Chapter IV

Hybrid-Learning Methods for Stock Index Modeling

Yuehui Chen, Jinan University, China

Ajith Abraham, Chung-Ang University, Republic of Korea

Abstract

The use of intelligent systems for stock market prediction has been widely established. In this paper, we investigate how the seemingly chaotic behavior of stock markets could be well represented using several connectionist paradigms and soft computing techniques. To demonstrate the different techniques, we consider the Nasdaq-100 index of Nasdaq Stock Market℠ and the S&P CNX NIFTY stock index. We analyze 7-year Nasdaq 100 main-index values and 4-year NIFTY index values. This chapter investigates the development of novel, reliable, and efficient techniques to model the seemingly chaotic behavior of stock markets. We consider the flexible neural tree algorithm, a wavelet neural network, local linear wavelet neural network, and finally a feed-forward artificial neural network. The particle-swarm-optimization algorithm optimizes the parameters of the different techniques. This paper briefly explains how the different learning paradigms could be formulated using various methods and then investigates whether they can provide the required level of performance — in other

words, whether they are sufficiently good and robust so as to provide a reliable forecast model for stock market indices. Experiment results reveal that all the models considered could represent the stock indices behavior very accurately.

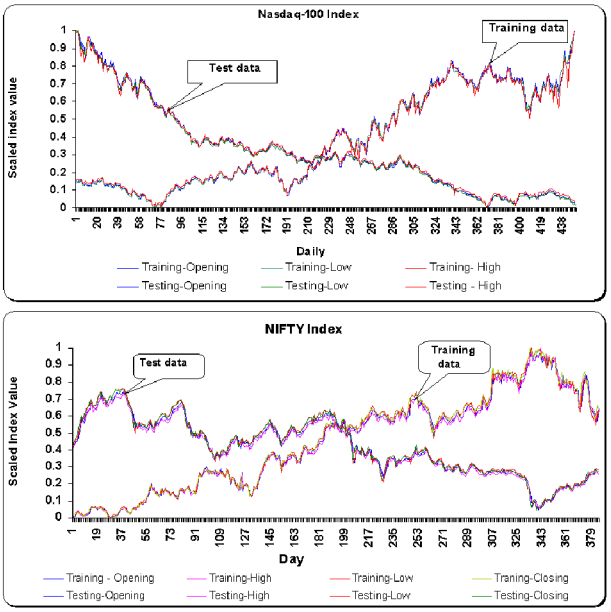
Introduction

Prediction of stocks is generally believed to be a very difficult task — it behaves like a random walk process and time varying. The obvious complexity of the problem paves the way for the importance of intelligent prediction paradigms (Abraham, Nath, & Mahanti, 2001). During the last decade, stocks and futures traders have come to rely upon various types of intelligent systems to make trading decisions (Abraham, Philip, & Saratchandran, 2003; Chan & Liu, 2002; Francis, Tay, & Cao, 2002; Leigh, Modani, Purvis, & Roberts, 2002; Leigh, Purvis, & Ragusa, 2002; Oh & Kim, 2002; Quah & Srinivasan, 1999; Wang, 2002). Several intelligent systems have in recent years been developed for modeling expertise, decision support, and complicated automation tasks (Berkeley, 1997; Bischi & Valori, 2000; Cios, 2001; Kim & Han, 2000; Koulouriotis, Diakoulakis, & Emiris, 2001; Lebaron, 2001; Palma-dos-Reis & Zahedi, 1999; Wuthrich et al., 1998). In this chapter, we analyse the seemingly chaotic behavior of two well-known stock indices namely the Nasdaq-100 index of Nasdaq^{SM} and the S&P CNX NIFTY stock index.

The Nasdaq-100 index reflects Nasdaq's largest companies across major industry groups, including computer hardware and software, telecommunications, retail/wholesale trade, and biotechnology. The Nasdaq-100 index is a modified capitalization-weighted index, designed to limit domination of the Index by a few large stocks while generally retaining the capitalization ranking of companies. Through an investment in Nasdaq-100 index tracking stock, investors can participate in the collective performance of many of the Nasdaq stocks that are often in the news or have become household names. Similarly, S&P CNX NIFTY is a well-diversified 50-stock index accounting for 25 sectors of the economy. It is used for a variety of purposes such as benchmarking fund portfolios, index-based derivatives, and index funds. The CNX indices are computed using the market capitalization weighted method, wherein the level of the index reflects the total market value of all the stocks in the index relative to a particular base period. The method also takes into account constituent changes in the index and importantly corporate actions such as stock splits, rights, and so on, without affecting the index value.

Our research investigates the performance analysis of four different connectionist paradigms for modeling the Nasdaq-100 and NIFTY stock market indices. We consider the Flexible Neural Tree (FNT) algorithm (Chen, Yang, and Dong, 2004), a Wavelet Neural Network (WNN), Local Linear Wavelet Neural Network (LLWNN) (Chen et al., 2005) and finally a feed-forward Neural Network (ANN) (Chen et al., 2004). The particle-swarm-optimization algorithm optimizes the parameters of the different techniques (Kennedy & Eberhart, 1995). We analysed the Nasdaq-100 index value from 11 January 1995 to 11 January 2002 and the NIFTY index from 01 January 1998 to 03 December 2001. For both indices, we divided the entire data into roughly two equal halves. No special rules were used to select the training set other than ensuring a reasonable representation of the

Figure 1. (a) Training and test data sets for the Nasdaq-100 index and (b) the NIFTY index



parameter space of the problem domain (Abraham et al., 2003). The complexity of the training and test data sets for both indices is depicted in Figure 1. In the section entitled "Hybrid-learning Models," we briefly describe the different learning algorithms. This is section is followed by the "Experimentation Setup and Results" section. This is, in turn, followed by the "Conclusions" section.

Particle-Swarm-Optimization (PSO) Algorithm

The PSO conducts searches using a population of particles that correspond to individuals in an Evolutionary Algorithm (EA). Initially, a population of particles is randomly generated. Each particle represents a potential solution and has a position represented by a position vector x_i. A swarm of particles moves through the problem space, with the moving velocity of each particle represented by a velocity vector v_i. At each time step, a function f_i — representing a quality measure — is calculated by using x_i as input. Each

particle keeps track of its own *best* position, which is associated with the best fitness it has achieved so far in a vector p_i. Furthermore, the *best* position among all the particles obtained so far in the population is kept track of as p_g. In addition to this global version, another version of PSO keeps track of the *best* position among all the topological neighbors of a particle. At each time step *t*, by using the individual best position, $p_i(t)$, and the global best position, $p_g(t)$, a new velocity for particle *i* is updated by:

$$v_i(t+1) = v_i(t) + c_1\phi_1(p_i(t) - x_i(t)) + c_2\phi_2(p_g(t) - x_i(t)) \quad (1)$$

where c_1 and c_2 are positive constants and ϕ_1 and ϕ_2 are uniformly distributed random numbers in $[0,1]$. The term c_i is limited to the range of $\pm V_{\max}$ (if the velocity violates this limit, it is set to its proper limit). Changing velocity this way enables the particle *i* to search around both its individual best position, p_i, and global best position, p_g. Based on the updated velocities, each particle changes its position according to:

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (2)$$

The PSO algorithm is employed to optimize the parameter vectors of FNT, ANN, and WNN.

Hybrid-Learning Models

Flexible Neural Tree Model

In this research, a tree-structured encoding method with specific instruction set is selected for representing a FNT model (Chen et al., 2005).

Flexible Neuron Instructor and FNT Model

The function set *F* and terminal instruction set *T* used for generating a FNT model are described as follows:

$$S = F \cup T = \{+_2, +_3, \cdots, +_N\} \cup \{x_1, x_2, \cdots, x_n\} \quad (3)$$

where $+_i (i = 2, 3, \cdots, N)$ denote nonleaf nodes' instructions and taking *i* arguments. $x_1, x_2, \cdots, x_n$ are leaf nodes' instructions and taking no other arguments. The output of a nonleaf

node is calculated as a flexible neuron model (see Figure 2). From this point of view, the instruction $+_i$ is also called a flexible neuron operator with i inputs.

In the construction process of a neural tree, if a nonterminal instruction, that is, $+_i (i = 2, 3, \cdots, N)$ is selected, i real values are randomly generated and used for representing the connection strength between the node $+_i$ and its children. In addition, two adjustable parameters a_i and b_i are randomly created as flexible activation function parameters.

For developing the FNT model, the following flexible activation function is used:

$$f(a_i, b_i; x) = \exp\left(-\frac{(x - a_i)^2}{b_i^2}\right) \tag{4}$$

The output of a flexible neuron $+_n$ can be calculated as follows. The total excitation of $+_n$ is:

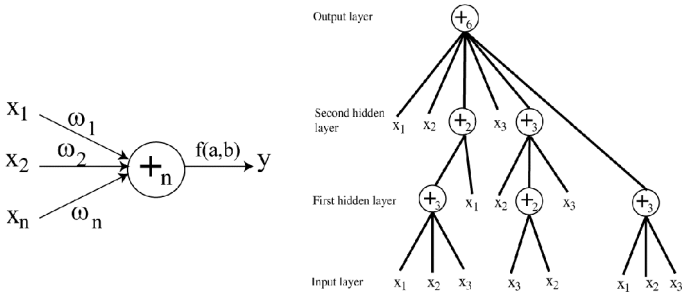
$$net_n = \sum_{j=1}^{n} w_j x_j \tag{5}$$

where $x_j (j = 1, 2, \cdots, n)$ are the inputs to node $+_n$. The output of the node $+_n$ is then calculated by:

$$out_n = f(a_n, b_n, net_n) = \exp\left(-\frac{(net_n - a_n)^2}{b_n^2}\right) \tag{6}$$

A typical flexible neuron operator and a neural tree model are illustrated in Figure 2. The overall output of the flexible neural tree can be recursively computed from left to right by the depth-first method.

Figure 2. A flexible neuron operator (left), and a typical representation of the FNT with function instruction set $F = \{+_2, +_3, \cdots, +_6\}$, and terminal instruction set $T = \{x_1, x_2, x_3\}$



Optimization of the FNT Model

The optimization of FNT includes both tree-structure and parameter optimization. Finding an optimal or near-optimal neural tree is formulated as a product of evolution. A number of neural tree variation operators are developed as follows:

- Mutation

Four different mutation operators were employed to generate offspring from the parents. These mutation operators are as follows:

(1) Changing one terminal node: Randomly select one terminal node in the neural tree and replace it with another terminal node.
(2) Changing all the terminal nodes: Select each and every terminal node in the neural tree and replace it with another terminal node.
(3) Growing: Select a random leaf in the hidden layer of the neural tree and replace it with a newly generated subtree.
(4) Pruning: Randomly select a function node in the neural tree and replace it with a terminal node.

The neural tree operators were applied to each of the parents to generate an offspring using the following steps:

(a) A Poission random number N, with mean λ, was generated.
(b) N random mutation operators were uniformly selected with replacement from the previous four-mutation operator set.
(c) These N mutation operators were applied in sequence one after the other to the parents to get the offspring.

- Crossover

Select two neural trees at random and select one nonterminal node in the hidden layer for each neural tree randomly, then swap the selected subtree. The crossover operator is implemented with a predefined probability of 0.3 in this study.

- Selection

Evolutionary-programming (EP) tournament selection was applied to select the parents for the next generation. Pairwise comparison is conducted for the union of μ parents and μ offspring. For each individual, *q* opponents are chosen uniformly at random from all the parents and offspring. For each comparison, if the individual's fitness is no smaller than the opponent's, it is selected. Then select μ individuals from parents and offspring that have most wins to form the next generation.

- Parameter Optimization by PSO

Parameter optimization is achieved by the PSO algorithm as described in the "The Particle-swarm-optimization (PSO) Algorithm" section. In this stage, the FNT architecture is fixed, as the best tree developed by the end of run of the structured search. The parameters (weights and flexible activation-function parameters) encoded in the best tree formulate a particle. The PSO algorithm works as follows:

(a) An initial population is randomly generated. The learning parameters c_1 and c_2 in PSO should be assigned in advance.

(b) The objective function value is calculated for each particle.

(c) Modification of search point — the current search point of each particle is changed using Equations 1 and 2.

(d) If the maximum number of generations is reached or no better parameter vector is found for a significantly long time (~100 steps), then stop, otherwise go to step (b).

The Artificial Neural Network (ANN) Model

A neural network classifier trained using the PSO algorithm with flexible bipolar sigmoid activation functions at hidden layer was constructed for the stock data. Before describing the details of the algorithm for training the ANN classifier, the issue of coding needs to be addressed. Coding concerns the way the weights and the flexible activation-function parameters of the ANN are represented by individuals or particles. A floating-point coding scheme is adopted here. For neural network (NN) coding, suppose there are *M* nodes in the hidden layer and one node in the output layer and *n* input variables, then the number of total weights is $n \times M + M \times 1$, the number of thresholds is $M + 1$ and the number of flexible activation-function parameters is $M + 1$, therefore the total number of free parameters in the ANN to be coded is $n \times M + M + 2(M + 1)$. These parameters are coded into an individual or particle orderly. The simple proposed training algorithm for a neural network is as follows:

Step 1: Initialization — Initial population is generated randomly. The learning parameters c_1 and c_2 in the PSO should be assigned in advance.

Step 2: Evaluation — The objective function value is calculated for each particle.

Step 3: Modification of search point — The current search point of each particle is changed using Equations 1 and 2.

Step 4: If the maximum number of generations is reached or no better parameter vector is found for a significantly long time (100 steps say), then stop, otherwise go to *Step 2*.

The WNN-Prediction Model

In terms of wavelet transformation theory, wavelets in the following form:

$$\psi = \{\psi_i = \left|a_i^{-\frac{1}{2}}\right| \varphi(\frac{x-b_i}{a_i}) : \ a_i, b_i \in R, i \in Z\} \tag{7}$$

$$x = (x_1, x_2, \ \mathbf{i}\ , x_n), \quad a_i = (a_{i1}, a_{i2}, \ \mathbf{i}\ , a_{in}), \quad b_i = (b_{i1}, b_{i2}, \ \mathbf{i}\ , b_{in})$$

are a family of functions generated from one single function $\varphi(x)$ by the operation of dilation and translation. $\varphi(x)$, which is localized in both the time space and the frequency space, is called a mother wavelet and the parameters a_i and b_i are named the scale and translation parameters, respectively.

In the standard form of a wavelet neural network, output is given by:

$$f(x) = \sum_{i=1}^{M} \omega_i \psi_i(x) = \sum_{i=1}^{M} \omega_i \left|a_i\right|^{-\frac{1}{2}} \varphi(\frac{x-b_i}{a_i}) \tag{8}$$

where ψ_i is the wavelet activation function of i-th unit of the hidden layer and ω_i is the weight connecting the i-th unit of the hidden layer to the output-layer unit. Note that for the n-dimensional input space, the multivariate wavelet-basis function can be calculated by the tensor product of n single wavelet-basis functions as follows:

$$\varphi(x) = \prod_{i=1}^{n} \varphi(x_i) \tag{9}$$

Before describing details of the PSO algorithm for training WNNs, the issue of coding needs to be addressed. Coding concerns the way the weights, dilation, and translation

parameters of WNNs are represented by individuals or particles. A floating-point coding scheme is adopted here. For WNN coding, suppose there are *M* nodes in the hidden layer and *n* input variables, then the total number of parameters to be coded is *(2n + 1)M*. The coding of a WNN into an individual or particle is as follows:

$$| a_{11}b_{11} \ \mathbf{i} \ a_{1n}b_{1n}\omega_1 \,|\, a_{21}b_{21} \ \mathbf{i} \ a_{2n}b_{2n}\omega_2 \,|\, \mathbf{i} \,|\, a_{n1}b_{n1} \ \mathbf{i} \ a_{nn}b_{nn}\omega_n \,|$$

The simple proposed training algorithm for a WNN is as follows:

Step 1: An initial population is randomly generated. The learning parameters, such as c_1, c_2 in PSO should be assigned in advance.

Step 2: Parameter optimization with PSO algorithm.

Step 3: if the maximum number of generations is reached or no better parameter vector is found for a significantly long time (~100 steps), then go to *Step 4*; otherwise go to *Step 2*.

Step 4: Parameter optimization with gradient-descent algorithm.

Step 5: If a satisfactory solution is found then stop; otherwise go to *Step 4*.

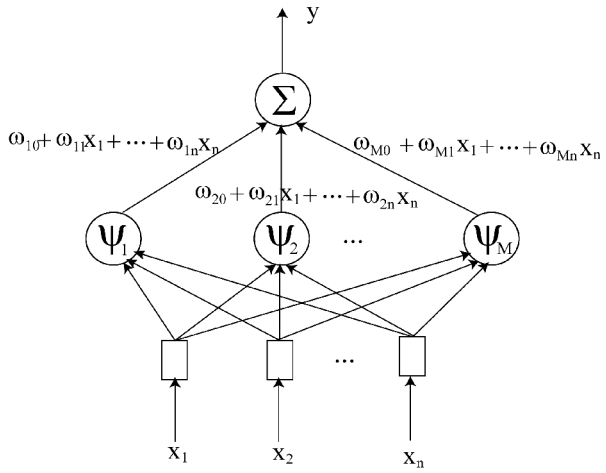
The Local Linear WNN Prediction Model

An intrinsic feature of basis-function networks is the localized activation of the hidden-layer units, so that the connection weights associated with the units can be viewed as locally accurate piecewise constant models whose validity for any given input is indicated by the activation functions. Compared to the multilayer perceptron neural network, this local capacity provides some advantages, such as learning efficiency and structure transparency. However, the problem of basis-function networks requires some special attention. Due to the crudeness of the local approximation, a large number of basis-function units have to be employed to approximate a given system. A shortcoming of the wavelet neural network is that for higher dimensional problems many hidden-layer units are needed.

In order to take advantage of the local capacity of the wavelet-basis functions while not having too many hidden units, here we propose an alternative type of WNN. The architecture of the proposed local linear WNN (LLWNN) is shown in Figure 3. Its output in the output layer is given by:

$$y = \sum_{i=1}^{M} (\omega_{i0} + \omega_{i1}x_1 + \ \mathbf{i} \ + \omega_{in}x_n)\psi_i(x) = \sum_{i=1}^{M} (\omega_{i0} + \omega_{i1}x_1 + \ \mathbf{i} \ + \omega_{in}x_n)|a_i|^{-\frac{1}{2}}\varphi(\frac{x-b_i}{a_i}) \quad (10)$$

Figure 3. Architecture of a local linear wavelet neural network



where $x = (x_1, x_2, \cdots, x_n)$. Instead of the straightforward weight ω_i (piecewise constant model), a linear model:

$$v_i = \omega_{i0} + \omega_{i1}x_1 + \wedge + \omega_{in}x_n \tag{11}$$

is introduced. The activities of the linear model v_i ($i = 1, 2, \cdots, M$) are determined by the associated locally active wavelet function $\psi_i(x)$($i = 1, 2, \cdots, M$), thus v_i is only locally significant. The motivations for introducing local linear models into a WNN are as follows: (1) Local-linear models have been studied in some neurofuzzy systems (Abraham, 2001) and offer good performances; and (2) Local-linear models should provide a more parsimonious interpolation in high-dimension spaces when modeling samples are sparse. The scale and translation parameters and local-linear-model parameters are randomly initialized at the beginning and are optimized by the PSO algorithm.

Experiment Setup and Results

We considered 7-year stock data for the Nasdaq-100 Index and 4-year for the NIFTY index. Our target was to develop efficient forecast models that could predict the index value of the following trading day based on the opening, closing, and maximum values on any given day. The training and test patterns for both indices (scaled values) are illustrated in Figure 1. We used the same training- and test-data sets to evaluate the

different connectionist models. More details are reported in the following sections. Experiments were carried out on a Pentium IV, 2.8 GHz Machine with 512 MB RAM and the programs implemented in C/C++. Test data was presented to the trained connectionist models, and the output from the network compared with the actual index values in the time series.

The assessment of the prediction performance of the different connectionist paradigms were done by quantifying the prediction obtained on an independent data set. The root-mean-squared error (*RMSE*), maximum-absolute-percentage error (*MAP*), mean-absolute-percentage error (*MAPE*), and correlation coefficient (*CC*) were used to study the performance of the trained forecasting model for the test data.

MAP is defined as follows:

$$MAP = max\left(\frac{\left|P_{actual,\ i} - P_{predicted,\ i}\right|}{P_{predicted,\ i}} \times 100\right) \tag{12}$$

where $P_{\text{actual},\ i}$ is the actual index value on day i and $P_{\text{predicted},\ i}$ is the forecast value of the index on that day. Similarly *MAPE* is given as:

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{\left|P_{actual,\ i} - P_{predicted,\ i}\right|}{P_{actual,\ i}}\right] \times 100 \tag{13}$$

where N represents the total number of days.

- FNT Algorithm

We used the instruction set $S = \{+_2, +_3, \cdots, +_{10}, x_0, x_1, x_2\}$ modeling the Nasdaq-100 index and instruction set $S = \{+_2, +_3, \cdots, +_{10}, x_0, x_1, x_2, x_3, x_4\}$ modeling the NIFTY index. We used the flexible activation function of Equation 4 for the hidden neurons. Training was terminated after 80 epochs on each dataset.

- NN-PSO Training

A feed-forward neural network with three input nodes and a single hidden layer consisting of 10 neurons was used for modeling the Nasdaq-100 index. A feed-forward neural network with five input nodes and a single hidden layer consisting of 10 neurons was used for modeling the NIFTY index. Training was terminated after 3000 epochs on each dataset.

- WNN-PSO

A WNN with three input nodes and a single hidden layer consisting of 10 neurons was used for modeling the Nasdaq-100 index. A WNN with five input nodes and a single hidden layer consisting of 10 neurons was used for modeling the NIFTY index. Training was terminated after 4000 epochs on each dataset.

- LLWNN-PSO

A LLWNN with three input nodes and a hidden layer consisting of five neurons for modeling Nasdaq-100 index. A LLWNN with five input nodes and a single hidden layer consisting of five neurons for modeling NIFTY index. Training was terminated after 4500 epochs on each dataset.

Figure 4. Test results showing the performance of the different methods for modeling the Nasdaq-100 index

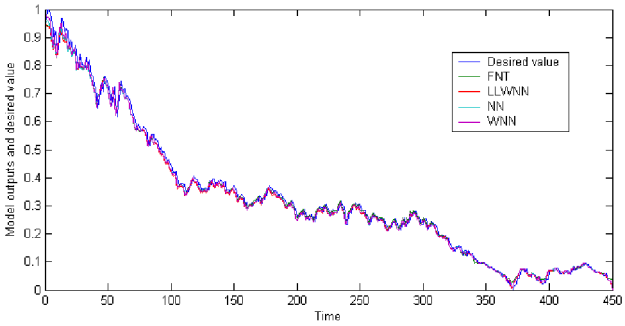


Figure 5. Test results showing the performance of the different methods for modeling the NIFTY index

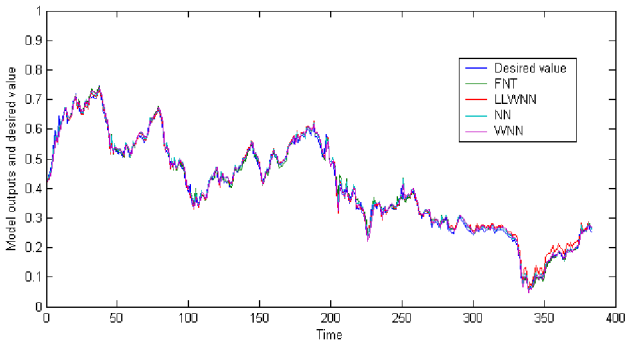


Table 1. Empirical comparison of RMSE results for four learning methods

	FNT	NN-PSO	WNN-PSO	LLWNN-PSO
	Training results			
Nasdaq-100	0.02598	0.02573	0.02586	0.02551
NIFTY	0.01847	0.01729	0.01829	0.01691
	Testing results			
Nasdaq-100	0.01882	0.01864	0.01789	0.01968
NIFTY	0.01428	0.01326	0.01426	0.01564

Table 2. Statistical analysis of four learning methods (test data)

	FNT	NN-PSO	WNN-PSO	LLWNN-PSO
	Nasdaq-100			
CC	0.997579	0.997704	0.997721	0.997623
MAP	98.107	141.363	152.754	230.514
MAPE	6.205	6.528	6.570	6.952
	NIFTY			
CC	0.996298	0.997079	0.996399	0.996291
MAP	39.987	27.257	39.671	30.814
MAPE	3.328	3.092	3.408	4.146

- Performance and Results Achieved

Table 1 summarizes the training and test results achieved for the two stock indices using the four different approaches. The statistical analysis of the four learning methods is depicted in Table 2. Figures 4 and 5 depict the test results for the 1-day-ahead prediction of Nasdaq-100 index and NIFTY index respectively.

Conclusion

In this chapter, we have demonstrated how the chaotic behavior of stock indices could be well-represented by different hybrid learning paradigms. Empirical results on the two data sets using four different learning models clearly reveal the efficiency of the proposed techniques. In terms of RMSE values, for the Nasdaq-100 index, WNN performed marginally better than the other models and for the NIFTY index, the NN approach gave the lowest generalization RMSE values. For both data sets, LLWNN had the lowest training error. For the Nasdaq-100 index (test data), WNN had the highest CC, but the lowest values of MAPE and MAP were achieved by using the FNT model. The highest CC together with the best MAPE/MAP values for the NIFTY index were achieved using the NN trained using the PSO model. A low MAP value is a crucial indicator for evaluating the stability of a market under unforeseen fluctuations. In the present example, the predictability ensures that the decrease in trade is only a temporary cyclic variation that is perfectly under control.

Our research was to predict the share price for the following trading day based on the opening, closing, and maximum values on any given day. Our experimental results indicate that the most prominent parameters that affect share prices are their immediate opening and closing values. The fluctuations in the share market are chaotic in the sense that they heavily depend on the values of their immediate forerunning fluctuations. Long-term trends exist, but are slow variations and this information is useful for long-term investment strategies. Our study focused on short-term floor trades in which the risk is higher. However, the results of our study show that even with seemingly random fluctuations, there is an underlying deterministic feature that is directly enciphered in the opening, closing, and maximum values of the index of any day making predictability possible.

Empirical results also show that there are various advantages and disadvantages for the different techniques considered. There is little reason to expect that one can find a uniformly best learning algorithm for optimization of the performance for different stock indices. This is in accordance with the no-free-lunch theorem, which explains that for any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class (Macready & Wolpert, 1997). Our future research will be oriented towards determining the optimal way to combine the different learning paradigms using an ensemble approach (Maqsood, Kahn, & Abraham, 2004) so as to complement the advantages and disadvantages of the different methods considered.

References

Abraham, A., Nath, B., & Mahanti, P. K. (2001). Hybrid intelligent systems for stock market analysis. In V. N. Alexandrov et al. (Eds.), *Computational Science* (pp. 337-345). Germany: Springer-Verlag.

Abraham, A. (2001). NeuroFuzzy systems: State-of-the-art modeling techniques. In J. Mira & A. Prieto (Eds.), *Proceedings of the 7th International Work Conference on Artificial and Neural Networks, Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence*, Granada, Spain (pp. 269-276). Germany: Springer-Verlag.

Abraham, A., Philip, N. S., & Saratchandran, P. (2003). Modeling chaotic behavior of stock indices using intelligent paradigms. *International Journal of Neural, Parallel & Scientific Computations, 11*(1-2), 143-160.

Berkeley, A. R. (1997). Nasdaq's technology floor: Its president takes stock. *IEEE Spectrum, 34*(2), 66-67.

Bischi, G. I., & Valori, V. (2000). Nonlinear effects in a discrete-time dynamic model of a stock market. *Chaos, Solitons & Fractals, 11*(13), 2103-2121.

Chan, W. S., & Liu, W. N. (2002). Diagnosing shocks in stock markets of Southeast Asia, Australia, and New Zealand. *Mathematics and Computers in Simulation, 59*(1-3), 223-232.

Chen, Y., Yang, B., & Dong, J. (2004). Nonlinear system modeling via optimal design of neural trees. *International Journal of Neural Systems, 14*(2), 125-137.

Chen, Y., Yang, B., & Dong, J. (in press). Time-series prediction using a local linear wavelet neural network. *International Journal of Neural Systems*.

Chen, Y., Yang, B., Dong, J., & Abraham, A. (in press). Time-series forecasting using flexible neural tree model. *Information Science*.

Cios, K. J. (2001). Data mining in finance: Advances in relational and hybrid methods. *Neurocomputing, 36*(1-4), 245-246.

Francis, E. H., Tay, & Cao, L. J. (2002). Modified support vector machines in financial time series forecasting. *Neurocomputing, 48*(1-4), 847-861.

Kennedy, J., & Eberhart, R. C. (1995). Particle swarm optimization. In *Proceedings IEEE International Conference on Neural Networks* (pp. 1942-1948).

Kim, K. J., & Han, I. (2000). Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert Systems with Applications, 19*(2), 125-132.

Koulouriotis, D. E., Diakoulakis, I. E., & Emiris, D. M. (2001). A fuzzy cognitive map-based stock market model: Synthesis, analysis and experimental results. In *Proceedings of the 10th IEEE International Conference on Fuzzy Systems, Vol. 1* (pp. 465-468).

LeBaron, B. (2001). Empirical regularities from interacting long- and short-memory investors in an agent-based stock market. *IEEE Transactions on Evolutionary Computation, 5*(5), 442-455.

Leigh, W., Modani, N., Purvis, R., & Roberts, T. (2002). Stock market trading rule discovery using technical charting heuristics. *Expert Systems with Applications, 23*(2), 155-159.

Leigh, W., Purvis, R., & Ragusa, J. M. (2002). Forecasting the NYSE composite index with technical analysis, pattern recognizer, neural network, and genetic algorithm: A case study in *romantic* decision support. *Decision Support Systems, 32*(4), 361-377.

Maqsood, I., Khan, M. R., & Abraham, A. (2004). Neural network ensemble method for weather forecasting. *Neural Computing & Applications, 13*(2), 112-122.

Macready, W. G., & Wolpert, D. H. (1997). *The no free lunch theorems. IEEE Transaction on Evolutionary Computing, 1*(1), 67-82.

Nasdaq Stock Market℠. (n.d.). Retrieved from http://www.nasdaq.com

National Stock Exchange of India Limited. (n.d.). Retrieved from http://www.nse-india.com

Oh, K. J., & Kim, K. J. (2002). Analyzing stock market tick data using piecewise nonlinear model. *Expert Systems with Applications, 22*(3), 249-255.

Palma-dos-Reis, A., & Zahedi, F. (1999). Designing personalized intelligent financial decision support systems. *Decision Support Systems, 26*(1), 31-47.

Quah, T. S., & Srinivasan, B. (1999). Improving returns on stock investment through neural network selection. *Expert Systems with Applications, 17*(4), 295-301.

Wang, Y. F. (2002). Mining stock price using fuzzy rough set system. *Expert Systems with Applications, 24*(1), 13-23.

Wuthrich, B., Cho, V., Leung, S., Permunetilleke, D., Sankaran, K., & Zhang, J. (1998). Daily stock market forecast from textual web data. *Proceedings IEEE International Conference on Systems, Man, and Cybernetics, 3*, 2720-2725.