses a better approximation of the error function than the gradient descent technique. The newton technique uses second derivatives and gradient information of the error function to determine the next step direction. This helps in reducing the number of steps taken to reach a minimum, thus achieving faster convergence. However, Newton optimization has the disadvantage of being computationally expensive because the inverse of the Hessian matrix needs to be calculated at each training step. Newton's optimization should preferably be used with neural networks with a few number of weights due to the cost of computing the inverse of the Hessian matrix[Darken *et al* 1992, Becker *et al* 1988].

3. **Pseudo newton optimization** is an adaptation of Newton's method. Pseudo newton optimization computes an approximation to the inverse Hessian matrix, and is therefore more computationally efficient than Newton's optimization. Pseudo newton optimization should be preferably used for neural networks with a moderate number of weights due to the cost of approximating the Hessian matrix [Darken *et al* 1992].

4. **Conjugate gradient optimization** is used for large optimization problems, since it does not require the computation and storage of the Hessian matrix. Conjugate gradient uses only gradient information. The objective of conjugate gradient is to minimize both the weight vector and a direction vector. Conjugate gradient is related to gradient descent optimization using momentum, because the weight search in conjugate gradient optimization combines the new gradient direction and the previous gradient direction. Each step involves computing a conjugate direction followed by

a line search, to get an approximate minimum in the conjugate direction. Conjugate gradient optimization increases speed of training and the convergence of the network [Becker *et al* 1988, Møller 1993].

5. **Simulated Annealing** can be used where the objective function (the error function in neural network training) is not differentiable. Optimization is performed by randomly perturbing the independent variables (inputs in this case) and keeping track of the best (lowest error) function value for each randomized set of variables. Simulated annealing can be combined together with other optimization algorithms such as conjugate gradient, where simulated annealing is used to find a good initial weight vector, after which conjugate gradient is used to find the local minimum [Masters 1993, Desai *et al* 1996].

### 2.3.5 Why Neural Networks?

Neural network applications emphasize areas where NNs appear to offer a more appropriate approach than traditional computing has. NNs can be used when data, on which conclusions are to be based, is noisy.

When the influential or informative patterns are subtle or hidden, a neural network has the ability to discover patterns which are not clear, or unknown, to the human researcher or standard statistical methods. For example, to determine the credit worthiness of a loan applicant, the information needed is hidden within data on the spending and the payment history of loan applicants. NNs have shown to provide decisions superior to those made by human beings [Masters 1993]. Neural networks have also been applied to data that exhibits significant unpredictable nonlinearity [Masters 1993]. NNs adapt to predict future values not based on strictly defined models, and offer possibilities for solving problems that require pattern recognition, pattern mapping, dealing with noisy data, pattern classification and

function approximation.

Specific areas where NNs have been applied include, amongst others:

- Neural networks have excelled in **pattern recognition**. NNs deal with the complexities inherent in many applications such as recognizing patterns in speech, radar and seismic readings. A real world application is the NETTALK, a neural network designed by Sejnowski and Rosenberg to produce phonetic strings which in turn specify pronunciation for written texts [Dayhoff 1990].

- NNs are used for **pattern classification**. Input patterns of a network are mapped into one or more classes. That is, each pattern belongs to one of the classes [Fausett 1994]. For example, NNs are used for medical diagnosis to identify diseases of the heart from electrocardiograms. NNs can also be used in plant classification to determine crop types from satellite photographs [Masters 1993].

- NNs have also been used in **adaptive control** applications such as in robots and automatic vehicles. Neural networks are used to control robots in the industry [Dayhoff 1990].

- Neural networks are used in **financial analysis** problems such as credit assessment and financial forecasting. NNs have also find application in optimization, scheduling and routing problems. A practical application is in optimizing resources for airlines [Dayhoff 1990].

- NNs are used in **function approximation** problems. A NN can learn a given function or time series problem when presented with training patterns representing that function or time series. This application has found its usefulness in forecasting, such as weather and in the stock exchange market.

- Neural networks are used for **database mining**. A major problem which surfaced in information retrieval is that explicit information can easily be retrieved while

implicit information can not. Implicit information is distributed across the patterns stored in the database and is difficult to extract by human experts. NNs are one of the most promising technologies available to extract such implicit information - a process referred to as data mining. [Towell *et al* 1993, Fu 1994].

## 2.4 Gradient Descent Optimization

Multilayer neural networks (MLNNs) perform excellently in most applications, especially in classification problems because of the inclusion of one or more hidden layer. Training a MLNN is not as straight forward, nor as easy, as training a single layer network. This section discusses training of MLNNs using gradient descent. Complete derivationΦs of the learning equations are given and problems with gradient descent optimization are discussed.

### 2.4.1 Introduction

NNs that are trained using GD are referred to as backpropagation neural networks (BPNNs). In order to train the network successfully, the output of the network is made to approach the desired output by continually reducing the error between the network's output and the desired output. This is achieved by adjusting the weights between layers: by calculating the approximation error and backpropagating this error from the final layer to the first layer. The weights are then adjusted in such a way to reduce the approximation error. The approximation error is minimized using the gradient descent optimization technique [Rogas 1996].

The gradient descent technique searches for the minimum of the error function in the weight

space. The combination of weights which minimizes the error function is considered to be the solution to the learning problem. When an input pattern is presented to the network, the network produces an output $o_k^{(p)}$ for output unit $o_k$ which is different from the target value $t_k^{(p)}$.

The objective of training is then to minimize the error arising from these two values over the entire training set. The error function is defined as the sum squared error function (SSE):

$$E = \frac{1}{2}\sum_{p=1}^{P}\sum_{k=1}^{K}\left(t_k^{(p)} - o_k^{(p)}\right)^2 \tag{2.3}$$

where $P$ is the total number of patterns in the training set, $K$ is the total number of output units, $t_k^{(p)}$ is the target value for $k$th output unit for pattern $p$, and $o_k^{(p)}$ is the output value for the $k$-th output unit for pattern $p$.

The gradient for the error function is computed and is used to adjust the weights. Weight adjustment can be done in two ways:

- **Batch training** which adjusts and updates the weights after presenting a number of training patterns. Weight changes are accumulated and applied once only. Batch training is also referred to as offline training.
- **Online Training** where the weights are adjusted after each pattern presentation. Online training has the advantage of not needing a separate memory to store the derivatives of patterns as is needed by the offline training.

Training using GD involves two passes:

1. **The forward pass:** During the forward phase, each input unit $z_i^{(p)}$ receives an input signal and distribute this signal to the hidden units $y_j$ for all $j = 1, \ldots, J$. Each hidden unit then computes its activation and sends the activation signal to each output unit at the output layer or to hidden units in the next hidden layer if

there are more than one hidden layer. As there are no connections within a layer, all the units in that layer can have their output computed in parallel, while the layers are dealt with in sequential order. The output layer provides the response of the network for a given input pattern.

2. **The backward pass:** Each output unit compares its computed activation $o_k^{(p)}$ with its target value $t_k^{(p)}$ to determine the associated error for that pattern. The error is backpropagated to all units in the previous layer and is used to update the weights between the output and hidden layers. The accumulated error at each hidden unit is then calculated, and backpropagated to adjust the weights between the input and hidden layers. The error value associated with each processing unit reflects the error of that unit. A larger error value indicates that a larger correction will be made to the corresponding weights.

### 2.4.2 Gradient descent training algorithm

Certain aspects have to be addressed before commencing training of multilayer networks. One important aspect is the activation function used in the hidden and output layers. GD requires the activation function to be continuous, differentiable and monotonically increasing. For the purposes of this thesis the logistic (sigmoid) function is assumed. Another issue is the data set: the output value of logistic function is always in the range (0,1), thus requiring scaling of the desired output (target) before training to fit into this range. Though it is not required to scale inputs, it is advisable to scale the inputs to [-1,1] if logistic function is used. The input values will then lie within the active range of the sigmoid function. The number of hidden layers also has to be considered. Although gradient descent can be applied to any number of layers, it has been shown that a single layer of hidden units is sufficient to approximate any function with many discontinuities to

arbitrary precision provided that the activation function is non-linear [Krose *et al* 1993]. This thesis assumes a single hidden layer.

The training algorithm is summarized below:

1. Weight initialization: Set all weights to small random values. Let $v_{ji}$ be the weight between the $j$-th hidden unit and $i$-th input unit, and $w_{kj}$ the weight between the $k$-th output unit and $j$-th hidden unit.

2. Calculate the activation of the units in the network, layer-by-layer, starting from the input layer.

   - The activation level of each input is the value of the training pattern applied to the input.
   - The activation of each hidden and output unit is calculated as:

$$y_j^{(p)} = f_{y_j}^{(p)}(\sum_{i=1}^{I} v_{ji} z_i - v_{j0}) \tag{2.4}$$

$$o_k^{(p)} = f_{o_k}^{(p)}(\sum_{j=1}^{J} w_{kj} y_j - w_{k0}) \tag{2.5}$$

   where $y_j^{(p)}$ is the activation of the $j$-th hidden unit, and $o_k^{(p)}$ is the activation of the $k$-th output unit for pattern $p$. $K$ is the total number of output units, $I$ is the total number of input units and $J$ is the total number of hidden units. $v_{j0}$ is the weight connected to the bias unit in the input layer, while $w_{k0}$ is the weight connected to the bias unit in the hidden layer. The term **bias** is discussed in the section 2.4.3.

3. Weight adjustment

   - Start at the output units and recursively propagate error signals to the input layer.

- Calculate the weight adjustments:

  The output $o_k^{(p)}$ is compared with the corresponding target value $t_k^{(p)}$ over the entire training set using the function

$$E^{(p)} = \frac{1}{2}\sum_{k=1}^{K}\left(t_k^{(p)} - o_k^{(p)}\right)^2 \tag{2.6}$$

  to express the error in the network's approximation of the target function. Minimization of $E^{(p)}$ by GD requires the partial derivative of $E^{(p)}$ with respect to each weight in the network to be computed. The change in weight is proportional to the corresponding derivative:

$$\Delta v_{ji}(t+1) = -\eta\frac{\partial E^{(p)}}{\partial v_{ji}} + \alpha\Delta v_{ji}(t) \tag{2.7}$$

$$\Delta w_{kj}(t+1) = -\eta\frac{\partial E^{(p)}}{\partial w_{kj}} + \alpha\Delta w_{kj}(t) \tag{2.8}$$

  where: $\eta$ is the learning rate which is in the step length in the negative gradient direction. The value of $\eta$ is usually between 0 and 1. The last term is a momentum term which is a function of the previous weight change. The concept of **momentum** is discussed in the section 2.4.3.

  For notational convenience, the $(p)$ superscript is dropped in the remainder of this section. The reader should keep in mind that the equations below are for a single pattern.

  The partial derivative of $E$ with respect to $w_{kj}$ is computed as

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial w_{kj}} \tag{2.9}$$

  The term $\frac{\partial E}{\partial o_k}$ in equation (2.9) is calculated as

$$\begin{aligned}\frac{\partial E}{\partial o_k} &= \frac{\partial}{\partial o_k}\left[\frac{1}{2}\sum_{k=1}^{K}(t_k - o_k)^2\right] \\ &= -(t_k - o_k)\end{aligned} \tag{2.10}$$

and

$$\frac{\partial o_k}{\partial w_{kj}} = o_k(1-o_k)y_j \tag{2.11}$$

From equations (2.10) and (2.11)

$$\frac{\partial E}{\partial w_{kj}} = -(t_k - o_k)o_k(1-o_k)y_j \tag{2.12}$$

Therefore,

$$\Delta w_{kj} = \eta(t_k - o_k)o_k(1-o_k)y_j \tag{2.13}$$

The contribution of hidden units to the output error is not readily known. However, the error measure can be written as a function of the error contribution over all output units.

$$\begin{aligned}\frac{\partial E}{\partial v_{ji}} &= \sum_{k=1}^{K} \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial y_j}\frac{\partial y_j}{\partial v_{ji}} \\ &= \frac{\partial y_j}{\partial v_{ji}}\sum_{k=1}^{K}\frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial y_j} \\ &= \frac{\partial y_j}{\partial v_{ji}}\sum_{k=1}^{K} -(t_k - o_k)o_k(1-o_k)y_j w_{kj}\end{aligned} \tag{2.14}$$

The partial derivative $\frac{\partial y_j}{\partial v_{ji}}$ is computed as

$$\frac{\partial y_j}{\partial v_{ji}} = y_j(1-y_j)z_i \tag{2.15}$$

Therefore,

$$\Delta v_{ji} = \eta\sum_{k=1}^{K}(t_k - o_k)o_k(1-o_k)y_j w_{kj}y_j(1-y_j)z_i \tag{2.16}$$

4. Update the weights:

$$w_{kj}(t+1) = w_{kj}(t) + \Delta w_{kj}(t+1) \quad (2.17)$$

$$v_{ji}(t+1) = v_{ji}(t) + \Delta v_{ji}(t+1) \quad (2.18)$$

where $t$ represents the current time step, $\Delta v_{ji}$ and $\Delta w_{kj}$ are the weight adjustments from equations (2.13) and (2.16).

5. Test for convergence, for example if an acceptable MSE has been reached, or the maximum number of epochs has been exceeded. Go to step (2) and repeat until convergence in terms of selected stopping criteria.
An iteration, which is referred to as an *epoch*, is one pass through the training set which includes presenting training patterns, calculating the activation values, and modifying the weights.

### 2.4.3 Additional Features To The Training Algorithm

Some features have been incorporated into the GD training algorithm to improve neural network learning.

- **Addition of neuron bias**: The addition of a bias to the neural networks is to offset the origin of the activation function. This allows more rapid convergence of the training process [Masters 1993, Wasserman 1989, Fausett 1994]. By adding a bias unit with a constant activation value of $-1$. The weight between the bias unit and a unit in the next layer serves as bias to that unit. These bias weights are trained in the same way as the other weights. Therefore, for hidden units

$$y_j = f(\sum_{i=0}^{I} v_{ji} z_i \quad (2.19)$$

and for output units

$$o_k = f(\sum_{j=0}^{J} w_{kj} y_j \quad (2.20)$$

with $z_0 = -1$ and $y_0 = -1$. $v_{j0}$ is the weight to the bias unit $z_0$ in the input layer and $w_{k0}$ is the weight to the bias unit $y_0$ in the hidden layer.

- Another prominent feature that can be added to improve the performance of the network is to add a **momentum term**. The addition of a momentum term helps to avoid oscillations in weight adjustments [Beale *et al* 1990]. Momentum is proportional to the magnitude of previous weight changes. Weight changes are then in the direction that is a combination of the current gradient and the previous gradient. Momentum allows the network to make reasonably large weight adjustments, as long as corrections are in the same direction for several patterns, while using a smaller learning rate. Momentum also reduces the chances of getting stuck in a local minimum [Wasserman 1989, Dayhoff 1990] - a problem of learning with the gradient descent technique which is discussed in the next section. In effect, momentum tries to find the global minimum of the error surface by repeatedly jumping in the downhill direction. Momentum is typically implemented by multiplying a numeric parameter between zero and one with the previous weight change (refer to equations (2.7) and (2.8)).

## 2.5 Learning Difficulties With Gradient Descent Optimization

Despite gradient descent's usefulness in training multilayer neural networks, there are difficulties associated with learning using gradient descent. Problems with GD include network paralysis, local minima and slow convergence.

One of the problems that occurs when GD is used is *network paralysis*. Network paralysis occurs when the weights are adjusted to very large values during training. Large

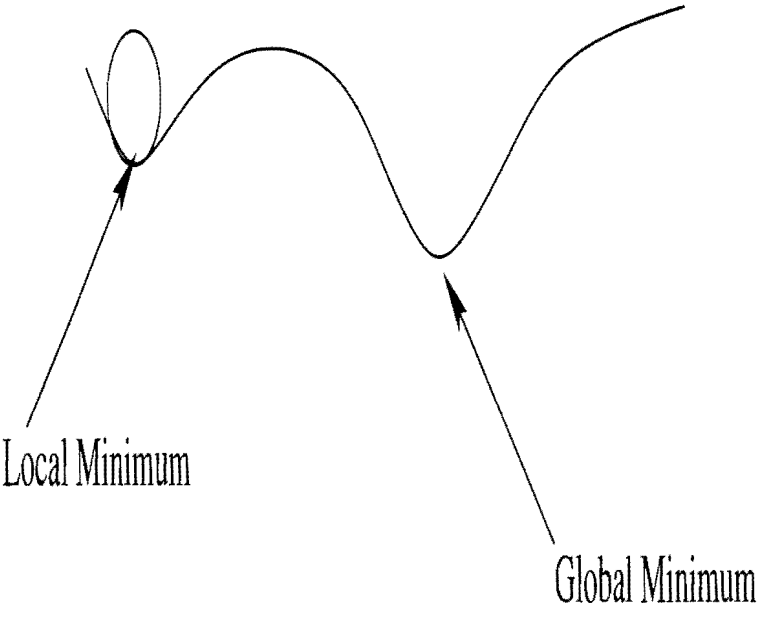



Figure 2.8: An illustration of local and global minimum

weights can force most of the units to operate at extreme values, in a region where the derivative of the activation function is very small. Since the error backpropagated is proportional to the derivative of the activation function (refer to equations (2.9) and (2.14), the training process can come to a stand still [Wasserman 1989].

A prominent problem with training using GD is the occurrence of *local minima* [Rumelhart *et al* 1986]. The network finds a combination of weights that that represents a local and not a global minimum. The gradient descent technique follows the slope of the error surface downward, constantly adjusting the weights towards the minimum. The error surface could be highly complex: full of hills, valleys, folds and gullies in high dimensional space. The network may therefore, get trapped in a local minimum (shallow valley), while there is a much deeper minimum nearby or elsewhere. Figure 2.8 illustrates the concept of local minimum and global minimum.

There is also the problem of *slow convergence*: A multilayer neural network requires many repeated presentations of the input patterns, for which the weights need to be adjusted before the network is able to settle down into an optimal solution. The

method of gradient descent could be very slow to converge for a complex problem due to the complexity of the error surface [Wasserman 1989].

*Overfitting* and *underfitting* are not unique problems of GD but general problems of any learning or regression algorithm. Overfitting occurs when a network has too many training units (an oversized architecture), causing the network to produce good results with the training data, but performing badly with data not seen during training. Rather than learning the basic structure of the data, the network learns the irrelevant details, for example noise in the training patterns [Sarle 1995, Schittenkopf *et al* 1997]. A low training error therefore does not always imply a good performance of the network. A network can also be *underfitted*, which occurs when the number of training units in a network is too few, i.e. an undersized architecture. Thus the network fails to approximate the true form of the relationship between inputs and targets.

### 2.5.1 Solutions to these learning difficulties

Many research efforts have been invested in the study of how to improve the learning of multilayer neural networks. Approaches to improve performance range from finding the optimal learning rate to finding the optimal network architecture. Some of the most promising approaches are discussed below:

1. **Adaptive learning rate and momentum factor:** Rather than using a fixed learning rate in training, the learning rate and momentum can be adjusted dynamically during training [Weir 1990, Fausett 1994]. Decreased training time and improved convergence have been achieved using adaptive learning rate and momentum. A careful selection of the learning rate is often necessary to ensure smooth convergence. A large learning rate can cause network paralysis and a small learning rate causes slow convergence. An advantage of a large learning

rate is to accelerate learning when a plateau is reached in the weight space. A small learning rate, on the other hand, is helpful in slowing down learning when a valley is approach in the search space [Yu *et al* 1997].

A momentum factor is used to smooth error oscillation. Plaut *et al* have shown that low momentum is good to maintain movement along a particular direction in the error surface, but should be increased when the learning procedure has settled in a stable direction of movement [Plaut *et al* 1986 ]. The learning rate and momentum should therefore be varied according to the region where the weight adjustment is. An optimal learning rate for a learning problem can also be found [Weir 1990]. However, the optimal learning rate is problem dependent.

2. **Random weight initialization:** The choice of initial weight values influences whether the network converges quickly or not [Fausett 1994]. The weight update between two units depends on both the derivative of the objective (error) function with respect to weights, as well as the activation value of units. Initial weights must not be too large, to ensure that the initial input signal of the a hidden unit or output unit does not fall in the region where the derivative of the sigmoid function is very small. If the derivative is small, the net input of the hidden or the output unit will be close to zero and will cause extremely slow learning due to small weight updates. Weights are initialized *randomly* to break symmetry [Rumelhart *et al* 1986]. Symmetry occurs when all weights are initialized to the same value. Consequently, the hidden units are assigned identical error values. All weights in the network are then adjusted in an identical manner, and thus prevent the error function from being reduced. Weights are usually initialized *randomly* to small values [Rumelhart *et al* 1986].

3. **Optimal network architecture selection:** The achievement of good performance in a trained network is through careful selection of the network size. An oversized network can lead to overfitting of the data but on the other hand, a

small sized (simple) network can lead to underfitting [Le Cun 1990].

Optimal architecture selection is adaptive in the sense that adjusting neural network size is incorporated into the network training. Research into optimal architecture selection is split into three areas: growing the network during training by adding more parameters to the network [Hirose *et al* 1991, Jutten *et al* 1995], pruning the network by removing redundant parameters during training [Sietsma *et al* 1988, Engelbrecht *et al* 1996, Le Cun 1990] or regularization through penalty terms added to the objective function [Weigend *et al* 1991, Kamimura *et al* 1994, Karayiannis *et al* 1993].

- **Network pruning** involves training an oversized network and removing redundant and irrelevant network parameters, including units and / or weights. Starting with an oversized network rather than a small or undersized network, the network is guaranteed to learn the desired input and output mapping [Le Cun 1990]. Once a network has learn a solution to a problem, the network can then be pruned to the minimum size [Sietsma *et al* 1988]. Pruning aims at solving the problem of the overfitting as well as reducing the computational cost of training and applying the network [Le Cun 1990]. Selecting the parameters to remove is the main focus of pruning methods and is based on different criteria proposed by different researchers. Le Cun *et al* introduced the concept of network pruning through their work on optimal brain damage (OBD) [Le Cun 1990]. Le Cun *et al* empirically showed that by removing unimportant weights from a network, several improvements could be achieved. These improvements include better generalization, fewer training examples and improved speed of learning. OBD reduces the size of a network by selectively deleting weights. The goal of OBD is to find a set of parameters, that when deleted would cause the least increase in the error function. To find such set of parameters, Le Cun *et al* defined the *saliency* of parameter as the change in error caused

by deleting that parameter. The parameter with least saliency is deleted. The second derivative information is used to calculate this saliency and therefore is computationally complex due to calculation of Hessian matrix. Hassibi *et al* extended OBD to remove the required retraining after pruning [Hassibi *et al* 1994]. Their approach, referred to as OBS, automatically computes the adjustments needed to the remaining weights due to the pruning of weights. Engelbrecht *et al* developed a pruning algorithm where the sensitivity of the output of the network to small parameter perturbations is used to identify irrelevant parameters [Engelbrecht *et al* 1996]. This algorithm prunes both input and hidden layers of feedforward neural networks. Units that have the least statistical influence on all units in the succeeding layers are pruned. An adaptation to this pruning algorithm was also proposed by Engelbrecht *et al* [Engelbrecht *et al* 1999b]. A new pruning heuristic based on variance analysis of sensitivity information is used to find irrelevant parameters.

- **Network growing** involves growing the network during training. Hidden units are added to the network when needed. Network growing reduces computational cost and complexity of the trained network [Jutten *et al* 1995]. A reduction in computational cost is achieved because the optimal architecture needed to train a network is problem dependent. A small network architechure have fewer weights than a large network and thus needs a few weight adjustments. Once the optimal solution for a problem is obtained, the resulting network has an optimal architecture [Jutten *et al* 1995]. Hirose *et al* also used network growing to solve the problem of local minima [Hirose *et al* 1991]. In their research, Hirose *et al* added more hidden units to a network being trained as soon as the network starts overfitting. The error function was used to detect local minima.
- **Regularization,** where all weights are penalized. Regularization

is achieved by adding a penalty term to the objective function [Weigend *et al* 1991]. In doing so, network complexity is penalized. The effect is that redundant weights are driven to zero, while active weights retain their importance [Kamimura *et al* 1994, Karnin 1990]. Weight decay is one form of regularization [Fu 1994].

4. **Training with jitter:** Jitter is artificial noise deliberately added to inputs during training. Training with jitter is a form of regularization, such as weight decay. An advantage of jitter is that the NN can be brought out of a local minimum [Beale *et al* 1990]. Injecting artificial noise into inputs during training is very effective in improving generalization performance when small training sets are used. Noise injected into inputs is assumed to have zero mean and a small variance in order not to change the distribution of the given training data.

5. **Adaptive learning function:** Activation functions can be adapted and trained just like the weights of a NN. This adaptation improves learning in terms of faster convergence and more accurate results [Zurada 1992a. Engelbrecht *et al* 1995, Fletcher *et al* 1994]. Zurada [Zurada 1992a] and Fletcher *et al* [Fletcher *et al* 1994] proposed a learning rule where the steepness or slope of the activation function used for learning is trained alongside with the weights. The learning rule produced better solutions and a faster convergence to problems when compared to conventional error backpropagation. Another research on adaptive learning functions is the gamma learning proposed by Engelbrecht *et al* [Engelbrecht *et al* 1995]. Gamma learning extends the lamda rule of Zurada, by dynamically adjusting the output range of the sigmoid activation function, thereby performing automatic scaling.

6. **Active learning** involves making optimal use of the training data. Much research has been done in developing active learning models [Engelbrecht *et al* 1998, Engelbrecht *et al* 1999a, Engelbrecht *et al* 1999c,

Zhang 1994, Röbel 1994a, Plutowski *et al* 1993, Cohn *et al* 1996]. Active learning refers to the selection of a subset of the available training data dynamically during training, where the subset contains the most informative data. Active learning has been found to save computational cost and reduce training time [Cohn *et al* 1996, Plutowski *et al* 1993, Röbel 1994a, Engelbrecht *et al* 1999a]. This thesis presents a survey and comparison of active learning algorithms for function approximation and time series problems. The next chapter elaborates on active learning.

## 2.6 CONCLUSION

This chapter discussed the training of the neural networks. A background introduction into multilayer neural networks was given. The chapter focused on training feedforward MLNNs using gradient descent optimization.

The learning equations were derived and the problems of training a NN using gradient descent as well as the solutions to these problems were discussed.

The next chapter discusses one of the methods to improve learning with gradient descent technique, i.e. *active learning*.

| *Problem* | *Equation* | $P_C$-$P_G$-$P_V$ | *Architecture* |
|---|---|---|---|
| F1 | (3.3) | 600 -200-200 | 2-5-1 |
| TS1 | (3.4) | 600-200-200 | 1-5-1 |
| TS2 | (3.5) | 600-200-200 | 2-5-1 |
| TS3 | (3.6) | 140-60-60 | 10-10-1 |
| TS4 | | 600-200-200 | 2-5-1 |
| TS5 | | 600-200-200 | 2-7-1 |

Table 3.1: Summary of the functions and time series used

that

$$\begin{aligned} D_C \cap D_V &= \emptyset \\ D_C \cap D_G &= \emptyset \\ D_G \cap D_V &= \emptyset \end{aligned}$$

Let $P_C$ be the number of training patterns in $D_C$, $P_V$ the number of training patterns in $D_V$ and $P_G$ the number of patterns in test set $D_G$. Table 3.1 shows the size of these sets for each problem. $D_C$ is the candidate training set from which training patterns are selected. $D_V$ contains data used to determine the generalization factor during training. $D_G$ contains data used to determine the generalization performance of the network.

The performance of the active learning algorithms was tested on clean and noisy data, as well as data containing outliers. Section 3.5.1 explains the experimental procedure, including a discussion of the performance criteria used to compare the learning algorithms. The results are compared in section 3.5.2.

The characteristics of the functions and time series used for experimentation are discussed next. The following functions and time series were used:

1. Function $F1$ is defined as (see figure 3.1(a))

$$F1 : F(z_1, z_2) = \frac{1}{2}(z_1^2 + z_2^2) \tag{3.4}$$

where $z_1, z_2 \sim U(-1, 1)$. All target values were scaled to the range [0,1].

2. Time series TS1 is a sine function defined as (see figure 3.1(b)),

$$TS1 : F(z) = \sin(2\pi z)e^{(-z)} + \zeta \tag{3.5}$$

where $z \sim U(-1, 1)$ and $\zeta \sim N(0, 0.1)$. Target values were scaled to the range [0,1].

3. Time series TS2 is the henon-map function defined as (refer to figure 3.1(c)),

$$\begin{aligned} TS2 : o_t &= z_t \\ z_t &= 1 + 0.3z_{t-2} + 1.4z_{t-1}^2 \end{aligned} \tag{3.6}$$

where $z_1, z_2 \sim U(-1, 1)$. The target values were scaled to the range [0,1].

4. Time series TS3 is a difficult time series, having 10 input parameters of which 7 are irrelevant (see figure 3.2(c)).

$$\begin{aligned} TS3 : o_t &= z_t \\ z_t &= 0.3z_{t-6} - 0.6z_{t-4} + 0.5z_{t-1} + 0.3z_{t-6}^2 - 0.2z_{t-4}^2 + \zeta_t \end{aligned} \tag{3.7}$$

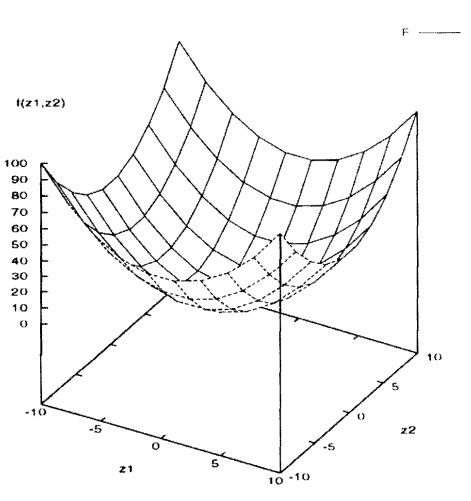
for $t = 1, \cdots, 10$, where $z_4, z_6, z_9 \sim U(-1, 1)$ and $\zeta_t \sim N(0, 0.05)$. All target values were scaled to the range [0,1].

5. Time series TS4 is a convolution of two discrete functions with outliers. Figure 3.2(a) shows an illustration of this function.

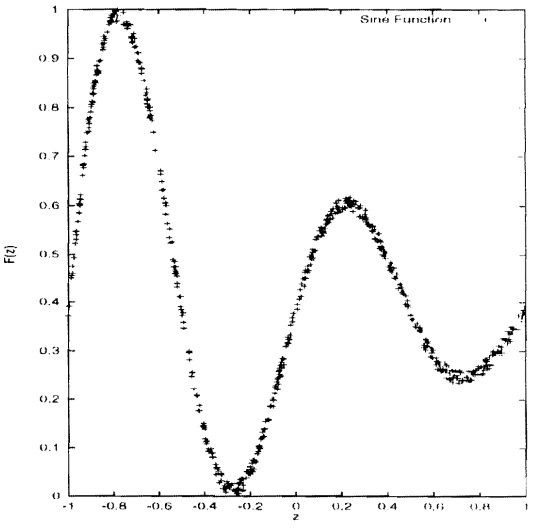
6. Time series TS5 is the sine function TS1 with 5% of the candidate training set consisting of outliers (see figure 3.2(b)).

### 3.5.1 Experimental Procedure

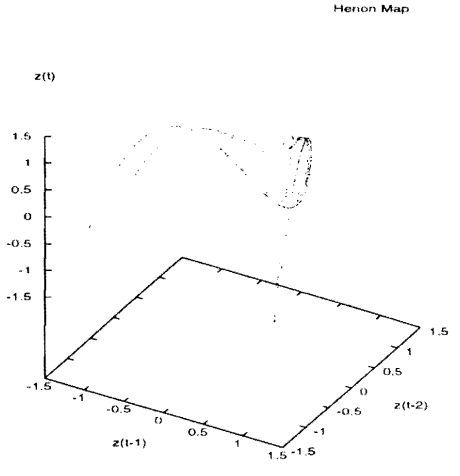
In order to obtain statistically valid assertions in comparing experimental results of the four learning algorithms, thirty simulations were performed for each problem.




(a) F1

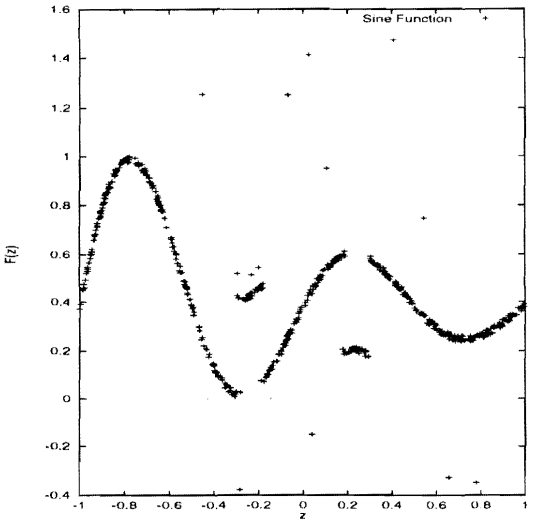



(b) TS1

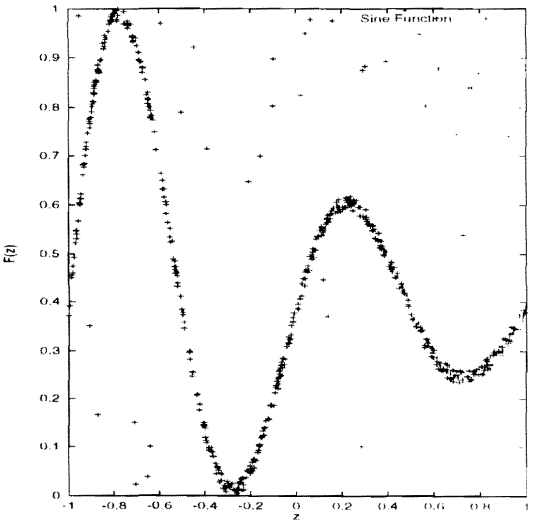



(c) TS2

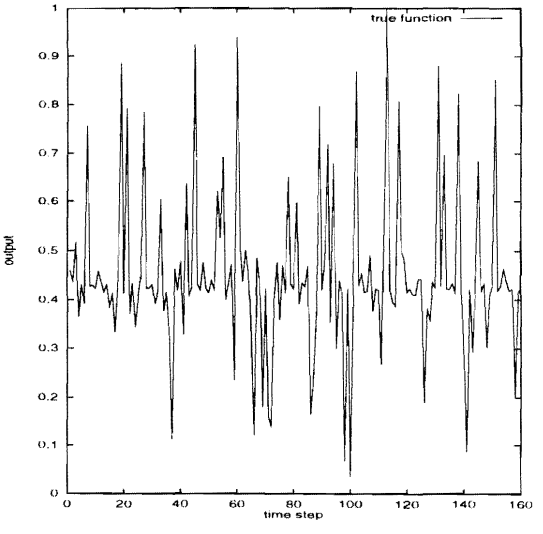
Figure 3.1: Function and Time series problems to be approximated




(a) TS4




(b) TS5




(c) TS3

Figure 3.2: Time series problems to be approximated

Online training was used for the active learning algorithms. The initial subset size for incremental learning algorithms consisted of one pattern and a subselection size of one pattern was used. Each simulations was executed for 2000 epochs. A learning rate 0.1 and momentum 0.9 were used for all the approximation problems . Results reported are averages over the 30 simulations together with 95% confidence intervals as obtained from the t-distribution.

The selective learning algorithm was not applied to F1, since F1 is not a time series problem. The $\tau$ value used in the subset selection criterion for AL was adjusted for each problem using a trial and error approach. For TS3, a high $\tau$ was used ($\tau = 1000$), a value of 100 was used for TS1, TS4 and TS5 while a value of 180 was used for TS2 and F1.

**Performance measures**

To evaluate the performance of each learning algorithm, the following performance criteria were used:

1. The mean squared error (MSE) was used as a measure of accuracy. The MSE measures how well a function is approximated by the network, and is defined as

$$MSE = \frac{\sum_{p=1}^{P} \sum_{k=1}^{K} \left(t_k^{(p)} - o_k^{(p)}\right)^2}{2\ K\ P}$$

.

A MSE value close to zero shows a small error between the target and the output function. The MSE over the three sets $D_V$, $D_G$ and $D_C$ were computed. The MSE over $D_G$, denoted by $E_G$ provides an unbiased estimate of the generalization error since the patterns in $D_G$ were not used for training.

2. Röbel's generalization factor $\rho$ was used to measure overfitting effects. The generalization factor was computed as $\rho = \frac{E_V}{E_C}$, where $E_C$ is the MSE over candidate training set $D_C$ and $E_V$ is the MSE over the validation set $D_V$. A network overfits when the value of $\rho$ increases substantially above 1.

3. The computational complexity of learning algorithms was also used as performance criterion. For the purpose of this thesis, computational cost is measured as the number of calculations needed to train the network. Calculations include subtraction, multiplication, addition and division.

   At any epoch $\xi$, the cost $C_{fc}$ of training a NN on a training set, is expressed as

   $$C_{fc} = (C_V + C_W) * P_T$$

   where $C_V$ is the cost of updating weights between input and hidden units and $C_W$ is the cost of updating weights between hidden and output units. $P_T$ is the number of patterns in the training subset $D_T$. For conventional backpropagation with fixed set learning, $P_T = P_C$. Thus the cost of training $C_{fsl}$ is computed as $C_{fsl} = (C_V + C_W) * P_C$.

   The costs of updating the weights are calculated as

   $$\begin{aligned} C_V &= C_v * (N_V) \\ C_W &= C_w * (N_W) \end{aligned}$$

   where $C_v$ is the cost of updating a single weight between the input and hidden layers, $C_w$ is the cost of updating a single weight between the hidden and output layers. $C_V$ is the total cost of updating the weight connections between the input and the hidden layers, and $C_W$ is the total cost of updating the weight connections between the hidden and output layer. $N_V$ is the total number of connections between the input and hidden layers and $N_W$ is the total number of connections between the hidden and output layers.

   The total number of connections $N_V$ and $N_W$ are expressed as

   $$N_V = (I + 1) * (J + 1)$$
   $$N_W = (J + 1) * (K)$$

   and

   $$C_v = 13$$

$$C_w = 11$$

Therefore,

$$\begin{aligned} C_V &= 13 * (I+1) * (J+1) \\ C_W &= 11 * (J+1) * K \end{aligned} \tag{3.8}$$

The cost of training a network using any active learning algorithm includes $C_{fc}$, the cost of selecting patterns for training and the cost of computing the subset termination criterion. Therefore, at any epoch $\xi$, the cost of training a network using SLA, SAILA, DPS and AL are:

$$\begin{aligned} C_{SL} &= C_{fc} + C_{sla} * P_C \\ C_{DP} &= C_{fc} + C_{dps} * (P_C - P_T) + (C_{S_{dps}} * P_T) \\ C_{AL} &= C_{fc} + C_{al} * (P_C - P_T) + (C_{S_{al}} * P_T) \\ C_{SA} &= C_{fc} + C_{sai} * (P_C - P_T) + (C_{S_{sai}} * P_T) \end{aligned}$$

For all the incremental learning algorithms, the subset selection criteria are tested on the remaining patterns in the candidate set $D_C$ which is equal to $P_C - P_T$. Also, for incremental learning algorithms, an additional cost of selecting pattern is incurred when a pattern is selected.

$C_{SL}$, $C_{SA}$, $C_{AL}$ and $C_{DP}$ are the cost of training a network using SLA, SAILA, AL and DPS respectively. $C_{sla} = 15$ is the cost of computing the subset selection criteria for SLA, $C_{dps} = 11$ is the cost of computing the subset selection criteria for DPS, $C_{al} = 4$ is the cost of computing the subset selection criteria for AL and $C_{sai} = 18$ is the cost of computing the subset selection criteria for SAILA. $C_{S_{dps}} = 2$, $C_{S_{al}} = 2$ and $C_{S_{sai}} = 7$ are the cost of selecting patterns into $D_T$ for DPS, AL and SAILA respectively.

Therefore,

$$C_{SL} = C_{fc} + 15P_C$$

$$
\begin{aligned}
C_{DP} &= C_{fc} + 11(P_C - P_T) + 2P_T \\
C_{AL} &= C_{fc} + 4(P_C - P_T) + 2P_T \\
C_{SA} &= C_{fc} * P_T + 18(P_C - P_T) + 7P_T
\end{aligned} \tag{3.9}
$$

From equation (3.9), the cost of training is directly proportional to the number of patterns selected for training. The more patterns are selected for training, the higher the computational cost. Initially, $P_T$ for SLA is greater than the other algorithms because DPS, AL and SAILA are incremental learning algorithm and a small initial trainig set and subset size is used in the simulations. Thus, $C_{SL}$ is expected to be greater than $C_{AL}$, $C_{DPS}$ and $C_{SA}$ initially. SAILA is computationally more expensive in selection criteria than the other algorithms because SAILA has more subset selection criteria to implement than the other algorithms.

Section 3.5.2 illustrates the costs for the different algorithms.

### 3.5.2 Results

This section presents the results of the simulations carried out on the active learning algorithms.

**Training error**

In order to compare the performance of the four active learning algorithms, the MSE over the candidate set $D_C$ was computed for the simulations and the average calculated. Tables (3.2) and ( 3.3) show the result over 2000 epochs for clean data and data with noise and outliers.

For TS1, DPS had a very low error with the lowest variance which means that all the errors of the simulations for DPS were all closer to the average error of 0.0003. Although, SLA had a low error as well. However SLA had a large variance when compared to DPS. AL had the largest error with a very large variance.

DPS achieved the smallest error for TS2, having a small variance. For TS3, all the algorithms had very low errors but SAILA had a high variance. DPS had the smallest error for F1 with a very small variance.

For TS4 and TS5, SLA achieved the smallest error with the lowest variance. AL had the largest error for TS4 and TS5. This is because AL selected and trained on just a single pattern for TS4 and an average of 4 patterns for TS5. Thus AL had high errors for TS4 and TS5.

The training errors for all the problems with noise and outliers were larger ( $\times 10^2$) than for problems with clean data. DPS had the lowest average error for clean data while SLA had the lowest error for noisy data.

**Generalization error**

To compare the generalization ability of the four active learning algorithms, the MSE over the generalization set, $E_G$, was computed and the average over the 30 simulations was plotted as a function of number of epochs. Figures 3.3 and 3.4 illustrates the trend of the generalization errors for the entire training period.

DPS achieved a very low average error faster than the other algorithms for F1 (refer to figure 3.3(a)). However, both SAILA and AL achieved a comparable result to DPS at epoch 1000. From table 3.3, DPS had the lowest error with the a very small variance ($5.07E-05$) which means all errors of the simulations are closer to the average.

For TS1, SAILA initially had the highest generalization error but decreased to a low level of error (see figure 3.3(b)). SLA initially had the lowest average error, which can be explained by the fact that SLA used more patterns initially than the other algorithms (refer to figure 3.7(b)). Although SLA and DPS had small errors, DPS had the smallest variance and thus DPS achieved the smallest error. AL had the largest error after 2000 epochs with a large confidence interval.

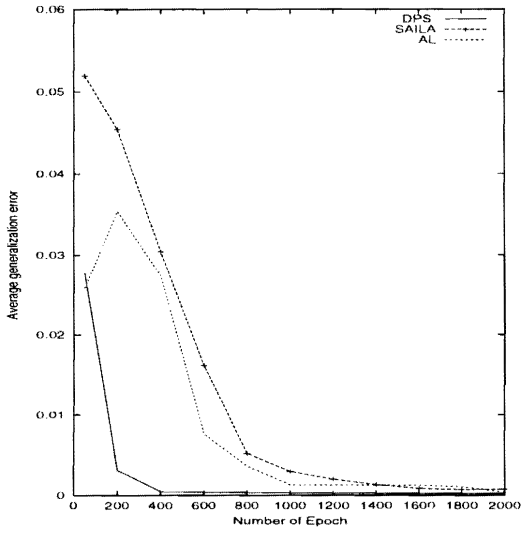
For TS2, DPS, AL and SLA achieved a very low average error before epoch 500.

| Function | Röbel | Zhang | Selective Learning | Sensitivity Analysis |
|---|---|---|---|---|
| **TS1** | | | | |
| Training Error | 0.00036 ± 0.00017 | 0.02172 ± 0.04186 | 0.00045 ± 0.00035 | 0.00346 ± 0.00712 |
| Generalization | 0.00039 ± 0.0002 | 0.02241 ± 0.04191 | 0.00047 ± 0.00040 | 0.0035 ± 0.00737 |
| Used Patterns | 485.43 ± 234.79 | 4.73 ± 0.92 | 270.93 ± 3.48 | 571.67 ± 88.91 |
| **TS2** | | | | |
| Training Error | 0.00014 ± 0.00011 | 0.00023 ± 0.00021 | 0.00029 ± 0.00038 | 0.00126 ± 0.00163 |
| Generalization | 0.00012 ± $0.25E-05$ | 0.00022 ± 0.00019 | 0.00029 ± 0.00037 | 0.00129 ± 0.00169 |
| Used Patterns | 411.77 ± 215.87 | 174.63 ± 61.48 | 272.57 ± 7.61 | 522.57 ± 173.37 |
| **TS3** | | | | |
| Training Error | 0.00039 ± 0.00086 | 0.00044 ± 0.00091 | 0.00050 ± 0.00085 | 0.00068 ± 0.00146 |
| Generalization | 0.00275 ± 0.00155 | 0.00253 ± 0.00133 | 0.00302 ± 0.00138 | 0.00225 ± 0.00174 |
| Used Patterns | 180 ± 0 | 180 ± 0 | 78.17 ± 1.53 | 180 ± 0 |

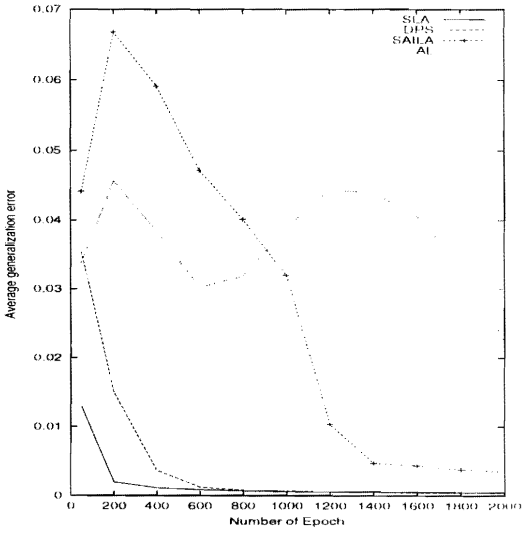
Table 3.2: Comparison results over 2000 epochs for times series problems

| Function | Röbel | Zhang | Selective Learning | Sensitivity Analysis |
|---|---|---|---|---|
| **F1** | | | | |
| Training Error | 0.000226 ± $5.07E-05$ | 0.000412 ± 0.000366 | | 0.000791 ± 0.001852 |
| Generalization | 0.000221 ± $5.2E-05$ | 0.000392 ± 0.000347 | | 0.000764 ± 0.001624 |
| Used Patterns | 320.2 ± 167.6698 | 82.8 ± 32.37935 | | 445.1333 ± 121.476 |
| **TS4** | | | | |
| Training Error | 0.01141 ± 0.00573 | 0.19935 ± 0.03093 | 0.00516 ± 0.00393 | 0.02828 ± 0.09522 |
| Generalization | 0.01077 ± 0.00534 | 0.19051 ± 0.02169 | 0.00478 ± 0.00349 | 0.02739 ± 0.09170 |
| Used Patterns | 493.03 ± 193.37 | 1 ± 0 | 245.23 ± 7.89 | 597.03 ± 27.41 |
| **TS5** | | | | |
| Training Error | 0.00683 ± 0.00468 | 0.10278 ± 0.08731 | 0.00155 ± 0.00158 | 0.00562 ± 0.00768 |
| Generalization | 0.00714 ± 0.00489 | 0.09904 ± 0.08932 | 0.00158 ± 0.00142 | 0.00595 ± 0.00842 |
| Used Patterns | 103.5 ± 20.14 | 4.67 ± 1.35 | 269.13 ± 9.89 | 584 ± 93.4 |

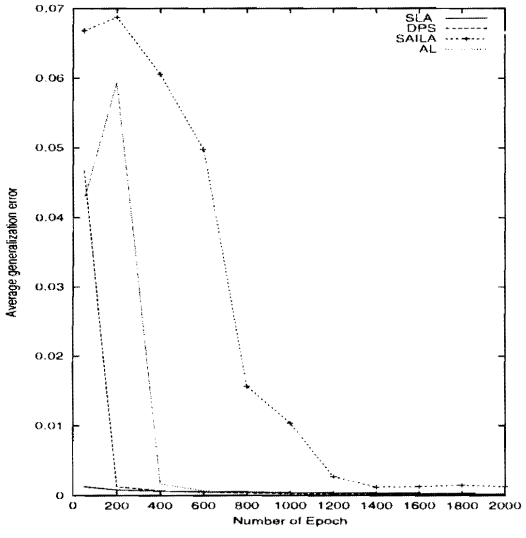
Table 3.3: Comparison results over 2000 epochs for problems F1 and times series with noise and outliers



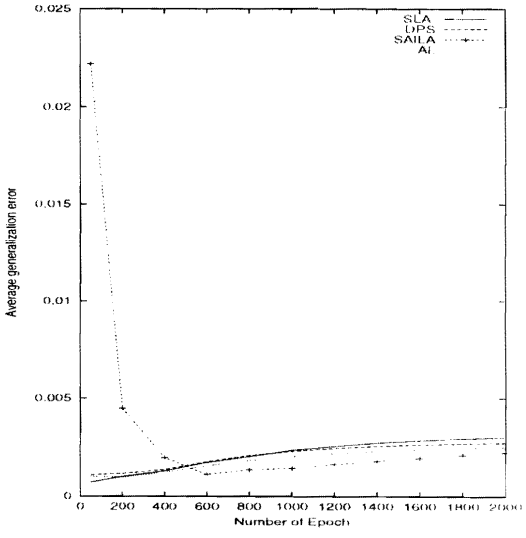

(a) Function F1




(b) TS1

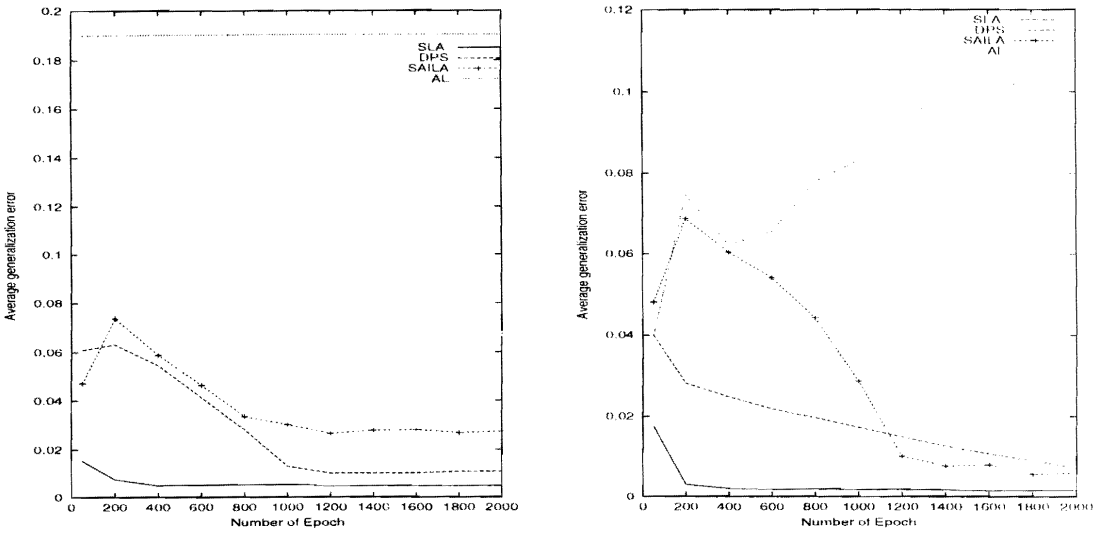



(c) TS2




(d) TS3

Figure 3.3: Average generalization error vs epoch




(a) TS4 (b) TS5

Figure 3.4: Average generalization error vs epoch

SAILA was slower to achieve a comparable low error but SAILA had a low error by the end of training. From the table 3.2, DPS had the smallest error with a very small variance after 2000 epochs, implying that all errors of the simulations are closer to the average.

For TS3, the generalization error for all the algorithms increased as the number of epochs increased except SAILA (see figure 3.3(c)).

From the table 3.3, SLA had the lowest generalization errors for TS4 and TS5. While AL had the largest error for TS4 and TS5. AL did not learn the functions (refer to figure 3.4(a)). AL selected a few patterns for training, thus had little information about the time series to be approximated and therefore AL had a bad generalization.

DPS had the lowest generalization errors for functions with clean data while SLA had the lowest generalization errors for functions with noise and outliers. Although DPS had better generalization with clean functions than SLA, DPS used more patterns than SLA to achieve the low generalization error in all the problems. AL had very large generalization errors for TS1, TS4 and TS5. This bad generalization can be

attributed to the extremely small training set sizes used by AL which is an indication of an inferior subset selection criterion. The subset selection criterion depends on the number of connections in the network. Redundant or irrelevant weights in the network will make the value of the performance level $\kappa$ very large which can cause the network to train on the current training subset $D_T$ too long without selecting additional patterns. Thus the network selects a few patterns, hence having insufficient information to train the network. On the other hand, too few weights in the network can make $\kappa$ small. Thus, the network selects patterns more often than are needed for training.

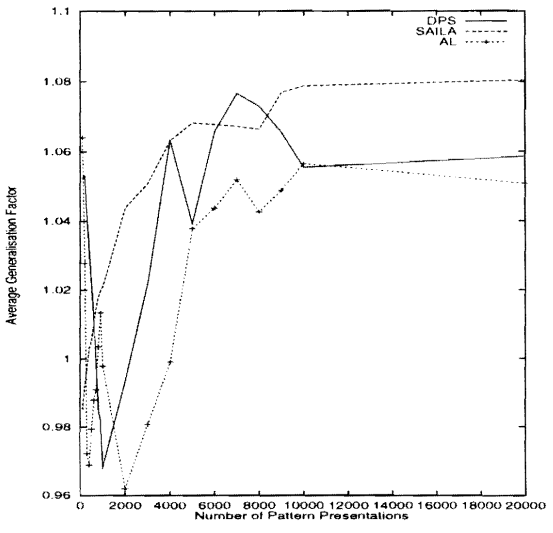
**Overfitting effects**

The average generalization factor $\rho$ for all the problems were computed over the 30 simulations. Figures 3.5 and 3.6 show the charts for the average generalization factors. The average generalization factors were plotted as function of pattern presentations. A pattern presentation represents one weight update.

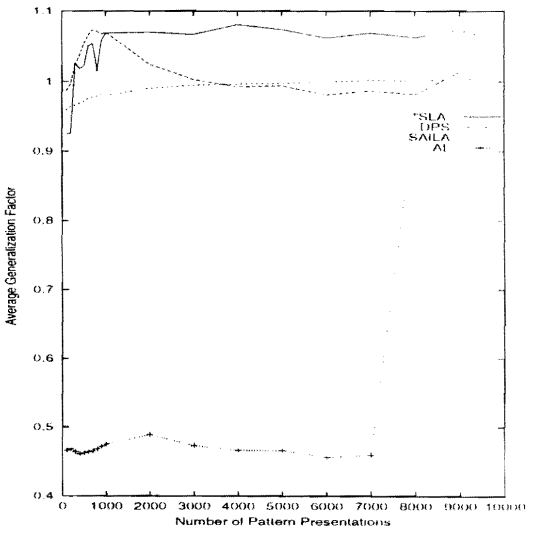
TS3 was the only function for which all the algorithms except SAILA, overfitted. SAILA had an average generalization factor of less than one, while the other algorithms had high generalization factors. For the entire training period for TS4, AL had a generalization factor constantly larger than 1, indicating that AL overfitted TS4. For the other functions, the average generalization factor values fluctuated. The flunctuation is due to the overfitting of a training subset until new patterns are selected for training. When new patterns are selected, the overfitting of the training subset is reduced. The average generalization factor for all the algorithms (except TS3) were slightly over one, and indicating a mild case of overfitting.

**Computational costs**

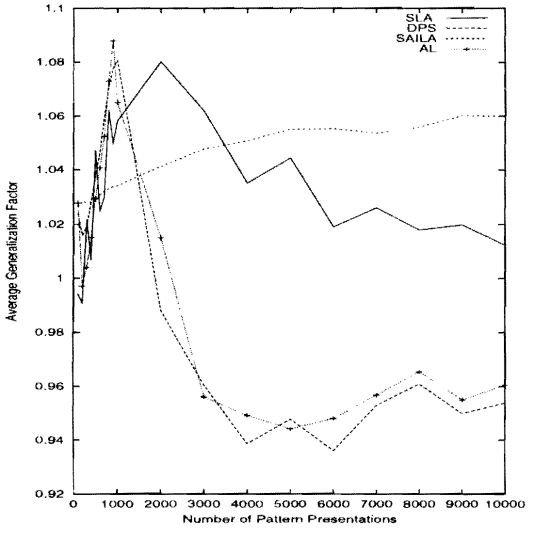
The computational costs for AL, DPS, SLA and SAILA were computed using equation (3.9) for specified epochs. The costs are plotted as a function of epochs as illustrated in figures (3.9) and (3.10).



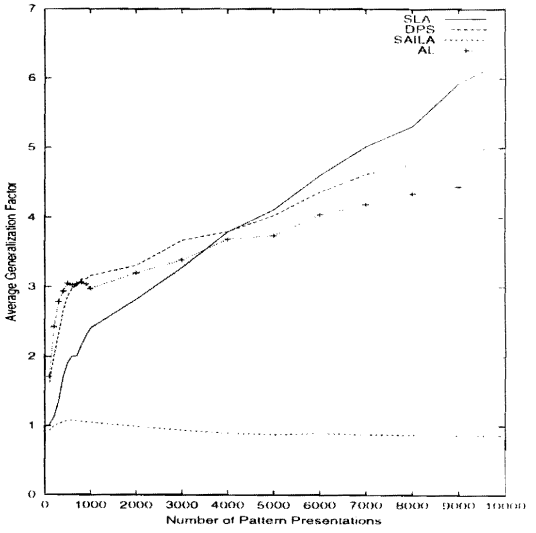

(a) Function F1




(b) TS1

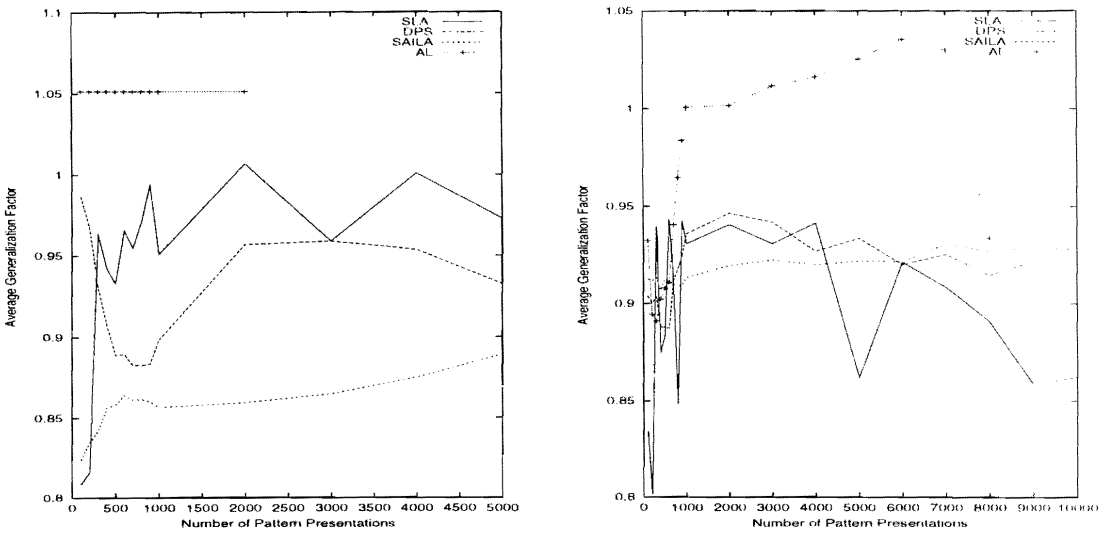



(c) TS2




(d) TS3

Figure 3.5: Average generalization factor vs pattern presentations




(a) TS4 (b) TS5

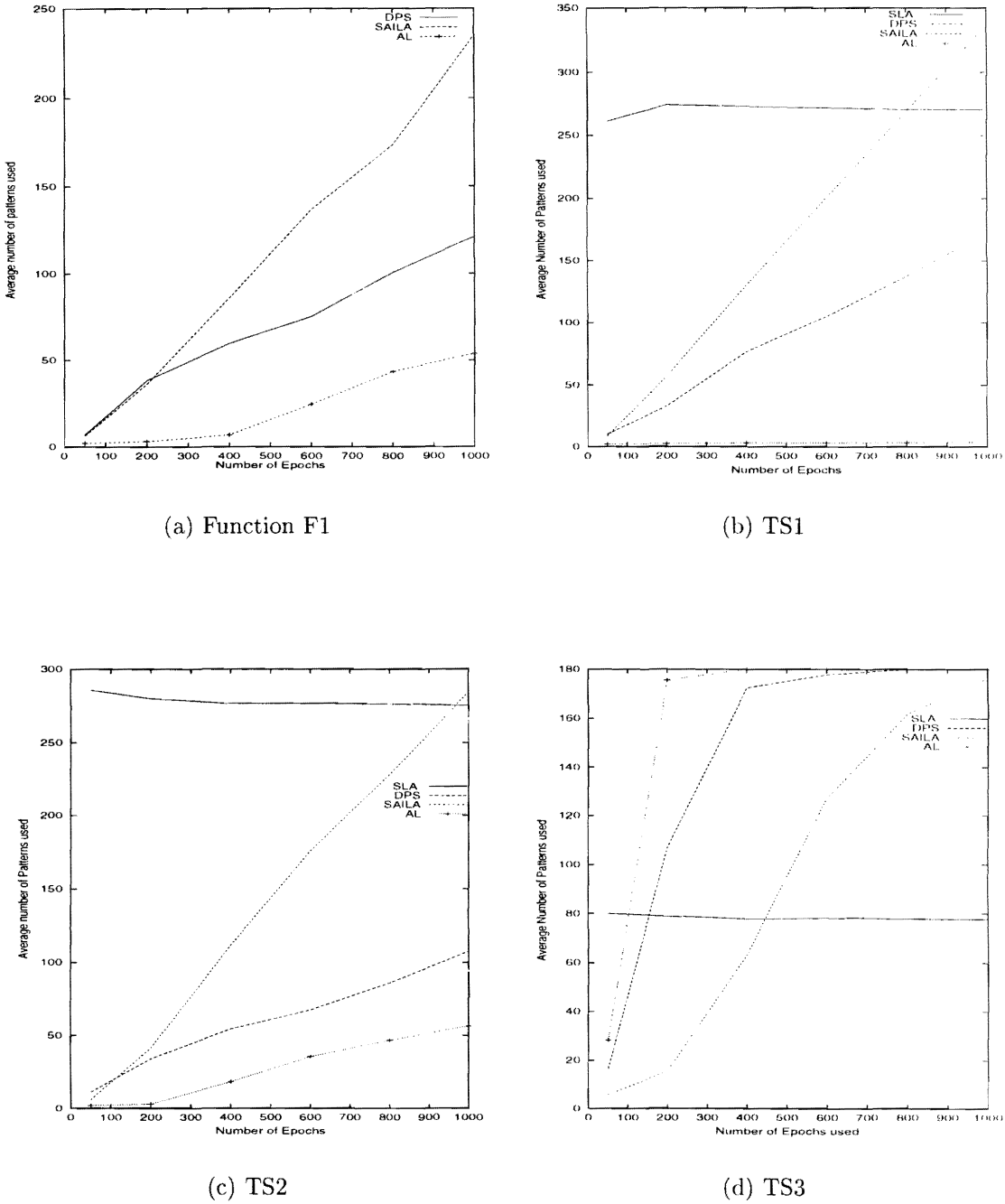
Figure 3.6: Average generalization factor vs pattern presentations

SAILA has the most expensive and AL has the least expensive subset selection criteria. However, AL performed badly (refer to tables 3.2 and 3.3). For all the approximation problems, DPS, AL and SAILA had increasing costs because they are incremental learning algorithms. More patterns were used as training progressed (refer to figures (3.7) and (3.8)).

For F1 and TS2, AL had the smallest cost (see figure 3.9(a) and (b)). These small costs can be attributed to the cheap cost of the subset selection criterion as well as the fact that AL used the smallest number of patterns for training.

Despite the fact that AL has the cheapest subset selection criterion and a simple selection criterion, AL had the highest cost for TS3. This is because AL selected all the patterns in $D_C$ within a short training interval (by epoch 400). SLA initially had the highest cost (first 350 epochs). However, at epoch 1000, the cost was half of the other algorithms.

For all the functions approximated, SLA initially had a higher training cost than the other algorithms - almost four times the training cost of other algorithms, because



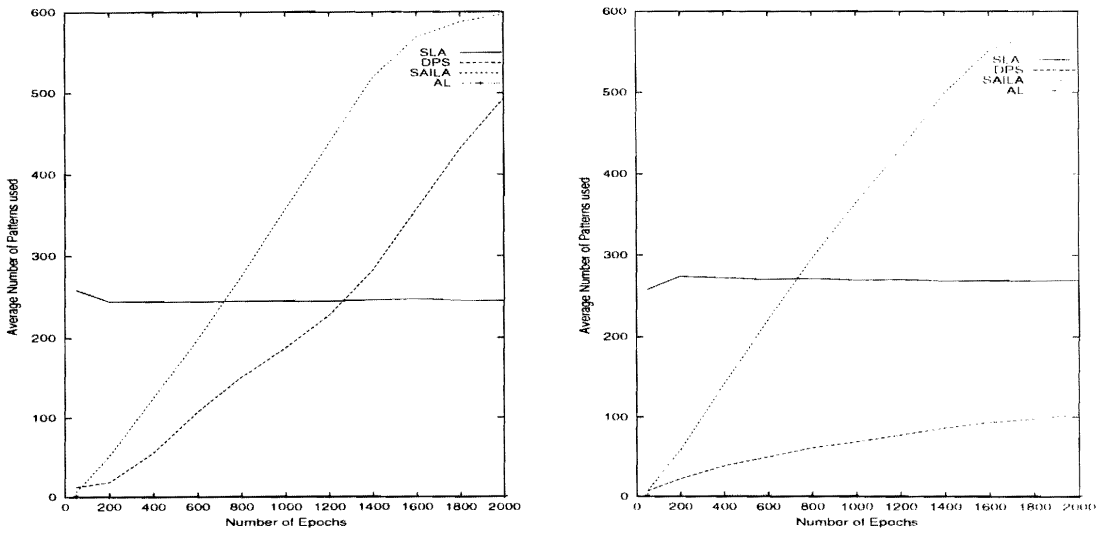

(a) Function F1

(b) TS1

(c) TS2

(d) TS3

Figure 3.7: Average number of patterns used per epoch




(a) TS4 (b) TS5

Figure 3.8: Average number of patterns used per epoch

SLA is a selective approach (see figure (3.9) and (3.10)).

From tables (3.2) and (3.2), SLA used less number of patterns after 2000 epochs than the other algorithms, thus SLA was computationally less expensive.

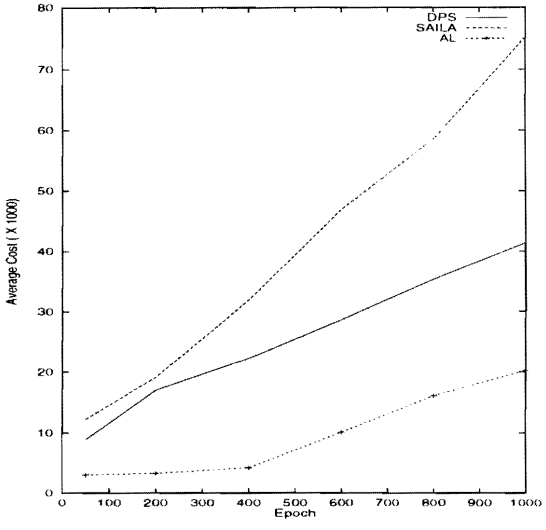
### Convergence

The convergence performance of the four active learning algorithms are compared in figures 3.11 and 3.12. These figures plot the percentage of simulations that reached specific generalization errors.

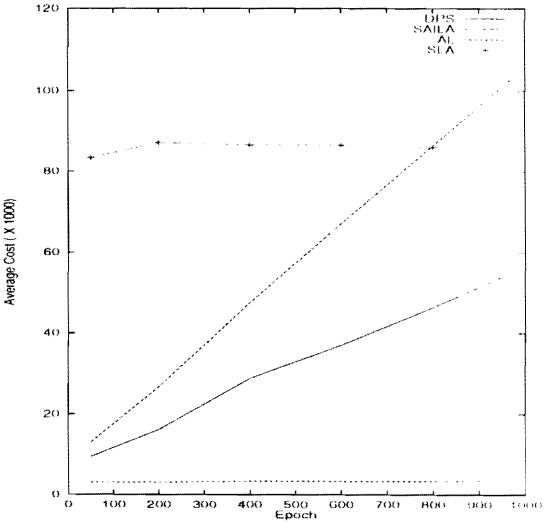
For F1, DPS had the best convergence, all the simulations converged to a very low error of 0.0004. AL also had a good convergence, more than half of the simulations converged to 0.0004 (refer to figure 3.11(a)).

None of AL's simulations converged to the specified error level for TS2. SLA and DPS achieved good convergence for TS2, as more than half of their simulations converged to a low error (refer to figure 3.11(b)).

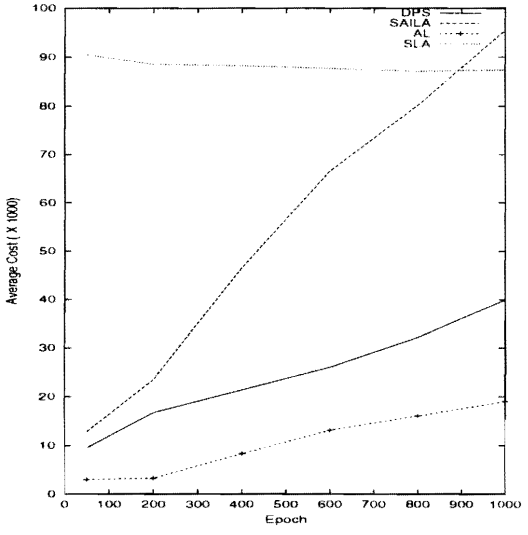
For TS2, DPS had the best generalization, most of all the simulations converged to a



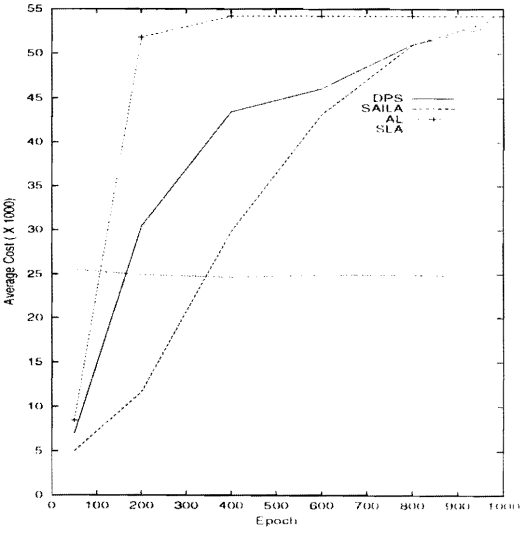

(a) Function F1




(b) TS1

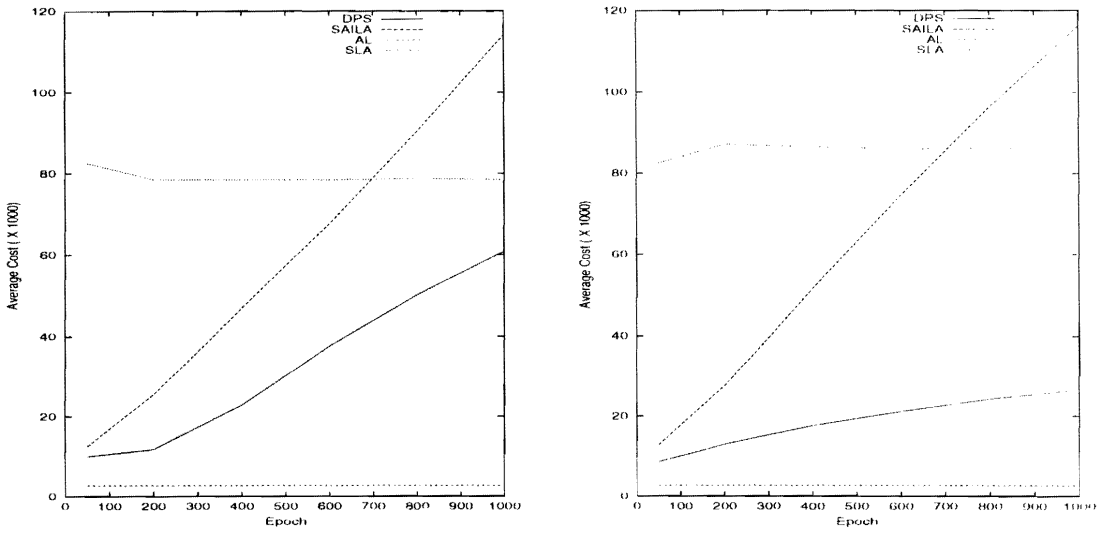



(c) TS2




(d) TS3

Figure 3.9: Average computational cost per epoch




(a) TS4 (b) TS5

Figure 3.10: Average computational cost per epoch

very low error (0.0002). SLA and AL also had good convergence (see figure 3.11(c)).

While the other algorithms had few converged simulations at 0.002, almost half of SAILA's simulations converged to this error (refer 3.11(d)).

AL had bad generalization for TS4 and TS5. None of AL's simulations converged to the specified error levels for TS4 while only a few converged for TS5.

SLA had the best generalization for TS4, with all the simulations converging to a low error of 0.004 (refer to figure 3.12(a)). SAILA had a good convergence with 40% of the simulations converging to this error of 0.004. Only a few of DPS's simulation and none of AL's simulation converged at this point.

SLA also had the best generalization for TS5. Almost all the simulations (74%) converged to a error level of 0.005 while only a few of the other algorithms simulations converged to this error level (see figure 3.12(b)).

SLA had the best convergence for data with outliers and noise. DPS had the best convergence for clean data, although SLA had good convergence for clean data. SAILA

had a good convergence for TS3. For all the sine functions (TS1, TS4 and TS5), AL had bad convergence, none of its simulations converged to the specified error levels. The errors specified for data with outliers and noise were larger than the errors specified for clean data. This is because the performance of all the algorithms were degraded in the presence of noise and outliers.

## 3.6 Conclusion

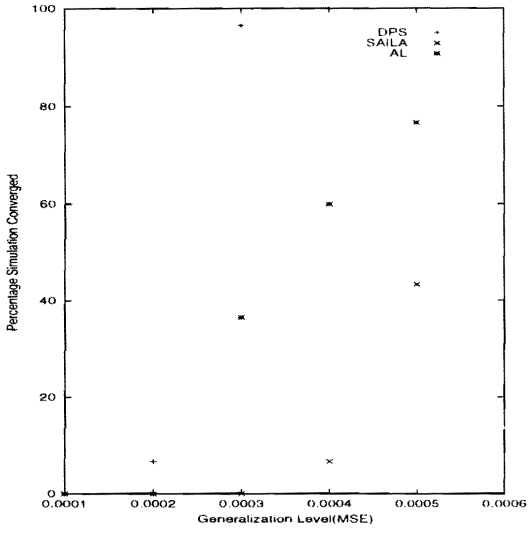
The objectives of the chapter were to present a new learning algorithm (SLA) and also to compare four active learning algorithms with respect to their accuracy, convergence and the complexity on both clean and noisy data as well as overfitting effects for the problems were also examined.

The results presented showed that AL was unstable, producing good results for the henon-map and F1 only. The bad training behavior can be attributed to the extremely small training set sizes used by AL, which is an indication of an inferior subset selection trigger.

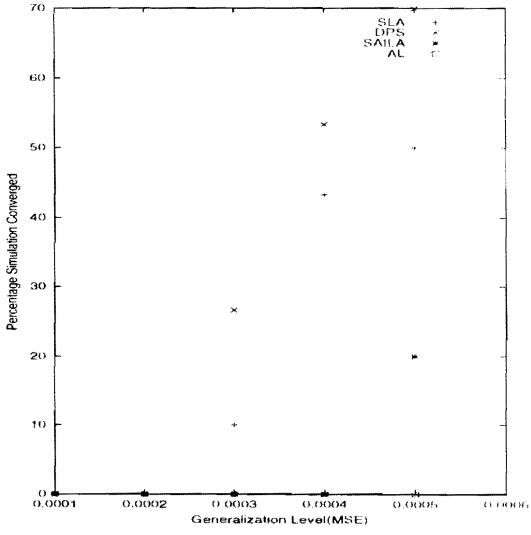
DPS and SLA performed very similar on the clean data, while SLA outperformed all the other algorithms on the noisy and outliers training data. The sensitivity analysis approach (SAILA) performed well under the occurrence of outliers and noisy time series, and very well for the complex function TS3. SAILA performed better than AL in TS1, TS4 and TS5, but worse than SLA and DPS. SAILA is computationally more expensive, requiring larger training subsets than the other algorithms.

As is expected, the performance of the error selection approaches degraded under the occurrence of outliers and noise. The degradation is due to the early selection of outliers, since outliers result in the largest prediction errors.

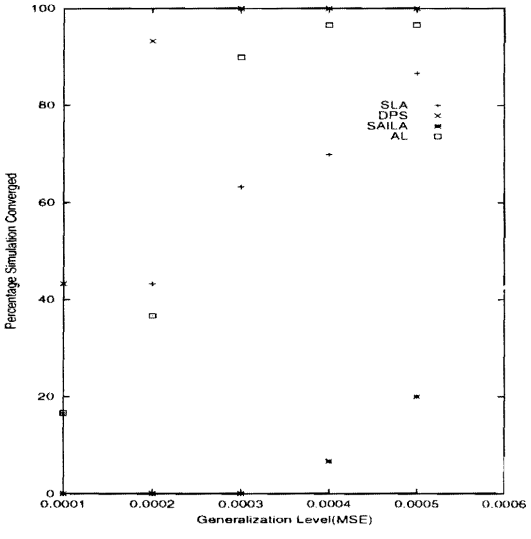
The comparison above showed that SLA had the best generalization performance, and lowest complexity. The selective learning approach (SLA) produced better accuracy



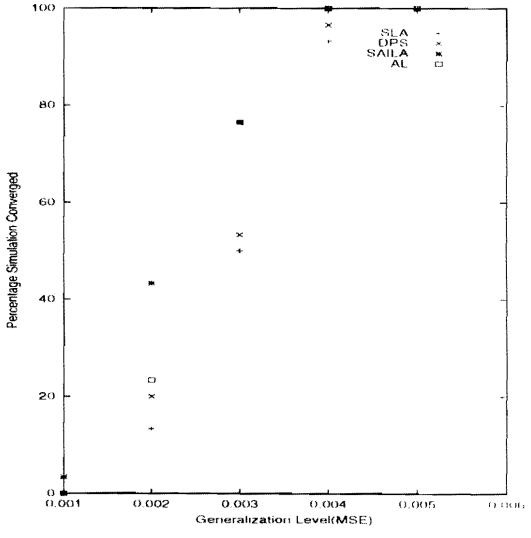

(a) F1




(b) TS1

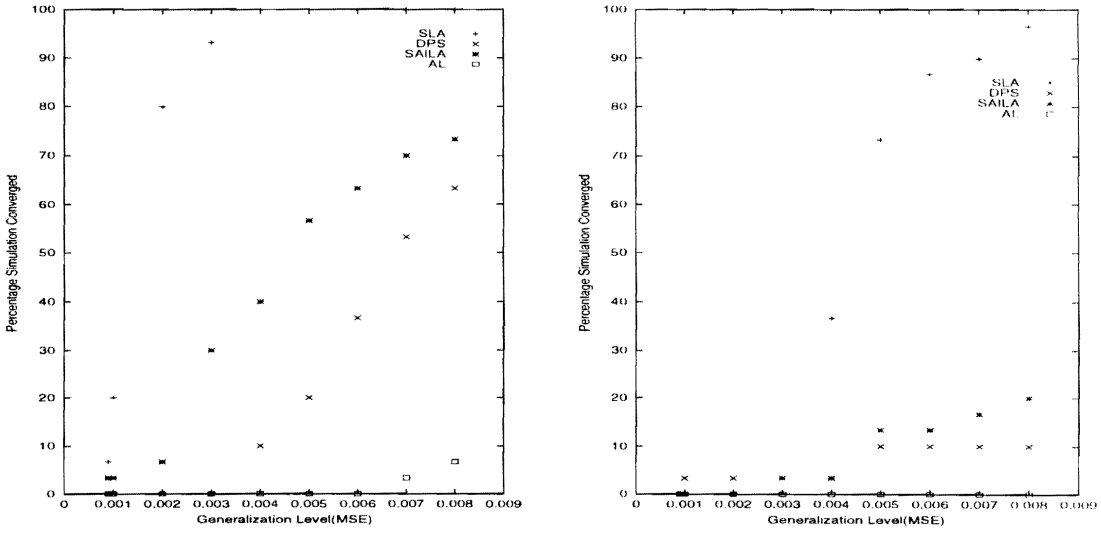



(c) TS2




(d) TS3

Figure 3.11: Percentage simulations that converged




(a) TS4 (b) TS5

Figure 3.12: Percentage simulations that converged

than the other approaches, and showed to be more robust in the occurrence of outliers and noise.

# Chapter 3

# ACTIVE LEARNING

One of the goals when designing and training a neural network is to maximize or to improve generalization, which is, the ability to give accurate response to data that the NN has not seen as part of the training process. In conventional backpropagation learning, all the available data are presented to the network for training. Learning on all the training data can be quite problematic especially when there are redundant data in the training set. The computational cost of training the network in terms of training time can become high, especially if these redundant data are included in the training set.

Studies have shown that selecting the most informative data for training rather than presenting all the available data to the network improves, or at least maintains the generalization performance. Selecting data for training also reduces training time and the data needed for training [Zhang 1994, Plutowski *et al* 1993, Engelbrecht *et al* 1998, Engelbrecht *et al* 1999a, Röbel 1994a].

This chapter discusses the concept and advantages of using of active learning. The objective of this chapter is to present a new selective learning algorithm and also to compare this new algorithm with three additional active learning algorithms with reference to their respective generalization performance, overfitting characteristics.

computational complexity and convergence characteristics.

## 3.1 Introduction

There have been various research efforts on improving the learning of BPNNs. These research efforts include finding optimal weight initialization [Rumelhart *et al* 1986], optimal learning rate and momentum [Plaut *et al* 1986 , Weir 1990, Yu *et al* 1997], finding the optimal architectures [Le Cun 1990, Karnin 1990, Hirose *et al* 1991, Pelillo *et al* 1993, Engelbrecht *et al* 1996], using second order optimization techniques [Becker *et al* 1988], adaptive activation functions [Zurada 1992a, Engelbrecht *et al* 1995, Fletcher *et al* 1998] and active learning [Zhang 1994, Engelbrecht *et al* 1999a, Engelbrecht *et al* 1998, Röbel 1994a]. This chapter concentrates on active learning as an approach to improve performance of NNs. Active learning is a technique in which patterns that have the highest influence on weight changes are dynamically selected by the NN learner from a candidate set of training patterns. The network utilizes current attained knowledge about the tasks to be learned as encapsulated in the current weights to select the most informative training patterns. There are two main approaches to active learning:

- **Incremental learning,** where patterns are selected and removed from a candidate training set. The selected patterns are added or injected into the actual training set. The effect is that the actual training set grows as training progresses, while the candidate training set is pruned.
- **Selective learning,** where a subset of the training patterns that satisfy a selection criterion is selected from a candidate training set and used for training. Unlike the incremental learning, the candidate training set is not pruned. At each pattern selection interval, all the patterns in the candidate set have a

chance to be selected. The candidate set remain fixed while the size of the actual training set varies from time to time.

A brief outline of this chapter is as follows: Section 3.2 discusses the concept of active learning and section 3.3 presents a general algorithm for active learning. A new selective learning strategy for time series problems is presented and compared with three other active learning algorithms in section 3.4. Section 3.5 presents results obtained from the different learning algorithms. Finally, section 3.6 highlights the conclusions, comments and observations.

## 3.2 Concept of Active Learning

Active learning has emerged as an efficient alternative to improve the performance of multilayer layer NNs. Active learning refers to the selection of a subset of the available training data dynamically during training, where the subset contains the most informative data. The objective of active learning algorithms is to identify, and train on the most informative patterns in a candidate training set. Active learning *efficiently* selects *optimal* training patterns from available training patterns for training the network. Efficiency refers to the complexity of the pattern selection mechanism which should be minimized. Optimal patterns are patterns that have useful information about the current state of the network and such patterns bring the network closer to the target function. The network plays an active role in data selection. Rather than being a passive learner, the network utilizes information based on its current state to gather useful information for further training.

Active learning addresses two fundamental questions:

– Which of the patterns should be selected for training from the candidate set?

– When should an additional set of patterns be selected?

Answers to these questions have resulted in the design of different approaches to active learning. These approaches mainly use the error in prediction as selection criterion [Cohn 1994, Röbel 1994a, Zhang 1994, Plutowski *et al* 1993] or changes in outputs due to perturbations in input parameters [Engelbrecht *et al* 1998, Engelbrecht *et al* 1999a, Engelbrecht *et al* 1999c]. Pattern selection has been the focus of many research. Infact, active learning has been called various names such as query learning, incremental learning, selective learning and dynamic pattern selection. All these terms refer to the same basic concept of selecting a subset of the most informative patterns from the candidate training set. Active learning algorithms aim at:

– Improving, or at least maintaining the generalization ability of the network.

– Reducing the cost of training the network in terms of the number of patterns needed for training. But selecting these patterns should not exceed the gain in computational cost reduction achieved by reducing the training set size.

– Improving the speed of convergence. Convergence is the ability to achieve certain generalization levels. Active learning aims at increasing the probability that the network will converge to given generalization levels, and doing so in as less time as possible.

Section 3.2.1 presents an overview of different approaches to active learning.

### 3.2.1 Overview on Active Learning

Plutowski and White used error in prediction as their selection criterion [Plutowski *et al* 1993]. The integrated squared bias (ISB) is used as the error term. Patterns that maximize the decrement in the ISB of the network resulting from

adding such patterns to the training subset are selected for training. Additional patterns are selected when the training error on the current training subset is sufficiently small. While Plutowski and White used the bias term of the MSE as the selection criterion, Cohn used the variance term of the MSE [Cohn 1994]. In this case, the most informative patterns are those that maximize the change in variance of the network resulting from adding these patterns to the network. The learner (NN) selects an additional training pattern at each time step, or epoch. In Mackay's algorithm, information theory was used to select patterns for training [Mackay 1992]. However, Mackay applied his active algorithm within bayesian framework. Fukumizu selected patterns that minimize the estimation error i.e. the expected value of the MSE [Fukumizu 1996] for training. Sung *et al* also used the error function as their selection criterion, but they considered both the bias and variance term [Sung *et al* 1996]. Patterns that minimize the expected misfit, i.e. the total output uncertainty between the target and the estimated target function are selected for training. Seung *et al* developed an active learning algorithm which they called Query by Committee (QBS) [Seung *et al* 1992]. In QBC, the degree of disagreement among the committee of learners (students) serves as an estimate of information value. The query that has the maximum disagreement among the committee of learners is chosen for training. That is, QBC selects an input classified as positive by half of the committee and negative by the other half. Freund *et al* presented a more complete and general analysis of QBC using the batch training algorithm [Freund *et al* 1997]. Hara *et al* applied an active learning algorithm to pattern classification problems [Hara *et al* 1988]. Patterns selected for training are those patterns that are close to the boundary of the pattern classes. Cohn *et al* combined active learning with statistical models (gaussian and weighted regression) [Cohn *et al* 1996]. Patterns that give the lowest expected model variance are selected for training.

Similar to Plutowski *et al*, Sung and Cohn, Zhang used the error in prediction as selection criterion [Zhang 1994]. Patterns whose addition cause the maximum approximation error to the network, are selected for training. However, the relative value rather than the absolute value of the error term is used. Röbel used the same error criterion as Zhang in selecting the most informative patterns for training[Röbel 1994a]. Röbel and Zhangs' algorithms only differ in the criterion that triggers the subset selection. Training on the current subset continues until some criteria are triggered (refer to section 3.4).

The change in outputs due to perturbations in input parameters can also used as selection criterion. Engelbrecht proposed an active learning algorithm where patterns with the highest influence on the outputs are selected for training [Engelbrecht *et al* 1999a]. First order derivatives of the output units with respect to the input units are used to quantify the influence a pattern has on the outputs. Another active learning algorithm that uses output perturbation as selection criterion is the selective learning algorithm (SLA) [Engelbrecht *et al* 1999c], developed in this thesis. Patterns that influence the output most are selected more for training than patterns that have little influence on the output. The influence on the output is reflected in the next-time-change in output values. Thus, patterns that have the large next-time-change in output values are selected more into the current training set than patterns with small-time-change in output.

This thesis selects four active learning algorithms based on their selection criteria for comparison. Two of these algorithms uses the error in prediction as selection criterion while the other two algorithms uses changes in output as their selection criteria. The algorithms selected are:

1. error based criterion
   - Accelerated learning using active learning, developed by Zhang [Zhang 1994].
   - Dynamic pattern selection (DPS), developed by Röbel [Röbel 1994a].

2. output based criterion
   - Sensitivity analysis incremental learning (SAILA), developed by Engelbrecht [Engelbrecht *et al* 1999a].
   - Selective learning algorithm (SLA), developed in this thesis.

While Cohn, Plutowski *et al* and Sung *et al* based their selection criteria on information theory, Zhang has shown that their approach is similar to selection of patterns using the prediction error as selection criterion [Zhang 1994]. Selection of patterns using the largest error is computationally less expensive than using information theoretic approaches. For these reasons, this thesis chose the algorithms developed by Röbel [Röbel 1994a] and Zhang [Zhang 1994] in its comparison instead of the information theoretic approach. The next section presents a general algorithm for active learning.

## 3.3 General Algorithm for Active Learning

This section presents a general algorithm for active learning and then discusses the algorithm design issues. Let $D_T$ be the current training set, which has all the patterns selected for training, $D_C$ be the candidate training set, which contains all the available patterns and $D_V$ be the validation set, which contains patterns not used as part of training and is used to test for overfitting.

A general algorithm for active learning is summarized below:

1. Initialize weights randomly as in conventional back propagation.
2. Select the most *informative* pattern(s) into training set $D_T$ from $D_C$
   - for incremental learning, add the selected pattern(s) into $D_T$ and remove them from $D_C$,

- for selective learning, select patterns into $D_T$.

3. Train the network for a training interval (i.e. adjust the weights) using $D_T$.
4. If the network has reached the desired accuracy or has reached the maximum number of epochs, stop training.
5. If a subset selection criteria triggers, repeat from step (2)
   otherwise repeat from step (3).

A training interval maybe one epoch for online training or $\epsilon$ epochs if batch training method is used.

**Design issues in active learning algorithm**

When designing and implementing an active learning algorithm, some issues have to be taken into consideration. One of these design issues is the number of patterns to select at each selection interval, referred to as the subset size. Although there is no heuristic to determine the size of the training subset, there are guidelines. For incremental learning, it is advisable to train with a small subset size because the subset grows during training. Selecting a small subset means patterns will be selected more often and thus increase the complexity of the network due to the cost of selecting the patterns. But, selecting a large number of patterns during training may defeat the aim of active learning, which is to reduce the number of patterns needed for training. Therefore, good selection criteria are desirable in selecting patterns that have useful information about the network. For selective learning, the subset size depends on the selection criteria. Therefore, selection criteria are important to ensure that the most informative patterns are selected for training.

Another important issue to consider is when to select additional patterns. One or more subsetselection criteria can be used. These subsetselection criteria should ensure that the NN does not train too long on a training subset because the network may overfit the training subset. The network should also train long enough to acquire maximum information on the current training subset. Different algorithms have

different subsetselection criteria. Patterns can be selected at each training epoch [Cohn 1994, Engelbrecht *et al* 1999c]. This criterion can have high computational cost because the subset selection function is applied at each epoch. An alternative is to select at each $\xi$ epochs. A reduction in computational cost will be achieved since the number of pattern selections is fewer. However, if the selection interval $\xi$ is too large, overfitting of the training subset may occur. Patterns can also be selected using the error on the training and the validation set. New patterns are selected into $D_T$ as soon as the error on the training or validation set reduces to a specified level [Zhang 1994, Engelbrecht *et al* 1999a].

Another criterion is to select new patterns as soon as the network overfits the training subset [Engelbrecht *et al* 1999a, Röbel 1994a]. Different algorithms use a criterion or combination of criteria to decide when to select additional patterns. The different subsetselection criteria used for the four selected active learning algorithms are discussed in more details in section 3.4.

## 3.4 A Comparative Study of Four Selected Active Learning Algorithms

The objective of this thesis is to compare selected active learning algorithms with reference to generalization performance and computational complexity. For this purpose, a new selective learning algorithm for time series problems developed in this thesis, sensitivity analysis for incremental learning [Engelbrecht *et al* 1999a], dynamic pattern selection of Röbel [Röbel 1994a] and accelerated learning using active example selection of Zhang [Zhang 1994], are compared with one another. This section presents an overview and critique of these algorithms.

### 3.4.1 A New Selective Learning Algorithm

Kohara presented an algorithm that performs pre-processing of the training set for time series problems [Kohara 1995]. Kohara's algorithm divides the training set into two sets. The one set contains all training patterns that reflect large-next-time changes in the time series, while the other contains patterns that reflect small-next-time changes. Kohara's algorithm assumes one output unit and also assumed a time ordering among the training patterns. Kohara uses target values to determine next-time changes. The next-time-change $\Upsilon_k^{(p)}$ for output $k$ is defined as

$$\Upsilon_k^{(p)} = t_k^{(p+1)} - t_k^{(p)}$$

The two training sets remain fixed during training. During training, patterns are more frequently selected from the large-next-time changes set than from the small-next-time changes set. Therefore, Kohara's algorithm is not considered as an active learning algorithm, because the neural network plays no role in the selection of patterns. Kohara's approach is rather referred to as a training set manipulation technique.

A new output based selective learning algorithm is proposed in this thesis based on Kohara's algorithm. Instead of using the target values to construct the two training subsets, the actual outputs of the network are used. Therefore, next-time-change $\Psi_k^{(p)}$ for output $k$ is defined as

$$\Psi_k^{(p)} = o_k^{(p+1)} - o_k^{(p)} \tag{3.1}$$

$\Psi_k^{(p)}$ can only be computed for the first $P_C - 1$ patterns, where $P_C$ is the number of patterns in the candidate training set $D_C$. The division of the original training set into large- and small-next-time changes sets is done after each selection interval, which is one epoch.

More patterns (80%) from the large-next-time changes set are randomly selected than from the small-next-time changes (20%) into the current training subset $D_T$. In doing this, the two subsets reflect the current knowledge of the learner, in that the set reflects what the learner perceives as large and small changes. More patterns are selected from the large-next-day changes set, since these patterns contain the most information about the characteristics of the time series. A large change in the output values, causes a large change in the the weights and a large change in weights means more information is gained in bringing the network's output closer to the target function (refer to weight update equations (2.13) and (2.16)).

Active learning is introduced by calculating the next-time-changes based on the actual output of the network and not on the target output values. At each epoch, the current training subset is discarded and a new subset $D_T$ is selected with training patterns. The training set $D_T$ is then used for training.

The algorithm for SLA is summarized below:

1. Initialize weights, learning rate and momentum.
2. Calculate the output $o_k^{(p)}$ of the network for each pattern $p$. Then, calculate next-time-changes as

$$\Psi_k^{(p)} = o_k^{(p+1)} - o_k^{(p)} \ , \ \forall_p \in D_C$$

3. Separate patterns into a small-time and a large-time-change set:
   - calculate the average next-time-changes:

$$\overline{\Psi} = \frac{\sum_{p=1}^{P_C-1} \Psi^{(p)}}{P-1}$$

   - divide the candidate patterns into the two training subsets:
     Add all patterns $p$ for which $\Psi_k^{(p)} > \overline{\Psi}$ to the large-change set and all other patterns into the small-change set.
4. Select the actual training set $D_T$ to consist of

- 80% of the large-change set, randomly selected
- 20% of the small-change set, randomly selected.

5. Train the network for one epoch using $D_T$
6. Repeat steps (2 - 5) until the number of epochs exceeds the maximum number of epochs allowed.

The subset size depends mainly on the size of the large-next-time changes set. The performance of the proposed algorithm is compared to the other algorithms and the results are discussed in the section 3.5

### 3.4.2 Sensitivity Analysis Incremental Learning Algorithm

The second algorithm to be studied is an *incremental learning* approach to active learning which uses an output based selection criterion, referred to as sensitivity analysis incremental learning algorithm (SAILA). SAILA is developed by Engelbrecht [Engelbrecht *et al* 1999a]. In SAILA, the most informative patterns are perceived as those patterns that maximally influence the output of the NN in the presence of small input perturbations. First order derivatives of the output units with respect to the input units are used to compute the influence the pattern has on the output value of the function approximated by the network. Patterns with the highest sensitivity cause the largest change in weights (large change in weights achieve maximum gain in bringing the approximation closer to the true function). These patterns lie in the region of the peaks' derivatives. Thus, the partial derivatives $\frac{\partial o_k}{\partial z_i^{(p)}}$ is calculated for each input and output for each pattern. Training on such patterns yield better generalization and faster convergence [Engelbrecht *et al* 1999a]. The sensitivity of each pattern is determined by computing the informativeness of the pattern, as

$$\Phi^{(p)} = \max\{S_{o,k}^{(p)}\} \tag{3.2}$$

where

$$S_{o,k}^{(p)} = \sqrt{\sum_{i=1}^{I} (\frac{\partial o_k}{\partial z_i^{(p)}})^2} \tag{3.3}$$

The larger the value of $\Phi^{(p)}$, the more informative that pattern is. Pattern(s) with the largest absolute value of $\Phi^{(p)}$ are selected into $D_T$.

SAILA continues training on the training subset to achieve maximum gain from the patterns before selecting additional patterns. At the same time, the network must not be allowed to memorize the training subset. The network should therefore not spend too much time on the current training set. SAILA uses the following subset selection criteria:

1. The algorithm limits the number of training epochs on the current subset. The criterion ensures that the NN does not train indefinitely on the subset. Engelbrecht limited the number to 100 in his implementation.
2. If the error on $D_T$, or the validation set $D_V$, decreases sufficiently, a new subset is selected for training. The criterion ensures that the NN achieves sufficient gain on the current training subset before selecting additional patterns. In Engelbrecht's implementation, an additional pattern is selected into $D_T$ as soon as the error on the $D_T$ or $D_V$ decreases by 80%.
3. A new subset is also selected if the average decrease in error on $D_T$ and $D_V$ since training started on the current subset is small. The criterion will prevent the learner from training on $D_T$ with achieving too little gain.
4. If the error $\mathcal{E}_V$ on the validation set increases too much, a new subset is selected. The subset selection criterion prevents the NN from memorizing the current training subset by triggering a new subset selection as soon as overfitting of $D_T$ is observed.

The sensitivity analysis incremental learning algorithm is summarized below [Engelbrecht *et al* 1999a]:

1. Initialize weights, momentum and learning rate. Initialize the subset size, $P_{S_s}$, i.e the number of patterns selected from the candidate set $D_C$. Construct the initial training subset $D_{S_o} \subset D_C$. Let $D_T \leftarrow D_{S_o}$ be the current training set.

2. Repeat

   - Repeat

     Train the NN on training set $D_T$

     until a termination criterion on $D_T$ is triggered (as discussed above).

   - Compute the new training subset $D_{S_s}$

     * For each $p \in D_C$, compute the sensitivity matrix $S^{(p)}_{oz,ki}$ for sigmoid activation functions:

$$S^{(p)}_{oz,ki} = f^{(p)'}_{o_k} \sum_{j=1}^{J} w_{kj} f^{(p)'}_{y_j} v_{ji}$$

     * Compute the output sensitivity vector $\vec{S}^{(p)}_{o,k}$ for each $p \in D_C$

$$\vec{S}^{(p)}_{o,k} = ||S^{(p)}_{oz}||$$

     * Compute the informativeness $\Phi^{(p)}$ of each pattern $p \in D_C$ using

$$\Phi^{(p)} = \max_{k=1,\cdots,K} \{|S^{(p)}_{o,k}|\}$$

     * Find the subset $D_{S_s}$ of the $P_{S_s}$ most informative patterns as

$$D_{S_s} \leftarrow \{p \in D_C | \Phi^{(p)}_{\infty} = \max_{q=1,\cdots,P_C} \{\Phi^{(q)}_{\infty}\}; \forall q \in D_C, not \;\; yet \;\; selected\}$$

     where $P_C$ is the number of patterns remaining in $D_C$. Then, let $D_T \leftarrow D_T \cup D_{S_s}$ and $D_C \leftarrow D_C - D_{S_s}$

   until convergence is reached.

In Engelbrecht's implementation, SAILA started training with one pattern, and selects only one new pattern at each subsetselection interval. Although the subset

selection criteria implemented by SAILA considers overfitting effect and generalization of the network, the cost of selecting patterns in SAILA could be high. Because SAILA has more criteria to implement when selecting new patterns for training. Results of SAILA when applied to function approximation and time series problems are presented in the section 3.5.

### 3.4.3 Dynamic Pattern Selection

The third active learning algorithm studied is Röbel's dynamic pattern selection technique (DPS). Unlike SAILA and SLA, DPS uses the error indication as selection criterion. DPS is an example of incremental learning , where informativeness of each pattern is measured using the prediction error. The prediction error is computed as $(t_k^{(p)} - o_k^{(p)})^2$. Patterns with the largest prediction error are the most informative and are selected for training.

Training continues on the current training subset until the subset starts to overfit. To measure overfitting, Röbel defined the generalization factor $\rho$ as

$$\rho = \frac{E_V}{(E_T + E_C)}$$

where $E_V$, $E_T$ and $E_C$ are the error functions on the validation set, training subset and the candidate training set respectively. By requiring that $\rho \leq 1.0$, overfitting is prevented. A value of $\rho$ greater than one means that the validation error is larger than the training the training error, hence bad generalization. New patterns are therefore selected into $D_T$ when $\rho$ grows beyond one. However, Röbel discovered that each pattern selected for training decreases the value of $\rho$ to a minimum value before $\rho$ slowly increases again and therefore takes a long time to reach the value of one. This means that the network will train too long on the current training subset if only the selection criterion of $\rho > 1.0$ is implemented. Thus, new patterns are also selected as soon as $\rho$ reaches a minimum threshold value. For these purpose, a

threshold $\phi_\rho$ is defined as

$$\phi_\rho(\xi) = \min\{\phi_\rho(\xi - 1), \overline{\rho} + \sigma_\rho, 1.0\}$$

where $\xi$ is the current training epoch, $\overline{\rho}$ and $\sigma_\rho$ are the average and standard deviation of the generalization factor for $M$ preceding epochs respectively. Röbel used 100 for the value of $M$. The DPS algorithm selects new patterns whenever $\rho(\xi) > \phi_\rho(\xi)$; $\rho(\xi)$ is the generalization factor for the current training epoch.

The algorithm for DPS is summarized below:

1. Initialize the weights and set threshold $\phi_\rho(\xi)$ to one with $\xi = 1$.
2. For each pattern $p$ in $D_C$, calculate the SSE as $E^{(p)} = \sum_{k=1}^{K} (t_k^{(p)} - o_k^{(p)})^2$.
3. Select pattern(s) with the highest error $E^{(p)}$ into $D_T$ and remove the selected pattern(s) from $D_C$.
4. Train the network.
5. If the number of epochs exceeds the maximum number of epochs or the error limit has been reached, stop training.
6. Calculate the generalization factor $\rho = \frac{E_V}{(E_T + E_C)}$
7. if $\rho$ is greater than $\phi_\rho(\xi)$, then set $\phi_\rho(\xi + 1) = \min(\rho(\xi), 1.0)$ and repeat from step (2), otherwise set $\phi_\rho(\xi + 1) = \min\{\phi_\rho(\xi), \overline{\rho} + \sigma_\rho, 1.0\}$ and repeat from step (4).

Röbel used a subset size of one pattern and selects a pattern when the subsetselection criterion is triggered. The online cross validation technique is used to check for the generalization. That is, a separate data is used compute the validation error of the network. Additional overhead is incurred in DPS for implementing the cross validation technique. If the training data is limited, having a separate set for validation may not be feasible.

While the selection criterion is easy to compute, performance degrades if the training data have outliers and noise. Outliers will be selected as the most informative patterns, since these patterns have the maximum prediction errors. The network is then biased towards the outliers. Consequently, the ability of the network to generalize deteriorates. Results of DPS when implemented are discussed in section 3.5.

### 3.4.4 Accelerated Learning by Active Example Selection

The last algorithm to be studied is accelerated learning by active example selection (AL) proposed by Zhang [Zhang 1994]. AL is an incremental learning approach. AL selects as the most informative patterns, those patterns that have the maximum prediction error, where the prediction error is computed as $(t_k^{(p)} - o_k^{(p)})^2$.

New patterns are selected for training when the error on the training subset is reduced to a specified performance level $\kappa$, where $\kappa$ is computed as

$$\kappa = \frac{J(I+K)}{\tau}$$

$\tau$ is the allowable error per connection. Zhang suggested a value of $\tau \in [100, 200]$.

Zhang motivates the selection criterion on the fact that the learning capacity of a NN is proportional to the total number of adjustable connections in the network, which is $J(I+K)$ [Zhang 1994].

The algorithm for Zhang's accelerated learning is

1. Initialize weights to small random values.
2. For each pattern $p$ in $D_C$, compute the SSE as

$$E^{(p)} = \sum_{k=1}^{K} (t_k^{(p)} - o_k^{(p)})^2$$

3. Select pattern(s) with the highest error into $D_T$ and remove from $D_C$.

4. Train the network.
5. If the maximum number of epochs is exceeded or if the error limit has been reached, stop training.
6. Compute $\kappa = \frac{J(K+I)}{\tau}$ and SSE on $D_T$ as $E_T = \sum_{p=1}^{P} \sum_{k=1}^{K} (t_k^{(p)} - o_k^{(p)})^2$.
7. If $\kappa \leq E_T$ then repeat from step (2),
   otherwise repeat from step (4).

Even though AL's subset termination criterion is less complex and easy to compute, AL does not test for generalization of the network. Overfitting may therefore still occur. The subselection termination criterion depends on the architecture of the network being trained. If the wrong architecture is selected (either undersized or oversized) this criterion may not perform well. Due to the selection of patterns with the largest prediction error, the performance of AL may deteriorate in the presence of outliers and or noise.

The results and performance of AL are discussed in section 3.5.

## 3.5 Experimental Results

Four approximation and times series problems of varying complexity were used to test the performance of SLA, SAILA, DPS and AL. These problems differ in input dimensions and the number of hidden units needed to train the network. Table 3.1 shows a summary of the NN architecture used for these problems. In table 3.1, the architecture of a NN is referred to as I-J-K where I is the number of input units, J is the number of hidden units and K is the number of output unit i.e. the notation 2-5-1 means two input, five hidden and one output units are used.

All the available data was split into three sets: the candidate training set $D_t$, validation set $D_V$ and generalization set $D_G$. The three sets were randomly created such

| *Problem* | *Equation* | $P_C$-$P_G$-$P_V$ | *Architecture* |
|---|---|---|---|
| F1 | (3.3) | 600 -200-200 | 2-5-1 |
| TS1 | (3.4) | 600-200-200 | 1-5-1 |
| TS2 | (3.5) | 600-200-200 | 2-5-1 |
| TS3 | (3.6) | 140-60-60 | 10-10-1 |
| TS4 | | 600-200-200 | 2-5-1 |
| TS5 | | 600-200-200 | 2-7-1 |

Table 3.1: Summary of the functions and time series used

that

$$\begin{aligned} D_C \cap D_V &= \emptyset \\ D_C \cap D_G &= \emptyset \\ D_G \cap D_V &= \emptyset \end{aligned}$$

Let $P_C$ be the number of training patterns in $D_C$, $P_V$ the number of training patterns in $D_V$ and $P_G$ the number of patterns in test set $D_G$. Table 3.1 shows the size of these sets for each problem. $D_C$ is the candidate training set from which training patterns are selected. $D_V$ contains data used to determine the generalization factor during training. $D_G$ contains data used to determine the generalization performance of the network.

The performance of the active learning algorithms was tested on clean and noisy data, as well as data containing outliers. Section 3.5.1 explains the experimental procedure, including a discussion of the performance criteria used to compare the learning algorithms. The results are compared in section 3.5.2.

The characteristics of the functions and time series used for experimentation are discussed next. The following functions and time series were used:

1. Function $F1$ is defined as (see figure 3.1(a))

$$F1 : F(z_1, z_2) = \frac{1}{2}(z_1^2 + z_2^2) \tag{3.4}$$

where $z_1, z_2 \sim U(-1, 1)$. All target values were scaled to the range [0,1].

2. Time series TS1 is a sine function defined as (see figure 3.1(b)),

$$TS1 : F(z) = \sin(2\pi z)e^{(-z)} + \zeta \tag{3.5}$$

where $z \sim U(-1, 1)$ and $\zeta \sim N(0, 0.1)$. Target values were scaled to the range [0,1].

3. Time series TS2 is the henon-map function defined as (refer to figure 3.1(c)),

$$\begin{aligned} TS2 : o_t &= z_t \\ z_t &= 1 + 0.3z_{t-2} + 1.4z_{t-1}^2 \end{aligned} \tag{3.6}$$

where $z_1, z_2 \sim U(-1, 1)$. The target values were scaled to the range [0,1].

4. Time series TS3 is a difficult time series, having 10 input parameters of which 7 are irrelevant (see figure 3.2(c)).

$$\begin{aligned} TS3 : o_t &= z_t \\ z_t &= 0.3z_{t-6} - 0.6z_{t-4} + 0.5z_{t-1} + 0.3z_{t-6}^2 - 0.2z_{t-4}^2 + \zeta_t \end{aligned} \tag{3.7}$$

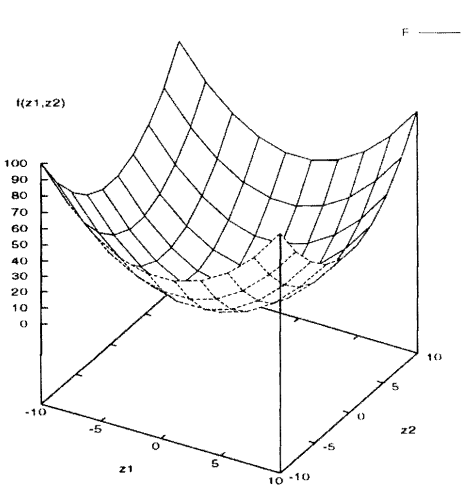
for $t = 1, \cdots, 10$, where $z_4, z_6, z_9 \sim U(-1, 1)$ and $\zeta_t \sim N(0, 0.05)$. All target values were scaled to the range [0,1].

5. Time series TS4 is a convolution of two discrete functions with outliers. Figure 3.2(a) shows an illustration of this function.

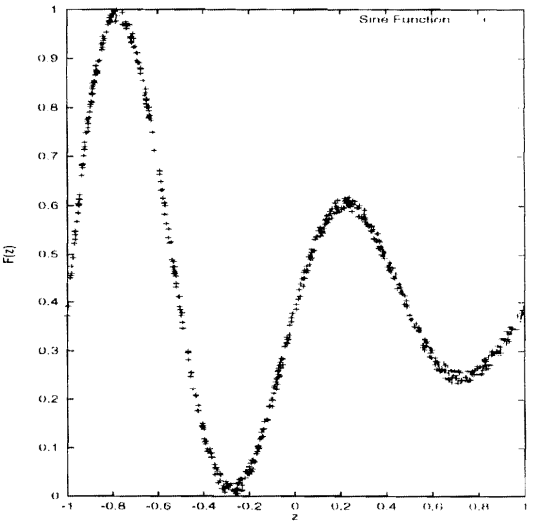
6. Time series TS5 is the sine function TS1 with 5% of the candidate training set consisting of outliers (see figure 3.2(b)).

### 3.5.1 Experimental Procedure

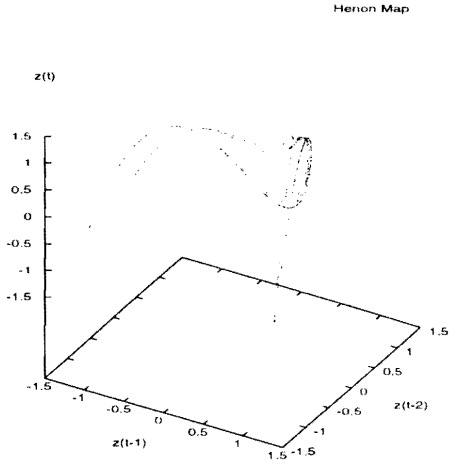
In order to obtain statistically valid assertions in comparing experimental results of the four learning algorithms, thirty simulations were performed for each problem.




(a) F1

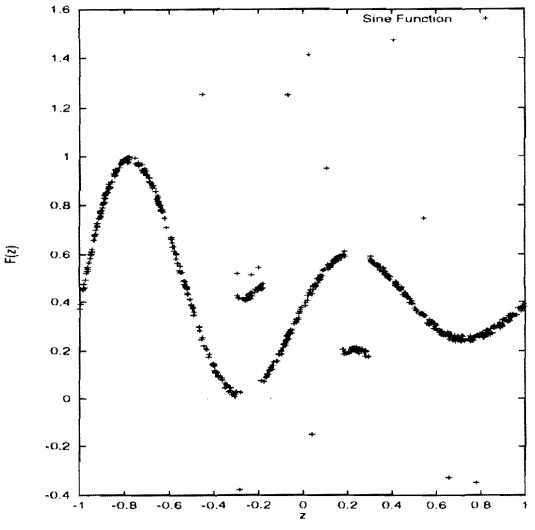



(b) TS1

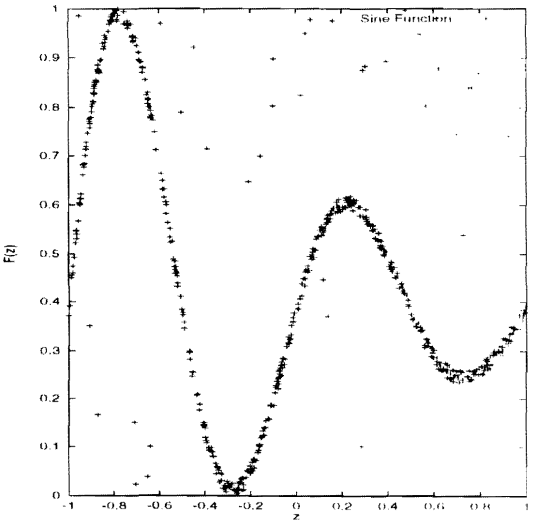



(c) TS2

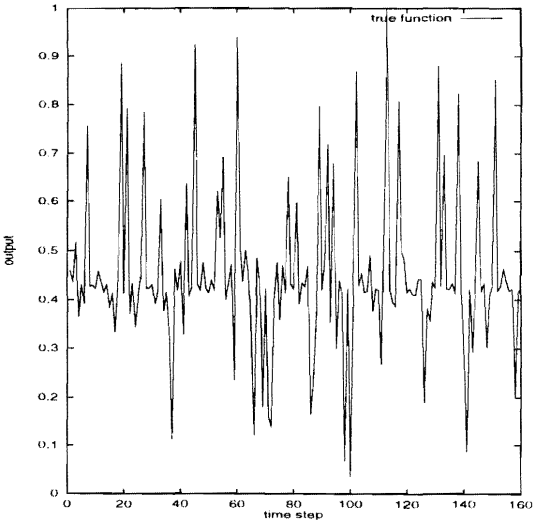
Figure 3.1: Function and Time series problems to be approximated




(a) TS4




(b) TS5




(c) TS3

Figure 3.2: Time series problems to be approximated

Online training was used for the active learning algorithms. The initial subset size for incremental learning algorithms consisted of one pattern and a subselection size of one pattern was used. Each simulations was executed for 2000 epochs. A learning rate 0.1 and momentum 0.9 were used for all the approximation problems . Results reported are averages over the 30 simulations together with 95% confidence intervals as obtained from the t-distribution.

The selective learning algorithm was not applied to F1, since F1 is not a time series problem. The $\tau$ value used in the subset selection criterion for AL was adjusted for each problem using a trial and error approach. For TS3, a high $\tau$ was used ($\tau = 1000$), a value of 100 was used for TS1, TS4 and TS5 while a value of 180 was used for TS2 and F1.

**Performance measures**

To evaluate the performance of each learning algorithm, the following performance criteria were used:

1. The mean squared error (MSE) was used as a measure of accuracy. The MSE measures how well a function is approximated by the network, and is defined as

$$MSE = \frac{\sum_{p=1}^{P} \sum_{k=1}^{K} \left(t_k^{(p)} - o_k^{(p)}\right)^2}{2\ K\ P}$$

.

A MSE value close to zero shows a small error between the target and the output function. The MSE over the three sets $D_V$, $D_G$ and $D_C$ were computed. The MSE over $D_G$, denoted by $E_G$ provides an unbiased estimate of the generalization error since the patterns in $D_G$ were not used for training.

2. Röbel's generalization factor $\rho$ was used to measure overfitting effects. The generalization factor was computed as $\rho = \frac{E_V}{E_C}$, where $E_C$ is the MSE over candidate training set $D_C$ and $E_V$ is the MSE over the validation set $D_V$. A network overfits when the value of $\rho$ increases substantially above 1.

3. The computational complexity of learning algorithms was also used as performance criterion. For the purpose of this thesis, computational cost is measured as the number of calculations needed to train the network. Calculations include subtraction, multiplication, addition and division.

   At any epoch $\xi$, the cost $C_{fc}$ of training a NN on a training set, is expressed as

   $$C_{fc} = (C_V + C_W) * P_T$$

   where $C_V$ is the cost of updating weights between input and hidden units and $C_W$ is the cost of updating weights between hidden and output units. $P_T$ is the number of patterns in the training subset $D_T$. For conventional backpropagation with fixed set learning, $P_T = P_C$. Thus the cost of training $C_{fsl}$ is computed as $C_{fsl} = (C_V + C_W) * P_C$.

   The costs of updating the weights are calculated as

   $$\begin{aligned} C_V &= C_v * (N_V) \\ C_W &= C_w * (N_W) \end{aligned}$$

   where $C_v$ is the cost of updating a single weight between the input and hidden layers, $C_w$ is the cost of updating a single weight between the hidden and output layers. $C_V$ is the total cost of updating the weight connections between the input and the hidden layers, and $C_W$ is the total cost of updating the weight connections between the hidden and output layer. $N_V$ is the total number of connections between the input and hidden layers and $N_W$ is the total number of connections between the hidden and output layers.

   The total number of connections $N_V$ and $N_W$ are expressed as

   $$N_V = (I + 1) * (J + 1)$$
   $$N_W = (J + 1) * (K)$$

   and

   $$C_v = 13$$

$$C_w = 11$$

Therefore,

$$\begin{aligned} C_V &= 13 * (I+1) * (J+1) \\ C_W &= 11 * (J+1) * K \end{aligned} \tag{3.8}$$

The cost of training a network using any active learning algorithm includes $C_{fc}$, the cost of selecting patterns for training and the cost of computing the subset termination criterion. Therefore, at any epoch $\xi$, the cost of training a network using SLA, SAILA, DPS and AL are:

$$\begin{aligned} C_{SL} &= C_{fc} + C_{sla} * P_C \\ C_{DP} &= C_{fc} + C_{dps} * (P_C - P_T) + (C_{S_{dps}} * P_T) \\ C_{AL} &= C_{fc} + C_{al} * (P_C - P_T) + (C_{S_{al}} * P_T) \\ C_{SA} &= C_{fc} + C_{sai} * (P_C - P_T) + (C_{S_{sai}} * P_T) \end{aligned}$$

For all the incremental learning algorithms, the subset selection criteria are tested on the remaining patterns in the candidate set $D_C$ which is equal to $P_C - P_T$. Also, for incremental learning algorithms, an additional cost of selecting pattern is incurred when a pattern is selected.

$C_{SL}$, $C_{SA}$, $C_{AL}$ and $C_{DP}$ are the cost of training a network using SLA, SAILA, AL and DPS respectively. $C_{sla} = 15$ is the cost of computing the subset selection criteria for SLA, $C_{dps} = 11$ is the cost of computing the subset selection criteria for DPS, $C_{al} = 4$ is the cost of computing the subset selection criteria for AL and $C_{sai} = 18$ is the cost of computing the subset selection criteria for SAILA. $C_{S_{dps}} = 2$, $C_{S_{al}} = 2$ and $C_{S_{sai}} = 7$ are the cost of selecting patterns into $D_T$ for DPS, AL and SAILA respectively.

Therefore,

$$C_{SL} = C_{fc} + 15P_C$$

$$\begin{aligned} C_{DP} &= C_{fc} + 11(P_C - P_T) + 2P_T \\ C_{AL} &= C_{fc} + 4(P_C - P_T) + 2P_T \\ C_{SA} &= C_{fc} * P_T + 18(P_C - P_T) + 7P_T \end{aligned} \tag{3.9}$$

From equation (3.9), the cost of training is directly proportional to the number of patterns selected for training. The more patterns are selected for training. the higher the computational cost. Initially, $P_T$ for SLA is greater than the other algorithms because DPS, AL and SAILA are incremental learning algorithm and a small initial trainig set and subset size is used in the simulations. Thus, $C_{SL}$ is expected to be greater than $C_{AL}$, $C_{DPS}$ and $C_{SA}$ initially. SAILA is computationally more expensive in selection criteria than the other algorithms because SAILA has more subset selection criteria to implement than the other algorithms.

Section 3.5.2 illustrates the costs for the different algorithms.

### 3.5.2 Results

This section presents the results of the simulations carried out on the active learning algorithms.

**Training error**

In order to compare the performance of the four active learning algorithms. the MSE over the candidate set $D_C$ was computed for the simulations and the average calculated. Tables (3.2) and ( 3.3) show the result over 2000 epochs for clean data and data with noise and outliers.

For TS1, DPS had a very low error with the lowest variance which means that all the errors of the simulations for DPS were all closer to the average error of 0.0003. Although, SLA had a low error as well. However SLA had a large variance when compared to DPS. AL had the largest error with a very large variance.

DPS achieved the smallest error for TS2, having a small variance. For TS3, all the algorithms had very low errors but SAILA had a high variance. DPS had the smallest error for F1 with a very small variance.

For TS4 and TS5, SLA achieved the smallest error with the lowest variance. AL had the largest error for TS4 and TS5. This is because AL selected and trained on just a single pattern for TS4 and an average of 4 patterns for TS5. Thus AL had high errors for TS4 and TS5.

The training errors for all the problems with noise and outliers were larger ( $\times 10^2$) than for problems with clean data. DPS had the lowest average error for clean data while SLA had the lowest error for noisy data.

**Generalization error**

To compare the generalization ability of the four active learning algorithms, the MSE over the generalization set, $E_G$, was computed and the average over the 30 simulations was plotted as a function of number of epochs. Figures 3.3 and 3.4 illustrates the trend of the generalization errors for the entire training period.

DPS achieved a very low average error faster than the other algorithms for F1 (refer to figure 3.3(a)). However, both SAILA and AL achieved a comparable result to DPS at epoch 1000. From table 3.3, DPS had the lowest error with the a very small variance ($5.07E-05$) which means all errors of the simulations are closer to the average.

For TS1, SAILA initially had the highest generalization error but decreased to a low level of error (see figure 3.3(b)). SLA initially had the lowest average error, which can be explained by the fact that SLA used more patterns initially than the other algorithms (refer to figure 3.7(b)). Although SLA and DPS had small errors, DPS had the smallest variance and thus DPS achieved the smallest error. AL had the largest error after 2000 epochs with a large confidence interval.

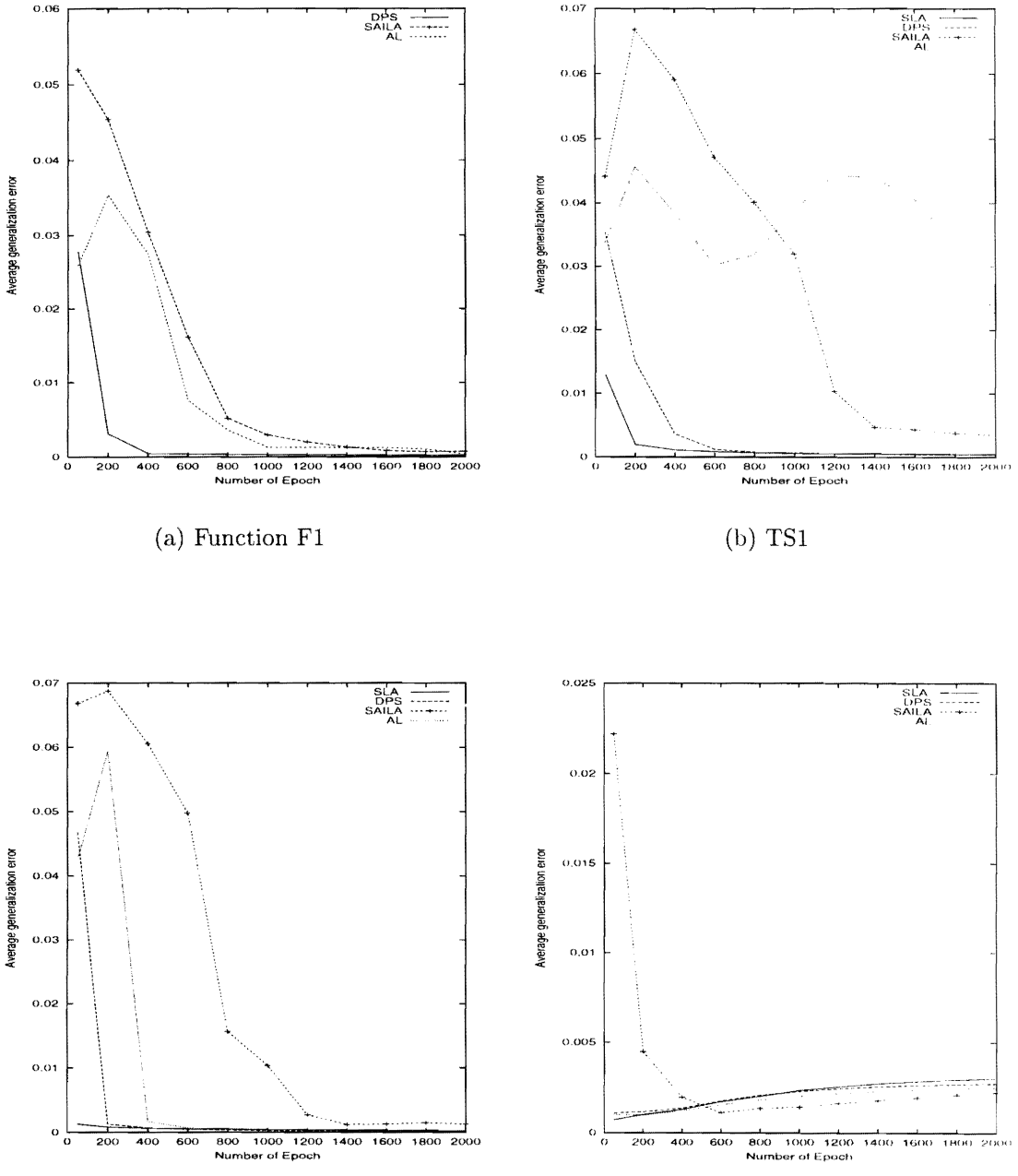
For TS2, DPS, AL and SLA achieved a very low average error before epoch 500.

| Function | Röbel | Zhang | Selective Learning | Sensitivity Analysis |
|---|---|---|---|---|
| **TS1** | | | | |
| Training Error | 0.00036 ± 0.00017 | 0.02172 ± 0.04186 | 0.00045 ± 0.00035 | 0.00346 ± 0.00712 |
| Generalization | 0.00039 ± 0.0002 | 0.02241 ± 0.04191 | 0.00047 ± 0.00040 | 0.0035 ± 0.00737 |
| Used Patterns | 485.43 ± 234.79 | 4.73 ± 0.92 | 270.93 ± 3.48 | 571.67 ± 88.91 |
| **TS2** | | | | |
| Training Error | 0.00014 ± 0.00011 | 0.00023 ± 0.00021 | 0.00029 ± 0.00038 | 0.00126 ± 0.00163 |
| Generalization | 0.00012 ± $0.25E-05$ | 0.00022 ± 0.00019 | 0.00029 ± 0.00037 | 0.00129 ± 0.00169 |
| Used Patterns | 411.77 ± 215.87 | 174.63 ± 61.48 | 272.57 ± 7.61 | 522.57 ± 173.37 |
| **TS3** | | | | |
| Training Error | 0.00039 ± 0.00086 | 0.00044 ± 0.00091 | 0.00050 ± 0.00085 | 0.00068 ± 0.00146 |
| Generalization | 0.00275 ± 0.00155 | 0.00253 ± 0.00133 | 0.00302 ± 0.00138 | 0.00225 ± 0.00174 |
| Used Patterns | 180 ± 0 | 180 ± 0 | 78.17 ± 1.53 | 180 ± 0 |

Table 3.2: Comparison results over 2000 epochs for times series problems

| Function | Röbel | Zhang | Selective Learning | Sensitivity Analysis |
|---|---|---|---|---|
| **F1** | | | | |
| Training Error | 0.000226 ± 5.07$E-05$ | 0.000412 ± 0.000366 | | 0.000791 ± 0.001852 |
| Generalization | 0.000221 ± 5.2$E-05$ | 0.000392 ± 0.000347 | | 0.000764 ± 0.001624 |
| Used Patterns | 320.2 ± 167.6698 | 82.8 ± 32.37935 | | 445.1333 ± 121.476 |
| **TS4** | | | | |
| Training Error | 0.01141 ± 0.00573 | 0.19935 ± 0.03093 | 0.00516 ± 0.00393 | 0.02828 ± 0.09522 |
| Generalization | 0.01077 ± 0.00534 | 0.19051 ± 0.02169 | 0.00478 ± 0.00349 | 0.02739 ± 0.09170 |
| Used Patterns | 493.03 ± 193.37 | 1 ± 0 | 245.23 ± 7.89 | 597.03 ± 27.41 |
| **TS5** | | | | |
| Training Error | 0.00683 ± 0.00468 | 0.10278 ± 0.08731 | 0.00155 ± 0.00158 | 0.00562 ± 0.00768 |
| Generalization | 0.00714 ± 0.00489 | 0.09904 ± 0.08932 | 0.00158 ± 0.00142 | 0.00595 ± 0.00842 |
| Used Patterns | 103.5 ± 20.14 | 4.67 ± 1.35 | 269.13 ± 9.89 | 584 ± 93.4 |

Table 3.3: Comparison results over 2000 epochs for problems F1 and times series with noise and outliers



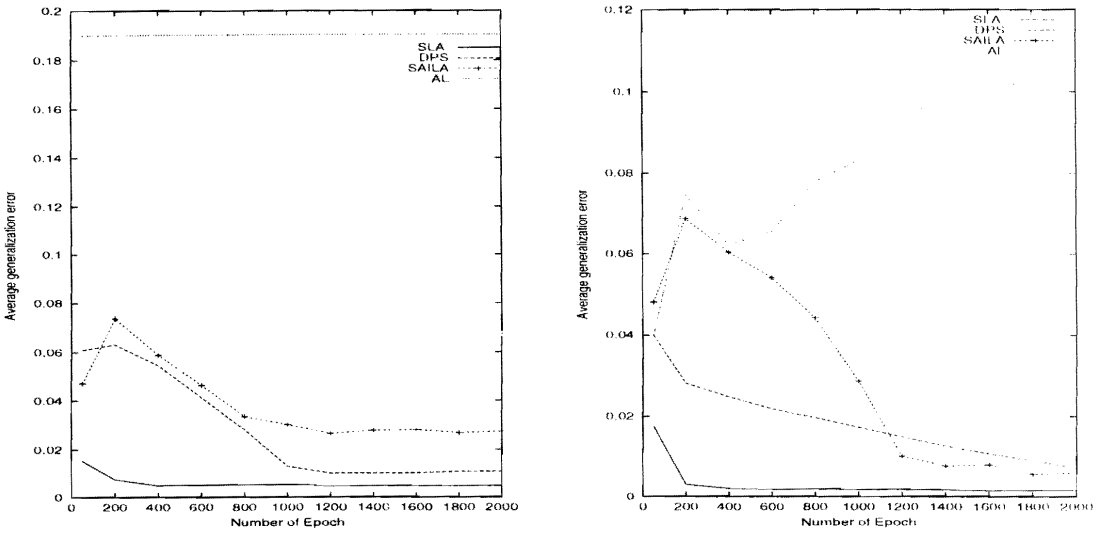

(a) Function F1

(b) TS1

(c) TS2

(d) TS3

Figure 3.3: Average generalization error vs epoch




(a) TS4 (b) TS5

Figure 3.4: Average generalization error vs epoch

SAILA was slower to achieve a comparable low error but SAILA had a low error by the end of training. From the table 3.2, DPS had the smallest error with a very small variance after 2000 epochs, implying that all errors of the simulations are closer to the average.

For TS3, the generalization error for all the algorithms increased as the number of epochs increased except SAILA (see figure 3.3(c)).

From the table 3.3, SLA had the lowest generalization errors for TS4 and TS5. While AL had the largest error for TS4 and TS5. AL did not learn the functions (refer to figure 3.4(a)). AL selected a few patterns for training, thus had little information about the time series to be approximated and therefore AL had a bad generalization.

DPS had the lowest generalization errors for functions with clean data while SLA had the lowest generalization errors for functions with noise and outliers. Although DPS had better generalization with clean functions than SLA, DPS used more patterns than SLA to achieve the low generalization error in all the problems. AL had very large generalization errors for TS1, TS4 and TS5. This bad generalization can be

attributed to the extremely small training set sizes used by AL which is an indication of an inferior subset selection criterion. The subset selection criterion depends on the number of connections in the network. Redundant or irrelevant weights in the network will make the value of the performance level $\kappa$ very large which can cause the network to train on the current training subset $D_T$ too long without selecting additional patterns. Thus the network selects a few patterns, hence having insufficient information to train the network. On the other hand, too few weights in the network can make $\kappa$ small. Thus, the network selects patterns more often than are needed for training.

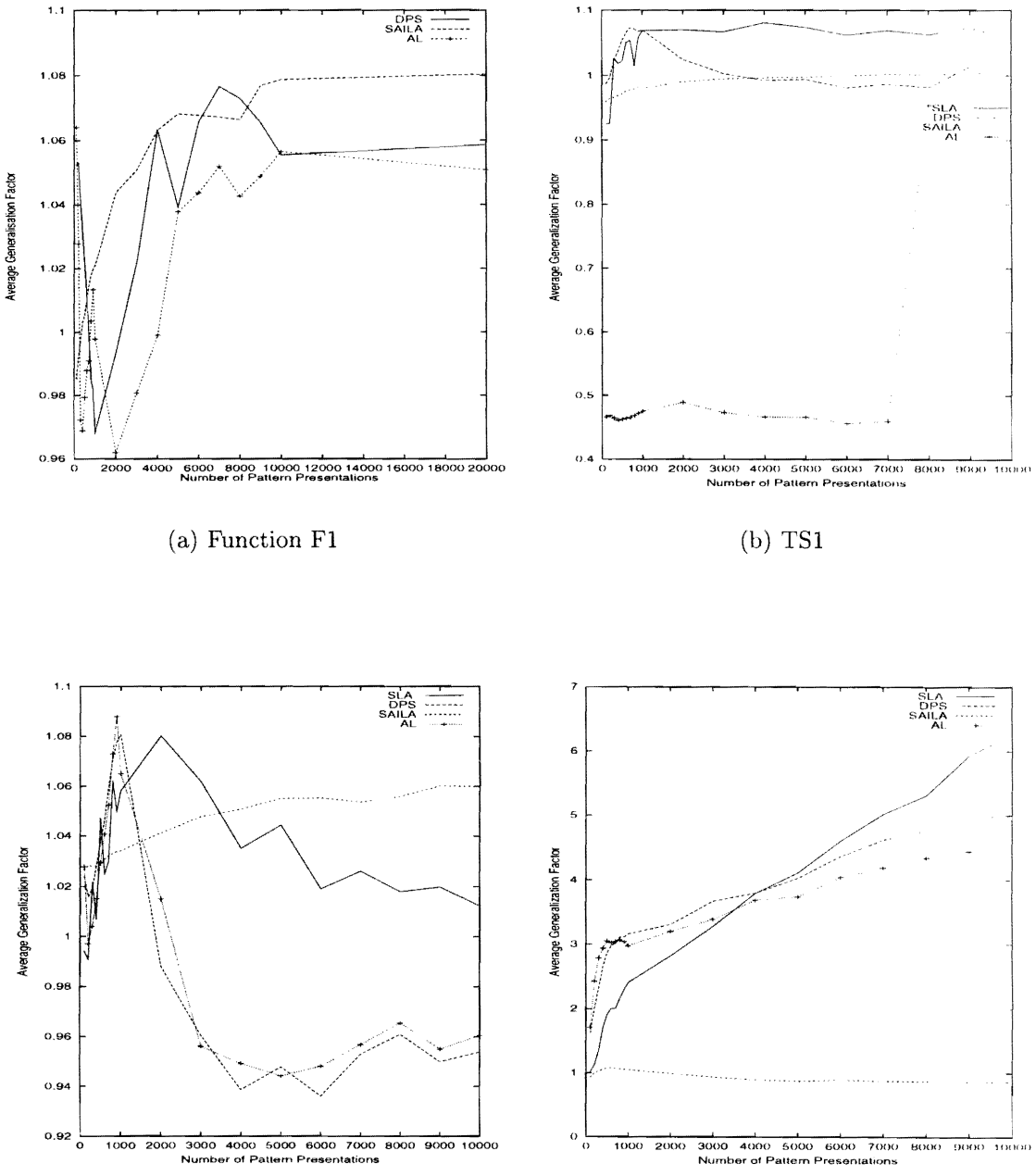
**Overfitting effects**

The average generalization factor $\rho$ for all the problems were computed over the 30 simulations. Figures 3.5 and 3.6 show the charts for the average generalization factors. The average generalization factors were plotted as function of pattern presentations. A pattern presentation represents one weight update.

TS3 was the only function for which all the algorithms except SAILA, overfitted. SAILA had an average generalization factor of less than one, while the other algorithms had high generalization factors. For the entire training period for TS4, AL had a generalization factor constantly larger than 1, indicating that AL overfitted TS4. For the other functions, the average generalization factor values fluctuated. The flunctuation is due to the overfitting of a training subset until new patterns are selected for training. When new patterns are selected, the overfitting of the training subset is reduced. The average generalization factor for all the algorithms (except TS3) were slightly over one, and indicating a mild case of overfitting.

**Computational costs**

The computational costs for AL, DPS, SLA and SAILA were computed using equation (3.9) for specified epochs. The costs are plotted as a function of epochs as illustrated in figures (3.9) and (3.10).



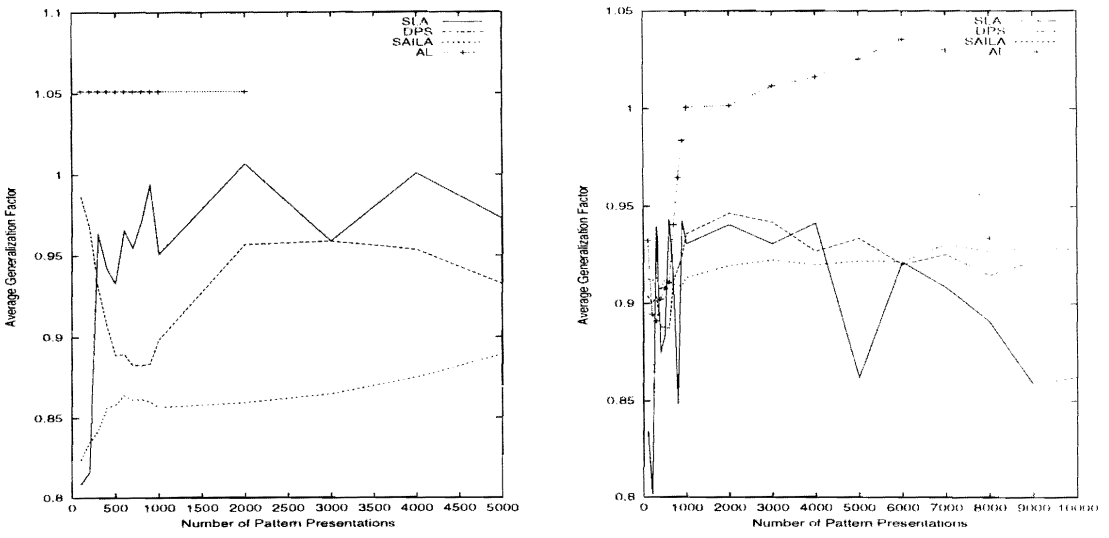

(a) Function F1

(b) TS1

(c) TS2

(d) TS3

Figure 3.5: Average generalization factor vs pattern presentations




(a) TS4 (b) TS5

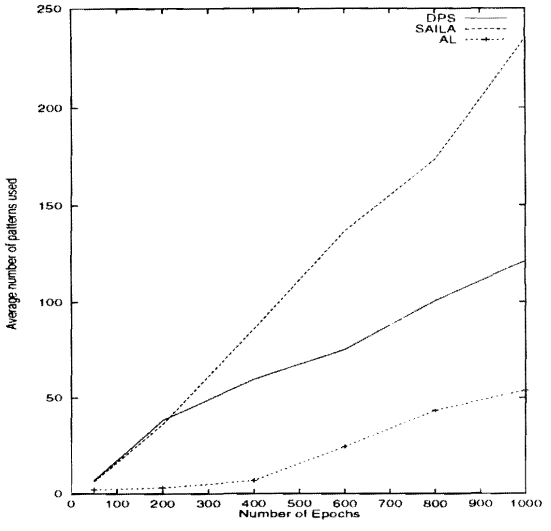
Figure 3.6: Average generalization factor vs pattern presentations

SAILA has the most expensive and AL has the least expensive subset selection criteria. However, AL performed badly (refer to tables 3.2 and 3.3). For all the approximation problems, DPS, AL and SAILA had increasing costs because they are incremental learning algorithms. More patterns were used as training progressed (refer to figures (3.7) and (3.8)).

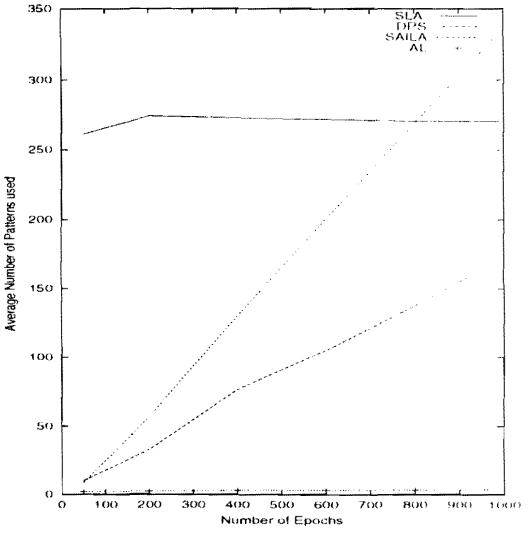
For F1 and TS2, AL had the smallest cost (see figure 3.9(a) and (b)). These small costs can be attributed to the cheap cost of the subset selection criterion as well as the fact that AL used the smallest number of patterns for training.

Despite the fact that AL has the cheapest subset selection criterion and a simple selection criterion, AL had the highest cost for TS3. This is because AL selected all the patterns in $D_C$ within a short training interval (by epoch 400). SLA initially had the highest cost (first 350 epochs). However, at epoch 1000, the cost was half of the other algorithms.

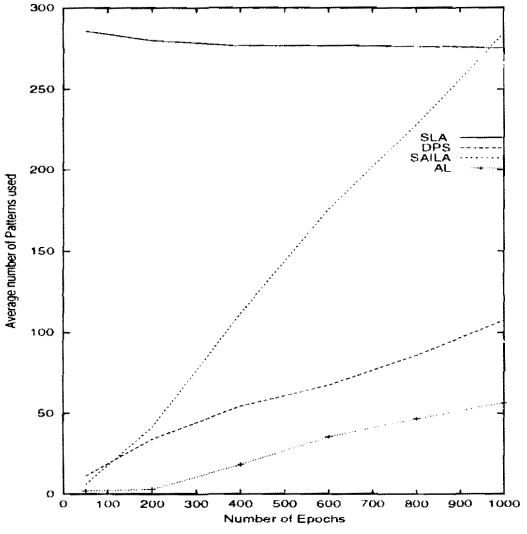
For all the functions approximated, SLA initially had a higher training cost than the other algorithms - almost four times the training cost of other algorithms, because



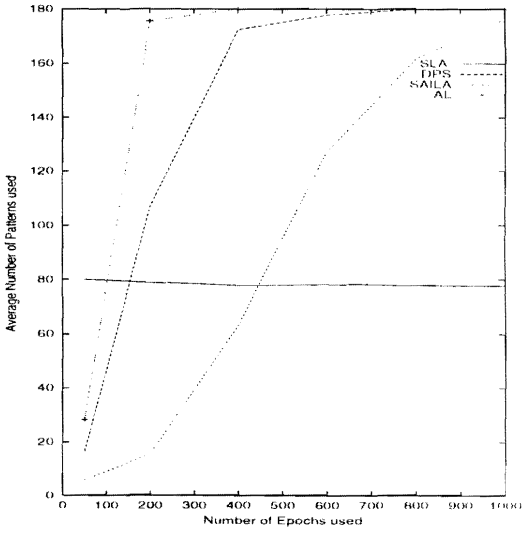

(a) Function F1




(b) TS1

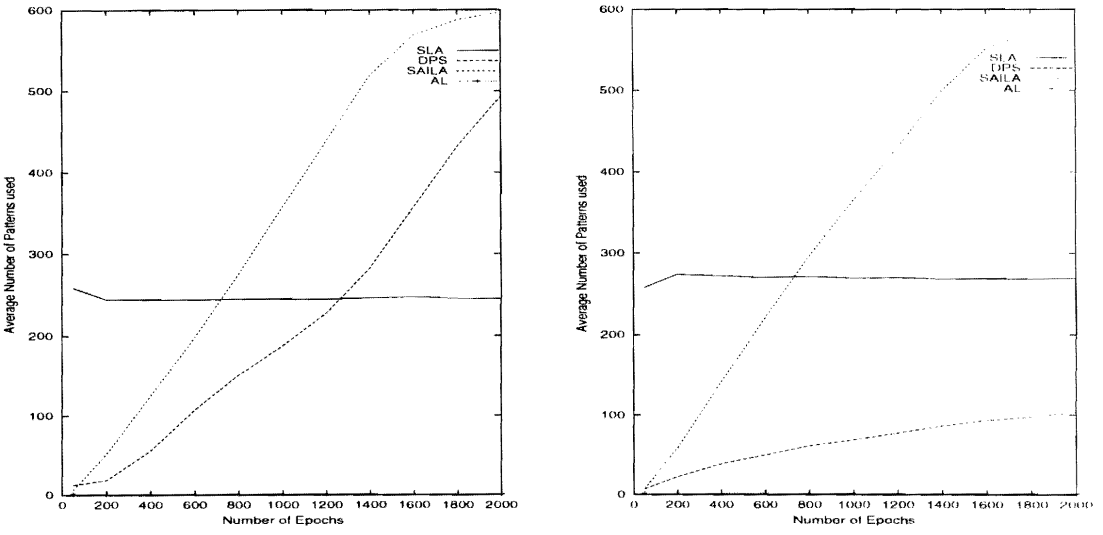



(c) TS2




(d) TS3

Figure 3.7: Average number of patterns used per epoch




(a) TS4 (b) TS5

Figure 3.8: Average number of patterns used per epoch

SLA is a selective approach (see figure (3.9) and (3.10)).

From tables (3.2) and (3.2), SLA used less number of patterns after 2000 epochs than the other algorithms, thus SLA was computationally less expensive.

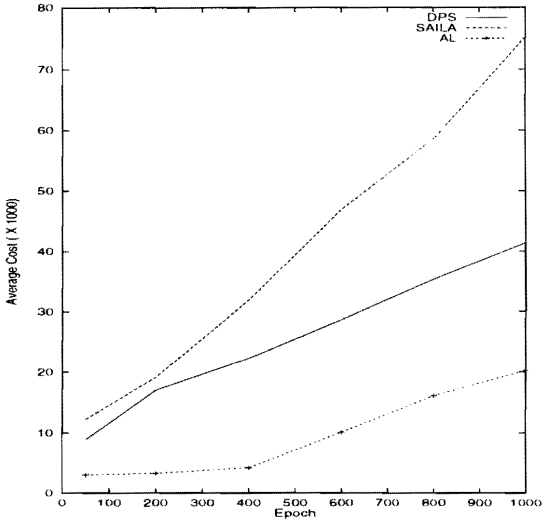
### Convergence

The convergence performance of the four active learning algorithms are compared in figures 3.11 and 3.12. These figures plot the percentage of simulations that reached specific generalization errors.

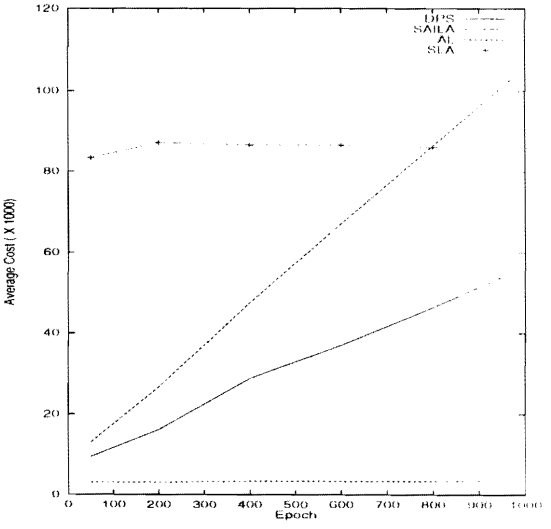
For F1, DPS had the best convergence, all the simulations converged to a very low error of 0.0004. AL also had a good convergence, more than half of the simulations converged to 0.0004 (refer to figure 3.11(a)).

None of AL's simulations converged to the specified error level for TS2. SLA and DPS achieved good convergence for TS2, as more than half of their simulations converged to a low error (refer to figure 3.11(b)).

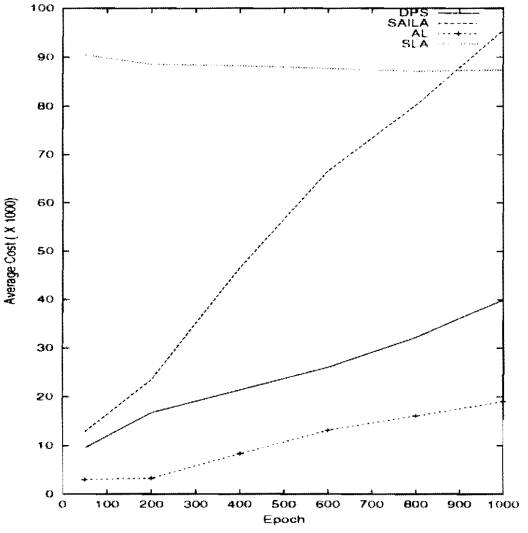
For TS2, DPS had the best generalization, most of all the simulations converged to a



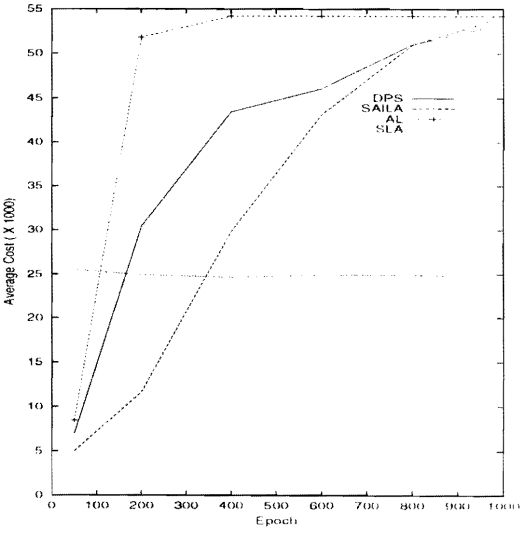

(a) Function F1




(b) TS1

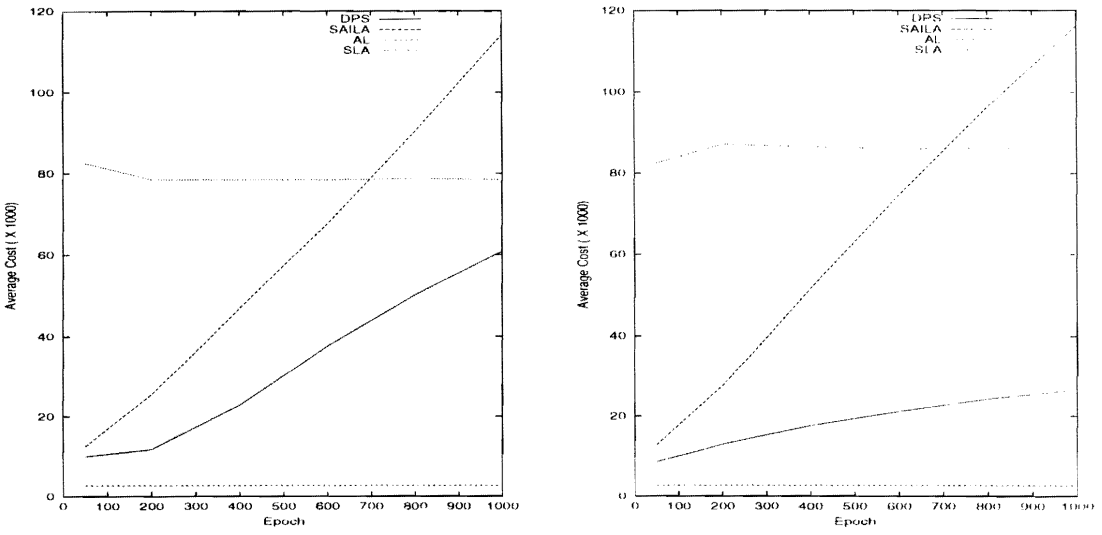



(c) TS2




(d) TS3

Figure 3.9: Average computational cost per epoch




(a) TS4 (b) TS5

Figure 3.10: Average computational cost per epoch

very low error (0.0002). SLA and AL also had good convergence (see figure 3.11(c)).

While the other algorithms had few converged simulations at 0.002, almost half of SAILA's simulations converged to this error (refer 3.11(d)).

AL had bad generalization for TS4 and TS5. None of AL's simulations converged to the specified error levels for TS4 while only a few converged for TS5.

SLA had the best generalization for TS4, with all the simulations converging to a low error of 0.004 (refer to figure 3.12(a)). SAILA had a good convergence with 40% of the simulations converging to this error of 0.004. Only a few of DPS's simulation and none of AL's simulation converged at this point.

SLA also had the best generalization for TS5. Almost all the simulations (74%) converged to a error level of 0.005 while only a few of the other algorithms simulations converged to this error level (see figure 3.12(b)).

SLA had the best convergence for data with outliers and noise. DPS had the best convergence for clean data, although SLA had good convergence for clean data. SAILA

had a good convergence for TS3. For all the sine functions (TS1, TS4 and TS5), AL had bad convergence, none of its simulations converged to the specified error levels. The errors specified for data with outliers and noise were larger than the errors specified for clean data. This is because the performance of all the algorithms were degraded in the presence of noise and outliers.

## 3.6 Conclusion

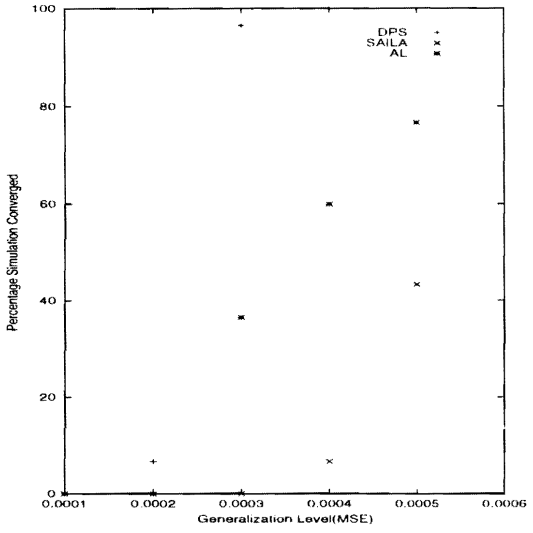
The objectives of the chapter were to present a new learning algorithm (SLA) and also to compare four active learning algorithms with respect to their accuracy, convergence and the complexity on both clean and noisy data as well as overfitting effects for the problems were also examined.

The results presented showed that AL was unstable, producing good results for the henon-map and F1 only. The bad training behavior can be attributed to the extremely small training set sizes used by AL, which is an indication of an inferior subset selection trigger.

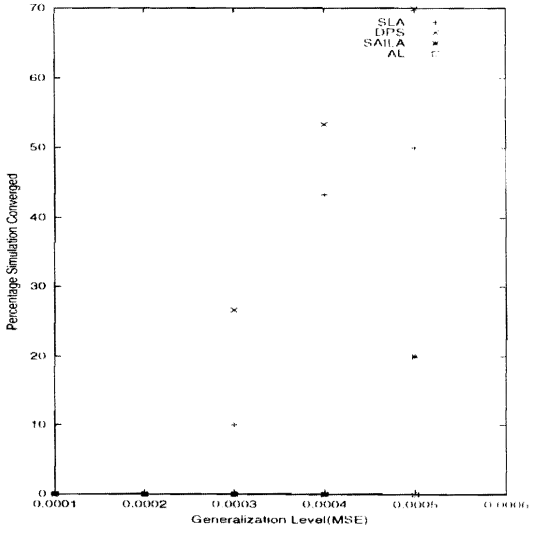
DPS and SLA performed very similar on the clean data, while SLA outperformed all the other algorithms on the noisy and outliers training data. The sensitivity analysis approach (SAILA) performed well under the occurrence of outliers and noisy time series, and very well for the complex function TS3. SAILA performed better than AL in TS1, TS4 and TS5, but worse than SLA and DPS. SAILA is computationally more expensive, requiring larger training subsets than the other algorithms.

As is expected, the performance of the error selection approaches degraded under the occurrence of outliers and noise. The degradation is due to the early selection of outliers, since outliers result in the largest prediction errors.

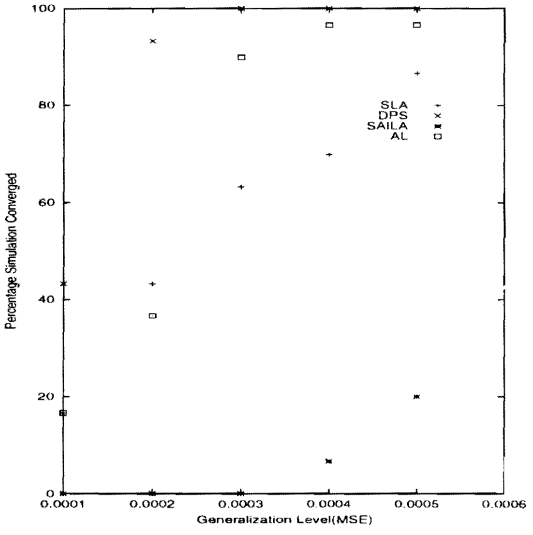
The comparison above showed that SLA had the best generalization performance, and lowest complexity. The selective learning approach (SLA) produced better accuracy



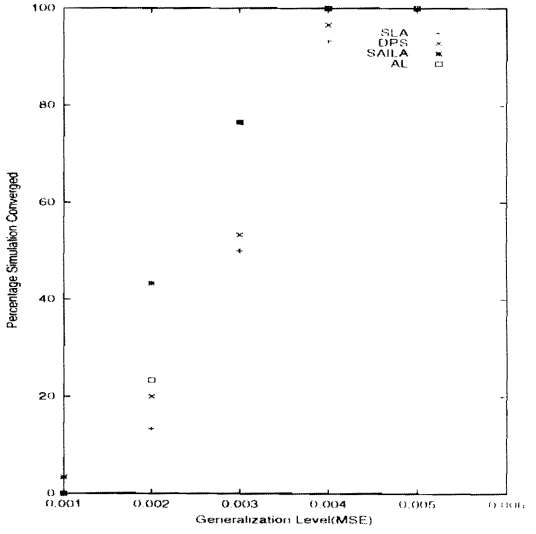

(a) F1




(b) TS1

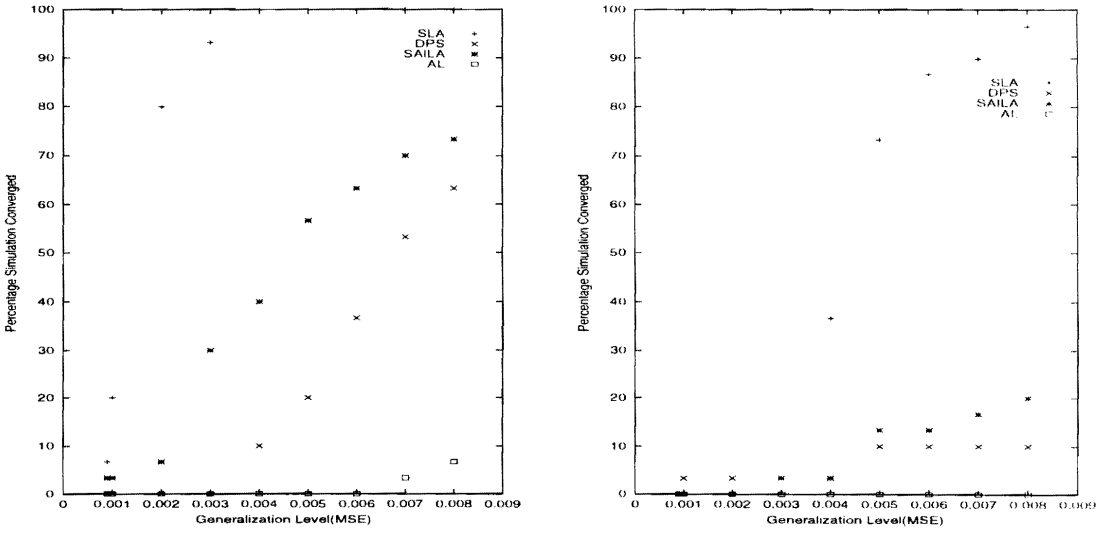



(c) TS2




(d) TS3

Figure 3.11: Percentage simulations that converged




(a) TS4 (b) TS5

Figure 3.12: Percentage simulations that converged

than the other approaches, and showed to be more robust in the occurrence of outliers and noise.

# Chapter 4

# CONCLUSION

The first objective of this thesis was to propose a new active learning algorithm using changes in output as selection criterion. The second objective of this thesis was to compare the performance of selected active learning algorithms on both clean and noisy data. The algorithms were compared in relation to their accuracy, generalization, convergence and computational cost. Four active learning algorithms were selected for this comparison. Two of these algorithms namely, DPS and AL use the error in prediction as selection criterion. That is, patterns are selected based on the error of the patterns. Two algorithms, SLA and SAILA, which use perturbations in output as selection criterion were also selected for comparison. Patterns which influence the change in output values most are selected for training, using the output selection criterion.

Röbel's algorithm (DPS) performed well with clean data (TS1, TS2 and F1), having a faster convergence and better generalization than the other algorithms. This performance can be attributed to the selection of patterns that contribute most to the error of the network. Training on such patterns took into account the current state of the network and thus brought the output closer to the target function. The performance of DPS degraded in the presence of outliers and noise in the training data, consequently the generalization ability deteriorated.

AL performed badly in all the functions except F1 and TS2. For the sine functions (TS1, TS2, TS3), AL selected very few patterns for training which resulted in very large errors.

SAILA achieved a considerable good accuracy for the function with outliers and the complex function TS3. This is because SAILA used perturbations in output values. i.e. changes caused to the output by the input as its selection criterion, thus avoiding the selection of outliers patterns. SAILA was however, slow in learning most of the functions, even for those functions for which a low generalization error has been obtained. SAILA's slow learning *can* be attributed to the fact that SAILA only chose patterns at the highest peak of the derivative and then tries to fit the network from this point. A suggestion to improve training using the SAILA algorithm is to select patterns at the lowest peak also, i.e at the turning point where derivative is zero in addition to the patterns selected at the highest peak. The network will then fit the problems being solved at the two extreme points of the derivatives simultaneously. A faster convergence and a lower training time maybe achieved compared to the current SAILA algorithm.

SLA achieved a good accuracy for both clean data and data with outliers and noise. SLA used much less patterns (i.e. a low computational cost) than all the algorithms for all the problems. Thus, SLA showed to be more robust in the occurrence of outliers and noise. SLA has demonstrated good and comparable results both in the training and generalization ability of the network.

Active learning algorithms using perturbations in output performed better with functions with noise and outliers while, algorithms using change in error as selection criteria performed better with clean data. A good subset selection criterion is very important in any active learning algorithm. AL had a poor subset selection criterion, selecting too few patterns for training. Even though, DPS and AL used the same selection criterion, DPS outperformed AL in all the problems. This better performance of DPS is a result of a better subset selection criterion used by Röbel. A network

trained with too few information will generalize badly, as in the case of AL.

For problems with clean data, DPS is preferred, though DPS used more patterns for training than SLA. However SLA is preferred with problems with outliers and noise. SLA is also preferred for clean data because of low computational costs.

## 4.1 Future of Active Learning in Neural Networks

Active learning has been shown to demonstrate a better performance than the conventional backpropagation algorithm. Various research have compared these two learning paradigms and have published their results [Zhang 1994, Röbel 1994c, Engelbrecht *et al* 1998]. Because of the demonstrated performance of active learning, research to improve on active learning must be continuously carried out.

A suggestion to further improve on active learning is to first cluster input patterns. A clustering algorithm can be used to group similar patterns into clusters, where similarity is measured as the Euclidean distance between input vectors. At each subsetselection interval, the most informative pattern is selected from each of the clusters. The clustering active learning approach can potentially reduce computational complexity and improve accuracy.

-

# Bibliography

[Adejumo *et al* 1999] A Adejumo, AP Engelbrecht, *A comparative Study of Neural Network Active Learning Algorithms*, In: VB Bajic and D Sha (eds). Development and Practice of Artificial Intelligence Techniques, Proceedings of the International Conference on Artificial Intelligence, Durban, South Africa. pp 29-31, 1999.

[Annema 1995] AJ Annema, *Feed Forward Neural Networks: Vector Decomposition Analysis, Modelling and Analog Implementation*, Kluwer Academic Publishers, 1995.

[Beale *et al* 1990] R Beale, T Jackson, *Neural Computing: Introduction*. Adam Hilger, 1990.

[Becker *et al* 1988] S Becker, Y Le Cun, *Improving the Convergence of Back-Propagation Learning with Second Order Methods*, DS Touretzky, GE Hinton, TJ Sejnowski (eds), Proceedings of the 1988 Connectionist Summer School, Morgan Kaufmann:Los Angeles, 1988.

[Carling 1992] A Carling, *Introducing Neural Networks*, Sigma Press, 1992.

[Cohn 1994] DA Cohn, *Neural Network Exploration using Optimal Experiment Design*, AI Memo No 1491, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1994.

[Cohn *et al* 1996] DA Cohn, Z Ghahramani, MI Jordan, *Active Learning with Statistical Models*, Journal of Artificial Intelligence Research, Vol 4. 1996. pp

129-145.

[Darken *et al* 1992] C Darken, J Moody, *Towards Faster Stochastic Gradient Search*, J Moody, SJ Hanson, R Lippmann (eds), Advances in Neural Information Processing Systems, Vol 4, 1992, pp 1009-1016.

[Dayhoff 1990] JE Dayhoff, *Neural Network Architechures: An Introduction*, Van Nostrand Reinhold, 1990.

[Depenau *et al* 1994] J Depenau, M Møller, *Aspects of Generalization and Pruning*, World Congress on Neural Networks, Vol 3, 1994, pp 504-509.

[Desai *et al* 1996] R Desai, R Patil, *SALO: Combining Simulated Annealing and Local Optimization for Efficient Global Optimization*, Proceedings of the 9th Florida AI Research Symposium, 1996, pp 233-237.

[Durbin *et al* 1989] R Durbin, DE Rumelhart, *Product Units: A Computationally Powerful and Biologically Plausible Extension to Backpropagation Networks*, Neural Computation, Vol 1, 1989, pp 133-142.

[Engelbrecht *et al* 1995] AP Engelbrecht, I Cloete, J Geldenhuys, JM Zurada, *Automatic Scaling using Gamma Learning for Feedforward Neural Networks*, International Workshop on Artificial Neural Networks, Torremolinos, Spain, June 1995, in J Mira, F Sandoval (eds), "From Natural to Artificial Neural Computing," in the series Lecture Notes in Computer Science, Vol 930, pp 374-381.

[Engelbrecht *et al* 1996] AP Engelbrecht, I Cloete, *A Sensitivity Analysis Algorithm for Pruning Feedforward Neural Networks*, IEEE International Conference in Neural Networks, Washington, Vol 2, 1996, pp 1274-1277.

[Engelbrecht *et al* 1998] AP Engelbrecht, I Cloete, *Selective Learning using Sensitivity Analysis*, IEEE World Congress on Computational Intelligence, International Joint Conference on Neural Networks, Anchorage, Alaska, 1998, pp 1150-1155.

[Engelbrecht *et al* 1999a] AP Engelbrecht, I Cloete, *Incremental Learning using Sensitivity Analysis*, accepted for IEEE International Joint Conference on Neural Networks, Washington DC, USA, 1999.

[Engelbrecht *et al* 1999b] AP Engelbrecht, L Fletcher, I Cloete, *Variance Analysis of Sensitivity Information for Pruning Feedforward Neural Networks*, accepted for IEEE International Joint Conference on Neural Networks, Washington DC, USA, 1999.

[Engelbrecht *et al* 1999c] AP Engelbrecht, A Adejumo, *A New Selective Learning Algorithm for Time Series Approximation using Feedforward Neural Networks*, In: VB Bajic and D Sha (eds), Development and Practice of Artificial Intelligence Techniques, Proceedings of the International Conference on Artificial Intelligence, Durban, South Africa, pp 29-31, 1999.

[Fausett 1994] L Fausett, *Fundamentals of Neural Networks: Architechures, Algorithms and Applications*, Prentice Hall, 1994.

[Fletcher *et al* 1994] GP Fletcher, CJ Hinde, *Learning the Activation Function for the Neurons in Neural Networks*, International Conference on Artificial Neural Networks, Vol 1, 1994, pp 611-614.

[Fletcher *et al* 1998] L Fletcher, V Katkovnik, FE Steffens, AP Engelbrecht, *Optimizing the Number of Hidden Nodes of a Feedforward Artificial Neural Network*, IEEE World Congress on Computational Intelligence, International Joint Conference on Neural Networks, Anchorage, Alaska, 1998, pp 1608-1612.

[Freund *et al* 1997] Y Freund, H Seung, E Shamir, N Tishby, *Selective Sampling using the Query by Committee Algorithm*, Machine learning, Kluwer Academic Publishers, 1997, pp 1-38.

[Fu 1994] LiMin Fu, *Neural Networks in Computer Intelligience*, McGraw Hill, 1994.

[Fukumizu 1996] K Fukumizu, *Active learning in Multilayer Perceptrons*. D Touretzky, M Mozer, M Hasselm (eds), To appear Advances in Neural Information Processing Systems, Vol 8, MIT Press, 1996.

[Hara *et al* 1988] K Hara, K Nakayama, *Training Data Selection Method for Generalisation by MLNN*, Institute of Electronics, Information and Communication Engineers Transaction Fundamentals, Vol E81-A, No 3. 1988, pp 374-381.

[Hassibi *et al* 1994] B Hassibi, DG Stork, G Wolff, *Optimal Brain Surgeon: Extensions and Performance Comparisons*, JD Cowan, G Tesauro, J Alspector (eds), Advances in Neural Information Processing Systems, Vol 6. 1994, pp 263-270.

[Hassoun, 1995] MH Hassoun, *Fundamentals of Artificial Neural Network*. MIT Press, 1995.

[Haykins, 1994] S Haykins, *Neural Networks: A Comprehensive Foundation*. MacMillan Academic and Professional, 1994.

[Hecht-Nielsen 1989] R Hecht-Nielsen, *Neurocomputing*, Addison-Wesley, 1989.

[Hirose *et al* 1991] Y Hirose, K Yamashita, S Hijiya, *Back-Propagation Algorithm which Varies the Number of Hidden Units*, Neural Networks, Vol 4. 1991, pp 61-66.

[Hopfield 1982] J Hopfield, *Neural networks and physical systems with emergent collective computational abilities*, Proceedings of the National Academy of Sciences, 79:2554-2558. Reprinted in Anderson Rosenfeld [1988], pp 460-464.

[Hussain *et al* 1997] A Hussain, JJ Soraghan, TS Durbani, *A New Neural Network for Nonlinear Time-Series Modelling*, NeuroVest Journal, Jan 1997, pp 16-26.

[Jacobs 1989] RA Jacobs, *Increased Rates of Convergence through Learning Rate Adaptation*, Neural Networks, 4(1), 1989.

[Jasić *et al* 1995] T Jasić, HL Poh, *Analysis of Pruning in Backpropagation Networks for Artificial and Real World Mapping Problems*, International Workshop on Artificial Neural Networks, in J Mira, F Sandoval (eds), "From Natural to Artificial Neural Computing," in the series Lecture Notes in Computer Science, Vol 930, 1995, pp 239-245.

[Jutten *et al* 1995] C Jutten, P Chentouf, *A New Scheme for Incremental Learning*, Neural Processing Letters, 2(1), 1995, pp 1-4.

[Kamimura *et al* 1994] R Kamimura, S Nakanishi, *Weight Decay as a Process of Redundancy Reduction*, World Congress on Neural Networks, Vol 3, 1994, pp 486-491.

[Karayiannis *et al* 1993] NB Karayiannis, AN Venetsanopoulos, *Artificial Neural Networks: Learning Algorithms, Performance Evaluation and Application*, Kluwer Academic Publishers, 1993.

[Karnin 1990] ED Karnin, *A Simple Procedure for Pruning Back-Propagation Trained Neural Networks*, IEEE Transactions on Neural Networks, 1(2), 1990, pp 239-242.

[Kohara 1995] K Kohara, *Selective Presentation Learning for Forecasting by Neural Networks*, International Workshop on Applications of Neural Networks in Telecommunications, Stockolm, Sweden, 1995, pp 316-323.

[Krose *et al* 1993] B Krose, P Van Der Smagt, *An Introduction to Neural Networks*, University of Amsterdam, 1993.

[Le Cun 1990] Y Le Cun, JS Denker, SA Solla, *Optimal Brain Damage*, D Touretzky (ed), Advances in Neural Information Processing systems, Vol 2, 1990, pp 598-605.

[Leerink *et al* 1995] LR Leerink, C Lee Giles, BG Horne, MA Jabri, *Learning with Product Units*, Advances in Neural Information Processing Systems, Vol 7, 1995, pp 537-544.

[Lisborn 1992] PGJ Lisborn, *Neural Networks: Current Applications*, Chapman and Hall, 1992.

[Mackay 1992] D Mackay, *Information-based objective functions for active data selection*, Neural Computation Vol 4(4), 1992, pp 305-318.

[Maren *et al* 1990] A Maren, C Harsten, R Pap, *Handbook of Neural Computing Applications*, Academic Press, 1990.

[Masters 1993] T Masters, *Practical Neural Network Recipes in C++*, Academic Press, 1993.

[Møller 1993] MF Møller, *A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning*, Neural Networks, Vol 6, 1993, pp 525-533.

[Müller *et al* 1990] B Müller, J Reinhardt, *Neural Networks: An Introduction*, Springer Verlag, 1990.

[Pelillo *et al* 1993] M Pelillo, AM Fanelli, *A Method of Pruning Layered Feed-Forward Neural Networks*, International Workshop on Artificial Neural Networks, 1993, pp 278-283.

[Plaut *et al* 1986 ] D Plaut, S Nowlan, G Hinton, *Experiment on Learning by Backpropagation*, Technical Report CMU-CS-86-126, Carnegie-Melon University, June 1986.

[Plutowski *et al* 1993] M Plutowski, H White, *Selecting Concise Training Sets from Clean Data*, IEEE Transactions on Neural Networks, 4(2), March 1993, pp 305-318.

[Röbel 1994a] A Röbel, *Dynamic Pattern Selection: Effectively Training Backpropagation Neural Networks*, International Conference on Artificial Neural Networks, Vol 1, 1994, pp 643-646.

[Röbel 1994b] A Röbel, *Dynamic Pattern Selection for Faster Learning and Controlled Generalization of Neural Networks*, European Symposium on Artificial Neural Networks, 1994.

[Röbel 1994c] A Röbel, *The Dynamic Pattern Selection Algorithm: Effective Training and Controlled Generalization of Backpropagation Neural Networks*. Technical Report, Institute für Angewandte Informatik, Technische Universität Berlin, March 1994, pp 497-500.

[Rogas 1996] R Rogas, *Neural Networks: A Systematic Introduction*, Springer Press, 1996.

[Rumelhart *et al* 1986] DE Rumelhart, GE Hinton, RJ Williams, *Learning Internal Representation by Error Propagation*, DE Rumelhart, JL McClelland and the PDP Research Group (eds), Parallel Distributed Processing, Vol 1, MIT Press, Cambridge, Mass, 1986.

[Sarle 1995] WS Sarle, *Stopped Training and Other Remedies for Overfitting*, Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics, 1995, pp 352-360.

[Schittenkopf *et al* 1997] C Schittenkopf, G Deco, W Brauer, *Two Strategies to Avoid Overfitting in Feedforward Neural Networks*, Neural Networks, 10(30), 1997, pp 505-516.

[Seung *et al* 1992] HS Seung, M Opper, H Sompolinsky, *Query by Committee*, Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory, 1992, pp 287-299.

[Sietsma *et al* 1988] J Sietsma, RJF Dow, *Neural Net Pruning - Why and How*, IEEE International conference, on Neural Networks, Vol 4, 1988.

[Simpson 1990] PK Simpson, *Artificial Neural Systems: Foundations, Paradigms, Applications and Implementation*, Pergamon Press, 1990.

[Sung *et al* 1996] KK Sung, P Niyogi, *A Formulation for Active Learning with Applications to Object Detection*, AI Memo No 1438, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1996.

[Towell *et al* 1993] GG Towell, JW Shavlik, *Extracting Refined Rules from Knowledge-Based Neural Networks*, Machine Learning, Vol 13, 1993, pp 71-101.

[Wasserman 1989] PD Wasserman, *Neural Computing: Theory and Practice*, Van Nostrand Reinhold, 1989.

[Weir 1990] MK Weir, *A Method for Self-Determination of Adaptive Learning Rates in Back Propagation*, Neural Networks, 1990, pp 371-379.

[Weigend *et al* 1991] AS Weigend, DE Rumelhart, BA Huberman, *Generalization by Weight Elimination with Application to Forecasting*, R Lippman, J Moody, DS Touretzky (eds), Advances in Neural Information Processing Systems, Vol 3, 1991, pp 872-882.

[Wessels *et al* 1992] LFA Wesseles, E Barnard, *Avoiding False Local Minima by Proper Initialization of Connections*, IEEE Transactions on Neural Networks, 3(6), 1992, pp 899-905.

[Yu *et al* 1997] X-H Yu, G-A Chen, *Efficient Backpropagation Learning using Optimal Learning Rate and Momentum*, Neural Networks, 10(3), 1997, pp 517-527.

[Zhang 1994] B-T Zhang, *Accelerated Learning by Active Example Selection*, International Journal of Neural Systems, 5(1), March 1994, pp 67-75.

[Zurada 1992a] J Zurada, *Lambda Learning Rule for Feedforward Neural Networks*, Proceedings of the IEEE International Conference in Neural Networks, San Francisco, 1992.

[Zurada 1992b] J Zurada, *Introduction to Artificial Neural Systems*, West Publishing Company, 1992.

[Zurada *et al* 1997] JM Zurada, A Malinowski, S Usui, *Perturbation Method for Deleting Redundant Inputs of Perceptron Networks*, Neurocomputing, Vol. 14, 1997, pp 177-193.

# Appendix A

# Symbols and Notations

The notation and symbols used in this thesis assume a three layer neural network (NN) architechure with one input layer, one hidden layer, and one output layer. This appendix summarizes the symbols used throughout this thesis with reference to the three layer architecture. The symbols are listed alphabetically with their interpretation.

| *Symbols* | *Interpretation* |
|---|---|
| $\alpha$ | momentum term |
| $\eta$ | learning rate |
| $\kappa$ | Zhang's notation for a specified performance level |
| $\rho$ | Röbel's generalization factor |
| $\tau$ | Zhang's notation for allowable error tolerance per connection |
| $\Phi$ | Engelbrecht's notation for pattern informativeness |
| $\Psi$ | used in this thesis as the next-time-change in output |
| $\Upsilon$ | used as the next-time-change in target for Kohara's algorithm |

| *Symbols* | *Interpretation* |
|---|---|
| $C_{al}$ | cost of the subsetselection criteria for Zhang's accelerated learning (AL) |
| $C_{dps}$ | cost of the subsetselection criteria for Röbel's dynamic pattern selection (DPS) |
| $C_{fc}$ | cost of training a NN on a training set |
| $C_{fsl}$ | cost of fixed set learning |
| $C_s$ | cost of selecting patterns |
| $C_{sai}$ | cost of the subsetselection criteria for SAILA |
| $C_{sla}$ | cost of the subselection criteria for selective learning algorithm (SLA) |
| $C_{S_{al}}$ | cost of selecting a pattern into $D_T$ for Zhang's accelerated learning |
| $C_{S_{dps}}$ | cost of selecting a pattern into $D_T$ for Röbel's dynamic pattern selection |
| $C_{S_{sai}}$ | cost of selecting a pattern into $D_T$ for SAILA |
| $C_{AL}$ | cost of training a network using Zhang's algorithm |
| $C_{DPS}$ | cost of training a network using Röbel's algorithm |
| $C_{SA}$ | cost of training a network using Engelbrecht's algorithm |
| $C_{SL}$ | cost of training a network using selective learning algorithm |
| $C_v$ | cost of updating a weight between the input and hidden layers |
| $C_V$ | cost of updating all weights between the input and hidden layers |
| $C_w$ | cost of updating a weight between the hidden and output layers |
| $C_W$ | cost of updating all weights between the hidden and output layers |
| $D_C$ | set of candidate training patterns |
| $D_G$ | test set or the generalization set |
| $D_T$ | actual training set |
| $D_V$ | validation set |
| $I$ | total number of input units |
| $J$ | total number of hidden units |
| $K$ | total number of output units |
| $N_V$ | total number of weights between input and hidden layers of a network |
| $N_W$ | total number of weights between hidden and output layers of a network |
| $p$ | a single pattern |
| $P_C$ | number of patterns in the candidate set $D_C$ |
| $P_G$ | number of patterns in the generalization set $D_G$ |
| $P_s$ | number of patterns in a subset $D_s$ |
| $P_T$ | number of patterns in the training set $D_T$ |
| $P_V$ | number of patterns in the validation set $D_V$ |
| $o_k$ | $k$-th output unit |
| $o_k^{(p)}$ | activation of output unit $o_k$ for pattern $p$ |
| $v_{ji}$ | weight between $j$-th hidden unit and $i$-th input unit |
| $w_{kj}$ | weight between $k$-th output unit and $j$-th hidden unit |
| $y_j$ | $j$-th hidden unit |
| $y_j^{(p)}$ | activation of hidden unit $y_j$ for pattern $p$ |
| $z_i$ | $i$-th input unit |

# Appendix B

# Definitions

This appendix summarizes definitions of key terms used in this thesis. The terms are defined in alphabetical order.

**Active learning:** Active learning is any form of learning in which the learning algorithm has some deterministic control during training over what part of the input space it receives information (page 10).

**Accelerated Learning:** Accelerated learning (AL) is Zhang's algorithm for active learning. Patterns with the highest prediction error are selected for training. New patterns are selected as soon as the error on the training subset is reduced to a specified performance level. AL is an incremental approach to active learning (page 61).

**Bias:** A bias is a unit or neuron added to the input and hidden layers with a constant activation value of $-1$. The purpose of adding a bias unit is to offset the origin of the logistic activation function (page 36).

**Dynamic Pattern Selection:** Dynamic pattern selection (DPS) is Röbel's algorithm for active learning. The most informative patterns are the patterns with the maximum prediction error and are selected for training. New set of patterns are selected for training as soon as the network overfits the current training

subset. DPS is an example of incremental learning (page 59).

**Epoch:** An epoch is one learning pass through the training set. One learning pass involves the presentation of training patterns, the calculation of the activation of each neuron and modification of the weights (page 36).

**Gradient Descent Optimization:** In Gradient descent optimization (GD) the minimum of the objective function is searched in the negative gradient of the objective function. In NNs, the objective function is the error function which is a function of the weights of the NN (page 27).

**Incremental Learning:** Incremental learning is a form of active learning, where a subset of the training patterns that satisfies a selection criterion is selected for training. Patterns are however selected and removed from the candidate set $D_C$ into the actual training set $D_T$. The effect of incremental learning is that the training set $D_T$ is grown while the candidate set $D_C$ is pruned during training (page 46).

**Momentum:** Momentum is a term added to weight adjustments to help avoid oscillations in weight changes during training. This term is proportional to the magnitude of previous weight changes (page 37).

**Mean Squared Error:** In the context of neural networks, the mean squared error (MSE) is defined as the mean of the squared sum of the error between target values $t_k^{(p)}$ and the actual NN output values $o_k^{(p)}$:

$$MSE = \frac{\sum_{p=1}^{P} \sum_{k=1}^{K} (t_k^{(p)} - o_k^{(p)})^2}{2PK}$$

where $P$ is the total number of patterns and $K$ is the number of output units (page 11).

**Pattern Presentations:** A pattern presentation is a single pattern presented to the network for training. Pattern presentations are the total number of patterns presented so far to the network at a particular epoch (page 76).

**Sensitivity Analysis for Incremental Learning:** Sensitivity analysis for incremental learning (SAILA) is Engelbrecht's algorithm for active learning. Patterns are selected for training using the changes in output caused by perturbations in input parameters (page 56).

**Selective Learning:** Selective learning is an active learning algorithm, where a subset of the training patterns that satisfies a selection criterion is selected for training. Unlike incremental learning, the candidate set remain fixed while the size of the actual training set varies from time to time (page 46).

**Selective Learning Algorithm:** The selective learning algorithm (SLA) is a new active learning algorithm proposed in this thesis, which uses information on the next-time-changes to select patterns for training (page 54).

**Subset selection Criterion:** Subset selection criteria are criteria tested to determine whether a NN should select additional patterns into the current training subset $D_T$ (page 52).

**Sum Squared Error:** SSE is the sum of squared errors, defined as

$$SSE = \frac{1}{2}\sum_{p=1}^{P}\sum_{k=1}^{K}(t_k^{(p)} - o_k^{(p)})^2$$

(page 31).