Financial Time Series Forecasting by Neural Network Using Conjugate Gradient Learning Algorithm and Multiple Linear Regression Weight Initialization

CHAN Man-Chung, WONG Chi-Cheong, LAM Chi-Chung
Department of Computing
The Hong Kong Polytechnic University
Kowloon, Hong Kong
csmcchan@comp.polyu.edu.hk csccwong@comp.polyu.edu.hk

Abstract

Multilayer neural network has been successfully applied to the time series forecasting. Steepest descend, a popular learning algorithm for backpropagation network, converges slowly and has the difficulty in determining the network parameters. In this paper, conjugate gradient learning algorithm with restart procedure is introduced to overcome these problems. Also, the commonly used random weight initialization does not guarantee to generate a set of initial connection weights close to the optimal weights leading to slow convergence. Multiple linear regression (MLR) provides a better alternative for weight initialization.

The daily trade data of the listed companies from Shanghai Stock Exchange is collected for technical analysis with the means of neural networks. Two learning algorithms and two weight initializations are compared. The results find that neural networks can model the time series satisfactorily, whatever which learning algorithm and weight initialization are adopted. However, the proposed conjugate gradient with MLR weight initialization requires a lower computation cost and learns better than steepest decent with random initialization.

Keywords: time series forecasting, technical analysis, learning algorithm, conjugate gradient, multiple linear regression weight initialization, backpropagation neural network

1. Introduction

Detecting trends of stock data is a decision support process. Although the Random Walk Theory claims that price changes are serially independent, traders and certain academics[4] have observed that there is no efficient market. The movements of market price are not random and predictable.

Statistical methods and neural networks are commonly used for time series prediction. Empirical results have shown that Neural Networks outperform linear regression[1,18,32] since stock markets are complex, nonlinear, dynamic and chaotic[22]. Neural networks are reliable for modeling nonlinear, dynamic market signals[15]. Neural Network makes very few assumptions as opposed to normality assumptions commonly found in statistical methods. Neural network can perform prediction after learning the underlying relationship between the input variables and outputs. From a statistician's point of view, neural networks are analogous to nonparametric, nonlinear regression models.

Backpropagation neural network is commonly used for price prediction. Classical backpropagation adopts first-order steepest descent technique as learning algorithm. Weights are modified in a direction that corresponds to the negative gradient of the error surface. Gradient is an extremely local pointer and does not point to global minimum. This hill-climbing search is in zigzag motion and may move towards a wrong direction, getting stuck in a local minimum. The direction may be spoiled by subsequent directions, leading to slow convergence.

In addition, classical backpropagation is sensitive to the parameters such as learning rate and momentum rate. For examples, the value of learning rate is critical in the sense that too small value will make have slow convergence and too large value will make the search direction jump wildly and never converge. The optimal values of the parameters are difficult to find and often obtained empirically.

Customary random weight initialization does not guarantee a good choice of initial weight values. The random weights may be far from a good solution or near local minima or saddle points of the error surface, leading to a slow learning.

To overcome the deficiencies of steepest descent learning and random weight initialization, some researches[19,29] have investigated the use of Genetic Algorithms and Simulated Annealing to escape local minimum. Some[5,20]

have attempted Orthogonal Least Squares. Some have adopted Newton-Raphson and Levenberg-Marquardt.

In this paper, conjugate gradient learning algorithm and multiple linear regression weight initialization are attempted. In next section, conjugate gradient learning algorithm is introduced. Section 3 mentions multiple linear regression weight initialization. The descriptions and the results of experiments on the performance of both learning algorithms and both weight initializations are reported in section 4. Finally, conclusion is drawn and further research is discussed in section 5.

2. Conjugate Gradient Learning Algorithm

The training phase of a backpropagation network is an unconstrained nonlinear optimization problem. The goal of the training is to search an optimal set of connection weights in the manner that the errors of the network output can be minimized.

Besides popular steepest descent algorithm, conjugate gradient algorithm is another search method that can be used to minimize network output error in conjugate directions. Conjugate gradient method uses orthogonal and linearly independent non-zero vectors. Two vectors $\boldsymbol{d}_i$ and $\boldsymbol{d}_j$ are mutually $\boldsymbol{G}$-conjugate if

$$\boldsymbol{d}_i^T \boldsymbol{G} \boldsymbol{d}_j = 0 \qquad for \quad i \neq j \tag{1}$$

The algorithm was firstly developed to minimize a quadratic function of *n* variables

$$f(\boldsymbol{w}) = \boldsymbol{c} - \boldsymbol{b}^T \boldsymbol{w} + \frac{1}{2} \boldsymbol{w}^T \boldsymbol{G} \boldsymbol{w} \tag{2}$$

where $\boldsymbol{w}$ is a vector with *n* elements and $\boldsymbol{G}$ is an $n \times n$ symmetric and positive definite matrix. The algorithm was then extended to minimization of general non-linear functions by interpreting (2) as a second order Taylor series expansion of the objective function. $\boldsymbol{G}$ in (2) is regarded as Hessian matrix of function *f*.

A starting point $\boldsymbol{w}_1$ is selected first. The first search direction $\boldsymbol{d}_1$ is set to negative gradient $\boldsymbol{g}_1$ (i.e. $\boldsymbol{d}_1 = -\boldsymbol{g}_1$). Conjugate gradient method is to minimize differentiable function (2) by generating a sequence of approximation $\boldsymbol{w}_{k+1}$ iteratively according to

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \alpha_k \boldsymbol{d}_k \tag{3}$$

$$\boldsymbol{d}_{k+1} = -\boldsymbol{g}_{k+1} + \beta_k \boldsymbol{d}_k \tag{4}$$

α and b are momentum terms to avoid oscillations.

Let $m = \frac{1}{1 + \boldsymbol{b}_k}$. Equation (4) can be rewritten as:

$$\boldsymbol{d}_{k+1} = \frac{1}{\mu} [\mu(-\boldsymbol{g}_{k+1}) + (1-\mu)\boldsymbol{d}_k] \tag{5}$$

The search direction can be viewed as a convex combination of the current steepest descent direction and the direction used in the last move.

The search distance of each direction is varied. The value of α_k can be determined by line search techniques, such as Golden Search and Brent's Algorithm, in the way that $f(\boldsymbol{w}_k + \alpha_k \boldsymbol{d}_k)$ is minimized along the direction $\boldsymbol{d}_k$, given fixed $\boldsymbol{w}_k$ and fixed $\boldsymbol{d}_k$.

b_k can be calculated by the following three formulae:

Hestenes and Stiefel's formula,

$$\beta_k = \frac{\boldsymbol{g}_{k+1}^T [\boldsymbol{g}_{k+1} - \boldsymbol{g}_k]}{\boldsymbol{d}_k^T [\boldsymbol{g}_{k+1} - \boldsymbol{g}_k]} \tag{6}$$

Polak and Ribiere's formula,

$$\beta_k = \frac{\boldsymbol{g}_{k+1}^T [\boldsymbol{g}_{k+1} - \boldsymbol{g}_k]}{\boldsymbol{g}_k^T \boldsymbol{g}_k} \tag{7}$$

Fletcher and Reeves' formula,

$$\beta_k = \frac{\boldsymbol{g}_{k+1}^T \boldsymbol{g}_{k+1}}{\boldsymbol{g}_k^T \boldsymbol{g}_k} \tag{8}$$

Shanno's inexact line search[15] considers the conjugate method as a memoryless quasi-Newton method. Shanno derives a formula for computing $\boldsymbol{d}_{k+1}$:

$$\boldsymbol{d}_{k+1} = -\boldsymbol{g}_{k+1} - \left[\left(1 + \frac{\boldsymbol{y}_k^T \boldsymbol{y}_k}{\boldsymbol{p}_k^T \boldsymbol{y}_k} \right) \frac{\boldsymbol{p}_k^T \boldsymbol{g}_k}{\boldsymbol{p}_k^T \boldsymbol{y}_k} - \frac{\boldsymbol{y}_k^T \boldsymbol{g}_k}{\boldsymbol{p}_k^T \boldsymbol{y}_k} \right] \boldsymbol{p}_k + \frac{\boldsymbol{p}_k^T \boldsymbol{g}_k}{\boldsymbol{p}_k^T \boldsymbol{y}_k} \boldsymbol{y}_k$$

where $\boldsymbol{p}_k = \alpha_k \boldsymbol{d}_k$ and $\boldsymbol{y}_k = \boldsymbol{g}_{k+1} - \boldsymbol{g}_k$. (9)

The method performs an approximate line minimization in a descent direction in order to increase numerical stability.

For *n*-dimensional quadratic problems, the solution is converged from $\boldsymbol{w}_0$ to $\boldsymbol{w}^*$ by *n* step moves along different $\boldsymbol{G}$-conjugate directions $\boldsymbol{d}_1, \boldsymbol{d}_2, ..., \boldsymbol{d}_n$. However, for non-quadratic problems, $\boldsymbol{G}$-conjugacy of the direction vectors deteriorates. Therefore, the direction vector is reinitialized to the negative gradient of the current point after every *n* steps. That is,

$$\boldsymbol{d}_k = -\boldsymbol{g}_k \quad \text{where } k = mn + 1, \quad m \in N \tag{10}$$

Conjugate gradient method has a second-order convergence property without complex calculation of the Hessian matrix. A faster convergence is expected than first order steepest descent approach. Conjugate-gradient approach finds the optimal weight vector $\boldsymbol{w}$ along the current gradient by doing a line-search. It computes the gradient at the new point and projects it onto the subspace defined by the complement of the space defined by all previously chosen gradients. The new direction is orthogonal to all previous

search directions. The method is simple. No parameter is involved. It requires little storage space and expected to be efficient.

The summary of conjugate gradient algorithm is describe below:

1. Set $k = 1$. Initialize $\boldsymbol{w}_1$.
2. Compute $\boldsymbol{g}_1 = \nabla f(\boldsymbol{w}_1)$.
3. Set $\boldsymbol{d}_1 = -\boldsymbol{g}_1$.
4. Compute α_k by line search,
   where $\alpha_k = arg\ min_\alpha [\ f(\boldsymbol{w}_k + \alpha_k \boldsymbol{d}_k)]$.
5. Update weight vector by $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \alpha_k \boldsymbol{d}_k$.
6. If network error is less than a pre-set minimum value or the maximum number of iterations has been reached, stop; else go to step 7.
7. If $k+1 > n$, then $\boldsymbol{w}_1 = \boldsymbol{w}_{k+1}$, $k = 1$ and go to step 2;
   Else a) set $k = k+1$
   b) compute $\boldsymbol{g}_{k+1} = \nabla f(\boldsymbol{w}_{k+1})$.
   c) compute $\hat{a}_k$.
   d) compute new direction: $\boldsymbol{d}_{k+1} = -\boldsymbol{g}_{k+1} + b_k \boldsymbol{d}_k$.
   e) go to step 4

To compute gradient in step 2 and 7b, the objective function is first defined. The aim is to minimize the network error that is dependent of the independent connection weights. The objective function is defined by the error function:

$$f(\boldsymbol{w}) = \frac{1}{2N} \sum_n \sum_j (t_{nj} - y_{nj}(\boldsymbol{w}))^2 \qquad (11)$$

where N is the number of patterns in the training set;
$\boldsymbol{w}$ is one-dimensional weight vector in which weights are ordered by layer and then by neuron;
t_{nj} and $y_{nj}(\boldsymbol{w})$ are the actual and desired outputs of the j-th output neuron for n-th pattern, respectively.

With the arguments in [34], the gradient is

$$g(\boldsymbol{w}) = \frac{1}{N} \sum_n \delta_{nj} y_{ni}(\boldsymbol{w}) \qquad (12)$$

For output nodes,

$$\delta_{nj} = -(t_{nj} - y_{nj}(\boldsymbol{w})) s_j'(net_{nj}) \qquad (13)$$

where $s_j'(net_{nj})$ is the derivative of the activation function of the input of the j-th neuron net_{nj} .

For the hidden node,

$$\delta_{nj} = s_j'(net_{nj}) \sum_k \delta_{nk} w_{jk} \qquad (14)$$

where w_{jk} is the weight from j-th to the k-th neuron.

3. Multiple Linear Regression Weight Initialization

Backpropagation is a hill-climbing technique. It runs the risk of being trapped in local optimum. The starting point of the connection weights becomes an important issue to reduce the possibility of being trapped in local optimum. Random weight initialization does not guarantee to generate a good starting point. It can be enhanced by multiple linear regression. In this method, weights between input layer and hidden layer are still initialized randomly but weights between hidden layer and output layer is obtained by multiple linear regression.

The weight w_{ij} between the input node i and the hidden node j is initialized by uniform randomization. Once input x_i^s of sample s has been fed into the input node and w_{ij}'s have been assigned values, output value R_j^s of the hidden node j can be calculated as

$$R_j^s = f(\sum_i w_{ij} x_i^s), \qquad (15)$$

where f is a transfer function. The output value of the output node can be calculated as

$$y^s = f(\sum_j v_j R_j^s) \qquad (16)$$

where v_j is the weight between the hidden layer and the output layer.

Assume sigmoid function $f(x) = \frac{1}{1+e^{-x}}$ is used as the transfer function of the network. By Taylor's expansion,

$$f(x) \cong \frac{1}{2} + \frac{x}{4} \qquad (17)$$

Applying the linear approximation in (17) to (16), we have the following approximated linear relationship between the output y and v_j's:

$$y^s = \frac{1}{2} + \frac{1}{4}(\sum_j^m v_j R_j^s) \qquad (18)$$

or $4y^s - 2 = v_1 R_1^s + v_2 R_2^s + \ldots + v_m R_m^s$

$$s = 1,2,..N \qquad (19)$$

where m is the number of hidden nodes;
N is the total number of training samples.

The set of equations in (19) is a typical multiple linear regression model. R_j^s's are considered as the regressors. v_j's can be estimated by standard regression method.

Once v_j's have been obtained, the network initialization is completed and the training starts.

4. Experiment

The daily trading data of eleven listing companies in 1994-1996 was collected from Shanghai Stock Exchange for technical analysis of stock price. The first 500 entries were used as training data. The rest 150 were testing data. The raw data is preprocessed into various technical indicators to gain insight into the direction that the stock market may be going. Ten technical indicators was selected as inputs of the neural network: the lagging input of past 5 days' change in exponential moving average ($D_1EMA(t-1)$, $D_2EMA(t-1)$, $D_3EMA(t-1)$, $D_4EMA(t-1)$, $D_5EMA(t-1)$), relative strength index on day t-1 ($RSI(t-1)$), moving average convergence-divergence on day t-1 ($MACD(t-1)$), MACD Signal Line on day t-1 ($MACD\ Signal\ Line\ (t-1)$), stochastic %K on day t-1 ($\%K(t-1)$) and stochastic %D on day t-1 ($\%D(t-1)$).

EMA is a trend-following tool that gives an average value of data with greater weight to the latest data. Difference of EMA can be considered as momentum. RSI is an oscillator which measures the strength of up versus down over a certain time interval (nine days were selected in our experiment). High value of RSI indicates a strong market and low value indicates weak markets. MACD, a trend-following momentum indicator, is the difference between two moving average of price. In our experiment, 12-day EMA and 26-day EMA were used. MACD signal line smoothes MACD. 9-day *EMA* of MACD was selected for the calculation of MACD signal line. Stochastic is an oscillator that tracks the relationship of each closing price to the recent high-low range. It has two lines: %K and %D. %K is the "raw" Stochastic. In our experiment, the Stochastic's time window was set to five for calculation of %K. %D smoothes %K – over a 3-day period in our experiment.

Neural network cannot handle wide range of values. In order to avoid difficulty in getting network outputs very close to the two endpoints, the indicators were normalized to the range [0.05, 0.95], instead of [0,1], before being input to the network.

Prediction of price change allows a larger error tolerance than prediction of exact price value, resulting in a significant improvement in the forecasting ability[8,10]. In order to smooth out the noise and the random component of data, exponential moving average of the closing price change at day t ($\Delta EMA(t)$) was selected as the output node of the network. $\Delta EMA(t)$ can then be transformed to stock closing price P_t by

$$P_t = P_{t-1} + \frac{1}{L}[\Delta EMA(t) - \Delta EMA(t-1)] + \Delta EMA(t-1)$$

A three-layer network architecture was used. The required number of hidden nodes is estimated by

No. of hidden nodes = $(M + N) / 2$

where M and N is the number of input nodes and output nodes respectively. In our network, there were ten input nodes and one output node. Hence, five hidden nodes were used.

The following scenarios have been examined:

a. Conjugate gradient with random initialization (CG/RI)
b. Conjugate gradient with multiple linear regression initialization (CG/MLRI)
c. Steepest descent with random initialization (SD/RI)
d. Steepest descent with multiple linear regression initialization (SD/MLRI)

In steepest descent algorithm, the learning rate and momentum rate was set to 0.1 and 0.5 respectively[27]. In conjugate gradient, Golden Search was used to perform the exact the line search of α. According to Bazarra's analysis[3], Polak and Ribiere's form was selected for the calculation of b. For all scenarios mentioned above, the training is terminated when mean square error (MSE) is smaller than 0.5%.

All the eleven company data were used for each of the above scenario. Each company data set ran 10 times. Figure 1 shows a sample result from testing phase.

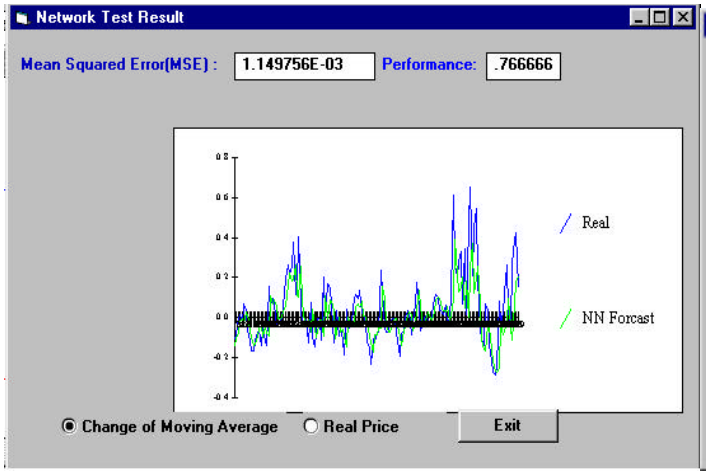

Fig 1a: Predicted $\Delta EMA(t)$ vs actual $\Delta EMA(t)$

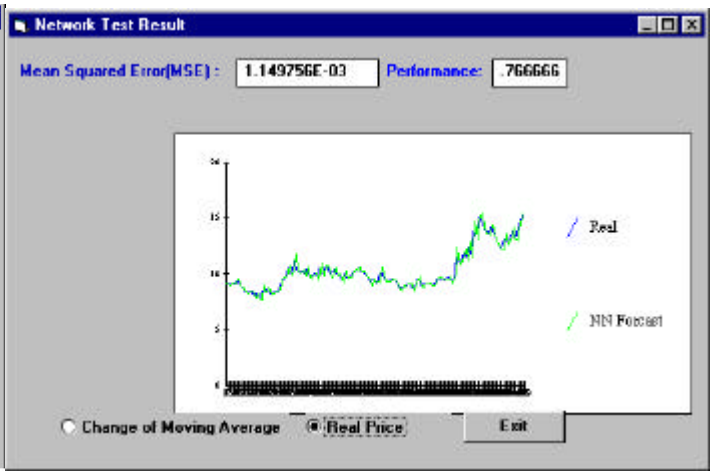

Fig 1b: Predicted stock price vs actual stock price

Figure 1: A sample result from neural network

In figure 1a, although predicted $\Delta EMA(t)$ and actual $\Delta EMA(t)$ have a relative great deviation in some regions, the network can still model the actual EMA reasonably well. On the other hand, after the transformation of $\Delta EMA(t)$ to exact price value, the deviation between actual price and predicted price is small. Two curves in figure 1b nearly coincide. This reflects the selection of the network forecaster was appropriate.

The performance of scenarios mentioned above is evaluated by average number of iterations required for training, average MSE in testing phase and the percentage of correct direction prediction in testing phase. The results are summarized in Table 1.

	Average number of iterations	Average MSE	% of correct direction prediction
CG / RI	56.636	0.001753	73.055
CG / MLRI	30.273	0.001768	73.545
SD / RI	497.818	0.001797	72.564
SD / MLRI	729.367	0.002580	69.303

Table 1: Performance evaluation for four scenarios

All scenarios, except for steepest descent with MLR initialization, achieve similar average MSE and percentage of correct direction prediction. All scenarios perform satisfactory. The mean square error produced is on average below 0.258% and more than 69% correct direction prediction is reached.

Conjugate gradient learning on average requires significant less number of iterations than steepest descent learning. Due to complexity of line search, conjugate gradient requires a longer computation time than steepest gradient per iteration. However, overall convergence of conjugate gradient neural network is still faster than steepest descent network.

In conjugate gradient network, MLR initialization requires less number of iterations required for training than random initialization, achieving similar MSE and direction prediction accuracy with random initialization. The positive result shows that regression provides a better starting point for the local quadratic approximation of the nonlinear network function performed by conjugate gradient.

However, in steepest descent network, regression initialization does not improve performance. It requires more number of iterations for training, produces a larger MSE and fewer correct direction predictions than random initialization. The phenomenon is opposite to the case in conjugate gradient network. It is attributed to the characteristics of the gradient descent algorithm that modifies direction to negative gradient of error surface, resulting in spoils of good starting point generated by MLP by subsequent directions.

5. Conclusion & Discussion

The experimental results show that it is possible to model stock price based on historical trading data by using a three-layer neural network. In general, both steepest descent network and conjugate gradient network produce the same level of error and reach the same level of direction prediction accuracy.

Conjugate gradient approach has advantages of steepest descent approach. It does not require empirical determination of network. As opposed to zigzag motion in steepest descent approach, its orthogonal search prevents a good point being spoiled. Theoretically, the convergence of second-order conjugate gradient method is faster than first order steepest descent approach. This is verified in our experiment.

In regard to initial starting point, the experimental results show the good starting point generated by multiple linear regression weight initialization is spoiled by subsequent direction in steepest descent network. On the contrary, regression initialization provides a good starting point, improving the convergence of conjugate gradient learning.

To sum up, the efficiency of backpropagation can be improved by conjugate gradient learning with multiple linear regression weight initialization.

It is believed that the computation time of conjugate gradient can be reduced by Shanno's approach[7]. The initialization scheme may be improved by estimating weights between input nodes and hidden nodes, instead of random initialization. Enrichment of more relevant inputs such as fundamental data and data from derivative markets may improve the predictability of the network. Finally, more sophisticated network architectures can be attempted for price prediction.

References

1. Arun Bansal, Robert J. Kauffman, Rob R. Weitz. Comparing the Modeling Performance of Regression and Neural Networks as Data Quality Varies: A Business Value Approach, *Journal of Management Information Systems*, Vol 10. pp.11-32, 1993.
2. Baum E.B., Haussler D. What size net gives valid generalization?, *Neural Computation* 1, pp.151-160, 1988.
3. Bazarra M.S. Nonlinear Programming, Theory and Algorithms, Ch 8, Wiley, 1993.
4. Burton G. Malkeil. A Random Walk Down Wall Street, Ch 5-8, W.W. Norton & Company, 1996.
5. Chen S., Cowan C.F.N., Grant P.M. Orthogonal Least Squares LearningAlgorithm for Radial Basis Function Networks, *IEEE Trans. Neural Network* 2(2), pp. 302-309, 1991.
6. Chris Bishop. Exact Calculation of the Hessian Matrix for the Multilayer Perceptron, *Neural Computation* 4, pp 494-501, 1992.
7. David F. Shanno. Conjugate Gradient Methods with Inexact Searches, *Mathematics of Operations Research*, Vol. 3, no. 3, 1978.
8. David M. Skapura. Building Neural Networks, Ch 5, ACM Press, Addison-Wesley.
9. Davide Chinetti, Francesco Gardin, Cecilia Rossignoli. A Neural Network Model for Stock Market Prediction, *Proc. Int'l Conference on Artificial Intelligence Applications on Wall Street*, 1993.
10. Edward Gately. Neural Networks for Financial Forecasting, Wiley, 1996.
11. Eitan Michael Azoff. Reducing Error in Neural Network Time Series Forecasting, *Neural Computing & Applications*, pp.240-247, 1993.
12. Emad W. Saad, Danil V.P., Donald C.W. Advanced Neural Network Training Methods for Low False Alarm Stock Trend Prediction, *Proc. of World Congress on Neural Networks*, Washing D.C., June 1996.
13. G. Thimm and E. Fiesler. High Order and Multilayer Perceptron Initialization, *IEEE Transactions on Neural Networks*, Vol. 8, No.2, pp.349-359, 1997.
14. Gia-Shuh Jang, Feipei Lai. Intelligent Stock Market Prediction System Using Dual Adaptive-Structure Neural Networks, *Proc. Int'l Conference on Artificial Intelligence Applications on Wall Street*, 1993.
15. Guido J. Deboeck. Trading on the Edge - Neural, Genetic, and Fuzzy Systems for Chaotic Financial Markets, Wiley, 1994.
16. Hecht-Nielsen R. Kolmogorov's mapping neural network existence theorem, *Proc. 1st IEEE Int'l Joint Conf. Neural Network*, June 1987.
17. Hong Tan, Danil V. P., Donald C. W. Probabilistic and Time-Delay Neural Network Techniques for Conservative Short-Term Stock Trend Prediction, *Proc. of World Congress on Neural Networks*, Washington D.C., July 1995.
18. Leorey Marquez, Tim Hill, Reginald Worthley, William Remus. Neural Network Models as an Alternate to Regression, *Proc. of IEEE 24th Annual Hawaii Int'l Conference on System Sciences*, pp.129-135, Vol VI, 1991.
19. Michael McInerney, Atam P. Dhawan. Use of Genetic Algorithms with Back-Propagation in training of Feed-Forward Neural Networks, *Proc. of IEEE Int'l Joint Conference on Neural Networks*, Vol. 1, pp. 203-208, 1993.
20. Mikko Lehtokangas. Initializing Weights of a Multilayer Perceptron Network by Using the Orthogonal Least Squares Algorithm, *Neural Computation*, 1995.
21. Referenes A-P., Zapranis A., Francis G. Stock Performance Modeling Using Neural Networks: A Comparative Study with Regression Models, *Neural Networks*, Vol. 7, no. 2, pp.375-388, 1994.
22. Robert R. Trippi, Jae L. Lee. Artificial Intelligence in Finance & Investing, Ch 10, IRWIN, 1996.
23. Robert R. Trippi. Neural Networks in Finance and Investing, Probus Publishing Company, 1993.
24. Roberto Battiti. First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method, *Neural Computation* 4, pp. 141-166, 1992.
25. Roger Stein. Preprocessing Data for Neural Networks, *AI Expert*, March, 1993.
26. Roger Stein. Selecting Data for Neural Networks, *AI Expert*, March, 1993.
27. Simon Haykin. Neural Networks, A Comprehensive Foundation, Ch 6, Macmillan, 1994.
28. Steven B. Achilis. Technical Analysis From A to Z, Probus Publishing, 1995.
29. Timothy Masters. Practical Neural Network Receipes in C++, Ch 4-10, Academic Press.
30. W. Press et al. Numerical Recipes in C, The Art of Scientific Computing, 2nd Edition, Cambridge University Press, 1994.
31. Wasserman P.D. Advanced Methods in Neural Computing, Van Nostrand Reinhold.
32. White H. Economic Prediction Using Neural Networks: The Case of IBM Daily Stock Returns, *Proc. of IEEE Int'l Conference on Neural Networks*, 1988.
33. Wood D., Dasgupta B. Classifying Trend Movements in the MSCI U.S.A. Capital Market Index - A comparison of Regression, ARIMA and Neural Network Methods, *Computer and Operations Research*, Vol. 23, no. 6, pp.611, 1996.
34. Rumelhart, D.E., Hinton G.E., Williams R.J. Learning Internal Representations by Error Propagation, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1: Foundations. MIT Press, 1986.