Combining Artificial Neural Networks and Statistics for Stock-Market Forecasting

Shaun-inn Wu
College of Arts and Sciences
California State University
San Marcos, CA 92069

Ruey-Pyng Lu
Douglas W. Mahoney
Department of Statistics
North Dakota State University
Fargo, ND 58102

Abstract: We have developed a stock-market forecasting system based on artificial neural networks. The system has been trained with the Standard & Poor 500 composite indexes of past twenty years. Meanwhile, the system produces the forecasts and adjusts itself by comparing its forecasts with the actual indexes. Since most of stock-market forecasting systems are based on some kind of statistical models, we have also implemented a statistical system based on Box-Jenkins ARIMA(p,d,q) model of time series. We compare the performance of the these systems . It shows that the artificial neural network's forecasting is generally superior to time series but it occasionally produces some very wild forecasting values. We then developed a transfer function model to forecast based on the indexes and the forecasts by the artificial neural networks.

1. Introduction

Most of stock-market forecasting systems are based on some kind of statistical models. They usually suffer from the huge computational cost of handling large volume data. However, in order to have a more accurate predication, these systems usually are required to forecast based on the data of the last ten or twenty years. Hence, most of them can only forecast based on monthly data of past years or the finer data of much shorter history in order to be computationally feasible.

We designed and implemented an *artificial neural network* (ANN) [7]. This ANN has been used to forecast the trends of stock market. The system is able to forecast based on much more fine-grained daily data without much computational cost.

ANNs date back to Rosenblatt's work with perceptrons in the 1960's [5]. They have experienced a resurgence in popularity as a paradigm for machine learning. A great deal of research is underway exploring the relative power of different types of ANNs using various algorithms [9]. ANNs have been applied to a wide variety of problems and have proven quite successful at tasks such as image restoration [4], speech recognition [3], loan processing [8], gene classification [11] and parts classification [2]. Recently, much research has been done to improve ANN and to test its ability in application areas.

2. Artificial Neural Network Forecasting

In order to understand the capability of ANN, we will have to discuss the structure of ANN. ANN attempts to simulate the functioning of biological neural systems on a simplified level. *Nodes* in the network represent neurons in a biological system. Each node has some inputs and output *connection* lines and can be in one of the two possible states, active or inactive. The output lines of a node are connected to the input lines of other nodes. An adjustable *weight* is assigned to each connection to represent the strength of that connection. A node computes its state as a function of the weights of the input connections to adjacent nodes and the states of the adjacent nodes.

2.1 Artificial Neural Network Structure

ANNs can be represented by an interconnected network of the nodes described above. They exist

in many variations. ANNs may be structured into disjoint sets of nodes called *layers* that are arranged logically on top of one another. Each layer may contain disjoint sets of nodes called *clusters* and the clusters may contain nodes which may be specialized or homogeneous. The ANN may be organized into an input layer, an output layer and a number of intermediate layers. An input pattern is applied to the ANN by activating certain nodes, and an output pattern is produced as a function of the weights of the connections. The ANN learns by comparing the resulting output pattern to the correct output and adjusting the weights of the connections accordingly.

We have designed and implemented a system to simulate the operation of ANNs. It is actually a management system for ANNs. The user can create ANNs of various sizes, save and retrieve the ANN states, put them through learning cycles, and test their learning progress. It was originally designed to create and maintain ANNs of any practical size. Both cluster- and non-cluster-based ANNs are supported.

The ANN we designed is made up at the lowest level by nodes [10]. These nodes are grouped into clusters that are used to regulate which nodes are active or inactive. Within a cluster only one node is allowed to be active at any time; nodes within clusters compete to be in the active state. These clusters in turn form the layers of an ANN. Each node in a layer is linked by a weighted connection to every node in the layer that precedes it. The weight of each connection affects how strongly signals from one node reach another.

All of these structures are further divided into three types: *input*, *hidden* and *output*, each serving some similar and some dissimilar functions. The layer at the bottom level is the input layer, and the layer at the top is the output layer. In between these two layers are one or more hidden layers. The input layer's nodes serve no purpose other than to hold their *states* for use by the other layers. Each node in a layer, except the input layer, has an input line connected to an output line of each node in the layer below it, that is, a node's output lines go to the input lines of the nodes above. Each line has a weight associated with it, representing the strength of that connection. The state of a node is determined by a function of the weights of its input lines and the states of the nodes with output lines connecting to it. The hidden and output layers' nodes hold additional information on the connections, or weights, between nodes and calculate their states from this information during their calculation iterations. The hidden and output layers differ in that when the ANN performs a learning iteration, the output layer's *errors* are externally set according to the application, while the hidden layers use internal calculations to determine their error.

2.2 Forward Propagation Process

ANN works by mapping input patterns to output patterns in the following manner: An input pattern is applied by activating certain nodes in the input layer to represent the pattern. According to the input application pattern, the states of the input nodes are set to *on* or *off* to best represent the data to be considered by the ANN. In each successive layer the nodes calculate their *activation levels* as a function of the weight values of the input lines that are connected to active nodes in the previous layer.

If the layers are subdivided into clusters, the nodes in the same cluster may compete with one another for the right to enter the active state. The clusters in the layer set the node with the highest activation level to an on state and set all the others to off. This is based on the competitive ANN model [6]. If the layers are not organized into clusters, a node goes into the active state when its activation level is greater than the threshold value. Thus each intermediate layer produces a pattern of activated nodes based on the layer below. This process is carried out up to the top layer, resulting in an output pattern for the applied input pattern.

This process is called *forward propagation*, since a pattern proceeds forward from the input layer through the hidden layers until the output layer is reached, and the states of the output nodes determine the outcome of the ANN. If the ANN has already been taught to perform the job, this is a complete iteration. However, if the ANN is in the learning mode, another process called *backward propagation* is applied.

2.3 Backward Propagation Process

After the forward propagation is finished, the ANN checks whether the resulting output pattern is correct and adjusts the weights accordingly. The correct or expected output pattern is presented to the ANN by a teacher, taken from a file, or generated by an algorithm. The ANN works backward to the input layer, adjusting the weights

of the network connections according to how much each one contributed to the overall error. Since the weights are initially randomly distributed, the ANN responds incorrectly at first, getting more accurate as it learns.

In the backward propagation process, the connection weights of all nodes are modified based on an error formula and weight adjustment formula to make the ANN respond with a particular output given a certain input. Backward propagation is initiated by setting the error of each of the output nodes directly from the correct output pattern.

Errors for nodes in the output layer are calculated as the difference between the returned outcome and the expected outcome. For the hidden layer, the error for a node is the amount of error that it contributed to the nodes in the output layer. A hidden layer node has an error only if it is active. If it is not active, then it cannot cause an output node to fire erroneously, therefore it has no error. Then the previous hidden layer calculates its error according to the weights of the connections between its nodes and the nodes in the next layer and the error value of each of those nodes. Input layer nodes usually have no error, since they are set to the active state to directly correspond with the input pattern.

After the errors have been calculated for all the nodes in the output and hidden layers, the weights are adjusted according to a function of the error of the node in the upper layer associated with the connection. If the node's error is positive this weight is then redistributed to the active input lines, thus strengthening the connections to the active nodes below. If the node's error is negative, the weight is redistributed to the inactive input lines, reducing the strength of the connections to the active nodes below. Hence, the weights of the output and hidden layers are adjusted to compensate for the error and reduce the tendency of the ANN to produce erroneous results.

The above backward propagation process continues through all the hidden layers.

In our ANN the sum of the weights of all the input connections for a node is normalized to the value one and the weight adjustment formula sustains this normalization. This maintains the "fairness" of the competition between nodes. If the sum of the input weights were unequal, a node in a cluster would have an advantage if the sum of its weights were greater than that of the other nodes in the cluster.

2.4 Forecasting Based On Artificial Neural Network

Currently, our ANN has been trained by presenting one S&P 500 index in each iteration. It then responds with another index as the output. Because each index is a five-digit number with two digits after the decimal point, we decided to do some preprocessing and convert the index to a binary representation. Hence, both the input and output layer have the same structure. We are thus working in a problem domain mapping an input pattern to an output pattern.

For our study purposes, we have temporarily limited the network size to three layers. The sizes of the input and output layers were also fixed to represent our sets of input and output patterns. The input and output layers are fixed to sixteen clusters of two nodes each. The size and cluster structure of the one hidden layer is user-specified upon creation of the ANN. Taking an experimental approach, we found some acceptable structure of the hidden layer in our ANN.

This weight adjusting process is actually where the learning takes place. The ANN is run through the forward and backward propagation cycle many times, using input patterns from our data set. The goal of the ANN is for the weights of the ANN to settle to a state where a high percentage of the output patterns being generated are close to the actual indexes of the following days. Once the ANN has been trained, it will respond correctly to a high percentage of very closed indexes. This ability to utilize the knowledge stored in the connection weights to forecast is the major strength of artificial neural networks.

3. Statistical Forecasting

A *time series* is a chronological sequence of observations on a particular variable. The components of a time series are trend, cycle, seasonal variations and irregular fluctuations. These components do not always occur all together. Generally, an additive model $Y_t = T_t+C_t+S_t+\varepsilon_t$ is used where Y_t is the value of response variable at time t, T_t is the contribution from the

trend, C_t is the cyclic effect, S_t is the seasonal effect, and ε_t is the unexpected random effect.

In our study, forecasting is the most concerned subject in analyzing the business and industrial time series data. One of the most important problems to be solved in forecasting is to match an appropriate forecasting model to the pattern of the available time series data.

3.1 Box-Jenkins Forecasting Method

The Box-Jenkins forecasting method we used consists of a four-step iterative procedure [1]:
1. Tentative model identification: historical data is used to tentatively identify an appropriate Box-Jenkins model (see the next section).
2. Parameter estimation: the historical data is used to estimate the parameters of the tentatively identified model.
3. Diagnostic checking: various diagnostics are used to check the adequacy of the tentatively identified model; an improved model may be suggested.
4. Forecasting: once a final model is obtained, it is used to forecast future time series values.

Given a time series data, $Y_1, Y_2, ..., Y_N$, we believe that there is a stochastic process $..., X_{-2}, X_{-1}, X_0, X_1, X_2, ...$ which has generated them. Once we identify the process, we use it for forecasting, i.e., $Y_{N+1} = E(X_{N+1} | X_1=Y_1, X_2=Y_2, ..., X_N=Y_N)$. For a stochastic process X_t, t=0, ±1, ±2, ..., let the *mean function* be $\mu_t = EX_t$, the *variance function* be $\sigma_t^2 = Var(X_t)$, the *auto-covariance function* be $r(s,t) = Cov(X_s, X_t) = E[(X_s - E(X_s))(X_t - E(X_t))]$, the *autocorrelation function* be $\rho(s,t) = r(s,t)/\sigma_s\sigma_t = r(s,t)/\sqrt{r(s,s)\, r(t,t)}$.

A time series is *stationary* if the statistical properties of the time series are essentially constant through time, i.e., μ_t is a constant and r(s,t) depends only on the difference |s-t|. Experience shows that, if the original time series values $Y_1, Y_2, ..., Y_N$ are nonstationary, using the first differencing transformation $Z_t = \nabla Y_t = Y_t - Y_{t-1}$ or the second differencing transformation $Z_t = \nabla^2 Y_t = (Y_t - Y_{t-1}) - (Y_{t-1} - Y_{t-2}) = Y_t - 2Y_{t-1} + Y_{t-2}$ will usually produce stationary time series values where $\nabla = 1 - B$ and B is the backward shift operator, i.e., $B\varepsilon_t = \varepsilon_{t-1}$.

3.2 Box-Jenkins Models

If a time series data $Z_1, Z_2, ..., Z_N$ has no trend, cyclic and seasonal variations, we believe there is a stationary process that generated the data.
1. White noise model: ε_t, t=0, ±1, ±2, ... is a sequence of independent identically distributed random variable with common mean 0 and variance σ^2. Here $Y_t = \varepsilon_t$, t=0, ±1, ±2, ..., and $r(s,t) = Cov(Y_s, Y_t) = Cov(\varepsilon_s, \varepsilon_t) = \sigma^2$ if t = s or 0 if $t \neq s$.
2. Moving average model of order q: MA(q). $Y_t = \varepsilon_t + \theta_1\varepsilon_{t-1} + ... + \theta_q\varepsilon_{t-q} = (1 - \theta_1 B - ... - \theta_q B^q)\varepsilon_t$, t=0, ±1, ±2, ...
3. Auto-regressive model of order p: AR(p). $Y_t = \varepsilon_t + \phi_1 Y_{t-1} + ... + \phi_p Y_{t-p}$, t=0, ±1, ±2, ... and $\varepsilon_t = Y_t - \phi_1 Y_{t-1} - ... - \phi_p Y_{t-p} = (1 - \phi_1 B - ... - \phi_p B^p) Y_t$.
4. Auto-regressive and moving average model of order (p,q): ARMA(p,q). $Y_t - \phi_1 Y_{t-1} - ... - \phi_p Y_{t-p} = \varepsilon_t - \theta_1\varepsilon_{t-1} - ... - \theta_q\varepsilon_{t-q}$.
5. General ARIMA(p,d,q) model: d is the number of times the differencing transformation is applied to the time series data $Y_1, Y_2, ..., Y_t$ such that $Z_t = \nabla^d Y_t$ is stationary.

3.3 Parameter Estimation

In the time series data $Y_1, Y_2, ..., Y_N$, $Z_t = \nabla^d Y_t = (1-B)^d Y_t$ is identified as ARMA(p,q) model. Let $\phi_p(B) = 1 - \phi_1 B - ... - \phi_p B^p$, $\theta_q(B) = 1 - \theta_1 B - ... - \theta_q B^q$, and $\delta = \mu(1 - \phi_1 - ... - \phi_p)$. We have $\phi_p(B)Z_t = \delta + \theta_q(B)\varepsilon_t$. Box and Jenkins favored the maximum likelihood approach to calculate the estimates $\hat{\phi}_1, ..., \hat{\phi}_p$ and $\hat{\theta}_1, ..., \hat{\theta}_q$ and $\hat{\delta}$ [1].

This approach could be somewhat difficult and costly to implement and so they suggested using the least squares approach to calculate estimates.

3.4 Diagnostic Checking

After a Box-Jenkins model is identified, the parameters of the model are estimated, the residuals are computed, the model is tested to find out if it fits well by performing a statistical test of hypothesis H_0: ε_t's are i.i.d. $N(0,\sigma^2)$, i.e., H_0: $\rho_1 = \rho_2 = ... = \rho_k = 0$. Ljung-Box statistic is calculated, $Q^*_k = n(n+2) \sum_{i=1}^{n} r_i/(n-i)$ where r_i is the sample autocorrelation of the residuals at lag i. A large value of Q^* indicates that the model is not adequate. Thus Q^* will be compared with χ^2_{k-p} where p is the number of parameters estimated. p-value = $P_r(\chi^2_{k-p} > Q^* \mid H_0)$ are computed with Q^*. In SAS/ETS, the values of Q^*_k for k = 6, 12, 18, 24 along with the corresponding p-values are given to assess the model adequacy.

3.5 Forecasting Based On Time Series

To predict future values of Y_t, t = n+1, ..., n+l, the prediction equation $\hat{Y}_{n+1} = \hat{\phi}^*_1 Y_{t-1} - \hat{\phi}^*_2 Y_{t-2} - ... - \hat{\phi}^*_{p+d} Y_{t-p-d} + \hat{\delta} + \hat{\varepsilon}_n - \hat{\theta}_1\hat{\varepsilon}_{t-1} - ... - \hat{\theta}_q \hat{\varepsilon}_{t-q}$ is used to obtain $\hat{Y}_{n+1}, \hat{Y}_{n+2}, ..., \hat{Y}_{n+l}$.

We have applied Box-Jenkins methodology to investigate the S&P 500 indexes in the following:
1. The first n = 100 observations were used to identify the most fitted ARIMA(p,d,q) model and to obtain the values of p, d, and q through iterative procedures. Then, the estimates of parameters $\hat{\phi}^*_i$ and $\hat{\theta}_j$ are calculated by least squares approach. Once the final model is obtained, the future l = 50 forecasts can be evaluated.
2. The above process is repeated for the last 50 observations of the previous 100 data and the next 50 observations until the end of data is reached.

3.6 Transfer Function Models

When employing the Box-Jenkins methodology, we use the term *transfer function model* to refer to a model that predicts future values of a time series on the basis of past values of the time series and on the basis of values of one or more other time series related to the one being predicted. Suppose X measures the level of an *input* to a system and influences the level of a system *output* Y. Because of the inertia of the system a change in X from one level to another will have effect on the output as a dynamic response. When using transfer function models to forecast, it involves identifying a model to describe the input series, identifying a preliminary transfer function model describing the output series, and using the residuals for the preliminary model to identify a model describing the error structure of the preliminary model and to form a final transfer function model.

Let X_t and Y_t be the t^{th} values of the input series and output series respectively. We assume that the same stationary transformation can be used to transform both X_t and Y_t into stationary time series values. The dynamic systems are often parsimoniously represented by the general linear difference equation $(1 + \xi_1\nabla + ... + \xi_r\nabla^r)Y_t = g(1 + \eta_1\nabla + ... + \eta_s\nabla^s)X_{t-b}$ which we refer to as *a transfer function model of order (b,r,s).* The difference equation may also be written as $(1 - \delta_1 B - ... - \delta_r B^r)Y_t = (\omega_0 - \omega_1 B - ... - \omega_s B^s)X_{t-b}$ or as $\delta(B)Y_t = \omega(B)X_{t-b}$. Hence, $Y_t - \delta_1 Y_{t-1} - ... - \delta_r Y_{t-r} = \omega_0 X_{t-b} - \omega_1 X_{t-b-1} - ... - \omega_s X_{t-b-s}$ or $Y_t = \delta^{-1}(B)\omega(B)X_{t-b}$. Let the noise of the system be η_t, the *general transfer function-noise model* is represented as $Y_t = \delta^{-1}(B)\omega(B)X_{t-b} + \eta_t$.

Box and Jenkin suggest that once we have obtained an appropriate model describing X_t, we should use this model to *prewhiten* the values of X_t and Y_t to determine their relationship. Let the prewhitened series of X_t and Y_t be $Z_t^{(x)}$, $Z_t^{(y)}$, respectively. For the $Z_t^{(x)}$ and $Z_t^{(y)}$ bivariate series, we will define the *cross-correlation coefficients between the X_t and Y_t at lag k* is defined as $r_k(X_t, Y_t) = Cov(X_t, Y_{t+k}) = E[(X_t - E(X_t))(Y_{t+k} - E(Y_t))]$, k=0, ±1, ±2, ... and the dimensionless quantity $\rho_k(X_t, Y_t) =$

$r_k(X_t,Y_t)/\sigma_x\sigma_y$ for k=0, ±1, ±2, ... is called the *cross-correlation coefficient at lag k of the bivariate process.* It is a measure of the linear relationship between the values of X_t and Y_t.

To identify (b,s,r) in the above model we will conduct the following steps:
1. After examining the sample cross-correlation (SCC) coefficient $r_k(X_t, Y_t)$, we can determine b which is the number of periods before the input series X_t begins to affect the output series Y_t. It is generally presented that there is a spike at lag b in the cross-correlation plot for k=0, ±1, ±2, ...
2. $\omega(B) = 1 - \omega_1 B - ... - \omega_s B^s$ is the $Z_t^{(x)}$ *operator of order s* where s represents the number of past $Z_t^{(x)}$ values influencing $Z_t^{(y)}$. Practice has shown that the first spike in the SCC will be followed by a *clear dying down pattern* that may be exponential or sinusoidal. The value s is set equal to the number of lags that reside between the first spike in the SCC and the beginning of the clear dying down pattern.
3. $\delta(B) = 1 - \delta_1 B - ... - \delta_r B^r$ is the $Z_t^{(y)}$ *operator of order r* where r represents the number of its past values to which $Z_t^{(y)}$ is related. If SCC dies down in a damped exponential fashion, it is reasonable to set r=1. If SCC dies down in a damped sine wave fashion, it is reasonable to set r=2.

The relationship between impulse response weight and the cross correlation is $v_k(X_t, Y_t) = \rho_k(X_t,Y_t)\sigma_y/\sigma_x$ for k=0, 1, 2, ..., and the sample impulse response weight can be calculated as $\hat{v}_k(X_t, Y_t) = r_k(X_t, Y_t)s_y/s_x$ for k=0, 1, 2, ...

After (b,s,r) being identified, we can solve the above equations with v(B), δ(B) and ω(B) involved to obtain the estimates of parameters ω_1, ..., ω_s and δ_1,..., δ_r and to calculate the residuals η_t. To identify a model describing η_t, a necessary condition for the validity of transfer function model is that the prewhitened input series $Z_t^{(x)}$ is statistically independent of the error component η_t. After examining the sample autocorrelation plot and sample partial autocorrelation of the residuals, we can determine the ARIMA(p,q) model for $\eta_t = \phi^{-1}(B)\theta(B)a_t$ and apply least squares method to the conditional sum of squares to obtain the estimates of the parameters involved.

We can use the residuals for diagnostic checking of the adequacy of the fitted transfer function model such as the white noise assumption of the residuals a_t and the zero cross correlation between input series and residuals a_t. After the adequacy of the transfer function has been checked, the prediction equation $\hat{Y}_{t+1} = \hat{\delta}_1 Y_t + ... + \hat{\delta}_r Y_{t-r+1} + \hat{\omega}_0 X_{t-b+1} - \hat{\omega}_1 X_{t-b} - ... - \hat{\omega}_s X_{t-b-s+1} + \hat{n}_{t+1}$ can be used iteratively to obtain m forecasts $\hat{Y}_{t+1}$, $\hat{Y}_{t+2}$ ..., $\hat{Y}_{t+m}$.

4. Discussion

In this research, we have trained our ANN with the S&P 500 indexes from 1971 to 1990. The ANN always forecasts the next index in the training stage, and then adjusts itself by comparing its forecast with the actual S&P 500 index.

We have also applied Box-Jenkins methodology to investigate the S&P 500 indexes. We analyzed the time series from February 11, 1981 to December 31, 1990 with a total of 2500 observations. A total of 2400 forecasting values are obtained which correspond to the S&P 500 indexes ranging from July 7, 1981 to December 31, 1990. Of forty-eight ARIMA(p,d,q) models, forty-six of them are ARIMA(0,1,0) models that are the random walk models, one of them is ARIMA(1,1,0) model, and the other is ARIMA(0,1,1) model.

The error of a particular forecast $\hat{Y}_t$ is $e_t = Y_t - \hat{Y}_t$. The overall performance of forecasting method can be best described using Mean Absolute Deviation $(MAD) = \sum_{t=1}^{n} |e_t|/n$. This measure is used to determine the magnitude of forecast errors generated by our forecasting methods.

After many experiments, we came up with an initial system configuration of our neural network with reasonable performance [12]. Comparing the forecasts from both systems with the actual indexes, we have the following results:

A. Artificial Neural Network: $MAD_N = 4.2006$ with standard error of 0.4166.
B. Time Series: $MAD_T = 9.1307$ with standard error of 0.2427.

Two interesting results are worth to mention. First, eleven wild forecasting values were made by the artificial neural network which increases the variation of the set of forecasting observations. The artificial neural network's forecasting is superior to that of the time series. If the outliers can be removed from the set of predictions, the artificial neural network will have even better performance. Second, if at some points the index data contain irregular fluctuation, then both methods cannot catch the movement of the market index although the artificial neural network seems to do a little better in this situation. The absolute deviation errors remain high during the period of fluctuation until the indexes reach certain stability.

Time series method is a powerful tool to model long-term stable observations. Our results indicate less acturate forecasting at the tail of forecasting periods. It may be improved when we try to predict less future indexes in each round. However, this is a labor-intensive task because the model identification can only be judged by the researchers from the available ARIMA(p,d,q) candidate models. For forty-eight groups of fifty forecasts each, twenty-four hundred forecasting values are obtained. On the contrast, the artificial neural network forecasts daily index very efficiently although it took some time initially to set up the right network structure.

Various methods can be applied to reduce or eliminate outliers in the forecasts of our artificial neural network. There are many different structures of hidden layer and different forward and backward propagation algorithms to be experimented in order to eliminate these outliers.

We also applied S&P 500 index as the input series and the neural network prediction as output series to the transfer function model. The model was fitted with 100 available bivariate series values, next 20 forecasts are produced along with 20 indexes. This process has been conducted repeatedly for 25 times for a total of 500 indexes. The prewhiten model are ARIMA(0,1,0) for the index and neural network, and the resulting transfer function models are (b,s,r) = (0,0,2) and (1,0,2). The residuals can be identified as white noise most of the times.

We compared the S&P 500 index with neural network forecasting, time-series forecasting and transfer function forecasting to decide whether the indexes were up or down. We define a 0.1 as tolerance interval and we say the index or forecast stays the same if it goes up or down within 0.1.

The following is the table for the index and the time-series forecasting:

		S&P 500 index			
		up	down	same	total
time-series forecasting	up	138	137	9	284
	down	132	72	10	214
	same	1	1	0	2
	total	271	210	19	500

The following is the table for the index and neural network forecasting:

		S&P 500 index			
		up	down	same	total
Neural-network forecasting	up	52	149	4	205
	down	213	58	11	282
	same	6	3	4	13
	total	271	210	19	500

The following is the table for the index and the transfer function model forecasting:

		S&P 500 index			
		up	down	same	total
transfer function model forecasting	up	112	113	10	235
	down	156	96	9	261
	same	3	1	0	4
	total	271	210	19	500

Since S&P 500 indexes are modelled as random walks in most cases of time series method, time-series forecasting seems preserve the small fluctuation of the index movement. The above tables indicate higher percentage of days time-series forecasting matching the index going up or down compared to the neural network forecasting matching the index movement. When time-series method predicts the index will go up next day, there is a probability of 138/271 to be correct, and when it predicts the index will go down there is a probability of 72/210 correct. Similar arguments can be made with respect to neural network forecasting, but the correct prediction probabilities are 52/271and 96/210. These are smaller than the probabilities of correct prediction using time-series method. However, neural network performs better in a stable market with a probability of

22/126 versus 2/126 for time-series forecasting. For the transfer function model the correct prediction probabilities are 112/271 and 96/210, respectively, these are comparable results compared with time-series method and neural network forecasting. However, neural network performs better in stable market with a probability of 22/126 versus 2/126 for time-series forecasting.

Based on the results of our comparisons, neural network forecasting is able to make closer forecasts but time-series forecasting is better in forecasting the market trends. However, these comparisons are made with a tolerance interval of 0.1 which is too small compared with the industrial standards of stock market forecasting. If the tolerance interval is changed, the probability of correct prediction will be different. Some further investigation will be made in comparing these forecasting methods in the future.

More investigation is worth to improve the performance of these systems because of the cost related to the forecasting errors and methods. For artificial neural network forecasting, more research will be done to determine the right structure and more efficient algorithms to improve its forecasting accuracy. For time series and transfer function forecasting, it will be experimented to determine the number of observations in identifying the ARIMA(p,d,q) model and the ratio of forecasting observations and the number of observations in fitting the model.

References

[1] Box, G.E.P. and G.M. Jenkins, **Time Series Analysis: Forecasting and Control**, 2nd ed., Holden-Day, San Francisco, 1976.

[2] Chen, Y.K., C.T. Su and W.T. Liou, *Classification of Mechanical Parts Using a Hybrid Recognition System*, **the Second International Computer Science Conference**, Hong Kong, December 13-16, 1992.

[3] De Mori, R., Y. Bengio and R. Cardin, *Data-driven Execution of Multi-layered Networks for Automatic Speech Recognition*, **Proc AAAI**, Saint Paul, Minnesota, 1988, pp. 734-736.

[4] Kulkarni, A.D., *Neural Nets for Image Restoration*, **Proc ACM Computer Science Conference**, Washington, D.C., 1990, pp. 373-378.

[5] Rosenblatt, F., **Principles of Neurodynamics: Perceptions and The theory of Brain Mechanisms**, Spartan, 1962.

[6] Rumelhart, D.E. and D. Zipser, *Feature Discovery by Competitive Learning*, **Cognitive Science** 9, 1985, pp. 75-112.

[7] Spartz, R.J. and S. Wu, *Comparative Implementations of An Artificial Neural Network*, **Proc. the 24th Small College Computing Symposium**, Morris, Minnesota, April 19-20, 1991, pp. 133-142.

[8] Srivastava, R.P., *Automating Judgemental Decisions Using Neural Networks: A Model for Processing Business Loan Applications*, **Proc the 20th Computer Science Conference**, Kansas City, Missouri, March 3-5, 1992, pp. 351-357.

[9] Valiant, L.G.,*Functionality in Neural Nets*, **Proc AAAI**, Saint Paul, Minnesota, 1988, pp. 629-634.

[10] Wasserman, P.D., **Neural Computing: Theory and Practice**, van Nostrand Reinhold, New York, 1989.

[11] Wu, C.H., G.M. Whitson and J.W. McLarty, *Artificial Neural System for Gene Classification Using a Domain Database*, **Proc ACM Computer Science Conference**, Washington, D.C., 1990, pp. 288-292.

[12] Wu, S., R. Lu and N. Buesing, *A Stock-Market Forecasting System With An Experimental Approach Using An Artificial Neural Network*, **Proc the 25th Small College Computing Symposium**, Grand Forks, North Dakota, April 24-25, 1992, pp. 183-192.