Forecasting Stock Price Index in Stock Exchange by Artificial Neural network

Sunandan barman
BE Computer Engineering
Maharastra Academy of Engineering
Pune, India
Sunandan.barman@gmail.com

Swapnil Agarwal
BE Computer Engineering
Maharastra Academy of Engineering
Pune , India
swapnil_a12@rediffmail.com

Abstract— **Artificial Neural Network has been shown to be an efficient tool for non-parametric modeling of data in a variety of different contexts where the output id non-linear functions of the inputs. These include business forecasting, credit scoring, bond rating, business failure prediction, medicine, pattern recognition, and image processing. A large number of studies have been reported in the literature with reference to use of ANN in modeling stock prices in the western countries. However, not much work along these lines has been reported in the Indian context. This paper presents a better prediction model by the use of neural network technique for the stock index. We will have a look at the Single layer networks and Multilayer networks and finally, the method used to train a Multilayer Networks: Backpropogation.**

Keywords-Artificial Nueral Network;BackPropogation

1. INTRODUCTION

Recently forecasting stock market return is gaining more attention, maybe because of the fact that if the direction of the market is successfully predicted the investors may be better guided. The profitability of investing and trading in the stock market to a large extent depends on the predictability. If any system be developed which can consistently predict the trends of the dynamic stock market, would make the owner of the system wealthy. More over the predicted trends of the market will help the regulators of the market in making corrective measures.

Many researchers and practitioners have proposed many models using various fundamental, technical and analytical techniques to give a more or less exact prediction. Fundamental analysis involves the in-depth analysis of the changes of the stock prices in terms of exogenous macroeconomic variables. It assumes that the share price of a stock depends on its intrinsic value and the expected return of the investors. But this expected return is subjected to change as new information pertaining to the stock is available in the market which in turn changes the share price. Moreover, the analysis of the economic factors is quite subjective as the interpretation totally lays on the intellectuality of the analyst. Alternatively, technical analysis centers on using price, volume, and open interest statistical charts to predict future stock movements. The premise behind technical analysis is that all of the internal and external factors that affect a market at any given point of time are already factored into that market's price.

Apart from these commonly used methods of prediction, some traditional time series forecasting tools are also used for the same. In time series forecasting, the past data of the prediction variable is analyzed and modeled to capture the patterns of the historic changes in the variable. These models are then used to forecast the future prices.

There are mainly two approaches of time series modeling and forecasting: linear approach and the nonlinear approach. Mostly used linear methods are moving average, exponential smoothing, time series regression etc. One of the most common and popular linear method is the Auto regressive integrated moving average (ARIMA) model (Box and Jenkins (1976)). It presumes linear model but is quite flexible as it can represent different types of time series, i.e. Auto regressive (AR), moving average (MA) and combined AR and MA (ARMA) series.

However, there is not much evidence that the stock market returns are perfectly linear for the very reason that the residual variance between the predicted return and the actual is quite high. The existence of the non-linearity of the financial market is propounded by many researchers and financial analyst.

During last few years there has been much advancement in the application of neural network in stock market indices forecasting with a hope that market patterns can be extracted. The novelty of the ANN lies in their ability to discover nonlinear relationship in the input data set without a priori assumption of the knowledge of relation between the input and the output. They independently learn the relationship inherent in the variables. From statistical point of view neural networks are analogous to nonparametric, nonlinear, regression model. So, neural network suits better than other models in predicting the stock market returns.

I. ARTIFICIAL NUERAL NETWORKS-THEORY

A neural network is a massively parallel distributed processor made up of simple processing unit which has a natural propensity for storing experiential knowledge and making it available for use. Neural networks have remarkable ability to derive meaning from complicated or imprecise data. They are used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. Neural networks have remarkable ability to derive meaning from complicated or imprecise data. They are used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques

Artificial Neural Networks were inspired by biological findings relating to the behavior of the brain as a network of units called neurons. The human brain is estimated to have around 10 billion neurons each connected on average to 10,000 other neurons. Each neuron receives signals through synapses that control the effects of the signal on the neuron. These synaptic connections are believed to play a key role in the behavior of the brain.

The fundamental building block in an ANN is the mathematical model of a neuron as shown in Figure

The three basic components of the (artificial) neuron are:

1. The synapses or connecting links that provide weights, wj , to the input values, xj *for* $j = 1, ...m$;

2. An adder that sums the weighted input values to compute the input to the activation function $v = \sum_{j=1}^{m} wjxj$, where $w0$ is called the bias (not to be confused with statistical bias in prediction or estimation) is a numerical value associated with the neuron. It is convenient to think of the bias as the weight for an input $x0$ whose value is always equal to one, so that $v = \sum_{j=0}^{m} wjxj$;

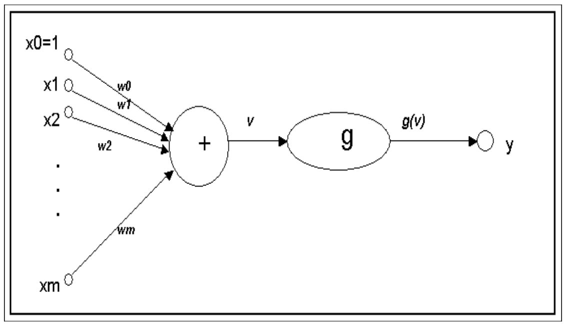


Fig.1

3. An activation function g (also called a squashing function) that maps v to $g(v)$ the output value of the neuron. This function is a monotone function.

While there are numerous different (artificial) neural network architectures that have been studied by researchers, the most successful applications in data mining of neural networks have been multilayer feedforward networks. These are networks in which there is an input layer consisting of nodes that simply accept the input values and successive layers of nodes that are neurons as depicted in Figure 1. The outputs of neurons in a layer are inputs to neurons in the next layer. The last layer is called the output layer. Layers between the input and output layers are known as hidden layers. Figure 2 is a diagram for this architecture.

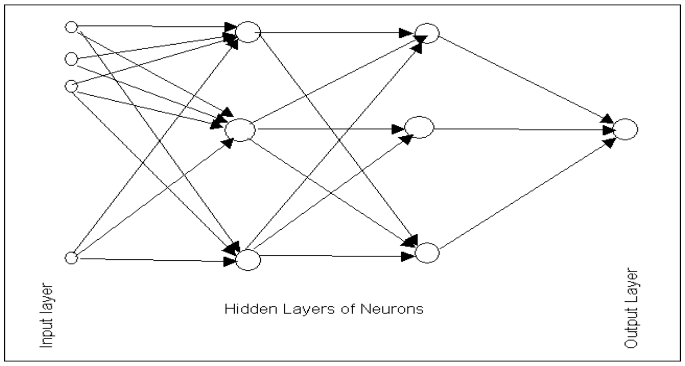


Fig.2

.

In a supervised setting where a neural net is used to predict a numerical quantity there is one neuron in the output layer and its output is the prediction. When the network is used for classification, the output layer typically has as many nodes as the number of classes and the output layer node with the largest output value gives the network's estimate of the class for a given input. In the special case of two classes it is common to have just one node in the output layer, the classification between the two classes being made by applying a cut-off to the output value at the node.

1.1 Single layer networks

Let us begin by examining neural networks with just one layer of neurons (output layer only, no hidden layers). The simplest network consists of just one neuron with the function g chosen to be the identity function, $g(v) = v$ for all v. In this case notice that the output of the network is $\sum_{j=0}^{m} wjxj$, a linear function of the input vector x with components x_j. If we are modeling the dependent variable y using multiple linear regression, we can interpret the neural network as a structure that predicts a value _y for a given input vector x with the weights being the coefficients. If we choose these weights to minimize the mean square error using observations in a training set, these weights would simply be the least squares estimates of the coefficients. The weights in neural nets are also often designed to minimize mean square error in a training data set. There is, however, a different orientation in the case of neural nets: the weights are"learned". The network is presented with cases from the training data one at a time and the weights are revised after each case in an attempt to minimize the mean square error. This process of incremental adjustment of weights is based on the error made on training cases and is known as 'training' the neural net. The almost universally used dynamic updating algorithm for the neural net version of linear regression is known as the Widrow-Hoff rule or the least-mean-square (LMS) algorithm. It is simply stated. Let $x(i)$ denote the input vector x for the i^{th} case used to train the network, and the weights *before* this case is presented to the net by the vector $w(i)$. The updating rule is $w(i+1) = w(i)+\eta(y(i)-\_y(i))x(i)$ with $w(0) = 0$. It can be shown that if the network is trained in this manner by repeatedly presenting test data observations one-at-a-time then for suitably small (absolute) values of η the network will learn (converge to) the optimal values of w. Note that the training data may have to be presented several times for $w(i)$ to be close to the optimal w. The advantage of dynamic updating is that the network tracks moderate time trends in the underlying linear model quite effectively. If we consider using the single layer neural net for classification into c classes, we would use c nodes in the output layer. If we think of classical discriminant analysis in neural network terms, the coefficients in Fisher's classification functions give us weights for the network that are optimal if the input vectors come from Multivariate Normal distributions with a common covariance matrix. For classification into two classes, the linear optimization approach that we examined in class, can be viewed as choosing optimal weights in a single layer neural network using the appropriate objective function. Maximum likelihood coefficients for logistic regression can also be considered as weights in a neural network to minimize a function of the residuals called the deviance. In this case the logistic function $g(v) =(e^v/1+e^v)$ is the activation function for the output node.

1.2 Multilayer Neural networks

Multilayer neural networks are undoubtedly the most popular networks used in applications. While it is possible to consider many activation functions, in practice it has been found that the logistic (also called the sigmoid) function $g(v) =(e^v/1+e^v)$ as the activation function (or minor variants such as the tanh function) works best. In fact the revival of interest in neural nets was sparked by successes in training neural networks using this function in place of the historically (biologically inspired) step function (the "perceptron"*)*. Notice that using a linear function does not achieve anything in multilayer networks that is beyond what can be done with single layer networks with linear activation functions. The practical value of the logistic function arises from the fact that it is almost linear in the range where g is between 0.1 and 0.9 but has a squashing effect on very small or very large values of v. In theory it is sufficient to consider networks with two layers of neurons–one hidden and one output layer–and this is certainly the case for most applications. There are, however, a number of situations where three and sometimes four and five layers have been more effective. For prediction the output node is often given a linear activation function to provide forecasts that are not limited to the zero to one range. An alternative is to scale the output to the linear part (0.1 to 0.9) of the logistic function. Unfortunately there is no clear theory to guide us on choosing the number of nodes in each hidden layer or indeed the number of layers. The common
practice is to use trial and error, although there are schemes for combining optimization methods such as genetic algorithms with network training for these parameters. Since trial and error is a necessary part of neural net applications it is important to have an understanding of the standard method used to train a multilayered network: backpropagation.

ALGORITHM:

Forward propagation is a supervised learning algorithm and describes the "flow of information" through a neural net from its input layer to its output layer.

The algorithm works as follows:

1. Set all weights to random values ranging from -1.0 to +1.0
2. Set an input pattern (binary values) to the neurons of the net's input layer

3. Activate each neuron of the following layer: - Multiply the weight values of the connections leading to this neuron with the output values of the preceding neurons - Add up these values - Pass the result to an activation function, which computes the output value of this neuron.

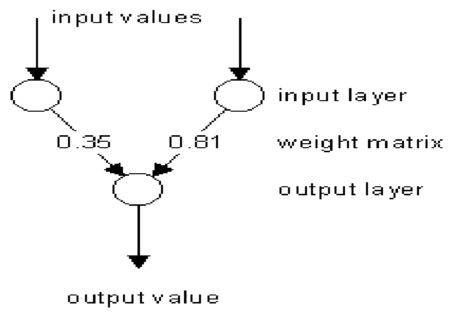
4. Repeat this until the output layer is reached.

5. Compare the calculated output pattern to the desired target pattern and compute an error value.

6. Change all weights by adding the error value to the (old) weight values.
7. Go to step 2

The algorithm ends, if all output patterns match their target patterns.

Example:
Suppose you have the following 2-layered Perceptron:



Forwardpropagation in a 2-layered Perceptron

Patterns to be learned:

input	target
0 1	0
1 1	1

First, the weight values are set to random values (0.35 and 0.81).
The learning rate of the net is set to 0.25.
Next, the values of the first input pattern (0 1) are set to the neurons of the input layer (the output of the input layer is the same as its input).

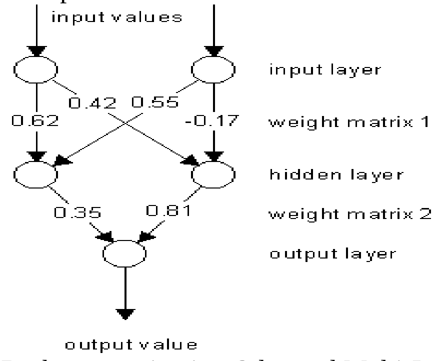
The neurons in the following layer (only one neuron in the output layer) are activated:
Input 1 of output neuron: 0 * 0.35 = 0
Input 2 of output neuron: 1 * 0.81 = 0.81
Add the inputs: 0 + 0.81 = 0.81 (= output)
Compute an error value by subtracting output from target:
0 - 0.81 = **-0.81**
Value for changing weight 1: 0.25 * 0 * (-0.81) = 0
(0.25 = learning rate)
Value for changing weight 2: 0.25 * 1 * (-0.81) = -0.2025
Change weight 1: 0.35 + 0 = 0.35 (not changed)
Change weight 2: 0.81 + (-0.2025) = 0.6075
Now that the weights are changed, the second input pattern (1 1) is set to the input layer's neurons and the activation of the output neuron is performed again, now with the new weight values:
Input 1 of output neuron: 1 * 0.35 = 0.35
Input 2 of output neuron: 1 * 0.6075 = 0.6075
Add the inputs: 0.35 + 0.6075 = 0.9575 (= output)
Compute an error value by subtracting output from target:
1 - 0.9575 = **0.0425**
Value for changing weight 1: 0.25 * 1 * 0.0425 = 0.010625
Value for changing weight 2: 0.25 * 1 * 0.0425 = 0.010625
Change weight 1: 0.35 + 0.010625 = 0.360625
Change weight 2: 0.6075 + 0.010625 = 0.618125
That was one learning step. Each input pattern had been propagated through the net and the weight values were changed.
The error of the net can now be calculated by adding up the squared values of the output errors of each pattern:
Compute the net error: $(-0.81)^2 + (0.0425)^2 = \mathbf{0.65790625}$

By performing this procedure repeatedly, this error value gets smaller and smaller.

Back propagation is a supervised learning algorithm and is mainly used by Multi-Layer-Perceptrons to change the weights connected to the net's hidden neuron layer(s).

The backpropagation algorithm uses a computed output error to change the weight values in backward direction.
To get this net error, a forwardpropagation phase must have been done before. While propagating in forward direction, the neurons are being activated using the sigmoid activation function.
The formula of **sigmoid activation** is:f(x)= $\frac{1}{1+e^{-input}}$

The algorithm works as follows:
1. Perform the forwardpropagation phase for an input pattern and calculate the output error
2. Change all weight values of each weight matrix using the formula weight (old) + learning rate * output error * output (neurons i) * output (neurons i+1) * (1 - output (neurons i+1))
3. Go to step 1
4. The algorithm ends, if all output patterns match their target patterns
Example:
Suppose you have the following 3-layered Multi-Layer-Perceptron:



Backpropagation in a 3-layered Multi-Layer-Perceptron
Patterns to be learned:

input	target
0 1	0
1 1	1

First, the weight values are set to random values: 0.62, 0.42, 0.55, -0.17 for weight matrix 1 and 0.35, 0.81 for weight matrix 2.
The learning rate of the net is set to 0.25.
Next, the values of the first input pattern (0 1) are set to the neurons of the input layer (the output of the input layer is the same as its input).

The neurons in the hidden layer are activated:
Input of hidden neuron 1: 0 * 0.62 + 1 * 0.55 = 0.55
Input of hidden neuron 2: 0 * 0.42 + 1 * (-0.17) = -0.17
Output of hidden neuron 1: 1 / (1 + exp(-0.55)) = 0.634135591
Output of hidden neuron 2: 1 / (1 + exp(+0.17)) = 0.457602059
The neurons in the output layer are activated:
Input of output neuron: 0.634135591 * 0.35 + 0.457602059 * 0.81 = 0.592605124
Output of output neuron: 1 / (1 + exp(-0.592605124)) = 0.643962658
Compute an error value by subtracting output from target: 0 - 0.643962658 = **-0.643962658**
Now that we got the output error, let's do the backpropagation.
We start with changing the weights in weight matrix 2:
Value for changing weight 1: 0.25 * (-0.643962658) * 0.634135591*0.643962658*(1-0.643962658)=-.023406638
Value for changing weight 2: 0.25 * (-0.643962658) * 0.457602059*0.643962658*(1-0.643962658)=-.016890593
Change weight 1: 0.35 + (-0.023406638) = 0.326593362
Change weight 2: 0.81 + (-0.016890593) = 0.793109407
Now we will change the weights in weight matrix 1:
Value for changing weight 1:
0.25* (-0.643962658) * 0* 0.634135591 * (1-0.634135591) = 0
Value for changing weight 2:
0.25* (-0.643962658) * 0* 0.457602059 * (1-0.457602059) = 0
Value for changing weight 3:
0.25* (-0.643962658) * 1* 0.634135591 * (1-0.634135591) = -0.037351064
Value for changing weight 4:
0.25*(-0.643962658) * 1* 0.457602059 * (1-0.457602059) = -0.039958271
Change weight 1: 0.62 + 0 = 0.62 (not changed)
Change weight 2: 0.42 + 0 = 0.42 (not changed)
Change weight 3: 0.55 + (-0.037351064) = 0.512648936
Change weight 4: -0.17+ (-0.039958271) = -0.209958271

The algorithm is successfully finished, if the net error is zero (perfect) or approximately zero.

Note that this algorithm is also applicable for Multi-Layer-Perceptrons with more than one hidden layer.

Let's outline the steps that we need to take to use the Neural Network .

1. First of all, we need data. Here we are taking Stock Index as the data. The only criteria is - the data must be sequential (a table with numbers in it's cells is a good example).

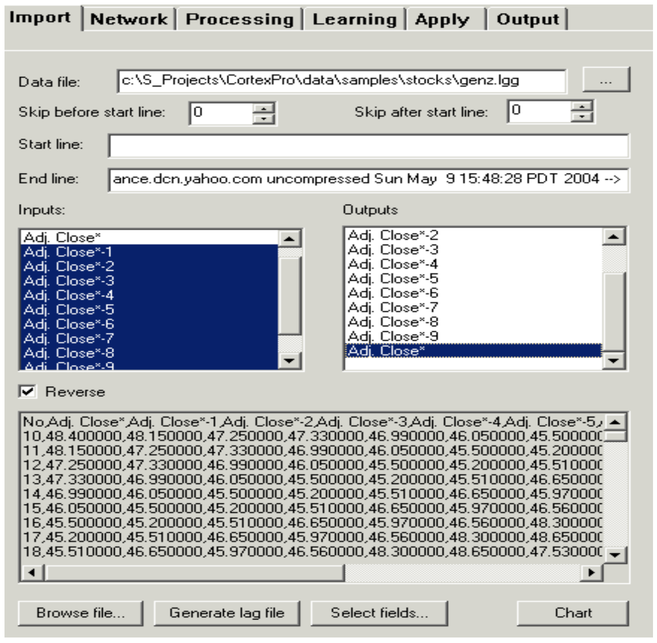
2. These data need to be fed to the *artificial neural networks application* one row in a time. Let's say, we want to do stock trading .To use a "one row in a time" approach, we need to make sure that this row contains all the historical data we need, for example, it can contain the today's data in the column one, the yesterday's data in a column two, and so on
3. We need to choose a *Neural Network* configuration - number of neurons, activation type and so on. The GUI presents a simple visual interface that allows us to do just that.
4. What we do next is *training neural network*. To do it, we run it against part of the data in the "back propagation" mode, using another part of the data to test the performance of the net. The first part of the data will be called a learning data set, the second part is called a "testing" data set. As we doing it, a *neural networks optimization* occurs.
5. After the *neural networks optimization* (training) is completed, we can use it on the "real" data. For example, we may teach it on the stock quotes for the last year, and then we expect it to predict the tomorrow's price, based on the price for today, and couple of days of history.
To do it we need to generate a lag file, and to run the data rows we want to analyze through the Neural Network. The resulting file will have the following columns:
Columns for the input : What we present to the Neural Network.
Columns for the output - what we were trying to predict.

Finally, after the Neural Network is created, we need to somehow call it from the trading software of our choice: TradeStation, MetaStocks, MetaTrader. This way we can test our GUI against real-world scenarios and get immediate results.
Click the "Select fields...". The first (after we found the start pattern and skipped extra lines) line will be broken into the column names and presented in the list box for inputs and outputs.
Select "Adj. Close*" both for the input and output. We are going to use past values of the stock closing price, to predict future values.
The "Adj. Close*" is now selected in the list boxes. We are about to generate what is called a lag file. The idea of the lag file is to represent today's data in the same table, side by side with the data for yesterday and so on. Press the "lag file" button. You will have the file with .lgg extention containing something like:

No	Close	Close-1	Close-2	...
10	5.200000	5.150000	5.180000	...
11	5.150000	5.180000	5.120000	...
12	5.180000	5.120000	5.100000	...

As you can see, the first value in Close-1 column was removed, and the entire column moved up. For the Close-2, TWO first values were removed and so on. Therefore, each line of this new file contains data for the current day AND data for nine previous days.

Let's select the new inputs. Click on the "Select fields" again, and select "Adj. Close*-1", "Adj. Close*-2"... "Adj. Close*-9" as inputs and "Adj. Close*" as output. This way we will be using nine PREVIOUS days to predict the coming price.



Data file: c:\S_Projects\CortexPro\data\samples\stocks\genz.lgg
Skip before start line: 0
Skip after start line: 0
Start line:
End line: ance.dcn.yahoo.com uncompressed Sun May 9 15:48:28 PDT 2004 -->

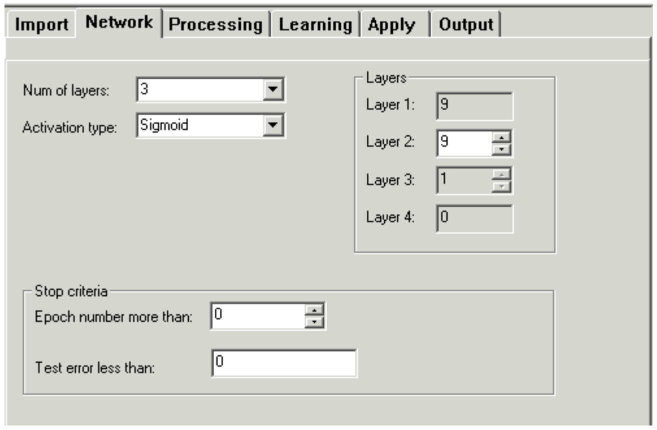
Inputs: Adj. Close*, Adj. Close*-1, Adj. Close*-2, Adj. Close*-3, Adj. Close*-4, Adj. Close*-5, Adj. Close*-6, Adj. Close*-7, Adj. Close*-8, Adj. Close*-9

Outputs: Adj. Close*-2, Adj. Close*-3, Adj. Close*-4, Adj. Close*-5, Adj. Close*-6, Adj. Close*-7, Adj. Close*-8, Adj. Close*-9, Adj. Close*

[✓] Reverse

```
No,Adj. Close*,Adj. Close*-1,Adj. Close*-2,Adj. Close*-3,Adj. Close*-4,Adj. Close*-5,
10,48.400000,48.150000,47.250000,47.330000,46.990000,46.050000,45.50000
11,48.150000,47.250000,47.330000,46.990000,46.050000,45.500000,45.20000
12,47.250000,47.330000,46.990000,46.050000,45.500000,45.200000,45.51000
13,47.330000,46.990000,46.050000,45.500000,45.200000,45.510000,46.65000
14,46.990000,46.050000,45.500000,45.200000,45.510000,46.650000,45.97000
15,46.050000,45.500000,45.200000,45.510000,46.650000,45.970000,46.56000
16,45.500000,45.200000,45.510000,46.650000,45.970000,46.560000,48.30000
17,45.200000,45.510000,46.650000,45.970000,46.560000,48.300000,48.65000
18,45.510000,46.650000,45.970000,46.560000,48.300000,48.650000,47.53000
```

Click on the "Network" tab.



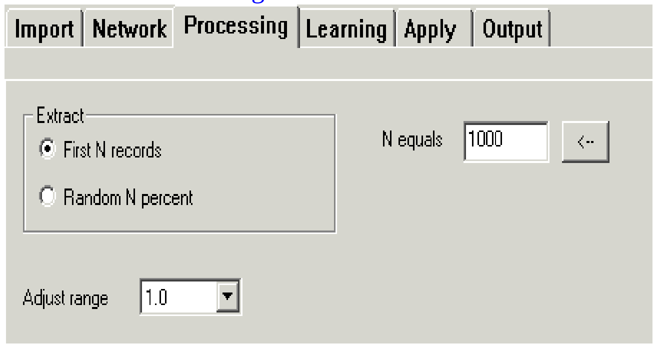
Num of layers: 3
Activation type: Sigmoid

Layers
Layer 1: 9
Layer 2: 9
Layer 3: 1
Layer 4: 0

Stop criteria
Epoch number more than: 0
Test error less than: 0

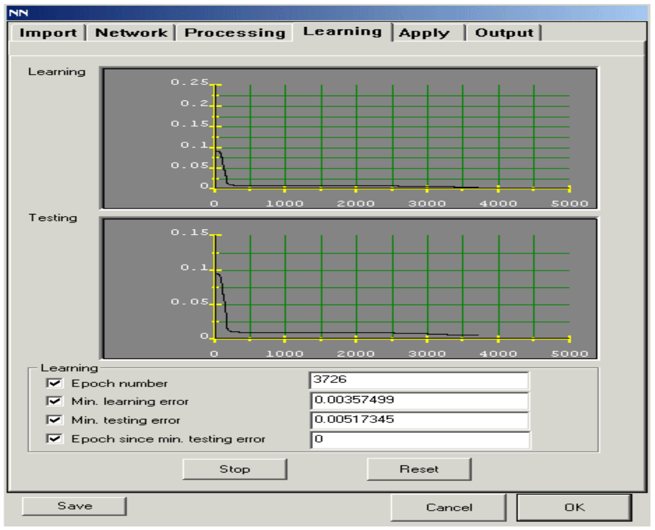
As you can see, you can specify the number of layers, number of neurons in hidden layers one of two activation functions , and stop criteria, if you want the learning process to stop automatically.

For this particular task let's select 7 neurons in the hidden layer.
Click on the "Processing" tab.



Extract
(● First N records) ○ Random N percent
N equals 1000
Adjust range 1.0

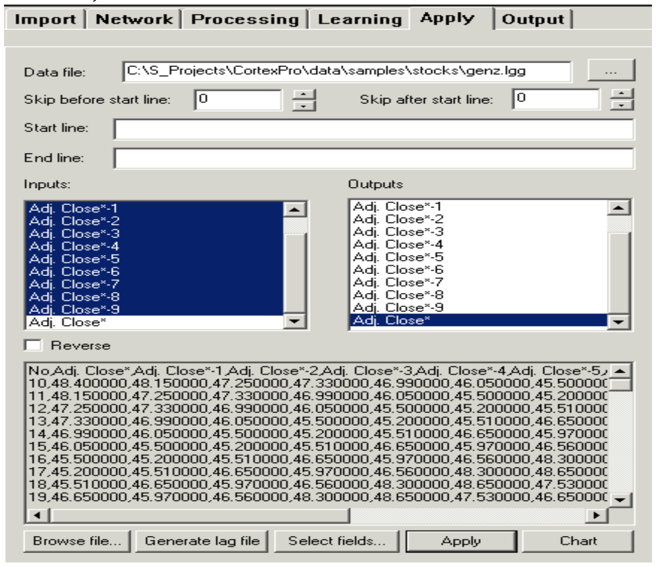
Here you can specify one of two ways of breaking the data to the "learning" and "testing" parts. You can use first N records (patterns) as a "learning" material and the rest - as a "testing" data. Or you can randomly select N % of the data. For this example we will choose the "First N records" option.
The "adjust range" combo box is important if there is a chance for our "test" data to get out from the range where the "learning" data are. To compensate, we can extend the range. Let's leave it at 1.0 as in our case the last 228 records are not out of range of the first 1000.
Click the "Learning" tab, select all check boxes and press Run. The Neural Network will begin the learning process. The number of epochs (how many times the entire data set was presented to the network), and the best (smallest) learning and testing errors will be displayed



Learning
[✓] Epoch number: 3726
[✓] Min. learning error: 0.00357499
[✓] Min. testing error: 0.00517345
[✓] Epoch since min. testing error: 0

As the learning continues, the error (representing the difference between the actual output of the network and the desired output) is decreasing. When we decide that it is small enough (and we can always go back and continue the training) we click "Stop" and go to the "Apply" tab.

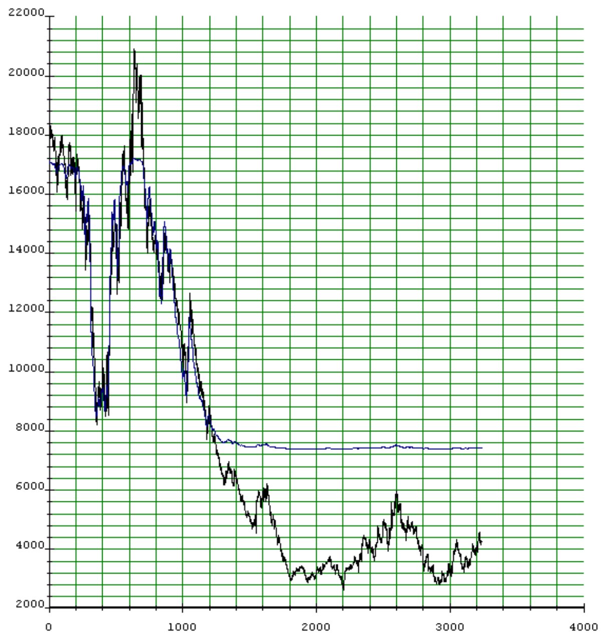
The "Apply" tab is an exact copy of a "Input" tab except for the "Chart" and "Apply" buttons. The functionality is different, however.



Data file: C:\S_Projects\CortexPro\data\samples\stocks\genz.lgg
Skip before start line: 0
Skip after start line: 0
Start line:
End line:
Inputs:
Adj. Close*-1
Adj. Close*-2
Adj. Close*-3
Adj. Close*-4
Adj. Close*-5
Adj. Close*-6
Adj. Close*-7
Adj. Close*-8
Adj. Close*-9
Adj. Close*
Outputs
Adj. Close*-1
Adj. Close*-2
Adj. Close*-3
Adj. Close*-4
Adj. Close*-5
Adj. Close*-6
Adj. Close*-7
Adj. Close*-8
Adj. Close*-9
Adj. Close*
Reverse

```
No,Adj. Close*,Adj. Close*-1,Adj. Close*-2,Adj. Close*-3,Adj. Close*-4,Adj. Close*-5,
10,48.400000,48.150000,47.250000,47.330000,46.990000,46.050000,45.50000
11,48.150000,47.250000,47.330000,46.990000,46.050000,45.500000,45.20000
12,47.250000,47.330000,46.990000,46.050000,45.500000,45.200000,45.51000
13,47.330000,46.990000,46.050000,45.500000,45.200000,45.510000,46.65000
14,46.990000,46.050000,45.500000,45.200000,45.510000,46.650000,45.97000
15,46.050000,45.500000,45.200000,45.510000,46.650000,45.970000,46.56000
16,45.500000,45.200000,45.510000,46.650000,45.970000,46.560000,48.30000
17,45.200000,45.510000,46.650000,45.970000,46.560000,48.300000,48.65000
18,45.510000,46.650000,45.970000,46.560000,48.300000,48.650000,47.53000
19,46.650000,45.970000,46.560000,48.300000,48.650000,47.530000,46.65000
```

Select the No (record number) as the input and Close and NN:Close as outputs - we are going to plot the Close and Predicted Close together on the same chart, to be able to compare them visually.
Use the "Chart" button, to plot the desired output (Close) vs. the output, produced by the Neural Net.
The following image is produced by the trained Neural Net



CONCLUSION

The predicted and the actual values are negligible difference in their values. Thus we conclude that the network is sufficiently trained and can predict stock market indexes accurately.

ACKNOWLEDGMENT

We gratefully acknowledge the able guidance of Mr. Soumitra Das , Assistant Professor in Maharastra Academy of Engineering without whose help the work done was not possible.

REFERENCES

[1] Abhyankar, A., Copeland, L. S., & Wong, W. (1997). Uncovering nonlinear structure in real-time stockmarket indexes: The S&P 500, the DAX, the Nikkei 225, and the FTSE-100. Journal of Business & Economic Statistics, 15, 1–14.

[2] Austin, M. Looney, C., & Zhuo, J. (1997). Security market timing using neural network models. New Review of Applied Expert Systems, 3, 3–14.

[3] Box, G. E. P., & Jenkins, G. M. (1970). Time series analysis: Forecasting and control, Holden Day.

[4] Brownstone, D., (1996). Using percentage accuracy to measure neural network predictions in stock market movements. Neurocomputing, 10, 237–250.

[5] Goutam Dutta, Pankaj Jha, Arnab Kumar Laha and Neeraj Mohan Artificial Neural Network Models for Forecasting Stock Price Index in the Bombay Stock Exchange Journal of Emerging Market Finance, Vol. 5, No. 3, 283-295.

[6] Kuvayev Leonid, (1996) Predicting Financial Markets with Neural Networks