FOREX PREDICTION USING AN ARTIFICIAL INTELLIGENCE SYSTEM

By

JINXING HAN GOULD

Bachelor of Science

Beijing University

Beijing, China

1983

Submitted to the Faculty of the
Graduate Collage of the
Oklahoma State University
In partial fulfillment of
The requirements for
The Degree of
MASTER OF SCIENCE
December, 2004

FOREX PREDICTION USING ARTIFICIAL

INTELLIGENT SYSTEM

Thesis Approved:

Dr. George E. Hedrick

Thesis Advisor

Dr. Debao Chen

Dr. Johnson Thomas

Dr. A. Gordon Emslie

Dean of the Graduate Collage

ACKNOWLEDGMENTS

I thank Dr. Hedrick, my advisor, for giving me the opportunity and support to pursue my research towards the completion of this project. I also thank Dr. Hedrick, Dr. Thomas and Dr. Cheng for participating in my thesis committee and for reviewing my work.

I greatly appreciate all the support I received from my husband, Ronald Gould; not only for all the love and care he gave, but also for many valuable discussions in board areas, especially language.

I wish to thank my parents for their continuous support in my academic endeavors and deep understanding of my living and studying abroad. It is their endless love that makes my life a happy journey.

I dedicate this thesis to my loving husband, Ronald Gould.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

LIST OF SYMBOLS

AI	Artificial Intelligence
ANN	Artificial Neural Networks
BP	Back-Propagation
CART	Classifications and Regression Trees
EANN	Evolutionary Artificial Neural Networks
FOREX	Foreign Currency Exchange
GA	Genetic Algorithm
Lr	Learning Rate
MLP	Multi Layer Perceptron
MARS	Multivariate Adaptive Regression Splines
MSE	Mean Square Error
RMES	Root Mean Square Error
RNN	Recurrent Neural Network
RW	Random-Walk

LIST OF FORMULA

Formula **Page**

CHAPTER 1

INTRODUCTION

FOREX (Foreign Currency Exchange) is concerned with the exchange rates of foreign currencies compared to one another. These rates provide significant data necessary for currency trading in the international monetary markets. FOREX rates are impacted by a variety of factors including economic and political events, and even the psychological state of individual traders and investors. These factors are correlated highly and interact with one another in a highly complex manner. Those interactions are very unstable, dynamic, and volatile. This complexity makes predicting FOREX changes exceedingly difficult. The people involved in the field of international monetary exchange have searched for explanations of rate changes; thereby, hoping to improve prediction capabilities. It is this ability to correctly predict FOREX rate changes that allows for the maximization of profits. [25] Trading at the right time with the relatively correct strategies can bring large profit, but a trade based on wrong movement can risk big losses. Using the right analytical tool and good methods can reduce the effect of mistakes and also can increase profitability.

Artificial Intelligence (AI) systems are systems designed for detecting knowledge in data without human interruptions. The systems are trained to create models of the underlying data making computers become more self-programming rather than implementing program structures for computers to execute. An AI system should be able

to generate it's own rule sets, decision strategies and numerical values based on input data and targets defined by a human being [20]. Such AI systems must develop successful behavior just as successful human traders who use market analysis and combine it with their knowledge and skills. With the advances made in computer technology, artificial intelligence can overcome some of the limitations of humankind, such as lower rates of performance, less efficiency and slower communication.

One popular technique of implementing an AI system for predictions of financial market (*e.g.,* foreign currency exchange) performance is Artificial Neural Networks (ANN). ANN is actually an information processing system that consists of a graph representing the processing system as well as various algorithms. ANN is a complex and sophisticated computer program. It is able to adapt, to recognize patterns, to generalize, and to cluster or to organize data. The ANN operation has been used to advise trading of FOREX to increase profitability. Today ANN can be trained to solve problems that are difficult when using conventional programming techniques or through the efforts of human beings.

In order for ANN to recognize patterns in the data, it is necessary for the neural network to "learn" the structure of the data set. Learning is accomplished by providing sets of connected input/output units where each connection has a weight associated with it. The ANN learns by adjusting the weights so that the application of a set of inputs produces the desired set of outputs. [3]

There are several advantages in using ANN for the analysis of data. First, once neural networks are trained, their performance does not significantly degrade when presented with data not encountered during the initial learning phase. Second, they

handle poor quality data very well. Such problems with data may occur from measurement errors or for other reasons. Third, it isn't necessary to make strong assumptions about the data presented to a neural network in the same way traditional statistical techniques require. Fourth, the neural network does not require the user to decide the importance of variables because the network itself will make those decisions. Using ANN to predict foreign exchange rates has a large potential for profit returns, if it is successful. [3] Therefore, the use of ANN might be able to predict the fluctuations for foreign currency rates and are much more efficient than other systems. For this reason, the decision was made to use ANN for this project.

CHAPTER 2

DESIGNING A NEURAL NETWORK FORECASTING MODEL

The background information used for this chapter was taken from Kaastra and Boyd [15]. The back-propagation neural network is the most widely used in financial time series forecasting. Standard back-propagation is a gradient descent algorithm. It was created using the Windrow-Hoof learning rule. The rule is the network weights and biases are updated or generated in the direction of the negative gradient of the performance function [8]. The performance function is measured by MSE (mean square error)---the average squared error between the network outputs and the target outputs. This network can be used to approximate a general function. It can approximate equally well any function with a finite number of discontinuities.

2.1 Variable selection

Understanding the problem that one must resolve is very important. The variables of the selected raw data are more numerous than is needed. Choosing the variables that are important to the market being predicted is critical. The frequency of data selection (*e.g.,* monthly or daily) depends on the goals of the researcher.

2.2 Data collection

The cost and quality of the data should be considered when collecting data for the variables chosen. Four issues need to be considered in the process of data selection; (1) the method of calculation, (2) data that cannot be modified retroactively, (3) an appropriate delay of the data, and (4) assurance that the source will continue to provide data in the future.

2.3 Data preprocessing

Data preprocessing is the process to assist the neural network in learning the data patterns. The input and output variables rarely are fed into the network in raw form; the process refers to transforming the input and output variables to minimize noise, to highlight important relationships, to flatten the variable distribution, and to detect trends. The raw data is usually scaled between 0 and 1 or –1 and +1, so it is consistent with the type of transfer function that is being used.

2.4 Training, testing and validation sets

Generally the process is to divide the time series (input data) into three parts called training, testing and validation (out of sample) sets.

The training set is the largest set and is used by the neural network to learn the patterns of the data.

The size of the testing set ranges from 10 percent to 30 percent of the training set. It is used to test the network models so that the researcher can select the model with the best performance. The testing set can be selected randomly either from the training set or

from a set of data immediately following the training set. A more rigorous approach in evaluating a neural network is the walk-forward testing routine (the training-testing-validation sets are overlapped) because it attempts to simulate real life trading. The validation data set is used to make a final check on the performance of the trained neural network. Validation sets should be selected from the most recent contiguous observations.

2.5 Neural network structure

There are infinite ways to build a neural network. Some properties of a network must be defined for each new network. For example, the number of input neurons, the number of hidden layers, the number of hidden neurons, the number of output neurons, and the transfer functions.

Input neurons represent independent variables. In practice, one or two hidden layers are used widely and have very good performance. One hidden layer is sufficient because increasing the number of hidden layers increases the danger of over-fitting. Selecting the best number of hidden neurons is dependent on the complexity of the problem. It involves experimentation. The rule is "always to select the network that has the best performance on the testing set with the least number of hidden neurons [15]." The three methods most often used are fixed, constructors and distracters. Since multiple outputs produce degraded results, a single neuron is often used. The transfer function is to prevent outputs from getting very large values that can disable neural networks. Most neural network models use the sigmoid, tangent or linear functions. The sigmoid function is used commonly for financial markets time series data, which is nonlinear and keeps changing. Depending on the problem, other functions also can be used.

2.6 Evaluation measure

In a financial trading system, neural network forecasting would be converted into buy/sell signals according to the analysis. The general neural network measurement is the error sum of squares. Low forecast errors and trading profits are not necessarily identical because one large trade predicted by the neural network could have caused most of the trading system's profits. Filtering the time series to eradicate many smaller price changes may increase the profitability of the neural network. Additionally, it has been suggested by Kaastra and Boyd [15] that neural networks may be useful if the time series behave more like counter trend systems that do not follow the patterns of more traditional forecasting methods.

2.7 Neural network training

The neural network training process uses a training algorithm that adjusts the weights to reach the global minimum error. The process requires a set of examples of proper network behavior—network inputs P and target outputs T. During training the weights and biases of the network are adjusted to minimize the network performance function. Either the training is stopped, or how many training iterations should be processed must be determined *a priori*. Both the convergence approach and the train-test approach are used to prevent over-fitting or over-training. Over-training occurs when the error on the training set is driven to a very small value, but when the new data is presented to the network the error is large. Training also is affected by many other parameters such as the choice of learning rate and momentum value. The network adjusts the weights between

neurons based on the learning rule, and the learning rate is the size of the change of the weight which could control the training speed, *e.g.,* increase the learning rate could speed up training time. The momentum term determines how past weight changes affect current weight changes. McClelland and Rumelhart (in Kaastra and Boyd [15]) indicate that the momentum term is especially useful in error spaces such as steep, high wall and sloping floor. If the error space is sloping floor, without a momentum term, a very small learning rate would be required to move down the floor of the ravine. It will take excessive training time. By dampening the oscillations, the momentum term can allow a higher learning rate to be used to speed up the training time.

Using artificial intelligence systems in predicting financial markets also combines many other kinds of techniques to develop successful behaviors similar to successful methods used by human traders; *e.g.,* various algorithms and network structures. There is no way to make 100 percent accurate predictions, so risk management analysis based on expert knowledge also is needed [21].

CHAPTER 3

LITERATURE REVIEW OF PRECEDING WORK

Piche [18] uses a trend visualization plot on a moving average oscillator. For example, he uses an exponential moving average oscillator method to compute the fractional returns and then uses the trend visualization algorithm to plot the trend visualization matrix. By setting different parameters on the currency exchange rates of various national currencies, the results show this method is useful in gaining insight into other aspects of the market.

Staley and Kim [23] have suggested that interest rates are the most important variable determining the currency exchange rates; "self-fulfilling" behavior may also contribute to the movements in the rates. Therefore, they use two inputs: One relates to the changes in interest rates, and the other is the short-term trend in the exchange rate to search for patterns in the data. They indicate the model could be improved if more variables were added and the results were tested. Additionally, confidence regions (or error bars) could be added to the predictions so more appropriate validation sets could be chosen. This information could suggest whether on not the prediction should be applied on any given day.

Demster, Payne, Romahi and Thompson [7] have shown two learning strategies based on a genetic (programming) algorithm (GA) and reinforcement learning, and on two simple methods based on a Markov decision problem and a simple heuristic technique. All methods generate significant in-sample and out-of-sample profit when transaction

costs are zero. The GA approach is superior for nonzero transaction costs. They also state that when in-sample learning is not constrained, then there is the risk of over-fitting.

Chen and Teong [4] use a simple neural network to improve regular technical analyses. The result of using a neural network not only enhances profitability but also turns losing systems into profitable ones. This provides one with the opportunity to enter and exit trades before a majority of other traders do. A neural network is also able to adapt itself to new patterns emerging in the market place. This is important because currency market conditions change very rapidly.

Refenes, Azema-Barac and Karoussos [22] demonstrate that by using a neural network system and an error back-propagation algorithm with hourly feedback and careful network configurations, short term training can be improved. Feedback propagation is a more effective method of forecasting time series than forecasting without a feedback neural network. They also considered the impact of varying learning times and learning rates on the convergence and generalization performance. They discovered that multi-step predictions are better than single-step predictions as well as the fact that appropriate training set selection is important.

Laddad, Desai and Poonacha [16] use a multi-layer perceptron (MLP) network for predicting problems. The raw data contained a considerable volume of noise so they decomposed the data into many less complex time series data and used a separate MLP to learn each of them. Another method is to use two new weight initialization schemes. Both methods provide faster learning and improved prediction accuracy. The authors use the Random Optimization Algorithm rather than back-propagation because it gives faster learning and a smaller mean squared error.

Chen and Lu [6] compare the performance of eight artificial neural networks (ANN), eight Evolutionary ANNs (EANN) and the random-walk (RW) model. They found that all neural network models could generally outperform the RW model. The EAAN is the best model within a large search space. Since predicting the complex nonlinear exchange rate is impacted by different elements, it is desirable to have an independent search mechanism such as an Evolutionary ANN.

Giles, Lawrence and Tsoi [11] use hybrid neural network algorithms for noisy time series prediction, especially for small samples and high noise data. By using inference and extraction, recurrent neural network (RNN) models are suited to the detection of temporal patterns compared with standard multi-layer perceptron neural networks. They also provide insight into both the real world system and the predictor.

Green and Pearson [12] used artificial intelligence tools including a neural network and a genetic algorithm to build a model system for financial market decision support. They combined the Policy and Risk Decision Support tools with a neural network to build a cognitive model. The decision output is, therefore, a weighted formulation of probability and likelihood. The implementation with a hybrid of hardware and software produces seamless system integration.

Quah, Teh and Tan [19] indicate the hybrid system demonstrated its strengths for use as a decision support tool. The system simulates human logic, merging expert systems based on economic statistical figures and neural networks that have efficient learning processes. This system not only has a learning capability but also can handle the fuzzy and biased nature of human decision processes.

Ip and Wong [14] apply the Dempster-Shafer theory of evidence to the foreign exchange forecasting domain based on evidential reasoning. This theory provides a means for interpreting the partial truth or falsity of a hypothesis and for reasoning under uncertainties. Within the mathematical framework of the theory, evidence can be brought to bear upon a hypothesis in one of three ways: to confirm, to refuse or to ignore. Different factors affect the exchange rate at different degrees at different times. Various competing hypotheses are assigned to the factors under consideration. Some factors reflect the economy of a country. The economy in turn provides evidence for the movement of its currency. Based on historical data that implicitly record trends and other external factors, the system is able to evolve. The accumulation of more data regarding time-varying parameters and past performance hypotheses is reflected in the accuracy of future hypotheses.

White and Racine [24] use ANN to provide inferences regarding whether a particular input or group of inputs "belong" in a particular model. The test of these inferences is based on estimated feed-forward neural network models and statistical resampling techniques. The results suggest foreign exchange rates are predictable, but the nature of the predictive relation changes through time.

Ghoshray [10] used a fuzzy inferencing method on the fuzzy time series data to predict movement of foreign exchange rates. A fuzzy inferencing method uses one of the ingredients of chaos theory, which is the results of the previous iterations fed back repeatedly into the next one. He used fuzzy functions to express the dynamics of deterministic chaos. After certain steps any specific predicted value of the data vector could be obtained. He also found that fundamental analysis is useful in predicting long-

term trends but of little use in predicting short-term movements of exchange rates. Even though technical analysis can be useful in predicting short-term period changes, there is a lack of consistent accuracy. The author has examined several forecasting techniques, considered the behavior of time series data and advanced a fuzzy inferencing technique to predict future exchange rate fluctuations.

Iokibe, Murata and Koyama [13] use Takens' embedding theorem and local fuzzy reconstruction technology to predict short-term foreign exchange rates. The Takens' theory is that the vector $X(t) = (y(t), y(t-\tau), y(t-2\tau), \ldots\ldots. y(t-(n-1)\tau))$ is generated from the observed time series y(t), where "τ" is a time delay. The embedded latest data vector is replaced with $Z(T) = (y(T), y(T-\tau), y(T-2\tau), \ldots\ldots. y(T-(n-1)\tau))$. After one step is fetched, the data vector including this data is replaced with Z(T+1). The value of the time series data is predicted by the local fuzzy reconstruction method after "s" steps. This sequence is iterated up to the last data by setting dimensions of embedding (n=5) and a delay time of (τ=2) and the number of neighboring data vectors (N=5) on the currency exchange rate data of different countries. The satisfactory results have proven this method suffers less on irregular phenomenon governed by contingencies of the financial market. .

Muhammad and King [17] indicate fuzzy networks provide better general logic for modeling non-linear, multivariate and stochastic problems by using four layers; *i.e.,* using fuzzy input, fuzzy rules, normalizing and defuzzifying sequences. This method not only improves the root mean square error (RMSE) but also gives a good track of the actual change in the foreign exchange market.

Abraham [1] compares the performance and accuracy of neural networks, neural-fuzzy systems, Multivariate Adaptive Regression Splines (MARS), Classifications and Regression Trees (CART), and a hybrid MARS-CART technique for predicting the monthly average FOREX rates for the next month. The test results show the hybrid CART-MARS has the smallest Root Mean Square Error (RMSE), compared to the other techniques. It was determined that the hybrid model predicts the FOREX rates more accurately than all the other models when applied individually.

Abraham and Chowdhury [2] used four Gaussian membership functions for each input variable, built a feed-forward two hidden layers (6-14-14-1) model, modified the Adaptive Neural Fuzzy Inference System to accommodate the multiple outputs and adopted the mixture of back-propagation (BP) algorithms and least mean squares estimate to decide the consequent parameters. This model reveals that correction models can predict the average FOREX rates accurately one month in advance. By comparing the feed-forward ANN which uses the scaled conjugate gradient algorithm they found out that the Takagi-Sugeno type neuron-fuzzy system had better performance (35 seconds < 200 seconds) and smaller RMSE (0.0248 < 0.251) on predicting the average monthly FOREX rates of the Australian dollar with respect to the Japanese yen, U.S. dollar, U.K. pound, Singapore dollar and the New Zealand dollar.

Carney and Cunningham [3] indicate that neural network models are better than the conventional model methods in predicting foreign exchange rates on Deutsche marks (DEM), British pounds (GBP), Swedish krona (SEK) and U.S. dollars (USD). They used two methods; single-step predictions and multi-step predictions. Additionally, they discovered multi-step models had more accurate predictions than the single-step models.

Chen and Leung [5] combine the advantages of neural networks and multivariate econometric models to predict one-month-ahead exchange rates. The results from a number of tests and different statistical measures show that the hybrid approach not only produces better exchange rate predictions but also produces higher investment returns compared to the single-stage econometric model, the single-stage regression neural network model, and the random walk model. The effect of risk conversion is also considered.

Yao and Tan [25] consider time series data and simple technical indicators, such as moving average, are fed to the neural networks to catch the movement in currency exchange rates between American dollars and five other major currencies. The authors indicate that without using extensive market data, useful predictions can be made and a paper profit can be achieved for out-of-sample data with a simple technical indicator. They also point out trading strategies need to be considered. In this example "buy-and-hold" strategy is better than "trend-follow".

CHAPTER 4

PROJECT OVERVIEW

One of the current challenges in time series forecasting approaches is in the area of financial time series. It is a function approximation problem. Pattern recognitions are performed on the monthly foreign currency exchange rate data to predict the future exchange rate. The future value is predicted by using time series analysis or regression techniques in this project. Regression involves the learning of the function that does this mapping. Regression assumes that the target data fits into some known type of function (*e.g.,* linear or logistics) then discerns the best function that models the given data. An appropriate type of error analysis (MSE, for example) is used to determine which function is the most efficient.

We use neural networks to recognize patterns within the data by adjusting the weights and biases so that the set of inputs generates the desired set of outputs.

4.1 Data description

We use the monthly time series of foreign currency exchange rate between Australian dollars with the currencies of other countries (*e.g.,* U.S. dollars, Japanese yen, Chinese yuan). The goal of this paper is to predict future FOREX rates. The data being used for training and testing is from 1969 to 1989 monthly foreign currency exchange rates. There are two types of basic input vectors. One is concurrent data and the other one is sequential data. Time series is sequential data. In this project the input data has two

dimensions. One is year and the other is month. The output data has one dimension, which is the exchange rate.

4.2 Data preprocessing

The utilized data has three columns, which are years, months and exchange rates. Thus the larger value input vector (e.g., year as compared to month) compares the month values and exchange rates value. This can lead to changes in the weights and biases that take a longer time for a much smaller input vector to overcome.

The regular rule for updating weighting is:

$$\Delta W = (T- A) P' = E P' \quad (1)$$

Where W is weight. T is target. A is output. E is error. P is input.

As shown above, the larger an input vector P, the larger is its effect on the weight vector W. Thus if an input vector is much larger than other input vectors, the smaller input vectors must be presented many times to have an effect.

The solution is to normalize the data; that is, compress the data into a smaller arrangement. One of the functions for normalizing data is as follows:

$$X n = (X – X \min) / (X \max – X \min) \quad (2)$$

This compresses the data into an arrangement between 0 and 1. From the author's experience in putting raw data and normalized data into the network for training, the performance of the network training will be more efficient after preprocessing the network data.

Fortunately, Matlab [8] has a built-in function, premmx, to carry out this operation. It scales the inputs and targets so that their values can fall in the range [-1, 1]. For this

project, the input data is years and months. The target data are currency exchange rates. We use the following Matlab command to preprocess (normalize) the input matrix P and target matrix T:

[Pn, minP, maxP, Tn, minT, maxT] = premm[P, T];

Now the original input data P and target T are normalized to input Pn and target Tn such that all fall in the interval [-1, 1]. The vector minP and maxP are the minimum and maximum value of the original P. The vector minT and maxT are the minimum and maximum value of the original T. Next Pn and Tn are put into the network to train it. After the network has been trained, these vectors should be used to transform any future input applied to the network. Subsequently, the post processing, postmmx function, is used to convert the output to the same units as the original target.

4.3 Partitioning of the data

In this project 70% of the data will be used for training and 30% of the data will be used for testing.

4.4 Creating the neural network

This section presents the architecture of the multi-layer feed-forward network. It is most commonly used with a back-propagation algorithm.

The basic type of connectivity of feed-forward networks is the connections are only to later layers in the structure. The network will use a two-layer network because, based on this project's research, a larger and more complicated neural network can cause over-fitting which occurs when a neural network is trained to fit one set of data almost exactly

but results in a very large error when new data is tested [15]. Therefore, smaller neural networks are recommended by the author.

4.5 Choosing transform functions

Each input vector X is weighted with an appropriate matrix W, which is the dot product of the matrix W with the input vector X. The bias b is summed with the weighted input and put into the transfer function f. The transfer function takes the output, which may have any value between plus and minus infinity, and compresses the output into the range between 0,1 or between –1,1. The range depends on which transfer function is chosen. The output of the node is $Yi = fi (\Sigma Wij Xji + bi)$. An additional reason to use a transfer function is to prevent noisy inputs from impacting the network analysis. There are many suggested transfer functions, including threshold, sigmoid, symmetric and Gussian. The Matlab Toolbox [8] has many commonly used transfer functions, including hard-limit, purelin, log-sigmoid and tan-sigmoid. We use tan-sigmoid as the hidden layer transfer function in our back-propagation network and use a linear function as the output layer transfer function. It is important for the back-propagation network to be able to calculate the derivative of any transfer function. Tan-sigmoid and purelin functions have corresponding derivative functions dtan-sinmiod and dpurelin. This is the reason they are chosen.

4.6 Creating a network for exchange rates from Australia to U.S. dollars

The first step for creating a network is to create the network object. The function (newff) creates a feed-forward network. The command shown below creates a two-layer

network. In this project there is one input vector with two elements; one element is year, which ranges from 1969 to 1989, the other one is month, arranged from 1 (January) to 12 (December). The hidden layer of this network has three neurons, and the output layer has one neuron. The Matlab code is as follows:

Net = newff ([1969, 1989 ; 1, 12], [3, 1], {'tangsig', 'purelin'}, traind)

The hidden layer transfer function is tansig, the output layer transfer function is purelin. The training function is traind. In the next section we present the training function.

This command creates a network object and also automatically initializes weights and biases of the network. The Matlab toolbox randomly assigns values to the weights and biases to initialize the network.

4.7 Training

4.7.1 Least mean square error (LMS) algorithm

Widrow and Hoff (in Demuth and Beale [8]) found that the mean square error could be estimated by using the squared error in each iteration. By deriving the squared error with respect to the weights and biases at the kth iteration, the change to the weight matrix and bias will be as follows:

$$W(k+1) = W(k) + 2\alpha E(k) P'(k) \quad (3)$$

$$b(k+1) = b(k) + 2\alpha E(k) \quad (4)$$

Here P (k) is the element of the input vector. E (k) is the mean square error. α is a learning rate which is related to the training speed and training stability. These two equations above form the basis of the Widrow-Hoff learning algorithm.

After the network is created and weights and biases are initialized, the network is ready for training for function approximation. The input matrix, P, and target vector, T, which is a sample of the function, are approximated, then sent to the network and through the training process. During the process, weights and biases of the network are repeatedly modified to minimize the mean square error. For the currency exchange rate prediction, this error also measures whether or not this network model is the best or the most accurate prediction for the problem.

4.7.2 Training function

Before the training process begins, the training function must be chosen. The training process is a procedure for modifying the weights, W, and biases, b, of the network. This procedure also is referred to as a training algorithm. The choosing of a training algorithm is very important for building the best possible model for an individual problem. It impacts the accuracy of prediction and the network performance. In this project, different algorithms are tested to find the best model for each problem and to illustrate how some of them can optimize network performance.

This project uses a supervised learning rule in which the weights and biases are modified according to the error, E, the difference between predicted target, T, and the network output, A. The formula is as follows:

$$\Delta W = (T - A) P' = E P' \quad (1)$$

$$\Delta b = (T - A) = E \quad (5)$$

In this formula, P is the input vector. The practical mean square error algorithm is used to adjust weights and biases. We want to minimize the average of the sum of these errors so that the network generates more accurate predictions. This is critical. There are many different training algorithms for feed-forward networks. It is just this variety that allows for building different models to fit individual problems to improve model performance and accuracy.

The back-propagation neural network is the most frequently used in financial time series. This network uses a back-propagation algorithm. It is a gradient descent algorithm in which the networks are adjusted along the negative of a gradient of the MSE. The basic implementation of a back-propagation algorithm is that the network weights and biases are upgraded in the direction of the most rapidly decreasing MSE. One iteration of this algorithm formula is as follows:

$$X(k+1) = X(k) - \alpha(k)\, G(k) \quad (6)$$

Where X (k) is the current weight and bias vector. G (k) is the current gradient descent. The $\alpha(k)$ is the learning rate.

There are two different methods to implement the gradient descent algorithm; one is incremental mode and the other one is batch mode. In the incremental mode, the gradient is calculated and weights and biases are modified after each input, then sent into the network. In the batch mode, the gradient is computed and weights and biases are changed after all inputs are applied to the network.

CHAPTER 5

EXPERIMENTAL RESULTS AND DISCUSSION

5.1 Performance comparison with different algorithms

There are many parameters we can adjust to ascertain whether or not they improve performance. In this project we only explore the concept of the accuracy and speed of training. This aspect of performance is very important because in the real world, training a huge and complex set of data can take hours, even days, to attain results.

There are many variations of back-propagation training algorithms provided in the Matlab Toolbox [8]. We will test the following algorithms to verify the performance of each:

- Batch gradient descent (traingd)
- Batch gradient descent with momentum (traingdm)
- Variable learning rate back-propagation (traingad)
- Resilient back-propagation (trainrp)
- Conjugate gradient (traincgf, traincgp, traincgb, trainscg)
- Quasi-Newton (trainbfg, trainoss)
- Levenberg-Marquardt (trainlm)

5.1.1 Batch gradient descent algorithm (traingd)

The training function for the batch gradient descent algorithm is traingd. The gradients are computed at each training input and added together to determine the changes in the weights and biases. The weights and biases are modified in the direction of the negative gradient of the performance function (MSE). We use the batch steepest descent training function, traingd, to train the network.

We adjust the parameters associated with the traingd function as follows:

Epochs: The number of the iterations processed before the training is stopped

Show: Shows the training results of every iteration of the algorithm

Goal: The minimum MSE to achieve before the training is stopped

Time: The training time

Min_grad: The minimum gradient before the training is stopped

Max_fail: The early stopping of the training technique

lr: lr is multiplied times the negative of the gradient to determine how much to change the weights and biases.

The system automatically assigns the values for each parameter as a default value.

The two dimensions raw input matrix P_0 = [1969 1969 1989 1989

7 8 11 12]

The one dimension raw target matrix T_0 = [1.1138 1.1091 0.7815 0.7929]

After normalizing P_0 and T_0, the value of P and T are between 0 and 1.

The real input matrix Pn = [0 0 1 1

0.45454 0.54545 0.8182 0.9091]

The real target matrix Tn = [0.57988 0.57459 0.2063 0.2189]

The following command creates the feed-forward network by using the batch gradient descent training algorithm. At this point we may want to change some of the default training parameters. The following codes modify the training parameters:

```
Net = newff (minmax(Pn), [3,1], {'transig','purelin'},'traingd');
Net.trainparam.epochs = 300;
Net.trainparam.show = 50;
Net.trainparam.lr = 0.05;
Net.trainparam.goal = 1e-3;
```

Now, we train the network. By putting the parameter object, Net, input, Pn, and target, Tn, into the training function, the training process is carried out. The training code is as follows:

```
(Net, tr) = train (Net, Pn, Tn);
```

The tr is the training record and it contains the information about the training process. For example, it is used as a diagnostic tool to plot the training results. The training result is as follows:

```
TRAINGD, Epoch 0/300, MSE 0.689155/0.001, Gradient 2.52655/1e-010
TRAINGD, Epoch 50/300, MSE 0.0322265/0.001, Gradient 0.0539035/1e-010
TRAINGD, Epoch 100/300, MSE 0.0267678/0.001, Gradient 0.0401758/1e-010
TRAINGD, Epoch 150/300, MSE 0.0236098/0.001, Gradient 0.0311962/1e-010
TRAINGD, Epoch 200/300, MSE 0.0216549/0.001, Gradient 0.0248931/1e-010
TRAINGD, Epoch 250/300, MSE 0.020386/0.001, Gradient 0.0202641/1e-010
TRAINGD, Epoch 300/300, MSE 0.0195325/0.001, Gradient 0.0167518/1e-010
TRAINGD, Maximum epoch reached, performance goal was not met.
```

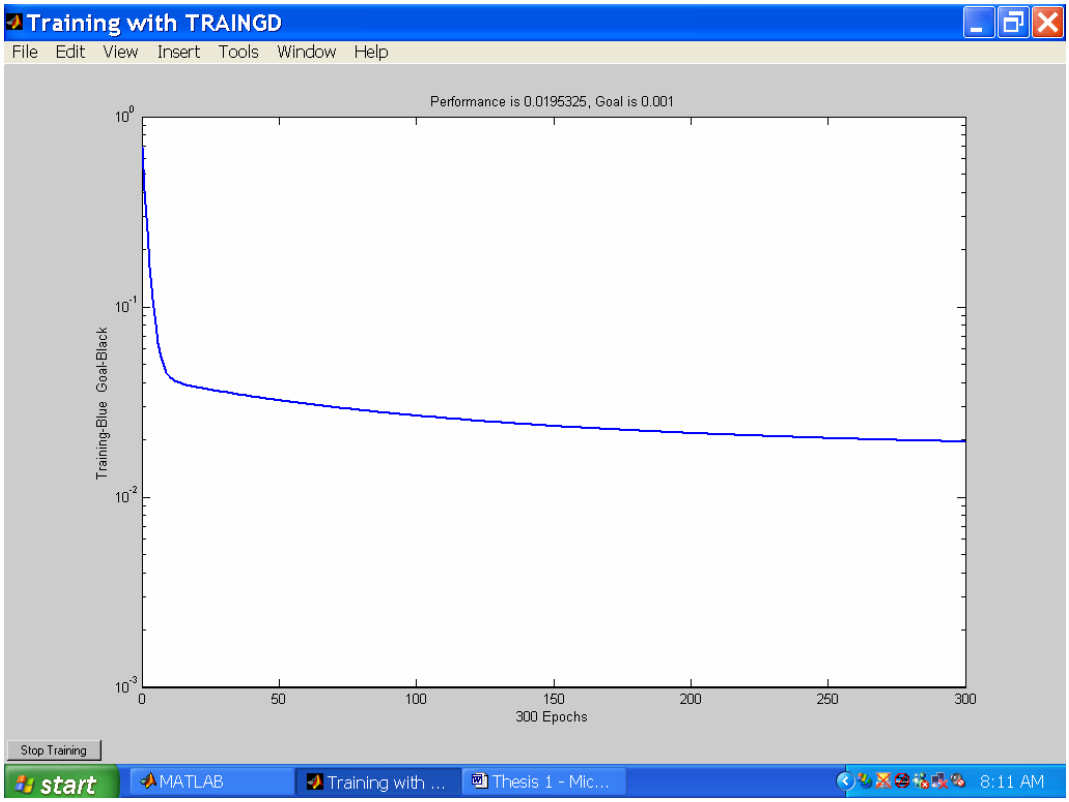

Figure 1. Training result of traingd

From Figure 1, we find the minimum MSE is 0.0195325 and estimated training time is more than 300 epochs since the MSE is still dropping.

5.1.2 Batch gradient descent with momentum (traingdm)

Momentum makes weight changes equal to the sum of a fraction of the last weight change and the new change suggested by the back-propagation rule. The momentum constant, mc, can be any number between 0 and 1. When mc equals 0, the new weight change can only rely on the gradient. When mc equals 1, the new weight change is set to the same as the last weight change and any gradient is simply ignored. In this project we set mc = 0.9, the code is net.trainParam.mc = 0.9 and the other parameters are the same as the batch gradient descent algorithm. The other commands are:

Net = newff (minmax(inputinit), [3,1], {‘transig’,’purelin’},’traingdm’);

(Net, tr) = train (Net, inputint, targetinit);

The training result is as follows:

TRAINGDM, Epoch 0/300, MSE 3.88657/0.001, Gradient 6.17719/1e-010

TRAINGDM, Epoch 50/300, MSE 0.0206684/0.001, Gradient 0.133611/1e-010

TRAINGDM, Epoch 100/300, MSE 0.016531/0.001, Gradient 0.0401495/1e-010

TRAINGDM, Epoch 150/300, MSE 0.0152145/0.001, Gradient 0.0179974/1e-010

TRAINGDM, Epoch 200/300, MSE 0.0145931/0.001, Gradient 0.0130241/1e-010

TRAINGDM, Epoch 250/300, MSE 0.0142507/0.001, Gradient 0.00988702/1e-010

TRAINGDM, Epoch 300/300, MSE 0.0140481/0.001, Gradient 0.00779517/1e-010

TRAINGDM, Maximum epoch reached, performance goal was not met.

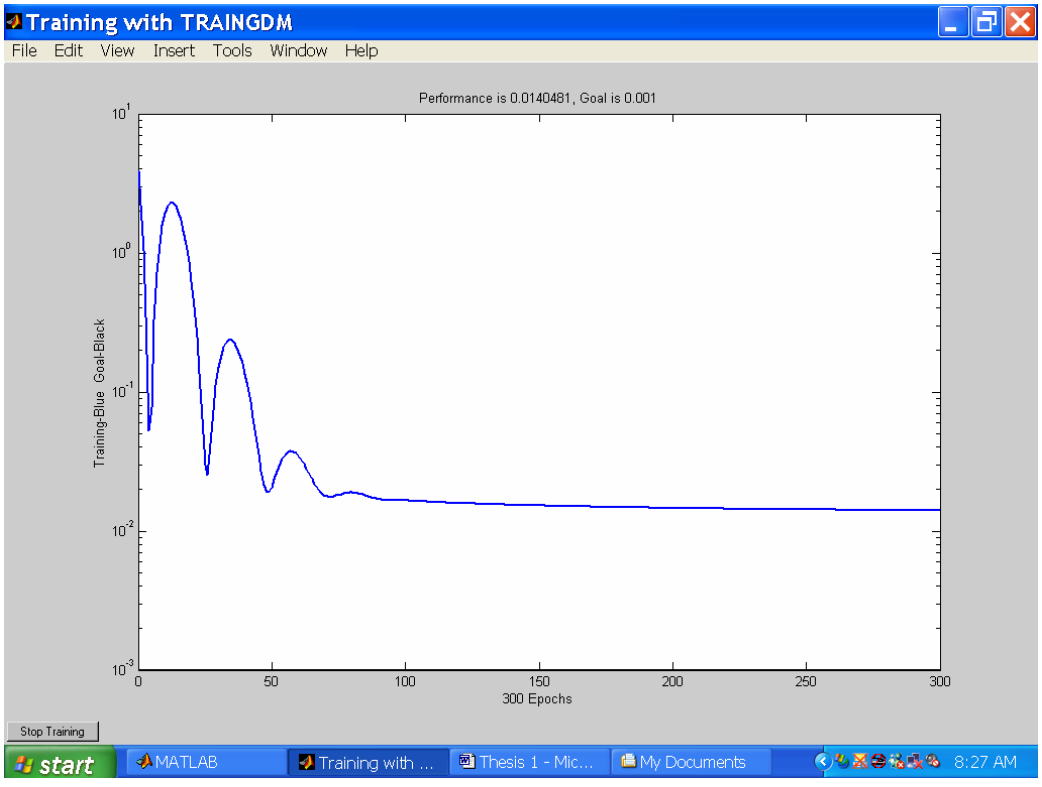

Figure 2. Training result of traingdm

From the training time parameter epoch in Figure 2, we can see the curve stabilizes earlier than in the previous algorithm. That means the batch gradient descent with momentum provides faster convergence because momentum makes the network respond not only to the local gradient but also to recent trends in the error surface. Momentum allows the network to ignore small features in the error area (surface), otherwise the network may get stuck in a shallow local minimum.

From the result, we find the minimum MSE is 0.0140481 and estimated training time is about 150 epochs since the MSE is very stable after that.

5.1.3 Variable learning rate back-propagation (traingda)

This training algorithm should be faster than the previous two back-propagation training algorithms. It uses heuristic techniques and is developed from an analysis of the performance of the standard steepest descent algorithm.

The variable learning rate back-propagation algorithm still uses the steepest descent algorithm. The difference is the learning rate is not held constant throughout training. Since the optimal learning rate changes during the training process, it is not possible optimally to set the learning rate before training, and the performance is very sensitive to the proper setting of the learning rate.

This algorithm tries to improve performance by allowing the learning rate to keep the learning step size as large as possible and keeping the learning stable. The learning rate is made responsive to the complexity of the local error surface. If the new error exceeds the old error by more than a predefined ratio max-perf-inc (maximum performance increase, typically 1.04) the new weights and biases are decreased. In addition, the learning rate is

decreased (typically lr_dec = 0.7). If the new error is less than the old error, the learning rate is increased (typically lr_inc = 1.05)

In this project we set lr_inc = 1.05 and lr_dec = 0.7. In the following code we train the network with the function traingdx with an adaptive learning rate. The other training parameters are the same as those in traingd except for the additional training parameters lr_inc and lr_dec. The training code is as follows:

```
Net = init (net);                    //this code is for reinitializing the weights and biases//
Net = newff (minmax(inputinit), [3,1], {'transig','purelin'},'traingda');
Net.trainparam.lr_inc = 1.05;        //or Net.trainparam.lr_dec = 0.7;//
(Net, tr) = train (Net, inputint, targetinit);
```

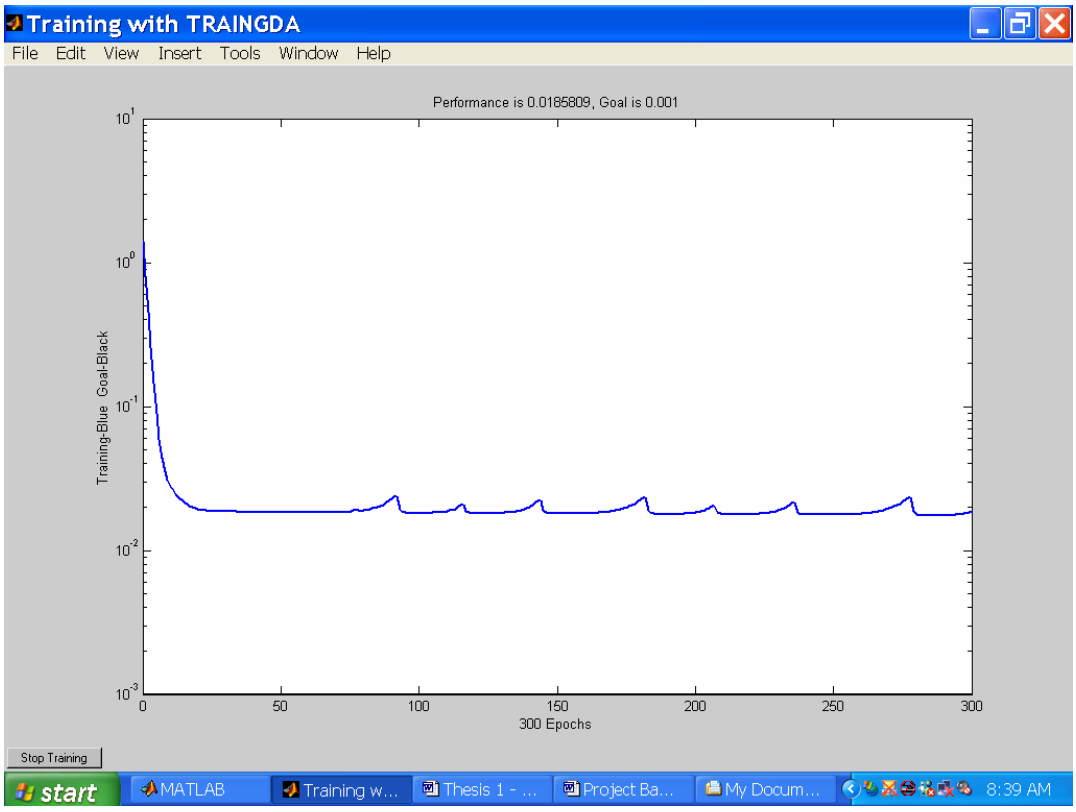

Figure 3. Training result of traingda

The training result is shown in Figure 3. From the result, we find the minimum MSE is 0.0185809. Its convergence is very fast, but the MSE is not stable. From the training performance, we find that choosing lr_inc is better than lr_dec. This is because this training process is without large error increases. The function, traingda, combines an adaptive learning rate with momentum training. This implementation shows this algorithm converges faster than the previous two algorithms.

5.1.4 Resilient back-propagation (trainrp)

Generally, a multi-player network uses a sigmoid transfer function in the hidden layer. This function compresses an infinite input range into a finite output range. This causes difficulties when using the steepest descent method to train a multi-layer network since the gradient can have a very small magnitude that can cause small changes in weights and biases. This condition makes that weights and biases have values far from optimal, and slows performance.

The resilient back-propagation (Rprop) training algorithm is utilized to eliminate harmful effects of the magnitudes. In this algorithm, only the sign of the derivative is used to determine the direction of the weights update. The size of the weight change is determined by a separate update value. If the derivative of the performance function with respect to that weight has the same sign for two successive iterations, the weight and bias is increased by the factor delt_inc. If the sign of the derivative changes from the previous iteration the weight and bias is decreased by the factor delt_dec. If the derivative is zero then the update value remains the same. By using this algorithm when weights are oscillating, the weight change is reduced. If the weights continue to change in the same

direction for a few iterations then the size of the weight change will be increased. This speeds the change of the weights and biases in the direction of their optimal values, and gives the process a better performance in terms of convergence time. The training parameters for trainrip are epoch_show, goal, time, min_grad, max_fail, delt_inc and delt_dec. The first three parameters are the same as the parameters for traingd. The other parameters are set as the default values. The training code is as follows:

Net = newff (minmax(inputinit), [3,1], {'transig','purelin'},'trainrp');

(Net, tr) = train (Net, inputint, targetinit);

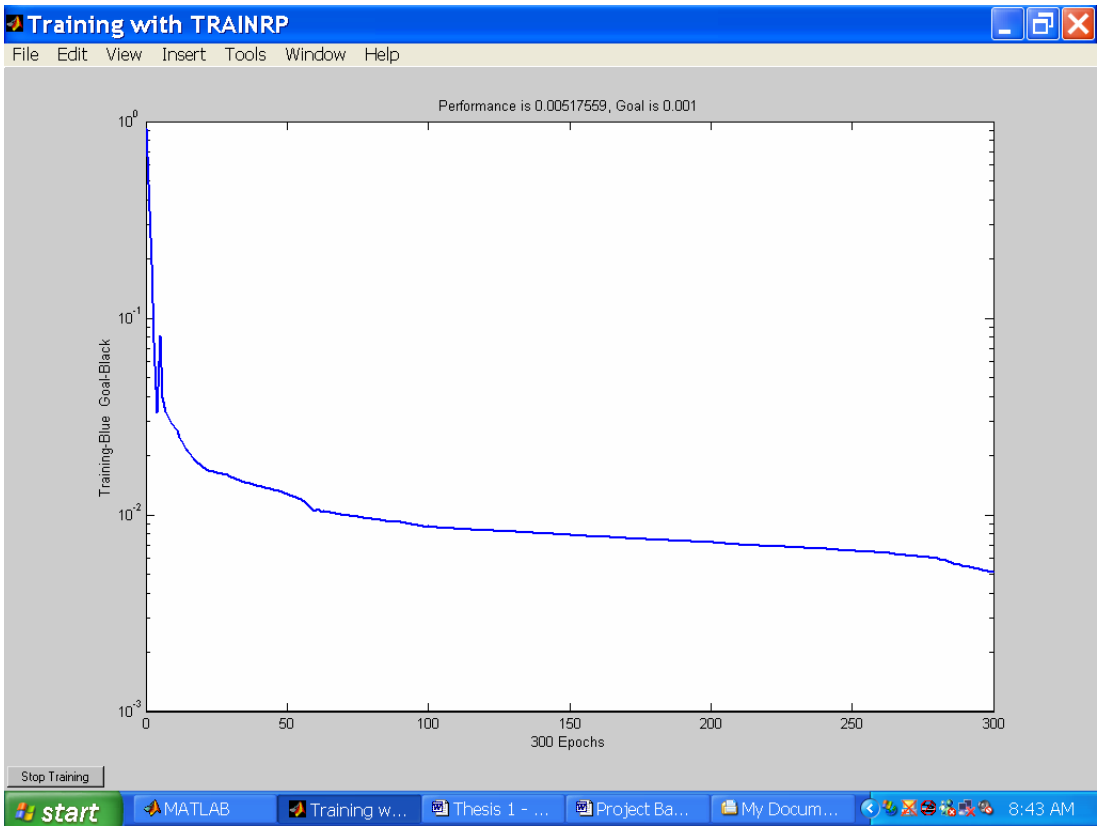

Figure 4. Training result of trainrp

The training result is illustrated in Figure 4. From the result, we find the minimum MSE is 0.00517559. From the training result, we find the Rprop algorithm is generally much faster than the standard steepest descent algorithm and has a smaller MSE.

5.1.5 Conjugate gradient (traincgf, traincgp, traincgb, trainscg)

The basic back-propagation algorithm adjusts the weights in the direction of the MSE decreasing most rapidly, which is the steepest descent direction (negative of the gradient), but it still does not necessarily have the fastest convergence.

In the conjugate gradient algorithms the weights are adjusted along conjugate directions. That generally produces faster convergence than the steepest descent direction searching in the steepest descent. This algorithm begins by searching in the steepest descent direction (negative of the gradient) on the first iteration $P_0 = -g_0$. Then a line search is used to determine the optimal distance to move along the current search direction. The search direction will be reset periodically to the negative of the gradient

$$X_{(k+1)} = X_k + \alpha_k P_k \tag{7}$$

Where α is the learning rate and X is the vector of weight and bias. The next search direction is to combine the new steepest descent direction with the previous search direction

$$P_k = - g_k + \beta_k P_{(k-1)}. \tag{8}$$

Where g is the gradient. There are four kinds of conjugate gradient algorithms, which depend on how the constant β_k is computed.

5.1.5.1 Fletcher-Reeves Update (traincgf)

The Fletcher-Reeves Update procedure is:

$$\beta_k = \frac{g_k^T g_k}{g_{k-1}^T - g_{k-1}} \tag{9}$$

This is the ratio of the normalized square of the current gradient to the normalized square of the previous gradient. The training code follows, and the whole conjugate gradient algorithm training parameters are the same as traingd:

Net = newff (minmax(inputinit), [3,1], {‘transig’,’purelin’},’traincgf’);

(Net, tr) = train (Net, inputint, targetinit);

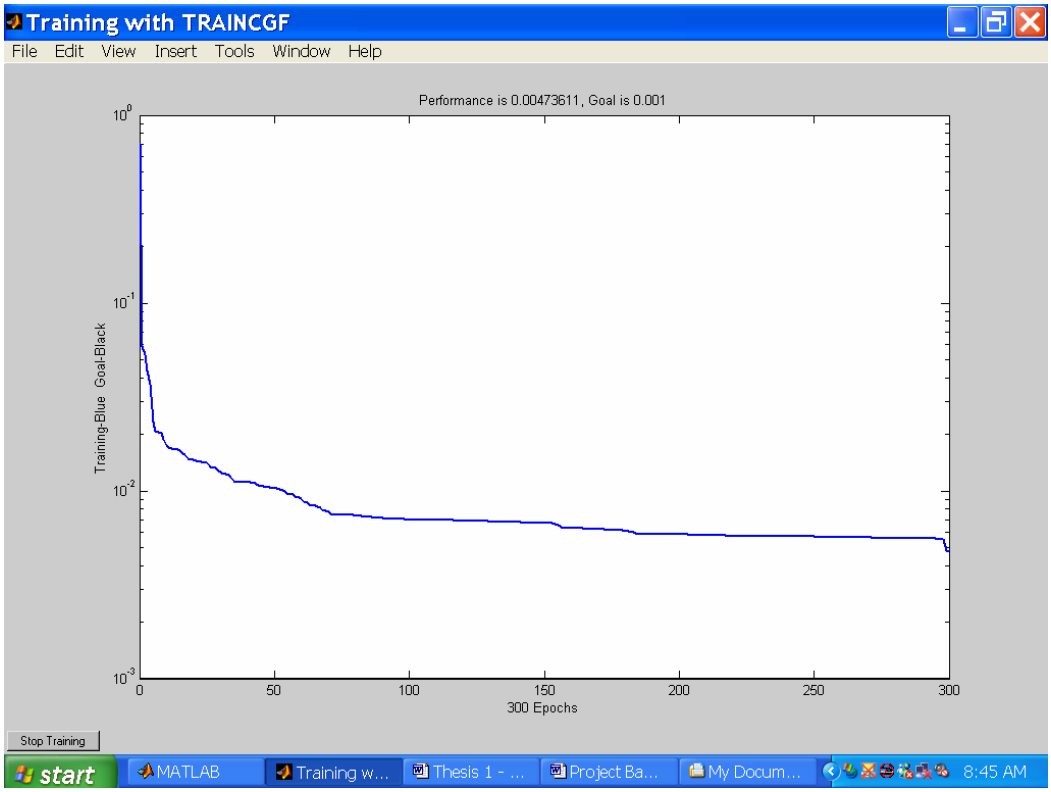

Figure 5. Training result of traincgf

The training result is illustrated in Figure 5. From the result, we find the minimum MSE is 0.00473611. The Fletcher-Reeves Update algorithm has a slightly better performance than the preview algorithms, and has a smaller MSE, but it has a slower convergence than some of the other algorithms.

5.1.5.2 Palak-Ribiere (traincgp)

The Palak-Ribiere algorithm upgrades the constant β_k as computed by

$$\beta_k = \frac{\Delta g_{k-1}^T g_k}{g_{k-1}^T - g_{k-1}} \tag{10}$$

The code is as follows:

Net = newff (minmax(inputinit), [3,1], {'transig','purelin'},'traincgp');

(Net, tr) = train (Net, inputint, targetinit);

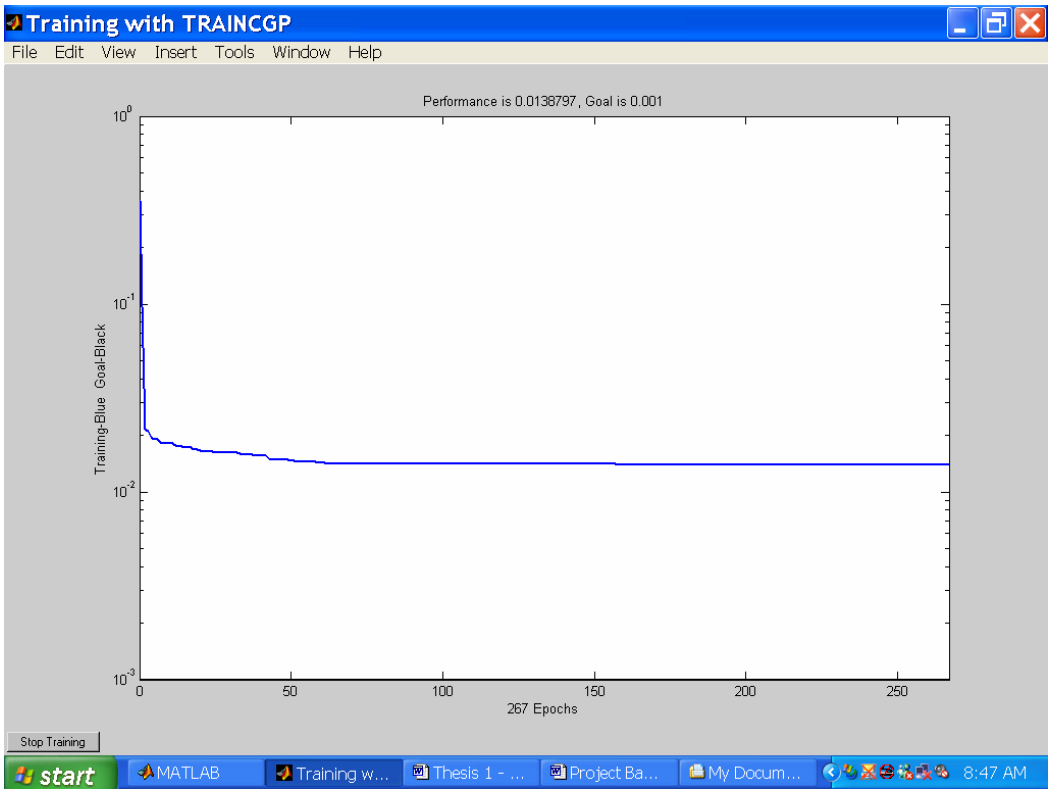

Figure 6. Training result of traincgp

The training result is illustrated in Figure 6. From the result, we find the minimum MSE is 0.0138797. The Palak-Ribiere algorithm does not have as good a performance as the previous algorithms, but it has a much faster convergence.

5.1.5.3 Powell-Beale Restarts (traincgb)

For all conjugate gradient algorithms, the search direction will be periodically reset to the negative of the gradient. The standard reset point is when the number of network parameters (weights and biases) is equal to the number of iterations [8]. For the Powell-Beale Restarts algorithm, if there is very little orthogonality left between the current gradient and previous gradient the search direction is reset to negative of the gradient.

Net = newff (minmax (inputinit), [3,1], {'transig','purelin'},'traincgb');

(Net, tr) = train (Net, inputint, targetinit);

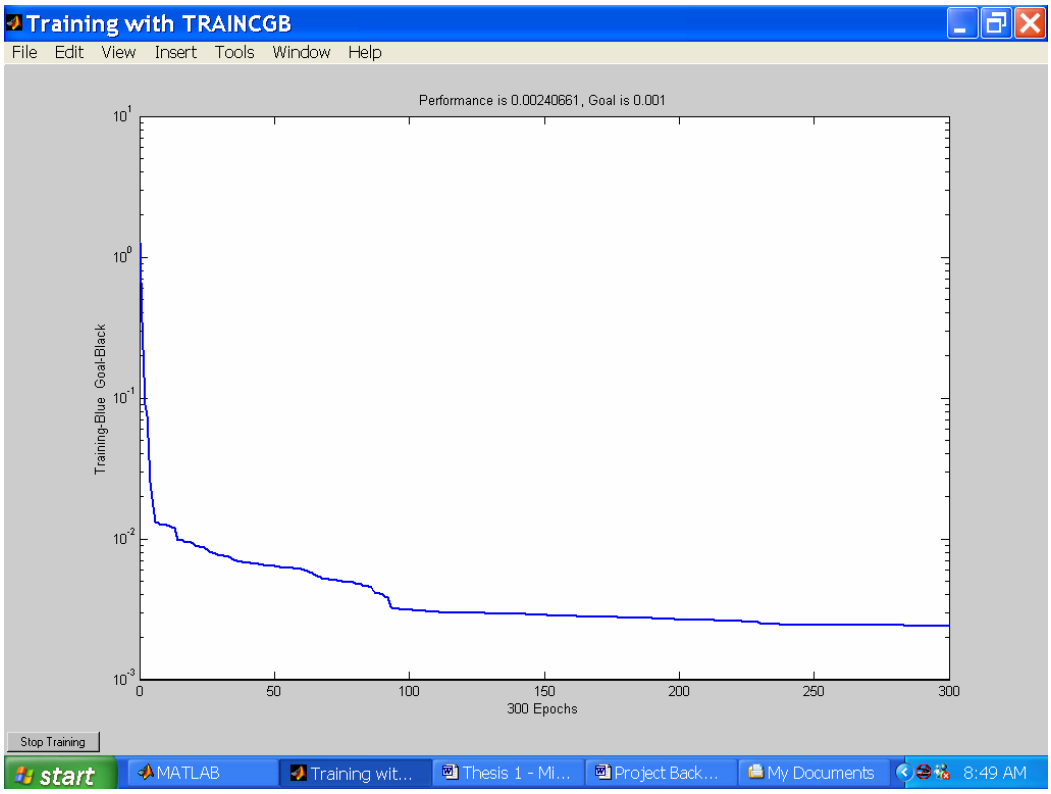

Figure 7. Training result of traincgb

The training result is illustrated in Figure 7. From the result, we find the minimum MSE is 0.00240661. The Powell-Beale Restarts algorithm has a much better performance

than the previous algorithms and has a smaller MSE, but the convergence is slower than some of the other algorithms.

5.1.5.4 Scaled conjugate gradient (trainscg)

Up to this point, each conjugate gradient algorithm has required a line search at each iteration. This line search is very expensive in terms of time. The scaled conjugate was designed to avoid the time-consuming line search. It combines the model-trust region approach with the conjugates gradient approach. The code is as follows:

Net = newff (minmax(inputinit), [3,1], {'transig','purelin'},'trainscg');

(Net, tr) = train (Net, inputint, targetinit);

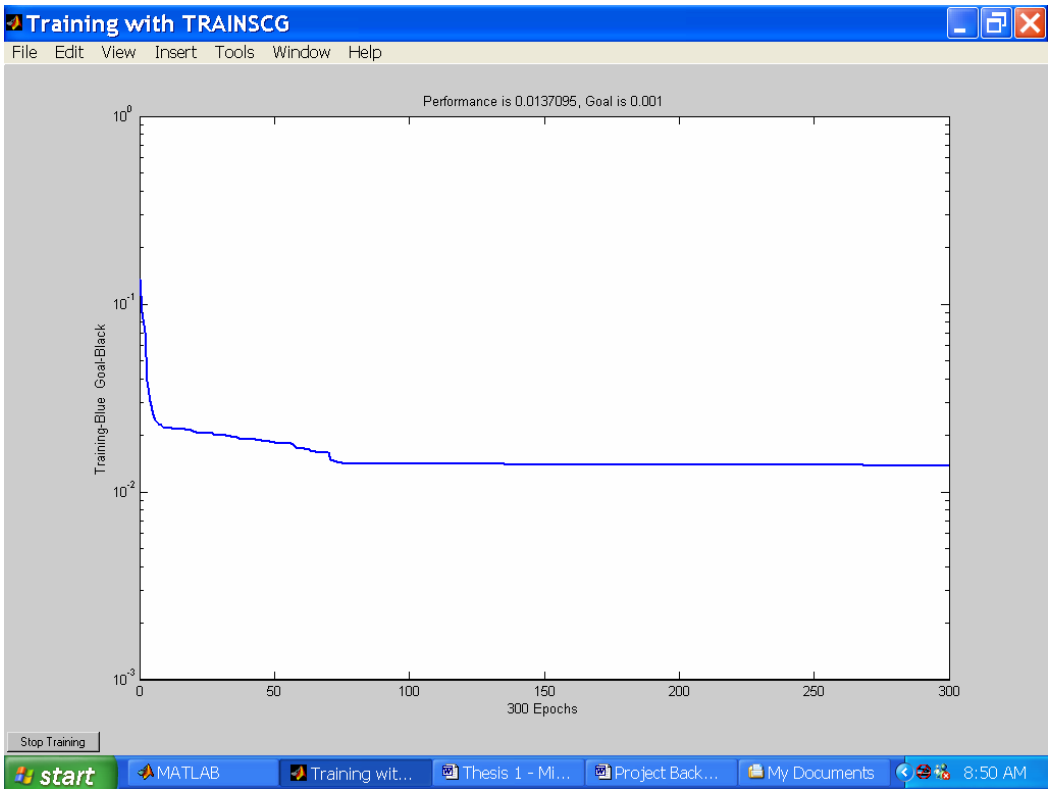

Figure 8. Training result of trainscg

The training result is illustrated in Figure 8. From the result, we find the minimum MSE is 0.0137095. The scaled conjugate gradient algorithm does not have as good a performance as the previous algorithms, but it has a slightly faster convergence than the other algorithms.

Comparing the above four conjugate gradient algorithms, some have better function performance (MSE); *e.g.,* Powell-Beale Restarts algorithm, and all of them have faster convergence than the previous algorithms.

5.1.6 Quasi-Newton (trainbfg, trainoss)

Newton algorithms are an alternative to the conjugate gradient algorithms for fast optimization. The weight and bias vector X are updated as in the following equation:

$$X_{(k+1)} = X_k - A_k^{-1} g_k. \quad (11)$$

Where A_k is the Hessian matrix (second derivative) of the performance index at the current values of the weights and biases.

Newton's method often converges faster than conjugate gradient methods because the algorithm does not require calculation of the second derivative. This may not apply to feed-forward neural networks because it is not difficult to compute the Hessian matrix.

5.1.6.1 BFGS algorithm (trainbfg)

This algorithm requires more computation in each iteration and more memory storage than the conjugate gradient algorithm because the Hessian matrix is n * n dimensionals and each n is equal to the total number of weights and biases in the network. However,

for smaller networks the BFGS algorithm can be an efficient training function, especially for this project. The code is as follows:

Net = newff (minmax(inputinit), [3,1], {'transig','purelin'},'trainbfg');

(Net, tr) = train (Net, inputint, targetinit);

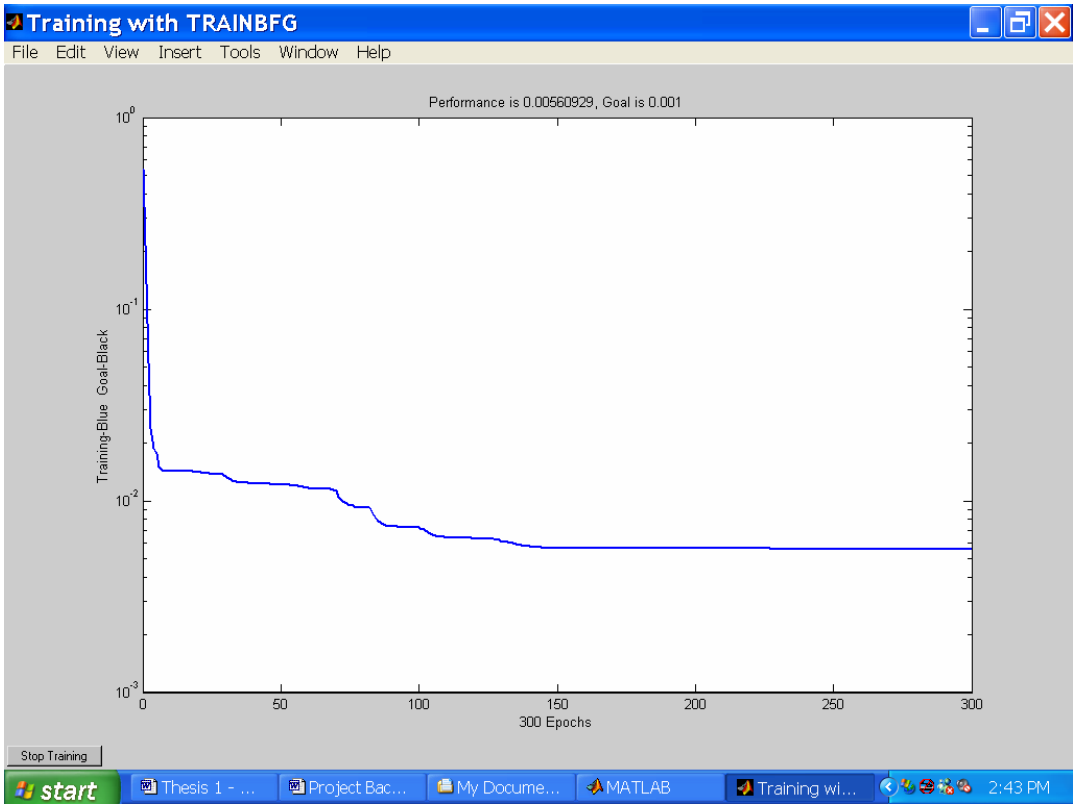

Figure 9. Training result of trainbfg

The training result is illustrated in Figure 9. From the result, we find the minimum MSE is 0.00560929. The BFGS algorithm has a good performance, but it has a slightly slower convergence than the conjugate gradient algorithms.

5.1.6.2 One-step second (OSS) algorithm (trainoss)

One-Step Second algorithm is trying to bridge the gap between the conjugate gradient algorithm and the Quasi-Newton (second) algorithm. This algorithm requires less

memory storage since it assumes that at each iteration the previous Hessian matrix was the identity matrix so it doesn't need to store the complete Hessian matrix. The code is as follows:

Net = newff (minmax(inputinit), [3,1], {'transig','purelin'},'trainoss');

(Net, tr) = train (Net, inputint, targetinit);

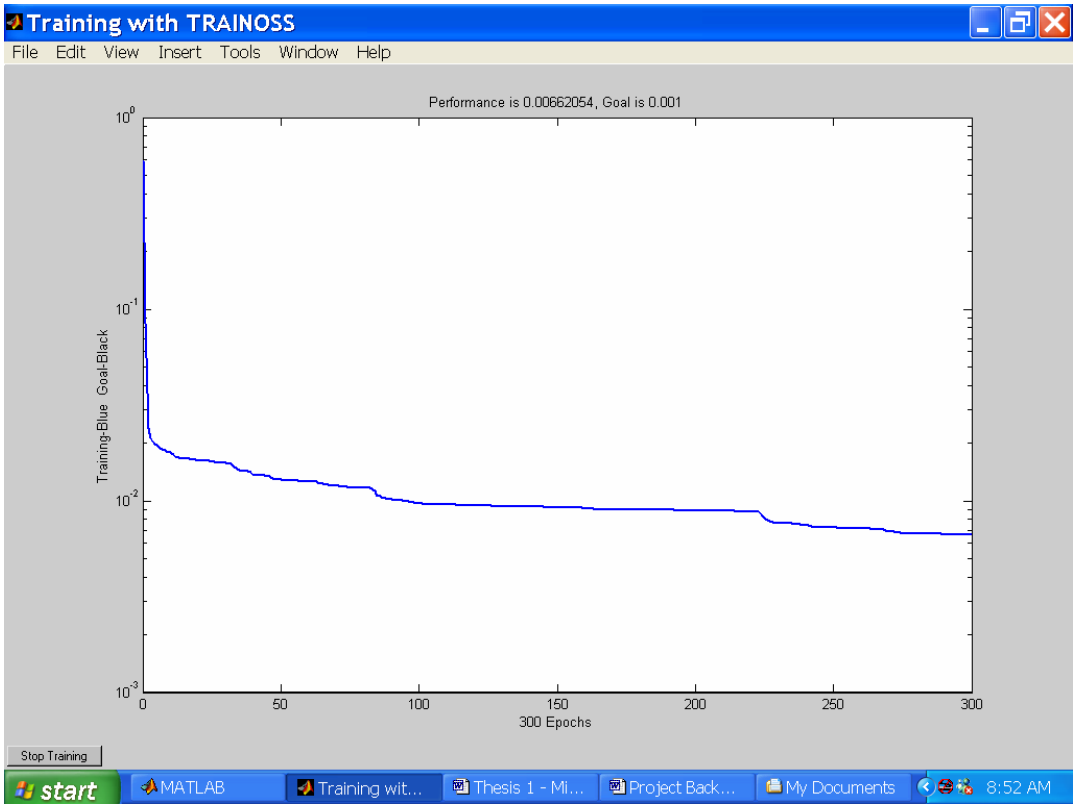

Figure 10. Training result of trainoss

The training result is illustrated in Figure 10. From the result, we find the minimum MSE is 0.00662054. The One-Step Second algorithm has a good performance, but it has slower convergence than the other algorithms.

5.1.7 Levenberg-Marquardt (trainlm)

Like the Qussi-Newton algorithm, the Levenberg-Marquardt algorithm also does not require calculation of second derivatives. The Hessian matrix can be approximated as follows:

$$H = J^T J \tag{12}$$

The gradient can be computed as follow:

$$g = J^T e \tag{13}$$

Where J is the Jacobian matrix, which contains the first derivations of the network errors with respect to the weights and biases and e is a vector of network error. The computation of the Jacobian matrix is much less complex than computing the Hessian matrix. Therefore, this algorithm seems to be the fastest method for training moderate-sized feed-forward neural networks. The Levenberg-Marquardt algorithm uses Newton-like updates and Hessian matrix approximations as follows:

$$X_{(k+1)} = X_k - [J^T J + \mu I]^{-1} J^T e. \tag{14}$$

When $\mu = 0$ it is Newton's method. When μ is large it becomes a gradient descent with a small step size. The code is as follows:

Net = newff (minmax(inputinit), [3,1], {'transig','purelin'},'trainlm');

(Net, tr) = train (Net, inputint, targetinit);

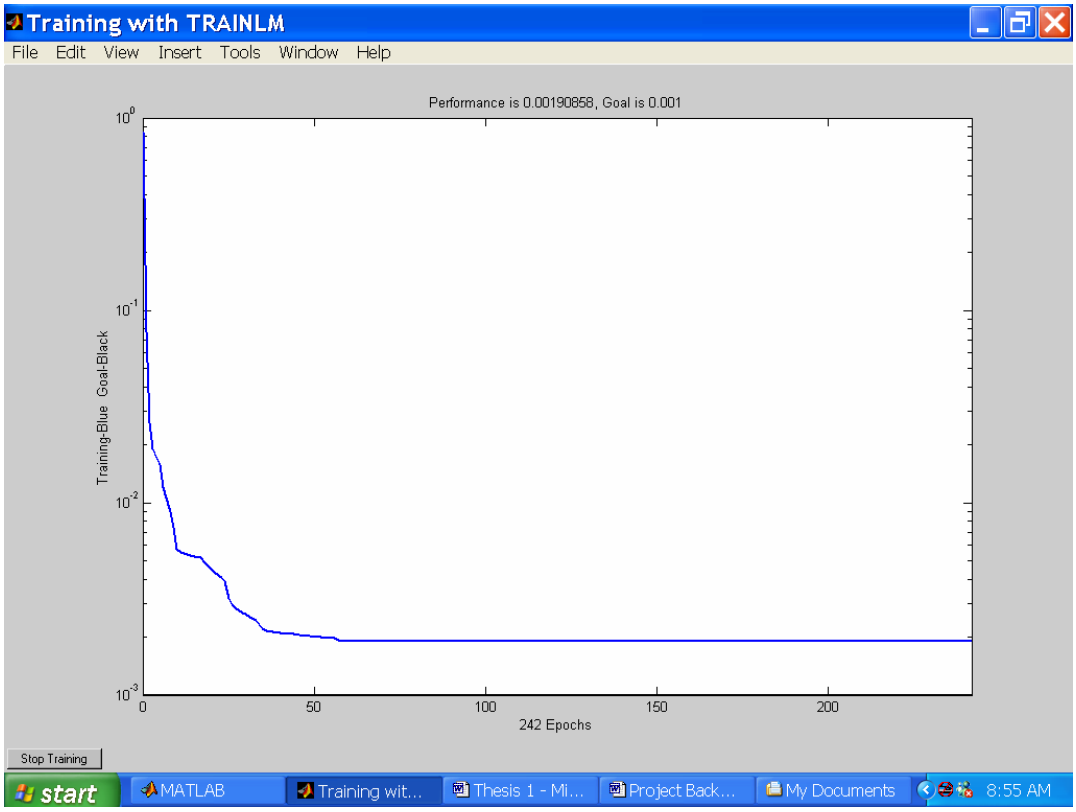

Figure 11. Training result of trainlm

The training result is illustrated in Figure 11. From the result, we find the minimum MSE is 0.00190858. The Levenberg-Marquardt algorithm has the best performance and faster convergence than the other algorithms. It has a very efficient Matlab implementation.

5.1.8 Summary of the performance comparison of different algorithms

The foreign exchange rate data is a simple function approximation problem. A 1-3-1 network, with tansig transfer functions in the hidden layer and linear transfer functions in the output layer, is used to approximate the trend of the exchange rate. The training results show the gradient descent algorithm is generally very slow because it requires small learning rates for stable learning. The conjugate gradient algorithms have fast

convergence performance, but function performances are not very good. The application of the Levenberg-Marquardt algorithm appears to be the fastest method for training moderate-sized feed-forward neural networks. The Levenberg-Marquardt algorithm is best suited to deal with a function approximation problem where the network has up to several hundred weights and the approximation must be very accurate. This project's research area involves that type of task. The algorithm also has a very efficient Matlab implementation since the solution of the matrix equation is a built-in function, therefore its attributes become even more marked in a Matlab setting. From the application result we also found the Levenberg-Marquardt algorithm has the smallest MSE.

5.2 Performance comparison with different problems

In this section, we compare two neural network training performances by using the Australian dollar versus U.S. dollar exchange rates, and the Australian dollar versus Chinese yuan exchange rates. We'll use different algorithms in the networks to see the general performance of each algorithm in the different problems. The following table lists the algorithms that are tested and the acronyms we use to identify them.

GDA : traingda-Variable Learning Rate Back-propagation

RP : trainrp-Resilient Back-propagation

CGF : traingrf-Fletcher-Powell Conjugate Gradient

CGP : traincgp-Polak-Ribiere Conjugate Gradient

CGB : traincgb-Conjugate Gradient with Powell/Beale Restarts

SCG : trainscg-Scaled Conjugate Gradient

BFG : trainfg-BFGS Quasi-Newton

OSS : trainoss-One-Step Secant

LM : trainlm-Levenberg-Marquardt

Table I shows the mean square error for each algorithm in the different iteration epoch of Australia dollar versus U.S. dollar exchange rate data training process. In the algorithm data in the first line, *e.g.,* 50 in 50/300, the first number represents the iterations and the second number, 300, is the maximum number of epochs. From this table we can see the relationship among the algorithms. There is further illustration in Figure 12, a plot of the time required to converge versus the mean square error convergence goal. From this figure we can see that as the error is reduced, the improvement provided by the LM algorithm becomes more pronounced. Some algorithms perform very well as the error goal is reduced (*e.g.,* CGB).

TABLE I

THE MSE FOR EACH ALGORITHM IN THE DIFFERENT EPOCH FOR AUSTRALIA DOLLAR VERSUS U.S. DOLLAR

Algorithm	0/300	50/300	100/300	150/300	200/300	250/300	300/300
GDA	1.418310	0.018448	0.018169	0.018003	0.018242	0.017690	0.018581
RB	0.918038	0.012756	0.008724	0.007889	0.007242	0.006544	0.005176
CGF	0.704973	0.010344	0.007042	0.006717	0.005870	0.005696	0.004736
CGP	0.355534	0.014596	0.014006	0.014020	0.013889	0.013883	0.013880
CGB	1.264740	0.006409	0.003119	0.002894	0.002668	0.002455	0.002407
SCG	0.136152	0.018363	0.014092	0.013999	0.013950	0.013889	0.013710
BFG	0.537822	0.012177	0.007255	0.005676	0.005645	0.005613	0.005609
OSS	0.592987	0.012926	0.009704	0.009258	0.008921	0.007247	0.006621
LM	0.842387	0.002006	0.001906	0.001909	0.001909	0.001909	0.001909

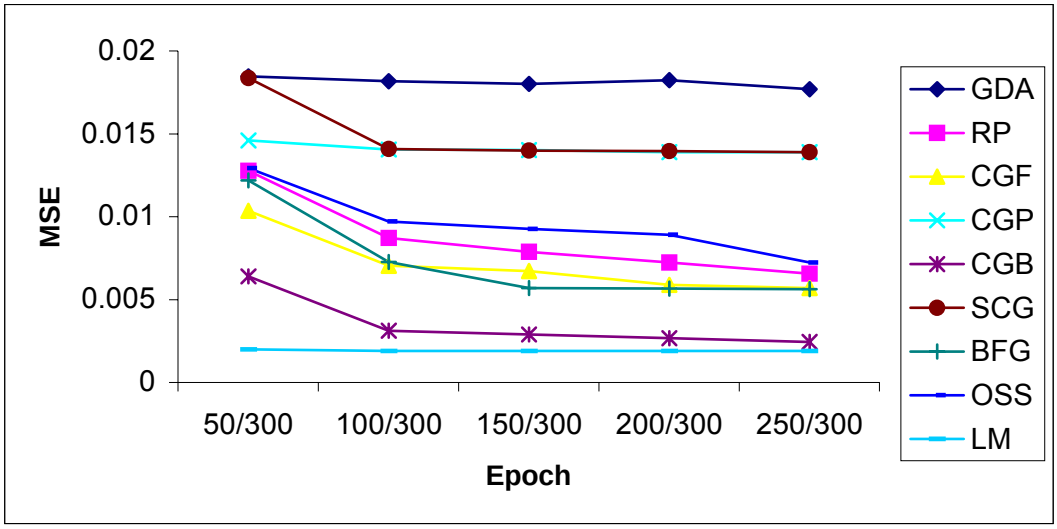

Figure 12. Performance comparison for Table I

Table II shows the mean square error for each algorithm in the different iteration epoch of the Australia dollar versus Chinese yuan exchange rate data training process. In the algorithm data in the first line, *e.g.,* 60 in 60/300, the first number represents the iterations and the second number, 300, is the maximum number of epochs. The performance of the various algorithms can be affected by the accuracy required of the approximation. From Table II, we find that in the LM algorithm the error decreases much more rapidly than the other algorithms. It converges very quickly from the beginning of the training process. The relationships among the algorithms are illustrated in Figure13, which plots the time required to converge versus the mean squire error convergence goal. Here we can see that as the error goal is reduced, the improvements provided by the LM algorithm and BFG algorithm become more pronounced.

TABLE II

THE MSE FOR EACH ALGORITHM IN THE DIFFERENT EPOCH FOR AUSTRALIA DOLLAR VERSUS CHINESE YUAN

Algorithm	0/300	60/300	120/300	180/300	240/300	300/300
GDA	0.215103	0.071355	0.030595	0.029037	0.028101	0.027857
RP	2.119290	0.026936	0.021331	0.016078	0.012360	0.010432
CGF	3.058830	0.028571	0.028333	0.028007	0.027796	0.027307
CGP	1.530740	0.019498	0.014582	0.009327	0.007522	0.006439
CGB	0.632063	0.027451	0.009045	0.006591	0.006372	0.006298
SCG	0.993480	0.027044	0.022846	0.013440	0.007731	0.007452
BFG	0.732348	0.005761	0.005004	0.004879	0.004878	0.004878
OSS	3.671140	0.028907	0.028469	0.028434	0.028346	0.028255
LM	0.417254	0.003597	0.003591	0.003590	0.003590	0.003589

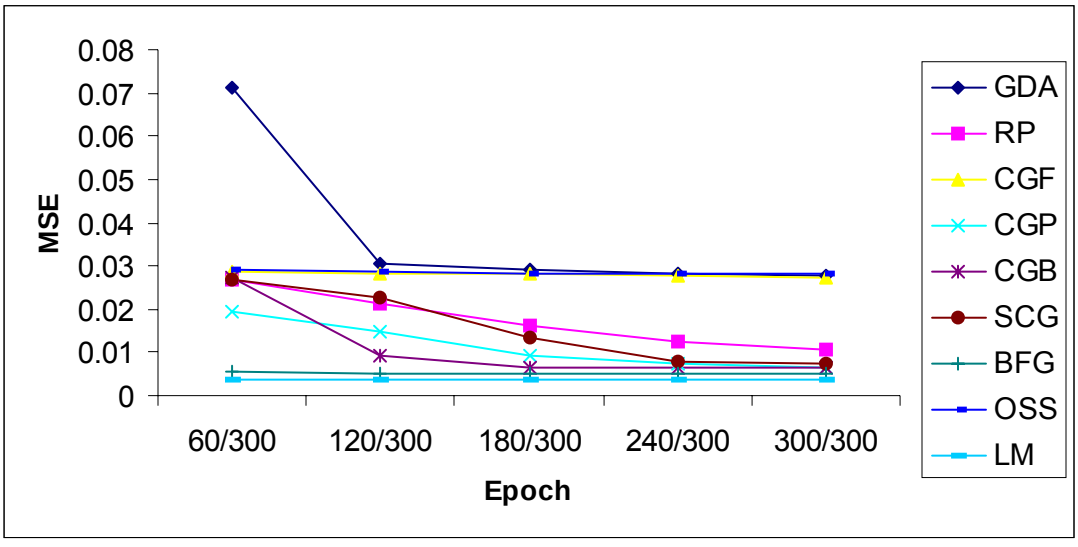

Figure 13. Performance comparison for Table II

The performance of the various algorithms can be affected by the accuracy required of the approximation. From the above two examples we find that in the LM algorithm, the error decreases more rapidly than in the other algorithms. It converges very quickly from the beginning of the training process.

There are several algorithm characteristics that we can deduce from the experiments we have completed.

LM: In general, the Levenberg-Marquardt algorithm will have the fastest convergence for networks that contain up to a few hundred weights. This advantage is especially noticeable if very accurate training is required. In many cases trainlm is able to obtain a lower mean square error than the other algorithms tested. However, as the number of weights in the network increase, for example, increasing the network layers and neurons, the advantage of the trainlm may decrease. The storage requirements of trainlm are larger than the other algorithms tested. By adjusting the mem_reduc parameter, the storage requirements can be reduced, but at a cost of increased execution time.

RP: The trainrp function does not perform well on function approximation problems. Its performance degrades as the error goal is reduced.

SCG: The conjugate gradient algorithm, in particular trainscg, seems to perform well over a wide variety of problems, particularly for networks with a large number of weights. The SCG algorithm is almost as fast as the LM algorithm on function approximation problems (fast for large networks). Its performance does not degrade as quickly as trainrp performance does when the error is reduced.

BFG: The trainbfg performance is similar to that of trainlm. It does not require as much storage as trainlm but the computation required does increase geometrically with the size of the network, since the equivalent of a matrix inverse must be computed at each iteration.

GDA: The traingda is usually much slower than the other methods and has about the same storage requirements as trainrp, but it can still be useful for some problems.

5.3 Simulation of the China currency exchange rate

In this section we put some of the ideas we previously covered together with an example of a typical training session. For this example, we performed a simulation of the Australia dollar versus Chinese yuan exchange rate to discover a pattern for this data. From this pattern we can predict the exchange rate.

We trained feed-forward networks on time series of foreign currency exchange rate between Chinese yuan and Australia dollars. We used the Matlab built-in preprocessing function to normalize the raw data.

The first step was to load the data into the workspace from an Excel database and use Matlab built-in preprocessing functions for scaling network inputs and targets. The function premnmx was used to scale input and targets so that they fall in the range [-1, 1]. The following code illustrates the use of this function:

```
xlsread ER-China;                          //load data into the workspace//
[inputn, mininput, maxinput, targetn, mintarget, maxtarget] = Premnmx(input, target);
```

A check of the size of the transformed data reveals:

```
[R, Q] = size (inputn);
```

The next step was to divide the data into training, validation and test subsets. We took one fourth of the data for the validation set, one fourth for the test set and one half for the training set. We chose the sets as equally spaced points throughout the original data.

```
iitst = 2:4:Q;
iival = 4:4:Q;
iitr = [1:4:Q 3:4:Q];
val.Input = inputpn(:,iival); val.Target = targetn(:, iival);
test.Input= inputn(:,iitst); test.Target = targetn(:, iitst);
inputtr = inputn(:,iitr); targettr = targetn(:, iitr);
```

We created a network and trained it. We tried a two-layer network, with a tan-sigmoid transfer function in the hidden layer and a linear transfer function in the output layer. This is a useful structure for function approximation (or regression) problems. We used three neurons in the hidden layer. The network had one output neuron since there was one target. We used the Levenberg-Marquardt algorithm for training, as follows;

```
net = newff(minmax(inputtr), [3 1], {'tansig' 'purelin'}, 'trainlm');
[net, tr] = train(net, inputtr, targettr, [ ], [ ], val, test);
```

When the network begins to overfit the data, the error on the validation set will typically begin to rise. The training stopped after 42 iterations because the validation error increased. It is useful to plot the training, validation and test errors to check the progress of training. It was done with the following commands:

```
Plot (tr.epoch, tr.perf, tr.epoch, tr.vperf, tr.epoch, tr.tperf);
Legend (‘Training’, ‘Validation’, ‘Test’, -1);
ylabel (‘Squared Error’); xlabel(‘Epoch’);
```

The result is shown in Figure 14. The result exhibited here is reasonable, since the test set error and the validation set error have similar characteristics, and it doesn't appear that any significant over-fitting has occurred.

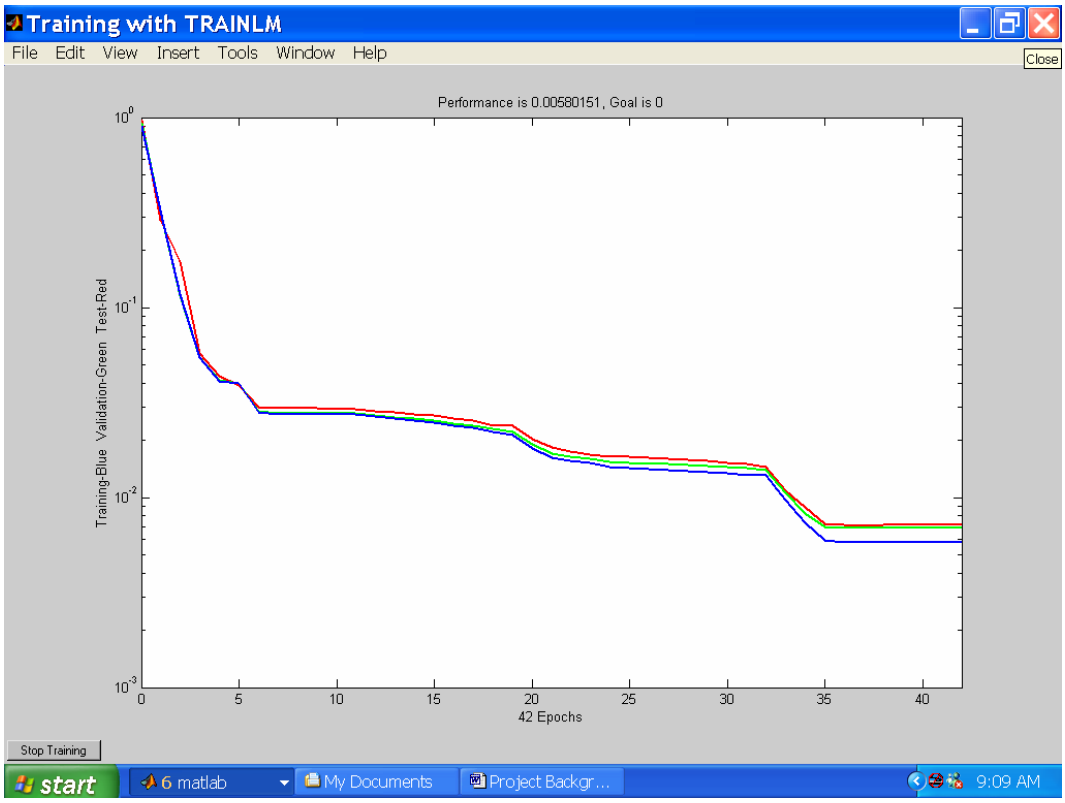

Figure 14. Performances for train, test and validation sets

The next step was to perform some analysis of the network response. We put the entire data set through the network (training, validation and test) and performed a linear regression between the network output and corresponding targets. Then we needed to unnormalize the network outputs by using the postprocessor function postmmx:

```
an = sim(net, pn);
a = postmmx(an, mint, maxt);
for i = 1:1
    figure(i)
```

[m(i), b(i), r(i)] = postreg(a(i,:), t(i,:));

end

The results are shown in Figures 15. The vertical axis is scaled exchange rate output data and the horizontal axis is scaled input data. The output seems to track the targets reasonably well.

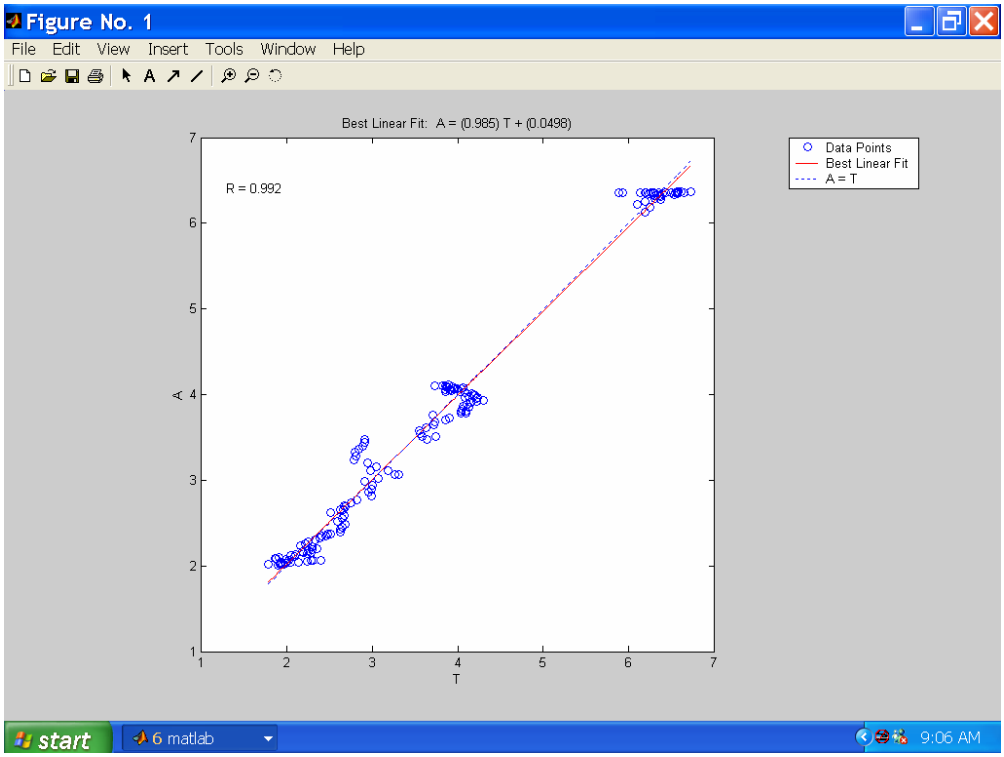

Figure 15. Comparison of simulation output and time series

5.4 Simulation prediction

The trained network is the model we built for this individual problem. Based on the experience of training the Australia dollar versus U.S dollar currency exchange rate data by using different network structures and different algorithms, we chose the 1-3-1 feed-forward network structure with tansig transfer functions in the hidden layer and linear transfer functions in the output layer to approximate the trend of the exchange rate. Since

the Levenberg-Marquardt algorithm appeared to be the fastest method for training moderate-sized feed-forward neural networks, especially for the function approximation problem, we used the LM algorithm as the training algorithm. It was the best suited for this individual problem. We used 70% of the data as training data. After this network was built, and the training process completed, the model for this individual problem was constructed (see section 5.1). Then we used this model to predict the value of future foreign exchange rates by using the function sim. For example, we used the remaining 30% of the data to test the model. The code is as follows:

```
Test_output  = sim (Net, Pt); where Pt is test input data.
```

Then we plotted the Test_output and Target data to check the accuracy of the prediction. The code for the plot is:

```
x = (1 : 1 : 74);
plot  (x, Target, '-*', x, Test_output);
```

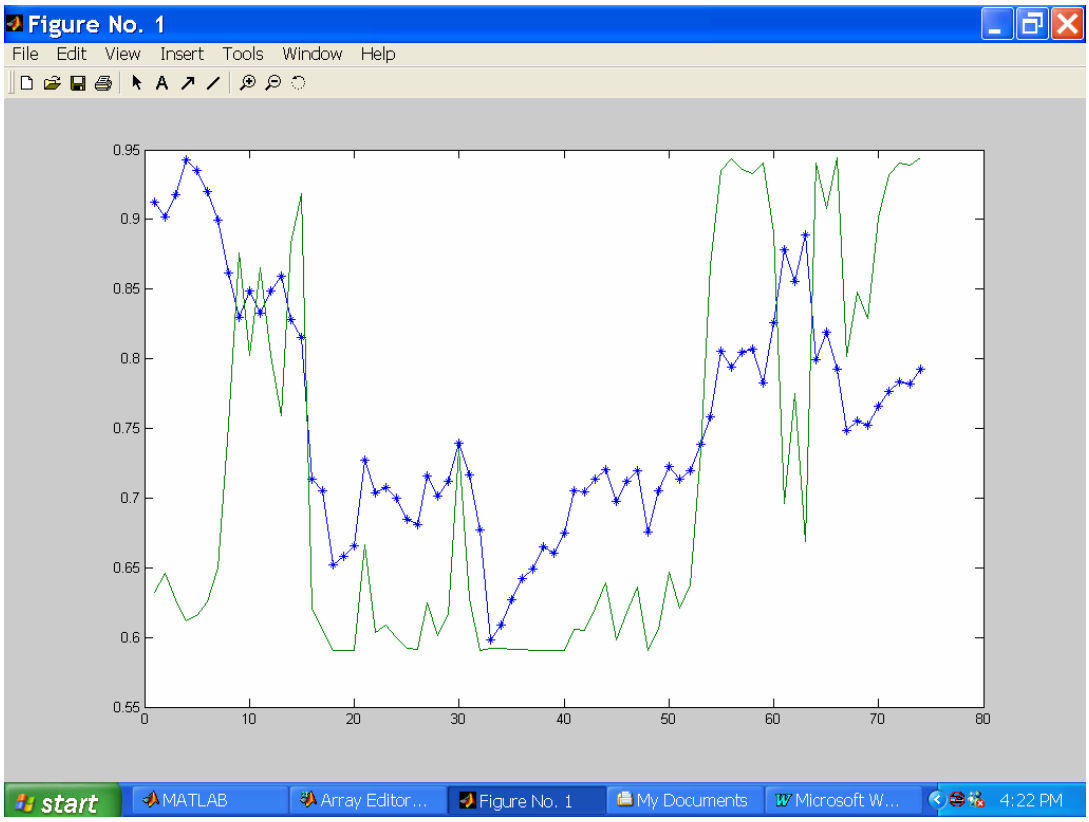

Figure 16 Simulation for the prediction

Figure 16 is the simulation for the prediction of the future currency exchange rate trend. The exchange rates are represented on the vertical axis and the epochs are represented on the horizontal axis. The output line seems to track the target line (*) reasonably well as far as the trend is concerned.

CHAPER 6

CONCLUSION AND FUTURE WORK

This paper is the implementation of back-propagation neural networks on foreign currency exchange rates. We attempt to prove that by using a back-propagation neural network, FOREX rates can be predicted correctly, allowing maximal profits. This paper describes how to build an artificial neural network model to predict trends in foreign currency exchange rates. The back-propagation neural network was chosen for this research because it is capable of solving a wide variety of problems and it commonly is used in time series forecasting. We use feed-forward topologies, supervised learning and back-propagation learning algorithms on the networks. We try different network structures and training algorithms on different kinds of data and compare the performance to ascertain what type network structure and training algorithm is the best fit for what kind of data.

This paper uses Matlab neural network software as a tool to test different kinds of algorithms and network structures to find the best model for the prediction of FOREX rates. To date we have found that a network model using the LM as the training algorithm and 1-3-1 as the network structure has the best performance for FOREX rate data.

Neural networks have been criticized because of the black box nature of their solutions, extreme training times, difficulty in obtaining a stable solution and large

number of parameters that must be selected experimentally to generate a good forecast. From our project, we have provided evidence that neural networks can be used to correctly predict FOREX rates; thereby, decreasing the risk of making unreasonable decisions. There are, however, still limitations and cautions that must be considered when using neural networks as predictive models.

Although a multi-layer back-propagation network with enough neurons can implement just about any function, back-propagation will not always find the correct weights for the optimal solution. While a network being trained may be theoretically capable of performing correctly, back-propagation and its variations may not always find a solution.

Picking the learning rate for a nonlinear network is a challenge. Unlike linear networks, there are no easy methods for picking a good learning rate for nonlinear multi-layer networks.

Networks are also sensitive to the number of neurons in their hidden layers. Too few neurons can lead to under-fitting, too many can contribute to over-fitting.

One may want to reinitialize the network and retrain several times to guarantee that one has the best solution.

One significant direction in which we would like to expand our work is to explore more properties of Matlab neural network software, testing more parameters to increase the accuracy of the prediction, decrease the time consumed in the process and reduce memory usage.

More dimensions and more complex data should be tested to explore the potential for data analysis in Matlab. Overall, using the right analytical tools and methods can

decrease the chance of making incorrect decisions and increase the possibility of profitability in the area of foreign currency exchange rates.

SELECTED BIBLIOGRAPHY

[1] Abraham, Ajith (2002) "Analysis of Hybrid Soft and Hard Computing Techniques for FOREX Monitoring System." In *IEEE International Conference on Fuzzy Systems*, Honolulu, Hawaii, pp 1616-1622.

[2] Abraham, Ajith and Chowdhury, Morshed U. (2001) "An Intelligent FOREX Monitoring System." In *Proceedings of IEEE International Conference on Info-tech and Info-net*, Beijing, China, pp 523-528.

[3] Carney, John G. and Cunningham, Padraig (1996) "Neural Networks and Currency Exchange Rate Prediction," *Trinity College Working Paper Foresight Business Journal*, http://www.maths.tcd.ie/pub/fbj/forex4.html. Date posted 1996 and last date accessed September 2002.

[4] Chan, C. C. Keith and Teong, Foo Kean (1995) "Enhancing Technical Analysis in the FOREX Market Using Neural Networks." In *Proceedings of IEEE Conference on Neural Networks*, Perth, Australia, Vol. 2, pp 1023-1027.

[5] Chen, An-Sing and Leung, Mark T. (2003) "Regression Neural Network for Error Correction in Foreign Exchange Forecasting and Trading," Computers & Operations Research, http://www.sciencedirect.com. Last updated 2003 and last date accessed January, 2004.

[6] Chen, Shu-Heng and Lu Chun-Fen (1999) "Would Evolutionary Computation Help in Designs of ANNs in Forecasting Foreign Exchange Rate?" In *Proceedings of IEEE of the 1999 Congress on Evolutionary Computation*, Washington, D.C., pp 267-274.

[7] Dempster, M. A. H., Payne, Tom W., Romahi, Yazann, and Thompson, G. W. P. (2001) "Computational Learning Techniques for Intraday FX Trading Using Popular Technical Indicators." In *IEEE Transactions on Neural Networks*, Vol. 12 No.4, pp 744-754.

[8] Demuth, Howard and Beale, Mark, (1992-2002) *Neural Network Toolbox User's Guide Version 4*, http://brookscole.com/engineering/nnd.html. Last updated 2002 and last accessed November 2003.

[9] Dunham, Margaret H. (2003) *Data Mining Introductory and Advanced Topics*, Prentice Hall, Pearson Education, Upper Saddle River, New Jersey.

[10] Ghoshray, Sabyasachi (1996) "Foreign Exchange Rate Prediction by Fuzzy Inferencing on Deterministic Chaos." In *Proceedings of IEEE/IAFE Computational Intelligence for Financial Engineering*. New York City, pp 96-102.

[11] Giles, C. Lee, Lawrence, Steve and Tsoi, Ah Chung (1997) "Rule Inference for Financial Prediction Using Recurrent Neural Networks," In *Proceedings of IEEE/IAFE Conference on Computational Intelligence for Financial Engineering.",* New York City, pp 253-259.

[12} Green, Henry G. and Pearson, Michael A. (1995) "Artificial Intelligence in Financial Markets," In *Proceedings of IEEE International Conference on Neural Networks.* Perth, Australia, pp 839-844.

[13] Iokibe, Tadashi, Murata, Shoji and Koyama, Masaya *(1995)* "Prediction of Foreign Exchange Rate by Local Fuzzy Reconstruction Method," In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics, Vancouver, B. C., Canada,* pp 4051-4054.

[14] Ip, Horace H. S. and Wong, Hon-Ming (1991) "Evidential Reasoning in Foreign Exchang Rates Prediction (1991) In *Proceedings of the International Conference on Artificial Intelligence Applications on Wall Street,* New York City, pp152-159.

[15] Kaastra, Iebeling and Boyd, Milton (1996) "Designing a Neural Network for Forecasting Financial and Economic Time Series," *Neurocomputing 10:215-236.*

[16] Laddad, R. R., Desai, U. B. and Poonacha, P. G. (1994) "Faster and Better Training of Multi-Layer Perceptron for Forecasting Problems," In *IEEE Workshop on Neural Networks for Signal Processing*, Ermoni, Greece,i pp 88-97.

[17] Muhammad, A. and King, G. A. (1997) "Foreign Exchange Market Forecasting Using Evolutionary Fuzzy Networks." In *Proceedings of IEEE/IAFE Computational Intelligence for Financial Engineering,* New York City, pp 213-219.

[18] Piche, Stephen W. (1995) "Trend Visualization", In *Proceedings of IEEE/IAFE Conference on Computational Intelligence for Financial Engineering*, New York City, pp 146-150.

[19] Quah, T. S., Teh, H. H. and Tan, C. L. (1995) "Neural Logic Hybrid System for Currency Option Trading." In *Proceedings of IEEE International Conference on Neural Networks,* Perth, Australia, pp 824-828

[20] Rabatin, Arthur (1998) "Risk Management for Intelligent Trading Portfolios" *Applied Derivatives Trading*: pp1-12 http://www.rabatin.com/papers/aptadp/i/risk.html. Last updated December 1998 and last date accessed May 2004.

[21] Rabatin Investment Technology Ltd (1999) *The Project Guide for Artificial Intelligence in Finance and Investing:* Rabatin Investment Technology Ltd: http://www.rabatin.com. Last updated April 1999 and last date accessed June 2004.

[22] Refenes, A. N., Azema-Barac, M. and Karoussos, S. A. (1992) "Currency Exchange Rate Forecasting by Error Back-Propagation." In *Proceedings of IEEE International Conference on Systems Sciences,* Honolulu, Hawaii, pp 504-515.

[23] Staley, Mark and Kim, Peter (1995) "Predicting the Canadian Spot Exchange Rate with Neural Networks." In *Proceedings of IEEE/IAFE Conference on Computational Intelligence for Financial Engineering,* New York City, pp 108-112.

[24] White, Halbert and Racine, Jeffrey (2001) "Statistical Inference, The Bootstrap, and Neural-Network Modeling with Application to Foreign Exchange Rates," In *IEEE Transactions on Neural Networks*, Vol. 12, pp 657-673.

[25] Yao, Jintao and Tan, Chew Lim (2000) "A Case Study on Using Neural Networks to Perform Technical Forecasting of FOREX," *Neurocomputing 34:79-98.*

VITA

Jinxing Han Gould

Candidate for the Degree of

Master of Science

Thesis: FOREX PREDICTION USING ARTIFICIAL INTELLIGENT SYSTEM

Major Field: Computer Science

Biographical:

Education: Graduated from Beijing 113 High School, Beijing, P.R. China, in February, 1977. Received Bachelor of Science degree with a major in Physics from Beijing University, Beijing, P.R. China, in February, 1983. Received Certificate of Achievement in Computer Information Systems, Systems Support Technician, Tulsa Community College, Tulsa, Oklahoma, in February, 1998. Completed the requirements for Master of Science degree with a major in Computer Science at Oklahoma State University in December, 2004.

Experience: Taught technical school classes in physics, mathematics, electronics, computer operations and English at Beijing Medical high School, Beijing, P.R. China from February, 1983, through April, 1997. Employed by Oklahoma State University as a Teaching Assistant from January, 2000, through May, 2003.