Pergamon

Neural Networks, Vol 7, No. 2, pp. 375–388, 1994

0893-6080/94 $6 00 + 00

CONTRIBUTED ARTICLE

Stock Performance Modeling Using Neural Networks: A Comparative Study With Regression Models

APOSTOLOS NICHOLAS REFENES,[1] ACHILEAS ZAPRANIS,[1] AND GAVIN FRANCIS[2]

[1]University College London and [2]County NatWest Investment Management

(*Received* 21 *December* 1992; *accepted* 17 *September* 1993)

Abstract—*We examine the use of neural networks as an alternative to classical statistical techniques for forecasting within the framework of the APT (arbitrage pricing theory) model for stock ranking We show that neural networks outperform these statistical techniques in forecasting accuracy terms, and give better model fitness in-sample by one order of magnitude We identify intervals for the network parameter values for which these performance figures are statistically stable Neural networks have been criticised for not being able to provide an explanation of how they interact with their environment and how they reach an outcome We show that by using sensitivity analysis, neural networks can provide a reasonable explanation of their predictive behaviour and can model their environment more convincingly than regression models*

Keywords—Neural networks, Multiple linear regression, Arbitrage pricing theory, Stock market modeling, Sensitivity analysis, Parameter significance estimation.

1. INTRODUCTION

A great deal of effort has been devoted to developing systems for predicting stock returns in the capital markets. Limited success has been achieved. It is believed that the main reason for this is that the structural relationship between an asset price and its determinants changes over time. These changes can be abrupt. For example, one month a rise in interest rates will strengthen sterling, whilst the next month a rise will weaken sterling. This phenomenon of unstable structural parameters in asset price models is a special case of a general fundamental critique of econometric and statistical models. A relationship might be established, for example, between consumer spending and personal income. A tax cut could then be analysed via its effect on personal income. Critics, however, assert that this cannot be done, because a change in policy (the tax cut) will not only change the level of income, but will change the relationship between spending and income.

Neural network architectures have drawn considerable attention in recent years because of their interesting learning abilities. They are capable of dealing with the problem of structural instability. Several researchers have reported exceptional results with the use of neural networks (Dutta & Shashi, 1988; Refenes, 1991; Refenes & Zaidi, 1992; Schoenenburg, 1990; White, 1988; Refenes & Azema-Barac, 1993). Neural networks are generally believed to be an effective modeling procedure when the mapping from the input to the output vector space contains both regularities and exceptions. They are, in principle, capable of solving any nonlinear classification problem, provided that the network contains a sufficiently large number of free parameters (i.e., hidden units and/or connections) (Hinton, 1987).

In this paper we investigate the performance of neural networks in the nontrivial application of stock performance modeling. The problem consists of a universe of stocks whose returns are linked to three factors. The idea is to model how the structural relationship between a stock's return and its determinants changes over time. The model can then be used in three ways:

- to predict the relative outperformance of each stock in the universe, 6 months in advance given the current values of its determinant factors.
- to test the hypothesis that excess returns required by investors for factor exposures, change *slowly* over time as the economic environment evolves, and that a period of 6 months is a reasonable time frame for dynamic remodeling.
- to analyse the factors that determine stock performance and to identify how the relative significance of these factors changes over time.

The whole process is part of a dynamic version of the APT model (arbitrage pricing theory). Currently,

Requests for reprints should be sent to Apostolos Nicholas Refenes, Department of Decision Science, London Business School, Sussex Place, Regents Park, London NW1 4SA, England

the modeling is done by linear regression. Our target is to outperform linear regression with respect to three performance metrics:

1. goodness of fit *in-sample* (convergence);
2. goodness of fit *out-of-sample* (generalisation);
3. stability of results with varying network parameters and different data sets.

We show that neural networks give better model fitness in-sample by one order of magnitude and outperform linear regression in out-of-sample forecasting. We identify intervals of values for the parameters that influence network performance over which the results are stable, and show that the same performance figures persist across different training/test sets. We show that by using sensitivity analysis, neural networks can provide a reasonable explanation of their predictive behaviour and can model their environment more convincingly than regression models.

In Section 2 of this paper we give a brief overview of the stock performance modeling application. In Section 3 we discuss the neural network setup. We also define the metrics for evaluating the convergence and generalisation ability of different network configurations and compare with multiple linear regression. In Section 4 we discuss our results using multiple linear regression (MLR) as a benchmark. In Section 5 we perform sensitivity analysis on the model and discuss its use as a stock modeling tool. Finally, in Section 6 we give some brief concluding remarks.

2. A DYNAMIC MULTIFACTOR MODEL OF STOCK RETURNS

2.1. The Arbitrage Pricing Theory

The APT (Ross & Ross, 1980) is widely used in portfolio management as an alternative to the capital asset pricing model (CAPM) (Sharpe, 1964). APT has the benefit of being a more powerful theory; it requires less stringent assumptions than the CAPM, yet it produces similar results. The difficulty with the APT is that it shows that *there is* a way to forecast expected asset returns but it does not specify *how* to do it.

The key idea of the theory is that there exist a *set of factors* such that *expected returns* can be explained as a linear combination of each asset's exposure to these factors.

The APT is based on a *no-arbitrage* assumption that can be stated as requiring an upper limit on the ratio of the *expected-excess-return* so that any risk investment divided by the *volatility* of that same investment is bounded. If this ratio was not bounded then it would be possible to get positive expected-excess-returns for very low levels of risk. With the no-arbitrage assumption there always exists a portfolio that we call portfolio Q, which is efficient and has the highest ratio of expected-excess-return to volatility. In this case any asset's (or portfolio P's) expected-excess-return will be proportional to their covariance with the portfolio Q. For portfolio P we have:

$$E[r(P)] = \gamma(n)E[r(Q)] \tag{1}$$

where $\gamma(n)$ is the "beta" of the asset with respect to the portfolio Q, $E[r(P)]$ is the expected-excess-return of the portfolio P, and $E[r(Q)]$ is the expected-excess-return of the portfolio Q.

The advantage of this is that it does not require any special assumptions to arrive at the result. The disadvantage is that we do not know the portfolio Q. The effort to uncover the portfolio Q usually takes the form of searching for *attributes* of Q rather than the actual portfolio holdings. The idea is that the particular portfolio Q will depend on the universe of assets we are considering and the properties of those assets at that time. The main proposition of the APT model is that:

PROPOSITION 1. *Stock returns can be explained in terms of a set of factors Traditionally, it has been assumed that the return is a linear combination of each stock's exposure to these factors, as shown in eqn* (2).

$$R_i = \alpha_i + \sum_{j=1}^{n} f_{ij}\beta_j + \varepsilon_i \tag{2}$$

where R_i is the return on stock i

- α_i is a constant for stock i
- f_{ij} is the exposure of stock i to factor j
- β_j is the return attributable to factor j
- ε_i is a random error with mean zero.

Using data on a universe of stocks, it is possible to estimate a model of the following form for the expected return on stock i, where λ_j is the estimate of β_j:

$$\bar{R}_i = \lambda_0 + \sum_{j=1}^{n} f_{ij}\lambda_j. \tag{3}$$

In the framework of APT, the λ's represent the excess expected return required by investors because of a security's sensitivity to the particular factor. By subtracting the mean from both sides of the equation above, one obtains the expected return of stock i above the average of all stocks:

$$\bar{R}_i - \bar{R} = \sum_{j=1}^{n} (f_{ij} - \bar{f}_j)\lambda_j \tag{4}$$

So a stock's return relative to the average is assumed to be a linear function of its relative exposure to various factors. These factors are chosen subjectively and are typically based on information derived from the balance sheet of the individual companies.

According to the ATP framework, three stages are necessary during the investment process:

1. preprocessing of data to calculate relative values for the factors involved,
2. ranking the stocks,
3. constructing the portfolio.

To rank the stocks in the universe, it is necessary to estimate the λ coefficients. However, before this can be done, the appropriate time frame has to be selected. The ATP model makes no proposition about the time frame over which λ coefficients are to be estimated, with practitioners often using static time frames. In our case, the appropriate time frame is determined by the DynIM™ model [DynIM (Dynamic Multi-Factor Model of Stock Returns) is a Trade Mark of County NatWest Investment Management Ltd.].

2.2. The DynIM Hypothesis

DynIM™ is a dynamic version of ATP. The main proposition of the model is the following:

PROPOSITION 2. *The excess returns required by investors for factor exposures tend to change slowly over time as the economic environment evolves.*

For example, small companies tend to be hit hard during a recession, but they do very well in a buoyant economy. Therefore, the DynIM™ model is designed to be dynamic in the sense that relationships are recomputed each month using sample data over the previous 6 months.

The purpose of stock ranking is the construction of a portfolio. Stock ranking is defined as the task of assigning ratings to different stocks within a universe. This is actually a classification problem: *given a set of classes and a set of input data instances, each described by a suitable set of features, assign each input data instance to one of the classes.* In our case, the different stocks form the set of input instances and the various ratings form the possible classes to which the input stocks can belong. Each stock instance can be described by a set of features which represent important financial information about the company which the stock represents.

More formally the problem statement is as follows: let S represent the space of stocks, $s_1, s_2, s_3, \ldots, s_n$, and R be the set of possible (mutually exclusive) stock ratings, $r_1, r_2, r_3, \ldots, r_m$. Let F represent the k-dimensional feature space, $f_1, f_2, f_3, \ldots, f_k$, describing each of the stocks. Each stock s_i can be considered as a k-tuple $(c_1, c_2, c_3, \ldots, c_k)$ in the Cartesian space $f_1 \times f_2 \times f_3 \times \ldots \times f_k$. Rating the stocks involves finding the one-to-one mapping function g (Dutta & Shashi 1988):

$$g \cdot f_1 \times f_2 \times f_3 \times \ldots \times f_k \rightarrow R. \tag{5}$$

In the application described we have an average of 143 stocks updated on a monthly basis. Each stock is described in a three-dimensional feature space by three factors (A, B, C) and the ranking is based on the predicted relative outperformance of each stock 6 months ahead.

So far, the DynIM™ has been applied quite successfully to UK stocks. For the purposes of estimating the λ coefficients, DynIM™ uses MLR. A natural extension to this work is to relax the first assumption of the model to allow for a more general nonlinear relationship between factors and resultant return. In this paper we show that neural networks are a superior substitute for linear regression.

3. EXPERIMENTAL SETUP

3.1. Training and Test Sets

The training and test sets consist of data provided in a preprocessed form and presented as factor A, factor B, factor C, and resultant outperformance Y. The factors A, B, and C are parameters extracted from the balance sheets of the companies in the universe of the UK stocks. Details of these factors are not specified. Table

TABLE 1
Sample Training Data

Y	A	B	C	Stock Code
−0.203553	+1.268286	−0.128681	+0.616215	6811.000000
−0.066618	−0.272814	−0.187851	−0.124382	6842.000000
+0.170599	+0.175118	+0.331097	+0.420197	6870.000000
−0.078946	−0.061965	−0.181817	+0.309313	6926.000000
+0.193520	−0.342653	+1.379686	+0.215267	8118.000000
+0.112259	+0.792553	−0.417726	+0.066999	8320.000000
−0.190763	+1.016654	+0.270754	+1.166889	8800.000000
−0.104398	−0.194282	+0.169641	−0.036239	8846.000000
−0.124752	−0.021836	+0.900211	+0.338135	9344.000000
−0.070049	−0.135540	−0 064680	+0.599856	9601.000000
−0.087188	+0.156639	+0.410285	−0.001720	9616.000000
+0.049878	+0.286707	+0.124161	−0.280837	10585.000000
−0.049638	+0.611714	+0.072714	+0.160638	10812.000000
+0.020342	+0.000000	−1.085925	−0.174620	11052.000000
+0.135470	+0.658805	+0.213187	+0.307542	11144.000000
+0.011012	+0.417831	−0.363035	−0.139603	13302.000000

1 gives an example of the training dataset we use in this application.

The rightmost column is a code for each stock in the said universe. A, B, and C are the inputs to the network with Y being the target output (i.e., the outperformance of the stock). The outperformance of the stock P in the tth month is the result of the application of an unknown function g on P_t/P_{t+6}, where P_t is the price of the stock P in the tth month, and P_{t+6} is the price of stock P after 6 months [in the $(t+6)$th month] More formally:

$$Y_t^P = g\left(\frac{P_t}{P_{t+6}}\right) \tag{6}$$

where Y_t^P stands for the outperformance of the stock P in month t.

The data set covers the period May 1985 to December 1991 on a monthly basis, and concerns 143 stocks. The overall size of the data set therefore is given by: $143 \times 12 \times 6 = 10{,}296$ training vectors.

In accordance with the DynIM™ proposition, the networks are trained on six monthly batches of data and evaluated for the next 6-month period. Thus, each training/test run consists of $143 \times 6 = 852$ training vectors. The intermediate size for each training run makes the problem nontrivial and allows for extensive tests on convergence and generalisation.

3.2. Network Setup

The learning algorithm used is the standard back propagation learning algorithm with a momentum term. All simulations run on SUN4 workstations; convergence is reached within 30,000 iterations typically requiring 3–4 days of CPU time.

The need for statistical stability in the results requires extensive experimentation with the parameters that influence network performance. Our target here is to identify intervals of values for these parameters that give statistically stable results, and to demonstrate that these results persist across different training and test sets. Below we give a list of these parameters and how they are varied in the simulations.

- *Network architecture* We examine layered, fully connected, feedforward networks. The number of neurons of the input and output layers are defined by the application and they are three for the input layer (one for each factor A, B, and C) and one for the output layer (representing the outperformance Y). The parameters with respect to network topology are the number of hidden layers and the number of neurons of each layer. The aim is to identify network topologies for which the results show small standard deviation (from a mean value higher than MLR's).
- *Gradient descent terms* The parameters here are the learning rate and the momentum term. The epoch is kept always equal to one, meaning that the weights are updated after each presentation of a training pattern. This is the on-line or *stochastic* version of back propagation, as opposed to the *batch* version where the weights are updated after the gradients have accumulated over the whole training set. Also, there isn't any offset added to the derivative of the transfer function. The objective is to find the ranges of momentum term and learning rate that yield stable performance for a given network architecture, as a function of the training time.
- *Training time* With the term training time we mean the number of presentations of the entire training set to the network (iterations). This parameter is of great importance because the effect of any network configuration on network performance must be seen as a function of the training time. The aim is to identify the largest possible intervals of network topology for which the results show small standard deviation.
- *Transfer function, cost function, and initial conditions* The transfer function is not a parameter; for all simulations we use the common asymmetric sigmoid. The cost function used for all simulations is the common quadratic error function, and we test the stability of the results with varying initial conditions.

The network architecture consists of three inputs (corresponding to A, B, and C) and one output (corresponding to Y). We experiment with several configurations of hidden units. The best configuration in terms of the trade-offs involved between convergence and generalisation and also giving a conveniently stable network is a 3-32-16-1 configuration. By the term *convenient stability* we mean a network whose generalisation performance remains stable (i.e., high mean, low standard deviation) for a wide range of values of control parameters. For example, a network whose generalisation is consistent for training times of say between 15K and 30K iterations is preferable to a network in which small variations in training times produce wide variations in outcome.

Figure 1 (right panel) shows the architecture of the 3-32-16-1 network. The notation used here will be use-

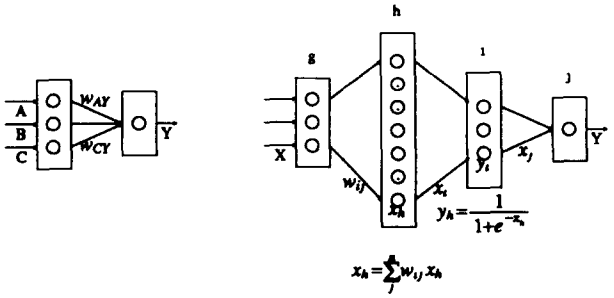

FIGURE 1. (Left) single layer network; (Right) three layer feed forward network with 3-32-16-1 connectivity.

ful for our discussion in the sensitivity analysis section. The network shown in Figure 1 (left panel) is solely depicted for the purposes of the analysis in Section 5.

3.3. Performance Metrics

It is apparent from the ATP and DynIM™ models that the main requirement on the estimator (i.e., MLR and/or the neural network) is to estimate as accurately as possible the λ coefficients; which means to fit an accurate model in the universe of stocks for the previous 6 months. The accuracy of fitness is typically measured by the mean squared error of observed against actual values for *Y in-sample* The DynIM™ hypothesis guarantees that the results of the estimator remain accurate for at least the trailing 6 months.

In theory, therefore, the only criterion on whether neural networks perform better than regression is if they converge to smaller mean squared error in-sample. In practice, however, any estimator will only *approximate* the actual structural relationship. Even if the DynIM™ hypothesis holds, the estimator will still contain an error element in its estimation. This error element can be divided in two components. The first component is due to the estimator's *bias* whilst the second component is due to the estimator's *variance*

Parametric estimators such as linear regression are *high-bias* estimators in that they assume an *a priori* model (e.g., linear relationship) but they are also *zero-variance* estimators. Neural networks, however, are analogous to nonparametric regression methods in that they make no *a priori* assumptions about the problem (i.e., let the data speak for itself); the main contributor to the global error is their *high variance.*

So, it is always possible that although neural networks may have produced better fit *in-sample* their actual estimation of the λ coefficients might be quite wrong. To avoid this problem we consider the following performance measures when comparing with MLR:

1. *convergence:* the *in-sample* performance of the network is important because it determines its convergence ability and sets a target of feasible *out-of-sample* performance that can be achieved by fine-tuning the network parameters and training discipline. The target here is to achieve better model fitness than MLR but without penalising generalisation (see 2).
2. *generalisation.* is the main property that should be sought. The aim here is to achieve (out-of-sample) generalisation performance (i.e., prediction of relative stock outperformance) that is better than that of linear regression. If the networks are capable of making better (out-of-sample) prediction, then this is the most objective metric that they have indeed managed to capture the structural relationship between a stock's performance and its determinant factors more accurately than MLR and any subsequent analysis is thus more reliable.
3. *stability:* neural networks have been known to produce wide variations in their predictive properties. This is to say that small changes in network design, learning times, initial conditions, etc., may produce large changes in network behaviour. Our target here is to identify intervals of values for these parameters that give statistically stable results, and to demonstrate that these results persist across different training and test sets.
4. *sensitivity.* the ultimate performance metric is the usefulness of the estimator as a qualitative decision-making tool in portfolio management. This involves analysing the sensitivity of a stock's exposure to changes in the values of its determinant factors, the ability to simulate "what-if " scenaria, and the ability to have a formal framework for reasoning about the model's prediction. In Section 5, we give an analytic framework for factor analysis and its practical application to this problem. We show that using this framework the network can provide reasonable explanation of its decisions.

To quantify the convergence and generalisation performance of the two methods (i.e., linear regression and neural networks) we use two metrics. The first metric is the common root mean square (RMS) error. The RMS error clearly is a measure of the correctness of prediction in terms of absolute values and can sometimes be misleading because of its averaging properties. Another metric that could be used instead is percentage of change in direction (POCID for short).

In terms of stock ranking, POCID is a measure of the relative outperformances in the stock universe. It provides an approximation of the *shape* of one's portfolio 6 months ahead. If our stock universe were a time-series, then POCID would give a metric of the direction of change. POCID is calculated by comparing the first differences $(t_2 - t_1), (t_3 - t_2), \ldots, (t_m - t_{m-1})$ and $(o_2 - o_1), (o_3 - o_2), \ldots, (o_m - o_{m-1})$, where t_i are the desired values and o_i are the predicted values of the outperformance and m is the total number of patterns in the training set for each pair of adjacent training patterns one by one. The POCID metric is defined as the number of pairs $((t_i - t_{i-1}), (o_i - o_{i-1}))$ that have the same sign for both differences $(t_i - t_{i-1})$ and $(o_i - o_{i-1})$, expressed as a percentage of the total number of such pairs $(m - 1)$.

POCID is, to a certain extent, sensitive to the order in which the training patterns are presented and therefore that order should remain the same for all simulations. Furthermore, it cannot be regarded as a measure of correct prediction in the direction of change of the outperformance, but as an indication of how well the network predicts the shape of the universe of outperformances.

4. STOCK PERFORMANCE MODELING—RESULTS

4.1. Overall Results: Comparison With MLR

The first target of this work is to show that the *in-sample* performance of the network gives a better fit than linear regression. The *in-sample* performance of the network is important because it determines its convergence ability and sets a target of feasible *out-of-sample* performance that can be achieved by fine-tuning the network parameters and training discipline. Figure 2 shows two scattergrams depicting the target vs predicted outperformance for MLR and a neural network with topology 3-32-16-1, learning rate $\eta = 0.3$, momentum rate $m = 0.3$, trained for 25,000 iterations.

The ideal shape in both scattergrams in Figure 2 would be a straight line with a slope of 45°, which crosses the origin. The reason is obvious; if the desired outperformance is, let us say, 0.2 the ideal would be that the predicted is also 0.2; if the desired is −0.1 the predicted should be the same, and so on. The points (0.2, 0.2), (−0.1, −0.1) define the line we described above. We see in Figure 2 that in the scattergram for MLR the dots are scattered all over the place, in contrast to the scattergram for the neural network where they resemble the shape of a straight line. This is reflected in the RMS values. The conclusion is evident: the neural network yields much better *in-sample* fitness than MLR.

4.2. Generalisation: Testing the DynIM Hypothesis

The second network performance metric is generalisation. Our goal here is to achieve *out-of-sample* performance comparable and if possible better than MLR. This would confirm that the model is indeed capable of producing better estimates for the λ coefficients. There are two out-of-sample periods. The (6-month) period immediately trailing the training period and the subsequent 6 months. For instance, if we are training from May to October 1985 we are using the values for A, B, C that are available at the time. However, to calculate Y there is a delay of up to 6 months until these values become available. So we would not know $Y_{oct} = g(y_{oct}/y_{march})$ until March 1986. In April 1986 we can use the model in one of three ways:

1. by supplying the network with the values of $(A, B, C)_{April}$ we can obtain a prediction for October 1986.
2. by supplying the network with the values of $(A, B, C)_{Nov\ 85}$ we can obtain a prediction for May 1986.
3. by supplying the network with the values of A, B, C from November 1985 through April 1986 we can obtain predictions for May 1986 through October 1986.

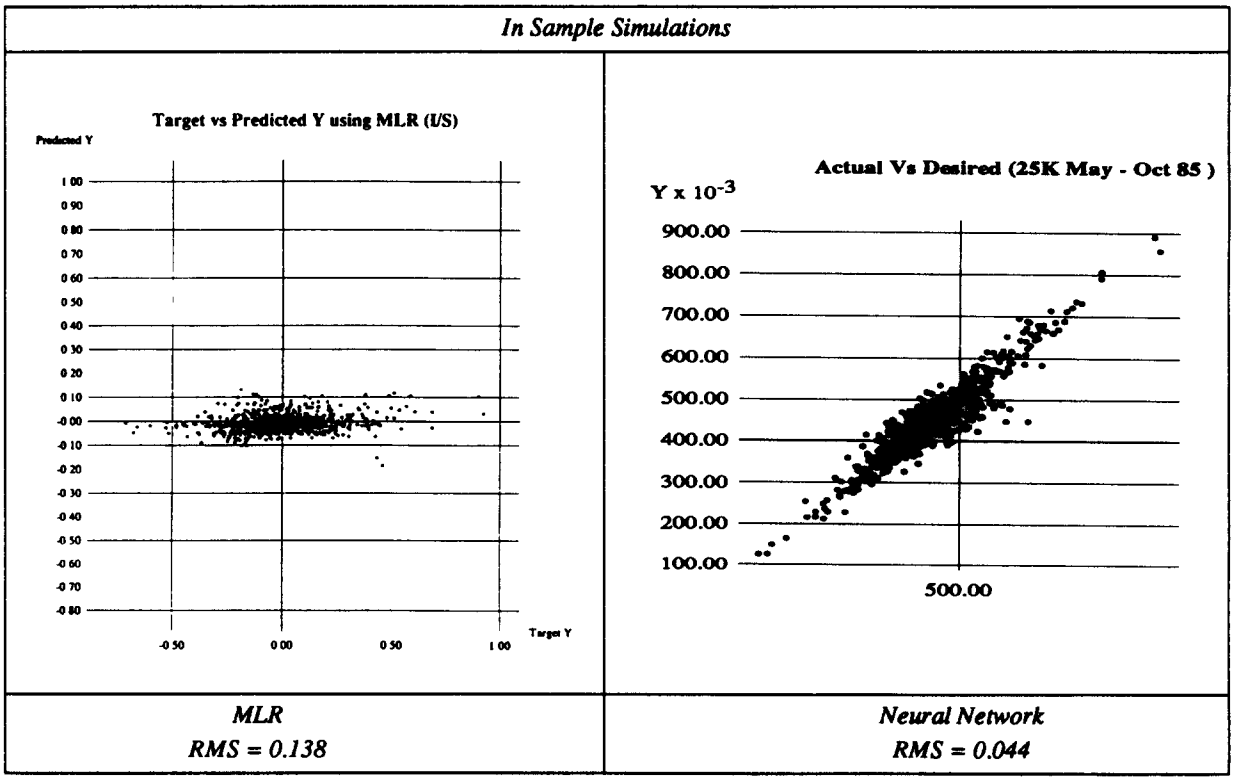

FIGURE 2. Target vs predicted outperformance, *in-sample* simulations (May 1985 to Oct 1985), with MLR and a 3-32-16-1 network with learning rate $\eta = 0.3$, momentum $m = 0.7$, trained for 25,000 iterations.

Assuming that the DynIM^{TM} hypothesis holds we are able to plot the relative outperformance of each stock up to 6 months in advance. Any further predictions will be less valuable as the relationship has changed according to the DynIM^{TM} hypothesis. The accuracy of these predictions (i.e., using A, B, C, values from November 1985 to April 1986) gives us the most objective criterion of the accuracy of the approximation that the network has made of the structural relationship between May 1985 and October 1985.

Figure 3 depicts the target vs predicted outperformance for MLR and the 3-32-16-1 neural network, for *out-of-sample* testing with factor values for November 1985.

Evidently the network has produced results that are much better than regression. Although not as good as in-sample, the prediction is reasonably accurate and certainly better than regression. The RMS error for this prediction is 0.066 compared 0.128 for MLR.

Figure 4 shows the RMS error (predicted vs actual) for each subsequent month up to and including April 1986. The solid line show RMS given by the neural network and the dotted line shows RMS given by MLR.

MLR starts with an in-sample RMS error of 0.12 and remains practically unchanged, which indicates that the model captures only some major features of the market, averaging out important trends and variations. The neural network, on the other hand, starts with an RMS error of 0.04 in-sample, which immediately increases to 0.066 for the following month and remains broadly unchanged for a further 5 months before it starts rising again. This type of behaviour is consistent with the second proposition of the DynIM^{TM} framework, which argues that the relationship between a stock's performance and its determinants changes over time but it does so relatively slowly.

The results presented above clearly outperform MLR. However, for a real-life application such as the one described here it is important to examine how these results vary with network parameters. The objective is to identify intervals of statistical stability for these parameters. This is analysed extensively in Zapranis (1992) and summarised in the following sections.

4.3. Stability With Network Architecture

We experimented with the architecture of the network, varying the number of hidden layers from one to three. The number of neurons of each layer were also varied. For all these simulations the learning rate (η) and the momentum rate (m) were fixed ($\eta = 0.3$ and $m = 0.7$). The number of iterations varied from 500 to 30,000. At the high end of training times, the RMS error *in-sample* remains very stable, varying from 0.044 to 0.072

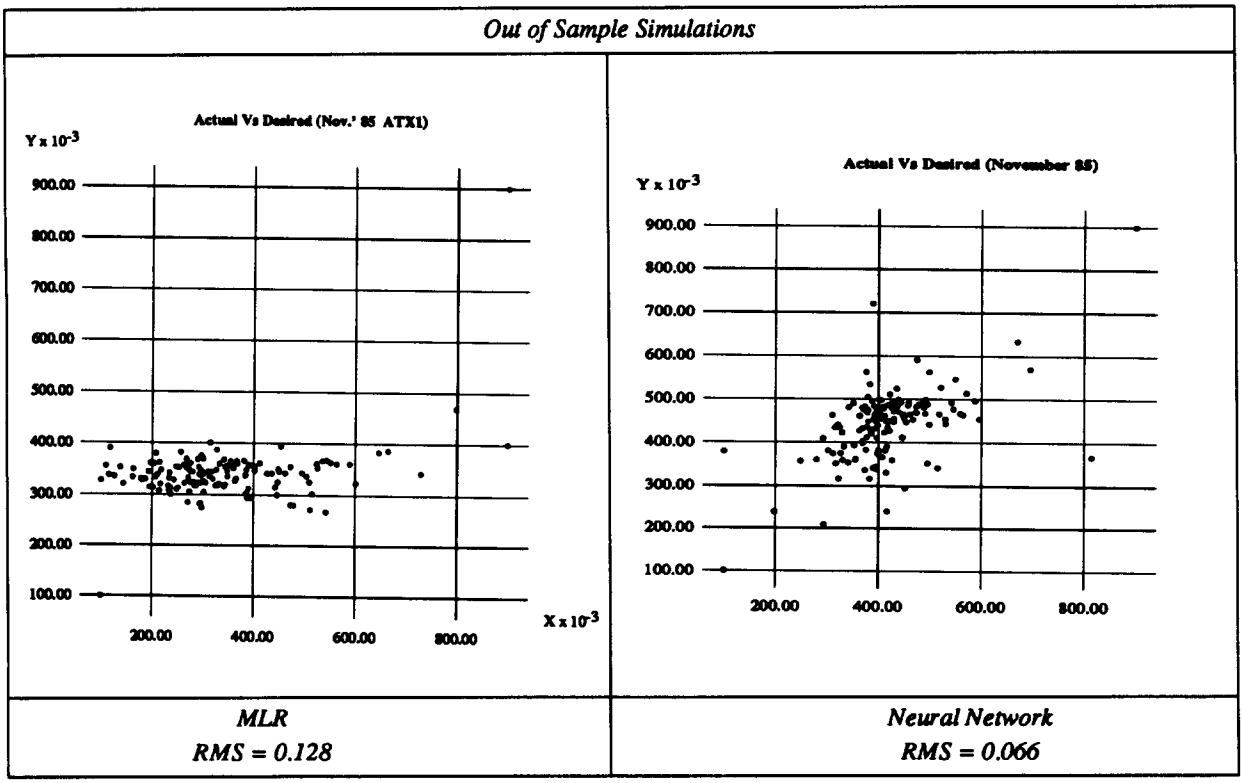

FIGURE 3. Target vs predicted outperformance, *out-of-sample* (A, B, C)s for November 1985 MLR vs a 3-32-16-1 network with learning rate $\eta = 0.3$, momentum $m = 0.7$, trained for 25,000 iterations.

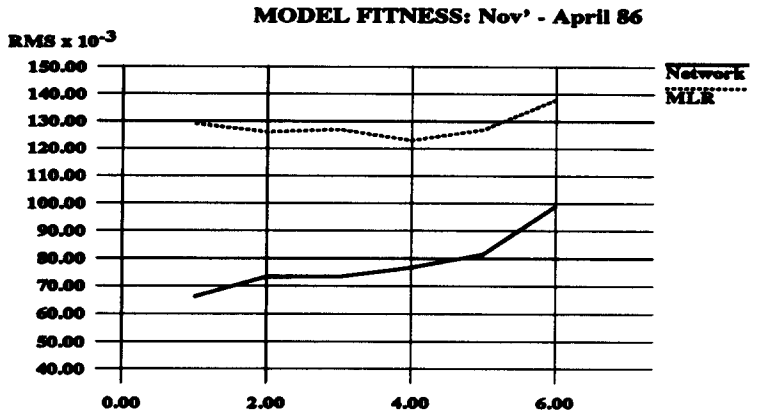

FIGURE 4. MLR vs Network—RMS error with *A*, *B*, *C* values from November 1985 through April 1986.

at worse. Single-layer networks give RMS error outside this range.

The *out-of-sample* performance also remains stable, varying from around 0.06 to around 0.10 (at worse) for networks with two layers of hidden units; POCID also remains stable around 85% (Zapranis, 1992) with a standard deviation of seven percentage points.

4.4. Stability With Gradient Descent Control Terms

A detailed analysis of the network performance stability can be found in Zapranis (1992). In summary, learning rates in the range 0.2 to 0.4 when combined with a momentum term of less than 0.7 yield better convergence (see Figure 5).

In general, one- and two-layered networks with a learning rate $\eta = 0.2$ and a momentum term $0.3 < m \leq 0.5$ yield the best combination of convergence and generalisation.

4.5. Stability With Training Time

For simple networks with one layer of hidden units, 5000 iterations were sufficient to stabilise the RMS to a virtually unchanged (with the number of iterations) value. For more complicated networks with more than one hidden layer, training times in excess of 15,000 iterations are necessary. The POCID metric behaves like the RMS, tending asymptotically to a maximum

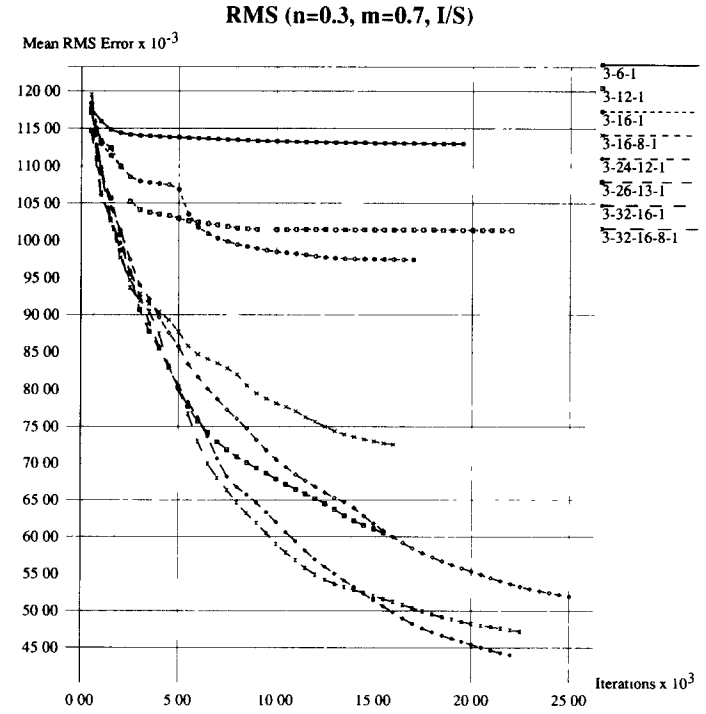

FIGURE 5. RMS (in-sample) for momentum $m = 0.7$, topology 3-16-1, different learning rates, and different numbers of iterations.

that is generally found after the 25,000 iterations limit, although the shape of the curve is more noisy.

For *out-of-sample* performance, we have the best results for large numbers of iterations. In general there is a smooth improvement in *out-of-sample* (generalisation) performance. The generalisation performance of the network improves steadily from 5000 through 10,000 iterations to 30,000 iterations. Even at the bottom end, at 5000 iterations, the network predicts significantly better than linear regression.

There are, however, regions with temporary performance drop-offs (both in terms of increased RMS or decreased POCID). We do not think, however, that these can be interpreted as signs of *overtraining*, because they appear rather early (mainly between 5000 and 10,000 iterations). Probably their existence implies that the network is still *undertrained*, and the better solutions are yet to come for larger numbers of iterations. This behaviour persists across different data sets.

4.6. Stability of Results With Different Training Samples

All the simulations mentioned so far were performed for the same training and testing datasets. The training data set contained monthly data for the period May 1985 to October 1985, and the testing data set contained data for November 1985 through April 1986 inclusive. To examine the effect of the data set on the performance of the network, we carried out simulations for the topology 3-32-16-1, using the mean values for the learning and network parameters.

The convergence and generalisation performance of the network does not alter significantly. It appears that the performance of the network is slightly worse than the mean performance in the previous data set but well within the range of the standard deviation (7% in terms of POCID, 0.005 in terms of RMS).

Figure 6 depicts RMS and POCID metrics for the new and old data sets. The profiles of the two curves are much the same.

4.7. Stability With Initial Conditions

Back propagation is known to be sensitive to the values of initial conditions, that is, randomised initial weight values. It is always desirable to observe the mean and standard deviation of the network performance measures for a large number of different initial conditions. We have so far performed only two random sampling runs for the same network configuration but with different initial conditions.

In-sample the curves for the RMS are very much the same, but for all other comparisons in and *out-of-*

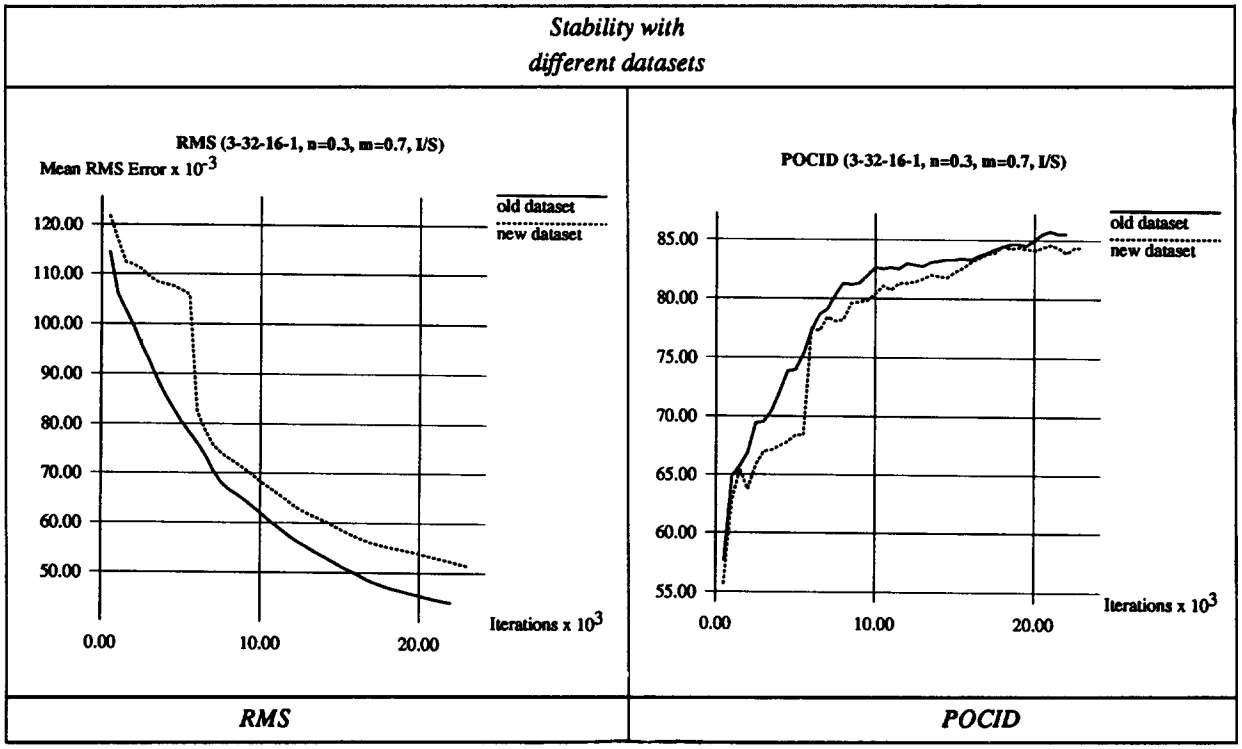

FIGURE 6. RMS in-sample, for topology 3-32-16-1, learning rate η = 0.3, momentum rate m = 0.7, old training data set May 1985 to October 1985, new training data set January 1986 to June 1986, old testing data set April 1986, and new testing data set December 1986. POCID in-sample, for topology 3-32-16-1, learning rate η = 0.3, momentum rate m = 0.7, old training data set May 1985 to October 1985, new training data set January 1986 to June 1986, old testing data set April 1986, and new testing data set December 1986.

sample the first set of initial weights marginally outperforms the second one. We believe that some adjustment to the network configuration while using the second set of initial weights, or if we insist on harder training, might help to bridge that gap in performance. It is clear that the starting point of the training phase can make a difference (perhaps not a great one) but it should be a consideration when training the network.

5. STOCK MARKET ANALYSIS

5.1. Factor Significance Estimation

In the framework of qualitative asset allocation, it is useful to have an estimate of the degree of impact that a factor has on the relative outperformance. As the structural relationship between factors and outcome changes over time, an accurate estimate of current degree of impact that each factor has (in the present economic environment) can be a very useful tool in asset reallocation. Suppose for example that *market capitalisation* were one of the factors. If it were observed that this factor is starting to become significant (wrt other factors), then the portfolio could be restructured to contain a larger component of assets with high/small market capitalisation as appropriate.

To interpret the relative significance of the various factors, the change in output (outperformance Y) relative to the change in an input variable (factor X), $\partial Y/\partial X$, $X \in \{A, B, C\}$, needs to be determined. Several researchers have attempted to interpret the network dynamics in similar ways (Gorman & Sejnowski, 1988). Here we perform parameter significance estimation in a way similar to Klimisaukas, Guiver, and Pelton (1989) and Sen, Oliver, and Sen (1992).

If the network is simple with no hidden layers it is trivial to compute the partial derivative, $\partial Y/\partial X$. We know that the output Y is given by the sigmoid function $f(z) = 1/1 + e^{-z}$ where $z = w_{AY} + w_{BY} + w_{CY} + \theta$, and θ is an internal threshold (see Figure 1 left panel). We also know that $f(z)$ is continuous and differentiable.

For multilayered networks, the partial derivative, $\partial Y/\partial X$, can be computed by applying the chain rule for derivatives repeatedly through the paths that connect the output node Y to the input node X. Starting from the output layer we know that $x_j = \sum w_{ij} y_i$, therefore $\partial x_j/\partial y_i = w_{ij}$. Using the chain derivative rule:

$$\frac{\partial Y}{\partial y_i} = w_{ij} \frac{\partial Y}{\partial x_j} \tag{7}$$

Proceeding to the next layer on the left and continuing until the input layer, with similar applications of the chain rule, we obtain:

$$\frac{\partial Y}{\partial X} = w_{gh} K_h \tag{8}$$

where h denotes the nodes in the middle hidden layer, g denotes nodes in the input layer, and K_h is given by:

$$K_h = \frac{\partial y_h}{\partial x_h} w_{hi} \frac{\partial y_i}{\partial x_i} w_{ij} \frac{\partial Y}{\partial x_j} \tag{9}$$

Like Sen, Oliver, and Sen (1992) we use this notation to impress that K_h is a function independent of any parameters in the input layer. The node at the output layer and the nodes in the input layer g are connected via multiple paths through the hidden layers. To obtain the partial derivative over all paths G connecting the nodes Y and $X \in \{A, B, C\}$ we sum all such $\partial Y/\partial X$. Using the notation in eqn (8) we obtain:

$$\left.\frac{\partial Y}{\partial X}\right|_G = \sum_h w_{gh} K_h \tag{10}$$

We note that the K_hs are the same for all input variables, X_g, because all changes reflected from the outer layer through the hidden layers pass through the same paths for all input variables. We note that this does not include the input layer weights. Therefore, a relative change in Y with respect to a change in a factor (A, B, or C) is influenced primarily by the weights in the input layer. If the absolute values of the weights in the input layer for a particular factor are high, then it is reasonable

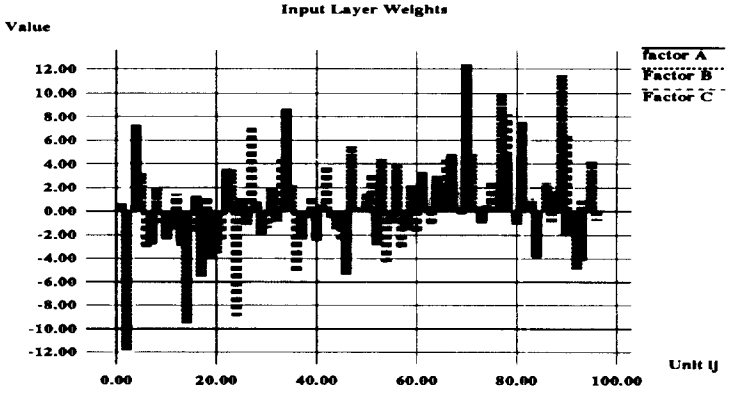

FIGURE 7. Factor significance estimation.

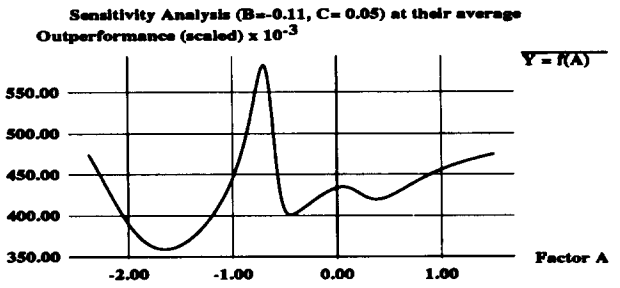

FIGURE 8. Sensitivity plots for factor *A.**

to expect that the result in Y will be very sensitive to changes in this input. The magnitude of this change cannot be determined because it depends upon the sign and magnitude of K_h, but an approximation can be estimated by considering a linear combination of the weights in the input layer. Klimisaukas et al. (1989) and Sen et al. (1992) note that this approximation is necessary only when the K_hs are not all of the same sign.

Figure 7 plots the values of the input layer weights for each factor. The sum of the absolute values of the input layer weights for each factor A, B, and C are 103, 110, and 91, respectively. The two factors with the highest significance for the period under consideration (i.e., May to October 1985) are A and B. This is a very important finding and we can use it as an integral part of the asset allocation process (see also later sections).

The estimates for $\partial Y/\partial X$ produced here are only approximations. Better estimates can be produced if the K_hs were known. However, the K_hs not only depend upon the weights in the hidden layers, they also depend upon the input vector. The K_hs change with each input vector. Sen et al. (1992) propose an approach for obtaining better estimates for the K_hs based on a weighted mean for K_h that can be obtained by combining all the weights and the inputs. The weighted mean can then be used in eqn (9) to obtain a better estimate of $\partial Y/\partial X$. A complementary approach would be to experimentally analyse the sensitivity of the output as the values of each factor are varied while holding the other factors constant at some fixed point in their range. This is very useful for simulating "what-if" scenaria in asset management.

5.2. Sensitivity Analysis

To perform sensitivity analysis we first determine the ranges of factors A, B, and C. For each factor, the minimum, maximum, and the mean value of the range are determined. The values of each factor are then varied one at a time while holding the values of the other factors constant at their mean (or median) value. For each factor being varied we step through its range smoothly using a relatively small step size. In our case the ranges for A, B, and C are $[-2.37, 1.50]$, $[-2.733, 3.86]$ and $[-1.44, 1.67]$, respectively. We generate approximately 300 values (A^*, B^*, C^*) for each factor. These *hypothetical* values are then fed to the neural network model that is used to compute the output. This output is treated as the likelihood of the sample being a target. Wan (1990) shows that the output of a multilayer network classifier can be considered to be a nonparametric estimate of a posterior probability of the sample belonging to a particular class.

For all three variables, the network outcome is plotted against the value of the factor. The plot for factor A is shown in Figure 8. It should be noted that the absolute value of the predicted outperformance, Y, is not as important as the *change* in the network output. As expected, the relationship between Y and factor A is a complex nonlinear function.

The sensitivity plots for a factor are interpreted to indicate that if all other factors were held constant at around the mean value, changes in this factor would affect the relative outperformance in the manner shown by the plot. For instance, Figure 8 shows that the out-

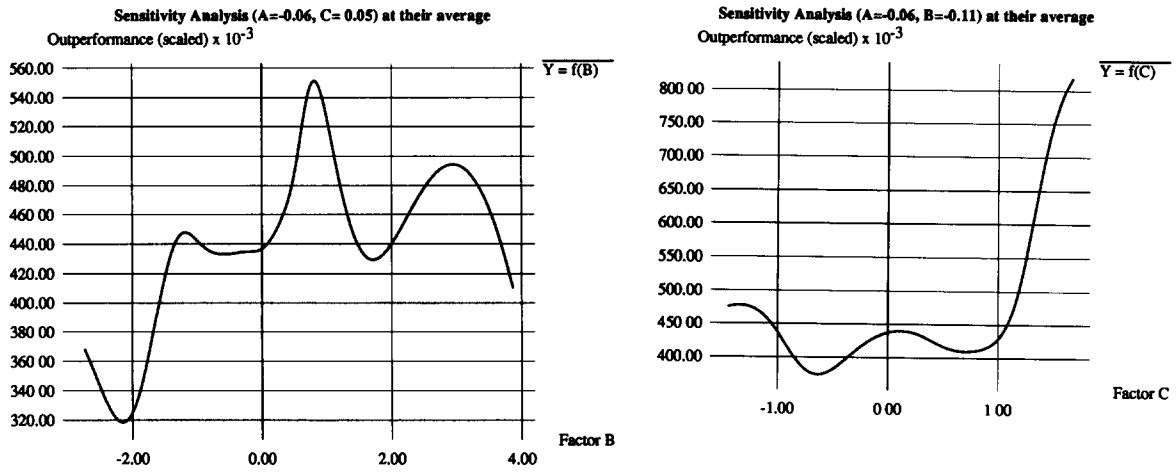

FIGURE 9. Sensitivity plots for factors *B and *C**.**

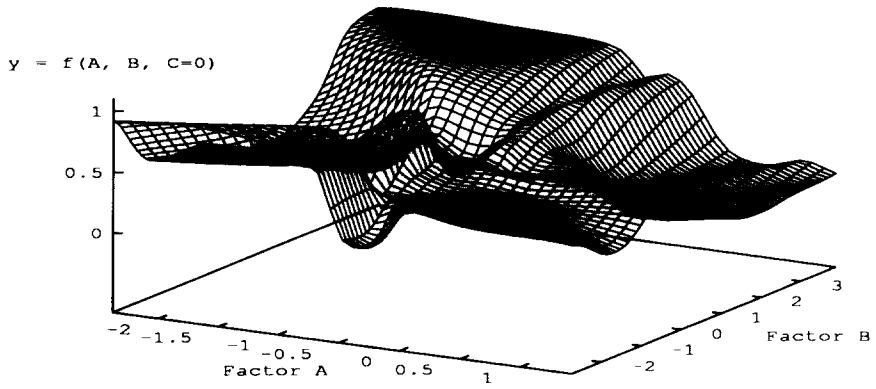

FIGURE 10. Estimated outperformance surface for factors *A* and *B* (May to October 1985); *C* remains fixed at its mean value.

performance is very sensitive to changes in factor A. For large negative values of A, stock performance is very low, and it increases with A in a nonlinear manner. The plot is consistent with general expectations. Regression models are unable to capture such information.

Factor B also shows a similar nonlinear behaviour. Factor C appears to be the least sensitive of all, in that only changes in its values at the extreme end of its range have a significant influence in outcome (see Figure 9).

These observations confirm the expectations predicted by the analysis in the previous section.

5.3. Market Profiling

The justification for the sensitivity analysis lies in the argument that if the model captures the relationship between predictor variables and the outcome accurately, then the relationship is very likely meaningful at least for the training sample. Sen et al. (1992) assert that the *quality* of the model can be relied upon if the predicted behaviour is not counter-intuitive. We argue that it is possible to quantify the quality of the model by observing its prediction accuracy for the *out-of-sample* period immediately following the training period. In Section 4.2 we used the *out-of-sample* RMS error to show that the quality of the neural model is superior to that of current *best practise*

In this section we extend the sensitivity analysis to produce a more complex profile of the stock market by keeping the least significant factor fixed at its midpoint and varying the most significant factors from the minimum to the maximum values in their range. We repeat this process for consecutive time frames to produce a 3-D visualisation of how the market evolves as the economic environment changes.

In Figure 10 we fix the least significant factor C at its mean value and vary the values of factors A and B using approximately 60 equal intervals (each) over their whole range. The neural network model is then used to compute the output. For both factors A and B (X and Y axis) the predicted outperformance is plotted on the Z axis. The estimated decision surface is viewed from an angle of 60°, 30°. The network was trained with data from May to October 1985.

As expected, the relationship between outperformance and the determinant factors is a complex nonlinear function. It is interesting to note the effects of the nonlinear combination of factors A and B. In particular, in the low end of the ranges $[-2, -1.5]$ the nonlinearly combined values of A and B have an overall positive effect.

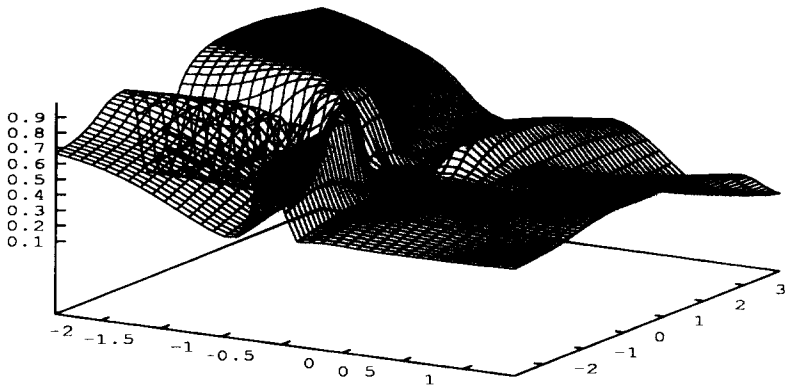

FIGURE 11. Estimated outperformance surface for variables *A* and *B* (November 1985 to March 1986).

Again, it should be noted that the *absolute* value of the predicted outperformance is not as important as the predicted *changes* in the outperformance. However, the fact that when we tested the predicted output against the actuals (in Section 4.2), we only found a relatively small RMS error provides important evidence that the predicted surface can be treated as being quite accurate even in absolute terms.

We now move on to the next 6-month period and repeat the process with a retrained network. Figure 11 shows the estimated outperformance surface produced by a network trained with data from November 1985 to March 1986. A slightly different surface is now predicted, with stocks in the far corner being the best suggested outperformers. Stocks in the left-hand corner (i.e., $A \in [-2, -1]$, $B \in [-2, -1]$) that were good performers started to drop off.

Figure 12 shows the estimated outperformance surface produced by a network trained with data from January 1986 to June 1986. There is now a definite concentration of *good* stocks in the far corner and the *bubble* between $A \in [0, 0.5]$ and $B \in [1, 3]$ is starting to migrate to the back right-hand side corner.

It must be noted that because the relationship will be continuously recalculated every month, the evolution of the market profile will be much smoother than shown in Figures 9 and 10. In fact, we deliberately produced the estimated outperformance surfaces to cover both nonoverlapping periods (i.e., May to October 1985 and November 1985 to March 1986) and overlapping periods (i.e., November 1985 to March 1986 and January 1986 to June 1986). It can be seen from Figures 9 and 10 that the overlapping period shows a smoother transition.

5.4. Testing the DynIM Hypothesis

The estimated outperformance surfaces support the DynIM™ hypothesis that the structural relationship between an asset price and its performance changes dynamically over time, but they do so relatively slowly. However, to test the DynIM™ hypothesis more rigorously, we need to ensure that the network predictions are accurate and reliable.

This is an impossible task because we have no way of confirming that the predicted outperformance for *all* simulated triples (A^*, B^*, C^*) is in fact the same as the actual one. The most objective way of doing so is to test the RMS error of the predicted against the actual outperformance for those out-of-sample triples (A, B, C) for which we know the outcome. If the network error for these triples is low, there is every chance that the network approximates the outperformance surface quite accurately. In Section 4.2 (Figures 3 and 4) we tested the DynIM hypothesis for the period May to October 1985 by comparing the RMS error between predicted vs actual for the subsequent 6 months. The findings supported the DynIM hypothesis and showed that the networks are more accurate than MLR. The RMS error was satisfactorily low. So there is every chance that the predicted outperformance surface following May to October 1985 (as shown in Figure 9) is quite accurate.

We now repeat the process for the periods November 1985 to March 1986 and January 1986 to June 1986. Figure 13 shows the RMS error for out-of-sample predictions from April 1986 through October 1986 on a month-on-month basis. Likewise, Figure 13 shows the RMS error for out-of-sample predictions from June 1986 through to December 1986.

Like Figure 3, in both cases the RMS error starts with a level close to 0.04 in-sample, which increases to 0.055 for the trailing month and remains low for a further 6 months.

6. CONCLUSIONS AND FURTHER WORK

Classical statistical techniques for prediction reach their limitations in applications with nonlinearities in the data set. Most forecasting methods are only capable of picking up general trends and have difficulty in modeling cycles that are by no means repetitive in ampli-

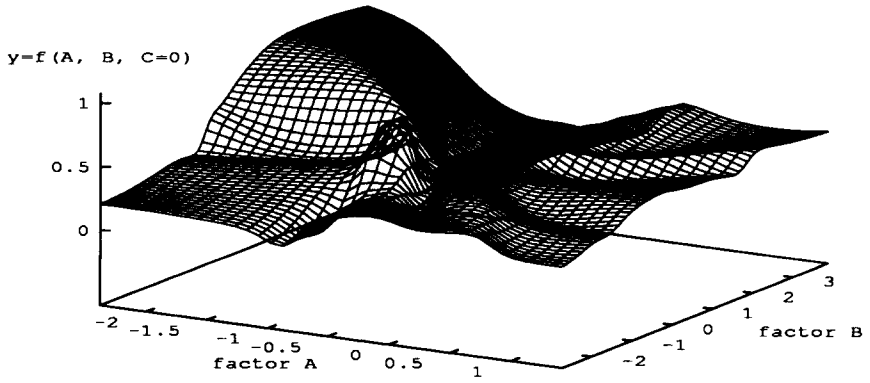

FIGURE 12. Estimated outperformance surface for variables *A* and *B* (January 1986 to June 1986); *C* remains fixed at its mean value.

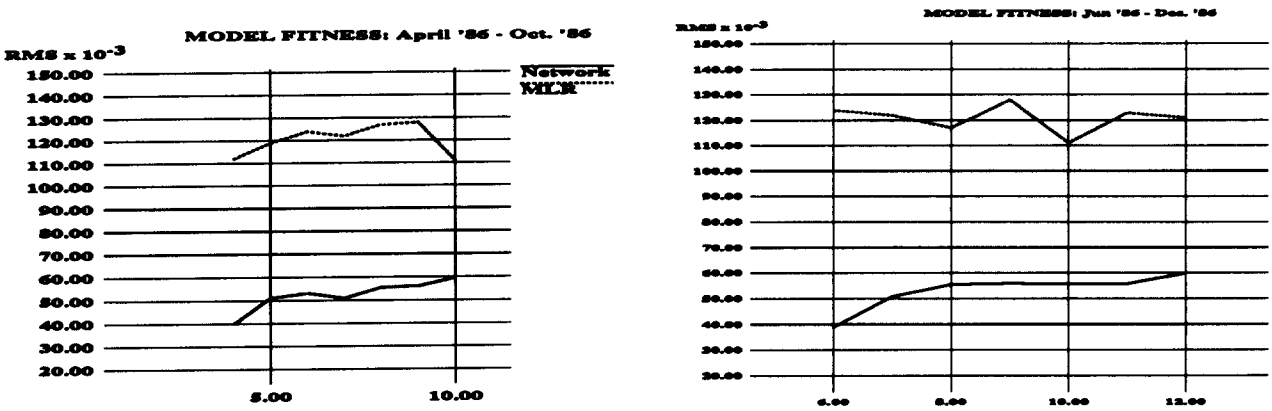

FIGURE 13. Out-of-sample RMS April 1986 through October 1986; out-of-sample RMS June 1986 through December 1986. It should be noted that in the overlap, predictions are made by two different networks.

tude, period, or shape. Despite their inadequacies, techniques such as multiple linear regression have proved to be a useful tool in the Capital Markets and are used routinely.

We showed that even simple neural learning procedures such as the back propagation algorithm far outperform current best practice in a typical application for stock ranking within the framework of the arbitrage pricing model. Their smooth interpolation properties allow neural models to fit better models to the data and to generalise significantly better.

We believe that the performance measures obtained here can be improved further with careful network design and preprocessing of the data. As far as the data is concerned, there is at least one obvious area of improvement. It concerns the existence of malicious vectors in the training set. These are vectors that lie close to the borders between classes (i.e., one-to-many mappings) and that the quadratic cost function used here finds difficult to learn (at best it averages). We have developed an algorithm for detecting such malicious vectors (Zapranis, 1992) and applied it to the training set. We found that up to 13% of the training data were classified as such vectors, and we are currently experimenting with various strategies for dealing with such vectors (Tuv & Refenes, 1993).

REFERENCES

Dutta, S., & Shashi, S. (1988). Bond rating A non-conservative application of neural networks. *Proceedings of the ICNN-88* (Vol II). San Diego, CA, July 24–27 1988.

Gorman, R. P., & Sejnowski, T P. (1988). Analysis of hidden units in a layered network trained to classify sonar targets *Neural Networks,* **1,** 75–89

Hinton, G (1987) *Connectionist learning procedures* Carnegie Melon University, Computer Science Department

Klimisaukas, C. C , Guiver, J , & Pelton, G. (1989). *Neural Computing* Pittsburgh: Neural Ware Inc.

Refenes, A. N. (1992) Constructive learning and its application to currency exchange rate prediction. In E. Turban & R Trippi (Eds.), *Neural network applications in investment and finance services* (Chap. 27). Chicago· Probus Publishing.

Refenes, A N , & Azema-Barac, M. (1993) Neural network applications in financial asset management. *Neural Computing & Applications Journal* (accepted)

Refenes, A., Azema-Barac, M., Chen, L , & Karoussos, S. A. (1991) Currency exchange rate prediction and neural network design strategies *Neural Computing & Applications Journal,* **1**(2).

Refenes, A N , & Zaidi, A (1992) Managing exchange rate prediction strategies with neural networks *Proceedings of the Workshop on Neural Networks Techniques & Applications,* Liverpool.

Ross, R L., & Ross, F. (1990) An empirical investigation of the arbitrage pricing theory. *Journal of Finance,* **44,** 1–18.

Schoenenburg, E (1990) Stock price prediction using neural networks: A project report. *Neurocomputing,* **2,** 17–27

Sen, T, Oliver, R., & Sen, N (1992). *Predicting corporate mergers using backpropagation neural networks A comparative study with logistic models* Virginia Tech, R. B Pamblin College of Business, Department of Accounting.

Sharpe, W. (1964). Capital asset prices. A theory of market equilibrium under conditions of risk *Journal of Finance,* **19,** 425–442

Tuv, E., & Refenes, A. N. (1993) Removal of catastrophic noise in hetero-associative training samples. *International Joint Conference on Neural Networks,* Nagoya, Japan.

Wan, E A. (1990). Neural network classification: A Bayesian interpretation. *IEEE Transactions on Neural Networks,* **1,** 303–305

White, H. (1988). *Economic prediction using neural networks The case of IBM daily stock returns* University of California, Department of Economics.

Zapranis, A. D. (1992) *Stock ranking using neural networks* (Proj Rep.). University College London, Department of Computer Science.