

PREDICTING FINANCIAL MARKETS USING NEURO
FUZZY GENETIC SYSTEMS

By

BRENT ARTHUR DOEKSEN

Bachelor of Computer Science and Mathematic

Oklahoma State University

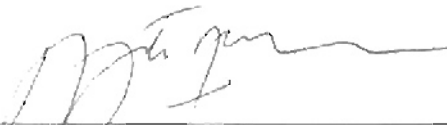
Stillwater, OK

1999

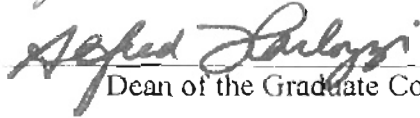
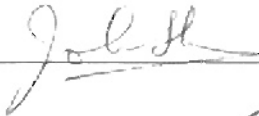
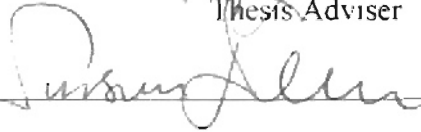
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2003

PREDICTING FINANCIAL MARKETS USING NEURO
FUZZY GENETIC SYSTEMS

Thesis Approved:



Thesis Adviser



Dean of the Graduate College

PREFACE

This study was conducted to provide knowledge in stock market prediction through the use of several different types of artificial intelligence systems. Many attempts have been made to accurately predict the stock market with only marginal success. This study shows that predicting the stock market is possible with very little input data and compares the abilities of several different methods: Neural Networks, Fuzzy Systems with Mamdani and Takagi Sugeno inference method. Mamdani Inference System was adapted using back propagation and genetic algorithms. Takagi Sugeno Fuzzy inference system was adjusted using back propagation learning and least squares method. This research is concluded with a yearlong profit simulation on two stocks, Microsoft and Intel. Thus showing how these models can be used to make profit.

I sincerely thank my master's committee Dr. Aijth Abraham (Chair), Dr. Dursun Delen, and Dr. Johnson Thomas for their support in the completion of this research.

TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION.....	1
Neural Networks.....	1
Conjugate Gradient.....	5
Fuzzy Logic.....	5
Genetic Algorithms.....	7
Decision Trees.....	10
Classification and Regression Tree.....	12
Objective of Study.....	13
Significance of Study.....	14
Data Set and Tools Used.....	14
II. LITERATURE REVIEW.....	17
Hurst Exponent.....	17
Scaling and Normalization.....	18
Overfitting and Overtraining.....	19
Learning.....	20
Splitting the Data Set.....	21
Recent Trends.....	22
III. HYBRID INTELLIGENCE SYSTEMS ARCHITECTURE.....	23
Stand Alone.....	23
Transformational Hybrid Intelligence System.....	23
Hierarchical Hybrid Intelligence System.....	24
Integrated Intelligence System.....	25
Conclusion of Hybrid Intelligence Systems.....	26

Chapter	Page
IV. HYBRID INTELLIGENCE SYSTEMS.....	20
ANFIS	27
Neuro Fuzzy.....	30
Takagi Sugeno Neuro Fuzzy.....	30
Mamdani Neuro Fuzzy.....	31
Input Selection.....	32
V. HURST EXPONENT ON DATA.....	33
VI. DATA PREPERATION.....	35
Input Reduction.....	35
Data Reduction.....	39
Data Transformation.....	41
VII. RESULTS.....	42
Testing Standards.....	42
Conjugate Gradient vs. Back Propagation.....	43
Takagi Sugeno Neuro Fuzzy Inference System Tests.....	45
Mamdani Neuro Fuzzy Tests.....	46
Classification vs. Regression.....	47
Random Data Sets.....	48
Even Split in Classification.....	50
VIII. SIMULATIONS AND DISCUSSIONS.....	51
IX. CONCLUSION.....	55
X. FUTURE WORK.....	57
BIBLIOGRAPHY	59

LIST OF TABLES

Table	Page
2.2 Normalization of a series.....	19
5.1 Hurst Exponent Calculations.....	35
6.1.1 Spearman Correlations.....	37
6.1.2 Greedy Input Reduction: 8 inputs.....	39
6.1.3 Greedy Input Reduction: 7 inputs.....	39
7.2 Conjugate Gradient vs. Back Propagation.....	45
7.3 Takagi Sugeno Neuro Fuzzy vs. Neural Networks.....	47
7.4 Mamdani Neuro Fuzzy Results.....	48
7.5 Classification vs. Regression.....	49
7.6 Classification across 3 Random Sets.....	50
7.7 Classification with Even Split.....	51
8.1 Profit Simulation.....	52
8.2 Profit on MSFT vs. INTC.....	55

LIST OF FIGURES

Figure	Page
1.1 Neural Network.....	3
1.3.1 Binary String Representation.....	8
1.3.2 Elitism.....	9
1.4 Example Decision Tree.....	10
1.7 Microsoft and Intel Stock Price.....	15
3.3 Hierarchical Hybrid Intelligent System.....	25
3.4 Integrated Hybrid Intelligent System.....	26
4.1 ANFIS.....	28
4.2.1 Takagi Sugeno Neuro Fuzzy Inference System.....	31
6.1 Sensitivity About the Mean: MSFT.....	38
7.2 Conjugate vs. Back Propagation.....	45
7.6 Random Data Sets.....	50
8.1 Microsoft Profit Simulation.....	53
8.2 Transactions.....	54
8.3 Intel Profit Simulation.....	55

NOMENCLATURE

AI	Artificial Intelligence
ANFIS	Artificial Neuro Fuzzy Inference System
CSI	Michigan's Consumer Sentiment Index
CCI	United State's Consumer Confidence Index
INTC	Intel's Trade Symbol
MF	Membership Function
MSFT	Microsoft's Trade Symbol
NMSE	Normalized Mean Squared Error
PE	Processing Element
RMSE	Root Mean Squared Error
TSK	Takagi Sugeno and Kang Neuro Fuzzy Inference System

Chapter 1

Introduction

Moore's law is still in tack and thus processors are doubling in speed approximately every 18 months. This new power is very helpful with artificial intelligence, which was only a mere conception a few decades ago. Now, thanks to abundance of processing power, we can even combine artificial intelligence techniques in ways not possible just 10 years ago. Inference systems can learn patterns in megabytes of data in only seconds, thus allowing for more and more data to be learned by the machines. The ability to parse through tons of data is critical in the financial world as there are gigabytes of data, and the ability to understand and learn the non-linear patterns in an unsupervised manor is critical to success. This paper will delve into some of the most popular artificial intelligence methods used today. These methods are: Neural Networks, Fuzzy Inference System, Genetic Algorithm, and Decision Trees. Once the strengths of these artificial intelligence methods are explained, we can then try and create a hybrid intelligence system that realizes the strength of two or more of these intelligence systems by combining them into one significantly more advanced system.

1.1 Neural Networks

Neural Networks is an attempt at creating a computer that thinks in a manner similar to humans. The term, neural networks, was originally coined in the 1940's. Neural Networks can handle complex problems that have as many as 200 independent variables [23]. Neural networks are great for problems that are complex and contain

uncertainty. Neural Networks can even determine trends over time [18], which is a limitation of decision trees and many other artificial intelligence mechanisms. Time series analysis is critical to any financial model because we must learn how the prices changes over time and what inputs are most critical to future prices.

Database marketing is an area that would benefit from neural networks. Database marketing often has hundreds of independent variables, which is well suited for a neural network. Figure 1.1 shows what a neural network could look like. Independent variables are inputted to every node in the hidden decision layer and their output is passed onto the next decision layer (depending on how many layers have been set up). Once the output is determined, its result is compared to the actual outcome and the result is backward propagated into the systems and is weighted according to the systems learning rate.

Selecting the right learning rate and momentum are critical to the success of the neural network. Good programming can avoid the user from having to proceed through the very lengthy process of trial and error. *Neural Planner* provides a "Smart Start" option that will search for the ideal setting for the learning rate and momentum [26].

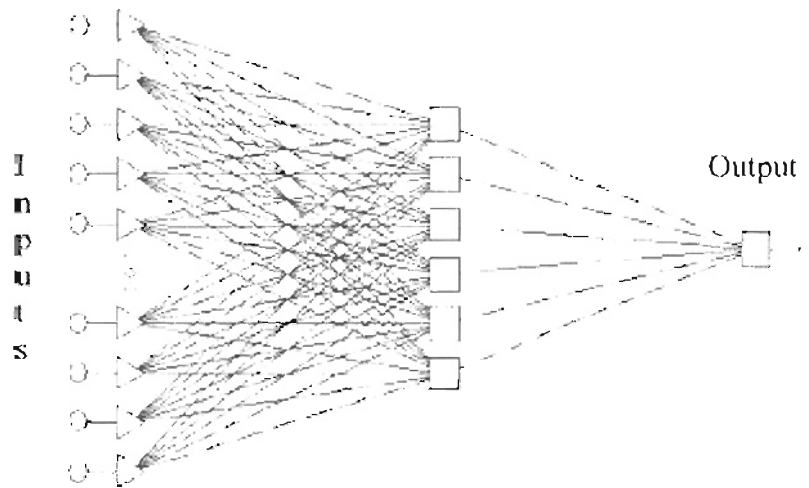


Figure 1.1 Neural Network

The two key points that must be followed when designing a neural network are: do not over train the neural network and pick a good starting point. If a neural network is over trained, it will memorize the noise of the dataset and not attain the true signal of the population. Also, if the wrong starting point is selected the results tend to be very poor as a local minima will be found. Luckily, good programming can solve both of these problems. The algorithm can check accuracy as it trains and as soon as the accuracy starts to dip, because of over training, the program can be rolled back to the best results. This is done in Weka [30] by setting a threshold variable to an integer x . When the network trains for x epochs and the performance doesn't get any better on the cross validation set, then the network is rolled back to the best result and stops training. To solve the wrong starting point, a program should choose several starting points at random and then pick the one with the best results. One final problem with a neural network is

the high price of software that runs this algorithm. ModelMAX, a tool used by many direct marketers can be an extravagant expense to many companies [24].

The positive side of using a neural network is it can adapt for areas of higher uncertainty and has the ability to solve larger problems. Neural networks are well suited for problems that are highly non-linear. These types of problems are very common in database marketing with hard to define variables such as customer satisfaction and even harder to define dependant variables such as customer loyalty. Another strength of neural networks is the ability to predict a continuous variable, whereas decision trees have problems with this topic. The ability to learn new situations and recognize trends is another reason that neural networks are popular. The ability of the neural network to reduce costs can easily outweigh the cost of buying the software for any mid-size or larger direct marketing company, making the decision to do so profitable and intelligent. The cost of the software will eventually come down, and today there are many tools that are available for academic use; however, many of the free tools, such as Weka and Fuzzy Cope 3, are not very user friendly. Plus they lack much of the power of the commercially available software.

The overall performance of neural networks are also improved though the use of different technique to find the global minimum. These new methods use gradient-based descent techniques such as conjugate gradient descent. To begin with, there is the Newton method, which attempts to find the steepest descent. Second, there is the Levenberg-Marquardt variant, which is discussed in the next section, and lastly there is the Quasi-Newton Method. All of these algorithms arrive at an optimal solution much

faster than back propagation and require fewer epochs resulting in less expensive hardware being required.

1.1.1 Conjugate Gradient

The conjugate gradient is a method, which uses an approximation of the second order derivative without actually calculating the second derivative. This process was originally discovered in the 1960s for solving linear systems [18]. This method is exceptionally fast and thus is very useful with solving large data sets or when many networks need to be built. The gradient uses a vector of previous points to determine the conjugate direction. Imagine that you are standing on step embankment that leads to a river that is going to your right and empties into the ocean. The ocean represents the minimum and so the best path is a straight line to the ocean. If you used a steepest descent method you would head straight down to the river first and once you got there turn head down the river. A conjugate gradient method would send you directly towards the ocean because it makes use of the second derivative. [4]

1.2 Fuzzy Logic

Fuzzy logic gives a set of natural language rules that are easily understood by humans. Fuzzy logic, which is based on the Fuzzy Set theory, was original discovered by Zadeh in 1965 [35]. Fuzzy set theory can be viewed as a generalization of the classical set theory [20]. The primary advantage of fuzzy logic is its readability [28]. Fuzzy Inference systems have been successfully used in many different areas such as automatic

control, data classification, decision analysis, time series prediction, and pattern recognition [16]. Petrovic *et al.* [24] use fuzzy logic in a multiple objective decision model for a manufacturing plant. The rules for a fuzzy system can be generated either by interviewing experts in the field or mechanical mechanisms used in a fuzzy inference system, which uses supervised learning to recognize patterns in the data. A typical fuzzy rule is given below:

If (*customer has high credit score*) and (*customer has high income*) then (*grant loan*).

Equation 1.2

In the above example it is obvious that there is no absolute definition for either statement. Not everyone is going to agree that \$50,000/year is a high income. Even if the limit was moved up to \$200,000/year a few people would still not consider that to be a high income. Thus, we must define a membership function to describe the fuzzy relationship between having a high income and any specific income. The membership function below helps determine the degree of “if” relating to whether or not a customer has high income.

$$\%HighIncome = \frac{1}{1 + 2 \left(\frac{30,000 - income}{30,000} \right)} \quad \text{Equation 1.3}$$

In the above example a customer with an income of \$30,000/year is said to have high income with a certainty of 50%. Similarly a customer with an income of \$90,000/year is said to have a high income with certainty of 80%. Knowing the certainty,

allows the system to weigh rules within the system and give preference to rules that the customer fits better.

As opposed to traditional probability theory not all possibilities must add up to 100% [29]. For example, let us say that there are two cases: a person is rich or a person is poor. It is possible that according to a membership function, Jack is rich (CF = 0.65) and Jack is poor (CF = 0.20). Except $0.65 + 0.20 \neq 1.00$ and this case is possible in fuzzy logic but not in probability theory.

Fuzzy Logic is used today in many different real world applications. One such example is an Anti-Lock braking system [29] where instead of the traditional anti-lock braking system, which uses an on/off pumping action to unlock the wheel, there are about 18 sensing factors. When a sensor begins to come close to being locked, the pressure on the brake is slightly released and therefore maximizing the friction helping the vehicle to stop sooner. Fuzzy Logic is commonly used in control systems such as a subway control system or other motor controls and navigation [5]. One system designed by Hall (1987) provided for small and medium size businesses to plan strategically for a single product based on how the company answered five questions regarding their strength and weakness. The system was known as STRATASSIST [29].

1.3 Genetic Algorithms

John Holland at the University of Michigan (1975) was the first to propose genetic algorithms. Genetic Algorithms have been used in modeling exchange rates [15] through a use of a multi-agent system. Genetic algorithms loosely mimic the concept of natural selection. Each member is made up of a chromosome, which is normally a binary

string. This chromosome defines the characteristic of the member of the population and that allows the algorithm to determine its fitness. A population is a group of members and changes from generation to generation through methods such as mutation and crossover. The fitness function is used at every generation to see which members are fit and most likely to survive to the next generation through a crossover operation that can be thought of as mating.

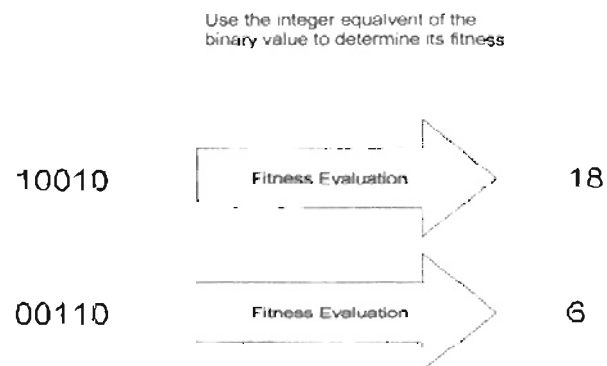


Figure 1.3.1 Binary String Representation

The process of creating genetic algorithms goes like this: Find a way of representing a member of the population at a bit string, this is known as **Encoding Schemas**. The second step called **fitness evaluation** finds a way of evaluating the fitness of this member. Step three known as **selection**, selects the members of the population that are the most fit. Finally the fourth step called **crossover**, mates the fittest members.

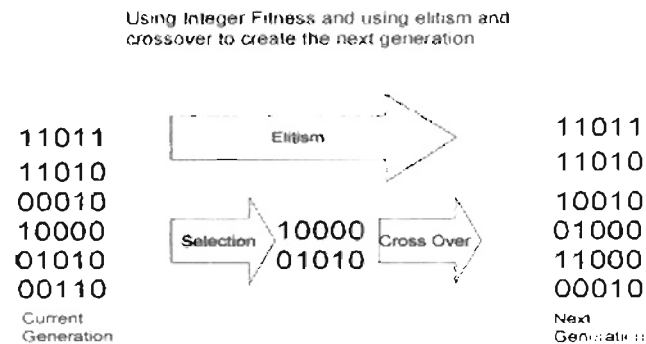


Figure 1.3.2 Elitism

We must also introduce some randomness to ensure more of the search space is covered and this can be done by **mutation**. Mutations can be done by simply flipping a bit in the string to produce a new mutated string. Mutation does not occur in every generation and can be changed by adjusting the mutation rate. It is also critical to avoid local minima. If the mutation rate is too high the generation will lose the chromosomes that were created through the selection process and the search will be very similar to that of a random search. In addition if the mutation rate is too low the algorithm is likely to get stuck in local minima, but generally speaking it is best to keep the mutation rate low. Once these steps are done go back to step 2 and repeat until the desired fitness is achieved.

Another common practice is called **elitism**, which keeps the very best members of the current generation and puts them in the next generation without any crossover or mutation. The rest of the population is then used for crossover and mutation to create the remaining members of the next generation.

Genetic Algorithms are an exceptionally powerful tool, as they are very effective at searching a predefined search space, and this ability helps genetic algorithms to be used in a hybrid manor with other tools.

1.4 Decision Trees

A decision trees can be used to predict an outcome for dependant variable based on many independent variables. The root node of the tree contains the most significant independent variable. As the tree is traversed, the node becomes less important to the outcome until a leaf node is reached and an outcome is predicted. Figure 1 below shows a simple decision tree that could be used by any direct marketer.

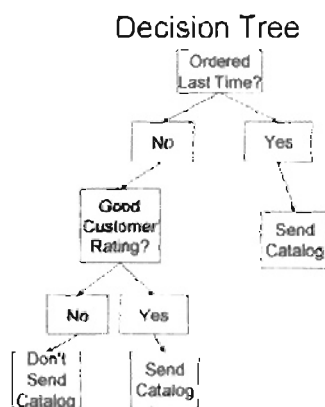


Figure 1.4 Example Decision Tree

The creation of a decision tree is done by determining the relative importance of each independent variable on the dependent variable. One such method of classification is the C4.5 tree. Equation 1.4 and 1.5 show how the importance of each independent variable is qualified. (Let p be the number of elements in class P and n be the number of

elements in class N .) For example, class P could be the people to receive catalog and class N could be the people who do not receive a catalog.

$$I(p, n) = -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n} \quad \text{Equation 1.4}$$

Set S is partitioned into sets $\{S_1, S_2, \dots, S_v\}$. For Set S_i , p_i is the number of p 's in the set and n_i is the number of n 's in the set. $I(p, n)$ is the importance to model. The higher the $I(p, n)$ is the better this combination is for a split. A value of zero means to attach no importance to $I(p, n)$ and a value of 1 means n and p have ideal values. $Gain(A)$ is the amount of information gained for an attribute A with a highest gain being the attribute to use as the root.

$$Gain(A) = I(p, n) - \sum_{i=1}^v \frac{p_i + n_i}{p+n} I(p_i, n_i) \quad \text{Equation 1.5}$$

The weakness of a decision tree is that it cannot adapt to trends in the data or changes that occur over time. If particular groups of people stop ordering from catalogs because of a new competitor, for example the Internet, the decision tree created will not capture this trend. A possible solution to this problem would be to rebuild the decision tree on regular intervals to catch any business or customer trends.

A benefit of the decision tree is that the results can easily be explained. For example a bank can easily explain to a customer why they were declined using a decision tree. Decision trees can be very accurate when enough data is available for training, even beating human experts. This process can be seen when looking at a bank trying to determine qualified loan applicants. American Express UK loan officer's manually determined if applicants in the grey area would be accepted for a loan, but they

where right only 50% of the time. After a decision support system was implemented, that used a decision tree, the success rate increased to 70% saving the company money [29].

1.4.1 Classification and Regression Tree

CART (Classification and Regression Tree) is a special case of a decision tree that can be constructed by examining data in a systematic approach; the CART grows through a series of splits. A CART determines the importance of each variable before adding a splitter in the tree. Starting from the root node an exhaustive search is performed on all inputs to determine which input creates the least error when picked. After finding the split, two disjoint sets are created according to the split and each set is then passed on down the tree and the process repeats with the best split picked at each level. This process terminates when the gain for each level does not meet a threshold or when a predefined error is reached.

This technique can also be useful in determining which variables are the most important in a data set and thus have the greatest impact on the output. The most important variables will be contained towards the top of the tree, and the least important will not exist in the tree. In this study a CART was designed to help pick which inputs have the most impact and which inputs could be removed from the models with very little impact on the performance of each network.

1.5 Objective of Study

The main focus of this study is to compare different performances of artificial intelligence paradigms on predicting the direction of individuals stocks, and how hybrid intelligence can be used to better solve problems. The first algorithm examined is Artificial Neural Network using conjugate gradient descent algorithm. The second algorithm used is a straightforward back propagation method. A Mamdani Neuro Fuzzy inference was built and then the membership functions were modified using back propagation and a Genetic Algorithm. This showed how effective Genetic Algorithms could be and provide a comparison with Takagi Sugeno Neuro Fuzzy model. The ANFIS model is based on Takagi Sugeno Fuzzy Inference System and was compared with a neural network on both performance and training time required [18]. A Neuro Genetic solution was also built which had a Neuro network as its base and used genetic algorithms to determine the strength of each input, thus eliminating some of the unnecessary inputs; this created a hierarchical hybrid intelligence system. Once all networks were built, the last part of the experiment was to use the best result from the above mentioned algorithms in a simulation to determine how much profit could be made using this method versus a simple buy and hold technique. For the simulation, any money left on the sidelines will earn a return of Prime rate. It has been suggested that stock prices take a random walk; if that is the case our network will do no better than the buy and hold technique. This study will disprove the random walk theory.

1.6 Significance of Study

The most recent studies compare indexes such as the S&P 500, NASDAQ, and the Dow Jones [2][8][28][31][32]. The experiments done in this project examine the chaotic behavior of actual companies that tend to be less stable and thus harder to predict. Studies have also shown that using direction as compared to prediction can generate higher profits, [8] and this study will try and capitalize on that idea. Also the prediction will examine a more realistic situation where an investor has the choice between multiple stocks, in this case 2, and chooses the stock that is mostly likely to increase in value. The experiments also compare many hybrid techniques and their abilities to predict a categorical output. The ability to predict the direction of the stock prices is the most important factor to making money using financial prediction. All the investor really needs to know is to buy if the stock is going up in value and to sell if it is decreasing in value.

1.7 Data Set and Tools Used

The data is comprised of prices for Microsoft and Intel Corp from January 2nd, 1990 until August 5th, 2003. This time period was a very violent time in the stock market, it include 2 recessions, a dot COM boom, and a dot COM bust. Also world events like the September 11th attack where included in the original data set. Figure 1.7 shows the violent nature of the Microsoft and Intel during the selected time period. It also shows the wider range and more chaotic behavior caused more problems for the networks in predicting Intel's stock price.

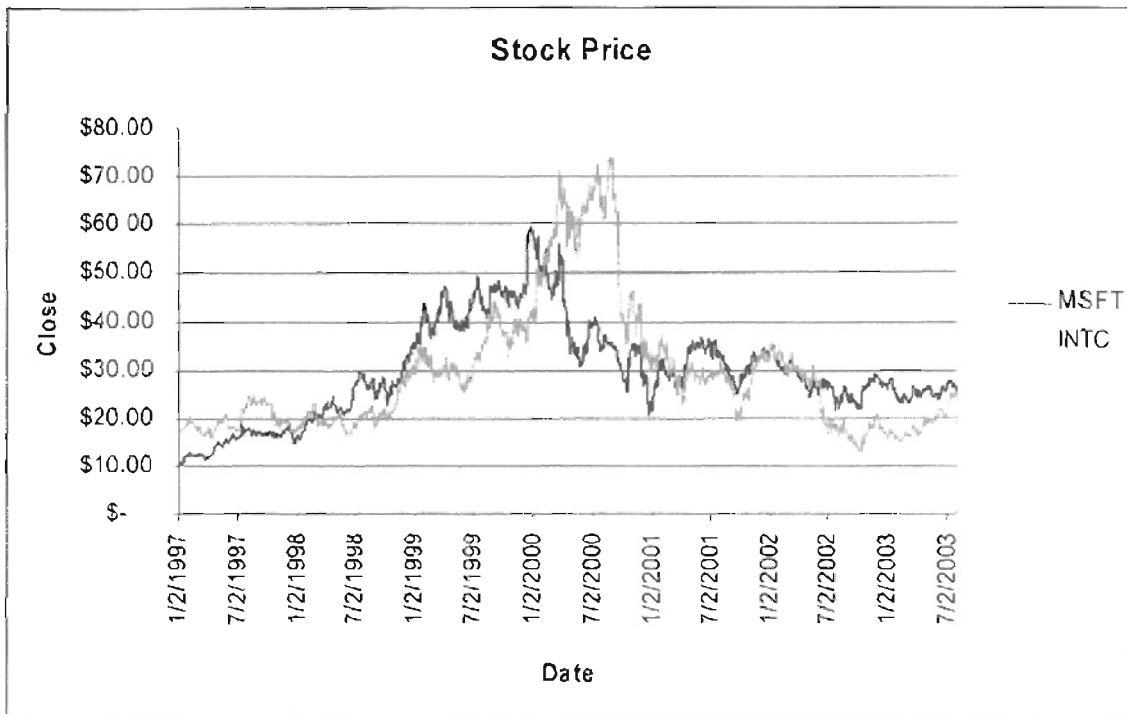


Figure 1.7 Microsoft and Intel Stock Price

Information contained for each daily report is the opening price, closing price, low price, high prices, and volume of shares traded. External data was also gathered to aid the network in its training. Economic indicators such as the current Prime rate, Michigan's Consumer Sentiment Index, and the United States Consumer Confidences where added. The software used to train the Mamdani Neuro Fuzzy and Mamdani Neuro Fuzzy with Genetic Algorithm is FuzzyCope 3[10]. The neural network using both conjugant gradient descent and traditional back propagation were trained using Neuro Solutions[22]. All testing was done on an Athlon 2000+ with 1 GB of memory. A java application was written to transform .CSV files into .IRN files that can be understood by FuzzyCope to save time from manual translation and improve the accuracy so that no row is lost. Similarly it was required to transform .csv into .RCJ. files, which FuzzyCope used to perform a test once the network was trained. Another stone-alone java

application was developed to randomize the rows of a .CSV file to ensure the network fully randomized this could have also been done using the preprocessing built into Neuro Solutions. FuzzyCope3 is designed to perform regression testing only and not classification. Thus it was necessary to writing an application to transform the predicted value to 0 or 1 and then do a comparison for accuracy. All Java applications were developed using JDeveloper by Oracle.

Chapter 2

Literature Review

2.1 Hurst Exponent

Some papers have used the Hurst Exponent [9][12][32][33] to prove that the data is not completely random but in fact has the correspondence between the input and the output data. The Hurst Exponent was originally discovered by Hurst et al. [14] in 1965. The Hurst Exponent can show the degree of correlation. If the exponent is 0.5 the data is completely random and thus no network will be able to predict the output and thus it is a waste of time to attempt to learn any pattern in the data. The closer the Hurst Exponent is to one, the greater the correlation between the input and output, and a Hurst Exponent of less than 0.5 means that the input and output are indirectly proportional. It is important to note the Hurst Exponent is confined to the range of 0 to 1.

$$\text{HurstExponent} = \frac{\log(R/S)}{\log(N)} \quad \text{Equation 2.1}$$

S is the standard deviation of the time series before normalization

R is the maximum and minimum cumulative deviations of the observation has compared with the mean of the series.

N is the number of observations

$$R_N = \max_{1 \leq t \leq N} [x_{t,N}] - \min_{1 \leq t \leq N} [x_{t,N}] \quad \text{Equation 2.2}$$

$x_{t,N}$, the cumulative deviation, is describe by

$$x_i = \sum_{n=1}^i (x_n - \mu_N) \quad \text{Equation 2.3}$$

μ_N is the mean of x_n for all N elements.

The Hurst exponent can be very useful in any set and allows a method of comparing sets of data. For example, a set with a Hurst Exponent of 0.55 is very difficult to predict and any network with decent results should be great. However, a data set with a Hurst Exponent of 0.95 should expect the network to be extremely accurate to be considered good.

2.2 Scaling and Normalization

It is important to smooth out the data and help the network to learn the signal of each input and not just memorize a single input with a very large numerical value given to it. Thus normalization of the data can help any network better obtain the correct signal of the network. There are many methods available today but no research has proven one to be superior to the others in all cases. Below is an example of one way of normalizing data.

$$n_i = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}} \quad \text{Equation 2.4}$$

In the above equation x is the original series with x_{\max} being the greatest value in the series and x_{\min} being the smallest value in the series. The new series denoted n will be on the range from [0,1]. The example below shows a series of stock prices before normalization (x -series) and after normalized (n -series).

x-series	n-series
35 25	0.5478
37 25	0.9462
37 52	1.0000
37 5	0.9960
34.87	0.4721
32 5	0.0000

Table 2.2 Normalization of a series

The above set of data shows how a data set can be spread out by using normalization, making it easier for the network to understand.

2.3 Overfitting and Overtraining

Many networks will memorize the patterns of the test set as opposed to learning the signal of the set. This is normally caused by over training. Setting a threshold in the cross validation data set can prevent this problem. after a certain predetermined number of meager results the network would stop training and would revert to the most ideal network. For example, if the accuracy of predicting against the cross validation set becomes worse for 20 epochs in a row, then the network has complete training and uses the weights specified by the network with the best prediction against the cross-validation set. Similarly overfitting can be caused by have too many neurons to define the problem. This problem can be fixed by reducing the number of hidden neurons. There have been a few papers published that have discussed a general rule of thumb to determine an approximate number of neurons. One such rule of thumb is the Freisleben rule of Thumb

[33]. Which states the correct number of hidden neurons is a multiple k times the number of inputs (n) minus one.

$$\# \text{ neurons} = (k * n) - 1 \quad \text{Equation 2.5}$$

A second rule of thumb popular in newsgroups is

$$\# \text{ neurons} = \sqrt{\text{inputs} * \text{outputs}} \quad \text{Equation 2.6}$$

$$H_{n+1} = \ln(H_n) \quad \text{Equation 2.7}$$

The Baum-Haussler rule for determining the correct number of hidden neurons is defined by the following function.

$$\# \text{ neurons} \leq \frac{N_{\text{records}} * E_{\text{tolerance}}}{N_{\text{inputs}} * N_{\text{outputs}}} \quad \text{Equation 2.8}$$

Using any of these rules of thumbs can prevent the networks from memorizing and thus catching the actual signal of the neural network. N_{records} is the number of records and $E_{\text{tolerance}}$ is the Error tolerance. N_{inputs} and N_{outputs} are the number of inputs and number of outputs respectively. H_n is then number of hidden neurons in layer n .

Both rules of thumbs give a good guideline, but there is no guarantee of the best number for every dataset. Hence it is best to try many different networks [6] to see which produces the best result for the data set in question.

2.4 Learning

Supervised learning is most common form of training a network, and it done by providing the network with both the input and the expected output. Thus the system can learn from known truths about the data. This learning technique should be used

whenever there is sufficient data with both inputs and outputs. When know outcome is available it is ideal to use supervised learning [26].

Unsupervised learning means the system attempts the recognized patterns in the data and doesn't have the expected outputs. Self-organizing maps are a common usage of unsupervised learning when the network attempts to recognize clusters of data and to group them according to similarities with other members. Unsupervised learning is done when the system doesn't know the expected output, and the system is then supposed to learn the patterns. A common tool used for unsupervised learning is a self-organizing map.

2.5 Splitting the Data Set

The data set should be split into 3 different sets: Training set, cross-validation set, and the testing set. A common break down is to use $2/3$ of the data set for training, $2/15$ for cross-validation, and $3/15$ for testing set. The above-mentioned split was used by Yao *et al* [32]. A two-thirds training and one-third validation set was used by Yao *et al* [32]; however, this not a good design because no cross validation is done thus preventing the network from over training. There is no guarantee for any split to be perfect for all datasets. Thus is it advisable to try all rules of thumbs and to vary the size of the sets to see what produces the best results. It is crucial to use a cross-validation set regardless of the size of the split to prevent the network from over training, also known as over-fitting the data.

2.6 Recent Trends

Many papers have dealt with input selection when it comes to mapping financial indexes and stocks [2][8][28][31][32]. Inputs have been broken into two different types of inputs, financial and political (which tend to be qualitative). Kuo *et al* [19] uses a genetic algorithm base fuzzy neural network to measure the qualitative effects on the stock price. Variable selection is critical to the success of any network and 5 key parts of the financial viability of a company were identified by Quah *et al* [26] as yield, liquidity, risk, growth, and momentum factors. These variables are widely available in qualitative form such as the P/E ratio can be used for yield and the return on equity could be used for growth etc. Macroeconomic factors such as inflation and short-term interest rate [8] have to shown to have direct impacts on the stock returns.

A better measure of fitness, which considers profit [31] has been suggested to replace a root means squared error. Yao and Poh [32] showed an example where a model with a low NMSE had a lower return than a model with a higher NMSE. Brownstone [6] recommends using percentages to measure performance so that the result can be better understood by traders and other people that might need their research and are not experts in the field. Chen *et al* [8] used a 68-day sliding window to predict the next day's price of the index. Commission is commonly overlooked when doing research relating to stock market prediction; however, if any model is actually implemented it is going to incur fees which could greatly affect the profit predicted by the model. Chen *et al* [8] considers 3 different levels of commissions and how it would affect the best buying strategy used by investors. Simulation [34] has been used to show how these models can produce profits on real world testing data that is not seen by the network.

Chapter 3

Hybrid Intelligence Systems Architecture

3.1 Stand Alone

“Stand-alone models consist of independent software components which do not interact in any way [1].” These systems can work in a parallel environment to allow the user to determine which model is the best fit to learn the signal of the data. Once the stand-alone system has aided in picking the best system that system would then be developed by itself to make the best possible single intelligent system. The advantage of this model is it is fast to build and uses software that is already available. A disadvantage is the system doesn't incorporate any strengths of the discarded system, and as a result the performance is not any better than a single intelligence system.

3.2 Transformational Hybrid Intelligent System

The system begins as one system and then transition into an entirely new system. Thus once the model is built on a system is required to be worked on. Like the stand-alone model this system suffers from not being able to use the strength of both systems. These systems also tend to be application-oriented [1]. A disadvantage of this system is there is not any really available software that supports this type of architecture.

3.3 Hierarchical Hybrid Intelligent System

The Hierarchical Hybrid Intelligent System uses the strengths of multiple types of artificial intelligence systems to produce the best possible intelligent systems. The design is broken up in layers with each layer having a single intelligence system to determine what is best at that layer. A common usage of hierarchical hybrid intelligent system is to use an evolutionary algorithm to produce the inputs or the best settings for another artificial intelligence system. Leigh (Forecasting the NYSE composite index) used a genetic algorithm to determine which of the 22 inputs were the most useful and which could be eliminated to generate a better R-squared correlation. The findings from the genetic algorithm were then used to create a better neural network. A hierarchical hybrid intelligent system is when the system begins

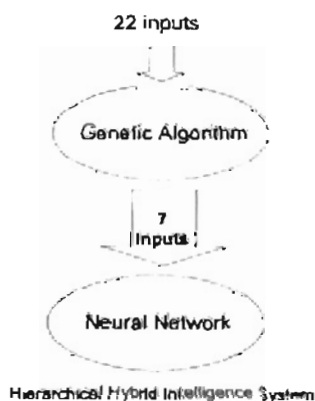


Figure 3.3 Hierarchical Hybrid Intelligent System

as one type of intelligent system, and then is transformed into a different type with the final product having no proof of ever being of the first type of intelligent system. The design shown in figure 3.3 was used in this study to reduce the number of inputs from 16 to 9 which were then given to the neural network for training. Hierarchical hybrid

intelligence systems show dramatic improvement over using a single intelligent system. This allows the user to focus on the bigger picture, and the computer can figure out the details of the design such as how many hidden neurons should be used.

3.4 Integrated Intelligent System

Integrated Intelligent Systems use fused architectures [1] that provide a single model with the best characteristic of all models. There are numerous advantages to this type of model. Integrated Intelligent systems provide increased performance and are more robust because it is both noise resistant and has the ability to explain itself. The biggest disadvantage of this system is its complexity; to design this type of system is a complex undertaking for any company. Nevertheless these types of systems are needed by companies and so are actually being developed. The hope is that as more Integrated Intelligent systems are developed, the aforementioned problems will begin to dissipate. One such model that is currently available is FuzzyCope, which provides a Neuro-Fuzzy model, and it is available at [10]. Similarly Neuro solution has an ANFIS (Artificial Neural Fuzzy Inference System) that uses an integrated intelligent system [22]. Hierarchical design has been very popular in recent studies Abraham [1] discusses a 5-layered system that evolves Neuro-Fuzzy-Evolutionary System (EvoNF). This type of system would require the largest computers systems available today to build its model, which is a huge disadvantage of the hierarchical architecture. The cost of the system to run these programs can be huge, but the biggest strength of these systems is their performance once the model has been built. Mamdani Fuzzy Inference shown in figure 3.4 is an example of an integrate intelligence system.

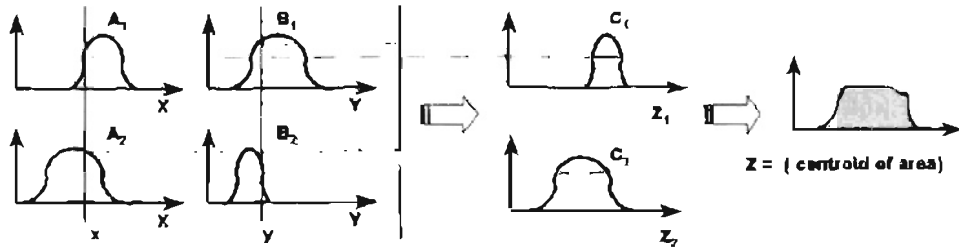


Figure 3.4 Integrated Hybrid Intelligent System [3][4]

3.5 Conclusion of Hybrid Intelligence Systems

The most interesting of the intelligence systems are the Integrated and Hierarchical hybrid because these two methods provide the most significant performance improvements and can realize the strength of many different intelligent systems. However, we are not limited to having to choose one of these two systems, in fact, it would be perfectly reasonable to create a **Hierarchical Integrated Hybrid Intelligence System**. This system would contain layers as in the hierarchical system, with one or more layers containing an integrated system.

Chapter 4

Hybrid Intelligence Systems

4.1 ANFIS

ANFIS, Adaptive Network-based Fuzzy Inference System have been shown to provide better result than artificial neural networks and fuzzy models [16].

A common model used today in ANFIS is the Takagi Sugeno Fuzzy Model. In the Sugeno model each different rule has its own function.

$$\text{if } (x \text{ is } A) \text{ and } (y \text{ is } B) \text{ then } z = f(x,y) \quad \text{Equation 4.1.1}$$

In the above function $f(x,y)$ is a crisp function and the sets A and B are fuzzy sets, thus they don't have absolute members, but rather a degree of membership. Jang [16] gives an excellent example of an ANFIS with only 2 inputs. The diagram below shows the procedure for inputs x and y . Each layer is then described below.

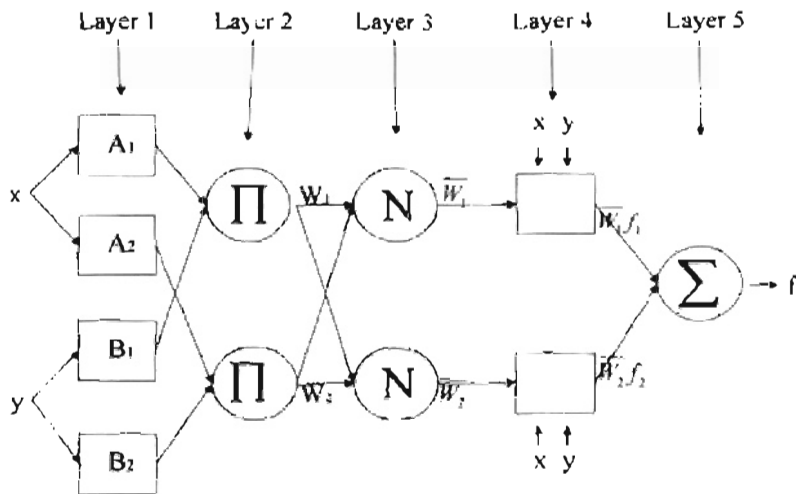


Figure 4.1 ANFIS [16]

The ANFIS consists of 5 different layers described below:

Layer 1 (Membership Function): This bell shaped graph determines if x is in A and to what degree it is a member. The bell shape of the graph can be manipulated by changing a value of any variable. Thus the end result is a bell shape that better matches the real world.

$$\mu_A = \frac{1}{1 + \left| \frac{x - c_i}{a_i} \right|^{2b_i}} \quad \text{Equation 4.1.2}$$

$a, b,$ and c are constants that determine the shape of the bell. A is the linguistic label (tall, short, etc) that is associated with the node.

Layer 2 (Firing Strength): Every node in layer two corresponds to the firing strength of a rule. Any T-norm operator could be used in this layer. Two common T-norm operators are the AND and MAX functions.

$$O_{2,i} = w_i = \mu_{A_i}(x)\mu_{B_i}(y), i = 1,2 \quad \text{Equation 4.1.3}$$

Layer 3 (Normalized Strength): In layer three calculate a normalized firing strength so that the output one node doesn't overshadow all other nodes.

$$O_{3,i} = \overline{w_i} = \frac{w_i}{w_1 + w_2}, i = 1,2 \quad \text{Equation 4.1.4}$$

Layer 4 (Adoptive function): Each node has a node function defined by w_i , normalized firing strength, and by 3 new constants p , q , and r . These three parameters are referred to as the consequent parameters.

$$O_{4,i} = \overline{w_i} f_i = \overline{w_i} (p_i x + q_i y + r_i) \quad \text{Equation 4.1.5}$$

Layer 5 (Calculate Output): A summation of all input signals is used in this signal node to compute the overall output as describe in the formula below.

$$\sum_i \overline{w_i} f_i = \frac{\sum_i w_i f_i}{\sum_i w_i} \quad \text{Equation 4.1.6}$$

The Mamdani fuzzy inference system is a special case of the Sugeno fuzzy model in which the order of the model is zero. Since the order of the system is zero then f is a constant.

4.2 Neuro Fuzzy

Neuro fuzzy systems are an attempt to combine natural linguistics used in fuzzy inference Systems with the proven capabilities of artificial neural networks [13]. The combined system's goal is to be more transparent like a fuzzy system giving the users a list of general and understandable rules while at the same time building in the ability of a neural network to predict non-linear trends in data. Central to this idea, is building a bridge from fuzzy logic using membership functions and artificial neural networks that possess quantitative adaptive number crunching power. Castellano *et al* [7] designed a Neuro-fuzzy model where the parameters of the fuzzy rules base were configured by a two-phase learning of the neural network.

4.2.1 Takagi Sugeno Neuro Fuzzy

A common fuzzy inference system (FIS) used today is Takagi Sugeno fuzzy inference system [27]. The idea was to formalize a systematic method for generating rules that a computer could use for any given data set [17]. Takagi Sugeno FIS has rules that follow the format:

$$\text{if}(\text{pressure is high}) \text{ then volume} = 2 * \text{pressure} \quad \text{Equation 4.2.1}$$

In a Takagi Sugeno FIS the consequent is a crisp function that can be expressed in terms of $f(x)$. A first-order Sugeno fuzzy model occurs when the function f is a first order polynomial. A zero-order Sugeno fuzzy model occurs when the function f is a constant. This can also be viewed as a special case of the Mamdani fuzzy inference system [17][18].

Takagi Sugeno has 2-step process of learning that occurs for every epoch through the training set. The first step holds the membership functions constant and updates the input patterns learned according to an iterative least squares method. The second part of the learning updates the membership function while the input patterns are held constant [3]. These steps provide for a very efficient learning tool.

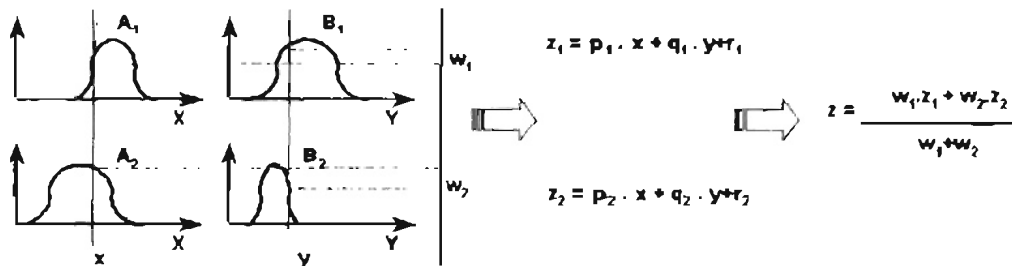


Figure 4.2.1 TSK Fuzzy Inference System [3][4]

4.2.2 Mamdani Neuro Fuzzy

A_i and B_i are input fuzzy sets and the result is the output of the fuzzy set [3][21].

A supervised learning technique is used to learn the membership functions in a Mamdani Fuzzy Inference system. The Mamdani system has 6 layers instead of 5 that are in Takagi Sugeno Model. The first layer is for the inputs. The second layer is a fuzzification layer. The third layer is the rule antecedent layer. Then the fourth rule is

the strength normalization rule, and the fifth is the consequent layer rule. The final layer in the Mamdani Neuro Fuzzy system is the rule inference layer.

4.3 Input Selection

In real world problems, there can be hundreds [16] of different possible inputs for any artificial intelligence system. For instance, in a financial model the inputs are not just limited to the stock price, dividends, and volume trade of a particular stock or index in question. However the indexes could extend to the overall performance of the market, the consumer confidence, Federal Reserve interest rates, or even world policies indicators such as how is the current war is proceeding. Once all these possible inputs have been found it is good to find a mechanism for reducing the sheer number of inputs, as having too many inputs can cause many problems such as complexity of computation and less transparency of the underlying model. Four rules have been found as a rule of thumb to guide input selection by Jang [17], and it is reasonable to believe that these rules are generalized enough that they could work for other models.

- 1) Remove noise/irrelevant inputs
- 2) Remove inputs that are dependant on other inputs
- 3) Inputs that create a more concise and transparent model
- 4) Reduce time for model construction

Chapter 5

Hurst Exponent on Data

Once the data was transformed in the most viable form to use in all the networks, the Hurst Exponent [9][12][32][33] was calculated to show that both the prediction is possible and that the prediction is going to be very difficult. The time series used to calculate the Hurst exponent consisted only of the percentage change in price from the previous day and the actual value was not used.

$$\text{PercentageChange} = \frac{P_{\text{today}} - P_{\text{yesterday}}}{P_{\text{yesterday}}} \quad \text{Equation 6.1}$$

This equation was performed on all 1398 days in the testing set from January 1st, 1997 to July 31, 2002. Then the $x_{t,n}$ was calculated for all days using equation 2.3. Once that was done, then the R_N (1.3387 for MSFT) could be found using equation 2.2. The standard deviation for MSFT was found to be 0.02742 that gave us all the information needed by equation 2.1 to determine the Hurst Exponent to be 0.537 for MSFT. This proves both points earlier stated. The data is not a complete random walk because neither network had a Hurst Exponent of 0.5. And second, it shows that good performance will be very difficult to achieve for any network, as the network is nearly random. Similar tests were run on Intel's data to produce a Hurst Exponent of 0.513. Thus based on the Hurst Exponent, Intel's data is more random and should be more difficult to produce good results. Figure 5.1 shows how the calculation for Microsoft was calculated to find a

t	Price Change	$X_{t,n}$
1	-0.0025	-0.0035
2	-0.0033	-0.0078
3	0.0642	0.0554
4	0.0585	0.1129
5	-0.0734	0.0386
6	0.0746	0.1122
7	-0.0946	0.0167
8	-0.0413	-0.0256
9	-0.0302	-0.0568
10	-0.0174	-0.0752
11	0.0149	-0.0613
12	-0.0108	-0.0731
13	-0.0012	-0.0753
14	-0.0197	-0.0960
15	0.0127	-0.0843
16	-0.0181	-0.1034
17	0.0053	-0.0990
18	-0.0351	-0.1351
19	0.0580	-0.0781
20	0.0078	-0.0713
21	-0.0232	-0.0955
22	-0.0371	-0.1335
23	-0.0040	-0.1385
...
1398	0.0364	0

Mean = 0.000985

$\max_{1 \leq t \leq N} [x_{t,N}] = 0.1129$

$\min_{1 \leq t \leq N} [x_{t,N}] = -1.2257$

R = 1.3397

S = 0.0274

N = 1398

$H = \frac{\text{Log}(R/S)}{\text{Log}(N)} = 0.536$

Table 5.1 Hurst Exponent Calculations

Hurst exponent of 0.5368.

Chapter 6

Data Preparation

A key when designing any artificial intelligence system is to present the data in the most meaningful and understandable format for the algorithm to understand. The 3 steps of presenting the data to the network to the system is to choose the best inputs, remove misleading or corrupt data rows, and transform the data.

6.1 Input Reduction

There are 16 inputs to begin with in each data model. This must be reduced to aid in the performance time of the neuro fuzzy engine since execution time grows exponential with the number of inputs. It takes about 3 days to train an ANFIS with five inputs and five MFs per input, thus the number of inputs must be reduced to no more than five. Also results with neural network tend to be better when unnecessary inputs are removed and duplicate or similar inputs are eliminated. This is shown to be the case in the testing of these networks. The original 16 inputs that were considered are: Consumer Confidence Index, the prime rate, Michigan Consumer Sentiment Index, price -1 (price yesterday), price -2 (price the day before yesterday), price -3 (price 3 days ago), price -4 (price 4 days ago), price -5 (price 5 days ago), volume -1 (volume of trades yesterday), low -1 (low price yesterday), high 1 (high price yesterday), open -1 (open price yesterday), low -2 (low price the day before yesterday), high -2 (high price the day before yesterday), open 1 (open price yesterday), and volume -2 (volume day before

yesterday). Several models were used to determine the least important inputs and then these inputs were removed before a more accurate and iterative approach could be used to determine the final inputs. Models were built using decision trees (CART) to determine which variable offered the most “gain.” Linear regression models were also designed to determine which inputs contributed the most to the model. Significance towards the price model was built using correlation models to determine each inputs contribution, which is shown below in figure 6.9.

Variable	Spearman Rank Order Correlations									
	Marked correlations are significant at $p < 0.05000$									
	Vol-1	Vol-2	Price -1	Price -2	Price -3	Price-4	Price-5	Prime	CCI	Price
Vol -1	1	0.876	-0.0138	-0.052	-0.079	-0.065	-0.071	-0.12	-0.045	-0.0431
Vol -2		1	-0.0402	-0.013	-0.053	-0.082	-0.064	-0.12	-0.046	-0.0445
Price -1			1	-0.023	-0.019	-0.016	-0.018	0.008	-0.015	-0.0289
Price -2				1	-0.023	-0.018	-0.016	0.008	-0.011	-0.0082
Price -3					1	-0.025	-0.017	0.009	-0.007	-0.0176
Price -4						1	-0.023	0.011	-0.007	-0.0196
Price -5							1	0.007	-0.011	-0.0045
Prime								1	0.709	-0.0062
CCI									1	-0.0207
Price										1

Table 6.1.1 Spearman Correlation

Also a genetic algorithm with a population of 50 was trained for 100 generations, and the results were examined to see what variable it had chosen to remove from the data model. Each network was trained for 10,000 epochs with a threshold of 500. The Mutation rate used was 0.1, uniform crossover, Roulette was used for selection, and rank basis fitness was used. A progression of generational was used instead of a steady state. After building a network the sensitivity for each input can be test by hold all variables constant except the one variable in question, then the network is feed a series of values for this variable. How the outputted result changes for each different input gives us the

sensitivity for this variable. The Sensitivity about the Mean test was conducted to see how important each input is relative to a particular neural network. The result for Microsoft data set is shown below in figure 6.1.

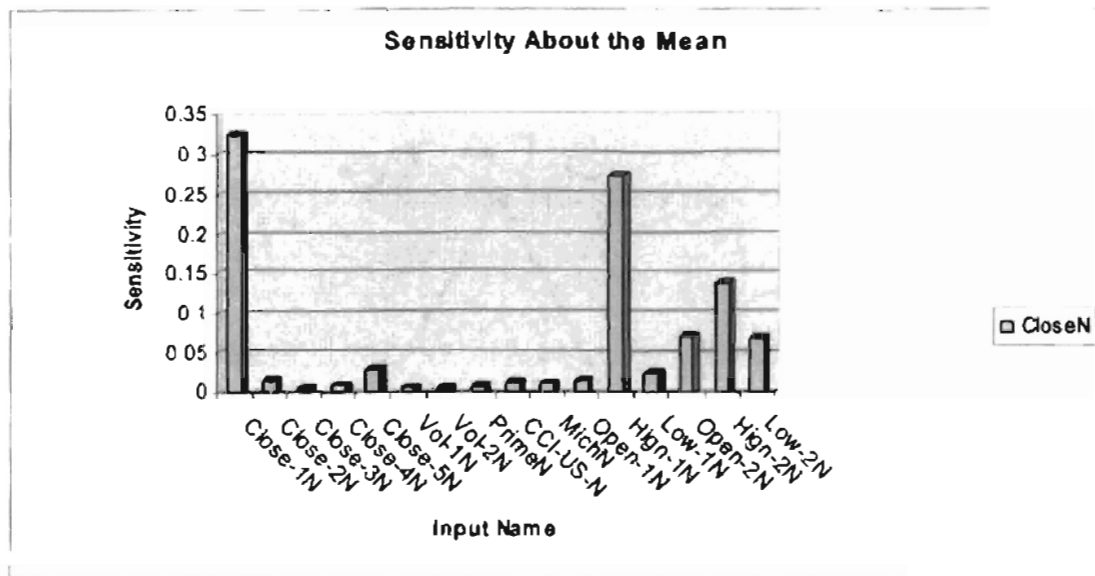


Figure 6.1 Sensitivity About the Mean: MSFT

All the results from the above experiments were then analyzed to determine which variables to use based on their importance in all the models. From the results given the best 9 inputs were picked. Using the models built, the number of inputs was reduced to a much more manageable size and from there a greedy systematic test was performed to eliminate the least important variable according to a neural network. Given the 9 inputs, a network was built and tested with one input missing. This iterative approach was impressive as improving performance, however ideally a power set should be created to pick the best inputs with every possible combination. Each network took about 20 minutes to train and test. For the original 16 inputs, the power set would contain $2^{16} = 65536$ combinations, which would take 2 years, 5 months, and 27 days, and this is not feasible for this study. The network which did the best on test was kept, thus after the

first iteration there where 8 inputs left. This process was repeated again with all 8 inputs being removed individually to find out which input was the least significant. Then that input was removed and so on until the dataset contained the 5 most significant inputs.

The graph below shows a cross table result that determined that CCI (Consumer Confidence Index) should be removed from Microsoft's inputs.

<u>Network Name</u>	<u>MSE cross validation</u>	<u>NMSE on Testing</u>
without CCI	0.000521	0.00712
without close-3	0.000535	0.00729
without volume	0.000527	0.00731
without high-2	0.000515	0.0074
without close-2	0.00053	0.00757
without open-1	0.000516	0.00765
without high-1	0.000513	0.00779
without close-1	0.0006	0.00880

Table 6.1.2 Greedy Input Reduction: 8 inputs

Interestingly, the network with CCI had a performance of 0.007506063, which was worse than the network that had the input removed. Similar the next iteration, which chooses volume to be removed, even out performed this network with a NMSE of 0.006871094.

This iteration is shown below.

<u>Network Name</u>	<u>MSE cross validation</u>	<u>NMSE on Testing</u>
without volume-1	0.000515	0.00687
without close-2	0.000518	0.00719
without high-2	0.000509	0.00726
without close-3	0.000520	0.00746
without high-1	0.000532	0.00766
without open-1	0.000524	0.00769
without close-1	0.000584	0.00809

Table 6.1.3 Greedy Input Reduction: 8 inputs

The final selection of inputs using this approach was: close-1, open-1, high-1, high-2, and close-3. The final performance on testing was 0.00693, which is just slightly worse than including 6 inputs. All networks in the above examples were trained 3 separate times

with randomly initialized weights chosen, and the best network then chosen by the system to perform the test on. Each run used conjugate gradient descent and 10,000 epochs and a threshold of 500. A threshold of 500 in this study means that if the error on the cross validation set does not improve for 500 epochs then terminate training. The network used contained 20 processing elements in layer one and 7 processing elements in the second layer. NMSE used by Neuro Solutions is the MSE of the network divided by a straight forward network that picked the average value each time and then calculated the MSE for this dumb network. Thus when NMSE is 1, the network has learned nothing about the data set and a value of 0 means the network is perfect.

$$NMSE_{network} = \frac{MSE_{network}}{MSE_{dumbNetwork}} \quad \text{Equation 6.1}$$

6.2 Data Reduction

There are many things to do for data clean up. First, the decision needs to be handled on what to do with missing or invalid data. If there is enough data, as in this model, it is wise just to throw away those records. Both data sets were examined for missing or incomplete data and none was found. Also we must decide what to do with outliers, which can throw a model off. In some cases, such as fraud detection, the outliers are the meat of the problem; however, that is not the case for our data and thus all outliers were removed from the set. All days in which the price increased or decreased by more than 10% in a single day were removed from the database as these outliers were most likely caused by external forces. Also days in which little, less than 0.1%, or no change occurred were removed because an action performed by the investor would not affect

there bottom line. The outliers for these models were determined to be days when trade volume is 4 times the average trade volume or more. For Microsoft, this value came out to any day that more than 105 million shares were traded. These days are mostly like caused by a natural or manmade disaster for which no network would have the ability to foresee. The prime was also used and the change in prime can greatly affect the market so any day that prime was changed was removed from the data. The daily value since 1947 of the prime rate is published at on the web [25] and it is updated every time the prime is changed. Removing this noisy data aids the networks to obtain better prediction. Data reduction is also necessary to give a more manageable size to the data set. Man made disasters such as 9/11 have a huge impact on the stock market, which could not have been for seen by any algorithm. Thus the entire week following September 11, 2001 was removed from the data sets of both Intel and Microsoft. The data sets used in this paper originally had data from January 1990 all the way to August 2003, and the price of the stocks had changed so much that very little would be learned by using the entire dataset. All the data prior to 1997 was removed from the data set. This provided 2 services: First it reduced the size of the data set and second it gave a better split of increasing days to decreasing days. Studies have shown in classification problems it is important to have equal representations of both cases in order to prevent the network from becoming biased towards the more common value, in this case increasing days. Intel's data set contained less than a 1% difference in the representation of increasing days as compared with decreasing days, thus further manipulation of the data set was not required. However Microsoft did much better during this period and had an increase in 59.6% of the days in the sample set. In order to prevent the network from heavily

favoring the increased prediction, the data set had increasing predictions randomly removed until a 55/45 split was achieved. This allowed experiments to be tested to show the difference with strong bias and without a bias. Thus the last year of data from August 1st, 2002 until July 31st, 2003 was held out of set to be used as an unseen testing set in the simulation.

6.3 Data Transformation

Transforming the data in the most meaningful manor for the network is essential for the success of the network. The daily price was modified from a stock price to a percentage change from the previous day in an attempt to help the network better understand the network. This gets the data closer to the actually attempted prediction of predicting if the stock price goes up or down not simply trying to guess the price, which we don't care about in this prediction paper. However, after days of testing this was found to actually hurt performance and a straight price model was then used. All data should also be standardized or normalized so that one column of values cannot completely dominate the prediction. There are many types of normalization of data such as min-max normalization, z-score normalization, and normalization by decimal scaling. Most of the better software in the industry handles the normalization for the user. The data in this paper was normalized using a min-max normalization specified in equation 2.4. Also z-score normalization was tried, and the results tended to perform worse than with the much more straightforward min-max normalization thus min-max normalization was used. Many neural network tools, such as WebStatistica and Neuro Solutions, automatically normalize the data for the user.

Chapter 7

Results

7.1 Testing Standards

The test standards listed below were used for all tests unless otherwise noted on the results. All tests on the neural network were given 10,000 epochs to train and cross validation was used to terminate training after 200 epochs with no improvement. Also all networks were ran 3 times each with randomized starting weights to insure the best possible network by minimizing the chance of obtaining a local minima. Neural Solutions allows for varying of a single parameter; so often it was the case that with everything being held constant, the number of hidden neurons in the first layer would vary from 10 to 50, with a step size of 2, to determine which network was best at learning the data. This type of gradual improvement was key to the success seen in the final networks that were to average around 63% correct which was much better than the original networks which had a very dismal performance of around 53%. When a 2-layered neural network was designed the second layer would contain the ceiling of the log of the number of neurons in the first layer. All networks used conjugate gradient descent unless otherwise noted. The transfer function used for all networks was TanhAxon. The data split for the networks were 65% test, 15% cross, and 20% testing, except in the simulation when the test set was the entire year. The training set was broken up as 80% training and 20% cross validation. Networks given for Fuzzy Cope used min-max normalization.

7.2 Conjugate Gradient vs. Back Propagation

The original neural network designed used the very straightforward method of back propagation. The error would filter through the network back on the learning rate and momentum of the network. Several new ideas have been published to provide faster learning and also to provide better results for the networks. This paper examines conjugate gradient descent and back propagation algorithm. Figure 7.2 shows that it takes about one-third the amount of time to train a conjugate gradient network than a back propagation network. Much of the improvement in speed was because conjugate gradient networks would normally cross the threshold (set at 200 epochs without improvement) before ever reaching the epoch limit of 10,000. Exact time required is not possible since the weights are randomly initialized, and the picking of the weights can greatly affect the convergence of the network. For example, it took just over an hour to train a series of conjugate gradient networks, with the number of processing elements

varying from 20 to 50 in a single layer and each network was built 3 times.

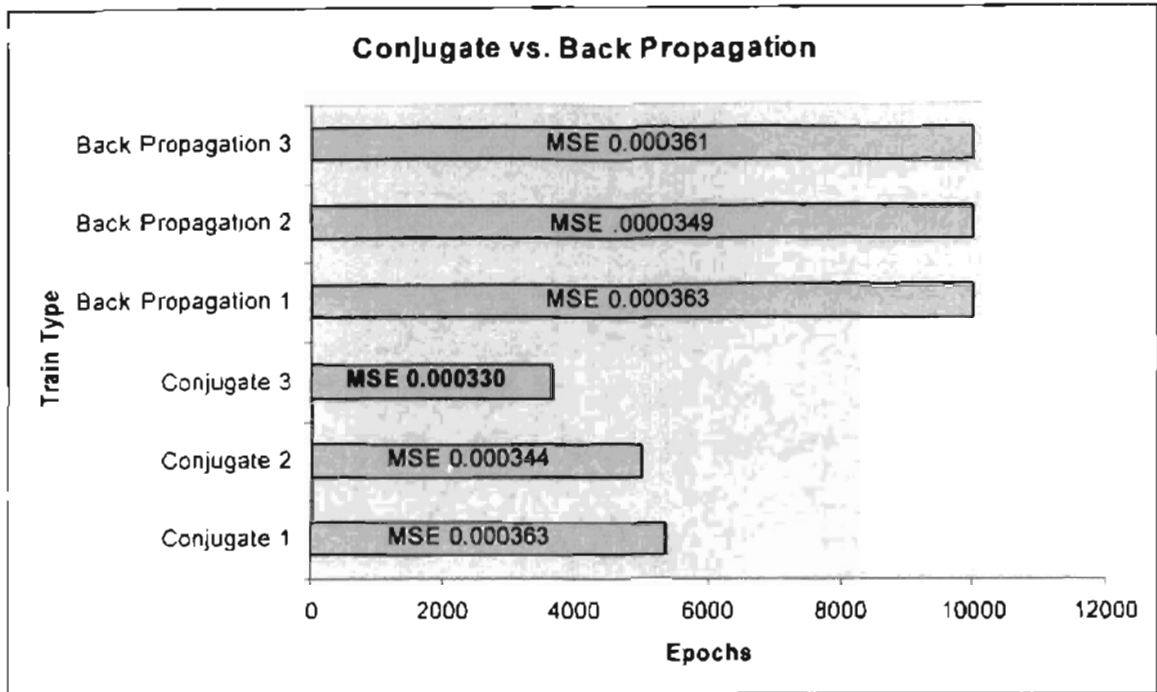


Figure 7.2 Conjugate vs. Back Propagation

The same test, with back propagation, takes about 4 hours. The chart below shows the performance of both back propagation and conjugate gradient, both using the default learning rates provided in Neuro Solutions. It is clear that Conjugate Gradient is better in almost every category in this regression test.

Back Propagation	<u>MSE Error</u>	<u>Overall Correct</u>	<u>Decrease</u>	<u>Increase</u>
30 PE* - 5 PE**	0.3725	59.98%	41.01%	78.72%
40 PE* - 6 PE**	0.3748	59.10%	41.60%	76.41%
36 PE*	0.3702	61.14%	38.92%	85.08%
Conjugate Gradient Descent				
24 PE - 5 PE	0.3702	59.98%	37.50%	82.19%
48 PE - 6 PE	0.3689	59.40%	35.16%	83.35%
22 PE	0.3745	62.59%	42.18%	82.77%

*Step size 1.0 first layer

**Step size 0.1 second layer

Random Set 1

Momentum 0.7 as default

Table 7.2 Conjugate Gradient vs. Back Propagation

7.3 Takagi Sugeno Neuro Fuzzy Inference System Tests

Neuro Solutions provide an ANFIS training kit, which they define as following: “The ANFIS (Co-Active Neuro-Fuzzy Inference System) model integrates adaptable fuzzy inputs with a modular neural network to rapidly and accurately approximate complex functions. Fuzzy inference systems are also valuable as they combine the explanatory nature of rules (membership functions) with the power of “black box” neural networks.” [22]The ANFIS package is not part of the educator version, and thus it had to be tested only in evaluation mode, limiting the number of exemplars to 300. This greatly reduced the networks ability to learn as the other networks had about 850 exemplars to be trained on. In order to make a comparison between a straight neural network and that of the ANFIS, the neural network was trained with the exact same restrictions. The results for both were mediocre at best. The significance here is simply to show that the Takagi Sugeno Neuro Fuzzy was superior to the Neural network in learning the signal and this also shows the extreme complexity of this network. The networks used 5 inputs: price yesterday, price a week ago, volume yesterday, high price yesterday, and high price 2 days ago. The last two inputs were discovered in Neural Trading solutions, a powerful stock market predictor software available for commercial use[22]. To show the extreme complexity of these networks, a rough approximation of time is given. It should be noted that many of the networks terminated, due to the cross validation set not having any improvement for 500 epochs, well before every reaching the 10,000 epochs maximum. The time complexity of using an ANFIS is its worst assets. It is simply impossible to test hundreds of different weights for a network when it takes 2 days to train a single network on one of today’s fastest machines.

<u>Type of Network</u>	<u>NMSE on testing</u>	<u>% Correct</u>	<u>Time</u>
3 MFs per input	0.003811	53.67%	30 min
4 MFs per input	0.004515	56.00%	4 hours
5 MFs per input	0.005447	55.33%	22 hours
NN-20-7	0.006474	55.00%	1 min
NN-10	0.008856	56.67%	15 sec
NN-9	0.006148	55.33%	15 sec
Randomize Records			
3 MFs per input	0.004685	55.33%	30 min

300 exemplars, Conjugate Gradient, TSK, Bell MF,
Axon Transfer, 10,000 epochs, 500 threshold, cross-validation

Table 7.3 Takagi Sugeno Neuro Fuzzy vs. Neural Networks

7.4 Mamdani Neuro Fuzzy Tests

The results reported by FuzzyCope3 were very poor. The software, simply put, lacks the power to learn such a complex network. It suffers from being unable to get out of a local minimum and thus hundreds of similar networks must be built. The software must then be reloaded to clear its memory so it does not find the same local minimum. This technique was able to find a good solution for the problem using 3 membership functions per input. However, the problem never converged when 4 and 5 membership functions were used per input despite around 100 attempts for both network using varying weights. NMSE error in these tests means, the mean of all the error squared on a normalized data set in which the data is in the range [0,1]. The results are shown below.

<u>MF per input</u>	<u>Step</u>	<u>Momentum</u>	<u>Epochs</u>	<u>RMS Training</u>	<u>NMSE Testing</u>	<u>% Correct</u>
3	0.2	0.8	1000	0.0232	0.0414	53.31%
3	0.2	0.8	2000	0.0230	0.0412	53.01%
4	0.2	0.8	2000	0.2059	0.4792	NA
5	0.2	0.8	2000	0.2059	0.4792	NA
	<u>Population</u>	<u>Min. Max</u>	<u>Generation</u>			
3 MF - GA	50	-20,20	50	0.0228	0.0412	53.01%
Cross Over Point: Uniform		Fitness Normalization Elitism		Selection: Tournament		

Table 7.4 Mamdani Neuro Fuzzy Results

The error for both 4 and 5 membership functions per input are actually 10 times worse than that of the network with 3 membership functions. This is caused by an error in FuzzyCope3, which is unable to converge for these networks.

7.5 Classification vs. Regression

This fundamental question comes down to where to translate the data before or after submitting to the neural network. When performing regression testing, all 5 inputs are given to the network, and the output is the stock price for the next day in dollars and cents. Once a test is completed the predicted price is compared with yesterday's price to see if the price increased or decreased. The actual price is compared with yesterday's price to determine if it increased or decreased. If both formulas produce the same output, then it is determined that the network correctly predicted the value. Classification is much more straightforward. Before presenting the data to the network the output is translated to 1, for increase, or 0 for decrease. And the objective of the network is to correctly predicted 0 or 1.

Regression	<u>Min MSE Cross</u>	<u>NMSE Testing</u>	<u>Testing split</u>	<u>Correct</u>	<u>Decrease</u>	<u>Increased</u>
28 PE - 5 PE	0.0004663	0.006659	50.29%	57.94%	37.50%	78.14%
42 PE - 6 PE	0.0004583	0.006789	50.29%	52.71%	31.65%	73.52%
26 PE	0.0004918	0.007338	50.29%	54.45%	36.33%	72.36%
Classification						
24 PE - 5 PE	0.3702	NA	50.29%	59.98%	37.50%	82.19%
48 PE - 6 PE	0.3689	NA	50.29%	59.40%	35.16%	83.35%
22 PE	0.3745	NA	50.29%	62.59%	42.18%	82.77%

Table 7.5 Classification vs. Regression

7.6 Random Data Sets

When dealing with random sets, it is often very important to show the data is actually able to be learned at the shown rate for more than a just a single set. As this set, which is chosen at random, could have been luckily. Showing the same tests for different randomly selected data sets shows the results are realizable. The Microsoft data set used in the previous two sections was randomized three different times and networks where built for each data set using the same standards discussed in previous sections to ensure the best possible networks were created.

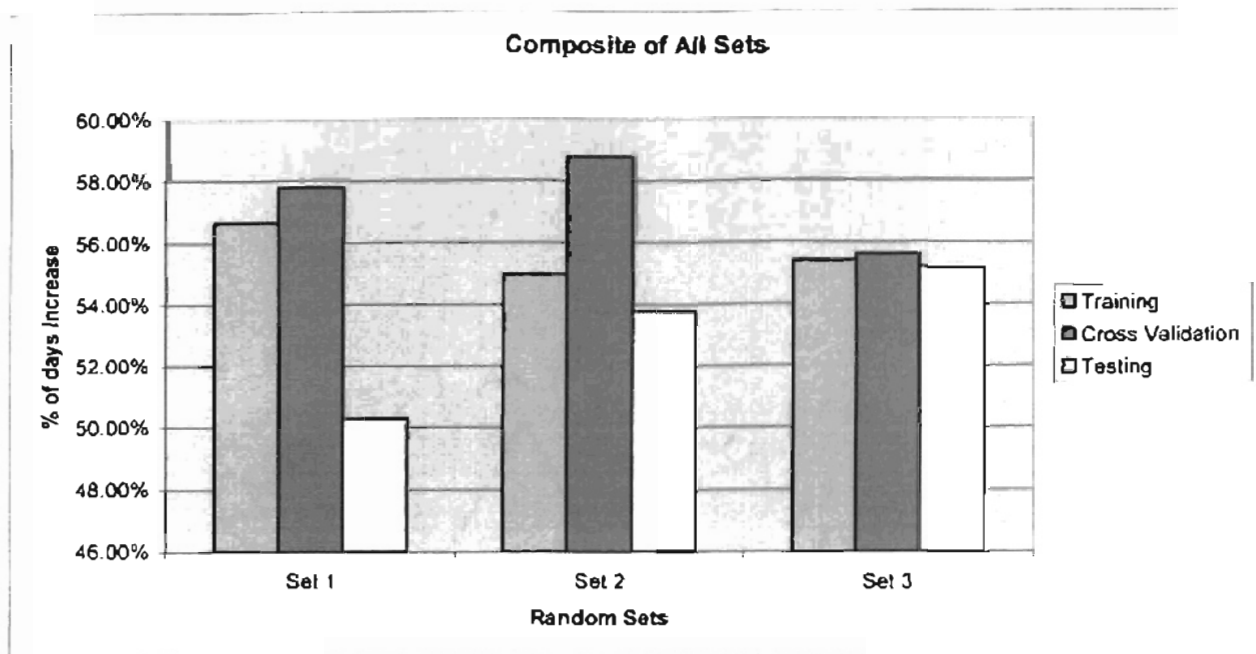


Figure 7.6 Random Data Sets

It is important that the training set and the test set have similar composites in order for the network to perform well. Set 1 has about a 6% difference in the make up training and testing of the randomly selected rows making prediction much more difficult.

Set 1	<u>MSE</u>	<u>Correct</u>	<u>Decrease</u>	<u>Increase</u>
22 PE	0.3745	62.59%	42.18%	82.77%
24 PE - 5 PE	0.3702	59.98%	37.50%	82.19%
48 PE - 6 PE	0.3689	59.40%	35.16%	83.35%

Set 2				
Classification				
34 PE	0.3920	55.91%	16.21%	90.03%
20 PE - 5 PE	0.3924	58.23%	10.40%	95.92%
38 PE - 6 PE	0.3905	59.40%	26.27%	87.86%

Set 3				
Classification				
42 PE	0.379365	60.85%	36.33%	80.49%
32 PE - 5 PE	0.382199	59.98%	36.99%	78.39%
54 PE - 6 PE	0.380529	60.85%	37.64%	79.44%

Conjugate Gradient

MSFT

Table 7.6 Classification across 3 Random Sets

The importance of the table above is to show that performance is comparable across all sets. Thus the results can be considerable reliable for any random grouping of the dataset.

7.7 Even Split in Classification

Also studies have shown that in classification it is important to have a similar split so that the network does not become biased. The data consists of MSFT stock price from 1997 randomized with 55% of the total set representing increasing days. In this study the inputs we use show the difference in performance on a set of even inputs and when the train set is not evened out. The results are below.

<u>Classification:</u>	<u>MSE Error</u>	<u>Correct</u>	<u>Decrease</u>	<u>Increase</u>
24 PE - 5 PE	0.37022	59.98%	37.50%	82.19%
48 PE - 6 PE	0.36889	59.40%	35.16%	83.35%
22 PE	0.37448	62.59%	42.18%	82.77%
<u>Classification Even Split:</u>				
22 PE - 5 PE	0.37520	58.23%	57.97%	58.49%
46 PE - 6 PE	0.37578	58.23%	61.48%	55.02%
18 PE	0.38088	61.14%	59.14%	63.12%

Table 7.7 Classification with Even Split

These finds are consistent with studies published on dataset splits. The overall performance is slightly better when not making the training a set an even split; however, this creates a big bias even when the data is split 55/45. As you can see in Table 7.6 the network that was not evenly split got over 80% for the network, for days that increased in value and less than 40% for days that decrease in value. However, the even split data was able to score around 60 correct, regardless of if the prediction day is increasing or decreasing.

Chapter 8

Simulations and Discussions

The best possible neural networks were designed using only data from January 1997 until July 2002. Data from these sets (MSFT and INTC) were removed in the manner mentioned in section 6.2. One additional step was taken after the results were shown to be good. A regression based neural network predicted all values in the training period and the results were compared with the actual price. Any day with a prediction error of more than 5% was removed from the training data set. These days were outliers and most likely caused by external forces. Each model was given \$100 at the beginning of the testing period, August 1, 2003, and the model bought if it predicted an increase with over a 50% certainty. The model would hold onto the stock until it came to a day, which had an increase certainty of less than 50%, and then it would sell the stock. The table below shows an example of this strategy. A similar model was also built which

<u>Increase Output</u>	<u>Action</u>	<u># of shares</u>	<u>Cash</u>
0.460738	STAY	0	\$ 100.00
0.617069	BUY	4.411116	\$ -
0.566166	HOLD	4.411116	\$ -
0.551663	HOLD	4.411116	\$ -
0.595844	HOLD	4.411116	\$ -
0.521784	HOLD	4.411116	\$ -
0.489340	SELL	0	\$ 106.93
0.552275	BUY	4.483247	\$ -
0.478012	SELL	0	\$ 107.69
0.651398	BUY	4.617822	\$ -
0.485218	SELL	0	\$ 113.78
0.557304	BUY	4.612206	\$ -
0.479946	SELL	0	\$ 114.29
0.461905	STAY	0	\$ 114.29

Table 8.1 Profit Simulation

earned the prime rate for any money that was not invested in the stock, this resulted in only marginal improvement for all models.

The results from the simulation were astonishing. Profit was improved as much as 889% over a straightforward buy and hold strategy. The figure below shows how profit increased dramatically by using the neural network to make decisions.

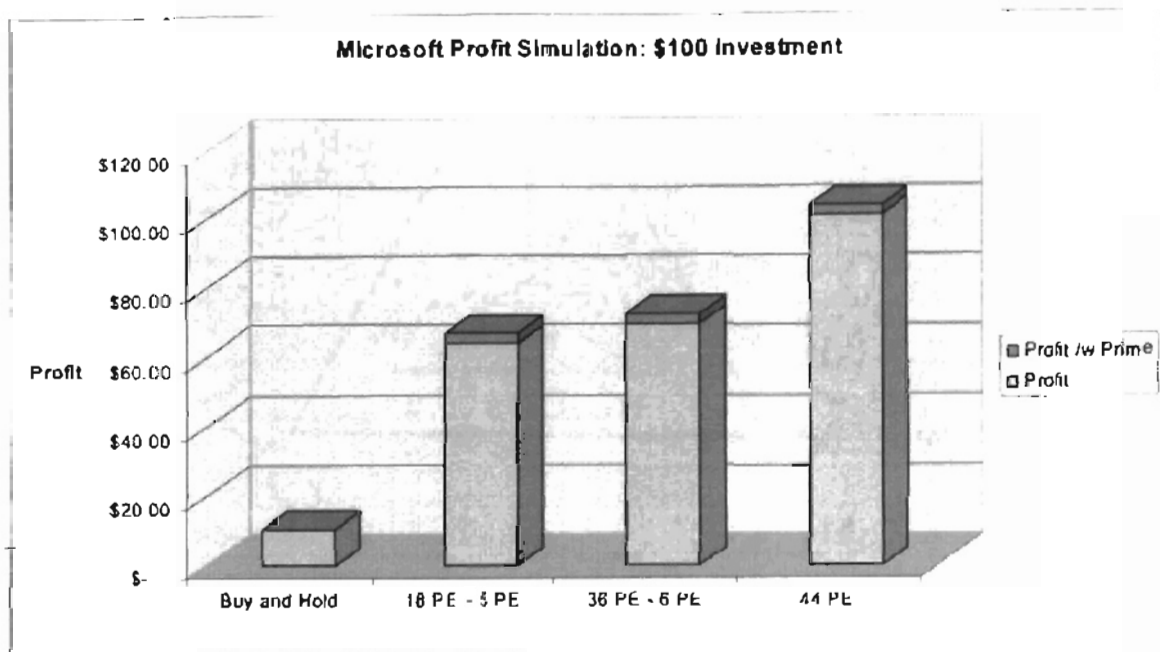


Figure 8.1 Microsoft Profit Simulation

The model, which was created with 44 processing elements in one layer, was able to make \$103.17 in a one-year period with an initial investment of \$100.00. For comparison, if the same money bought stocks at the beginning of the period and sold the stocks at the end of the period it would have made a profit of \$10.43, this is referred to as a buy and hold strategy. This model predicted the direction of the stock correctly 63% of the time. Despite the lack luster performance the model was able to predict correctly when it counts most. The biggest draw back of such a scheme in the real world is

commission. The model mentioned above bought and sold stock a total of 143 times during the 252 trading days in the simulation period. The total number of trades is shown in the table below (that shows the Microsoft model, which was much more accurate, trading more often).

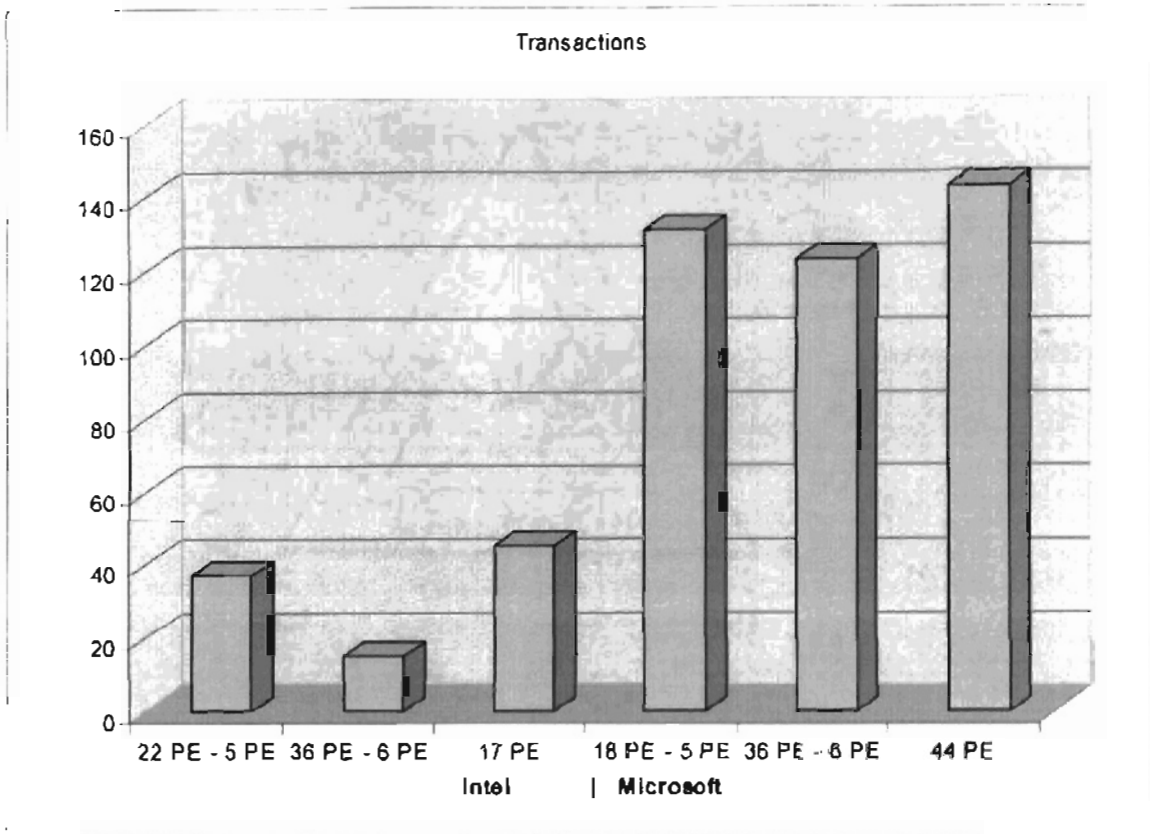


Figure 8.2 Transactions

The prediction on Intel's stock was inferior in all cases. Two of the 3 models were able to beat the buy and hold strategy. The only plus side of Intel performance is it minimized the number of transactions, thus it would incur less commission if it were actually implemented. The table below shows the best models picked by the lowest MSE on the cross validation set for both Intel and Microsoft.

<u>Network</u>	<u>MSE Training</u>	<u>MSE Cross</u>	<u>Correct</u>	<u>Profit</u>
MSFT 44 PE	0.3959	0.3968	63.32%	\$ 103.17
INTEL 17 PE	0.3957	0.3972	54.98%	\$ 49.23

Table 8.2 Profit on MSFT vs. INTC

The profit seen by the best model for Intel increased profit over the buy and hold strategy by 55%. The graph below shows the results of all 3 models on Intel's data.

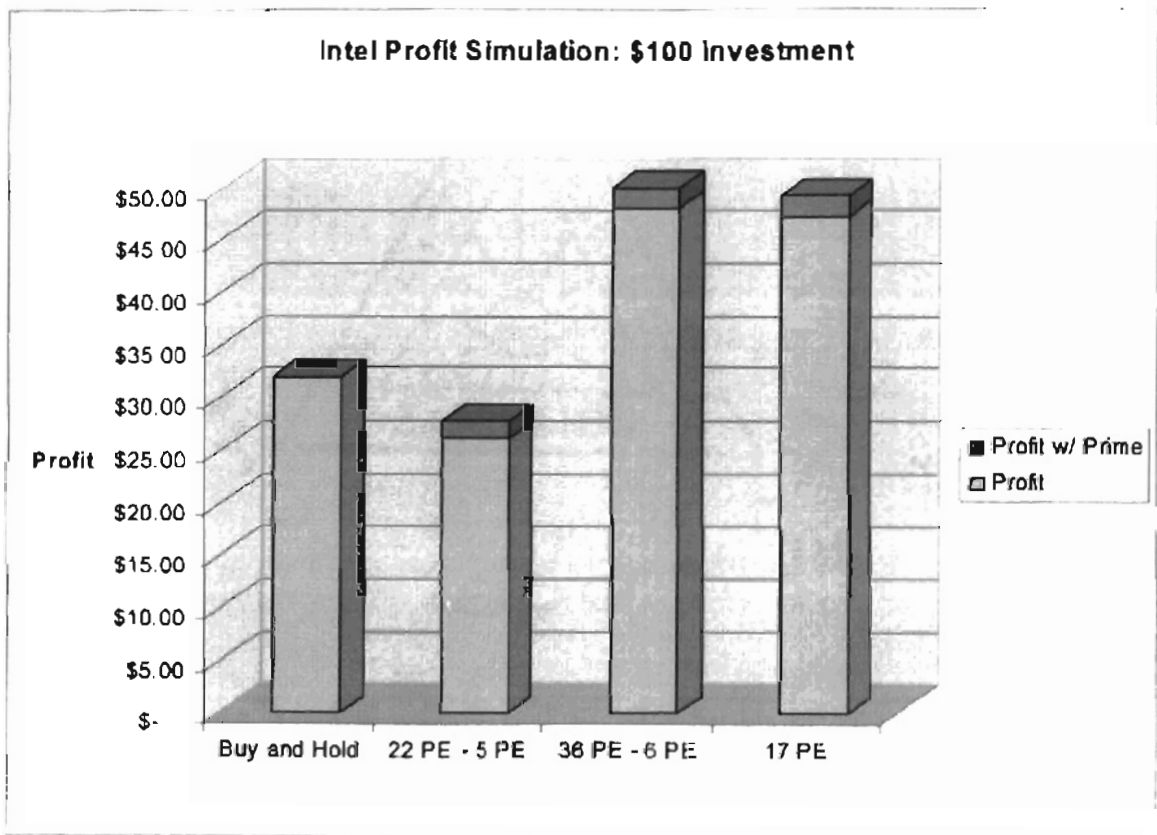


Figure 8.3 Intel Profit Simulation

Chapter 9

Conclusion

The ability to predict stocks on a daily bases is very difficult for even the most advance AI techniques. The Hurst Component confirmed the hypothesis, which said prediction is possible but especially difficult. Surprisingly, the Consumer Confidence Index and Prime Rate were not able to improve the predictability of these networks. Through the use of many techniques, it is possible to correctly predict the direction of the stock 63% of the time for a large company like Microsoft. The study demonstrated that a Takagi Sugeno (TSK) Neuro Fuzzy System was able to produce a much better result than a pure neural network when given the same training set. The TSK Neuro Fuzzy system requires a great deal of processing power, and a network with five membership functions per input took as long at 22 hours to training. The Takagi Sugeno Neuro Fuzzy model was superior in prediction to the Mamdani Neuro Fuzzy Inference System, and both networks required about the same amount of time to train. Genetic Algorithms are a very efficient way of determining which inputs are the most valuable and by removing unnecessary inputs the performance of the network can be increased. Genetic Algorithms provide a “good enough” solution when searching the entire search space could take years of CPU time on even the best system. Genetic Algorithms were used to fine turn the membership function in the Mamdani Neuro Fuzzy system, and the result was a marginal improvement in the RMSE; however it didn't improve the percentage of directional correctness, which was the goal of this study. The GA - Mamdani didn't perform as well as the ANFIS.

The neural network using conjugate gradient descent were able to achieve 63% correct on the test set, thus this research provides the groundwork for a great deal of profit. Even the worst model used for Microsoft produced a return on investment of 66%, and the best network scored an astounding 103% return. This study also demonstrated that picking the correct stock is as important as building the best network (as the best network for Intel was outperformed by the worst network on Microsoft data). The best network for Intel scored a mediocre return on investment of just under 55%. The biggest downfall of these networks was that the transaction cost of buying and selling stocks would be very costly. However, it would be feasible to fine tune the buy and sell strategy to lower this cost.

Chapter 10

Future Work

There are many areas in which this research can be extended. The many area of interest is the stock market simulation.

It is clear from all the tests that the networks were able to learn the pattern in Microsoft's data much more easily than Intel's data, thus it might be possible that another stock is more learnable than Microsoft. More stocks could be picked to determine which have the most learnable pattern. There also exist many different types of networks such as Support Vector Machines, Generalized Feed Forward, Jordan/Elman Network, ANFIS (attempted but the software version used limited its ability) and Time-Lag Recurrent Network. Any one of these might actually out perform the neural networks designed here. the only way to be for sure is to design all the networks and see which does the best. Similarly, there are many different transfer functions and trying many different combinations might also improve performance. Each problem was given well over a hundred networks before picking the best network; however, there are an infinite number of networks and thus many more networks could be built hoping to find a more optimal solution. The data sets themselves might not be optimized for learning and thus trying more than just 3 random data sets could improve performance. The input space could be searched more thoroughly. The original 6 inputs were cut down to 5 inputs using multiple techniques such as Genetic Algorithms and an iterative approach. However, all 65,536 could be checked to find the best network for each approach. The simulation was performed using a static model, and the research has been done showing that a moving

model that is updated with each day can outperform a static model. Thus it might be possible that in the simulation if a new neural network is built everyday given the most recent information it could improve its performance. The cost of commission is omitted from this research, as profit is not the goal of this study. However, if an actual real world model were to be used for making profit, it would be necessary to modify the entire model to consider commission and thus reduce the number of trade performed by the network. Many of the networks had 150 trades over the 252 days the market was open during the testing period. It would be very advisable to set a threshold that would not perform the predicted action unless the likelihood was much better than just 50%. One last method for improving the network performance would be to remove more outlier from the networks and with different combinations. All data in the training and cross validation set that was missed by more than 5% by a well trained regression network, were removed. The value of 5% should be changed many times to find an ideal value for learning. From all the above recommendations, it is clear that this problem is tractable, with so many combinations, and that not all possible networks could ever be built. So we must be satisfied with a good enough result.

Bibliography Page

1. Abraham, A. Recent Advances in Intelligent Paradigms and Applications: Studies in Fuzziness and Soft Computing. Ed. Ajith Abraham, Lakhmi C. Jain, and Janusz Kacprzyk: Physica-Verlag, (2002): 1-28.
2. Abraham, A., Philip, N. S., and Saratchandran, P. "Modeling Chaotic Behavior of Stock Indices Using Intelligent Paradigms." Neural, Parallel and Scientific Computations. 11 (2003): 143-160.
3. Abraham, A. "It is Time to Fuzzify Neural Networks." Intelligent Multimedia, Computing and Communications: Technologies and Applications of the Future. (2001): pp. 253-263.
4. Abraham, A. "Meta Learning Evolutionary Artificial Neural Networks." Neurocomputing Journal, Elsevier Science, Netherlands, (2003) (forthcoming).
5. Bartos, F.J. "Motion Control Tunes into AI Methods." Control Engineering 46 No. 5 May (1999).
6. Brownstone, D. "Using Percentage Accuracy to Measure Neural Network Predictions in Stock Market Movements." Neurocomputing. 10 (1996): 237-250.
7. Castellano, G., Castiello, C., Fanelli, A.M., and Giovannini, M. "A Neuro-Fuzzy Framework for Predicting Ash Properties in Combustion Processes." Neural, Parallel and Scientific Computations. 11 (2003): 69-82.
8. Chen, A.S., Leung, M.T., and Daouk, H. "Application of Neural Networks to an Emerging Financial Market: Forecasting and Trading the Taiwan Stock Index." Computers and Operations Research. 30 (2003): 901-923.

9. Feder, Jens. Fractals. Pelnum Press, New York, New York, 1988.
10. Fuzzy Cope 3. <http://kel.otago.ac.nz/software/FuzzyCOPE3/>, 11/2003
11. Garver, M. S. "Try new data-mining techniques" Marketing News. Volume 36, Issue 19 Sept (2002).
12. Grossglauser, M. and Bolot, J-C. "On the Relevance of Long-Range Dependence in Network Traffic." ACM SIGCOMM '96. August (1996).
13. Harris, C.J. and Hong, X. "Neuro-Fuzzy Network Model Construction Using Bezier Bernstein Polynomial Functions." IEEE Proc D Control Theory and Application in Press. 47 (2000): 337-.
14. Hurst, H.E., Black, R.P. and Simaika Y.M. Long-Term Storage: An Experimental Study. London, England. Constable, 1965.
15. Izumi, K. and Ueda, K. "Analysis of Exchange Rate Scenarios Using an Artificial Market Approach." Proceeding of the International Conference on Artificial Intelligence. 2 (1999): 360-366.
16. Jang, J.S.R. "ANFIS: Adaptive-Network-Based Fuzzy Inference System." IEEE Transactions on Systems, Man, and Cybernetics. 23 (1993): 665-684.
17. Jang, J.S.R. "Input Selection for ANFIS Learning." Proceeding of IEEE International Conference on Fuzzy Systems. Sep. 1996.
18. Jang, J.S.R., Sun, C.-T., and Mizutani, E. Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence. New Jersey: Prentice Hall, 1997.
19. Kuo, R.J., Chen, C.H., and Hwang, Y.C. "An Intelligent Stock Trading Decision Support System through Integration of Genetic Algorithm Based Fuzzy Neural

- Network and Artificial Neural Network.” Fuzzy Sets and Systems. 118 (2001): 21-24.
20. Labbi, A., Gauthier, F. “Combining Fuzzy Knowledge and Data for Neuro-Fuzzy Modeling.” Journal of Intelligent Systems. 7 N1/2 (1997).
21. Mamdani, E H and Assilian, S. “An experiment in Linguistic Synthesis with a Fuzzy Logic Controller.” International Journal of Man-Machine Studies. Vol. 7. No.1, pp. 1-13, (1975).
22. Neuro Solutions. <http://www.nd.com>, 11/2003
23. O’Brian, T. V. “Neural Nets for Direct Marketers” Marketing Research. Volume 6, Issue 1 Dec (1994).
24. Petrovic-Lazarevic, Sonja, and Abraham, A. “Hybrid Fuzzy-Linear Programming Approach for Multi Criteria Decision Making Problems.” Neural, Parallel and Scientific Computations. 11 (2003): 53-68.
25. Prime Rate. <http://research.stlouisfed.org/fred2/data/PRIME.txt>, 11/2003.
26. Quah, T.S. and Srinivasan, B. “Improving Returns on Stock Investment through Neural Network Selection.” Expert Systems with Applications. 17 (1999): 295-301.
27. Sugeno M, “Industrial Applications of Fuzzy Control.” Elsevier Science Pub Co., (1985).
28. Tsaih, R., Hsu, Y., and Lai, C.C. “Forecasting S&P 500 Stock Index Futures with a Hybrid AI System.” Decision Support Systems. 23 (1998) 161-174.
29. Turban, E. and Aronson, J. E. Decision Support Systems and Intelligent Systems. Delhi, India: Pearson Education, Inc., 2001.

30. Weka. <http://www.cs.waikato.ac.nz/ml/weka>, 11/2003.
31. Yao, J.T. and Tan, C.L. "A Study on Training Criteria for Financial Time Series Forecasting." Proceedings of International Conference on Neural Information Processing. Nov. 2001: 772-777.
32. Yao, J. and Poh, H.L. "Forecasting the KLSE Index Using Neural Networks." IEEE International Conference on Neural Networks. 2 (1995) 1012-1017.
33. Yao, J., Tan, C.L., and Poh, H.L. "Neural Networks for Technical Analysis: A Study of KLCI." International Journal of Theoretical and Applied Finance. 2 (1999) 221-241.
34. Yao, J., Poh, H. L. "Equity Forecasting: A Case Study on the KLSE Index." NNCM '95. (3rd International Conference on Neural Networks in the Capital Markets). Oct (1995) 341-353.
35. Zadeh, L. A. "Fuzzy Sets." Information and Control. June (1965), 8(3): 338-353.



VITA

Brent Arthur Doeksen

Candidate for the Degree of

Master of Computer Science

Thesis: Predicting Financial Markets Using Neuro Fuzzy Genetic Systems

Major Field: Computer Science

Biographical:

Personal Data: Born in Stillwater, Oklahoma, On November 14, 1976, the son of Gerald and Cheryl Doeksen

Education: Graduated from Stillwater High School, Stillwater, Oklahoma in May 1995; received Bachelor of Science degree in Computer Science and Mathematics from Oklahoma State University, Stillwater, Oklahoma in December 1999.

Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in December, 2003.

Experience: Brent has been a professional software developer since 1999 and has worked for several companies across the United States: Sabre Inc., J.B. Hunt, Phillip Morris, OneOK, and Williams Communications Group.