University of Fort Hare

Together in Excellence

The generalization ability of Artificial Neural Networks in forecasting TCP/IP network traffic trends.

By

Vusumuzi Moyo

Supervisor

Dr K. Sibanda

A thesis submitted in fulfilment of the requirements of the
Degree of Master of Science in Computer Science
at the University of Fort Hare.

August 2014

Declaration

I hereby declare that *"The generalization ability of Artificial Neural Networks in forecasting TCP/IP network traffic trends"* is my original work and it has not been submitted before for any degree or examination at any other University. All sources I have used, consulted or quoted are duly indicated and acknowledged herein.

August 2014

Publications

Part of the research work presented in this thesis has been published or has been submitted for publication or review in the following papers:

V. Moyo and K. Sibanda, *The generalization ability of Artificial Neural Networks in forecasting TCP/IP network traffic trends*. In Proceedings of the 2013 Southern African Telecommunications Networks and Applications Conference, Stellenbosch wine Spier estates, Western Cape, South Africa.

V. Moyo and K. Sibanda, *On the optimal Artificial Neural Network architecture for forecasting TCP/IP network traffic trends*. In Proceedings of the 2014 Southern African Telecommunications Networks and Applications Conference, Port Elizabeth, Eastern Cape, South Africa.

V. Moyo and K. Sibanda, *On the optimal learning rate size for the generalization ability of Artificial Neural Networks in forecasting TCP/IP traffic trends* [Accepted to be published in International Journal of Computer Applications].

V. Moyo and K. Sibanda, *The learning rate and momentum: How much does each matter in forecasting TCP/IP network traffic trends?* [Accepted to be published in International Journal of Computer Science Applications].

V. Moyo and K. Sibanda, *The effects of the training set size on the generalization ability of Artificial Neural Networks in forecasting TCP/IP network traffic trends*: [Submitted to be reviewed in International Journal of Computers Communications and Control].

V. Moyo and K. Sibanda, *The generalization ability and convergence of Artificial Neural Networks in forecasting TCP/IP network traffic trends: Which Backpropagation algorithm is most effective*: [Submitted to be reviewed in South African Computer Journal].

Dedication

To my parents, with utmost gratitude

Acknowledgements

First and foremost I would like to thank the Almighty, whose invisible hand continues to lead me through life. I owe special gratitude to the research supervisor Dr K. Sibanda for insight, guidance, support and constructive criticism during the course of this research. He continually and realistically conveyed a spirit of adventure about research, and an excitement regarding teaching. Without his guidance and persistent help, this thesis would not have been possible. A special thank you to the staff of the Department of Computer Science at UFH, to the H.O.D Mr. S. Scott for his continual jokes and support, Professor M. Thinyane for his advice and concern throughout this research, Ms N. Moorosi for her valuable suggestions in conducting the research, Mr S. Ngwenya and many others I did not mention. To the Masters' class of 2014, thank you guys for making it worthwhile, life would never have been the same without you guys. To Caro, Munya and Pride, thank you for the priceless moments.

I would like to thank my parents Mr. and Mrs. K. Moyo for being there and believing in me in all things. I am eternally grateful to you mum and dad. I thank my entire family, to my sister Mamo, thank you for your continued support. Gracious, words can never express how I love you guys.

This work is based on the research undertaken within the Telkom CoE in ICTD supported in part by Telkom SA, Tellabs, Saab Grintek Technologies, Easttel, Khula Holdings, THRIP and National Research Foundation of South Africa (UID: 90434). The opinions, findings and conclusions or recommendations expressed here are those of the authors and none of the above sponsors accepts any liability whatsoever in this regard.

Abstract

Artificial Neural Networks (ANNs) have been used in many fields for a variety of applications, and proved to be reliable. They have proved to be one of the most powerful tools in the domain of forecasting and analysis of various time series. The forecasting of TCP/IP network traffic is an important issue receiving growing attention from the computer networks. By improving upon this task, efficient network traffic engineering and anomaly detection tools can be created, resulting in economic gains from better resource management. The use of ANNs requires some critical decisions on the part of the user. These decisions, which are mainly concerned with the determinations of the components of the network structure and the parameters defined for the learning algorithm, can significantly affect the ability of the ANN to generalize, i.e. to have the outputs of the ANN approximate target values given inputs that are not in the training set. This has an impact on the quality of forecasts produced by the ANN. Although there are some discussions in the literature regarding the issues that affect network generalization ability, there is no standard method or approach that is universally accepted to determine the optimum values of these parameters for a particular problem. This research examined the impact a selection of key design features has on the generalization ability of ANNs. We examined how the size and composition of the network architecture, the size of the training samples, the choice of learning algorithm, the training schedule and the size of the learning rate both individually and collectively affect the ability of an ANN to learn the training data and to generalize well to novel data. To investigate this matter, we empirically conducted several experiments in forecasting a real world TCP/IP network traffic time series and the network performance validated using an independent test set. MATLAB version 7.4.0.287's Neural Network toolbox version 5.0.2 (R2007a) was used for our experiments. The results are found to be promising in terms of ease of design and use of ANNs. Our results indicate that in contrast to Occam's razor principle for a single hidden layer an increase in number of hidden neurons produces a corresponding increase in generalization ability of ANNs, however larger networks do not always improve the generalization ability of ANNs even though an increase in number of hidden neurons results in a concomitant rise in network generalization. Also, contradicting commonly accepted guidelines, networks trained with a larger representation of the data, exhibit better generalization than networks trained on smaller representations, even though the larger networks have a significantly

greater capacity. Furthermore, the results obtained indicate that the learning rate, momentum, training schedule and choice of learning algorithm have as much a significant effect on ANN generalization ability. A number of conclusions were drawn from the results and later used to generate a comprehensive set of guidelines that will facilitate the process of design and use of ANNs in TCP/IP network traffic forecasting. The main contribution of this research lies in the identification of optimal strategies for the use of ANNs in forecasting TCP/IP network traffic trends. Although the information obtained from the tests carried out in this research is specific to the problem considered, it provides users of back-propagation networks with a valuable guide on the behaviour of networks under a wide range of operating conditions. It is important to note that the guidelines accrued from this research are of an assistive and not necessarily restrictive nature to potential ANN modellers.

Table of Contents

Table of Figures

Table of Tables

List of Acronyms

ANN	Artificial Neural Networks
ARIMA	Auto -Regressive Model
BP	Backpropagation
CSV	Comma separated values
Dos	Denial of service
FFNN	Feedforward Neural Network
GUI	Graphical User Interface
IP	Internet Protocol
ISP	Internet Service Provider
Lr	Learning rate
Mc	Momentum
MLP	Multilayer perceptron
MRTG	Multi Router Grapher
MSE	Mean Square Error
NN	Neural Network
NNE	Neural Network Ensemble
Qos	Quality of service
RMSE	Root Mean Square Error
SLP	Single layer Perceptron

STWNN	Sliding Time Window Neural Network
TCP	Transmission Control Protocol
TENET	South African Tertiary Institution Network
UFH	University of Fort Hare
VC	Vapnik Chervonenkis

Chapter 1: Introduction

This chapter serves as an introduction to the research and to the chapters that follow. A comprehensive background into the research is given. Thereafter, the problem statement is provided, followed by research goals that guide investigation of the research. This is followed by a brief section that presents and clarifies the scope of research detailing methodology used for investigating the research problem highlighted. The final section is devoted to the description of the thesis structure and conclusion of the chapter.

1.1 Research Background

Time Series Forecasting (TSF) deals with the prediction of a chronologically ordered variable (Cortez et al. 2006). As more applications vital to today's society migrate to TCP/IP networks it is essential to develop techniques that better understand and predict the behaviour of these systems. TCP/IP network traffic forecasting is vital for the day- to- day running of large-/medium-scale organizations (Cortez et al. 2010). By improving upon this task network providers can optimize resources (e.g. adaptive congestion control and proactive network management), allowing an overall better Quality of Service (QoS). TCP/IP forecasting also helps to detect anomalies in the network. Security attacks like Denial-of-Service (DoS) or even an irregular amount of SPAM can be detected by comparing the real traffic with the values predicted by forecasting algorithms, resulting in economic gains from better resource management. Two of the most recent discoveries of the statistics of internet traffic over the last ten years are that internet traffic exhibits self-similarity and non-linear behaviour (Chabaa 2010). This nature of network traffic makes highly accurate prediction difficult. Several TSF methods have been proposed, such as Autoregressive models (ARIMA), Box Jenkins model, Holt Winters and Artificial Neural Networks (ANN). Literature has indicated that unlike other methods, ANNs can approximate almost any function regardless of its degree of nonlinearity and without prior knowledge of its functional form (Klevecka 2009; Zhang & Kline 2007). This positions them as good candidates for modelling non linear and self- similar time series such as TCP/IP network traffic. Currently, there are several journals devoted to ANN research and a considerable number of textbooks published illustrating their applications in a diversity of fields.

One of the earliest studies discussing the use of ANNs techniques for forecasting TCP/IP network traffic is reported by Jiang and Apavasillou (2004), following which a number of researchers have adopted different ANN models for forecasting and modelling network traffic (Wang et al. 2008; Allacorn -Quiono & Arria 2006). One typical example is in Cortez et al. (2010), where a Neural Network Ensemble (NNE) for the prediction of TCP/IP traffic using TSF point of view was presented. The NNE approach was compared with TSF methods (e.g. Holt -Winter and ARIMA) and the NNE was found to compete favourably with the other methods. Although the use of ANNs is widespread and well documented, and despite their promising prospects, ANNs suffer from several deficiencies; mostly related to problems in the selection of optimal network architecture and learning parameters for good generalization ability (Uwahuro 2008; Halenár & Libošvárová 2012). These deficiencies restrict their general acceptability, particularly in the forecasting community (Baum & Haussler 1989).

Work done by Zhani and Elbiaze (2009) in comparing the performance of ANNs to other methods in forecasting TCP/IP traffic showed that whilst ANNs were the best at making short-term forecasts, they fared poorly in long- term forecasts and were computationally expensive to set up contrasted to the other methods. Equally, some authors such as Conway (1998) have argued that although ANNs greatly outperform multiple linear regression models in performing prediction tasks, however, due to their inability to generalize well, they cannot compete with algorithmic or other sophisticated models. Consequently, several techniques have been studied, such as fuzzification of input vector, regularization, result feedback and Optimal Brain surgeon (Canuto 2001). Although these methods are reported to improve the generalization ability of ANNs to some extent, the problem of ANNs' generalization ability is universally still not solved or not completely solved (Walleed et al. 2009; Baum & Haussler 1989; Neeharika 1996). Furthermore, issues that have an effect on the generalization ability of ANNs, such as the appropriate choice of features for input to the network, the training methodology, and the best network topology have all been identified, but completely satisfactory solutions have not been offered for any of these problems (Richards 1991; Maier & Dandy 1998; Kavzoglu 1999).

While most researchers find that ANNs can outperform linear methods under a variety of situations, the conclusions are not consistent. In any case most of these studies have been conducted on synthetic datasets (e.g. Glass-Mackey time series), making the solutions thereof difficult to apply to real world problems. This clearly makes the subject of the generalization ability of ANNs still an open research area. The main aim of this research is to investigate the effects of the size of the network architecture, size of the training set, size of the learning rate, the choice of learning algorithm and the schedule of training on the generalization ability of ANNs in forecasting of a real world TC/IP network traffic time series.

Reports on the relationship between network architecture and ANN generalization ability indicate that whilst the input and output layers are important for learning, it is the specification of the size of the hidden layer(s) that is critical for the ANN's capabilities of learning and generalization (Wijayasekara et al. 2011; Teixeira & Fernandes 2012). Despite the fact that the effects of employing too small or too large network architectures are known in a general sense, the exact nature of the impact of the size of the hidden layers on ANN generalization has not been fully investigated.

Although several heuristics have been proposed none is universally accepted for estimating the optimal number of hidden layers and/or neurons for a particular problem. Consideration of previous research results also indicates that the size of the training set is crucial insofar as the generalization ability of ANNs is concerned (Chakraborty & Pal 1997). Whilst it is important for the ANN to have a sufficient architecture, this same ANN cannot be expected to generalize well from those examples which it does not memorize unless they contain sufficient information regarding the similarities which must be extracted in order for a good generalization to occur (Richards 1991). An appropriate number of training samples has to be measured to ensure a correct presentation of the forecasting problem to the ANN. As to the exact size of the training set that is required for an ANN to generalize well to new examples, we are uncertain. Literature indicates that whilst some authors are in favour of smaller training sets, others prefer the complete opposite, and to further add to the quagmire, some suggest it is largely dependent on the network architecture and the level of complexity introduced by the problem (Chakraborty & Pal 1997; Lange & Manner 1994; Karelson et al. 2006). To date, it is clear that the training set is crucial for any successful ANN implementation, however the exact number of training samples

required to ensure good generalization ability of ANNs for a particular problem remains largely subjective.

Research also indicates that the size of the learning rate, training schedule and choice of learning algorithm are frequently unemphasized factors in the pantheon of factors that affect ANN generalization ability (Wilson & Martinez 2001; Mahmoud et al. 2007; Kiran et al. 2010). To define an optimum size of the learning rate, to select the best learning algorithm and to choose an appropriate training schedule are among the biggest difficulties encountered in the design of ANNs. Foody et al. (1996), describes these three parameters as being factors that the analyst may have control over and that strongly influence ANN performance, especially in terms of speed and accuracy. He also states that a major limitation associated with ANNs is that they are semantically poor. On the other hand, Richards (1991) describes them as being factors which have a profound influence on the generalization ability of ANNs. Although individual studies have been conducted and some form of heuristics provided for the selection of these parameters, none has had the theoretical rigor of revealing optimal or at least near-optimal solutions. In any case most of these studies have been conducted on synthetic datasets e.g. (Glass-Mackey time series) making the solutions thereof difficult to apply to real world problems. In fact, until a number of experiments have been done, it is unknown which parameter values of the aforementioned factors will provide optimum solutions. Hence new users of ANNs, usually blindly employ trial-and-error strategies to determine the optimal values for these parameters without any substantive guidelines. This results in the addition of more time to the already slow process of training an ANN. To date, we are not aware of any research study that has attempted to consider the influence these and other factors have on the generalization ability of ANNs in the context of the forecasting of TCP/IP network traffic trends. This has created a huge gap in as far as the successful design and implementation of ANNs in this domain is concerned. Motivated by the this fact, the present study is conducted in order to gain some insights into understanding the behaviour of ANNs under different operating conditions, thus facilitating the steps of ANN design and parameter setting. It is hoped that this research will help to disprove to some extent the statement that ANNs are semantically poor.

1.2 Statement of problem

Generalization in ANNs is the acquisition of connection strengths which reflect similarities extracted from the representations used in training the network (Kavzoglu 1999). Several factors have been reported in literature as having an influence on the ability of ANNs to generalize (Chakraborty & Pal 1997; Lange & Manner 1994). These are: size of network architecture, size of the training set, size of learning rate, the learning algorithm and the training schedule. As yet we do not understand the exact conditions an ANN requires in order to do a good job of extracting similarities from the representations used in training the network. Research has shown that in forecasting TCP/IP network traffic trends, given an appropriate architecture, sufficient training data, reasonably sized learning rate, learning algorithm, and an appropriate training schedule, an ANN can generally find a function that will memorize the training set, however what still remains unknown is if that function will permit the ANN to generalize well to data it has never seen before (Richards 1991; Fearn 2004).

1.3 Research Goals

ANNs represent a field that is rooted in many disciplines. They have been used in many fields to solve a number of problems by virtue of an important property*: the ability to learn from input data with or without a teacher*. For this research it is essential that a detailed overview of the many facets of ANN learning is undertaken so as to better comprehend the structure and functionality of ANNs, in particular the Multilayer Perceptron (MLPs) which has found significance in many real world applications of ANNs.

Therefore, the first objective of the research is:

- To determine the state of the art in TSF by reviewing ANNs

Currently there exist many methods of TSF. Each of these has their relative strengths and weaknesses. It is essential that a thorough investigation is done on the state of the art in TSF highlighting the current methods of TSF and the relative limitations and advantages of ANNs over conventional statistical methods in TSF.

Therefore, the second objective of the research is:

- To determine the current methods of TSF.

The third objective of the research is:

- To investigate how the size of the network architecture, size of training set and the size of the learning rate both individually and collectively affect the generalization ability of ANNs.

The fourth objective of the research is:

- To determine how much influence the learning algorithm and the training schedule both individually and collectively have on the generalization ability of ANNs.

1.4 Contribution of Study

In situations where ANN modelling approaches are adopted, modellers are faced with a number of alternatives at each stage of the model development process. It is certainly one of the user's expectations that ANNs should be easy to use by means of understanding each design step taken. These include the choice of learning algorithm, the sizes of the training set, architecture, and learning rate, as well as the appropriate training schedule. It is essential that the options available to ANN modellers at each of these steps are discussed and the issues that should be considered highlighted. Some guidelines with sound foundations are required, describing the possible effects of various configurations of ANN geometry and parameter settings. Therefore the major contributions of this research are:

- To provide sound and practical guidelines on the efficient design of ANNs to new and novice users. Through empirically conducting experiments we have explored the effects of certain ANN design parameters on the generalization ability of ANNs. The results of all the experiments are discussed in detail and used to form the basis of further work, directed at continuing to refine the processes involved during the design of ANN models. It is common knowledge that a generalized ANN avoids overfitting of the data and yields models with better predictability (Staufer & Fischer 1997). Several general guidelines have been established, and these are presented. They will provide the new or inexperienced user with clear instructions on the design of ANNs, particularly in the context of forecasting TCP/IP network traffic.
- To contribute theoretically to the school of knowledge on TSF in particular that of TCP/IP network traffic engineering.

1.5 Scope of the research

To achieve the research goals mentioned in section 1.3, the following methodology is employed:

- A technical study of ANNs, particularly in the domain of TSF is essential, accompanied by wide research on case studies pertaining to ANN generalization ability. The literature and case studies are reviewed to provide background knowledge necessary for understanding, developing and investigating the process of designing efficient ANNs, particularly in the domain of TSF.
- The main aim of the research is the investigation of the question: how do the size of the network architecture, size of the training set, size of the learning rate, learning algorithm choice and training schedule, both individually and collectively affect the ability of an ANN not just to learn the training data but to generalize well to previously unseen data? The case study under consideration is the forecasting of a real world TCP/IP network traffic time series. To answer this question, several simulation experiments are conducted whereby each of the aforementioned factors are investigated through varying certain ANN parameters in a controlled manner. Matlab programs or codes for conducting these experiments are written and Neural Network toolbox Version 7.4.2 (R2007a) is used for this task.
- Preparation of the data and pre-processing is done in steps. Firstly, Linear interpolation is done to fill in the missing values or observations, secondly identification of seasonal components is done to identify seasonality and outliers in the data, thirdly the ANN inputs and targets are computed, fourthly normalization of the data is done to scale the ANN input and targets within the range of the activation function, and finally partitioning of the time series into train and test set.
- The primary evaluation criterion is the Root Mean Square Error (RMSE).To assess the generalization ability of the ANNs, the RMSE on the independent test set is computed. The RMSE is given by the overall formula:

$$\boldsymbol{RMSE} = \sqrt{\left(\frac{\mathbf{1}}{\boldsymbol{PK}}\sqrt{\left(\sum_{\boldsymbol{p=1}}^{\boldsymbol{P}}\sum_{\boldsymbol{k=1}}^{\boldsymbol{K}}(\boldsymbol{t_{k,p}} - \boldsymbol{o_{k,p}})^2\right)}\right)} \qquad \textbf{(1.1)}$$

where P is the total number of training patterns or observations, K is the number of training outputs, $t_{k,p}$ is the target output for the k^{th} value of the p^{th} pattern, $o_{k,p}$ is the

actual output generated by the k^{th} output for the p^{th} pattern. An ANN with good generalization ability should consistently show low RMSE values on the test set.

1.6 Overview of the Dissertation

The thesis consists of six chapters, including this introductory chapter in order to achieve the aims and objectives defined above.

Chapter 1: introduces the research by presenting background to the study and does a brief literature review of related work in relevant areas. It provides the underlying reasons for the research, presents the research problem and the approach used to reach the corresponding solutions.

Chapter 2: gives a detailed literature review of ANNs, in particular the Multilayer Perceptrons (MLPs). The chapter gives the reader insight into the theoretical background of MLPs and their application in the domain of TSF. The chapter also reviews past work done on TSF using ANNs and presents theory on some important and relevant aspects of ANN learning. A special section is devoted to the problems encountered in the use of ANNs in TSF. These problems will be the main concern of later chapters dealing with the analysis of the effects of certain ANN design parameters.

Chapter 3: concentrates on the problems encountered in the design and use of ANNs, which this research has identified. Whilst the design of ANNs is mainly related to the size of the network architecture, the effective use of ANNs is associated with the selection of appropriate rates for the learning parameters such as learning rate, learning algorithm, training set size and training schedule. After providing an extensive literature review on these issues, discussions and comparisons are carried out of the heuristics (or rules of thumb) recommended by different researchers on the selection of these parameters, pointing out certain flaws and strengths of each case study.

Chapter 4: presents on the methodology and data techniques utilised in this research.

Chapter 5: describes the results and presents the evaluations of the research. The goals of the experiments are stated, described and the results are analysed. It discusses the performance of the ANNs with experiment configurations done under varying training parameter scenarios.

Chapter 6: presents the conclusions drawn from the research. It specifically comprises a number of guidelines for the use and design of ANNs extracted from the results produced and experience

gained during the research. Finally, this chapter gives suggestions for further research that could be conducted on this specific research topic.

In addition to the chapters outlined above, **Appendix A** gives some of the programs (MFiles) used in the research. Snapshot of the Multi Router Traffic Grapher (MRTG) software which monitors traffic load on network links from various institutions in South Africa are available in **Appendix B.** A plot of the TCP/IP network traffic time series is available in **Appendix C**. Available in **Appendix D** is a snapshot of one of the ANN models during one of the many training sessions in the Neural Network toolbox Version 5.0.2 (R2007a). A snapshot of a Performance plot curve of one of the ANNs is also available in **appendix E**. Graphs, depicting results of the different sizes of the sliding time windows are shown in **Appendix F.**

1.7 Conclusion

This chapter introduced the research study by presenting background to the study, stating research goals and addressing the methodology followed in accomplishing the research work. The chapter highlighted the contribution of the research and concluded with the structure of the thesis. An extensive literature review of ANNs and their relative applications in the domain of Time series forecasting follows in the next chapter.

Chapter 2: Artificial Neural Networks and applications in Time Series Forecasting of TCP/IP network traffic.

This chapter provides an extensive review of ANNs. It focuses on a particular structure of ANNs, the MLP which is the most popular and widely-used paradigm in many applications of ANNs, including forecasting. This chapter provides a review of the literature concerning the areas of MLP analysis, learning and the Backpropagation (BP) algorithm as well as the Logistic sigmoid and Linear activation functions. It also discusses an important theorem in ANNs: the Universal approximation property. The chapter concludes by briefly highlighting important aspects of the application of ANNs in forecasting TCP/IP network traffic. The intent is to give the general background of each topic area, and to highlight any aspects of these subjects applicable to this research.

2.1 General overview of Artificial Neural Networks

2.1.1 Biological inspiration

The puzzle of the human brain is as old as human history itself. Making machines that mimic the behaviour of the human brain has been a dream for centuries. The study of ANNs has been inspired in part by the observation that biological learning systems are built of very complex webs of interconnected neurons (Shavlik et al. 1991). In a rough analogy, ANNs are built out of a densely interconnected set of simple neurons, where each neuron takes a number of real-valued inputs (possibly the outputs of other neurons) and produces a single real-valued output, which may become input to other neurons (Batbayar 2008). To develop a feel for this analogy, let us consider a few facts from neurobiology. The neuron is the basic unit of the brain that processes and transmits information. A neuron has the following structural components: synapses, dendrites, cell body, and axon (Lange et al. 1997). The biological neuron receives inputs along the dendrites and adds them up (‘summates’ them). The dendrites are connected to the outputs of other neurons via special junctions known as synapses. The synapses alter the effectiveness with which the signal is transmitted. The signals are accumulated in the cell body and when their quantity reaches a certain level (usually referred to as the threshold), the new signal is released and carried to other neurons through the axons (Balestrassi et al. 2009).

One motivation for ANN systems is to capture this kind of highly parallel computation based on distributed representations. While ANNs are loosely motivated by biological neural systems, there are many complexities to biological neural systems that are not modelled by ANNs, and many features of ANNs are known to be inconsistent with biological systems (Lange et al. 1997).

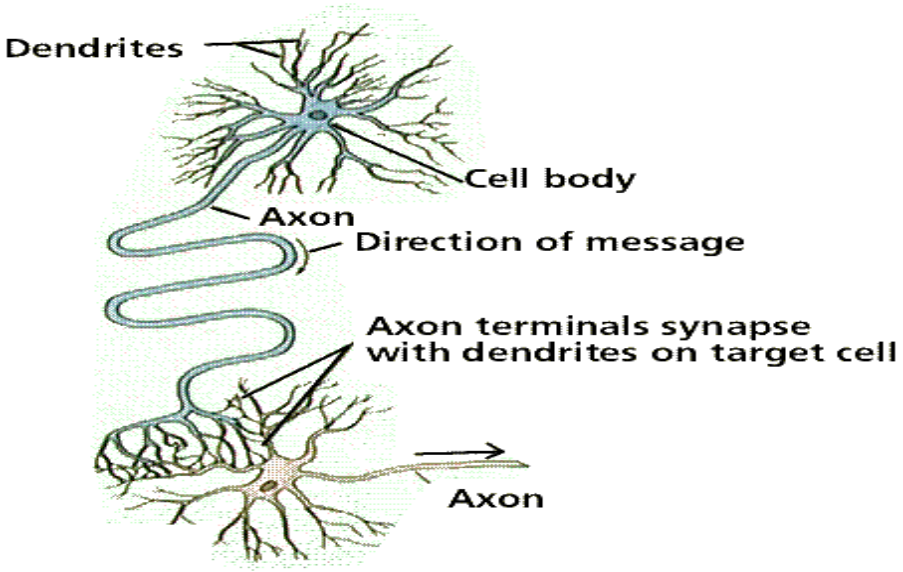


Figure 2.1: The Biological neuron (Control 2010).

2.1.2 What are ANNs?

ANNs have attracted increasing attention from researchers in many fields during the last decade. There have been several attempts to mimic the human brain. These date back to the early 1930s, 1940s and 1950s by Alan Turing, Warren McCullough, Walter Pitts, Donald Hebb and James von Neumann. The first ANN was produced in 1943 by Warren McCullogh and Walter Pitts, following which a lot of research has been devoted to ANNs (Kiran et al. 2010). ANNs go by many names, such as Neural Networks, Parallel distributed processing models, Connectivist or Connectionism models, Adaptive systems, Self-organizing systems, Neurocomputing, or Neuromorphic systems (Barabas et al. 2011). Each name may mean something slightly different; in this work we use the term Artificial Neural Network, or simply ANN.

A number of attempts have been made by various authors to define an ANN. Zhang (2001) defines ANNs as "*physical systems which can acquire, store and utilize experimental knowledge*". Hartono and Hashimoto (2007) define ANNs as "*a massively parallel distributed processor that has a natural propensity for storing experimental knowledge and making it available for use*". Haykin (1999) defines neural computing as "*the study of adaptable nodes which, through a process of learning from task examples, store experimental knowledge and make it available for use*".

One can conclude from these definitions that an ANN

- consists of several simple processing elements called neurons;
- is well suited for parallel computations, with each neuron operating independently of other neurons;
- contains a high degree of interconnections between neurons;
- contains links between neurons, each with a weight (scalar value) associated with it;
- has adaptable weights that can be modified during training;
- can be trained to do a specific task, i.e. produce desired output for given input.

2.1.2.1 A model of an Artificial Neuron

A neuron is an information processing element that is fundamental to the operation of an ANN. A neuron consists of three basic elements (Haykin 1999):

- A set of synapses or connecting links, each with its own strength or weight. Specifically, a signal x_j at the input of synapse j connected to neuron k is multiplied by the synaptic weight w_{kj}. The first subscript refers to the neuron in question and the second subscript refers to the input end of the synapse to which the weight refers.
- An accumulator Σ that computes the weighted sum of the signals.
- An activation function for limiting the amplitude of the output of a neuron. The activation function is also referred to in literature by Haykin (1999) as a squashing function in that it squashes (limits) the permissible amplitude range of the output signal to some finite value. Typically, the normalized amplitude range of the output of a neuron is written as the closed unit interval [0, 1] or alternatively [-1, 1].

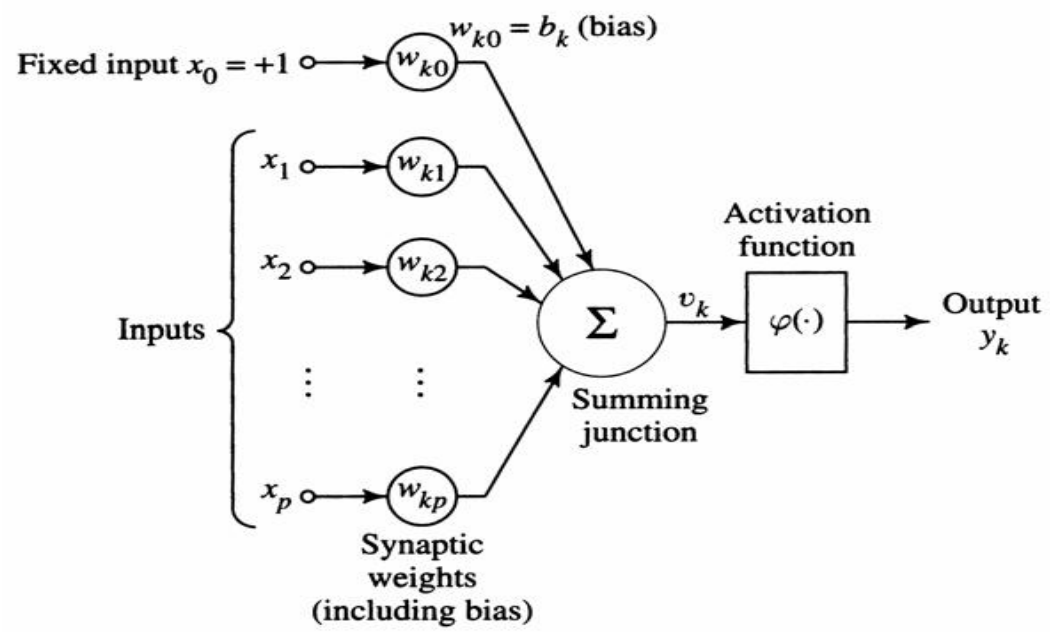


Figure 2.2: An Artificial Neuron (Svozil et al. 1997).

Figure 2.2 shows the basic structure of an artificial neuron. The inputs are denoted by $x_1, x_2, x_3 \ldots x_p$ and weights are denoted by $w_{ko}, w_{k1}, w_{k2} \ldots w_{kp}$. The inputs are sent through the connections to the accumulator and the connections carry the weights which are applied to the inputs. These weights are applied to the inputs x_i by multiplying the input by its corresponding weight w_{ki}. The products are then added, and the information sent to the accumulator is of the form:

$$\boldsymbol{w_k, x = \sum_{i=1}^{p} w_{ki} x_i} \qquad \textbf{(2.1)}$$

The accumulated sum of the inputs is compared to a value called the threshold, such that if the sum is greater than the threshold, then the output is 1, and if less, it is 0. The threshold adds a charge known as the bias (b_k) to the sum of the signals. A bias is added to the hidden and output neurons only. The output produced is then given as:

$$\boldsymbol{y_k = f(\sum_{i=1}^{p} w_{ki} x_i + b_k)} \qquad \textbf{(2.2)}$$

where f is the activation function and y_k is the output signal.

2.2 Classes of ANN applications

ANNs have been successfully applied to a number of problems. We review briefly the different classes of ANN applications that can be found.

- Clustering: The objective of clustering is to group similar patterns into groups, or clusters. This is mainly achieved by the Euclidean distance between patterns (Kvale 2010). Clustering is done by the ANN through inspection of the data for regularities (Uwahuro 2008). It has been applied to document classification to enhance information retrieval (Kvale 2010).
- Control: There have been a number of successful applications to control systems. Applications of ANNs to this field range from process control, robotics and industrial manufacturing to aerospace applications and automobile control (Moustafa et al. 2011). The basic objective of control is to provide the appropriate input signal to a given physical process to yield its desired response. ANNs for control were developed by Wei et al. (2004) and Werbos (1989).
- Optimization: The objective of ANNs in optimization applications is to optimize certain cost functions (Haykin 1999). ANNs have been successfully applied to optimization problems such as job-shop scheduling. Problems that are simpler but which belong to the same group of optimization tasks include scheduling classrooms to classes, hospital patients to beds, etc. (Zurada 1994).
- Association or Pattern Recognition: In Association each training pattern is associated with an image stored in the ANN (Kvale 2010). Association can be subdivided into autoassociation and heteroassociation. In autoassociation an ANN is repeatedly presented with a set of patterns to be stored by the ANN. After training, a partial description of the original pattern is presented to the ANN; the task is then to retrieve the original pattern. In heteroassociation an arbitrary set of patterns is paired with another arbitrary set of patterns. After training, when a partial description of the original pattern of the first set is presented to the ANN, the task is to retrieve the pattern paired off with the original pattern (Hussain & Browse 1998). Applications include the 'Human Face Detection Network' of Rowley and Shumeet (1996) and the NETtalk ANN of Bartlett et al. (2002) that produced phonetic strings which specified pronunciation for English text.

- Approximation: Approximation requires an ANN to approximate a non-linear function or time-series given a set of patterns in the form of input and desired (target) output pairs (Svozil et al. 1997). Once trained, the ANN is used to calculate an output for patterns not used in training (i.e. the ANN interpolates). An example of an application of approximation is in TCP/IP network traffic forecasting, which is the case study for this research.

2.3 The strengths and weaknesses of ANNs

According to Gonzalez (2000) the power of ANN techniques rests in their unique advantages that may be listed as follows:

- High tolerance to noisy data.
- Ability to classify untrained patterns.
- Well-suited for continuous valued inputs and outputs.
- Successful on a wide array of real world data.
- Adaptive learning: Ability to adapt its weights to changing environments.

However they are prone to limitations such as:

- Long training time.
- Require a number of parameters typically best determined empirically, e.g. network topology or structure.
- Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights of hidden neurons in the network.
- ANNs are not very good at performing symbolic computations. They cannot be used effectively for rule- based reasoning and arithmetic operations.

2.4 The architectures of ANNs

ANNs can be divided into Recurrent Neural Networks (RNN) and Feedforward Neural Networks (FFNN) (Zhang 2001).

RNN architecture employs feedback connections in order to learn temporal characteristics of data presented for learning. The feedback connections allow the network to produce complex

time varying outputs in response to simple static input (EngelBretch et al. 1995). RNNs exhibit properties very similar to short term memory in human beings (Churchland & Sejnowski 2005). There are different types of RNNs, e.g. Jordan and Elman RNNs.

In FFNN architectures, data flows strictly from the input layer to the output layer. There is neither feedback connection in the whole network, nor a connection between neurons in a single layer (Zhou et al. 2006). A FFNN has no memory and the output is solely determined by the current input and weights values (Gers 2001). The output of each layer serves as input to the next layer. The absence of recurrences makes FFNNs much easier to train, but also less powerful (Zhou et al. 2006). A Feed-Forward structure makes things simpler because there is a simple flow of computation through the ANN (Gers 2001). This research concentrates on FFNNs, and studies network learning in FFNNs as well as problems associated with learning in FFNNs.

2.4.1 Feedforward Neural Networks

There are mainly two types of FFNNs, the Single Layer Perceptron (SLP) and the Multilayer Perceptrons (MLP) (Haykin 1999).

2.4.1.1 SLP

In a SLP, there are only input and output neurons (Sarle & Warren 1991).The input layer of neurons is not counted as a layer since no computation is performed in this layer. An example of a SLP network is a Perceptron Neural Network. These networks use the step functions as their threshold activation functions (Zurada 1994). SLP networks are the simplest kind of ANNs, and can only solve linear or mildly non-linear problems (Sarle & Warren 1991).

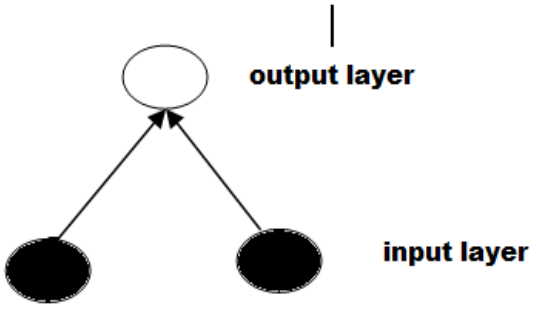


Figure 2.3: Single layer perceptron (Badri 2010)

Badri (2010) sets out some of the limitations of SLP networks. He notes that the output values of a SLP are limited as the ANN can take only one of two values (true or false). He further suggests that since SLPs can only classify linearly separable sets of vectors, a straight line or plane can be drawn to separate the input vectors into their correct categories; and provided the input vectors are linearly separable the perceptron will find the solution. This is further supported by Balestrassi et al. (2009), who state, "*When two classes are linearly separable, the perceptron training procedure will be guaranteed to converge to an answe*r, *however, if the classes are not linearly separable, the procedure will not converge and will keep on cycling through the data forever*." We further explore the concept of linear separability in subsequent sections.

2.4.1.2 MLP

A MLP contains at least one hidden layer situated between the input and output layers (Zhang & Fukushige 2002). MLPs are trained by the standard Backpropagation (BP) algorithm. They are supervised networks so they require a desired response to be trained (Abbas et al. 2013). MLPs contain a set of inputs connected to hidden layers whose function is to accept the input data samples and process them. There may be more than one hidden layer used to process the data to produce a network output similar to the desired output (Hagan & Menhaj 1994). In this type of ANN there are no links between neurons in the same layer, thus no computational dependencies between neurons in the same layer (Churchland & Sejnowski 2005). This allows the outputs of these neurons to be computed in parallel. The neurons in the input layer receive signals from the environment and distribute the signals to the next layer in the ANN (Churchland & Sejnowski 2005). MLPs can model functions of almost any arbitrary complexity, with the number of layers, and the number of neurons in each layer, determining the function complexity (Cortez et al. 2006). The major difference between the SLP and the MLP perceptron is in their activation functions. The former uses only the Threshold functions for activation, while the latter uses a combination of Sigmoid and/or Linear activation functions (Tsilo 2008). This research uses the MLP architecture in the implementation of the ANN models.

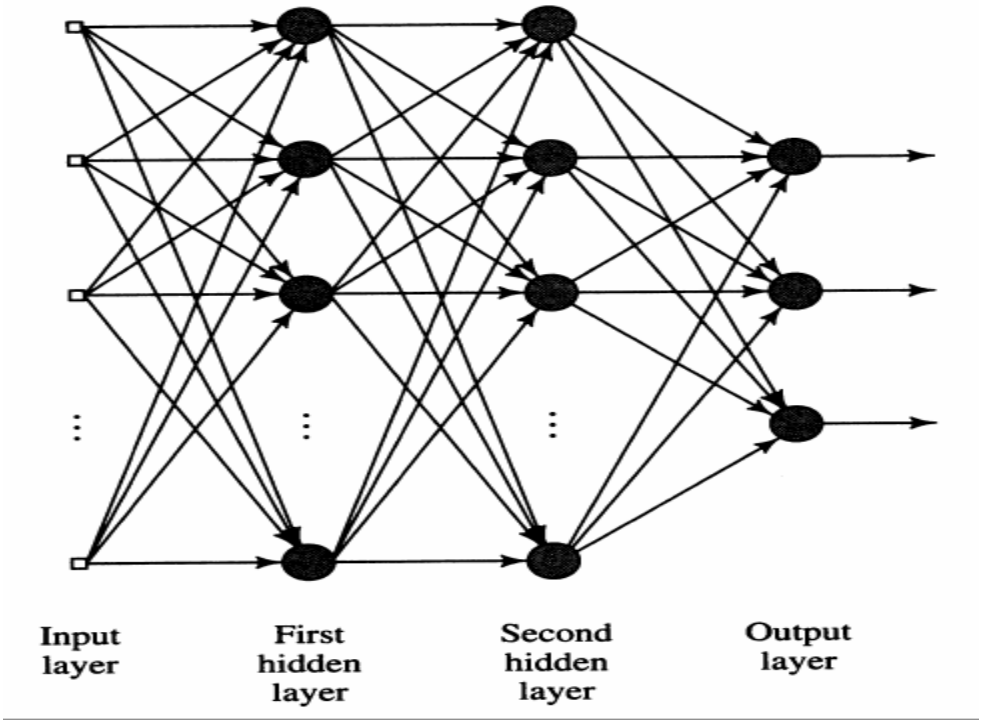


Figure 2.4: Multilayer perceptron(Cortez et al. 2006).

Tsilo (2008) states that MLPs have two properties which make them unique:

- Ability to classify linearly inseparable variables.
- Universal approximation property.

2.4.1.2.1 Ability to classify linearly inseparable problems

The SLP has wide applications for linearly separable problems. Linearly separable problems, are cases in which there exists a hyperplane that separates the inputs into classes. The SLP finds weights and biases to achieve such classification. An example of a linearly separable problem is illustrated in Figure 2.5.

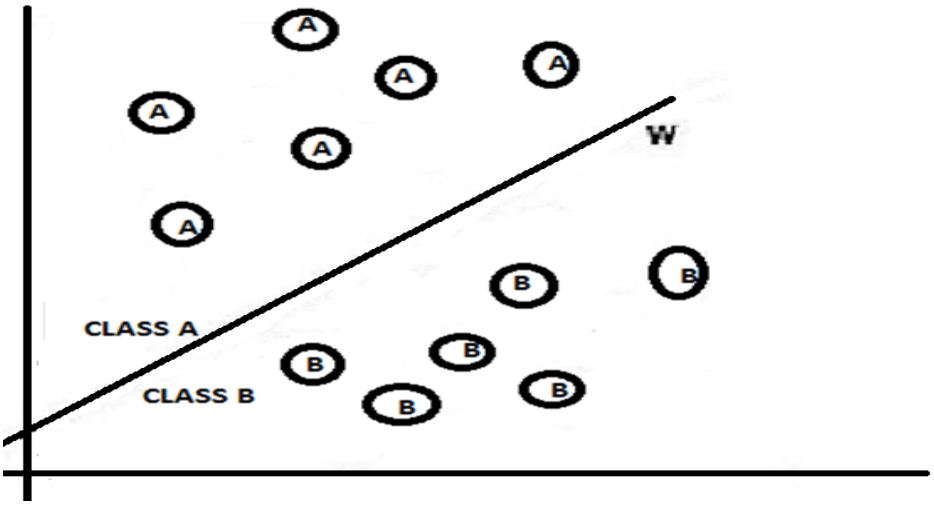


Figure 2.5: Classification to illustrate linear separability (Tsilo 2008).

Consider a situation in which there are inputs $x = (x_1, x_2)$ belonging to classes A and B as illustrated in Figure 2.5. Notice that these classes can be separated with a single line w. They are known as *linearly separable* patterns. The processing neuron of a SLP is able to categorize a set of patterns into two classes as the Threshold function defines their linear separability. An example, however of a problem that is not linearly separable is the XOR problem (Figure 2.6).

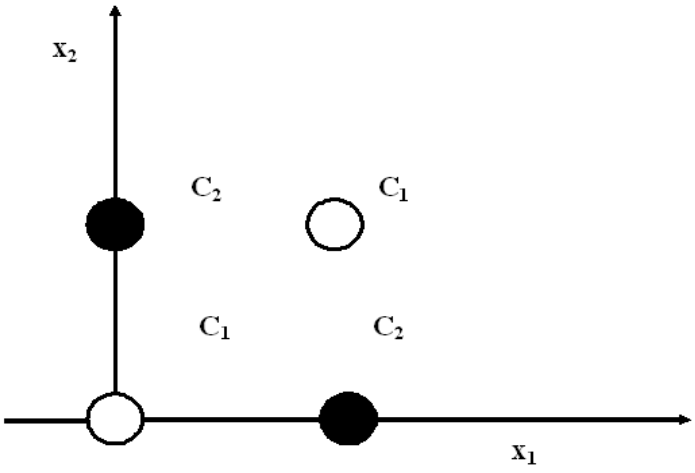


Figure 2.6: The XOR problem (Tsilo 2008).

Figure 2.6 illustrates the XOR problem. Suppose there are inputs $x = (x_1, x_2)$ belonging to two classes of $c = (c_1, c_2)$ as illustrated in Figure 2.6. In this example vectors *x = (0, 0)* and *x = (1,*

1) belong to c_1 while *x = (0, 1)* and *(0, 1)* belong to c_2. A straight line *w* in this case cannot be drawn to separate c_1 from c_2 points. This is an example of a linearly inseparable problem. SLP cannot correctly classify such a problem. This is largely due to their usage of the Threshold activation function. The use of this function removes information vital to the ANN. Tsilo (2008) states that a correct approach would be to modify the activation function to give details of the weights so that learning can take place. An example for such an activation function is the Logistic sigmoid function which is the activation function used in MLPs.

2.4.1.2.1.1 Activation functions

The firing of any neuron is governed by its activation function (Joy 2011). The activation functions used in ANNs are classified into Threshold, Linear and nonlinear activation functions. Loosely speaking, any differentiable function can qualify as an activation function (Chen & Miikkulainen 2001). In MLPs, hidden layers should have nonlinear activation functions while the output layer can have both Linear and nonlinear activation functions (Uwahuro 2008). We review the Logistic sigmoid and the Linear activation functions which have been widely used by researchers in the design of ANN models for forecasting.

2.4.1.2.1.1.1 The Logistic sigmoid activation function

Several activation functions are in use today, however the Logistic sigmoid activation function is the most favoured by a host of researchers (Kim 2005). This is largely attributed to its real valued and continuously differentiable properties (Cardoso et al. 2011). The Logistic sigmoid function produces outputs in the range of [0, 1] to emulate the brain. Within the brain, each nerve cell can respond to as many as 200,000 inputs, although 1,000 to 10,000 inputs are more typical. The inputs are either 'fire' or 'do not fire,' similar to a 0-1 variable (Nelson et al. 1999). The Logistic sigmoid function is defined by the following expression:

$$\boldsymbol{f(z) = \frac{1}{1+e^{-\alpha z}}} \qquad \textbf{(2.3)}$$

where *z* is the net input of the units in the ANN and α is the gradient parameter of the Logistic sigmoid activation function.

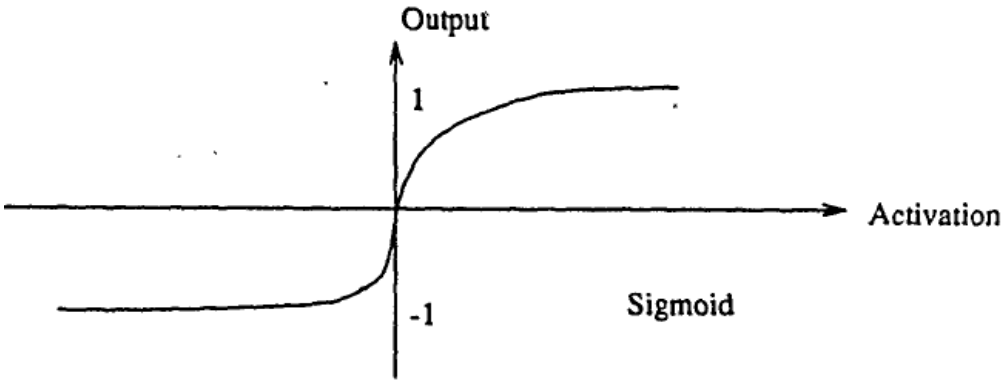


Figure 2.7: The Logistic sigmoid activation function (Haykin 1999).

2.4.1.2.1.1.2 The Linear activation function

The Linear activation function is a common activation function for ANNs. It produces only positive numbers over the entire real number range. It is the most useful function for the output layer of a predictive ANN because it has an output of continuous target values (EngelBretch et al. 1995). The linear activation function is given by the expression:

$$f(z) = z \qquad (2.4)$$

where z is the net input of the units in the ANN.

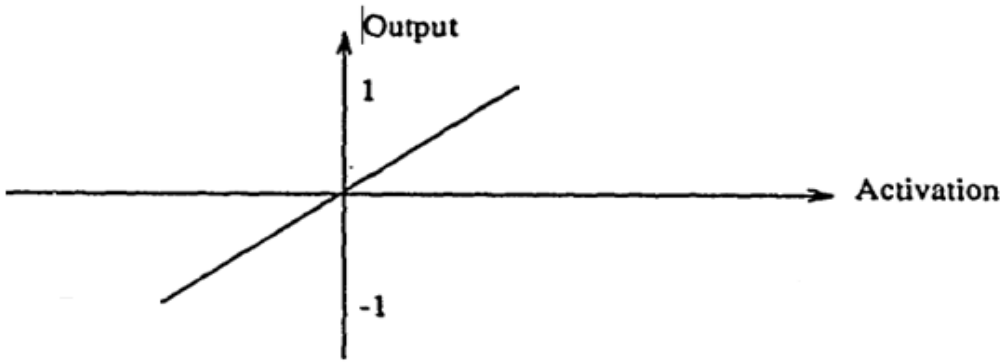


Figure 2.8: The Linear activation function (Haykin 1999).

2.4.1.2.2 The Universal Approximation property of MLPs

Cybenko (1989) proved that a MLP with hidden neurons, a sufficient number of hidden layers, of the Logistic sigmoidal activation type, and a single linear output neuron is capable of

approximating any continuous function. He formulated a theorem called the Universal theorem of Approximation which states:

THEOREM 2.1. *Given any measurable* $F: R^n \rightarrow R^m$*, there exists a feed-forward network with one hidden layer which can approximate that function well to any desired accuracy.*

Also, Zhang (2001) dealt with either one or two hidden layers in the multilayer network. With one hidden layer, he noted that the multilayer network can "*implement most decision surfaces, and can closely approximate any decision surface"*. Regarding the three layer network which included two hidden layers Zhang (2001), lists its attributes as being able to "*implement any separating decision surface when sufficient hidden units are represented in the two layers*". Zhang (2001) complements the more capable network with two hidden layers, he concludes that two hidden layers are sufficient and that "*additional layers do not add any representational power to the discrimination*".

2.5 Learning processes

Among the many interesting properties of ANNs is the ability of the ANN to learn from its environment and to improve its performance through learning. An ANN learns about its environment through an iterative process of adjustments applied to its synaptic weights and thresholds. Zhang and Kline (2007) define learning in the context of ANNs as: "*a process by which the free parameters of a neural network are adapted through a continuing process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place"*.

From this definition one can conclude that the learning process implies the following sequence of events:

- the ANN is stimulated by an environment;
- the ANN undergoes changes as a result of this stimulation;
- the ANN responds in a new way to the environment because of the changes that have occurred in its internal structure.

2.5.1 Learning paradigms

There are basically two learning paradigms: Supervised, and Unsupervised learning (Wei et al. 2004).

2.5.1.1 Unsupervised learning

Unsupervised learning, also referred to as self-organization, requires no target or desired outputs and relies only upon local information during the entire learning process (Wei et al. 2004). The task of Unsupervised learning is to learn to group together patterns that are similar of a given training set (Wan et al. 2009). In this mode of learning, the ANN must discover for itself any possibly existing patterns, regularities, separating properties, etc (Wan et al. 2009). While discovering these, the ANN undergoes change of its parameters, which is referred to as self-organization. A major disadvantage of this type of learning is that error information cannot be used to improve ANN behaviour, since the desired response is not known (Tsai & Lee 2011). With no information being available as to the correctness or incorrectness of responses, learning must somehow be accomplished based on observations of responses to inputs about which the ANN has little or no knowledge. Examples of unsupervised learning are Kohonen's self-organizing feature maps and Hebbian learning (Tsai & Lee 2011).

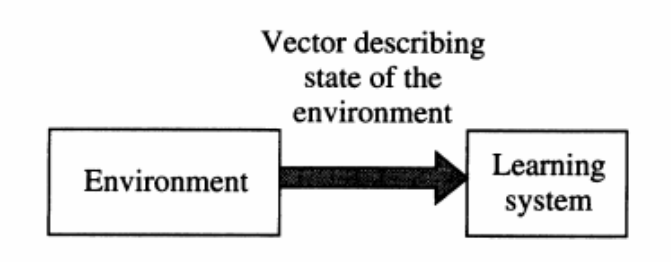


Figure 2.9: Unsupervised learning (Tsai & Lee 2011).

2.5.1.2 Supervised learning

In Supervised learning, a supervisor (or teacher) provides the ANN with an input pattern and the associated target, or desired response (Eberhart et al. 1990). The difference between the actual output of the ANN and the target output serves as an error measure and is used in correcting synaptic weights (Maier & Dandy 1997). The weights are adjusted gradually, by updating them at each step of the learning process so that the error between the ANN's output and corresponding desired output is reduced.

This adjustment is carried out iteratively in a step-by-step fashion with the aim of eventually making the ANN emulate the teacher. Since adjustable weights are assumed, the teacher may implement a reward and punishment scheme to adapt the ANN's weights (Canuto 2001). Supervised learning rewards accurate classifications or associations and punishes those that yield inaccurate responses (Piotrowski & Napiorkowski 2013). The reward or punishment is based on the teacher's estimate of the negative error gradient direction. An example of Supervised learning is error-correction learning, of which gradient-descent by back-propagation is an example. In this work we follow the Supervised model of learning.

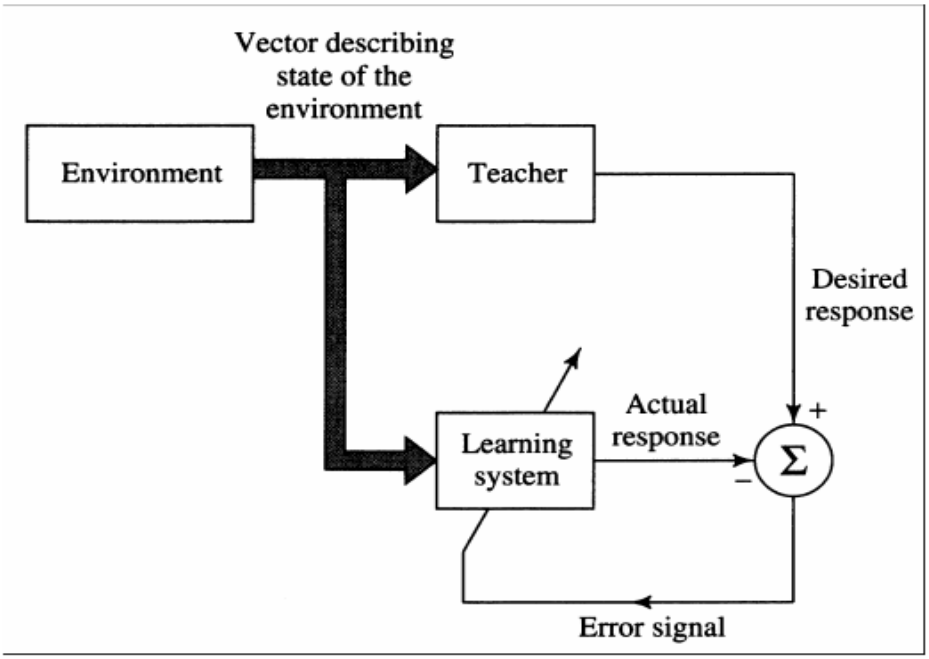


Figure 2.10: Supervised learning (Tsai & Lee 2011).

2.5.2 Modes of learning

Mode of learning refers to the type of weight adjustment implemented during training. Weights can be updated in two ways, namely Batch and On-line modes (Aamodt 2010).

- Batch learning/ Off-line learning: In this type of learning, weight changes are done only after the entire training set has been presented to the ANN. Weight changes for each presented pattern are accumulated and updated after each epoch (i.e. one complete presentation of the entire training set during the training process) (Badri 2010). In Off-line learning, once the ANN has been trained and enters recall mode (i.e. when the ANN is in operation) the weights are fixed and not modified at all (Abbas et al. 2013).

Off-line training provides a more accurate estimate of the gradient vector even though it requires more storage space than On-line training (Balestrassi et al. 2009).

- On-line learning: In this type of learning, the weights are adjusted after each pattern is presented to the ANN (Abbas et al. 2013). Once the ANN has been trained and enters recall mode, the weights are fixed and not modified at all. The advantage of On-line learning is that it requires less storage space than Batch training (Churchland & Sejnowski 2005).

This work assumes On-line learning since literature indicates that Batch learning requires more training time than On-line training on most problems of practical interest without any corresponding improvement in accuracy.

2.5.3 Learning algorithm

There are many training strategies developed for different ANN models. For training FFNNs the most popular technique is the Backpropagation (BP) algorithm introduced by Rummelhart (1967). The BP algorithm has been used in about 70% of ANN applications (Hillberg et al. 2009). It is appropriate for problems with the following characteristics (EngelBretch et al. 1995):

- Instances are represented by many attribute value pairs: The target function to be learned is defined over instances that can be described by a vector of predefined features, such as the pixel values. These input attributes may be highly correlated or independent of one another.
- The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.
- The training examples may contain errors: ANN learning methods are quite robust to noise in the training data.
- Long training times are acceptable: ANN learning algorithms typically require longer training times than decision tree learning algorithms. Training times can range from a few seconds to many hours, depending on factors such as the number of weights in the network, the number of training examples considered, and the settings of various learning algorithm parameters.
- Fast evaluation of the learned target function may be required: Although ANN learning times are relatively long, evaluating the learned network, in order to apply it to a subsequent instance, is typically very fast.

- The ability of humans to understand the learned target function is not important: The weights learned by ANNs are often difficult for humans to interpret.

We now review the mathematics behind the Backpropagation learning algorithm.

2.5.3.1 Backpropagation learning algorithm

The BP algorithm consists of two distinct phases: (a) the forward phase and (b) the backward phase (Churchland & Sejnowski 2005). For the forward pass, during training, a sample is presented to the ANN as input. For each layer, the output from the previous layer is used as an input to the next hidden layer until the output layer is reached and the output is produced. The output response is then compared to the known target output. Based on the value of the error, the connection weights are adjusted. In the backward pass weights are adapted to ensure that the minimum error between the targets and the actual outputs is achieved. The error is then propagated backwards.

2.5.3.1.1 Forward phase

In the forward phase, a pattern is presented to the ANN via the input layer to be propagated to the hidden layer. The Logistic sigmoid activation signal is computed for each layer and then passed to the other layers until it reaches the output layer. The output layer provides the response of the network for a given input pattern. The error *E* in the output layer is then computed as follows,

$$\boldsymbol{E = \frac{(O_k - O_k^*)^2}{2}} \qquad \textbf{(2.5)}$$

where O_k is the ANN output and O_k^* is the targeted output from the ANN.

2.5.3.1.2 Backward phase

In the backward pass, which starts at the output layer, the error computed in the forward pass is propagated backwards through the ANN, layer by layer, and *E*, i.e. the local error or gradient, for each neuron is computed recursively. To minimize the error it is essential that the weights are updated in the direction opposite to the gradient of the error function, i.e. $\frac{\delta E}{\delta w_{j,k}}$ has to be minimized. The chain rule is used to modify this,

$$\boldsymbol{\frac{\delta E}{\delta w_{j,k}} = \frac{\delta E}{\delta O_k}\frac{\delta O_k}{\delta I_k}\frac{\delta I_k}{\delta w_{j,k}}} \qquad \textbf{(2.6)}$$

Compute each of the terms in equation 2.6 as

$$\frac{\delta E}{\delta O_k} = \frac{\delta}{\delta O_k}\frac{(O_k - O_k^*)^2}{2} = (o_k - o_k^*) \quad \textbf{(2.7)}$$

$$\frac{\delta o_k}{\delta I_k} = \frac{\delta}{\delta I_k}\left(\frac{1}{1+e^{-I_k}}\right) = o_k(1 - o_k) \quad \textbf{(2.8)}$$

$$\frac{\delta I_k}{\delta w_{j,k}} = \frac{\delta}{\delta w_{j,k}}\sum_k w_{j,k}\, o_j = o_j \qquad (2.9)$$

assuming a learning rate of β and substituting the formulae in equations (2.7, 2.8, 2.9), the weight update rule which is updated by the vector $\Delta w_{j,k}$ so that

$$\Delta w_{j,k} = -\beta(o_k - o_k^*)o_k(1 - o_k)o_j \quad \textbf{(2.10)}$$

to compute for the hidden layers we need to adjust or minimize the summed error in the ANN. Minimize,

$$\Delta w_{i,j} = \sum_k \frac{\delta E}{\delta w_{i,j}} \qquad \textbf{(2.11)}$$

By adjusting the connection weight at $w_{i,j}$ this gives

$$\Delta w_{i,j} = \sum_k \frac{\delta E}{\delta O_K}\frac{\delta O_K}{\delta I_k}\frac{\delta I_k}{\delta o_j}\frac{\delta o_j}{\delta I_j}\frac{\delta I_j}{\delta w_{j,i}} \qquad \textbf{(2.12)}$$

The components can be computed as in equations (2.7, 2.8, and 2.9),

$$\frac{\delta E}{\delta o_k} = \frac{\delta}{\delta o_k}\frac{(o_k - o_k^*)^2}{2} = (o_k - o_k^*) \quad \textbf{(2.13)}$$

$$\frac{\delta o_k}{\delta I_k} = \frac{\delta}{\delta I_k}\frac{1}{1+e^{I_k}} = o_k(1 - o_k) \qquad \textbf{(2.14)}$$

$$\frac{\delta I_k}{\delta o_j} = w_{j,k} \qquad \textbf{(2.15)}$$

$$\frac{\delta o_j}{\delta I_j} = \frac{\delta}{\delta I_j}\frac{1}{(1+e^{-I_j})^2} = O_j(1-o_j) \qquad \textbf{(2.16)}$$

$$\frac{\delta I_j}{\delta w_{i,j}} = \frac{\delta}{\delta w_{i,j}}\sum_j w_{i,j}o_i = o_i \qquad \textbf{(2.17)}$$

the overall weight update algorithm is given by

$$\Delta w_{i,j} = -\beta(\sum_k (o_k - o_k^*)o_k(1-o_k)w_{j,k}\, o_j(1-o_j)o_i \quad \textbf{(2.18)}$$

where β is the learning rate ,o_j,o_i, are the output signals from neuron j and neuron i,O_k is the ANN output, O_k^* is the target output from the network, $w_{i,j}$ is the weight of connection from hidden neuron j to input neuron i and $w_{j,k}$ is the weight from output neuron k to neuron hidden neuron j. To better illustrate the principle behind the BP algorithm consider Figures 2.11 and 2.12.

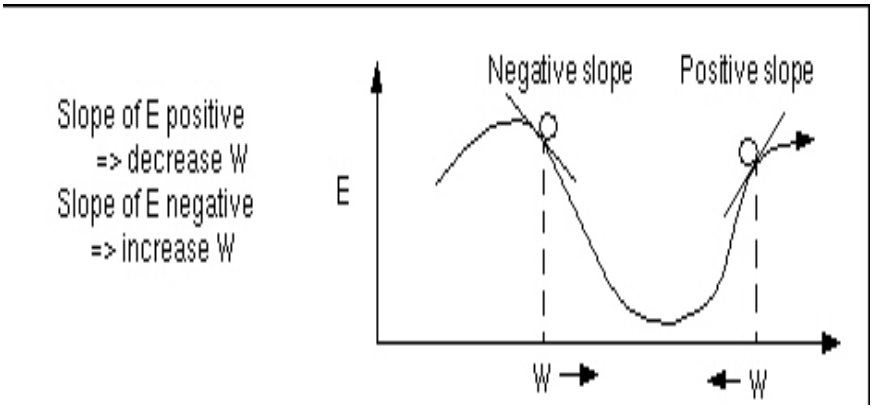


Figure 2.11: Backpropagation (Eberhart et al. 1990).

Figure 2.11 shows the relation between the behaviour of E (error function) and one weight w (in most ANNs there are many weights). To decrease the value of E, we always must move in the reverse direction of the gradient (slope). If the gradient of E is negative, we must increase w to move forward towards the minimum. If E is positive, we must move backwards to the minimum. By repeating this over and over, we move downhill in E until we reach a minimum, where $\nabla E(\vec{w}) = 0$, so that no further progress is possible, as illustrated in Figure 2.12.

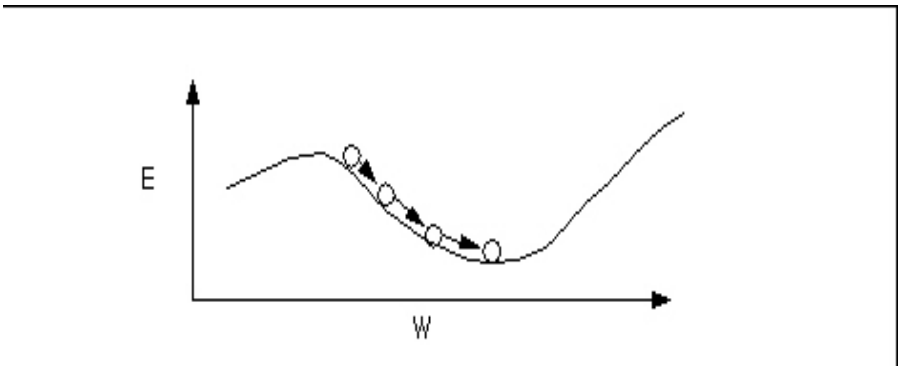


Figure 2.12: Gradient descent minimum attained (Eberhart et al. 1990).

There are many improvements to the BP algorithm, such as momentum term and weight decay, described in the appropriate literature (Haykin 1999; Control 2010). Cortez et al. (2006) praises the BP algorithm as an "*easy and elegant way of performing on-line (or stochastic) gradient descent to train neural networks*". Also according to Haykin (1999) the BP algorithm is popular for two primary reasons: the algorithm is easy to use and it provides effective solutions to large and difficult problems. However Tsilo (2008) lists several potential problems with the BP algorithm. He calls it a 'user's nightmare.' The potential problems he highlights include the following: choosing the appropriate learning rate and momentum term, choosing to train by epoch or by pattern (case), finding an effective initial random starting state, and deciding when to stop. He states that changing any of these parameters may have a 'major' impact upon the results.

Literature from other researchers also indicates problems associated with BP learning strategies, which include:

- Local minima in the error function surface: In practice, the error function can have, apart from the global minimum, multiple local minima. There is a danger of the algorithm landing in one of the local minima and thus not being able to reduce the error to highest extent possible by reaching a global minimum (Coulibaly et al. 2000).
- Insufficient search direction (Zurada 1994).
- For a step size that is too high, the gradient moves around the minimum but does not converge to the minimum (Zurada 1994).
- Slow convergence in case of small step size (Hessami et al. 2004).

- Relatively bad generalization at certain instances (Zurada 1994).

2.6 Time series forecasting

Forecasting is the art of predicting the future. Such activities are critical in many areas including macroeconomics, finance, production and facilities planning, sales and inventory control. In general, a forecast is often required when a decision is to be made regarding an uncertain future. To handle the increasing variety and complexity of forecasting problems, many new techniques have been developed. Each has its special use, and care must be taken to select the correct technique for a particular application. The selection of a method depends on many factors, including the context of the forecast, the degree of accuracy desired, the relevance and availability of historical data, the time period to be forecast, and the analyst's time available for performing the analysis (Control 2010).

2.6.1 TCP/IP network forecasting using ANNs

Various application domains exist for forecasting and analysing time series, however for the purposes of this research we concentrate solely on forecasting TCP/IP network traffic. Internet traffic prediction plays a fundamental role in network design, management, control, and optimization. Two of the most recent discoveries of the statistics of internet traffic over the last ten years are that internet traffic exhibits self-similarity and non-linearity (Kvale 2010; Choudhury et al. 2012). This self-similar and non-linear nature of network traffic makes highly accurate prediction difficult (Cortez et al. 2006). In the past several years, many methods have been proposed for TCP/IP network traffic prediction. Statistical models such as moving average, exponential smoothing methods, autoregressive models (AR), linear regression models, autoregressive moving average (ARMA) models, and machine learning models such as Kalman filtering and Box Jenkins have been widely used in practice for time series analysis and prediction of TCP/IP network traffic (Zhang et al. 1998). While some of these methods do improve the prediction performance for self-similar time series, most of them have proved to be both time-consuming and largely ineffective. Literature has indicated that unlike other methods, ANNs can approximate almost any function regardless of its degree of nonlinearity. This positions them as good candidates for modelling non linear and self similar time series such as network traffic (Chaoba 2009). Literature also shows their performance and success in predicting

TCP/IP network traffic trends increasing in accuracy from 64.3% in earlier approaches (Werbos 1989) to 70% and better (Cortez et al. 2010).

Researchers have applied many different ANN models to explore the traffic modelling task. Models, such as the Time Delay Neural Network (Waibel et al. 1989), Radial Basis Neural Network (Rivas et al. 2004) and Hopfield Neural Networks (Guang 2013) have been effectively utilised by several authors. The Sliding Time Window Neural Network (STWNN) model (Zhang 2001) which takes as inputs the time lags used to build a forecast, has also featured prominently as one of the most widely used ANN models, mainly due to its simplicity and the fact that it optimises the network performance by providing a method of predicting data that exhibit nonlinear time varying behaviour (Eberhart et al. 1990). In our work we adopt the STWNN approach for the forecasting models. We review it accordingly.

2.6.2 Sliding Time Window Forecasting Model

In this type of model data are presented to the ANN as a sliding window over the time series history. The ANN tries to learn the underlying data generation process during training, so that valid forecasts are made when new input values are provided (Moustafa et al. 2011). Figure 2.13 shows a STWNN forecasting model.

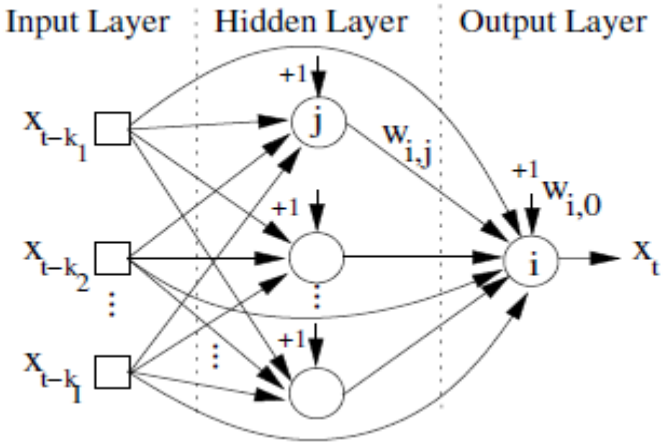


Figure 2.13: A FFNN with time window as a non-linear model (Moustafa et al. 2011).

Consider Figure 2.13, let $\boldsymbol{X_t} = (\ \boldsymbol{X_{1t}} \ldots\ldots \boldsymbol{X_{kt}})$ denote a multivariate series, where X_{ij} is the j^{th} chronological observation on variable *i*, and *k* is the number of distinct time variables.

Then:

$$\boldsymbol{X_{pt} = f(X_{1t-1}, \ldots\ldots X_{1t-n}, \ldots\ldots X_{rt-1}, \ldots\ldots X_{rt-n})} \quad \textbf{(2.19)}$$

$$\boldsymbol{e_t = X_{p,t} - X_{pt}} \quad \textbf{(2.20)}$$

where X_{pt} denotes the estimated value for the p^{th} variable at time *t*, f is the activation function of the forecasting model, and e_t is the error. The overall model is given by the expression

$$\boldsymbol{X_{p,t} = w_{o,0} + \sum_{i=I+H}^{I+H} f \sum_{s=1}^{k} \sum_{r=1}^{w_s} X_{st-L_{sr}} w_{i,j}} \quad \textbf{(2.21)}$$

where $w_{i,j}$ is the weight of the connection from node *j* to *i* (if *j*=0 then it is a bias connection), *o* denotes the output node and f is the Logistic sigmoidal activation function. Examples of ANN models of this type used for time series prediction purposes can be found in Singh et al.(2011) and Cortez et al. (2006).

2.7 Conclusion

This chapter gave a detailed review of literature and theoretical background of the fundamentals of ANNs. MLPs in particular were extensively reviewed, their basic architecture and relative weaknesses and strengths over SLPs were discussed in detail. The types of learning associated with ANNs, as well as the Logistic sigmoid and Linear activation functions whose application in the domain of ANN learning is well established, were also reviewed. This chapter also gave a detailed overview of the Backpropagation learning rule, which is the algorithm used to train the ANN models in this research. The mathematics behind this algorithm was presented and explained in well- articulated equations. Finally the Chapter ended by discussing in detail the Sliding Time Window Neural Network forecasting model, which is the model we adopt for implementation of the ANNs in this research. This chapter emphasized that, although the ANN approach can give considerably better results than conventional statistical techniques in many real world domains, it has some handicaps (or problems) that should be taken into consideration. The information given in this chapter is important in order to understand the concepts to be discussed in the following chapters. Chapter 3 will deal with the generalization ability of ANNs.

Chapter 3: Generalization of Artificial Neural Networks

To design and train ANNs that generalize well to new examples after having been trained on a sufficient set of training examples is a major goal of ANN learning. Several factors have been reported to affect generalization ability of ANNs, therefore considerable interest has been directed at understanding conditions under which good generalization occurs in ANNs. In this chapter we review some of the existing research into the generalization ability of ANNs and describe some of the limitations and strengths of that research. We lay particular emphasis on the five factors whose relationship to the generalization exhibited by an ANN this research investigates. Special attention is paid to the components of the network structure (i.e. input, hidden and output layers) and the learning parameters (i.e. training set size, learning algorithm, training schedule and the learning). We carefully examine previous work done by academia and industry in various case studies that looked promising at the time of this writing. The intent is to point out the value each case study offers, and then draw conclusions in the final section of this chapter.

3.1 Generalization overview

The essence of intelligence is plasticity of learning. Learning includes both memorization of the concepts which are required to be learned and generalization to new concepts which have not yet been encountered. Richards (1991) defines memorization as *"a search for a function that successfully maps or relates training inputs to their corresponding outputs"*. Kavzoglu (1999) defines generalization as "*the ability to map novel inputs to correct corresponding outputs based upon properties discovered in the training set*". The power of an ANN depends on how well it describes new data after completing the training process (Haykin 1999). Generalization is a measure of how well the ANN performs on the actual problem once training is complete. Once the ANN can generalize well, it is capable of dealing with new situations such as a new problem or a new point on the curve or surface (Mi et al. 2005). Figure 3.1 illustrates how generalization may occur in a hypothetical network. The points labelled *training data* are the nonlinear input-output mapping computed by the ANN as a result of learning. The points labelled *Generalization* are the result of interpolation performed by the ANN.

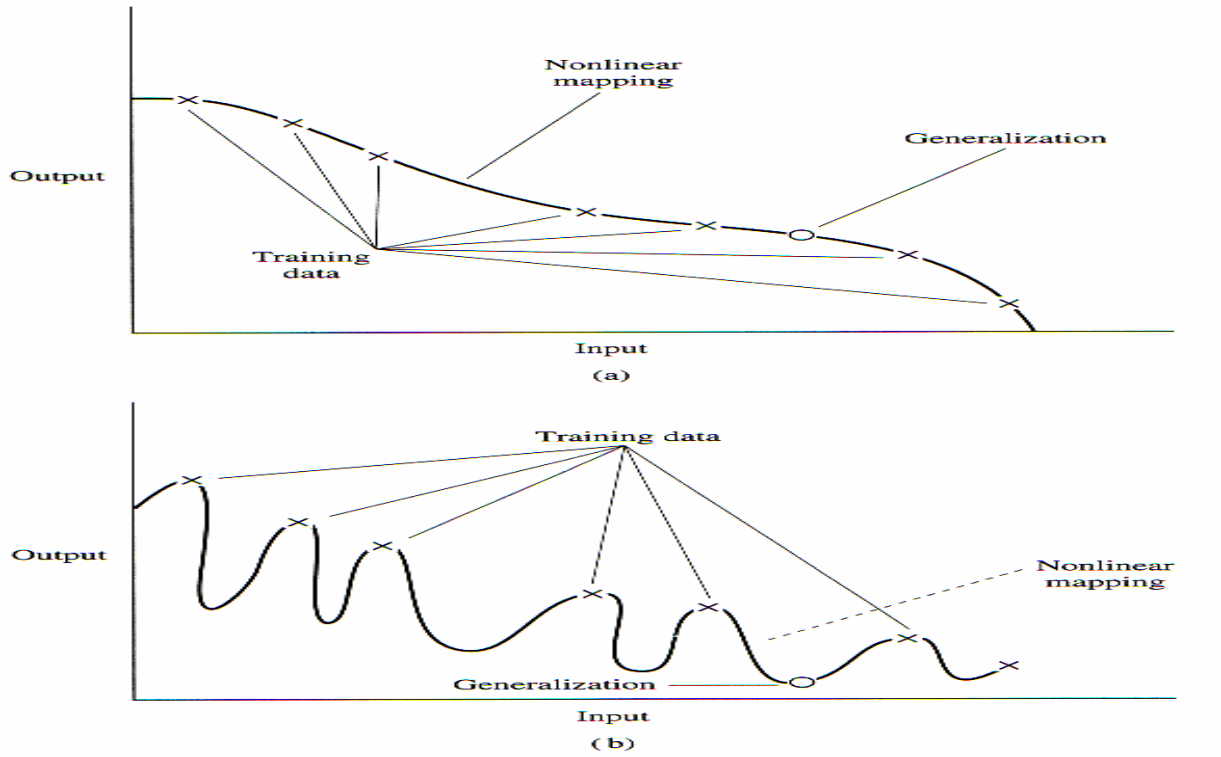


Figure 3.1:(a) Properly fitted data (good generalization). (b) Overfitted data (poor generalization) (Fearn 2004)**.**

Figure 3.1a shows a hypothetical performance of an ANN that generalizes well, as indicated by the smooth fit between the *Training* data points and the *Generalization* point. Figure 3.1 b shows an output from an ANN trained on a similar dataset as in Figure 3.1a. In this case however, the *Training* data points and the *Generalization* point appear not to be well correlated. This is a typical example of an ANN that has a bad generalization. ANNs have two phenomena that lead to their failure to generalize well. These are overfitting and underfitting (Schneider & Bishof 1992). The overfitting phenomenon occurs when the ANN has trained the noisy or imprecise data during the learning phase, and the underfitting problem is about the output of the ANN being far away from the actual underlying function of a given data set. Generally, these two phenomena identify the bias and variance dilemma which is described in detail in the next section. Research on generalization ability of ANNs can be classified into three categories: (1) theoretical prediction of expected generalization; (2) structural adaptation of network before, after, or during training for improving the generalization ability, and (3), empirical estimation of generalization ability of trained network. All three research directions are important since they

complement each other. A better theoretical understanding provides methods to implement better systems, a good estimation of generalization ability of a network provides methods of assessing accuracy of a new theory, and experimental results are often useful to form a hypothesis for establishing new theories. In this research we place particular emphasis on the third aspect, which is the empirical estimation of the generalization ability of trained ANNs. For a survey of literature for the first two research categories refer to Tsai and Lee (2011) and the references therein.

3.2 Bias, variance dilemma and generalization errors

Overall generalization errors come from two terms: bias and variance (Gonzalez 2000). Bias occurs when the ANN tries to fit all data points, including the noise. Variance addresses the smoothness of an approximated model in comparison with the actual underlying function that generated the training data (Wan et al. 2009). The overfitting problem occurs when the model has small bias and a large variance. Underfitting occurs when a model has a large bias and a small variance. There is always a trade-off between bias and variance. Bias and variance are complementary quantities and the best generalization performance will occur when a model achieves the optimal trade-off between the two, as indicated in Figure 3.2 (Teixeira & Fernandes 2012).

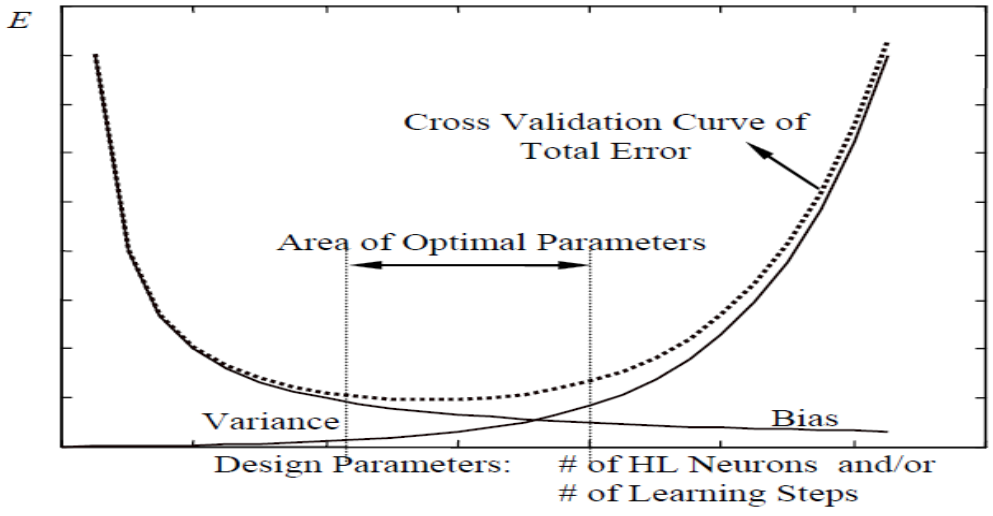


Figure 3.2: The optimal trade-off between bias and variance (Renals et al. 1992).

It has been experimentally proven that the degree to which a model overfits or underfits the data is related to five factors (Wan et al. 2009; Swingler 1996):

- the size of the network architecture,
- the size of the training set,
- the learning algorithm,
- the choice of training schedule,
- the size of the learning rate,

The choice of these parameters has an immense bearing on the performance of ANNs, however most pertinent is the extent to which each of these factors affects the ability of ANNs to generalize to new data. Research to date is inconclusive on this crucial aspect of ANN learning. With a plethora of research already published on this subject, we review some of the most crucial contributions.

3.3 Generalization: architecture and training set size.

By far the greatest amount of research into the ANN generalization problem has concentrated on exploring the impact of network architecture and training set size on ANN training and generalization. Network architecture means the set of input neurons, hidden neurons and output neurons together with the connections between neurons and the neuron groupings which combine to form a network of computing elements (Kavzoglu 1999). The training set, on the other hand, can be defined simply as all the data used to train the ANN. Haykin (1999) suggests that we should view the issue of generalization from two different perspectives:

- The architecture of the ANN is fixed (hopefully in accordance with the physical complexity of the underlying problem), and the issue to be resolved is that of determining the size of the training set needed for a good generalization to occur.
- The size of the training set is fixed, and the issue of interest is that of determining the best architecture of the ANN for achieving a good generalization.

Both viewpoints are valid in their own individual respects since they complement each other quite extensively. Several empirical results have been reported concerning the appropriate training set size for a given architecture, or reducing the capacity of an ANN to accommodate a fixed size training set. We undertake a detailed analysis of each of these perspectives.

3.3.1 Size of network architecture

For a given data set there may be an infinite number of network structures relevant to learn the characteristics of the data (Fearn 2004). The question however remains: "*What size of network will be best for a specific data set*?" Unfortunately, this is not an easy question to answer. The network architecture is of great importance as it dictates the number of free parameters available during training. It also affects learning time, but most importantly as motivation for this work, it is professed to affect the generalization capabilities of the network (Fearn 2004). Due to the difficulties encountered in analysing the behaviour of ANNs using nonlinear activation functions, theoretical work on the relationship between size of network architecture and generalization has been greatly limited (Richards 1991). The layer of input neurons, hidden neurons and output neurons are the most important attributes of a network (Nakamura 2005). Each of these units affects the performance of an ANN in diverse ways, we examine each of these.

3.3.1.1 The number of input neurons

The sole role of the input layer of neurons is to relate the external inputs to the neurons of the hidden layer (Lange et al. 1997). Selecting the most appropriate number of input neurons for any task is transparent and relatively easy to do. In a time series forecasting problem, the number of input neurons corresponds to the number of lagged observations used to discover the underlying pattern in a time series and to make forecasts for future values (Conway 1998). It has been experimentally proven that a linear relationship exists between the number of input neurons and the time necessary to train a network (Zhang et al. 1999; Gopal & Woodcock 1996; Grimm & Yarnold 1997). More input neurons in the ANN require more time for training. The input layer can be expanded simply by adding new data sources as additional neurons, but this increases the computation time (t) by the order of t^2 (Becker & Cun 1986). Therefore Wijayasekara et al. (2011) suggest that new network inputs sets should be added only if they contribute to a significantly improved performance. Every input neuron should represent some independent variable that has an influence on the output of the ANN (Teixeira & Fernandes 2012).

3.3.1.2 The number of output neurons

The number of output neurons is also relatively easy to specify as it is directly related to the problem under study (Poh et al. 1998). For a time series forecasting problem, the number of

output neurons often corresponds to the forecasting horizon (Bebis & Georgiopoulos 1997). There is a broad consensus amongst many ANN practitioners that the overall effect of the output neurons on the performance of ANNs is relatively negligible (Gallagher & Downs 1997; Karsoliya 2012). The only significant deviation from this viewpoint comes from Zhang and Kline (2007) who in the task of forecasting various quarterly seasonal time series resolved that the greater the number of output neurons, the more complex the nature of the problem to be solved, due to the separation of input space into more specific regions. They advise that it is important to choose the optimum number of output neurons to avoid unnecessary training.

3.3.1.3 The number of hidden neurons and layers

The hidden layer is the collection of neurons which has activation function applied on it as well as providing an intermediate layer between the input layer and the output layer. The numbers of hidden neurons and layers play very important roles for many applications of ANNs (Baum & Haussler 1989; Cherkassky & Zhong 2001; Kavzoglu 1999). It is the hidden neurons in the hidden layers that allow ANNs to detect the feature, to capture the pattern in the data, and to perform complicated nonlinear mapping between input and output variables. It is the hidden neurons in the hidden layers that define the complexity and power of the ANN model to be used to delineate underlying relationships and structures inherent in a particular dataset (Fearn 2004). More importantly, it is the hidden neurons in the hidden layers that determine the generalization ability of ANNs (Richards 1991; Weigend et al. 1990; Swingler 1996).

A major issue in the design of ANN models is the determination of the optimum number of hidden neurons. A common sentiment shared by many ANN practitioners is that the number of neurons in the hidden layer(s) should be large enough for the correct representation of the problem, but low enough to have adequate generalization capabilities (Chester 1990; Rasmussen 1993). Most of these authors tend to apply some form of Occam's razor to their networks. The Occam's razor principle attributed to the 14th century logician and Franciscan friar William of Ockham, states "*For a given dataset unnecessarily complex models should not be preferred to simpler ones*" (Rasmussen 1993). This approach, although favoured by some in the ANN research community, is emphatically rejected by others. Several researchers have investigated systems which vary the number of hidden units in the ANN in an attempt to determine the best generalization ability of a network for a given training set size. One such example is Sietsma and

Dow (1988). These workers developed a technique to test the hypothesis that ANNs with few hidden neurons on the first hidden layer generalize better than ANNs with many hidden neurons on the first hidden layer. They trained an ANN to classify sine waves of different frequencies. Two test sets were generated: one consisted of sine waves of different frequencies with different phase shifts from the training set, while the second consisted of similar sine waves corrupted with random noise added to the test patterns. The initial network architecture had 64 input neurons, 20 hidden neurons on the first hidden layer, 8 neurons on the second hidden layer and 3 output neurons. Results were obtained for training the ANN with several different initial conditions. The ANN was then re-trained while unused or non-contributing neurons were pruned from the ANN. At the end of their experiments, they resolved that reducing the ANN to the smallest size possible of classifying the training set degraded the generalization capability of the network. They state "*in some instances networks with many hidden neurons generalize better than networks with few hidden neurons*."

Another set of interesting findings comes from Gorman and Sejnowski (1998), who trained ANNs of 60 input neurons and 2 output neurons. They varied the number of hidden neurons from 0 to 24. We provide a summary of their results in Table 3.1.

Table 3.1: Results from experiments conducted by Gorman & Sejnowski (1998).

Average performance		
Hidden	**Training %**	**Testing %**
0	79.3	73.1
2	96.2	85.7
3	98.1	87.6
6	99.4	89.3
12	99.8	90.4
24	100.0	89.2

From Table 3.1 note that as the number of hidden neurons increases from 0 to 12, both the training recognition and testing generalization increase. Note also, however, that at 24 hidden

neurons the training recognition continued to rise while the testing generalization exhibits a very slight decrease. Gorman and Sejnowski (1998) conclude that the generalization ability of their ANNs increased with an increase in the number of hidden neurons. However careful analysis of their results points to some rather flawed conclusion. They seem to notice the increase in generalization from 0 to 12 hidden neurons only, without offering any explanation as to the essentially similar generalization trend exhibited across the range of 6 to 24 hidden neurons, an omission that could potentially render their conclusions questionable. Both studies from Gorman and Sejnowski (1998) and Sietsma and Dow (1988) are a far cry from the theorem enunciated by the Franciscan friar William in his Occam's razor principle. Furthermore, Lawrence et al. (1996) carried out a large-scale numerical study on the optimum network size. They found that networks with sizes larger than optimal size can sometimes generalize better. They caution, however, that this does not necessarily contradict the Occam's razor principle, as the quality of the different networks should be taken into consideration before any conclusions could be reached on the validity of the Occam's razor principle. Lawrence et al. (1996) share some form of an "Occam's Razor" bias and, at least to some degree, prefer simpler models.

On a rather contrary note, Weigend et al. (1990) trained an ANN with one hidden layer having a nonlinear activation in the hidden layers and a Linear activation at the output layer to predict time series in sunspot activities. They assumed that the best generalizations occur when the smallest network is still able to fit the training data. They began with a large ANN and gradually pruned it. At the end of their investigation they conclude that as the ANN size is gradually pruned the generalization ability of the ANN increases to a certain point beyond which it begins to decrease. The authors however do not offer any rationale as to how they selected their ANNs or what they meant by the term 'large', as in some cases 'large' could be deemed small by certain standards. Similar observations to those of Weigend et al. (1990) were made by Hinton (1988) during experiments on handwriting recognition using ANNs. Also, Morgan and Bourlard (1989) explored the impact of the number of hidden neurons on the generalization ability of ANNs in Phenome recognition. The number of hidden layer neurons was varied from 20 to 200, performance on the test set was monitored and training discontinued when test set performance showed no further improvement. They report that as the number of hidden neurons increased

from 20 to 200, performance on the training set increased from 75.7% to 86.7%, although corresponding performance on the generalization test set was found to decrease from 62.7% to 59.6%. We provide a summary of their results in Table 3.2.

Table 3.2: Results on generalization ability of Phenome recognition ANNs (Morgan & Bourlard 1989)

Phenome Recognition		
Hidden neurons	**Training (%)**	**Testing (%)**
5	62.8	54.2
20	75.7	62.7
50	73.7	60.6
200	86.7	59.6

From Table 3.2, note that as the number of hidden neurons increases from 5 to 200 there is a correlative increase in training recognition, however this is accompanied by a decrease in testing generalization. These results, however, may be passed off as slightly limited because of the restricted range of hidden neurons they chose to investigate. For an ANN with 1888 input neurons they only chose to explore a single hidden layer with 5 to 200 hidden neurons. It was also highly possible that ANNs could exhibit some intermittent behaviour in this range, for example, training recognition could increase to beyond 86.7% and then decrease to 86.7%. It is not clear why Morgan and Bourlard (1989) did not study ANNs having more than 200 hidden neurons, since the 200 hidden neuron exhibited poor generalization ability. Their findings are nevertheless consistent with Occam's razor principle, the essence of which is that all things being equal, the simplest solution tends to work the best.

Recent studies in the literature seem to suggest that the optimal network architecture for a good generalization ability of ANNs depends solely on the number of training patterns presented to the system, introducing yet another dimension to this debate. Martin and Pittmann (1990) used real world data in their examinations of the generalization problem in ANNs. In exploring the effect of network architecture on ANN generalization ability, they manipulated the network

architecture in various ways: varying the number of hidden neurons, limiting the connectivity between layers and distributing the hidden neurons across both one and two layers, and sharing connections weights between hidden neurons. After completion of their investigations, it turned out that using ANNs with fewer hidden neurons did not improve generalization, neither did limiting the connectivity between layers. They state "*The fact that we do not find no advantage to reducing the number of connections conflicts with Baum's and Haussler's estimates and the underlying assumption that network architecture plays a strong role in determining generalization*." They conclude their investigations by stating "*Given an architecture that enables relatively high training performance, we find only small effects of network architecture on generalization performance. It is probably better to devote limited resources to collecting a very large, representative training set than to extensive experimentation with different net architectures. The variations in network capacity that we have examined do not sufficiently affect generalization performance for sufficiently large training sets*." The results from Martin and Pittmann (1990) attest to the divergence in opinion by researchers as to the influence the network architecture has on the generalization ability of ANNs. Commensurate with the findings of Martin and Pittmann (1990), Lange et al. (1997) obtained an optimal generalization performance for a 10 dimensional input, 2 class output and 208 training set at only 8 hidden neurons, whilst Lippmann (1987) used only 8 hidden neurons to obtain good results for a 2 dimensional input, 2 class output and 200 input training patterns, highlighting the negligible effects of the network architecture on the generalization ability of their ANNs.

In the absence of an exact paradigm to estimate an optimally or near-optimally performing ANN architecture, literature has been inundated with several strategies and heuristics to estimate the optimum number of hidden layer neurons for an ideal generalization ability of ANNs. However, none of these methods has had the theoretical rigor of revealing optimal or at least near-optimal solutions. We give some examples of these heuristics in Table 3.3.

Table 3.3: Heuristics for selecting the optimal number of hidden neurons in ANNs for optimal generalization ability.

Heuristic	**researcher**
$2N_i$	Kanellopoulos and Wilkinson (1997)
$3N_i$	Hush et al. (1992)
$2N_i + 1$	Hecht-Nielsen (1987)
$2\frac{N_i}{3}$	Wang and Zhou (2005)
$\frac{N_i + N_o}{2}$	Ripley (1993)
$\frac{N_p}{r(N_i + N_0)}$	Stathakis (2009)

The number of input and output layer neurons is represented by N_i and N_o respectively, and the number of training samples (or patterns) is represented by N_p.

Several systematic approaches have also been proposed to obtain automatically the optimal number of hidden neurons in an ANN. These approaches are largely divided into two parts. One is to start with a small ANN and iteratively increase the number of neurons in the hidden layer(s) until satisfactory learning is achieved. The techniques based on this approach are called Constructive techniques (Hirose et al. 1991). One typical example of these includes the Adaptive method of architecture determination, suggested by Hashash et al. (2004).This technique begins with an arbitrary, but small, number of neurons in the hidden layers during training, and as the ANN approaches its capacity, new neurons are added to the hidden layers, and new connection weights are generated. Training is continued immediately after the new hidden neurons are added to allow the new connection weights to acquire the portion of the knowledge base which was not stored in the old connection weights. However, Bebis and Georgiopoulos (1997) raised an objection that this approach could be problem prone as small ANNs are sensitive to initial conditions and learning parameters. These ANNs are also more likely to become trapped in a local minimum as the error surface of a smaller ANN is more complicated and includes more

local minima than the error surface of a larger ANN. As a consequence of the algorithm, a number of ANNs must be trained to find the optimum network structure, which takes a long time. The second approach is to begin with a larger ANN and make it smaller by iteratively eliminating neurons in the hidden layer(s) or interconnections between neurons. These types of algorithms are called Pruning techniques. Optimum brain damage, Optimum brain surgeon, and Skeletonization are the major pruning techniques in use. After an ANN is trained to a desired solution with the training data, units (hidden layer neurons or interconnections) are analysed to find out which ones are not participating in the solution, then they are eliminated Kavzoglu (1999). However these techniques have also been found lacking in many respects and ultimately the selection of the most appropriate number of hidden neurons comes down to trial and error.

Equally important is the number of hidden layers. Available also in literature are heuristics for selecting the optimal number of hidden layers, but unfortunately they remain largely subjective and inconclusive. Reports from Lippmann (1987) and Cybenko (1989) indicate that a single hidden layer is generally sufficient for most ANNs to achieve a good generalization capability, particularly for forecasting tasks. Hecht-Nielsen (1987) provides a proof that a single hidden layer of neurons, operating a sigmoidal activation function, is sufficient to model any solution surface of practical interest. On the contrary, Flood and Kartam (1994) states that many solution surfaces are extremely difficult to model using a Sigmoidal network using one hidden layer. This assertion is further supported by Renals et al. (1992) and Srinivasan and Liew (1994), who insist that a second hidden layer does offer some added performance benefits. In fact, Srinivasan and Liew (1994), in one of their experiments, use two hidden layers in their network. They claim this resulted in a more compact architecture, which achieved a higher generalization performance than the single hidden layer network. Lapedes and Faber (1987) provide more practical proof that two hidden layers are sufficient, and according to Chester (1990), the first hidden layer is used to extract the local features of the input patterns while the second hidden layer is useful to extract the global features of the training patterns. However, Masters (1993) states that using more than one hidden layer often slows the training process dramatically and increases the chance of getting trapped in local minima.

The relationship between the network architecture and the generalization ability exhibited by an ANN has always been a cause of concern to ANN users. Lack of clear-cut guidelines on the selection of the appropriate network architecture for good generalization ability poses a major difficulty and obstacle for new users, thus undermining the popularity of ANNs. It is critical that this issue is further explored in a much more empirical and systematic manner. A wider range of ANNs of different sizes trained on a fixed training set, should be investigated, so as to ascertain their generalization capabilities. In performing these investigations, it is important that manipulation of the network structure should not just be done on the hidden neurons but on the hidden layers as well so as to arrive at more conclusive findings. If we train an Ensemble of ANNs with different numbers of hidden neurons, but with the same validation error, Occam's razor indicates that the smaller ANNs would generalize better than the larger ANNs during testing. More systematic simulations on some benchmarking network architectures should be conducted in order to ascertain whether this trend is apparent.

3.3.2 Training set size

The size of the training set employed at the learning stage has a significant impact on the performance of ANNs. The training set plays a major role in adjusting weights during the ANN learning process (Eskander et al. 2007). A general consensus amongst ANN practitioners is that if the number of training samples is not sufficient, the ANN cannot correctly learn the actual input-output relation of the system. On the other hand if the number of training samples is too large it may cause the ANN to overfit the data and increase training time (Chakraborty & Pal 1997; Karelson et al. 2006; Mencer & Parisi 1992). The training set must therefore contain enough information to learn the task. That then presents a fundamental problem in model selection: that of the selection of a concise training set. Without prior knowledge about the learning task, it is very difficult to obtain a representative training set.

There have been several attempts in the literature to estimate the optimum size of the training samples in relation to the network size and the generalization level desired of an ANN. Indications from most of these studies point to a conflict of opinions and findings on the part of the researchers. On one hand, some authors have reported a correlative increase in generalization ability of their ANNs with an increase in training set size, whilst on the other hand the opposite is the norm.

One typical example is provided by Leung and Zue (1989), who explored the impact of training set size on the generalization ability of an ANN having a fixed architecture in classifying speech data. The ANN was trained with sets consisting of 80 to over 8000 training samples. With an increase in training set size, training recognition results reached an asymptote of about 80%, while speaker independent testing results show that the generalization performance increased monotonically with training set size from 30% to 54%, with the greatest increase in accuracy occurring between 80 and over 8000 tokens.

Another group of authors to make known their findings are Chakraborty and Pal (2003) in forecasting electric load using ANNs. Their results show that with an increase in training set size, the generalization ability of the ANN increased from a mere 20% to over 75%. Similar sentiments were echoed by Lange and Manner (1994) in testing the effect of different training sample sizes on general forecasting tasks. They conclude that as the training sample size increased, the ANN forecaster performed better, with improved generalization ability. Furthermore, Hush et al. (1992), after carrying out a series of experiments on ANNs with fixed architectures and varying the training set size, conclude that the more training samples there are, the more incorrect functions the ANN is able to reject, and the more likely it is to find the correct function, leading to better generalization ability. Rivas et al. (2004), who did extensive work in forecasting currency exchange rates, indicate that larger quantities (periods of time) of data will produce more comprehensive models and better generalization.

Contrary to that, Ahmad and Tesauro (1998) trained an ANN to learn a linearly separable problem in order to study the relationship between the size of the network, the number of training samples, and the generalization exhibited by the ANN. Their results indicate that for a given input the generalization ability of the ANN decreased exponentially with the number of training samples. They also discovered that the number of training patterns required to achieve a fixed performance level increased linearly with the network size. Also, Zhuang and Hindi (1990) explored the size of the training set required by ANN to classify features. They conclude that using only a small percentage of about 5-10% of the data was more than adequate to train the ANN for a good generalization. They assert that any greater size of the training set would have been effectively detrimental to generalization ability of their model. Both studies are a stark

contradiction to the previous studies which had presumptuously concluded that the generalization ability of ANNs tends to increase as the training set size is enlarged. Also, experimental results by Lange et al. (1997) show that more training examples do not necessarily improve generalization. In their paper ”*Neural networks for pattern recognition*”, Lange et al. (1997) introduce the notion of a critical training set size. Through experimentation they found that examples beyond this critical size do not improve generalization, illustrating that an excess of patterns offers no real gain. They further state that this critical training set size is problem dependent.

In light of the dilemma of selecting an optimum training set size, several authors have proposed some rules of thumb for determining the appropriate training set size *vis-a-vis* network architecture and generalization thereof. However none of these heuristics has been universally accepted. Wei et al. (2004) suggest that one should never use more hidden layer neurons than training set samples. They state that the number of hidden layer neurons should always be much smaller than the size of training samples, otherwise the ANN will memorize the training samples, which leads to failure in generalization of new and unseen data.

Garson (1998) go on to further suggest that the number of training samples should be at least 10 times the number of input and middle layer neurons in the ANN for an appropriate generalization ability. Tong (1983) and Tong et al. (2005) suggest that for nonlinear forecasting models, to achieve good generalization at least 80% of any data sample should be held back for training. Flood and Kartam (1994) suggest using all the available data for training and using so-called synthetic time series for testing so as to reduce the data requirement in building ANN forecasters. They suggest this approach would be more sensible in cases where there is a deficiency of data, which is usually the case in many real world problems. This they claim will result in optimal generalization performance of any ANN. Some of the heuristics provided by various authors on the determination of an optimal training set size are summarized in Table 3.4. The number of input neurons is represented by N_i and the total number of weights is represented by N_w.

Table 3.4: Heuristics on the selection of optimal training set size.

Heuristic	**researcher**
$5 \times N_w$	Klimasauskas (1993)
$10 \times N_w$	Baum and Haussler (1989)
$30 \times N_i \times (N_i + 1)$	Hush (1989)
$60 \times N_i \times (N_i + 1)$	Hush and Horne (1993)
$30 \times N_w$	Garson (1998)

A more elaborate proposal comes from Vapnik and Cervonenkis (1968). They insist that the best way of determining the size of the training set is to consider the generalization error of the network, which is defined as the difference between the generalization on the training data and the generalization on the actual problem. In most cases, it is found that the difference between the two generalizations can be bounded, and by increasing the number of training samples this bound can be made arbitrarily small (Pissarenko 2002). This bound can be established when the number of training samples exceeds the Vapnik- Chervonenkis dimension (VC dimension). The VC dimension is a measure of the capability of the network (Haykin 1999). The VC dimension of a one-hidden-layer ANN with full connectivity between the layers is in the range (Zhang & Fukushige 2002):

$$\mathbf{2\left[N_h/_2\right]N_i \leq VC\ Dimension\ \leq 2N_w In\ (N_n)} \quad \mathbf{(3.1)}$$

where $N_h\ is$ the number of hidden neurons, N_i is the number of input neurons, N_w is the total number of weights in the ANN, and N_n is the total number of neurons in the ANN. The upper bound holds, irrespective of the number of layers and the connectivity (Mitchell 1997). This approach, whilst heavy on the technical aspect, has generated more questions than answers, with divergent views being expressed by authors. Sontag (1992) suggests that the training set size should be at least twice as large as the VC dimension. On the contrary, Baum and Haussler (1989) propose that if a generalization level of 90% is desired, the number of training samples should be about 10 times the VC dimension, or the number of weights in the ANN. However not

to be outdone Jain (1996), in one of his tutorial notes, states that the VC dimension is largely pessimistic. He goes on to give reasons as to how he arrived at this conclusion. He states that computing an upper bound on the expected risk is not practical in various situations and the VC dimension cannot be accurately estimated for nonlinear models such as ANNs. Furthermore he claims that the VC dimension may be infinite, requiring an infinite amount of data, and the upper bound may sometimes be trivial. However, one area of weakness in his conclusions is that they lack a practical basis as they are mainly based on complex mathematical equations. However his claims are further supported by Hasegawa et al. (1996) who in their various investigations concluded that the VC-dimension provides overly pessimistic bounds on the number of training examples, often leading to an overestimation of the required training set. Furthermore, experimental results have shown that acceptable generalization performances can be obtained with training set sizes much less than that specified by the VC-dimension (Cruz 1989).

On a more recent note, some researchers have come up with the idea of Active learning (also referred to in the literature as example selection, sequential learning, query-based learning) (Engelbreetch 2007). An active learning strategy allows the learner to dynamically select training examples, during training, from a candidate training set as received from the teacher (supervisor). The learner capitalizes on current attained knowledge to select examples from the candidate training set that are most likely to solve the problem, or that will lead to a maximum decrease in error. Rather than passively accepting training examples from the teacher, the ANN is allowed to use its current knowledge about the problem to have some deterministic control over which training examples to accept, and to guide the search for informative patterns. By adding this functionality to an ANN, the network changes from a passive learner to an active learner. This approach has led to shorter training times and better generalization, provided that the added complexity of the example selection method does not exceed the reduction in training computations (Conway 1998).

Authors such as Bretscher and Cohn (1970) show through average case analysis that the expected generalization performance of active learning is significantly better than passive learning. Similar sentiments are echoed by Seung et al. (1992) and Xie et al. (2002). In fact results presented by Seung et al. (1992), indicate that generalization error decreases more rapidly

for active learning than for passive learning. In spite of its advantages, Active learning is not without flaws. Seung et al. (1992), after performing extensive experiments on selecting training data sets, conclude active learning is time-consuming and is most effective for Ensemble networks. David and MacKay (2007) state that the technique of Active learning is extremely computationally expensive. Fukumizu (1992) conclude that one drawback of the active learning algorithms is that they rely on the inversion of an information matrix. If the information matrix is singular, the inverse of that matrix may not exist.

There is no doubt that the size of the training set plays a crucial role in the performance of ANNs, the intrinsic relation between the size of the training set, the network architecture and the generalization ability thereof cannot be over emphasized. Research on the selection of an optimal training set size for ANNs has been nothing short of abundant, but questions still remain unanswered, fundamentally: *for a given network architecture how big should the training set be for a good generalization?*. The answer to that question is to date largely an academic affair. All of the heuristics provided presently remain contentious and as such, render the ultimate process of selecting an appropriate training set size for ANN models a matter of trial and error. ANN researchers who have built ANN forecasting models for TCP/IP network traffic have typically utilized training data from one year e.g. Cortez et al. (2006) to sixteen years e.g. Cortez et al. (2010), including various training set sizes in between the two extremes, e.g. Chabaa (2010). However, once researchers have obtained their training data, they typically use all the data in building the ANN forecasting model, with no attempt at comparing data quantity effects on the quality of the produced forecasting models. To optimize the performance of ANNs in forecasting TCP/IP traffic trends, robust strategies need to be employed in an attempt to determine the optimal size of the training set that will guarantee good generalization. One potential direction would be to explore empirically the generalization ability of various sizes of training sets against different network architectures. The results from such experiments may then be used to define a critical training set size for ANN applications, in particular those dealing with TCP/IP forecasting problems.

3.3.3 Summary

Literature on the effects of network architecture and training set size on the generalization ability of ANNs indicates a plethora of views and opinions, most of them contradictory. The various views point to one of the following possibilities:

- Smaller ANNs tend to generalize better than larger ones in accordance with Occam's razor principle (Sietsma & Dow 1988).
- Larger ANNs generalize better than smaller ones contradicting Occam's razor principle (Gorman & Sejnowski 1998).
- Generalization ability of ANNs is directly proportional to the network size to a certain point, beyond which the relationship becomes inverse (Weigend et al. 1990).
- The effects of network architecture on generalization of ANNs are negligible (Martin & Pittmann 1990).
- Larger training sets results in better generalization of ANNs (Leung & Zue 1989).
- Smaller training set results in better generalization of ANNs (Zhuang & Hindi 1990).

We have presented an array of research work dedicated to answering some of the most crucial questions with regard to the relationship between the network architecture, training set size and the generalization ability of ANNs. We critically analysed each work, pointing out the necessary flaws and limitations, for instance we questioned the validity of Morgan and Bourlard's (1989) results after they conducted an investigation on a limited number of hidden neurons. We queried the work of Weigend et al.(1990), who did not give any rationale as to how they selected their ANNs or what they meant by the term "large" network. We also questioned the validity of conclusions made by Jain et al. (1996), which were purely based on mathematical computations without any real applications. However some of the work was thoroughly done, for instance Leung and Zue (1989), in their quest to determine the appropriate training set size for a good generalization, went out of their way to conduct training of up to 8000 samples. Also the investigation by Martin and Pittmann (1990) on exploring the appropriate network size for optimal generalization ability of ANNs was a fairly thorough job. They used real world data and manipulated the network architecture in various ways: varying the number of hidden neurons, limiting the connectivity between layers and distributing the hidden neurons across both 1 and 2 layers, and sharing connections weights between hidden neurons. The only downside of all these

studies is that none of them attempted a truly systematic investigation of the relationship between network architecture and training set size on the generalization ability of ANNs. They are also silent on a host of other factors which could potentially affect the generalization ability of the networks, such as learning rate, momentum, initial weights and learning algorithm. They merely focus on the training set size and network architecture without paying attention to the aforementioned factors. In our view this is a huge omission as these factors could influence the way in which the network architecture and training set respond to the generalization ability of the ANN. More insight into this subject is a necessity taking into account these and other factors.

3.4 Generalization: Other factors

There is no doubt that a huge portion of the research on the generalization ability of ANNs has mainly concentrated on the network architecture and the size of the training set. There is an extensive body of literature reporting on a variety of other different factors which also influence the ability of ANNs to generalize, namely: the size of the learning rate, the learning algorithm and the training schedule (Neeharika 1996; Richards 1991). During the course of our reviews it has emerged that a handful of studies has been conducted on adequately exploring the effects these factors have on the generalization ability of ANNs. It is also worth pointing out that in most of these studies the authors do not provide a full comprehensive set of results on generalization ability of the ANNs. Results are usually focused on other aspects such as speed of training, computational cost and probability of successful convergence (ability of the ANN to converge to specified error levels). In this section we review some of the most pertinent work and literature in so far as studies conducted on the relationship between these additional factors and the generalization exhibited by an ANN is concerned.

3.4.1 Learning rate

The learning rate, also referred to as the step size parameter, determines how much the weights can change in response to an observed error on the training set. It is considered a key parameter for a successful ANN application because it controls the size of each step toward the minimum of the objective function (Attoh-Okine 1999). The learning rate is usually a value chosen between 0 and 1.

A general consensus amongst ANN practitioners is that a learning rate that is too large often moves too far in the correct direction, resulting in overshooting a valley or minimum in the error surface. This indelibly leads to longer training times, because it is continually overshooting its objective and unlearning what it has learned, leading to poor generalization ability of the ANN. On the other hand, a learning rate that is too small increases training time, resulting in the ANN taking many more steps than necessary to reach the goal and a greater likelihood of becoming trapped in a local minimum, or a plateau on the error surface, also resulting in poor generalization ability (Conway 1998; Plaut et al. 1986). Quite obviously, the best learning rate for good generalization is not so obvious. The selection of the appropriate size of the learning rate has been a huge problem for a number of ANN users. Studies pertaining to this aspect of ANN learning, though few, have presented mixed conclusions.

Wilson and Martinez (2001) conducted experiments on speech digit recognition. They conclude that a learning rate that is too large often hurts generalization accuracy and also slows down training. They further state that, once the learning rate is small enough, further reductions in size would result in a waste of computational resources without any further improvement in generalization ability. Richards (1991) also did extensive studies on the relationship between learning rate and the generalization ability of ANNs in phenome recognition. Their conclusion was that ANNs trained with higher learning rates tend to have a better generalization ability than those trained with lower learning rates. However they state that this assertion holds true only for ANNs trained with a single hidden layer; when the hidden layers are doubled the opposite is quite true. The authors do not offer much explanation for this phenomenon except suggesting that it's an area that needs further insight. Another group of authors, Mohammad and Luis (2011), using different learning rates, tested the performance of ANNs for several time series which had been previously reported to have worse results with ANNs. They conclude that high learning rates are good for less complex data and low learning rates should be used for more complex time series data.

In light of the uncertainty surrounding the selection of the most appropriate learning rate for ideal generalization ability of ANNs, several authors have attempted to offer some sort of heuristics. In practice these heuristics are frequently used as points of departure for subsequent

search by trial and error. Plaut et al. (1986) proposed that the learning rate should be inversely proportional to the fan-in of a neuron. This approach has been theoretically justified through an analysis of the Eigen value distribution of the Hessian matrix of the objective function (Becker & Cun 1986). Reports from Swingler (1996) suggest that starting with a large value for the learning rate of 0.75 and reducing to 0.25 and then to 0.1 as the network starts to oscillate is a good way of reaching the global minimum of error, leading to higher generalization ability of ANNs. Accounts from Becker and Cun (1986) suggest that for a given neuron, the learning rate should be inversely proportional to the square root of the synaptic weights made to that neuron for an optimal generalization ability of the ANNs to be attained. Gonzalez (2000) reports that one- and two-layered networks with a learning rate of 0.2 and a momentum of 0.3 yield the best combination for good generalization ability. Unfortunately, this study was only based on simulated data. Richards (1991) shows that for fixed parameters, almost optimal generalization requires learning to go to zero at an appropriate rate, however Katidiotis et al. (2000) contend that a more nearly constant learning rate might well be preferable if parameters are time varying. Traditionally, learning rates remain fixed during training, but some authors have come up with proposals to dynamically adjust the learning rate during training. One proposal comes from Jacobs (1998), who suggests that one should assume that each weight has a different learning rate n_{kj}. The following rule is then applied to each weight before that weight is updated: if the direction in which the error decreases at this weight change is the same as the direction in which it has been decreasing recently, then n_{kj} is increased; if not, n_{kj} is decreased. The direction in which the error decreases is determined by the sign of the partial derivative of the objective function with respect to the weight. An alternative is to use an annealing schedule to gradually reduce a large learning rate to a smaller value (Vogl et al. 1988). This allows for large initial steps, in the region of the minimum. However, Shavlik et al. (1991) discourage this approach by stating that these methods require the selection of parameters which determine the rate at which the learning rate is to be adjusted. They also state that the optimization of these parameters is highly problem-dependent and "… *may lead to over adjustment of the weights, resulting in dramatic divergence".*

It is evident that the debate on the most appropriate learning rate for an optimal generalization ability of ANNs is largely ongoing. Analysis of the existing studies shows a lack of thorough

probing on the part of researchers. For instance there is a widely held perception that the momentum attributed to smoothing bumps in the error surface by referring to the previous weight is closely linked to the learning rate (Attoh-Okine 1999). If this assertion is remotely true, as indeed proven by Attoh-Okine (1999) in predicting pavement conditions, and Tsai and Lee (2011) in a hand gesture recognition system, then any investigation done on the learning rate cannot exclude the momentum term. None of the authors mentioned thus far has dared to fully conduct systematic investigations in that direction. Although some heuristics on selecting the optimum combination of learning rate and momentum are found in literature and presented in Table 3.5, most of them have not been universally accepted as they are largely based on toy problems with no relevance to the real world.

Table 3.5: Heuristics for optimum learning rate and momentum term.

Learning rate and momentum combination	**Author**
0.01, 0.00005	Paola and Schowengerdt (1997)
0.05, 0.5	Wilson and Martinez (2001)
0.1, 0.9	Foddy et al. (1996)
0.5, 0.9	Hara et al. (1994)
0.8, 0.2	Staufer and Fischer (1997)
0.15, 0.075	Eberhart et al. (1990)
0.05,0.5	Yates et al. (1996)
0.2,0.6	Govy et al. (1996)

At present we are not aware of any major investigations on the effects of the learning rate on the generalization ability of ANNs, taking into account the possible influence of a momentum term. Most data used for previous research were simulated, making it difficult to generalize the conclusions of these studies. Furthermore, these studies tend to focus on network convergence without paying much dividend on generalization ability, which is an equally important aspect of ANN learning. The only rigorous studies we are aware of which have come close to addressing this issue are those conducted by Maier and Dandy (1999), who experimented with various

learning rates and momentum combinations in forecasting salinity in Rivieree river, Australia. They empirically found that a learning rate of 0.6 and momentum of 0.2 ensured reasonably good convergence and generalization ability of their networks. Richards (1991), underscored a critical observation on how the sensitivity of networks to learning rates largely depends on the architecture of the networks. He found that multi hidden layer networks are more sensitive to the learning rate than single hidden layers. Although he conducted limited investigations of only 2 learning rates, 0.1 and 0.01. This is an area which, if fully investigated, has potential for significant strides insofar as the generalization ability of ANNs is concerned. Therefore continued research on this crucial aspect of ANN design remains an ardent necessity.

3.4.2 Learning algorithm

The choice of learning algorithm has a significant impact on the performance of ANNs (Shavlik et al. 1991). The Backpropagation (BP) algorithm has been widely used, well investigated, and is one of the most popular learning algorithm classes for ANNs. Its effectiveess and flaws have been well documented in various publications, for example an online BP learning algorithm for time-varying inputs (Mahmoud et al. 2007), an adaptive learning algorithm with reduced complexity (Wang & Zhou 2005), a fast-learning algorithm based on the gradient descent of neuron space (Mencer & Parisi 1992), and a general BP learning algorithm for FFMLP (Hara et al. 1994). For a detailed review of the mathematics behind the BP algorithm the reader is advised to consult section 2.4.

Many variations of the BP learning algorithm are provided in the Matlab Toolbox and other Neural Network software. These include Batch gradient descent (traingd), Batch gradient descent with momentum (traingdm), Variable learning rate back-propagation (traingad), Resilient back-propagation (trainrp), Conjugate gradient (traincgf, traincgp, traincgb, trainscg), Quasi-Newton (trainbfg, trainoss), Levenberg-Marquardt (trainlm), and the Batch gradient descent algorithm (traingd). Although the BP algorithm has been a significant milestone in ANN research, it is known as an algorithm with a very poor convergence rate, increasing the chances of being trapped in local minima (Lahmiri 2011). Many attempts have been made to speed up the BP algorithm. Commonly known heuristic approaches such as momentum term (Jacobs 1998), variable learning rate (Hagan & Menhaj 1994), or stochastic learning (Barry 2000), have only led to a slight improvement in that regard. Although many of the studies on BP algorithms have

mainly focused on optimizing the speed of convergence, a select few have gone on to further study how the generalization exhibited by an ANN responds to a change in learning algorithm. One most notable example is Neeharika (1996). He did extensive studies on the generalization ability of ANNs for pattern recognition tasks; he states that due to the slow convergence rate of the BP algorithm it was difficult to train the ANN on a large number of training examples as the time required to train the network became very large. He concludes that this had a huge impact on the generalization ability of the ANNs, and suggest methods of accelerating the learning algorithm in an effort to train the network with large number of examples in finite time. Also, Richards (1991), while studying the generalization ability of ANNs in speech recognition tasks made a very profound contribution to the study of Connectionist generalization. After performing extensive comparative investigations on the generalization ability of two ANNs trained on the same size of the network architecture and on similar training set sizes, with one having the Gradient Descent algorithm and the other Conjugate Gradient learning algorithm, they conclude "*Different training algorithms may facilitate or hinder the development of generalization*."

Another interesting set of results is reported by Lahmiri (2011) in performing a comparative study on BP algorithms in financial prediction. The findings indicate that BP algorithm trained with Conjugate (BFGS) and the Levenberg-Marquardt (trainlm) provides the best generalization accuracy according to RMSE. Also, Mahmoud et al.(2007) performed a comparative investigation on the effect of using different learning algorithms on the performance of ANNs with BP algorithm in image coding and compression. Based on the obtained results the Gradient Descent algorithm took less time during training than did the Conjugate Gradient methods and Quasi Newton methods. However Quasi Newton performed better in terms of generalization ability, such that the image compressed by Quasi Newton had a higher quality. They also observed that, in general, one hidden layer network had a better generalization performance than the three hidden layer network. Piotrowski and Napiorkowski (2011) compared Levenberg-Marquardt and the Conjugate gradient algorithms for stream-flow forecasting and determination of lateral stress in cohesionless soils. They found that Levenberg- Marquardt algorithm was faster and achieved better convergence and generalization performance than the other algorithm during training. Esugasini et al. (2005) considered the problem of breast cancer diagnosis and

compared the generalization ability of the Standard Steepest Descent against that of the Gradient Descent with momentum and adaptive learning, Resilient BP, Quasi-Newton and Levenberg-Marquardt algorithm. The simulations show that the ANN using the Levenberg-Marquardt algorithm achieved the best generalization performance.

In their research, Ahmad et al. (2008) employed three ANNs with different algorithms to the problem of intrusion detection in computer and network systems. The learning algorithms considered by the authors were the Standard, the Batch, and the Resilient BP algorithms. They conclude that the Resilient algorithm had a better generalization performance.
Finally, Lahmiri (2011) compared the performance of the standard BP and Levenberg-Marquardt algorithm in the prediction of a radio network planning tool. They found that the standard BP algorithm achieved the minimum error and outperformed the Levenberg-Marquardt algorithm in terms of speed of convergence and generalization accuracy.

In the context of TCP/IP network traffic forecasting, the BP is, indeed, the most employed algorithm to train ANNs (Chabaa 2010; Nguyen & Chan 2004). However, of the studies conducted thus far, comparative research on the generalization ability of the various BP learning algorithms is conspicuously absent. Also no special mention of the accuracy and the relative learning times of the different algorithms on novel examples has been made. More insight is therefore needed to address these concerns and to shed more light on the relative merits of the different approaches to Backpropagation learning.

3.4.3 Training schedule

The effects of overtraining on ANNs has drawn significant attention with sometimes apparently conflicting results, even about the mere presence or absence of the effect. An operational definition of overtraining is that the out- of- sample error starts to increase with training time after having gone through a minimum (Piotrowski & Napiorkowski 2013). A standard interpretation of this would be that the ANN at this point starts to pick up some idiosyncrasies of the training set that do not generalize to the test set.
The overtraining phenomenon is well documented in ANN literature (Badri 2010). The effects of overtraining on the generalization ability of ANNs have been receiving growing attention from ANN practitioners. Pissarenko (2002) states that a major objective of training in ANNs is to find

a set of weights between the neurons that determine the global minimum of the error function. He insists that unless the model is overtrained, this set of weights should provide a good generalization. Zhang et al. (1998) further affirm these observations by stating *"A network is overtrained when it conforms exactly to the training data rather than generalizing it. This produces an optimal network with very low approximation errors, but only for the training dataset as when data with subtle differences are introduced to such a network, the errors start to dramatically increase"*. Morgan and Bourlard (1989) performed an empirical study between the number of weights in an ANN and the ability of the network to generalize well to new examples. They used both simulated data and speech data to train the network. They conclude, "*While both studies show the expected effects of overtraining, which are poor generalization and sensitivity to overtraining in the presence of noise, perhaps the most significant result is that it was possible to greatly reduce the sensitivity to the choice of network size by directly observing the network performance on an independent test set during the course of learning*".

Although some researchers are agreed that overtraining is synonymous with bad generalization, not all of them are of that mind. One such example is Richards (1991). He studied the generalization ability of ANNs in phenome recognition. He trained his two hidden layer networks with a learning rate of 0.01 for a hefty 128000 epochs, and at the end of his exploration he concludes "*Contrary to the results obtained by Bourlard my networks did not seem to exhibit any loss in generalization due to extensive training*". Such a statement is evidence of the conflicting sentiments shared by the various researchers on this subject and a clarion call for continued research on this issue.

Closely linked to the overtraining phenomena is the training schedule. It is a crucial element of ANN training as it determines the level of training that a particular ANN model is subjected to. Actually one of the major difficulties in the use of ANNs is to determine the appropriate training schedule that will ensure the learning process is terminated at a point where good generalization ability is achieved. This is an important yet relatively explored area. Several suggestions have been made by various authors to help determine the actual point at which the learning process should be halted. These suggestions are largely divided into 3 parts: (1) User defined error level, (2) Early stopping and (3) Fixed iteration. As to which of these approaches is more effective in

as far as the generalization ability of ANNs is concerned is largely subjective. ANN practitioners have raised contrasting opinions with regard to each of these techniques. Some researchers such as Wilson and Martinez (2001) suggest the first approach is suitable for extremely large datasets only, whilst some, such as Rasmussen (1993) suggest the second approach may prevent the BP algorithm to stop learning before its full learning capability is attained. On the other hand, Lahmiri (2011) suggests the third approach limits the BP to its potential to learn more and further minimize its error rate. Although each of these perspectives is credible and admissible of further investigations, for the purposes of this research we focus on the third approach. We briefly review some of the pertinent literature on this subject. For literature on the other two stopping criteria, the reader is advised to consult Mitchell (1997).

3.4.3.1 Fixed iteration

One of the most important issues in ANN learning is the adjustment of weights. The number of training iterations (or epochs) is a core parameter in weight adjustment (Abbas et al. 2013). An epoch is defined as one weight update or training iteration. The epoch size used can be equal to the entire training set or a subset of the training set and is usually selected randomly (Richards 1991). For each epoch, the BP learning algorithm builds a different model, *i.e.* an ANN with a different set of weights. If an ANN is trained up to 1000 epochs, it means the learning algorithm has to move through 1000 different models to reach an optimal solution. The choice of this parameter is therefore extremely important for ANN training. If the number of training iterations is set too low the ANN may not reach a sufficient level of learning, resulting in underfitting, whilst too many iterations can cause the ANN to overfit the data, both cases leading to reduced generalization ability (Paola & Schowengerdt 1997).

A common method of stopping the training process of an ANN is to train the network a pre-defined number of iterations, hoping that the network will have reached the global minimum of the error. Zhang et al. (1999) employed this strategy to terminate the training process and recorded good performance and accuracy on their ANNs. Kaastra and Boyd (1996) did extensive research in forecasting financial markets. They state that it is difficult to determine a general value for the maximum number of training iterations required for optimal generalization ability because training is affected by many parameters such as the choice of learning rate, momentum values, and proprietary improvements to the BP algorithm, which vary from study to study.

Some of the ANN studies that mention the number of training iterations, report maximum generalization and convergence for their ANNs from 85 to 5000 training epochs. This range, however, can be as wide as 50000 to 191400 training iterations, resulting in long training times of 60 hours Klimasauskas (1993).

Balestrassi et al. (2009) state that the only appeal of this method as a stopping criterion is simplicity, as it does not use any information or feedback from the system before or during training. They conclude "*When the number of iterations is capped at a predefined value, there is no guarantee that the learning machine has found coefficients that are close to the optimal values*." Also, Maier and Dandy (1998) during their investigations in determining the effects of internal parameters in forecasting salinity in Rivierre river, Australia, state three conclusions: (1) There was no significant difference between the best forecasts obtained when different epoch sizes were used. (2) Learning speed was much greater for the ANNs with smaller epoch sizes. (3). There was a steady decrease in the RMSE prediction error for all ANNs until a local minimum 'plateau' in the error surface was reached. Maier and Dandy (1998) give an overall conclusion by stating quite succinctly *"For the data set used, there was no advantage in using larger epoch sizes. The number of normalized weight updates required to obtain the best result was independent of epoch size. As the time taken to perform one weight update increases with increasing epoch size, learning is much faster with smaller epoch sizes. The generalization ability of the network was found to be unaffected by epoch size*." However, Tsai and Lee (2011), in their studies on the parameter effect on performance in ANN for a hand gesture recognition system, state that *"...epoch size affected the learning ability of the network*". These findings were in contrast to those of Maier and Dandy (1998) and perhaps put into fruition claims made by Kaastra and Boyd (1996) that the selection of an appropriate number of training epochs depends largely on the problem. Tsai and Lee (2011) conclude by stating "…..*when the epoch was in the 1000 to 3500 range, training set error was minimal. However, to avoid over fitting, the number of the epochs was limited to 3000.*" Further to that, Lodwich et al. (2009), in their experiments on an analysis of stopping criteria in ANNs, state that using a maximum number of 600 epochs deprived their network of sufficient training. They state "*.... However, the total absolute error on the testing set appeared to decrease slightly even after epoch 600*." They

suggest starting off with a large number of epochs and gradually reducing the number until the true minimum RMSE on the test set is achieved.

The number of training iterations is a common phrase in ANN vocabulary, but not much effort has been afforded on a thorough analysis of the influence this design factor has on the efficient training of ANNs. Most studies on the number of training iterations have mainly focused on convergence rate without mentioning much on the generalization ability of the ANNs, which is a crucial part of a successful ANN implementation. This could effectively be one of the areas overlooked by many researchers which has potential to spur the discourse on the generalization ability of ANNs in an altogether different trajectory.

3.5 Conclusion

The chapter has discussed major literature in as far as the studies pertaining to the generalization ability of ANNs is concerned. While not exhaustive, it has reviewed and acknowledged a number of current case studies intended to resolve the fundamental question of the generalization ability of ANNs. The main purpose of this chapter was to provide both theoretical knowledge available to date by referring to recent studies, and heuristics developed by researchers as a result of their experience. Some of the studies conducted were thorough, but some were lacking in many respects. Of note are the experiments conducted by Martin and Pittmann (1990) in a clear and articulate attempt to explore the appropriate network size for optimal generalization ability of ANNs. They used real world data and empirically manipulated the network architecture in various ways. Yet others such as Jain (1996), in an attempt to solve a similar problem, resorted to the use of purely mathematically-based simulations without any real applications. Needless to say, a similar pattern was noted for most of the case studies reviewed on each of the five factors whose relationship to the generalization ability of ANNs this research investigates. Literature on most of the case studies reviewed indicates a contrast of opinions and findings on the part of the researchers concerned. However, it is vital to note that these projects have been done in different countries under different experimental conditions, hence the diversity of results. One striking omission in the literature is on studies pertaining to the generalization ability of ANNs in time series forecasting of TCP/IP network traffic trends. If the forecasting of TCP/IP network traffic is to be optimized by any standard it is imperative that studies on the generalization ability of ANNs be directed in that direction. Perhaps the most important conclusion gathered in this

Chapter is the fact that although several heuristics exist in determining various parameters for generalization ability of ANNs, most of these have not been universally accepted and/or a largely based on toy problems with no real significance in the real world. Hence to date, for a given problem the ultimate method of determining appropriate ANN learning parameters remains that of trial and error. Chapter 4 presents the methodology and data techniques of the research.

Chapter 4: Methodology and Data Techniques

This chapter discusses the methods and data techniques used in this study for investigating the generalization ability of ANNs in forecasting TCP/IP network traffic trends. It covers the critical decisions and techniques employed in this evaluation. The chapter is divided into 7 sections. Each section describes an aspect of the ANN model development process, highlighting the choices made during development. Section 4.1 describes the case study under consideration. Section 4.2 describes the software and computer specifications used for performing the investigations. Section 4.3 gives an account of the data and its composition including the source. Section 4.4 describes various techniques employed for pre-processing the data prior to conducting the experiments. Section 4.5 gives a description of the various experiments conducted during ANN training, as well as the experimental strategic and tactical decisions highlighting the rationale behind them. Section 4.6 relates the measures of performance and the various evaluation criteria used to judge the performance of the ANN models in our research, followed by a conclusion and summary of the chapter in Section 4.7.

4.1 The case study

Prior to discussing the materials, methods and the experimental procedures related to our research, we firstly turn to the subject of how the most appropriate problem domain to conduct our investigations was selected. To rigorously conduct our investigations through manipulating the structure of an ANN in various ways and observing the resultant effect upon generalization, we needed to have a domain in which meaningful variations in ANN structure could be explored. An easy route would have been to develop an artificial domain in which to carry out our investigations. To do so would require making assumptions about the underlying structures of that domain, notwithstanding that, using an artificial domain could well have resulted in criticism of our research based on several objections. The domain could have been passed off as being a "toy "domain, some criticism could have been raised concerning the underlying justifications for the task requirements and it could have been claimed that the results of such an investigation simply might not scale up to larger ANNs using larger representations of collected experimental data. Using a real domain for our investigations was therefore the best alternative as it avoids both these objections. In addition to this, it also eliminated the time and effort involved in attempting to develop an appropriate artificial domain: such time and effort was better spent

investigating the question that forms the focus of our research. We had to look for a real domain which would be easy to manipulate and well formulated to address our research objectives. The forecasting of TCP/IP network traffic at UFH was the most viable option, for several reasons. Firstly the ease of accessibility of the data was quite crucial, apart from the network security concerns raised by the Systems Administrator; TCP/IP network traffic data was readily available for collection. Secondly we needed a domain in which the task carried out as well as the representations appropriate to the domain would assist in the development of alternate network architectures appropriate for use in our investigations. We also needed a domain which would permit more than one quality or level of information content of the data to allow several investigations to be conducted. TCP/IP network traffic forecasting satisfied both these requirements, hence it was chosen as the problem domain for conducting this research. To date network Administrators at University of Fort Hare (UFH) have had to struggle with a great deal of unexpected traffic congestion. They have had to simply rely on intuition to predict traffic congestion. Through the efficient forecasting of TCP/IP network traffic, the task of traffic engineering at UFH could be handled in a more economical manner.

4.2 The software

The software used for the purposes of this research is Matlab Version 7.4.0.287 (R2007a). Matlab is an application software and programming language with interfaces to Java, C/C++ and FORTRAN. It has many toolboxes which are dedicated to areas such as aerospace technology and bioinformatics. In this study, Matlab provides an environment for creating programs with built-in functions for performance metrics and forecasting using its Neural Networks toolbox Version 5.0.2 (R2007a). The toolbox provides comprehensive support for many proven paradigms, as well as Graphical User Interfaces (GUIs) that enable one to design and manage ANNs. The modular, open and extensible design of the toolbox simplifies the creation of customized functions and networks. Some of its features include:

- Comprehensive set of training and learning functions.
- Modular network representation that enables an unlimited number of input setting layers, network interconnections and a graphical view of network training architectures.
- Pre-processing, post-processing functions and simulink blocks for improving ANN training and assessing ANN performance.

- Visualization functions and GUIs for viewing ANN performance and monitoring the training process.

The computer used to conduct this research is an Intel(R) Core(TM) 2CPU6300@1.86GHz. Some of the programs (MFiles) used in this study can be found in Appendix A. A summary of what each code is intended to achieve is also found in Appendix A. The codes were created to assist in the importation of the data into Matlab, and the pre/post processing of the time series. Some create the ANN inputs and their respective targets. There are codes used to achieve training. There are codes that deal specifically with evaluating the performance of the ANNs.

4.3 Data collection

Data collection is a significant part of designing an ANN. Kaastra and Boyd (1996) made the following observations on the factors that must be considered when collecting data for use in ANN training:

- Time and cost- The data must be collected within the shortest time possible and at a reasonable cost.
- Data significance – The data should comprise only those factors which might influence the predicted value the most.
- The source of data must be reputable and its databases must be constantly updated.

In line with these guidelines, the data for this research were collected from the South African Tertiary Institutions Network (TENET) website (www.TENET.ac.za). TENET is a major academic network backbone which utilizes the Multi Router Traffic Grapher (MRTG) software to monitor traffic load on network links from various institutions in South Africa. MRTG is a software that quantifies the traffic passing through every network interface with reasonable accuracy. It is widely deployed by every Internet Service Provider (ISP) in South Africa, such that the collection of these data did not induce any extra traffic on the network. MRTG also generates HTML pages containing PNG images which provide a live visual representation of network traffic. Snapshots of this software are available in Appendix B. This study analysed network traffic data which comprised inbound data in (bits/ sec) from the University of Fort Hare VC Alice Boardroom 101 – Fa 0/1 router. The data spanned from the 1st of March 2010 from 02:00 hours to the 21st of September 2012 02:00 hours in daily intervals, equating to 730 observations.

Matlab Code (see appendix A) for computing the mean, standard deviation and the median of the time series data which are vital statistical functions were written and are available in Appendix A. Table 4.1 shows the respective values thereof.

Table 4.1: Annotations.

Annotation	**Value**
Mean	9.9e+0.03
Standard Deviation	3.3e+0.06
Median	9.5e+0.05

The data were downloaded as Comma Separated Values (CSV) format and subsequently converted to MS Excel format which is compatible with the Matlab software. Code (see Appendix A) was written to import the data into Matlab from MS Excel.

4.4 Data pre-processing

As in most ANN applications, data pre-processing is a crucial step for achieving a good prediction performance when applying ANNs to forecasting tasks (Atiya & Ji 1997). The input and target variables are rarely fed into the ANN model in raw form. At the very least, the data must be scaled between the upper and lower bounds of the activation function (Shavlik et al. 1991). According to Kaastra and Boyd (1996), pre-processing the data before it is used in training the ANN is important for three reasons:

- Reducing the dimensionality of the data is crucial; high dimensional data is difficult to work with; more features introduce the likelihood of noise which prevents a good approximation of a function by the ANN. A reduction of data dimensions leads to an efficient pruning of the data space.
- It enables the relevant parts of the data to be identified and isolated from the irrelevant features so that they can be amplified.
- It improves the efficiency of ANN training, giving rise to faster training time and more precise solutions.

Due to the self similar and cyclical nature of TCP/IP network traffic, a number of pre-processing methods and algorithms have been adopted by various authors. Some of these methods include Fourier Series Analysis (Krzyovt 2008), Principal Component Analysis (Baldi & Hornik 1989)

and Tukey's 53H Procedure (Ahmad & Tesauro 1998). Following recommendations from Kaastra and Boyd (1996), who have done extensive work in designing ANN models for forecasting purposes, we performed the following pre- and post- processing techniques: Linear interpolation, identification of seasonal components and Normalization of the data.

4.4.1 Linear interpolation

As in all practical applications the data suffered from several deficiencies which needed to be remedied before training. During the transmission of data through an unreliable network such as the internet, packets can be corrupted or prevented from reaching their destinations, causing many problems. Most notable is the issue of missing or null values, which in our dataset amounted to 7 such recordings. Various heuristics have been proposed for dealing with missing data in regression and forecasting problems. Naturally, it is common simply to discard those missing patterns from the dataset, however in our case this approach would have been highly precarious considering that we had roughly 100 weeks of observations, which by normal standards is a slightly limited amount of data to train an ANN (Nguyen & Chan 2004). Discarding any values would have effectively modified the data distribution which would have had a detrimental effect on training. It was necessary therefore to make full use of the information which was potentially available from the incomplete values. It is also worth mentioning that within this domain it is difficult to collect more than $^2/_3$ months of data, since network servers often reboot or pass through update maintenance changes.

Another approach to dealing with missing data suggested by Hong et al. (2010) is to fill in the missing values first and then train the ANN using some standard method. Each missing value is typically replaced by the mean of the corresponding variable over those patterns for which its value is available. Such an approach can however lead to poor results as reported by Nakamura (2005) and Tan et al. (2012).

A more elaborate technique favoured by the majority of authors is Linear interpolation (Liyong et al. 2007; Wagner & Gross 1996). According to David and MacKay (2007) Linear interpolation is fairly robust against departures from the normality assumption. It expresses any variable which has missing values in terms of a regression over the other variables using the

available data. The regression function is then used to fill in the missing values. Linear interpolation is defined by the overall formula:

$$y = y_1 - \left(\frac{(x_1 - x)(y_1 - y_2)}{x - x_2}\right) \qquad (4.1)$$

where y is the the y co-ordinate of the data point on a straight line, and x, is the x co-ordinate of a point on straight line. We performed Linear interpolation on the data to fill in the missing or null values. Code for this procedure is available in Appendix A.

4.4.2 Identification of periodic components

In most instances TCP/IP traffic data comprise two periodic components, namely seasonal and cyclical periods (Zhang & Kline 2007). The seasonal period, often referred to as seasonality, is a characteristic of a time series in which the data experience regular and predictable changes recurring every calendar year. Seasonal effects are different from cyclical effects which can span time periods shorter or longer than one calendar year (Zhang & Kline 2007). According to Cortez et al. (2010) an effective method of identifying periodic components in a dataset is through a simple plot of the time series. A plot of the time series (see Appendix C) was done in order to get a visual indication of any seasonal effects, trends, changes in variance, and to enable the detection of outliers. The code used for plotting the time series is available in Appendix A. Visual inspection of the plot indicates a highly seasonal time series, with visible monthly, weekly and daily trends. Evidently there is also a yearly pattern that is difficult to model due to lack of sufficient data. The time series also has some apparent outliers: unordinary traffic behaviour was observed in the last 1.5 weeks of the year (i.e. time observation 250). This could well be associated with the effect of the Christmas and the New Year's Eve holidays with few students being resident on campus. Also the last two weeks of the year (i.e. time observation 247-252) show aberrant behaviour. Also apparent is a progressive increase in network traffic over the period in question, although the trend is not that linearly visible. To get rid of linearity and to enhance smoothness a constant de-trend was run on the time series. Matlab code for this process is also available in Appendix A.

4.4.3 Network inputs and targets

One of the most challenging tasks in developing a forecasting model is to determine appropriate model inputs beyond which values of the input time series have no significant effect on the output time series (Lange et al. 1997). For the ANN to be used as a tool for prediction, input-

output patterns should be constructed such that the ANN training reflects the relationship existing between input and output parameters. When developing models for forecasting, the order of the inputs can be determined using empirical and/or analytical approaches (Maier & Dandy 1998). In the empirical approach, *a priori* knowledge about the processes that generated the time series is used to determine the lags of the inputs. The aim of the analytical procedures is to determine the strength of the relationship between a series and itself at various lags. The lags that indicate strong correspondence between the series and itself can then be used as an indicator of the important inputs for the model (Maier & Dandy 1998). However, analytical approaches generally are not used to determine the inputs for ANN models. The main reason for this is that ANNs belong to the class of data-driven approaches, whereas conventional statistical methods are model-driven (Piotrowski & Napiorkowski 2013). In model-driven approaches, the structure of the model has to be determined first, which is done with the aid of the empirical or analytical approaches before the unknown model parameters can be estimated. Data-driven approaches, on the other hand, have the ability to determine which model inputs are critical, so there is no need for *a priori* rationalization about relationships between variables. However, presenting a large number of inputs to ANN models and relying on the network to determine the critical model inputs usually has a number of disadvantages, such as increasing training time, increasing the amount of data required to efficiently estimate the connection weights, and increasing the number of local minima in the error surface, which makes it more difficult to obtain the near-optimal combination of weights for the problem under consideration (Piotrowski & Napiorkowski 2013). Consequently, there are distinct advantages in using an analytical technique to help determine the inputs for ANN models.

Research suggests two different methodologies of selecting the ANN inputs and targets. The first is called the Haugh and Box method, which determines the inputs to ANN model using correlation analysis to determine the strength of the relationship between the input time series and the output time series at various lags (Maier & Dandy 1997). However, when this analysis is used, the time series needs to be pre-whitened to obtain the true correlation relationship (Zhang 2007). A second approach, which we adopted for this work uses a stepwise linear regression to identify the significant time lags in a time series and uses those as the input vectors (Zhang 2007). Under this setup a stepwise model is fitted to the data and the significant time lags are

used as inputs to the ANN. For each ANN model, a maximum lag (time window) k_{max} is chosen, and values at lags 1; 2; 3…… k_{max} are used as model inputs. If *a priori* knowledge about the relationship between the input and output time series is available, k_{max} is chosen so that the lags of the input time series that exceed k_{max} are not suspected to have any significant effect on the output time series. If no *a priori* knowledge is available, k_{max} has to be chosen empirically (Maier & Dandy 1998). A sliding time window k_{max} of size 150 was empirically chosen as the input for all the ANN models in this research. This value was selected based on a series of preliminary experiments conducted with Professor Mike Burton at Rhodes University using a similar time series dataset whereupon a single hidden layer ANN of 90 hidden neurons was trained through experimentally varying the size of the sliding time window. The sizes of sliding time window which were considered for examination are 20, 40, 60, 80, 100, 150, 170 and 200. For these experiments, the learning rate and momentum were set at zero, the size of the training set was set at 620 and the test set at 200 samples. The Logistic sigmoid and the Linear activation functions were used in the hidden and output layers respectively; whilst the Backpropagation Levenberg–Marquardt algorithm was the learning algorithm. Training was stopped after 1000 iterations, and a plot of the activations (predicted values) versus the targets (actual values) on the train and test sets was done. The best-performing sliding time window would result in the activations trailing the targets reasonably well in both the train and test sets. After extensive experiments a sliding time window of size 150 yielded the best results. Appendix F shows the results of those preliminary investigations. The code for conducting the training of the ANN is available in Appendix A. For this research, since we did not have any significant knowledge of the underlying process which generated the time series pattern, a supervising script (see Appendix A) was written which organized the ANN input patterns to be columns of a matrix $p_{r\times m}$ and the corresponding targets to be the columns of a matrix$t_{s\times m}$. The supervising script performs regression by varying the window length and keeps a record of the networks and their success, measured by correlation. Once it has done that it converts the network inputs from linear to cyclic. Each day number *dn,* in the time series was replaced by a suitable pair (*dnc, dns)* where

$$dnc = cos 2\boldsymbol{\pi} \frac{\boldsymbol{dn}}{\mathbf{365}} \qquad \textbf{(4.3)}$$

$$dns = sin 2\boldsymbol{\pi} \frac{\boldsymbol{dn}}{\mathbf{365}} \qquad \textbf{(4.4)}$$

thereby placing the days on a parameter of a circle whose circumference is of length 365 to ensure that the inputs become cyclic rather than linear for proper training of the ANNs. This approach has been used in literature with promising results (Uwahuro 2008).

4.4.4 Normalization

Due to the volatility of the time series under study, the measured values fluctuated sharply in a short period, and it was therefore necessary to scale the inputs and targets to a particular range. Normalization brings the range of data within the limits of the ANN (Sola & Sevilla 1997). In the Matlab environment it is advised that the data be normalized so that the ANN can perform significantly better (Hudson et al. 2014). Literature reports four methods of normalization (Ahmad & Tesauro 1998):

- Along channel normalization: A channel is defined as a set of elements in the same position over all input vectors in the training or test set. Each channel can be thought of as an independent input variable. The Along channel normalization is performed column by column if the input vectors are put into a matrix. In other words, it normalizes each input variable.
- Across channel normalization: This type of normalization is performed for each input vector independently, i.e., normalization is across all the elements in a data pattern.
- Mixed channel normalization: this method uses some kind of combinations of Along and Across normalization.
- External normalization: All the training data are normalized into a specific range. It is defined by the formula:

$$\boldsymbol{x_n = \frac{x_0}{x_{max}}} \qquad \textbf{(4.5)}$$

where x_n and x_0 represent the normalized and original data; x_{max} is the maximum devation along the columns or rows.

Before using the data for ANN training, External normalization of the ANN inputs and targets was done to keep the connection weights from becoming too large and swamping the network during training. Matlab Neural Network toolbox has a built-in function, *mapminmax* to carry out this operation. It automatically scales the data down before processing so that it has zero mean and unity standard deviation and then scales it up again after processing so as to produce outputs with zero mean and unity standard deviation.

The partitioning of the time series data into training and testing sets is typically done at this stage. The columns of the input pattern matrix, *p*, and the target matrix, *t*, need to be partitioned into training set: *ptrain, ttrain*; and test sets: *ptest, ttest*. The training set error is used for computing the gradient and updating the network weights and biases, while the test set is used to verify the prediction performance. The test error is representative of the error which we can expect from absolutely new data for the same problem (David & MacKay 2007). However, in certain circumstances, usually where the dataset is very large a third set called the validation set is used. The validation set comprises of samples used to find the best ANN configuration and training parameters, however in many practical situations, the available data are small enough to be solely devoted to model training and testing, and therefore collecting any more data for validation is difficult. The decision to partition the data is normally left to Matlab. The default function for partitioning the data in Matlab is *dividerand*, which randomly partitions the data into 60% for training, 20% for testing and 20% for validation. This could however be changed by specifying the index data division function, *dividend.*

In an attempt to find the optimal proportion of the data to use for training, testing and validation, Atiya & Ji (1997) investigated the impact of the proportion of data used in various subsets on ANN performance for a case study of settlement prediction of shallow foundations and found that there is no clear relationship between the proportion of data for training, testing and validation and model performance. They did find, however, that the best result was obtained when 20% of the data were used for validation and the remaining data were divided into 70% for training and 10% for testing. Barry (2000) addresses the concept of sufficiency of data. He notes that the lack of data is more often the limiting factor. Given this potential dearth of data, he recommends splitting the data in the following proportions: 2 for the training set and 1 for the testing set. In the majority of ANN applications in TCP/IP network traffic forecasting, the data are divided into their subsets on an arbitrary basis. However, recent studies have found that the way the data are divided can have a significant impact on the results obtained. Chabaa (2010) in predicting and forecasting internet traffic used a ratio of train to test sets of 3:1 with reasonable success. Piotrowski and Napiorkowski (2013) in the same task used a similar ratio and report that their networks did not suffer from any overfitting. Following similar convention as Piotrowski

and Napiorkowski (2013) we allocated 75% of the data to the training and 25% to the test set (unless stated otherwise). Generally, there were two ways to treat the training versus the testing data. First, they could both have been in a larger data set from which the training and testing exemplars were drawn (Zhou et al. 2006). Alternatively, the study could have used distinct training and testing sets (Hagan & Menhaj 1994). For purposes of simplicity we chose the former approach. Before training, the data was randomly split into train and test sets to avoid the training results becoming biased towards a particular section of the database. Matlab codes for the normalization and partitioning of the data are available in Appendix A.

4.5 Artificial Neural Network training program

Training an ANN to learn patterns in the data plays a major role in the successful application of ANNs. In section 2.5, learning in terms of ANNs is defined as *"a process by which the free parameters of a neural network are adapted through a continuing process of stimulation by the environment in which the network is embedded.*" (Balestrassi et al. 2009). Training can therefore be defined as the "*procedure by which the ANN learns"* (Shavlik et al. 1991). The major objective of training an ANN is to find the set of weights between the neurons that determine the global minimum of the error function (Kaastra & Boyd 1996). During training, the ANN goes through an iterative process, each of the connection weights, *w*, and biases, *b*, of the network is adjusted and the most appropriate set of weights is found to represent the model. The ANN training program has to go through a series of steps in order to adequately train an ANN. Firstly the program begins by initializing the weights connecting both the input layer and hidden layer and the hidden layer and output layer. Next an example problem from the training data is presented to the ANN. Each training example presented to the ANN may be represented by a certain subset of the total set of functions available to the ANN. The program then computes the activation of the hidden layer and output layers, after which the activations of the output layer are compared to the desired (known) output and the appropriate weights updated. After all the training vectors have been presented to the ANN, the weights are held constant and the validation set (if present) is presented to the ANN (Haykin 1999). The error rate of the validation set (in conjunction with the error rate of the training set) is used as an indicator of the performance of the ANN. Each presentation of the set of training vectors is defined as an epoch (Engelbreetch 2007). By collecting the error rate for the entire test set at the end of each training epoch, the program constructs an error curve and a minimum error is observed somewhere

along the curve. During the training phase, the difference between the guess made by the ANN and the correct value for the output is assessed by the program, and the weights are changed in order to minimize the error. The program checks if an acceptable error level has been attained, and when the acceptable error has been achieved the program stops training and presents the test set for evaluating unseen examples during training (Engelbreetch 2007). This process is repeated over an entire set of training data. For a flow chart description of the ANN training program which was followed during the training of the ANNs in this research, see Figure 4.1**.**

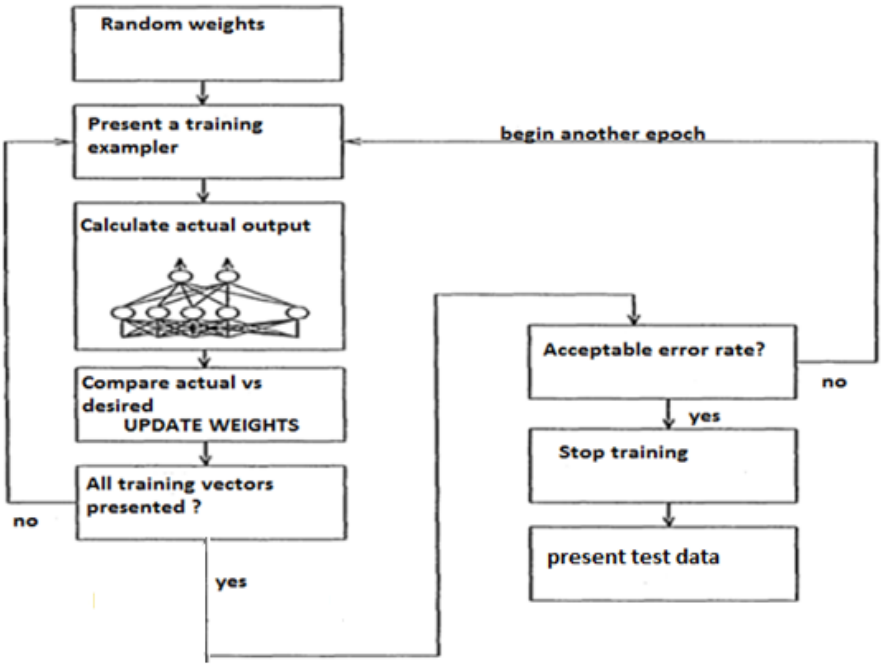


Figure 4.1: Training program for an ANN flowchart (Mi et al. 2005).

Training for the ANNs in our research was carried out by invoking the *train* function in the Matlab Neural Network software. The structure of the overall code for the training of ANNs in this research is available in Appendix A. The code

- Loads and organizes the data in the Neural Network toolbox.
- Partitions the data into training and test sets.
- Automatically scales the data down before processing and scales it up afterwards.

- Has a supervising script which organizes the network input patterns to be columns of a matrix $p_{r\times m}$ and the corresponding targets to be the columns of a matrix $t_{s\times m}$ and converts them from linear to cyclic.
- Configures an ANN.
- Trains the ANN.
- Simulates the ANN performance on training data.
- Tests the performance of the ANN on unseen test data.

Available in Appendix D is a snapshot of an ANN during one of the many training sessions in the Neural Network toolbox Version 5.0.2 (R2007a).

4.5.1 Experimental strategies and procedures

The five independent variables whose relationship to each other as well as to the generalization exhibited by an ANN this research studies are: size of network architecture, size of training set, size of learning rate, learning algorithm and training schedule. These variables are not all equally easy to manipulate. It was therefore important that before conducting our investigations and experiments we had to make certain strategic and tactical decisions. In our approach for the investigations, we used the experimental method which is a proven paradigm for testing and exploring cause and effect relationships. The experimental method is a systematic and scientific approach to research in which the researcher manipulates one or more variables, controls and measures any change in other variables. An experiment is typically carried out by manipulating a variable, called the independent variable, affecting the experimental group. The effect that the researcher is interested in, the dependent variable(s), is measured. Identifying and controlling non-experimental factors which the researcher does not want to influence the effects is crucial to drawing a valid conclusion. The benefit of using this method is that it allows the control of variables thereby enabling the isolation of a particular variable to observe effects on other variables. This further acts as a light in the tunnel that guides us towards making conclusions about cause and effects of variables. Moore and McCabe (1993) describe this approach as the best method of establishing causation in which the effect of possible lurking variables is controlled. Also, Gay (1992) states, "*The experimental method is the only method of research that can truly test hypotheses concerning cause-and-effect relationships. To experiment means to actively change x and to observe the response in y*".

We resolved to conduct our investigations in five separate training sessions, whereupon each training session comprised several experiments. This was so as to ensure that each of the five independent variables that this research identified could be adequately and independently investigated. The first training session investigated the effect of the size of the network architecture on the ability of ANNs to generalize to new data. The second training session investigated the generalization ability of ANNs as a function of the amount of training data. The third training session investigated the effect of learning algorithm choice on the generalization ability of ANNs. The fourth training session investigated the generalization ability of ANNs on different learning rates. The fifth training session explored the most appropriate training schedule for maximum generalization ability of ANNs. For compatibility of results, in all experiments an input layer with a single neuron, and 150 inputs corresponding to the maximum length of our sliding time window, and an output layer of a single neuron, automatically determined by the Matlab software were used. Since the number of hidden neurons and layers can significantly affect the training time and performance of ANNs, the general practice in the literature of having one hidden layer was favoured for this research (Bebis & Georgiopoulos, 1997; Teixeira & Fernandes, 2012). The decision of how many neurons to place on the hidden layer was more or less arbitrary. The Logistic sigmoid due to its real valued and continuously differentiable properties and the Linear activation function due to its ability to output continuous target values were used in the hidden and output layers respectively of each of the ANNs in this research. The initial random choice of weights can affect the performance of Backpropagation algorithm, hence in all the experiments weights were randomly initialized between [-0.5, 0.5]. The Backpropagation Levenberg–Marquardt algorithm (trainlm) was the standard learning algorithm throughout the duration of our investigations (unless stated otherwise).

As already mentioned in sub-section 4.15, prior to performing the experiments, the data were partitioned into training and testing sets. We designated 547 of the 730 samples for training and 182 samples for the test set. Reports from Zhang et al. (1998) suggest that a training set size which is 75% of the total data is sufficiently large to avoid overfitting. Following this guideline the results for 547 training samples would provide a legitimate standard comparison for different generalization results without overfitting the data as this number safely matches Zhang et al's (1998) recommendations. The training and testing sets were presented in a random fashion to the

networks to avoid possible bias in the presentation order of the sample patterns to the ANNs. The software package enables the connection weights to be saved at pre-determined intervals (i.e. at particular learn counts). Consequently, the training and testing samples were presented to the networks and progress of training procedure checked after every 50 epochs. The current literature does not provide a consensus for the optimal values of the learning rate, hence we had to rely on the experience of adept researchers such as Maier and Dandy (1997) who carried out a similar case study, where they assessed the effect of different internal parameters and network geometries on network performance in forecasting salinity in Murray River. They found that the size of the steps taken in weight space during training had a significant effect on learning speed. A learning rate of 0.1 in conjunction with a momentum value of 0.2 was found to give good results, and hence was used for all their models. In our research we adopted a similar approach and a learning rate of 0.1 and momentum of 0.2 was used in all our models (unless stated otherwise). In fact in most experiments to avoid local mimina problems we avoided completely the use of the momentum and in those experiments where the momentum was used we were careful that it did not exceed a value of 0.2, which is deemed to be fairly safe against overshooting of global minima (Jacobs 1998; Maier & Dandy 1997). With the exception of experiments where the learning rate was the variable of concern, training for all experiments was accomplished with a limited manipulation of the learning rate. Training was stopped whenever one of the following stopping criteria was satisfied: either the performance error goal was achieved or the maximum allowable number of training epochs was met. The number of training epochs did not exceed 1000, neither was training allowed to go beyond a performance error goal of 0.001. To reduce statistical fluctuations and to negate the effects of random weights, we ran each experiment over two training and testing sessions and averaged the results. Most importantly, for each of the five independent variables that we sought to investigate, we ensured that during experimentation all other variables that could potentially influence the results remained constant. One parameter was varied and all other parameters were kept constant in order to assess the effect due to changes in that parameter. For example during the investigation of the effects of the size of the network architecture on generalization ability, we ensured that the size of the training set, the learning rate, learning algorithm and the number of training iterations remained constant throughout the investigations. This was done so as to avoid any external factors which could potentially compromise the quality of results. A similar trend was

noted for the other four variables. The training procedure was conducted iteratively, covering in sample learning and out of sample testing. The experimental conditions outlined above were subject to change depending on the variable being investigated. We now turn to the subject of performance evaluation where we describe the evaluation criteria and measures which formed the bases on which we arrived at any given conclusions with regard to the main research objective.

4.6 Performance evaluation

A critical question to be answered is: how does one measure the generalization ability of an ANN? Testing the performance of an ANN on a test set is generally considered to be a global measure of the generalization ability of an ANN (Choudhury et al. 2012). The ANN is biased towards the training set, so the test set must be used to determine generalization (Fearn 2004). Given a limited amount of available data, it is important to hold aside a certain subset during the training process. After the ANN has been trained, the errors made by the trained network are computed on this test set. The test set errors will then give an indication of how the ANN will perform in the future; they are a measure of the generalization capability of the ANN (Walleed et al. 2009). According to Walleed et al. (2009), in order for the test set to be a valid indicator of generalization capability, there are two important things to keep in mind. First, the test set must never be used in any way to train the ANN, or even to select one network from a group of candidate ANNs. The test set should only be used after all training and selection is complete. Second, the test set must be representative of all situations for which the ANN will be used. Literature suggests two schools of thought regarding this issue. The first approach measures the ability of the ANN to generalize to new data from examples of a dataset previously encountered during training. This is the kind of test usually done to measure the ability of an ANN to exhibit good generalization to novel inputs from the same domain. The second approach measures the ability of an ANN to generalize to new data from examples of a dataset never before encountered during training. This is a test of the ANN's ability to generalize to novel inputs from a similar but different domain (Richards 1991). Whilst both of these viewpoints are valid in their own respect, for the purposes of this research we adopted the former approach, simply because we are of the opinion that analyzing the generalization performance of an ANN trained and tested on data from the same domain is far more easy to accomplish than on an ANN trained on one domain, and its performance tested on another domain, albeit similar.

The main aim of our analysis was to evaluate all the ANN models in our research and determine the model(s) that could simulate the data with highest generalization ability. The test set was removed from the data set before training began; it was then used at the completion of training to measure generalization capability by calculating the RMSE between the network activations (actual) and targets (predicted) values (see Appendix A for the Matlab code). RMSE is a frequently used measure of the difference between values predicted by a model and the values actually observed from the environment being modelled. These individual differences are also called residuals, and the RMSE serves to aggregate them into a single measure of predictive power. The RMSE is directly interpretable in terms of measurement units, and as such a better measure of performance than other criteria (Nakayama et al. 2004). It is defined by the overall formula:

$$\boldsymbol{RMSE = \sqrt{(\frac{1}{PK}\sqrt{(\sum_{p=1}^{P}\sum_{k=1}^{K}(t_{k,p} - o_{k,p})^2)})}} \qquad \textbf{(4.6)}$$

where P is the total number of training patterns or observations, K is the number of training outputs, $t_{k,p}$ is the target output for the k^{th} value of the p^{th} pattern, $o_{k,p}$ is the actual output generated by the k^{th} output for the p^{th} pattern. An ANN with good generalization ability should consistently show low RMSE values for the test set.

To further evaluate the model's performance, two other criteria were used. The Correlation statistic R was selected to measure the linear correlation between the predicted values and the actual values of the ANNs under consideration. Correlation indicates the strength and direction of a linear relationship between two variables (Cawley et al. 1993). A number of different correlation coefficients are used for different situations. The best known is the Pearson product-moment correlation coefficient (also called Pearson correlation coefficient), which is obtained by dividing the covariance of the two variables by the product of their standard deviations. It is given by the formula:

$$\boldsymbol{r_k = \frac{\sum_{t=1}^{p-k}(y_t - \overline{y})(y_{t+k} - \overline{y})}{\sum_{t=1}^{p}(y_t - \overline{y})}} \qquad \textbf{(4.7)}$$

where y_1, y_2, y_p stand for the time series and $\overline{y}$ for the series' average. The correlation is unity in the case of a perfect increasing linear relationship and -1 in case of a decreasing linear relationship, and the values in between indicate the degree of linear relationship between the predicted values and the actual values. A correlation coefficient of 0 means that there is no linear relationship between the variables. The square of the Pearson correlation coefficient (R^2), known as the coefficient of determination, describes how much of the variance between the two variables is described by the linear fit. The optimum R^2 value is unity. In this research an R and R^2 value on the train or test set smaller than 0.7 was assumed to be problematic. Codes for computing the R and R^2 are available in Appendix A.

While the ANN trains, the figure window for the Neural Network Training Tool (*nntraintool*) pops up. It provides information on how well the ANN is performing. The Performance Plot shows the RMSE on the training set as training progresses. The slope of the train curve can be a useful indicator of effects of overfitting. We observed the Performance plot as training progressed. If the curve went flat for many hundreds of epochs it was likely that the ANN was being trained with little improvement in performance. This may well indicate that the ANN may have been overfitting the data and therefore training had to be stopped at that point. The position at which the flatness started was noted and training was commenced just beyond that point. A snapshot of a Performance plot during training of one of the ANN models investigated in this research is available in appendix E.

4.7 Conclusion

This chapter has presented an extensive discussion of the methods and data techniques used in the study. The problem domain in which our investigations were conducted was discussed, along with the data collection and pre/post processing techniques employed. The hardware and software used in this research, and the training methodology for the ANNs were also presented. The experimental setup and procedures for the various experiments conducted in this study were clearly articulated. The methodological approach for the research, which is the Experimental method was also detailed. The chapter ended by presenting the performance evaluation criteria used to judge the performance of the ANNs. This chapter has described a detailed methodology of the development of a STWNN -based prediction model for investigating the generalization

ability of ANNs in forecasting TCP/IP network traffic trends. Chapter 5 presents the results of the experiments conducted, and the relevant discussions and findings.

Chapter 5: Results and Discussion

This chapter presents and discusses the results from experiments described in Chapter IV. The Chapter is divided into 3 sections, section 5.1 presents the results of the various experiments conducted, 5.2 interprets and analyses those results and 5.3 concludes the chapter. By presenting, interpreting and analysing a series of graphs from the experiments the chapter explores how the generalization ability of an ANN (empirically computed using RMSE on the test set) is affected by size of network geometry and a host of several design parameters in forecasting TCP/IP network traffic trends. Also of interest is the performance of the ANNs in terms of R and R^2 on the train and test sets. To put the results in proper context, the initial ANN settings and experimental conditions are first enumerated.

The experiments were conducted according to the following three steps:

- *Informal preliminary tests- experimental search for optimal network design. The goal in this step was to select a set of ANNs for further detailed, repetitive tests. Also several combinations of training parameters (learning rate, momentum, number of training cycles etc) were tested to make close to optimal the range of network parameters to be investigated in the various experiments.*
- *Detailed tests- the main part of the experiments. Pre-selected ANNs were trained with the appropriate parameters and their generalization ability tested.*
- *Additional repetitive tests- limited number of extra tests aimed at tracing out intrinsic limits of the models and to negate the random effects of network weights on results.*

While experimental results cannot cover all practical situations, our results do help to explain common behaviour which does not agree with some theoretical expectations. The chapter enables deductions to be made on some of the most critical components of designing ANN models.

5.1 The Results

In this section we present the results of the experiments conducted. Discussion and interpretation of the results follows in section 5.2. In order to maintain consistency and to avoid repetition the results of the preliminary experiments are not included in this thesis.

5.1.1 Training scenario 1: Network architecture

In the first training scenario, the focus was on investigating the effect of network architecture on the generalization ability of ANNs trained in forecasting TCP/IP network traffic trends. We sought to analyse the generalization ability of different network architectures and to study how this property changes with network size. Since literature has shown that the input and output layers of an ANN have negligible effect on the generalization ability of ANNs, the relationship between ANN generalization and network architecture was investigated by manipulating the hidden layer structures of an ANN trained on a fixed dataset. The aim was to determine the architectural design which would exhibit the best generalization ability for each of the network topologies examined. Richards (1991) states two basic guidelines for the design of ANN models:

- The overfitting guideline: If an ANN can learn a problem, then the fewer the number of free parameters in the network the better the ANN is likely to generalize.
- The dataset size guideline: The larger the number of free parameters in an ANN the more data needed to train the ANN.

Taking these guidelines into consideration and keeping with the *Universal Theorem of Approximation* (section 2.11), we saw fit to start off by allocating the smallest number of hidden units (neurons and layers) possible which would enable the ANN to learn the training set, and then incrementally add more hidden units as training progressed. This would enable us to determine the smallest possible network architecture that is able to capture adequately the relationship to be modelled with optimal generalization ability. Fixing the size of the training set at 547 samples and the test set at 182 samples, randomly setting network weights in the range [-0.5, 0.5], with the Logistic sigmoid and Linear activation functions in the hidden and output layers respectively, and the BP Levenberg- Marquardt (trainlm) algorithm as the weight update rule, ANNs with a single input and output neuron was trained on several network architecture configurations. The tested architectures ranged from 5 hidden neurons in a single hidden layer to architectures of 80 hidden neurons in 3 hidden layers. This range was selected for testing because

preliminary experiments conducted using fewer hidden neurons produced large RMSE (on both train and test sets), whilst architectures with more neurons in more layers was too computationally intensive for practical use. In order not to interfere with the training parameters the learning rate and momentum were kept at zero. Training for each experiment was conducted for 1000 iterations and the performance of the models tested on the independent test set of 182 samples. Each experiment was run twice, using different initial weights for training and testing and the results averaged. Following conventional guidelines from Maier and Dandy (1997), who performed similar investigations in forecasting Salinity on the Riverre river in Australia, we performed the following experiments:

5.1.1.1 Single hidden layer networks

The first set of experiments examined the architecture of an ANN with a single hidden layer. In this series of tests, the generalization ability of an ANN with a single hidden layer was analysed and investigated. We trained the ANN using different numbers of hidden neurons on separate occasions. From the results of the preliminary experiments the number of hidden neurons that were considered in this analysis are: 10, 20, 30, 40, 50, 60, 70, 80 and 90 respectively. The results for the experiments conducted are shown in Figure 5.1.

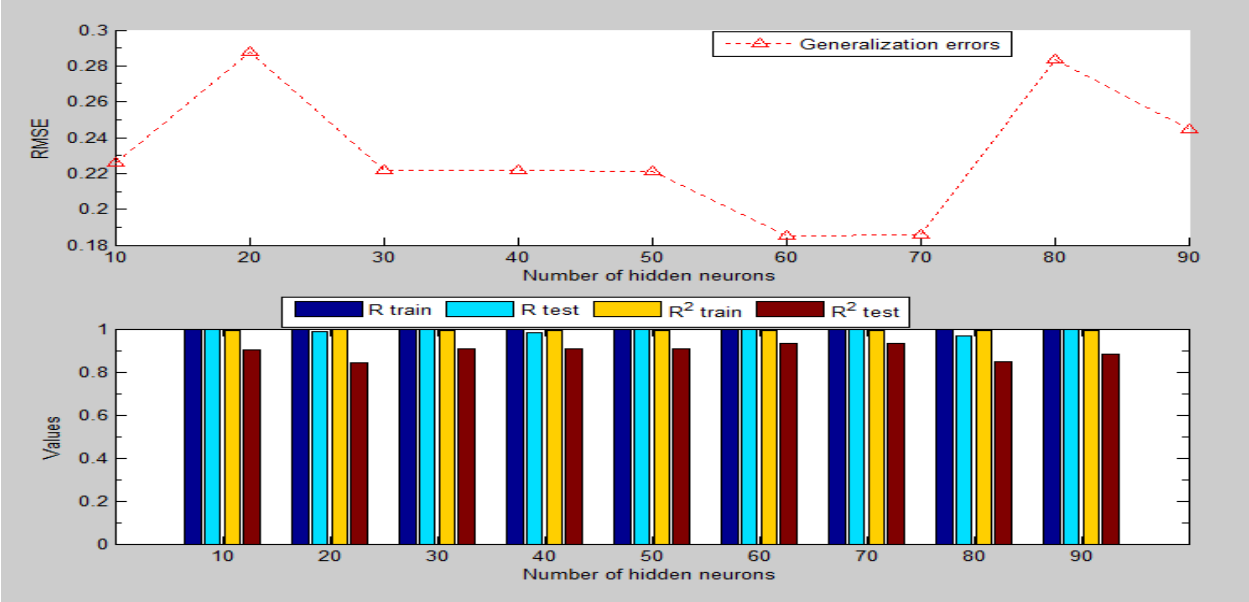


Figure 5.1: One hidden layer: Generalization errors in RMSE (top); R and R^2 for train and test sets (bottom).

5.1.1.2 Two Hidden layer networks

It has been suggested by certain authors that an ANN having many hidden layers is capable of generalizing better than an ANN having a single hidden layer (Kavzoglu 1999; Richards 1991). The rationale behind this school of thought is that the added layers provide the ANN with additional dimensions in which to reorganize the material to be learned. The relationship between ANN generalization and network architecture for multi-hidden layer models has never been adequately explored. This is doubtless due to the multitude of potential network architectures that can be specified once one opens the door to the possibility of having more than one hidden layer. A possible approach to this dilemma could be to fix the number of hidden neurons in a network and then distribute them equally over the multiple hidden layers, so that each layer has the same number of hidden neurons. Another approach could be to use different ratios of first to second hidden neurons. We explored each of these approaches.

5.1.1.2.1 Two hidden layers (equal number of first to second hidden neurons)

The second set of experiments explored the generalization ability of an ANN with two hidden layers. In this series of experiments the number of neurons was distributed equally over 2 hidden layers. From the preliminary experiments the following network architectures were considered for examination: (5:5), (10:10), (15:15), (20:20), (30:30), (35:35), (40:40) and (45:45). Where (*x: y*) denotes *x* hidden neurons in first hidden layer and *y* hidden neurons in second hidden layer. The results for the experiments conducted are shown in Figure 5.2.

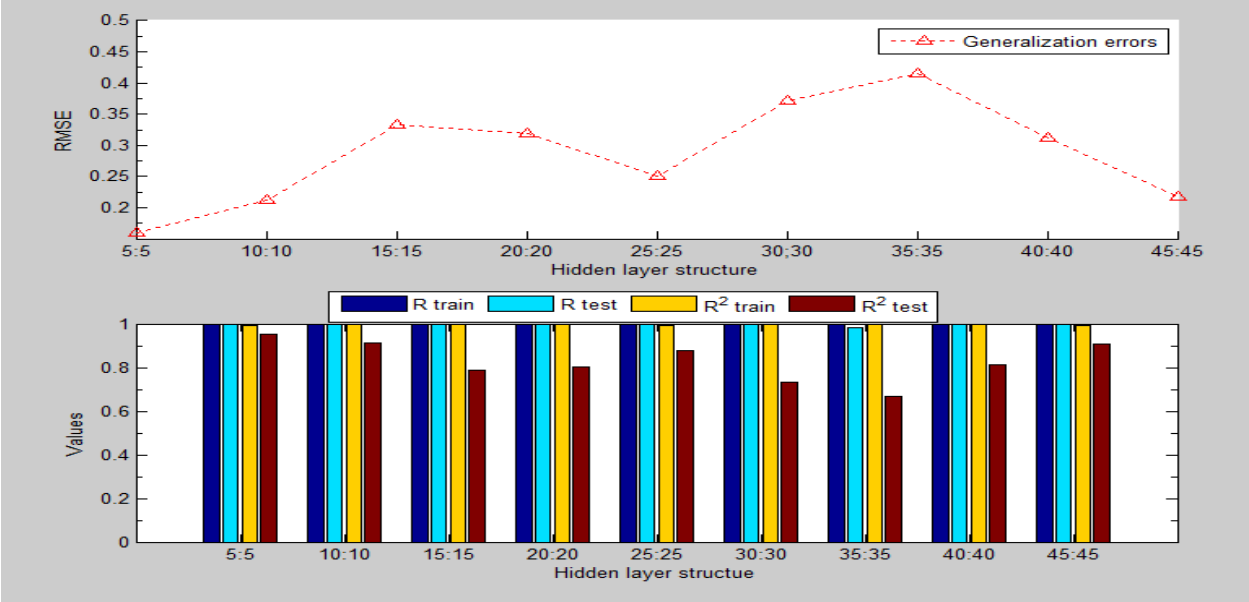


Figure 5.2: Results for different numbers of hidden layer structures: Generalization errors in RMSE (top); R and R^2 for train and test sets (bottom).

5.1.1.2.2 Two hidden layers (different ratios of first to second hidden neurons).

In this series of tests, the effect of using different ratios of first to second hidden layer neurons on the generalization ability of ANNs was assessed. Under this configuration, we conducted two sets of experiments. In the first set we kept the number of neurons in the second hidden layer constant at 5 neurons whilst we varied the number of neurons in the first hidden layer. Thus the network architectures explored were (5:5), (10:5), (15:5), (20:5), (25:5), (30:5), (35:5), (40:5), (45:5) and (50:5). The results for the experiments are shown in Figure 5.3.

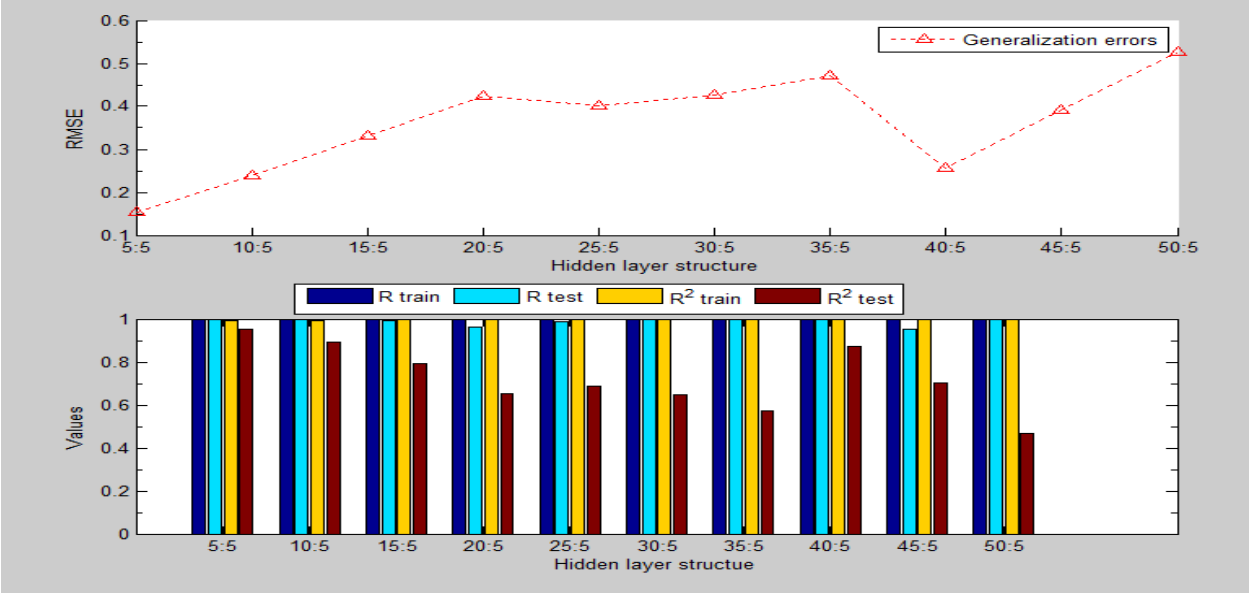


Figure 5.3: Results for different numbers of hidden layer structures: Generalization errors in RMSE (top); R and R^2 for train and test sets (bottom).

We then reversed the previous trend and kept the number of neurons in the first hidden layer constant at 5 neurons whilst we varied the number of neurons in the second hidden layer. The architectures explored were thus (5:5), (5:10), (5:15), (5:20), (5:25), (5: 30), (5:35), (5:40), (5:45) and (5:50). The results for the experiments are shown in Figure 5.4.

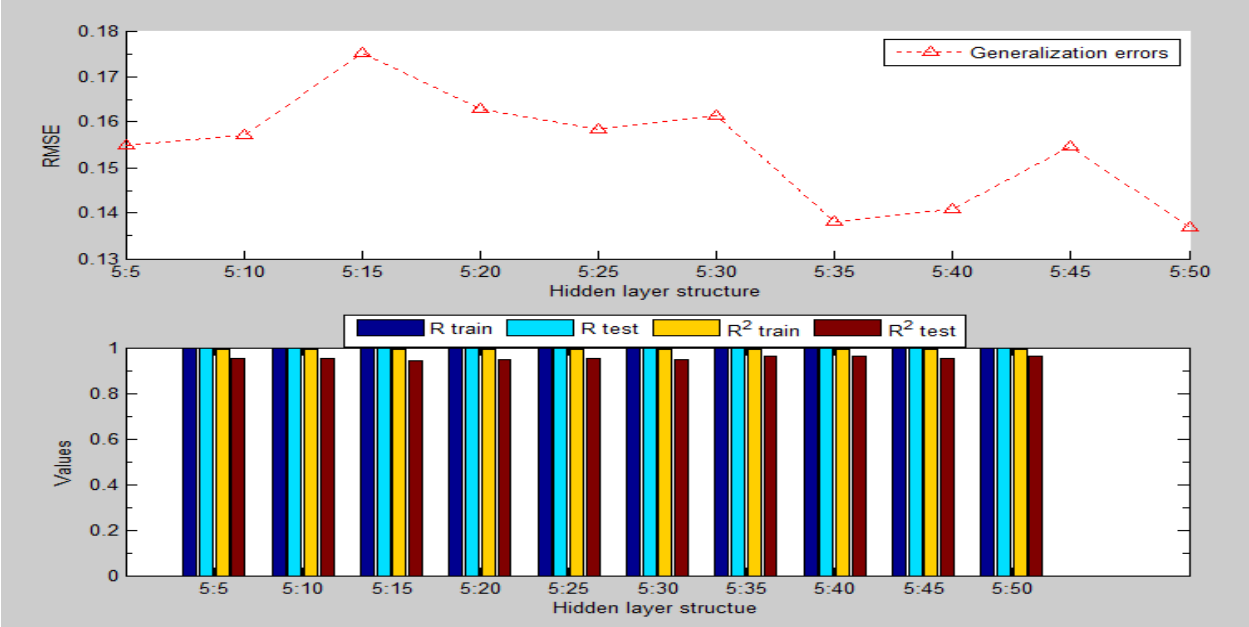


Figure 5.4: Results for different numbers of hidden layer structures: Generalization errors in RMSE (top); R and R^2 for train and test sets (bottom).

5.1.1.3 Three hidden layers

Although literature largely indicates no theoretical or practical basis for using a third hidden layer, out of curiosity we decided to examine how a third hidden layer could possibly impact on the performance of an ANN trained to predict trends in TCP/IP network traffic. This curiosity was brought about by results of a study by Philip (2001). His study reveals that an ANN with three hidden layers had generalization accuracy of 62.3% while the one with one hidden layer achieved generalization accuracy of 59.3%. Similarly Kwon (1991) reports that an ANN with three hidden layers reached generalization accuracy of 62.5%, while an ANN with a single hidden layer reached 62.7% generalization accuracy. In this series of experiments, we conducted a limited investigation into several three hidden layer architectures, in addition to that in order to get an overall picture of how the size of the network architecture affects the generalization ability of ANNs we compared the performance of 1, 2 and 3 hidden layer network architectures. We kept an equal ratio of first to second (in the case of 2 layers), and first to second to third (in the case of 3 layers) hidden layer neurons. From the results of our preliminary investigations it was decided to examine the following network architectures: (20), (40), (60), (80), (20*2), (40*2), (60*2), (80*2), (20*3), (40*3), (60*3) and (80*3). Here (*x** *y*) denotes *x* hidden neurons in y hidden layers. Figure 5.5 shows the results from the experiments.

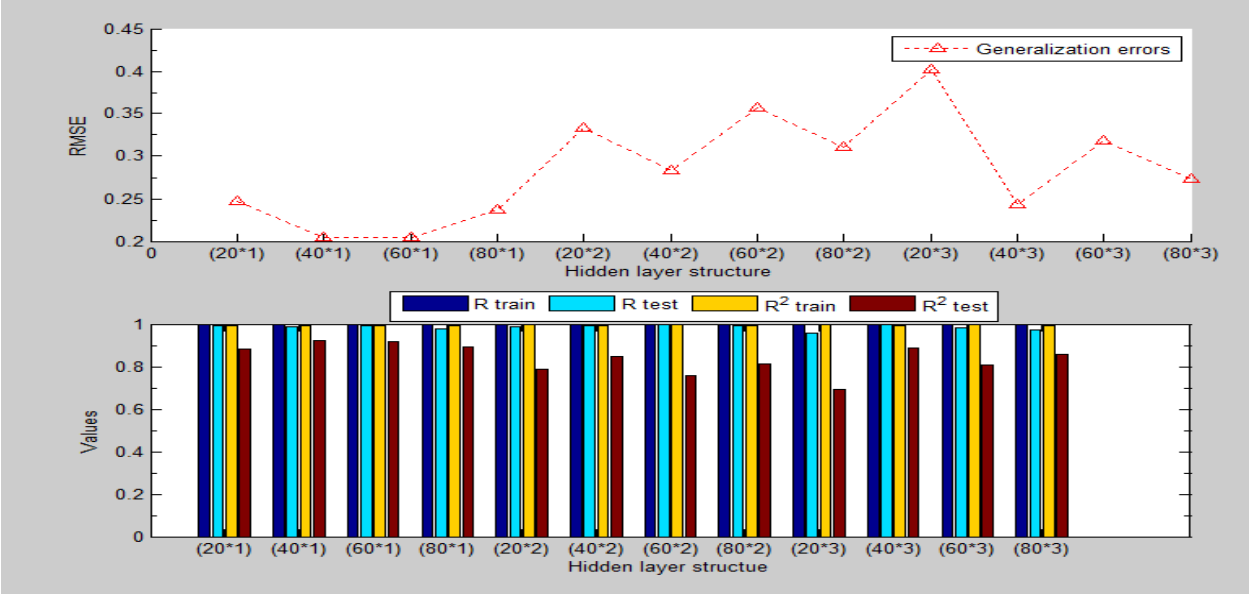


Figure 5.5: Results for different numbers of hidden layer structures: Generalization errors in RMSE (top); R and R^2 for train and test sets (bottom).

5.1.2 Training scenario 2: Training set size

Of 730 samples, 547 were allocated to the training set. If we could reduce the size of the training set and still achieve equivalent generalization ability, it would be possible to decrease the amount of time required to train an ANN. In the second training scenario we investigated the impact of training set size on the generalization ability of ANNs. The ANNs were trained to forecast TCP/IP network traffic trends either using the full training set of 547 samples or reduced variations of the training set. In order to conduct this investigation, unlike Kavzoglu (1999), who used two separate datasets, it was necessary to select a single dataset large enough to be separated into subsets. The reason behind this decision was that comparing the test results of two completely different datasets was likely to introduce the added complexity of determining whether the datasets themselves are in fact suitably comparable. Breaking the dataset down into subsets at least ensured that there was maximum compatibility within the dataset itself. Ten different training sets were generated as subsets chosen from the full training dataset. These subsets were created at 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% and 100% of the full training set of 547 samples as these were deemed appropriate sizes for subset selection. The choice of only ten subsets was made due to time constraints as testing a larger number of subsets was highly unlikely to produce more telling results.

Since general and purely experimental approaches to optimal network architecture can be extremely time consuming, our analysis was constrained to some *a priori* selected network architectures, hence the sizes of hidden layers and neurons in this investigation were limited to some predefined set of values. We conducted several sets of experiments where we examined the performance of both single and double hidden layer ANNs. During training the network weights were randomly initialized in the range [-0.5, 0.5], training for each experiment was halted after 1000 epochs thereafter the independent test set presented to the ANNs for assessing generalization ability. The Back-propagation Levenberg- Marquardt training rule was used to update the weights, the Logistic sigmoid and Linear activation functions were used in the hidden and output layers of the ANN models respectively. For each experiment, two training-testing runs per simulation were done to refine the accuracy of the results.

In the first phase of these investigations we examined a single hidden layer ANN of 60 neurons. We chose this network architecture because it exhibited the best generalization amongst all the architectures examined in the previous experiments conducted in sub-section (5.1.1.1). We trained the ANN using both the full training set of 547 samples and reduced variations of it on different runs. The results of the experiments are shown in Figure 5.6.

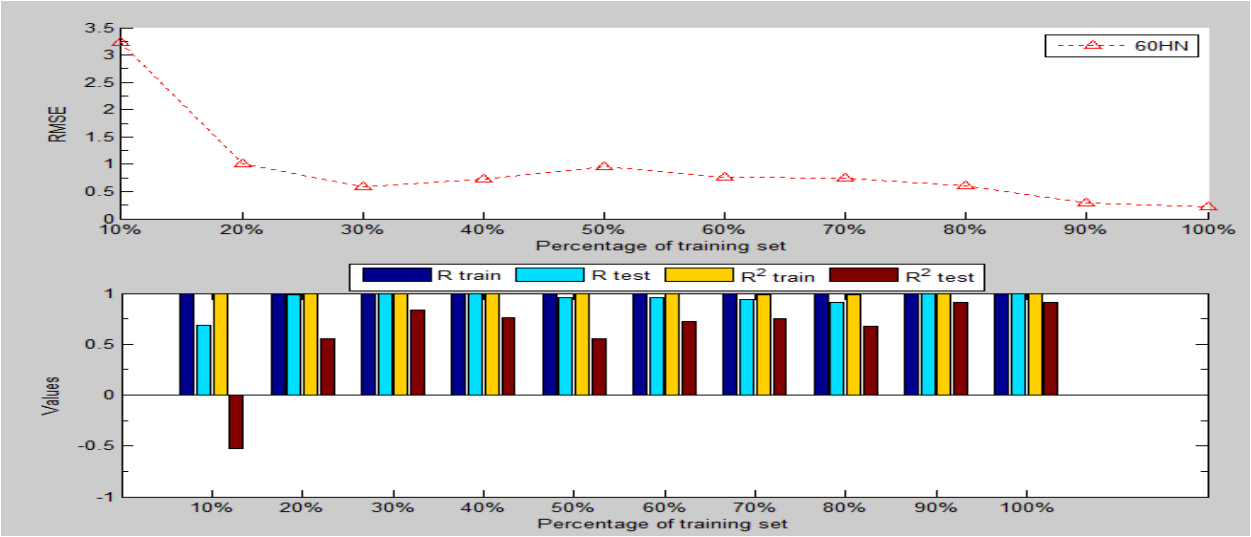


Figure 5.6: Results for different training set sizes for 60HN ANN: Generalization errors in RMSE (top); R and R^2 for train and test sets (bottom).

Secondly, we then examined the performance of a two hidden layer ANN of network architecture (5:35), *i.e.* 5 first hidden layer neurons and 35 second hidden layer neurons. As in the previous case, this network architecture exhibited substantially better generalization performance than the other two hidden layer architectures examined in sub-section (5.1.1.2). Furthermore, Maier and Dandy (1997) carried out a similar investigation using an almost similar network architecture of (8:30) with some measure of success. We trained the ANN using both the full training set of 547 samples and reduced variations of it on different runs. The results of the experiments are shown in Figure 5.7.

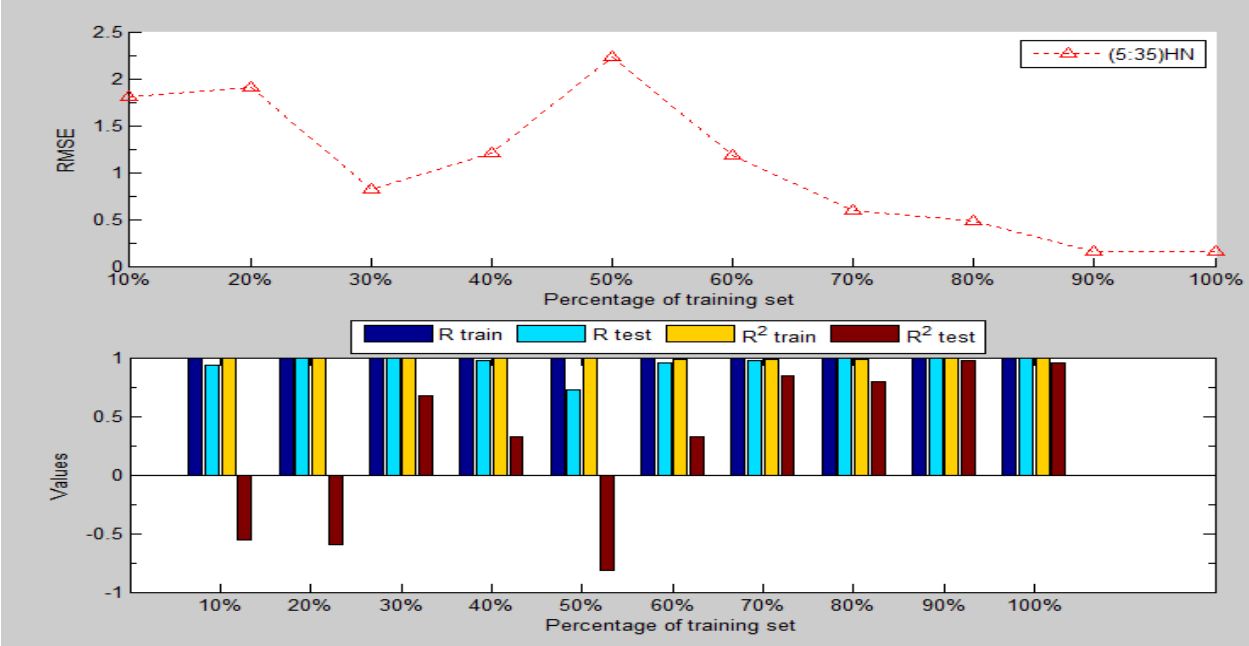


Figure 5.7: Results for different training set sizes for 5:35HN ANN: Generalization errors in RMSE (top); R and R^2 for train and test sets (bottom).

Finally, we performed a comparative analysis of some of the heuristics given in Table 3.4 (sub-section 3.3.2), with respect to their effectiveness and reliability towards determination of the optimum training set size for generalization ability of ANNs in forecasting TCP/IP network traffic. For these investigations we once again trained a single hidden layer ANN of 60 hidden neurons on different training set sizes as suggested by the different heuristics. The results of the investigation are shown in Figure 5.8.

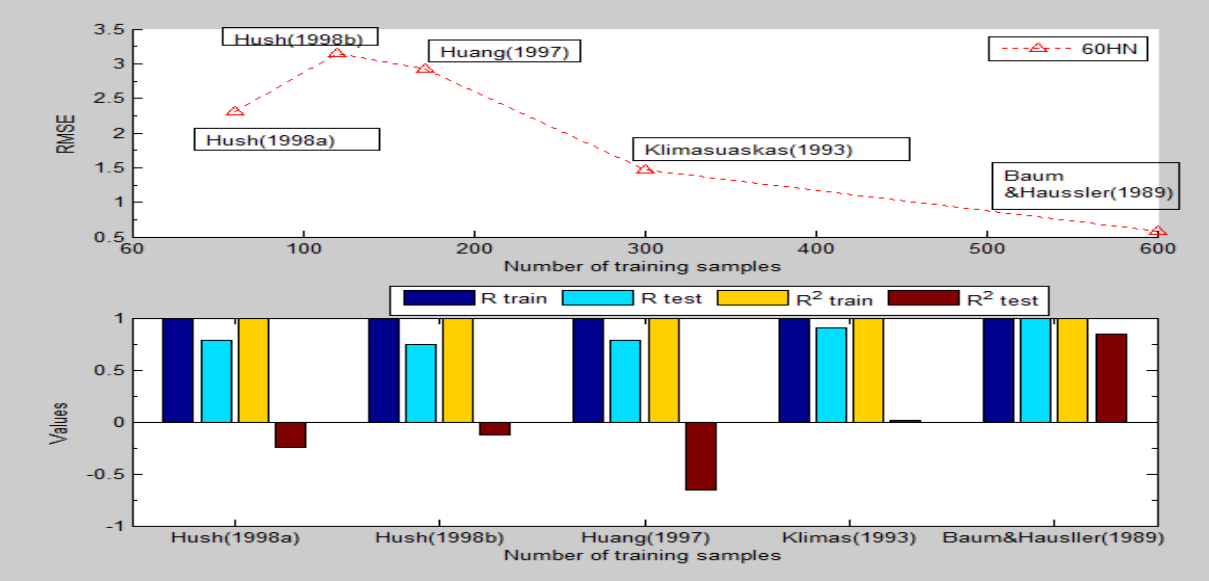


Figure 5.8: Results for different heuristics of training set size on 60HN ANN: Generalization errors in RMSE (top); R and R^2 for train and test sets (bottom).

5.1.3 Training scenario 3: Learning algorithm

There are many variations of the Backpropagation (BP) learning algorithm; some provide faster speed of convergence and better generalization ability, while others give smaller memory requirements. In the third training scenario, 5 BP learning algorithms were evaluated primarily in terms of their generalization ability, further to that a limited investigation was also conducted in terms of their relative speed of convergence and accuracy of forecasts. The Gradient descent (traincgb), Resilient backpropagation (trainrp), Conjugate Gradient (trainscg), Quasi-Newton (traincgf) and Levenberg-Marquardt (traingdm) algorithms were considered in the task of forecasting a TCP/IP network traffic time series.

In the first phase of the investigations, several ANNs with one input neuron, one hidden layer with a varying number of hidden neurons and one output layer were trained using the five different BP algorithms on separate occasions. The number of neurons in the hidden layers were arbitrarily varied as 20, 40, 60 and 80. The Logistic sigmoid and Linear activation functions were used in the hidden and output layers respectively. Throughout the duration of the experiments the training set was maintained at 547 samples, whilst the weights were randomly set between the range [-0.5, 0.5]. To make more precise comparison all the learning algorithms were trained for the same number of iterations (1000 epochs) and the learning parameters were

held constant during the training process. The learning rate for training was fixed at 0.1 and momentum at 0.4 throughout the training. The choice of learning rate and momentum were adopted from a similar empirical study conducted by Attoh-Okine (1999) in predicting inversion of pavement deflection where a learning rate of 0.1 and momentum of 0.4 yielded the best results. Training was terminated when the number of epochs reached 1000. The performance of the ANNs was then evaluated on the 182 independent test patterns. To negate the effects of random weights each training-testing run was repeated twice and the results of the different runs averaged. Figure 5.9 shows the generalization errors from the different algorithms and Table 5.1 shows the results of the R and R^2 on both train and test sets for the experiments conducted.

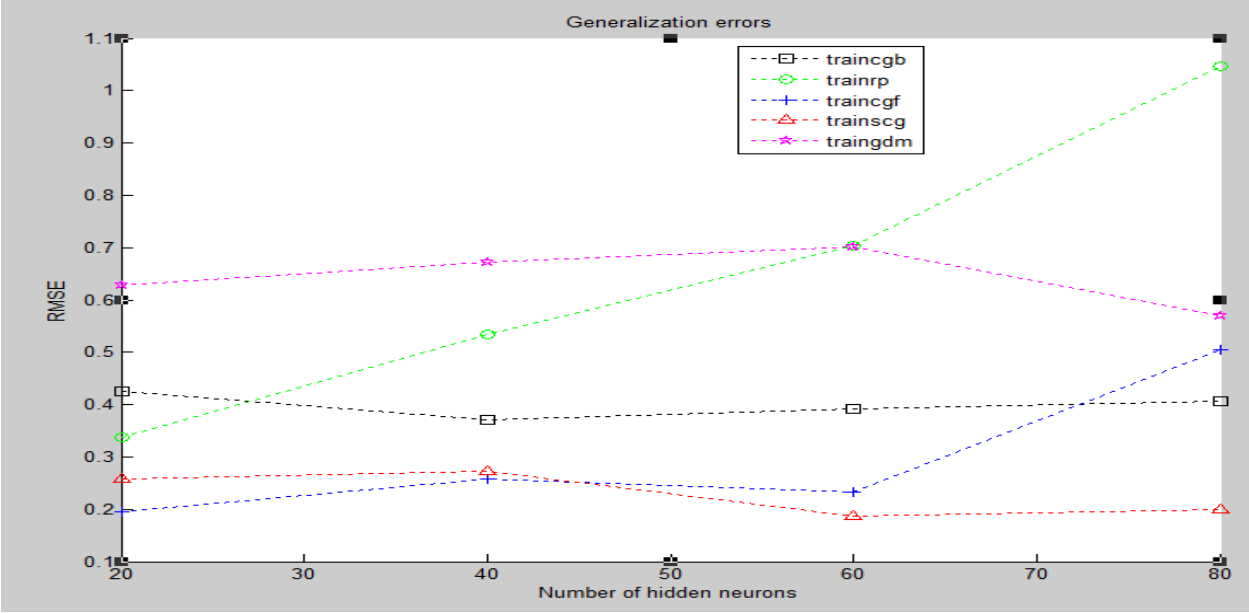


Figure 5.9: Generalization errors (RMSE) for the different BP algorithms.

Table 5.1: R and R^2 results for the different BP algorithms for train and test sets.

H.N	traincgp				trainrp			
	R train	R test	R2 train	R2 test	R train	R test	R2 train	R2 test
20	1.0000	0.9902	0.9961	0.6535	1.0000	0.9947	0.9949	0.8819
40	1.0000	0.9775	0.9970	0.7366	1.0000	0.9129	0.9936	0.8526
60	1.0000	0.9333	0.9957	0.7056	1.0000	0.9414	0.9960	0.8468
80	1.0000	0.9817	0.9967	0.8823	1.0000	0.8144	0.9968	0.9058
H.N	traincgf				trainscg			
	R train	R test	R2 train	R2 test	R train	R test	R2 train	R2 test
20	1.0000	0.9985	0.9939	0.9261	1.0000	0.9909	0.9951	0.8725
40	1.0000	0.9928	0.9951	0.8718	1.0000	0.9840	0.9934	0.8579
60	1.0000	1.0000	0.9948	0.8279	1.0000	0.9925	0.9905	0.9335
80	1.0000	0.9172	0.9956	0.8107	1.0000	0.9980	0.9924	0.9235
H.N	traingdm							
	R train	R test	R2 train	R2 test				
20	1.0000	0.9959	0.8282	0.2399				
40	1.0000	0.9943	0.8413	0.1290				
60	1.0000	0.9870	0.7807	0.0553				
80	1.0000	0.9961	0.8070	0.3775				

In the second phase of the investigations in order to compare the speed of convergence and accuracy of the different algorithms in terms of minimum Mean square error (MSE), a two hidden layer ANN with a network configuration of (5:35) and having the Logistic sigmoid and Linear activation functions in the hidden and output layers respectively, was trained using each of the five different BP algorithms on separate occasions. The speed of convergence and accuracy of ANNs is very important because in the real world, training a huge and complex set of data can take hours, even days, to attain results, hence studying the conditions which would result in decreased training time would be beneficial. MSE is defined as the square of the RMSE.

As in the previous experiments the learning rate was fixed at 0.1 and momentum at 0.4 throughout the duration of the training. The size of the training set was maintained at 547 samples and the weights were randomly initialized in the range [-0.5, 0.5]. Training was terminated when the network converged (i.e. every pattern learned) to a tolerance of 0.001 error goal or when the number of epochs reached 1000, whichever came first.

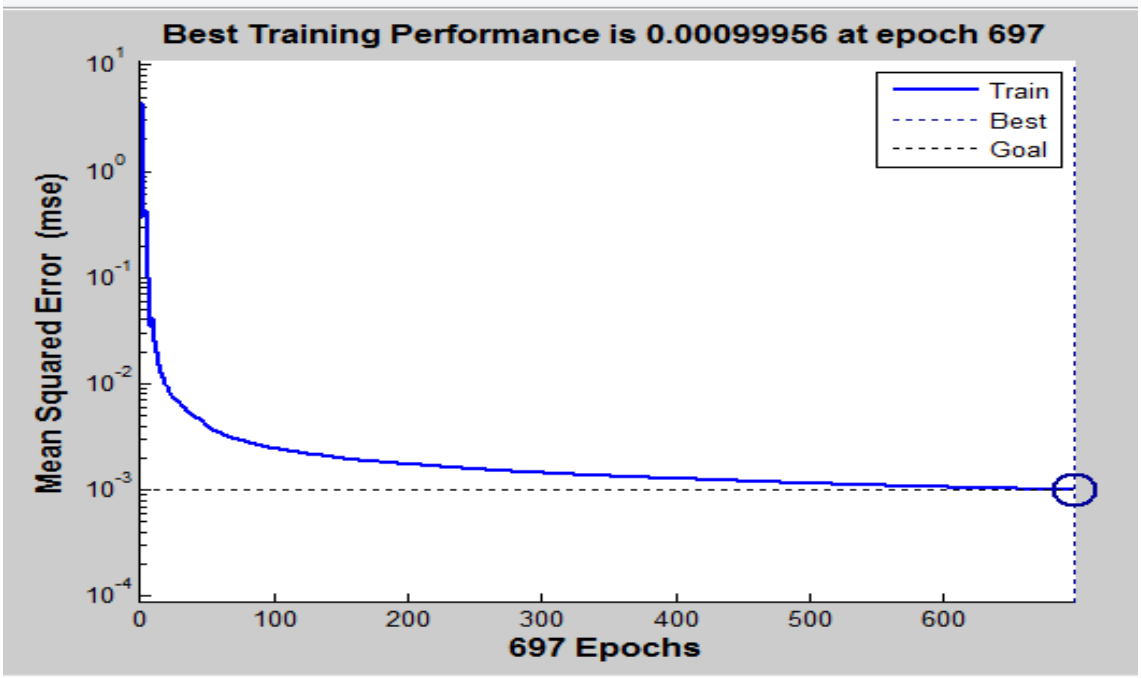


Figure 5.10: Results of traincgp algorithm.

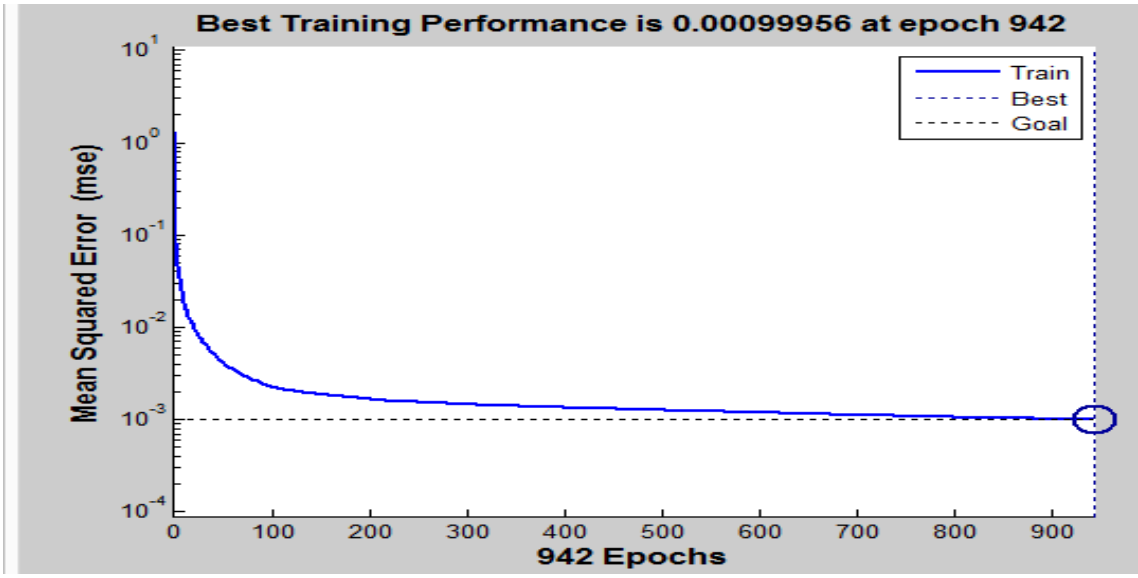


Figure 5.11: Results of trainrp algorithm.

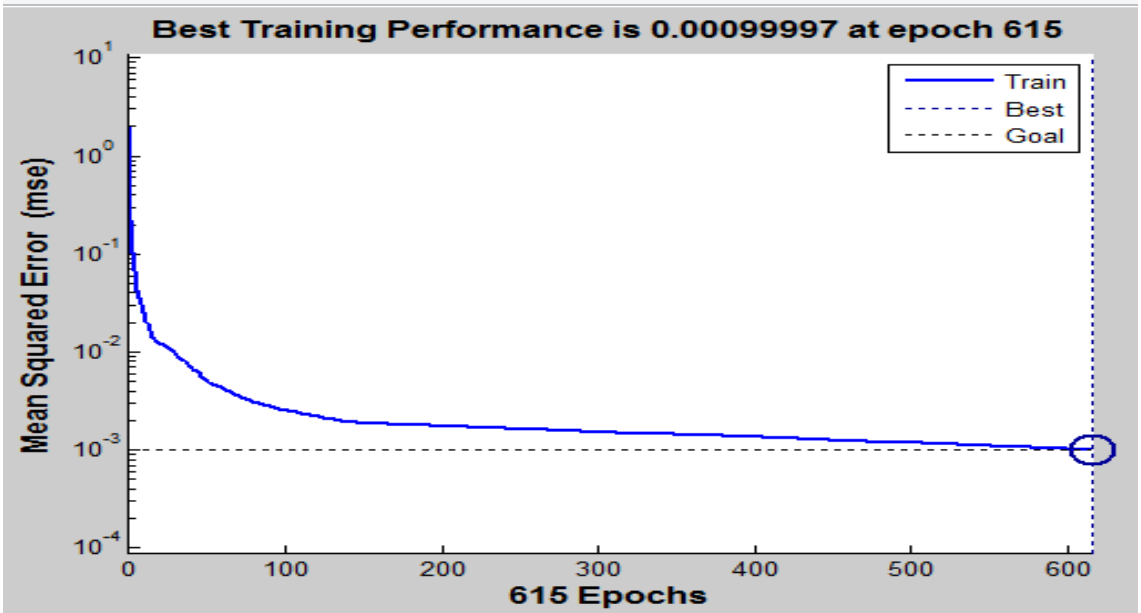


Figure 5.12: Results of traincgf algorithm.

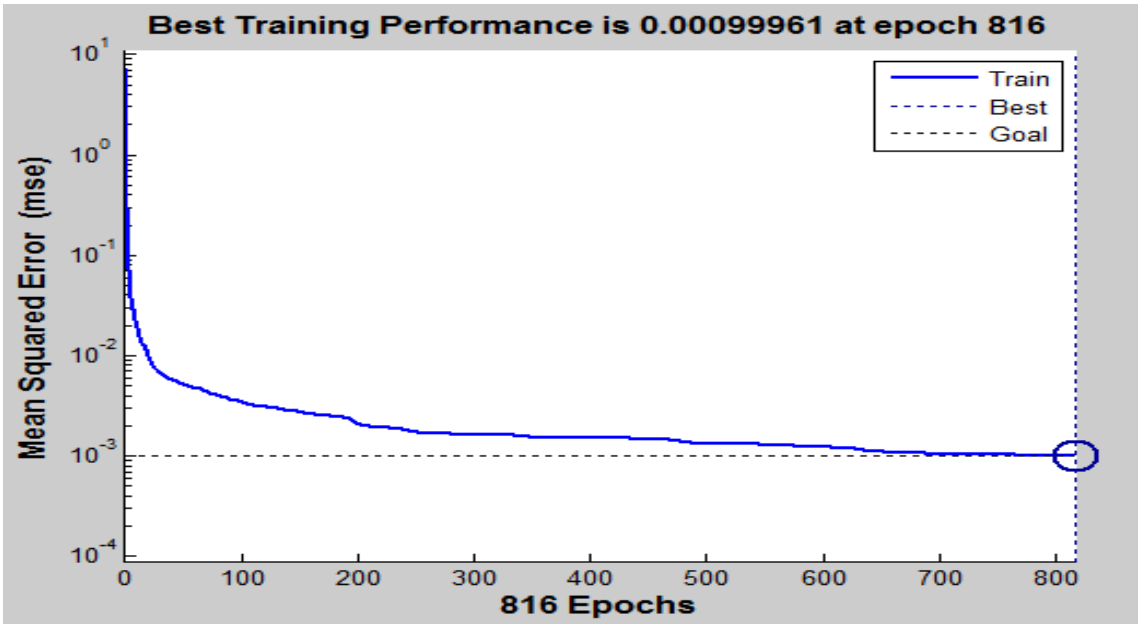


Figure 5.13: Results of trainscg algorithm.

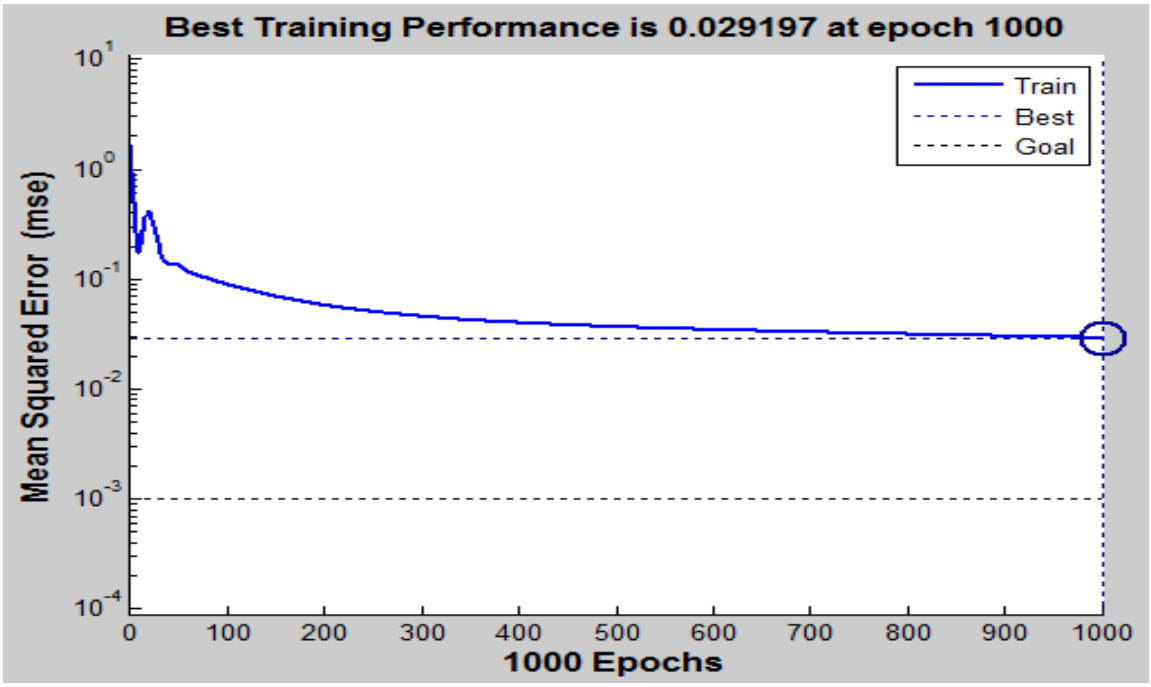


Figure 5.14: Results of traingdm algorithm.

5.1.4 Training scenario 4: Learning rate

To investigate the effect of learning rate on the generalization ability of ANNs in predicting TCP/IP network traffic trends, various layered, fully connected ANNs with a single input neuron, Logistic sigmoid activation function in the hidden layers and a Linear output neuron were examined. We investigated both single and double hidden layer ANNs. The architectures we chose exhibited the best generalization performance in section sub-sections 5.1.1.1 and 5.1.1.2.

The first set of experiments involved several single hidden layer ANNs trained on various learning rates on separate occasions. The ANNs selected for examination had 20, 40, 60 and 80 hidden neurons. These were selected based on the stability times during preliminary investigations. The learning rates were varied according to 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 and 1. The choice of the learning rates was purely done on an arbitrary basis. Fixing the size of the training set at 547 samples and of the test set at 182 samples, randomly initialising network weights in the range of [-0.5, 0.5], having the BP (trainlm) training rule as the weight updating algorithm and a fixed momentum of 0.2, we systematically trained the ANNs. Training was stopped after 1000 training cycles and the generalization ability of the ANNs tested on the independent test set of 182 samples. Figure 5.15 shows the outcome of the experiments in terms of the generalization errors incurred for each learning rate on various ANNs, and Table 5.2 shows the results of the R and R^2 on both train and test sets for the experiments conducted.

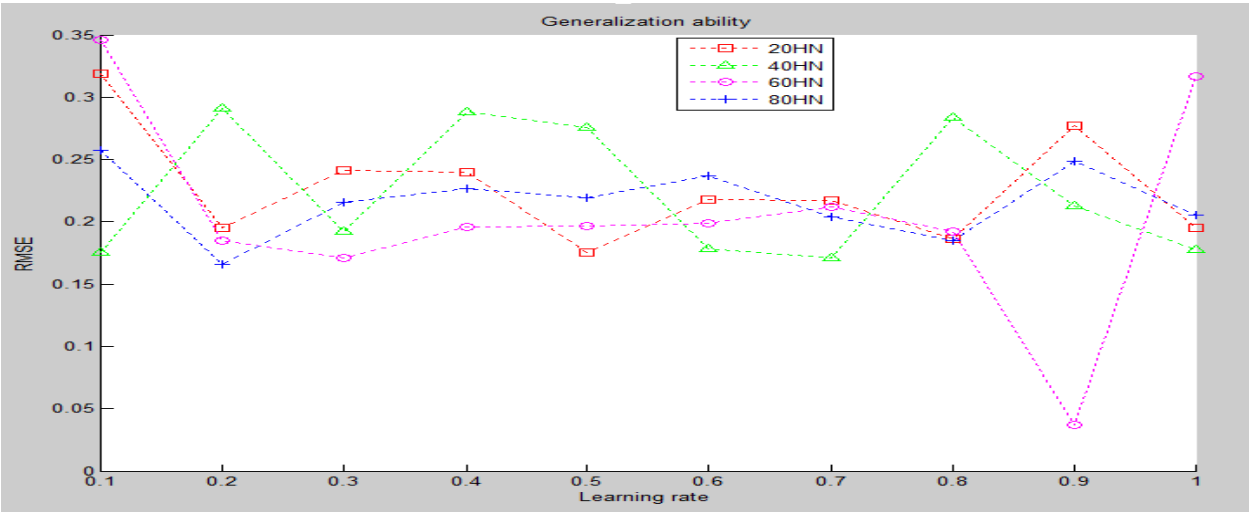


Figure 5.15: Generalization errors (RMSE) for the different learning rates at various network architectures.

Table 5.2: R and R^2 on train and test sets for different learning rates at various network architectures.

HN	**Learning rate (R train values)**									
	0.1	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
20	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
40	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
60	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
80	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
HN	**Learning rate (R test values)**									
	0.1	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
20	0.9842	0.9969	0.9938	0.9999	0.9991	1.0000	0.9982	0.9989	0.9965	0.9952
40	1.0000	0.9896	0.9982	0.9686	0.9979	0.9992	0.9997	0.9984	0.9983	0.9997
60	0.9765	0.9977	0.9980	0.9999	0.9990	0.9975	0.9996	0.9947	0.9997	0.9889
80	0.9874	0.9979	0.9937	0.9951	0.9979	0.9869	0.9911	0.9995	0.9951	0.9989
HN	**Learning rate (R2 train)**									
	0.1	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
20	0.9952	0.9909	0.9942	0.9953	0.9920	0.9926	0.9939	0.9924	0.9932	0.9922
40	0.9921	0.9951	0.9916	0.9942	0.9953	0.9916	0.9911	0.9948	0.9948	0.9928
60	0.9940	0.9914	0.9914	0.9922	0.9920	0.9913	0.9914	0.9919	0.9927	0.9972
80	0.9928	0.9915	0.9932	0.9925	0.9923	0.9929	0.9900	0.9927	0.9928	0.9921
HN	**Learning rate (R2 test)**									
	0.1	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
20	0.8046	0.9266	0.8885	0.8897	0.9409	0.9086	0.9096	0.9331	0.8528	0.9265
40	0.9406	0.8374	0.9292	0.8406	0.8536	0.9391	0.9437	0.8449	0.9132	0.9396
60	0.7695	0.9341	0.9438	0.9264	0.9260	0.9240	0.9139	0.9292	0.9288	0.8071
80	0.8724	0.9470	0.9107	0.9015	0.9076	0.8921	0.9203	0.9341	0.8812	0.9188

Secondly, we then selected to train a single hidden layer ANN of 60 hidden neurons, at learning rates of 0.2, 0.4, 0.6, and 0.8 for a different number of epochs to ascertain how the impact of learning rate on generalization ability varies with an increase in number of training iterations. The choice of the values of the learning rates was motivated from a previous similar case study by Attoh-Okine (1999) who experimented with the same values. The reason an ANN with 60 hidden neurons was selected for investigation was because it exhibited the best generalization ability amongst the single hidden layer architectures examined in experiments in sub-section 5.1.1.1. The size of the training set was fixed at 547 samples and of the test set at 182 samples, weights were randomly initialised in the range of [-0.5, 0.5], the BP (trainlm) training rule was the weight updating algorithm and the momentum was fixed at 0.2. Training was stopped after 1000 training cycles for each training run and the performance of the ANN tested on the test set of 182 samples. Figure 5.16 shows the generalization errors from the experiment, and Table 5.3 shows the results of the R and R^2 on both train and test sets for the experiments conducted.

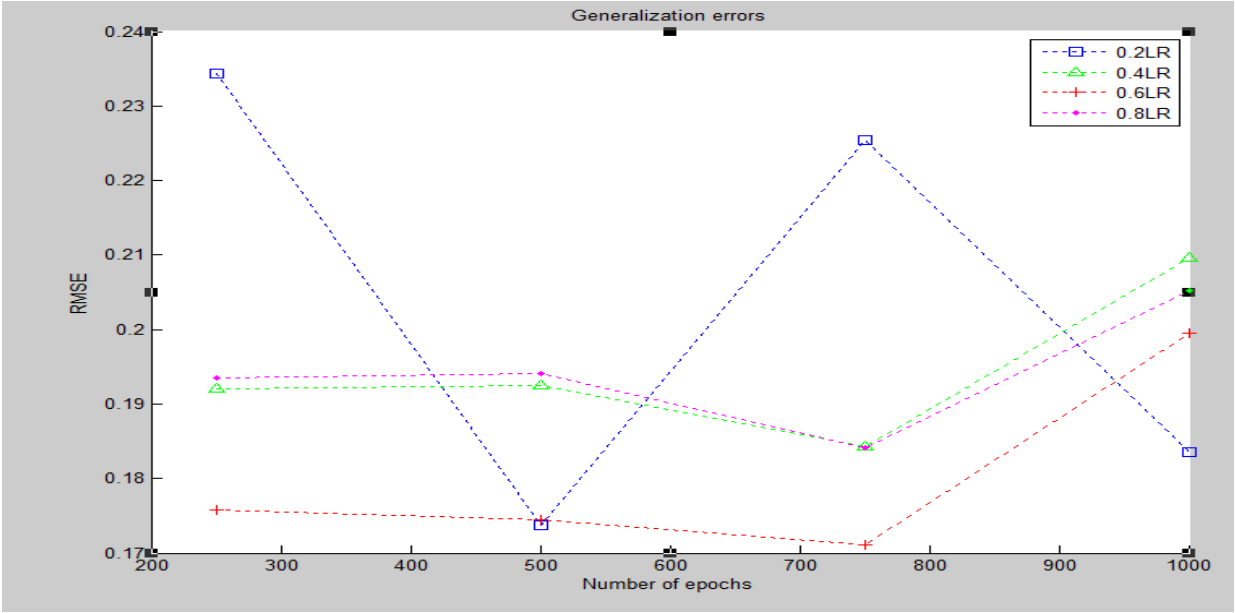


Figure 5.16: Generalization errors (RMSE) for the different learning rates at various training iterations.

Table 5.3: R and R^2 on train and test sets for different learning rates at various training iterations.

Number of epochs	Learning rate (R train values)				Learning rate (R test values)			
	0.1	**0.2**	**0.3**	**0.4**	**0.1**	**0.2**	**0.3**	**0.4**
250	1.0000	1.0000	1.0000	1.0000	0.9922	0.9998	0.9995	0.9957
500	1.0000	1.0000	1.0000	1.0000	0.9977	0.9968	0.9973	0.9974
750	1.0000	1.0000	1.0000	1.0000	0.9977	0.9990	0.9990	0.9952
1000	1.0000	1.0000	1.0000	1.0000	0.9998	0.9981	0.9977	0.9966
Number of epochs	**Learning rate (R2 train values)**				**Learning rate (R2 test values)**			
	0.1	**0.2**	**0.3**	**0.4**	**0.1**	**0.2**	**0.3**	**0.4**
250	0.9808	0.9843	0.9847	0.9875	0.8944	0.9292	0.9406	0.9280
500	0.9905	0.9901	0.9878	0.9908	0.9420	0.9288	0.9414	0.9276
750	0.9905	0.9918	0.9903	0.9911	0.9420	0.9347	0.9437	0.9348
1000	0.9915	0.9928	0.9916	0.9922	0.9182	0.9156	0.9235	0.9191

We then conducted another batch of experiments which examined a two hidden layer network architecture. After conducting several experiments on various network architectures from experiments in section (5.1.1.2) a two hidden layer ANN of network architecture (5:35) was chosen as the bases for conducting our investigations. The basis for arriving at the chosen architecture was the fact that this architecture exhibited better generalization ability than all the other two hidden layer architectures examined in sub-section 5.1.1.2. We trained the ANN on two different learning rates of 0.1 and 0.01 per run. The sizes of the training set and test sets were maintained at 547 and 182 samples respectively. A momentum value (m) of 0.2 was maintained throughout the experiments. The results of the generalization errors are shown in Figure 5.17, and Table 5.4 shows the results of R and R^2 on both train and test sets for the experiments conducted.

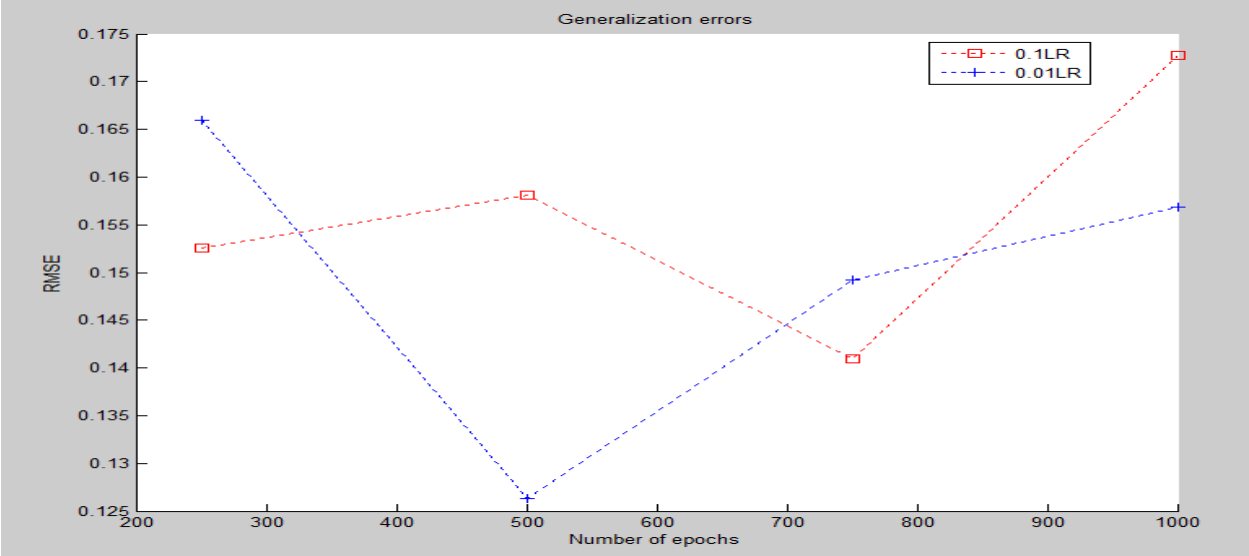


Figure 5.17: Generalization errors (RMSE) for different learning rates at various training iterations on (5:35) HN ANN.

Table 5.4: R and R^2 on train and test sets for different learning rates at various training iterations on (5: 35) HN ANN.

Number of epochs	Learning rate (R train values)		Learning rate (R test values)		Learning rate (R2 train values)		Learning rate (R test values)	
	0.1	**0.01**	**0.1**	**0.01**	**0.1**	**0.01**	**0.1**	**0.01**
250	1.0000	1.0000	0.9999	0.9280	0.9877	0.9911	0.9519	0.9694
500	1.0000	1.0000	0.9999	1.0000	0.9877	0.9875	0.9519	0.9694
750	1.0000	1.0000	1.0000	0.9999	0.9900	0.9899	0.9618	0.9572
1000	1.0000	1.0000	1.0000	0.9999	0.9913	0.9908	0.9427	0.9527

Finally, in order to reach fair conclusions additional experiments were conducted using different combinations of learning rates and momentum values so as to assess whether the effect of different learning rates is the same regardless of momentum. These two parameters have been suggested to be quite closely related (Attoh-Okine 1999). We trained, as in the previous case, an ANN of network architecture (5:35) using different heuristics of learning rates and momentum provided by some of the authors listed in Table 3.5. Likewise training began with random initial weights of [-0.5, 0.5] and was halted after 1000 training cycles, after which the generalization performance of the ANN was tested on the test set of 182 samples. The generalization errors, as well as R and R^2 for train and test sets, are shown in Figure 5.18 and Table 5.5 respectively.

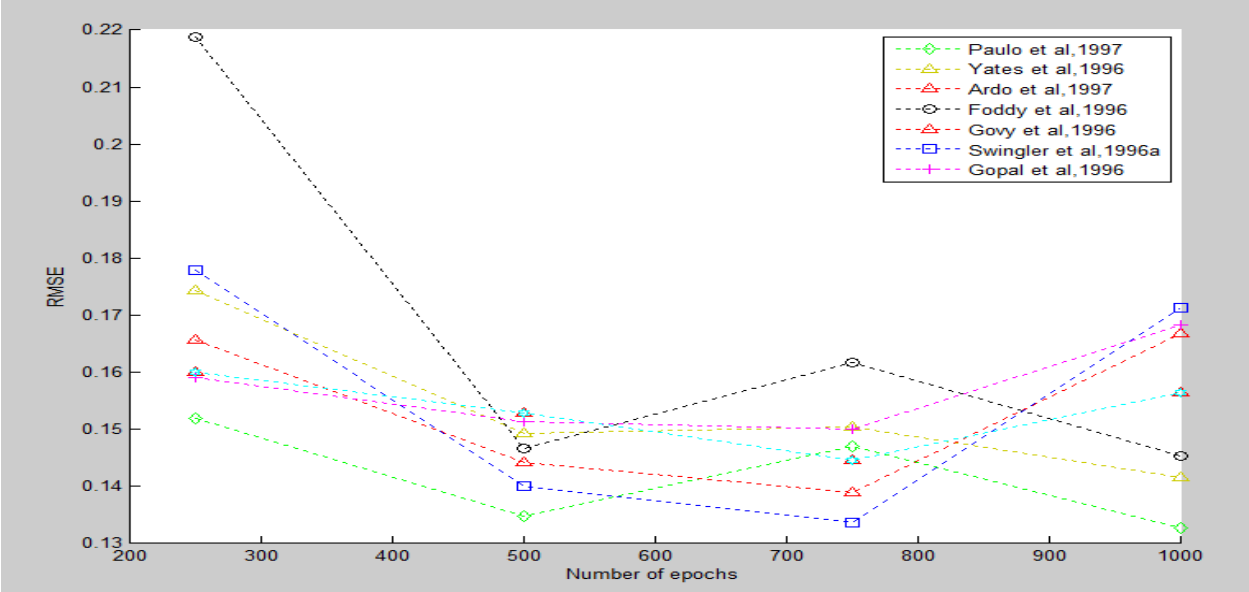


Figure 5.18: Generalization errors (RMSE) for different learning rates and momentum combinations at various training iterations on (5:35) HN ANN.

Table 5.5: R and R^2 on train and test sets for different learning rates and momentum combinations at various training iterations on (5: 35) HN ANN.

Number of epochs	R train values for different learning rates and momentum						
	Paulo	**Yates**	**Ardo**	**Foddy**	**Govy**	**Swingler**	**Gopal**
250	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
500	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
750	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
1000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Number of epochs	**R test values for different learning rates and momentum**						
	Paulo	**Yates**	**Ardo**	**Foddy**	**Govy**	**Swingler**	**Gopal**
250	0.9994	1.0000	1.0000	1.0000	0.9996	1.0000	0.9996
500	1.0000	0.9999	0.9999	0.9996	0.9999	1.0000	0.9999
750	0.9998	1.0000	0.9996	1.0000	0.9998	1.0000	0.9998
1000	0.9999	0.9999	0.9984	1.0000	0.9997	1.0000	1.0000
Number of epochs	**R2 train values for different learning rates and momentum**						
	Paulo	**Yates**	**Ardo**	**Foddy**	**Govy**	**Swingler**	**Gopal**
250	0.9802	0.9838	0.9800	0.9769	0.9835	0.9798	0.9858
500	0.9896	0.9895	0.9897	0.9892	0.9879	0.9878	0.9900
750	0.9893	0.9912	0.9893	0.9912	0.9885	0.9906	0.9909
1000	0.9894	0.9902	0.9904	0.9898	0.9915	0.9914	0.9898
Number of epochs	**R2 test values for different learning rates and momentum**						
	Paulo	**Yates**	**Ardo**	**Foddy**	**Govy**	**Swingler**	**Gopal**
250	0.9556	0.9416	0.9472	0.9080	0.9507	0.9391	0.9514
500	0.9651	0.9572	0.9600	0.9586	0.9551	0.9623	0.9560
750	0.9584	0.9564	0.9629	0.9497	0.9598	0.9657	0.9567
1000	0.9661	0.9615	0.9465	0.9594	0.9530	0.9436	0.9455

5.1.5 Training scenario 5: Training schedule

In the fifth training scenario the focus was on determining the most appropriate training schedule for ANNs in forecasting TCP/IP network traffic trends. In this series of experiments we sought to establish how different stopping criteria impact on the generalization ability of ANNs, furthermore we intended to examine the relationship between the generalization ability and overtraining of ANNs. As usual 547 out of 730 samples were allocated to the training set, whilst 182 data samples were allocated to test the performance of the trained ANNs. Yet again, an ANN of architecture (5:35), and an ANN with 60 hidden neurons were trained for a different number of iterations. Learning epochs of 100, 300, 500, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000 and 10000 were selected for examination in these investigations. These were largely selected on an arbitrary basis. On training, weights were randomly initialized in the range of [-0.5, 0.5], and the learning rate constant and momentum were fixed at 0.2 and 0.6 respectively. The popular BP (trainlm) algorithm was used in updating weights and the Logistic sigmoid and Linear activation functions were used in the hidden and output layers respectively. The resulting ANNs were later assessed on the independent test set of 182 samples. As noted, the majority of ANNs investigated thus far were trained for up to 1000 epochs. To investigate the effects of the overtraining phenomena on the generalization ability of ANNs, we trained the ANNs for up to 10 000 epochs, the performance window plot was monitored while training progressed. If the curve went flat for many hundreds of epochs it was likely that the ANNs were overfitting the data and training would be stopped immediately. To negate the effects of random initial weights each training-testing run was performed twice and the results averaged. Results of the experiments are shown in Figures 5.19 and 5.20 respectively.

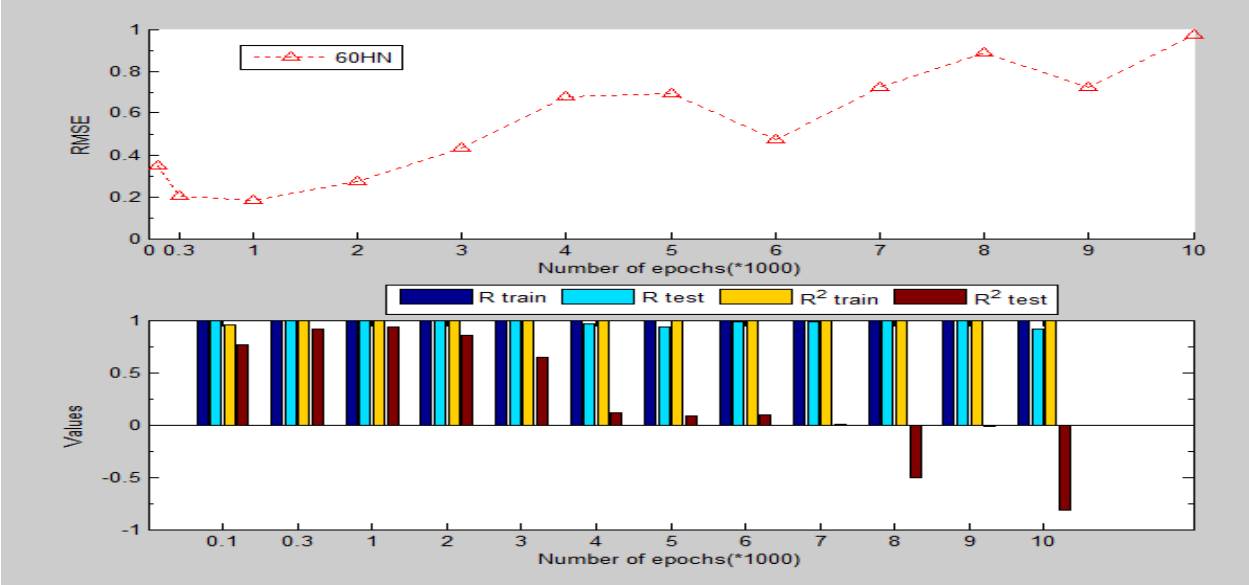


Figure 5.19: Results for different training iterations on a 60HN network: Generalization errors in RMSE (top); R and R^2 for train and test sets (bottom).

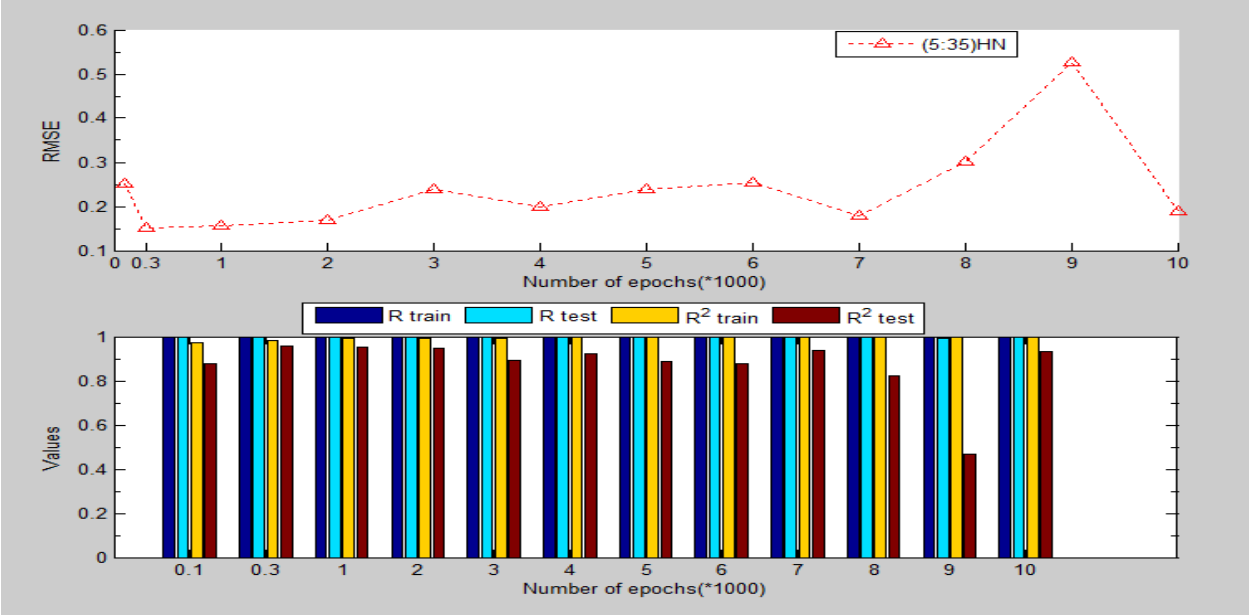


Figure 5.20: Results for different training iterations on a (5:35) HN network: Generalization errors in RMSE (top); R and R^2 for train and test sets (bottom).

5.2 Discussion of results

In this section the results presented in section 5.1 are discussed in detail. Where applicable we compare our findings with previous studies conducted within similar research frameworks. It must be noted, however, that in any analysis of these results, attempting to completely disentangle the effects on ANN performance of each of the five parameters under investigation in this research is practically impossible. In so far as possible, in order to ensure that the results presented are indicative of the effects due to changes in the particular parameter under observation, in each set of experiments carried out, one parameter was varied and all other parameters were kept constant. Where there was an interaction of parameters, for instance the results of ANNs trained on various learning rates at different training epochs, we discuss the results in terms of what we believe is the primary factor effecting changes in ANN performance and generalization ability, whilst noting the possibility of effects from other factors. The reader is however entitled to disagree as to which is the most influential parameter in a particular situation: this of course is the essence of research. Also, as in any practical situation there are bound to be some erratic or unordinary behaviours. Where possible and to the best of our understanding, we try to explain any such kind of behaviour exhibited by the ANNs under investigation in this research.

5.2.1 Size of network architecture

We begin with the results of the effects of size of network architecture on the ability of ANNs to generalize well to novel data in the task of forecasting trends in a TCP/IP network traffic time series. Firstly we examine the results exhibited by a single hidden layer ANN architecture. The results from Figure 5.1 show that when the number of hidden neurons is in the range of 10 to 20, in a single hidden layer ANN, the generalization errors increase as more hidden neurons are continually added, however, this trend is suddenly reversed after 20 hidden neurons, where a precipitous decrease in generalization errors takes precedence. Figure 5.1 illustrates that, to a certain extent and in general, an increase in the number of hidden neurons is concomitantly accompanied by a corresponding increase in generalization ability of the ANN. This trend is confirmed by the gradual decrease in generalization errors as more hidden neurons were continually added after 20 hidden neurons, with the lowest errors attained between the ranges of 60 to 70 hidden neurons.

Quite interestingly, our findings are in sharp contrast to Occam's razor principle and to the Overfitting guideline which would have us believe that the network with 10 hidden neurons, by virtue of the fact that it's the smallest, is bound to have better generalization ability than the rest. It is apparent from the results in Figure 5.1 that as good or better generalization ability in the task of forecasting TCP/IP network traffic occurs when there are more than those numbers of hidden neurons in a single hidden layer ANN. Our results reveal that an ANN having 50 hidden neurons exhibits much better generalization ability than an ANN having 20 hidden neurons, and an ANN having 70 hidden neurons exhibits much better generalization than an ANN having 50 hidden neurons. However we note that when the number of hidden neurons exceeds a certain threshold of 70 hidden neurons, the generalization errors begin to drastically increase again. This comes as no surprise, as literature from Teixeira and Fernandes (2012) indicates that an increase in network size results in a more complex ANN that tends to overshoot the decision boundaries causing generalization problems. Also the poor network performance between 10 and 20 hidden neurons can probably be attributed to the fact that the small number of hidden neurons did not have enough power to model and learn the data. Our results from Figure 5.1 are similar in principle to the factual findings of Morgan and Bourlard (1989) who recorded an increase in the generalization ability of their ANNs from 5 to 50 hidden neurons, after which the generalization abilities of their ANNs began to drop drastically, reaching an asymptote at 250 hidden neurons. They gave no valid reason as to why the ANNs behaved in such a manner, apart from highlighting the need for further research. Also interesting to note is that there appear to be two regions along the generalization error curve in Figure 5.1, where the ANN's performance seems to be oblivious to changes in size of the network. This pattern is apparent between 30 to 50 hidden neurons and between 60 to 70 hidden neurons. Observe that at 30 hidden neurons the ANN has a RMSE of 0.0221. Adding a further 20 hidden neurons did not have any significant effect in terms of RMSE. In a strikingly similar fashion, at 60 hidden neurons the network has a RMSE of 0.186. Adding a further 10 hidden neurons does not appear to effect any changes in that regard. We suspect this may be due entirely to the initial random weights between those two ranges, since all other parameters were constant, perhaps the initial weights could have been oscillating about a fixed starting point. Analysis of the statistical measures in Figure 5.1 shows relatively good performance of the ANN for all the architectures examined. Values of R and (R^2) are above 0.8 on both the train and test sets, indicating a good linear correlation and a

positive fit between the activations (predicted values) and targets (actual values). The ANN did not show any signs of overfitting during the course of these investigations.

Turning to the results produced by two hidden layer architectures, we carefully scrutinize results from Figure 5.2. Results from Figure 5.2 reveal that using an equal ratio of first to second hidden neurons does not have any significant impact on the generalization ability of ANNs. On examining Figure 5.2, there is some variation in terms of the generalization errors for the various network architectures examined, but no clear trend can be observed to warrant substantial conclusions as regards their ability to generalize to new data. For instance the generalization errors seem to be on an increasing trajectory from network architecture (5:5) to network architecture (15:15). They suddenly begin decreasing from network architecture (15:15), reaching a minimum at network architecture (25:25), where once again the errors begin to increase until network architecture (35:35), then a successive decrease in errors is noted until network architecture (45:45). It appears that the network performance fluctuates after every successive 10 hidden neurons. Whilst these results are not exclusively unique to us, as Teixeira and Fernandes (2012) report a similar occurrence in their experiments, we could not find any substantial reason for this kind of erratic network behaviour, either in literature or through our own analysis. However, once again we cannot completely rule out the possibility of random effects of network weights. Judging from the nature of our results in Figures 5.1 and 5.2, it is safe to say that the generalization ability of a single hidden layer ANN with any number of hidden neurons is better than that of a two hidden layer model with an equal number of neurons, albeit distributed over the two hidden layers. We reached this conclusion based on the following observations: In Figure 5.1, note that the generalization error of an ANN with 60 hidden neurons in a single hidden layer is a RMSE of 0.185, whereas in Figure 5.2 the generalization error of an ANN having 60 hidden neurons distributed equally over the 2 hidden layers is an excessive RMSE value of 0.35. This is more than a 65% decrease in generalization accuracy. Suffice to say, this trend is apparent in all the architectures examined and whose results are shown in both Figures 5.1 and 5.2. These empirical results add weight to the theoretical findings reported by Judd (1990) "*for a given number of hidden neurons, it is better to contain those hidden neurons in a single hidden layer than to distribute them over two or more hidden layers*".

Analysis of the statistical measures from Figure 5.2 indicates a relatively good fit and a positive linear correlation between the targets and activations for most of the network architectures examined. The only notable exception was the ANN trained on a network architecture of (35:35), which attained an (R^2) value of less than 0.7 on the test set, indicating perhaps that the size of the sliding time window was not sufficient to make accurate forecasts at that particular architecture.

We now turn to the results where different ratios of first to second hidden neurons were explored. Results from Figure 5.3 indicate that when the number of second hidden neurons is fixed, an increase in the number of the first hidden neurons is generally accompanied by a decrease in the generalization ability of the ANNs. Figure 5.3 shows an increase in generalization errors as the size of the network is increased through the addition of more first hidden layer neurons. We note that from network architecture (5:5) to network architecture (20:5) there is linear increase in generalization errors as the size of the network increases, but from network architecture (20:5) to (35:5), although an increase in generalization errors can be noted, it is rather less linear and bordering on constant. A sudden dip in generalization errors occurs at network architecture (40:5) corresponding to a ratio of first to second hidden neurons of 8:1. After this the generalization errors continue on their upwards incline. This trend is most probably attributable to the increase in computation as the size of the ANN increases. Evaluation of R values on both the train and test sets demonstrate a fairly reasonable linear correlation for the network architectures concerned, as any R value above 0.8 is indicative of a positive relationship between the targets and activations; however a gradual decrease in the goodness of fit is noted as the size of the ANN increased. Note that R^2 values on the test set range from 0.4 to 0.6 for lager networks, *i.e.* network architectures (20:5) to (50:5). However in terms of the generalization ability, an ANN of architecture (40:5) performed exceptionally above the nominal average.

Moving on to the results illustrated in Figure 5.4, we note that when the previous trend is reversed and the number of first hidden neurons is instead fixed, an increase in the size of the second hidden neurons is typically accompanied by an increase in generalization ability. However as in most cases this trend is only true up to a certain point. Careful analysis of the

generalization error curve shown in Figure 5.4, reveals that whilst the generalization errors seem to increase from network architecture (5:5) to (5:15), an immediate but unsteady decrease of the same follows for a greater part of the curve, culminating in a margin at network architecture (5:35), which plateaus right up to network architecture (5:40). A temporary increase in error is noted at network architecture (5:45), followed by a mild dip in errors at network architecture (5:50). In Figure 5.4 we note that fixing the size of the first hidden neurons and varying the second hidden neurons caused the generalization errors to fluctuate quite extensively. This perhaps leads to the assumption that the first layer hidden neurons are important in adding some stability to the ANN performance. Analysis of the statistical measures in Figure 5.4, shows comparatively good performance for the network architectures examined. Values of R and (R^2) are above 0.8 on both train and test sets, indicating a positive linear correlation and good fit between the activations and targets. Our results from Figures 5.3 and 5.4 are in stark contrast to the empirical findings of Kudryci (1988), who concluded that in forecasting tasks a ratio of 3:1 between the first and second hidden layer neurons would certainly warrant a good generalization ability. In our case this seemed not to be the case as the best generalization performance was attained when a ratio of first to second hidden neurons was 1:8, which is a far cry from the heuristic provided by Kudryci (1988).

The final set of results in this sub-section summarizes the effects of different layers and neurons on the generalization performance of ANNs in forecasting TCP/IP network traffic trends. Results from Figure 5.5 indicate that, based on the generalization errors, compared to the one hidden-layer model, almost all two hidden-layer and three hidden-layer models did not perform significantly better. As Figure 5.5 shows, the generalization errors achieved by one hidden layer architectures were in most instances better than two and three hidden layers architectures considering the low number of neurons. This confirms the findings of researchers such as Zhang et al. (1998) who proposed that simple ANN models are often adequate for forecasting linear time series. Zhang et al. (1998) claim that one hidden neuron was sufficient for optimal generalization and accurate forecasting. This is despite the fact that the number of hidden neurons in their study was from 1 to 10, which is rather limited compared to our study which explored architectures as huge as 80 hidden neurons in as much as three hidden layers. In Figure 5.5, the best generalized architecture was a single hidden layer of 60 hidden neurons. A further

enlargement of the ANN by way of a second hidden layer or a third hidden layer of neurons did not have a significant impact on network performance. The use of two or three hidden layers did not in any significant way improve the generalization ability of the ANN, in fact in some cases it degraded it. However for some strange reason using a three hidden layer ANN gave a slightly better generalization performance than two hidden layers, especially when the two hidden layer model either had too few neurons or too many. It is also interesting to note that the best generalization performance for a two hidden layer ANN of network architecture (40*2) was worse than the least generalized single hidden layer architecture of 20 hidden neurons; equally the best generalized three hidden layer model, *i.e.* (40*3), performed in an almost similar fashion to the worst generalized single hidden layer architecture. These findings put into effect claims made by researchers such as Cybenko (1989) that two hidden layers are sufficient to approximate any function and that additional layers would be computationally redundant. Insofar as the statistical measures in Figure 5.5 are concerned it is safe to say that all network architectures indicated a good fit and a positively correlated linear relationship between the activations and targets, judging by the quality of results from R and R^2 values, all of which are above 0.7, in some cases 0.9, however all single hidden layer models performed marginally better than the other models. It is important also to note that for all the network architectures considered there were no visible signs of overfitting.

5.2.2 Training set size

In this sub-section, we turn our attention to the results showing the relationship between the size of the training set and the generalization exhibited by an ANN in forecasting TCP/IP network traffic trends. We begin with the results given in Figure 5.6, which show the generalization errors versus the size of the training set for a single hidden layer ANN with 60 hidden neurons. On examining Figure 5.6, one cannot shy away from the exponential decrease in generalization errors as the size of the training set is enlarged. This trend is unambiguously uninterrupted until a minimum RMSE in generalization errors is attained at the full training set of 547 samples. Our results are similar to those of Kavzoglu (1999), who in empirically conducting experiments on ANNs in speech synthesis recorded an exponential decrease in generalization errors as the size of the training set increased. Quite evidently from the results in Figure 5.6, the generalization ability of a single hidden layer ANN trained on a reduced training set is significantly degraded juxtaposed with an ANN trained on the full training set of 547 samples. Perhaps one possible

reason for this is that the full training set was more representative of the problem space than the altered variations of it. This undoubtedly adds another dimension to the whole debate, that of the quality of the training set. Our findings are contrary to those of Zhang et al. (1998), who suggested that a training set size of only 5% of the total data is sufficient for a good generalization ability, especially of a single hidden layer ANN. Also, Lange et al. (1997) introduced the idea of a critical training set size. Through experimentation they found that examples beyond this critical size do not improve generalization, illustrating that an excess of training patterns has no real gain. They state that this critical training set size is problem-dependent. Although we are not entirely sympathetic to this viewpoint it appears that the argument raised by Lange et al. (1997) is quite valid, judging by the nature of the generalization error curve. At the full training set of 547 samples, the generalization errors seem to plateau, raising potential doubts that any increase in training set size beyond that point would have further decreased the errors, we are however reluctant to derive substantial conclusions in that regard due to the fact that we did not train the networks beyond the full allocated training set size of 547 samples. On analysis of the statistical measures in Figure 5.6, we note that at lower proportions of the training set values of the R^2 on the test set were largely lower than at higher proportions of the training set. In fact at 10% of the training set a negative R^2 value on the test set was recorded, indicating a very bad fit between the activations and targets. Generally the ANN exhibited poor performance on both R and R^2 for training set sizes below 80% of the full training set.

The next set of results is those exhibited by a two hidden layer ANN with a network architecture of (5:35). Once again the trend in Figure 5.7 indicates a decrease in generalization errors as the size of the training set is increased. However this time, the trend is not as exponentially smooth as in the previous case. This is evidenced by the huge fluctuations in generalization errors witnessed from 10% to 50% of the full training set of 547 samples, after which the generalization error curve gradually assumes a smooth exponential decline in errors, reaching a minimum at the full training set. Whilst we are not exactly certain as to the actualities resulting in the slightly erratic behaviour of the ANN between 10% to 50% of the full training set of 547 samples, one possible reason could be that within this range, particular instances within the ANN are a better representation of the problem space than others. If these instances are randomly spread

throughout the data set then the likelihood of their being selected for training is equally random. Another possible cause of the erratic behaviour of the ANN in the concerned range of training samples could have been due to the addition of an extra hidden layer. Perhaps the addition of a second hidden layer caused the ANN to be unstable. Analysis of statistical measures indicates that from 60% and below of the full training set, values of R^2 (on the test sets) range mostly between 0.6 to -0.8, revealing a poor fit between the activations and targets. As largely expected the best performances in terms of R and R^2 were recorded at sample sizes larger than 70% of the training set. The generalization results show that the best generalization performance of a single hidden layer architecture in Figure 5.6 is attained at a RMSE of approximately 0.25, whilst for a double hidden layer architecture in Figure 5.7, the same is attained at a comparably similar error level. These RMSE values were both attained at the same percentage of the training set i.e. at 547 samples. What is perhaps most striking about these results is the comparative equivalency of the generalization errors in Figures 5.6 and 5.7, despite the different network architectures. Our results seem to contradict the empirical findings suggested by Baum and Haussler (1989) and Weigend et al.(1990) who state that more training data are required to achieve good generalization ability for larger ANNs. For this research, generalization ability was not affected by using larger ANNs, despite the fact that limited training data were available. This quite indelibly is also in stark contrast to the Dataset size guideline of Richard (1991).

Moving on to the results illustrated by Figure 5.8, which show the generalization performances of different heuristics provided by various authors in selecting the optimal training set size for generalization ability of ANNs. It can be noted again that as the number of training samples increases, there is a gradual, almost logarithmic decrease in generalization errors. For the case study considered the heuristic proposed by Hush et al. (1992) suggests an insufficiently small number of training samples, whilst on the other hand the heuristics recommended by Klimasauskas (1993) and Baum and Haussler (1989) indicate training set samples that are relatively close to the optimum solution. R and R^2 values at or above 0.8 are statistically significant. Of the results in Figure 5.8, only Baum and Haussler (1989) heuristics satisfy that condition. None of the networks examined showed any signs of overfitting.

5.2.3 Learning algorithm

We now turn the subject of our discussion to the experiments that examined the effect the choice of learning algorithm has on the generalization ability of ANNs in forecasting TCP/IP network traffic trends. Figure 5.9 shows the generalization errors of several single hidden layer ANNs trained on each of the five different BP algorithms. We note that the difference between the generalization errors due to the various algorithms became more pronounced as the size of the networks increased. This effect is clearly illustrated in Figure 5.9. Note that at 20 hidden neurons ANNs trained on the Conjugate gradient backpropagation with Fletcher-Reeves updates (traincgf), Scaled conjugate gradient backpropagation (traincgp) and Resilient backpropagation (trainrp) algorithms performed best in terms of generalization ability as they constantly displayed the lowest generalization errors. An ANN trained on the Gradient descent with momentum backpropagation (traingdm) algorithm was the least generalized. This pattern of results remains virtually undiminished up to 60 hidden neurons, at which point the generalization errors steadily become divergent. The Conjugate gradient backpropagation with Polak-Ribiére updates (trainscg) algorithm, which was previously performing relatively below the Conjugate gradient backpropagation with Fletcher-Reeves updates algorithm, gradually outperforms the later as more hidden neurons are added. Further, the Gradient descent with momentum backpropagation algorithm which was previously the least performing algorithm suddenly outperforms the Resilient backpropagation algorithm from 60 hidden neurons to 80 hidden neurons. Results also show that the performance of the Conjugate gradient backpropagation with Powell-Beale restarts (traincgb) algorithm was least affected by the size of the network in comparison with the other learning algorithms. Note also that at 80 hidden neurons the Conjugate gradient backpropagation with Powell-Beale restarts algorithm performs marginally better than the Conjugate gradient backpropagation with Fletcher-Reeves updates algorithm.

The simulation results show strong evidence of the superiority of Conjugate gradient backpropagation with Polak-Ribiére updates and Conjugate gradient backpropagation with Fletcher-Reeves updates algorithms in terms of generalization. On the other hand, Resilient backpropagation and Gradient descent with momentum backpropagation algorithms were the worst performing algorithms. What is not surprising though is the fact that all the Conjugate family of algorithms, *i.e.* Conjugate gradient backpropagation with Powell-Beale restarts,

Conjugate gradient backpropagation with Polak-Ribiére updates and Conjugate gradient backpropagation with Fletcher-Reeves updates performed much better than all the other algorithms used in the study, a fact widely verified in literature (Jacobs 1998; Joy 2011). Our results in Figure 5.9 show that Backpropagation Neural Networks (BPNN) trained on the Conjugate gradient backpropagation with Polak-Ribiére updates exhibits the best generalization ability in forecasting TCP/IP traffic trends. Analysis of the statistical measures in Table 5.1, shows that for the ANNs trained on 4 of the 5 algorithms, there was a positive linear relationship between the targets and activations, as indicated by the *R* train and test values, which are all above 0.8. The results also show that for the same networks a good fit between the predicted values and actual values was realized, judging by the R^2 values on the train and test set, which are above 0.8. The most obvious exception were the ANNs trained on the Resilient backpropagation and Gradient descent with momentum backpropagation algorithm, which consistently exhibited extremely low R^2 values below 0.2 on the test set. Also, none of the ANNs examined showed any signs of overfitting.

On a rather different note, limited investigations on the speed of convergence and the accuracy of the different learning algorithms were also conducted. A two hidden layer ANN with a network architecture of (5:35) was trained on the five various algorithms and the performance of the learning algorithms assessed. We begin by examining the results exhibited by the Conjugate gradient backpropagation with Polak-Ribiére updates algorithm. From Figure 5.10, we note that an ANN trained on the Conjugate gradient backpropagation with Polak-Ribiére updates algorithm achieved a mimimum prediction MSE of 0.00099956, and convergence was achieved after 697 epochs. These results indicate that Conjugate gradient backpropagation with Polak-Ribiére updates algorithm has a fast convergence. Figure 5.11 shows the results of the performance of the Resilient backpropagation algorithm. Note that the mimimum MSE attained is 0.00099956, and convergence was achieved after 942 epochs. The Resilient backpropagation algorithm acheived the same levels of accuracy as the Conjugate gradient backpropagation with Polak-Ribiére updates algorithm but it was much slower than the Conjugate gradient backpropagation with Polak-Ribiére updates algorithm, as shown by the fact that convergence was attained after 942 epochs, compared to the former's 697 epochs.

Figure 5.12 shows the results of the performance of the Conjugate gradient backpropagation with Fletcher-Reeves updates algorithm. This implementation shows that this algorithm converges faster than the previous two. We note that the curve stabilizes earlier than in the previous algorithms, however from Figure 5.12, we find that the minimum MSE is 0.00099997, and convergence was attained after 615 epochs. Therefore the results indicate that whilst the Resilient backpropagation algorithm is faster than the previous algorithms, its shortfall is that it has a very unstable accuracy. Figure 5.13 shows the performance of the Scaled conjugate gradient backpropagation algorithm. From the results we note that the minimum MSE is 0.00099961, and convergence was reached after 816 epochs. From the training results, we deduce that the Scaled conjugate gradient backpropagation algorithm is generally much faster than the Scaled conjugate gradient backpropagation algorithm; but it is less accurate. Figure 5.14 shows the training result of the Gradient descent with momentum backpropagation algorithm. The minimum MSE is 0.029197, and the results show that the algorithm failed to converge after 1000 epochs. This means that for the case study under consideration, the Gradient descent with momentum backpropagation algorithm was the slowest and least accurate.

In summary, judging by the quality of results, if we are to make a fair assessment in terms of a balance between the speed of convergence and accuracy of results, the Conjugate gradient backpropagation with Polak-Ribiére updates algorithm had the best performance. This is most probably due to the fact that in the Conjugate family of algorithms the weights are adjusted along conjugate directions. That generally produces faster convergence and better accuracy than the rest of the BP algorithms. It seems to have a very efficient Matlab implementation; however we commit our findings to further scrutiny due to the limited nature of these investigations.

5.2.4 Learning rate

In this sub-section we discuss the empirical results of experiments regarding the relationship between ANN generalization and learning rate. We begin our analysis by examining the results in Figure 5.15, which show the generalization errors of various single hidden layer ANNs trained on various learning rates. From Figure 5.15 it appears that indeed selecting the optimum learning rate for a given problem is a complex task. From the results, we note that the best performing ANN was a 60 hidden architecture which was trained on a learning rate of 0.9. We note that for many of the architectures examined the best generalization errors occurred at lower learning

rates, mostly between 0.2 to 0.7, however it is difficult to pinpoint a single universal value which we can safely conclude to be effective in all cases. In many cases though a learning rate of 0.6 seemed to give the most reasonable generalization errors. Our results indicate that the smaller ANNs performed badly in these experiments, especially the 20 hidden neuron and 40 hidden neuron architectures. These two architectures also displayed the most erratic behaviour, whether trained on smaller or larger learning rates, indicating the intrinsic relationship between learning rates and network architecture, thereby emphasizing the sensitivity of larger networks to learning rate. It was also noted that small learning rates produced consistent and better results, whereas large learning rates appeared to cause oscillations and inconsistent results. The statistical measures in Table 5.2 show relatively good performance of the ANN for all the architectures concerned. Values of R and (R^2) are above 0.8 on both the train and test sets, indicating a good linear correlation and positive fit between the activations (predicted values) and targets (actual values). The ANNs examined did not show any signs of overfitting.

In the same vein, we trained an ANN having 60 hidden neurons at different learning rates of 0.2, 0.4, 0.6, and 0.8 for a different number of epochs in order to ascertain how the impact of learning rate on generalization ability varies with an increase in the number of training iterations. The results of that endeavour are shown in Figure 5.16. From Figure 5.16 we note that the best generalization performance was attained by the ANN when trained on a learning rate of 0.6. It is evident from the results that the generalization ability of the ANN increased as the size of the learning rate increased, however this was only true up to a certain threshold, beyond which any further increment in learning rate resulted in adverse effects. This is made especially true judging by the generalization errors incurred. From Figure 5.16 we note that the ANN performed the worst and was mostly erratic at a learning rate of 0.2. When the learning rate was increased to 0.4 the generalization ability of the ANN dramatically improved. At a learning rate of 0.6 the ANN was at its best in terms of generalization performance, however when a learning rate of 0.8 was used instead of following a similar antecedent, the generalization ability of the ANN decreased. It is also interesting to note that for all the learning rate-number of iteration combinations the ANN seemed to perform best in terms of generalization ability at 750 epochs. Additionally, training time was fast for learning rates of 0.2 and 0.4, and reasonably fast for rates of 0.6 and 0.8. We noted that larger learning rates took many epochs to reach their maximum

generalization accuracy, which was ultimately poor anyway. The small learning rates took longer to reach the same accuracy, but yielded no further improvement in accuracy. Those seeking simply for the learning rate that produces the fastest convergence would probably settle on a learning rate of 0.4. However, doing so would mean sacrificing generalization accuracy, which could be more efficiently achieved by using a larger learning rate: a balance between that trade-off is an ardent necessity. Unfortunately for these experiments we did not require the training process to converge, rather, the training process is used to perform a direct search of a model with superior generalization performance. Once again the statistical measures in Table 5.3 show relatively good performance for the ANN during the course of the investigations. Values of R and (R^2) are above 0.8 on both the train and test sets, indicating a good linear correlation and positive fit between the activations (predicted values) and targets (actual values). The ANN also did not show any signs of overfitting.

We now turn our discussion to the results shown in Figure 5.17 which exhibit the generalization performance of a two hidden layer ANN of architecture (5:35). The results in Figure 5.17 indicate that for the forecasting of a TCP/IP network traffic time series, the ANN was more sensitive to smaller leaning rates than larger ones. The generalization errors show that the ANN had significantly better generalization ability when trained on a learning rate of 0.01 than 0.1. This was more pronounced at 500 epochs. Although a learning rate of 0.1 outperformed the 0.01 learning rate between 750 to 800 epochs, we did not read much into this as it was short-lived. Statistical measures in Table 5.4 show extremely good results for the R and (R^2) values, with values above 0.8 for both learning rates examined.

To conclude with the discussion in this sub-section we examine the results of a different combination of learning rates and momentum heuristics given by various authors. The reason for this was to ascertain whether the effect of different learning rates on generalization ability is the same regardless of momentum. The results of this assessment are given in Figure 5.18. Several conclusions can be deduced from these results regarding the effect of different learning rate and momentum combinations on the generalization ability of the ANNs.

From the results we note that the 0.1-0.9 learning rate-momentum combination given by Foody et al. (1996) failed to produce satisfactory generalization performance. We note extensive

oscillatory behaviour in terms of generalization errors at those values of learning rate and momentum. As noted by Kavzoglu (1999) this could perhaps be attributed to the fact that the use of large momentum term increases the effect of oscillations by extending the steps taken in a faulty direction. Perhaps the ANN could have been stuck in a local minimum resulting from the selection of a large momentum term. It can be seen from Figure 5.18 that the best generalization results were overally produced by the combinations that used small learning rate-momentum combinations, such as 0.25-0.2 of Swinger (1996) and 0.1-0.3 of Ardö et al. (1997). In fact, for both cases, consistently good generalization results were produced. One other important observation we made was that the addition of the momentum term to the training considerably slows down the learning process. Although the selection of an appropriate combination of learning rate-momentum is a mammoth task, it appears that a very small learning rate, roughly 0.1, and a moderately high momentum term between 0.2–0.4, provided near-optimal solutions for the task considered.

Now to answer the fundamental question: How does the use of a momentum term affect the performance of the learning rate? From Figure 5.18, in general, the results obtained in these tests were in agreement with those obtained when a momentum value of 0.2 was used in previous experiments. However, the behaviour of the ANNs was less controlled with increasing momentum, as a result of the larger steps taken in weight space. In fact, when a momentum value of 0.8 was used in conjunction with a learning rate of 0.2, and when a momentum value of 0.9 was used in conjunction with learning rates of 0.1, the steps taken in weight space were too large, and divergent behaviour occurred during training. Discussion on the sizes of training epochs as a function of the network structure and effect on learning rate and momentum terms need further insight so that a more generalized result can be proposed. The statistical measures shown in Table 5 display R and R^2 values above 0.8, indicating that the inputs were sufficient in mapping the outputs. None of the ANNs examined in these experiments showed any signs of overfitting.

5.2.5 Training schedule

Finally we turn the subject of our discussion to the results regarding the effects the training schedule has on the generalization ability of ANNs in the task of forecasting a TCP/IP network traffic time series. We begin by discussing the results exhibited by a single hidden layer ANN having 60 hidden neurons which was trained for a various number of epochs. These results are

shown in Figure 5.19. The errors on the generalization curve reveal a general increase in generalization ability as the ANN is trained with an increasing number of iterations. This pattern however is valid up to 1000 iterations, after which the generalization performance takes on a negative trajectory. One would observe that the generalization errors immediately after 1000 epochs begin to increase, become constant between 4000 and 5000, marginally drop at 6000, and increase again after 6000, only to marginally dip at 9000 epochs. This indicates that after 1000 epochs the generalization errors were largely unstable and fluctuated quite extensively as the ANN struggled to find the best set of weights to approximate the model training parameters. Another interesting observation is that the generalization error generated at 3000 epochs is nearly equal to the generalization error incurred at 6000 epochs, illustrating that the addition of more training iterations did not assist the ANN to learn any better, in fact the latter only resulted in a more computationally expensive and time- consuming operation. Our results indicate that 1000 training iterations were sufficient to allow the model to reach an optimal solution, this in pure agreement with the empirical findings of Attoh-Okine (1999), who conducted similar investigations in forecasting stock prices and arrived at the same conclusions. Analysis of the statistical measures in Figure 5.19 shows that for up to 3000 epochs the ANN exhibited a positive linear relationship between the targets and activations as indicated by the R train and test values, which were all above 0.7. The results also show that for the same number of epochs a good fit between the predicted values and actual values was realized, judging by the R^2 values on the train and test set, which are above 0.8. After 3000 epochs, we note that although the R values on the train and test set, and the R^2 values on the train set remained sufficiently large, the R^2 values on the test set suffered hugely as a result of additional training iterations. This resulted in the ANN attaining negative R^2 values for networks trained for 8000, 9000 and 10 000 iterations. Suffice to say, contrary to the findings of Morgan and Bourlard (1989) the ANN did not seem to show any signs of overfitting even though it was "overtrained" for a considerable number of epochs - as much as 10 000 epochs. Although Morgan and Bourlard (1989) trained their ANNs for up to 280 000 epochs, we believe 10 000 epochs is reasonably sufficient to warrant any conclusions with regard to the relationship between ANN generalization and overtraining, especially considering that Richards (1991) in his quest to investigate the same question on the relationship between overtraining and ANN generalization trained his ANNs for

up to 50 000 epochs reaching an almost similar conclusion to ours. This leads us to the question: is there such a thing as overtraining?

Turning to the results of a two hidden layer model with a network architecture of (5:35), careful analysis of the generalization curve in Figure 5.20 shows the generalization errors gradually decreasing as the epochs are increased from 100 to 300. The generalization errors are then constant from 300 to 2000 epochs, after which a mild fluctuation in errors can be noted up to 7000. The errors begin to rise drastically, reaching a peak at 9000 epochs and then suddenly dropping at 10 000. This kind of behaviour, as of that exhibited by the ANN from 7000 to 10000 epochs, is nothing short of unordinary and erratic, and whilst we may not really be certain as to why the ANN behaved in the manner in which it did, one thing we are certain of is that it was not due to any form of overfitting as the ANN did not show any signs of overfitting throughout the course of these investigations. We also note that, although the generalization errors between 300 to 2000 epochs remain almost constant, one cannot ignore the marginal difference in generalization errors accrued by the network between 300 and 7000 epochs. It is safe to say that, within that range, the addition of more training iterations really did not have a significant impact on the generalization ability of the ANN. Perhaps Ockham was right in his assessment *"it is pointless to do with more what can be done with less"*(Zurada 1994). For further analysis of the model's prediction performance we examined the statistical measures. The statistical measures in Figure 5.20 on the other hand tell a completely different story. Apart from the training conducted at 9000 epochs, which resulted in a value of the R^2 on the test set below the nominal value of 0.7, the network exhibited a good linear correlation and a positive fit between the network activations (predicted values) and network targets (actual values). In conclusion, our results indicate that in forecasting TCP/IP traffic the epoch size chosen should be a function of the relative importance of training speed and generalization ability. An epoch size larger than 1000 is less likely to improve generalization ability but it will certainly increase training time (on average).

5.3 Conclusion

The major issues that this research has identified to be of primary importance for the generalization ability of ANNs have been investigated in this chapter. The chapter has presented ANN performance under several real world experimental configurations. Comprehensive

description and analysis of each experiment was conducted and results presented and analysed. The experiments were done under different settings and shared the same variables of measurements. The results were presented comprehensively in 20 conclusive graphs. Although a number of conclusions and observations can be deduced from the results of the experiments, they were not enumerated in this chapter in order to avoid possible repetition. Instead, they will be given in the next chapter, Chapter VI, since that chapter is intended to present some guidelines that will be largely extracted from the results given in this chapter. It is hoped that such guidelines derived from a vast number of experiments will be useful and beneficial for a wide variety of ANN applications in time series forecasting of TCP/IP network traffic.

Chapter 6: Conclusions and Future work

In research just as in life hindsight is always better than foresight. Our research has been an empirical investigation of the complex question: how do the size of the architecture of a network, size of the training set, size of the learning rate, choice of Backpropagation learning algorithm and schedule of training both individually and collectively affect the ability of an ANN to learn the training data and to generalize well to previously unseen data? Utilizing a real world TCP/IP network traffic time series we explored this question in depth by empirically conducting several experiments. This chapter presents a discussion of the findings of the research in parallel with the research objectives presented in the first chapter. Stating the extent to which the research statement has been fulfilled by providing the main achievements and conclusions of the research. A rundown of the thesis is presented and a comprehensive summary of the important results of the research is offered. In addition, the chapter concludes by describing potential subjects of interest for future research that could be extended and done in this area.

6.1 Specific conclusions

In this section we detail the specific conclusions with regards to our research objectives, the overall conclusions follow at the end of the chapter. The research goals provide the structure of this section, each research objective as stated in Chapter 1 is recalled, answered and discussed accordingly. As a whole, the research done and discussed in this thesis pertains to the research topic: "*The generalization ability of Artificial Neural Networks in forecasting TCP/IP network traffic trends"*. All the six chapters of this thesis explain in detail how the research was undertaken in providing comprehensive analysis and deductions to the research problem. The research findings have achieved research goals and made contributions to the existing volumes of knowledge on TSF using ANNs by providing and elaborating comprehensive conclusions and contributions to the problem statement defined in Chapter 1. The research goals, outlined in section 1.3, are revisited and followed in the same order according to the full scope of this research. Addressing the individual research goals, conclusions can be made for each one.

6.1.1 First objective of the research

- To determine the state of the art in TSF by reviewing ANNs.

The research came up with an extensive review of ANNs. The structure of MLPs as well as their advantages and limitations over the SLPs were reviewed accordingly in chapter 2. The Backpropagation algorithm which is the most common learning algorithm for training ANNs was equally reviewed. The research came up with the following general conclusions on ANNs.

- they consists of several simple processing elements called neurons;
- they are well suited for parallel computations, with each neuron operating independently of other neurons;
- they contain a high degree of interconnections between neurons;
- they contain links between neurons, each with a weight (scalar value) associated with it;
- they have adaptable weights that can be modified during training;
- they can be trained to do specific task, i.e. produce desired output for given input;
- the most popular architecture of ANNs is the MLP;
- the most popular algorithm to train ANNs is the Backpropagation;
- they have been used in a number of fields with a great deal of success.

6.1.2 Second objective of the research

- To determine the current methods of TSF

The research looked at the state of the art in time series analysis and forecasting. A number of different time series forecasting methods were reviewed, their advantages and limitations, mostly in comparison with the ANNs. The use of ANNs for this work was then justified by considering their strengths over the traditional and conventional methods of TSF. The research came up with the following conclusions on TSF methods both traditional and current.

- Time series methods are better suited to short-term forecasts (i.e., less than a year).
- TSF relies on sufficient past data being available and that the data is of a high quality and truly representative.
- Time series methods are best suited to relatively stable situations. Where substantial fluctuations are common and underlying conditions are subject to extreme change, then time series methods may give relatively poor results.
- ANNs unlike other methods can model any function without prior knowledge of the data generating process.

6.1.3 Third objective of the research

- The third objective of this research was to investigate how the generalization ability of an ANN is affected both collectively and individually by the following factors:

6.1.3.1 Size of network Architecture

The experimental results regarding the relationship between the size of the network architecture and the generalization exhibited by an ANN were discussed in section 5.2.1. Each of the ANNs when trained with a fixed training set, exhibited comparable levels of performance and generalization across a broad range of hidden units. ANNs having fewer hidden neurons generally did not generalize better than ANNs having more hidden neurons, even though the increase in hidden neurons implied a concomitant increase in total generalization ability of the ANN. Also, ANNs having two or three hidden layers did not perform significantly better than ANNs having one hidden layer. The following conclusions can be drawn from the results:

- Small ANNs learn tasks more quickly, but not necessarily better.
- Increasing the number of neurons improves the performance of the system. Nevertheless, too many neurons decrease the generalization ability of the model due to over-memorizing. This means that too many or too few neurons in the model is detrimental to the generalization ability of ANNs.
- Although the size of the ANN affects the generalization ability of an ANN, it has little or no influence on the goodness of fit and linear correlation of the models. In our experiments the R and R^2 values for the train and test sets indicate a good fit and linear correlation between the targets and activations despite the size of the network examined.
- When the number of neurons in the models is the same or fewer, the generalization ability of one-hidden layer ANN is better than that of two and three hidden layer models.
- Hecht-Nielsen (1987) provided a proof that a single hidden layer of neurons, operating a Sigmoidal activation function, is sufficient to model any solution surface of practical interest. Our results seem to support this conclusion as one hidden layer was sufficient for good generalization for our ANNs, further reaffirming the assertion made by Hush et al. (1992) regarding the sensitivity of ANNs to the number of hidden layers.
- Optimal network architecture is highly problem-dependent.

- Perhaps the most important conclusion derived from this study is that large networks do not always improve the generalization ability of ANNs.

6.1.3.2 Training set size

The experimental results regarding the relationship between the size of the training set and the generalization exhibited by an ANN are discussed in section 5.2.2. The results indicate that altering the size of the dataset used to train an ANN has an obvious impact on its generalization ability. Evidence has been presented that contradicts some of the current ANN design schools of thought, which imply that smaller quantities of training data necessarily produce better-quality forecasting models. The empirical results from this research indicate that frequently a larger quantity of training data will produce a better-generalized Backpropagation ANN model.

Stathakis (2009), who arrived at conclusions similar to our own, proposed that this trend is apparent mostly due to one of two reasons: 1) the ANN is suffering from overfitting when training on small data sets, or 2) the ANN is really being affected by the size of the data.

After carefully observing the performance plot during training of our ANNs it was determined that the ANNs in our research were in fact not suffering from the effects of overfitting and that it was instead the size of the training set that was actually causing the ANNs to perform poorly. We concluded therefore that the poor performance on small training sample sizes could be attributed to one of two possibilities: 1) the data set was so diverse that the ANNs could not represent it accurately without seeing 100% of the instances during training, 2) particular instances within the ANN are a better representation of the problem space than others, and if these instances are spread throughout the data set, the likelihood of their being selected for training is proportionate to the size of the subset.

Heuristics proposed to estimate the optimum number of training samples were compared, the results show that for the case study considered the heuristics proposed by Baum and Haussler (1989) $10 \times N_W$ and Klimasauskas (1993), $5 \times N_W$ are frequently good choices.

6.1.3.3 Size of learning Rate

The experimental results regarding the relationship between the learning rate and ANN generalization were discussed in section 5.2.4. We discovered that for a single hidden layer ANN, a learning rate of 0.6 gave the most reasonable generalization errors, particularly at an epoch size of 750, and a double hidden layer was more sensitive to larger learning rates than to

smaller ones. We also noted that smaller learning rates decreased the training time quite significantly.

With regard to the impact the momentum term has on the learning rate and generalization ability of ANNs, we discovered that a very small learning rate, roughly 0.1, and a moderately high momentum term between 0.2 and 0.4, provided near-optimal solutions for the task considered. However, the behaviour of the ANNs was less controlled with increasing momentum, as a result of the larger steps taken in weight space. We conclude that the degree of effects of learning rate and momentum on the model differ, as stated by Wilson and Martinez (2001). The learning rate is more powerful than momentum, as when a large learning rate and small momentum are achieved, the result is more precise than the opposite.

6.1.4 Fourth objective of the research

- The fourth objective of this research was to determine how much influence each of the following factors both individually and collectively have on the generalization ability of ANNs:

6.1.4.1 Learning algorithm

With respect to the results regarding the impact the choice of Backpropagation learning algorithm has on the generalization ability of ANNs, which are discussed in section 5.2.3, we reached several conclusions, including the fact that the Conjugate gradient backpropagation with Polak-Ribiére updates was the best- performing algorithm in terms of generalization ability of the ANNs. This trend became more pronounced as the size of the ANNs increased. We also discovered that all the Conjugate family of algorithms, *i.e.* Conjugate gradient backpropagation with Powell-Beale restarts, Conjugate gradient backpropagation with Polak-Ribiére updates and Conjugate gradient backpropagation with Fletcher-Reeves updates performed much better than all the other algorithms used in the study, a fact widely verified in literature (Joy 2011; Jacobs 1998).

As regards the limited investigation into the rate of convergence and accuracy of prediction of the different algorithms, we discovered that the Conjugate gradient backpropagation with Polak-Ribiére updates algorithm was also the fastest and most accurate. This was most probably due to

the fact that in the Conjugate family of algorithms the weights are adjusted along conjugate directions. That generally produces faster convergence and better accuracy than the rest of the BP algorithms (Karelson et al. 2006). We were careful not to draw extensive conclusions from these experiments due to the limited nature of our investigations.

6.1.4.2 Training schedule

The experimental results on the relationship between the training schedule and generalization performance of ANNs were discussed in section 5.2.5. From the results we conclude that for the case study considered, epoch size affected the model performance. Intuitively, more training iterations yielded a better generalization result. However, too many training iterations had a detrimental effect on the ANN generalization. An epoch size of 1000 was found to be sufficient for a single hidden layer ANN to attain good generalization, whilst for a double hidden layer model, an epoch size of 2000 seemed equally adequate to achieve the same. For the latter case, however, we noted that 300 epochs was in fact sufficient to warrant good generalization. With regard to the training schedule we also discovered that our ANNs did not suffer from any overfitting effects despite the fact of the ANNs being "overtrained" for 10 000 epochs. This led us to question the whole concept of network "overtraining", *does it really exist*?

6.2 Guidelines for effective use of ANNs in forecasting TCP/IP network traffic trends.

The conclusions produced and the experience gained during this study can be used to formulate a number of guidelines that can greatly facilitate the process of design of ANNs, especially for new users. These guidelines are particularly useful for defining the network structure and configuring the various learning parameters. It should be noted that these guidelines are specifically intended for a similar dataset and problems associated with forecasting TCP/IP network traffic trends, however general users of ANNs may also find them useful. Some suggestions are also made for post-processing to improve the generalization capabilities of networks. The list of guidelines is given as follows:

- **Size of network architecture**: An ANN large enough to learn the characteristics of the data is usually sufficient; under this setup we recommend a single hidden layer with roughly 60 hidden neurons as being sufficient from a generalization viewpoint. We recommend that for TCP/IP forecasting tasks one should start with a small number of

hidden neurons, preferably 20, and slightly increase the size of the network until no further improvement in generalization ability of the ANN is achieved. Although this number is not ideal for all situations, it could be used as a starting point for the search towards the optimum number of hidden layer neurons. We are not really keen on the idea of a second hidden layer but should one for any reason feel they should use a second hidden layer, we would suggest they use a ratio of first to second hidden neurons of 1:8, as this would warrant pretty good generalization ability. We refute the idea of using three hidden layers: it does not improve generalization performance by any standards and it is a total computational liability. We support the assertion made by Judd (1990) that for a given number of hidden units it is better to contain the units in a single hidden layer than to distribute them over two or more layers. In our view keeping the number of neurons to a minimum has several other advantages, as it (a) reduces the computational time needed for training, (b) helps avoid overfitting, and (c), allows the trained ANN to be analysed easily. As regards the input and output layers we advise one to define the number of input and output layer neurons by considering the nature of the problem and the availability of ground reference data.

- **Training set size**: Although it may not be possible to give a theoretical recommendation on the size of the training sample for all practical problems, our experimental results suggest that larger training sample sizes perform significantly better than smaller ones in as far as generalization ability is concerned. This however may introduce important practical implications as smaller ensembles require less computational efforts (Becker & Cun 1986). We found from both simulation experiments and literature that the best training sample size can be obtained by selecting training samples randomly, between the interval $5 \times N_W$ and $10 \times N_W$ in number, depending on the difficulty of the problem under consideration.
- **Size of learning rate**: Set the learning rate to 0.1 for the standard Backpropagation algorithm, and to either 0.1 or 0.2 if used in conjunction with the momentum term of 0.5 or 0.6. Do not set the momentum term too large as it would cause the ANN to be greatly unstable. If possible we advise minimal use of the momentum term as it greatly interferes with the training process of ANNs.

- **Learning algorithm**: If one is interested in the speed of convergence using the Conjugate gradient, Backpropagation with Polak-Ribiére updates algorithm is advisable, however if one is interested in an algorithm that has good generalization ability, any of the conjugate family of Backpropagation algorithms would be equal to the task.
- **Training epochs**: Training epoch size of between 1000 and 2000 training iterations is sufficient for the model to discover an appropriate set of weights for the model.
- **Initial weights**: Set the initial weights to a small range of between [-0.5, 0.5]. This range is quite effective for safeguarding against the effects of overfitting. Our ANNs did not show any signs of overfitting during the course of our investigations; this we suspect was largely attributed to good choice of initial weights. If the range of initial weights is set too small, training may be paralyzed. On the other hand, if the range is too large, premature saturation of the neurons may occur, which in turn will slow down training and result in the cessation of training at suboptimal levels (Hara et al. 1994) .
- **Test set***:* Employ an independent test set to evaluate the generalization performance of the ANN. Generalization will assist in indicating how the ANN is performing on datasets not seen. It is vital that this test set should not have been used as part of the training process in any capacity. Depending on the size of the data sample we advise using a test set not less than 5 % of the total data.
- **Size of the sliding time window**: Careful selection of the sliding time window is necessary for the efficient training of the ANN. Depending on the nature of the data under consideration it is wise to select time windows which are indicative of periods which could have profound influence on the activity of consecutive observations. The sliding time window should not be too large or too small. In our study we used information of the first 150 days to predict the next day iteratively. We also advise on use of a supervising script as it helps keep a record of the networks and their success.
- **Check for trends in the time series:** Until recently, the issue of stationarity has been rarely considered in the development of ANN models. However, there are good reasons why the removal of deterministic components in the data (i.e. trends, variance, seasonal and cyclic components) should be considered. It is generally accepted that ANNs cannot extrapolate beyond the range of the training data (Nguyen & Chan 2004) and (Plaut et al. 1986). Consequently, it is unlikely that ANNs can account for trends and

heteroscedasticity in the data. One way to deal with this problem is to remove any deterministic components using methods commonly used in time series modelling, such as classical decomposition or differencing. Matlab version 7 has an option in the Neural Network toolbox, '*Detrend data'* which helps you conduct this task with ease.

- **Normalization of the data: Normalize your data** always so that the ANN does not give unnecessarily high values of RMSE. Normalization is important in order to ensure that all variables receive equal attention during the training process. In addition, the variables have to be scaled in such a way as to be commensurate with the limits of the activation functions used in the output layer. If using software like Matlab version 7, this will be done automatically for you.
- **Check for the effects of overfitting:** This is necessary so that training may be stopped when the ANN begins to overfit the data. In our study this was done by trailing the performance plot curve on the train set. We believe this is a better indicator of the effects of overfitting than a cross validation set which might deplete already insufficient data.

6.3 Future work and recommendations

As with almost any area of research, progress leads toward more questions. Based on the research carried out in this study, our results suggest considerable potential for future work. We plan on extending our investigations to new self- similar and chaotic time series and to other ANN models and learning parameters. In addition, more testing is needed to evaluate the applicability of our guidelines to other datasets to be able to make claims about their robustness and to validate the effectiveness of the conclusions reached in this research. In order to improve and extend the investigations reported in this work, in addition to constant learning rates, the use of adaptive learning rate strategies could be examined and their generalization performance results compared to those produced by their counterparts. Another issue which we can possibly look at is using a variable momentum value. A variable momentum value is currently being researched and its impact upon the generalization error is unknown to date. Also, the effect of employing different transfer functions, such as the Gaussian basis and tangent hyperbolic function in the learning process, which are also reported to have significant effect on ANN performance, needs investigating (Sibi et al. 2013). Throughout the duration of these investigations we did not have a validation data set due to the slightly limited size of our time series data, hence it would be a good idea to look into the effectiveness of Early stopping and

other techniques such as Bayesian regularization, which have also been widely reported in improving the generalization ability of ANN (Pissarenko 2002; Noriega 2005). Also, Richards (1991) suggests that although the size of the training set is of considerable importance, the characteristics and the distributions of the data as well as the sampling strategy used are crucial. Simply stated this means that not only the quantity but also the quality of the training set is crucial for a successful ANN application. Any future investigations carried out on the training set should take into consideration that valuable fact, as the quality of the training set could be a potential game changer in many respects. We intend to look into this area quite extensively in our future explorations. As this study was limited to Feed-forward ANN learning problems with the Backpropagation learning algorithm, it could be also beneficial to investigate the effects of the network structure and the learning parameters on the performance and generalization ability of other ANN models, including Self Organizing Maps (SOM) and Learning Vector Quantization (LVQ), with the aim of deriving some general conclusions that can be used to construct guidelines for users in the design of these particular network models.

6.4 Overall conclusions

ANNs have been widely used in an extensive number of applications. However, the focus in this study has been on the forecasting of a real world TCP/IP network traffic time series. It is the case that ANNs are more robust than conventional statistical techniques. Therefore they are of great importance for forecasting studies. It is believed that ANNs will continue to maintain their importance and validity for time series forecasting problems in the future despite the advent of new and sophisticated methods, such as decision trees and genetic algorithms. This research sought to strengthen their importance by providing extensive analyses on the effect of the size of the network architecture, size of training set, learning algorithm, training schedule and size of learning rate on their ability to perform and generalize well to new data. At the beginning of this research all previous experience and research seemed to indicate that the most significant effect on network generalization would be that resulting from the size of the architecture and the training set. At the end of the project and after extensive explorations, it is clear, the choice of learning algorithm, the size of the learning rate and the schedule of training have as much effect on the ability of the ANNs to generalize to novel data. It is hoped that this study makes some contributions to the understanding of the role of ANNs in TCP/IP forecasting, and will be beneficial for their design and use. By applying the suggestions made in this research, more

accurate forecasting results and better generalization ability can be produced. Finally, by evaluating the impact of the choices of network architecture, size of the training set, learning algorithm, training schedule and learning rate, it is hoped that users of ANNs will have a clearer idea of the way these networks function, so that they are no longer considered to be 'black-boxes'.

References

Aamodt, R., 2010. *Using Artificial Neural Networks To Forecast Financial Time Series*. Norwegian university of science and technology.

Abbas, Q., Haider Bangyal, W. & Ahmad, J., 2013. The Impact of Training Iterations on ANN Applications Using BPNN Algorithm. *International Journal of Future Computer and Communication*, 2(6), pp.567–569. Available at: http://www.ijfcc.org/index.php?m=content&c=index&a=show&catid=43&id=492 [Accessed July 18, 2014].

Ahmad, I., Ansari, M.A. & Mohsin, S., 2008. Performance Comparison between Backpropagation Algorithms Applied to Intrusion Detection in Computer Network Systems. , pp.231–236.

Ahmad, S. & Tesauro, G., 1998. Scaling and Generalization in Neural Networks. In *International Conference on Neural Infromation presseing systems*. pp. 177–185. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=134256.

Allacorn -Quiono, V. & Arria, J.., 2006. Multi- resolution FIR neural-network-based learning algo- rithm applied to network traffic prediction. *IEEE Transactions on Systems, Man and Cybernetics – Part C*, 3(6), pp.208–220.

Ardö, J., Pilesjö, P. & Skidmore, A., 1997. Neural Networks, multitemporal Landsat Thematic Mapper data and topographic data to classify forest damages in the Czech Republic. *Canadian Journal of Remote Sensing*, 23(2), pp.217–229.

Atiya, a & Ji, C., 1997. How initial conditions affect generalization performance in large networks. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 8(2), pp.448–51. Available at: http://www.ncbi.nlm.nih.gov/pubmed/18255648.

Attoh-Okine, N.O., 1999. Analysis of learning rate and momentum term in backpropagation Neural Network algorithm trained to predict pavement performance. *Advances in Engineering Software*, 30(4), pp.291–302. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0965997898000714.

Badri, L., 2010. Development of Neural Networks for Noise Reduction. *International Arab Journal of Infromation Technology*, 7(3), pp.289–294.

Baldi, P. & Hornik, K., 1989. Determining the weights of a Fourier series neural network on the basis of the multidimensional discrete fourier transform. *Journal of Neural Networks*, 2(2), pp.53–58.

Balestrassi, P.P. et al., 2009. Design of experiments on Neural Network's training for nonlinear time series forecasting. *Neurocomputing*, 72(4-6), pp.1160–1178. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0925231208001513 [Accessed July 18, 2014].

Barabas, M. et al., 2011. Evaluation of network traffic prediction based on neural networks with multi-task learning and multiresolution decomposition. In *2011 IEEE 7th International Conference on Intelligent Computer Communication and Processing*. Ieee, pp. 95–102. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6047849.

Barry, R., 2000. Artificial Neural Network prediction of wavelet sub bands. University of Manitoba.

Bartlett, M.S., Movellan, J.R. & Sejnowski, T.J., 2002. Component Analysis. In *IEEE Transactions of neural networks*. pp. 1450–1464.

Batbayar, B., 2008. *Improving Time Efficiency of Feedforward Neural Network Learning*. National University of Mongolia.

Baum, E. & Haussler, D., 1989. What size net gives valid generalization. *Neural Computing*, 1(1), pp.159–161.

Bebis, G. & Georgiopoulos, M., 1997. Optimal feed-forward Neural Network architectures. In *IEEE Potentials*. pp. 27–31.

Becker, S. & Cun, L., 1986. Improving the Convergence of Back- Propagation Learning with Second Order Methods. In *Proceedings of the 1988 Connectionist Summer School*. pp. 67–56.

Bretscher, P. & Cohn, M., 1970. A Theory of Self-Non-self Discrimination. *Science Journal*, 1(69), pp.1042–1049.

Canuto, D.P., 2001.Combining Neural Networks and Fuzzy logic for applications in character recognition. University of Kent.

Cardoso, D.O. et al., 2011. Clustering data streams with weightless Neural Networks. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning.* pp. 27–29.

Cawley, G.C., Edgington, M. & Heath, M., 1993. Generalization in Neural speech synthesis. *In Proceedings of the world Conference on Neural Networks*. Oregon,USA, pp. 122–125.

Chabaa, S., 2010. Identification and Prediction of Internet Traffic Using Artificial Neural Networks. *Journal of Intelligent Learning Systems and Applications*, 02(03), pp.147–155. Available at: http://www.scirp.org/journal/PaperDownload.aspx?DOI=10.4236/jilsa.2010.23018 [Accessed July 18, 2014].

Chakraborty, D. & Pal, N.R., 1997. Expanding the Training Set for Better Generalization in MLP. *Electronics and Communications*, 3(1), pp.6–9.

Chaoba, A., 2009. Internet traffic modelling and forecasting. Kansas state university.

Chen, C.J. & Miikkulainen, R., 2001. Creating Melodies with Evolving Recurrent Neural Networks. *In Proceedings of the 2001 International Joint Conference on Neural Networks*, 2(2), pp.20–60.

Cherkassky, V. & Zhong, S., 2001. Factors controlling generalization ability of MLP networks. In *International Joint Conference on Neural Networks. Proceedings.* Ieee, pp. 625–630. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=831571.

Chester, D., 1990. Why two hidden layers are better than one. In *Proceedings of the International Joint Conference on Neural Networks*. Washington, USA, pp. 265–268.

Choudhury, T. a., Hosseinzadeh, N. & Berndt, C.C., 2012. Improving the Generalization Ability of an Artificial Neural Network in Predicting In-Flight Particle Characteristics of an Atmospheric Plasma Spray Process. *Journal of Thermal Spray Technology*, 21(5), pp.935–949. Available at: http://link.springer.com/10.1007/s11666-012-9775-9 [Accessed July 18, 2014].

Churchland, P.S. & Sejnowski, T.J., 2005. Neural representation and Neural computation. *Philosophical perspectives*, 4(1990), pp.343–382.

Control, E., 2010. DSMS for monitoring home electricity change. Arab Open university.

Conway, A.J., 1998. Time series , Neural Networks and the future of the Sun. *New Astronomy Reviews*, 42(June), pp.343–394.

Cortez, P. et al., 2006. Internet Traffic Forecasting using Neural Networks. *IEEE International Joint Conference on Neural Network Proceedings*, 2(september), pp.2635–2642. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1716452.

Cortez, P. et al., 2010. Multi-scale Internet traffic forecasting using Neural Networks and time series methods. *Expert Systems*, 29(2), p.no–no. Available at: http://doi.wiley.com/10.1111/j.1468-0394.2010.00568.x [Accessed July 18, 2014].

Coulibaly, P., Anctil, F. & Bobe, B., 2000. Daily reservoir inflow forecasting using artificial neural networks with stopped training approach. *Journal of Hydrology*, 230(January-February), pp.244–257.

Cruz, S., 1989. Learnability and the Vapnik-Chervonenkis Dimension. *Journal of Artificial intelligence*, 36(4), pp.929–965.

Cybenko, G., 1989. Approximation by superpositions of a sigmoidal function. *Maths control and signal systems*, 2(2), pp.303–314.

David, J. & MacKay, C., 2007. Bayesian interpolation. *Systems Computation and Neural*, 2(3), pp.139–174.

Eberhart, R., Kennedy, J. & Dobbins, R.W., 1990. *Neural Network PC Tools - A Practical Guide*, London,GB: London: Academic Press, Inc.

Engelbreetch, A., 2007. Computational Intelligence. , pp.60–67.

EngelBretch, P., Geldenhyus, J. & Cloete, 1, 1995. An investigation of Neural Networks for linear time-series forecasting. *Automatic scaling in Artificial Neural Networks*, pp.1–5.

Eskander, G.S. et al., 2007. Round Trip Time Prediction Using the Symbolic Function Network Approach. *2007 International Symposium on Information Technology Convergence (ISITC 2007)*, 3(4), pp.3–7. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4410595.

Esugasini, S. et al., 2005. Using Various Back Propagation Algorithms. *Neural computing*, 2(2), pp.123–130.

Fearn, C., 2004. *The generalization ability of Neural Networks*. New York,USA: Elsevier B.V.

Flood, I. & Kartam, N., 1994. Neural networks in civil engineering. In *Principles and understanding*. J Comp civil engineers, pp. 669–682.

Foody, G., Lucas, R. & Curran, M., 1996. Estimation of the areal extent of land cover classes that only occur at a sub-pixel level. *Canadian Journal of Remote Sensing*, 22(4), pp.428–432.

Fukumizu, K., 1992. Active Learning in Multilayer Perceptrons. *Advances in Neural Information Processing Systems*, 8(2), pp.295–301.

Gallagher, M. & Downs, T., 1997. Visualisation of learning in Neural Networks using principal component analysis. In *Proceedings of International Conference on Computational Intelligence and Multimedia Applications*. pp. 327–331.

Garson, G., 1998. *Neural Networks: An Introductory Guide for Social Scientists*, SAGE Publications).

Gay, L.R., 1992. *Educational research* 4th ed., New York,United states of America: Merrill.

Gers, F., 2001. *Long Short-Term Memory in Recurrent Neural Networks*. Ecole Polytechnique federale de la Usanne.

Gonzalez, S., 2000. *Neural Networks for Macroeconomic Forecasting.*, Canada.

Gopal, S. & Woodcock, C., 1996. Remote sensing of forest change using Artificial Neural Networks. In *IEEE Transactions on Geoscience and Remote Sensing*. pp. 398–403.

Gorman, P. & Sejnowski, T., 1998. Analysis of hidden units in a layered network trained to classify sonar targets. *Journal of Neural computing*, 1(1), pp.75–89.

Govy, P., Pu, R. & Chen, J., 1996. Mapping ecological land systems and classification uncertainties from digital elevation and forest-cover data using Neural Networks. *Photogrammetric Engineering and Remote Sensing*, 6(2), pp.124–129.

Grimm, G. & Yarnold, P.(, 1997. Reading and Understanding Multivariate Statistics. *Canadian Journal of Remote Sensing*, 4(2), pp.143–148.

Guang, S., 2013. Network Traffic Prediction Based on the Wavelet Analysis and Hopfield Neural Network. , 2(2), pp.101–105.

Hagan, M.T. & Menhaj, M.B., 1994. Training Feedforward Networks. In *IEEE Transactions on Neural Networks*. pp. 989–992.

Halenár, I. & Libošvárová, A., 2012. The Impact of the Neural Network Structure by the Detection of Undesirable Network Packets. In *Proceedings of the world conference on Engineering and Computer sciences*.

Hara, Y. et al., 1994. Application of Neural Networks to radar image classification. In *IEEE Transactions on Geoscience and Remote Sensing*. p. 1994.

Hara, Y. et al., 1994. Application of Neural Networks to radar image classification. In *IEEE Transactions on Geoscience and Remote Sensing*. pp. 100–109.

Hartono, P. & Hashimoto, S., 2007. Learning from imperfect data. *Applied Soft Computing*, 7(1), pp.353–363. Available at: http://linkinghub.elsevier.com/retrieve/pii/S1568494605000773 [Accessed July 18, 2014].

Hasegawa, A., Itoh, K. & Ichioka, Y., 1996. Generalization of shift invariant Neural Networks: Image processing of corneal endothelium. *Neural Networks*, 9(2), pp.345–356. Available at: http://linkinghub.elsevier.com/retrieve/pii/0893608095000542.

Hashash, Y., Jung, S. & Ghabousi, J., 2004. Numerical Implementation of a neural network based material model in finite element anslysis. *International Journal for numerical Methods in Engineering*, 59(7), pp.989–1005.

Haykin, S., 1999. *Neural Networks: A comprehensive foundation* second. Pearson, ed., Pearson.

Hecht-Nielsen, R., 1987. Kolmogorov's mapping neural network existence theorem. In *Proceedings of the First IEEE International Conference on Neural Networks*. San Diego, CA, USA, pp. 11–14.

Hessami, M., Anctil, F. & Viau, A.A., 2004. Selection of an Artificial Neural Network model for the Post-calibration of Weather Radar Rainfall Estimation. *Journal of Data Science*, 2(November), pp.107–124.

Hillberg, P.A., Sengupta, S. & Til, R.P. Van, 2009. A Comparative Study of Three Predictive Tools for Forecasting a Transfer Line ' s Throughput. , 16(1), pp.32–40.

Hinton, G., 1988. Connectionist learning procedures. *Journal of Artificial intelligence*, 3(2).

Hirose, Y., Yamashita, K. & Hijiya, K., 1991. Back-propagation algorithm which varies the number of hidden units. *Neural Networks*, 4(2), pp.61–67.

Hong, X., Panchi, L. & MAJON, C., 2010. Process Neural Network Algorithm Based on Piecewise Linear Interpolation Function. In *2nd international conference on information technology and technology*. pp. 1–4.

Hudson, M., Hag, M. & Demuth, H., 2014. *Neural Network Toolbox™ User's Guide* 8.2 ed., New York,United states of America: The MathWorks, Inc.

Hush, D., 1989. Classification with Neural Networks. In *Proceedings of the IEEE International Conference on Systems Engineering*. Dayton, Ohio, USA, pp. 50–57.

Hush, D. & Horne, B., 1993. Progress in supervised Neural Networks. *IEEE Signal Processing Magazine*, pp.8–39.

Hush, D.R., Horne, B. & Salas, J.M., 1992. Error surfaces for multilayer perceptrons. In *IEEE Transactions on Systems, Man and Cybernetics*. Ohio,USA.

Hussain, T.S. & Browse, R.A., 1998. Genetic Encoding of Neural Networks using Attribute Grammars. In *Proceedings of the 2010 Interservice/Industry Training, Simulation and Education Conference*. pp. 1550–1558.

Jacobs, R., 1998. Increased rates of covergence through learning rate adaptation. *Neural Networks*, 1(4), pp.295–308. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=938431.

Jain, A., 1996. Artificial Neural Networks:A tutorial. *Journal of Neural Computing*, 2(2), pp.400–409.

Jiang, J. & Apavasillou, S., 2004. Detecting network attacks in the internet via statistical network traffic normality prediction. *Journal of Network and Systems Management*, 1(2), pp.51–72.

Joy, C.U., 2011. Comparing the Performance of Backpropagation Algorithm and Genetic Algorithms in Pattern Recognition Problems. *International Journal of Computer Information Sstems*, 2(5), pp.7–12.

Judd, S., 1990. *Neural Network design and the complexity of learning*, Massachutes,USA: The MIT press publishers.

Kaastra, L. & Boyd, M., 1996. Designing a Neural Network for forecasting financial and economic time series. *Neurocomputing*, 10(2), pp.215–236.

Kanellopoulos, I. & Wilkinson, J., 1997. Strategies and best practice for Neural Network image classification. *International Journal of Remote Sensing*, 18(4), pp.711–725.

Karelson, M. et al., 2006. Neural networks convergence using physicochemical data. *Journal of chemical information and modeling*, 46(5), pp.1891–7. Available at: http://www.ncbi.nlm.nih.gov/pubmed/16995718.

Karsoliya, S., 2012. Approximating Number of Hidden layer neurons in Multiple Hidden Layer BPNN Architecture. *International Journal of Engineering Trends and Technology*, 3(6), pp.714–717.

Katidiotis, A., Tsagkaris, K. & Demestichas, P., 2000. *Performance Evaluation of Artificial Neural Network - based Learning Schemes for Cognitive Radio Systems*,

Kavzoglu, T., 1999. Determining Optimum Structure for Artificial Neural Networks. In *Proceedings of the 25th Annual Technical Conference and Exhibition of the Remote Sensing Society*. Cardiff, UK, pp. 675–682.

Kim, D., 2005. Forecasting Solar Cycle 24 using Neural Networks. *Journal of the Eastern Asia Society for Transportation Studies,*, 6(2), pp.2629–2638.

Kiran, N.V.N.I., Devi, M.P. & Lakshmi, G.V., 2010. Training Multilayered Perceptions for Pattern Recognition : A Comparative Study of Three Training Algorithms. *International Journal of information technology and Management*, 2(July-December), pp.579–584.

Klevecka, I., 2009. Forecasting Network Traffic: A Comparison of Neural Networks ad Linear models. In *Proceedings of the 9th international Conference "Realibility and Statistics in Transportation and Communication."*pp. 170–177.

Klimasauskas, C., 1993. *Applying Neural Networks* R. Turban & E. Trippi, eds.,

Krzyovt, H., 2008.Determining the weights of a Fourier series Neural Network on the basis of a tri Fourier analysis. *Journal of applications*, 18(3), pp.369–375.

Kudryci, T., 1988. *Neural Network implementation of a medical diagnosis system*. Cincinatti University.

Kvale, D.T., 2010. *Artificial Neural Networks Based Approaches for modelling Radiated Emisssions from Printed Circuit Board Structures and Shields*. The university of Toledo.

Kwon, A., 1991. *Investigation of a Neural Network Methodology to predict Transient Performance in FMS*. Oklahoma State University.

Lahmiri, S., 2011.A comparative study of Backpropagation algorithms in financial prediction. *International Journal of Computer science, Engineering and Applications*, 1(4), pp.15–21.

Lange, N., Bishop, C.M. & Ripley, B.D., 1997. Neural Networks for Pattern Recognition. *Journal of the American Statistical Association*, 92(440), p.1642. Available at: http://www.jstor.org/stable/2965437?origin=crossref.

Lange, R. & Manner, R., 1994. Quantifying a critical training set size for generalization and over fitting using teacher Neural Networks. In *Proceedings of the international conference on Artificial Neural Networks*. London,GB, pp. 497–500.

Lapedes, A. & Faber, R., 1987. *Non-Linear Signal Processing Using Neural Networks: Prediction and System Modelling*, Los Alamos National Laboratory, USA,.

Lawrence, S., Giles, C.L. & Tsoi, A.C., 1996. What Size Neural Network Gives Optimal Generalization ? Convergence Properties of Backpropagation. *Neural computing*, 2(2), pp.1–35.

Leung, H. & Zue, W., 1989. On the Generalization Capability of Multi-Layered Networks in the Extraction of Speech Properties. In *International Conference on Acoustics,Speech and Signal Processing*. New York,USA, pp. 422–425.

Lippmann, R., 1987. An introduction to computing with Neural Nets. *EEE ASSP Magazine*, pp.4–22.

Liyong, S., Shen, Y. & Ma, J., 2007. Local spatial properties based image interpolation using Neural Neworks. In *4th international symposium on neural networks*. pp. 875–878.

Lodwich, A., Rangoni, Y. & Breuel, T., 2009. Evaluation of robustness and performance of Early Stopping Rules with Multi Layer Perceptrons. In *Proceedings International Joint Conference on Neural Networks*, 2(4), pp.1877–1884. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5178626.

Mahmoud, O., Anwar, F. & Salami, M.J.E., 2007.On Multilayer feedforward Artificial Neural Networks performance. *Journal of Engineering Science and Technology*, 2(2), pp.188–199.

Maier, H. & Dandy, G., 1998. The effect of internal parameters and geometry on the performance of back-propagation neural networks: an empirical study. *Environmental Modelling and Software*, 13(2), pp.193–209. Available at: http://linkinghub.elsevier.com/retrieve/pii/S1364815298000206.

Maier, H.R. & Dandy, C., 1999. Empirical comparison of various methods for training feed-forward Neural Networks for salinity forecasting in the River. *Water Resources Journal*, 35(8), pp.2591–2596.

Maier, H.R. & Dandy, G.C., 1997. Determining Inputs for Neural Network Models of Multivariate Time Series. *Computer-Aided Civil and Infrastructure Engineering*, 12(5), pp.353–368. Available at: http://doi.wiley.com/10.1111/0885-9507.00069.

Martin, G. & Pittmann, J., 1990. Recognizing hand printed letters and digits. *In advances in Neural information processing systems.*, 2(2), pp.405–415.

Masters, T., 1993. *Practial Neural Network Recipes in C++*, Toronto, ON: Academic Press, Inc.

Mencer, F. & Parisi, D., 1992. Genetic evolution of the topology and weight distribution of neural networks. *Biological Cybernetics*, 6(6), pp.238–289.

Mi, X. et al., 2005. Testing the generalization of Artificial Neural Networks with cross-validation and independent-validation in modelling rice tillering dynamics. *Ecological Modelling*, 181(4), pp.493–508. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0304380004003746 [Accessed July 13, 2014].

Mitchell, T., 1997. *Machine learning* E. Munson, ed., McGraw Hill Publishers.

Mohammad, O. & Luis, J., 2011. Performance evaluation of communication networks and systems. *Journal of Networks*, 6(4), pp.533–540.

Moore, D. & McCabe, D., 1993. *Introduction to the practice of statistics.*, New York,United states of America: Freeman.

Morgan, N. & Bourlard, H., 1989. Generalization and parameter in Feedforward Nets: Some Experiments. *In advances in neural network processing systems.* Berkeley,USA, pp. 630–638.

Moustafa, A.A. et al., 2011. Performance Evaluation of Artificial Neural Networks for Spatial Data Analysis. *WSEAS Transactions on Computers*, 10(4), pp.115–124.

Nakamura, E., 2005. Inflation forecasting using a Neural Network. *Economics Letters*, 86(3), pp.373–378. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0165176504003088.

Nakayama, K., Hirano, A. & Fukumura, K. -i., 2004. On generalization of Multilayer Neural Network applied to predicting protein secondary structure. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*. Ieee, pp. 1209–1213. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1380114.

Neeharika, A., 1996. Generalization capability of feedforward Neural Networks for pattern recognition tasks. Indian Institute of Technology.

Nelson, M., Hill, T. & Remus, W., 1999. Time series forecasting using Neural Networks: should the data be deseasonalized first? *Journal of forecasting*, 18(5), pp.359–367.

Nguyen, H.H. & Chan, Æ.C.W., 2004. Multiple Neural Networks for a long term time series forecast. *Neural Networks*, 3(4), pp.90–98.

Noriega, L., 2005. Multilayer Perceptron Tutorial. , pp.1–12.

Paola, J.D. & Schowengerdt, R.A., 1997. The Effect of Neural-Network Structure on a Classification. *Neural computing*, 85721(2), pp.1–7.

Philip, N., 2001. *Studies in Artificial Neural Network Modelling*. Cochin University of science and Technology.

Piotrowski, A.P. & Napiorkowski, J.J., 2013. A comparison of methods to avoid overfitting in Neural Networks training in the case of catchment runoff modelling. *Journal of Hydrology*, 476, pp.97–111. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0022169412008931 [Accessed July 18, 2014].

Piotrowski, A.P. & Napiorkowski, J.J., 2011. Optimizing Neural Networks for river flow forecasting – Evolutionary Computation methods versus the Levenberg–Marquardt approach. *Journal of Hydrology*, 407(1-4), pp.12–27. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0022169411004501 [Accessed July 18, 2014].

Pissarenko, D., 2002. *Neural Networks For Financial Time Series Prediction :* Univesity of london.

Plaut, D., Nowlan, S. & Hinton, G., 1986. *Experiments on Learning by Back Propagation*,

Poh, H., Yao, J. & Jas, T., 1998. Neural Networks for the Analysis and Forecasting of Advertising and. *International Journal of intelligent systems in accounting,Finance and management*, 7(June), pp.253–268.

Rasmussen, C.E., 1993. *Generalization in Neural Networks*, Denmark.

Reed, R., 1993. Pruning and Regularization Techniques for Feed Forward Nets applied on a Real World Data Base. In *IEEE Transactions on Neurla Networks*. pp. 201–215.

Renals, S., Morgan, N. & Cohen, M., 1992. Connectionist Probability is the decipher speech recognition system. In *Proceedings IEEE International Conference on Acoustics,Speech and signal processing*. pp. 601–604.

Richards, E., 1991. *Generalization in Neural Networks,Experiments in Speech Recognition*. University of Colarado.

Ripley, B., 1993. *Statistical aspects of Neural Networks* O. Barndorff-Nielsen, J. Jensen, & W. Kendall, eds.,

Rivas, V.M., Merelo, J.J. & Castillo, P.A., 2004. Evolving RBF neural networks for time-series forecasting with EvRBF. *IEEE Transactions on acoustics,signal and hearing*, 165(September 2003), pp.207–220.

Rowley, H.A. & Shumeet, B., 1996. Neural Network-Based Face Detection. *Computer Vision and Pattern Recognition*, 2(November), pp.1670–1674.

Rummelhart, D., 1967.The effects of inter-presentation intervals on performance in a contiua. Stanford University.

Sarle, S. & Warren, T., 1991. Neural Networks and Statistical Models. In *Proceedings of the 1991 Connectionist Models Summer School,*. San Mateo,Carlifornia, pp. 1538–1550.

Schneider, W. & Bishof, H., 1992. Improving neural network generalisation. *1995 International Geoscience and Remote Sensing Symposium, IGARSS '95. Quantitative Remote Sensing for Science and Applications*, 30(3), pp.1255–1257. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=521718.

Seung, H., Opper, M. & Sompolinsky, H., 1992. Query by Committee. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*. pp. 287–299.

Shavlik, J.W., Mooney, R.J. & Towell, G.G., 1991. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6(2), pp.111–143. Available at: http://link.springer.com/10.1007/BF00114160.

Sibi, P., Jones, S. & Siddarth, P., 2013. Analysis of different activation functions using back propagation neural networks. *Journal of theoretical and applied Information Technology*, 47(3), pp.1264–1268.

Sietsma, J. & Dow, R., 1988. Neural net pruning - why and how ? In *Proceedings of IEEE International Conference on Neural Networks*. San Diego, CA, USA, pp. 325–333.

Singh, S., Bhambri, P. & Gill, J., 2011. Time Series based Temperature Prediction using Back Propagation with Genetic Algorithm Technique. *International Journal of Computer science*, 8(5), pp.28–32.

Sola, J. & Sevilla, J., 1997. Importance of input data normalization for the application of neural networks to complex industrial problems. In *IEEE Transactions on Nuclear Science*. pp. 1464 – 1468.

Sontag, E., 1992. Feedback stabilization using two hidden layer nets. *IEEE Trans.Neural Networks*, 3(6), pp.34–60.

Srinivasan, D. & Liew, A., 1994. A Neural Network short term load forecaster. *Electrical power systems research*, 4(28), pp.227–234.

Stathakis, D., 2009. How many hidden layers and nodes? *International Journal of Remote Sensing*, 30(8), pp.2133–2147. Available at: http://www.tandfonline.com/doi/abs/10.1080/01431160802549278 [Accessed July 15, 2014].

Staufer, P. & Fischer, M.M., 1997. *Spectral pattern recognition by a two-layer perceptron: effects of training set size* I. Kanellopoulos & G. . Wilkinson, eds., London,GB: London: Springer.

Svozil, D., Kvasnieka, V. & Pospichal, J., 1997. Introduction to Multi-layer feed-forward neural networks. *Introduction to multi layer feed forward Neural Networks*, 39(October-June), pp.43–62.

Swinger, K., 1996. Financial prediction. *Journal of Neural Computing and Applications*, 4(4), pp.192–197.

Swingler, K., 1996. *Applying Neural Networks:A practical Guide*, New York,USA: Harcourt Brace and Company.

Tan, J.Y.B., Bong, D.B.L. & Rigit, A.R.H., 2012. Time Series Prediction using Backpropagation Network Optimized by Hybrid K-means-Greedy Algorithm. *Engineering Letters*, 3(2), pp.18–20.

Teixeira, J.P. & Fernandes, P.O., 2012. Comparison of Artificial Neural Network architectures in the Task of Tourism Time Series Forecast. *World academy of science,Engineering and Technology*, 66(5), pp.978–983.

Tong, H. et al., 2005. Internet Traffic Prediction by W-Boost : Classification and Regression ∗. *Neural computing*, 2(973), pp.397–402.

Tong, H., 1983. *Threshold Models in Non-Linear Time Series Analysis*, New York,USA: Springer-Verlag.

Tsai, C.-Y. & Lee, Y.-H., 2011. The parameters effect on performance in ANN for hand gesture recognition system. *Expert Systems with Applications*, 38(7), pp.7980–7983. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0957417410014491 [Accessed July 18, 2014].

Tsilo, L.C., 2008. Protein secondary structure prediction using Neural Networks. Rhodes University.

Uwahuro, J., 2008. *Forecasting Solar Cycle 24 using Neural Networks*. Rhodes University.

Vapnik, V. & Cervonenkis, A., 1968. *The uniform convergence of frequencies of the appearance of events to their probabilities*, Russia: Nauk SSSR.

Vogl, T. et al., 1988. Accelerating the Convergence of the Back-Propagation Method. *Biological Cybernetics*, Vol 5(9), pp.257–263.

Wagner, D. & Gross, E.., 1996. KD trees and Delaunay-based linear interpolation for function learning: a comparison to neural networks with error backpropagation. In *IEEE Transactions on control systems*. pp. 649–693.

Waibel, A., Hanzawa, T. & Koyihuro, S., 1989. waibel1989.pdf. In *IEEE Transactions on acoustics,signal and hearing*. Osaka,Japan, pp. 328–350.

Walleed, A.., Rashhed & Hamza, A., 2009. Generalization Aspect of Neural Networks on Upgrading Assimilation Structure into Accommodating Scheme Waleed , A . J . Rasheed and Hamza A . Ali Department of Software Engineering , Faculty of Computer Science ,. *Journal of Computer sciences*, 5(3), pp.177–183.

Wan, W. et al., 2009. Enhancing the generalization ability of Neural Networks through controlling the hidden layers. *Applied Soft Computing*, 9(1), pp.404–414. Available at: http://linkinghub.elsevier.com/retrieve/pii/S1568494608000902 [Accessed July 18, 2014].

Wang, C., Hang, Z. & Heng, L., 2008. An internet traffic forecasting model adopting radi- cal based on function Neural Network optimized by genetic algorithm. In *Proceedings of IEEE Workshop on Knowledge Discovery and Data Mining (WKDD08)*. Adelaide, Australia, pp. 367–370.

Wang, L. & Zhou, N., 2005. Optimal Size of a Feedforward Neural Network : How Much does it Matter ? In *Proceedings of the joint International conference on Autonomous and Automatic Systems and international Conference on networking and Services*. pp. 300–306.

Wei, H., Yoshiteru, N. & Shouyang, W., 2004. A general approach based on autocorrelation to determine input variables of Neural Networks for Time Series Forecasting. *Journal of Systems Science and Complexity*, 17(3), pp.297–305.

Weigend, A.., Rumelhart, D.. & Huberman, B.., 1990. Predicting the future: A connectionist approach. *International Journal of Neural Systems*, 1(3), pp.193–209.

Werbos, P.J., 1989. Backpropagation through time. In *Proceedings of the IEEE conference*. Washington DC, pp. 1549–1550.

Wijayasekara, D. et al., 2011. Optimal Artificial Neural Network architecture selection for performance prediction of compact heat exchanger with the EBaLM-OTR technique. *Nuclear Engineering and Design*, 241(7), pp.2549–2557. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0029549311003530 [Accessed July 18, 2014].

Wilson, D.R. & Martinez, T.R., 2001. The need for small learning rates on large problems. *IJCNN'01. International Joint Conference on Neural Networks.* In *Proceedings (Cat. No.01CH37222)*, 1, pp.115–119. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=939002.

Xie, X., Zhang, W. & Yang, Z., 2002. A Dissipative Particle Swarm Optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*. pp. 1456–1461.

Yates, W.., Patridge, G. & PAOLA, G., 1996. Replicability of Neural computing experiments. *Complex Systems*,, 10, pp.257–258.

Zhang, G. et al., 1999. Artificial Neural Networks in bankruptcy prediction : General framework and cross-validation analysis. *Europena journal of operational research*, 116, pp.16–32.

Zhang, G. & Kline, D., 2007. Quarterly Time-Series Forecasting With Neural Networks. *IEEE Transactions on Neural Networks*, 18(6), pp.1800–1814. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4359174.

Zhang, G., Patuwo, B.E. & Hu, M.Y., 1998. Forecasting with Artificial Neural Networks : The state of the art. *International Journal of forecasting*, 14(July), pp.35–62.

Zhang, G.P., 2007. A Neural Network ensemble method with jittered training data for time series forecasting. *Information Sciences*, 177(23), pp.5329–5346. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0020025507003003 [Accessed July 18, 2014].

Zhang, G.P., 2001. An investigation of Neural Networks for linear time-series forecasting. *Computers & Operations Research*, 28(12), pp.1183–1202. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0305054800000332.

Zhang, T. & Fukushige, A., 2002. Forecasting time series by Bayesian Neural Networks. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*, 2(November), pp.382–387. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1005502.

Zhani, M.F. & Elbiaze, H., 2009. Analysis and Prediction of Real Network Traffic. *Journal of Networks*, 4(9), pp.855–865.

Zhou, Z., Member, S. & Liu, X., 2006. Training Cost-Sensitive Neural Networks with Methods Addressing the Class Imbalance Problem. In *IEEE Transactions on knowledge and data engineering.* pp. 1–14.

Zhuang, L. & Hindi, K., 1990. Mean Value analysis for multiclass closed queueing network models of flexible manufacturing systems with lilited buffers. *Europena Journal of Operational Research*, 4(6), pp.366–379.

Zurada, J., 1994. Computational Mechanism Design. *Computational intelligence imitating life*. New York: IEEE Press, pp. 62–82.

Appendix A. Multilayer perceptron program

We would like to acknowledge Professor Mike Burton from the Rhodes University Department of Applied Mathematics for his invaluable assistance during the formulation of these codes.

The codes available in this Appendix are for:

- Importing the time series into Matlab workspace
- Linear interpolate, detrend the time series and compute the Annotations of the time series
- Train script which does the following
    - Loads the data in the Matlab Neural Network toolbox from Matlab workspace.
    - Partitions the data into training and test sets.
    - Automatically scales the data down before processing and scales it up afterwards.
    - Constructs an ANN.
    - Trains the ANN and saves the variables in a file named VUSI9.mat
- Test script which does the following
    - Loads the variables saved on the train script
    - Simulates the network activations and targets on train and test data.
    - Tests the performance of the ANN on unseen test data
- Supervising script which keeps a record of ANN performance and inputs and targets and converts ANN inputs and targets from linear to cyclic.
- Correlation function which computes the R and R^2 on the train and test sets.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This is a script which imports and saves the time series data into Matlab %
% workspace.
Function importfile(fileToRead1)
%IMPORTFILE(FILETOREAD1)
% Imports data from the specified file
% FILETOREAD1: file to read
% Import the file
sheetName='Sheet1';
[numbers, strings] = xlsread(fileToRead1, sheetName);
if ~isempty(numbers)
newData1.data = numbers;
end
if ~isempty(strings)
    newData1.textdata = strings;

end
% Create new variables in the base workspace from those fields.
vars = fieldnames(newData1);
for i = 1:length(vars)
assignin('base', vars{i}, newData1.(vars{i}));
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%this is a script which interpolates the missing data, detrends the time
%series, computes the standard deviation, mean and median of the time series.
%Interpolating missing data
data = resample(data,data.Time);
%Removing rows missing data
Irowexcl = isnan(data.Data(:,[1]));
data = delsample(data,'Index',find(Irowexcl));
%Detrending time series
data= detrend(data,'constant',[1]);
%Calculating standard deviation of time series
ts_std=std(data);
ts_std = std(ts,data,Value);
%Calculating mean of the time series
ts_mn=mean(data);
ts_mn = mean(ts,data,Value);
%Calculating median of the time series
ts_med=median(data);
ts_med = median(ts,data,Value);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This script loads and organises the data into Neural network Toolbox,
%constructs the neural network,
%trains it and
%saves the variables to a file: VUSI9.mat
x=xlsread ('Book.xls','d2:731');
myfigure
plot(x)
xlabel('time periods(days)')
ylabel('network traffic(bits/sec)')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%define size of the sliding time window
[p, t]=delay(x,150);
m=size(p,2);
%normalise all data network targets and activations
[pn,ps]=mapminmax(p);
[tn,ts]=mapminmax(t);
%partition the data define index for the test and train sets
%size of test set
s=150;
%ttest
tti=[m-s:m];
%ptest index
pti=[m-s:m];

%test set: last s
ptestn=pn(:,pti);
ttestn=tn(:,tti);
ptest=p(:,pti);
ttest=t(:,tti);

%training index
ttri=[1:m-s-1];
ptri=[1:m-s-1];
```

```
%train set patterns and targets
ptrainn=pn(:,ptri);
ttrainn=tn(:,ttri);
%define network parameters
s1=60;
s2=1;
net=newff(minmax(pn),[s1,s2],{'logsig','purelin'},'trainscg');
%net=feedforwardnet([s1],'trainscg');
net.divideFcn='';
%Net training
net.TrainParam.epochs=1000;
% net.performfcn='msereg';
% net.performparam.ration=.7;
%net.trainparam.max_fail=5000;
%initiate the weights and biases
net=init(net);
%train the net
%myfigure
net=train(net,ptrainn,ttrainn);
%save the variables in a file named gully9.mat
VUSI9net=net;
saveVUSI9.mat
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This is a script which tests the generalization performance of the trained
%network on new unseen data.
%The script loads the variables stored in VUSI9.mat
%simulates the network on training and tests data
%assesses the performance of the network on unseen test data
Closeall;clear all;clc
loadVUSI9.mat
%simulate the activations
atrainn=sim(VUSI9net,ptrainn);
%postprocess activation on training set
atrain=mapminmax('reverse',atrainn,ts);
ttrain=mapminmax('reverse',ttrainn,ts);
%Print the values of R and R2
[r2 r]=correlation(atrain,ttrain);
fprintf('Train: r %5.4f r2 %5.4f\n',r,r2)
atestn=sim(VUSI9net,ptestn);
atest=mapminmax('reverse',atestn,ts);
[r2 r]=correlation(atest,ttest);
%plot a figure of the network activations and targets.
fprintf('Test: r %5.4f r2 %5.4f\n',r,r2)
myfigure
%myfigureposition(pos1)
plot([1:length(atest)],atest,[1:length(ttest)],ttest,'o')
set(gca,'xticklabel',[m+2-s:m+2])
title('testing')
xlabel(sprintf('targets %d to %d\n',m+2-s,m+2))
an=sim(VUSI9net,pn);
a=mapminmax('reverse',an,ts);
myfigure
%myfigureposition(pos2)
plot([1:length(a)],a,[1:length(a)],t,'o')
title('all')
%calculate the error measures
sse_train=sum((ttrain-atrain).^2)%sum of squared errors train set
mse_train=sse_train/length(ttrain)%mean square error train set
rmse_train=(sqrt(mse_train));%% root mean square error on the train test
```

```
sse_test=sum((ttest-atest).^2)%sum of squared errors test set
mse_test=sse_test/length(ttest)%mean square error test set
rmse_test=(sqrt(mse_test));%root mean square error on the test set
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This supervising script defines the function delay to be used to
%compute the pattern of inputs and output from the sliding time window.
function [p t]=delay(x,r)
%produces a moving window of length r from the sequence x
%from x1 x2 x3 .......
%produces

% p=
% x1   x2                x(n-r)
% x2
%
% xr   x(r+1)             x(n-1)

% t=
% x(r+1)                  xn
%There are n-r input patterns p and targets t
n=size(x);

for k=1:n-r
p(:,k)=x([k:k+r-1]);
t(k)=x(k+r);
end
```

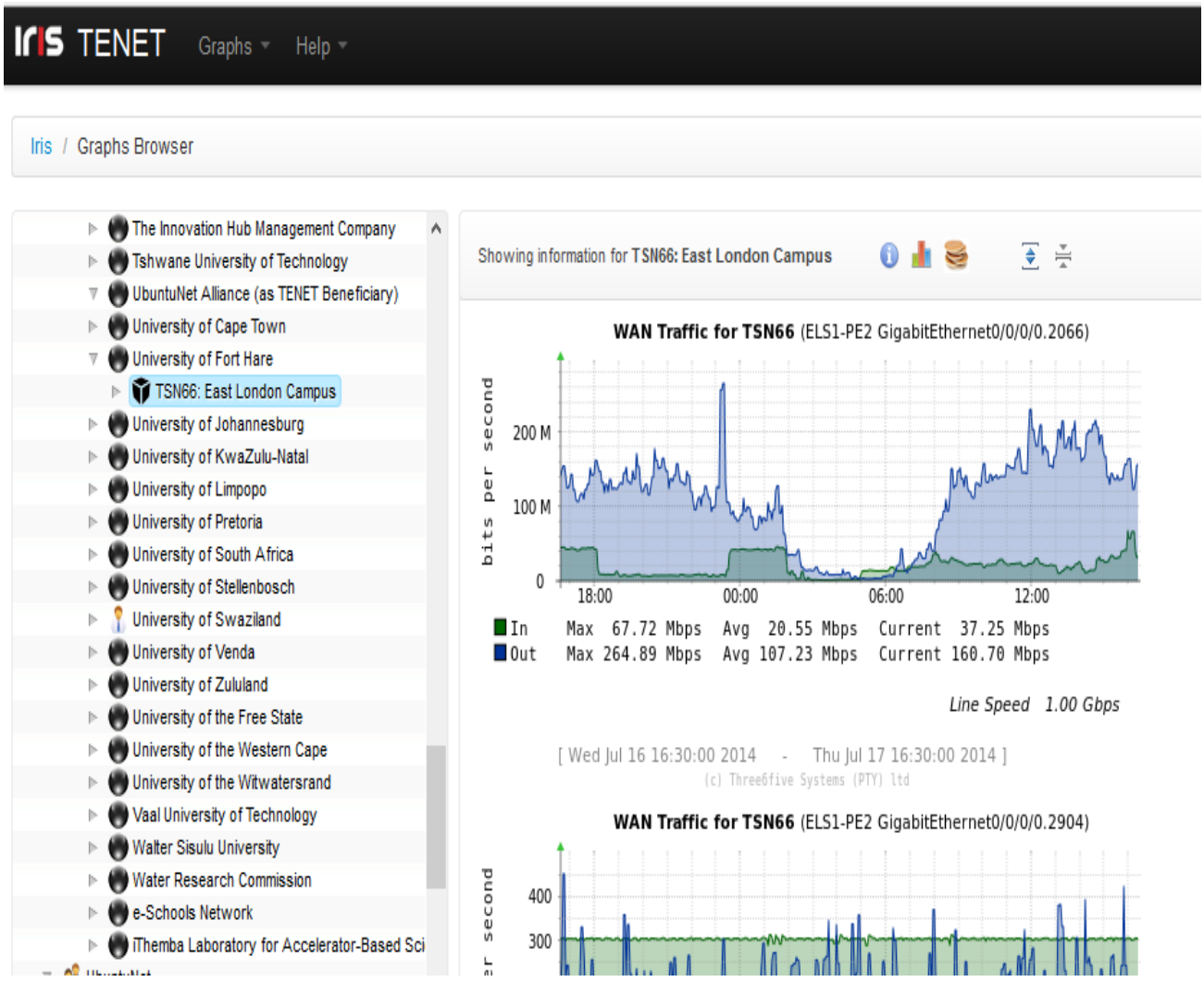
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This is a function which computes the R^2 statistic and correlation
%coefficient R for
%a: activation vector
%t: target vector
%calculating R2:
function [r2 r]=correlation(a, t)
tbar=mean(t,2);
abar=mean(a,2);
tBar=repmat(tbar,1,length(t));
ss=sum((a-t).^2,2)/length(t);
sstBar=sum((tBar-t).^2,2)/length(t);
r2=1-ss./sstBar;
%calculating correlation coefficient R
et=t-repmat(tbar,1,length(t));
ea=t-repmat(abar,1,length(a));

if nargout>1
for i=1:size(et,1)
       norms(i)=(norm(et(i,:)))*norm(ea(i,:));
end
norms=norms(:);
   r=sum(et.*ea,2)./norms;
end

%for i=1:size(t,1)
% figure
% hold on
% plot(t(i,:),0,'o')
% plot(t(i,:),t(i,:))
% plot(t(i,:),a(i,:),'*')
% hold off
% title(sprintf('%s component %d\n','name',i))
% xlabel('targets')
% ylabel('activation')
%end
```

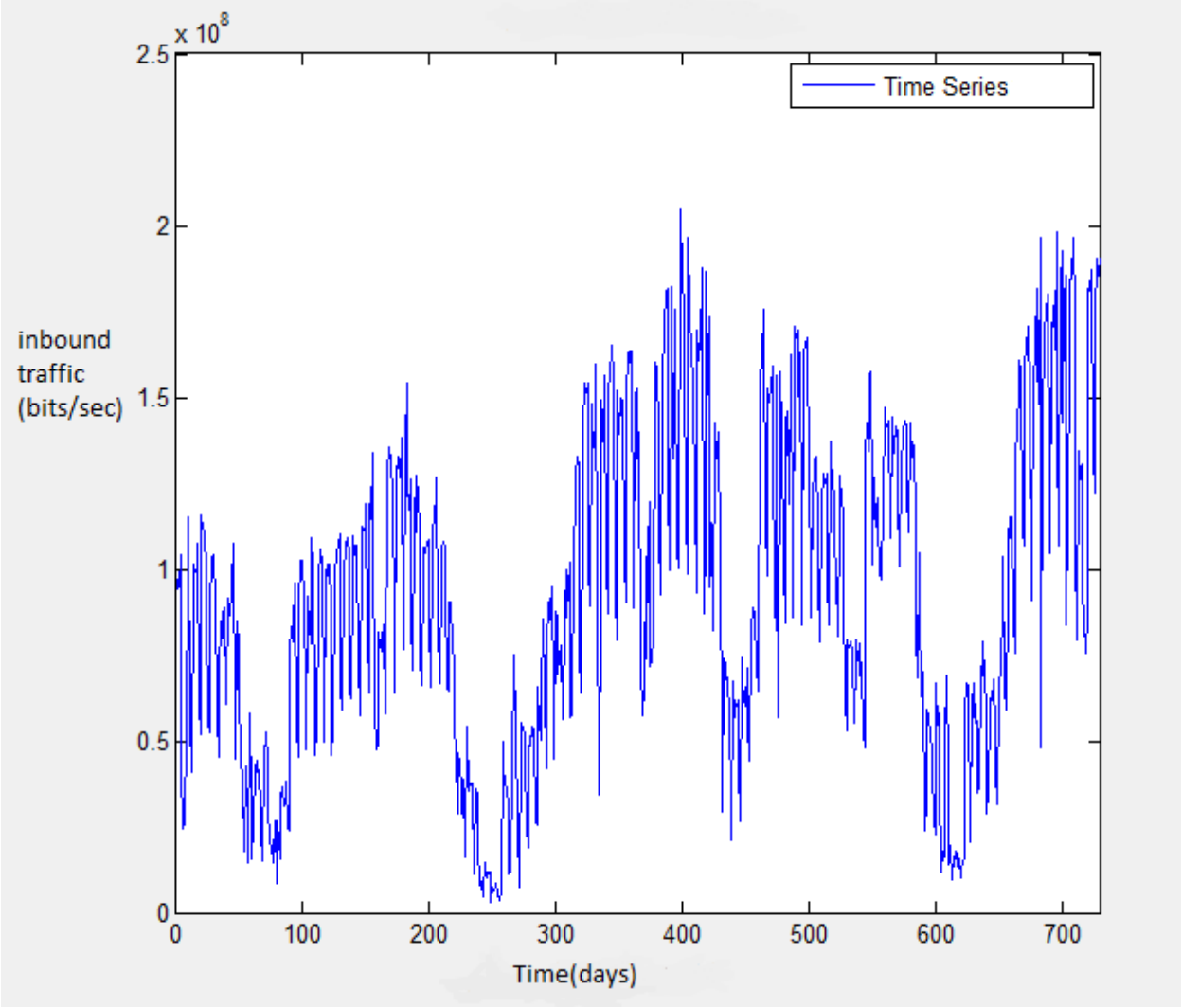
Appendix B. Multi Router Grapher software

Snapshot of the Multi Router Traffic Grapher (MRTG) software which monitors traffic load on network links from various institutions in South Africa.



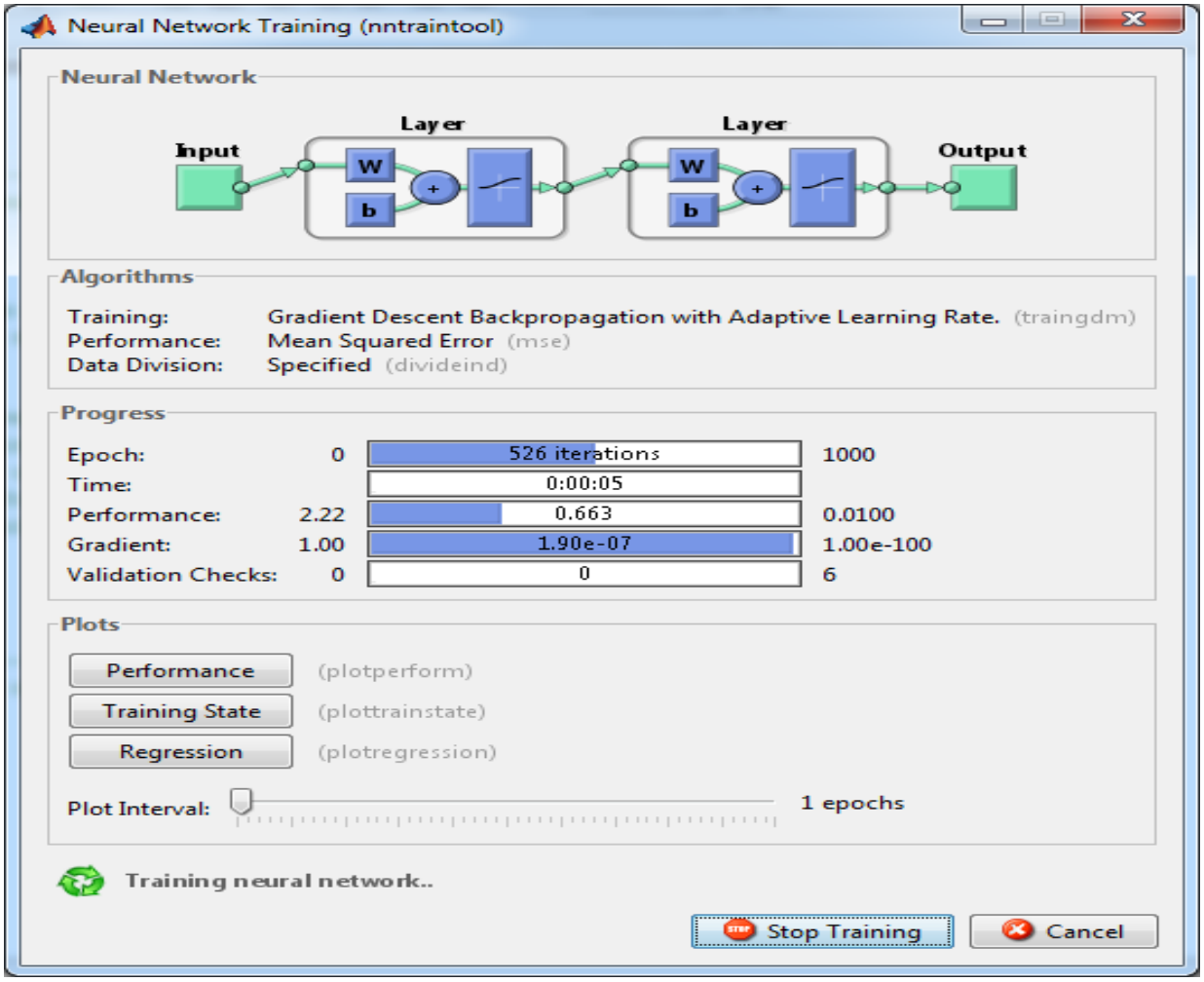
Appendix C. Snapshot of time series

Snapshot of the time series as imported into Matlab.



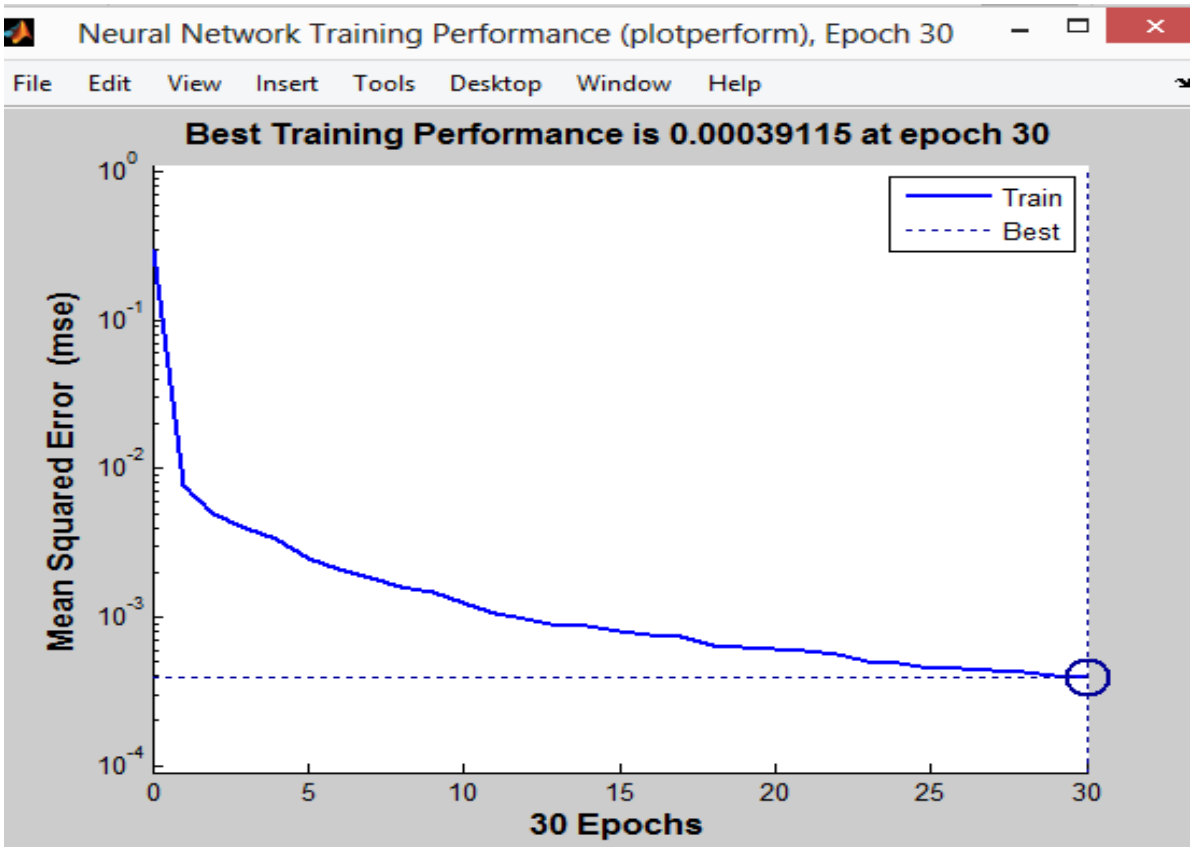
Appendix D. Snapshot of Artificial Neural Network during training

A snapshot of the Neural Network during one of the many training sessions in the Neural Network toolbox Version 5.0.2 (R2007a).



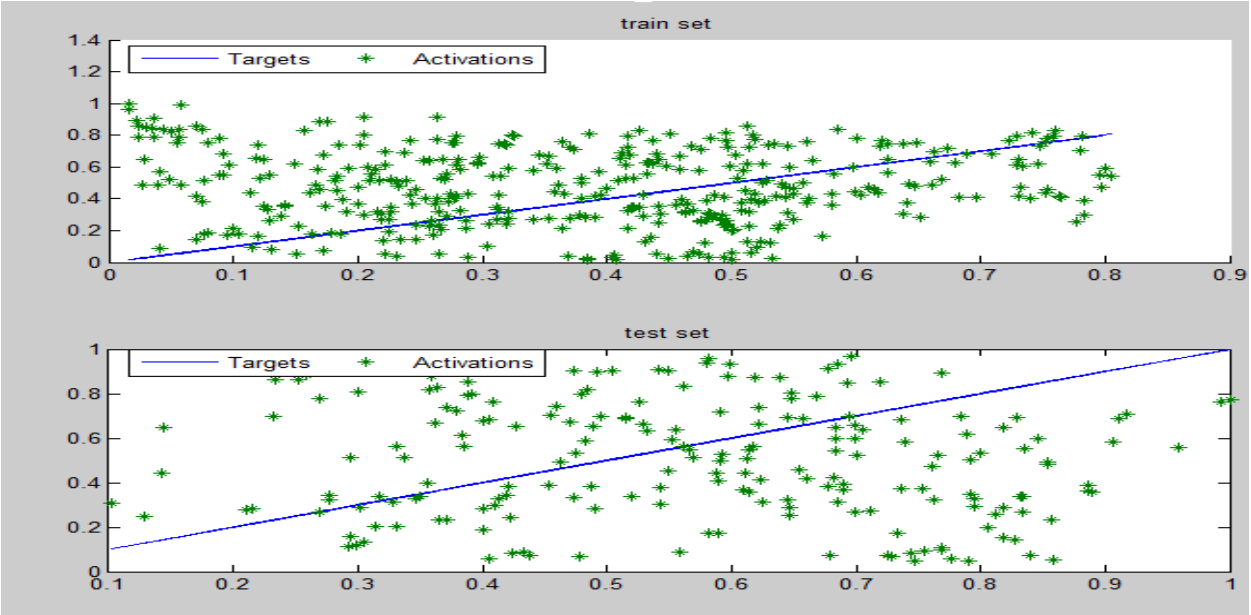
Appendix E. Snapshot of performance plot

A snapshot of a Performance plot during training of one of the ANNs.

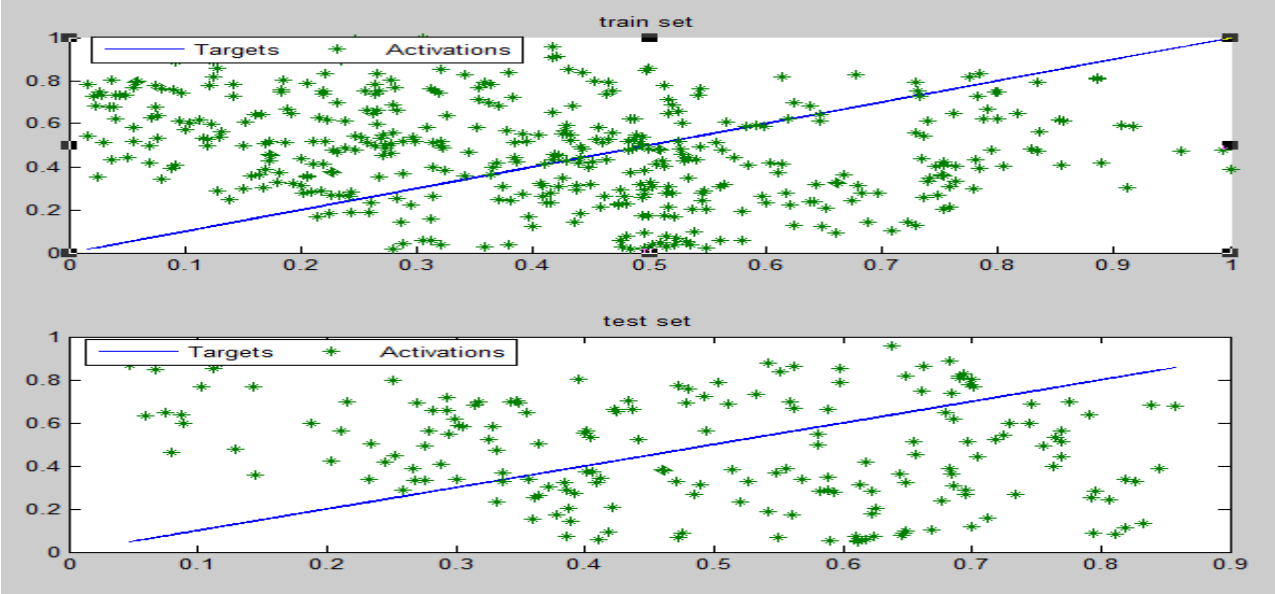


Appendix F. Results of the different sliding time windows

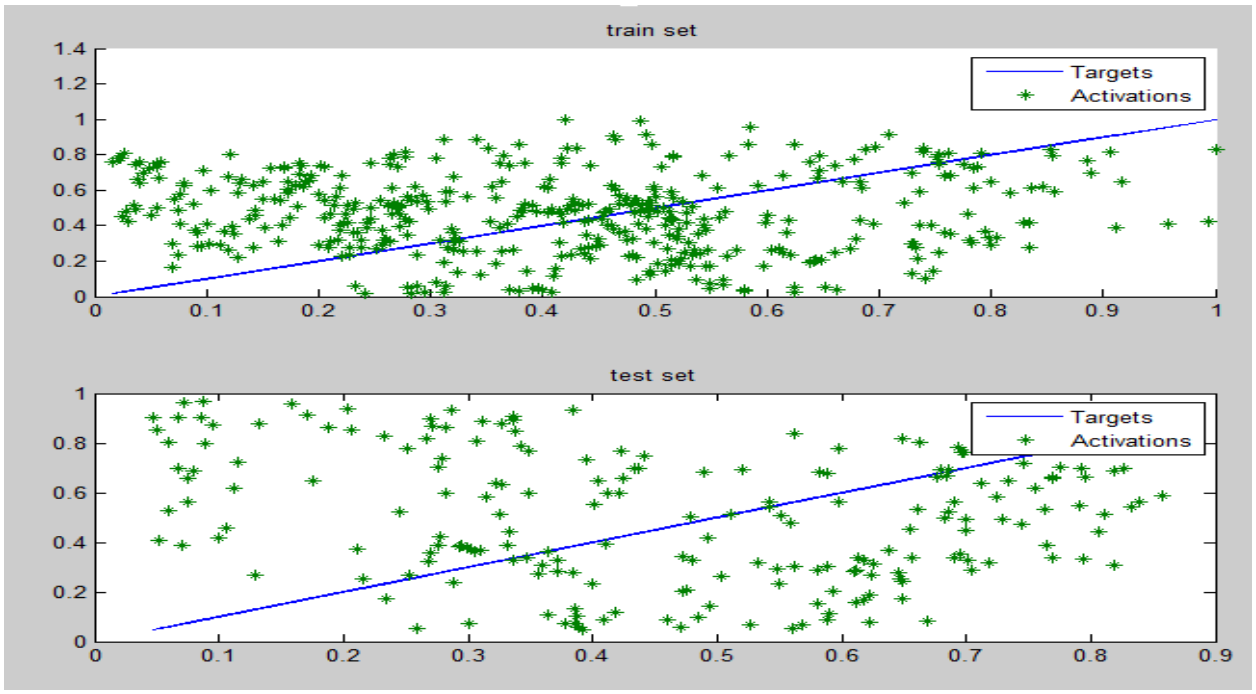
Results of sliding time window size 20



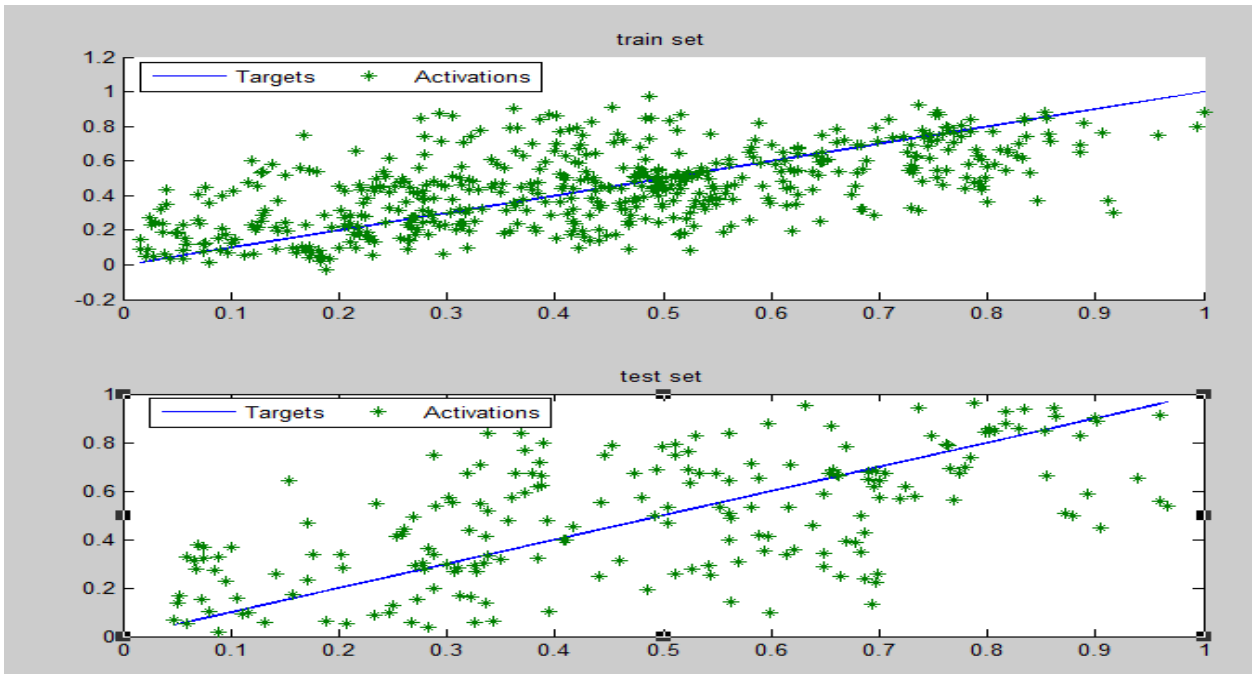
Results of sliding time window size 40



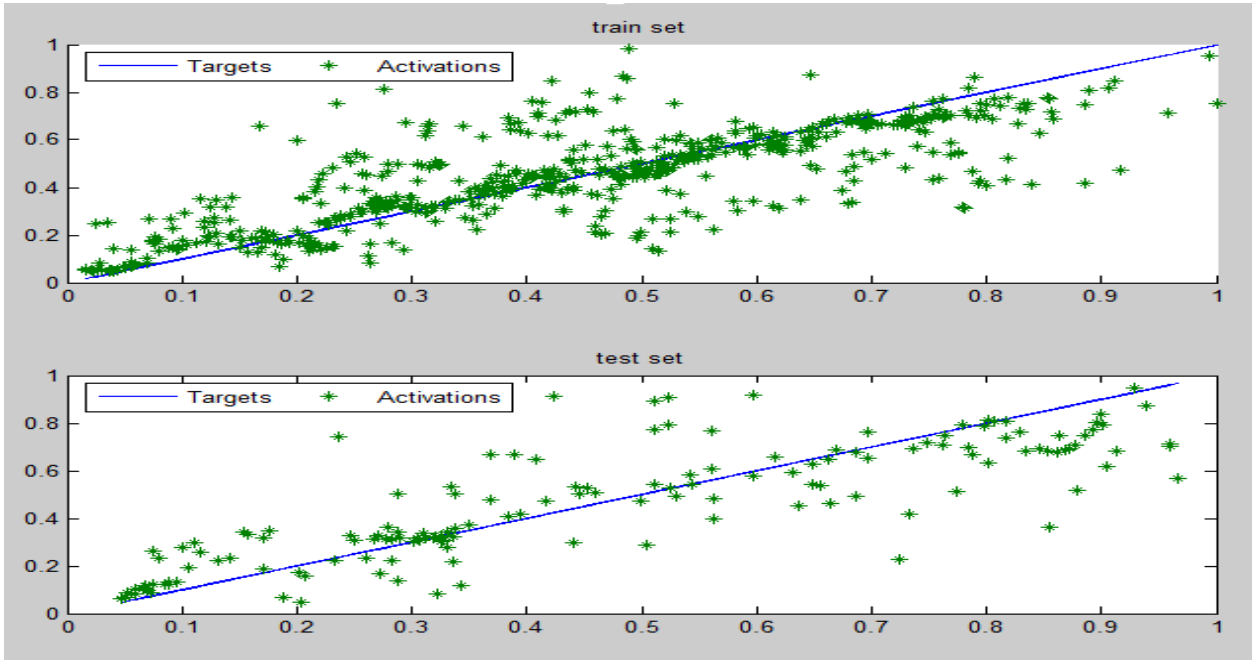
Results of sliding time window size 60



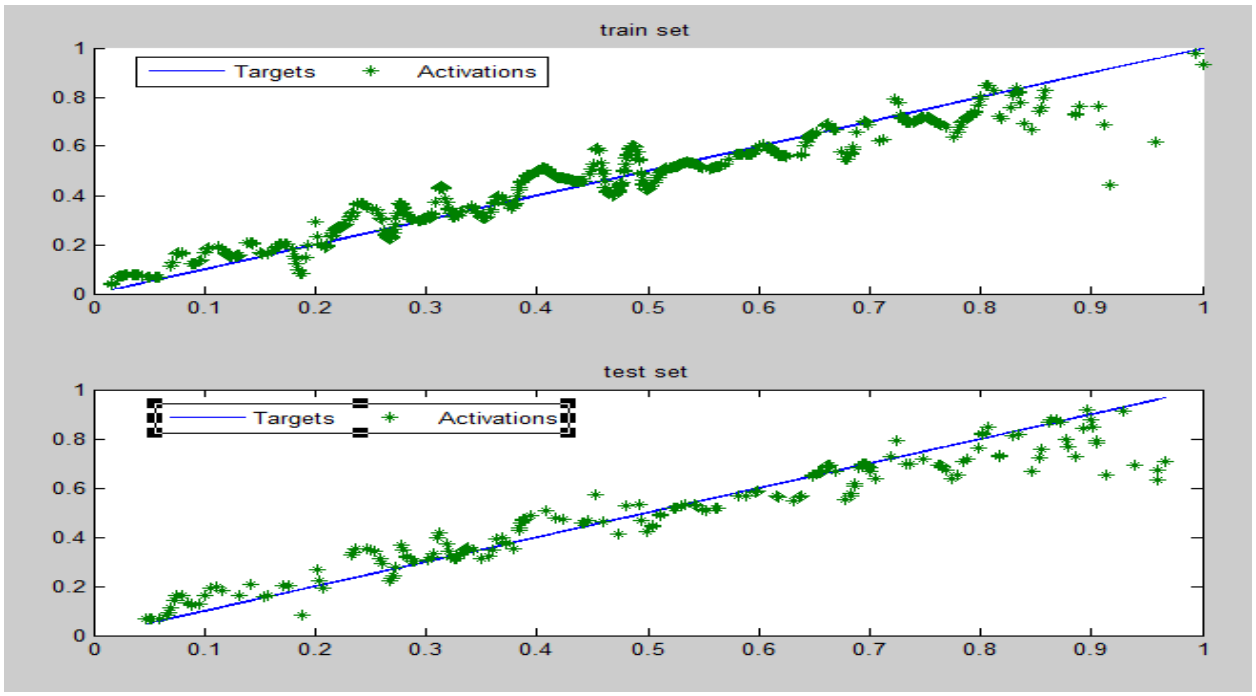
Results of sliding time window size 80



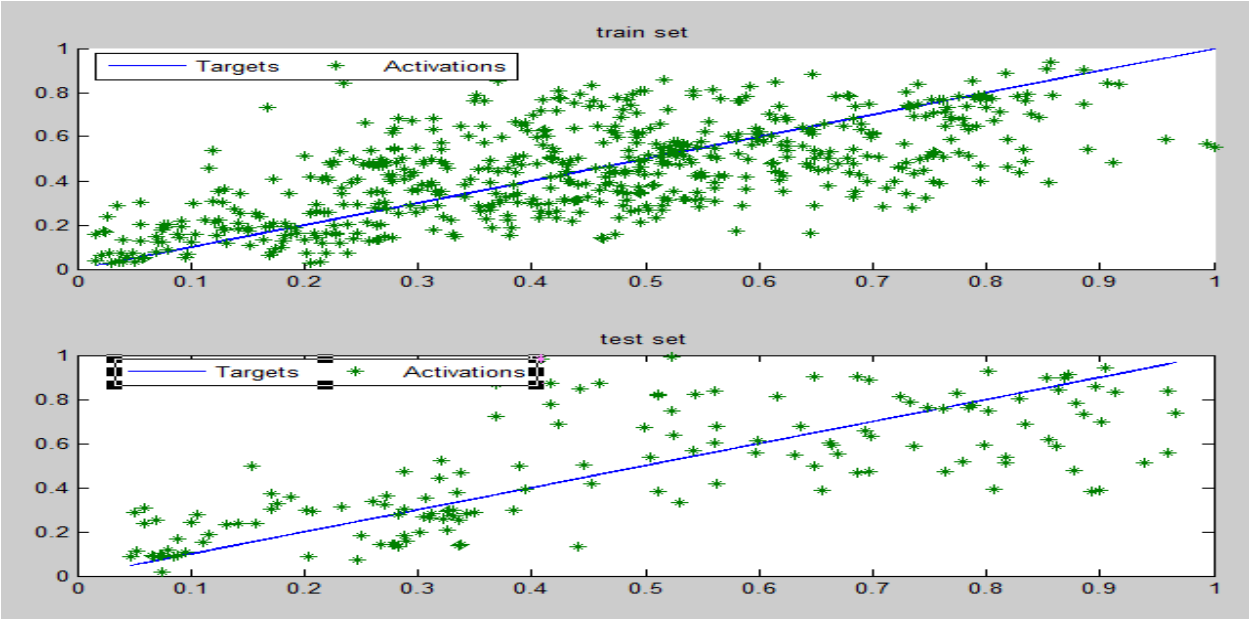
Results of sliding time window size 100



Results of sliding time window size 150



Results of sliding time window size 170



Results of sliding time window size 200