Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A hybrid evolutionary dynamic neural network for stock market trend analysis and prediction using unscented Kalman filter

Ranjeeta Bisoi, P.K. Dash*

Multidisciplinary Research Cell, Siksha O Anusandhan University, Bhubaneswar, Odisha, India

A R T I C L E   I N F O

Article history:
Received 24 May 2012
Received in revised form
24 November 2013
Accepted 22 January 2014
Available online 7 February 2014

Keywords:
IIR filter NN
UKF
DE
Stock indices
Trend prediction

A B S T R A C T

Stock market prediction is of great interest to stock traders and investors due to high profit in trading the stocks. A successful stock buying/selling generally occurs near price trend turning point. Thus the prediction of stock market indices and its analysis are important to ascertain whether the next day's closing price would increase or decrease. This paper, therefore, presents a simple IIR filter based dynamic neural network (DNN) and an innovative optimized adaptive unscented Kalman filter for forecasting stock price indices of four different Indian stocks, namely the Bombay stock exchange (BSE), the IBM stock market, RIL stock market, and Oracle stock market. The weights of the dynamic neural information system are adjusted by four different learning strategies that include gradient calculation, unscented Kalman filter (UKF), differential evolution (DE), and a hybrid technique (DEUKF) by alternately executing the DE and UKF for a few generations. To improve the performance of both the UKF and DE algorithms, adaptation of certain parameters in both these algorithms has been presented in this paper. After predicting the stock price indices one day to one week ahead time horizon, the stock market trend has been analyzed using several important technical indicators like the moving average (MA), stochastic oscillators like K and D parameters, WMS%R (William indicator), etc. Extensive computer simulations are carried out with the four learning strategies for prediction of stock indices and the up or down trends of the indices. From the results it is observed that significant accuracy is achieved using the hybrid DEUKF algorithm in comparison to others that include only DE, UKF, and gradient descent technique in chronological order. Comparisons with some of the widely used neural networks (NNs) are also presented in the paper.

1. Introduction

Price index forecasting is one of the most important problems in financial markets that make the investors worry about their investments. In the past decades, the stock market prediction has played a vital role for the investment brokers and individual investors and researchers are on the constant lookout for a reliable method for predicting stock market trends. The nonlinear and nonstationary characteristics of the stock market make it difficult and challenging for forecasting stock indices in a reliable manner. Traditionally the basic methodology for financial time series has been a series of statistical methods like ARMA, ARIMA, GARCH, etc., which have been used over the years but suffer from the assumption of linear variation of the stock prices during a certain period of time. In general, the statistical models do not yield a process that can be easily automated, as it requires adaptation and changes at every stage requiring certain regularities and stationary nature in the target data. Accordingly, traditional statistical methods cannot be used to track the complexity and nonstationary nature of the stock markets.

In their search for methods for addressing these shortcomings, many market analysts, traders, and researchers have investigated the various intelligent system techniques for analyzing the stock markets and making trading decisions. The various tools of computational intelligence include artificial NN (ANN) [1–4], fuzzy logic systems [5–8], support vector machines (SVM) [9,10]. The different types of ANNs used for stock market analysis include radial basis function NN (RBFNN) [11], recurrent NN (RNN) [12,13], multilayer perceptron (MLP) [14–16], generalized regression NN (GRNN) [17], random vector functional link net (FLANN) [18,19], local linear wavelet NN (LLWNN) [20,21], wavelet NN (WNN) [22], etc. These neural models, however, do not perform so successfully due to the dimensionality, volatility, and noise of the stock price data. Fuzzy logic theory is preferred by several investigators due to its superior capabilities to handle uncertainties in the data and a synergic combination of fuzzy logic systems and NN have evolved for stock market prediction. These models include fuzzy NN (FNN) [23], adaptive network fuzzy information system (ANFIS) [24], wavelet fuzzy NN (WLFNN) [25], etc. To improve the accuracy of prediction and automate stock market forecasting and trend analysis other

* Corresponding author. Tel.: +91 674 2725336; fax: +91 674 2725312.
E-mail address: pkdash.india@gmail.com (P.K. Dash).

1568-4946/$ – see front matter © 2014 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2014.01.039

inputs like the technical indicators and qualitative factors due to political effects, etc. are used along with the price data. Even with these indicators, the NN models suffer from computational complexity because they need as many as 800 neurons in the input layer and 600 neurons in the hidden layers for predicting reasonably well stock price indices over a certain time frame. Further search techniques like the genetic algorithm (GA), particle swarm optimization (PSO) [26], bacterial foraging optimization (BFO), DE [27], etc. are used to obtain optimal parameters of the network that includes the network weights, the number of neurons in the hidden layer, etc. to improve the accuracy of the prediction.

DNN models provide an excellent means for forecasting and prediction of nonstationary time series. In particular, a neural architecture, known as locally recurrent NN (LRNN) [28], is preferred to the traditional MLP because the time varying nature of a stock time series can be better represented using LRNN. The use of LRNN has demonstrated superioriority in comparison to other NN approaches for temporal sequence processing and a number of successful applications exists in the areas of electrical load forecasting, non linear system prediction, and economic time series prediction.

Therefore, in this paper, a simple feed forward DNN comprising one or more layers of dynamic neurons is presented for forecasting stock price indices and profits from one day ahead to 30 days in advance. The DNN includes one or more IIR (infinite impulse filter) filters in the forward path providing feedback connections between outputs and inputs. This allows signal flow in both forward and backward directions, giving the network a dynamic memory useful to mimic dynamic systems. Further for globally recurrent networks, the stability is hard to be proved, but locally recurrent networks allow easy checking of the stability by the examination of poles of their internal filters However, training these networks presents difficulties due to the feedback connections. The possible algorithms that can be used for training RNNs are: (1) the back propagation through time (BPTT) [29] where the recurrent network is unfolded into a multilayer feed forward network that increases by one at each time step; (2) the real time recurrent learning (RTRL) which is based on the supervised learning method where the feedback from the output of a unit is replaced by the output of the true system in subsequent computations. However, the RTRL algorithm suffers from low convergence speed and multiple local minima.

Considering the chaotic nature of the time series prediction and noise problem, extended Kalman filter (EKF) [30] and unscented Kalman filter (UKF) [31,32] have been proposed for fast convergence and good tracking performance. A key property of the EKF is that it is the minimum-variance estimator for the state of a nonlinear dynamical system and when applied to IIR filter NN training, it produces faster convergence than gradient-based algorithms; also, it overcomes the vanishing-gradient problem. The EKF algorithm provides first-order approximations to optimal nonlinear estimation through the linearization of the nonlinear system, which might produce large errors in the true posterior mean and covariance of the transformed (Gaussian) random variable, leading to suboptimal performance, and sometimes, filter divergence. The UKF first proposed by Julier and Uhlmann [31] and further extended by Wan and van der Merwe [32], is an alternative to the EKF algorithm and provides third-order approximation of process and measurement errors for Gaussian distributions. Consequently, the UKF does not require the computation of Jacobeans, needed for linearizing the process and measurement equations and thus produces better estimates of the weights of the recurrent NN.

UKF is based on the unscented transformation (UT) theory, and has a greater ability to cope with non-linearities in one-step ahead prediction. Instead of linearizing a nonlinear system, a statistical distribution of the state is propagated through the non-linear system that provides better estimates of the actual state and the posterior covariance matrices. Thus, instead of using the standard gradient or back propagation algorithm for adjusting the weights and some parameters of the DNN, an UKF is used in this paper to provide fast convergence and better accuracy with respect to chaotic variations in the inputs. However, its accuracy significantly reduces, if Signal-to-Noise Ratio is low and the noise co variances and some of the parameters used in unscented transformation are not chosen correctly. Thus to obtain an optimal forecasting performance, it is proposed in this paper to use DE technique for optimizing the objective function obtained from the UKF algorithm alternately during the beginning of the learning cycle. Once initialized using DE, the optimized UKF algorithm updates the weights of the DNN and exhibits superior convergence in tracking a chaotic time series like the stock market indices. Unlike traditional EAs (Evolutionary algorithms), DE employs the difference of the parameter vectors to explore the objective function landscape. The advantage of DE is that there is a possibility of finding the global minimum of a multimodal function irrespective of the values of its initial parameters. Additionally, DE takes very few control parameters (typically the population size *Np*, the scale factor *F*, and the crossover rate *Cr*), which makes it easy to use.

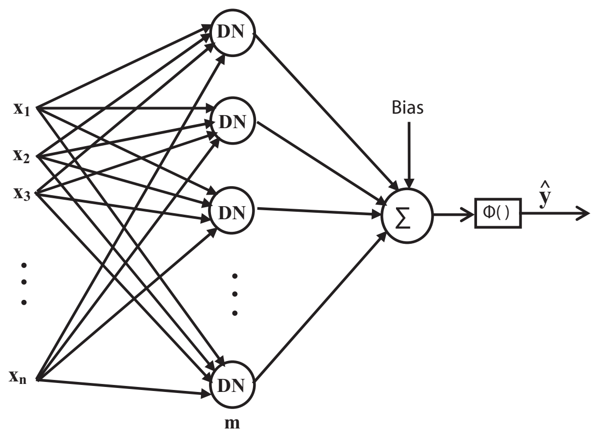
It is well known that the technical indicators play a great role in predicting the dynamic behavior of the stock market indices, and hence they can be used to predict the next day's trend whether the stock indices are going up or down. Turning point of this trend can be used as a buy or sell decision for the traders and investors in the market. The technical indicators chosen for this purpose are the 25 and 65 days moving average, relative strength index, trading volume, stochastic oscillators, and William indicator, etc. With the help of a few simple rules, the up and down trend of the stock indices can be ascertained, which can be used for automatic buying or selling decision.

This paper is organized as follows: in Section 2, the architecture of the DNN is proposed. The state space model of the UKF is described in Section 3 along with the learning strategy of the DNN and this section also deals with the DE technique and various steps in implementing the algorithm are described. In Section 4 the performance of the prediction algorithms are analyzed using various technical indicators, while the stock market trend analysis is presented in Section 5. Section 6 provides the original datasets and an overview of input selection for the various stock market data and the stock value prediction results and the errors for different data sets with and without the use of DE. Trend prediction using certain technical indicators and a pertinent rule base are also given in this section. Finally, conclusion is given in Section 7. Further some of the well known NNs like the FLANN, LLWNN, and a slightly modified RBFNN are used for comparison.

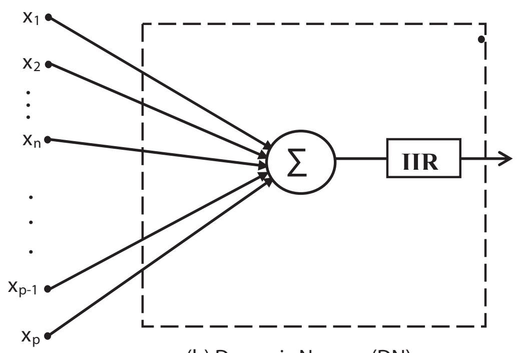
2. Feed forward DNN architecture

The IIR-MLP NN architecture considered in this paper is similar to the multi-layered feed-forward one comprising locally recurrent dynamic neurons. The proposed model exhibits dynamic properties by introducing an infinite impulse response (IIR) filter into the neuron structure. Fig. 1 shows the structure and the topology of the dynamic neuron and multilayered dynamic neural network (DNN). Consequently incorporating an IIR filter between input weights and the output, the neuron can reproduce its own past inputs using two signals: the lagged input pattern input and filter output. For the *j*th dynamic neuron, the weighted sum of the inputs (lagged stock indices and technical indicators) to it is computed at the *k*th instant as

$$s_{j,k} = \sum_{i=1}^{p} w_{ji} x_i \tag{1}$$



(a) A Neural model for IIR Multilayered Perceptron



(b) Dynamic Neuron (DN)

Fig. 1. A neural model for IIR multilayered perceptron.

where p is the number of inputs $(x_1, x_2, x_3, \ldots, x_p)$, and the weight vector is given by

$\mathbf{W}_j = [w_{j1}, w_{j2}, \ldots, w_{jp}]^T$. The output of the filter is obtained from a linear difference equation of the form

$$z_{j,k+1} = \sum_{i=0}^{n} b_{ji} s_{j,k-i} + \sum_{i=1}^{m} a_{ji} z_{j,k-i} \tag{2}$$

where the feedback and feedforward filter weights are given by

$$\mathbf{A} = [a_1, a_2, a_3, \ldots, a_m]^T, \quad \mathbf{B} = [b_1, b_2, \ldots, b_n]^T \tag{3}$$

The final output is obtained from the jth dynamic neuron with tan h activation function as

$$y_{j,k+1} = \tan h(\theta_1 z_{j,k+1} + \sigma_1) \tag{4}$$

where θ_1, and σ_1 are tunable parameters.

In a multilayered formulation the neurons transmit signals from one layer to the next one, till it reaches the output neuron without any feedback loop. The IIR filter, however, provides the dynamic feedback thus giving recurrent nature to the network. Considering a simple DNN architecture comprising an input layer, a hidden layer and an output layer with one neuron, the final output of the IIR-MLP is given by

$$y_{k+1} = \tan h\left(\sum_{j=1}^{q} \theta_j z_{j,k+1} + \sigma\right) \tag{5}$$

where q is the total number of dynamic neurons in the hidden layer, and $\theta_1, \theta_2, \ldots, \theta_q, \sigma$ are tunable parameters.

The parameters to be predicted can be arranged as a state vector given by

$$\mathbf{x} = [\mathbf{W}, \mathbf{A}, \mathbf{B}, \boldsymbol{\theta}, \sigma]^T \tag{6}$$

where the weight matrix $\mathbf{W}$, and the vector θ are obtained as

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \ldots & w_{1n} \\ w_{21} & w_{21} & \ldots & w_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ w_{n1} & w_{n2} & \ldots & w_{nn} \end{bmatrix} \tag{7}$$

$$\boldsymbol{\theta} = \left[\theta_1, \theta_2, \ldots, \theta_q\right]^T$$

Although a simple architecture has been used for this model, more numbers of dynamic neurons can be added to match the nonlinear nature of the stock prices. The next section describes four algorithms that include the RTRL algorithm based on gradient descent, the UKF, and the DE, and a hybrid of DE and UKF (DEUKF) for learning the parameters of the above model in a recursive manner.

3. Learning algorithms

The most common gradient based algorithms used for on-line training of recurrent IIR filter based NNs are back-propagation (BP) algorithms and real-time recurrent learning (RTRL) [29]. In both these algorithms gradient descent based on first order derivatives is used to update the synaptic weights of the network. However, both these approaches, exhibit slow convergence rates because of the small learning rates required, and most often they become trapped to local minima. On the other hand, the UKF has been used [31,32] in the weight update recursions, where better conditioned training is accomplished in comparison to BP and the RTRL method. The following learning algorithms are used in this paper for training the DNN for the prediction of stock market indices.

3.1. Real time recurrent learning algorithm (RTRL)

For the IIR filter based NN the formula for updating the current filter coefficients is more complex due to the dependencies of the recursive IIR formulation on the past filter coefficients. Normally the limits of AR part of the IIR filter are set within −1 and +1 for stable operation [36,37]. For simplicity an Adaline and IIR filter neural system is considered for updating the weights in the following way:

In this approach the mean squared error is minimized to yield the updating formula for the moving average and filter coefficients as

$$a_{i,k} = a_{i,k-1} + \mu e_k \mathrm{sec}^2 h(\theta_1 z_{1,k} + \sigma_1)\frac{\partial z_{1,k}}{\partial a_{i,k}} \tag{8}$$

$$b_{i,k} = b_{i,k-1} + \mu e_k \mathrm{sec}^2 h(\theta_1 z_{1,k} + \sigma_1)\frac{\partial z_{1,k}}{\partial b_{i,k}} \tag{9}$$

$$w_{i,k} = w_{i,k-1} + \mu e_k \mathrm{sec}^2 h(\theta_1 z_{1,k} + \sigma_1)\frac{\partial z_{1,k}}{\partial w_{i,k}} \tag{10}$$

and the error e_k is obtained as

$$e_k = y_k^d - y_k,$$

where y_k^d is the desired stock price on the forecasting day and y_k is the corresponding output from the DNN.

Further the gradients in the above equations are updated recursively as

$$\frac{\partial z_{1,k}}{\partial a_{i,k}} = z_{k-i} + \sum_{j=1}^{m} a_{j,i}\frac{\partial z_{k-j}}{\partial a_{i,k}}, \quad i = 1, 2, \ldots, m \tag{11}$$

$$\frac{\partial z_{1,k}}{\partial b_{i,k}} = s_{k-i} + \sum_{j=1}^{n} b_{j,i} \frac{\partial z_{k-j}}{\partial b_{i,k}}, \quad i = 1, 2, \ldots, n \tag{12}$$

$$\frac{\partial z_{1,k}}{\partial w_{i,k}} = x_{k-i} + \sum_{j=1}^{p} w_{j,i} \frac{\partial z_{k-j}}{\partial w_{i,k}}, \quad i = 1, 2, \ldots, p \tag{13}$$

This procedure does not yield in a closed form expression for the gradient but it does produce a recursive relation by which the gradients can be generated using Eq. (13). In the beginning of the learning cycle all the gradients are made zero or very small. It is well known that the use of this recursion is to make the problem nonlinear in terms of the coefficient parameters. Also, the current filter parameters now depend directly upon previous time-varying filter coefficients, often leading to mean square error surfaces that are not quadratic in nature and thus lead to local minima.

Any EKF-based training algorithm is a second order, recursive procedure that is particularly effective in training both recurrent and feed forward NN architectures for a wide variety of problems. It typically requires fewer training cycles than does its RTRL counterpart and tends to yield superior input–output mapping. Recently, Julier and Uhlmann [12–14] have introduced a new filter founded on the intuition that it is easier to approximate a Gaussian distribution than it is to approximate arbitrary nonlinear functions. They named this filter as unscented Kalman filter (UKF). The UKF leads to more accurate results than EKF and in particular it generates much better estimates of the covariance of the state. Therefore, a more robust algorithm using UKF is presented in the next section.

3.2. Adaptive UKF for IIR filter NN training

The UKF overcomes some of the limitations of the well known extended Kalman filter (EKF) as it does not use linearization avoiding the loss of higher order terms in the Taylor series expansion for the Jacobean, resulting in the improvement of the performance of the predictor. Unlike the EKF, the estimates obtained by UKF are not biased and the computational advantages accrue due to absence of matrix inversion at each iterative step. The UKF belongs to the family of sigma-point filters and an unscented transformation (UT) is used to evaluate the statistics of a random variable undergoing nonlinear transformation. Further the UT relies on the principle that a Gaussian distribution is relatively easy to approximate than a nonlinear function and provides higher order information about the distribution of the system state, even using a small number of sigma points. Instead of linearizing the Jacobean matrices, a deterministic sampling approach is used in the UKF to obtain mean and covariance estimates with a minimal set of $2 \times n+1$, sigma points based on a square-root decomposition of the prior covariances, and n represents the order of the system input state. Propagating these sigma points through the nonlinearity, a weighted mean and covariance is found. Like the EKF, the UKF uses a recursive algorithm that uses the system model, measurements, and known statistics of the noise mixed with the signal. The UKF was originally designed to estimate the states of a dynamic system and for nonlinear control applications.

For training the RNN, the following discrete-time equations are formulated as

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + \mathbf{w}(k) \\ \mathbf{y}_k &= g(\mathbf{x}_k) + \mathbf{v}(k) \end{aligned} \tag{14}$$

and,

$$\mathbf{x}_k = [\mathbf{W}, \mathbf{A}, \mathbf{B}, \boldsymbol{\theta}, \sigma], \quad g(\mathbf{x}_k) = \tan h \left(\sum_{j=1}^{q} \theta_j z_{j,k+1} + \sigma \right) \tag{15}$$

where $\mathbf{w}(k)$ and $\mathbf{v}(k)$ are process noise and measurement noise, and $\boldsymbol{x}(k)$ and $\boldsymbol{y}(k)$ stand for parameter vector and observation vector, respectively. Given a state vector at $\mathbf{x}$ at the step $k-1$, sigma points are computed and stored in the columns of $n \times (2n+1)$ sigma point matrix $\mathbf{X}_{k-1}$, where n = dimension of the state vector. The unscented transform (UT) is used to calculate a set of vectors known as sigma points. Thus for estimating the stock market prediction model with 10 state variables, n becomes equal to 10 and thus $\mathbf{X}_{k-1}$ is a 10×21 matrix. The UKF algorithm is summarized in the following steps:

Step 1: Unscented transformation and sigma points calculation

Given a state vector $\mathbf{x}$ at time step $k-1$, sigma points are computed and stored in the columns of $L \times (2L+1)$ sigma point matrix $\mathbf{X}$. The sigma points are computed as

$$\begin{aligned} \mathbf{X}_{0,k-1} &= \hat{\mathbf{x}}_{k-1}, \quad i = 0 \\ \mathbf{X}_{i,k-1} &= \hat{\mathbf{x}}_{k-1} + \left(\sqrt{(n+\lambda)\mathbf{P}_{k-1}} \right)_i, \quad i = 1, 2, \ldots, n \\ \mathbf{X}_{i+n,k-1} &= \mathbf{x}_{k-1} - \left(\sqrt{(n+\lambda)\mathbf{P}_{k-1}} \right)_i, \quad i = n+1, \ldots, 2n \end{aligned} \tag{16}$$

$\left(\sqrt{(n+\lambda)\mathbf{P}_{k-1}} \right)_i$ is the ith column of the matrix square root [22,23] of $(n+\lambda)\mathbf{P}_{k-1}$.

The parameter λ is used to control the covariance matrix, and is given by

$$\lambda = \alpha^2(n+\kappa) - n \tag{17}$$

The scaling parameters λ and κ determine the spread of the sigma points around $\hat{\mathbf{x}}$. Also the factor κ has the effect of reducing the higher order errors of the mean and covariance approximations. The constant α is usually set to $-4 \le \alpha \le 1$, and $\kappa = 3 - n$, or zero.

The weight vector of the IIR recurrent NN and the UKF state, process and noise covariance matrices are initialized as

$$\begin{aligned} &\hat{\mathbf{x}}(0) = E[\mathbf{x}] \\ &\mathbf{P}(0) = E[(\mathbf{x} - \hat{\mathbf{x}}(0))(\mathbf{x} - \hat{\mathbf{x}}(0))^T = \varepsilon^{-1}\mathbf{I}, \quad \mathbf{R}(0) = r, \quad \mathbf{Q}(0) = q\mathbf{I} \end{aligned} \tag{18}$$

where ε, r, and q are chosen appropriately and $\mathbf{I}$ is a unit matrix of appropriate dimension.

After computing the sigma points the time update of state estimates are given by

$$\hat{\mathbf{x}}_k^- = \sum_{i=0}^{2n} W_i^m \mathbf{X}_{i,k|k-1} \tag{19}$$

where the weights $W_i^{(m)}$ are defined by

$$W_0^{(m)} = \frac{\lambda}{n+\lambda}, W_i^{(m)} = \frac{\lambda}{2(n+\lambda)}, W_{i+L}^{(m)} = \frac{1}{2(n+\lambda)}, \quad i = 1, \ldots., n \tag{20}$$

The a priori error covariance is given by

$$\bar{\mathbf{P}}_k = \sum_{i=0}^{2n} W_i^{(c)} [\mathbf{X}_{i,k|k-1} - \hat{\mathbf{x}}_k^-][\mathbf{X}_{i,k|k-1} - \hat{\mathbf{x}}_k^-]^T + \mathbf{Q}_k \tag{21}$$

$$W_0^{(c)} = \frac{\lambda}{(n+\lambda)} + (1 - \alpha^2 + \beta), \quad W_i^{(c)} = \frac{1}{2(n+\lambda)} + (1 - \alpha^2 + \beta), \tag{22}$$

$$W_{i+L}^{(c)} = \frac{1}{2(n+\lambda)}, \quad i = 1, \ldots., n \tag{23}$$

The sigma points are propagated through the nonlinear stock market output prediction model to estimate the mean and covariance of **y**. The estimated output becomes equal to

$$\mathbf{Y}_{i,k|k-1} = g_k(\mathbf{X}_{i,k|k-1}) \tag{24}$$

$$\hat{\mathbf{y}}_k^- = \sum_{i-0}^{2L} W_i^{(m)} \mathbf{Y}_{i,k|k-1} \tag{25}$$

Step 2: Measurement update

The unscented transformation yields the covariance matrix of the vector **Y** as

$$S_k = \sum_{i=0}^{2L} W_i^{(c)} [\mathbf{Y}_{i,k|k-1} - \hat{\mathbf{y}}_k^-][\mathbf{Y}_{i,k|k-1} - \hat{\mathbf{y}}_k^-]^T + R_k \tag{26}$$

Further the cross covariance matrix between **X** and **Y** is obtained as

$$\mathbf{U}_k = \sum_{i=0}^{2L} W_i^{(c)} [\mathbf{X}_{i,k|k-1} - \hat{\mathbf{x}}_k^-][\mathbf{Y}_{i,k|k-1} - \hat{\mathbf{y}}_k^-]^T \tag{27}$$

Then the Kalman gain $\mathbf{K}_k$, the state estimate $\hat{\mathbf{x}}_k$, and the error covariance $\mathbf{P}_k$ are obtained as

$$\mathbf{K}_k = \mathbf{U}_k \mathbf{S}_k^{-1} \tag{28}$$

The observation error is obtained as

$$\mathbf{e}_k = (\mathbf{y}_k - \hat{\mathbf{y}}_k^-) \tag{29}$$

and the updated parameter vector is obtained as

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k \mathbf{e}_k \tag{30}$$

To improve the tracking performance of the filter, the error covariance matrix $\mathbf{P}_k$ is obtained as

$$\mathbf{P}_k = [\bar{\mathbf{P}}_k]^{-1} - \mathbf{K}_k \mathbf{S}_k \mathbf{K}_k^T \quad \text{for} \quad \mathbf{S}_k > \varepsilon \bar{\mathbf{S}}_k \tag{31}$$

$$\text{and} \quad \mathbf{P}_k = [(\bar{\mathbf{P}}_k)^{-1} - \gamma^{-2}\mathbf{I}]^{-1} - \mathbf{K}_k \mathbf{S}_k \mathbf{K}_k^T, \quad \text{otherwise.} \tag{32}$$

$$\bar{\mathbf{S}}_k = \begin{cases} \mathbf{e}_k \mathbf{e}_k^T, & k = 0 \\ \dfrac{\xi \bar{\mathbf{S}}_{k,k-1} + \mathbf{e}_k \mathbf{e}_k^T}{\xi + 1}, & k > 0 \end{cases} \tag{33}$$

and ξ is a forgetting factor and is chosen as 0.98.where $\bar{\mathbf{S}} = E[\mathbf{e}_k \mathbf{e}_k^T]$ is the real covariance matrix of the innovation (measurement error). The parameter $\varepsilon > 0 (\varepsilon = 0.9)$ is used to tune the threshold while the filter is implemented. Further another tuning parameter γ is introduced to minimize the estimation of errors by the filter due to sudden fluctuations of the data. It is seen that the robustness of the filter increases by reducing γ, but the mean square error also increases. Thus a suitable value of γ needs to be selected for making the filter robust and reduce the mean square error simultaneously. The design objective for the robustness is to guarantee the norm of the transfer function between the external disturbances (modeling errors and system noises) and the estimation error to be less than a prescribed attenuation level γ.

$$\frac{||\mathbf{I} \cdot (\hat{\mathbf{x}}_k - \hat{\mathbf{x}}_k^-)||^2}{||\mathbf{S}_k||^2 + ||\bar{\mathbf{P}}_k||^2} < \gamma \tag{34}$$

Normally a value of $\gamma = 1.98$ is used to ensure good tracking performance.

Step 3: Adaptation procedure

The performance of the UKF algorithm depends on a proper choice of the parameters α, β, and the initial values of the covariances **Q** and **R**. The covariance matrices **Q** and **R** are the elements that determine the performance and stability of the unscented filter and hence in this paper these two parameters are adaptively tuned. This is required as the uncertainties in the system model and fluctuations in the measured values will affect the UKF gain matrix. For this purpose a time-averaged innovation covariance is computed as

$$\mathbf{G}_k = \frac{1}{N_1} \sum_{i=k-N+1}^{k} \boldsymbol{\upsilon}_k \boldsymbol{\upsilon}_k^T \tag{35}$$

where N_1 is the estimation window size and $\upsilon_k = (y_k - \hat{y}_k^-)$, and the filter covariance is obtained as

$$\hat{\mathbf{G}}_k = \sum_{i=0}^{2L} W_i^{(c)} [\mathbf{Y}_{i,k|k-1} - \hat{\mathbf{y}}_k^-][\mathbf{Y}_{i,k|k-1} - \hat{\mathbf{y}}_k^-]^T + \mathbf{R}_k \tag{36}$$

A cost function to minimize $\hat{\mathbf{G}}_k$ is formulated as

$$V_k = tr[(\mathbf{G}_k - \hat{\mathbf{G}}_k)^2] \tag{37}$$

The cost function can be minimized by choosing the value of q of the **Q** matrix as

$$q_k = q_{k-1} - \mu \left[\frac{(V_k - V_{k-1})}{(q_{k-1} - q_{k-2)})} \right] \tag{38}$$

In a similar way the measurement covariance matrix is updated as

$$\mathbf{R}_k = \rho \mathbf{R}_{k-1} + \mathbf{K}_k \left(y_k - \sum_{i-0}^{2L} W_i^{(m)} \mathbf{Y}_{i,k|k-1} \right)^2 \tag{39}$$

ρ is a forgetting factor and is initially taken as 0.9.

The forgetting factor can be further tuned as

$$\rho_k = \rho_{\min} + (\rho_{\max} - \rho_{\min}) \exp(-|(r_k/r_0)|) \tag{40}$$

where the estimation error covariance is obtained iteratively as,

$$r_k = \rho_{k-1} r_{k-1} + (y_k - \hat{y}_k^-)^2 \tag{41}$$

and the value of r_0 is obtained by summing the measurement error over a few samples (10 samples for example), $\rho_{\min}$, $\rho_{\max}$ are suitably chosen parameters having values $\rho_{\min} = 0.85$, $\rho_{\max} = 0.99$.

Further to improve the estimation performance of the adaptive UKF, a stochastic optimization technique like the DE is used to obtain the initial values of α, β, **Q**, and **R**, instead of trial and error approach.

The optimization is aimed at minimizing the MSE or maximizing the fitness value. Normally a sample size of 10 is used in this paper to obtain the suitable parameters α, β, q, r for starting the UKF program. Further **Q** and **R** matrices are tuned recursively after the filter weights are initialized. This new formulation is known as DEUKF method. Another alternative learning procedure is the use of DE for the entire training data and the IIR NN weights. The next section describes the DE algorithm.

3.3. DE for learning DNN parameters

The DE algorithm is a simple yet powerful population-based stochastic search technique that provides effective global optimization in the continuous search domain. Many successful applications of DE include diverse areas like power systems, pattern recognition, communications, etc. Although there exists several trial vectors generation paradigms in DE, a few may be appropriate in yielding the optimal solutions for a particular problem [38,39]. Also the optimization performance of DE is governed by three crucial control parameters like the population size, scaling factor, and crossover rate. Further the DE algorithm aims at evolving a population of *Np* *D* – dimensional parameter vectors, known as individuals, which encode the candidate solutions toward the global optimum. The pseudo code for DE implementation is given below:

3.4. *Pseudo code for DEUKF implementation*

Input: population size Np, no. of variables to be optimized (Dimension D), initial scaling and mutation parameters F_1, F_2, and Cr, upper and lower limits of variables, total number of generations G. Strategy candidate pool: "DE/rand/2/bin".

(1) Generation:

The parameters of the ith vector for the generation g are given by Eq. (42).

$$\mathbf{XX}^g_{(i)} = \left\{xx^g_{(i,1)}, xx^g_{(i,2)}, \ldots, xx^g_{(i,j)}, \ldots, xx^g_{(i,D)}\right\} \tag{42}$$

A random initialization is done for the jth parameter of the ith vector at the 1st generation as

$$xx^1_{(i,j)} = xx^{\min}_j + rand_{(i,j)} \times \left(xx^{\max}_j - xx^{\min}_j\right) \tag{43}$$

While stopping criterion is not satisfied

do

(2) Mutation

for $i = 1$ to Np

$$\begin{aligned} F_1 &= 0.5 + 0.45 \quad rand(0, 1) \\ F_2 &= 0.3 + 0.5 \quad rand(0, 1) \end{aligned} \tag{44}$$

Generate the mutant vector from the target vector using Eq. (42)

$$\mathbf{V}^g_{(i)} = \left\{v^g_{(i,1)}, v^g_{(i,2)}, \ldots, v^g_{(i,j)}, \ldots, v^g_{(i,D)}\right\}$$

$$\mathbf{V}^g_{(i)} = \mathbf{XX}^g_{(\mu)} + F_1 \times \left(\mathbf{XX}^g_{(\delta)} - \mathbf{XX}^g_{(\gamma)}\right) + F_2 \times \left(\mathbf{XX}^g_{(\zeta)} - \mathbf{XX}^g_{(\eta)}\right) \tag{45}$$

The indices, $\mu, \delta, \gamma, \xi, \eta \in [1, Np]$

end for

(3) Crossover

for $i = 1$ to Np

$$j_{rand} = rndinit(1, D)$$

for $j = 1$ to D

To increase the diversity of population crossover is used for each target vector, and the crossover rate is adapted as

$$Cr = \begin{cases} rand3, & if \quad rand\,4 \le 0.1 \\ Cr, & \text{otherwise} \end{cases} \tag{46}$$

where *rand* 3 and *rand* 4 are random numbers between [0,1].

Generate a trial vector $u^g_{(i,j)}$ using Eq. (47)

$$\mathbf{U}^g_{(i)} = \left\{u^g_{(i,1)}, u^g_{(i,2)}, \ldots, u^g_{(i,j)}, \ldots, u^g_{(i,D)}\right\}$$

$$u^g_{(i,j)} = \left\{ \begin{matrix} v^g_{(i,j)} & \text{if}(rand_{(i,j)} \le Cr) \quad \text{or} \quad j = j_{rand} \\ x^g_{(i,j)} & \text{otherwise} \end{matrix} \right\} \tag{47}$$

end for

end for

(4) Selection:

for $i = 1$ to Np

Evaluate the objective function of the trial vector

$$\mathbf{U}^g_{(i)} = \left\{u^g_{(i,1)}, u^g_{(i,2)}, \ldots, u^G_{(i,j)}, \ldots, u^g_{(i,D)}\right\}$$

The objective function is obtained from the UKF algorithm as $f(\mathbf{U}^g_{(i)}) = (1/MM)\sum_{k=1}^{MM} e^2_k$, MM = iteration number used for evaluation from Eq. (29).

and the fitness of the ith vector

$$F_i = \frac{1}{1 + f(\mathbf{U}^g_{(i)})} \tag{48}$$

$$\mathbf{XX}^{g+1}_{(i)} = \left\{ \begin{matrix} \mathbf{U}^g_{(i)} & \text{if} \;\; f(\mathbf{U}^g_{(i)}) \le f(\mathbf{XX}^g_{(i)}) \\ \mathbf{XX}^g_{(i)} & \text{if} \;\; f(\mathbf{U}^g_{(i)}) > f(\mathbf{XX}^g_{(i)}) \end{matrix} \right\} \tag{49}$$

end for

g = g + 1

(5) **end** While

4. Performance evaluation

Although the MSE measure may be used to drive the prediction model in the training process, it cannot be considered alone as a conclusive measure for comparison of different prediction models [27]. For this reason, other performance criteria should be considered for allowing more robust performance assertiveness. In order to measure the performance of the proposed model, the Root mean square error (RMSE), and mean absolute percentage error (MAPE) have been used as

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{50}$$

$$\text{MAPE} = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100 \tag{51}$$

In order to avoid the adverse effect of very small stock indices over a long period of time, another MAPE known as AMAPE is adopted and compared for all the learning models:

$$\text{AMAPE} = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{(1/N)\sum_{i=1}^{N} y_i}\right| \times 100 \tag{52}$$

To measure the impact of the learning model uncertainty using the DNN on forecasting accuracy, the variance of the forecast errors over a certain span of time is needed and this parameter is obtained as

$$\sigma^2_E = \frac{1}{ND}\sum_{i=1}^{ND}\left\{\frac{|y_i - \hat{y}_i|}{(1/ND)\sum_{i=1}^{N} y_i} - \sum_{i=1}^{ND}\frac{|y_i - \hat{y}_i|}{(1/ND)\sum_{i=1}^{N} y_i}\right\}^2 \tag{53}$$

where ND is the no. of days over which the variance is evaluated.

The smaller is the variance, the less uncertain is the learning model or more accurate is the forecast results.

5. Stock market trend prediction

Technical analysis and computation of technical indicators are important to predict the future behavior of stock markets. Important technical indicators are used in this paper for extracting relevant information from the stock time series to predict the stock market trend [34,35]. From these technical indicators, a rule based system is developed for predicting one-day-ahead trends in the stock market. Classification of the trend into up, down and no trend was done according to the definition in [15] in the following way: The market is formally classified as being in an uptrend (downtrend) when all the following conditions are satisfied:

5.1. MA indicator

1. If $C_t > MA_{25}$, and $MA_{25} > MA_{65}$, and $\Delta MA_{25} < 0$, and $\Delta MA_{65} > 0$ Then Uptrend (54)

2. If $C_t < MA_{25}$, and $MA_{25} < MA_{65}$ and, $\Delta MA_{25} < 0$, and $\Delta MA_{65} < 0$ Then Downtrend (55)

where C_t is the closing price of the tth trading day; MA_{25} = 25 day moving average; MA_{65} = 65 day moving average, Δ represents the change.

5.2. Stochastic line, KD

After determining the turning points trading decisions [33] can be taken with the help other technical parameters like the stochastic line KD and WMS%R indicator (or simply the R-index). The KD indicator comprises two smooth moving average lines, K line and D line. Both these indicators range between 0 and 100; and this indictor is relied by the investors in buying or selling the stocks. When the KD indicators are ≥80, and the D indicator crosses the K indicator side with a negative slope, the stock may be sold. On the other hand, when the KD indicators are ≤20, and the D indicator crosses the K indicator with a positive slope, the stock may be bought. The accuracy of forecasting result will decrease when KD remains less than 20 or greater than 80 for all times. Based on this, the William indicator should be adopted to assist the turning point decision.

5.3. WMS%R indicator (R-index)

The WMS%R of the tth trading day is obtained as

$$\text{WMS\%Rt} = \frac{H - C_t}{H - L} \times 100\% \tag{56}$$

where H, L are the highest and lowest stock prices during the latest t trading days, and C_t is the closing price of the tth trading day. A simple trading rule base is given below:

Rule-1: If WMS%Rt > threshold, then the stock price is undervalued.
Rule-2: If WMS%Rt < threshold, then the stock price is overvalued.

The WMS%Rt or simply the R-index will be adopted to assist the KD intersection for judging the points of buying/selling.

6. Experimental results

The model proposed in this study chooses several variables as inputs to the DNN for prediction of future stock price; one day ahead of time. The experimental data is obtained from several stock markets such as BSE, IBM stock market, Reliance stock market (RIL), and Oracle stock market, etc. The daily closing prices of past six days of the dataset, i.e., price on day $(d-1)$, $(d-2)$, $(d-3)$, $(d-4)$, $(d-5)$, $(d-6)$, along with two technical indicators like the moving average and integrated moving average have been taken as inputs to the model, to find out the price on day (d). Each dataset contains 905 closing price indices, starting from 3rd Jan 2005 to 13th August, 2008. Each dataset is divided into two phases: training and testing. Out of the 905 data samples, first 500 data samples are used for training and the rest 400 for testing. Further the system is tested in different time horizons and the errors are measured by different error accuracy measurement formulas. All the values of the datasets are normalized between 0 to 1 using the standard formula given in Eq. (57), and the normalized values along with technical indicators are fed into DNN as inputs for future stock price prediction. The scaling is done as

$$\tilde{X}^i = \frac{X^i - X_{\min}}{X_{\max} - X_{\min}} \tag{57}$$

where $\tilde{X}^i$ is the scaled price, X^i is the current day Closing price, $X_{\max}$ and $X_{\min}$ are the maximum and minimum prices of the dataset, respectively. The total number of variables to be estimated for the DNN comprising 8 inputs (six past stock closing prices, and two technical indicators clubbed together) and two coefficients of the IIR block become equal to 10. The enhanced DE performs best with the last strategy in DE. The number of population for UKF parameter optimization Np = 10; dimension D = 5; number of generations G = 50, iterations for UKF loop MM = 5. Following the standard DE literature, the scaling factors F_1, and F_2, and the mutation factor Cr are chosen as 0.5, 0.3, and 0.65, respectively at the start of the DEUKF program execution. Subsequently F_1 and F_2 are varied at each generation according to Eq. (44). The normal distribution function (represented by ϕ_1^g and ϕ_2^g) generates positive and negative values in continuous domain with mean as zero. For DE learning procedure the Np = 30; D = 12 for one dynamic neuron; G = 500. A flow chart for the implementation of the hybrid learning and prediction scheme is shown in Fig. 2(a and b), respectively. In the DEUKF algorithm the total number of data samples used is 250 for optimizing the UKF parameters, which are found as $\alpha = 0.453$, $\beta = 2$, $q = .0011$, $r = 0.186$, $\theta_1 = 1.05$, $\sigma = 0.1$. The initial covariance matrix P is chosen equal to 1000I I is an unit matrix of appropriate dimension. Thereafter, the UKF algorithm does the prediction on its own using the optimized parameters and adapting **Q** and **R** iteratively as outlined in Section 3.2.

The prediction results for BSE, RIL, and IBM stock indices using the proposed dynamic neural model trained with DE and DEUKF learning approaches are shown in Fig. 3(a–f), respectively. From these figures, it is quite evident that the DNN when trained with DEUKF shows excellent tracking performance for all the stock market indices in comparison to the DE based learning approach. Further the stock market performance results shown in Tables 1–4 exhibit a similar trend, where the DNN trained with optimized UKF shows significant reduction in MAPE, AMAPE, and RMSE. Since the UKF does not have any matrix inversion and the number of variables to be optimized is only 5 of the UKF parameters and the number of generations is limited to 50 (with the number of iterations for UKF is limited to 10 for the objective function), the time of execution is not significantly increased. The accuracy of prediction can be further improved by performing more iteration for the UKF between generations and using multilayered DNN instead of the single layer architecture. Also these results reveal that the gradient based RTRL algorithm produces the worst prediction performance with a MAPE of nearly 4.5% for a data window of 400 samples. However, when the data window is reduced to be 200, the MAPE reduces to approximately 3.9%. Other performance indices show similar trends. Although the DEUKF performs better than the DE alone in predicting the stock indices, its performance can be further increased by increasing the number of iterations from 5 to 10 or more used in the UKF loop at the expense of more execution time. Since in stock market prediction is not a real-time problem, the prediction accuracy is more important than the execution timings.

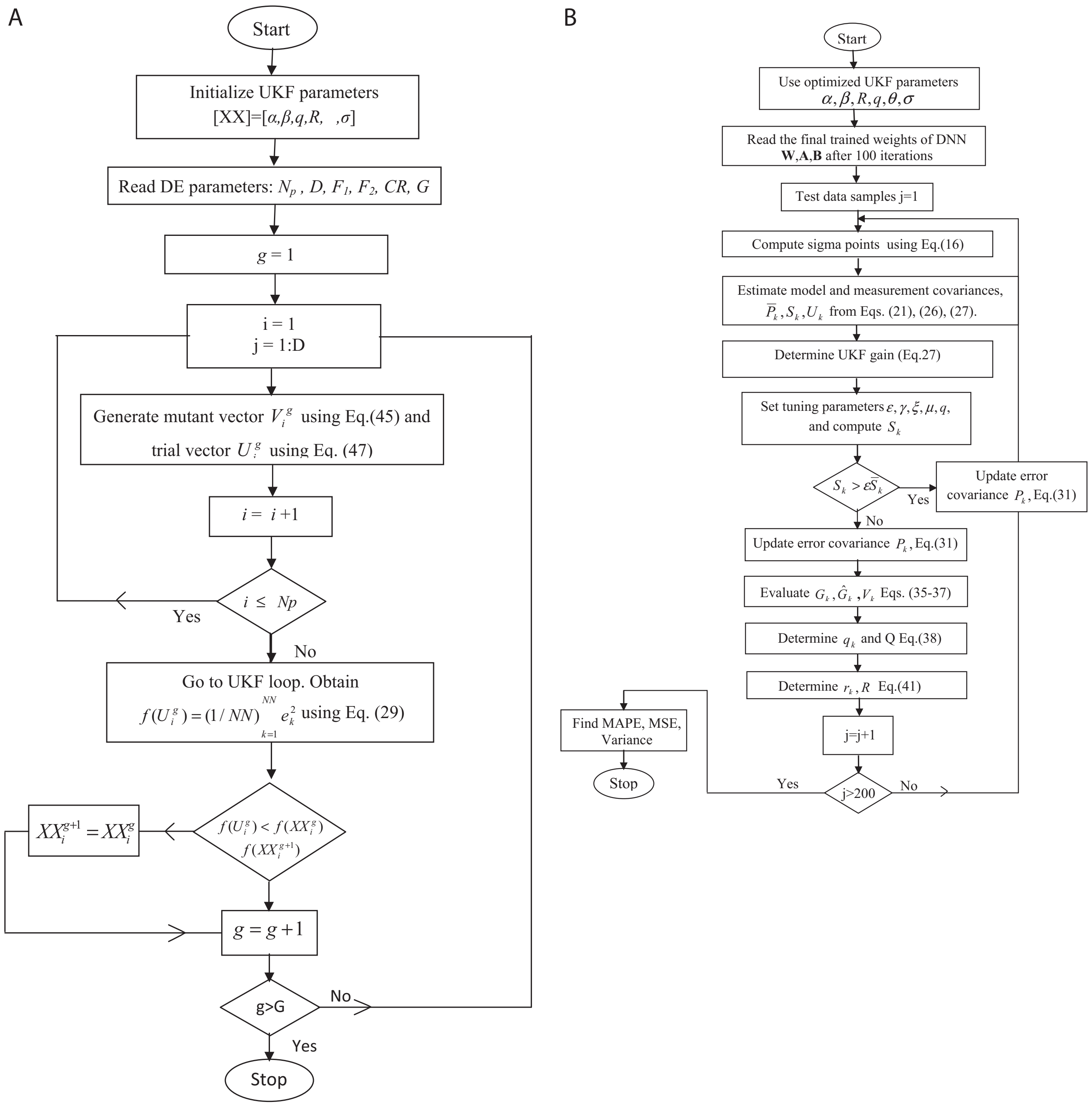
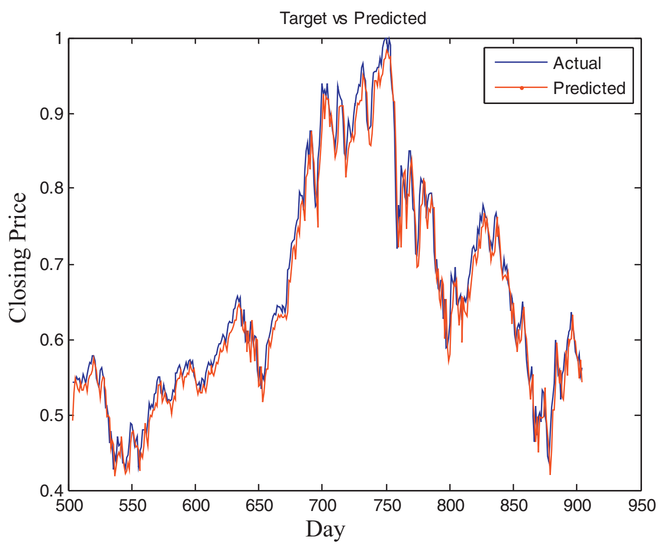

A
Start
Initialize UKF parameters [XX]=[α,β,q,R, ,σ]
Read DE parameters: N_p, D, F_1, F_2, CR, G
g = 1
i = 1
j = 1:D
Generate mutant vector V_i^g using Eq.(45) and trial vector U_i^g using Eq. (47)
i = i +1
i ≤ Np
Yes
No
Go to UKF loop. Obtain $f(U_i^g) = (1/NN)\sum_{k=1}^{NN} e_k^2$ using Eq. (29)
$f(U_i^g) < f(XX_i^g)$
$f(XX_i^{g+1})$
$XX_i^{g+1} = XX_i^g$
g = g +1
g>G
No
Yes
Stop
B
Start
Use optimized UKF parameters α, β, R, q, θ, σ
Read the final trained weights of DNN W,A,B after 100 iterations
Test data samples j=1
Compute sigma points using Eq.(16)
Estimate model and measurement covariances, $\bar{P}_k, S_k, U_k$ from Eqs. (21), (26), (27).
Determine UKF gain (Eq.27)
Set tuning parameters ε, γ, ξ, μ, q, and compute S_k
$S_k > \varepsilon \bar{S}_k$
Yes
Update error covariance P_k, Eq.(31)
No
Update error covariance P_k, Eq.(31)
Evaluate $G_k, \hat{G}_k, V_k$ Eqs. (35-37)
Determine q_k and Q Eq.(38)
Determine r_k, R Eq.(41)
j=j+1
j>200
Yes
No
Find MAPE, MSE, Variance
Stop

Fig. 2. Flow chart for training and testing of the proposed DEUKF model.

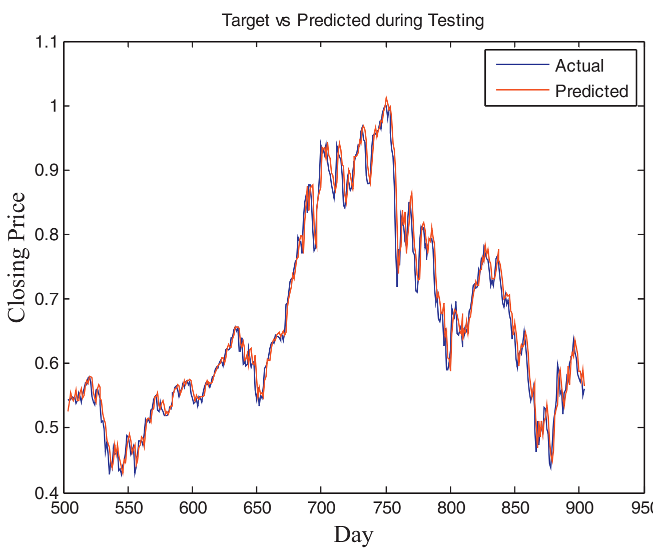
Table 1
Comparison amongst DE, UKF, and DE-UKF using BSE stock data.

Stock	MAPE, MSE, RMSE-measured for 400 and 200 data sets							
	UKF		DE		DE-UKF		RTRL	
BSE	(400 data) (504–903)	(200 data) (504–703)	(400 data) (504–903)	(200 data) (504–703)	(400 data) (504–903)	(200 data) (504–703)	(400 data) (504–903)	(200 data) (504–703)
MAPE	3.60	3.42	2.97	2.55	2.55	2.0450	4.36	4.13
AMAPE	3.52	3.32	2.92	2.54	2.51	2.0424	4.38	4.08
RMSE	0.0387	0.0076	0.0315	0.020	0.0243	0.0143	0.092	0.013

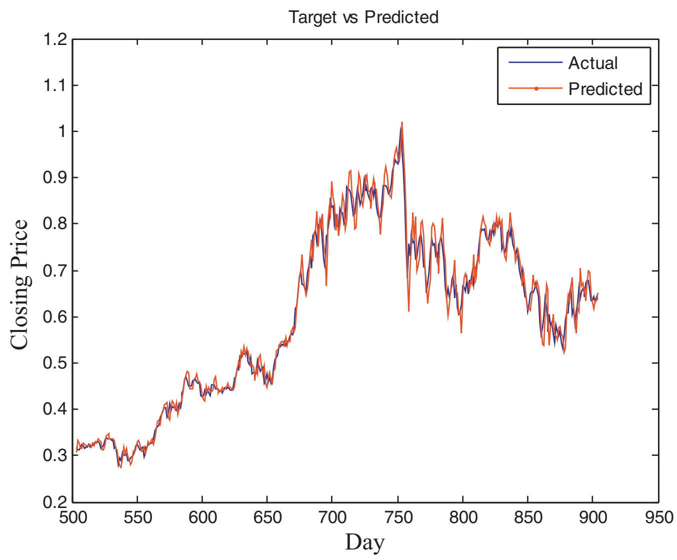
Since DEUKF combines the estimation accuracy and robustness of UKF and the parametric optimization capability of DE, it outperforms all other learning strategies in predicting the chaotic variations of the future stock closing prices. However, there is 10–20% reduction in MAPE for DEUKF in comparison to the DE alone for different stock market indices. In DE alone, a difficulty arises in fixing the upper and lower limits of all the parameters and this affects the accuracy of prediction. Besides as the stock time series



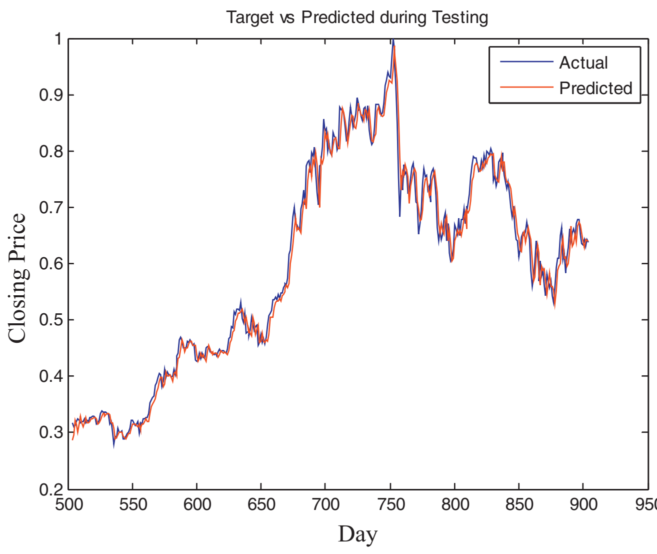
(a) BSE Stock Prices (DE)



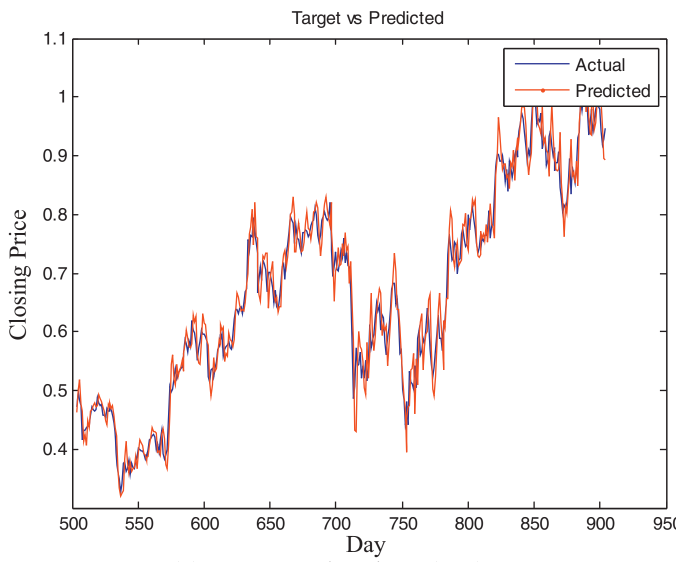
(b) BSE stock Prices (DEUKF)



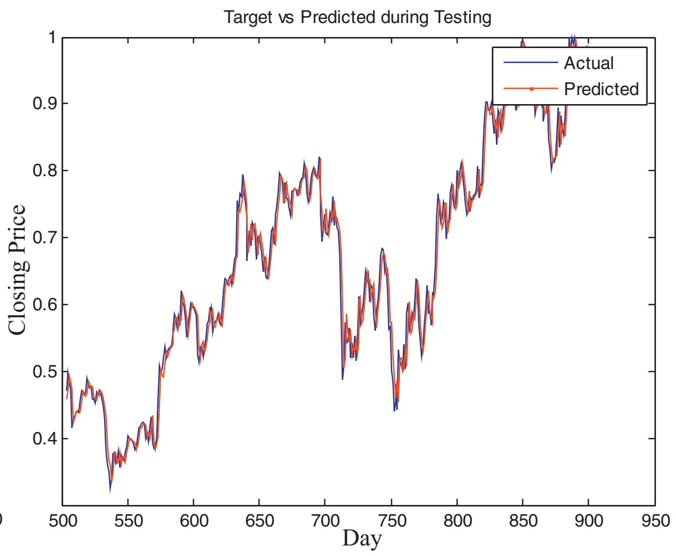
(c) RIL Stock Prices (DE)



(d) RIL Stock Prices (DEUKF)



(e) IBM Stock Prices (DE)



(f) IBM Stock Prices (DEUKF)

Fig. 3. Prediction results of BSE, RIL and IBM using DE and DEUKF.

Table 2
Comparison amongst DE, UKF, and DE-UKF using Oracle stock data.

	MAPE, MSE, RMSE-measured for 400 data and 200 data sets							
	UKF		DE		DEUKF		RTRL	
Oracle	(400 data) (504–903)	(200 data) (504–703)	(400 data) (504–903)	(200 data) (504–703)	(400 data) (504–903)	(200 data) (504–703)	(400 data) (504–903)	(200 data) (504–703)
MAPE	4.58	3.80	3.79	3.41	3.62	3.192	5.32	4.39
AMAPE	4.62	3.82	3.78	3.36	3.70	3.248	5.362	4.421
RMSE	0.0624	0.04898	0.0346	0.0360	0.0300	0.0297	0.0905	0.0818

Table 3
Comparison amongst DE, UKF, and DEUKF using RIL stock data.

Stock	MAPE, MSE, RMSE-measured for 400 data and 200 data							
	UKF		DE		DEUKF		RTRL	
RIL	(400 data) (504–903)	(200 data) (504–703)	(400 data) (504–903)	(200 data) (504–703)	(400 data) (504–903)	(200 data) (504–703)	(400 data) (504–903)	(200 data) (504–703)
MAPE	4.68	3.54	3.21	2.66	2.70	2.205	5.45	4.36
AMAPE	4.74	3.69	3.33	2.75	2.73	2.43	5.48	4.41
RMSE	0.0529	0.0213	0.0278	0.0191	0.0226	0.0138	0.0700	0.0616

Table 4
Comparison amongst DE, UKF, and DE-UKF using IBM stock data.

Stock	MAPE, MSE, RMSE-measured for 400 data and 200 data							
	UKF		DE		DEUKF		RTRL	
IBM	(400 data) (504–903)	(200 data) (504–703)	(400 data) (504–903)	(200 data) (504–703)	(400 data) (504–903)	(200 data) (504–703)	(400 data) (504–903)	(200 data) (504–703)
MAPE	4.62	4.16	3.96	3.55	3.13	2.758	5.78	5.36
AMAPE	4.34	4.09	3.72	3.52	2.97	2.648	5.653	5.302
RMSE	0.0458	0.0412	0.0331	0.0261	0.0263	0.0205	0.0787	0.0700

Table 5
Error variance of BSE and RIL stock data using different ANN models using DE.

Time period	Error Variance (DEUKF) (BSE)	Error Variance (DE) (BSE)	Error Variance (DEUKF) (RIL)	Error Variance (DE) (RIL)
30 Days	0.00028614	0.0004182	0.00010225	0.000402
January, 08	0.0015	0.0024	0.0021	0.0034
March, 08	0.00084366	0.0013	0.00042020	0.0056
May, 08	0.00029185	0.000396	0.00038887	0.00062
July, 08	0.00052033	0.000807	0.00047518	0.0071

is highly chaotic in nature, it is always difficult to predict the accuracy of the forecast for a given method unless all the methods are tried. On the other hand, once the objective function is optimized and the initial UKF parameters are set, the UKF can track the time varying stock indices more accurately than any other method, its convergence is rapid in comparison to the DE and it gives a lesser MAPE, AMAPE, and RMSE. A t-statistic was conducted for the MAPE and RMSE for the DE and DEUKF learning methods for the stock indices and the t and p values are 2.688, 0.0181, and 2.156, and 0.0372, respectively. The corresponding values between DEUKF and UKF are 9.88, 3.08$e-$05, and 4.81, and .0015, respectively. Since the value of p is less than .05 (5% threshold) the performance of DEUKF is superior to those of DE and UKF used for training separately. These values clearly indicate that the DEUKF produces better prediction accuracy in comparison to the DE alone. The t-test for RTRL also shows a much inferior performance in comparison to DEUKF algorithm. A similar conclusion is obtained by using the Friedman nonparametric rank-sum test for both the DE and DEUKF algorithms. Further Table 5 shows the error variance for a period of 30 days computed on different dates for both DE and DEUKF learning models, which clearly reveal the prediction accuracy of DEUKF over DE.

It is well known that the DE is a stochastic algorithm with initial starting parameters chosen arbitrarily and hence a few runs may not be sufficient to optimize the parameters and thus 25, 50, and 100 generations were undertaken for two stocks namely the BSE and RIL. The UKF is run for 10, 5, 2 iterations in each generation. The number of inputs for the DNN is kept at 8, and the total number of samples N = 275,300,300 for the three above generations, respectively. The performance parameters are listed in Table 6 for the two considered stock indices. From the Table it is observed that starting with chosen parameter values nearly 50 generations for DE and the

Table 6
Effect of UKF training.

Stock	MAPE-testing for 200 data samples	
	Window size	1 Day ahead
BSE	25 (DE) + 250 (UKF)	2.15
	50 (DE) + 250 (UKF)	1.92
	100 (DE) + 250 (UKF)	2.46
RIL	25 (DE) + 250 (UKF)	2.83
	50 (DE) + 250 (UKF)	2.29
	100 (DE) + 250 (UKF)	3.15

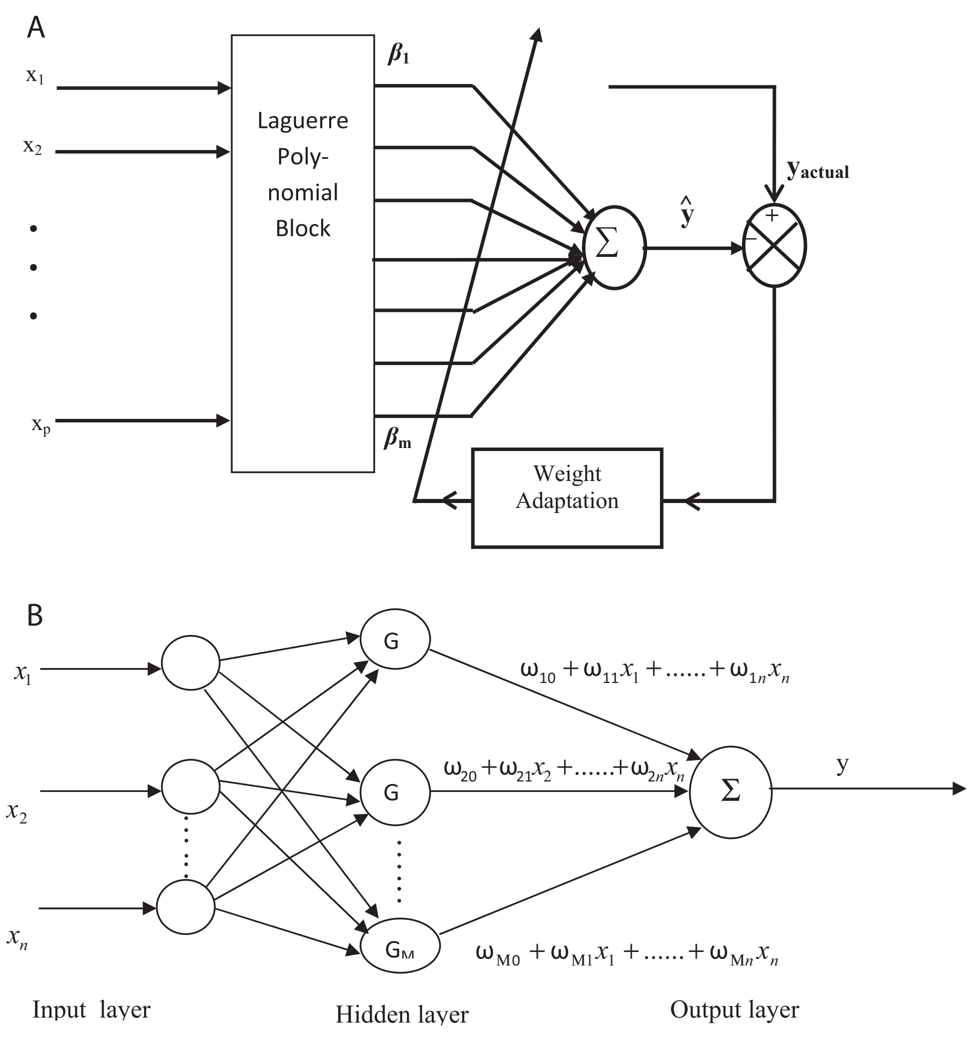

A
x1
x2
xp
Laguerre Poly-nomial Block
β1
βm
Σ
ŷ
yactual
+
−
Weight Adaptation
B
x1
x2
xn
G
G
GM
ω10 + ω11x1 + ...+ ω1nxn
ω20 + ω21x2 + ...+ ω2nxn
ωM0 + ωM1x1 + ...+ ωMnxn
Σ
y
Input layer
Hidden layer
Output layer

Fig. 4. Neural network architectures for comparison.

rest out of a total 500 data points for the UKF, the least MAPE value is obtained. However, the nature of the data will determine the optimal mix of the DE and UKF iterations and this may require a few trials. A compromise may, however, be required for excessive run times of the DE algorithm. Another important parameter is the size of the input window and three window sizes have been used for illustration as shown in Table 7. The optimum DE and UKF combination is used to provide a comparison of MAPE values for both the BSE and RIL stock indices. From Table 7 it is observed that a lower window size produces better one day ahead forecast than bigger window size. However, a bigger window is beneficial for increasing the prediction horizon from one day ahead to seven days ahead as shown in the Table.

Table 7
Effect of window size.

Stock	MAPE-testing for 200 data samples		
	Window size	1 Day ahead	7 Days ahead
BSE	8	1.92	5.03
	10	2.34	4.87
	12	2.83	4.66
RIL	8	2.2	5.84
	10	2.74	5.32
	12	2.91	5.14

In order to compare the performance of the DNN and DEUKF model, other modified NN architectures like the local linear wavelet neural network (LLWNN), the local linear RBFNN (LLRBFNN), and the Laguerre FLANN shown in Fig. 4 are considered. Also it can be noted here that the number of weights in the DNN model is 10, whereas the FLANN model has 56 and both LLWNN and LLRBFNN have both 72 weights. The advantage of using local linear architecture is used here to avoid an arbitrary choice of neurons in the hidden layer for multilayered NNs. Fig. 5 shows the actual and predicted stock price indices for the above NN models. Besides these three NN models, the performance of other NNs like the MLP, ANFIS, and RBF networks are evaluated for the BSE price indices using a time frame varying from 3-days ahead to 15 days ahead and the MAPE values are shown in Table 8. All the neural models except the ANFIS comprise an input layer, a hidden layer, and an output layer. On the other hand, the ANFIS has 5 layers including a rule layer. For the MLP, and RBF, the architecture is 8,13,1 and 8, 6, 1, respectively. The Gaussian membership functions are used for RBFNN, and ANFIS, with center and spread updated along with the weights. The weights of these networks are updated with DE algorithms, the number of variables is 117 for MLP, 18 (which includes the weights center and breadth of each of the radial basis functions) for the RBF and 50 for the ANFIS with 5 rules and 5 fuzzy set for each variable.

To illustrate the superiority of DEUKF over DE learning algorithms, 3-days ahead and 7-days ahead prediction results of the

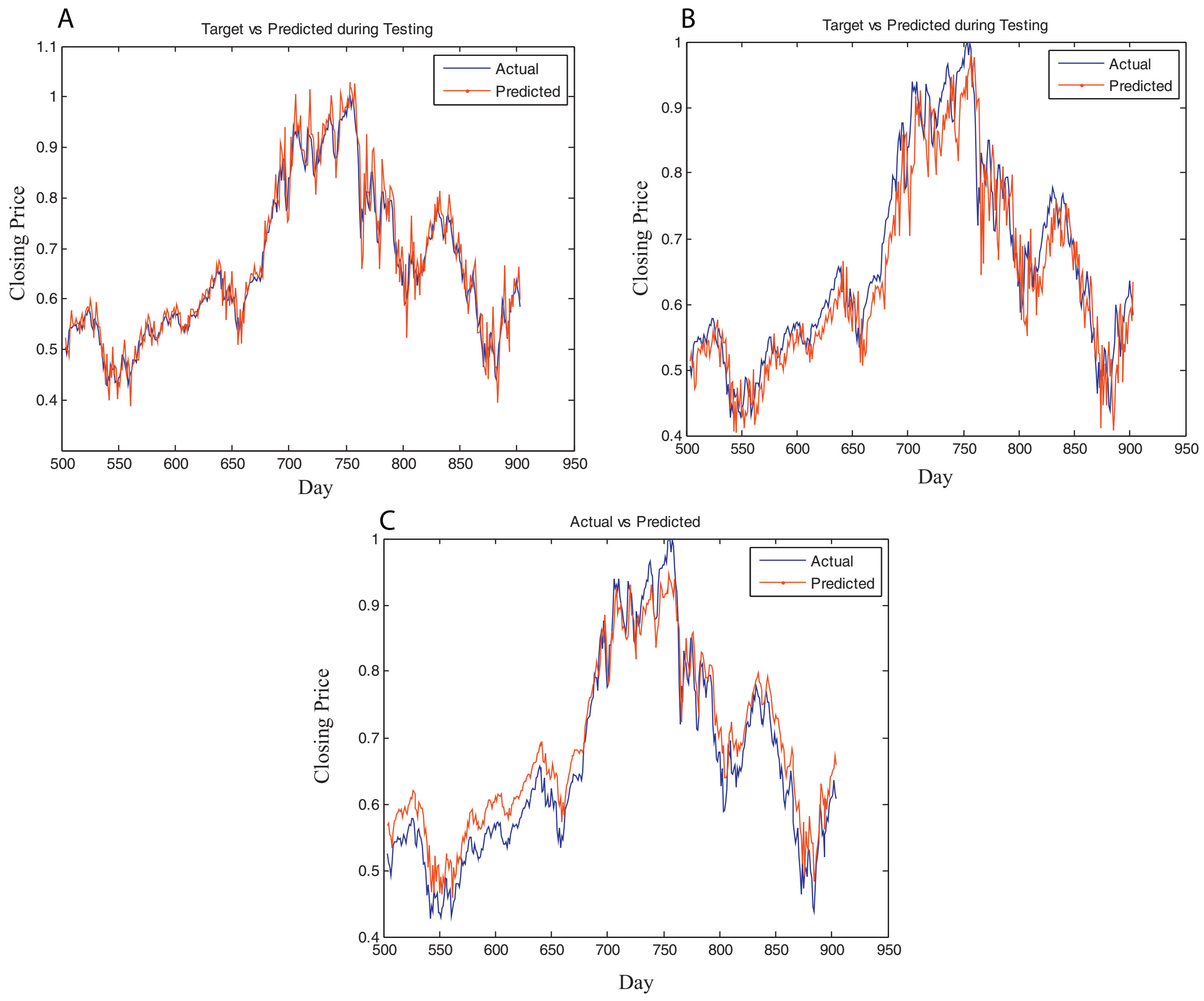


Fig. 5. Comparison results of BSE data using LLWNN, LLRBFNN, and FLANN (Laguerre).

BSE stock index are shown in Fig. 6. Also from the results shown in Table 8 it is clear that the simple DNN performs the best in comparison to other NN models using the DEUKF learning algorithm. Most of the NN models used for comparison have large number of neurons in the hidden layer. Also the DNN performance can be further enhanced by increasing its complexity by using a multilayered architecture shown in Fig. 1(a).

Finally after predicting the next day's stock price index, it will be worthwhile to analyze the weekly stock index trend. For illustrating these two stock markets namely the IBM and Oracle are taken for weekly trend analysis. For finding the R-index, maximum and minimum predicted stock prices variations are obtained using the DEUKF learning approach. A graphical analysis of moving averages reveals that the 25-day moving average lags the 65-day moving average for the IBM stock, while it leads for the Oracle stock. This is also observed from Tables 9 and 10 for both the IBM and Oracle stock closing price data samples and thus it can be inferred that the two week data samples for IBM stock exhibit a down trend using the rules in (58) and (59). On the other hand the Oracle stock closing price data for the same period showed an uptrend. After obtaining the stock price trend, the next step will be to obtain the stock turning points and the technical indicators K, D, and

Table 8
MAPE errors during testing of BSE data set using different ANN models using DE.

Days ahead	MLP	RBFNN	ANFIS	FLANN	DNN (DEUKF)	LLRBFNN	LLWNN
1	2.9825	3.2276	3.0741	2.3441	1.9188	2.8752	2.924
3	4.5576	5.1677	4.7244	4.2055	3.4547	4.4619	4.976
5	5.4390	5.4759	5.2157	4.8933	4.2082	5.3918	5.281
7	6.2503	6.4708	6.2177	5.8592	5.2163	6.4466	6.612
10	7.8726	7.9583	7.3432	6.9007	6.2392	7.8579	7.986
15	9.0252	9.3138	8.5982	8.8549	7.8064	9.6480	9.610

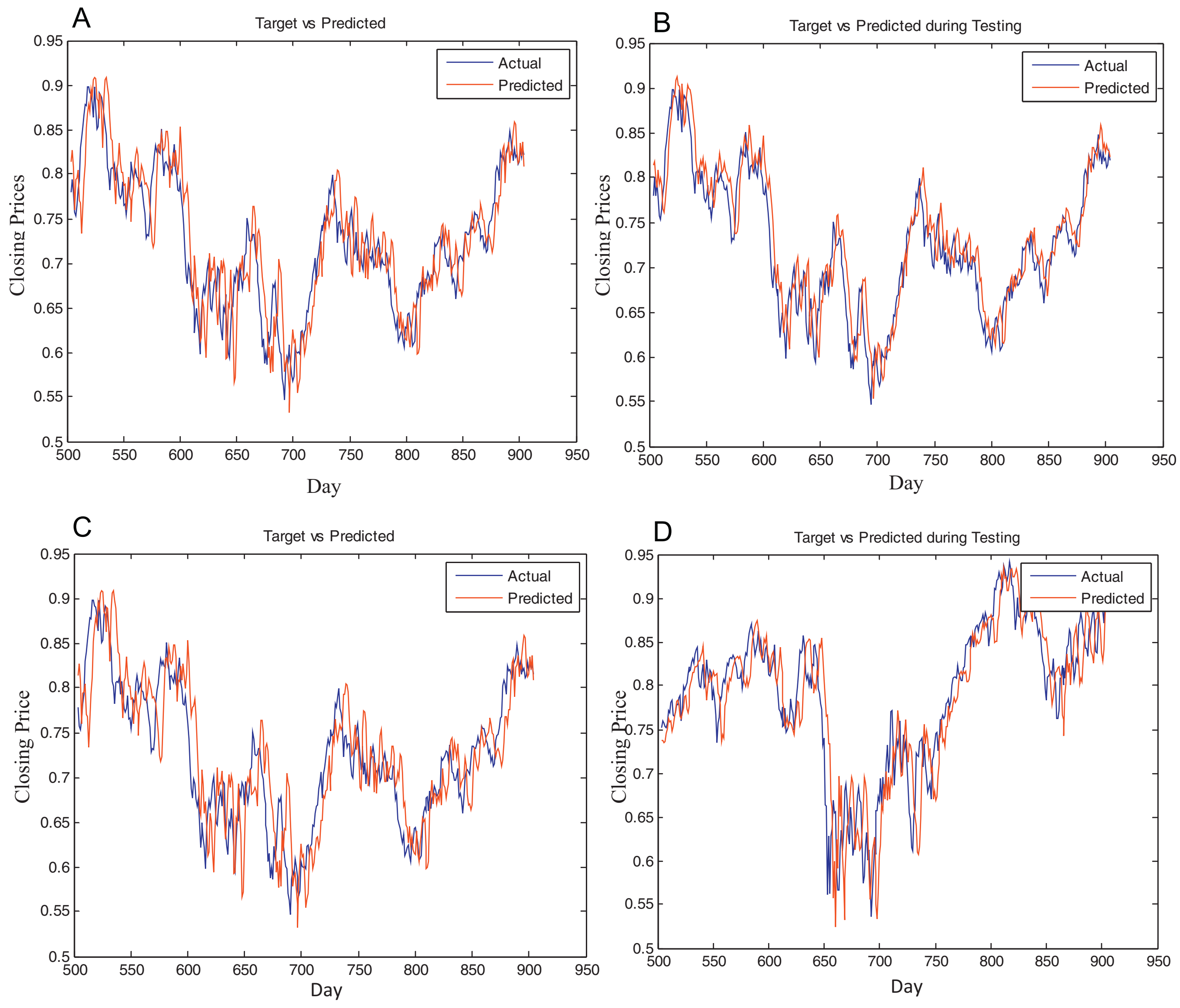


Fig. 6. 3 Days and 7 days prediction results of BSE stock using DE and DEUKF.

Table 9
Technical indicators (MA_25, MA_65, K-index, D-index, R-index) for IBM data.

Stock market	Technical indicators					
	MA_25 (15 data) (750–765)	MA_65 (15 data) (750–765)	K index (750–765)	D index (750–765)	R index (750–765)	Predicted prices (750–765)
IBM	0.5528	0.6349	18.4176	9.6217	81.5824	0.6903
	0.5496	0.6359	4.1584	10.0758	95.8416	0.6807
	0.5479	0.6372	41.3043	21.2934	58.6957	0.6624
	0.5468	0.6386	34.9125	26.7918	65.0875	0.6724
	0.5433	0.6401	35.4167	37.2112	64.5833	0.6353
	0.5402	0.6422	33.9749	34.7680	66.0251	0.6160
	0.5385	0.6452	65.7025	45.0313	34.2975	0.5952
	0.5374	0.6483	43.1818	47.6197	56.8182	0.5595
	0.5402	0.6524	93.5950	67.4931	6.4050	0.5342
	0.5447	0.6574	94.2693	77.0154	5.7307	0.4783
	0.5521	0.6623	69.5814	85.8153	30.4186	0.4960
	0.5575	0.6673	73.8605	79.2371	26.1395	0.4894
	0.5618	0.6719	84.2791	75.9070	15.7209	0.4763
	0.5669	0.6779	76.9645	78.3680	23.0355	0.5532
	0.5703	0.6833	90.9187	84.0541	9.0813	0.5453
	0.5738	0.6893	97.0642	88.3158	2.9358	0.5314

Table 10
Technical indicators (MA_25, MA_65, K-index, D-index, R-index) for Oracle data.

Stock market	Technical indicators					
	MA_25 (15 data) (750–765)	MA_65 (15 data) (750–765)	K index (750–765)	D index (750–765)	R index (750–765)	Predicted prices (750–765)
Oracle	0.8412	0.7328	27.2321	16.4507	72.7679	0.9850
	0.8339	0.7287	1.7778	16.8896	98.2222	0.9829
	0.8261	0.7254	44.4444	24.4848	55.5556	0.9840
	0.8167	0.7222	11.2108	19.1443	88.7892	0.9827
	0.8043	0.7192	47.3451	34.3334	52.6549	0.9446
	0.7946	0.7164	1.6389	20.0649	98.3611	0.9551
	0.7837	0.7135	2.1364	17.0401	97.8636	0.9583
	0.7706	0.7094	1.5280	1.7678	98.4720	0.8930
	0.7620	0.7071	3.0704	2.2449	96.9296	0.8999
	0.7525	0.7047	3.0704	2.5563	96.9296	0.8248
	0.7475	0.7031	2.1408	2.7606	97.8592	0.8821
	0.7391	0.7008	2.0845	2.4319	97.9155	0.8683
	0.7305	0.6989	1.5493	1.9249	98.4507	0.8392
	0.7236	0.6985	2.1127	1.9155	97.8873	0.9046
	0.7148	0.6974	2.9014	2.1878	97.0986	0.8588
	0.7067	0.6980	3.2676	2.7606	96.7324	0.8667

R-index will be used to obtain the stock trading decision. Thus out of the tested 400 samples the 351st sample exhibits a buying point, whereas the 359th sample shows a selling point. However, more technical indicators are needed for ascertaining these results.

6.1. Computational complexity of DEUKF

The accurate computation of UKF alone depends on the sampling strategy of the Sigma point and increasing its number results in greater precision while resulting in greater computational overhead. It has been shown in [40] that by using different Sigma point calculation strategy, its computational complexity can be reduced significantly. In this paper the 250 steps of UKF with 10 variables take nearly 11.6 s in a dual core processor, while combining it with DE in an alternate fashion, it takes nearly 18 s for stock indices prediction. In [40], the computational complexity of UKF alone has been given, which is similar to the one in this paper. Implementation of DE alone takes more than 15 s for 50 generations. The computation will increase substantially with DE alone for multilayered dynamic neural networks.

7. Conclusion

In this paper an innovative learning paradigm like the hybrid UKF and DE (DEUKF) is presented to predict the next day's trend in the four Indian stock markets like Bombay stock exchange (BSE), IBM, Oracle, and the Reliance Industries Limited (RIL) using a simple DNN architecture. In the new DEUKF both the DE and UKF algorithms are executed alternately for a few iterations, before the UKF completely takes over the training process. In addition to the various historical pricing data of the above stock markets several relevant technical and financial indicators are used to improve the accuracy of the trend prediction. Also several performance indices are used to compare the trend prediction accuracies of the proposed four learning strategies for each of the four Indian stocks. From the various computational results, it is observed that the hybrid DEUKF algorithm outperforms other three learning strategies in terms of the various technical indicators such as the lowest MAPE of around 2%, and very small MSE and RMSE. The other three learning methods also perform more or less accurately in predicating stock market trends. The paper also analyses some of the technical indicators in predicting the turning points in the trend thus providing a future framework for buying and selling decisions for booking profits. A comparison with other neural architecture reveals that a simple DNN with less number of weights can outperform other neural architecture when an optimized UKF is used for the learning cycle. The computational complexity of the adaptive UKF can be further reduced by using Rao-Blackwellised unscented Kalman filter (RBUKF) and making it robust by using an H-infinity norm. Also a larger DNN architecture can be used for further reducing the MAPE and RMSE.

Appendix A. Appendix

A.1. Functional link artificial neural network (FLANN)

A wide variety of FLANNs with functional expansion using orthogonal trigonometric functions, Chebyshev polynomial, Laguerre polynomial, and Legendre orthogonal polynomial have been used successfully to forecast stock market price indices. The well known back-propagation algorithm is commonly used to update the weights of the FLANN. In this paper the well known Laguerre polynomials are used as functional blocks for the input data comprising stock market prices and the technical indicators. The Laguerre polynomials are given by

$$\begin{bmatrix} La_0(s) \\ La_1(s) \\ La_2(s) \\ La_3(s) \\ La_4(s) \\ La_5(s) \\ La_6(s) \end{bmatrix} = \begin{bmatrix} 1 \\ (1-s) \\ \left(\frac{s^2}{2} - 2s + 1\right) \\ \left(-\frac{s^3}{6} + \frac{3s^2}{2} - 3s + 1\right) \\ \left(\frac{s^4}{24} - \frac{2s^3}{3} + 3s^2 - 4s + 1\right) \\ \left(-\frac{s^5}{120} + \frac{5s^4}{24} - \frac{5s^3}{3} + 5s^2 - 5s + 1\right) \\ \left(\frac{s^6}{720} - \frac{s^5}{20} + \frac{5s^4}{8} - \frac{10s^3}{3} + \frac{15s^2}{2} - 6s + 1\right) \end{bmatrix} \quad (58)$$

The output from the Laguerre FLANN as shown in Fig. 4(a) is obtained as

$$y_k^{\text{Laguerre}} = \beta_{(0)} + \begin{bmatrix} \beta_{(1,1)} \\ \beta_{(2,1)} \\ \vdots \\ \vdots \\ \beta_{(m-1,1)} \\ \beta_{(m,1)} \end{bmatrix}^T \begin{bmatrix} La_1(x_1) \\ La_2(x_1) \\ \vdots \\ \vdots \\ La_{m-1}(x_1) \\ La_m(x_1) \end{bmatrix} + \cdots + \begin{bmatrix} \beta_{(1,p)} \\ \beta_{(2,p)} \\ \vdots \\ \vdots \\ \beta_{(m-1,p)} \\ \beta_{(m,p)} \end{bmatrix}^T \begin{bmatrix} La_1(x_p) \\ La_2(x_p) \\ \vdots \\ \vdots \\ La_{m-1}(x_p) \\ La_m(x_p) \end{bmatrix} \quad (59)$$

In a functional form it is represented as

$$y_k^{\text{Laguerre}} = f\left(\underbrace{\beta_{(0)}, \beta_{(1,1)}, \ldots, \beta_{(m,n)}}_{\text{variables}}, \underbrace{x_1, \ldots, x_p}_{\text{constants}}\right) \quad (60)$$

and the error between the FLANN output and desired stock value is obtained as

$$e_k = y_k^d - y_k^{\text{Laguerre}} \quad (61)$$

The parameters of the Laguerre FLANN $\underbrace{\beta_{(0)}, \beta_{(1,1)}, \ldots, \beta_{(m,n)}}_{\text{variables}}$ are obtained by DE learning algorithm.

A.2. Wavelet neural network

The output of wavelet neural network is given by

$$y_k(x) = \sum_{m=1}^{M} \omega_m \psi_m(x) = \sum_{m=1}^{M} \omega_m |a_m|^{-(1/2)} \psi_m\left(\frac{x - b_m}{a_m}\right) \quad (62)$$

where ψ_m is the wavelet activation function of the mth unit of the hidden layer and ω_m is the weight connecting the mth unit of the hidden layer to the output layer unit. The $\psi_m(x)$ is called as mother wavelet and the parameters a_m and b_m are the scale and translation parameters and x represents inputs to the WNN model.

The mother wavelet is given by

$$\psi_m(x) = -x \exp\left(-\frac{x^2}{2}\right) \quad (63)$$

WNN is a type of basis function neural network in the sense that the wavelets consists of the basis function. For higher dimension problems WNN requires more hidden units because a large number of basis function units have to be employed to approximate a given system. Thus instead of WNN, a slightly modified WNN known as local linear wavelet neural network is used in this paper for stock market prediction.

A.3. Local linear wavelet neural network

In LLWNN the connection weights between the hidden layer and output layer are replaced by a local linear model. The output of local linear wavelet neural network is given by

$$y = \sum_{m=1}^{M} (\omega_{m0} + \omega_{m1}x_1 + \omega_{m2}x_2 + \cdots + \omega_{mp}x_p)\psi_m(x)$$

$$y = \sum_{m=1}^{M} (\omega_{m0} + \omega_{m1}x_1 + \omega_{m2}x_2 + \cdots + \omega_{mp}x_p)\psi_m(x)$$
$$= \sum_{m=1}^{M} (\omega_{m0} + \omega_{m1}x_1 + \omega_{m2}x_2 + \cdots + \omega_{mp}x_p)|a_m|^{-(1/2)}\psi_m\left(\frac{x - b_m}{a_m}\right) \quad (64)$$

where $x = [x_1, x_2, \cdots, x_p]$ and the linear model is given by

$$L_m = (\omega_{m0} + \omega_{m1}x_1 + \omega_{m2}x_2 + \cdots + \omega_{np}x_p) \quad (65)$$

The activities of linear models $L_m (m = 1, 2, \ldots, M)$ are determined by the associated locally active wavelet functions. A simple wavelet basis function (Mexican Hat) is chosen in this paper for stock market prediction as

$\psi_m(x) = (1 - x^2)\exp(-x^2/2\sigma_s{}^2)$, a typical value of σ_s is chosen between 0.5 and 0.7.

Also the number of hidden layer neurons is equal to the number of inputs. A back propagation learning algorithm is used to obtain the weight matrix W

$$W = \begin{bmatrix} \omega_{01} & \omega_{11} & \omega_{12} & \ldots & \omega_{1p} \\ \omega_{02} & \omega_{21} & \omega_{22} & \ldots & \omega_{2p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \omega_{0p} & \omega_{p1} & \omega_{p2} & \ldots & \omega_{pp} \end{bmatrix} \quad (66)$$

In this model, a local linear RBFNN (LLRBFN) is used instead of the well known RBFNN. The RBFNN converges faster than MLP with less neuron. In order to take advantage of the local capacity of the RBFs while not having too many hidden units, here an alternative type of RBFNN is proposed. The architecture of the proposed LLRBFNN is similar to the one for LLWNN except that the wavelet basis functions are replaced by radial basis functions.

RBFNN has a hidden layer whose activation functions are calculated based on distance of the inputs and center of the activation functions. The hidden layer takes the local linear weight as the input and computes the output. The activation function of the Mth hidden neuron is defined by a Gaussian Kernel as

$$G_M(x) = e^{-||x - C_M||^2/2\sigma_M^2} \quad (67)$$

where σ_M^2 is the parameter for controlling the smoothness of the activation function and C_M is the center of the hidden node and $||x - c_M||$ indicates the Euclidean distance between the inputs and the function center.

LLRBFNN are characterized by localized activation of the hidden layer units. The connecting weights associated with the hidden nodes can be viewed as locally accurate linear models. The mapping function of the network with Gaussian activation function and weighted linear summation in output neuron is given by

$$y = \sum_{m=1}^{M} L_m G_m(L_m) \tag{68}$$

where

$$G_m(L_m) = e^{(-||(L_m - C_M)||^2/2\sigma_M{}^2)} \tag{69}$$

The local linear model L_m is given by

$$[L]_{m=1}^{M} = [\omega_{m0} + \omega_{m1}x_1 + \omega_{m2}x_2 + \cdots + \omega_{nm}x_n] \tag{70}$$

The output at the output layer is given by

$$y = \sum_{m=1}^{M} (\omega_{m0} + \omega_{m1}x_1 + \omega_{m2}x_2 \ldots \omega_{mn}x_n) \cdot e^{(-||(L_m - C_M)||^2/2\sigma_M{}^2)} \tag{71}$$

The objective function is to minimize the error and the mean square error is given by

$$\text{MSE}(e) = \frac{1}{M}\sum_{m=1}^{M}(D_m - \theta_m)^2 \tag{72}$$

where 'D' is the desired vector.

All the above neural architectures use DE learning algorithm.

References

[1] A.P.N. Refenes, A.N. Burgess, Y. Bentz, Neural networks in financial engineering: a study in methodology, IEEE Transactions on Neural Networks 8 (1997) 1222–1267.

[2] H.-J. Kim, K.S. Shin, A hybrid approach based on neural networks and genetic algorithms for detecting temporal patterns in stock markets, Applied Soft Computing 7 (2007) 569–576.

[3] H.S. Kim, S.H. Chun, Graded forecasting using an array of bipolar predictions: application of probabilistic neural networks to a stock market index, International Journal of Forecasting 14 (1998) 323–337.

[4] G. Armano, M. Marchesi, A. Murru, A hybrid genetic-neural architecture for stock indexes forecasting, Information Sciences 170 (2004) 3–33.

[5] I.-H. Kuo, S.-J. Horng, Y.-H. Chen, R.-S. Run, T.-W. Kao, R.-J. Chen, J.-L. Lai, T.-L. Lin, Forecasting TAIFEX based on fuzzy time series and particle swarm optimization, Expert System with Applications 37 (2010) 1494–1502.

[6] M.H.F. Zarandi, B. Rezaee, I.B. Turksen, E. Neshat, A type-2 fuzzy rule-based expert system model for stock price analysis, Expert Systems with Applications 36 (2009) 139–154.

[7] J.-C. Hung, Applying a combined fuzzy system and GARCH model to adaptively forecast stock market volatility, Applied Soft Computing 11 (2011) 3938–3946.

[8] P.C. Chang, C.H. Liu, A TSK type fuzzy rule based system for stock price prediction, Expert Systems with Applications 34 (2008) 135–144.

[9] C.F. Huang, A hybrid stock selection model using genetic algorithms and support vector regression, Applied Soft Computing 12 (2012) 807–818.

[10] S.-C. Huang, Forecasting stock indices with wavelet domain kernel partial least square, Regressions Applied Soft Computing 11 (2011) 5433–5443.

[11] Y.F. Sun, Y.C. Liang, W.L. Zhang, H.P. Lee, W.Z. Lin, Optimal partition algorithm for the RBF neural network for financial time series forecasting, Neural Computing and Applications 14 (2005) 36–44.

[12] S. Yumlu, F.G. Gurgen, N. Okay, A Comparison of global, recurrent and smoothed-piecewise neural models for Istanbul stock exchange prediction, Pattern Recognition Letters 26 (2005) 2093–2103.

[13] T.J. Hsieh, H.F. Hsiao, W.C. Yeh, Forecasting stock markets using wavelet transforms and recurrent neural networks: an integrated system based on artificial bee colony algorithm, Applied Soft computing 11 (2011) 2510–2525.

[14] C.R. Harvey, K.E. Travens, M.J. Costa, Forecasting emerging market returns using neural networks, Emerging Markets Quarterly 4 (2000) 43–55.

[15] M. Qi, G.P. Zhang, Trend time series modeling and forecasting with neural networks, in: IEEE International Conference on Computational Intelligence for Financial Engineering, 2003, pp. 331–337.

[16] C.-T. Lee, Y.-P. Chen, The efficacy of neural networks and simple technical indicators in predicting stock markets, International Conference on Convergence Information Technology (2007) 2292–2297.

[17] M. Bildirici, O. Ersin, Improving forecasts of GARCH family models with the artificial neural networks: an application to the daily returns in Istanbul stock exchange, Expert System with Applications 36 (2009) 7355–7362.

[18] B. Majhi, S. Hasan, F. Mowafak, FLANN based forecasting of S&P 500 index, Information Technology Journal 4 (2005) 289–292.

[19] R. Majhi, G. Panda, G. Sahoo, Development and performance evaluation of FLANN based model for forecasting stock market, Expert systems with Applications 36 (2009) 6800–6808.

[20] Y. Chen, B. Yang, J. Dong, Time series prediction using a local linear wavelet neural network, Neurocomputing 69 (2006) 449–465.

[21] Y. Chen, X. Dong, Y. Zhao, Stock index modeling using EDA based local linear wavelet neural network, in: IEEE Proceedings of International Conference on Neural Networks and Brain, vol. 3, 2005, pp. 1646–1650.

[22] Y. Zhao, Y. Zhang, C. Qi, Prediction model of stock market returns based on wavelet neural network, in: Asia Pacific Workshop on Computational Intelligence and Industrial Application, 2008, pp. 31–36.

[23] C.-F. Liu, C.-Y. Yeh, S.-J. Lee, Application of type-2 neuro-fuzzy modeling in stock price prediction, Applied Soft Computing 12 (2012) 1348–1358.

[24] L.-Y. Wei, T.-L. Chen, T.-H. Ho, A hybrid model based on adaptive-network-based fuzzy inference system to forecast Taiwan stock market, Expert Systems with Applications 38 (1) (2011) 13625–21363.

[25] P.C. Chang, C.Y. Fan, A hybrid system integrating a wavelet and TSK fuzzy rules for stock price forecasting, IEEE Transactions on Man, Machine, and Cybernetics: Part C 38 (2008) 802–815.

[26] Y. Wang, Z. Cai, Q. Zhang, Differential evolution with composite trial vector generation strategies and control parameters, IEEE Transactions on Evolutionary Computation 15 (2011) 55–66.

[27] F.-Y. Huang, Integration of an improved particle swarm optimization algorithm and fuzzy neural network for Shanghai stock market prediction, in: IEEE Workshop on Power Electronics and Intelligent Transportation System, 2008.

[28] T.J. Hsieh, H.F. Hsiao, W.C. Yeh, Forecasting stock markets using wavelet transform and recurrent neural networks: an integrated system based artificial bee colony algorithm, Applied Soft Computing 11 (2011) 2510–2525.

[29] P. Campolucci, A. Uncini, F. Piazza, B.D. Rao, On-line learning algorithms for locally recurrent neural networks, IEEE Transactions on Neural Networks 10 (1999) 253–270.

[30] K. Xiong, H. Zhang, L. Liu, Adaptive robust extended Kalman filter for nonlinear stochastic systems, IET Control Theory and Applications 2 (2008) 239–250.

[31] E.A. Wan, R. van der Merwe, The unscented Kalman filter for nonlinear estimation, in: Proceedings of the IEEE Adaptive Systems for Signal Processing, Communications and Control Symposium, 2000, pp. 153–158.

[32] E.A. Wan, R. van der Merwe, in: S. Haykin (Ed.), The Unscented Kalman Filter, Kalman Filtering and Neural Networks, Wiley, New York, 2001.

[33] P.C. Chang, C.H. Liu, J.L. Lin, C.Y. Fan, C.S.P. Ng, A neural network with a case based window for stock trading prediction, Expert Systems with Applications 36 (2009) 6889–6898.

[34] B.B. Nair, V.P. Mohandas, N.R. Sakthivel, A decision tree-rough set hybrid system for stock market trend prediction, International Journal of Computer Applications 6 (2010) 1–6.

[35] B.B. Nair, N.M. Dharini, V.P. Mohandas, A stock market trend prediction system using a hybrid decision tree-neuro fuzzy system, in: IEEE International Conference on Advances in Recent Technologies in Communication and Computing, 2010, pp. 381–385.

[36] A.C. Tsoi, A.D. Back, Locally recurrent globally feedforward networks: a critical review of architectures, IEEE Transactions on Neural Networks 5 (1994) 229–239.

[37] K. Patan, Stability analysis and the stabilization of a class of discrete-time dynamic neural network, IEEE Transactions on Neural Networks 18 (2007) 660–673.

[38] W.-L. Mao, W.-M. Wang, J. Sheen, P.-H. Chen, The UKF-based RNN predictor for GPS narrowband interference suppression, in: IEEE Australian Communication Theory Workshop, 2012, pp. 7–12.

[39] A. Slowik, M. Bialkoa, Design of IIR digital filters with non-standard characteristics using differential evolution algorithm, Bulletin of the Polish Academy of Sciences 55 (2007) 359–363.

[40] Y. Hao, Z. Xiong, F. Sun, X. Wang, Comparison of Unscented Kalman Filters, in: Proceedings of the 2007 IEEE International Conference on Mechatronics and Automation, Harbin, China, August 5–8, 2007.