DATA PREDICTION USING NEURAL NETWORK

A DISSERTATION

Submitted in partial fulfilment of the requirements for the award of the degree

of

MASTER OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

SANDEEP SINGH RAWAT

ER & DCI
NOIDA

IIT Roorkee-ER&DCI, Noida
C-56/1, "Anusandhan Bhawan"
Sector 62, Noida-201 307

FEBRUARY, 2003

CANDIDATE'S DECLARATION

I hereby declare that the work presented in this dissertation titled "DATA PREDICTION USING NEURAL NETWORK", in partial fulfillment of the requirements for the award of the degree of **Master of Technology** in **Information Technology**, submitted in **IIT, Roorkee – ER&DCI Campus, Noida,** is an authentic record of my own work carried out during the period from August 2002 to February, 2003 under the guidance of **Mr. M. K. Bhattacharya**, Senior Project Manager, Electronics Research and Development Centre of India, Noida.

The matter embodied in this dissertation has not been submitted by me for award of any other degree or diploma

Date : 21 - 02 - 2003

Place: Noida

[illegible]

(Sandeep Singh Rawat)

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

[illegible]

Date : 22 - 02 - 2003

Place: Noida

(Mr. M. K. Bhattacharya)

Senior Project Manager

ER&DCI, Noida.

ACKNOWLEDGEMENT

The work presented in this report would not have been completed without the guidance and support of many people. Firstly , I would like to thank **Prof. Prem Vratt,** Director, IIT Roorkee and **Mr. R.K.Verma,** Executive Director, ER&DCI, Noida. I would like to thank **Prof. A.K.Awasthi,** Dean PGS&R, IIT Roorkee. for providing good environment. I am also grateful to **Prof. R.P.Agrawal,** Course Coordinator, M.Tech (IT), IIT Roorkee and **Mr. V.N.Shukla,** Course Coordinator, M.Tech (IT), ER&DCI, Noida for his support and guidance.

I would like to thank my guide, **Mr. M.K.Bhattacharya,** Senior Project Manager ER&DCI for his constant support, incredible enthusiasm and encouragement. I am also grateful to **Mr. Munish Kumar,** Project Engineer for the cooperation extended by him in the successful completion of this report. I would also like to thank Ms. Vaishali Kale, for his timely help and constant guidance.

Most of all I would like to thank my family. My parents provided me a perfect environment for my studies and supported me throughout. Finally, I would like to extend my gratitude to all those persons who directly or indirectly helped me in the process and contributed towards this work.

[illegible]

(Sandeep Singh Rawat)

Enrollment No. 019043

CONTENTS

CANDIDATE'S DECLARATION	(i)
ACKNOWLEDGEMENT	(ii)
ABSTRACT	1
1. INTRODUCTION	3
1.1 Objective	4
1.2 Artificial Neural Systems	4
1.3 Scope	6
1.4 Problem Definition	6
1.5 Organization of Dissertation	7
2. LITERATURE SURVEY FOR DATA PREDICTION	9
3. TRADING SCHEME	11
3.1 Mechanic Concepts	11
3.2 System Development Life Cycle	12
3.2.1 Investigation	12
3.2.2 Design	12
3.2.3 Implementation	12
3.2.4 Evaluation	13
4. ARTIFICIAL NEURAL NETWORKS	15
4.1 Single and Multi-layer Perceptrons	15
4.2 Backpropagation Neural Network	18
4.3 Learning and Training	20
5. DATA SELECTION	23
5.1 Choosing the Output and Objective	23
5.2 Choosing the Inputs	23

5.3 Preprocessing Data 24

5.3.1 Highlight Features 24

5.3.2 Transform 24

5.3.3 Scale and bias 24

5.3.4 Reduce Dimensionality 25

5.4 Highlight Features in the Data 26

5.4.1 Normalizing the Range 27

5.4.2 The target 27

6. DATA PREDICTION AND IMPLEMENTATION DETAILS 29

7. RESULTS AND DISCUSSIONS 35

7.1 Training and Test Results 35

7.2 Traditional Statistical Approach Pitfalls 38

7.2.1 Evaluate the right thing 38

7.2.2 Evaluation data 39

7.3 Benefits and limitations of neural network 40

7.3.1 Benefits 40

7.3.2 Limitations 41

8. CONCLUSION 43

REFERENCES 45

APPENDIX A: Training file

APPENDIX B: Test file

APPENDIX C: Blind test file

APPENDIX D: Weight file

APPENDIX E: Output file

ABSTRACT

Neural networks (NNs), as artificial intelligence methods, have become very important in making stock market predictions. Much research on the applications of NNs for solving business problems have proven their advantages over statistical and other methods that do not include AI, although there is no optimal methodology for a certain problem. The system has been trained with the Standard & Poor (S&P) 500 composite indexes of past twelfth years. It can be concluded from analysis that NNs are most implemented in forecasting stock prices, returns, and stock modeling, and the most frequent methodology is the Backpropagation algorithm. Inspite of many benefits, there are limitations that should be investigated.

Stocks are commonly predicted on the basis of daily data, although some researchers use weekly and monthly data. Additionally, future research should focus on the examinations of other types of networks that were rarely applied, such as Hopfiled's, Kohonen's, etc. This data prediction can be used in weather forecasting also. End user for this data prediction, either the stockbroker or else who wants to predict the future record, based on the past data, but the key to all applications though, is how we present and enhance data, and working through parameter selection by trial and error.

INTRODUCTION

Stock market prediction is believed to be a very difficult task. Huge amount of immeasurable and unknown variables, unknown relationships between those variables and relative small number of observations makes stock market prediction as a complex problem. At this moment, no fixed trading rule provides an everlasting profit from stock market. Since stock investment is a popular way to achieve a great amount of return in a short time period, many people like to find out a successful method to beat stock markets.

Serious people will treat this complex prediction task in a scientific way because an efficient stock market prediction method is possible to gain several millions of money or more. Then they are very careful at the prediction algorithm construction and try to reduce faults as much as possible within the entire process. Around the financial world, many scientific stock market prediction methodologies are commonly used and they are derived by some different approaches. The first approach is to study economic and industrial conditions about organizations. Fundamental analysis is a popular method based on this approach. To cut the matter short, this methodology is to study organization's accounting information, such as revenues, earnings, debt ratio, return on equity and profit margins, to determine the underlying value and the future potential of organization, as the future trend about stock prices [13]. However, investors cannot obtain those figures easily or it can be said investors require costs for getting those kinds of information because some information is said to be confidential for view of organization. Another prediction approach is to study past organization stock prices and related data to determine the future stock prices. Technical and diagrammatic analyses are based on this approach. In addition, same data can treat as different ways, such as stock price time series and stock chart patterns. Therefore, financial experts have ever suggested many prediction indicators and patterns to try to achieve prediction task. As many experts usually use past stock price data to interpret the future trend of stock price, statistical approach becomes a popular forecasting methodology. Regression and time-series analyses are typical methods. However, those methods assume linear model property and stock prices always do not satisfy this property. Then it is difficult to achieve success on. stock market prediction by those methods. A new generation of methodologies, including neural networks, expert systems, chaos theory and genetic algorithms, have attracted attention for stock market prediction. In particular, neural network approach is being used extensively. Human brain consists of a huge number of neurons. Those neurons are connected by axons and they combine into a network called 'biological' neural network. An 'artificial' neural network is a computational system simulating the activities and properties of a biological neural network. In simple words, an artificial neural network is a collection of artificial neurons, connected through links called connections. The aim of an artificial neural network is to achieve the ability of a biological neural network, that is, performing pattern recognition, classification, memorization and complex problem solving. Neural networks are generally regarded as 'black boxes'. Since actual model structures are not known, raw data is submitted into neural networks and training process will be applied to the networks. Then the networks will have recognition ability for patterns at the same problem domain [4]. The aim of project is to construct stock market trading and to maximize the profit by applying neural network principle.

1.1 Objective

The objective of project is to try to apply neural network approach to find out some patterns to appear increases or decreases of data for e.g. stock price levels and financial forecasting.

1.2 Artificial Neural Systems

In order to understand how an artificial neural network functions, we must understand the biological neural network first. The human brain is a vast communication network in which around 100 billion brains cells called neurons are interconnected to other neurons. Each neuron contains a soma, nucleus, axon, yet they don't play an important role in receiving and outputting electrical impulses. Each neuron has several dendrites, which connect to other neurons, and when a neuron fires (sending electrical impulse), a positive or negative charge is sent to other neurons. When a neuron receives signals from other neurons, spatial and temporal summation occurs where spatial summation converts several weak signals into a large one, and temporal summation converts a series of weak signals from the same source into a large signal. The electrical impulse is then transmitted through axon to terminal buttons to other neurons. The axon hillock plays an important role because if the signal is not strong enough to pass through it, no signal will be transmitted. The terminal buttons shown on figure 1.1 are connected to other neurons or muscle cells.

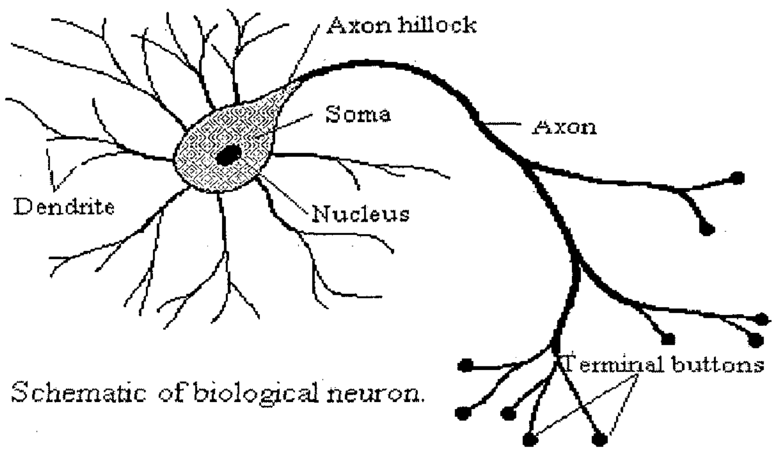


Figure 1.1: Schematic of biological neuron

The gap between the two neurons is called the synapse. The synapse also determines the "weight" of the signal transmitted. The more often a signal is sent through the synapse, the easier it is for the signal to be sent through. In theory, this is how humans memorize or recognize patterns; which is why when humans practice certain tasks continuously, they become more familiar or used to the tasks [18].

Because the neural network mimics the biological neural network, an ANN has to resemble essential parts of a BNN, such as neuron, axons, hillock and The output of each neuron in the figure 1.2 is the sum of all the inputs multiplying the weights plus the offset value and through a non-linear function

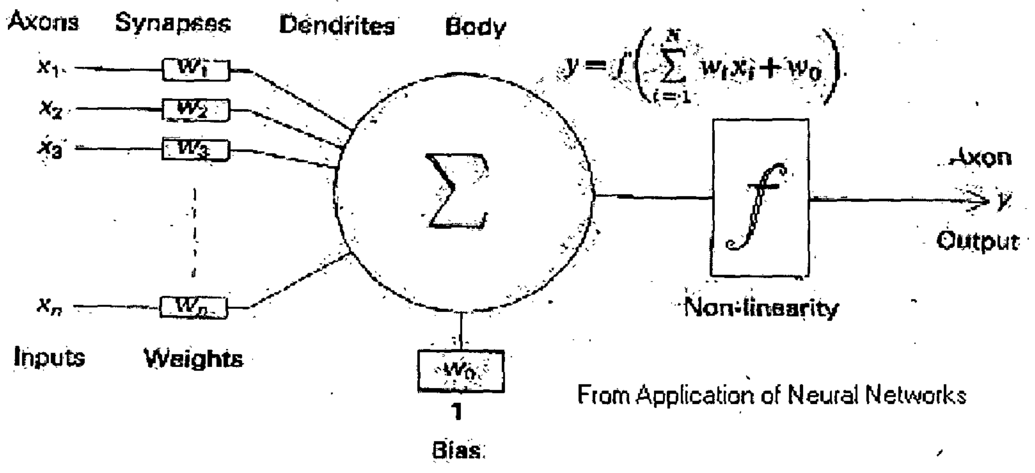


Figure 1.2: Clearly demonstrates parts of a BNN in terms of ANN.

1.3 Scope

This data prediction can be used anywhere where we have past data and we want to know the future value. Like weather forecasting etc. The key to all applications though, is how we present and enhance data, and working through parameter selection by trial and error.

1.4 Problem Definition

Many strategies and methodologies were put forward for data prediction. However, no fixed trading rule provides everlasting profits. So I would like to use scientific methods. Around the financial world, fundamental and technical analyses are popular methods to determine when to buy or sell. The aim of fundamental analysis is to determine the future trends of data value by studying everything from the economy and industry conditions. Usually, revenues, earnings, debt ratio, return on equity, profit margins are used to determine an organization's underlying value and potential for future growth, as financial forecasting and its trend. However, investors cannot obtain the accurate figures anytime. They always estimate by organization's annual report and study the economy or industry conditions. Then it becomes a qualitative framework and the forecast accuracy cannot be evaluated in a scientific way. The aim of technical analysis is to determine by recognizing patterns on financial data or statistical indicators by past financial data prices and volumes[15]. It is a quantitative methodology. Nowadays, plenty of indicators and patterns are recommended to apply to predict data. Analogue to fundamental analysis, distinct chartists always provide opposite views for the same financial, by different kinds of indicators or patterns. So an `honest' strategy is required for most of investors and artificial neural network is a reasonable methodology to predict financial data because neural networks can be trained with raw data to product outputs or classify raw data without knowledge or understanding the model structure. Thus, in this project, I would like to study the data prediction using neural network.

1.5 Organization of Dissertation

We shall see below the brief description of the chapter in the Thesis Report.

Chapter 2 Describe the Literature Survey for Data Prediction.

Chapter 3 This section presents a concept for building a trading scheme. A number of guidelines are suggested.

Chapter 4 Describe the Artificial Neural Network to understand how an artificial neural network functions.

Chapter 5 Describe the Data Selection, how preprocessing is done, which is very important in most of the neural network model.

Chapter 6 Gives the Implementation Details, gives the functions which is used in the program also shows the data flow diagram.

Chapter 7 Gives the Training and Testing Result and discusses the Benefits and limitations of Neural Network.

Chapter 8 Includes the Conclusion and Future Scope.

LITERATURE SURVEY FOR DATA PREDICTION

A famous principle called Efficient Market Hypothesis (EMH). In the weak form of EMH, current stock price has reflected all information about past prices. That is, past stock prices are not useful for forecasting future trend and the movements of stock prices are unpredictable. If EMH holds, nobody can gain benefit in the stock market and the stock market forecast performance will be not better than random guesses[11]. Seiler et al.[12] performed some time series analysis for all stocks listed on New York Stock Exchange. They concluded stock prices are followed random walk model. Although EMH was held in many papers, many people proposed new methods to "beat" the stock market. Mathematicians used principles of statistics to predict stock prices. Time series is a traditional and reasonable methodology. Traditional statistic principle assumes the model is linear and normally distributed. As EMH holds, random walk model is identified which the next period stock price is just the current price plus an unpredictable white noise (sometimes with a positive or negative expected return). However, Aparicio et al. [1] stated that stock prices are not normally distributed. So classic statistic approach may not be applicable at chaotic financial market. Technical and diagrammatic analyses are popular prediction approaches to study past stock prices and related data to determine the future stock price trend. In practical, financial experts recommend many indicators. Moving average and relative strength index (RSI) are the most commonly used. The former one is based on a statistical topic called time series analysis and the latter one is not recommended by its discover. However, Jones [9] stated that rules for making decisions from charts are not unique. As many different patterns are used and patterns are difficult to be accurately recognized, the techniques are inconclusive. Many articles applied neural network approach to forecast stock market and had a good performance. A neural network paradigm called multi-layer perceptrons or Feedforward network is commonly used for many studies. Chenoweth et al. [2] built up stock prediction models applying buy-and-hold strategy, neural network and technical analysis knowledge respectively. The best result among models was the system applying neural networks. Zekic [15] performed a survey for comparing effectiveness of different neural network approaches. 12 representative papers were chosen to analyze their problem domain, model and forecasting performance. The comparisons showed that stock market prediction using neural networks outperform other statistical models. Also, MLP with backpropagation training algorithm and data pre-processing performs good results. The correctness was about 70~100%. Multi-layer perceptrons look like a nice approach for stock market prediction by experiment results of many papers. However, de la Maza [4] criticized application of MLP for financial time series prediction. They reported sum-squared-like errors were not good measures and they suggested Sharpe ratio as the measure for financial application, which the objective is to maximum investment profit. De Bodt et al. held similar paper [3] that they applied theories of statistics and suggested price returns as expected model outputs, not stock prices. Diagrammatic analysis is a commonly used technical analysis methodology. Its aim is to identify specific patterns to represent the future trend of stock prices. Since the pattern recognition is subjective, some researchers suggested an unsupervised neural network paradigm called self-organizing maps (SOM) to achieve this task. Kohonen [16] developed self-organizing map to transform an arbitrary dimensional input data into lower dimensional (one or two) graphs to represent similarity without specifying output classes. Kaski [10] stated SOM can be used to perform a K-means clustering application. Deboeck [5] demonstrated an application of mutual fund selection by applying appropriate data into a SOM. Experiment results showed that SOM classification could reduce labor effort at classification process, which is working by fund managers. Fu, Chung and Luk [8] used self-organizing maps to recognize stock time series patterns. Good results are obtained by time series from Hong Kong Hang Seng Index.

Chapter 3

TRADING SCHEME

This section presents a concept for building a trading scheme. A number of guidelines are suggested. In addition, to construct an efficient and effective trading scheme, systematic approaches are used and software development life cycle approach is recommended to use.

3.1 Mechanic Concepts

In order to yield a huge amount of profit and avoid risks from financial market, a scientific and systematic approach is recommended. Here some design rules were refined. The first rule is simplicity. As distinct financial experts recommended many different kinds of technical indicators, fundamental data and chart patterns, excess number of decision rules will lead to conflicts and common investors are difficult to judge by opposite decisions from different indicators. The second guideline is quantification. An ordinal and numerical measure can lead to many advantages. Obvious, mathematical knowledge can be applied for quantitative measurement. Another point is the forecast performance of quantitative methods can be evaluated easily by applying mathematics. After this, clarity is another keyword for a good trading scheme. The meaning is quite similar to that of simplicity but their focuses are different. Simplicity concentrates on the rules to make decision and clarity focuses on the result representation after decision making. The trading signals must be as simple as possible because this reduces efforts for system users to make trading decisions. Relative Strength Index (RSI) is an example to show the importance of clear signals. It is an oscillating indicator discovered by Wilder. It ranges from 0 to 100 and represents the status of stock. Large RSI value means the stock is overbought and small RSI value indicates the stock is oversold. By inspection, investors must buy stocks at RSI value is small and sell them at a large RSI value. However, different sets of values are recommended by different experts, e.g. 30/70, 20/80, 30/80. This leads to confusion for investors that they cannot make decisions when the RSI value lies between 20 and 30, or between 70 and 80. Then faults will be made easily and consequently investors become losers. Consequently, a good trading scheme only includes clear indicators for buying and selling. Another consideration for designing a trading scheme is robustness. A well-established trading scheme must adapt different conditions, such as time periods and financial markets. It is very important because the trading scheme must keep its profit making performance consistently over time. Otherwise, investors will face a risk of system unpredictability. In addition, lack of robustness makes investor to build different plans for different stocks and this increases their efforts [12].

3.2 System Development Life Cycle

Similar to software development, trading scheme construction is also an engineering work and a systematic approach can achieve a qualified deliverable, i.e. effective and efficient trading scheme. To make it simpler, a four-phase model is introduced. This is linear sequential model see Figure 3.1.

3.2.1 Investigation

At the beginning, studies about financial markets must be held. The objectives of the trading scheme must be defined in a quantitative view. Also, data collection must be performed such as financial data like stock price sequences. After enough information is obtained and reviewed, a feasibility test may be conducted to make go or not-go decision.

3.2.2 Design

The objective of design is to construct a trading scheme design by the information gathered by the investigation stage. A trading scheme design may include several components, including input attributes, model architecture, operation procedures and trading signal interpretation.

3.2.3 Implementation

The aim of this stage is to translate the trading scheme design into a workable system. For computerized trading schemes, programming is the primary activity in the implementation stage. If the system can be performed manually, the procedure must be well defined and structured for convenience at use.

3.2.4 Evaluation

Once the system is implemented, it can be tested. Two phases are included in evaluation stage: 'code' testing and performance evaluation. The former phase consists of validation and verification processes to guarantee the system is correctly implemented. The later one is to fit some real data into the system to find out the system performance and behaviors. So, evaluation criteria must be predefined.

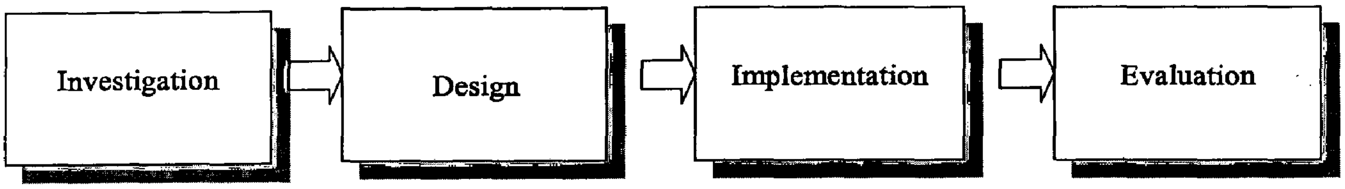


Figure 3.1: Linear Sequential Model

Chapter 4

ARTIFICIAL NEURAL NETWORK

A Neural Network is an interconnected assembly of simple processing elements, units or nodes, whose functionality is loosely based on the animal neuron. The processing ability of the network is stored in the inter-unit connection strengths, or weights, obtained by a process of adaptation to, or learning from, a set of training patterns.

4.1 Single and Multi-layer Perceptrons

A perceptron is a simple neural network model introduced by Frank RosenBlatt in 1958.A single layer perceptron is used to classify an input vector into several classes. In a single layer perceptron, the input values and activation level of the perceptron are either –1 or 1; weights are real-valued (between 0 and 1). The activation level is given by summing the weighted input values $\Sigma x_i w_i$. Perceptrons use a simple hard-limiting threshold function, where activation above a threshold results in an output value of 1, and –1 otherwise.

$$\begin{aligned} \text{Perceptron output} &= \mathrm{sign}(\Sigma x_i w_i) \\ &= 1 \quad \text{if } \Sigma x_i w_i >= t \\ &= -1 \quad \text{if } \Sigma x_i w_i <= t \end{aligned}$$

The perceptron uses a simple form of supervised learning. The way a perceptron learns to distinguish patterns is through modifying its weights to reduce error. The adjustment for the weight Δw_i on the ith component of the input vector is given by:

$$\Delta w_i = c\, x_i\, \delta$$

where c = learning rate

d = desired output

δ = (desired output) – (actual output) = $d - \mathrm{sign}(\Sigma x_i w_i)$

Single layer perceptrons can only solve problems where the solutions can be divided by a line (or hyperplane). The classes to be distinguished should be linearly separable. Therefore, a single layer perceptron cannot express non-linear decisions like the XOR problem. Single layer perceptron is shown in figure 4.1.

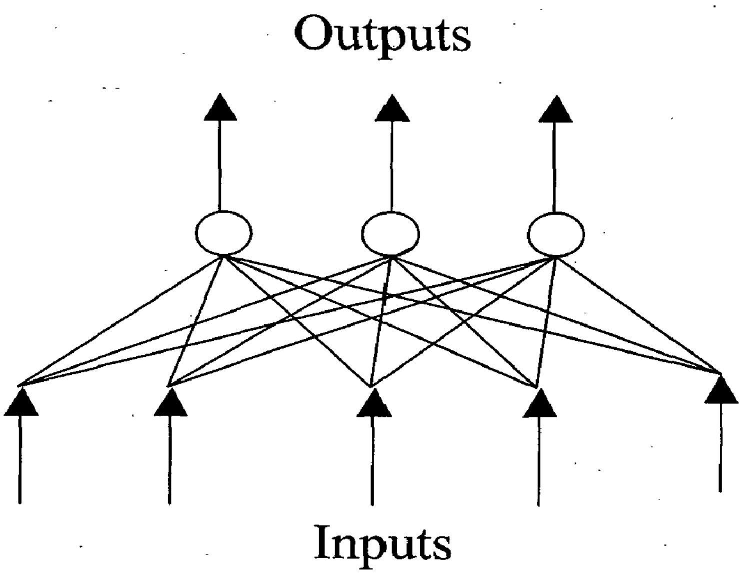


Figure 4.1: Single-layer Perceptron

Multi-layer perceptions are feed-forward nets with one or more layers of nodes between the input and output nodes. These additional layers contain hidden units or nodes that are not directly connected to both the inputs and outputs. Multi-layer perceptions overcome many of the limitations of the single layer perceptions. Multi layer perceptron is shown in figure 4.2. The capabilities of multi-layer perceptions stem from the non-linearity used within nodes. In multi-layer networks, when adjusting a weight anywhere in the network, one has to be able to tell what effect this will have on the overall effect of the network. To do this, one has to look at the derivative of the error function with respect to that weight.

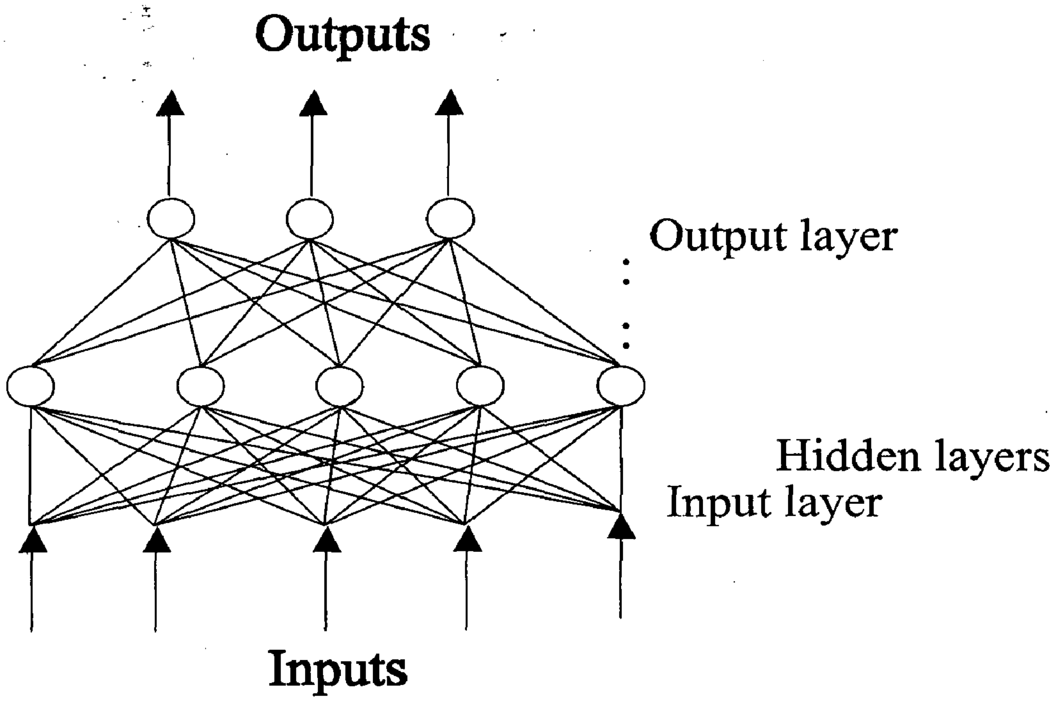
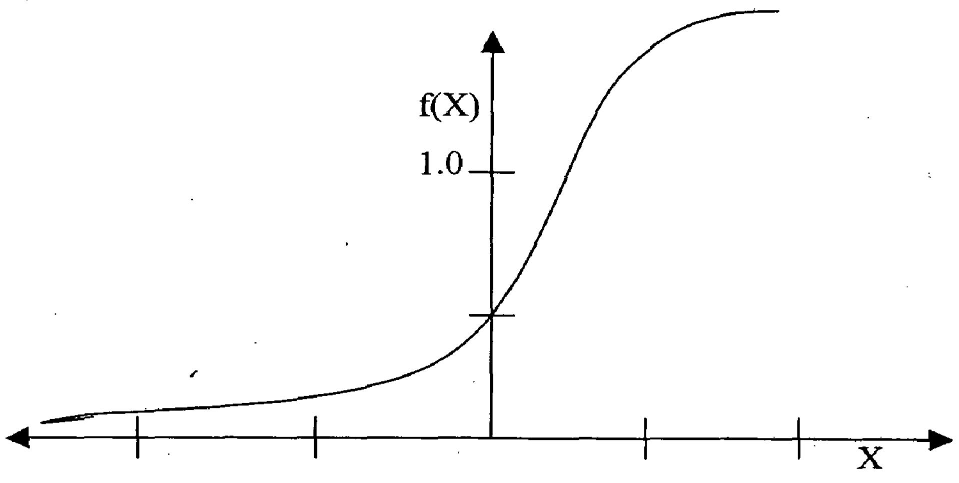


Figure 4.2: Multi-layer Perceptron

The most popular continuous activation function used within backpropagation nets is the sigmoid function or the logistic function given by the equation:

$$f(net) = 1 / (1 + e^{-\lambda * net}), \text{ where net} = \Sigma x_i w_I$$

The shape of the sigmoid function is shown in the figure 4.3. As λ (called the squashing parameter) gets large, the sigmoid function approaches a linear threshold function ~~over {0, 1}; as it gets closer to 1, it approaches a straight line. This activation~~ function is non-linear, scaled and differentiable.



$f(X) = 1/(1 + e^{-x})$

Figure 4.3: Sigmoid Function

4.2 Backpropagation Neural Network

The backpropagation algorithm is perhaps the most widely used supervised training algorithm for multilayered feedforward networks. The backpropagation training algorithm is an iterative gradient algorithm designed to minimize the mean square error between the actual output of a multilayer feedforward perception and the desired output. In the backpropagation algorithm, a feedforward phase is first done on an input pattern to calculate the net error. Then, the algorithm uses this computed output error to change the weight values in the backward direction. The error is slowly propagated backwards through the hidden layers - and hence its name. Errors in the output determine measures of hidden layer output errors, which are used as a basis for adjustment of connection weights between the input and hidden layers. Adjusting the two sets of weights between the pairs of layers and recalculating the outputs is an iterative process that is carried on until the errors fall below a tolerance level. Learning rate parameters scale the adjustments to weights. A momentum parameters can also be used in scaling the adjustments from a previous iteration and adding to the adjustments in the current iteration. The feedforward backpropagation network maps the input vectors to output vectors. It does not have feedback connections, but errors are backpropagated during training. Pairs of input and output vectors are chosen to train the network first. Once training is completed, the weights are set and the network can be used to find outputs for new inputs. The dimension of the input vector determines the number of neurons in the input layer, and the number of neurons in the outputs layer is determined by the dimension of the outputs. If there are k neurons in the input layer and m neurons in the output layer, then this network can make a mapping from k-dimensional space to m-dimensional space. Mapping is depends on pair of patterns or vectors are used as exemplars to train the network, which determine the network weights. Once trained, the network gives you the image of new input vectors under this mapping.

The architecture of Feedforward backpropagation network is shown in Figure 4.4. The number of neurons in the input layer and output layer are determined by the dimensions of the input and output patterns, respectively. There can be many hidden layers here illustrate with only one hidden layer. It is not easy to determine how many neurons are needed for the hidden layer. Figure 4.4 show with five input neurons, three neurons in the hidden layer, and five output neurons, with a few representative connections.

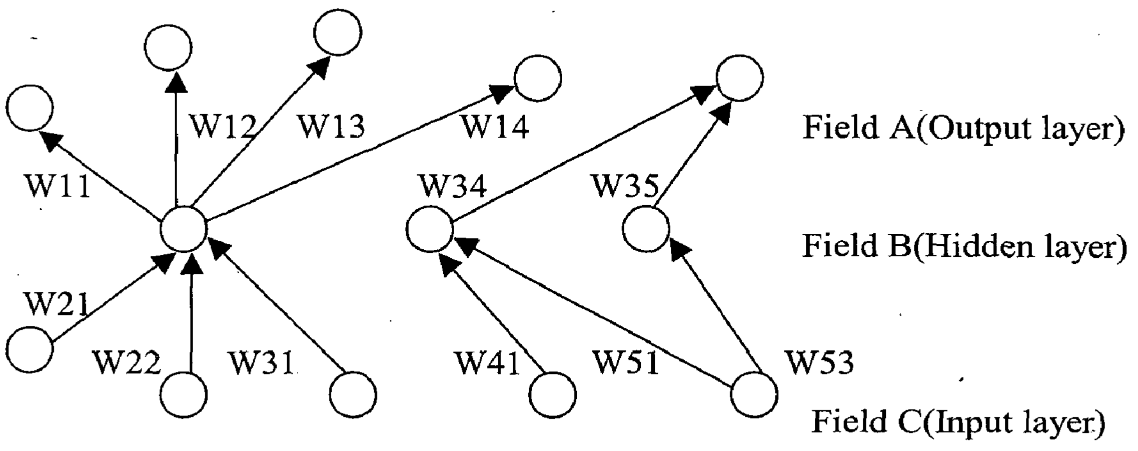


Figure 4.4: Layout of a Feedforward Backpropagation Network

The network has three fields of neurons: one for input neurons, one for hidden processing elements, and one for the output neurons. As already stated, connections are for feed forward activity. There are connections from every neurons in field A to every one in field B, and, in turn,, from every neuron in field B to every neuron in field C. Thus, there are two sets of weights, those figuring in the activation of hidden layer neurons, and those that help determine the output neuron activation. In training, all of these weights are adjusted by considering cost function in terms of the error in the computed output pattern and the desired output pattern.

The feedforward backpropagation network undergoes supervised training, with a finite number of pattern pairs consisting of an input pattern and a desired or target output pattern. An input pattern is presented at the input layer. The neurons here pass the pattern activation to the next layer neurons, which are in a hidden layer. The outputs of the hidden layer neurons are obtained by using perhaps a bias, and also a threshold function with the activation determined by the weights and the inputs. These hidden layer outputs become inputs to the output neurons, which process the inputs using an optional bias and a threshold function. The final output of the network is determined by the activation from the output layer.

The computed pattern and the input pattern are compared, a function of this error for each component of the pattern is determined, and adjustment to weights of connections between the hidden layer and the output layer is computed. A similar computation, still based on the error in the output layer, is made for the connection weighs between the input and hidden layers. The procedure is repeated with each pattern pair assigned for training the network. Each pass through all the training patterns is called a cycle or an epoch. The process is then repeated as many cycles as needed until the error is within a prescribed tolerance.

4.3 Learning and Training

A neural network maps a set of inputs to a set of outputs. This nonlinear mapping can be thought of as a multidimensional mapping surface. The objective of learning is to mold the mapping surface according to a desired response, either with or without an explicit training process.

A network can learn when training is used, or the network can also in the absence of training. The difference between supervised and unsupervised training is that, in supervised training, external prototypes are used as target output for specific
inputs, and the network is given a learning algorithm to follow and calculate new connection weights that bring the output closer to the target output. Unsupervised learning is the sort of learning that takes place without a teacher. Here learning algorithm

may be given but target outputs are not given. In such a case, data input to the network gets clustered together; similar input stimuli cause similar responses.

When a neural network model is developed and an appropriate learning algorithm is proposed, it would be based on the theory supporting the model. The learning equations are initially formulated in terms of differential equations. After solving the differential equations, and using any initial conditions that are available, the algorithm could be simplified to consist of an algebraic equation for the changes in the weights.

The delta rule is also known as the least mean squared error rule (LMS). We first calculate the square of the errors between the target or desired values and computed values, and then take the average to get the mean squared error. This quantity is to be minimized. For this, realize that it is a function of the weights themselves, since the computation of output uses them. The set of values of weights that minimizes the mean squared error is what is needed for the next cycle of operation of the neural network. So the delta rule, which is also the rule used first by Widrow and Hoff, in the context of learning in neural networks, is stated as an equation defining the change in the weights to be affected.

Suppose we fix our attention to the weight on connection between the ith neuron in one layer and jth neuron in the next layer. At time t, this weight is Wij(t). After one cycle of operation, this weight becomes Wij(t+1). The difference between the two is Wij(t+1) – Wij(t), and is denoted by ΔWij as :

$\Delta Wij = 2\mu Xi$ (desired output value – computed output value)j

Here, μ is the learning rate, which is positive and much smaller than 1, and Xi is the ith component of the input vector.

The actual derivations for the different formulas used in the backpropagation algorithm come from the generalized delta rule. The delta rule is based on the idea of the error surface. The error surface represents cumulative error over a data set as a function of the network weights. Each possible network weight configuration is represented by a point on this error surface. By taking the partial derivative of the network error with respect to each weight we will learn a little about the direction the error of the network is moving. In fact, if we take the negative of this derivative (i.e.the rate change of the error as the value of the weight increases) and then proceed to add it to the weight, the error will decrease until it reaches a local minimum. The taking of these partial derivatives and then applying them to each of the weights, takes place starting from the output layer to hidden layer weights, then, from the hidden layer to input layer weights [18].

Chapter 5

DATA SELECTION

5.1 Choosing the Output and Objective

Our objective is to forecast the S&P 500 ten weeks from now. The output we choose is the change in the S&P 500 from the current week to 10 weeks from now as a percentage of the current week's value.

5.2 Choosing the Inputs

The inputs to the network need to be weekly changes of indicators that have some relevance to the company index. Here we choose a data set that represents the state of the financial markets and the economy. The inputs chosen are listed as:

- Previous price action in the company index, including the close or final value of the index.
- Breadth indicators for the stock market, including the number of advancing issues and declining issues for the stocks in the New York Stock Exchange (NYSE).
- Other technical indicators, including the number of new highs and new lows achieved in the week for NYSE market. This gives some indication about the strength of an uptrend or downtrend.
- Interest rates, including short-term interest rates in the Three-Month Treasury Bill Yield, and long-term rates in the 30-year Treasury Bond Yield.

The input and output layers are fixed by the number of inputs and outputs we are using. In our case, the output is a single number, the expected change in the company index 10 weeks from now. The input size will be dictated by the number of inputs we have after preprocessing.

5.3 Preprocessing Data

Four substeps in the area of preprocessing [2]

- Highlight Features
- Transform
- Scale and bias
- Reduce Dimensionality

5.3.1 Highlighting Features in the Input Data

One should present the neural network, as much as possible, with an easy way to find patterns in our data. For time series data, like stock market prices over time, one may consider presenting quantities like rate of change as example. Other ways to highlight data is to magnify certain occurrences. For example, if you consider Central bank intervention as an important qualifier to foreign exchanges rates, then you may include as an input to your network, a value of 1 or 0, to indicate the presence or lack of presence of Central bank intervention.

5.3.2 Transform the data If Appropriate

For time series data, we may consider using a Fourier transform to move to the frequency-phase plane. This will uncover periodic cyclic information of it exists. The Fourier transform will decompose the input discrete data series into a series of frequency spikes that measure the relevance of each frequency component. If the stock market indeed follows the so-called January effect, where prices typically make a run up, then you would expect a strong yearly component in the frequency spectrum.

5.3.3 Scale your Data

Neurons like to see data in a particular input range to be most effective. if we present data like the S&P 500 that varies 200 to 550 will not be useful. We should choose data that fit a range that does not saturate, or overwhelm the network neurons. Choosing inputs from –1 to 1 or 0 to 1 is good idea.

to zero input will mean that the weight will not participate in learning. To avoid such situations, we can add a constant bias to our data to move the data closer to 0.5, where the neurons respond very well.

5.3.4 Reduce Dimensionality

We should try to eliminate inputs wherever possible. This will reduce the dimensionality of the problem and make it easier for neural network to generalize. Suppose that we have three inputs, x, y, z and one output, o. now suppose that we find that all of our inputs are restricted only to one plane. We could redefine axes such that we have x' and y' for the new plane and map your inputs to the new coordinates. This changes the number of inputs to our problem to 2 instead of 3, without any loss of information. Here we have so 22 fields in the raw data.

There are a couple of ways we can start **preprocessing** the data to reduce the number of inputs.

- Use Advances/Declines ratio instead of each value separately.
- Use New Highs/New Lows ratio instead of each value separately.

Finally we have following fields: see table 5.1

1. Three-Month Treasury Bill Yield
2. 30-Year Treasury Bill Yield
3. NYSE Advancing/Declining issues
4. NYSE New Highs/New Lows
5. Company closing price

Presently we have data available for the period from January 1980 to December 1992.

Date	3 MoTBills	30 yrTBonds	NYSE Adv./Dec	NYSE NewH/NewL	Closing Price
1/4/80	12.11	9.64	4.209459	2.764706	106.52
1/11/80	11.94	9.73	1.649573	21.28571	109.92
1/18/80	11.9	9.8	0.881335	4.210526	111.07
1/25/80	12.19	9.93	0.793269	3.606061	113.61
2/1/80	12.04	10.2	1.16293	2.088235	115.12
2/8/80	12.09	10.48	1.338415	2.936508	117.95
2/15/80	12.31	10.96	0.338053	0.134615	115.41
2/22/80	13.16	11.25	0.32381	0.109091	115.04
2/29/80	13.7	12.14	1.676895	0.179245	113.66
3/7/80	15.14	12.1	0.282591	0	106.9
3/14/80	15.38	12.01	0.690286	0.011628	105.43
3/21/80	15.05	11.73	0.486267	0.027933	102.31
3/28/80	16.53	11.67	5.247191	0.011628	100.68
4/3/80	15.04	12.06	0.983562	0.0117647	102.15
4/11/80	14.42	11.81	1.565854	0.0310345	103.79
4/18/80	13.82	11.23	1.113287	0.146341	100.55
4/25/80	12.73	10.59	0.849807	0.473684	105.16

Table 5.1: A sample of a few lines looks the following data.

5.4 Highlight Features in the Data

For each of the five inputs, we want use a function to highlight rate of change types of features. We will use the following function (Proposed by Jurik) for this purpose.

Where: input (t) is the input's current value and BA (t-n) is a five unit block average of adjacent values centered around the value n periods ago.

Below table 5.2 after doing the block averages

Example: BA3MOBills for 1/18/80 = (3MoBills(1/4/80) + 3MoBills(1/11/80) + 3MoBills(1/18/80) + 3MoBills(1/25/80) + 3MoBills(2/1/80)) / 5.

Date	ROC3 Mo	ROC3 Bond	ROC3 A/D	ROC3 H/L	ROC3SP C	ROC10 _3Mo	ROC10_B nd	ROC10_ AD	ROC10_ H/L	ROC10_ SP
1/4/80										
1/11/80										
1/18/80										
1/25/80										
2/1/80										
2/8/80	0.002238	0.030482	-0.13026	-0.39625	0.02924					
2/15/80	0.011421	0.044406	-0.55021	-0.96132	0.008194					
2/22/80	0.041716	0.045345	-0.47202	-0.91932	0.001776					
2/29/80	0.0515	0.069415	0.358805	-0.81655	-0.00771					
3/7/80	0.089209	0.047347	-0.54808	-1	-0.03839					
3/14/80	0.073273	0.026671	-0.06859	-0.96598	-0.03814					
3/21/80	0.038361	0.001622	-0.15328	-0.51357	-0.04203					
3/28/80	0.065901	-0.00748	0.766981	-0.69879	-0.03816	0.15732	0.048069	0.502093	-0.99658	-0.04987
4/3/80	-0.00397	0.005419	-0.26054	0.437052	-0.01753	0.111111	0.091996	-0.08449	-0.96616	-0.05278
4/11/80	-0.03377	-0.00438	0.008981	0.803743	0.001428	0.87235	0.069553	0.268589	-0.78638	-0.04964
4/18/80	-0.0503	-0.02712	-0.23431	0.208545	-0.01141	0.055848	0.030559	0.169062	-0.84766	-0.06888
4/25/80	-0.08093	-0.0498	-0.37721	0.58831	0.015764	0.002757	-0.01926	-0.06503	-0.39396	-0.04658

Table 5.2: Data after Highlight Feature

5.4.1 Normalizing the Range

We now have values in the original five data columns that have a very large range. we have to reduce the range by some method. We use the following function:

New value = (old value – Mean) /(Maximum Range)

5.4.2 The target

The objective is that predicts the percentage change 10 weeks into the future. We need to shift the S&P 500 10 weeks back, and then calculate the value as percentage change as follows:

Result = 100 * ((S&P 10 weeks ahead) – (S&P this week)) / (S&P this week)

This gives the value that varies between –14.8 to and + 33.7. This is not in the form we need yet. The output comes from a sigmoid function that is restricted to 0 to +1. We first add 14.8 to all values and then scale them by a factor of 0.02. This will result in a scaled target that varies from 0 to 1.

Scaled target = (result + 14.8) * 0.02

The final data file with the scaled target shown along with the scaled original six columns of data is shown in below table 5.3.

DATE	S_3MOBILL	S_LNGBND	S_A/D	S_H/L	S_STC	RESULT	SCALE TARGET
3/28/80	0.534853	-0.01616	0.765273	-0.07089	-0.51328	12.43544	0.544709
4/3/80	0.391308	0.055271	-0.06356	-0.07046	-0.49236	12.88302	0.55366
4/11/80	0.331578	0.009483	0.049635	-0.06969	-0.46901	9.89498	0.4939
4/18/80	0.273774	-0.09674	-0.03834	-0.07035	-0.51513	15.36549	0.60331
4/25/80	0.168365	-0.21396	-0.08956	-0.06903	-0.44951	11.71548	0.53031

Table 5.3: Normalized ranges for original columns and scaled target.

After getting the output we need to un-normalize the data back to get the answer in terms of the change in the S&P 500 index.

DATA PREDICTION AND IMPLEMENTATION DETAILS

In this chapter, we shall make an attempt to implement neural nets. The model that has been implemented as neural network for data prediction, which is based on backpropagation algorithm.

Algorithm:

(1) Initialize weights and offsets:

Set all weights and node biases to small random values.

(2) Present input and desired outputs:

Present a continuous valued input vector $\mathbf{x_0}$, $\mathbf{x_1}$, ..., $\mathbf{x_{N-1}}$ and specify the desired outputs $\mathbf{d_0}$, $\mathbf{d_1}$, ... , $\mathbf{d_{M-1}}$. If the net is used as a classifier, then all desired outputs are typically set to 0 except for that output corresponding to the class the input is from, which is set to 1. The input could be new on each trial

(3) Calculate actual outputs:

Use the following formulas to calculate outputs $O_1, O_2, \ldots, O_{M-1}$ of every neuron in the network.

$$O_i = f(\Sigma x_i w_i + b_i)$$

$$f(y) = 1 / (1 + e^{-\lambda * y})$$

Where **x** → input vector, **w** → weight vector denoting to weights linking the neuron unit to the previous neuron layer, **b** → bias vector, **λ** → squashing parameter.

(4) Adapt weights:

Use a recursive algorithm starting at the output nodes and working back to the first hidden layer. This has many sub-steps.

Step a: Compute the sum-squared error of the network.

$$\text{Error} = ½ \Sigma_{i \, \varepsilon \, \text{outputs}} (d_i - O_i)^2$$

Step b: Calculate the error term of each neuron in the output layer,

$$\delta_i = O_i (1 - O_i) (d_i - O_i)$$

Step c: Calculate the error term of each neuron in the hidden layer,

$$\delta_i = O_i (1 - O_i) \Sigma_j \delta_j w_{ij}$$

where, j is the index of the nodes in the next layer to which i's signals fan out.

Step d: Compute the weight deltas. η is the learning rate.

A low learning rate can ensure more stable convergence. A high learning rate can speed up convergence in some cases.

$$\Delta w_{ki} = \eta \, \delta_k \, x_{ki}$$

where w_{ki} is the weight from the hidden (or input) node **k** to node **i**.

Step e: Add the weight deltas to each of the weights

$$w_{ki}(t+1) = w_{ki}(t) + \Delta w_{ki}$$

where t denotes the iteration step.

(5) Repeat by going to step 2.

The following are definitions in the layer base class. Here number of inputs and outputs are protected data members, which means that they can be accessed by descendants of the class.

```
int num_inputs;
int num_outputs;
float *outputs;
float *inputs;
friend network;
```

A layer contains neurons and weights. The layer is responsible for calculating its output (calc_out()) , and errors (calc_error()) for each of its respective neurons. The input class does not have any weights associated with it and therefor is a special case. It does not need to provide any data members or function members related to errors or backpropagation. The only purpose of the input layer is to store data to be forward propagated to the next layer. With the output layer, there are a few more arrays present. First for storing backpropagated errors. There is a weight array and finally, for storing the expected values that initiates the error calculation process. The network class is used to set up communication channels between layers and to feed and remove data from the network. The network class performs the interconnection of layers by setting the pointer of an input array of a given layer to the output array of a previous layer. The network class is also responsible for setting the pointer of an output_error array to the back_error array of the next layer.

Adding momentum term sometimes results in much faster training is the addition of a momentum term. The training law for backpropagation implemented as:

Weight change = Beta * output_error * input + Alpha * previous_weight_change

The momentum term is an attempt to try to keep the keep the weight change process moving.

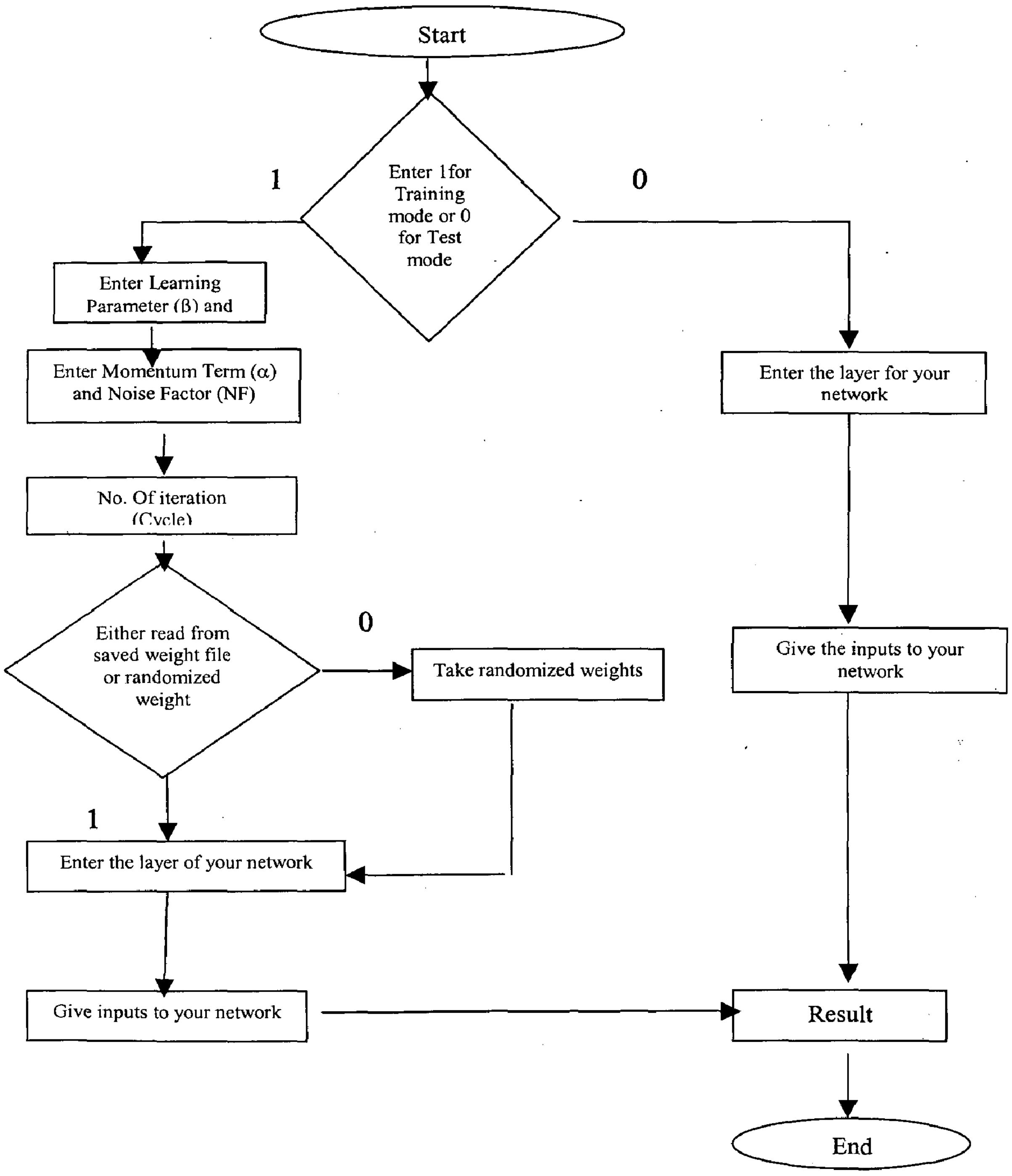
To enhance generalization ability introduces some noise in the inputs during training. A random number is added to each input component of the input vector as it is applied to the network. This is scaled by an overall noise factor, NF, which has a 0 to 1 range. We don't want noise at that time when we close to a solution and have reached a satisfactory minimum. Another reason for using noise is to prevent memorization by the network. We are effectively presenting a different input pattern with each cycle so it becomes hard for the network to memorize patterns.

At the top of the program, there are two **#define** statements, which are used to set the maximum number of layers that can be used, and the maximum number of training or test vectors that can read into an I/O buffer. This is currently 100. We can increase the size of the buffer for better speed at the cost of increased memory usage. Figure 6.1 show that the flow of the program.

The following is a listing of the functions used in program along with a brief statement.

- **void set_training (const unsigned &)** Sets the value of the private data member, training; use 1 for training mode , and 0 for test mode.
- **unsigned get_training_value()**Gets the value of the training constant that gives the mode in use.
- **void get_layer_info ()** Gets information about the number of layers and layer sizes from the user.
- **void set_up_network ()** This routine sets up the connections between layers by assigning pointers appropriately.
- **void randomize_weights ()** At the beginning of the training process, this routine is used to randomize all of the weights in the network.
- **void update_weights (const float)** As part of training, weights are updated according to the learning law used in backpropagation.
- **void write_weights(FILE *)** This routine is used to write weights to file.
- **void read_weights(FILE *)** This routine is used to read weights into the network from a file.
- **void list_weights()** This routine can be used to list weights while a simulation is in progress.
- **void write_outputs(FILE *)** This routine writes the outputs of the network to a file.
- **void list_outputs()** This routine can be used to list the outputs of the network while a simulation is in progress.
- **void list_errors()** Lists errors for all layers while a simulation is in progress.
- **void forward_prop()** Performs the forward propagation.
- **void backward_prop(float &)** Performs the backward error propagation.
- **int fill_Iobuffer(FILE *)** This routine fills the internal IO buffer with data from the training or test data sets.
- **void set_up_pattern(int)** This routine is used to set up one pattern from the IO buffer for training.
- **inline float squash(float input)** This function performs the sigmoid function.

- **inline float randomweight (unsigned unit)** This routine returns a random weight between –1 and 1; use 1 to initialize the generator, and 0 for all subsequent calls.



Start
Enter 1for Training mode or 0 for Test mode
1
0
Enter Learning Parameter (β) and
Enter Momentum Term (α) and Noise Factor (NF)
No. Of iteration (Cycle)
Either read from saved weight file or randomized weight
0
Take randomized weights
1
Enter the layer of your network
Give inputs to your network
Enter the layer for your network
Give the inputs to your network
Result
End

Figure 6.1: Flow Chart Showing the execution of program

RESULTS AND DISCUSSION

7.1 Training and Testing Results

Much of the process of determining the best parameter for this application is trial and error. We need to spend a great deal of time evaluating different options to find the best fit for our problem. We have to literally create hundreds if not thousands of networks either manually or automatically to search for the best solution. Many commercial neural network programs use genetic algorithms to help to automatically arrive at an optimum network. A genetic algorithm makes up possible solutions to a problem from a set of starting genes. Analogous to biological evolution, the algorithm combines genetic solutions with a predefined set of operators to create new generations of solutions, who survive or perish depending on their ability to solve the problem. The key benefit of genetic algorithms is the ability to traverse an enormous search space for a possibly optimum solution.

The numbers of inputs are 15, and the number of outputs is 1. A total of three layers are used with the middle layer of size 5. The optimum sizes and number of layers is found by much trial and error. After each run, we can look at the error from the training set and from the test set. We obtain the error for the test set by running the program in Training mode for one cycle with weights loaded from the weight file. This approach has been taken with five runs of the simulator for 500 cycles each.

The error gets lesser and lesser with each run up to run #4. For run#5, the training set error decreases, but the test set error increases, indicating the onset of memorization. Run#4 is used for the final network results, showing RMS error of 13.9% and training set error of 6.9%. If we find the test set error does not decrease much, whereas the training set error continues to make substantial progress, then this that memorization is starting to set in (run#5 in table 7.1).

Run#	Tolerance	Beta	Alpha	NF	Max cycles	Cycles run	Training set error	Test set error
1	0.001	0.5	0.001	0.0005	500	500	0.150938	0.25429
2	0.001	0.4	0.001	0.0005	500	500	0.114948	0.185828
3	0.001	0.3	0	0	500	500	0.0936422	0.148541
4	0.001	0.2	0	0	500	500	0.068976	0.139230
5	0.001	0.1	0	0	500	500	0.0621412	0.143430

Table 7.1: Summary of the results along with parameters used.

```
After training mode the output file shows
for input vector:
-0.198290 -0.061950 -0.092800 -0.047110 0.431060 -0.003910 0.000175 -0.171230
0.263442 0.006145 0.025126 0.004309 -0.094330 0.512339 0.007212
output vector is:
0.179065
expected output vector is:
0.174338
```

It is important to monitor the test set that are used, while we are training to make sure that good, generalized learning is occurring versus memorization of overfitting the data. In the table shown, the test set error continued to improve until run#5, where the test set error degraded.

After test mode the output file shows the following data. Here some of the data are shown because output file is too large.

```
for input vector:
-0.207920 -0.080260 -0.145350 -0.059460 0.419245 -0.008650 -0.005620 -0.358000
-0.228550 0.002479 0.025126 -0.000530 -0.349090 -0.053160 0.003975
output vector is:
0.208678
----------------------
for input vector:
```

```
-0.198290 -0.100410 -0.121130 -0.063570 0.387074 -0.001790 -0.009630 -0.228080
-0.466090 -0.006660 0.025126 -0.006030 -0.260190 -0.281140 -0.004120
output vector is:
0.202962
---------------------
for input vector:
0.534853 -0.016160 0.765273 -0.070890 -0.513280 0.065901 -0.007480 0.766981 -
0.698790 -0.038160 0.009582 0.084069 0.502093 -0.996580 -0.049870
output vector is:
0.566802
---------------------
for input vector:
-0.198290 -0.061950 -0.092800 -0.047110 0.431060 -0.003910 0.000175 -0.171230
0.263442 0.006145 0.025126 0.004309 -0.094330 0.512339 0.007212
output vector is:
0.179252
---------------------
```

Now we need to un-normalize the data back to get the answer in terms of the change in the S&P index.

Steps:

1. Take the predicted scaled target value and calculate, the result value as Result = (Scaled target/0.02) – 14.8
2. Take the result from above (which is the percentage change 10 weeks from now) and calculate the projected value, Projected S&P 10 weeks from now = (This week's S&P value) (1+ Result/100)

Example: In this case I got the Scaled target 0.551020 for the date 03/28/80(mm-dd-yy). Then for un-normalize put the value in step 1, after calculating I got the Result = 12.75. For calculating the projected value we put the result in step 2. Here I have taken the S&P closing price for the date 04-03-80, which is 102.15 and after 10 week which is 06-13-80 closing price is 115.31.so put the value in step 2, which follows:

Projected (04-03-80) S&P 10 weeks from now = (102.15) (1+ 12.75/100)

= 115.17

which is equivalent to (06-13-80) S&P projected value.

7.2 Traditional Statistical Approach Pitfalls

Proper evaluation is critical to a prediction system development. First, it has to measure exactly the interesting effect, e.g. trading return, as opposed to prediction accuracy. Second, it has to be sensitive enough as to distinguish often-minor gains. Third, it has to convince that the gains are no merely a coincidence.

7.2.1 Evaluate the right thing

Financial forecasts are often developed to support semi-automated trading (profitability), whereas the algorithms underlying those systems might have different objective. Thus, it is important to test the system performing in the setting it is going to be used, a trivial, but often missed notion. Also, the evaluation data should be of exactly the same nature as planned for real-life application, e.g. index-futures trading performed for index data used as a proxy for futures price, but real futures data degraded it. Some problems with common evaluation strategies follow.

Accuracy – percentage of correct discrete (e.g. up/down) predictions; common measure for discrete systems, e.g. ILP/decision trees. It values instances equally, disregarding both instance's weight and accuracy for different cases, e.g. a system might get high score predicting the numerous small changes whereas missing the big few. Actually, some of the best-performing systems have lower accuracy than could be found for that data.

Square error – sum of squared deviations from actual outputs – is a common measure in numerical prediction, e.g. ANN. It penalizes bigger deviations, however if sign is what matters this might not be optimal, e.g. predicting -1 for -0.1 gets bigger penalty than predicting +0.1, though the latter might trigger going long instead of short. Square error minimization is often an intrinsic part of an algorithm such as ANN Backpropagation, and changing it might be difficult. Still, many such predictors, e.g. trained on bootstrap samples, can be validated according to the desired measure and the best picked.

Reliability – predictor's confidence in it's forecast – is equally important and difficult to develop as the predictor itself. A predictor will not always be confident – it should be able to express this to the trading counterpart, human or not. e.g. by an output 'undecided'. No trade on dubious predictions is beneficial in many ways: lower errors, commissions, and exposure. Reliability can be assessed by comparing many predictions coming from an ensemble, as well as done in one step and multiple step fashion.

Performance measure should incorporate the predictor and the (trading) model it is going to benefit. Some points: Commissions need to be incorporated – many trading 'opportunities' exactly disappear with commissions. Risk/variability – what is the value of even high return strategy if in the process one gets bankrupt? Data difficult to obtain in real time, e.g. volume, might mislead historic data simulation [13].

7.2.2 Evaluation data

It should include different regimes, markets, even data errors, and be plentiful. Dividing test data into segments helps to spot performance irregularities (For different regimes). Overfitting a system to data is a real danger. Dividing data into disjoint sets is the first precaution: training, validation for tuning, and test set for performance estimation. A pitfall may be that the sets are not as separated as seem, e.g. predicting returns 5 days ahead, a set may end at day D, but that instance may contains return for day D+5 falling into a next set. Thus data preparation and splitting should be careful. Another pitfall is using the test set more than once. Here, 1 out of 20 trials is 95% above average, 1 out of 100, 99% above etc. In multiple test, significance calculation must factor that in, e.g. if 10 tests are run and the best appears 99.9% significant, it really is 99.9%10 = 99%. Multiple use can be avoided, for the ultimate test, by taking data that was not available earlier. Another possibility is to test on similar, not tuned for, data – without any tweaking until better results, only with predefined adjustments for the new data, e.g. switching the detrending preprocessing on.

Non/Parametric tests, most statistical tests have preconditions. They often involve assumptions about sample independence and distributions – unfulfilled leading to unfounded conclusions. Independence is tricky to achieve, e.g. predictors trained on overlapping data are not independent. If the sampling distribution is unknown, as it usually is, it takes least 30, better 100, observations for normal distribution statistics.
If the sample is smaller than 100, non/parametric test are preferable, with less scope for assumption errors. The backside is they have less discriminatory power – for the same sample size. A predictor should significantly win (non/parametric) comparisons with naive predictors:

1) Majority predictor outputs the commonest value all the time, for stocks it could be the dominant up move, translating into the buy and hold strategy.
2) Repeat previous predictor for the next value issues the (sign of the) previous one.

Sanity checks involve common sense. Prediction errors along the series should not reveal any structure, unless the predictor missed something. Do predictions on surrogate (permuted) series discover something? If valid, this is the bottom line for comparison with prediction on the original series.

7.3 Benefits and limitations of Neural Network

7.3.1 Benefits

Most of the benefits in the articles depend on the problem domain and the NN methodology used. A common contribution of NN applications is in their ability to deal with uncertain and robust data. Therefore, NN can be efficiently used in stock markets, to predict either stock prices or stock returns. It can be seen from a comparative analysis that the Backpropagation algorithm has the ability to predict with greater accuracy than other NN algorithms, no matter which data model was used. The variety of data models that exist in the papers could also be considered a benefit, since it shows NNs flexibility and efficiency in situations when certain data are not available. It has been proven that NN outperform classical forecasting and statistical methods, such as multiple regression analysis and discriminant analysis. When joined together, several NNs are able to predict values very accurately, because they can concentrate on different characteristics of data sets important for calculating the output. Analysis also shows the great possibilities of NN methodology in various combinations with other methods, such as expert systems. The combination of the NN calculating ability based on heuristics and the ability of expert systems to process the rules for making a decision and to explain the results can be a very effective intelligent support in various problem domains [13].

7.3.2 Limitations

Some of the NN limitations are:

(1) NNs require very large number of previous cases

(2) "The best" network architecture (topology) is still unknown

(3) For more complicated networks, reliability of results may decrease

(4) Statistical relevance of the results is needed

(5) A more careful data design is needed.

The first limitation is connected to the availability of data, and some researchers have already proven that it is possible to collect large data sets for the effective stock market predictions. The limitation still exists for the problems that do not have much previous data, e.g. new founded companies. The second limitation still does not have a visible solution in the near future. Although the efforts of the researchers are focused on performing numerous tests of various topologies and different data models, the results are still very dependent on particular cases. The third limitation, concerning to the reliability of results, requires further experiments with various network architectures to be overcome. The problem with evaluating NN reliability is connected with the next limitation, the need for more complex statistical relevance of the results. Finally, the variety of data models shows that data design is not systematically analyzed. Almost every author uses a different data model, sometimes without following any particular acknowledged modeling approach for the specific problem. There are some other limitations, concerning the problems of evaluation and implementation of NN, that should be discussed in order to improve NN applications [13].

CONCLUSION

Large number of research is done and implemented by companies that are not published in scientific indexes analyzed. It can be concluded that:

(1) NNs are efficiency methods in the area of stock market predictions, but there is no "recipe" that matches certain methodologies with certain problems.

(2) NNs are most implemented in forecasting stock prices and returns, although stock modeling is very promising problem domain of its application.

(3) Most frequent methodology is the Backpropagation algorithm, but the importance of integration of NN with other artificial intelligence methods is emphasized by many authors.

(4) Benefits of NN are in their ability to predict accurately even in situations with uncertain data, and the possible combinations with other methods.

(5) End user must know all the concept of NN because the key to all applications though, is how we present and enhance data, and working through parameter selection by trial and error.

Limitations have to do with insufficient reliability tests, data design, and the inability to identify the optimal topology for a certain problem domain.

Finally, almost all emphasize the integration of NNs with other methods of artificial intelligence as one of the best solutions for improving the limitations. Since NNs are relatively new methods and still not adequately examined, they open up many possibilities for combining their methods with new technologies, such as intelligent agents, Active X, and others. Those technologies could help in intelligent collecting of data that includes searching, selecting, and designing the large input patterns. Furthermore, with its intelligent user interfaces, those methods could improve the explanation of NNs results and their communication with user. NNs researchers improve their limitations daily and that is the valuable contribution to their practical importance in the future.

REFERENCES

1. Aparicio, F. M. and Estrada, J. "Empirical Distributions of Stock Returns: European Securities Markets, 1990-95".
2. Chenoweth, T., Obradovic, Z. and Lee, S., "Embedding Technical Analysis into Neural Network Based Trading Systems." Applied Artificial Intelligence, vol. 10 1996.
3. De Bodt, E., Rynkiewicz, J. and Cottrell, M. (2001) "Some known facts about financial data." European Symposium on Artificial Neural Networks 2001 proceedings, Bruges (Belgium), April 2001.
4. De la Maza, M. and Yuret, D."A critique of the standard neural application to financial time series analysis". The Magazine of Artificial Intelligence in Finance, 1995.
5. Deboeck, G. J. "Financial Applications of Self-Organizing Maps". Neural Network World, Vol. 8,1998.
6. Deboeck, G. J. "Self-Organizing Maps Facilitate Knowledge Discovery in Finance." Financial Engineering News, January 1999.
7. Elaine Rich & Kevin Knight, Artificial Intelligence, TMH 1991
8. Fu, T. C., Chung, F. L., Ng, V. and Luk, R. (2001) "Pattern Discovery from Stock Time Series Using Self-Organizing Maps." Workshop Notes of KDD2001 Workshop on Temporal Data Mining, Aug., 2001.
9. Jones, C. P. Investments: Analysis and Management, 6th edition. John Wiley & Sons, New York 1998.
10. Kaski, S."Data Exploration using Self-organizing Maps." Acta Polytechnica Scandinavica, Mathematics, Computing and Management in Engineering Series, March 1997.
11. Pistolese, C. Using Technical Analysis, Revise Edition. Probus, Chicago 1994.
12. Seiler, M. J. and Rom, W. "A Historical Analysis of Market Efficiency: Do Historical Returns Follow a Random Walk?" Journal of Financial And Strategic Decisions, Vol. 10 1997.

13. Swingler, K. "Financial prediction: Some pointers, pitfalls and common errors." Neural Computing and Applications, Vol. 4 1996.

14. Yao, J. T., Tan, C. L. and Poh, H. L. "Neural Networks for Technical Analysis: a Study on KLCI." International Journal of Theoretical and Applied Finance, Vol. 2 1999.

15. Zekic, M. (1998) "Neural Network Applications in Stock Market Predictions. A Methodology Analysis."

16. Kohonen, T. (2001) Self-organizing Maps, 3rd edition. Springer, New York.

17. http://www.soi.city.ac.uk/~livantes/DATA/DataFinancial.html

18. Dr. Roger S Gaborski, Research Paper: Introduction to Artificial Intelligence, July 2001

APPENDIX A

Shows the training file, which is input plus the expected output. Here 1st fifteen fields are inputs and sixteen filed is expected field. Here only few inputs are given because training file is too large.

0.534853	-0.01616	0.765273	-0.07089	-0.51328
0.065901	-0.00748	0.766981	-0.69879	-0.03816
0.009582	0.084069	0.502093	-0.99658	-0.04987
0.544709				
0.391308	0.055271	-0.06356	-0.07046	-0.49236
-0.00397	0.005419	-0.26054	0.437052	-0.01753
0.008478	0.091996	-0.08449	-0.96611	-0.05278
0.55366				
0.331578	0.009483	0.049635	-0.06969	-0.46901
-0.03377	-0.00438	0.008981	0.803743	0.001428
0.011763	0.069553	0.268589	-0.78638	-0.04964
0.4939				
0.273774	-0.09674	-0.03834	-0.07035	-0.51513
-0.0503	-0.02712	-0.23431	0.208545	-0.01141
0.010664	0.030559	0.169062	-0.84766	-0.06888
0.60331				
0.168765	-0.21396	-0.08956	-0.06903	-0.44951
-0.08093	-0.0498	-0.37721	0.58831	0.015764
0.009566	-0.01926	-0.06503	-0.39396	-0.04658
0.53031				
-0.01813	-0.2451	-0.0317	-0.06345	-0.44353
-0.14697	-0.04805	-0.25956	0.795146	0.014968
0.008472	-0.0443	0.183309	0.468658	-0.03743
0.528241				

APPENDIX B

Shows the test file, which contains the input. Here only few inputs are given because test file is too large.

-0.20792	-0.08026	-0.14535	-0.05946	0.419245	-0.00865
-0.00562	-0.358	-0.22855	0.002479	0.025126	-0.00053
-0.34909	-0.05316	0.003975			
-0.19829	-0.10041	-0.12113	-0.06357	0.387074	-0.00179
-0.00963	-0.22808	-0.46609	-0.00666	0.025126	-0.00603
-0.26019	-0.28114	-0.00412			
-0.2031	-0.09674	-0.15712	-0.06887	0.344085	-0.0032
-0.00593	-0.35262	-0.82344	-0.01638	0.02	-0.00443
-0.41118	-0.72318	-0.01502			
-0.18287	-0.09125	-0.00139	-0.07065	0.278462	0.009225
-0.00257	0.201525	-0.97209	-0.02849	0.016431	-0.00336
0.155841	-0.95153	-0.02719			
-0.18673	-0.0766	-0.10979	-0.07025	0.270491	0.006569
0.002298	-0.02104	-0.85708	-0.02238	0.011407	-0.00062
-0.11894	-0.84148	-0.02785			
-0.17806	-0.05828	0.532249	-0.06966	0.295686	0.010179
0.007397	0.683779	-0.54675	-0.00976	0.006432	0.002629
0.619579	-0.46275	-0.02083			
-0.17131	-0.0363	0.148156	-0.06722	0.320313	0.010545
0.011551	0.174271	0.228072	0.001138	0.001506	0.006267
0.275451	-0.57734	-0.0177			
-0.16746	-0.01982	-0.13201	-0.0705	0.250704	0.009615
0.012339	-0.46621	-0.5711	-0.0115	0.029283	0.008745
-0.30682	-0.95299	-0.03446			
-0.15493	0.003989	0.253694	-0.0672	0.32074	0.012535
0.014206	0.195437	0.487841	0.008363	0.029283	0.015167
0.410297	-0.68825	-0.02108			

APPENDIX C

Shows the blind test data.

-0.20792	-0.08026	-0.14535	-0.05946	0.419245	-0.00865
-0.00562	-0.358	-0.22855	0.002479	0.025126	-0.00053
-0.34909	-0.05316	0.003975			
-0.19829	-0.10041	-0.12113	-0.06357	0.387074	-0.00179
-0.00963	-0.22808	-0.46609	-0.00666	0.025126	-0.00603
-0.26019	-0.28114	-0.00412			
-0.2031	-0.09674	-0.15712	-0.06887	0.344085	-0.00326
-0.00593	-0.35262	-0.82344	-0.01638	0.02	-0.00443
-0.41118	-0.72318	-0.01502			
-0.18287	-0.09125	-0.00139	-0.07065	0.278462	0.009225
-0.00257	0.201525	-0.97209	-0.02849	0.016431	-0.00336
0.155841	-0.95153	-0.02719			
-0.18673	-0.0766	-0.10979	-0.07025	0.270491	0.006569
0.002298	-0.02104	-0.85708	-0.02238	0.011407	-0.00062
-0.11894	-0.84148	-0.02785			
-0.17806	-0.05828	0.532249	-0.06966	0.295686	0.010179
0.007397	0.683779	-0.54675	-0.00976	0.006432	0.002629
0.619579	-0.46275	-0.02083			
-0.17131	-0.0363	0.148156	-0.06722	0.320313	0.010545
0.011551	0.174271	0.228072	0.001138	0.001506	0.006267
0.275451	-0.57734	-0.0177			

APPENDIX D

Shows the weight file.

```
1 -14.788936 -23.214375 7.410563 2.342502 6.208565
1 -1.495994 0.073311 6.112903 -7.998254 4.467839
1 5.144179 3.322052 0.913615 -1.913263 -1.645980
1 0.942062 -0.017056 0.429377 3.644783 -0.047803
1 4.951530 5.664355 9.940310 0.947816 7.236715
1 5.216183 12.293064 4.796335 -1.135227 3.514790
1 10.004666 3.738953 6.464227 -0.903403 -0.364508
1 1.161117 3.658407 -0.991595 2.104930 0.194777
1 -0.558064 -0.322594 1.101481 0.055280 0.823635
1 -11.754412 6.081999 0.016447 0.421185 0.439200
1 7.767948 -12.617767 -4.839843 -1.803454 5.027760
1 -0.251463 0.286134 3.102932 -0.288077 -2.275205
1 -1.353341 -2.198585 0.337839 3.307456 0.583523
1 -1.648586 -2.483641 3.405509 0.646407 2.132873
1 10.865354 -1.654573 -0.245387 -3.177217 5.383827
2 -11.801553
2 11.343361
2 3.372332
2 0.518942
2 -5.035735
```

APPENDIX E

After the test mode the output file looks like this, here are some of the data.

```
for input vector:
-0.207920 -0.080260  -0.145350  -0.059460  0.419245 -0.008650  -0.005620
-0.358000  -0.228550  0.002479  0.025126  -0.000530  -0.349090  -0.053160
0.003975
output vector is:
0.208678
----------------------
for input vector:
-0.198290 -0.100410 -0.121130  -0.063570  0.387074  -0.001790 -0.009630
-0.228080 -0.466090 -0.006660  0.025126 -0.006030  -0.260190 -0.281140
-0.004120
output vector is:
0.202962
----------------------
for input vector:
0.534853  -0.016160  0.765273  -0.070890  -0.513280  0.065901  -0.007480
0.766981  -0.698790  -0.038160  0.009582  0.084069  0.502093  -0.996580
-0.049870
output vector is:
0.566802
----------------------
for input vector:
-0.198290  -0.061950  -0.092800 -0.047110  0.431060  -0.003910  0.000175
-0.171230  0.263442  0.006145 0.025126  0.004309  -0.094330  0.512339
0.007212
output vector is:
0.179252
----------------------
```