A COMPARISON OF NEURAL NETWORKS
FOR STOCK SELECTION

By

YIMIN YANG

Bachelor of Arts
Hebei Teacher's University
Shijiazhuang, Hebei
People's Republic of China
1990

Bachelor of Science
Renmin University of China
Beijing, P.R.China
1992

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
July, 1999

A COMPARISON OF NEURAL NETWORKS

FOR STOCK SELECTION

Thesis Approved:

J Chandler

Thesis Advisor

[illegible]

[illegible]

Wayne B. Powell

Dean of the Graduate College

ACKNOWLEDGMENTS

I would like to extend my sincere appreciation to my committee members for their guidance and support. I am particularly grateful to my advisor, Dr. J. P. Chandler, for his time and efforts invested in my whole graduate program and his sincere friendship with me. My thanks also go to Dr. B. E. Mayfield and Dr. H. K. Dai for their great help in my education and their time of serving as members of the committee.

My special thanks and love go to my wife, Yingjie Dong, for her great love and encouragement in the whole journey of my graduate study and the past twelve years. Every progress I made has had her contribution. My special thanks are also extended to my parents, Shiguo Yang and Jianying Ma, especially to my mom who is very intelligent but was robbed by the Communist party government of all the fortune she should inherit, and all working opportunities. Even in such a hard environment, she continuously encouraged and supported us brothers in attending the best universities in China and studying abroad. I completely understand her deep love for us.

My thanks also go to my brother, Fengming Yang, for his help in solving problems in neural networks and debugging of the programs.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Statement of the Problems

Individual and institutional investors apply various selection and optimization strategies in the process of creating a securities portfolio, which they hope will provide high return on their capital investment. Such strategies are frequently based on research in a particular industry, historical price data for individual stocks, analysis of fundamental data, etc. A common problem in forming such a strategy is, however, the overwhelming amount of the available market information, which can't be readily interpreted. For this reason, investors are confined to specific techniques, which rely on limited subsets of data. In addition, stock selection is a complex process. You may list more than a hundred factors that can affect the price movements of a particular stock. There are thousands of stocks listed in a market. For example, there are more than 2000 stocks listed in New York Stock Exchange (NNSE). Selecting several stocks from all stocks traded in the markets requires dealing with millions of records of various data. It is not the work that can be done in a short time by human brains. But decisions need to be made in a short time to catch up with the rapidly changing markets, so computers are used to deal with these large amounts of numerical information. Traditionally, computers have been used to do such work as computation, sorting and comparison, according to the user's order. But that is not enough for stock selection. Sophisticated techniques are required to analyze and combine disparate information that can potentially impact security price.

In the past few years, neural networks have become popular in solving problems in many fields, from interest rate prediction to speech recognition. The main attractive character of the neural networks is that they can work like human brains. They can learn from experience and make predictions according to what they have learned. As a human brain can predict it's going to rain or not according to the clouds in the sky and other factors, a neural network can predict the price movements of a particular stock according to its past trading information and other information. Unlike traditional artificial intelligence systems (also called expert systems) which are rule-based and don't need training, a neural network can learn from examples and modify itself according to changing conditions. Training is very important to neural networks.

1.2 Objectives

A comparison will be made of selected neural networks for stock selection.

Multi-Layer Perceptron (MLP) neural networks and Radial Basis Function (RBF) neural networks will be used in the stock selection systems.

Each system will be trained to select stocks from all stocks traded in a market; the performances of the selected stocks will be expected to be among the top ones in the near future (two weeks); tests will be made and results will be compared.

1.3 What Is New About the Systems.

1. The RBF systems in this thesis are the first stock-selection systems using RBF neural networks the author has seen although much work in this area is unpublished.
2. One of the more advanced learning algorithm (Conjugate Gradient) rather than the backpropagation algorithm will be used in the MLP neural networks.
3. The systems are mainly for speculators.
4. Some financial indicators (see following sections) developed by the author will be used in the software to ensure that this software is unique.

CHAPTER 2

REVIEW OF THE LITERATURE

2.1 Neural Network and Artificial Intelligence

The modern era of neural networks is said to have begun with the pioneering work of McCulloch and Pitts (1943); their classic paper [68] described a logical calculus of neural networks. This paper was widely read at the time and still is.

In 1948, Wiener's famous book "Cybernetics" was published [69], describing some important concepts for control, communications and statistical signal processing. In 1949, Hebb gave the clear statement about the concept of learning rule in his book "The Organization of Behavior"[70]. This was the first time the concept of a learning rule was formally presented which encouraged the development of computational models of learning and adaptive systems.

In 1952, Ashby's book, "Design for a Brain: The Origin of Adaptive Behavior"[71], was published, which is just as fascinating to read today as it must have been then. The book was concerned with the basic notion that adaptive behavior is not inborn but rather learned.

In 1954, Minsky wrote a "neural network" doctorate thesis at Princeton University [72], which was entitled "Theory of Neural-Analog Reinforcement Systems and Its

Application to the Brain-Model Problem". In 1961, an excellent paper by Minsky on AI, entitled "Steps Toward Artificial Intelligence", was published [72]; this later paper provide lots of terms used today in the neural network field.

Also in 1954, Gabor, one of the early pioneers of communication theory, and the inventor of holography, proposed the idea of a nonlinear adaptive filter [35].
Another topic that was investigated in the 1950s is "associative memory", which was initiated by Taylor (1956) [33].

An issue of particular concern in the context of neural networks is that of designing a reliable network with neurons that may be viewed as unreliable components. Von Neumann (1956) [26] using the idea of redundancy, which motivated Winograd and Cowan (1963) [17] to suggest the use of distributed redundant representations for neural networks, solved this important problem. Also in 1962, Frank Rosenblatt "Set forth the principles, motivation, and accomplishment of perceptron theory in their entirety" [36].

An important problem encountered in the design of a multilayer perceptron is the credit assignment problem (i.e. the problem of assigning credits to hidden neurons in the network). The terminology "credit assignment" was first used by Minsky (1961) [66], under the title "Credit Assignment Problem for Reinforcement Learning Systems".

An important event that did take place in the 1970s was self-organizing maps using competitive learning. The computer simulation work done by von der Malsburg (1973) [22] was perhaps the first to demonstrate self-organization.

In the 1980s, major contributions to the theory and design of neural networks were made on several fronts, and with it there was a resurgence of interest in neural networks.

Grossburg (1980) [69], building on his early work on competitive learning, established a new principal of self-organization that combines bottom-up adaptive filtering and contrast enhancement in short term memory.

In 1982, Hopfield [73] used the idea of an energy function to formulate a new way of understanding the computation performed by recurrent networks. Another important development in 1982 was the publication of Kohonen's paper [74] on self-organizing maps using a one or two dimensional lattice structure.

In 1984, Braitenberg's book, "Vehicles: Experiments in Synthetic Psychology", was published [76].

In 1986, the development of the back-propagation algorithm was reported by Rumelhart, Hinton, and Williams [77].

In 1988, Linsker described a new principle for self-organizing in a perceptual network (Linsker, 1988) [63]. The principle is designed to keep maximum information about input patterns. Also in 1988, Broomhead and Lowe [78] described a procedure for the design of layered feedforward networks using radial basis functions, which provide an alternative to multilayer perceptrons. We'll use two radial basis function systems in this project.

In 1989 Mead's book "Analog VLSI and Neural Systems" [79] was published, which provides an unusual mix of concepts drawn from neurobiology and VLSI technology.

Perhaps the 1982 paper by Hopfield and the 1986 two volume book by Rumelhart and McCLelland [80] were the most important publications which brought about the recovery of interest in neural networks in the 1980s.

In above, we give an outline of the development of the neural networks. Because the most exciting part of the neural network is its ability of learning, we will focus our discussion on learning algorithms in the following.

Each learning algorithm is related to some specific kind of neural networks. There are four different classes of neural network architectures [58]:

1. Single-Layer Feedforward Networks

   A single-layer feedforward network has in fact two layers of neurons; the input layer and the output layer. But the computation happens only in the output layer and it is strictly feedforward.

2. Multi-layer Feedforward Networks

   The second class of feedfoward networks is different from the first class is that a multi-layer feedforward network has more than one hidden (computation) layers

3. Recurrent Networks

   A recurrent network distinguishes itself from a feedforward network in that it has at least one feedback loop.

4. Lattice Structure

A lattice consists of a one-dimensional, or high-dimensional array of neurons. A related input array is also needed. A lattice network is really a feedforward network with output neurons arranged in rows and columns.

Of the four classes of networks as we described above, multi-layer feedforward networks are the most widely used ones. There are two different groups of networks among multi-layer feedforward networks. They are multi-layer perceptron neural networks (MLP) and radial basis function neural networks (RBF).

A traditional MLP network uses backpropogation learning algorithms. An MLP with such learning algorithms is successfully used in many fields [58]. But it has also some drawbacks of which, needing lots of time on training is the major one. Some advanced algorithms such as quasi-newton method, conjugate gradient method are used to MLPs to overcome these drawbacks.

The first conjugate gradient method was proposed by Hestenes [106] in 1980. Several other conjugate gradient methods such as the Fletcher-Reeves method, the Polak-Ribiere method appeared since then. According to Meishan Cheng [107], these conjugate gradient methods are relatively equivalent. Some experiments report that MLPs with conjugate gradient method have great improvement over MLPs with backpropogation method [43].

Broomhead and Lowe were the first to use radial basis functions in the design of neural networks [108]. Other contributions in this field were made by Moddy and Darken [109], and Poggio and Girosi [110]. The range of application of RBFs is quite broad though RNF networks are not as popular as MLP networks.

2.2. An Introduction to Speculators in the Stock Market.

Webster gives a number of definitions on speculation. Among them we find: [32]

(1) "mental view of anything in its various aspects; intellectual examination";

(2) "the act or practice of buying land or goods, etc., in expectation of the rise of price and of selling them at an advance".

To the second he added the complacent observation that " a few men have been enriched but many have been ruined by speculation" [32]. According to Webster, the motive is the test by which we must distinguish between an investment and a speculative transaction.

According to Philip L. Carret ("The Art of Speculation", 1930), speculation may be defined as " The purchase or sale of securities or commodities in expectation of profiting by fluctuations in their prices".

There are many fields in which one can speculate: commodity markets, stock markets, bond markets, currency markets, and gold markets. There are many methods by which one can speculate: futures, options, sellinf short, buying long in actual objects. This software only considers the stock market. It specializes in finding stocks that will perform among the top ones in the near future.

2.3. Neural Network Applications in Finance and Investment

In a global economy, the finance and investment industries need to deal with large amounts of data and they are facing more and more hard competition from many corporations and many countries. They will develop and use every new technique to help them keep leading positions in the market. Naturally computer technologies were the most widely used technologies. The neural network is one of the most attractive techniques.

Many of the first neural network researchers were excited by the idea of learning that the neural network can achieve through similarity to a small part of a human brain. Although most early attempts to apply neural networks to financial decision making were rudimentary, clumsy, and generally unsuccessful, recent innovation in the technology and improvements in our understanding of the strengths and weaknesses of neural networks are now resulting in commercially successful systems.

Neural networks are especially suited for simulating intelligence in pattern recognition, association, and classification activities. These problems frequently arise in such areas as credit assessment, security investment, and financial forecasting [89]. It is worth noting that, after the Department of Defense in 1989 investing on a five-year, multimillion-dollar program for neural network research, financial institutions have been the second largest sponsor of research in neural network application.

Lapedes and Robert performed one of the first applications of neural networks in forecasting [90]. They designed networks for forecasting chaotic time series. Early applications of neural network forecasting to the stock market were reported to be unsuccessful [91].

Weigend et al. [92] made a foreign exchange rate prediction using a neural network with two output neurons, one for return and one for the sign of the return.

Chakraborty et al [93] applied neural networks to forecasting monthly flour price indices in three commodity exchanges.

Bergerson and Wunsch [94] developed a hybrid neural network and expert system for predicting the S&P 500 stock index. The targets were selected as buy/sell/no-action signals by a human expert in a labor-intensive exercise, in contrast to predicting every rise and fall of the index.

Zaremba [95] described three neural network forecasting models for the S&P 500, US Treasury Bonds, and gold futures. He reported moderate success in actual trading. A number of articles have appeared in *Technical Analysis of Stocks and Commodities* on forecasting with neural networks. Shih [96] described a neural network trained on long and short indicators. Fishman et al. [97] trained a neural network on the S&P 500 price difference to predict a week ahead. Katz [98] discussed development issues in neural forecasting, and advised the use of separate neural networks, one trained to predict top and the other bottom turning points. The journal *Futures* has also featured articles on the subject. Jurik [99] discussed aspects of preprocessing input data based on harmonic analysis.

Yoon and Swales in 1991 [100] adopted a four-layered network with nine input parameters. They reported that, on the average, 77.5 percent of test cases were correctly classified. This result outperformed the classifying power of multiple linear discriminant analysis, which correctly classified only 65 percent [88].

Kamijo and Tanigawa (1990) [102] developed a recurrent neural network model to predict a price pattern, called the "triangle" pattern, from a candlestick chart. After the network was trained on 15 patterns by iterating 2,000 times, it correctly classified 15 out of 16 test cases, but this model may use so few patterns such that the model may memorize all patterns.

Kimoto and Yoda in 1990 [101] developed a network to determine optimal buy-and-sell timing for the TOPIX (Tokyo Stock Exchange Price Index). The terminal investment value of 3,129 in September 1989, which would have resulted from following the neural network model recommendations, exceeded the 2,642 TOPIX value at that date by 487 points—a significant difference.

In a study by Jiang and Lai in 1994 [103] using Taiwan stock data, the annual rates of return generated by a neural network-driven buy-and-sell strategy were considerably higher than those of a competing buy-and-hold strategy.

Some investment banks and journals provide lists of stocks, which show a tendency of irregular movements. A big investment bank can employ many analysts to follow all stocks, but the personal investor cannot do that, so software is very helpful in determining which stock to buy.

Some work has been done on neural network software by Neurostock Company (http://www.neurostock.com) which provides a variety of products of neural network software.

Several applications of neural nets to the domain of finance are already known in the art.

U.S. Pat. No.5,109,475 to Kosaka et al covers a system that uses a Hopfield neural network for selection of time series data. It only considers historical risk and return data, and assumes that the relation between risk and return factors will remain about the same in the future, but in a rapidly changing market environment such an assumption may not be accurate.

U.K. Pat. Application 2 253 081 A to Hatano et al. covers a system that describes a neural network stock-selection scheme using only price data as input.

U.S. Pat. No. 5,761,442 to Barr, et al. is called "Predictive neural network; means and method for selecting a portfolio of securities wherein each network has been trained using data relating to a corresponding security". This patent was granted in June 1998, so it can be considered as the most recent job done in this field. It is a good system, trying to cover as many as factors as possible in predicting the price of a particular security. As we said above, a lot of research on quantitative investing remains unpublished and secret, however.

Differences between Barr's System and This RBF System:

1. Barr's system provides methods for selecting securities and constructing an investment portfolio, which is expected to provide a return that is superior to the broad index benchmarks of a given capital market. Its goal is just to beat the index. The goal of this system is to provide methods for selecting securities to maximize the return.

2. Barr's system can be used by individual investors and institutions whereas the systems in this thesis can be better used by speculators.

3. Barr's system uses both fundamental and technical factors as input, while this system uses only technical factors as input.

4. This system uses different technical factors from that of Barr's system.

5. Different neural networks (RBF) will be used in this system from that of Barr's system.

6. Only stocks having the best appreciation potentials will be considered by this system, whereas Barr's system also considers those stocks having the worst appreciation potentials.

7. This system will use the same neural network for each stock, whereas Barr's system uses different neural networks for different stocks.

CHAPTER 3

MAIN IDEAS, DESIGN, AND PROBLEM SOLVING

3.1 Main Ideas of the Software

The purpose of this software is to help someone win in the stock market. How to achieve this goal is critical to the design and implementation of this program. If a stock shows a tendency to be upgoing, it may be (by computing examples and borrowing research results of other articles) that its price will increase at least, say 35% in the near future (say in one month). If one wants to win as much as this increase, this software should discover as early as possible these stocks and recommend them to the user. But how to find them is difficult. Fortunately, such stocks may show some common signs in the early stages of their upgoing [16]. These common signs are that the stock's price and trading volume will be strong and go beyond the general index [12]. So I design this software focusing on finding and listing those stocks whose price and trading volume has gone strongly beyond the general index in a fixed period of time, say, five days or ten days.

There are two primary approaches of stock market analysis: fundamental and technical; these systems belong to the technical approach. These systems do not care whether it is a bull or bear market; they are for stock selection, not for whole-market analysis. Some technical analysis is based on the idea that the future price of a stock is predictable; others on the idea that it is not predictable. This system is based on predictability.

Some technical analysts think there exist trends in the stock market; others think the price of the stock fluctuates like a random walk. This software is based on the idea that there exist trends in the stock market. There are mainly three kinds of technicians; they are tide-watchers, manipulators, and purists. This software is better for tide watchers. There are many aspects of stock market analysis. There are thousands of books written on it; there are hundreds of indicators created for it. As said above, this software focuses on stock selection, and behaves like a tide watcher, thus, its domain is narrowed greatly. In this narrowed field, like other software, this software also builds its forecasts on price and volume analysis of a stock. There are two famous indicators existing in this field for many years; they are "Relative Strength" and "On-Balance Volume". I will, in the following, briefly describe the ideas of these two indicators and show the differences between them and my indicators used in the software.

3.1.1 On-Balance Volume (OBV)

One technique for measuring accumulation and distribution, first developed by Joseph Granville (Granville's *New Key to Stock Market Profits*) [6], the On-Balance Volume system is based on the assumption that volume trends lead price trends. The cumulative on-balance volume series is calculated by adding successive daily OBV figures. The total is increased when the stock gains in price and is decreased when the stock falls. If cumulative on-balance volume is positive, meaning the price will go up, the stock is under accumulation, and if cumulative OBV is negative, the stock is being distributed (meaning the price will go down).

3.1.2 Relative Strength Indicator (RSI)

RSI was first developed by Robert A. Levy, a successful fund manager and computer researcher. Levy summarized his most important research conclusion in the November, 1967 issue of the *Financial Analysts Journal* (*"Random Walks: Reality or Myth")*. His study was based on an analysis of weekly closing price of 200 NYSE common stocks for the 260-week period beginning October 25,1960 and ending October 15, 1965. Each week, the 200 stocks were ranked on the basis of their relative price performance over the preceding 26 weeks (especially, the extent to which the current price was above or below the average price of the last 26 weeks). The experiments with various portfolio strategies are based on different strength ranks. The validity of RSI has been confirmed in numerous other studies but there is reason to believe that the finest work in this field remains unpublished.

This software will use moving averages and general indices. The stock exchanges or other security corporations such as Dow Industries, Standard & Poors 500, and the Shanghai Index provide the general indexes. As for moving averages, this will use price moving averages and volume moving averages, for 5-day or 7-day or other periods. There are two versions of moving averages popularly used in practice:

3.1.3 Simple Moving Average

A simple ten-day moving average of the Dow Jones Industries consists of successive average of its ten most recent days' closing values. Just add up the latest ten values and divide the total by ten.

This simple moving average is subject to criticism on two counts. First, it assigns equal weight to each of the base observations. Second, as a simple average moves through time, its point-to-point fluctuations are strictly dependent upon only two numbers, the one being dropped and the one being added.

3.1.4 Weighted Moving Average

This is based upon the assignment of greater weight (hence greater important) to more recent observations and lesser weight to old values. Each method has its adherents, but the weighted moving average is probably the best.

There are no magic numbers in trend following. Some technicians assert that a ten-day moving average is not optimal, that a 5-day or 20-day or perhaps some thing such as a 13-day moving average is superior on a long-term basis. Some analysts might insist that a 10-week, 30-week or 40-week moving average is best, while others suggest that perhaps some odd moving average length say, 7 or 39 weeks is optimal. As for my software, it is better to use short period moving averages because the requirement of my software is to find as early as possible the targeted stocks. If the price of the stock has already increased greatly when my software recommends the stock, then my software is a failure. So I will test 5-day moving averages.

The above indicators are used in this software as support indicators. I will create new indicators with the help of them. This software will produce results according to the new indicators I created.

3.2 Design of the System

According to Valluru [85], there are 12 steps in building a forecasting model, as listed below.

1. Decide on what your target is and develop a neural network for each target.
2. Determine the time frame that you wish to forecast.
3. Gather information about the problem domain.
4. Gather the needed data and get a feel for each input relation to the target.
5. Process the data to highlight features for the network to discern.
6. Transform the data as appropriate.
7. Scale and bias the data for the network, as needed.
8. Reduce the dimensionality of the input data as much as possible.
9. Design network architecture (topology, number of layers, size of layers, parameters, and learning paradigm).
10. Go through the train/test/redesign loop for a network.
11. Eliminate and correct inputs as much as possible, while in step 10.
12. Deploy your network on new data and test it and refine it as necessary.

3.2.1 Target of the Neural Networks

The targets of the neural networks will be trained to predict the price of a stock two weeks into the future according to the movements of the stock in the past two weeks.

3.2.2 Choosing of Data

There are some choices for choosing data for the training and testing of the neural networks. One method is to choose few stocks with longer period. For example, we may use three year's data of one stock to train the network. The other method is to choose many stocks with shorter period. For example, we may use two week's data of 100 stocks to train the networks. The first method is better for predicting a particular stock. This method was used in Barr's system [83]. The second method is better for stock selection, because if the network learns from only one stock, its generalization will be bad. So the second method is used to choose data for training and testing of the networks in this project. An ideal method would be to use many stocks with a very long period (say 20 years); this would be beyond the scope of this project.

The data we will use to train the neural networks are chosen arbitrarily from New York Stock Exchange (NYSE). It covers from 1996 to 1998. Such choice of data also has some drawbacks.

1. The period of time covered may not be typical. The longer the period is, the better the neural networks are trained. A good experiment may need as long as 20 years or more of data coverage. Due to insufficient time and equipment (we can't access a database), we just use the 17 days data for 100 stocks to train our neural networks. This may affect the generalization of the networks.

For future work, we recommend that fully selected data should be used to train the neural networks.

2. In our systems, the past two-week's information is used to predict two weeks into the future. A further study may try use more than two week of past data to predict two weeks into the future, say using four weeks of data to predict two weeks into the future.

3. For the systems to be universally used, the data used to train the neural networks may be chosen from several major markets rather than from only one market.

The data we chose directly from stock is called raw data. It is saved in a file named "rawdata" which will be processed by the preprocessor to make a training file for the neural network. Another group of data chosen in the same way is saved in a file called "testdata" which will also be processed by the preprocessor to make a test file for the neural network simulators. The form of each line in file "rawdata" and "testdata" will be listed below.

Line 1+17*x, x=1,2,...N has two fields.

Field 1. The stock name

Field 2. The highest price in two weeks from the last day of chosen data. It is used as the expected output.

Each of the 17 lines following line 1+17*x has five fields.

Field 1. Trading date.

Field 2. Trading volume of the whole market.

Field 3. Closing value of the market index (Dow Industries Average in our case)

Field 4. Trading volume of the particular stock

Field 5. Closing price of the particular stock.

3.2.3 Decision of Inputs to the Neural Network

One can list more than 100 factors that can affect the price movements of a stock. so a decision must be made as to which factors should be used as the input to the network. The decision should be based on the target of the network. The targets of these networks are to predict two weeks into the future. That is a short period, so we only need to consider those factors that can affect the stock's price in the near future. Based on this principle, the following eight factors were chosen:

PAN	PDP	PRN	PRG?

number of days in which the price is greater than the 5-day price moving average(n1)
0<=n1<=10
if(n1=10)
return 10/10
if(n1=4) 4/10
if(n1=3) 3/10
if(n1=2) 2/10
	Number of days in which the price increase percentage is greater than that of Dow Jones index (n2) 0<=n2<=10
	if(n2=5) return 5/10
	if(n2=4) 4/10
	if(n2=3) 3/10
	if(n2=2) 2/10
	if(n2=1||n2=0) 1/10
		Number of days in which the price of current day is greater than the last day(n3)
		0<=n3<=5
		if(n3=5) return 5/10
		if(n3=4) 4/10
		if(n3=3) 3/10
		if(n3=2) 2/10
		if(n3=1||0) 1/10
			5 day price increase percentage is greater than that of Dow Jones index?(n4)
			if(n4=1)(true)
			return num1-num2

Figure 3.1 The inputs to the network

VAN	VDV	VRN	VRG

number of days in which the volume is greater than the 5-day volume moving average(n1)
0<=n1<=10
if(n1=5) return 5/10
if(n1=4) 4/10
if(n1=3) 3/10
if(n1=2) 2/10
if(n1=1||=0) 1/10
	Number of days in which the volume increase percentage is greater than that of Dow Jones index (n2) 0<=n2<=10
	if(n2=5) return 5/10
	if(n2=4) 4/10
	if(n2=3) 3/10
	if(n2=2) 2/10
	if(n2=1||n2=0) 1/10
	5P%>DP%=20;
		Number of days in which the volume of current day is greater than the last day(n3)
		0<=n3<=10
		if(n3=5) return 5/10
		if(n3=4) 4/10
		if(n3=3) 3/10
		if(n3=2) 2/10
		if(n3=1||0) 1/10
			5 day price increase percentage is greater than that of Dow Jones index?(n4)
			if(n4=1)(true)
			return num1-num2;

Figure 3.2 The inputs to the network

Other factors may be added to the input vector easily.

The choice in this way is arbitrary. A more thorough study would choose the best factors from a large set. As we said above, there are more than one hundred factors that can affect the price movement of a particular stock. A more thorough study should begin with many factors and eliminate factors step by step until similar results are obtained with as few factors as possible. That means if a factor doesn't affect or only slightly affects the result of the neural network (after testing), this factor will be removed from the input vector.

3.2.4 Transform of the Data

The input vector obtained from the raw data must be transformed into the range between -1 and 1 to feed the neural networks. This work is done by the preprocessor of the system.

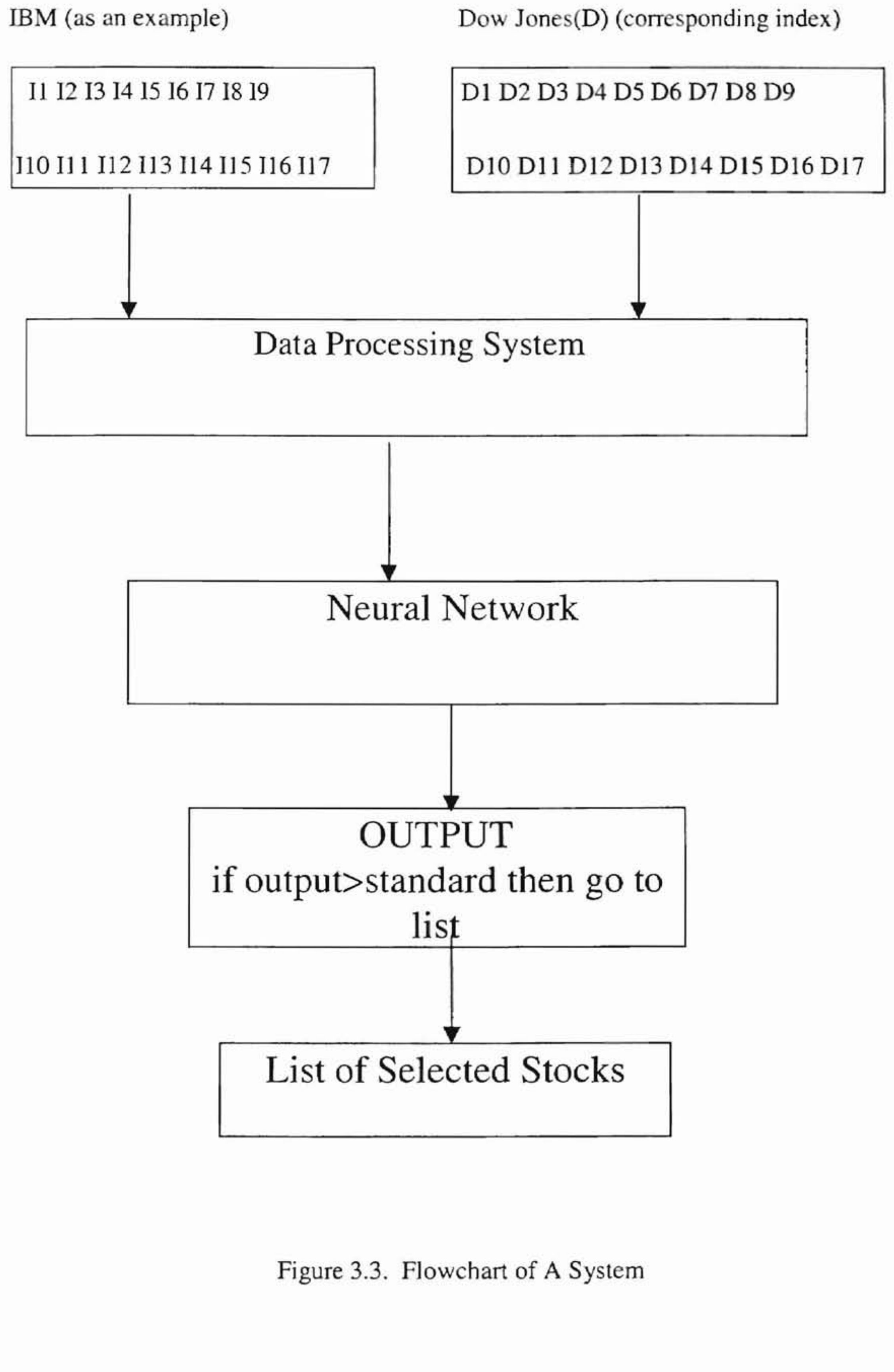
IBM (as an example)
Dow Jones(D) (corresponding index)
I1 I2 I3 I4 I5 I6 I7 I8 I9
I10 I11 I12 I13 I14 I15 I16 I17
D1 D2 D3 D4 D5 D6 D7 D8 D9
D10 D11 D12 D13 D14 D15 D16 D17
Data Processing System
Neural Network
OUTPUT
if output>standard then go to list
List of Selected Stocks

Figure 3.3. Flowchart of A System

3.2.5 Design of Neural Network

According to Hush and Horne [81], neural networks can be partitioned into two categories: static networks and dynamic networks. Static works are characterized by node equations that are memoryless. Representative static networks are:

1. Multi-layer Perceptron networks (MLP).

2. Radial Basis Function networks (RBF).

Dynamic networks are systems with memory. Their node equations are typically described by differential or difference equations. Representative networks are Hopfield networks and recurrent neural networks.

In the following I will describe the two most widely used static networks (MLP and RBF) and make a comparison of them.

3.2.5.1. MLP in General.

Typically, an MLP consists of a set of source nodes that form the input layer, one or more hidden layers of computation nodes, and an output layer of computation nodes. The input signal propagates through the network in a forward direction, on a layer-by-layer basis.

MLPs have been applied successfully to solve some difficult problems by training them in a supervised manner with a highly popular algorithm known as the *error back-propagation algorithm*. This algorithm is based on the *error-correcting learning rule* (also known as the *delta rule*).

Basically, the error backpropagation process consists of two passes through the different layers of the network: a forward pass and a backward pass. In the forward pass, a pattern (input vector) is processed forward through the first hidden layer, its outputs from the first hidden layer continue forward to the next layer and its effects propagate through the network, layer by layer. Finally, one or some outputs are produced as the actual outputs of the network. Then the error signal is computed by the formula:

error = expected output – actual output.

The error is then propagated backward through the network; in this way it comes by the name " error backpropagation". During the forward pass the weights of the neural network are all fixed. During the backward pass, on the other hand, the weights are all adjusted in accordance with the error correction rule.

The backpropagation method is based on the Steepest Gradient Descent method. Other optimization methods include Conjugate Gradient, Newton, Quasi-Newton, and Genetic Algorithms. For the MLP network, this thesis will use a Conjugate Gradient method, which shows a great improvement in speed over backpropagation method.

An MLP has two distinctive characteristics:

a. At least one nonlinear computation will take place in MLP neural networks. Some MLP neural networks use this computation only in the hidden layer; other MLP networks use this computation also in the output layer. A commonly used nonlinearity is a sigmoidal nonlinearity defined by the logistic function [81]:

$$Yj = \frac{1}{1+\exp(-Vj)} \qquad (3.1)$$

where Vj is the net internal activity level of neuron j, and Yj is the output of the neuron.

b. The network contains one or more layers of hidden neurons that are not part of the input or output of the network. More hidden layers don't necessarily mean better performance. A typical MLP neural network usually has no more than three hidden layers [82]; the most common numbers of hidden layers are one and two.

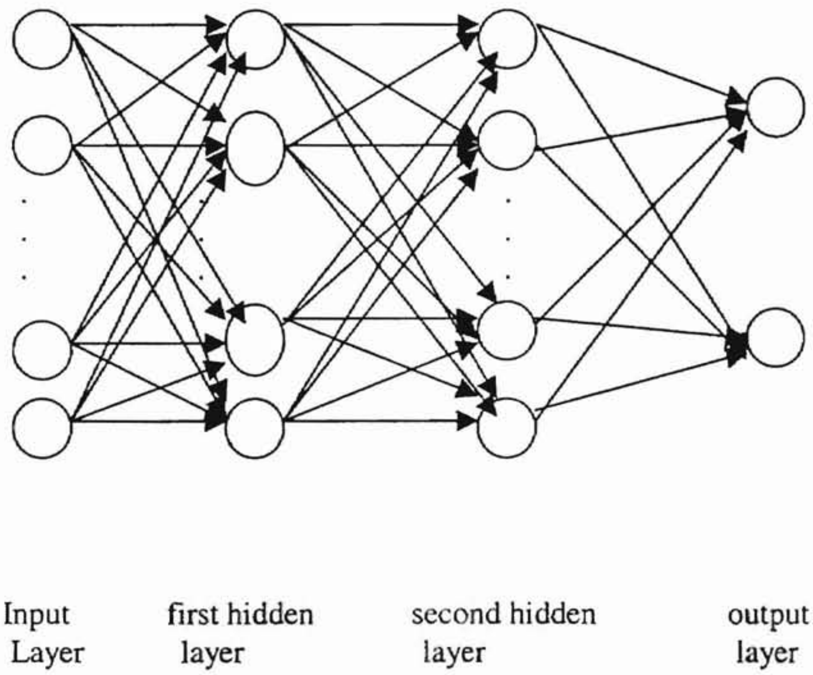

Figure 3.4 Architectural graph of a MLP with two hidden layers [81].

The MLP can complete both approximation and pattern recognition tasks. However, there are a number of practical concerns:

1. The first is the choosing of the network size.
2. The second is the time used on learning (training). That is, we may ask if it is possible to learn the desired mapping in a reasonable amount of time.

3. The third is the ability of the neural network to generalize: that is, its ability to produce accurate results on new samples outside the training set.
4. A key shortcoming in the MLPs as often implemented is the large amount of training time required by the backpropagation algorithm [82]. But backpropagation is not required in an MLP, and we will not be using it .

3.2.5.2. RBF in General

RFB networks have been used successfully in many fields in which they have proved to be good enough to replace multilayer perceptrons (MLPs). These fields include chaotic time-series prediction, speech recognition, and data classification. In addition, the RBF network is a widely used approximator if given enough hidden units. This means an RBF may need more hidden nodes than an MLP neural network. The RBF architecture has exactly three layers (Figure 2), an input layer which performs no computation, a hidden layer, and an output layer. You may notice that there are no weights between the input layer and the hidden layer. Instead, there is a function used to map each input vector with each hidden node. The defining feature of an RBF as opposed to other neural networks is that the basis functions, the transfer functions of the hidden units, are radially symmetric.

The function computed by a general RBF network is therefore of the form

$$F(\xi, w) = \sum_{b=1}^{k} w_b s_b (\xi) , \tag{3.2}$$

where ξ is the vector applied to the units and s_b denotes a basis function [81].

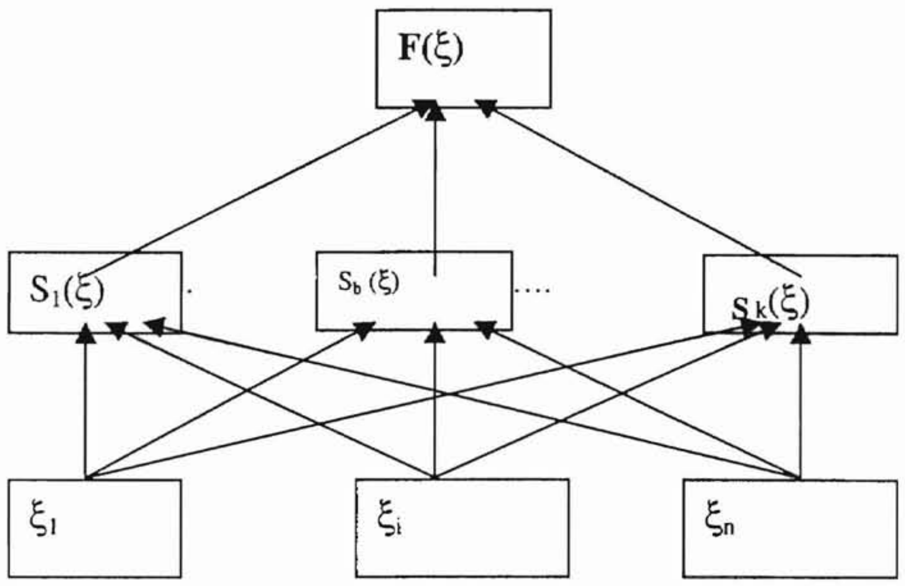

Figure 3.5 A radial basis function network. Each of the N components of the input vector ξ feeds forward to K basis functions whose outputs are linearly combined with weights $\{w_b\}_{b=1}^{K}$ into the network output f (ξ) [81].

The most common choice for the basis functions is the Gaussian, in which case the function computed becomes [81]

$$F(\xi, w) = \sum_{b=1}^{k} w_b \exp\left(\frac{-\| \xi - m_b \|^2}{2\sigma^2_b}\right), \tag{3.3}$$

where each hidden node is parameterized by two quantities: a center **m** in input space, corresponding to the vector defined by the weights between the node and the input nodes, and a width σ_b.

There are two commonly used methods for training RBFs.

1. One method used to train the RBF neural networks is to fix both the radial basis function center and widths. The centers can be set using an unsupervised training method such as clustering, or the user can just randomly choose input vectors from the training file as centers. After setting the centers, only the hidden-to-output weights are adaptable when training, which makes the problem linear in those weights. Although fast to train, this approach has some drawbacks due to its fixed centers and widths.

2. The other method is to adapt the hidden-layer parameters, either just the center position or both center positions and widths. The second method is more general in form. Some reports show that the second method performs better than the first method [58]. The great investment of time in adjusting the positions of the centers should yield considerable improvement, one could hope.

3.2.5.3 Comparison of RBF Networks and MLP Networks

Radial basis function networks and multilayer perceptrons are examples of nonlinear layered feedforward networks. They can both do tasks like approximation and pattern recognition. You may replace an MLP neural network with an RBF neural network to solve the same problem. However, there are still some distinctive differences between them.

a. An RBF network only has one hidden layer, whereas an MLP may have one or more hidden layers.

b. Usually, a sigmoid function will be used both in hidden layers and the output layer in an MLP neural network. But in RBF neural networks, a radial basis function is used only in the hidden layer, and usually a linear function is used in the output layer. This means that both hidden layers and output layer are nonlinear in MLP neural networks whereas in RBF networks, the hidden layer is nonlinear and the output layer is linear.

c. The radial basis function of each hidden unit in an RBF network computes the *Euclidean norm (distance)* between the input vector and the center of that unit. On the other hand, the activation function of each hidden unit in an MLP computes the inner product of the input vector and the weight vector of that unit.

d. The training of an RBF network can be done separately, that is, its hidden layer and output can be trained separately using different methods, thus saving training time. MLP neural networks can't be trained separately; therefore more time would be needed in training MLP networks. But in order to reach a similar result, the number of radial basis functions needed for an RBF may have to be very large.

e. The MLP is a traditional neural network technique, where RBF is the most recent popular one. Barr's patent [83] uses MLPs networks implemented using the generalized delta rule (also called error-correcting learning rule). The hidden layer of nodes is split into two groups: a) a group of nodes using a Gaussian activation function; and b) a second group of nodes using a Gaussian complement activation function. The output node layer uses a logistic activation function.

Kosaka's patent [84] uses a Hopfield network. No training method or activation function information is provided.

This thesis will build an MLP neural network stock selection system using a conjugate gradient optimization method and two RBF neural network stock selection systems using the two methods we described above, train and test them with the same data, compare their results, and make a report. Some experiments report that an RBF network is better than an MLP network in applications of prediction [58]. We will see if the results of our experiments support that conclusion.

3.2.5.4 MLP Network Using Conjugate Gradient Optimization

MLP networks with backpropagation (based on the steepest decent method) is the most popular type of neural network in past years. But more advanced optimization methods have been used in recent years, of which conjugate gradient algorithms, quasi-newton methods, simulated annealing algorithms and genetic optimization algorithms are the widely used ones [105]. The conjugate gradient algorithms gained notice in the optimization world because for a wide class of problems they promise convergence to an optimal solution in a finite number of steps. This is a big improvement over steepest descent, which is guaranteed to require an infinite number of steps if it does not reach an optimal solution on its first iteration.

The name "conjugate" comes from the use of "conjugate direction" vectors. In a vector space of dimension D, a set of D-vectors $\{P_1, \ldots P_d\}$ is said to form a set of A-conjugate directions if

$$P_i * A_{pj} = 0 \text{ for } i \neq j \qquad (3.4)$$

where A is a D by D positive definite matrix (a positive definite matrix is a square matrix all of whose eigenvalues are positive). Vectors satisfying Equation (3.4) are said to be A-conjugate [105].

How does the conjugate gradient algorithm achieve finite convergence? Suppose we wish to minimize the quadratic function

$$F(w)=(b-Aw)^*(b-Aw) \tag{3.5}$$

where b and w are D-vectors, and A is as said above. Suppose we are iteratively searching for the optimal W* that minimizes F(w), and we currently have some guess W0. Pick a nonzero vector P1 which serves as the search direction for the next iteration. Choose W1 to be the vector

$$W1=W0+\alpha P1 \tag{3.6}$$

where the α has been chosen to minimize $F(W0 + \alpha P1)$. Now comes the key point. The optimal direction to go in the next iteration, must form an A-conjugate pair with P1. The optimal direction is W*-W1, so the A-conjugate condition

$$(W^*-W1) * A\ P1=0$$

must be satisfied. We don't know W* at this point. However, in a D-dimension vector space, there can be only D-1 independent vectors that form an A-conjugate pair with P1. Thus, we have only a finite number of directions to search in order to find the optimal direction.

The version of the conjugate gradient algorithm which we will implement is shown in Figure 3.6

```
0. Initialize the weight vector W, and compute the gradient $G=grad_E(W)$
   of        E at W. Set the initial direction vector p equal to $-G/||G||$.
1. Find $\alpha$ which minimizes $E(W + \alpha p)$. Set new $W=W + \alpha p$.
2. If E(new W)<error tolerance, or the stepsize becomes very small,
   then Stop. Else compute new direction:
   New $G=grad_E$(new W).
   If (# iterations mod #weights)=0 then new direction =
   -(new G)/|| new G||;
   Else{
         $\beta$= (new G) **(new G)/G*G**
         new direction vector= -(new G) + $\beta$p/||-(new G) + $\beta$||}
3. Replace G with new G, and p with new direction vector. Go to 1.
```

Figure 3.6 The Conjugate Gradient Algorithm [105].

3.2.5.5 RBF Self-Organized Selection of Centers Neural Network

In RBF self-organized networks, the centers of the hidden layer are set using an unsupervised training method. A common choice for this is the k-mean cluster algorithm. The linear weights of the output layer are computed using a supervised learning rule, usually the least mean square method. In other words, the network undergoes a hybrid learning process [58]. The self-organized training method (k-mean) can set the centers only in these places where the most input data fall.The number of the centers of the network is fixed. For the self-organized selection of the hidden unit's centers, we may use the standard k-means clustering algorithm [74].

```
Procedure K_MEANS
  Initialize the cluster centers Wj, j=1,2,...,N
   //typically these are set equal to the first N training examples
   repeat
       //group all patterns with the closest cluster center
       for all Xi do
           Assign Xi to Wj* where Wj*=min||Xi-Wj||
       Endloop
       //compute the simple mean
       for all Wj do
           $W_j = \frac{1}{M_j} \sum X_i$;
           Where Wj*∋Xi;
       Endloop
   Until there is no change in cluster asssignments
End {K_MEANS}
```

Figure 3.7 K-Means Clustering Algorithm [74]

In Figure 3.7, the norm $\|X_i-W_j\|$ is a weighted norm; it usually is the Euclidean norm [58]:

$$D_{ij} = \|X_i-X_j\| = \left[\sum_{n=1}^{N} (X_{in}-X_{jn})^2\right]^{1/2} \qquad (3.7)$$

For the output layer, we may use the supervised least-mean-square (LMS) learning method; the outputs of the hidden units in the RBF network serve as the inputs of the LMS algorithm. Other optimization methods also can be used for the output layer of the RBF neural networks. For comparison purposes, we only use the traditional methods.

```
Procedure LMS
    Initialize the weights to small random value j=1,2,...,N
    Repeat
        Choose next training pair (Ul,D);
        //compute outputs
        for all j do
        Yj=Wj*Ul;
        Endloop
        //compute error
        for all j do
        Ej=Yj-Dj;
        Endloop

        //Update weights
        for all j do
        Wj(k+1)=Wj(k)-u*Ej*Ul;
        Endloop
      Until termination condition reached
End; {LMS}
```

Figure 3.8 LMS Algorithm [74]

The fixed center RBF is considered to be a "sensible" approach [58], because the centers are fixed. Only when the training data are distributed in a representative manner for the problem at hand can we finish the task effectively. To solve this problem, an adaptive center method can be used in selecting the centers.

3.2.5.6 RBF Adaptive Center Network

In this method, the centers of the radial basis functions and all other free parameters of the network undergo a supervised learning process; in other words, the RBF network takes on its most generalized form. In addition to what we do in self-organized RBF network described above, we allow the number of centers to increase under certain conditions. That means, we keep a record of the biggest distance (D1) between the selected centers. When a new pattern arrives, we compute its distance from all the selected centers. If the biggest distance (B2) between the new pattern and the selected centers is bigger than B1, then a new center is created. That is, the new pattern will be set as the new center. In this way, even if the training data are not distributed in a representative manner, we will not worry about the spread of the centers.

Another characteristic of the adaptive center network is that after the selection of the centers using the k-mean clustering algorithm, we still allow the centers to be modified according to some criteria. The usual criterion is that only the nearest center to the coming pattern can be modified whereas other centers are unchanged. So in this method, both hidden layer and output layer undergo a supervised training.

3.2.6 Implementation of the Networks

3.2.6.1 Stopping Criteria:

1. The maxmum training cycles reach.
2. The average error per cycle goes below the tolerance rate.
3. The step sizes becomes very small.

3.2.6.2 On-line or Off-line Training

On-line learning means that if a pattern is input and an error produced, the weights will be updated at once. Off-line learning means that the updating of the weights will be postponed until all training patterns have been processed and all errors accumulated. In the on-line mode, it is sometimes claimed that the learning process is more sensitive to each pattern [104], so it may help to find the global minimum. For an adaptive RBF network, both on-line training and off-line training are used. For the k-means cluster algorithm, only off-line training can be used. For modification of the centers, usually on-line training is used, because each time we update the centers, we only update the nearest center.

3.2.6.3 Outputs of the Networks

Type	Architecture	Cycles	Learning Rate	Average Error In Training Last Cycle	Average Error In Test File	Largest Individual Error
FRBF	8-10-1	500	0.00009	0.0592784	0.0611237	0.205
AFRBF	8-10-1	500	0.0013	0.0599917	0.0657163	0.275
MLP	8-10-1	432		0.0399974	0.0635255	0.285

Table 3.1 Records for NNS Training and Test

3.2.6.4 Explanation of the preprocessor and simulators

All the coding of the programs is in C++ , and the programs can be run in Unix or Visual C++ 6.0

All function names and variables are meaningful; necessary comments are included.

The preprocessor (Appendix C)

The preprocessor will process raw data from file "rawdata" and produce input factors for the neural network simulators. It also transforms the input factor value to lie between –1 and 1. The output from processing file "rawdata" is saved in file "training" which will be used by all neural network simulators of this project as input. It will also process the file "testdata" and write its outputs into a file named "testfile". The test file will also be used by all simulators to test their performances. Each line in file "rawdata" and "testdata" has the form we described in section 3.2.2. Each line in file "training" and file "testfile" has the following form (see Appendix A and Appendix B):

Line n+1, n = 0,1,2,...N are inputs to the simulator (also called one pattern or input vector);

Line n+2, n = 0,1,2,...N are the desired output for the above eight inputs.

The Simulators

The fixed-center radial basis function simulator uses the method described in section 3.3.5.5. Training of the simulator is conducted on the file "training" produced by the preprocessor, and testing is conducted on the file "testfile". After compiling and begin to run, there appears on the screen a user interface, telling you to choose an architecture and other parameters for the simulator. You may follow the instructions easily. Though only 8-10-1 and 8-4-1 architectures have been tested, you may have as many as 100 hidden nodes if needed.

You may see the output for "testfile" in Table 3.2. In Table 3.2. each four lines represent the result of a pattern from testfile.

Line 1: "for pattern"

Line 2: the inputs to the simulator (one pattern)

Line 3: target (desired output)

Line 4: actual output from simulator

The MLP Simulator uses Conjugate Gradient Optimization method described in section 3.2.5.4. Tt uses the same training file and test file as described above for the FRBF simulator.Though only 8-10-1 and 8-4-1 architectures have been tested, you may have as many as 100 hidden nodes if needed. Also you may try two or more hidden layers for this simulator.

You may see the output for the "testfile" in Table 3.3. In Table 3.3. each five lines represent the result of a pattern from the testfile. The first four lines function the same as the FRBF output described above. The fifth line is the following:

Line 5: the difference between the target output and the actual output.

The Adaptive Radial Basis Function Simulator (ARBF) uses the method described in section 3.3.5.6. It uses the same training file and test file as described above. You may see the output for "testfile" in Table 3.4. In Table 3.4. each first two lines have the same functions as described above for FRBF and MLP simulators. The third line is:

Line 3: actual output and desired output

The following is the output from the FRBF simulator

```
---------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------

 for pattern:
 0.444444 0.222222 0.222222 0.199083 0.555556 0.444444 0.666667 1
 target is: 0.0275229
 actual output from neural network is: 0.106739

 for pattern:
 0.666667 0.333333 0.444444 0.0242453 0.333333 0.111111 0.333333 1
 target is: 0.043755
 actual output from neural network is: 0.117221

 for pattern:
 0.666667 0.555556 0.444444 0.111211 0.222222 0.444444 0.444444 1
 target is: 0.0031348
 actual output from neural network is: 0.11455

 for pattern:
 0.333333 0.555556 0.555556 -0.0120284 0.444444 0.444444 0.444444 1
 target is: 0.297297
 actual output from neural network is: 0.115015

 for pattern:
 0.888889 0.555556 0.666667 0.021788 0.333333 0.444444 0.444444 0
 target is: 0.0317757
```

```
actual output from neural network is: 0.0447694

for pattern:
0.111111 0.222222 0.333333 -0.0434186 0.333333 0.444444 0.444444 1
target is: 0.0925926
actual output from neural network is: 0.113244

for pattern:
0.555556 0.333333 0.444444 0.123203 0.222222 0.444444 0.555556 0
target is: 0.0532915
actual output from neural network is: 0.0572898

for pattern:
0.555556 0.333333 0.444444 -0.0134872 0.333333 0.444444 0.222222 1
target is: 0.0401753
actual output from neural network is: 0.120587

for pattern:
0.333333 0.444444 0.555556 -0.010056 0.111111 0.444444 0.333333 0
target is: 0.0708661
actual output from neural network is: 0.0603566

for pattern:
0.333333 0.555556 0.333333 -0.0192748 0.222222 0.444444 0.333333 1
target is: 0.0510949
actual output from neural network is: 0.11952

for pattern:
0.444444 0.222222 0.222222 -0.0120284 0.222222 0.444444 0.444444 0
target is: 0.0536585
actual output from neural network is: 0.0646472

for pattern:
0.444444 0.444444 0.444444 -0.0246181 0.333333 0.333333 0.333333 0
target is: 0.0348259
actual output from neural network is: 0.0630041

for pattern:
0.555556 0.444444 0.444444 0.0520486 0.555556 0.333333 0.555556 0
target is: 0.167183
actual output from neural network is: 0.0527304

for pattern:
0.555556 0.555556 0.555556 -0.0215317 0.444444 0.333333 0.444444 0
target is: 0.0564103
actual output from neural network is: 0.0537227

for pattern:
0.111111 0.333333 0.333333 -0.08085 0.444444 0.222222 0.555556 0
target is: 0.170147
actual output from neural network is: 0.0592208

for pattern:
0.666667 0.555556 0.444444 0.0253106 0.555556 0.333333 0.555556 0
target is: 0.0217391
actual output from neural network is: 0.0487529

for pattern:
0.444444 0.444444 0.444444 0.0161982 0.444444 0.333333 0.444444 0
target is: 0.346405
actual output from neural network is: 0.0594768
```

```
for pattern:
0.666667 0.444444 0.444444 0.040317 0.444444 0.333333 0.666667 0
target is: 0.142857
actual output from neural network is: 0.0498557

for pattern:
0.444444 0.444444 0.555556 0.0657433 0.333333 0.333333 0.444444 0
target is: 0.0110497
actual output from neural network is: 0.0589694

for pattern:
0.444444 0.333333 0.444444 0.052823 0.555556 0.333333 0.555556 0
target is: 0.1375
actual output from neural network is: 0.0554948

for pattern:
0.555556 0.444444 0.444444 0.00251368 0.333333 0.333333 0.333333 0
target is: 0.106918
actual output from neural network is: 0.0613181

for pattern:
0.222222 0.222222 0.333333 -0.0534004 0.222222 0.333333 0.444444 0
target is: -0.0273556
actual output from neural network is: 0.0669927

the average error for testfile is : 0.0768373
Total patterns is :  22
```

Table 3.2 Output for FRBF Test File (above)

The following is output from the MLP simulator

```
For pattern :
0.444444 0.222222 0.222222 0.199083 0.555556 0.444444 0.666667 1

The actual output from network is:  0.0465932

The target output is : 0.0275229

The difference between target and actual is:0.0190703
For pattern :
0.666667 0.333333 0.444444 0.0242453 0.333333 0.111111 0.333333 1
The actual output from network is:  0.0518808
```

```
 The target output is :  0.043755

 The difference between target and actual is:0.00812577
 For pattern :
0.666667 0.555556 0.444444 0.111211 0.222222 0.444444 0.444444 1
The actual output from network is:   0.0523276

 The target output is :  0.0031348

 The difference between target and actual is:0.0491928
 For pattern :
0.333333 0.555556 0.555556 -0.0120284 0.444444 0.444444 0.444444 1
The actual output from network is:   0.0418075

 The target output is :  0.297297

 The difference between target and actual is:0.25549
 For pattern :
0.888889 0.555556 0.666667 0.021788 0.333333 0.444444 0.444444 0
The actual output from network is:   0.081526

 The target output is :  0.0317757

 The difference between target and actual is:0.0497503
 For pattern :
0.111111 0.222222 0.333333 -0.0434186 0.333333 0.444444 0.444444 1
The actual output from network is:   0.0379364

 The target output is :  0.0925926

 The difference between target and actual is:0.0546562
 For pattern :
0.555556 0.333333 0.444444 0.123203 0.222222 0.444444 0.555556 0
The actual output from network is:   0.0738734

 The target output is :  0.0532915

 The difference between target and actual is:0.0205819
 For pattern :
0.555556 0.333333 0.444444 -0.0134872 0.333333 0.444444 0.222222 1
The actual output from network is:   0.043139

 The target output is :  0.0401753

 The difference between target and actual is:0.00296368
 For pattern :
0.333333 0.444444 0.555556 -0.010056 0.111111 0.444444 0.333333 0
The actual output from network is:  0.0624653

 The target output is :  0.0708661

 The difference between target and actual is:0.00840076
 For pattern :
0.333333 0.555556 0.333333 -0.0192748 0.222222 0.444444 0.333333 1
The actual output from network is:  0.0398034

 The target output is :  0.0510949

 The difference between target and actual is:0.0112915
 For pattern :
0.444444 0.222222 0.222222 -0.0120284 0.222222 0.444444 0.444444 0
The actual output from network is:  0.0619681

 The target output is :  0.0536585

 The difference between target and actual is:0.00830961
 For pattern :
0.444444 0.444444 0.444444 -0.0246181 0.333333 0.333333 0.333333 0
The actual output from network is:  0.0599728

 The target output is :  0.0334825

 The difference between target and actual is:0.0264903
 For pattern :
0.555556 0.444444 0.444444 0.0520486 0.555556 0.333333 0.555556 0
The actual output from network is:  0.0661583

 The target output is :  0.167183

 The difference between target and actual is:0.101025
 For pattern :
0.555556 0.555556 0.555556 -0.0215317 0.444444 0.333333 0.444444 0
The actual output from network is:  0.0658878

 The target output is :  0.0564173

 The difference between target and actual is:0.00947054
 For pattern :
0.111111 0.333333 0.333333 -0.08085 0.444444 0.222222 0.555556 0
The actual output from network is:  0.0546183
```

```
 The target output is :  0.170147

 The difference between target and actual is:0.115529
 For pattern :
0.666667 0.555556 0.444444 0.0253106 0.555556 0.333333 0.555556 0
The actual output from network is:   0.0684791

 The target output is :  0.0217391

 The difference between target and actual is:0.04674
 For pattern :
0.444444 0.444444 0.444444 0.0161982 0.444444 0.333333 0.444444 0
The actual output from network is:   0.0613634

 The target output is :  0.346405

 The difference between target and actual is:0.285042
 For pattern :
0.666667 0.444444 0.444444 0.040317 0.444444 0.333333 0.666667 0
The actual output from network is:   0.0761588

 The target output is :  0.142875

 The difference between target and actual is:0.0667162
 For pattern :
0.444444 0.444444 0.555556 0.0657433 0.333333 0.333333 0.444444 0
The actual output from network is:   0.0665221

 The target output is :  0.0110497

 The difference between target and actual is:0.0554724
 For pattern :
0.444444 0.333333 0.444444 0.052823 0.555556 0.333333 0.555556 0
The actual output from network is:   0.0637077

 The target output is :  0.1375

 The difference between target and actual is:0.0737923
 For pattern :
0.555556 0.444444 0.444444 0.00251368 0.333333 0.333333 0.333333 0
The actual output from network is:   0.0634703

 The target output is :  0.106918
```

The difference between target and actual is:0.0434477
For pattern :
0.222222 0.222222 0.333333 -0.0534004 0.222222 0.333333 0.444444 0
The actual output from network is: 0.0586475

The target output is : -0.0273556

The difference between target and actual is:0.0860031
The average error for testfile is: 0.0635255
The number of predict error less than 0.05 is: 14

The number of all test pattern is: 22

Table 3.3 MLP Output for Test file

The following is output from the ARBF simulator

```
 for pattern:
0.444444 0.222222 0.222222 0.199083 0.555556 0.444444 0.666667 1
 the output is :  0.0708336   expected output is  0.0275229
 for pattern:
0.666667 0.333333 0.444444 0.0242453 0.333333 0.111111 0.333333 1
 the output is :  0.0789903   expected output is  0.043755
 for pattern:
0.666667 0.555556 0.444444 0.111211 0.222222 0.444444 0.444444 1
 the output is :  0.0782648   expected output is  0.0031348
 for pattern:
0.333333 0.555556 0.555556 -0.0120284 0.444444 0.444444 0.444444 1
 the output is :  0.0860295   expected output is  0.297297
 for pattern:
0.888889 0.555556 0.666667 0.021788 0.333333 0.444444 0.444444 0
 the output is :  0.0661415   expected output is  0.0317757
 for pattern:
0.111111 0.222222 0.333333 -0.0434186 0.333333 0.444444 0.444444 1
 the output is :  0.0751913   expected output is  0.0925926
 for pattern:
0.555556 0.333333 0.444444 0.123203 0.222222 0.444444 0.555556 0
 the output is :  0.0406358   expected output is  0.0532915
 for pattern:
0.555556 0.333333 0.444444 -0.0134872 0.333333 0.444444 0.222222 1
 the output is :  0.0775074   expected output is  0.0401753
 for pattern:
0.333333 0.444444 0.555556 -0.010056 0.111111 0.444444 0.333333 0
 the output is :  0.0500109   expected output is  0.0708661
 for pattern:
0.333333 0.555556 0.333333 -0.0192748 0.222222 0.444444 0.333333 1
```

```
 the output is :  0.0799577   expected output is  0.0510949
 for pattern:
0.444444 0.222222 0.222222 -0.0120284 0.222222 0.444444 0.444444 0
 the output is :  0.0315316   expected output is  0.0536585
 for pattern:
0.444444 0.444444 0.444444 -0.0246181 0.333333 0.333333 0.333333 0
 the output is :  0.0607042   expected output is  0.0348259
 for pattern:
0.555556 0.444444 0.444444 0.0520486 0.555556 0.333333 0.555556 0
 the output is :  0.0710698   expected output is  0.167183
 for pattern:
0.555556 0.555556 0.555556 -0.0215317 0.444444 0.333333 0.444444 0
 the output is :  0.0792015   expected output is  0.0564103
 for pattern:
0.111111 0.333333 0.333333 -0.08085 0.444444 0.222222 0.555556 0
 the output is :  0.062005   expected output is  0.170147
 for pattern:
0.666667 0.555556 0.444444 0.0253106 0.555556 0.333333 0.555556 0
 the output is :  0.0738595   expected output is  0.0217391
 for pattern:
0.444444 0.444444 0.444444 0.0161982 0.444444 0.333333 0.444444 0
 the output is :  0.0718764   expected output is  0.346405
 for pattern:
0.666667 0.444444 0.444444 0.040317 0.444444 0.333333 0.666667 0
 the output is :  0.0625335   expected output is  0.142857
 for pattern:
0.444444 0.444444 0.555556 0.0657433 0.333333 0.333333 0.444444 0
 the output is :  0.0667353   expected output is  0.0110497
 for pattern:
0.444444 0.333333 0.444444 0.052823 0.555556 0.333333 0.555556 0
 the output is :  0.0670063   expected output is  0.1375
 for pattern:
0.555556 0.444444 0.444444 0.00251368 0.333333 0.333333 0.333333 0
 the output is :  0.0545737   expected output is  0.106918
 for pattern:
0.222222 0.222222 0.333333 -0.0534004 0.222222 0.333333 0.444444 0
 the output is :  0.0414382   expected output is  -0.0273556
 the average error for test fie is 0.0657163 total pats are: 22
```

Table 3.4 ARBF Output to Test File (above)

CHAPTER 4

DISCUSSION OF RESULTS

4.1 Things said before making comparison

1. If more time could have been spent on this project, and if this project could access a database, the results from the simulators would be better than they appear in this thesis. Though the output is also one of our concerns, our main concern lies in comparison of the performances of different neural network simulators in the same conditions. If the condition is bad, it is bad for all simulators. In such a case, we may make a comparison of their performances.

2. For the architecture of the neural networks, we only choose 8-10-1. The main concern in doing so is that the RBF networks need more hidden nodes to make a good performance but MLP networks need to have few hidden nodes to avoid overfitting. Typically, the number of hidden nodes for a RBF network should be about twice the number of nodes in its input layer or at least, and the number of hidden nodes should be greater than the number of input nodes [58]. Some books [67] say that the number of hidden nodes in RBF networks can be equal to the number of the total training patterns. In our case, the number of hidden nodes in the RBF networks should be greater than 8; that means at least 9. Now consider our MLP network: the total number of patterns in our training file is 100, and the number of

input nodes is 8. Assume we choose X hidden nodes for our networks, then we can compute the number of weights for an MLP network by

Number of Weights $= 8 * X + X * 1 = 9X$.

The number of weights should be less than the number of total patterns to avoid overfitting. In our case, there are 100 patterns. We get

$9X<100$.

By solving the above inequality, we get the largest X to be 11. This means our MLP network can only afford at most 11 hidden nodes. As said above, our RBF networks need at least 9 hidden nodes. In such a way, we are left only with three choices in setting the number of hidden layer. That is 9,10,11. To make a compromise, 10 was chosen in our project. Such a result may greatly affect the performance of RBF networks. For this reason, the comparison we will make may be less than persuasive. It may be that requiring the MLP and RBF networks to have the same numbers of nodes ia an artificial restriction that should be relaxed in future work.

3. The learning rates for RBF networks were chosen through many tests. Learning rates less or greater than the chosen rate showed no better results. More training than 500 cycles did not bring much benefit to the performance of the networks. The error tolerance for the MLP was 0.04. It hit this limit after 432 cycles and stopped.

From Table 3.1 to Table 3.4, we may get another table:

Type	Architecture	LIE	SIE	NL1	NL3	NL5	NL6
MLP	8-10-1	0.285	0.00296	5	9	14	15
FRBF	8-10-1	0.205	0.004	1	8	11	11
ARBF	8-10-1	0.275	0.0126	0	7	12	16

Table 4.1. Outputs comparison (total patterns are 22)

LIE is the largest individual error found in the output from corresponding simulators. SIE is the smallest individual error. NL1 is the number of patterns from the output of a simulator whose error is less than 0.01. NL3 is the number of patterns whose error is less than 0.03. NL5 is the number of patterns whose error is less than 0.05. NL6 is the number of patterns whose error is less than 0.0632. 0.0632 is the average of all three simulator average testfile errors (see Table 3.1)

We are confused by comparing the performances of the networks through Table 3.1 and Table 4.1. MLP ranks first in SIE, NL1, NL3 and NL5, but it ranks last in LIE. ARBF ranks first in NL6 but last in Average Error. FRBF also has some firsts and lasts. We can tell which one is best according to an particular indicator, but that may be too simple to judge. How we can tell which one is the best overall? We need a complex method. So, by merging Table 3.1 and Table 4.1, we may get a rank table for these simulators.

Type	LIE	SIE	NL1	NL3	NL5	NL6	AETR	AETE	SUM
MLP	3	1	1	1	1	2	1	2	12
FRBF	1	2	2	2	3	3	2	1	16
ARBF	2	3	3	3	2	1	3	3	20

Table 4.2 Rank table for all simulators

AETR is the average error in training for the last cycle. AETE is the average error for the test file.

We get the ranks by comparing the values in each indicator. For example, in LIE, MLP is 0.285, FRBF is 0.205, ARBF is 0.275. MLP has the largest individual error, so it ranks last, and FRBF, whose LIE is the smallest one, ranks first. But in NL1, MLP is 5, FRBF is 1 and ARBF is 0; MLP ranks first in this indicator because it has predicted five patterns with an error less than 1%.

The indicators used here have general representatives. Adding or removing some indicators does not materially change the conclusions we can get.

4.2 Now we can discuss the results of the simulators

1. The number of data patterns (NP) in our training file is100. The number of weights (NW) in our neural networks is: NW=8*10+10*1=90. Generally speaking, if NW $\geq$ NP, overfitting will occur. That means the network can remember all possibilities of

the training file. It will get very good results for the training file, but when tested with new cases, it will produce poor results. Usually, the more NP is greater than NW, the better the results are from the test file. Some reports show that NP should be 3 times NW [85] to get satisfying results from the test file. Some books say that NP can be equal to NW [58] for RBF networks. Because the input data for this project was typed in by hand, only a limited number of patterns were provided which will affect the results of the neural networks.

2. Traditionally, MLP neural networks need more training time than do RBF neural networks. But the records (Table 3.1) show that the MLP in our experiments use only 432 cycles to reach below the error tolerance 0.04 while the two RBF networks in our experiments need 500 cycles to reach an error tolerance below 0.06. The reason here is that traditional MLP networks use the backpropagation learning method based on the steepest gradient optimization method. The MLP network used in our experiments used the conjugate gradient optimization method which, as we described in section 3.2.5.4, is a great improvement over backpropagation. In an experiment using the same training file to a backpropagation neural network described by Valluru, Rao [85] showed that 5000 cycles are needed for that network to reach an error tolerance of 0.06.

   The results of this experiment will be described in my later articles.

3. The average errors for the test file are a little larger than that for the training files in the last cycle. This show some suspicion of overfitting. Some similar experiments [88][100] report the same results. Maybe such results are reasonable.

4. Both the training errors and the test errors from the networks are about 0.06, and not much improvement was obtained from further training. This shows that neural networks (even using the most advanced methosd) can't predict exactly the price movements of stocks. Though the best work done in this field may remain unreleased, the financial reports from all mutual funds and other investment institutions tell us that not much better work is done in using neural networks to predict stock price movements, otherwise, the profits on their reports would be much better. There is always a great difference between the price movement in practice and that predicted. Based on this, some researchers claim that stock prices are unpredictable.
5. From Table 4.2, we may get the conclusion that MLP with Conjugate Gradient Optimization performs best in our experiment. Of course we can't say it will surely perform better than RBF networks in other conditions. If you remember what we discussed in section 4.1. you would argue that RBF networks may perform better than MLP networks with more hidden nodes. Things may be that way. That is one of the future works we recommend.
6. Generally, ARBF networks have performed better than FRBF networks for other researchers [74]. But our experiments get the opposite conclusion. We haven't found satisfying reasons for this result.
7. Some readers may find, by looking deep into Table 4.1, that an interesting point appears in indicator NL1. MLP networks trained with the Conjugate Gradient method successfully predicted five patterns with an error less than 1% whereas ARBF predict none. That is a great improvement. After making a deep study, we think the

Conjugate Gradient Optimization method mainly contributed to this success.

Backpropagation method is obsolete.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

Compared with the blueprint we provide in Section 5.2, we only did perhaps 5% of the possible work in this thesis. But we can learn some lessons from what we did in this thesis.

1. Neural networks can't predict exactly the direction of some complicated problems like stock price movements.
2. Advanced optimization methods may play a crucial role in the applications of neural networks, so we should pay close attention to the invention of new optimization methods and actively use them in our projects.
3. Traditional MLP networks need much time in training using backpropagation, but MLP networks with advanced optimization methods can overcome this drawback.
4. Considering the errors from the simulators, the average error is 0.0632. This error is acceptable, so the application of neural networks may help to make a decision on stock selection.
5. A project such as we did in this thesis needs more time and necessary equipment (a large database) to get better results. Full training and extensive choosing of

data will surely bring better results, as will better design of inputs to the simulators.

6. Some measurements should be settled upon to measure and compare the performances of different simulators. The method we provided in this thesis only tries to reach a reasonable standard.
7. A more friendly user interface should be designed to make the application more convenient to use.

5.2 Recommendation

Only 100-stock and two-week data was used in training the simulators because of time and money limit. More data should be used in future works. For comparison purpose, we chose the same architecture for all the simulators. An MLP may need more layers and an RBF may need more hidden neurons to perform the best. Various architectures should be used to each kind of neural networks in future works. The inputs to the neural network simulators were decided according to the author's experiences, a scientific method should be used in future works.

Future works of this project may lead to the invention of the super computer investor using neural networks

Invention of "NeuroInvestor I"

One proposed plan of future research could follow these lines:

"NeuroInvestor I" could be the product of an experiment which consists of 600 NeuroInvestors. All these NeuroInvestors are stock-selection systems using artificial neural networks. These 600 NeuroInvestors are divided into two groups, 300 for each group. Group 1 will be created using a random choice method. Group 2 will use a carefully designed method.

Group 1 will randomly choose its properties from:

1. All kinds of neural networks
2. All kinds of layers
3. All kinds of neurons
4. All possible inputs (more than 100) to the neural network
5. All kinds of transfer functions
6. All kinds of learning rules

The researchers would carefully design each NeuroInvestor of Group 2:

1. Neural network architecture.
2. Layers and neurons
3. Inputs to the neural network.
4. Transfer functions and learning rules

All NeuroInvestors would be trained and tested with the same carefully chosen data. After that, each NeuroInvestor would be given $1 million (virtual) as its capital and begin investing according to real market operations. No intervention would be given to these NeuroInvestors during their operation. After a year's virtual operation, returns from all NeuroInvestors would be ranked. The best NeuroInvestor would be selected. This is the prototype of the "NeuroInvestor I". Research and further work are needed ……

Bibliography

[1] Norman G. Fosback, "*Stock Market Logic*", Dearborn Financial Publishing, Inc. 1995

[2] Thomas R. DeMark, "*New Market Timing Techniques*", John Wiley & Sons, Inc. 1997

[3] Thomas R. DeMark, "*The New Science of Technical Analysis*", John Wiley & Sons, Inc., 1994

[4] Frank Cormier, "*Wall Street's Shady Side*", Public Affairs Press, 1962

[5] Robert M. Barnes, "*Trading in Choppy Markets, Breakthrough Techniques for Exploiting Nontrending Markets*", Times Mirror Higher Education Group Inc. Company. 1997

[6] Steven B. Achelis, "*Technical Analysis From A To Z*", Probus Publishing. 1995

[7] Brendan Moynihan, "*Trading on Exceptions*", John Wiley & Sons, Inc 1997

[8] Thomas J. Dorsey, "*Point and Figure Charting*", John Wiley & Sons, Inc. 1995

[9] Richard J. Maturi, "*Dividing the DOW*", Probus Publishing Company. 1993

[10] John Kenneth Galbraith, "*A Short History of Financial Euphoria*", Viking Penguin, a division of Penguin Books USA Inc. 1993

[11] Victor Sperandeo, "*Trader VIC II—Principles of Professional Speculation*", John Wiley & Sons, Inc 1994

[12] Tushar S. Chande and Stanley Kroll, "*The New Technical Trader, Boost Your Profit by Plugging into the Latest Indicators*", John Wiley & Sons, Inc 1994

[13] Ivan F. Boesky, "*Merger Mania, Arbitrage: Wall Street's Best Kept Money-Making Secret*", Holt Rinehart and Winston 1985

[14] MERGERSTAT—a division of Howlihan Lockey Howard & Zukin, *"1997 MergerSTAT Review"*.

[15] Connie Ferdinandson, "*Merger & Acquisition Sourcebook 1989 Edition*", Quality Service Company, 1989

[16] Peter Lynch & John Rothchild, *"Learn to Earn"*, Simon & Schuster Inc. 1995

[17] J.D. Cowan, "The Problem of Organismic Reliability", Progress in Brain Research 17, 9-63. 1963.

[18] Thomas A. Meyers, *"The Technical Analysis Course, A Winning Program for Investors & Traders"*, McGraw-Hill, 1994

[19] Henry R. Oppenheimer, *"Common Stock Selection, An Analysis of Benjamin Graham's "Intelligent Investor" Approach"*, UMI Research Press. 1981

[20] Benjamin Graham, David L. Dodd, Sidney Cottle, *"Securities Analysis, Principles and Tectonics"*, McGraw-Hill Book Company, Inc. 1962

[21] Spencer McGowan, *"The Investor's Information Sourcebook"*, New York Institute of Finance. 1995

[22] von der Malsburg, "Self-organization of Orietation Sensitive Cells in the Striate Cortex", *Kybernetik* 14, 85-100. 1973.

[23] Fred W. Frailey, *"How to Pick Stocks"*, The Kiplinger Washington Editors, Inc. 1997

[24] Phyllis S. Pierce, *"The IRWIN Investor's Handbook 1995"*, Irwin Professional Publishing 1995

[25] Matthew Bishop and John Kay, *"European Merger and Merger policy"*, Oxford University Press. 1993

[26] Von Neumann, "Probabilistic Logics And the Synthesis of Reliable Organisms From Unreliable Components.". In *Automatic Studies* (C.E. Shannon and J. McCathy. ED), 43-98, Princeton University Press, 1956.

[27] Michael Firth, *"Share Prices and Mergers"*, Saxon House, 1976

[28] Khian Thong Lim, *"Machine Learning Algorithms and Fuzzy Neural Networks: An Experimental Comparison"*, Master's Thesis, Computer Science Department, Oklahoma State University, Stillwater OK 1996

[29] Ping Jiang, *"A Penalty Method to Reduce Overfitting in Artificial Networks"*, Master's Thesis, Computer Science Department, Oklahoma State University, Stillwater, OK, 1996

[30] Douglas Gerlach, *"Investor's Web Guide; Tools and Strategies for Building Your Portfolio"*, Lycos Press. 1997

[31] Roy C. Smith. *"The Money Wars"*, Truman Talley Books, Dutton, 1990

[32] "Webster's Third New International Dictionary of the English Language Unabridged", Merriam-Webster, 1993.

[33] W. K. Talor, "Electrical Simulation of Some Nervous System Function Activities", *In Information Theory* (E.C. Cherry, ED). Vol. 3. 314-328. London Butterworth. 1956

[34] Mark F.Stein, *"Moody's Handbook of Common Stocks, Winter 1988-89"*, Moody's Investors Service. 1989

[35] George W. Stroke, *"An Introduction to Coherent Optics and Holography"*, Academic Press, 1966.

[36] Frank Rosenblatt, *"Principles of Neurodynamics; Perceptrons and the Theory of Brain Mechanism"*, Spartan Books, 1962.

[37] Philip L. Carret, *"The Art of Speculation"*, John Wiley & Sons, Inc. 1997

[38] Maureen Caudill and Charles Butler, *"Understanding Neural Networks: Computer Exploration, Volume 2, Advanced Networks*, MIT Press. 1992

[39] Seymour Schoen and Wendell G. Sykes, *"Putting Artificial Intelligence to Work, Evaluating & Implementing Business Applications"*, John Wiley & Sons, Inc. 1987

[40] Dobrivoje Popovic, Vijay P. Bhatkar, *"Methods and Tools for Applied Artificial Intelligence"*, Marcel Dekker, Inc. 1994

[41] IUI 1998, *"International Conference on Intelligence User Interfaces"*, ACM Press, 1998

[42] Patrick Henry Winston, *" Artificial Intelligence"*, Addison-Wesley Publishing Company, 1984

[43] Ben Du Boulay, David Hogg, Luc Steels, *"Advances in Artificial Intelligence-II"*, North-Holland. 1987

[44] John Haugeland, *"Artificial Intelligence: the Very Ideas"*, MIT Press, 1985

[45] Maureen Caudill and Charles Butler, *"Understanding Neural Networks: Computer Explorations, Volume 1, Basic Networks"*, MIT Press. 1992

[46] Philip C. Jackson, Jr. *"Introduction to Artificial Intelligence*", Mason & Lipscomb Publishers Inc. 1974

[47] Stephen J. Andriole, *"Applications in Artificial Intelligence"*, Petrocelli Books, Inc. 1985

[48] Eric L.Grimson and Ramesh S.Patil, *"AI in the 1980s and Beyond, an MIT Survey"*, MIT Press, 1987

[49] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *"Design Patterns, Elements of Reusable Objected-Oriented Software"*, Addison-Wesley Publishing Company, 1995

[50] Robert Hecht-Nielson, *"Neurocomputing"*, Addison-Wesley Publishing Company, 1990.

[51] John Mylopoulos and Michael L. Brodie, *"Artificial Intelligence&Database"*, Morgan Kaufmann Publishers, Inc. 1989

[52] Robert J. Schalkoff, *"Artificial Intelligence: An Engineering Approach"*, McGraw-Hill, Inc. 1990

[53] Judith E. Dayhoff, *"Neural Network Architectures, An Introduction"*, Van Nostrand Reinhold 1990.

[54] James A. Freeman, David M. Skapura, *"Neural Networks, Algorithms, Applications, and Programming Techniques"*, Addison-Wesley Publishing Company 1992

[55] Derek Patridge and Yorich Wilks, *"The Foundations of Artificial Intelligence A Source Book"*, Cambridge University Press 1990

[56] K.L. Diamantaras and S.Y.Kung, *"Principal Component Neural Networks, Theory and Applications"*, John Wiley & Sons, Inc. 1996

[57] Mohamad, H. Hassoun, *"Fundamentals of Artificial Neural Networks"*, MIT Press, 1995

[58] Simon Haykin, *"Neural Networks, A Comprehensive Foundation"*, Macmillan College Publishing Company, 1994

[59] Joey Rogers, *"Objected-Oriented Neural Networks In C++"*, Academic Press. 1997

[60] Timothy Budd, *"An Introduction to Object-Oriented Programming"*, Addison-Wesley Longman, Inc. 1997

[61] Andreas Paepcke, *"Object-Oriented Programming, the CLOS Perspective"*, MIT Press.1993

[62] Lewis J. Pinson, Richard S. Wiener, *"Applications of Object-Oriented Programming"*, Addison-Wesley Publishing Company. 1992

[63] R. Linsker, "Self-organization In A Perceptron Network", *Computer* 21, 105-117,1988

[64] J.J. Florentine, *"Object Oriented Programming System"*, Chapman & Hall. 1991

[65] Peter Coad and Jill Nicola, *"Object-Oriented Programming"*, Prentice Hall.1993

[66] M.L. Minsky, "Steps Toward Artificial Intelligence", *Processings of the Institute of Radio Engineer* 49, 8-30. 1961.

[67] M.T. Hagan, Howard B. Demuth, Mark Beale, *"Neural Network Design"*, PWS Publishing Company. 1996

[68] W. Pitts, and W.S. McCulloch, 1947. *"How We Know Universals: The Perception Of Auditory And Visual Forms"*, Bulletin of Mathematical Biophysics 9, 127-147.

[69] N. Wiener, 1948. *"Cybernetics"*, New York: Wiley.

[70] D.O. Hebb, 1949. *"The Organization of Behavior"*, New York: Wiley.

[71] W.R. Ashby, 1952. *"Design for a Brain"*. New York: Wiley.

[72] M.L. Minsky, 1954. *"Theory Of Neural-Analog Reinforcement Systems And Its Application To The Brain-Model Problem"*, Ph.D. Thesis, Princeton University, Princeton, NJ.

[73] J.J. Hopfield, 1982. "Neural Networks And Physical Systems", *Proceedings of the National Academy of Sciences of the U.S.A* 79, 2554-2558.

[74] T. Kohonen, 1982. "Self-organized Formation Of Topologically Correct Feature Maps", *Biological Cybernetics* 43, 59-69.

[75] S. Kirpatrick, C.D. Gellat, Jr., and M.P. Vecchi, 1983, *"Optimization By Simulated* Annealing", *Science* 220, 671-680.

[76] V. Braitenberg, 1984. *"Vehicles: Experiments in Synthetic Psychology"*, Cambridge, MA: MIT Press.

[77] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, 1986a, *"Learning Representations* by Back-propagating Errors", *Nature* (London), 323, 533-536.

[78] D.S. Broomhead, and D. Lowe, 1988. "Multivariable Functional Interpolation and Adaptive Networks", *Complex Systems* 2, 321-355

[79] C.A. Mead, and M. Ismail, 1989 S. *"Analog VLSI Implementation of Neural Systems"*, Boston, MA: Kluwer.

[80] D.E. Rumelhart, and J.L. McClelland, 1986. *"Parallel Distributed Processing: Explorations in the Microstructure of Cognition"*, Cambridge, MA: MIT Press.

[81] Don R. Hush and Bill G. Horne, " Progress in Supervised Neural Networks", *IEEE Signal Processing Magazine*, Jan 1993, 8-38.

[82] Cornelius T. Leondes, *" Algorithms and Architectures"*, Academic Press, 1998

[83] Dean S. Barr, *" Predictive neural network means and method for selecting a portfolio of securities wherein each network has been trained using data relating to a corresponding security"*, United State Patent 5,761,442 in 1998

[84] Michitaka Kosaka, *"Method and a system for selection of time series data"*, United States Patent, 5, 109, 475 in 1992

[85] Rao Valluru *" Neural Networks & Fuzzy Logic"*, MIS Press 1995

[86] M. Azoff, *"Neural Network Time Series Forecasting of Financial Markets"*, John Wiley and Sons, New York, 1994.

[87] Robert Trippi *"Neural Networks in Finance and Investing"*, Probus Publishing, 1993

[88] E.M. Azoff, *"Neural Network Time Series Forecasting of Financial Markets"*, John Wiley & Sons, 1994

[89] Robert Trippi, Jae K. Lee, *"Artificial Intelligence in Finance & Investment"*, Irwin Professional Publishing, 1996

[90] Alan Lapedse and Farbe Robert, *"Nonlinear Signal Processing Using Neural Networks, Prediction and System Modelling"*. Los Alamos Report LA_UR-87-2662. Los Alamos National Laboratory, 1987

[91] Brian O'Reilly, "Computers That Think Like People", *Fortune*, 58-61, 27 Feb 1989

[92] D.E. Rumelhart. "Generalization by Weight-Elimination with Application to Forecasting", *Advances in Neural Information Processing Systems* 3, 875-882.

[93] Kanad Chakraborty, "Forecasting the Behavior of Multivariate Time Series Using Neural Networks", *Neural Networks*, Vol. 5, 961-970, 1992.

[94] Karl Bergersonand, "A Commodity Trading Model Based on a Neural Network-Expert System Hybrid." *Proc. IJNN Seattle* 1991, Vol. 1, 289-293. IEEE, Piscataway, NJ, 1991.

[95] Zaremba Thomas. *"Technology in Search of a Buck.* ", Academic Press, 1990.

[96] Shih Y. Lung, "Neural Nets in Technical Analysis", *Technical Analysis of Stocks and Commodities*, Vol. 9, No.2, p 62, February 1991.

[97] Mark B. Fishman and Dean S. Barr, " Using Neural Nets in Market Analysis", *Technical Analysis of Stocks and Commodities*, Vol. 9, No4. Page 18 April 1991

[98] Jeffrey O. Katz, "Developing Neural Network Forecasters for Trading", *Technical Analysis of Stocks and Commodities*, Vol. 10, No 4, p 58, April 1992

[99] M. Jurik, "The Care and Feeding of a Neural Network", *Futures,* Vol. XXI, No 12, 40-44, October 1992.

[100] Y. Yoon, and G. Swales. "Predicting Stock Price Performance: A Neural Network Approach", *Proceedings of the 24th Annual Hawaii International Conference on System Science*, Hawaii, IEEE Computer Society Press, Vol. 4, 1991, pp. 156-162

[101] K. Kimoto, and M. Yoda. "Stock Market Prediction System with Modular Neural Networks", *Proceedings of the International Joint Conference on Neural Networks*, San Diego, IEEE Network Council. Vol. 1, 1990, pp. 1-6.

[102] K. Kamijo, and T. Tanigawa. "Stock Price Patern Recognition: A Recurrent *Neural* Network Approach", *Proceedings of International Joint Conference on Neural Networks*, San Diego, IEEE Neural Network Council, Vol. 1, 1990, pp. 215-221

[103] W. Jiang, and J. Lee. *"Intelligent Trading of an Emerging Market"*, John Wiley & Sons 1994.

[104] Liya Wang, *"The Damped Newton Method—An ANN Learning Algorithm"*, Master's Thesis, Computer Science Department, Oklahoma State University, 1995

[105] Stephen T. Welstead, *"Neural Network and Fuzzy Logic Applications In C/C++"*, John Wiley & Sons, Inc.1994.

[106] M. Hestenes, *"Conjugate Direction Methods in Optimization"*, Springer-Verlag, 1980.

[107] Meishan Cheng, *" A Survey and Comparison of Conjugate Gradient Methods for Optimization"*, Master's Thesis, Computer Science Department, Oklahoma State University, 1993.

[108] D.S. Broomhead and D. Lowe, "Multivariable Functional Interpolation and Adaptive Networks", *Complex Systems* 2, 321-355.1988

[109] J.E. Moody and C.J. Darken, "Fast Learning in Networks of Locally-tuned Processing Units", *Neural Computation* 1, 281-29. 1989

[110] T. Poggio and F. Girosi, "Regularization Algorithms for Learning That Are Eqivalent to Multilayer Networks", Science 247, 978-982. 1990.

APPENDIX A Training File

```
0.888889 0.555556 0.777778 0.11025 0.444444 0.333333 0.444444 0
0.0869565
0.444444 0.222222 0.222222 -0.0803906 0.333333 0.333333 0.333333 0
0.0447431
0.444444 0.444444 0.333333 0.126176 0.555556 0.333333 0.555556 0
0.228448
0.666667 0.333333 0.555556 0.334313 0.444444 0.333333 0.444444 0
0.0626536
0.222222 0.222222 0.111111 -0.120054 0.222222 0.333333 0.222222 0
0.100917
0.333333 0.555556 0.555556 -0.102176 0.333333 0.333333 0.333333 0
0.143113
0.222222 0.333333 0.222222 0.0198263 0.555556 0.333333 0.555556 0
0.012766
0.222222 0.222222 0.333333 -0.0294607 0.555556 0.333333 0.555556 0
0.0145985
0.444444 0.555556 0.444444 0.246215 0.444444 0.111111 0.444444 1
0.210383
0.555556 0.555556 0.555556 0.179637 0.444444 0.333333 0.333333 0
0.120141
0.777778 0.555556 0.666667 0.0324084 0.333333 0.333333 0.444444 0
0.000385356
0.777778 0.444444 0.333333 0.103651 0.222222 0.333333 0.444444 1
0.0816641
0.555556 0.444444 0.444444 0.0420602 0.222222 0.111111 0.333333 1 -
0.0533333
0.666667 0.555556 0.444444 0.0214079 0.222222 0.333333 0.222222 1 -
0.0125523
0.222222 0.444444 0.222222 0.144002 0.444444 0.333333 0.666667 0
0
0.444444 0.444444 0.666667 0.00934334 0.333333 0.222222 0.555556 0
0.0714286
0.111111 0.111111 0.111111 -0.045472 0.555556 0.333333 0.444444 1
0.106383
0.333333 0.333333 0.333333 0.0630935 0.111111 0.333333 0.333333 1
0
0.666667 0.666667 0.555556 0.102554 0.333333 0.333333 0.333333 0
0.180534
0.333333 0.444444 0.444444 0.12489 0.222222 0.333333 0.333333 1
0.229682
0.444444 0.333333 0.333333 -0.25835 0.444444 0.333333 0.333333 1
0.315068
0.333333 0.333333 0.333333 -0.0592513 0.333333 0.333333 0.555556 0
0.024911
0.444444 0.333333 0.444444 -0.01066 0.333333 0.222222 0.555556 0
0.0488145
```

```
0.111111 0.444444 0.444444 -0.0332537 0.444444 0.333333 0.333333 1
0.027027
0.666667 0.444444 0.444444 0.0570308 0.444444 0.111111 0.444444 0
0.0896057
0.555556 0.555556 0.444444 0.0159926 0.444444 0.333333 0.444444 1
0.122449
0.333333 0.111111 0.333333 -0.0298175 0.555556 0.222222 0.444444 1
0.0612245
0.444444 0.444444 0.222222 0.0450185 0.333333 0.444444 0.333333 0
0.0666667
0.444444 0.333333 0.333333 -0.0355578 0.333333 0.444444 0.444444 0
0.156627
0.777778 0.333333 0.444444 0.0401636 0.222222 0.444444 0.333333 0
0
0.888889 0.444444 0.555556 0.119817 0.222222 0.444444 0.555556 0
0.0322581
0.777778 0.333333 0.333333 0.0270914 0.444444 0.333333 0.555556 0
0.0541176
0.666667 0.444444 0.222222 0.0979716 0.111111 0.444444 0.333333 0
0.117117
0.444444 0.444444 0.444444 0.147392 0.333333 0.444444 0.555556 1
0
0.333333 0.333333 0.444444 -0.00797985 0.444444 0.333333 0.555556 0
0.225806
0.333333 0.666667 0.333333 0.0797147 0.222222 0.444444 0.444444 0
0.00840336
0.555556 0.444444 0.555556 0.12045 0.222222 0.444444 0.333333 0
0.0867925
0.777778 0.777778 0.666667 0.121863 0.333333 0.222222 0.222222 0
0.00369004
0.777778 0.333333 0.333333 -0.00402843 0.444444 0.444444 0.555556 1
0.0793651
0.888889 0.555556 0.333333 0.202631 0.333333 0.333333 0.333333 1
0.0689655
0.888889 0.666667 0.444444 0.0817216 0.555556 0.444444 0.444444 1
0.0901099
0.666667 0.444444 0.333333 0.0366822 0.444444 0.444444 0.444444 1
0.0491803
0.444444 0.444444 0.333333 -0.287028 0.333333 0.444444 0.333333 1
0.37931
0.333333 0.555556 0.444444 -0.0203618 0.222222 0.444444 0.333333 1
0.00840336
0.444444 0.444444 0.222222 0.310005 0.444444 0.444444 0.444444 1
0.025641
0.444444 0.555556 0.444444 -0.0376695 0.222222 0.333333 0.444444 0
0.473684
0.333333 0.333333 0.222222 -0.0120284 0.333333 0.444444 0.444444 0
0.0607735
0.333333 0.444444 0.222222 -0.0120284 0.555556 0.444444 0.555556 0
0.0562347
0.444444 0.555556 0.555556 0.131455 0.444444 0.333333 0.555556 0
0.271967
0.666667 0.888889 0.777778 0.218644 0.444444 0.333333 0.333333 0 -
0.115385
0.333333 0.555556 0.333333 0.0248231 0.222222 0.333333 0.222222 0
0.055409
0.444444 0.333333 0.333333 -0.039905 0 0.333333 0.222222 0
0.165049
0.555556 0.555556 0.333333 0.0390815 0.444444 0.333333 0.444444 0
0.0357143
```

```
0.444444 0.333333 0.444444 0.0694087 0.222222 0.222222 0.333333 0
0.0218805
0.333333 0.222222 0.222222 -0.0105438 0.444444 0.333333 0.333333 0
0.167883
0.444444 0.555556 0.555556 0.0383919 0.333333 0.333333 0.666667 0
0.00330033
0.777778 0.555556 0.555556 0.216119 0.222222 0.222222 0.444444 0
0.0352941
0.333333 0.555556 0.444444 -0.0118752 0.444444 0.222222 0.444444 0
0.0786517
0.555556 0.555556 0.333333 0.0460173 0.555556 0.222222 0.444444 0
0.0134357
0.444444 0.444444 0.333333 -0.0893093 0.333333 0.222222 0.444444 0
0.0607735
0.222222 0.333333 0.333333 -0.182944 0.333333 0.222222 0.444444 0
0.0827068
0.444444 0.333333 0.444444 -0.0164103 0.555556 0.333333 0.444444 0
0.00900901
0 0.444444 0.444444 -0.209899 0.333333 0.333333 0.555556 0
 0
0.444444 0.333333 0.222222 -0.0194224 0.555556 0.333333 0.666667 0
 0
0.444444 0.444444 0.444444 -0.0194224 0.444444 0.333333 0.444444 0
0.125
0.444444 0.333333 0.333333 -0.0280184 0.222222 0.333333 0.333333 0
0.0635838
0.666667 0.555556 0.333333 0.0222856 0.222222 0.333333 0.333333 0
0.0524309
0.444444 0.555556 0.555556 0.0745947 0.333333 0.111111 0.444444 0
0.157552
0.666667 0.333333 0.333333 0.149527 0.444444 0.333333 0.444444 0
0
0.444444 0.333333 0.333333 0.0093451 0.444444 0.111111 0.333333 0
0.0588235
0.333333 0.444444 0.444444 -0.0873995 0.555556 0.111111 0.444444 0
0.0126582
0.222222 0.222222 0.333333 -0.0669091 0.444444 0.111111 0.555556 0
0.0332717
0.333333 0.333333 0.333333 -0.040597 0.333333 0.222222 0.333333 1
0.0444243
0.444444 0.555556 0.333333 -0.0581889 0.444444 0.111111 0.555556 0
0.023569
0.666667 0.444444 0.444444 0.0069882 0.333333 0.111111 0.333333 0
0.00502513
0.222222 0.222222 0.111111 -0.0430358 0.333333 0.222222 0.444444 1
0.152355
0.222222 0.444444 0.333333 -0.042755 0.444444 0.222222 0.555556 1
0.00584795
0.555556 0.444444 0.222222 0.0019377 0.333333 0.111111 0.333333 0
0.0531646
0.666667 0.666667 0.222222 0.0629457 0.444444 0.111111 0.333333 0
0
0.444444 0.444444 0.555556 0.000482092 0.555556 0.111111 0.666667 0
0.00291545
0.666667 0.555556 0.555556 0.108709 0.444444 0.222222 0.444444 1
0.129412
0.222222 0.555556 0.222222 0.0269264 0.333333 0.333333 0.333333 0
 -0.00230947
0.555556 0.555556 0.444444 0.03731 0.555556 0.333333 0.555556 0
0.0031348
```

```
0.666667 0.666667 0.555556 0.0982793 0.666667 0.333333 0.444444 0
0.00437956
0.888889 0.777778 0.777778 0.169725 0.333333 0.222222 0.444444 0
0
0.666667 0.555556 0.333333 -0.0117934 0.444444 0.333333 0.555556 0
0.0153483
0.555556 0.222222 0.333333 0.00713583 0.555556 0.222222 0.666667 0
0.0750988
0.555556 0.666667 0.444444 0.0992384 0.444444 0.333333 0.555556 0
 0
0.444444 0.555556 0.444444 -0.066817 0.444444 0.333333 0.333333 0
0.0628931
0.222222 0.444444 0.333333 -0.0148621 0.333333 0.333333 0.444444 0
0.0684932
0.444444 0.555556 0.444444 0.0103997 0.444444 0.222222 0.222222 0
0.0104325
0.222222 0.555556 0.333333 -0.018949 0.222222 0.333333 0.444444 0
0.0455446
0.222222 0.555556 0.333333 0.0505856 0.111111 0.333333 0.333333 0
0.103797
0.222222 0.222222 0.222222 -0.0789444 0.333333 0.333333 0.444444 0
 -0.00911854
0.666667 0.555556 0.444444 0.0586252 0.444444 0.333333 0.333333 0
0
0.333333 0.444444 0.444444 -0.0191384 0.444444 0.333333 0.444444 0
0.0457516
0.666667 0.555556 0.444444 0.150738 0.333333 0.333333 0.666667 0
0.00266667
0.666667 0.555556 0.666667 0.127101 0.444444 0.222222 0.555556 0
0.0146082
0.555556 0.555556 0.555556 0.13124 0.222222 0.333333 0.444444 0
0.0416667
0.111111 0.222222 0.333333 -0.0220671 0.333333 0.333333 0.333333 0
-0.00376648
```

APPENDIX B Test File

```
--------------------------------------------------------------------------------------------------------

0.444444 0.222222 0.222222 0.199083 0.555556 0.444444 0.666667 1
0.0275229
0.666667 0.333333 0.444444 0.0242453 0.333333 0.111111 0.333333 1
0.043755
0.666667 0.555556 0.444444 0.111211 0.222222 0.444444 0.444444 1
0.0031348
0.333333 0.555556 0.555556 -0.0120284 0.444444 0.444444 0.444444 1
0.297297
0.888889 0.555556 0.666667 0.021788 0.333333 0.444444 0.444444 0
0.0317757
0.111111 0.222222 0.333333 -0.0434186 0.333333 0.444444 0.444444 1
0.0925926
0.555556 0.333333 0.444444 0.123203 0.222222 0.444444 0.555556 0
0.0532915
0.555556 0.333333 0.444444 -0.0134872 0.333333 0.444444 0.222222 1
0.0401753
0.333333 0.444444 0.555556 -0.010056 0.111111 0.444444 0.333333 0
0.0708661
0.333333 0.555556 0.333333 -0.0192748 0.222222 0.444444 0.333333 1
0.0510949
0.444444 0.222222 0.222222 -0.0120284 0.222222 0.444444 0.444444 0
0.0536585
0.444444 0.444444 0.444444 -0.0246181 0.333333 0.333333 0.333333 0
0.0334825
0.555556 0.444444 0.444444 0.0520486 0.555556 0.333333 0.555556 0
0.167183
0.555556 0.555556 0.555556 -0.0215317 0.444444 0.333333 0.444444 0
0.0564173
0.111111 0.333333 0.333333 -0.08085 0.444444 0.222222 0.555556 0
0.170147
0.666667 0.555556 0.444444 0.0253106 0.555556 0.333333 0.555556 0
0.0217391
0.444444 0.444444 0.444444 0.0161982 0.444444 0.333333 0.444444 0
0.346405
0.666667 0.444444 0.444444 0.040317 0.444444 0.333333 0.666667 0
0.142875
0.444444 0.444444 0.555556 0.0657433 0.333333 0.333333 0.444444 0
0.0110497
0.444444 0.333333 0.444444 0.052823 0.555556 0.333333 0.555556 0
0.1375
0.555556 0.444444 0.444444 0.00251368 0.333333 0.333333 0.333333 0
0.106918
0.222222 0.222222 0.333333 -0.0534004 0.222222 0.333333 0.444444 0
-0.0273556
```

APPENDIX C   Code for Preprocessor

```
// pr.h   Head file for preprocessor
// Feb, 1999 By Yimin Yang. Computer Science Department
// Oklahoma State University

#define M_DAY 5
#define DAY_NUM 17
#define MAX 9

typedef struct raw
{
public:
      long date;
      char *name;
      double closing_price;
      long volume;
        double target;
}rawnode;

inline rawnode* buildnode ( char *name1, double tar, long dat,long vo,
double pri){
            rawnode *node;
            node=new rawnode();
            node->date=dat;
            node->name=new char[5];
            strcpy(node->name,name1);
            node->closing_price=pri;
            node->volume=vo;
                node->target=tar;
            return node;
      }




rawnode* array[20];
rawnode* index[20];


class result
{
private:
        double pa,pia;
        long   va,iva;
      double pan, pdp, prn,prg;
      double van, vdv, vrn, vrg;

      FILE* output;
      //comput com;
```

```
protected:
      double targ;


//    double target;

public:
      result(FILE*out=0, double tar=0):pan(0),pdp(0),prn(0),prg(0),
            van(0),vdv(0),vrn(0),vrg(0){ output=out; targ=tar;}
//    result(FILE*, double);
      void print();
        void average();
};//end of class result

class comput: public result
{
private:
      double num1,num2;
      double pm(int);
      long vm(int);
public:
      comput(){}

      double pan();
      double pdp();
      double prn();
      double prg();
      double van();
      double vdv();
      double vrn();
      double vrg();
      double target();
};//end of class compute


/**********************************************************
 *     Preprocessor for Neural Networks                    *
 *   Feb, 1999. By Yimin Yang. Computer Science Department  *
 *   Oklahoma State University                             *
 **********************************************************/
// This program read raw data from file "rawdata" and produce
//inputs to the neural networks, the results saved in the file
//training. The raw data has the following form:
//line 1+17*x x=1,2,… 1800 has two fields one is stock name
//       the second is the highest price in two weeks from the
//       last day in the rawdata
//other lines have five fields each line.
//      first field is the trading date
//      second is the trading volume of the whole market
//      third is the closing value of the market index(Dow Jones
      Industies average
//      fourth is the trading volume of the stocks
//     fifth is the closing price of the stock that day
..........................................................................



#include<iomanip.h>
#include<fstream.h>
```

```
#include<stdio.h>
#include<iostream.h>
#include<string.h>
#include<math.h>
#include<stdlib.h>

#include "pr.h"


void result::print()
{
      comput com;
      pan=com.pan();
      pdp=com.pdp();
      prn=com.prn();
      prg=com.prg();
      van=com.van();
      vdv=com.vdv();
      vrn=com.vrn();
      vrg=com.vrg();
      targ=com.target();
        cout<<"\n act target is : "<<array[0]->target;
      //cout<<pan;
       // cout<<pdp;
       // cout<<prn<<prn<<prg;
       // cout<<endl;
      //
cout<<van<<setw(3)<<vdv<<setw(3)<<vrn<<setw(3)<<vrg<<targ<<"\n";
        double l=0;
        l=pan+pdp+prn+prg+van+vdv+vrn+vrg;
        l=l/8;
        l=l-0.2;
        cout<<"\n reasonable output should be:  "<<l;
        double tar=0;
        double s=0;
        s=array[DAY_NUM-1]->closing_price;
        tar=l*s+s;
        cout<<"\n reasonable target should be : "<<tar;


        ofstream outp("training",ios::app);
        if(outp.fail())
        cerr<<"OUTPUT file open error";
outp<<pan<<" "<<pdp<<" "<<prn<<" "<<prg<<" "<<van<<" "<<vdv<<" "<<vrn<<"
"<<vrg<<" "<<targ<<"\n";

}//end of print()

//comput::comput()
//{
//}//end of comput

//number of days in which the price is greater than its moving average

double comput::pan()
{
      int i=0;
      int num=0;
      double num1=0.0000,num2=0.0000;

      for(i=DAY_NUM-1;i>DAY_NUM-MAX;i--)
```

```
        {
                if(array[i]->closing_price>pm(i))
                        num++;
        }
        num1=num;
        num2=MAX;
        return num1/num2;

//      return num/MAX;
}//end of compute pan

//compute simple moving average
double comput::pm(int x)
{
    double total=0.0000;
    int i=0;

    for(i=x;i>x-M_DAY;i--)
    {
            total+=array[i]->closing_price;
    }
    return total/M_DAY;
}

//function p%>dp% compute number of days that the price increase
percentage is
//greater than that of index
double comput::pdp()
{
        int i=0;
        int num=0;
        double pc1,pc2;

        for(i=DAY_NUM-1;i>DAY_NUM-MAX;i--)
        {
                pc1=array[i]->closing_price-array[i-1]->closing_price;
                pc2=index[i]->closing_price-index[i-1]->closing_price;
                if(pc1/array[i-1]->closing_price>pc2/index[i-1]-
>closing_price)
                        num++;
        }
        num1=num;
        num2=MAX;
        return num1/num2;
//      return num/MAX;
}//need to pass the index array


//ber of days in which the price of current is greater than that of last
day
double comput:: prn()
{
    int i=0;
    int num=0;

    for(i=DAY_NUM-1;i>DAY_NUM-MAX;i--)
    {
            if(array[i]->closing_price>array[i-1]->closing_price)
                    num++;
    }
    num1=num;
    num2=MAX;
    return num1/num2;
  // return num/MAX;
}

//computer if the price increase of MAX days is greater than that of the
index

double comput::prg()
{
        int m=DAY_NUM-1;
        int n=DAY_NUM-MAX-1;
      double num1, num2;

      num1=(array[m]->closing_price-
    array[n]->closing_price)/array[n]->closing_price;
      num2=(index[m]->closing_price-
   index[n]->closing_price)/index[n]->closing_price;
      return (num1-num2);
}

//compute number of days in which the volume is greater than its moving
average
double comput::van()
{
      int i=0;
      int num=0;

      for(i=DAY_NUM-1;i>DAY_NUM-MAX;i--)
      {
            if(array[i]->volume>vm(i))
                  num++;
      }
      num1=num;
      num2=MAX;
      return num1/num2;
      //return num/MAX;
}

//compute volume moving average
long comput::vm(int x)
{
      int i=0;
      long total=0;

      for(i=x;i>x-M_DAY;i--)
      {
            total+=array[i]->volume;
      }
      return total/M_DAY;
}

//compute number of days in which the vovule increase percentage is
graeter than
double comput::vdv()
{
      int i=0;
      int num=0;
      double num3=0;
      double num4=0;
```

```
	for(i=DAY_NUM-1;i>DAY_NUM-MAX;i--)
	{
		num3=(array[i]->volume-array[i-1]->volume)/array[i-1]-
>volume;
		num4=(index[i]->volume-index[i-1]->volume)/array[i-1]-
>volume;
	  if(num3>num4)
			num++;
	}
	num1=num;
	num2=MAX;
	return num1/num2;
	//return num/MAX;
}

//compute number of days in which the volume of current day is graeter
than last
double comput::vrn()
{
	int i=0;
	int num=0;

	for(i=DAY_NUM-1;i>DAY_NUM-MAX;i--)
	{
		if(array[i]->volume>array[i-1]->volume)
			num++;
	}
	num1=num;
	num2=MAX;
	return num1/num2;
//	return num/MAX;
}

//compute if the whole volume is greater than that of index
double comput::vrg()
{
	double num3;
	double num4;
	long total1;
	long total2;
	int i=0;

	for(i=DAY_NUM-1;i>DAY_NUM-MAX;i--)
	{
		total1+=array[i]->volume;
		total2+= index[i]->volume;
	}

	  double m,n;
	  m=total1/MAX;
	  n=total2/MAX;
	num1=(m-array[0]->volume)/array[0]->volume;
	num2=(n-index[0]->volume)/index[0]->volume;
	  if(num1>0&&num1>num2)
	   return 1;
	   else
	   return 0;

}

//scalint target output
```

```
double comput::target()
{
      double cp;
      cp=array[DAY_NUM-1]->closing_price;
      return (array[0]->target-cp)/cp;

}

int main()
{
      char *line;

      int i=0, index1;
      long da,vo,ido;
      double cp,target,icp;
      char *name;
      char *name1;

      line=new char[80];
      name=new char[30];
      name1=new char[30];

        ifstream inp("rawdata");
        ofstream outp("training");

        if(inp.fail())
        {
         cerr<<"Error opening rawdata"<<endl;
         exit(1);
        }
 /*       if(outp.fail())
        {
         cerr<<"Error opening training"<<endl;
         exit(2);
        }  */

        while((!inp.eof()))
      {
        inp>>name;
          inp>>target;
          index1=0;
          for(i=0;i<DAY_NUM;i++)
          {
            inp>>da>>ido>>icp>>vo>>cp;
          //  cout<<da<<ido<<icp<<vo<<cp;
            array[index1]=buildnode(name,target,da,vo,cp);
            index[index1]=buildnode(name,target,da,ido,icp);
            index1++;
          }
          result re;
          re.print();
        }//end of while loop

}//end of main
```

APPENDIX D   Code for FRBF Simulator

```
/*********************************************
    Fixed Center Radial Basis Function Simulator

     Feb, 1999. By Yimin Yang. Computer Science Department
     Oklahoma State University
..................................................................
  This program is written using the method described in
Haykin' book "Neural Networks, A Comprehensive Foundation"
Simon Haykin, Macmillan College Publishing Company 1994
Before training the program, the file "training" must be set
Before testing the simulator, the file "testfile" must be
set. The program is written in C++, can run in both Unix and
PC Visual C++ envirament
*********************************************************/


#include <stdio.h>
#include <iostream.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include<fstream.h>

double tor;
 ofstream out("output");
class network;
class input_layer
{
public:

      int num_inputs;
//    double input_vector[10];


    input_layer(int);
      //initial_centers();
      ~input_layer();
      void put_input();
      friend network;
};

input_layer::input_layer( int i)
{
  num_inputs=i;
  //input_vector= new double[i];

}
input_layer::~input_layer()
```

```
{
  //delete [num_inputs] input_vector;
}

class middle_layer
{
   public:
         int num_centers;
      // double center_vector[10][10];
         input_layer *in;

      middle_layer();
      middle_layer(int,input_layer *);
      ~middle_layer();
      double calc_gaussian(int );
      double calc_distance(int);
      double max_distance();
      void initial_centers();
       void read_centers();
      void write_centers();
      double calc_two_points(int, int);
        void update_centers(float);
        int shortest_center();
      friend network;
};


middle_layer::middle_layer(int i, input_layer *m)
{
  num_centers =i;
  in=m;

//  center_vector=new double[i][num_inputs];

 // center_vector=new double[i]
}

middle_layer::~middle_layer(){
  // delete in;
}


class output_layer
{
private:
      double weights[100];
      double expected_value;
      input_layer *in;
      middle_layer *mid;

           friend network;

public:


      output_layer(input_layer*, middle_layer*);
      ~output_layer();
      void initial_weights();
      double calc_out();
      double  calc_error();
      void update_weights(float);
      //void list_weights();
      void write_weights();
      void read_weights();
      //void initial_centers();
      void write_output();
      void initial_centers();
      //void list_errors();
      //void list_outputs();
};
output_layer::output_layer(input_layer* i, middle_layer* j)
{
  in=i;
  mid=j;

}
output_layer::~output_layer()
{
  //delete [num_centers] weights;
}



class network

{

private:

      input_layer *inl;
      middle_layer *ml;
      output_layer      *ol;
      int numin, numcen,numout, training;

public:
    network();
    ~network();
      void set_training(const unsigned &);
      unsigned get_training_value();
      void get_layer_info();
        void training_centers();
      double get_error();


      void set_up_network();
      void set_up_patterns( int);
      void randomize_weights();
      void update_weights( float);
      //void update_momentum();
      void write_weights();
      void read_weights();
      //void list_weights();
      void write_outputs();
      void update_weights1(float);


};
network::network(){
      numin=0;
      numcen=0;
```

```
        numout=0;
        training=0;
}

 double train[150][50];
 int row;
 double center_vector[100][50];
 double input_vector[10];


inline float squash(float input)
// squashing function
// use sigmoid -- can customize to something
// else if desired; can add a bias term too
//
{
if (input < -50)
      return 0.0;
else  if (input > 50)
            return 1.0;
      else return (float)(1/(1+exp(-(double)input)));

}


inline double randomweight()
{
int num;
// random number generator
// will return a floating point
// value between -1 and 1

//if (init==1)     // seed the generator
      srand ((unsigned)time(NULL));

num=rand() % 100;

return 2*(double(num/100.00))-1;

}

void output_layer::initial_centers()
{
  int num=0;
  //int flag=0;
 // double unit;
  //ifstream select("training");
  //if(select.fail())
  //cerr<<"Input file open error";
  //else
   //{
      for(int i=0; i<mid->num_centers;i++)
       {
         num=0;
       //  flag=0;
//         srand((unsigned) time(NULL));
  //        num=rand()%100;
              for(int m=0;m<in->num_inputs;m++)
                    center_vector[i][m]=train[i][m];
        //  select>>unit;
         // flag++;
```

```
          // while(flag/9!=num)
           // {
            //   select>>unit;
             //  flag++;
             //}
             //if(flag/9==num)
              //{
             //           center_vector[i][0]=unit;
          //      for(int m=1;m<in->num_inputs;m++)
           //      select>>center_vector[i][m];
              //}
        }//end of i loop
     //end of else
   // select.close();
}

void middle_layer::update_centers( float beta)
{
  int i,x;
   i=shortest_center();
    for(x=0;x<in->num_inputs;x++)
     {
       center_vector[i][x]+=beta*(input_vector[x]-center_vector[i][x]);
     }
}

int middle_layer::shortest_center()
{
  int i, result, m;
  double x, y, shortest=1000.0000;

   for(i=0;i<num_centers;i++)
   {
     x=0.0000;
     y=0.0000;
     for(m=0;m<in->num_inputs;m++)
      {
        double unit=input_vector[m]-center_vector[i][m];
        x= x + unit*unit;
       }
       y=sqrt(x);
       if(shortest>y)
        {
          shortest=y;
          result=i;
        }
    }
       return result;
}

double middle_layer::calc_gaussian(int i)
 {
   double x, d, d1, d2, m,n;

    d2=calc_distance(i);      //compute ||x-ti||
    d2=d2*d2;
    d=max_distance();
   d1=d*d;
//     x=1/(1+d2);
      m=sqrt(2*num_centers);
      n=d/m;
   // x=exp(-num_centers/d1*d2);
      x=exp(-d2/d1);
    return x;
}
//this function compute ||x-ti||^2
double middle_layer::calc_distance(int i)
{
  int m;
   double x=0.0000;
      double y=0.0000;
   for(m=0;m<in->num_inputs;m++)
   {
     y=(input_vector[m]-center_vector[i][m]);
       x+=y*y;
   }
   return sqrt(x);
}

//compute max distance of chosen centers
double middle_layer::max_distance()
{
  double max_dis;
  double x=0.0000, y=0.0000;
  int i,j;

   for(i=0;i<num_centers;i++)
    for(j=1;j<num_centers;j++)
     {
       x=calc_two_points(i,j);
       if(max_dis<x)
        max_dis=x;
     }
     return max_dis;
}

//compute Euclidean distance between the tow centers
double middle_layer::calc_two_points(int i, int j)
{
      double x=0.0000,y=0.0000,z=0.0000;
      for(int m=0;m<in->num_inputs;m++)
      {
            x=(center_vector[i][m]-center_vector[j][m]);
                  y=x*x;
            z+=y;

      }
      return sqrt(z);
}

//read the centers from file center

void middle_layer::read_centers()
{
  ifstream readcenter("center");
   if(readcenter.fail())
   cerr<<"center file open error";
   else
    {
      for(int i=0;i<num_centers;i++)
        for(int j=0;j<in->num_inputs;j++)
         {
            readcenter>>center_vector[i][j];
          }
      }
       readcenter.close();
 }

 void middle_layer::write_centers()
{
       ofstream writecenter("center");
  int m=0;
   if(writecenter.fail())
    cerr<<"write center file open error";
    else
     {
       for(int i=0;i<num_centers; i++)
         {
         for(int j=0;j<in->num_inputs;j++)
              {
          writecenter<<center_vector[i][j]<<endl;
             //  m++;
               //if((m%8)==0)
                   //  cout<<"\n";
               }
              cout<<endl;
          }

      }
      writecenter.close();
}


void output_layer::initial_weights()
{
   //weights=new double[num_centers];
   for(int i=0;i<mid->num_centers;i++)
   weights[i]=randomweight();
}

void output_layer::update_weights( float beta)
 {
   double actual_out, error;
   actual_out=calc_out();
   error=expected_value-actual_out;
   for(int i=0;i<mid->num_centers;i++)
    {
      weights[i]+=beta*error*mid->calc_gaussian(i);//may use sigmoid
    }
 }

 void output_layer::read_weights()
{
   ifstream readweights("weight");
    if(readweights.fail())
     cerr<<"readweights file open error";
     else
      {
        for(int i=0;i<mid->num_centers;i++)
          readweights>>weights[i];
      }
      readweights.close();
  }

  void output_layer::write_weights()
{
         int m=0;
   ofstream writeweights("weight");
    if(writeweights.fail())
     cerr<<"writeweights file open error";
      else
       {
          for(int i=0;i<mid->num_centers;i++)

           writeweights<<weights[i]<<endl;



        }
    writeweights.close();
 }

double output_layer::calc_out()
{
  int i;
  double total=0.000000, unit, gau, result;

  for(i=0;i<mid->num_centers;i++)
   {

     gau=mid->calc_gaussian(i);
     unit=weights[i]*gau;
     total+=unit;
   }
   //result=(double)(1/(1+exp(-(double)total)));
  //return result ;
   return total;
}

void output_layer::write_output()
{
}


network::~network()
{
  delete inl;
   delete ol;
  delete ml;
}

void network::set_training(const unsigned & value)
{
training=value;
}

unsigned network::get_training_value()
{
return training;
}


void network::get_layer_info()
```

```
{


cout << " Enter in the layer sizes separated by spaces.\n";
cout << " For a network with 3 neurons in the input layer,\n";
cout << " 2 neurons in a hidden layer, and 4 neurons in the\n";
cout << " output layer, you would enter: 3 2 4 .\n";
//cout << " You can have up to 3 hidden layers,for five maximum entries
:\n\n";

      cin >> numin>>numcen>>numout;


// -------------------------------------------------------
// size of layers:
//         input_layer               layer_size[0]
//         output_layer              layer_size[number_of_layers-1]
//         middle_layers             layer_size[1]
//                             optional: layer_size[number_of_layers-3]
//                             optional: layer_size[number_of_layers-2]
//-------------------------------------------------------


}

void network::set_up_network()
{
//int i,j,k;



inl=new input_layer(numin);
ml=new middle_layer(numcen,inl);
ol=new output_layer(inl,ml);


}


//training the center before training the weights using k-mean method
void network::training_centers()
{
    ol->initial_centers();

   // ifstream inpu("training");
    //if(inpu.fail())
   // cerr<<"inpu open error";
    double cluster[100][100];
      for(int p=0;p<10;p++)
            for(int pp=0;pp<10;pp++)
                  cluster[p][pp]=0;
   // double iv[10];
   // int index=0;
    int m=0;
      int index [10][1];
      for(int k=0;k<10;k++)
            index[k][0]=0;
      double md=0;
    //while(!inpu.eof())
      int u;
      for(u=0;u<row;u++)
```

```
     {
        for(int i=0;i<inl->num_inputs;i++)
         input_vector[i]=train[u][i];
         m=ml->shortest_center();
         for(int l=0;l<inl->num_inputs;l++)
              {


                 cluster[m][l]
                     =cluster[m][l]+input_vector[l];


              }
              index[m][0]++;

     }
       for(int r=0;r<ml->num_centers;r++)
       {
               for(int n=0;n<inl->num_inputs;n++)
               {
                     center_vector[r][n]=cluster[r][n]/index[r][0];
               }
       }



     }
void network::randomize_weights()
{
       //ol->initial_centers();
      ol->initial_weights();
}


void network::update_weights( float beta)
{
      ml->update_centers(beta);
      ol->update_weights(beta);
}

void network::update_weights1(float beta)
{
      double ae=0;
      ae=tor/100;
      for(int i=0;i<ml->num_centers;i++)
            ol->weights[i]+=beta*ae;
}



void network::write_weights()
{
  ml->write_centers();
  ol->write_weights();
}


void network::read_weights()
{
        ml->read_centers();
        ol->read_weights();
}


void network::write_outputs()
{
                out<<"\n target is: "<<input_vector[8];
                out<<"\n actual output from neural network is: ";

            out<<ol->calc_out()<<endl;
}


double network::get_error()
{
        double x,y;
        x=ol->calc_out();
        y=ol->expected_value-x;
//      tor=tor+y*y;
        if(y<0)
              y=0-y;
        return y;
}



void network::set_up_patterns(int i)
{


        if(i==1)
              ol->expected_value=input_vector[8];
}





void main()
{

double error_tolerance=0.1;
double total_error=0.0;
double avg_error_per_cycle=0.0;
double error_last_cycle=0.0;
double avgerr_per_pattern=0.0; // for the latest cycle
double error_last_pattern=0.0;
float learning_parameter=0.02;
//float alpha; // momentum parameter

//float NF; // noise factor
//float new_NF;
tor=0;


unsigned temp, startup, start_weights;
//long int vectors_in_buffer;
```

```
long int max_cycles;
long int patterns_per_cycle=0;

long int total_cycles, total_patterns;
int i=0;


// create a network object
network rbf;
ifstream input("training");
ifstream test("testfile");


// enter the training mode : 1=training on     0=training off
cout << "--------------------------------------------------\n";
cout << " C++ Neural Networks \n";
cout << "    RBF simulator \n";
cout << "          version 1 \n";
cout << "--------------------------------------------------\n";
cout << "Please enter 1 for TRAINING on, or 0 for off: \n\n";
cout << "Use training to change weights according to your\n";
cout << "expected outputs. Your training.dat file should contain\n";
cout << "a set of inputs and expected outputs. The number of\n";
cout << "inputs determines the size of the first (input) layer\n";
cout << "while the number of outputs determines the size of the\n";
cout << "last (output) layer :\n\n";

cin >> temp;

row=0;
rbf.set_training(temp);

if (rbf.get_training_value() == 1)
      {

      cout << "--> Training mode is *ON*. weights will be saved\n";
      cout << "in the file weights.dat at the end of the\n";
      cout << "current set of input (training) data\n";
       while(!input.eof())
       {
             for(int q=0;q<9;q++)
             {
                   input>>train[row][q];
             }
             row++;
       }

      }
else
      {
      cout << "--> Training mode is *OFF*. weights will be loaded\n";
      cout << "from the file weights.dat and the current\n";
      cout << "(test) data set will be used. For the test\n";
      cout << "data set, the test.dat file should contain\n";
      cout << "only inputs, and no expected outputs.\n";
      while(!test.eof())
      {
            for(int q=0;q<9;q++)
            {
                  test>>train[row][q];
            }
```

```
                row++;
        }
        }

if (rbf.get_training_value()==1)
        {
        // ----------------------------------------
        //     Read in values for the error_tolerance,
        //     and the learning_parameter
        // ----------------------------------------
        cout << " Please enter in the error_tolerance\n";
        cout << " --- between 0.001 to 100.0, try 0.1 to start --\n";
        cout << "\n";
        cout << "and the learning_parameter, beta\n";
        cout << " --- between 0.01 to 1.0, try 0.5 to start -- \n\n";
        cout << " separate entries by a space\n";
        cout << " example: 0.1 0.5 sets defaults mentioned :\n\n";

        cin >> error_tolerance >> learning_parameter;

        cout << "Please enter the maximum cycles for the simulation\n";
        cout << "A cycle is one pass through the data set.\n";
        cout << "Try a value of 10 to start with\n";

        cin >> max_cycles;

        cout << "Do you want to read weights from weights.dat to
start?\n";
        cout << "Type 1 to read from file, 0 to randomize starting
weights\n";

        cin >> start_weights;

        }



// training: continue looping until the total error is less than
//          the tolerance specified, or the maximum number of
//          cycles is exceeded; use both the forward signal propagation
//          and the backward error propagation phases. If the error
//          tolerance criteria is satisfied, save the weights in a file.
// no training: just proceed through the input data set once in the
//          forward signal propagation phase only. Read the starting
//          weights from a file.
// in both cases report the outputs on the screen


// intialize counters
total_cycles=0; // a cycle is once through all the input data
total_patterns=0; // a pattern is one entry in the input data
//new_NF=NF;


// get layer information
rbf.get_layer_info();

// set up the network connections
rbf.set_up_network();
```

```
// initialize the weights
if ((rbf.get_training_value()==1)&&(start_weights!=1))
      {
      // randomize weights for all layers; there is no
      // weight matrix associated with the input layer
      // weight file will be written after processing

      rbf.randomize_weights();
      // set up the noise factor value
      //backp.set_NF(new_NF);
      }
else
      {
            rbf.read_weights();

      }



// main loop
// if training is on, keep going through the input data
//          until the error is acceptable or the maximum number of
cycles
//          is exceeded.
// if training is off, go through the input data once. report outputs
// with inputs to file output.dat

startup=1;
total_error = 0;
double temp1[100];
int total_pats=0;

if(rbf.get_training_value()==1)
{



      rbf.training_centers();


         //ifstream input("training");
         //if(input.fail())
         //cerr<<"input file open error";
       i=1;
       avgerr_per_pattern=100.0000;

       total_cycles=0;
       double unit_error=0.000000;
       double total_error=0.000000;


while((avgerr_per_pattern>error_tolerance)&&(total_cycles<max_cycles))
       {
             patterns_per_cycle=0;
                 total_error=0.0000;
                 total_pats=0;
             //while(!input.eof())
                         for(int r=0;r<row;r++)
             {
                    for(int l=0;l<9;l++)
                          input_vector[l]=train[r][l];


              rbf.set_up_patterns(i);

              total_patterns++;
              total_pats++;
              unit_error=rbf.get_error();
              total_error+=unit_error;

              rbf.update_weights(learning_parameter);
              }
              total_cycles++;
              rbf.update_weights1(learning_parameter);
              double pre=avgerr_per_pattern;
              avgerr_per_pattern=total_error/total_pats;
             // avgerr_per_pattern-=0.1;
                  cout<<total_cycles<<"    new error is:
"<<avgerr_per_pattern<<endl;
              if(pre<avgerr_per_pattern)
              {
              cout<<"Weights are blowing up, try a small learning rate";
                    exit(1);
              }
              //input.close();
               //input.open("training");
             // if(input.fail())
       //           cerr<<"input file open error";

        }
        //input.close();
        rbf.write_weights();
        rbf.write_outputs();

cout << "         weights saved in file weights.dat\n";
cout << "\n";
cout << "---->average error per cycle = " << avg_error_per_cycle << " <-
--\n";
cout << "---->error last cycle = " << error_last_cycle << " <---\n";
cout << "->error last cycle per pattern= " << avgerr_per_pattern << " <-
--\n";



cout << "------------>total cycles = " << total_cycles << " <---\n";
cout << "------------>total patterns = " << total_patterns << " <---\n";
cout << "-----------------------------------------------\n";
}//end of if training

if(rbf.get_training_value()==0)
{
      int i=1;
       // ifstream input("training");
      for(int r=0;r<row-1;r++)
      {
            for(int m=0;m<9;m++)
             input_vector[m]=train[r][m];

                out<<"\n for pattern:  ";
                for(int j=0;j<8;j++)
```

```
                    out<<input_vector[j]<<" ";
                rbf.set_up_patterns(i);
                rbf.write_outputs();
                    total_error+=rbf.get_error();
                    total_pats++;
          }
            out<<"\n the average error for testfile is : ";
            out<<total_error/total_pats;
            out<<"\n Total patterns is :  "<<total_pats;
          //input.close();
          cout<<"\n end of test, you may see result in output file";
    }

           out.close();

    }
```

Appendix E

(Code for Adaptive Center Radial Basis Function Simulator)

```
/*********************************************************
  Adaptive Center Radial Basis Function Simulator
  Feb, 1999. By Yimin Yang, Computer Science Department
  Oklahoma State University
..........................................................
   This program is written using the method described in
Haykin's book "Neural Networks, A Comprehensive Foundation"
Simon Haykin, Macmillan College Publishing Company. 1994
Before training the program, the file "training" must be set
Before test the simulator, the file "testfile" must be set
*********************************************************/



#include <stdio.h>
#include <iostream.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include<fstream.h>

class network;
ofstream results("result");

class input_layer
{
public:

      int num_inputs;
//    double input_vector[10];


    input_layer(int);
      //initial_centers();
      ~input_layer();
      void put_input();
      friend network;
};
```

```
input_layer::input_layer( int i)
{
  num_inputs=i;
  //input_vector= new double[i];

}
input_layer::~input_layer()
{
  //delete [num_inputs] input_vector;
}

class middle_layer
{
   public:
         int num_centers;
      // double center_vector[10][10];
         input_layer *in;

      middle_layer();
      middle_layer(int,input_layer *);
      ~middle_layer();
      double calc_gaussian(int );
      double calc_distance(int);
      double max_distance();
      void initial_centers();
       void read_centers();
      void write_centers();
      double calc_two_points(int, int);
        void update_centers(float);
        int shortest_center();
      friend network;
};


middle_layer::middle_layer(int i, input_layer *m)
{
  num_centers =i;
  in=m;

//  center_vector=new double[i][num_inputs];

 // center_vector=new double[i]
}

middle_layer::~middle_layer(){
  // delete in;
}


class output_layer
{
private:
      double weights[100];
      double expected_value;
      input_layer *in;
      middle_layer *mid;

          friend network;

public:
      output_layer(input_layer*, middle_layer*);
      ~output_layer();
      void initial_weights();
      double calc_out();
      double  calc_error();

      void update_weights(float);
      //void list_weights();
      void write_weights();
      void read_weights();
      //void initial_centers();
      void write_output();
      void initial_centers();
      //void list_errors();
      //void list_outputs();
};
output_layer::output_layer(input_layer* i, middle_layer* j)
{
  in=i;
  mid=j;

}
output_layer::~output_layer()
{
  //delete [num_centers] weights;
}



class network

{

private:

      input_layer *inl;
      middle_layer *ml;
      output_layer      *ol;
      int numin, numcen,numout, training;

public:
    network();
    ~network();
      void set_training(const unsigned &);
      unsigned get_training_value();
      void get_layer_info();
        void training_centers();
      double get_error();


      void set_up_network();
      void set_up_patterns( int);
      void randomize_weights();
      void update_weights( float);
      //void update_momentum();
      void write_weights();
      void read_weights();
      //void list_weights();
      void write_outputs();
```

```
};
network::network(){
       numin=0;
       numcen=0;
       numout=0;
       training=0;
}

 double train[150][50];
 int row;
 double center_vector[100][50];
 double input_vector[10];


inline float squash(float input)
// squashing function
// use sigmoid -- can customize to something
// else if desired; can add a bias term too
//
{
if (input < -50)
       return 0.0;
else  if (input > 50)
              return 1.0;
       else return (float)(1/(1+exp(-(double)input)));

}


inline double randomweight(int init)
{
int num;
// random number generator
// will return a floating point
// value between -1 and 1

if (init==1)       // seed the generator
       srand ((unsigned)time(NULL));

num=rand() % 100;

return 2*(double(num/100.00))-1;

}

void output_layer::initial_centers()
{
  int num=0;
  //int flag=0;
 // double unit;
  //ifstream select("training");
  //if(select.fail())
  //cerr<<"Input file open error";
  //else
   //{
      for(int i=0; i<mid->num_centers;i++)
       {
         num=0;
       //  flag=0;
//          srand((unsigned) time(NULL));
  //         num=rand()%100;
               for(int m=0;m<in->num_inputs;m++)
                     center_vector[i][m]=train[i][m];
         //  select>>unit;
          // flag++;
          // while(flag/9!=num)
           // {
            //   select>>unit;
             //  flag++;
             //}
             //if(flag/9==num)
              //{
             //          center_vector[i][0]=unit;
           //      for(int m=1;m<in->num_inputs;m++)
            //      select>>center_vector[i][m];
              //}
        }//end of i loop
     //end of else
  // select.close();
}

void middle_layer::update_centers( float beta)
{
  int i,x;
   i=shortest_center();
    for(x=0;x<in->num_inputs;x++)
     {
       center_vector[i][x]+=beta*(input_vector[x]-center_vector[i][x]);
     }
      results<<"\n after update centers are:  "<<endl;
      for(int m=0;m<in->num_inputs;m++)
            results<<center_vector[i][m]<<" ";

}

int middle_layer::shortest_center()
{
  int i, result, m;
  double x, y, shortest=1000.0000;

   for(i=0;i<num_centers;i++)
   {
     x=0.0000;
     y=0.0000;
     for(m=0;m<in->num_inputs;m++)
      {
        double unit=input_vector[m]-center_vector[i][m];
        x= x + unit*unit;
       }
       y=sqrt(x);
       if(shortest>y)
        {
          shortest=y;
          result=i;
         }
    }
       return result;
}

double middle_layer::calc_gaussian(int i)
```

```
 {
   double x, d, d1, d2,m,n;

    d2=calc_distance(i);      //compute ||x-ti||
    //d2=d2*d2;
    d=max_distance();
    d1=d*d;
    //x=exp(-num_centers/d1*d2);
      m=2*num_centers;
      m=(double)sqrt(m);
      n=d/m;
      //x=1/(1+d2);
      x=exp(-n*d2);
      x=x-0.5;


      results<<" \n gaussian "<<i<<"  is"<<x<<endl;
    return x;
}
//this function compute ||x-ti||^2
double middle_layer::calc_distance(int i)
{
  int m;
   double x=0.0000;
      double y=0.0000;
   for(m=0;m<in->num_inputs;m++)
   {
     y=(input_vector[m]-center_vector[i][m]);
       x+=y*y;
   }
  //return sqrt(x);
   results<<"\n for pattern :"<<endl;
   for(int n=0;n<8;n++)
         results<<input_vector[n]<<" ";
   results<<"\n";
   results<<"distance to center:"<< " "<<i<< " is: " <<x<<endl;
   return sqrt(x);

}

//compute max distance of chosen centers
double middle_layer::max_distance()
{
  double max_dis;
  double x=0.0000, y=0.0000;
  int i,j;

   for(i=0;i<num_centers;i++)
    for(j=1;j<num_centers;j++)
     {
       x=calc_two_points(i,j);
       if(max_dis<x)
        max_dis=x;
     }
      results<<"max distance is  "<<max_dis<<endl;

     return max_dis;
}

//compute eculid distance between the tow centers
double middle_layer::calc_two_points(int i, int j)
```

```
{
       double x=0.0000,y=0.0000,z=0.0000;
       for(int m=0;m<in->num_inputs;m++)
       {
             x=(center_vector[i][m]-center_vector[j][m]);
                   y=x*x;
             z+=y;

       }
       return sqrt(z);
}

//read the centers from file center

void middle_layer::read_centers()
{
  ifstream readcenter("center");
   if(readcenter.fail())
   cerr<<"center file open error";
   else
    {
      for(int i=0;i<num_centers;i++)
        for(int j=0;j<in->num_inputs;j++)
         {
           readcenter>>center_vector[i][j];
         }
     }
      readcenter.close();
 }

 void middle_layer::write_centers()
{
        ofstream writecenter("center");
  int m=0;
   if(writecenter.fail())
    cerr<<"write center file open error";
    else
     {
       for(int i=0;i<num_centers; i++)
         {
         for(int j=0;j<in->num_inputs;j++)
             {
          writecenter<<center_vector[i][j]<<endl;
            //  m++;
              //if((m%8)==0)
                   //  cout<<"\n";
              }
             cout<<endl;
         }

     }
     writecenter.close();
}


void output_layer::initial_weights()
{
   //weights=new double[num_centers];
      int w=1;
   for(int i=0;i<mid->num_centers;i++)
```

```
    weights[i]=randomweight(w);
    results<<"initial weights are:"<<endl;
    for(int m=0;m<mid->num_centers;m++)
          results<<weights[m]<<" "<<endl;
}

void output_layer::update_weights( float beta)
 {
    double actual_out, error;
    actual_out=calc_out();
    error=expected_value-actual_out;
    for(int i=0;i<mid->num_centers;i++)
     {
       weights[i]+=beta*error*mid->calc_gaussian(i);//may use sigmoid
     }
    results<<"\n after update weights, weights are: "<<endl;
    for(int m=0;m<mid->num_centers;m++)
          results<<weights[m]<<" ";

 }

 void output_layer::read_weights()
{
    ifstream readweights("weight");
     if(readweights.fail())
      cerr<<"readweights file open error";
      else
       {
         for(int i=0;i<mid->num_centers;i++)
          readweights>>weights[i];
       }
       readweights.close();
  }

  void output_layer::write_weights()
{
          int m=0;
    ofstream writeweights("weight");
     if(writeweights.fail())
      cerr<<"writeweights file open error";
       else
        {
           for(int i=0;i<mid->num_centers;i++)

            writeweights<<weights[i]<<endl;



          }
     writeweights.close();
 }

double output_layer::calc_out()
{
  int i,w=0;
  double total=0.0000, unit, gau, result;

  for(i=0;i<mid->num_centers;i++)
   {

     gau=mid->calc_gaussian(i);
```

```
      unit=weights[i]*gau;
      total+=unit;
        w++;
    }
    result=(double)(1/(1+exp(-(double)total)));
   //return result;
    results<<"output is  "<<total<<endl;
    return total +0.5;
}

void output_layer::write_output()
{
  ofstream writeoutput("output");
   if(writeoutput.fail())
    cerr<<"writeoutput file open error";
     else
      {
        for(int i=0;i<in->num_inputs;i++)
         writeoutput<<input_vector[i];
         cout<<endl;
         cout<<" the result is:"<<endl;
         cout<<calc_out();
       }
}


network::~network()
{
  delete inl;
   delete ol;
  delete ml;
}

void network::set_training(const unsigned & value)
{
training=value;
}

unsigned network::get_training_value()
{
return training;
}


void network::get_layer_info()
{


cout << " Enter in the layer sizes separated by spaces.\n";
cout << " For a network with 3 neurons in the input layer,\n";
cout << " 2 neurons in a hidden layer, and 4 neurons in the\n";
cout << " output layer, you would enter: 3 2 4 .\n";
//cout << " You can have up to 3 hidden layers,for five maximum entries
:\n\n";

      cin >> numin>>numcen>>numout;


// -----------------------------------------------------
// size of layers:
//          input_layer              layer_size[0]
//          output_layer            layer_size[number_of_layers-1]
//          middle_layers           layer_size[1]
//                            optional: layer_size[number_of_layers-3]
//                            optional: layer_size[number_of_layers-2]
//-------------------------------------------------------


}

void network::set_up_network()
{
//int i,j,k;



inl=new input_layer(numin);
ml=new middle_layer(numcen,inl);
ol=new output_layer(inl,ml);


}


//training the center before training the weights using k-mean method
void network::training_centers()
{
    ol->initial_centers();


      int u,v,h;
      double x,y;
      int w=1;
      x=ml->max_distance();
      h=ml->num_centers;
      for(u=0;u<row;u++)
    {
       for(int i=0;i<inl->num_inputs;i++)
        input_vector[i]=train[u][i];
         for(int j=0;j<ml->num_centers;j++)
         {
                y=ml->calc_distance(j);
                if(y>x)
                {
                       for(int g=0;g<inl->num_inputs;g++)
                       {
                              center_vector[ml-
>num_centers][g]=input_vector[g];

                       }
                       ml->num_centers++;
                       ol->weights[ml->num_centers]=randomweight(w);
                       x=y;

                }
                break;
         }
    }
      v=ml->num_centers-h;
      results<<"\n after adaptive  "<<v<<" centers are added"<<endl;
}
```

```
void network::randomize_weights()
{
        //ol->initial_centers();
       ol->initial_weights();
}


void network::update_weights( float beta)
{
       ml->update_centers(beta);
       ol->update_weights(beta);
}




void network::write_weights()
{
  ml->write_centers();
  ol->write_weights();
}


void network::read_weights()
{
       ml->read_centers();
       ol->read_weights();
}



void network::write_outputs()
{
  ofstream out("output");
  if(out.fail())
        cerr<<"output file open error";
       else
       {
             for(int i=0;i<inl->num_inputs;i++)
                   out<<input_vector[i];
             out<<ol->calc_out();
       }
       out.close();
}


double network::get_error()
{
       double x,y;
       x=ol->calc_out();
       y=ol->expected_value-x;
         y=y*y;       //square
         y=y+0.04;    //plus bias
       return y;
}




void network::set_up_patterns(int i)
{
        if(i==1)
              ol->expected_value=input_vector[8];
}




void main()
{

double error_tolerance=0.1;
double total_error=0.0;
double avg_error_per_cycle=0.0;
double error_last_cycle=0.0;
double avgerr_per_pattern=0.0; // for the latest cycle
double error_last_pattern=0.0;
float learning_parameter=0.02;
//float alpha; // momentum parameter

//float NF; // noise factor
//float new_NF;


unsigned temp, startup, start_weights;
//long int vectors_in_buffer;
long int max_cycles;
long int patterns_per_cycle=0;

long int total_cycles, total_patterns;
int i=0;


// create a network object
network rbf;
ifstream input("training");
ifstream test("testfile");


// enter the training mode : 1=training on     0=training off
cout << "-----------------------------------------------\n";
cout << " C++ Neural Networks \n";
cout << "   RBF simulator \n";
cout << "          version 1 \n";
cout << "-----------------------------------------------\n";
cout << "Please enter 1 for TRAINING on, or 0 for off: \n\n";
cout << "Use training to change weights according to your\n";
cout << "expected outputs. Your training.dat file should contain\n";
cout << "a set of inputs and expected outputs. The number of\n";
cout << "inputs determines the size of the first (input) layer\n";
cout << "while the number of outputs determines the size of the\n";
cout << "last (output) layer :\n\n";

cin >> temp;

row=0;
rbf.set_training(temp);
```

```
if (rbf.get_training_value() == 1)
      {
      cout << "--> Training mode is *ON*. weights will be saved\n";
      cout << "in the file weights.dat at the end of the\n";
      cout << "current set of input (training) data\n";
       while(!input.eof())
       {
             for(int q=0;q<9;q++)
             {
                   input>>train[row][q];
             }
             row++;
       }

      }
else
      {
      cout << "--> Training mode is *OFF*. weights will be loaded\n";
      cout << "from the file weights.dat and the current\n";
      cout << "(test) data set will be used. For the test\n";
      cout << "data set, the test.dat file should contain\n";
      cout << "only inputs, and no expected outputs.\n";
      while(!test.eof())
      {
            for(int q=0;q<8;q++)
            {
                  test>>train[row][q];
            }
            row++;
      }
      }

if (rbf.get_training_value()==1)
      {
      // -----------------------------------------
      //    Read in values for the error_tolerance,
      //    and the learning_parameter
      // -----------------------------------------
      cout << " Please enter in the error_tolerance\n";
      cout << " --- between 0.001 to 100.0, try 0.1 to start --\n";
      cout << "\n";
      cout << "and the learning_parameter, beta\n";
      cout << " --- between 0.01 to 1.0, try 0.5 to start -- \n\n";
      cout << " separate entries by a space\n";
      cout << " example: 0.1 0.5 sets defaults mentioned :\n\n";

      cin >> error_tolerance >> learning_parameter;

      cout << "Please enter the maximum cycles for the simulation\n";
      cout << "A cycle is one pass through the data set.\n";
      cout << "Try a value of 10 to start with\n";

      cin >> max_cycles;

      cout << "Do you want to read weights from weights.dat to
start?\n";
      cout << "Type 1 to read from file, 0 to randomize starting
weights\n";

      cin >> start_weights;
      }



// training: continue looping until the total error is less than
//          the tolerance specified, or the maximum number of
//          cycles is exceeded; use both the forward signal propagation
//          and the backward error propagation phases. If the error
//          tolerance criteria is satisfied, save the weights in a file.
// no training: just proceed through the input data set once in the
//          forward signal propagation phase only. Read the starting
//          weights from a file.
// in both cases report the outputs on the screen


// intialize counters
total_cycles=0; // a cycle is once through all the input data
total_patterns=0; // a pattern is one entry in the input data
//new_NF=NF;


// get layer information
rbf.get_layer_info();

// set up the network connections
rbf.set_up_network();

// initialize the weights
if ((rbf.get_training_value()==1)&&(start_weights!=1))
      {
      // randomize weights for all layers; there is no
      // weight matrix associated with the input layer
      // weight file will be written after processing

      rbf.randomize_weights();
      rbf.training_centers();
      // set up the noise factor value
      //backp.set_NF(new_NF);
      }
else
      {
            rbf.read_weights();

      }



// main loop
// if training is on, keep going through the input data
//          until the error is acceptable or the maximum number of
cycles
//          is exceeded.
// if training is off, go through the input data once. report outputs
// with inputs to file output.dat

startup=1;
total_error = 0;
double temp1[100];
```

```
int total_pats=0;

if(rbf.get_training_value()==1)
{



//     rbf.training_centers();


         //ifstream input("training");
         //if(input.fail())
         //cerr<<"input file open error";
       i=1;
       avgerr_per_pattern=100.0000;

       total_cycles=0;
       double unit_error=0.000000;
       double total_error=0.000000;


while((avgerr_per_pattern>error_tolerance)&&(total_cycles<max_cycles))
       {
             patterns_per_cycle=0;
                 total_error=0.0000;
                 total_pats=0;
             //while(!input.eof())
                         for(int r=0;r<row;r++)
             {
                   for(int l=0;l<9;l++)
                         input_vector[l]=train[r][l];


             rbf.set_up_patterns(i);

             total_patterns++;
             total_pats++;
             unit_error=rbf.get_error();
             total_error+=unit_error;

             rbf.update_weights(learning_parameter);
             }
             total_cycles++;
             double pre=avgerr_per_pattern;
             avgerr_per_pattern=total_error/total_pats;
                 cout<<"new error is:  "<<avgerr_per_pattern<<endl;
             if(pre<avgerr_per_pattern)
             {
             cout<<"Weights are blowing up, try a small learning rate";
                   exit(1);
             }
             //input.close();
              //input.open("training");
            // if(input.fail())
      //              cerr<<"input file open error";

       }
       results.close();
       rbf.write_weights();
       rbf.write_outputs();
cout << "         weights saved in file weights.dat\n";
cout << "\n";
cout << "---->average error per cycle = " << avg_error_per_cycle << " <-
--\n";
cout << "---->error last cycle = " << error_last_cycle << " <---\n";
cout << "->error last cycle per pattern= " << avgerr_per_pattern << " <-
--\n";



cout << "------------>total cycles = " << total_cycles << " <---\n";
cout << "------------>total patterns = " << total_patterns << " <---\n";
cout << "------------------------------------------------\n";
}//end of if training

if(rbf.get_training_value()==0)
{
      int i=0;
       // ifstream input("training");
      for(int r=0;r<row;r++)
      {
            for(int m=0;m<8;m++)
             temp1[m]=train[r][m];

            rbf.set_up_patterns(i);
            rbf.write_outputs();
      }
      //input.close();
      cout<<"end of test, you may see result in output file";
}



}
```

APPENDIX F Code for MLP Simulator

```
/*********************************************
    Multi-layer Perceptron Simulator Using Conjugate
     Gradient method

     Feb, 1999. By Yimin Yang. Computer Science Department
     Oklahoma State University
...........................................................
  This program is written using the method described in
Welstead's book "Neural Networks and Fuzzy Logic
Applications In C/C++", Stephen T. Welstead, John Wiley &
Sons, Inc.1994
Before training the program, the file "training" must be set
Before testing the simulator, the file "testfile" must be
set. The program is written in C++, can run in both Unix and
PC Visual C++ enviranment
*******************************************************/
```

```
#include<iostream.h>
#include<fstream.h>
#include<math.h>
#include<time.h>
#include<stdlib.h>
#include<stdio.h>
#include<assert.h>
//#include<fcntl.h>


double train[110][15];
int index;
const double cg_zero_check=1e-8;

class conjugate{
private:
      int la,lb,lc,no_of_wts;
      double threshold, target,actual;
      double weights[100];
      double input_vector[20];
      double hide_out[20];
      float error_tol;
      void rand_init();
      float vect_norm(double *, int, int);
      void gradient_of_obj_fn(double*, double*);
```

```
        void calc_hide(double*,double*);
        void minus(double*, double*, int, int);
        void unit_vector(double*,double*,int, int);
        void find_min(double*,double*, int,int,
float,float,double*,float*);
        float obj_fn(double*);
        void form_new_vector(float, double*,double *, double*,int,int);
        double find_beta(double*, double*, int,int);
        double dot_prod(double*, double*,int, int);
        void write_weights();
        //void print();


public:
        //conjugate();
        conjugate(int,int,int,double,float);
        conjugate(int,int,int);
/*      conjugate(int m1,int m2, int m3, double threshold1, float er){
                la=m1;
                lb=m2;
                lc=m3;
                target=0.0;
                actual=0.0;
                threshold=threshold1;
                no_of_wts=la*lb+lb*lc+1;
                input_vector=new double[la+1];
                hide_out=new double[lb];
                weights=new double[no_of_wts];
                error_tol=er;
        }*/
        ~conjugate(){
        //      delete  weights[100];
        //      delete input_vector[20];
        //      delete hide_out[20];
        }

        void submain();
        void print();
};//end of class
conjugate::conjugate(int m1,int m2, int m3, double threshold1, float
er){
                la=m1;
                lb=m2;
                lc=m3;
                target=0.0;
                actual=0.0;
                threshold=threshold1;
                no_of_wts=(la+1)*lb+lb*lc;
        //      input_vector=new double[100];
        //      hide_out=new double[50];
        //      weights=new double[100];
                error_tol=er;
        }
conjugate::conjugate(int r1,int r2,int r3)
{
        la=r1;
        lb=r2;
        lc=r3;
}

void conjugate::submain()
```

```
{
//     double *gradient,*new_gradient,
*neg_new_gradient,*new_vector,*neg_gradient, *new_weights;
            double gradient[100];
             double new_gradient[100];
            double neg_new_gradient[100];
                  //double test[100][10];
                  //test=new double[100][10];

       double new_vector[100];
       double neg_gradient[100];
       double new_weights[100];
      //double *direction_vector;
       double direction_vector[100];

      long iter=0;
      double alpha, beta, the_error;
      float obj_value, step_size, total_error;

      rand_init();//initialize the weights


      step_size=vect_norm(weights, 1,no_of_wts);
      gradient_of_obj_fn(weights,gradient);
      minus(gradient,neg_gradient,1,no_of_wts);
      unit_vector(neg_gradient,direction_vector,1,no_of_wts);
      the_error=vect_norm(gradient,1,no_of_wts);
    total_error=obj_fn(weights);
      find_min(weights,direction_vector,1,no_of_wts,total_error,step_siz
e,&alpha,&obj_value);
      cout<<"\nAlpha is: "<<alpha<<" obj value is: "<<obj_value;
      cout<<"\n grad norm is "<<the_error;
      cout<<" error tolerence is "<<error_tol;

form_new_vector(alpha,weights,direction_vector,new_weights,1,no_of_wts);
      for(int i=0;i<no_of_wts;i++)
            weights[i]=new_weights[i];
//    cout<<"\n Press key: ";
//getch();
     while(obj_value>error_tol)
       {
             iter++;
             gradient_of_obj_fn(weights,new_gradient);
             minus(new_gradient,neg_new_gradient,1,no_of_wts);
             if(iter%no_of_wts==0)

unit_vector(neg_new_gradient,direction_vector,1,no_of_wts);
             else
             {
                   beta=find_beta(gradient,new_gradient,1,no_of_wts);
                   cout<<"beta is : "<<beta;

form_new_vector(beta,neg_new_gradient,direction_vector,new_weights,1,
no_of_wts);
                   unit_vector(new_vector,direction_vector,1,no_of_wts);
             }//end of else
             for(int x=0;x<no_of_wts;x++)
                   gradient[i]=new_gradient[i];
             the_error =vect_norm(gradient,1,no_of_wts);
             double d=1e-6;
             if(the_error>d)
```

```
                    step_size=5*(obj_value)/the_error; //5*y divided by
the slope
              else
                    step_size=0.5*step_size;
               total_error=obj_fn(weights);

find_min(weights,direction_vector,1,no_of_wts,total_error,step_size,&alp
ha, &obj_value);
               cout<<"\nObj value is: "<<obj_value;
               cout<<"\n now is cycles is:   "<<iter<<endl;

form_new_vector(alpha,weights,direction_vector,new_weights,1,no_of_wts);
               for(int m=0;m<no_of_wts;m++)    //copy new weights to
weights
                      weights[m]=new_weights[m];

       }

       write_weights();

}

void conjugate::write_weights()
{
      ofstream out("weights_file");
      if(out.fail())
            cerr<<"Weights file open error";
      int m=0;
      for(int i=0;i<no_of_wts;i++)
      {
       out<<weights[i]<<" ";
         m++;
         if(m%8==0)
               out<<endl;
         }
}


double conjugate::find_beta(double *we,double *dir,int m,int len)
{
      double denom;
      denom=dot_prod(we,we, m,len);
      if(denom<cg_zero_check)
            denom=cg_zero_check;
      return dot_prod(dir,dir,m,len)/denom;
}

double conjugate::dot_prod(double *v1,double *v2,int start,int len)
{
      int i;
      double sum=0.0;
      for(i=0;i<len;i++)
            sum+=v1[i+start]*v2[i+start];
      return sum;
}
```

```
void conjugate::find_min(double *we,double *dir,int m,int len,float
te,float st,double *a,float*ov)
{
  //brute force method
      const double f_zero_toler=1e-8, x_zero_toler=1e-4;
      int i, no_of_pts=100;
      float opt_value, value, opt_x, x, xinc,xmin,range;
      //double *v;
      double v[100];
      range=st;
      do{
            xmin=-range;
            xinc=2*range/no_of_pts;
            x=xmin;
            form_new_vector(x,we,dir,v,m,len);
            opt_value=obj_fn(v);
            opt_x=x;
            for(i=1;i<no_of_pts;i++)
            {
                  x+=xinc;
                  form_new_vector(x,we,dir,v,m,len);
            //    for(int h=0;h<len;h++)
                        //v[h]=we[h]+x*we[h];
                        //v[h]=we[h]+x* dir[h];
                  value=obj_fn(v);
                  if(value<opt_value)
                  {
                        opt_x=x;
                        opt_value=value;
                  }
            }
            *a=opt_x;
            *ov=opt_value;
            range=range*0.1;
            //repeat until either a sizable move(x) is found, or the
func is zero
      }while ((abs(*a)<x_zero_toler)
&&(opt_value>f_zero_toler)&&(range>x_zero_toler));

//    if(range<x_zero_toler)
//          cerr<<"\nfind_min cannot find min in this direction\n";
}

void conjugate::form_new_vector(float a,double* w1,double* w2,double*v,
int start, int len)
{
      int i;
      for(i=0;i<len;i++)
      {
            v[i]=w1[i]+a*w2[i];
      }
      return;
}


float conjugate::obj_fn(double* we)
{
      int i,m;
      //double *pattern;
      double sum=0;
       double pattern [20];
      pattern[0]=1;
       for(i=0;i<index;i++)
       {
              for(m=0;m<la+1;m++)

                     pattern[m+1]=train[i][m];
                     calc_hide(pattern,we);
                     double x=(pattern[la+1]-actual);

                            x=x*x;
                     sum+=x;
       }
            sum=sum/index;
          // double g=exp(-sum);
             //float f=1/(1+g);
             return sum;
       //    return 1/(1+exp(-sum));
}


void conjugate::unit_vector(double *neg, double *dir, int m,int len)
{
       double denom;
       int i;
       denom=vect_norm(neg,m,len);
       if(denom<cg_zero_check)
              denom=cg_zero_check;
       for(i=0;i<len;i++)
              dir[i]=neg[i]/denom;
       return;
}

void conjugate::minus(double *grad, double *neg,int m,int len)
{
       int i;
       for(i=0;i<len;i++)
              neg[i]=-grad[i];
}


float conjugate::vect_norm(double *w,int i,int len)
{
       int n;
       double sun=0.0;
       for( n=0;n<len;n++)
              sun+=w[n]*w[n];
        float x=sqrt(sun);
        return x;
}

void conjugate::gradient_of_obj_fn(double *wei, double* grad)
{
       int i,j,k,p;
       double factor,sum;
       weights[0]=threshold;
       grad[0]=threshold;

       for(i=1;i<no_of_wts+10;i++)
              grad[i]=0.0;
       for(p=0;p<index;p++)
       {
            input_vector[0]=1;
            for(i=1;i<la+2;i++)
            input_vector[i]=train[p][i-1];
            target=input_vector[la+1];
            calc_hide(input_vector,wei);
            k=0;
            //input_layer weights
            for(i=1;i<la+1;i++)
              for(j=0;j<lb;j++)
              {
                    k++;
                    sum=0.0;
                    factor=hide_out[j]*(1.0-
hide_out[j])*input_vector[i];
                    sum=(target-actual)*actual*(1.0-
actual)*weights[la+j]*factor;
                    grad[k]+=sum;
              }
             //output layer weights
              for(j=0;j<lb;j++)
              {
                    k++;

                          double x=(target-actual)*actual*(1.0-
actual)*hide_out[j]*weights[la*lb+1+j];
                          assert(x>-1&&x<1);
                    grad[k]+=x;
              }

      }
}


void conjugate::print()
{
      int i,m,l=0;
      double pat[20];
      double wei[100];
      double sum=0;

    ofstream output("output");
      ifstream read_weights("weights_file");
      if(read_weights.fail())
            cerr<<"\nWeights read file open error\n";
      for(i=0;i<la*lb+lb+1;i++)
            read_weights>>wei[i];
      for(i=0;i<index;i++)
      {
            pat[0]=1;
            for(m=1;m<la+2;m++)
                  pat[m]=train[i][m-1];

            calc_hide(pat,wei);
            output<<"\n For pattern :\n";
            for(m=1;m<la+1;m++)
                  output<<pat[m]<<" ";
            output<<" \nThe actual output from network is:
"<<actual<<endl;
            output<<"\n The target output is :  "<<pat[la+1]<<endl;
      //    output<<"\n The difference is:  "<<
                  double unit=(pat[la+1]-actual);//*(pat[la+1]-actual);
```

```
        sum+=unit;
            output<<"\n The difference between target and actual
is:"<<unit;
              if((pat[la+1]-actual)<0.03)
                    l++;
      }
     output<<"\n The average error for testfile is:  "<<sum/index;
      output<<"\n The number of predict error less than 0.03 is:
"<<l<<endl;
      output<<"\n The number of all test pattern is: "<<index;
      double rate=0.00;
      rate=l/index;
      output<<"\n Predicting correct rate is:  "<<rate;
}



void conjugate::calc_hide(double *input_vect,double*we)
{
      int m,y;
      double n,num;
   for( m=0;m<lb;m++)
   {
          n=0.0,num=0.0;
          y=0;
         for(y=0;y<la+1;y++)
         //for( p=0;p<la+1;p++)
               n+=input_vect[y]*we[m*la+y];

         hide_out[m]=1/(1+exp(-n));
   }
    n=0.0;
   for(y=0;y<lb;y++)
         n+=hide_out[y]*we[la*lb+1+y];
       actual=1/(1+exp(-n));

}




void conjugate::rand_init()
{
      int i;
      double r;
      srand(1);
      for(i=0;i<no_of_wts+10;i++)
      {
            r=2.0*(rand()%100)/105.0000-1.0;//randon number betwwen -1
and 1
            weights[i]=r*0.5;
      }
}


void main()
{
      int l1,l2,l3;
      double threshold;
      float err;
      ifstream input("training.txt");
     int type;
      if(input.fail())
            cerr<<"\n input File open error";
      cout<<"Please enter training or test, for training enter 1, for
test enter 0\n";
      cin>>type;

      cout<<"\n Please enter nodes for network :";
      cout<<"\n each layer has a space like 8 4 1\n";

      cin>>l1>>l2>>l3;
      if(type==0)
      {
            ifstream te("testfile.txt");
            if(te.fail())
                  cerr<<"\nTestfile open error\n";
            while(!te.eof())
            {
               for(int u=0;u<l1+1;u++)
                  te>>train[index][u];
                 index++;
            }

            conjugate test(l1,l2,l3);
            test.print();
            te.close();
      }

      else
      {


      cout<<"\n Please enter threshold:";
      cout<<"\n between 0,0000001 and 1\n";
      cin>>threshold;
      cout<<"\n Please enter error tolerence betwween 0.00 to 1.00\n";
      cin>>err;
      index=0;

      while(!input.eof())
      {
            for(int i=0;i<l1+l3;i++)
                  input>>train[index][i];
            index++;

            if(index==100)
                  break;
      }


       conjugate  con(l1,l2,l3,threshold,err);
      con.submain();

      input.close();
      }
}
```

VITA

YIMING YANG

Candidate for the Degree of

Master of Science

Thesis: A COMPARISON OF NEURAL NETWORKS FOR STOCK SELECTION

Major Field: Computer Science

Biographical:

Personal Data: Born in Hebei Province, P.R. China, September 28, 1965, the son of Shiguo Yang and Jianying Ma.

Education: Received Bachelor of Arts Degree in English Literature and Language from Hebei Teacher's University in June 1990. Received Bachelor of Science Degree in International Politics from Renming University of China in June 1992; completed requirements for the Master of Science degree in Computer Science at Oklahoma State University in May, 1999.

Professional Experience: Portfolio Manager, China Xinxing Group, Beijing, 1994-1997; China International Trust and Investment Corporation (CITIC), Beijing, 1993-1994; China Poly Group, Beijing, 1992-1993.