OPTIMAL SELECTION OF STOCKS USING COMPUTATIONAL INTELLIGENCE METHODS

Brain Leke Betechuoh

A Dissertation submitted to the Faculty of Engineering and the Built Environment, University of the Witwatersrand, in fulfilment of the requirements of the degree of Master of Science.

Johannesburg 2004

To Mom and Dad

&

To Maphefo

PREFACE

The work described in this dissertation was carried out in the University of Witwatersrand, School of Electrical and Information Engineering in 2004. I would like to acknowledge a couple of people who made this thesis a possibility by an extensive support. Firstly, Prof Tshilidzi Marwala, who was my supervisor for this project and who put so much insight into the development of the methodologies. I also thank him for all the support, be it educational, financial or social. Thank you. I would also like to thank my fellow masters students Mr Gerasimos Tselentis, Mr Opeyemi Oni, Mr. Lukasz Machowski and Mr Micheal-Phillips Powell for their companionship and who always gave me some insights and details into my project which I may have neglected if not of their intervention. I also want to thank a couple of people who even though did not intervene technically in this project made it a possibility through their moral and social support. Firstly, my parents, Mr. Leke Casimir and Mrs. Leke Agatha, who have supported me all through my educational years emotionally and financially. My family, (Gwendoline, Clarence, Sydonie and Collins) for their support. Ms. Maphefo Matjeke, for always being there with her spiritual guidance and emotional support. To Mr. Patrick Katabua, Mr. Mphake Manyatshe, Ms. Loveness Msuku and Mr Tebatso Gillian for the friendship and helping me focus. Except where reference is made to the work of others, I affirm that this thesis is a result of my own original work. No part of this work has already been, or is currently being, submitted for any other degree, diploma or other qualification. This thesis is 136 pages in length with approximately 20814 words.

B. Leke Betechuoh

December 2004

Abstract

Various methods, mostly statistical in nature have been introduced for stock market modelling and prediction. These methods are, however, complex and difficult to manipulate. Computational intelligence facilitates this approach of predicting stocks due to its ability to accurately and intuitively learn complex patterns and characterise these patterns as simple equations. In this research, a methodology that uses neural networks and Bayesian framework to model stocks is developed. The NASDAQ all-share index was used as test data. A methodology to optimise the input time-window for stock prediction using neural networks was also devised. Polynomial approximation and reformulated Bayesian frameworks methodologies were investigated and implemented. A neural network based algorithm was then designed. The performance of this final algorithm was measured based on accuracy. The effect of simultaneous use of diverse neural network engines is also investigated. The test result and accuracy measurements are presented in the final part of this thesis.

Key words: Neural Networks, Bayesian framework and Markov Chain Monte Carlo

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction: The Stock Market

1.1 Background

The stock market appears in the news everyday [1], [2]. Every time it reaches a new high or a new low there is talk about it. But what is the stock market? The stock market is believed to have started at Wall Street. This is where the world's largest financial market was born and prospered. From Wall Street sprang a new industry with its own language and terminology. Wall Street can trace its name back to 1653. Originally it was set up for defense and not for commerce. What helped Wall Street rise to pre-eminence was the emergence of two great Stock Exchanges, which gave order to the chaotic trading and gave birth to the financial markets as it is known today. In 1790 at Philadelphia in the United States of America, the first stock exchange was founded. Two years later a group of New York merchants met to discuss how to take command of the securities business. The merchants founded what is now known as the New York Stock Exchange. But in early 1817, the merchant group from New York, distressed at the sorry state of their stock exchange, sent a representative to Philadelphia to observe how things were being done. Upon arriving with news about the robust exchange in Philadelphia, the New York Stock

and Exchange Board was soon formally organized. In the early 1900s massive amounts of money were made and lost on Wall Street. But the boom period could not be sustained indefinitely. And in 1929, the stock market seared the global-psyche and triggered what was to be called the Great Depression. But the stock market crash of 1929 was just the beginning of sorrows for Wall Street. For a while the economy eventually recovered from its catastrophic losses, the market excesses that had factored into the crash in the late 1920s seeped back into the picture. The result was the stock market crash of 1987, which saw the Dow Jones suffer what was the largest single-day loss in the stock market's history. The stock markets are now an integral part of the global economy, and so proper safeguards to reduce the risks of another disastrous crash are necessary. A market can be defined as a place which introduces a buyer to a seller. In the case of stocks the buyer and seller are dealing in small ownership portions of companies or shares. A stock symbolizes ownership in a company. The more stock investors hold in a particular company, the larger the percentage of the company they own. For instance, if a corporation has 20000 shares of stock outstanding and a person owns 1000 of them, then he or she actually owns 5 percent of the corporation. Those who own stock become shareholders or stockholders in the company from which they purchased the stock, and they remain shareholders for as long as they own the stock. In this way, the stocks investors own, reflects the percentage of the company they own. Stock markets perform the following functions:

- Connecting those who seek money with those who can provide it.
- Create an auction mechanism in which prices can be decided for investments.
- Distributing the future risk of investments across many millions of individuals.
- Providing the claim tickets upon which future wealth can be staked.

- Connecting financial institutions together to create money.

The stock market is an important entity in a country because it indicates the state of the economy. This state of the economy gives an indication of its stability, thereof, which can in turn be linked to the stability of the nation. This information can be used as a comparison of the nation's economy to other well established economies. The stock market has also become the very symbol of commerce in the modern world. They are truly unique in their scope and in the complexity of the number of transactions they handle each day. The economy of the world relies on the stock exchanges to facilitate even trade in the stocks of companies. On an individual level, the stock market is a high risk but high profit yielding investment. Due to the high risks involved in such an investment, it is beneficial that some sort of analytical tool, which reliably predicts future prices of stocks, be developed. The investments in the stock market are done by the trading of stock shares based on intelligent decisions. These intelligent decisions are generally made by stock brokers who based on analytic and statistical calculations decide on whether a stock is viable for investment or not. In our current era anyone can easily acquire for themselves the most popular stocks just by opening an online brokerage account. Direct interaction with the selling floor of the exchanges gives the modern investor more control than any other generation. There are a number of options available for investors who want to learn the complexities of the stock market. One popular way is to take a course on the stock market. What makes these lessons useful is that they usually enable participants to take a proactive approach to the trading process without having to assume any financial risk. Lessons dealing with the stock market, for instance, may require participants to attend a class setting where they get into groups. These groups will

then represent companies with CEOs, employees, human resources etc. These companies then make various decisions and, according to the stipulations of the game, are asked to react to various variables in the marketplace that emerge from time to time. Depending on how these companies respond to the variables, their stock prices could go up or down. What these stock market lessons do is to allow participants to gain an insider's view of the inner workings of a company and how these inner workings ultimately impact the public's perception of the company's value. Participants are often asked to respond to geo-political events, the need for layoffs, fiscal pressure, economic shifts and other factors real-life companies have to deal with on a day-to-day basis. Another twist on the educational front is to enable individual investors to learn about the stock market from the perspective of someone who wants to purchase stocks in a company. This risk-free option will enable participants to learn about the market without having to lose any real money in the process. In such a program, participants invest in the stock market and regularly research the companies they have invested in. Participants also learn how to determine the best time to buy and sell their stocks. The stock market has an interesting property in that since all of the buying and selling is done at one place the prices of the stock can be known every second of the day. When it comes to investing in the stock market, investors should know when to hold on to stocks and know when to unload them. Most financial experts believe that the buy-and-hold strategy, which requires investors to buy stocks and then keep them for the long term, is the best method for ultimately making money on the stock markets. The rationale behind this strategy is that, while the markets will likely experience ups and downs stemming from numerous factors, over time the stock markets tend to push upwards. This means that those who use the buy-and-

hold strategy stand to make money over time. While there are many experts who still hold to this strategy, others point to some of the more catastrophic stock market crashes of the past as proof that investors can literally lose everything they had gained in a bull market (a bull market refers to the stock market when stock prices have gone up for a certain period of time) to the impact of the bear market (a bear market occurs when the stock market drops for a given period of time, caused often by lower than expected quarterly earning reports, economic pressures or some other reason that gets market participants jittery). This may be due to result of inflation and political instability. Rather than adopt a buy-and-hold strategy, some financial professionals recommend that investors take a more sophisticated approach to buying and selling stocks. This necessitates monitoring market conditions and making changes as fluctuations in the markets warrant change. What it does not mean is making change just for the sake of making change. Some investors choose to go with a broker so as to bypass the pressures of managing their own stock portfolios. Doing so requires them to look around for a good broker, one who has a proven methodology and a solid track record. There are a number of statistical analysis applications available these days to meet the needs of individual investors or large corporations. Key components of any statistical analysis software include the ability to:

- perform data management
- present data in graphs and reports
- access data at a moment's notice

A number of websites also offer plenty when it comes to stock market analysis. For investors willing to pay, some companies provide a mix of services related to stock

market analysis. For instance, some interpret the financial news of the day, highlighting the implications of various developments and explaining how these developments could impact the marketplace, in general, and investors, in particular. Other service, which providers may offer relates to providing:

- details on which stocks could be hot or cold on any particular day
- analysis of earnings reports and what they mean
- updates on important events when they happen
- Signing up for free newsletters is another way to keep on top of developments in the stock market as they happen.

Due to the high necessity of reliable software to do the analytical manipulation of stocks, many different software systems have been developed of which this document focuses on the development of such a software and also the design of the relevant methodology. The next section deals with the literature survey of the previous methodologies developed for analyzing stock portfolios.

1.2 Research Focus and Motivation

The aim of this project is to develop a computational intelligence procedure to predict the future prices of stocks. The research also focuses on optimising the input time-window required for the prediction of stocks and this was motivated by the fact that upon analysis of the literature review, the optimisation of the time-window, which forms an important part of the prediction process, had not been done before. In this thesis, the computational intelligent method used is neural networks. The proposed procedure is to be tested on the NASDAQ index [3]. This project will consist of the following tasks:

1. Create an intelligent engine using computational intelligence methods. The aim of this engine will be to predict the future stock prices from historical data.
2. Train the network method using a Bayesian framework [4].
3. Identify the optimal input time-window using polynomial approximation and by redesigning the neural method to account for optimal selection of time-window and compare the two approaches.
4. Investigate the effect of the simultaneous use of diverse computational intelligence engines on the accuracy of the prediction.

The output of the design is thus a methodology that can be used to optimally select the input time-window as well as predict the future stock prices.

1.3 Literature Review

Most of the conventional sales forecasting methods use time series data to determine forecast. Lachtermacher and Fuller [5] conducted a survey which indicated that artificial neural networks (ANN) are more appropriate for time-series data rather than conventional regression methods. They developed a calibrated ANN model using the Box-Jenkins methods to identify the input variables and also developed a methodology to suggest the number of hidden units needed by the model. However, they did not suggest a methodology to accurately choose an optimal time-window.

Bigus [6] used promotion, time of the year, end-of-month and weekly sales as inputs to the ANN to forecast weekly demand. The results show a high degree of accuracy, however, in his paper Bigus does a weekly forecast by using a number of inputs with no

mention of how this number of inputs was determined. Agrawal and Schorling [7] showed that ANN is able to predict future share prices quite well from time-series data without the additional inputs that were used by the Bigus model [6].

Wang [8] proposed a methodology for the prediction of stock prices using a fuzzy grey prediction system. In his paper, he uses a fuzzy grey prediction system with two modules which are: the prediction agent and the graphic display agent. This method proved to be unsuitable for predicting the behavior of the system due to the fact that an inaccurate forecasting step was used. This method has the limitation in that it does not present a methodology of selecting the optimal time-window needed for the prediction of the future stock prices but states that 5 days can be used to predict future 2 days.

Kuo, Wu, Wang [9] proposed a methodology that uses artificial neural networks and fuzzy neural networks with fuzzy weight elimination for prediction of share prices. Previously, statistical methods which include the regression methods and moving average methods were used for such prediction. These methods have the limitation in that they are efficient only for data which are seasonal or cyclical. The results proved to be more accurate than the conventional statistical methods. In their paper they use historical time series data. Again, just as in Lachtermacher and Fuller [5], this methodology had the shortfall in that there was no mention of a methodology to select the optimal time-window even though the methodology gave adequately accurate results.

Chapter 2 of this thesis focuses on introducing neural networks as well as providing a rationale behind the increase in awareness of neural networks. This chapter also introduces the fundamentals of neural networks. It begins by giving definitions for neural networks. The different kinds of neural network architectures are looked into. It also gives a layout on the advantages and benefits neural networks give as well as the different applications in which neural networks are being used in. A brief discussion is given on the present and future of neural networks as well as their limitations.

Chapter 3 of this document focuses on statistical analysis methods that have been applied to the stock market.

Chapter 4 of this document then focuses on the Bayesian framework optimisation method. This part of the document introduces the Markov Chain Monte Carlo (MCMC) methods as well as the Metropolis-Hastings Algorithm, which is used to sample the posterior distribution resulting from the implementation of Bayesian framework.

The methodologies developed are then discussed in the later chapters. Two approaches are proposed to select the optimal time window. The first method is to use polynomial approximation and the second one to reformulate the neural network architecture such that the optimal time window is an inherent variable to be learned during the training stages. The first approach entails the generation of a polynomial mapping the error function to the number of inputs and minimizing this error function to get the optimal input time-window required to give the best prediction. The second approach involves

using the Markov Chain, Monte Carlo and Markov Chain Monte Carlo methods to optimally select the appropriate time window while in the training stages of the neural network. This will thus involve the reformulation of the Bayesian networks to suit the optimal selection of the input time-window.

The focus of the second part of this document is on the design process: from the analysis of the problem specification, to the choice of appropriate architectures, and finally to the actual neural network design.

Some chapters end with remarks and conclusions, which give the relevance of the section to the project discussed. These remarks give the relevance of such chapters to the project.

Chapter 2

Neural Networks

2.1 Introduction

The recent rapid advances in neural network technology in many pattern recognition systems, as opposed to the conventional statistical theory, have been attributed to the ability of these neural networks to model any kind of a system, be it a linear or non-linear. Due to the difficulty and complexity of all the various statistical methods employed and the high level of expertise required for such methods such as; moving averages and regression methods, there has been a significant increase in usage of neural networks. This increase has also been due to the fact that neural networks can be applied to virtually every field in the industry, such as the medical field e.g. AIDS modelling, engineering e.g. control of the product quality. Neural network has gathered enormous momentum in recent years and this field of study is currently being introduced in many universities with the industry demanding more products which need neural networks.

This document constitutes a neural network design for:

- Modelling stocks and
- Optimally selecting input time-window for stock market prediction.

2.2 Neural Networks

2.2.1 What Are Neural Networks?

Neural networks (NN) were first introduced in the early 40s based on the understanding of neurology. An artificial neural network is a network consisting of neurons and paths connecting the neurons. They are interconnected assemblies of simple processing nodes whose functionality is loosely based on the animal neuron. NN can also be defined as generalizations of classical pattern-oriented techniques in statistics and engineering areas of signal processing, system identification and control. Figure 1 shows a neural network model with the major components of the network. Each input is multiplied by weights along its path and the weighted inputs are then summed and biased. This weighted input is then biased by adding a value unto the weighted input. The output of the summation is sent into a function which the user specifies (linear, logistic). The output of the function block is fed to the output neuron.

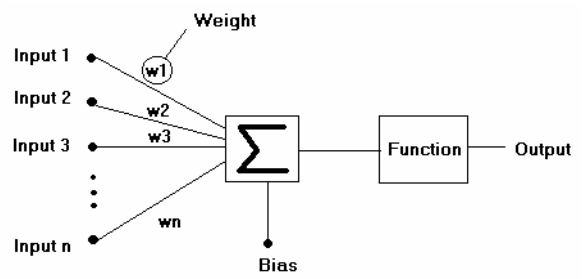


Figure 1: Architecture of a neuron.

Neural networks (NN) consist of simple processing units which communicate with each other by sending signals over a large number of weighted connections. The various

aspects of the NN models are; neurons (a set of processing units); a state of activation for every unit, equivalent to the output of the unit; connection between the units (each connection is defined by a weight which determines the signal the unit j has on unit k); a propagation rule (this determines the effective input of a unit from its external inputs); an external input or bias for each unit; and a learning rule. NN are adaptable systems that can learn relationships through repeated presentation of data, and are capable of generalizing to new, previously unseen data. For Figure 1, the NN output equation is:

$$Output_k = \sum_j w_{jk} y_j + b_k \tag{2.1}$$

Where w_j represents the *j-th* layer's weights, b represents the bias at the node, y_j represents the output at the *j-th* layer's node and k represents the output node.

2.2.2 Why Neural Networks?

Neural network has been motivated by the fact that [10, 11, 12] scientists are challenged to use machines more effectively for tasks currently solved by humans. Neural networks assist in systems where an algorithmic solution cannot be formulated. NN possess the property of adaptive learning which is the ability to learn how to do tasks based on the data given for training or initial experience [10]. NN can create their own organization or representation of the information it receives during learning time from the data observed. NN also possess the ability to represent any function and are known as universal approximators. NN are insensible to noise or unreliable data. There is also no restriction on the output type in neural networks. Neural networks are performed in very short computational times.

2.2.3 Neural Networks versus Other Methods and Linear Statistics

Statistical techniques on handling data have many drawbacks which neural networks do not possess [12]. They impose restrictions on the number of input data which NN do not. The regressions are performed using simple dependency functions (linear and logarithmic), which are quite unrealistic. There is no need for intensive mathematical methods to transform data for NN models meanwhile statistical methods require intensive mathematical transformations. NN are non-linear hence are better able to account for complexity of human behaviour. NN also give tolerance to missing or erroneous values.

2.2.4 Applications of Neural Networks [12]

Neural networks (NN) are currently being applied to nearly every field in the industry. NN are used in the banking sector to predict the issuing of loans, and to predict the recovery of bad loans (NN are used to predict the behaviour of new customers before offering them loans). NN are also used in the finance market to predict share prices. This helps for portfolio and asset management. NN are used in industry for the prediction of product or service demand in order to do better production planning. NN are used in administration for analysing and predicting crime, and tax return analysis for fraud detection. They are used in the medical field to analyse the spread of AIDS and future growth of the disease. NN are also used in game playing for games like Chess, Checkers and Backgammon in order to learn new moves which may not have initially been stored in the database.

2.2.5 Future of Neural Networks

NN are already being used in intelligent refrigerators which do stock taking and order those that are in shortage in the refrigerator. It is also predicted that neural networks, integrated with other computational intelligence technologies and other technologies such as genetic engineering will be used for the generation of life-forms whether man, machine, or a hybrid. Neural networks will give humans the capability to explore new dimensions which are currently only available through extensive training and discipline.

2.2.6 Limitations of Neural Networks [13]

The major issue in industry of NN is the integration of neural networks into the modern environment. These results from the fact that NN sometimes become unstable when applied to large scale problems and they also neglect the effect of noise hence would tend not to react appropriately to sharp changes. There is also the problem that neural networks are viewed as black boxes whose rules are unknown. The results obtained from neural networks are thus not explained.

2.3. Neural Networks Architectures

There exist many kinds of network architectures, such as: [14, 15]

- Multi-Layer Perceptron (MLP)
- Radial Basis Function (RBF)
- Recurrent Neural Networks(RNN)
- (Hierarchical) Mixtures of Experts (HME) and
- Self-Organizing Map (SOM).

2.3.1 Multi-layer Perceptron

The simplest network architecture consists of a single layer with directed inputs, weighted connections to the output unit. These are very simple learning algorithms which find the weights for linear and binary activation functions. However, these algorithms can only work for a limited number of functions. The limitations are overcome by adding one or more layers, known as hidden layers which are nonlinear units between the input and the output. The architecture is a feedforward structure whereby each unit receives inputs only from the lower layers units. Gradient methods are used to find the sets of weights that work accurately for the practical cases. Backpropagation is also used to compute derivatives, with respect to each weight in the network, of the error function. The error function generally used in the neural network computation is the squared difference between the actual and desired outputs. The activities for each unit are computed by forward propagation through the network, for the various training cases. Starting with the output units, backward propagation through the network is used to compute the derivatives of the error function with respect to the input received by each unit. The representation of such a network is as follows:

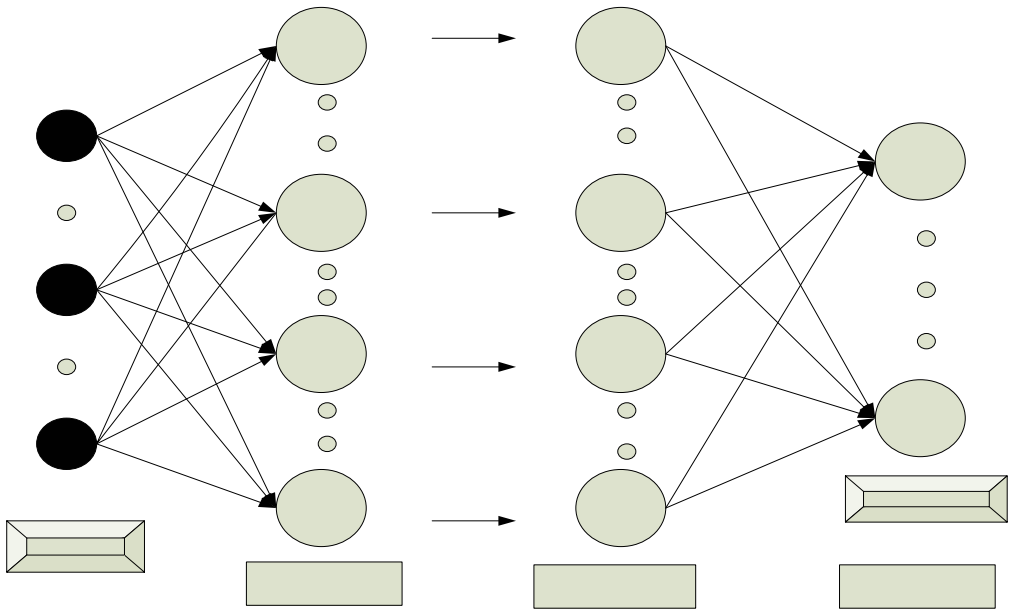


Figure 2: 2-Layer multi-layer perceptron neural network

The learning algorithm and number of iterations determines how good the error on the training set is minimized meanwhile the number of learning samples determines how good the training samples represent the actual function. In multi-layer perceptron, a number of layers are fully connected. The input to the activation function then becomes a scalar product of the layer weight vector w_i and input i, that is:

$$Output = actfn(w_i \times i) \qquad (2.2)$$

The different kinds of activation functions with their equations are as shown Table 1. The perceptron learning rule is a method for finding the weights in a network. The perceptron has the property that if there exist a set of weights that solve the problem, then the perceptron will find these weights. This rule follows a linear regression approach, that is, given a set of inputs and output values, the network finds the best mapping from inputs to outputs. Given an input value which was not in the set, the trained network can predict

the most likely output value. This ability to determine the output for an input the network was not trained with is known as generalization.

Table 1: Table of activation functions with the respective functions

NAME	FUNCTION
Linear	A
Sigmoid	$\frac{1}{\left(1+e^{-a}\right)}$
Tanh	$\frac{\left(e^{a}-e^{-a}\right)}{\left(e^{a}+e^{-a}\right)}$
Exp	e^{a}
Softmax	$\frac{e^{a}}{\sum_{j} e_{j}^{a}}$

Multi-layer networks are known as approximators. Two-layer networks with a sigmoid transfer function in the hidden layer and linear transfer functions in the output layer can approximate any function provided a sufficient number of hidden units are available [16]. These hidden units make use of non-linear activation functions.

2.3.1.1 Linear Regression

Linear regression is the algorithm used to fit a model unto a set of data. If a data set of inputs and outputs is given or can be obtained, it is then possible to fit in a model such that based on this model outputs can be determined for an input which is not in the

original set. The simplest model that can be fit is the linear model which has the following equation:

$$y = w_1 x + w_2 \tag{2.3}$$

This equation describes a straight line with a slope w_1 and an intercept w_2. The major problem in neural networks is choosing the parameters w_1 and w_2 for the given model, which would imply choosing a line which goes through the data. This method uses a supervised learning algorithm since the target values are available.

2.3.1.2 Perceptron Algorithm

The perceptron algorithm is as follows; initialize the weights, pick a learning rate η (this is generally a number between 0 and 1) [15] and iterate until stopping condition is satisfied, modifying the weights. For each training pattern (x, t) the following is done; compute the activation function $y=f(w,x)$, if $y = t$, do not change the weights else update the weights. It should be noted that the choice of the learning rate does not matter because it just changes the scaling of the weights, w and the perceptron is guaranteed to converge in a finite number of steps if the problem is separable but may be unstable if the problem is inseparable.

2.3.1.3 Activation Functions

There exist several activation functions. These are; identity function, step function, logistic function (sigmoid), radial basis functions, derivatives and softmax.

a) Identity Function

The identity function is characterised by the equation:

$$f(x) = x \tag{2.4}$$

b) Step Function

The step function is characterised by the function:

$$f(x) = 0 \text{ if } x \leq 0 \text{ and } f(x) = 1 \text{ if } x > 0 \tag{2.5}$$

c) Logistic Function (Sigmoid)

This function has the form $f(x) = \frac{1}{1 + e^{-ax}}$ where *a* determines how steep the function is.

The larger *a* is the steeper the function. The sigmoid function is generally used for a two class problem that has Gaussian input distributions.

d) Radial Basis Function

A radial basis function is simply a Gaussian; $f(x) = e^{-ax^2}$. It is zero everywhere except in a small region.

e) Derivatives

The derivative of the various functions above also form activation functions. The derivative of the identity function yields 1. The derivative of the step function is undefined, the derivative of the sigmoid function are easy to compute and yields:

$$\frac{df}{dx} = f(x)(1 - f(x)) \tag{2.6}$$

The tanh function is also used as an activation function and its derivative is:

$$\frac{df}{dx} = 1 - f(x)^2. \tag{2.7}$$

f) Softmax Function

The softmax function is characterised by the equation:

$$\frac{e^{a}}{\sum_{j} e_{j}^{a}} \tag{2.8}$$

The softmax function is generally used for a multi-class problem.

2.3.2 Radial Basis Function Networks

These kinds of networks consist of 2 layers, stacked together. The first layer with a Gaussian activation function and the second layer with a linear activation function. The input to the activation function is the distance between the layer weight vector and *i*, that is, $output = act(i - w)$. These networks are fast in training because the first layer can be initialised with meaningful values and the second layer is found through matrix inversion techniques. An iterative optimization technique is then used to refine the solution. RBF network is not used in this project due to the fact that RBF networks require more parameters than MLP neural networks. The computation nodes of the hidden layers of such a network are different and serve a different purpose from the output layer of the network as opposed to the MLP where the hidden and output layers share a common neuron model. The hidden layer, as discussed above, for the RBF network is non-linear and the output layer is linear hence the inability to approximate non-linear functions whereas in MLP both layers are non-linear [14]. The RBF network has the following architecture:

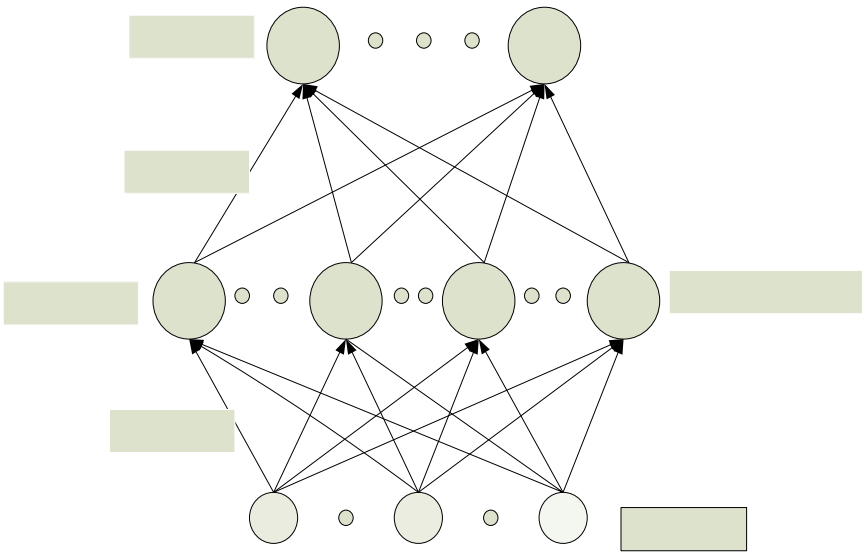


Figure 3: Architecture of a RBF neural network

With the following equations representing this network:

$$y_k(x) = \sum_{j=0}^{n} w_{kj}\phi_j(x) + b_j \qquad (2.9)$$

And

$$\phi_j(x) = \exp\left(-\frac{\|x-\mu_j\|^2}{2\sigma_j^2}\right) \qquad (2.10)$$

Where μ represents the centres and σ represents the widths of the network (training parameters to be optimised).

2.3.3 Recurrent Neural Networks

In these networks, there is the presence of recurrent or loop connections. These recurrent connections can, however, be unfolded to form feed-forward neural networks. These networks make efficient use of time varying information but are, however, complex to

design. This complexity arises from the fact that in order to use backpropagation algorithms with such architecture, there is a need to make the architecture feed-forward first, hence adding some computational expense. The inputs and outputs of this architecture are of arbitrary length sequences of vectors, not vectors. This also makes the handling of the input and outputs difficult to follow.

2.3.4 Hierarchical Mixtures of Experts

These networks are built out of modules, experts and gates, of which can be any of the other neural network types. The experts work on the problem in a small domain; meanwhile, the gates mix the opinions of the experts. The building of structure is data driven which poses a problem since as the structure would tend to fit the particular data it was trained for hence leading to over-fitting, which is a phenomenon to be avoided.

2.3.5 Self-Organising Map Networks

SOM is mainly used in the biomedical field such as in coronary heart risk assessment. It is relatively easy to implement and evaluate and is computationally not expensive. However, SOM has the problem of overcrowding and underutilization of the neurons in the network due to the fact that the size and shape of the network is fixed before the training phase begins.

2.3.6 Remarks on Network Architectures

The above sections have discussed briefly the different architectures available for neural network. Each section has given the short-falls of the various networks. MLP are,

however, the most appropriate network architecture for the project at hand since RBF networks require more parameters than MLP, RNN are complex to design due to the fact that they need to be unfolded, HME networks lead to over-fitting of the data and SOM networks have the problem of overcrowding and underutilization of the neurons in the network.

Chapter 3

Conventional Statistical Methods

This section focuses on introducing the various statistical methods, which have been applied to the stock market for stock prediction. The methods discussed in this chapter are the moving average methods [17, 18], exponential smoothing and linear regression. The chapter concludes with remarks and limitations of these statistical methods on the stock prediction.

3.1 Moving Average Methods

3.1.1 Simple Average

This method is suitable for data series with no trend/horizontal series. That is;

$$y_t = \beta_0 + \varepsilon_t, \quad \varepsilon_t \sim N(0, \sigma^2) \tag{3.1}$$

where β_0 may change slowly with time and ε_t is a noise signal with zero mean and a standard deviation σ. In this approach, the first n data points are averaged and used to forecast the next period as shown by formula (3.2).

$$F_{t+1} = \sum_{t=1}^{n} y_t / n \tag{3.2}$$

The simple average method is updated to a moving simple average whereby the n-period moving average (MA) calculated at time period t-1 is the average of the n most recent observations and this can be written as:

$$M_{t-1} = \frac{y_{t-1} + y_{t-2} + \ldots + y_{t-n+1} + y_{t-n}}{n} \tag{3.3}$$

As each new observation becomes available, a new moving average can be computed by dropping the oldest value and including the newest one

$$M_t = \frac{y_t + y_{t-1} + \ldots + y_{t-n+1}}{n} \tag{3.4}$$

M_t can also be calculated by:

$$M_t = M_{t-1} + \frac{y_t - y_{t-n}}{n} \tag{3.5}$$

We use the moving average calculated at time t to forecast the y value at time $t + 1$

$$F_t = M_t = \frac{y_t + y_{t-1} + \ldots + y_{t-n+1} + y_{t-n+1}}{n} \tag{3.6}$$

It should be noted that when the data have large randomness, a large n is used. Otherwise a small n is used. Data taken over a particular time always has some randomness associated therein. There exist methods for reducing or canceling the effect due to random variation. An often-used technique in industry is smoothing. This technique,

when properly applied, reveals more clearly the underlying trend, seasonal and cyclic components. There are two distinct groups of smoothing methods and these are; averaging methods and exponential smoothing methods. An alternative way to summarize the past data is to compute the mean of successive smaller sets of numbers of past data as follows; consider the set of numbers 9, 8, 9, 12, 9, 12, 11, 7, 13, 9, 11, 10 which is the Rand amount of 12 suppliers selected at random. Let us set *M*, the size of the smaller set equal to 3. Then the average of the first 3 numbers is: (9 + 8 + 9) / 3 = 8.667. This is called smoothing (i.e., some form of averaging). This smoothing process is continued by advancing one period and calculating the next average of three numbers, dropping the first number. Table 2 summarizes the process, which is referred to as *Moving Averaging*. The general expression for the moving average is

$$M_t = [Xt + X_{t\text{-}1} + \ldots + X_{t\text{-}N+1}] / N \qquad (3.7)$$

a) Results of Moving Average

Unfortunately, neither the mean of all data nor the moving average of the most recent *M* values, when used as forecasts for the next period, is able to cope with a significant trend. There exists a variation on the MA procedure that often does a better job of handling trend. It is called Double Moving Averages for a Linear Trend Process. It calculates a second moving average from the original moving average, using the same value for *M*. As soon as both single and double moving averages are available, a computer routine uses these averages to compute a slope and intercept, and then forecasts one or more periods ahead

Table 2: The Mean Square Error (MSE) = 2.018 as compared to 3 for a simple averaging process.

Supplier	R	MA	Error	Error squared
1	9			
2	8			
3	9	8.667	0.333	0.111
4	12	9.667	2.333	5.444
5	9	10.000	-1.000	1.000
6	12	11.000	1.000	1.000
7	11	10.667	0.333	0.111
8	7	10.000	-3.000	9.000
9	13	10.333	2.667	7.111
10	9	9.667	-0.667	0.444
11	11	11.000	0	0
12	10	10.000	0	0

3.2 Exponential Smoothing

Exponential smoothing is a technique used in time series analysis. This differs from the simple moving average in that, whereas in the simple moving average, past observations are weighted equally, exponential smoothing assigns exponentially decreasing weights as the observation (data) gets older thereby ensuring that recent observations are given

relatively more weight in the forecasting than the older data. There exists single, double, and triple exponential smoothing which would be described in the next subsections.

3.2.1 Single Exponential Smoothing

In this technique, the first smoothed exponential prediction S_2 is taken as the actual observed value. For any time period, the exponential prediction is:

$$S_t = \alpha y_{t-1} + (1-\alpha)S_{t-1} \qquad 0 < \alpha \leq 1 \qquad t \geq 3 \qquad (3.8)$$

Where the parameter α is called the smoothing constant. The initial observation is computed by averaging the first four or five observations or initializing it to y_1.

3.2.2 Double Exponential Smoothing

It should be noted that single exponential smoothing does not excel in data where there is a trend. As such, this situation can be improved by the introduction of a second equation with a second constant, γ, which is chosen in conjunction with α. These two equations are:

$$S_t = \alpha y_t + (1-\alpha)(S_{t-1} + b_{t-1}) \qquad 0 \leq \alpha \leq 1 \qquad (3.9)$$

$$b_t = \gamma(S_t - S_{t-1}) + (1-\gamma)b_{t-1} \qquad 0 \leq \gamma \leq 1 \qquad (3.10)$$

$$F_{t+m} = (S_t + mb_t) \qquad \text{Forecast for m - periods - ahead} \qquad (3.11)$$

S_1 is generally set as y_1, which is the original observation. Parameter b_1 can be chosen as follows:

$$b_1 = y_2 - y_1 \qquad (3.12)$$

$$b_1 = [(y_2 - y_1) + (y_3 - y_2) + (y_4 - y_3)]/3 \qquad (3.13)$$

$$b_1 = (y_n - y_1)/(n-1) \tag{3.14}$$

The first equation adjusts S_t for the trend of the previous period, b_{t-1}, by adding it to the last value, S_{t-1}. Meanwhile the second equation updates the trend. The value of the parameters α and γ are obtained through non-linear optimization techniques, such as the Marquardt Algorithm [19, 20].

3.2.3 Triple Exponential Smoothing

If the data, however, involves trend and seasonality, the double smoothing does not work. A third equation is then introduced which takes care of the seasonality. There are thus three equations and these sets of equations are known as the "Holt-Winters" (HW) equations named after the inventors. The equations are:

$$S_t = \alpha \frac{y_t}{I_{t-L}} + (1-\alpha)(S_{t-1} + b_{t-1}) \quad \text{Overall Smoothing} \tag{3.15}$$

$$b_t = \gamma(S_t - S_{t-1}) + (1-\gamma)b_{t-1} \quad \text{Trend Smoothing} \tag{3.16}$$

$$I_t = \beta \frac{y_t}{S_t} + (1-\beta)I_{t-L} \quad \text{Seasonal Smoothing} \tag{3.17}$$

$$F_{t+m} = (S_t + mb_t)I_{t-L+m} \quad \text{Forecast} \tag{3.18}$$

Where y is the observation, S is the smoothed observation, b is the trend factor, I is the seasonal index, F is the forecast at m periods ahead, t is an index denoting the time period, and α, β and γ are constants to be estimated such that the mean square error is minimized.

A full season's data is required in order to determine the seasonal parameter *I*, thereby initializing the equations. The trend factor requires two complete seasons for its determination since as a full season has L periods.

a) Initial values for the trend factor

Consider the example where the data consists of 6 years with 4 periods (that is, 4 quarters) per year. Then

Step 1: Compute the averages of each of the 6 years

$$A_p = \sum_{i=1}^{4} y_i \qquad p=1, 2,\ldots, 6 \tag{3.19}$$

Step 2: Divide the observations by the appropriate yearly mean

Table 3: Table of yearly means

1	2	3	4	5	6
y_1/A_1	y_5/A_2	y_9/A_3	y_{13}/A_4	y_{17}/A_5	y_{21}/A_6
y_2/A_1	y_6/A_2	y_{10}/A_3	y_{14}/A_4	y_{18}/A_5	y_{22}/A_6
y_3/A_1	y_7/A_2	y_{11}/A_3	y_{15}/A_4	y_{19}/A_5	y_{23}/A_6
y_4/A_1	y_8/A_2	y_{12}/A_3	y_{16}/A_4	y_{20}/A_5	y_{24}/A_6

Step 3: Now the seasonal indices are formed by computing the average of each row. Thus the initial seasonal indices (symbolically) are:

$$I_1 = \left(y_1/A_1 + y_5/A_2 + y_9/A_3 + y_{13}/A_4 + y_{17}/A_5 + y_{21}/A_6\right)/6 \tag{3.20}$$

$$I_2 = \left(y_2/A_1 + y_6/A_2 + y_{10}/A_3 + y_{14}/A_4 + y_{18}/A_5 + y_{22}/A_6\right)/6 \tag{3.21}$$

$$I_3 = \left(y_3/A_1 + y_7/A_2 + y_{11}/A_3 + y_{15}/A_4 + y_{19}/A_5 + y_{23}/A_6\right)/6 \tag{3.22}$$

$$I_4 = \left(y_4/A_1 + y_8/A_2 + y_{12}/A_3 + y_{16}/A_4 + y_{20}/A_5 + y_{24}/A_6\right)/6 \quad (3.23)$$

3.3 Linear Regression

The linear least squares regression is the most widely used modeling method [21, 22]. It is sometimes referred to as "regression", "linear regression" or "least squares" to fit a model to their data set. It has also been adapted to a broad range of situations. Linear least squares regression can be used to fit data with any function of the form:

$$f\left(\underset{x}{\rho} : \underset{\beta}{\mu}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots \quad (3.24)$$

Where $\beta_0, \beta_1, \beta_3 \ldots$ are model parameters to be estimated.

In the least squares method the unknown parameters are estimated by minimizing the sum of the squared deviations between the data and the model. The minimization process reduces the over determined system of equations formed by the data to a sensible system of *P* (where *P* is the number of parameters in the functional part of the model) equations in *P* unknowns. This new system of equations is then solved to obtain the parameter estimates. Linear models are not limited to being straight lines or planes, but include a fairly wide range of shapes. For example, a simple quadratic curve

$$f(x; \underset{\beta}{\mu}) = \beta_0 + \beta_1 x + \beta_{11} x^2 \quad (3.25)$$

is linear in the statistical sense. A straight-line model in log(x)

$$f(x; \underset{\beta}{\mu}) = \beta_0 + \beta_1 \ln(x) \quad (3.26)$$

or a polynomial in sin(x)

$$f(x;\vec{\beta}) = \beta_0 + \beta_1 \sin(x) + \beta_2 \sin(2x) + \beta_3 \sin(3x) \tag{3.27}$$

is also linear in the statistical sense because they are linear in the parameters, though not with respect to the observed explanatory variable, x. Though there are types of data that are better described by functions that are nonlinear in the parameters, many processes in science and engineering are well-described by linear models. This is because either the processes are inherently linear or because, over short ranges, any process can be well-approximated by a linear model. The theory associated with linear regression is well-understood and allows for construction of different types of easily-interpretable statistical intervals for predictions, calibrations, and optimizations. These statistical intervals can then be used to give clear answers to scientific and engineering questions. The main disadvantages of linear least squares are limitations in the shapes that linear models can assume over long ranges, possibly poor extrapolation properties, and sensitivity to outliers. Linear models with nonlinear terms in the predictor variables curve relatively slowly, so for inherently nonlinear processes it becomes increasingly difficult to find a linear model that fits the data well as the range of the data increases. As the explanatory variables become extreme, the outputs of the linear model will also always be more extreme. This means that linear models may not be effective for extrapolating the results of a process for which data cannot be collected in the region of interest. Of course extrapolation is potentially dangerous regardless of the model type. Finally, while the method of least squares often gives optimal estimates of the unknown parameters, it is very sensitive to the presence of unusual data points in the data used to fit a model.

3.3.1 Least Squares

In least squares (LS) estimation, the unknown values of the parameters, $\beta_0, \beta_1, ...,$ in the regression function, $f(\underset{\rho}{x}; \underset{\rho}{\beta})$, are estimated by finding numerical values for the parameters that minimize the sum of the squared deviations between the observed responses and the functional portion of the model. Mathematically, the least (sum of) squares criterion that is minimized to obtain the parameter estimates is

$$Q = \sum_{i=1}^{n} \left[y_i - f(\underset{\rho}{x_i}; \underset{\rho}{\beta}) \right]^2 \tag{3.28}$$

As previously noted, $\beta_0, \beta_1, ...,$ are treated as the variables in the optimization and the predictor variable values, $x_1, x_2, ...$ are treated as coefficients. To emphasize the fact that the estimates of the parameter values are not the same as the true values of the parameters, the estimates are denoted by $\hat{\beta}_0, \hat{\beta}_1, ...$ For linear models, the least squares minimization is usually done analytically using calculus. For nonlinear models, on the other hand, the minimization must almost always be done using iterative numerical algorithms. To illustrate, consider the straight-line model,

$$y = \beta_0 + \beta_1 x + \varepsilon \tag{3.29}$$

For this model the least squares estimates of the parameters would be computed by minimizing

$$Q = \sum_{i=1}^{n} \left[y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i) \right]^2 \tag{3.30}$$

Doing this by

1. taking partial derivatives of Q with respect to $\hat{\beta}_0$ and $\hat{\beta}_1$,
2. setting each partial derivative equal to zero, and
3. solving the resulting system of two equations with two unknowns

yields the following estimators for the parameters:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{3.31}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x} \tag{3.32}$$

These formulas are instructive because they show that the parameter estimators are functions of both the predictor and response variables and that the estimators are not independent of each other unless $\bar{x} = 0$. This is clear because the formula for the estimator of the intercept depends directly on the value of the estimator of the slope, except when the second term in the formula for $\hat{\beta}_0$ drops out due to multiplication by zero. This means that if the estimate of the slope deviates a lot from the true slope, then the estimate of the intercept will tend to deviate a lot from its true value too. This lack of independence of the parameter estimators, or more specifically the correlation of the parameter estimators, becomes important when computing the uncertainties of predicted values from the model. Although the formulas discussed in this paragraph only apply to the straight-line model, the relationship between the parameter estimators is analogous for more complicated models, including both statistically linear and statistically nonlinear models. Like the parameters in the functional part of the model, σ is another measure of the average quality of the fit of a regression function to a set of data by least squares,

which is generally not known, but can be estimated from the least squares equations. The formula for the estimate is

$$\hat{\sigma} = \sqrt{\frac{Q}{n-p}} \tag{3.33}$$

$$= \sqrt{\frac{\sum \left[y_i - f(\underset{\rho}{x}_i; \underset{\rho}{\hat{\beta}}) \right]^2}{n-p}} \tag{3.34}$$

with n denoting the number of observations in the sample and p is the number of parameters in the functional part of the model. Parameter $\hat{\sigma}$ is often referred to as the "residual standard deviation" of the process. Because σ measures how the individual values of the response variable vary with respect to their true values under $f(\underset{\rho}{x}; \underset{\mu}{\beta})$, it also contains information about how far from the truth quantities derived from the data, such as the estimated values of the parameters, could be.

3.4 Remarks and Conclusion

Statistical methods employed in the prediction of stock market prices have been presented in this chapter. These methods, however, involve complex and rigorous manipulations as the data set tends to increase. As presented in Section 3.1.1, simple average methods are suitable for data series with no trend (horizontal series) hence will not be suitable in the stock market, which sometimes has trends. The selection of the moving average model parameters in the statistical model, also involve further complexities such as non-linear techniques, Marquardt Algorithm. Linear regression methods discussed in Section 3.3 are disadvantageous in that they are limited in the shape they assume over long ranges hence leading to poor extrapolation properties. Also, linear

models with non-linear terms curve relatively slowly. Finally, while the method of least squares often gives optimal estimates of the unknown parameters, it is very sensitive to the presence of unusual data points in the data used to fit a model. To this effect the neural network model is more beneficial for the prediction of stock market prices since the mathematical process involved is minimal and a non-linear model can be developed with less computation required compared to the statistical methods.

Chapter 4

Markov Chain Monte Carlo Sampling

A Markov chain is a succession of elements each of which can be generated from a finite (usually small) number of elements preceding it, possibly with some random element added. A Markov chain can also be considered as a sequence of random values whose probabilities at a time interval depends upon the value of the number at the previous time. A simple example is the no returning random walk, where the walkers are restricted to not go back to the location just previously visited. Sampling methods which rely on Markov chain theory are iterative: the principle is to build a succession of states, and once convergence is reached, the consecutive states are assumed to be drawn from the target probability distribution. With these methods, it is possible to sample from general probability distributions, whereas direct sampling algorithms only apply to specific probability distributions such as the Gaussian distribution. The probability distribution can be a posterior distribution in a Bayesian context, which makes Markov Chain Monte

Carlo (MCMC) methods very attractive in Bayesian computation. Markov chain Monte Carlo is a technique used by Bayesian practitioners to sample from the posterior distribution. The Monte Carlo method is, in general terms, any technique used for obtaining solutions to deterministic problems using random numbers.

Markov chain Monte Carlo methods can be used in importance sampling, when in generating each point not only random numbers are used, but the previously generated point(s) enter with some weight, in the simplest case by a random walk, where $x_{new} = x_{old} + r$, with r a random vector. The controlling factor in a Markov chain is the transition probability; it is a conditional probability for the system to go to a particular new state, given the current state of the system. Fairly efficient estimates can be determined from the proper transition probabilities. Markov chains can be used to solve a very useful class of problems in a rather remarkable way. Suppose we wanted to find the value of the vector x that is the solution to,

$$x = Ax + f \tag{4.1}$$

where the $n \, x \, n$ matrix A, and the vector f are known. By setting up a random walk through the matrix A we can solve for any single component of x. A little mathematics is needed to see how this would work. First lets symbolically solve (4.1),

$$x = (I - A)^{-1} f \tag{4.2}$$

This can be expanded to,

$$x = f + Af + A^2 f + A^3 f + \ldots = \sum_{m=0}^{\infty} A^m f \tag{4.3}$$

Now lets suppose we have an $n \; x \; n$ matrix of probabilities, P, such that,

$$p_{ij} \geq 0 \tag{4.4}$$

$$\sum_j p_{ij} \leq 1 \tag{4.5}$$

and we have an array,

$$\Im_i = 1 - \sum_j p_{ij} \tag{4.6}$$

further we will define,

$$\nu_{ij} = \begin{cases} \dfrac{\Im_{ij}}{p_{ij}} & \text{if } \mathrm{p_{ij}} \neq 0 \\ 0 & \text{if } \mathrm{p_{ij}} = 0 \end{cases} \tag{4.7}$$

P can then describe a Markov chain where the states of the chain are n integers. The element p_{ij} gives the transition probability for the random walk to go from state i to state j. As long as g is not zero the walk will eventually terminate. The probability that the walk will terminate after state i is given by $\Im_i$. While taking the random walk we need to accumulate the product,

$$V_m = \nu_{i_0 i_1} \nu_{i_1 i_2} \ldots \nu_{i_{m-1} i_m} \tag{4.8}$$

and the sum,

$$W_k = \frac{1}{g_k} \sum_{m}^{k} V_m f_m \tag{4.9}$$

The final W value is important because it's mean value (averaged over the walks that start at index i) is,

$$\overline{W} = \sum_{k} \sum_{i_1} \cdots \sum_{i_k} p_{ii_1} \ldots p_{i_{k-1}i_k} v_{ii_1} \ldots v_{i_{k-1}i_k} f_{i_k} / g_{i_k} \tag{4.10}$$

$$= \sum_{k} \sum_{i_1} \cdots \sum_{i_k} a_{ii_1} \cdots a_{i_{k-1}i_k} f_{i_k} \tag{4.11}$$

$$= f_i + (Af)_i + (A^2 f)_i + (A^3 f)_i + \cdots \tag{4.12}$$

Notice that the final form of (4.12) is exactly the *i-th* element from (4.3). So to solve this problem we have three major steps:

- Set up the probabilities p and g and start off the system at the index at which we want to solve for x, lets call that index i.
- Then we take a random walk until the walk terminates, accumulating the product V and the sum W.
- Then we take the average of the W values over several walks to obtain our estimate of x_i.

This will work as long as (4.3) converges, this will happen if the norm of A,

$$\|A\| = \max_i \left(\sum_j |a_{ij}| \right) \tag{4.13}$$

is less than one (the smaller $\|A\|$ is the faster the Monte Carlo estimate will converge). If the norm is larger than one, all is not lost; there is usually some manipulation that can be done to get a new matrix that has a small norm. It turns out we can use this idea for all sorts of problems that have the same general form as (4.1). If we write (4.1) as,

$$x = Ax + f$$

and now consider A to be any linear operator that can operate on x, not just a matrix multiply. Given the appropriate operator for a given problem, we can use the above method to solve several kinds of problems. We can do a matrix inverse, i.e. solve

$$f = Hx$$

if we let $A = I - H$. Starting out at index i, will give us row i of H^{-1}. If we restrict the chains to start at index i and end at index j, then we obtain a single element of the inverse, H_{ij}^{-1}. Other problems that can be solved this way include the determination of eigenvalues and eigenvectors, and integral equations of the second kind such as,

$$x(t) = \int_a^b A(s,t)x(s)ds + f(t) \tag{4.14}$$

Notice that (4.14) has the same kind of form as (4.1), (integration is a linear operator). If we made a discrete grid upon which we wanted to solve (4.14) then we could use exactly the same code that we used to solve (4.1). However, in a practical application the

dimension of (4.14) would be extremely large, or $A(s,t)$ would be so complicated to calculate that it is not really practical to create a giant matrix to approximate the integral. Instead we free up our random walk to apply continuously within the range $[a,b]$. These probability density functions are explained in the next subsection.

4.1 Probability Density Function

If a random variable X has a cumulative distribution (Section 4.2) function $F(x)$ which is differentiable, the probability density function is defined as $f(x) = dF/dx$. The probability of observing X in the interval $x \le X \le x + dx$ is then $f(x)dx$. For several variables $X_1, X_2, ..., X_n$ the joint probability density function is

$$f(x_1, x_2, ..., x_n) = \partial^n / (\partial x_1 \partial x_2 ... \partial x_n) F(x_1, x_2, ..., x_n) \tag{4.15}$$

The transformation of a given probability density function $f(x)$ to the probability density function $g(y)$ of a different variable $y = y(x)$ is achieved by

$$g(y) = \frac{f(x)}{|dy/dx|} \tag{4.16}$$

The assumption has to be made for $y(x)$ to be an increasing or decreasing function, in order to have a one-to-one relation.

4.2 Distribution

A distribution of measurements or observations is the frequency of these measurements shown as a function of one or more variables, usually in the form of a histogram.

Experimental distributions can thus be compared to theoretical probability density functions. The term distribution function is short for cumulative distribution function and describes the integral of the probability density function: a random variable X has the (cumulative) distribution function $F(x)$, if the probability for an experiment to yield an $X < x$ is:

$$F(x) = P(X < x) = \int_{-\infty}^{x} f(\xi) d\xi \tag{4.17}$$

For several random variables $\overline{X} = (X_1, X_2, ..., X_n)$ the joint distribution function is

$$F(\underset{\sim}{x}) = F(x_1, x_2, ..., x_n) = P(X_1 < x_1, X_2 < x_2, ..., X_n < x_n) \tag{4.18}$$

The next section deals with the Monte Carlo methods.

4.3 Monte Carlo Methods

The systematic use of samples of random numbers in order to estimate parameters of an unknown distribution by statistical simulation. Methods based on this principle of random sampling are indicated in cases where the dimensionality and/or complexity of a problem make straightforward numerical solutions impossible or impractical. The method is ideally adapted to computers, its applications are varied and many, its main drawbacks are potentially slow convergence (large variances of the results), and often the difficulty of estimating the statistical error (variance) of the result. Monte Carlo problems can be formulated as integration of a function $f = f(\underset{\sim}{x})$ over a (multi-dimensional) volume V, with the result

$$\int_V f dV = V\ \bar{f}, \tag{4.19}$$

Where the average of f, $\bar{f}$ is obtained by exploring randomly the volume V.

Most easily one conceives a simple (and inefficient) *hit-and-miss Monte Carlo*: assume, for example, a three-dimensional volume V to be bounded by surfaces difficult to intersect and describe analytically; on the other hand, given a point (x,y,z), it is easy to decide whether it is inside or outside the boundary. In this case, a simply bounded volume which fully includes V can be sampled uniformly (the components x,y,z are generated as random numbers with uniform probability density function), and for each point a weight is computed, which is zero if the point is outside V, one otherwise. After N random numbers, $n \leq N$ will have been found inside V, and the ratio n/N is the fraction of the sampled volume which corresponds to V.

Another method, *crude Monte Carlo*, may be used for integration: assume now the volume V is bounded by two functions $z(x, y)$ and $z'(x, y)$, both not integrable, but known for any x, y, over an interval Δx and Δy. Taking random pairs (x,y), evaluating $\Delta z = |z(x, y) - z'(x, y)|$ at each point, averaging to $\langle \Delta z \rangle$ and forming $\Delta x \Delta y \langle \Delta z \rangle$, gives an approximation of the volume. Often, the function to be sampled is, in fact, a probability density function. Variance-reducing techniques will then be indicated, like importance sampling or stratified sampling.

4.3.1 Monte Carlo Integration

There are two major Monte Carlo techniques for evaluating such integrals. The first method is based upon an idea similar to the rejection method of generating random variables for arbitrary distribution functions. Suppose we wish to evaluate the integral,

$$I = \int_a^b g(x)dx \tag{4.20}$$

If we put a bounding box around the function $g(x)$, then the integral of $g(x)$ can be understood to be the fraction of the bounding box that is also within $g(x)$. So if we choose a point at random uniformly within the bounding box, the probability that the point is within $g(x)$ is given by the fraction of the area that $g(x)$ occupies. The integration scheme is then to take a large number of random points with the box and count the number that is within $g(x)$ to get the area,

$$I \approx \frac{n^*}{n} V \tag{4.21}$$

where, n^* is the number of points within $g(x)$, n is the total number of points generated, and V is the volume of the bounding box.

This method is very inefficient. Many points are required to make (4.21) converge towards (4.20) with any degree of precision. A more efficient approach is to note that we can write (4.20) as,

$$I = \int_a^b g(x) f(x) dx = \frac{1}{V} \int_a^b g(x) f(x) V dx \tag{4.22}$$

if we define $f(x)$ as,

$$f(x) = \begin{cases} 1 & \text{if x is in the domain} \\ 0 & \text{otherwise} \end{cases} \tag{4.23}$$

(again V is the volume of the domain). (4.22) can be interpreted as the expectation of the function, $h(x) = g(x) f(x) V$ for the random variable x, which is uniformly distributed within the domain. This then gives an approximate procedure,

$$I \approx \frac{1}{n} \sum_i^n h(x_i) = \frac{V}{n} \sum_i^n g(x_i) \tag{4.24}$$

Estimates based upon (4.24) converge much more quickly than those based upon using (4.21). If pseudo-random numbers are used for the Monte Carlo evaluation of integrals then, because of the clumps and voids in any given sample, there will be regions of the integral that are under represented as well as overrepresented. In the long run it is not a problem since we know that the numbers represent a uniform distribution well. But the long run means using lots of iterations.

Probably the most effective way to speed up the convergence of Monte Carlo integration is to use quasi-random numbers instead of pseudo-random numbers for choosing the points. In general this change will cause the integration estimate to converge towards the actual solution like $(\ln n)^N / n$ (where N is the number of dimensions in the integral)

instead of the usual $1/\sqrt{n}$. This improved convergence is considerably better, almost as fast as $1/n$.

4.3.2 Variance Reducing Technique

4.3.2.1 Importance Sampling

Importance sampling is a technique for numerically approximating an integral. It is also called biased sampling and is one of the variance-reducing techniques in Monte Carlo methods. It is mentioned here as a basis for the numerical concepts which follow. It is similar to stratified sampling in that the fundamental idea is that the sampling process is distorted, to take into account the weighting of the underlying distribution. An example of importance sampling in a Monte-Carlo context, but the basic principle is as follows; In wanting to estimate:

$$I = \int_{-\infty}^{\infty} g(x) f(x) dx \qquad (4.25)$$

where $f(x)$ is a density function, one could sample n values of x from $f(x)$ and then approximate with

$$\hat{I} = \frac{1}{n} \sum_{i=1}^{n} g(x_i) \qquad (4.26)$$

Alternatively, m values of x could be sampled from another density $h(x)$ and then I could be estimated using

$$\hat{I} = \frac{1}{m}\sum_{i=1}^{m}\frac{g(x_i)f(x_i)}{h(x_i)} \tag{4.27}$$

Consideration can then be made as to how $h(x)$ may be chosen so that the estimator is most efficient. It turns out that the most efficient form for $h(x)$ samples from areas where $g(x)$ is large, provided that $f(x)$ is not small, [23]. Such ideas are important in any method when simulating from the posterior.

4.3.2.2 Stratified Sampling

Consider a set of N types of job within an organization, which has a total of M employees. Let J_j where $1 \le j \le N$ be the number of people who have a job of type j with all people doing the same type of job getting paid the same salary. Then, clearly;

$$\sum_{j=1}^{N} J_j = M \tag{4.28}$$

If interested in the average salary paid and if M is very large the average may be approximated as follows;

$$\mu_X = \overline{X} = \frac{1}{m}\sum_{i=1}^{m} X_i \tag{4.29}$$

where we sample a total of m people from the organization and X_i is the salary paid to the i^{th} person we sampled. Ordinary random sampling would involve picking the m people uniformly from the total population of M people in the organization. However, another method would be to ensure that the probability of choosing a person from job

type *j* is the number of people doing job type *j* divided by the total number of people, *M*. This latter idea is just stratified sampling and is an important and well known sampling technique.

4.4 Metropolis Hastings Algorithm

The Metropolis-Hastings algorithm is a Markov chain Monte Carlo method as described previously. The algorithm sets about constructing a Markov matrix which has as its equilibrium distribution some target density ϕ, of interest to the operator. The algorithm requires the specification of a proposal density, q_j^i, which is a probability density for *j* and may depend upon *i*. This is then used in order to propose transitions from *i*. The condition of detailed balance is then imposed in the following fashion.

Construct α_j^i by imposing detailed balance, so that the matrix with entries given by $M_j^i = q_j^i \bullet \alpha_j^i$ is a Markov matrix. This is done as follows:

$$If\ q_j^i \phi_i = q_i^j \phi_j, \text{ then } \alpha_j^i = \alpha_i^j = 1 \tag{4.30}$$

Otherwise, assume (without loss of generality) that:

$$q_j^i \phi_i > q_i^j \phi_j, \tag{4.31}$$

then setting $\alpha_i^j = 1$, and constructing:

$$\alpha_j^i = \frac{q_i^j \phi_j}{q_j^i \phi_i}, \tag{4.32}$$

detailed balance holds. Thus, by defining in general:

$$\alpha_j^i = \min\left[\frac{q_i^j \phi_j}{q_j^i \phi_i}, 1\right], \tag{4.33}$$

detailed balance is satisfied. In order that what has been constructed is a Markov matrix which will generate a chain having ϕ as the invariant distribution, it remains to show that *M* is indeed Markov. This imposes conditions on the form of q which is related in turn to ϕ. The conditions are as referred to before, aperiodicity and connectedness. These are indeed satisfied for quite a large family of densities [24, 25].

The algorithm then, works as follows;

1. Set $i = 0$; Set N = some large value; Choose an initial state x_0.
2. Propose y from $q_y^{x_i}$.
3. Accept the proposal with probability $\alpha_j^{x_i}$.
4. If accepted, set $x_{i+1} = y$, else set $x_{i+1} = x_i$.
5. If $i<N$ set $i=i+1$; and back to step 2.

Although theory demonstrates that a chain constructed using this algorithm has a limiting distribution which is the target distribution, the question of the rate at which the limiting distribution is attained is still open.

Note that the samples $x_0, x_1, ..., x_j, ...$ generated by the chain will depend upon the choice of x_0 and only when close to the limiting distribution are the samples to be considered as having come from the target distribution. What size should *N* be, and for what minimum *j* should x_j be considered as a sample from the target? A number of methods have been proposed in order to answer these questions. Diagnostic methods of Gelman and Rubin [26] and others are reviewed by Cowles and Carlin [27]. Murdoch and Green have developed methods of demonstrating convergence [28], but these methods are far less practical than the heuristic diagnostics described elsewhere. A review of methods to date including those of Murdoch and Green is provided by Brooks and Roberts [29]. The Metropolis-Hastings algorithm is valid for sampling from the $\phi(x)$, for $x \in \Re^n$, that is for a general vector, *x*. However, in practice it can be more natural to consider *x* as the combination of subvectors $x = \{x_1, x_2\}$. It turns out [30] that a transition matrix for a chain which converges to the target $\phi(x)$ may be constructed by considering matrices for a chain which samples from $\phi(x_1 \mid x_2)$ a The Metropolis-Hastings algorithm is a Markov chain Monte Carlo method as described previously. The algorithm sets about constructing a Markov matrix which has as its equilibrium distribution some target density ϕ, of interest to the operator. The algorithm requires the specification of a proposal density, q_j^i which is a probability density for *j* and may depend upon *i*. This is then used in order to propose transitions from *i*.

4.5 Remarks

For the sampling schemes mentioned above, the target distribution is invariant and the rate of convergence issues will be important. Two issues arise for consideration, which are:

- When will the samples be independent of the initial value, x_0?
- What number of samples, N is needed?

The first question refers to the fact that the initial value is just some arbitrary guess and is unlikely to have come from the target distribution and it may actually take a while for the estimate to come from the actual target distribution, which is that time after which the samples can be used. This time is referred to as the burn-in time. Methods to determine this burn-in time are, however, tedious if not impracticable. The second point depends on what is being estimated and how accurate the estimator needs to be. The choice of the proposal distribution is determinative for the rate of convergence. It is also important that the target distribution be explored by the Markov chain. The acceptance rate is a measure of the level of exploring. If it is too low, then the chain is too stationary and does not move around much. If the acceptance rate is too high, this indicates that the chain does not have the opportunity to sample from the tails of the distribution.

CHAPTER 5

Bayesian Methodology for Statistical Modeling

This chapter focuses on the statistical methodology employed in the design process of this project which is the Bayesian approach. In Bayesian data analysis, all uncertain quantities are modeled as probability distributions, and inference is performed by constructing the posterior conditional probabilities for the unobserved variables of interest, given the observed data sample and some prior assumptions. This chapter focuses on the Bayesian approach for computational intelligence (Neural Networks). The major difficulty in neural networks model building is controlling the level of complexity of the model. With the standard neural network techniques, the correct model complexity is often chosen by crude methodology and is generally computationally expensive. Bayesian approach handles these issues by defining vague priors for the hyperparameters that determine the model complexity. The resulting model is averaged over all model complexities weighted by their posterior probability given the data sample. Another problem with standard neural network models is the lack of tools for analyzing the

results. The Bayesian analysis caters for this with the posterior distributions for the variables been estimated. In this chapter, the Bayesian approach in statistical modeling is discussed (5.1) [31].

5.1 Bayesian Approach

Bayesian inference is different from classical inference. In Bayesian inference, previous information is important. The key principle of Bayesian approach is to construct the posterior probability distributions for all the unknown parameters of the model, given the data sample. Bayesian statistics incorporates prior information directly into the analysis and it has a naturally formulated decision structure. Use of the posterior probabilities requires a definition of the prior probabilities for the unknown parameters.

5.1.1 Bayesian Methodology

Statistics is concerned with the estimation of numerical quantities. In the Bayesian context, the quantities of interest will be random variables or parameters. Before an experiment or survey, the prior knowledge about the quantities of interest is summarized in the form of a probability statement. Let the parameters of interests be θ and the model be represented by H. Also, H represents all the hypotheses and assumptions that are made when defining the model, for example the choice of multi-layer perceptron networks. The probability statement about initial beliefs is denoted $P(\theta / \mathrm{H})$ and is termed the prior belief. Since this is a probability statement it takes the form of a probability distribution and is often referred to as the prior distribution, or more simply the prior.

5.1.2 Prior Knowledge

It is essential, when considering θ as a random variable, to assign prior probabilities, simply because such must exist. In the case where prior knowledge shows that no particular value or values of θ are more likely than any others, then θ will be uniformly distributed. That is to say, $P(\theta / \mathrm{H})$ tends to one. The prior might also take the form of a normal distribution with some mean and (perhaps large) variance. It should be noted that all generalization is based on prior knowledge. The training samples provide information only at those points, and the prior knowledge provides the necessary link between the training samples and the future samples not yet determined.

5.1.3 Model or Likelihood

The idea of likelihood is common to all statistical inference, and is well understood by Bayesian statisticians. The relationship between the parameters of a model and the observables is fundamental to the process of updating knowledge of parameters based upon the data. The likelihood is sometimes termed the model, and takes the form of a probability statement $P(D / H)$, where D represents the given data of the system. Note that the likelihood is a conditional probability statement as to how likely it is for D to be observed if the parameters take the value θ. In a statistical analysis, it is the knowledge of θ which is of interest, that is to say, the distribution of θ given that D is observed. This is termed the posterior, and is dealt in the next section. Other methods of inference concentrate on the likelihood in their analysis, in which case the focus is $P(D / \theta)$ as a function of θ for fixed D. While $\int_{\nabla D} P(D / \theta)\, dD = 1$, the same is not true of the integral

with respect to θ. For this reason, and to avoid confusion, the likelihood is sometimes written $L(D/\theta)$.

5.1.4 Posterior Distribution

Of interest to the modeler, then, is the conditional distribution of the parameters, given the data, that is $P(\theta/D)$. The posterior probability for the parameters θ in a model H given the data D is, according to Bayes' rule,

$$p(\theta \mid D,H) = \frac{p(D \mid \theta,H)\,p(\theta \mid H)}{p(D \mid H)}, \tag{5.1}$$

Where $p(D \mid \theta,H)$ is the likelihood of the parameters θ (discussed in the previous section), $p(\theta/\mathrm{H})$ is the prior probability of θ, and $p(D/H)$ is a normalizing constant, called the evidence of the model H. The $p(\theta \mid D,H)$ distribution is termed the posterior distribution and describes the current state of knowledge about θ, given the initial knowledge of θ, together with the model H, such knowledge having been updated by information from the probability

$$p(D/H) = \int_{\theta} p(D \mid \theta,H)\,p(\theta \mid H)\,d\theta. \tag{5.2}$$

This normalization constant is the marginal probability of the data, conditional on H, integrated over everything with the chosen assumptions H, and prior distribution $p(\theta/\mathrm{H})$. The Bayesian method is then quite straightforward:

- construct a model, obtaining a likelihood $p(D/\theta)$;

- elicit a prior distribution $p(\theta / \mathrm{H})$;
- derive the posterior density $P(\theta / D, H)$ as above.

5.2 Bayesian Learning for MLP Networks

In this section of the chapter, a short overview of the Bayesian approach for neural networks is given. This section concentrates on the Multi-Layer Perceptron (MLP) networks and Markov Chain Monte Carlo (MCMC) methods for computing the integrations. The result of Bayesian modeling is the conditional probability distributions of the unknown parameters of interest, given the known data. In Bayesian MLP, the end parameters are the predictions of the model for new inputs. The posterior predictive distribution of output y^{new} for the new input x^{new} given the training data $D = \left\{\left(x^{(1)}, y^{(1)}\right), ..., \left(x^{(n)}, y^{(n)}\right)\right\}$ is obtained by integrating the predictions of the model with respect to the posterior distribution of the model,

$$p(y^{new} \mid x^{new}, D) = \int p(y^{new} \mid x^{new}, \theta) p(\theta \mid D) d\theta, \tag{5.3}$$

Where θ denotes all the model parameters and hyperparameters of the prior structures. The probability model for the measurements, *p(y|x, θ)*, contains the chosen approximation functions and noise models. It defines also the likelihood part in the posterior probability term, *p(θ|D)* α *p(D|θ)p(θ)*. The probability model in a regression problem with additive error is:

$$y = f(x : \theta_w) + e, \tag{5.4}$$

Where *f()* is the MLP function:

$$f(x, \theta_w) = b^2 + w^2 \tanh(b^1 + w^1 x). \quad (5.5)$$

θ_w denotes all the parameters w^1, b^1, w^2, b^2, which are the hidden layer weights and biases, and the output layer weights and biases, respectively.

5.3 Markov Chain Monte Carlo Method

In Markov Chain Monte Carlo (MCMC) the complex integrals in the marginalization are approximated via drawing samples from the joint probability distribution of all the model parameters and hyperparameters. For example, with squared error loss the best guess for model prediction (with additive zero-mean noise model), corresponds to the expectation of the posterior predictive distribution in (5.3):

$$\hat{y}^{new} = E\left[y^{new} \mid x^{new}, D\right] = \int f\left(x^{new}, \theta\right) p(\theta \mid D) d\theta. \quad \textbf{5.6}$$

This is approximated using a sample of values $\theta^{(t)}$ drawn from the posterior distribution of parameters:

$$\hat{y}^{new} \approx \frac{1}{N} \sum_{t=1}^{N} f(x^{new}, \theta^{(t)}). \quad \textbf{5.7}$$

In the MCMC, samples are generated using a Markov chain that has the desired posterior distribution as its stationary distribution. Choosing the initial values with early-stopping can be used to reduce the burn-in time, when the chain has not yet reached the equilibrium distribution. In general, the author's experience suggests that the convergence of the MCMC methods for MLP is slower than usually assumed, so that in

many of the published studies, the MCMC chains may have still been in the burn-in stage, producing a sort of early-stopping effect to the selection of the model complexity.

5.4 Conclusion and Remarks

Presented in this chapter is the Bayesian approach for statistical modeling. In the Bayesian approach, previous information is important for the analysis. This prior knowledge forms the prior probability distribution. The underlying principle is to construct posterior probability distributions for all the unknown parameters of the model, given the data sample. Also presented in this chapter is Bayesian learning for MLP networks. This is very important for the reformulated network methodology created (Section 1.2). The MCMC Bayesian learning implementation is also presented in this chapter, in Section 5.3. This was the implementation utilized for optimizing the reformulated network parameters, hence forms an important part of the thesis. The next chapter deals with the algorithms implemented.

Chapter 6

Input Time-Window Optimization Algorithms

This thesis started by introducing the stock market, and then proceeded to introducing neural network and the last chapter dealt with the Bayesian approach. The algorithms were implemented in neural networks using the various networks and later optimizing these networks. The methodology also included the use of Bayesian analysis. Both algorithms use the neural network NETLAB© toolbox that runs in MATLAB® [32].

The data used for this design was data obtained from the National Association of Securities Dealers (NASDAQ). The design process was divided into various stages. The following procedures are followed in designing the neural network architecture in this project:

1. Specify and process the data required by the neural network for training, validation and testing.

2. Create a neural network and train the neural network with the data in Step 1.

3. Optimize the input time-window using polynomial approximation.

4. Optimize the input time-window by reformulating neural networks methods (Bayesian framework).

5. Create an integrated infrastructure.

6. Comparison of the different networks.

This chapter would elaborate on the procedures stated above.

6.1 Factors Specification and Processing

The data used for the network design is obtained from the NASDAQ stock exchange. The NASDAQ all-share index was used as the sample data. The data for a two-year period was used as the analysis data. The output data set is obtained by calculating the average of the all-share index over 5 successive days. The data is then divided into 3 sets; training, validation and testing sets. The training data set is used to train the initial network. The validation data set is used to validate the network and the testing data set is used to confirm the predictability of the network. The division of data into three sets is to ensure that over-fitting and under-fitting are avoided. Over-fitting occurs when the network does not generalize but rather fits training data meanwhile underfitting occurs when the network does not follow the data at all.

The output and input data sets are first preconditioned by normalising them before the network is trained. Normalising the data sets makes the data lie between 0 and 1. This caters for over-fitting since large inputs and outputs values used during training results in the learning rates in the different layers being different by significant amounts. With the large values, a very small learning rate will be needed meaning a lot of steps will be

required to move the bias across the network. The normalizing is done by getting the minimum and maximum values in the data set and conditioning the data so that they lie between zero and one. This reduces the error during training. The data is normalized by using the following formula:

$$X_{norm} = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \tag{6.1}$$

Where X is the actual data, X_{min} is the smallest data in the data set (minimum data) and X_{max} is the largest data in the data set (maximum data). The normalised training data set was then used in the next stage to train the neural networks.

6.2 Creating the Neural Networks

The neural networks used are created using the MLP and RBF network architectures, which are very suitable for regression problems. In creating the neural networks, the number of inputs was assumed to be arbitrary. This will be optimized at a later stage of the project. This stage entailed getting an optimal architecture for the neural networks that will yield good predictions. Designing the neural networks thus involved choosing the right number of neurons and the appropriate network architecture which would yield the most accurate results. The number of neurons is then optimised by minimising an error function mapping the number of hidden neurons to the root mean square error obtained from the output related to the target output, for both the training data set and the validation data set. The root mean square (RMS) error is calculated by averaging the sum

of the square of the difference between the actual data output and the network output over the whole data set as per the formula below:

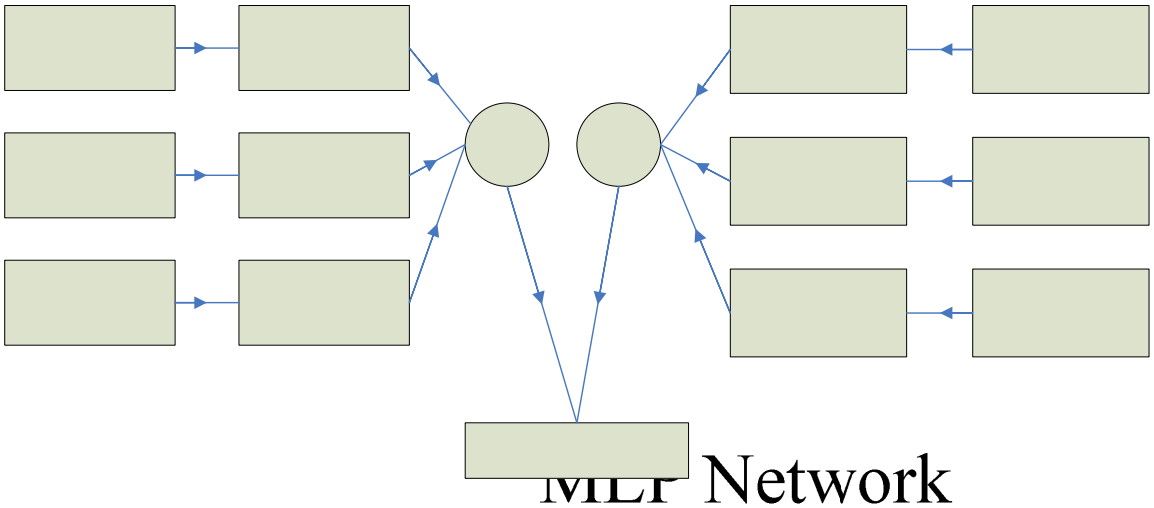
$$E_{RMS} = \frac{\sum_{i=1}^{n} (y_i - t_i)^2}{n} \tag{6.2}$$

Where y is the network output value, t is the *actual output* value and n is the number of data samples.

The hidden neurons were tested by incrementing the neurons from 5 hidden neurons to 40 hidden neurons in steps of 1 and training the network with the hidden neuron number. The root mean square error for each neuron number and each network architecture was then obtained. A committee of networks comprising of the average output of the MLP and the RBF networks was also obtained for each neuron number and the root mean square error of the output was also obtained. The output of the committee of networks was computed by averaging the outputs from the different network types as follows:

$$y = \frac{y_{MLP} + y_{RBF}}{2} \tag{6.3}$$

A more complex committee of networks [33, 34] comprising of six MLP and six RBF networks was also created. The network structure is as shown in Figure 4.

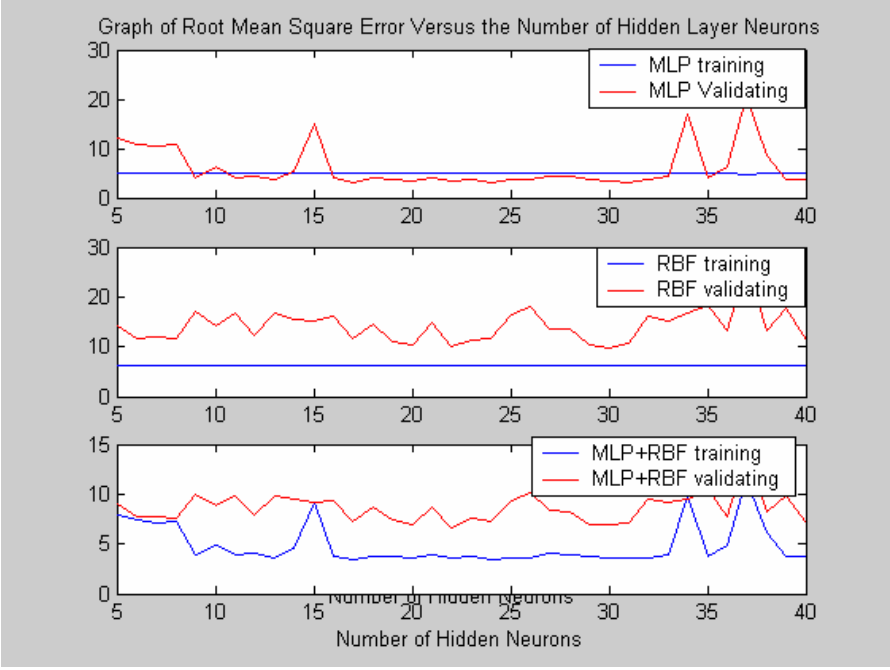


MLP Network MLP

1

Figure 4: Committee of networks for prediction

The networks are then validated using the validation data sets. The root mean square error obtained for eight inputs for the various number of neurons is shown in Figure 5.



rk MLP

rk MLP

Figure 5: Relationship between RMS error hidden layer neurons for the different architecture types

Upon getting the optimal network architecture which was the MLP and the optimal number of hidden units which was 12 hidden units, the design for the optimal input time-window was tackled. The RBF network architecture also gave accurate results with 11 hidden units being the optimal number of hidden neuron units.

6.3 Optimizing the Input Time-Window Using Polynomial Approximation

The next stage of the design was to optimally select an input time-window using polynomial approximation. The optimal network architecture, MLP and the optimal number of hidden neurons 12 was used to predict the input time-window. The RBF network was also verified by using the optimal number of hidden neurons 11. A set of networks was created with the number of days required to predict the output (average index of the next five days) ranging from 5 to 12. The output square error for each of the input days is then plotted and the optimal number of days required to predict the average of the next five days is obtained from the error function. Figure 6 shows the error plot obtained for various days:

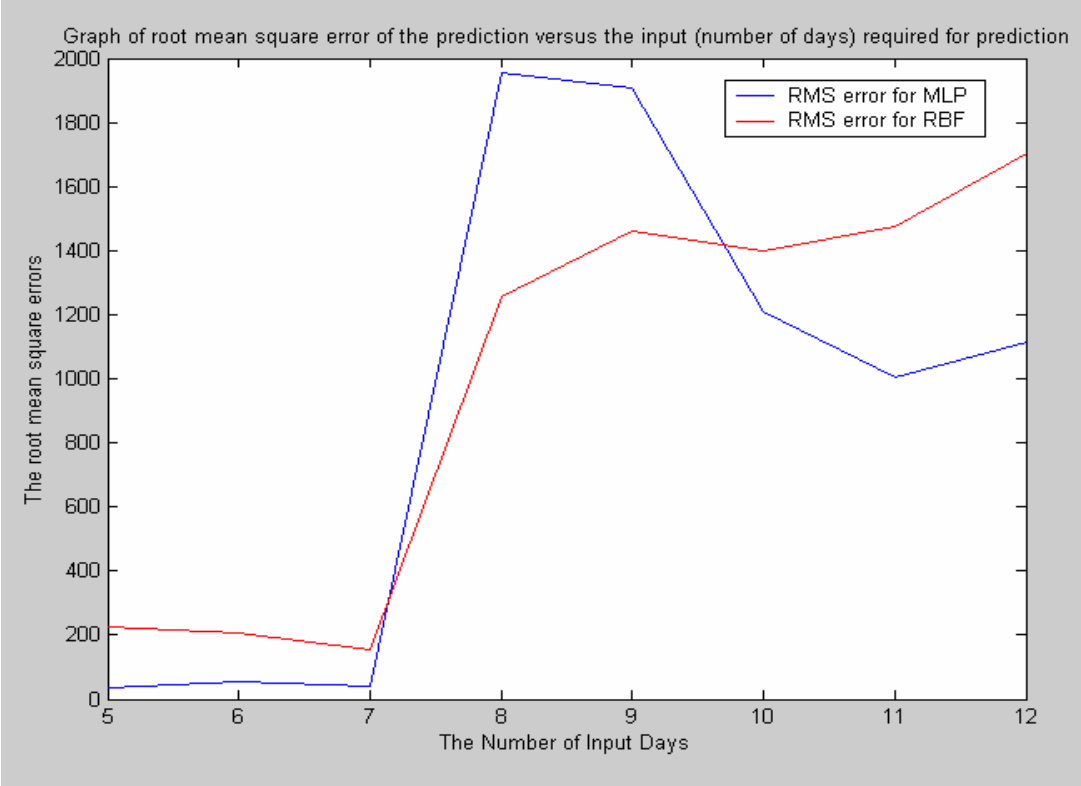


Figure 6: Relationship between the RMS error and the number of input days for MLP and RBF networks

As can be seen from Figure 6, the lowest error was obtained for the MLP and was 7 days. The RBF architecture also had an optimal input time-window of 7 days as can be seen in Figure 6 depicted by the red curve. The optimal input time-window from polynomial approximation is thus 7 days. Upon obtaining the optimal input time-window, the next stage of the design was then embarked on.

6.4 Optimizing the Input Time-Window by Reformulation of Bayesian Framework

This stage comprised the design of a neural network model to predict the average of the next five days by reformulating the Bayesian framework. The Markov Chain Monte Carlo (MCMC) method was used in this design stage. The Metropolis-Hastings algorithm (Section 4.4) [35] was employed whereby if the energy function is greater than a threshold then a new state is accepted else an old state is accepted. This threshold value is the error tolerance and is specified during the training stage. The neural network model as explained above in Chapter 2, Section 2.1 comprises of weights and biases. A new parameter, *a*, was introduced into the network such that the network now looked as follows:

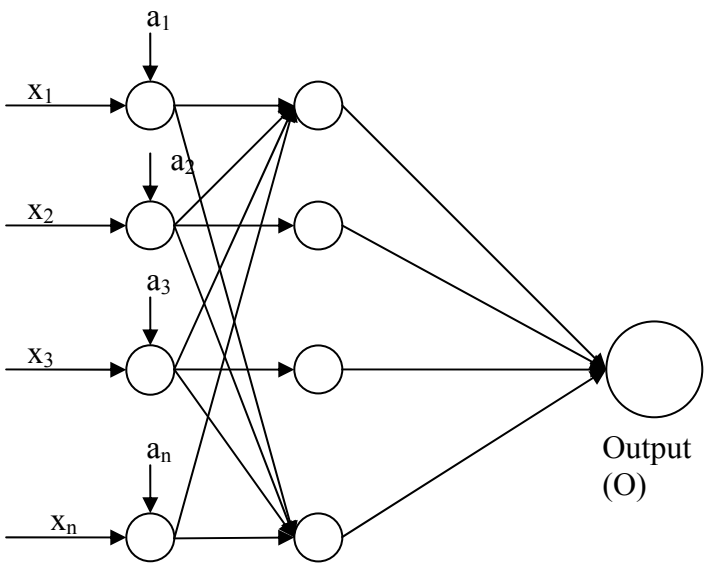


Figure 7: The reformed network with discrete parameter

This network parameter, *a*, is a discrete parameter which can have a value of 0 or 1. The

value of this parameter at the input neuron is dependent on the importance of the input at that neuron to the target output. A value of 1 means an input day is important for the prediction of the output and a value of 0 implies that an input day is not important towards the output prediction. The design involved the following steps:

6.4.1 Creating the Network Architecture

The first step involved creating the network architecture which had the binary (discrete) parameter. New network architecture types, discrete multi-layer perceptron (DMLP) and discrete radial basis function (DRBF) were created which will be initialised by stating the number of inputs, the number of hidden neurons in each layer, the number of outputs and the network function type. This multi-layer perceptron network contains 5 hidden parameters, *w1*, *b1* (hidden layer 1 weights and biases), *w2*, *b2* (hidden layer 2 weights and biases) and *a*, which is the input layer (discrete) parameter. The RBF network was also created and contains 5 hidden parameters, *c*, *wi* (first layer centres and widths), *w2*, *b2* (hidden layer 2 weights and biases) and *a*, the discrete parameter. In this step all the parameters will be initialised with random values, which will later be optimised.

6.4.2 Creation of the Discrete Feed-Forward Multi-Layer Perceptron

In this step the feed-forward discrete MLP and RBF were written whereby the MLP and RBF network formula (5.5), (2.8) and (2.9) were modified with *x* being replaced by x', (where $x' = a \bullet x$) and the equation now becomes:

$$y_k = f_{outer}\left(\sum_{j=1}^{M} w_{kj}^{(2)} f_{inner}\left(\sum_{i=1}^{d} w_{ji}^{(1)} x_i' + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \tag{6.4}$$

This function takes in an initialised discrete network together with an array of input data sample and gives an output data set.

6.4.3 Optimisation/Training Algorithm

In this step the optimisation/training algorithm was created. A discrete/continuous network optimisation algorithm was written using the Metropolis-Hastings algorithm [34]. In this algorithm, the initial values of the weight are taken as the initial state of the variables. The continuous and the discrete variables are separated. The old energy state is computed by summing and averaging the difference between the actual output and the network output squared. The variables are then updated. The continuous variables are updated by adding a random number unto them as per the Metropolis algorithm. The discrete parameter is updated by choosing a new random number between 0 and 1 and rounding off the number to the nearest integer thereby ensuring that the number is a zero or a one. The new energy is calculated using (6.4) and the new parameters. The probability difference between the two energies is then obtained from:

$$P = e^{(Eold - Enew)} \tag{6.5}$$

This probability is compared with a threshold value and if the probability is greater than the threshold value, the new state of the variables is kept as good samples. If the probability is, however, lower than the threshold value, then the old state is kept as the best sample. The number of samples used in training the network was 25000 for the MLP

network and 30000 for the RBF network. These numbers of samples was found to be the optimal values during validation. The variables were then obtained by taking the average of the samples stored for the continuous variables, and by rounding off the average for the discrete variable samples. The index averages are then predicted for the MLP network.

6.4.4 Prediction of outputs by RBF network

The RBF network was also used to predict the future average index and to select the input time-window optimally. The first stage in the RBF prediction was the selection of the input time-window. The algorithm in Section 6.4.3 was used to obtain the discrete parameter *a*, which depicts the input time-window. Upon obtaining the input-time window parameter (discrete parameter), the input was then pre-multiplied with this parameter and was used to train the second network which will do the prediction of the index average.

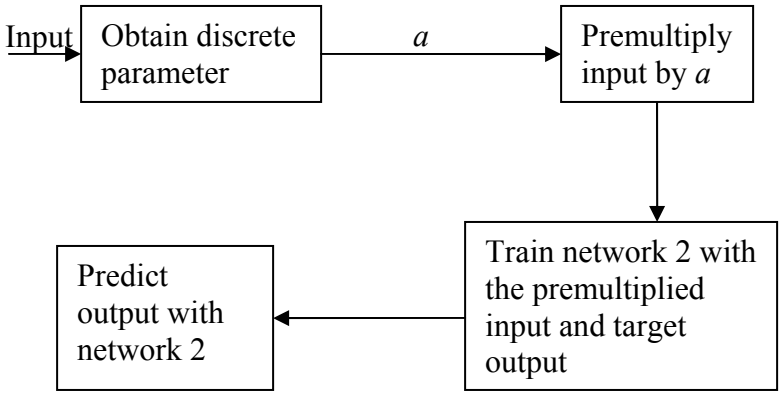


Figure 8: Diagrammatic representation of the RBF input time-window optimisation methodology

6.5 Simulation Results

Figures 9 and 10 were obtained for the training and validating data set. Table 4 shows the value for alpha for Figures 9 and 10 as well as the network architecture type. An input window of 13 days was used in the training and validation and Table 4 shows the optimal time-window. The discrete parameter has a value of one where the input is important for the prediction yielded and a zero where the input is not important for the prediction yielded. Table 4 also contains the fraction of the samples rejected during the optimisation process.

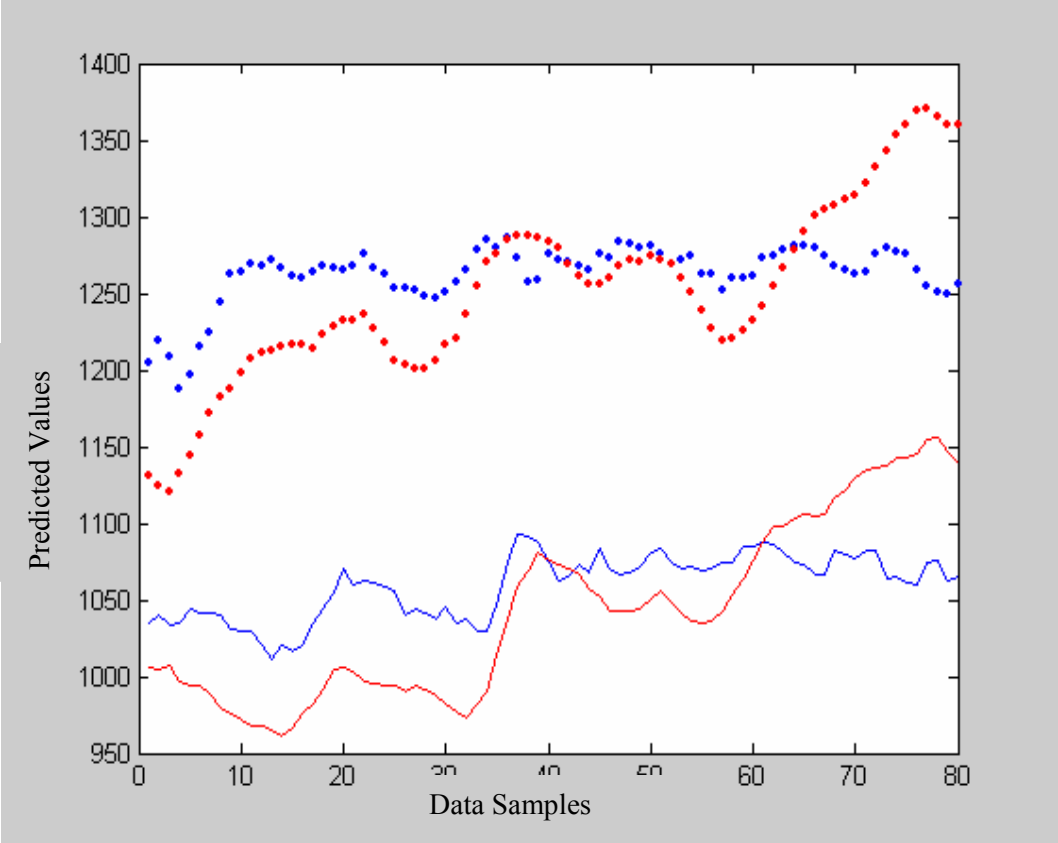


Figure 9: Predicted output by reformed MLP network

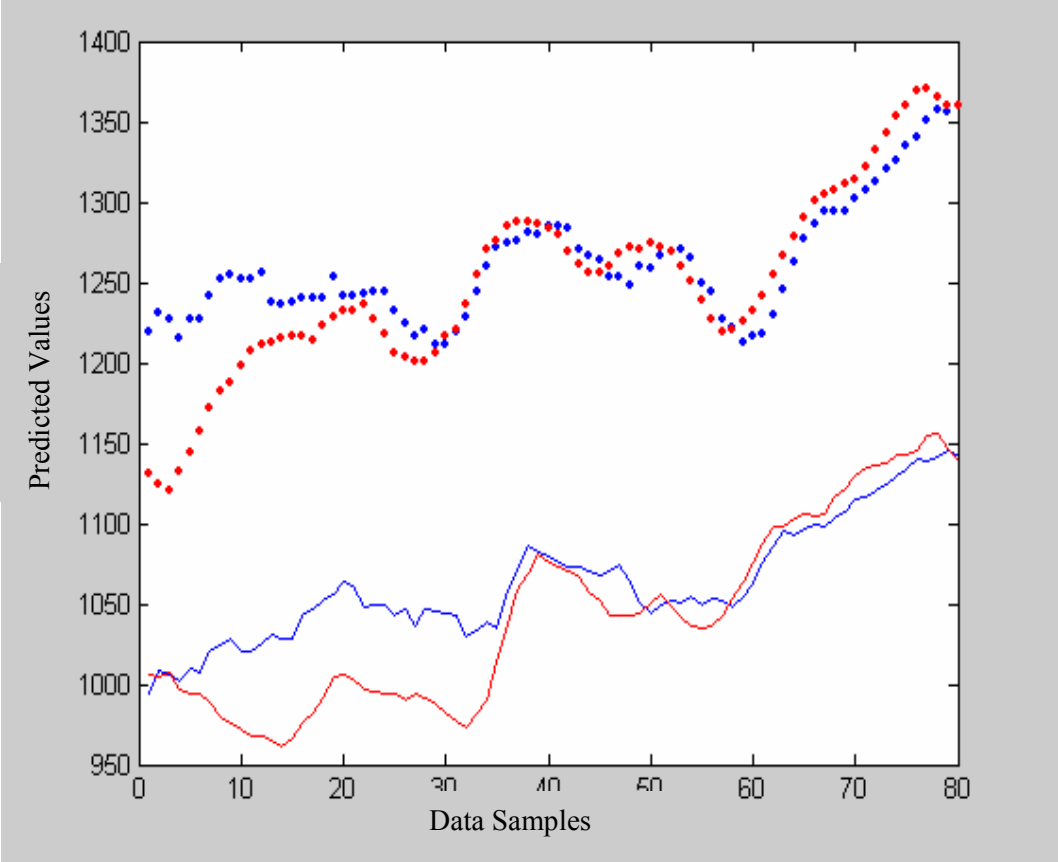


Figure 10: Predicted output by reformed RBF network

The red graphs are the actual output values meanwhile the blue graphs are the network predictions. The solid graph is the training set data meanwhile the dotted graph is the validation data set.

Table 4: The values of *a*, for the Figures 9 and 10

Fig	Network Type	Value Of *a*	Fraction of Rejected Samples
9	MLP	[0 1 1 0 1 0 0 1 1 0 0 1 1]	0.99968
10	RBF	[1 0 1 0 1 1 1 0 0 0 0 1 0]	0.8642

6.5.1 Testing and Comparison of Different Networks

The optimal design from each type of networks are tested and compared. The networks are trained and validated as explained above. The networks are then tested and compared using the testing data. The mean square errors of the outputs of the networks are shown in Table 5.

Table 5: The mean square errors of the outputs of the tested networks.

	MLP	**RBF**	**MLP+RBF**
Training			
RMS Error	0.0252	0.0548	0.0375
Validation			
RMS Error	0.0251	0.0466	0.0299
Testing			
RMS Error	0.2930	0.3750	0.3318

The optimal prediction for the reformulated MLP and RBF frameworks were tested with the test data set and yielded the Figures 11 and 12.

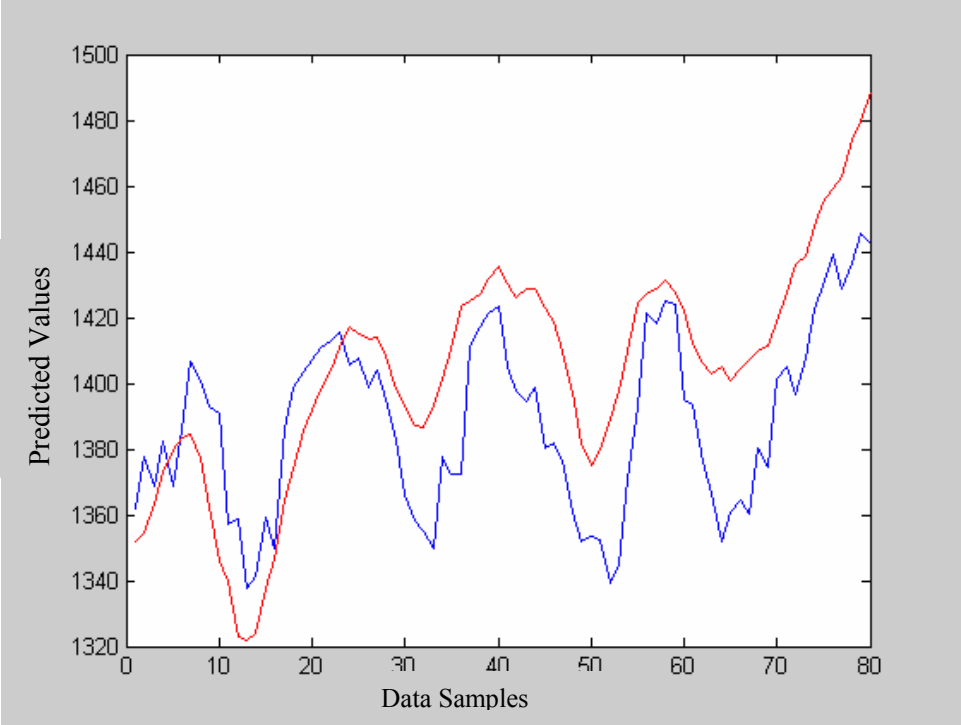


Figure 11: Predicted output of the reformulated MLP network for test data

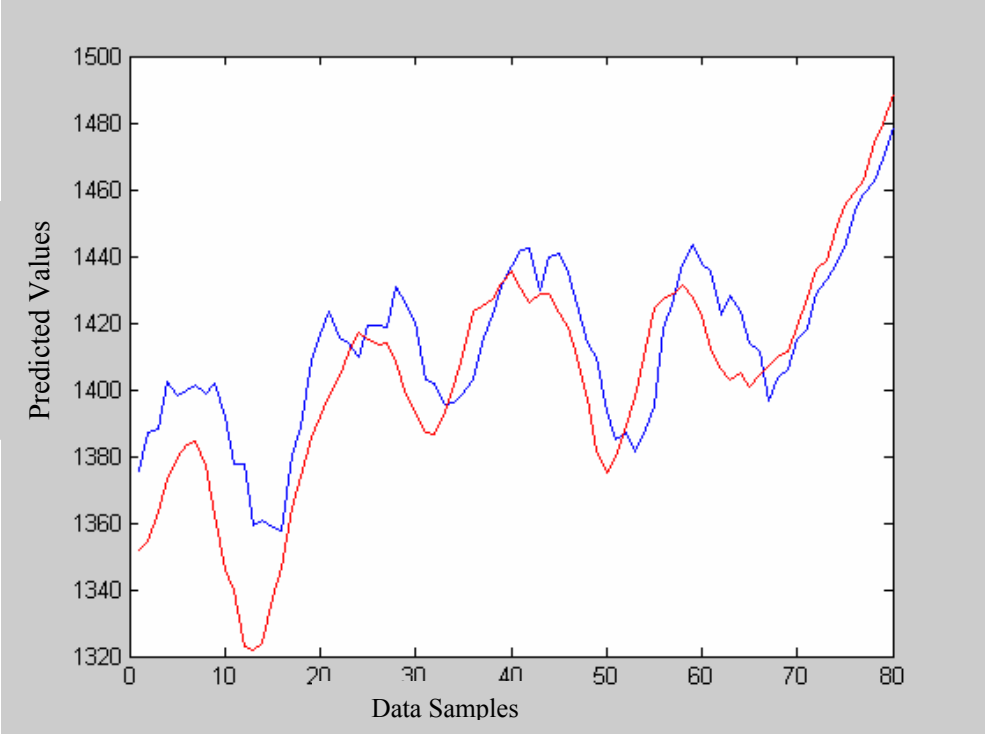


Figure 12: Predicted output of the reformulated RBF network for test data

Where the red is the actual values and the blue is the predicted value. Hence the optimal value is the Figure 10 entry in Table 4. The root mean square error for the normalised data was 0.2686 for the reformulated MLP and 0.0133 for the reformulated RBF. The index average calculated by the MLP network has the lowest mean square error for the polynomial approximation meanwhile the RBF network has the lowest mean square error for the reformulated Bayesian framework architecture. The mean square error is high when RBF network calculates the average indices for the polynomial approximation.

The outputs from the validation data for most cases have the least mean square error. The difference between the mean square errors of the outputs calculated using the training data and the mean square errors of the outputs calculated using the validation or the testing data is not significant. The difference is about 0.2 in the case of testing data and 0.01 for the validating data (normalised value). This indicates that problem of over-fitting is not present. The outputs from the networks using the testing data are shown in Figures 11, 12 and 13. The outputs from the network are compared with the actual index average. The output trends from each network confirm the mean square error calculations. The MLP network predicts the index average most accurately, meanwhile the RBF predicts the index average least accurately for the polynomial approximation. The networks predicts fairly accurately to the general trends of the target output.

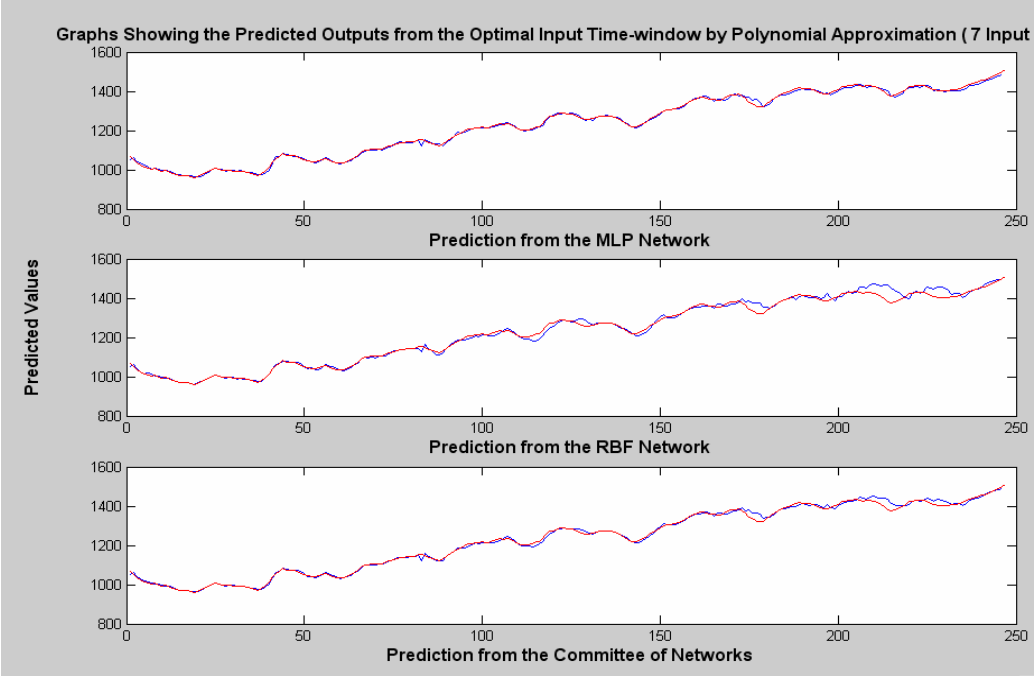


Figure 13: Networks predictions of the average indices using testing samples for polynomial approximation.

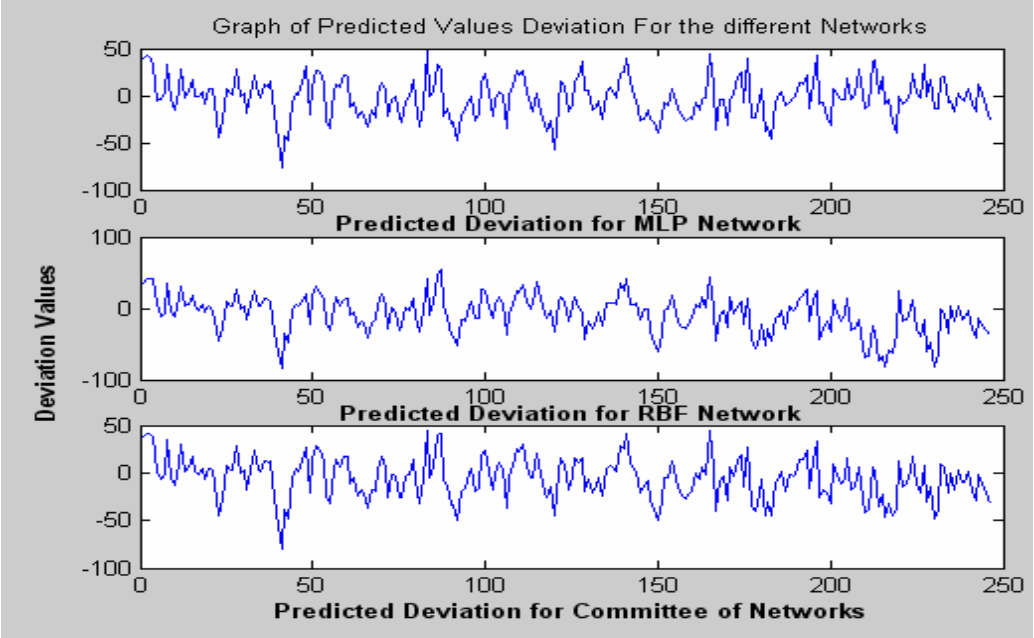


Figure 14: Networks predicted output standard deviations for the polynomial approximation

The deviations are small hence the index averages shown in Figure 13 can thus be considered as an accurate prediction.

The last network under investigation was the complex committee of networks comprised of six MLP and six RBF networks combined. The committee was found to work effectively and had a root mean square error of 0.0054 for the normalized test data and also a root mean square of 9.8497e-004 for the validating normalized data. The committee, which is as shown in Figure 4, yields Figure 15 for the training, validating and testing data:

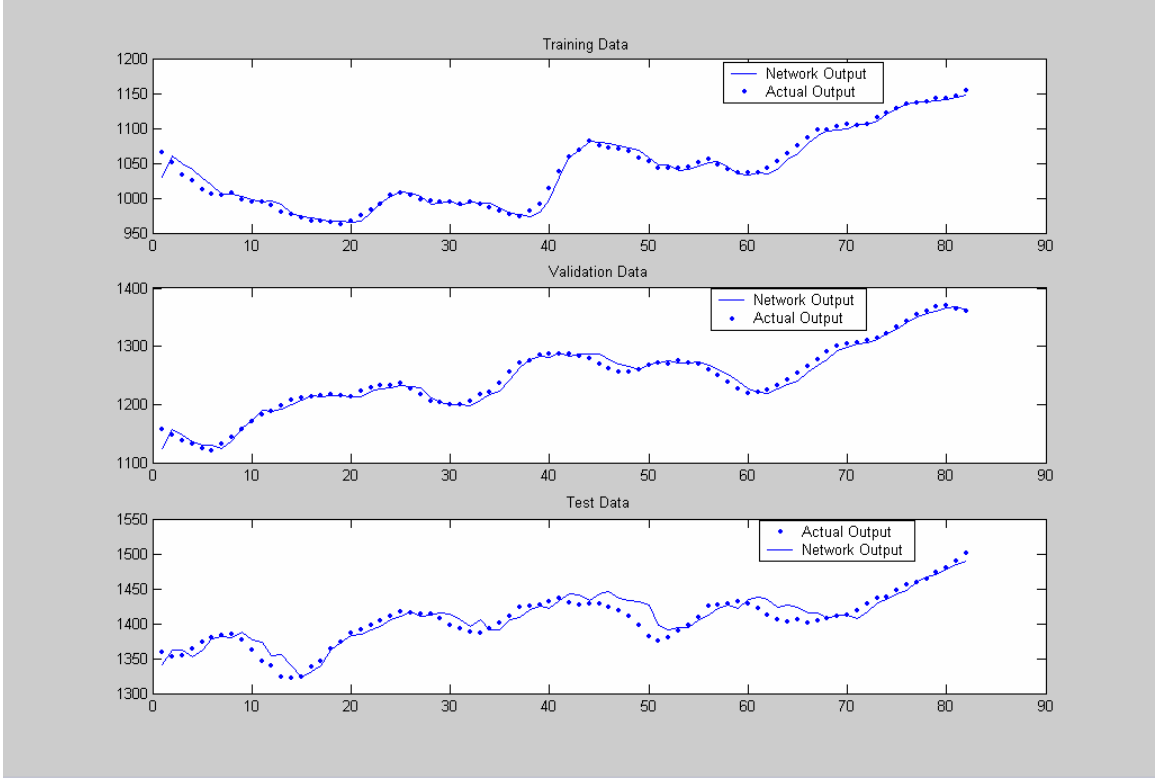


Figure 15: Committee of networks predicted output for the training, validating and testing data

The committee of networks thus gives the most efficient output with a very high accuracy level compared to all the other networks. The committee of networks is, however, just an extended implementation of the polynomial neural network.

6.3 Conclusion on Implementation

In this chapter, two methodologies were implemented using neural networks to optimally select the input time-window. The first methodology uses polynomial approximation and neural networks. The second methodology uses Bayesian analysis with the neural networks and Markov Chain Monte Carlo (MCMC) methods to optimally select this time window for the MLP and RBF networks. The sampling algorithm used for this methodology is the Metropolis-Hastings algorithm discussed in Section 4.4. The algorithm testing and validation has been done in this chapter too. A committee of networks was also investigated in this chapter and yielded the most accurate result compared to all the other networks. The next chapter would focus on the evaluation of the methodology (the network created by the methodology) on the basis of accuracy. The testing is done on NASDAQ data not used during training time, validation or testing time.

Chapter 7

Application of the Methodology Designed

In this chapter, the previous discussions are concluded. In Chapter 2, neural network was introduced. In Chapter 4, the Markov Chain Monte Carlo methods was described and in Chapter 5, the Bayesian approach was also described. In Chapter 6 two algorithms were implemented. The first algorithm exceeded the second in performance but the second methodology, however, gave the detailed representation of the important days required for the prediction. In this chapter the performance of the methodologies is further analyzed using unseen data which is obtained from the NASDAQ stock market. The chapter concludes by giving a thorough analysis of the results and also relating the methodology's performance to the existing methodologies for prediction. The background literature in Chapter 2, however, proved that there has not been methodologies implemented for the selection of an optimal input time-window, hence the methodologies implemented in the previous chapter as well as the network created in the

chapter are a novelty. The accuracy could be increased even further but such would be computationally expensive such as for the second methodology where the optimization of the parameters requires a lot of samples in order to converge as has been discussed in the Metropolis-Hastings algorithm section. The trade-off thus in the design was the convergence of the true result vis-à-vis the time required for the analysis. The results will, however, be analyzed in the last section of this chapter.

7.1 Analysis Data

The data to be used for the analysis is the NASDAQ data from 02 of January 2003 to 31 of December 2003. This data was pre-conditioned as per the algorithm such that the data can be normalized between 0 and 1 and the normalized data was then used to test the network for predictability. It should be noted that the NASDAQ all-share index (N100 index) is used as the analysis data set.

7.2 Performance Measurement

To determine the performance of the algorithms, two measures are used. The first measure is the root mean square error for the predicted average index and the second measure is the standard deviation of such predictions. The output of the algorithm (predicted value) is compared to the actual average index from the available data. The error can thus be obtained by just getting the squared difference between the actual value and the predicted value. Figure 16 shows the actual index for the NASDAQ data stated above:

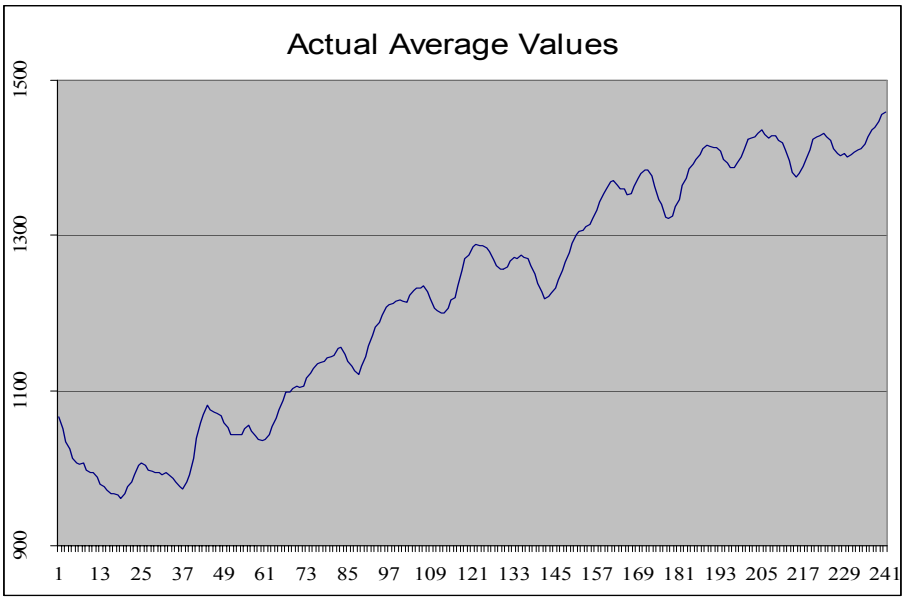


Figure 16: NASDAQ test data set used for the analysis of the methodologies

7.3 Methodology Analysis

The output from the polynomial approximation methodology was obtained and is shown in the Figure 17.

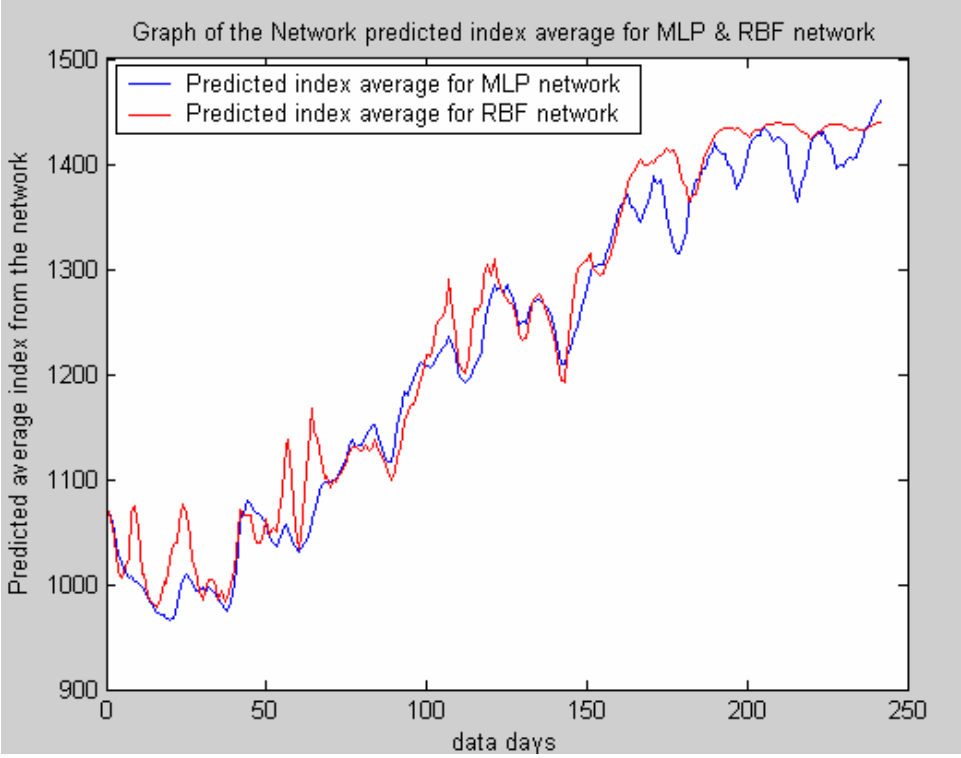


Figure 17: Graph of the network predicted index average for MLP & RBF network for polynomial approximation

In Figure 17 the root mean square errors were computed to be, 1.5978e-004 and 0.0040 for the MLP and the RBF networks, respectively. The error profile for the data was obtained and is shown in Figure 18.

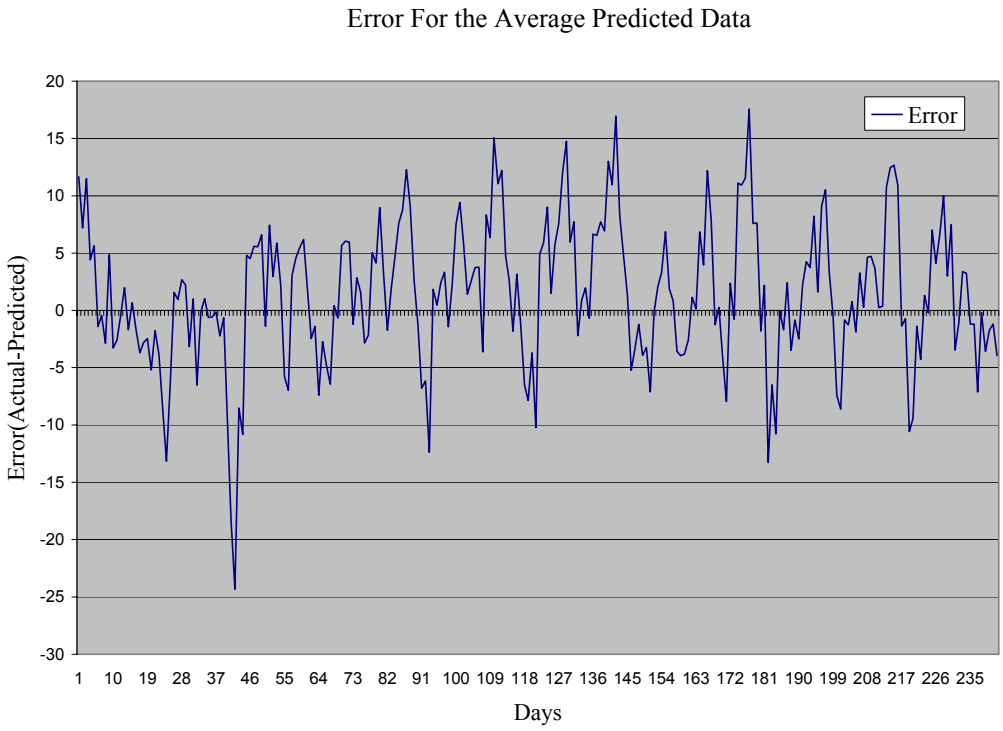


Figure 18: The error between the predicted and the actual values

The committee of networks was also investigated with this analysis data and yielded Figure 19.

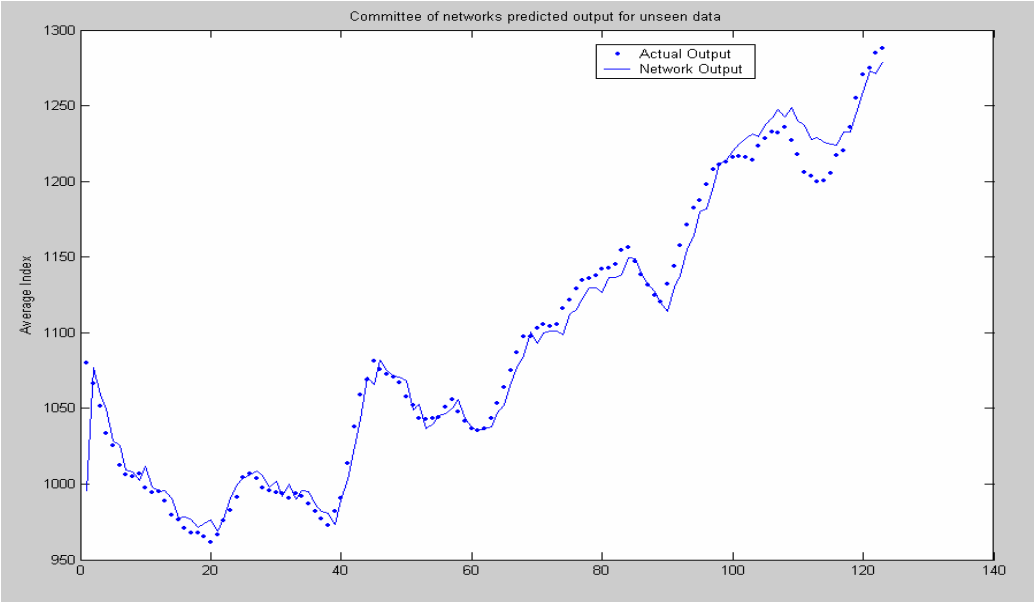


Figure 19: Committee of networks predicted output for unseen data

The root mean square error for the committee of networks prediction was 0.0016 for the normalized value, which was quite low. This network can thus predict the stock price average index accurately. Table 6 shows some of the data used in the analysis of the polynomial approximation together with the error obtained for this data.

Table 6: Table showing the actual values for the analysis and the predicted value from the network as well as the error of prediction

Actual	Predicted	Error	Actual	Predicted	Error	Actual	Predicted	Error
1080.056	1068.4	11.656	965.188	967.64	-2.452	1075.524	1071	4.524
1066.102	1058.9	7.202	961.458	966.64	-5.182	1072.4	1066.8	5.6
1051.286	1039.8	11.486	966.466	968.24	-1.774	1070.666	1065.1	5.566
1033.616	1029.2	4.416	975.456	979.27	-3.814	1067.004	1060.4	6.604
1025.432	1019.8	5.632	982.31	990.96	-8.65	1057.724	1059.1	-1.376
1012.376	1013.8	-1.424	991.25	1004.4	-13.15	1052.036	1044.6	7.436
1006.14	1006.6	-0.46	1004.054	1010.4	-6.346	1043.248	1040.3	2.948
1004.636	1007.5	-2.864	1006.574	1005	1.574	1042.68	1036.8	5.88
1006.646	1001.8	4.846	1003.448	1002.5	0.948	1043.106	1040.9	2.206
997.214	1000.5	-3.286	997.174	994.5	2.674	1043.904	1049.7	-5.796
994.588	997.16	-2.572	995.84	993.62	2.22	1050.828	1057.8	-6.972
994.72	995.16	-0.44	994.604	997.78	-3.176	1055.562	1052.5	3.062
988.784	986.8	1.984	993.88	992.89	0.99	1047.574	1043	4.574
979.266	980.95	-1.684	990.62	997.15	-6.53	1041.39	1035.9	5.49
976.274	975.62	0.654	993.764	993.89	-0.126	1036.478	1030.3	6.178

The Bayesian framework yielded the following discrete parameter a = [1 0 1 0 1 0 0 1 1 1 0 0 1].

7.3 Effect of the Simultaneous Use of Diverse Neural Networks on the Accuracy of Prediction

The committee of neural networks architecture was presented in Section 6.2 of this thesis. This network was found to have a higher level of accuracy compared to the MLP and RBF networks. This implies thus that the simultaneous use of diverse neural networks is beneficial to the overall system. A more complex combination of diverse neural networks will, therefore, yield even better results.

7.4 Conclusion

An analysis of the methodologies, presented in the previous chapters, has been done in this chapter. The NASDAQ all-share index from 01 January 2003 to 31 December 2003 was used as the test data. The MLP network yielded better results than the RBF network. The effect of the simultaneous use of diverse neural networks was also investigated in this chapter. This showed that the diverse neural network tracks the stock pattern for the average index more accurately than the MLP and RBF networks and also has a low rms error. To this effect, it can be concluded that the use of a committee of neural networks is beneficial and more efficient in the prediction of the stock prices and for trend tracking. This chapter illustrated the fact that a committee of networks increases the accuracy of the prediction; hence creating a more complex committee of networks will yield even more accurate results.

Chapter 8

Conclusion

Methods to optimally select the input time-window in the prediction of stocks were designed and implemented using polynomial approximation and also by reformulating the Bayesian framework to include a discrete parameter. This discrete parameter attaches a value to the importance of a particular day with respect to the output value whereby, a value of one implies importance and a zero implies unimportance, of the particular day towards the prediction of the output. The architectures tested were the multi-layer perceptron (MLP), radial basis function (RBF) and two integrated infrastructures comprising of the two networks, simultaneously, which were presented in Chapters 6 and 7. The methodology employed in designing was to first specify and process the data to be used for the design. At this stage, the data was normalized so that the values lie between zero and one, thereby reducing the effect of over-fitting (which leads to poor

generalization). This stage is known as preconditioning. Upon obtaining the preconditioned data, the polynomial approximation design was embarked on. The initial design phase of the polynomial approximation methodology involved optimizing the MLP and RBF networks by getting an optimal number of neurons in the hidden layer. The relationships between the number of neurons in the hidden layer and the mean square error of the outputs were then used to find the optimal parameter values for the MLP and RBF networks. It was found that 12 and 11 hidden neurons were optimal for the MLP and RBF networks, respectively. The performances of the two networks and the integrated network (committee of networks) were then compared using the testing data. The MLP network is best in predicting the index average meanwhile the RBF network is the worst in predicting the index average. The committee of networks yielded even better results than both the MLP and the RBF networks. This performance analysis done in Chapter 6, Section 6.2, was used to get the optimal architecture which was the MLP based on the root mean square (RMS) errors. Thus, the MLP is used for the first stage of the polynomial approximation to select an optimal input time-window design. Upon analyzing and optimizing the error function mapping the RMS error between the actual output and the predicted output, to the input days, an optimal time-window of seven days was obtained. The second design methodology involved the redesigning and reformulation of the Bayesian framework using the Metropolis-Hastings algorithm. The methodology employed here was firstly to create a discrete network for the MLP and RBF networks. This network contained a discrete parameter a. The parameter a, had a value of zero if a particular input day was not important towards the prediction, and the parameter had a value of one if the particular input day was important for the prediction.

This network architecture was presented in Chapter 6, Section 6.4. The discrete feedforward MLP and RBF networks were then created. Upon getting the discrete feedforward MLP and RBF networks, the Metropolis-Hastings algorithm was used to optimize the weights, biases and the discrete parameter in a discrete network as has been presented in previous sections. It was found that this reformed network yielded good result and could intuitively chose the number of days required to predict the index average. The average number of days required as the input time-window was found to be seven, which corresponded to the polynomial approximation as well. This methodology is beneficial since the number of input days required to predict the index average over the next five days do not have to be stated. A pool of data is rather entered into the network and the network is trained with this pool of the data. The network then recursively selects the input time-window. This methodology, however, had the limitation in that it was computationally expensive. The computational times spanned a period of more than 12 hours compared to the 3 minutes, which the polynomial approximation approach takes. A faster Markov Chain Monte Carlo (MCMC) algorithm will thus result in the increase of the efficiency of this methodology vis-à-vis computational expense. The trade-offs then had to be computational time or higher accuracy. The MLP architecture can thus be used to predict the index average over the next five days accurately and can also be used with polynomial approximation to select an optimal input time-window. A reformulated MLP and RBF with a discrete parameter can also be used to predict index average over the next five days without specifications of what previous days' data the network should use. The polynomial approximation is more efficient with respect to computational expense meanwhile the reformulated Bayesian neural networks are more efficient with respect to

accuracy. The effect of simultaneous use of diverse neural networks was also investigated and was found to yield more accurate results. The accuracy can thus be further increased by implementing a more complex committee of networks. Conclusively, neural networks using the MLP and RBF networks can be used with polynomial approximation to optimize the input time-window. A reformulated Bayesian MLP and RBF network can also be used to optimize this time-window. However, the MLP polynomial approximation is chosen as the optimal design since as the degree of accuracy is not much different from the Bayesian framework design, but the computational times is significantly different. Also, the use of simultaneous neural network engines as demonstrated in Chapters 6 and 7 results in the increase of accuracy and is thus encouraged.

Bibliography

[1] CNN Money: Markets and Stocks. www.money.com/markets. Last accessed: 2004 - 10 – 20.

[2] Stock Market News – World Markets – Stock Quotes, www.forbes.com/markets last accessed: 2004 - 10 – 20.

[3] NASDAQ monthly share prices, http://www.marketdata.nasdaq.com/mr4b.html, last accessed: 2004-05-30.

[4] Vehtari, A., and Lampinen, J. "Bayesian MLP neural networks for image analysis". *Journal of Pattern Recognition Letters*, 21, 2000, pp.1183-1191.

[5] Lachtermacher, G., and Fuller, J.D. "Backpropagation in time-series forecasting". *Journal of Forecasting*, 14, 1995, pp. 381-393.

[6] Bigus, J.P. "Data mining with neural networks: Solving business problems-from application development to decision support". *Neural Networks*, 15, 1996, pp. 909-925.

[7] Agrawal, D., and Schorling, C. "Market shares forecasting: An empirical comparison of artificial neural networks and multinomial logit model". *Journal of Retailing*, 72: (4), 1997, pp. 383-408.

[8] Wang, Y.-F. "Predicting stock price using fuzzy grey prediction system". *Journal of Expert Systems with Applications,* 22, 2002, pp. 33-39.

[9] Kuo, R. J., Wu, P., and Wang, C.P. "An intelligent sales forecasting system through integration of artificial neural network and fuzzy neural networks with fuzzy weight elimination". *Neural Networks*, 15, 2002, pp. 909-925.

[10] http://www.avaye.com/ai/nn/introduction/index.html. Last accessed: 2004-10-03.

[11] http://www.cs.stir.ac.uk/~lss/NNIntro/InvSlides.html#what. Last accessed: 2004-10-11.

[12] http://www.willamette.edu/~gorr/classes/cs449/. Last accessed: 2004-10-08.

[13] http://www.avaye.com/ai/nn/overfitting/index.html. Last accessed: 2004-10-03

[14] Haykin, S. *Neural Networks; A Comprehensive Foundation*. Macmillan, 1994, pp. 45 – 87.

[15] http://wwwinf.ethz.ch/~schraudo/NNcourse/linear2.html. Last accessed: 2004-10-10

[16] Bishop, C.M. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[17] Technical Analysis 101. "Moving Averages". http://www.fimi.com/studies/moving_averages.htm . Last accessed: 2004 – 10 – 20.

[18] "Forecasting Methods Used in ezForecaster". http://www.ezforecaster.com/fcmethod.htm . Last accessed: 2004 – 10 – 20.

[19] Efe, O.M., and Kaynak, O., "A novel optimization procedure for training of fuzzy inference systems by combining variable structure systems technique and Levenberg – Marquardt algorithm." *Fuzzy Sets and Systems*, 122, 2001, pp. 153 – 165.

[20] Ranganathan, A. "The Levenberg-Marquardt Algorithm", June 2004. http://www.cc.gatech.edu/~ananth/lmtut.pdf . Last accessed: 2004 – 10 – 21.

[21] G.E. Dallal. "Introduction to Simple Linear Regression". http://www.tufts.edu/~gdallal/slr.htm . Last accessed: 2004 – 10 – 20.

[22] Montgomery, D.C., Peck, E.A., and Vining, G.G. *Introduction to Linear Regression Analysis*. 3rd edition, New York: Wiley, 2001.

[23] Kleijnen, J.P.C. *Statistical Techniques in Simulation*, New York: Marcel Dekker, 1974.

[24] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H. and Teller, E. "Equations of state calculations by fast computing machines", *Journal of Chemical Physics*, 21, 1953, pp. 1087-1092.

[25] Tierney, L. "Markov chains for exploring posterior distributions", *Annals of Statistics*, 22: (4), 1994, pp. 1701-1728.

[26] Gelman, A., and Rubin, D.B. "Inference from iterative simulation using multiple sequences," *Journal of Statistical Science*, 7, 1992, pp. 457-511.

[27] Cowles, M.K., and Carlin, B.P., "Markov Chain Monte-Carlo convergence diagnostics: a comparative study," *Journal of the American Statistical Association*, 91, 1996, pp. 883-904.

[28] Murdoch, D.J., and Green, P.J. "Exact sampling from a continuous state space," *Scandinavian Journal of Statistics*, 25 :(3), 1998, pp. 483-502.

[29] Brooks, S.P., and Roberts, G.O. "Assessing convergence of Markov Chain Monte Carlo algorithms", *Journal of Statistics and Computing*, 8: (4), 1998, pp. 319-335.

[30] Chib, S., and Greenberg, E. "Understanding the Metropolis-Hastings algorithm", *The American Statistician* 49: (4), 1995, pp. 327-335.

[31] Lampinen J., and Vehtari A., "Bayesian approach for neural networks-review and case studies", *Neural Networks*, 14: (3), April 2001, pp. 7-24.

[32] Nabney, I.T. *NETLAB*: *Algorithms for Pattern Recognition*, Springer – Verlag, London, Great Britain, 2003, pp. 325 – 365.

[33] Sridhar, D.V., Bartlett, E.B., and Seagrave, R.C., "An information theoretic approach for combining neural network process models", *Neural Networks*, 12, 1999, pp. 915 – 926.

[34] Bakker B., and Heskes T., "Clustering ensembles of neural network models", *Neural Networks*, 16, 2003, pp. 261 – 269.

[35] Renshaw, E. "Metropolis-Hastings from a stochastic population dynamics perspective". *Journal of Computational Statistics and Data Analysis*, 45: (4), May 2004, pg 765-786.

Appendix A

The Implemented Code for Methodologies Developed

This appendix contains the different MATLAB codes implemented for the methodologies explained above in Chapters 6 and 7.

A.1 Matlab Code to Optimize the Network Architecture

```
clear all;
load dataf2
i=1;
n=3;
m=2;
e1=length(datanasdaq7);
f=floor(e1/3);
g=2*f;
e=3*f;

while(i<=f),
   p=i;
   q=1;
   n=p+6;
   for a=p:n,
      inpt(q,i)=datanasdaq(a,1);
      q=q+1;
   end

   outpt(1,i)=datanasdaq7(i,1);

      i=i+1;

   end
```

```
h=1;
while(i<=g),
p=i;
q=1;
n=p+6;

for a=p:n,
  inpv(q,h)=datanasdaq(a,1);
  q=q+1;
end
outpv(1,h)=datanasdaq7(i,1);
      h=h+1;
      i=i+1;
   end

k=1;

while(i<=e),
p=i;
q=1;
n=p+6;

for a=p:n,
  inpte(q,k)=datanasdaq(a,1);
  q=q+1;
end
outpte(1,k)=datanasdaq7(i,1);

      k=k+1;
      i=i+1;
      n=n+1;
end

  %Training data set

   day1dt=inpt(1,:);
   day2dt=inpt(2,:);
   day3dt=inpt(3,:);
   day4dt=inpt(4,:);
   day5dt=inpt(5,:);
   day6dt=inpt(6,:);
   day7dt=inpt(7,:);
   day8dt=outpt(1,:);

  %Validating data set
   day1dv=inpv(1,:);
   day2dv=inpv(2,:);
   day3dv=inpv(3,:);
   day4dv=inpv(4,:);
   day5dv=inpv(5,:);
```

```
day6dv=inpv(6,:);
day7dv=inpv(7,:);
day8dv=outpv(1,:);


%Testing data set
day1dte=inpte(1,:);
day2dte=inpte(2,:);
day3dte=inpte(3,:);
day4dte=inpte(4,:);
day5dte=inpte(5,:);
day6dte=inpte(6,:);
day7dte=inpte(7,:);
day8dte=outpte(1,:);


%normalise the parameters

%Training data
minday1t=min(day1dt);
maxday1t=max(day1dt);
minday2t=min(day2dt);
maxday2t=max(day2dt);
minday3t=min(day3dt);
maxday3t=max(day3dt);
minday4t=min(day4dt);
maxday4t=max(day4dt);
minday5t=min(day5dt);
maxday5t=max(day5dt);
minday6t=min(day6dt);
maxday6t=max(day6dt);
minday7t=min(day7dt);
maxday7t=max(day7dt);
minday8t=min(day8dt);
maxday8t=max(day8dt);

day1t=(day1dt-minday1t)/(maxday1t-minday1t);
day2t=(day2dt-minday2t)/(maxday2t-minday2t);
day3t=(day3dt-minday3t)/(maxday3t-minday3t);
day4t=(day4dt-minday4t)/(maxday4t-minday4t);
day5t=(day5dt-minday5t)/(maxday5t-minday5t);
day6t=(day6dt-minday6t)/(maxday6t-minday6t);
day7t=(day7dt-minday7t)/(maxday7t-minday7t);
day8t=(day8dt-minday8t)/(maxday8t-minday8t);


%Validation data
minday1v=min(day1dv);
maxday1v=max(day1dv);
minday2v=min(day2dv);
maxday2v=max(day2dv);
```

```
minday3v=min(day3dv);
maxday3v=max(day3dv);
minday4v=min(day4dv);
maxday4v=max(day4dv);
minday5v=min(day5dv);
maxday5v=max(day5dv);
minday6v=min(day6dv);
maxday6v=max(day6dv);
minday7v=min(day7dv);
maxday7v=max(day7dv);
minday8v=min(day8dv);
maxday8v=max(day8dv);

day1v=(day1dv-minday1v)/(maxday1v-minday1v);
day2v=(day2dv-minday2v)/(maxday2v-minday2v);
day3v=(day3dv-minday3v)/(maxday3v-minday3v);
day4v=(day4dv-minday4v)/(maxday4v-minday4v);
day5v=(day5dv-minday5v)/(maxday5v-minday5v);
day6v=(day6dv-minday6v)/(maxday6v-minday6v);
day7v=(day7dv-minday7v)/(maxday7v-minday7v);
day8v=(day8dv-minday8v)/(maxday8v-minday8v);


%Testing data set

minday1te=min(day1dte);
maxday1te=max(day1dte);
minday2te=min(day2dte);
maxday2te=max(day2dte);
minday3te=min(day3dte);
maxday3te=max(day3dte);
minday4te=min(day4dte);
maxday4te=max(day4dte);
minday5te=min(day5dte);
maxday5te=max(day5dte);
minday6te=min(day6dte);
maxday6te=max(day6dte);
minday7te=min(day7dte);
maxday7te=max(day7dte);
minday8te=min(day8dte);
maxday8te=max(day8dte);

day1te=(day1dte-minday1te)/(maxday1te-minday1te);
day2te=(day2dte-minday2te)/(maxday2te-minday2te);
day3te=(day3dte-minday3te)/(maxday3te-minday3te);
day4te=(day4dte-minday4te)/(maxday4te-minday4te);
day5te=(day5dte-minday5te)/(maxday5te-minday5te);
day6te=(day6dte-minday6te)/(maxday6te-minday6te);
day7te=(day7dte-minday7te)/(maxday7te-minday7te);
day8te=(day8dte-minday8te)/(maxday8te-minday8te);
```

```
a=length(day8t);
q=1;
for t=1:a,
if t==1
day8det(t,1)=0;
day8dev(t,1)=0;
day8dete(t,1)=0;
end
if t>1,
   day8det(t,1)=day8t(q);
   day8dev(t,1)=day8v(q);
   day8dete(t,1)=day8te(q);
   q=q+1;
end
end
   for s=1:f,

   invart(s,1)=day1t(s);
   invart(s,2)=day2t(s);
   invart(s,3)=day3t(s);
   invart(s,4)=day4t(s);
   invart(s,5)=day5t(s);
   invart(s,6)=day6t(s);
   invart(s,7)=day7t(s);
   invart(s,8)=day8det(s);
   outvart(s,1)=day8t(s);

end

j=1;
f1=f+1;

for s=f1:g,

   invarv(j,1)=day1v(j);
   invarv(j,2)=day2v(j);
   invarv(j,3)=day3v(j);
   invarv(j,4)=day4v(j);
   invarv(j,5)=day5v(j);
   invarv(j,6)=day6v(j);
   invarv(j,7)=day7v(j);
   invarv(j,8)=day8dev(j);
   outvarv(j,1)=day8v(j);
   j=j+1;

end

k=1;
g1=g+1;
for s=g1:e,
   invarte(k,1)=day1te(k);
```

```
   invarte(k,2)=day2te(k);
   invarte(k,3)=day3te(k);
   invarte(k,4)=day4te(k);
   invarte(k,5)=day5te(k);
   invarte(k,6)=day6te(k);
   invarte(k,7)=day7te(k);
   invarte(k,8)=day8dete(k);
   outvarte(k,1)=day8te(k);
   k=k+1;
end

for u=5:40,

%Initialising neural network parameters
nin=8;%Number of input units is 8
nhiddenm=u;%u hidden layers
nhiddenr=u;%u hidden layers used
nout=1;%One output value
alpha=0.01;

net1=mlp(nin, nhiddenm, nout, 'linear', alpha); %Linear MLP algorithm chosen
net2=rbf(nin,nhiddenr,nout,'gaussian','linear',alpha);%RBF network architecture

options = zeros(1,18);
options(1) = 1;
options(14) = 1000;%1000 iterations found to be accurate enough
[net1, options, varargout] = netopt(net1, options, invart, outvart, 'scg');

%RBF Optimisation
[net2, options, varargout] = netopt(net2, options, invart, outvart, 'scg');
yout1= mlpfwd(net1, invart);
yout2= mlpfwd(net1,invarv);

yout1r=rbffwd(net2,invart);
yout2r=rbffwd(net2,invarv);


Day8t1=(yout1(:,1)*(maxday8t-minday8t))+minday8t;
Day8t2=(yout2(:,1)*(maxday8v-minday8v))+minday8v;

Day8t1r=(yout1r(:,1)*(maxday8t-minday8t))+minday8t;
Day8t2r=(yout2r(:,1)*(maxday8v-minday8v))+minday8v;

Day8av1=(Day8t1+Day8t1r)/2;
Day8av2=(Day8t2+Day8t2r)/2;

m=length(Day8t1);
sum1=0;
sum2=0;
sum1av=0;
sum2av=0;
```

```
sum1r=0;
sum2r=0;

for n=1:m,
   sum1=sum1+((Day8t1(n)-day8dt(n))^2);
   sum2=sum2+((Day8t2(n)-day8dv(n))^2);
   sum1r=sum1r+((Day8t1r(n)-day8dt(n))^2);
   sum2r=sum2r+((Day8t2r(n)-day8dv(n))^2);
   sum1av=sum1av+((Day8av1(n)-day8dt(n))^2);
   sum2av=sum2av+((Day8av2(n)-day8dv(n))^2);


end

sum1=(sum1/m)^(1/2);
sum2=(sum2/m)^(1/2);
sum1av=(sum1av/m)^(1/2);
sum2av=(sum2av/m)^(1/2);
sum1r=(sum1r/m)^(1/2);
sum2r=(sum2r/m)^(1/2);
p=u-4;
eror1t(p)=sum1;
eror2t(p)=sum1r;
eror1av(p)=sum1av;
eror1v(p)=sum2;
eror2v(p)=sum2r;
eror2av(p)=sum2av;
end

x=[5:1:40];
plot(x,eror1t,'b')legend('error for training data-MLP')
hold on
plot(x,eror2t,'r')legend('error for training data-RBF')
plot(x,eror1v,'b.')legend('error for validating data-MLP')
plot(x,eror2v,'r.')legend('error for validating data-RBF')
plot(x,eror1av,'k')legend('error for training data-MLP&RBF')
plot(x,eror2av,'k.')legend('error for validating data-MLP&RBF')
```

A.2 Matlab Code for the Polynomial Approximation Optimisation of the Input Time-Window

The function input number below gets the RMS error for using the various number of days where ***mastdaysn*** represents a function which uses the number of days specified by the function to compute the RMS error with ***n*** representing the number of days. The generic function is as shown in A.2.2 below.

A.2.1 Input Number:

```
function inputnum=erran()
clear all;
for i=5:12,
   if(i==5)
   optstring=['mastdays5'];
end
if(i==6)
   optstring=['mastdays6'];
end
if(i==7)
   optstring=['mastdays7'];
end
if(i==8)
   optstring=['mastdays8'];
end
if(i==9)
   optstring=['mastdays9'];
end
if(i==10)
   optstring=['mastdays10'];
end
if(i==11)
   optstring=['mastdays11'];
end
if(i==12)
   optstring=['mastdays12'];
end
   [error1(i-4),error2(i-4)]=feval(optstring);

end

x=[5:1:12];
save errorf error1 error2
```

A.2.2 *Mastdaysn*

```
function [eror1,eror2]=mastdaysn()

clear all;
load dataf2
i=1;
e1=length(datanasdaqn);
f=floor(e1/3);
g=2*f;
e=3*f;

while(i<=f),
```

```
    p=i;
    q=1;
    t=p+n-1;
    for a=p:t,
        inpt(q,i)=datanasdaq(a,1);
        q=q+1;
    end

    outpt(1,i)=datanasdaqn(i,1);

        i=i+1;

    end

h=1;
while(i<=g),
p=i;
q=1;
t=p+n-1;

for a=p:t,
   inpv(q,h)=datanasdaq(a,1);
   q=q+1;
end
outpv(1,h)=datanasdaq6(i,1);
        h=h+1;
        i=i+1;
    end

k=1;

while(i<=e),
p=i;
q=1;
t=p+n-1;

for a=p:t,
   inpte(q,k)=datanasdaq(a,1);
   q=q+1;
end
outpte(1,k)=datanasdaq6(i,1);

        k=k+1;
        i=i+1;
    end

   %Training data set

    day1dt=inpt(1,:);
    day2dt=inpt(2,:);
    day3dt=inpt(3,:);
```

```
day4dt=inpt(4,:);
day5dt=inpt(5,:);
.       .   .
.       .   .
.       .   .
dayndt=inpt(n,:);
day(n+1)dt=outpt(1,:);

%Validating data set
day1dv=inpv(1,:);
day2dv=inpv(2,:);
day3dv=inpv(3,:);
day4dv=inpv(4,:);
day5dv=inpv(5,:);
.       .   .
.       .   .
.       .   .

dayndv=inpv(n,:);
day(n+1)dv=outpv(1,:);


%Testing data set
day1dte=inpte(1,:);
day2dte=inpte(2,:);
day3dte=inpte(3,:);
day4dte=inpte(4,:);
day5dte=inpte(5,:);
.       .   .
.       .   .
.       .   .
dayndte=inpte(n,:);
day(n+1)dte=outpte(1,:);


%normalise the parameters

%Training data
minday1t=min(day1dt);
maxday1t=max(day1dt);
minday2t=min(day2dt);
maxday2t=max(day2dt);
minday3t=min(day3dt);
maxday3t=max(day3dt);
minday4t=min(day4dt);
maxday4t=max(day4dt);
minday5t=min(day5dt);
maxday5t=max(day5dt);
.       .   .
.       .   .
.       .   .
```

```
   mindaynt=min(dayndt);
   maxdaynt=max(dayndt);
   minday(n+1)t=min(day(n+1)dt);
   maxday(n+1)t=max(day(n+1)dt);

   day1t=(day1dt-minday1t)/(maxday1t-minday1t);
   day2t=(day2dt-minday2t)/(maxday2t-minday2t);
   day3t=(day3dt-minday3t)/(maxday3t-minday3t);
   day4t=(day4dt-minday4t)/(maxday4t-minday4t);
   day5t=(day5dt-minday5t)/(maxday5t-minday5t);
   .     .          .
   .     .          .
   .     .          .
   daynt=(dayndt-mindaynt)/(maxdaynt-mindaynt);
   day(n+1)t=(day(n+1)dt-minday(n+1)t)/(maxday(n+1)t-minday(n+1)t);


   a=length(day(n+1)t);
q=1;
for t=1:a,
if t==1
day(n+1)det(t,1)=0;
end
if t>1,
   day(n+1)det(t,1)=day7t(q);
   q=q+1;
end
end
   for s=1:f,

   invart(s,1)=day1t(s);
   invart(s,2)=day2t(s);
   invart(s,3)=day3t(s);
   invart(s,4)=day4t(s);
   invart(s,5)=day5t(s);
   .     .    .
   .     .    .
   .     .    .
   invart(s,n)=daynt(s);
   invart(s,n+1)=day(n+1)det(s);
   outvart(s,1)=day(n+1)t(s);

end

%Initialising neural network parameters
nin=n+1;%Number of input units is (n+1)
nhiddenm=12;%12 hidden layers used (optimum value)
nhiddenr=11;%11 hidden layers used (optimum value)
nout=1;%One output value
alpha=0.01;
```

```
net1=mlp(nin, nhiddenm, nout, 'linear', alpha); %Linear MLP algorithm chosen
net2=rbf(nin,nhiddenr,nout,'gaussian','linear',alpha);%RBF network architecture

options = zeros(1,18);
options(1) = 1;
options(14) = 1000;%1000 iterations found to be accurate enough
[net1, options, varargout] = netopt(net1, options, invart, outvart, 'scg');

%RBF Optimisation
[net2, options, varargout] = netopt(net2, options, invart, outvart, 'scg');
yout1= mlpfwd(net1, invart);

yout1r=rbffwd(net2,invart);


Day(n+1)t1=(yout1(:,1)*(maxday7t-minday7t))+minday7t;

Day(n+1)t1r=(yout1r(:,1)*(maxday7t-minday7t))+minday7t;

Day7av1=(Day7t1+Day7t1r)/2;

m=length(Day(n+1)t1);
sum1=0;
sum1av=0;
sum1r=0;

for t=1:m,
   sum1=sum1+((Day(n+1)t1(t)-day(n+1)dt(t))^2);
   sum1r=sum1r+((Day(n+1)t1r(t)-day(n+1)dt(t))^2);
   sum1av=sum1av+((Day(n+1)av1(n)-day(n+1)dt(n))^2);

end

sum1=sum1/m;
sum1av=sum1av/m;
sum1r=sum1r/m;


eror1=sum1;
eror2=sum1r;
```

A.3 The Matlab Codes Created for the Second Methodology

A.3.1 The discrete MLP and RBF Networks

a) MLP

```
function netbin=mlpbin(nin,nhidden,nout,outfunc,prior,beta)

netbin.type='mlpbin';
netbin.nin=nin;
netbin.nhidden=nhidden;
netbin.nout=nout;

netbin.nwts=nin + (nin+1)*nhidden + (nhidden+1)*nout;

outfns={'linear','logistic','softmax'};

if sum(strcmp(outfunc,outfns))==0
   error('undefined activation function.Exiting.');
else
   netbin.outfn=outfunc;
end

if nargin>4
   if isstruct(prior)
      netbin.alpha=prior.alpha;
      netbin.index=prior.index;

   else if size(prior)==[1 1]
         netbin.alpha=prior;
      else
         error('prior must be a scalar or a structure');
      end
   end
end

   netbin.a=round(rand(1,nin));
   netbin.w1=randn(nin,nhidden)/sqrt(nin+1);
   netbin.b1=randn(1,nhidden)/sqrt(nin+1);
   netbin.w2=randn(nhidden,nout)/sqrt(nhidden+1);
   netbin.b2=randn(1,nout)/sqrt(nhidden+1);

   if nargin==6
      netbin.beta=beta;
   end
```

b) RBF

```
function netbin = rbfbin(nin, nhidden, nout, rbfunc, outfunc, prior, beta)

netbin.type = 'rbfbin';
netbin.nin = nin;
netbin.nhidden = nhidden;
netbin.nout = nout;

% Check that function is an allowed type
actfns = {'gaussian', 'tps', 'r4logr'};
outfns = {'linear', 'neuroscale'};
if (strcmp(rbfunc, actfns)) == 0
  error('Undefined activation function.')
else
  netbin.actfn = rbfunc;
end
if nargin <= 4
   netbin.outfn = outfns{1};
elseif (strcmp(outfunc, outfns) == 0)
   error('Undefined output function.')
else
   netbin.outfn = outfunc;
 end

% Assume each function has a centre and a single width parameter, and that
% hidden layer to output weights include a bias.  Only the Gaussian function
% requires a width
netbin.nwts=nin*(1+nhidden) + (nhidden + 1)*nout;

if strcmp(rbfunc, 'gaussian')
  % Extra weights for width parameters
  netbin.nwts = netbin.nwts + nhidden;
end

if nargin > 5
  if isstruct(prior)
    netbin.alpha = prior.alpha;
    netbin.index = prior.index;
  elseif size(prior) == [1 1]
    netbin.alpha = prior;
  else
    error('prior must be a scalar or a structure');
  end
  if nargin > 6
    netbin.beta = beta;
  end
end
a=round(rand(1,nin));
w1 = randn(1, (netbin.nwts-nin));
w=[a,w1];
```

```
netbin = rbfunpakbin(netbin, w);

% Make widths equal to one
if strcmp(rbfunc, 'gaussian')
  netbin.wi = ones(1, nhidden);
end

if strcmp(netbin.outfn, 'neuroscale')
  netbin.mask = rbfpriorbin(rbfunc, nin, nhidden, nout);
end
```

A.3.2 Optimisation Algorithms for the MLP and RBF Networks

```
function [x1,options,samples, energies, diagn] = metropbin(f, x1, options, gradf, varargin)


if nargin <= 2
  if ~strcmp(f, 'state')
    error('Unknown argument to metrop');
  end
  switch nargin
    case 1
      % Return state of sampler
      samples = get_state(f);         % Function defined in this module
      return;
    case 2
      % Set the state of the sampler
      set_state(f, x1);               % Function defined in this module
      return;
  end
end

display = options(1);
if options(14) > 0
  nsamples = options(14);
else
  nsamples = 100;
end
if options(15) >= 0
  nomit = options(15);
else
  nomit = 0;
end
if options(18) > 0.0
  std_dev = sqrt(options(18));
else
  std_dev = 1.0;   % default
end

netbin=varargin{1};
```

```
netbin=netbinunpak(netbin,x1);

b=netbin.a;
x2=[netbin.w1(:)',netbin.b1,netbin.w2(:)',netbin.b2];
nparams=length(x2);
nparams1=length(b);

f=fcnchk(f,length(varargin));

samples1=zeros(nsamples,nparams);
samples2=zeros(nsamples,nparams1);

if nargout>=2
   en_save=1;
   energies=zeros(nsamples,1);
else
   en_save=0;
end

if nargout>=3
   diagnostics=1;
   diagn_pos=zeros(nsamples,nparams);
   diagn_pos1=zeros(nsamples,nparams1);
   diagn_acc=zeros(nsamples,1);
else
   diagnostics=0;
end

x1=[netbin.a,netbin.w1(:)',netbin.b1,netbin.w2(:)',netbin.b2];
n=-nomit+1;
Eold=feval(f,x1,varargin{:});
nreject = 0;
t=0;

while n<=nsamples
   xold=x2;
   aold=b;

   x2=xold+randn(1,nparams)*std_dev;
  b=round(rand(1,nparams1));


   x1=[b,x2];
   Enew=feval(f,x1,varargin{:});

   p=exp(Eold-Enew);

   if (diagnostics & n>0)
      diagn_pos(n,:)=x2;
      diagn_pos1(n,:)=b;
```

```
      diagn_acc(n,:)=p;
   end

   if (display>1)
      fprintf(1,'New position is \n');
      disp(x1);
   end

   if p>rand(1)
      Eold=Enew;
      if (display>0)
         fprintf(1,'Finished step %4d Threshold: %g \n',n,p);
      end

   else
      if n>0
         nreject=nreject+1;
      end

      x2=xold;
      b2=aold;
      if (display>0)
         fprintf(1,'Sample rejected %4d. Threshold: %g \n',n,p);
      end
   end

   if n>0
      samples1(n,:)=x2;
      samples2(n,:)=b;
      if en_save
         energies(n)=Eold;
      end
   end
   n=n+1;
end
if (display>0)
   fprintf(1,'\n Fraction of samples rejected: %g \n',nreject/nsamples);
end
if diagnostics
   diagn.pos=diagn_pos;
   diagn.acc=diagn_acc;
   diagn.pos1=diagn_pos1;
end

options(8) = Eold;

x3=sum(samples1)/(n-1);
b2=round(sum(samples2)/(n-1));

x1=[b,x3];
```

```
% Return complete state of the sampler.
function state = get_state(f)

state.randstate = rand('state');
state.randnstate = randn('state');
return

% Set state of sampler, either from full state, or with an integer
function set_state(f, x)

if isnumeric(x)

 rand('state', x);
  randn('state', x);
else
  if ~isstruct(x)
    error('Second argument to metrop must be number or state structure');
end
  if (~isfield(x, 'randstate') | ~isfield(x, 'randnstate'))
    error('Second argument to metrop must contain correct fields')
end
  rand('state', x.randstate);
  randn('state', x.randnstate);
end
return
```

A.4 Matlab Code for the Optimisation of the Input Time-Window

A.4.1 MLP Network

```
clear all;
load dataf2
i=1;
n=3;
m=2;
e1=length(datanasdaq12);
f=floor(e1/3);
g=2*f;
e=3*f;

while(i<=f),
    p=i;
    q=1;
    n=p+11;
    for a=p:n,
        inpt(q,i)=datanasdaq(a,1);
        q=q+1;
    end
```

```
    outpt(1,i)=datanasdaq12(i,1);

        i=i+1;

    end

h=1;
while(i<=g),
p=i;
q=1;
n=p+11;

for a=p:n,
   inpv(q,h)=datanasdaq(a,1);
   q=q+1;
end
outpv(1,h)=datanasdaq12(i,1);
        h=h+1;
        i=i+1;
    end

k=1;

while(i<=e),
p=i;
q=1;
n=p+11;

for a=p:n,
   inpte(q,k)=datanasdaq(a,1);
   q=q+1;
end
outpte(1,k)=datanasdaq12(i,1);

        k=k+1;
        i=i+1;
        n=n+1;
end

   %Training data set

    day1dt=inpt(1,:);
    day2dt=inpt(2,:);
    day3dt=inpt(3,:);
    day4dt=inpt(4,:);
    day5dt=inpt(5,:);
    day6dt=inpt(6,:);
    day7dt=inpt(7,:);
    day8dt=inpt(8,:);
    day9dt=inpt(9,:);
    day10dt=inpt(10,:);
```

```
day11dt=inpt(11,:);
day12dt=inpt(12,:);
day13dt=outpt(1,:);

%Validating data set
day1dv=inpv(1,:);
day2dv=inpv(2,:);
day3dv=inpv(3,:);
day4dv=inpv(4,:);
day5dv=inpv(5,:);
day6dv=inpv(6,:);
day7dv=inpv(7,:);
day8dv=inpv(8,:);
day9dv=inpv(9,:);
day10dv=inpv(10,:);
day11dv=inpv(11,:);
day12dv=inpv(12,:);
day13dv=outpv(1,:);


%Testing data set
day1dte=inpte(1,:);
day2dte=inpte(2,:);
day3dte=inpte(3,:);
day4dte=inpte(4,:);
day5dte=inpte(5,:);
day6dte=inpte(6,:);
day7dte=inpte(7,:);
day8dte=inpte(8,:);
day9dte=inpte(9,:);
day10dte=inpte(10,:);
day11dte=inpte(11,:);
day12dte=inpte(12,:);
day13dte=outpte(1,:);


%normalise the parameters

%Training data
minday1t=min(day1dt);
maxday1t=max(day1dt);
minday2t=min(day2dt);
maxday2t=max(day2dt);
minday3t=min(day3dt);
maxday3t=max(day3dt);
minday4t=min(day4dt);
maxday4t=max(day4dt);
minday5t=min(day5dt);
maxday5t=max(day5dt);
minday6t=min(day6dt);
maxday6t=max(day6dt);
```

```
minday7t=min(day7dt);
maxday7t=max(day7dt);
minday8t=min(day8dt);
maxday8t=max(day8dt);
minday9t=min(day9dt);
maxday9t=max(day9dt);
minday10t=min(day10dt);
maxday10t=max(day10dt);
minday11t=min(day11dt);
maxday11t=max(day11dt);
minday12t=min(day12dt);
maxday12t=max(day12dt);
minday13t=min(day13dt);
maxday13t=max(day13dt);

day1t=(day1dt-minday1t)/(maxday1t-minday1t);
day2t=(day2dt-minday2t)/(maxday2t-minday2t);
day3t=(day3dt-minday3t)/(maxday3t-minday3t);
day4t=(day4dt-minday4t)/(maxday4t-minday4t);
day5t=(day5dt-minday5t)/(maxday5t-minday5t);
day6t=(day6dt-minday6t)/(maxday6t-minday6t);
day7t=(day7dt-minday7t)/(maxday7t-minday7t);
day8t=(day8dt-minday8t)/(maxday8t-minday8t);
day9t=(day9dt-minday9t)/(maxday9t-minday9t);
day10t=(day10dt-minday10t)/(maxday10t-minday10t);
day11t=(day11dt-minday11t)/(maxday11t-minday11t);
day12t=(day12dt-minday12t)/(maxday12t-minday12t);
day13t=(day13dt-minday13t)/(maxday13t-minday13t);



%Validation data
minday1v=min(day1dv);
maxday1v=max(day1dv);
minday2v=min(day2dv);
maxday2v=max(day2dv);
minday3v=min(day3dv);
maxday3v=max(day3dv);
minday4v=min(day4dv);
maxday4v=max(day4dv);
minday5v=min(day5dv);
maxday5v=max(day5dv);
minday6v=min(day6dv);
maxday6v=max(day6dv);
minday7v=min(day7dv);
maxday7v=max(day7dv);
minday8v=min(day8dv);
maxday8v=max(day8dv);
minday9v=min(day9dv);
maxday9v=max(day9dv);
minday10v=min(day10dv);
```

```
maxday10v=max(day10dv);
minday11v=min(day11dv);
maxday11v=max(day11dv);
minday12v=min(day12dv);
maxday12v=max(day12dv);
minday13v=min(day13dv);
maxday13v=max(day13dv);

day1v=(day1dv-minday1v)/(maxday1v-minday1v);
day2v=(day2dv-minday2v)/(maxday2v-minday2v);
day3v=(day3dv-minday3v)/(maxday3v-minday3v);
day4v=(day4dv-minday4v)/(maxday4v-minday4v);
day5v=(day5dv-minday5v)/(maxday5v-minday5v);
day6v=(day6dv-minday6v)/(maxday6v-minday6v);
day7v=(day7dv-minday7v)/(maxday7v-minday7v);
day8v=(day8dv-minday8v)/(maxday8v-minday8v);
day9v=(day9dv-minday9v)/(maxday9v-minday9v);
day10v=(day10dv-minday10v)/(maxday10v-minday10v);
day11v=(day11dv-minday11v)/(maxday11v-minday11v);
day12v=(day12dv-minday12v)/(maxday12v-minday12v);
day13v=(day13dv-minday13v)/(maxday13v-minday13v);


%Testing data set

minday1te=min(day1dte);
maxday1te=max(day1dte);
minday2te=min(day2dte);
maxday2te=max(day2dte);
minday3te=min(day3dte);
maxday3te=max(day3dte);
minday4te=min(day4dte);
maxday4te=max(day4dte);
minday5te=min(day5dte);
maxday5te=max(day5dte);
minday6te=min(day6dte);
maxday6te=max(day6dte);
minday7te=min(day7dte);
maxday7te=max(day7dte);
minday8te=min(day8dte);
maxday8te=max(day8dte);
minday9te=min(day9dte);
maxday9te=max(day9dte);
minday10te=min(day10dte);
maxday10te=max(day10dte);
minday11te=min(day11dte);
maxday11te=max(day11dte);
minday12te=min(day12dte);
maxday12te=max(day12dte);
minday13te=min(day13dte);
maxday13te=max(day13dte);
```

```
   day1te=(day1dte-minday1te)/(maxday1te-minday1te);
   day2te=(day2dte-minday2te)/(maxday2te-minday2te);
   day3te=(day3dte-minday3te)/(maxday3te-minday3te);
   day4te=(day4dte-minday4te)/(maxday4te-minday4te);
   day5te=(day5dte-minday5te)/(maxday5te-minday5te);
   day6te=(day6dte-minday6te)/(maxday6te-minday6te);
   day7te=(day7dte-minday7te)/(maxday7te-minday7te);
   day8te=(day8dte-minday8te)/(maxday8te-minday8te);
   day9te=(day9dte-minday9te)/(maxday9te-minday9te);
   day10te=(day10dte-minday10te)/(maxday10te-minday10te);
   day11te=(day11dte-minday11te)/(maxday11te-minday11te);
   day12te=(day12dte-minday12te)/(maxday12te-minday12te);
   day13te=(day13dte-minday13te)/(maxday13te-minday13te);

a=length(day13t);
q=1;
for t=1:a,
if t==1
day13det(t,1)=0;
day13dev(t,1)=0;
day13dete(t,1)=0;
end
if t>1,
   day13det(t,1)=day13t(q);
   day13dev(t,1)=day13v(q);
   day13dete(t,1)=day13te(q);
   q=q+1;
end
end
   for s=1:f,

   invart(s,1)=day1t(s);
   invart(s,2)=day2t(s);
   invart(s,3)=day3t(s);
   invart(s,4)=day4t(s);
   invart(s,5)=day5t(s);
   invart(s,6)=day6t(s);
   invart(s,7)=day7t(s);
   invart(s,8)=day8t(s);
   invart(s,9)=day9t(s);
   invart(s,10)=day10t(s);
   invart(s,11)=day11t(s);
   invart(s,12)=day12t(s);
   invart(s,13)=day13det(s);
   outvart(s,1)=day13t(s);

end

j=1;
f1=f+1;
```

```
for s=f1:g,

   invarv(j,1)=day1v(j);
   invarv(j,2)=day2v(j);
   invarv(j,3)=day3v(j);
   invarv(j,4)=day4v(j);
   invarv(j,5)=day5v(j);
   invarv(j,6)=day6v(j);
   invarv(j,7)=day7v(j);
   invarv(j,8)=day8v(j);
   invarv(j,9)=day9v(j);
   invarv(j,10)=day10v(j);
   invarv(j,11)=day11v(j);
   invarv(j,12)=day12v(j);
   invarv(j,13)=day13dev(j);
   outvart(j,1)=day13v(j);
   j=j+1;

end

k=1;
g1=g+1;
for s=g1:e,
   invarte(k,1)=day1te(k);
   invarte(k,2)=day2te(k);
   invarte(k,3)=day3te(k);
   invarte(k,4)=day4te(k);
   invarte(k,5)=day5te(k);
   invarte(k,6)=day6te(k);
   invarte(k,7)=day7te(k);
   invarte(k,8)=day8te(k);
   invarte(k,9)=day9te(k);
   invarte(k,10)=day10te(k);
   invarte(k,11)=day11te(k);
   invarte(k,12)=day12te(k);
   invarte(k,13)=day13dete(k);
   outvarte(k,1)=day13te(k);
   k=k+1;
end

%Initialising neural network parameters
nin=13;%Number of input units is 13
nhidden=19%19 hidden layers used (optimum value)
nout=1;%One output value
alpha=0.01;

net1=mlpbin(nin, nhidden, nout, 'linear', alpha); %Linear discrete MLP algorithm chosen

options = zeros(1,18);
```

```
options(1) = 1;

options(14) = 25000;%25000 iterations found to be accurate enough

[net1, options] = netoptbin(net1, options, invart, outvart, 'metropbin');

yout1= mlpbinfwd(net1, invart);
Day4=(yout1(:,1)*(maxday13t-minday13t))+minday13t;

yout2= mlpbinfwd(net1, invarv);

Dy4=(yout2(:,1)*(maxday13t-minday13t))+minday13t;

c=length(Dy4);

erro=0;

for d=1:c,
Err1(c)=Dy4(c)-day13dt(c);
erro=erro+(Err1(c))^2;
end

erro=((erro)^(1/2))/c;

plot(day13dt,'b')
hold on
plot(Dy4,'k')
plot(Day4,'r')
plot(day13dv,'b.')
net1.a
```

A.4.2 RBF Network

```
clear all;
load dataf2
i=1;
n=3;
m=2;
e1=length(datanasdaq12);
f=floor(e1/3);
g=2*f;
e=3*f;

while(i<=f),
   p=i;
   q=1;
   n=p+11;
   for a=p:n,
      inpt(q,i)=datanasdaq(a,1);
      q=q+1;
```

```
    end

    outpt(1,i)=datanasdaq12(i,1);

        i=i+1;

    end

h=1;
while(i<=g),
p=i;
q=1;
n=p+11;

for a=p:n,
   inpv(q,h)=datanasdaq(a,1);
   q=q+1;
end
outpv(1,h)=datanasdaq12(i,1);
        h=h+1;
        i=i+1;
    end

k=1;

while(i<=e),
p=i;
q=1;
n=p+11;

for a=p:n,
   inpte(q,k)=datanasdaq(a,1);
   q=q+1;
end
outpte(1,k)=datanasdaq12(i,1);

        k=k+1;
        i=i+1;
        n=n+1;
end

   %Training data set

    day1dt=inpt(1,:);
    day2dt=inpt(2,:);
    day3dt=inpt(3,:);
    day4dt=inpt(4,:);
    day5dt=inpt(5,:);
    day6dt=inpt(6,:);
    day7dt=inpt(7,:);
    day8dt=inpt(8,:);
```

```
day9dt=inpt(9,:);
day10dt=inpt(10,:);
day11dt=inpt(11,:);
day12dt=inpt(12,:);
day13dt=outpt(1,:);

%Validating data set
day1dv=inpv(1,:);
day2dv=inpv(2,:);
day3dv=inpv(3,:);
day4dv=inpv(4,:);
day5dv=inpv(5,:);
day6dv=inpv(6,:);
day7dv=inpv(7,:);
day8dv=inpv(8,:);
day9dv=inpv(9,:);
day10dv=inpv(10,:);
day11dv=inpv(11,:);
day12dv=inpv(12,:);
day13dv=outpv(1,:);


%Testing data set
day1dte=inpte(1,:);
day2dte=inpte(2,:);
day3dte=inpte(3,:);
day4dte=inpte(4,:);
day5dte=inpte(5,:);
day6dte=inpte(6,:);
day7dte=inpte(7,:);
day8dte=inpte(8,:);
day9dte=inpte(9,:);
day10dte=inpte(10,:);
day11dte=inpte(11,:);
day12dte=inpte(12,:);
day13dte=outpte(1,:);


%normalise the parameters

%Training data
minday1t=min(day1dt);
maxday1t=max(day1dt);
minday2t=min(day2dt);
maxday2t=max(day2dt);
minday3t=min(day3dt);
maxday3t=max(day3dt);
minday4t=min(day4dt);
maxday4t=max(day4dt);
minday5t=min(day5dt);
maxday5t=max(day5dt);
```

```
minday6t=min(day6dt);
maxday6t=max(day6dt);
minday7t=min(day7dt);
maxday7t=max(day7dt);
minday8t=min(day8dt);
maxday8t=max(day8dt);
minday9t=min(day9dt);
maxday9t=max(day9dt);
minday10t=min(day10dt);
maxday10t=max(day10dt);
minday11t=min(day11dt);
maxday11t=max(day11dt);
minday12t=min(day12dt);
maxday12t=max(day12dt);
minday13t=min(day13dt);
maxday13t=max(day13dt);

day1t=(day1dt-minday1t)/(maxday1t-minday1t);
day2t=(day2dt-minday2t)/(maxday2t-minday2t);
day3t=(day3dt-minday3t)/(maxday3t-minday3t);
day4t=(day4dt-minday4t)/(maxday4t-minday4t);
day5t=(day5dt-minday5t)/(maxday5t-minday5t);
day6t=(day6dt-minday6t)/(maxday6t-minday6t);
day7t=(day7dt-minday7t)/(maxday7t-minday7t);
day8t=(day8dt-minday8t)/(maxday8t-minday8t);
day9t=(day9dt-minday9t)/(maxday9t-minday9t);
day10t=(day10dt-minday10t)/(maxday10t-minday10t);
day11t=(day11dt-minday11t)/(maxday11t-minday11t);
day12t=(day12dt-minday12t)/(maxday12t-minday12t);
day13t=(day13dt-minday13t)/(maxday13t-minday13t);


%Validation data
minday1v=min(day1dv);
maxday1v=max(day1dv);
minday2v=min(day2dv);
maxday2v=max(day2dv);
minday3v=min(day3dv);
maxday3v=max(day3dv);
minday4v=min(day4dv);
maxday4v=max(day4dv);
minday5v=min(day5dv);
maxday5v=max(day5dv);
minday6v=min(day6dv);
maxday6v=max(day6dv);
minday7v=min(day7dv);
maxday7v=max(day7dv);
minday8v=min(day8dv);
maxday8v=max(day8dv);
minday9v=min(day9dv);
maxday9v=max(day9dv);
```

```
minday10v=min(day10dv);
maxday10v=max(day10dv);
minday11v=min(day11dv);
maxday11v=max(day11dv);
minday12v=min(day12dv);
maxday12v=max(day12dv);
minday13v=min(day13dv);
maxday13v=max(day13dv);

day1v=(day1dv-minday1v)/(maxday1v-minday1v);
day2v=(day2dv-minday2v)/(maxday2v-minday2v);
day3v=(day3dv-minday3v)/(maxday3v-minday3v);
day4v=(day4dv-minday4v)/(maxday4v-minday4v);
day5v=(day5dv-minday5v)/(maxday5v-minday5v);
day6v=(day6dv-minday6v)/(maxday6v-minday6v);
day7v=(day7dv-minday7v)/(maxday7v-minday7v);
day8v=(day8dv-minday8v)/(maxday8v-minday8v);
day9v=(day9dv-minday9v)/(maxday9v-minday9v);
day10v=(day10dv-minday10v)/(maxday10v-minday10v);
day11v=(day11dv-minday11v)/(maxday11v-minday11v);
day12v=(day12dv-minday12v)/(maxday12v-minday12v);
day13v=(day13dv-minday13v)/(maxday13v-minday13v);


%Testing data set

minday1te=min(day1dte);
maxday1te=max(day1dte);
minday2te=min(day2dte);
maxday2te=max(day2dte);
minday3te=min(day3dte);
maxday3te=max(day3dte);
minday4te=min(day4dte);
maxday4te=max(day4dte);
minday5te=min(day5dte);
maxday5te=max(day5dte);
minday6te=min(day6dte);
maxday6te=max(day6dte);
minday7te=min(day7dte);
maxday7te=max(day7dte);
minday8te=min(day8dte);
maxday8te=max(day8dte);
minday9te=min(day9dte);
maxday9te=max(day9dte);
minday10te=min(day10dte);
maxday10te=max(day10dte);
minday11te=min(day11dte);
maxday11te=max(day11dte);
minday12te=min(day12dte);
maxday12te=max(day12dte);
minday13te=min(day13dte);
```

```
   maxday13te=max(day13dte);

   day1te=(day1dte-minday1te)/(maxday1te-minday1te);
   day2te=(day2dte-minday2te)/(maxday2te-minday2te);
   day3te=(day3dte-minday3te)/(maxday3te-minday3te);
   day4te=(day4dte-minday4te)/(maxday4te-minday4te);
   day5te=(day5dte-minday5te)/(maxday5te-minday5te);
   day6te=(day6dte-minday6te)/(maxday6te-minday6te);
   day7te=(day7dte-minday7te)/(maxday7te-minday7te);
   day8te=(day8dte-minday8te)/(maxday8te-minday8te);
   day9te=(day9dte-minday9te)/(maxday9te-minday9te);
   day10te=(day10dte-minday10te)/(maxday10te-minday10te);
   day11te=(day11dte-minday11te)/(maxday11te-minday11te);
   day12te=(day12dte-minday12te)/(maxday12te-minday12te);
   day13te=(day13dte-minday13te)/(maxday13te-minday13te);

a=length(day13t);
q=1;
for t=1:a,
if t==1
day13det(t,1)=0;
day13dev(t,1)=0;
day13dete(t,1)=0;
end
if t>1,
   day13det(t,1)=day13t(q);
   day13dev(t,1)=day13v(q);
   day13dete(t,1)=day13te(q);
   q=q+1;
end
end
   for s=1:f,

   invart(s,1)=day1t(s);
   invart(s,2)=day2t(s);
   invart(s,3)=day3t(s);
   invart(s,4)=day4t(s);
   invart(s,5)=day5t(s);
   invart(s,6)=day6t(s);
   invart(s,7)=day7t(s);
   invart(s,8)=day8t(s);
   invart(s,9)=day9t(s);
   invart(s,10)=day10t(s);
   invart(s,11)=day11t(s);
   invart(s,12)=day12t(s);
   invart(s,13)=day13det(s);
   outvart(s,1)=day13t(s);

end

j=1;
```

```
f1=f+1;

for s=f1:g,

   invarv(j,1)=day1v(j);
   invarv(j,2)=day2v(j);
   invarv(j,3)=day3v(j);
   invarv(j,4)=day4v(j);
   invarv(j,5)=day5v(j);
   invarv(j,6)=day6v(j);
   invarv(j,7)=day7v(j);
   invarv(j,8)=day8v(j);
   invarv(j,9)=day9v(j);
   invarv(j,10)=day10v(j);
   invarv(j,11)=day11v(j);
   invarv(j,12)=day12v(j);
   invarv(j,13)=day13dev(j);
   outvart(j,1)=day13v(j);
   j=j+1;

end

k=1;
g1=g+1;
for s=g1:e,
   invarte(k,1)=day1te(k);
   invarte(k,2)=day2te(k);
   invarte(k,3)=day3te(k);
   invarte(k,4)=day4te(k);
   invarte(k,5)=day5te(k);
   invarte(k,6)=day6te(k);
   invarte(k,7)=day7te(k);
   invarte(k,8)=day8te(k);
   invarte(k,9)=day9te(k);
   invarte(k,10)=day10te(k);
   invarte(k,11)=day11te(k);
   invarte(k,12)=day12te(k);
   invarte(k,13)=day13dete(k);
   outvarte(k,1)=day13te(k);
   k=k+1;
end

%Initialising neural network parameters
nin=13;%Number of input units is 13
nhidden=19%19 hidden layers used (optimum value)
nout=1;%One output value
alpha=0.01;

net1=rbfbin(nin,nhidden,nout,'gaussian','linear'); % discrete RBF network chosen

options = zeros(1,18);
```

```
options(1) = 1;
options(14) = 30000;


[net1, options] = netoptbin(net1, options, invart, outvart, 'metropbin1');



[p1,p2]=size(invart);
for n1=1:p1
   invart1(n1,:)=invart(n1,:).*net1.a;
   invarv1(n1,:)=invarv(n1,:).*net1.a;
   invarte1(n1,:)=invarte(n1,:).*net1.a;
end
options(14) = 1000;%1000 iterations found to be accurate enough

[net2, options, varargout] = netopt(net2, options, invart1, outvart, 'scg');

yout1r=rbffwd(net2,invart1);
yout2r=rbffwd(net2,invarv1);

yout1= rbfbinfwd(net1, invart);
y3=rbfbinfwd(net1,invarv);

Day13t1=(yout1(:,1)*(maxday13t-minday13t))+minday13t;
Day13t2=(yout1r(:,1)*(maxday13t-minday13t))+minday13t;
Day13t3=(yout2r(:,1)*(maxday13v-minday13v))+minday13v;



subplot(2,1,1),plot(Day13t1,'b')
hold on
subplot(2,1,1),plot(day13dt,'r')
subplot(2,1,1),plot(Day13t2,'y')
subplot(2,1,2),plot(Day13t3,'b.')
hold on
subplot(2,1,2),plot(day13dv,'r.')
```