Neural Network Applications For Finance

by

Ronjon Nag

B.Sc. (Hons), University of Birmingham, UK (1984)
Ph.D., University of Cambridge, UK (1988)

Submitted to the Alfred P. Sloan School of Management
in partial fulfillment of the requirements for the degree of

Master of Science in Management

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1991

Signature of Author ....
Alfred P. Sloan School of Management
January 25th, 1991

Certified by ....
Andrew W. Lo
Associate Professor, Finance
Thesis Supervisor

Accepted by ....
Jeffrey A. Barks
Associate Dean, Master's and Batchelor's Programs

Neural Network Applications For Finance

by

Ronjon Nag

Submitted to the Alfred P. Sloan School of Management
on January 25th, 1991, in partial fulfillment of the
requirements for the degree of
Master of Science in Management

Abstract

Neural networks have recently emerged as a popular technique for pattern matching in the fields of speech and vision recognition. This thesis presents applications of neural networks in the field of finance. The contents presents the basic theory, outlines recent work showing the usefulness of neural networks in finance and presents original research results of applications of neural networks to stock market prediction.

Research in the literature has demonstrated the potential usefulness of neural networks for share trading but has not explicitly compared the technique with conventional mathematical models. In this work, a neural network is compared with a multiple regression model for the task of predicting AT&T stock returns. While the neural network's prediction error was better than that of the multiple regression model, the residual autocorrelations were higher indicating that overall performance may not necessarily be better. Although the difficulty of a simple neural network model to handle time series forecasting problems relates to generalization properties of the network on limited training data, neural networks may be more applicable to categorization problems such as credit verification. The thesis makes suggestions for improvements to the simple neural network model for the case of time series forecasting.

Thesis Supervisor: Andrew W. Lo
Title: Associate Professor, Finance

Contents

1 Introduction 5

1.1 What are neural networks ? 7

1.2 Limitations of the single layer perceptron 8

1.3 Recent improvements 9

1.4 Outline of the thesis 11

2 Backpropagation Algorithm 12

2.1 The training stage 14

2.2 Recognition 17

2.2.1 Choosing the learning rate α and the momentum μ 18

2.3 How many nodes ? 18

2.4 Summary 19

3 Neural Networks for Stock Price Prediction 21

3.1 The problem of stock price prediction 22

3.2 Previous work in neural nets for stock price prediction 23

3.3 An experiment in predicting stock prices with neural nets 23

3.4 Prediction of AT&T stock price 24

3.5 Using a neural network to learn how to trade 27

3.6 Summary of effectiveness in trading applications 28

4 Other neural network applications in finance **31**

4.1 Application to bankruptcy prediction 32

4.2 Application to bond rating prediction 32

4.3 Application to arbitrage pricing model testing 33

5 Discussion and Conclusion **35**

5.1 Surmounting the overfitting problem 36

5.2 Categorization vs time series forecasting 36

5.3 Genetic algorithms 38

5.4 Implementation considerations 38

Chapter 1

Introduction

Neural networks (also known as *connectionist models and parallel distributed processing models* have recently emerged as a popular technique for learning patterns in the fields of speech and vision recognition. This thesis presents applications of neural networks in the field of finance. The contents presents the basic theory, outlines recent work showing the usefulness of neural networks in finance and presents original elementary research results of applications of neural networks to stock market prediction. The thesis also makes suggestions for further improvements to the algorithm for the case of time series forecasting.

The stimulus for neural network computing has been to attempt to design mathematical formalisms which are consistent with the current understanding of the workings of the human brain. In spite of this aim, the most popular algorithms in the field have severe deficiencies to current brain theory (Crick, [4]). Despite this, useful results have been obtained in the physical sciences; and psychologists have used neural network theory as a crude tool to visualise how the microstructure of the brain may or may not work (Rumelhart [29]).

Unlike conventional computing paradigms, such as the von Neumann computer, which process computer programs in a serial fashion, neural computers are massively parallel architectures, considering many different hypotheses in concurrent fashion by the use of *massively parallel nets* composed of computing *nodes* linked by *weights.*

Neural networks are specified by three factors:

1. The computation characteristics of each node: what does each node compute ?
2. The network topology: how are the nodes connected ?
3. The learning algorithm: how does the network learn, and what are the limitations

While the neural architecture does offer the application to parallel processing hardware, many researchers apply the algorithms to serial computers which simulate parallel operation. The main advantages of neural networks over other learning paradigms are:

- Fault tolerance: can handle certain amounts of noisy data
- Learning by example: no expert system type rules to write
- Ability to *generalize* from the data: infer trends in the data
- Fewer assumptions: weaker assumptions relative to traditional statistical techniques- potentially more useful for non-linear problems

What are neural networks used for ? There are three main applications in the field:

1. *Pattern matching*: which category does an input belong to, given that either the training or testing data may be noisy
2. *Content-addressable associative memory*: essentially a decoding schema, whereby only part of an input pattern is available and the full pattern is required as an ouput, eg. a page of writing with an ink blot
3. *Vector quantisation*: also known as clustering

Note that none of the above refer explicitly to time series forecasting problems. Even in the physical sciences, the application of connectionist models to time series problems have met with difficulty, though there have been some attempts (Waibel, [31], Weigend [32]). The central theme of all neural network algorithms is that computational architectures are made up of massively parallel nets as described above. However, there are many different topologies and training algorithms depending on the emphasis of the application. In this thesis, effort is concentrated on pattern matching style networks which are more readily extendable to the time series forecasting case. *Single* and *Multiple Layer Perceptrons*, which have been shown to be particularly useful in pattern matching, are described in detail. Other commonly used network architectures are Hopfield Nets (Hopfield, [9]) for content-addressable associative memories, and Kohonen networks (Kohonen [13]) for clustering.

1.1 What are neural networks ?

The simplest neural network can be represented by figure 1-1. This formalism is often described as a *Perceptron* (Rosenblatt, [28]). The particular example in figure 1-1 consists of only one node. This type of network can be used to determine whether an input belongs to one of two categories only. The node consists of a number of variables x_i being input into the node. What does the node do ? The node acts a simple processing element which outputs a function called the *activation* $a(x)$ of the node, where $a(x)$ is given by:

$$a(x) = f\left(\sum_{\forall i} w_i x_i + \theta\right) \tag{1.1}$$

where θ is a *bias* value and w_i are *weights* which effect on each input x_i. the network therefore simply computes a weighted sum of the input variables, and subtracts a threshold. This result is then passed through a function $f()$ which is a non-linearity such that a(x) is between +1 or −1, which in turn corresponds to the classification decision between the two categories.

Usually the outputs of this node act as inputs to other nodes to form a neural network *topology* or *architecture*. The simplest extension of the single node perceptron is to arrange the nodes in two layers. Such an example is called a *single layer perceptron* and is shown in Figure 1-2 with example data input variables.

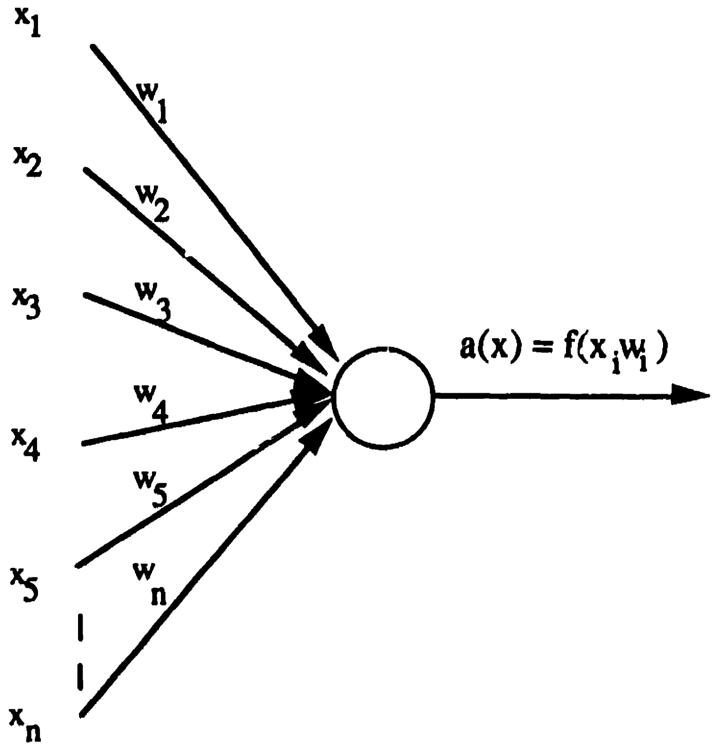


Figure 1-1: Single Perceptron Node

How are the weights and bias terms computed ? There are several algorithms for computing these parameters in the single layer perceptron case. The reader is referred to Lippman [16] or Rosenblatt [28] for a descriptions of computing the parameters in a technique known as the *perceptron convergence procedure*. The thrust of this thesis is to present the more general *back propagation algorithm* which is capable of training more complex architectures.

1.2 Limitations of the single layer perceptron

The initial presentation of the single layer perceptron aroused significant interest at the time of introduction (Rosenblatt, [28]). It was shown that the single layer perceptron could be used for separating classes which could be separated by a hyperplane as shown in Figure 1-3. However, if the classes cannot be a separated by a single hyperplane, then the network fails to learn the

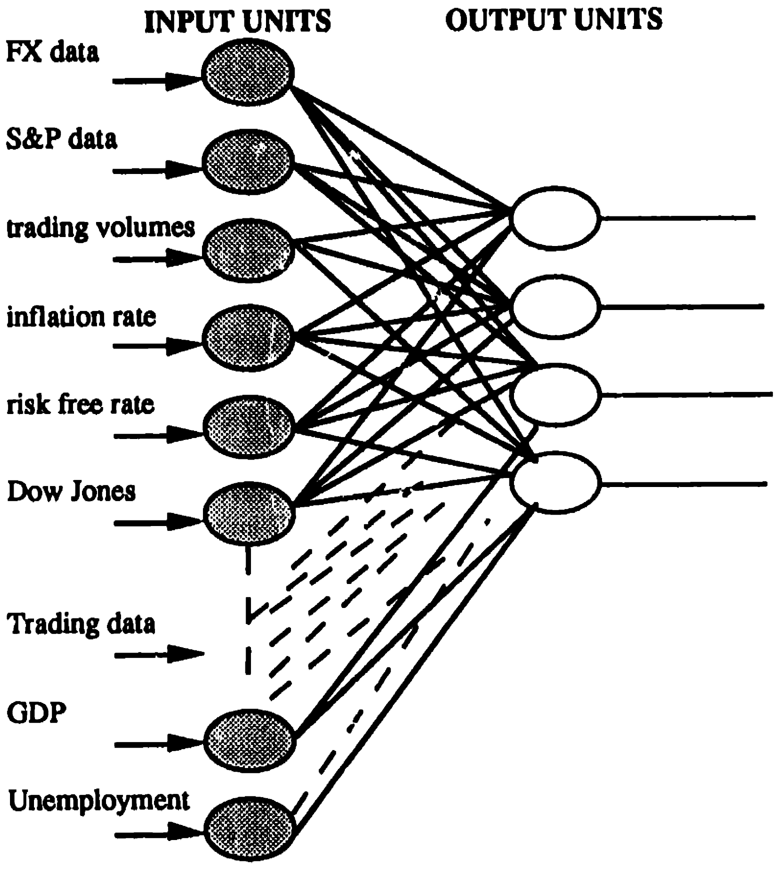


Figure 1-2: Single Perceptron Layer with Example Financial Inputs

categorization. A classic experiment undertaken by Minsky and Papert [20] to solve for the classes of the exclusive-OR problem demonstrates this. Figure 1-4 presents a diagram of the problem. Note that it is not possible to fit a hyperplane to separate these two classes. As a consequence of this result, research into *connectionist models* declined significantly in favour of rule-based artifical intelligence research (Rappa [26]).

1.3 Recent improvements

In recent years, methods for learning nonlinear functions have been proposed. These results rely on designing the network so that:

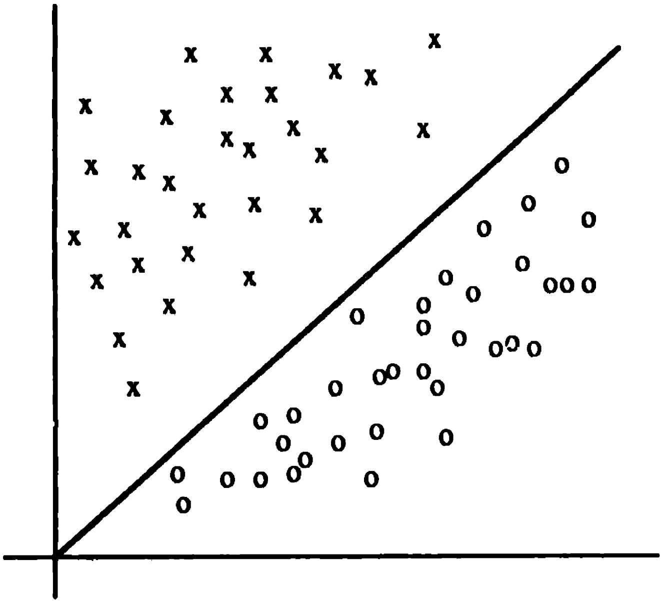


Figure 1-3: Example of a categorization problem that could be solved by a single layer perceptron

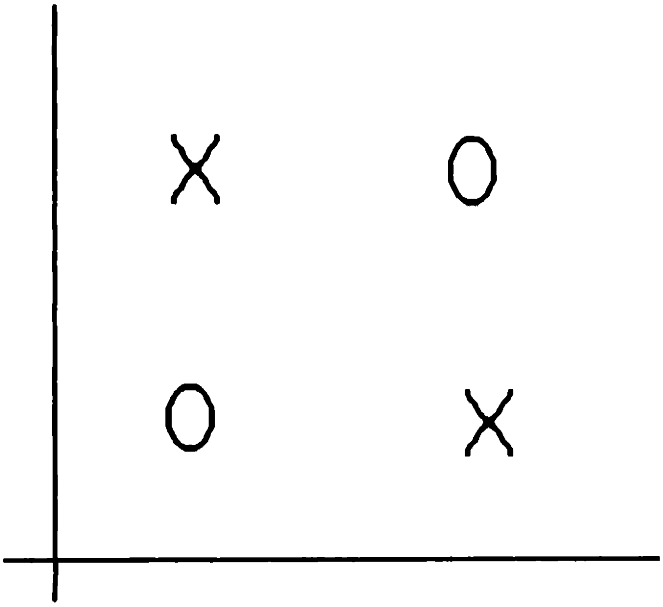


Figure 1-4: Example of a categorization problem that can not be solved by a single layer perceptron

- The neural network has at least one layer of nodes in between the input and ouput nodes - known as the hidden layer. This gives rise to the term *multi-layer perceptron.*
- The activation functions are differentiable so that learning can take place using the *back-propagation algorithm.*

By satisfying these conditions, the neural network has been shown to be able to learn nonlinear functions. It is with this motivation that it has been applied to difficult problems in the physical sciences.

The backpropagation algorithm has been a major force behind the popularity of connectionist models. It has been useful a tool to estimate the parameters of a neural network. It became well known to the community by the work of mathematical psychologists at Stanford (Rumelhart, [29]) but was also developed independently by other researchers at MIT Sloan School of Management (Parker, [22]).

1.4 Outline of the thesis

The basic concepts of neural networks have been introduced. The remainder of this thesis is concerned with describing in detail the learning algorithm and its applications to finance. Chapter 2 describes the backpropagation algorithm in detail; chapter 3 presents original experiments to demonstrate the prediction peroformance of a simple neural network model; chapter 4 describes other applications in finance; and chapter 5 concludes with a discussion of potential enhancements and implementation issues.

Chapter 2

Backpropagation Algorithm

The backpropagation algorithm was first popularised by psychologists at Stanford (Rumelhart [29]) who were investigating methods in artificial intelligence to aid research in learning. However, the algorithm was also proposed independently by other researchers at MIT Sloan School of Management (Parker, [22]) and Harvard (Werbos, [33]).

The backpropagation algorithm is typically used to train *feedforward networks*. A feedforward network consists of *layers* of nodes connected by feedforward connections. In the general case it is possible to have several inner or *hidden* layers. Each inner layer of *hidden nodes* is not, except in the single hidden layer case, directly connected to both the input and output layers. Rather, each layer takes its inputs from nodes in the previous layer and outputs to nodes in the successive layer. Figure 2-1 demonstrates a two hidden layer network. However, it has been shown that a network with just one hidden layer can theoretically learn any function (Lippman, [16]). The explanation of the algorithm is explained for the case of a three layer (one hidden layer) architecture.

The advantage of the multilayer perceptron is that the incorporation of at least one hidden layer results in the ability to learn non-linear functions. The capabilities are a direct result of the use of non-linear activation functions in the processing nodes.

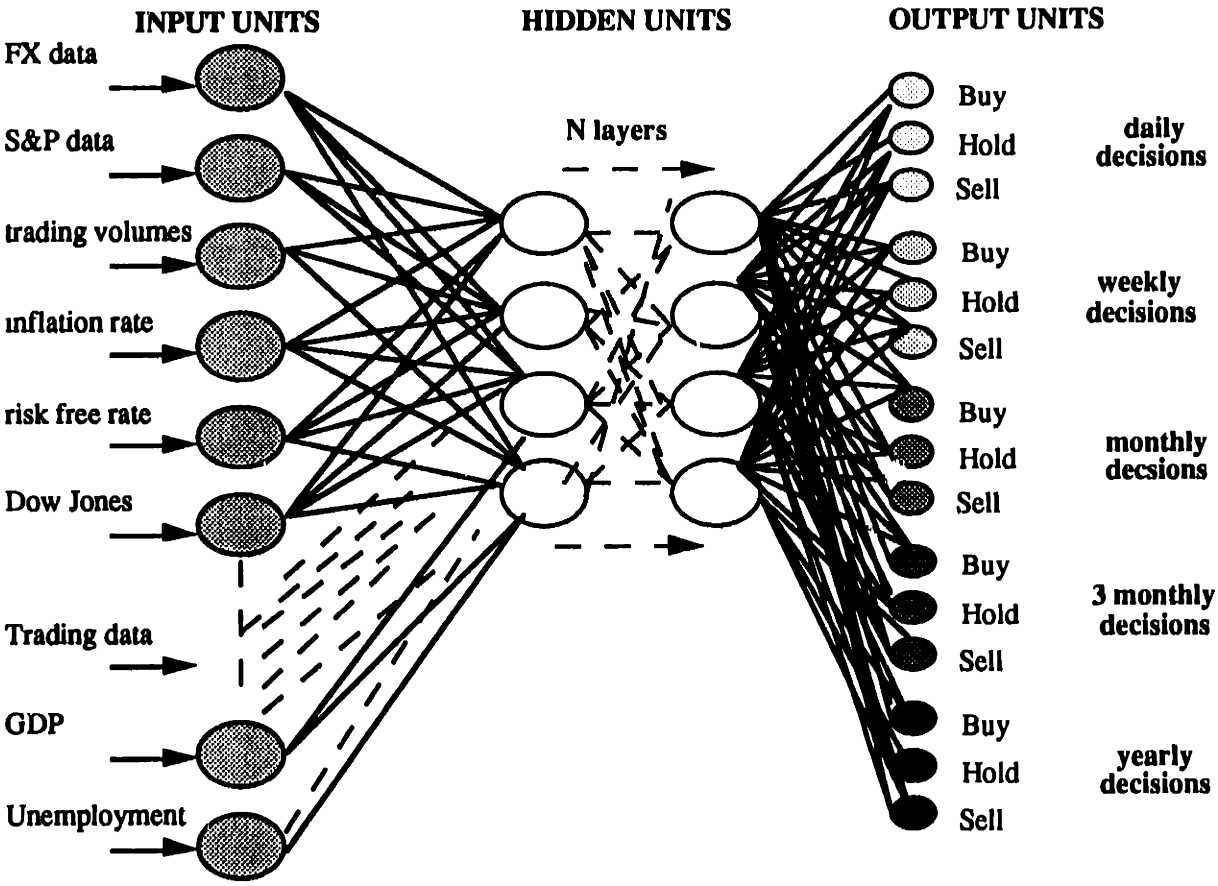


Figure 2-1: Example of a multilayer feedforward neural network with financial data inputs and decision outputs

2.1 The training stage

Neurons communicate over synaptic links as shown previously in Figures 2-1, 1-2 and 1-1. During recognition, information flows unidirectionally from the input layers through the hidden layers and out through the ouput layer. In the training stage, however, information flows in the reverse direction from the output to the input in a technique known as backpropagation (Parker, [22]; Rumelhart, [29]).

The backpropagation training algorithm is a method for updating the parameter values of the weights of the connection links and the activation values of the processing element nodes. In order to perform the input-output mapping, the algorithm minimizes a cost function of the error between the computed output of the network and the desired (target) ouput. Usually this cost function is simply the Euclidean distance between the computed output value and the desired value for each output processing element across all patterns in the data set. Other cost functions have been suggested however (Weigend, [32]).

In any case, whatever cost function is implemented, the procedure to adjust the weights is derived by computing the change in cost function with respect to the change in each weight. The algorithm can be generalized to adjust the weights connecting every layer, keeping in view that the error at each node is a proportionately weighted sum of the errors produced by the previous layer.

Let $(A_k, C_k), k = 1, 2, ..., m$ represent input-target pairs, where A_k represents the k'th occurence of an input pattern vector presentation and C_k represents the corresponding target vector., so that $A_k = (a_1^k, \ldots, a_H^k)$ and $C_k = (c_1^k, \ldots, c_J^k)$. The backpropagation algorithm learns offline, operates in discrete time and is represented by a three-layer feedforward topology , where the H processing elements of input layer A represent the A_k components and the J processing elements of output layer C correspond to the C_k components. The hidden layer is defined below.

Consider therefore that the network has input nodes, h, $(0 \leq h \leq H)$, hidden nodes i, $(0 \leq i \leq I)$, with thresholds θ_i and output nodes j, $0 \leq j \leq J)$ with thresholds Γ_j.

The algorithm can be described as follows:

1. Assign random values in the $(1,-1)$ section to the folowing:
    - all connections v_{hi} between the first layer and and the hidden layer, where v_{hi} is the weight between node h in the first layer and node i in the hidden layer;
    - all connections w_{ij} between the hidden layer and and the output layer, where w_{ij} is the weight between node i in the hidden layer and node i in the output layer;
    - to each processing element threshold θ_i in the hidden layer nodes;
    - to each processing element threshold Γ_j in the output layer.
2. For each of m pattern pairs (A_k, C_k), for $k = 1, 2, \ldots\ldots m$, do the following:
    (a) Compute the hidden layer activations, b_i:

    $$b_i = f\left(\sum_{h=1}^{H} a_h v_{hi} + \theta_i\right) \tag{2.1}$$

    for all $i = 1, 2, \ldots, p$, where b_i is the activation value and θ_i is the bias value of the i'th hidden node.

    (b) The activation function is signified by $f()$. This function is usually set as a linear function or more popularly as a logistic sigmoid function so that:

    $$b_i = \frac{1}{1 + \exp - (\sum_h a_h v_{hi} + \theta_i)} \tag{2.2}$$

    (c) Compute the output layer activations, c_j:

    $$c_j = f\left(\sum_{i=1}^{I} b_i w_{ij} + \Gamma_i\right) \tag{2.3}$$

    for all $j = 1, 2, \ldots, p$, where c_i is the activation value and θ_i is the bias value of the j'th output node. The activation function is signified by $f()$. As in the hidden nodes, this function is usually set as a logistic sigmoid function so that:

    $$c_j = \frac{1}{1 + \exp - (\sum_i b_i w_{ij} + \theta_j)} \tag{2.4}$$

(d) Compute the error δ_j between the desired target value and the actual computed output by using:

$$\delta_j = \left(c_j^k - c_j\right) c_j' \tag{2.5}$$

where c_j' is given as the derivative of c_j with respect to its total input $net_{pj} = \sum w_{ji} b_j + \Gamma_j$. This derivative is given by:

$$\frac{\delta c_j}{\delta net_j} = c_j \left(1 - c_j\right) \tag{2.6}$$

By substituting the derivative in equation 2.6 into equation 2.5, we obtain:

$$\delta_j = c_j \left(1 - c_j\right) \left(c_j^k - c_j\right) \tag{2.7}$$

Compute the above equation for all $j = 1, 2, \ldots, J$ where J is the number of output nodes.

Incidentally, were this network to have many layers, the error for an arbitrary hidden unit with output x_j would be given by:

$$\delta_{pj} = x_j \left(1 - x_j\right) \sum_l \delta_l w_{lj} \tag{2.8}$$

where l is for all nodes in the layers *above* node j.

(e) For the case of only one hidden layer, compute the error of each hidden unit activation b_j relative to each $\delta_j$with the equation

$$e_i = b_i \left(1 - b_i\right) \sum_{j=1}^{J} w_{ij} \delta_j \tag{2.9}$$

for all hidden units $i = 1, 2, \ldots, I$ where e_i is the i'th node error of I hidden nodes in the middle (second) layer.

(f) Adjust the weights connecting nodes from the hidden layer to nodes in the output layer using the equation

$$\Delta w_{ij} = \alpha b_i \delta_j \tag{2.10}$$

for all $i = 1, 2, \ldots, I$, and all $j = 1, 2, \ldots J$, where Δw_{ij} is the adjustment made to the connection from the i'th node in the hidden layer to the j'th node in the output layer and α is a positive constant which controls the learning rate.

(g) The bias terms Γ in the output nodes are adjusted in a similar manner:

$$\Delta\Gamma_j = \alpha\delta_j \tag{2.11}$$

for all $j = 1, 2, \ldots, q$, where $\Delta\Gamma_j$ is the amount of change to the j'th node's bias value.

(h) The weights connecting the nodes in the input layer to those in the hidden layer are adjusted in a similar way to that of step (f):

$$\Delta v_{hi} = \beta a_h e_i \tag{2.12}$$

for all $h = 1, 2, \ldots, n$, and all $i = 1, 2, \ldots, p$, where δv_{hi} is the amount of change made to the connection from the h'th node in the input layer to the i'th node in the hidden layer, and β is a positive constant controlling the learning rate.

(i) Adjust the bias terms in the hidden layer nodes:

$$\Delta\theta_i = \beta e_i \tag{2.13}$$

for all $i = 1, 2, \ldots, n$, where $\Delta\theta_i$ is the adjustment to the i'th output node's bias value.

3. Repeat step 2 until the error between the desired and computed outputs for each pattern is below a preset small threshold.

For a more theoretical, as opposed to pragmatic, explanation of the workings of the back-propagation algorithm, see Rumelhart [29] and Lippman [16].

2.2 Recognition

In the recognition phase we simply use the formulae quoted above on presentation of a new pattern. Given inputs in nodes h, $(1 \leq h \leq H)$:

1. Compute the hidden layer activations, b_i:

$$b_i = f\left(\sum_{h=1}^{H} a_h v_{hi} + \theta_i\right) \tag{2.14}$$

for all $i = 1, 2, \ldots, I$, where b_i is the activation value and θ_i is the bias value of the i'th hidden node. The activation function is signified by $f()$. and then

2. Compute the output layer activations, c_j:

$$c_j = f\left(\sum_{i=1}^{n} b_i w_{ij} + \Gamma_i\right) \tag{2.15}$$

for all $j = 1, 2, \ldots, p$, where c_i is the activation value and θ_i is the bias value of the j'th output node. The node with high activations are then used to classify the pattern.

2.2.1 Choosing the learning rate α and the momentum μ

Convergence is sometimes faster if a momentum term μ is added and weight changes are smoothed by:

$$\Delta w_{ij} = \mu(\delta_j o_j) + \alpha \Delta w_{ij}(n) \tag{2.16}$$

where n is the pattern presentation iteration.

2.3 How many nodes ?

In designing a network architecture, we need to consider several attributes. In particular, with small amounts of data, the network achitecture design and its training process must be thought out carefully. The key consideration is the number of parameters of the network that need to be estimated. If we have a network that is too large then we can be in a position of *overfitting* the data. For example, if we have more parameters than observations, then it may be possible to *learn* the training data very well, by incorporating an equation for every possible target value, but offering very poor categorization on test data. Conversely, if the network is too small then there is less scope for learning any inherent functions in the data set. Currently there are no precise recipes on how to design the architecture, given a data set. However, some people have proposed statistical arguments which imply that the amount of data required (Denker et al, [5]; Baum & Hausler [3]) is proportional to the number of weights in the network. Another rule of thumb which is often cited is that the amount of weights should be less than one tenth of the number of observations.

Along with the question of how many nodes, comes the question of how many layers. Kolmogorov presents a theorem which states that any continuous function of N variables can be computed with only linear summations and non-linear but increasingly functions of only one variable, described in Lorentz [18]. The theorem applied to neural networks states that a 3-layer perceptron with $N(2N+1)$ nodes using continuously increasing nonlinear activation functions can compute any continuous function of N variables. Even though the theorem states that there is no need to use more than one hidden layer, we still do not know how to select weights, activation functions and nonlinearities in the network.

2.4 Summary

The backpropagation training algorithm has been presented. While the algorithm improves the performance of the network with successive iterations, there is no clear recipe for the design of the actual network. Furthermore, training data requirements become very large as the network becomes larger.

With the use of gradient descent techniques employed in the backpropagation algorithm, there is a chance of being stuck in a local minima in the least mean squares cost function instead of a global minimum. Despite this effect, simulated annealing approaches do not give significant improvements in performance in the physical sciences (Paul et al [23]). An argument to explain this anomaly betwen theory and performance is that perhaps a global optimum does not give maximum performance. Perhaps features are being detected which are irrelevant to the problem are being learned: the telephone line characteristics in the speech recognition problem for example. Methods for surmounting the local minimum problem include: (a) increasing number of hidden units; (b) decreasing the learning rate; and (c) restarting the algorthm with different training weights.

Chapter 3

Neural Networks for Stock Price Prediction

While nonlinear domain pattern matching tasks have been accomplished across many applications, there has been relatively little work on using neural nets to forecast time series relative to the categorization problem. Examples of forecasting applications include Lapedes & Farber, [15] who also applied neural networks to decoding genetic protein sequences [14], and demonstrated that neural networks show some capability for decoding deterministic chaos.

In the case of time series such as stock prices, sunspot activity, etc., there is a tendency to think that there is insufficient data to use neural networks. As a rule of thumb, it is often said that the number of weights in a network should be less than on tenth of the number of observations. If we have only 1000 weekly observations, then this severely limits our network design. Furthermore, there may be econometric arguments which state that not all time series are relevant owing to heteroscedacity phenomena in the data, so that 1930's data for example may not be relevant.

In this chapter we present some original experiments for stock price prediction. A neural network prediction of AT&T stock returns is compared with the performance of a multiple regression model.

3.1 The problem of stock price prediction

Is it possible to predict stock prices ? A popular theory known as the capital asset pricing model states that markets are efficient and that it is not possible to make excess returns from investing in the stockmarket without accepting a risk premium. The capital asset pricing model is an application of the *efficient markets hypothesis* which asserts that stock prices follow a *random walk* [19]. Essentially, this states that stock prices are completely unpredictable, from past prices only, save for an expected appreciation equivalent to the risk free rate plus a risk premium. While non-financial markets are considered inefficient, financial markets are considered more efficient. The arguments for this is that the following enviromental factors are at work:

- Markets are are organized and regulated
- Low transactions costs
- Many partcipants with information
- Short positions are allowed
- Arbitrage relations exist

Recently there has been some evidence against the random walk hypothesis which involves computing variance ratios of multi-period returns [17]. Intuitively we can argue other reasons why the theory may not be correct:

- If markets are truly efficient, who then bothers to trade ?
- Bounded rationality arguments [30]: people are not superintelligent beings able to process all information instantaneously
- Concepts of relative efficiency at work: weak, semi-strong and strongly efficient markets which vary in degrees of information (insider, public, and pure time series) accounted for
- Nonfrictionless markets: transaction costs exist

In any case a key result of the efficient markets hypothesis is that the ability to earn excess returns from the stockmarket does not preclude the innovation in the financial markets, such as, for example, application of neural networks. However, the efficient markets hypothesis would predict that as new technologies diffuse into the marketplace, it would then be more difficult to earn excess returns.

3.2 Previous work in neural nets for stock price prediction

The best known work for the applicability of neural networks in time series modelling is that of Lapedes & Farber [14] who achieved some success in predicting a deterministic chaotic function. White [34] demonstrated some initial results for the prediction of the IBM stock price which did not disprove the efficient markets hypothesis. A network with 5 input units, 5 hidden units and one output unit was trained on 500 days of data of the IBM stock price, and tested on pre- and post- samples of data consisting of 500 days data. Kimoto et al [11] used a set of multiple networks using inputs of market turnover, stock price time series, foreign exchange rates and interest rates with one output node learning the effect of each variable on the buy and sell timing (i.e. a 1 or 0 binary teaching input). Kamijo et al [10] used *recurrent networks* (where the output values are fed back into some of the input nodes) to detect technical analysis style *triangles* in the Japanese stockmarket, using stock prices with corresponding high and low values for the time period.

3.3 An experiment in predicting stock prices with neural nets

In this section, experiments in predicting stock prices are presented. Two experiments are described:

- Prediction of actual ATT returns using the past values of IBM, Kodak and ATT returns
- Learning how to make daily trade decisions (i.e. buy or sell each day) of AT&T stock based on simple returns of past values of IBM, Kodak and ATT returns

In both cases a 3 layer multilayer perceptron architecture is used, trained using the back-propagation algorithm. The input nodes have linear activation functions while the hidden nodes have sigmoidal activation functions. Output nodes have linear and sigmoidal activation functions depending on whether the network is being used to actually predict (use linear activations) or to trade (use sigmoidal). The neural networks in each experiment are compared with the performance of a multiple regression model.

The inputs to the net are the previous time series values of returns. In security price analysis simple and continuous returns are common inputs in forecasting models. If the price at time t is P_t then the return R_t over a period τ, is given by:

$$\frac{P_{t+\tau}}{P_t} = \frac{P_{t+1}}{P_t}.\frac{P_{t+2}}{P_t}\cdots\frac{P_{t+\tau}}{P_{t+\tau-1}} \tag{3.1}$$

The *simple return* is then computed by:

$$1 + R_t(\tau) = (1 + R_{t+1}).(1 + R_{t+2})\ldots(1 + R_{t+\tau}) \tag{3.2}$$

To compute *continously compounded returns* X_t, logarithms are taken so that:

$$X_{t+1} = \log\frac{P_{t+1}}{P_t} \tag{3.3}$$

All inputs are scaled so that the inputs are in the $0 - 1$ section. Transaction costs are ignored throughout.

3.4 Prediction of AT&T stock price

In this example, past simple returns of IBM, Kodak and ATT stock are used to predict the AT&T return for the next day. The motivation for using these inputs are that there may be lead-lag effects occurring in the relationships of these stock prices. The neural network was trained on daily observations from 1/2/85 to 3/2/88 and tested on this set and a hold-out set of 450 observations from 3/3/88 to 12/11/89. Each daily observation consisted of the day's simple return for AT&T's stock as a teaching target and the previous 5 days of simple daily returns of AT&T, IBM, Eastman Kodak, and the Value Weighted CRSP [1] index, making a total of 20 inputs. The network was trained on the training set for 10,000 iterations.

The actual error is shown in Figure 3-1. The mean square error for the training set was 0.000333 for the training data and 0.000211 for the hold out sample, with autocorrelation values for the residuals being −0.011 and −0.045 respectively. A multiple regression analysis was also performed on the same data; the residual plot is shown in Figure 3-2. The mean square error for this analysis was 0.000359 for the training portion and 0.000204 for the hold out sample, with an autocorrelation value of residuals equal to 0.003 and −0.074 respectively. In both the neural network and multiple regression analyses, the mean square error calculations have been computed by converting the normalized teaching data and predictions back to actual values.

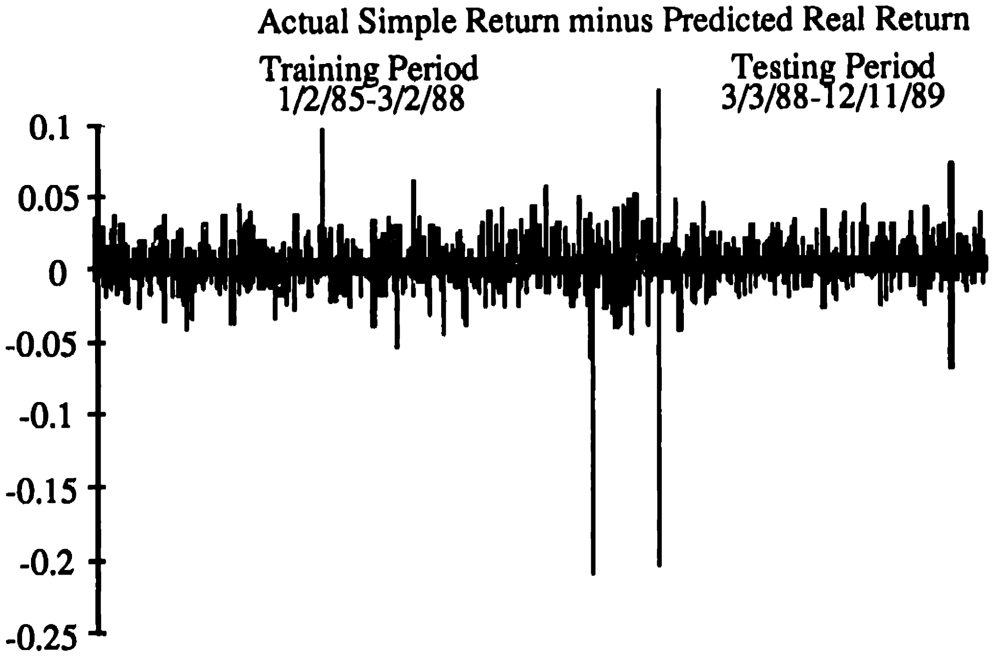


Figure 3-1: Actual daily errors between neural network prediction and actual value of simple daily return of AT&T stock

To test the trading ability of the predictive neural network model, the performance was compared with a buy and hold strategy, that is buy $1 worth of AT&T stock at the start and

---

[1] Centre for Research on Security Prices

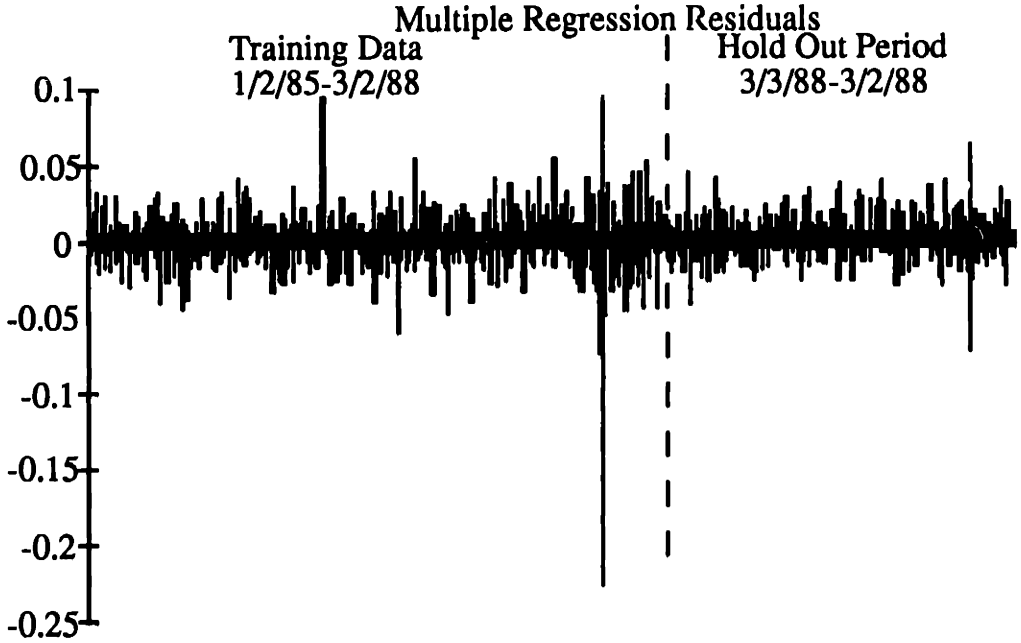


Figure 3-2: Actual daily errors between multiple regression prediction and actual value of simple daily returns of AT&T stock

hold on to it. The neural network was allowed to trade each day and was allowed to be in the market completely or out of the market completely, with no half measures. If the neural network had predicted the market to go down, i.e. return less than 0, then the model would not hold AT&T stock. Conversely, if the model predicted the day's return to be greater than 0, then the model would hold AT&T stock.

The result is shown in Figure 3-3. An interesting point is that the network timed and traded perfectly the crash of 1987, by selling before the crash and buying after the crash. However, the network fails to trade well during the hold out portion of the data.

If we compare the neural network's trading ability with a conventional multiple regression estimator, we find that although, the regression estimator fails to trade well during the crash, it it trades better at other times. Figure 3-4 shows that a multiple regression model would have obtained a return of 518% over the entire period, with a return of 198% over the training period, and 107% over the testing period. The return figures for the buy and hold strategy

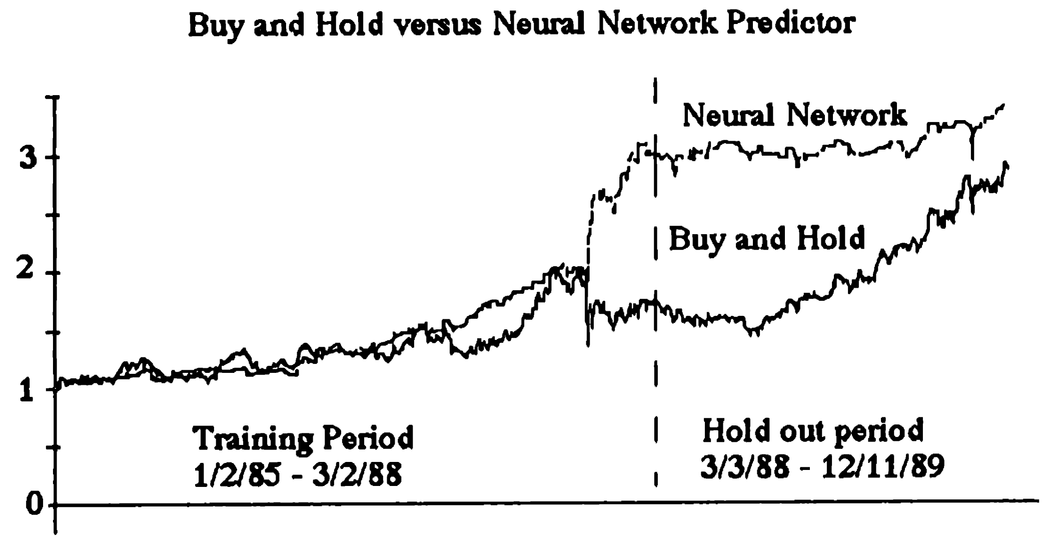


Figure 3-3: Comparison of buy and hold and neural network prediction trading strategies

would have been 186% overall, 65% training, and 72% testing.

3.5 Using a neural network to learn how to trade

In this experiment, rather than presenting the returns of AT&T stock as target values to the outputs of the neural network, decisions of how to trade are presented. For each day's parameter vector, which is the same 20 parameter vector, the target value is 1 if the model should be in the market for AT&T, that is, it will achieve a positive return for that day, and 0 if it should be out of the market, that is it would have achieved negative return that day. No interest is paid on *out of the market* holdings and no transaction costs are incurred, as before. The network was *bootstrapped* with the weights of the previous prediction of returns experiment and trained for 3800 iterations. Note that the network was the same as that used for the prediction experiment except that the output node activations were linear instead of sigmoidal.

The result of trading with this network is shown in Figure 3-5. The return over the entire period from 1/2/85 to 12/11/89 amounted to 516% compared with 186% in buy and hold case. Over just the training period, the return amounted to 201% versus 65% in the buy and hold case. Over the testing period, the return amounted to 104% compared to 72% in the buy

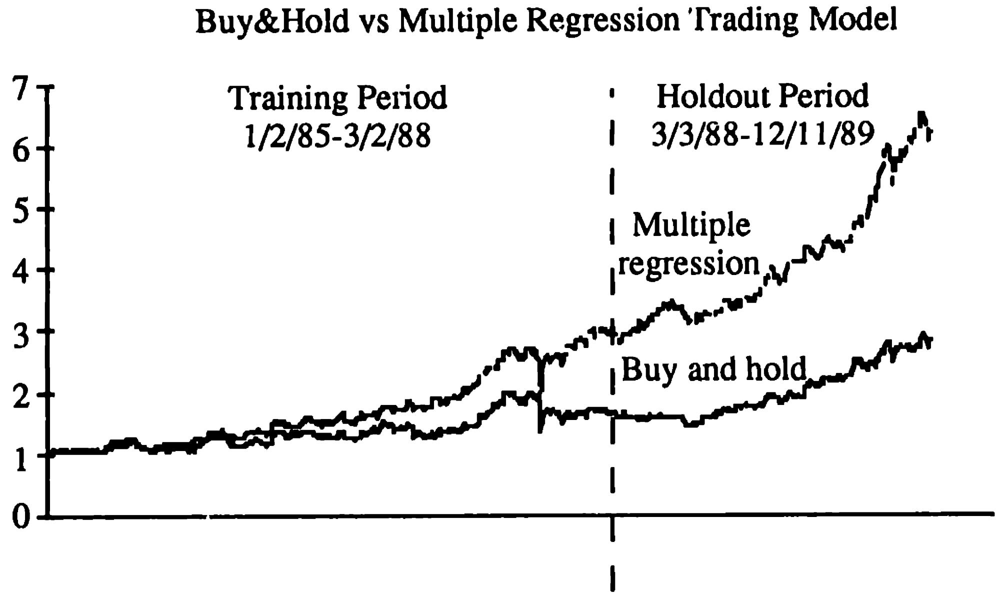


Figure 3-4: Comparison of buy and hold and multiple regression trading strategies

and hold case. The neural network made the correct decision correctly 61% of the time in the training period and 59.5% in the testing period, with the average hold being around 5 days. The maximum possible return with perfect timing would have been 3718%.

3.6 Summary of effectiveness in trading applications

In the example presented here neural networks do not appear to be as good as a conventional multiple regression model in terms of prediction of the actual simple return values. Although the mean square error values for the network are less than those for multiple regression in the training set, they are nevertheless similar, and the residual autocorrelations for the neural network are higher for both the training and hold out sets. The neural network fared reasonably well with a buy and hold strategy during the training period but was worse in performance over the holdout period. The neural network performed well on the crash of October 1987, selling before the crash and buying afterwards - substantially better than the multiple regression model - but remember that this occurrence was during the training period.

The neural network performed much better when asked to perform actual trading deci-

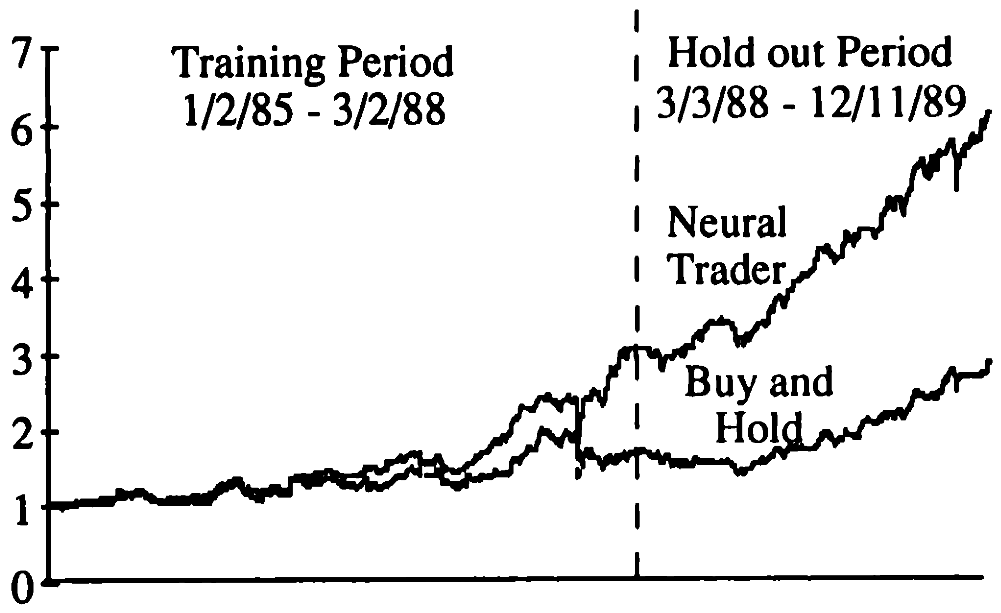


Figure 3-5: Comparison of buy and hold and neural network trading strategies

sions. However, the return was not significantly better than the multiple regression model with predicted values regressed against the 20 variables. The network was able to provide a return significantly larger than a buy and hold strategy, namely 104% against 72% in the tesing period and 201% versus 65% in the training period. The market timing decisions were still not perfect and fell far short of perfect timing.

Possible reasons why the network failed to perform well on the prediction task could be:

1. Insufficient data: 800 observations to train 220 parameters may be too small
2. Inadequate topology: even though a 3 layer neural network can in theory learn any arbitrary function (Lorentz [18]), we still have to know the activation function types to use - a network with more layers may be easier to train - linear activation functions at the output may not be optimum for a forecasting task.
3. Incorrect data inputs: maybe the data inputs used in the experiment are not appropriate for training a neural net, for example continuously compounded returns may be more useful

We must also remember that the neural network model employed was perhaps the simplest type of network that one could use. A textbook implementation of the algorithm was employed, with no embellishments. Further enhancements and more research into the kind of data inputs that may be useful will determine the eventual usefulness of a neural network. Previous results in the field (Kimura [12]) have presented neural network results, but have not explicitly compared the results with other models. This work shows that although a neural network may be useful after experimentation with different inputs and architectures, a simple implementation will perhaps only be as good as a multiple regression model.

Chapter 4

Other neural network applications in finance

In contrast to the management science literature, the literature of neural networks contains a few examples of applications of neural networks in finance and management science. The first demonstration of a potentially useful application was an application to the *travelling salesman problem* (Hopfield and Tank, [8]). Initially, the result showed that although neural networks were capble of solving this problem, results were not as good as more traditional techniques for cases with more than 30 cities. In recent times, newer architectures and training algorithms have been advanced which solve this problem for upto 200 cities with superior performance over the tradtional method of simulated annealing (Peterson, [24]). In this chapter, three applications are presented:

- Bankruptcy prediction
- Bond rating prediction
- Testing the arbitrage pricing model

These applications revolve around the pattern matching ability of neural networks rather than time series modelling.

4.1 Application to bankruptcy prediction

Some work has been undertaken in applying a simple 3 layer (5 inputs, 5 hidden nodes and one output node) neural network to the problem of bankruptcy prediction [21]. The 5 input variables were: (i) Working capital/total assets; (ii) retained earnings/total assets; (iii) earnings before interest and taxes/ total assets; (iv) market value of equity/total debt; (v) sales/total assets. The neural network method was found to outperform the usual method of discriminant analysis [2].

4.2 Application to bond rating prediction

The default risk of a bond relates to the chance that the promised coupon and principal vaulues will not be paid. Independent organizations like Standard & Poors and Moody's rate actively traded bonds for a fee paid by the bond issuer. Typically issuers are willing to do this because it adds credibility to the bond issue. A rating of AAA would be applied to a high rating bond where both the capital and the interest have high probability of payemnt. A rating of BBB might be used for the a bond where the company is currently able to pay interest and principal but may have weakened capacity to pay in adverse economic conditions. Investors use these ratings to reflect how much they should pay for a bond.

It is not entirely clear how rating agencies rate bonds. Typically a committee analysis of the bond may occur to evaluate attributes such as:

- ability to repay
- willingness to repay
- protective provisions for an issue

Whilst protective provisions and ability to repay may be quantifiable, willingness to repay may be difficult to attach quantitative attributes to. Hence it is difficult to obtain a mathematical model which is capable of predicting the rating of a bond. Similarly developing an expert system to undertake the solution of this problem is likely to be a formidable task. Some reseracher have used multiple regression and factor analysis techniques to predict bond rating categories — A prediction of 67% was obtained by Pinches & Mingo [25] for hold-out samples in 1967-68.

Duttar & Shekhar [6] proposed a neural network formalism which acted as a category classifier, where the categories relate to the different bond rating categories. In fact only two categories were used : is the bond of category AA; or is it of another category. As such the bond ratings were scaled linearly to convert the ratings of the bonds. One ouput unit was used with two or three hidden layer multilayer perceptrons trained with the standard backpropagation algorithm. Based on the results of Pinches [25], between 6 and 10 input variables were input into the network. The data included bond ratings and values of the 10 variables taken from the April 1986 issues of the ValueLine Index and the S&P Bond Guide.

The neural network results were found to give a prediction error an order of magnitude lower than that for multiregression, with the training data. Furthermore, the prediction accuracy on test data was found to be 88.3% compared to 64.7% in the regression model. While more hidden layer gave better prediction error results on the training data, there was no difference with test data. This demonstrates the trade-off of losing *generalisation* properties as we increase the number of hidden layers.

4.3 Application to arbitrage pricing model testing

The arbitage pricing model [27] is a generalisation of the capital asset pricing model and states that the expected return E_i of a stock i is given by;

$$E_i = \lambda_0 + \lambda_1 b_{i1} + \lambda_2 b_{i2} + \ldots + \lambda_K b_{iK} \tag{4.1}$$

where b_{ij} are the factor sensitivities of the return E_i on the i'th asset to the j'th factor; and λ_K are coefficients relating to factor risk premiums.

Problems of the Arbitrage pricing theory arise from the difficulty of finding the factors to incorporate into the model. Typically this undertaken by the use of factor analysis. However, in any group of securities in which factor analysis is undertaken, the factors may be different from those found in other groups in both number and kind. Ahmadi [1] reported a method using a variety of inputs that are usually used in the APT model as inputs into a neural network. The method describes a technique of inspecting what the weights of the network actually corresponded to in physical terms. However, although the method was described, no results were published.

Chapter 5

Discussion and Conclusion

This thesis has presented the basic theory of neural networks. The basic model consists of many non-linear computational nodes interconnected with each other, usually in the form of parallel feedforward layers. The application of neural networks in various areas of finance and management science has been reviewed and a simple experiment of an application to stock market prediction has been presented. The results of the experiment show that although the network had some forecasting value, it was not in fact as significantly better than a traditional multiple regression model for the particular stock and the inputs used in the experiment. Nevertheless, the experiment was a simple one using a very basic neural network model. The result is sufficiently good that it does not preclude a more enhanced model being more effective.

In this chapter several possible improvements are suggested which may affect the performance of a neural network. In particular: the problem of over learning is addressed; the problem of time series prediction vs categorization is discussed by introducing a recurrent neural network as a better way for time series forecasting; genetic algorithms for obtaining better networks which work in combination are presented; and the problems of computer limitations of implementation.

5.1 Surmounting the overfitting problem

As explained in the previous chapters, the network design has a significant effect on the learning characterisitics. If the network is too small, then it is difficult for the network to have sufficient parameters to learn functions lying within the data. If the network is too large then too many free parameters will allow the network to learn the training data set too closely with little prediction and generalization capabilities.

There are two main methods for overcoming the overfitting problem. One concerns using a separate validation set, during training. This validation set is used as a test set during training. Once performance on the validation set degrades, this is taken as a sign that the network is overfitting to the training data, and training on the current training set is suspended. Another method concerns the gradual elimination of weights so that there are fewer parameters for the training data to fit too, as the network learns. Both these methods are well described by Weigend et al [32].

If the network has a number of weights which equates to the order of the number of training observations, then the network is referred to as *oversized.* The objective of weight elimination is to obtain a *minimal* network which is capable of *generalizing* the input data whilst not overfitting it. Essentially the same basic backpropagation training algorithm is employed.The one exception is that the error function is made a little more sophisticated so as to penalize low value weights by allocating a cost to each each connection when computing the error function.

5.2 Categorization vs time series forecasting

Many of the neural network applications in the physical sciences have been related to problems of categorization or pattern matching rather than time series forecasting. The success of neural networks in finance has also been more prevalent in categorization applications such as the credit verification problem, the bond rating task and the bankruptcy prediction task. In contrast applications in time series forecasting are less prevalent under the standard feedforward architecture. An alternative architecture which may be more relevant to time series problems is that of the recurrent neural network. In this type of network, output values from nodes are fed back into a set of nodes in the input layer. The basic argument is then that there is some time series *memory* in the network by virtue of some of the hidden nodes having inputs from the previous time period. An example of a recurrent network is shown in Figure 5-1.

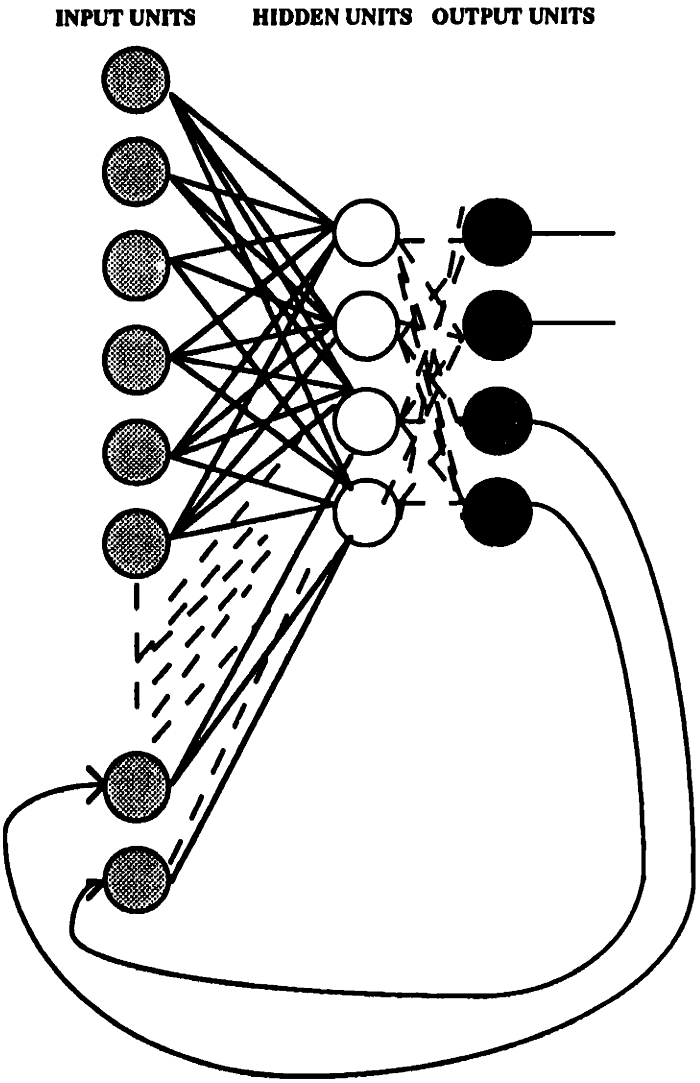


Figure 5-1: Example of a recurrent neural network

5.3 Genetic algorithms

Genetic algorithms (Goldberg [7]) have recently been proposed as a method of selecting which networks are good, bad and which to use in combination. The basic argument is that we start off with a number of network architectures and train and test all the networks in parallel. As some networks become weaker in performance, they are discarded, while the reamining networks *breed* to form new networks. Networks continue to be bred and discarded until a desired performance has been reached.

5.4 Implementation considerations

All of the algorithms and experiments presented in this thesis have been implemented in C on a Texas Instruments TMS320c30 Digital Signal Signal Processor Card installed in a IBM-compatible 386 PC. The resulting speed performance at 33 MFLOPS (million floating point instructions per second) is perhaps some 50 times faster than a regular PC. While the cost of such hardware is not prohibitive (say $5000 plus price of a PC), some form of accelerating hardware is required to undertake experiments in a productive manner. It takes between 6 and 48 hours to train a neural network on digital signal processor based system. One is not likely to be able to undertake experiments on an unaccelerated PC in sufficient quantity to be able to determine useful conclusions.

Another disadvantage of the neural network approach is that it is difficult to see how the network is making its decision. This is in sharp contrast to expert system based credit authorization approaches which place great emphasis on the telephone clerk being able to rationalize from the computer readout why a fraud case may be possible for a particular transaction. Although it is possible to inspect the weights linking the nodes, in practice it is not straightforward to trace back rationale from a neural network.

Bibliography

[1] H. Ahmadi. Testability of the arbitrage pricing theory by neural network. In *IEEE Joint Conference on Neural Networks*, San Diego, June 1990.

[2] E.I. Altman. Financial ratios, discriminant analysis and the prediction of corporate bankruptcy. *Journal of Finance*, September 1968.

[3] E.B. Baum and D. Hausler. What size net gives valid generalization. *Neural Computation*, 1:151, 1989.

[4] F. Crick. Neural networks. *Nature*, June 1988.

[5] J.S. Denker. Large automatic learining, rule extraction and generalization. *Complex Systems*, 1:877, 1987.

[6] S. Dutta and S. Shekhar. Bond rating: A non-conservative application of neural networks. In *International Joint Cnference on Neural Networks*, pages 443–450, 1987.

[7] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1988.

[8] J.J. Hopfield and D.W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52(141), 1985.

[9] J.J. Hopfield and D.W. Tank. Computing with neural circuits: A model. *Science*, 233:625–633, August 1986.

[10] K. Kamijo and T. Tanigawa. Stock price pattern prediction — a recurrent neural network approach. In *IEEE-INNS International Joint Conference on Neural Networks*, pages 1.215–1.1.221, San Diego, June 1990.

[11] T. Kimoto, K. Asakawa, M. Yoda, and M. Takeoka. Stock market prediction with modular neural networks. In *IEEE-INNS International Joint Conference on Neural Networks*, pages 1.1 -1.6, San Diego, June 1990.

[12] D. Kimura. Cerebal dominance and the perception of verbal stimuli. *Can. Jnl. Psychology*, 15:166–171, 1961.

[13] T. Kohonen, K. Makisara, and T. Saramaki. Phonotopic maps—insightful representation of phonological features for speech recognition. In *IEEE Proc. International Conference on Pattern Recognition*, page 184, Montreal, 1984.

[14] A. Lapedes and R. Farber. Genetic data base analysis with neural nets. *IEEE Conference on Neural Information Processing Systems - Natural and Synthetic*, 1987.

[15] A. Lapedes and R. Farber. Nonlinear signal processing using neural networks. *IEEE Conference on Neural Information Processing Systems - Natural and Synthetic*, 1987.

[16] R. Lippman. An introduction to neural nets. *ASSP magazine*, 4(2):4–22, April 1987.

[17] A.W. Lo and A.C. MacKinlay. Stock market prices do not follow random walks: Evidence from a simple specification test. *The Review of Financial Studies*, 1:41-66, 1988.

[18] G.G. Lorentz. American Mathematical Society,, R.I., 1976.

[19] R. Malkiel. *A Random Walk down Wall Street.* Norton, New York, 1985.

[20] M. Minsky and Pappert. *Perceptrons.* MIT Press, Boston, 1969.

[21] M.D. Odom and R. Sharda. A neural network model for bankruptcy prediction. In *IEEE Joint Conference on Neural Networks*, San Diego, June 1990.

[22] D.B. Parker. Learning logic. *MIT Sloan School of Management Center for Computational Research in Economics and Management Science, Working Paper TR-47*, 1985.

[23] D. Paul. Training of hmm recognizers by simulated annealing. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 13–16, Tampa, Florida, 1985.

[24] C. Peterson. Parallel distributed approaches to combinatorial optimization: Benchmark studies on travelling salesman problem. *Neural Computation*, 2(3), Fall 1990.

[25] G.E. Pinches and K.A. Mingo. A multivariate analysis of industrial bond ratings. *Journal of Finance*, March 1977.

[26] M. Rappa. Analyzing the diffusion of neural network technology with bibliographic techniques. *Sloan School of Management Working Paper*, June 1988.

[27] R. Roll and S. Ross. An empirical investigation of the arbitrage pricing theory. *Journal of Finance*, 35:1073–1103, December 1980.

[28] R. Rosenblatt. *Principles of Neurodynamics.* Spartan Books, New York, 1959.

[29] D. Rumelhart, G.E. Hinton, and R.J. Williams. *Learning Internal Representations by Error Propagation.* MIT Press, 1985.

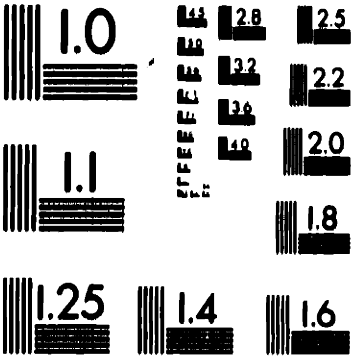
[30] H Simon. A behavioral model of rational choice. *Quarterly Journal of Economics*, 69:99–118, 1955.

[31] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang. Phoneme recognition using time-delay neural networks. *ATR Research Report TR-10006*, No. 7, October 1987.

[32] A.S. Weigend, B.A. Huberman, and D.E. Rumelhart. Predicting the future: A connectionist approach. *International Journal of Neural Systems*, 1:193–209, 1990.

[33] P. Werbos. *Beyond Regression.* PhD thesis, Harvard University, 1975.

[34] H. White. Economic prediction using neural networks: The case of ibm daily stock returns. In *IEEE International Joint Conference on Neural Networks*, pages 451–458, 1988.



24·1