Chapter-2

A Hybrid Evolutionary Dynamic Neural Network for Stock Market Trend Analysis and Prediction using Unscented Kalman Filter

2.1. Introduction

Dynamic neural network (DNN) models provide an excellent means for forecasting and prediction of nonstationary time series. A neural network architecture, known as locally recurrent neural network ((LRNN) [71], is preferred to the traditional multilayer perceptron (MLP) because the time varying nature of a stock time series can be better represented using LRNN. The use of LRNN has demonstrated superiority in comparison to the widely used neural networks like the multilayered perceptron (MLP) network, radial basis function (RBF) neural network, and wavelet neural networks (WNN), etc. in predicting highly fluctuating time series databases like electrical load, electricity price, and financial markets.

Therefore, in this chapter, 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 response filter) filters in the forward path providing feedback connections between outputs and inputs. This allows input and output signals to flow in a bidirectional way thereby correctly representing a dynamic system. The only problem, however, is the concern regarding stability of the network when the feedback occurs in a global manner rather than a local one. The local recurrence for an IIR structure can guarantee stability by suitable selection of pole-zero pattern. 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) [72] 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) [72] 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 two important minimum variance estimators like the extended Kalman filter (EKF) [73] and unscented Kalman filters (UKF) [74-76] have been used by researchers for their excellent tracking and predictive abilities. The EKF algorithm uses a first order approximation for linearizing the nonlinear dynamical system to obtain the Jacobian and this may lead to errors in obtaining the posterior mean and covariances of the transformed system. On the other hand the, unscented Kalman filter initially proposed in [74] and extended further in [75,76] is based on third order approximation of the Gaussian process and measurement noise covariances. Also it does not use linearization like EKF and thereby avoids matrix inversion of the Jacobians at every iterations and thus, produces better estimates of the weights of the recurrent NN.

The unscented transformation based on a statistical distribution of the nonlinear system state is the basic starting block of the UKF algorithm and results in a more accurate prediction of the state and noise 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 chapter to provide an accurate prediction even when the inputs fluctuate in a chaotic manner. There are, however, some problems when the input has substantial outliers and noise in the data exhibits a low signal to noise ratio (SNR) in dB. Also the choice of the initial values of the UKF algorithm plays an important role in producing quick convergence and accuracy and to achieve this, it is
proposed to use an evolutionary optimization technique like the differential evolution (DE) [77,78] in this chapter. The DE optimizes 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. Differential evolution (DE) is a population-based stochastic function optimizer, which uses a rather greedy and less stochastic approach for problem solving in comparison to classical evolutionary algorithms, such as genetic algorithms, evolutionary programming, and PSO, etc.. DE combines simple arithmetical operators with the classical operators of recombination, mutation, and selection to evolve from a randomly generated starting population to a final solution. Also, it is easy to implement as there are a few control parameters required to implement the evolutionary algorithm.

Trend analysis in stock markets is an important area of research since it provides the direction of stock market movement that are essential for the investors to decide whether to buy, sell or hold the stocks. As the stock market data are highly nonlinear and chaotic in nature, it is always difficult to predict the nature of future movement of stocks in an upward or downward trend with significant accuracy. Several researchers have used intelligent system techniques [1-25] like neural networks (ANN), support vector machines (SVM), hybrid systems combining ANN or SVM with genetic algorithm (GA) to predict the stock market trend [26-29] on day to day or weekly and fortnightly basis. In their studies, they have used a selected number of technical indicators also known as features and have used regression analysis to predict the future movement of stock price. A widely used data mining approach like the construction of a decision tree [27] to handle the features to classify whether a stock is in uptrend or downtrend or has no significant movement. Once the stock market trend for the next day or week is established, trading decisions like the buying or selling the stocks and the profits that accrue from it can be decided.

It is well known that the technical indicators play a great role in forecasting the stock market indices and thus are also very useful for trend analysis in finding out 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 chapter is organized as follows: In Section 2.2, the architecture of the DNN is proposed. The state space model of the UKF is described in Section 2.3 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 2.4, some of the well known NNs like the FLANN, LLWNN, and a slightly modified RBFNN are used for comparison. Section 2.5 presents the performance evaluation methods used for stock price prediction. Section 2.6 describes the experimental results of stock price prediction. It also 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. In Section 2.7 the computational complexity of DEUKF is described. Section 2.8 presents the stock market trend analysis using certain technical indicators and pertinent rule bases for trend prediction, where as Section 2.9 described the trading rules generation using technical indicators. Finally, conclusion is given in Section 2.10.

2.2. Feed Forward DNN Architecture

The IIR-MLP NN architecture considered in this chapter is similar to the multi-layered feed-forward one comprising locally recurrent dynamic neurons. The proposed model exhibits dynamic properties when IIR (infinite impulse response) filtering units are embedded into the neural network. Fig.2.1 shows the structure and the topology of the dynamic neuron and multilayered dynamic neural network (DNN).

Consequently the dynamic neuron can reproduce its past with an IIR filter between the input and the output by using the lagged filter outputs which become inputs in the successive iterations. 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

$$s_{j,k} = \sum_{i=1}^{p} w_{ij} x_i$$  \hspace{1cm} (2.1)

where $p$ is the number of inputs ($x_1, x_2, x_3, \ldots, x_p$).
(a) A Neural model for IIR Multilayered Perceptron

(b) Dynamic Neuron (DN)

Fig. 2.1 Dynamic neural network architecture for stock price prediction
The weight vector of the DNN is given by

\[ \mathbf{W}_j = [w_{j1}, w_{j2}, \ldots, w_{jm}]^T. \]

The output of the filter is obtained from a linear difference equation of the form

\[ z_{j,k+1} = \sum_{i=0}^{n} b_{ji} s_{j,k-i} + \sum_{i=1}^{m} a_{ji} z_{j,k-i} \]  \hspace{1cm} (2.2)

where the feedback and feedforward filter weights are given by

\[ \mathbf{A} = [a_1, a_2, a_3, \ldots, a_m]^T, \mathbf{B} = [b_1, b_2, \ldots, b_n]^T \] \hspace{1cm} (2.3)

The final output is obtained from the \( j \)th dynamic neuron with tanh activation function as

\[ y_{j,k+1} = \tanh(\theta_1 z_{j,k+1} + \sigma_1) \] \hspace{1cm} (2.4)

where \( \theta_1 \) and \( \sigma_1 \) 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

\[ y_{k+1} = \tanh(\sum_{j=1}^{q} \theta_j z_{j,k+1} + \sigma) \] \hspace{1cm} (2.5)

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

\[ \mathbf{x} = [\mathbf{W}, \mathbf{A}, \mathbf{B}, \mathbf{\theta}, \mathbf{\sigma}]^T \] \hspace{1cm} (2.6)

where the weight matrix \( \mathbf{W} \), and the vector \( \mathbf{\theta} \) are obtained as
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.

2.3. Learning Algorithms

The most common gradient based algorithms used for on-line training of recurrent IIR filter based NNs belong to back propagation (BP) and RTRL learning paradigms [72]. In both these algorithms, gradient descent based first order derivatives are used for adjusting the weights of the recurrent neurons. However, both these approaches use small learning rates and exhibit slow convergence leading to local minima. On the other hand, the UKF has been used [75,76] 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 chapter for training the DNN for the prediction of stock market indices:

2.3.1. 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 [99, 100]. 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

\[
\mathbf{W} = \\
\begin{bmatrix}
w_{11} & w_{12} & \cdots & w_{1n} \\
w_{21} & w_{22} & \cdots & w_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
w_{n1} & w_{n2} & \cdots & w_{nn}
\end{bmatrix}
\] (2.7)

\[
\theta = [\theta_1, \theta_2, \ldots, \theta_q]^T
\]
\[ a_{i,k} = a_{i,k-1} + \mu e_k \sec^2 h(\theta_i z_{i,k} + \sigma_i) \frac{\partial z_{1,k}}{\partial a_{i,k}} \]  
(2.8)

\[ b_{i,k} = b_{i,k-1} + \mu e_k \sec^2 h(\theta_i z_{i,k} + \sigma_i) \frac{\partial z_{1,k}}{\partial b_{i,k}} \]  
(2.9)

\[ w_{i,k} = w_{i,k-1} + \mu e_k \sec^2 h(\theta_i z_{i,k} + \sigma_i) \frac{\partial z_{1,k}}{\partial w_{i,k}} \]  
(2.10)

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

\[ \frac{\partial z_{1,k}}{\partial a_{i,k}} = z_{k-i} + \sum_{j=1}^{m} a_{j,i} \frac{\partial z_{k-j}}{\partial a_{i,k}}, i = 1,2,\ldots,m \]  
(2.11)

\[ \frac{\partial z_{1,k}}{\partial b_{i,k}} = s_{k-i} + \sum_{j=1}^{n} b_{j,i} \frac{\partial z_{k-j}}{\partial b_{i,k}}, i = 1,2,\ldots,n \]  
(2.12)

\[ \frac{\partial z_{1,k}}{\partial w_{i,k}} = x_{k-i} + \sum_{j=1}^{p} w_{j,i} \frac{\partial z_{k-j}}{\partial w_{i,k}}, i = 1,2,\ldots,p \]  
(2.13)

The above equations are used iteratively to generate gradients that are required to update the weights of the locally recurrent dynamic neurons. In the beginning of the learning cycle all the gradients are made zero or very small. It is well known that the use of the above equations make the weights nonlinear so that it will effectively track the nonlinear dynamic system like the stock market indices. However, the relationship between the present and past time-varying filter parameters often leads to mean square error surfaces that are not quadratic in nature and thus, lead to local minima. It is well known that the EKF algorithm produces more effective learning of the DNN parameters due to its second order approximation when compared to the gradient descent algorithm given above. Further, the unscented transformation and third order approximation used
in the UKF algorithm are able to approximate a nonlinear Gaussian function, thereby, achieving a more accurate noise and state covariances. This procedure makes the UKF algorithm more robust and hence, describes the use of this algorithm for DNN training.

2.3.2. Adaptive UKF for IIR Filter NN Training

As mentioned earlier the UKF overcomes limitations of losing the higher terms in the Taylor series expansion in the process of linearization adopted for the EKF to build the Jacobian matrix that results 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 unscented transformation of the system state forms the basis of sigma-point filters which is responsible for evaluating the nonlinear statistics of a random variable undergoing transformation. The unscented Kalman filter belongs to the family of sigma-point filters and several variants exist in the literature for nonlinear state estimation. Further, the UT relies on the principle that a Gaussian distribution is relatively easy to approximate than a nonlinear function and even with a small number of sigma-points can produce viable information set for the state of the nonlinear system. Instead of linearizing the Jacobian matrices, a deterministic sampling and square-root decomposition approach with minimal set of sigma points (2×order of the system state n) is used in the UKF to obtain mean and covariance estimates. Further, the generated sigma-points are propagated through the nonlinear system to obtain the weighted mean and noise and state covariances. The various steps used in the formulation of a nonlinear recursive estimator like the UKF are similar to those of the EKF that include the model and actual system measurements assuming a given noise statistics initially.

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

\[
x_{k+1} = x_k + w(k)
\]

\[
y_k = g(x_k) + v(k) \tag{2.14}
\]

and

\[
x_k = \begin{bmatrix} W, A, B, \theta, \sigma \end{bmatrix}, \quad g(x_k) = \tanh\left( \sum_{j=1}^{q} \theta_j z_{j,k+1} + \sigma \right) \tag{2.15}
\]
where \( w(k) \) and \( v(k) \) are process noise and measurement noise, and \( x(k) \) and \( y(k) \) stand for parameter vector and observation vector, respectively. 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 \( X_{k-1} \) is a 10\( \times \)21 matrix. The UKF algorithm is summarized in the following steps:

**Step 1: Unscented transformation and Sigma Points Calculation**

The \( L \times (2L+1) \) sigma points are computed for a state vector \( \mathbf{x} \) to obtain the new state matrix \( \mathbf{X} \) as

\[
X_{0,k-1} = \hat{x}_{k-1}, \quad i = 0
\]

\[
X_{i,k-1} = \hat{x}_{k-1} + \left( \sqrt{(n+\lambda)P_{k-1}^i} \right) \left( (n+\lambda)P_{k-1}^i \right), \quad i = 1, 2, ..., n
\]

(2.16)

\[
X_{i+n,k-1} = \hat{x}_{k-1} - \left( \sqrt{(n+\lambda)P_{k-1}^i} \right) \left( (n+\lambda)P_{k-1}^i \right), \quad i = n+1, ..., 2n
\]

\( \left( \sqrt{(n+\lambda)P_{k-1}^i} \right) \) is the \( i \)th column of the matrix square root [22,23] of \( (n+\lambda)P_{k-1} \).

The parameter \( \lambda \) is used to control the covariance matrix, and is given by

\[
\lambda = \alpha^2 (n+\kappa) - n
\]

(2.17)

The scaling parameters \( \lambda \) and \( \kappa \) determine the spread of the sigma points around \( \hat{x} \). Also the factor \( \kappa \) has the effect of reducing the higher order errors of the mean and covariance approximations. The constant \( \alpha \) is usually set to \(-4 \leq \alpha \leq 1\), 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

\[
\hat{x}(0) = E[\mathbf{x}]
\]

\[
P(0) = E[(\mathbf{x} - \hat{x}(0))(\mathbf{x} - \hat{x}(0))^T] = \varepsilon^{-1} \mathbf{I}, \quad R(0) = r, \quad Q(0) = q \mathbf{I}
\]

(2.18)
Where \( \varepsilon, r, \) and \( q \) are chosen appropriately and \( I \) is a unit matrix of appropriate dimension.

After computing the sigma points the time update of state estimates are given by
\[
\hat{x}_k = \sum_{i=0}^{2n} W_i^{m} X_{i,k|k-1}
\]
(2.19)
where the weights \( W_i^{(m)} \) are defined by
\[
W_0^{(m)} = \frac{\lambda}{n + \lambda}, \quad W_i^{(m)} = \frac{\lambda}{2(n + \lambda)}, \quad W_{i+L}^{(m)} = \frac{1}{2(n + \lambda)}, \quad i = 1, \ldots, n
\]
(2.20)

The a priori error covariance is given by
\[
\bar{P}_k = \sum_{i=0}^{2n} W_i^{(c)} [x_{i,k|k-1} - \hat{x}_k][x_{i,k|k-1} - \hat{x}_k]^T + Q_k
\]
(2.21)
\[
W_0^{(c)} = \frac{\lambda}{n + \lambda} + (1 - \alpha^2 + \beta), \quad W_i^{(c)} = \frac{1}{2(n + \lambda)} + (1 - \alpha^2 + \beta), \quad W_{i+L}^{(c)} = \frac{1}{2(n + \lambda)}, \quad i = 1, \ldots, n
\]
(2.22)
(2.23)

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
\[
Y_{i,k|k-1} = g_k(X_{i,k|k-1})
\]
(2.24)
\[
\hat{y}_k = \sum_{i=0}^{2L} W_i^{(m)} Y_{i,k|k-1}
\]
(2.25)

**Step 2: Measurement Update**

The unscented transformation yields the covariance matrix of the vector \( Y \) as
\[
S_k = \sum_{i=0}^{2L} W_i^{(c)} \left[ Y_{i,k|k-1} - \hat{y}_k \right] \left[ Y_{i,k|k-1} - \hat{y}_k \right]^T + R_k
\]
(2.26)
Further the cross covariance matrix between \( X \) and \( Y \) is obtained as

\[
U_i = \frac{2L_t}{\sum_i W_i^{(c)}} \left[ X_{i,k|k-1} - \hat{x}_k \right] \left[ Y_{i,k|k-1} - \hat{y}_k \right]^T
\]  

(2.27)

Then the Kalman gain \( K_k \), the state estimate \( \hat{x}_k \), and the error covariance \( P_k \) are obtained as

\[
K_k = U_k S_k^{-1}
\]  

(2.28)

The observation error is obtained as

\[
e_k = (y_k - \hat{y}_k)
\]  

(2.29)

and the updated parameter vector is obtained as

\[
\hat{x}_k = \hat{x}_k + K_k e_k
\]  

(2.30)

To improve the tracking performance of the filter, the error covariance matrix \( P_k \) is obtained as

\[
P_k = [\bar{P}_k]^{-1} - K_k S_k K_k^T \quad \text{for} \quad S_k > \varepsilon \, \bar{S}_k
\]  

(2.31)

and

\[
P_k = [(\bar{P}_k)^{-1} - \gamma^{-1}I]^{-1} - K_k S_k K_k^T, \quad \text{otherwise.}
\]  

(2.32)

\[
\bar{S}_k = \begin{cases} 
  e_k e_k^T, & k = 0 \\
  \xi \frac{S_{k,k-1} + e_k e_k^T}{\xi + 1}, & k > 0 
\end{cases}
\]  

(2.33)

and \( \xi \) is a forgetting factor and is chosen as 0.98.

where \( \bar{S} = E[e_k e_k^T] \) 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 \( \gamma \) 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 \( \gamma \), but the mean square error also increases.

Thus, a suitable value of \( \gamma \) needs to be selected for making the filter robust and reduce
the mean square error simultaneously. Thus, it requires to minimize the following
transfer function norm between the state estimation error and the noise covariances to
satisfy the attenuation level constraint \( \gamma \):

\[
\frac{\| \mathbf{I} \cdot (\hat{x}_k - \hat{x}_k^-) \|^2}{\| S_k \|^2 + \| P_k \|^2} < \gamma
\]

(2.34)

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 \( \alpha \), \( \beta \), 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 chapter 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

\[
G_k = \frac{1}{N_1} \sum_{i=k-N_1}^{k} v_k v_k^T
\]

(2.35)

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

\[
\hat{G}_k = \sum_{i=0}^{2L} W_i^{(c)} \left[ y_i, k|k - 1 = \hat{y}_k^- \right] \left[ y_i, k|k - 1 = \hat{y}_k^- \right]^T + R_k
\]

(2.36)

A cost function to minimize \( \hat{G}_k \) is formulated as

\[
V_k = tr[(G_k - \hat{G}_k)^2]
\]

(2.37)

The cost function can be minimized by choosing the value of \( q \) of the \( Q \) matrix as

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

(2.38)
In a similar way the measurement covariance matrix is updated as

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

(2.39)

$\rho$ is a forgetting factor and is initially taken as $= 0.9$.

The forgetting factor can be further tuned as

$$\rho_k = \rho_{\text{min}} + (\rho_{\text{max}} - \rho_{\text{min}}) \exp(-\left(\frac{r_k}{r_{0}}\right))$$

(2.40)

Where the estimation error covariance is obtained iteratively as,

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

(2.41)

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

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 $\alpha, \beta, Q$, 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 chapter to obtain the suitable parameters $\alpha, \beta, q, r$ 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.

### 2.3.3. 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, forecasting, 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. 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 encodes the candidate solutions towards the global optimum for a population size \( N_p \) having \( D \)-dimensional vectors of variables to be optimized. The parameters of the \( i^{th} \) vector for the generation \( g \) (\( g = 1,2,3, \ldots, G \)) are given in equation (2.42):

\[
XX^g_{(i)} = \{x_{x_{(i,1)}^g}, x_{x_{(i,2)}^g}, \ldots, x_{x_{(i,j)}^g}, \ldots, x_{x_{(i,D)}^g}\}
\]  
(2.42)

\( i = 1,2,3, \ldots, N_p \), and \( G \) = total number of generations used in the optimization process.

The initial population should be bounded by the prescribed minimum and maximum parameter limits to cover the entire search space

\[
XX_{\text{min}} = \{x_{x_{1}^\text{min}}, x_{x_{2}^\text{min}}, \ldots, x_{x_{j}^\text{min}}, \ldots, x_{x_{D}^\text{min}}\}
\]  
(2.43)

\[
XX_{\text{max}} = \{x_{x_{1}^\text{max}}, x_{x_{2}^\text{max}}, \ldots, x_{x_{j}^\text{max}}, \ldots, x_{x_{D}^\text{max}}\}
\]  
(2.44)

In the first generation, the \( j^{th} \) parameter of the \( i^{th} \) vector is initialised in a random manner as

\[
x_{x_{(i,j)}^1} = x_{x_{j}^\text{min}} + \text{rand}_{(i,j)} \times (x_{x_{j}^\text{max}} - x_{x_{j}^\text{min}})
\]  
(2.45)

where \( \text{rand}_{(i,j)} \in (0,1) \).

**Step 1: Mutation Operation**

For each target vector \( XX \) at the generation \( g \), the mutant vector \( V \) is generated in the following manner:

\[
V^g_{(i)} = \{v_{v_{(i,1)}^g}, v_{v_{(i,2)}^g}, \ldots, v_{v_{(i,j)}^g}, \ldots, v_{v_{(i,D)}^g}\}
\]  
(2.46)
Further, the five most frequently used mutation strategies used in DE are listed below:

1) “DE/rand/1”:

\[ V_{(i)}^g = XX_{(\mu)}^g + F \times (XX_{(\delta)}^g - XX_{(\gamma)}^g) \]  
(2.47)

2) “DE/best/1”:

\[ V_{(i)}^g = XX_{best}^g + F \times (XX_{(\mu)}^g - XX_{(\delta)}^g) \]  
(2.48)

3) “DE/rand-to-best/1”:

\[ V_{(i)}^g = XX_{(i)}^g + F_1 \times (XX_{best}^g - XX_{(i)}^g) + F_2 \times (XX_{(\mu)}^g - XX_{(\delta)}^g) \]  
(2.49)

4) “DE/best/2”:

\[ V_{(i)}^g = XX_{best}^g + F_1 \times (XX_{best}^g - XX_{(i)}^g) + F_2 \times (XX_{(\mu)}^g - XX_{(\delta)}^g) \]  
(2.50)

5) “DE/rand/2”:

\[ V_{(i)}^g = XX_{(\mu)}^g + F_1 \times (XX_{(\delta)}^g - XX_{(\gamma)}^g) + F_2 \times (XX_{(\mu)}^g - XX_{(\delta)}^g) \]  
(2.51)

The indices, \(\mu, \delta, \gamma, \zeta, \eta\) within the range \([1, Np]\) are mutually exclusive integers and are randomly generated once for each mutant vector in a particular generation \(g\). The scaling factors \(F, F_1\) and \(F_2\) are positive control parameters for generating the mutant vector at each generation. Like many other evolutionary algorithms, the DE may get stuck at a local optimum without yielding an optimal solution and hence, the scaling factors which control the mutation operation are adapted in a time varying manner. More feasible optimal solutions are possible due to the variation of the decision variables with the changes in scaling factors.

The scaling factors \(F, F_1\) and \(F_2\) are changed in the following way:

\[ F^g = F^* + \delta \cdot \phi^g \]  
(2.52)

\[ F_1^g = F_1^* + \delta_1 \cdot \phi_1^g \]  
(2.53)

\[ F_2^g = F_2^* + \delta_2 \cdot \phi_2^g \]  
(2.54)
where $F^e$ values are the time varying enhanced scaling factors $F^r$ values are the constant offsets which act as the mean around which the scaling factors oscillate. The $\phi$ values represent random values which follow normal distribution function with zero as mean. The $\vartheta$ values are the multipliers which amplify the random values.

**Step 2: Crossover Operation**

After the mutation phase, the trial vector $U_{(i)}^g$ is generated from each pair of target vector $XX_{(i)}^g$ and its mutant $V_{(i)}^g$ in the crossover operation as

$$U_{(i)}^g = \{ u_{(i,1)}^g, u_{(i,2)}^g, \ldots, u_{(i,j)}^g, \ldots, u_{(i,D)}^g \}$$

$$u_{(i,j)}^g = \begin{cases} v_{(i,j)}^g & \text{if } rand_{(i,j)} \leq Cr \text{ or } j = j_{rand} \\ x_{(i,j)}^g & \text{otherwise} \end{cases}$$

In Eq.(2.56), the crossover rate $Cr$ is a user-specified constant within the range $(0, 1)$. The binomial crossover operator copies the $j$th parameter of the mutant vector $V_{(i)}^g$ to the corresponding element in the trial vector $U_{(i)}^g$ if $rand_{(i,j)} \leq Cr$ or $j = j_{rand}$; Otherwise, it is copied from the corresponding target vector $XX_{(i)}^g$.

$$u^{g+1}_{(i,j)} = \begin{cases} v^{g+1}_{(i,j)} & \text{if } randb(j) \leq CR \text{ or } j = rnbr(i) \\ x^{g}_{j,i} & \text{if } randb(j) > CR \text{ and } j \neq rnbr(i) \end{cases}$$

**Step 3: Selection Operation**

If the values of some parameters of a newly generated trial vector exceed the corresponding upper and lower bounds, we randomly and uniformly reinitialize them within the pre-specified range. Then, the objective function values of all trial vectors are evaluated. After that, a selection operation is performed. The objective function value of each trial vector $f(U_{(i)}^g)$ is compared to that of its corresponding target vector $f(XX_{(i)}^g)$ in the current population. If the trial vector has less or equal objective function value than the corresponding target vector, the trial vector will replace the target vector and enter the population of the next generation. Otherwise, the target vector will remain in
the population for the next generation. The selection operation can be expressed as follows:

$$XX_{(i)}^{g+1} = \begin{cases} U_{(i)}^g & \text{if } f(U_{(i)}^g) \leq f(XX_{(i)}^g) \\ XX_{(i)}^g & \text{if } f(U_{(i)}^g) > f(XX_{(i)}^g) \end{cases}$$

(2.58)

**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 $i^{th}$ vector for the generation $g$ are given by Eq. (2.59).

$$XX_{(i)}^g = \{xx_{(i,1)}^g, xx_{(i,2)}^g, \ldots, xx_{(i,D)}^g\}$$

(2.59)

A random initialization is done for the $j^{th}$ parameter of the $i^{th}$ vector at the 1$^{st}$ generation as

$$xx_{(i,j)}^1 = xx_{j}^{\text{min}} + rand_{(i,j)} \times (xx_{j}^{\text{max}} - xx_{j}^{\text{min}})$$

(2.60)

While stopping criterion is not satisfied
\[ \text{do} \]

2. Mutation

for $i = 1$ to $Np$

$$F_1 = 0.5 + 0.45 \times rand(0,1)$$

$$F_2 = 0.3 + 0.5 \times rand(0,1)$$

(2.61)

Generate the mutant vector from the target vector using Eq.(2.59)

$$V_{(i)}^g = \{v_{(i,1)}^g, v_{(i,2)}^g, \ldots, v_{(i,D)}^g\}$$

$$V_{(i)}^g = XX_{(i)}^g + F_1 \times (XX_{(i)}^g - XX_{(i)}^g) + F_2 \times (XX_{(i)}^g - XX_{(i)}^g)$$

(2.62)
The indices, $\mu, \delta, \gamma, \xi, \eta \in [1, Np]$ 

end for

3. Crossover

for $i = 1$ to $Np$

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

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

$$Cr = \begin{cases} 
\text{rand3}, & \text{if rand4} \leq 0.1 \\
Cr, & \text{otherwise}
\end{cases} \quad (2.63)$$

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

Generate a trial vector $u_{(i,j)}^g$ using Eq.(2.64)

$$U_{(i)}^g = \left\{ u_{(i,1)}^g, u_{(i,2)}^g, \ldots, u_{(i,j)}^g, \ldots, u_{(i,D)}^g \right\}$$

$$u_{(i,j)}^g = \begin{cases} 
\nu_{(i,j)}^g, & \text{if } (rand_{(i,j)} \leq Cr) \text{ or } j = j_{rand} \\
x_{(i,j)}^g, & \text{otherwise}
\end{cases} \quad (2.64)$$

end for

end for

4. Selection:

for $i = 1$ to $Np$

Evaluate the objective function of the trial vector

$$U_{(i)}^g = \left\{ u_{(i,1)}^g, u_{(i,2)}^g, \ldots, u_{(i,j)}^g, \ldots, u_{(i,D)}^g \right\} \quad (2.65)$$
The objective function is obtained from the UKF algorithm as

\[ f(U(x)) = \frac{1}{MM} \sum_{k=1}^{MM} e_k^2, \]

where \( MM \) = iteration number used for evaluation from Eq.(2.29).

and the fitness of the \( i \)th vector \( F_i = \frac{1}{1 + f(U(x))} \) \hspace{1cm} (2.66)

\[ XX^{g+1} = \begin{cases} U(x) & \text{if } f(U(x)) \leq f(XX) \\ XX & \text{if } f(U(x)) > f(XX) \end{cases} \] \hspace{1cm} (2.67)

end for

\[ g = g + 1 \]

5. end While

2.4 Other Types of Neural Architectures for Stock Market Prediction

The following neural network architectures [18-22] are used for comparing the prediction performance of the proposed dynamic neural network trained by evolutionary unscented Kalman filter.

2.4.1 Functional Link Artificial Neural network (FLANN)

A wide variety of FLANNs [18,19] 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 chapter, 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:
The output from the Laguerre FLANN as shown in Fig.2.2 (a) is obtained as

\[
y_k^{\text{Laguerre}} = \beta_0 + \sum_{i=1}^{m} \beta_i \cdot \sum_{j=0}^{n} La_j(x_i)\]

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

(2.68)

Fig.2.2 (a) Laguerre FLANN Model
In a functional form it is represented as

\[ y_k^{\text{Laguerre}} = f \left( \beta(0), \beta(1,1), \ldots, \beta(m,n), x_1, \ldots, x_p \right) \]

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

\[ e_k = y_k^d - y_k^{\text{Laguerre}} \]

The parameters of the Laguerre FLANN \( \beta(0), \beta(1,1), \ldots, \beta(m,n) \) are obtained by DE learning algorithm.

2.4.2 Wavelet Neural Network (WNN)

The well known wavelet transform (WT) decomposes a time series into a set of constitutive time series which exhibits an uniform pattern thereby making the overall prediction more accurate. The wavelet neural network (WNN) exploits the nature of WT and uses wavelet basis function for forecasting and classification.

The output of the WNN is obtained as

\[ y_k(x) = \sum_{i=1}^{M} \omega_i \phi_i(x) = \sum_{i=1}^{M} \omega_i \left[ a_i \right]^\frac{1}{2} \phi_i \left( \frac{x - b_i}{a_i} \right) \]

and \( \phi_i = \) WNN activation function of the ith hidden layer,

\( a_i, b_i = \) scaling and translation parameters

\( \omega_i = \) weight parameter between the ith hidden layer to the output layer.

The wavelet basis function chosen here is given by

\[ \phi_i(x) = -x \exp \left( -\frac{x^2}{2} \right) \]
In a wavelet neural network the choice of the number of hidden layer neurons is a difficult task and may require trial and error approach till the correct number is found. This, however, depends on the dimension of the problem, and higher is the dimension, the more complex it is. Thus, instead of WNN, a slightly modified WNN known as LLWNN (local linear wavelet neural network) is used for stock market prediction considered in this chapter.

### 2.4.3 Local Linear Wavelet Neural Network

In this type of network shown in Fig.2.2 (b) the connection weights between the ith hidden layer and the output layer comprises a local linear model of the form

\[ \omega_i = \omega_{i0} + \omega_{i1}x_1 + \omega_{i2}x_2 + \ldots + \omega_{ip}x_p \]  

(2.74)

![Neural Network Architectures for comparison](https://via.placeholder.com/150)

\[ y = \sum_{i=1}^{M} \left( \omega_{i0} + \omega_{i1}x_1 + \omega_{i2}x_2 + \ldots + \omega_{ip}x_p \right) \phi_i(x) \]

\[ = \sum_{i=1}^{M} \left( \omega_{i0} + \omega_{i1}x_1 + \omega_{i2}x_2 + \ldots + \omega_{ip}x_p \right) a_i \left[ \frac{1}{2} \phi_i \left( \frac{x - b_i}{a_i} \right) \right] \]  

(2.75)
where \( x = [x_1, x_2, \ldots, x_p] \) and the linear model is given by
\[
L_m = (\omega_{m0} + \omega_{m1}x_1 + \omega_{m2}x_2 + \ldots + \omega_{mp}x_p)
\]  
(2.76)

where \( m=1,2,\ldots,M \)

A simple wavelet basis function (Mexican Hat) is chosen in this chapter for stock market prediction as
\[
\phi_i(x) = (1 - x^2)\exp(-x^2/2\sigma_i^2), \text{ a typical value of } \sigma_i \text{ is chosen between 0.5 and 0.7.}
\]

The weight matrix \( W \) shown below for the local weight models is obtained as
\[
W = \begin{bmatrix}
\omega_{01} & \omega_{11} & \omega_{12} & \ldots & \omega_{1p} \\
\omega_{02} & \omega_{21} & \omega_{22} & \ldots & \omega_{2p} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\omega_{0p} & \omega_{p1} & \omega_{p2} & \ldots & \omega_{pp}
\end{bmatrix}
\]  
(2.77)

Another variation of the LLWNN is the LLRBFNN (local linear radial basis function neural network) in which the wavelet basis functions are replaced by the radial basis functions of the form
\[
\mu(x) = \exp\left(\frac{(x - c)^2}{\sigma^2}\right)
\]  
(2.78)

where \( c \) and \( \sigma \) represent the centre and standard deviation of the RBF unit.

Thus, with \( n \) number of input units the output of the ith hidden neuron is obtained as
\[
\psi_i = \exp(-0.5 \exp\left[\frac{(x_1 - c_i)^2}{\sigma_i^2} + \frac{(x_2 - c_i)^2}{\sigma_i^2} + \ldots + \frac{(x_p - c_i)^2}{\sigma_i^2}\right])
\]  
(2.79)

\[
= \exp\left[\frac{(x - c)^2}{\sigma}\right] \text{, a Gaussian kernel}
\]  
(2.80)

and
\| x - c \|_2 = \text{the Euclidean distance between the inputs and the radial basis function center.}

Using \( m = M \) number of local units the total output of the LLRBFNN is given by

\[
O = \sum_{i=1}^{M} L_i \psi_i = \sum_{i=1}^{M} L_i \exp\left( \frac{(x - c)^2}{\sigma^2} \right)
\]  

(2.81)

To obtain the weights of the local elements of the LLRBFNN an objective function of the form given below is used:

\[
\text{Objective function } Obj = \frac{1}{K} \sum_{k=1}^{K} (\text{Des}(k) - O(k))^2
\]  

(2.82)

\( \text{Des}(k), O(k) = \) desired and output values at the kth iteration, respectively.

All the above neural architectures use DE learning algorithm.

### 2.5 Performance Evaluation

In the time series prediction problems, several performance measures like the mean square error (MSE), root mean square (RMSE), the mean absolute percentage error (MAPE), average MAPE, variances, etc. are widely used to determine the accuracy of the forecasting paradigm [80]. However, many researchers have other measures for determining the effectiveness of the prediction process, but they are not used here. The following measures are mathematically depicted as:

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

(2.83)

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

(2.84)

In order to avoid the adverse effect of very small stock indices over a long period of time, another MAPE known as AMAPE given in eqn.(2.85) is adopted and compared for all the learning models. 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 given in eqn.(2.86).

\[
\text{MAPE} = \frac{1}{N} \sum_{i=1}^{N} \left| \frac{y_i - \hat{y}_i}{\frac{1}{N} \sum_{i=1}^{N} y_i} \right| \times 100
\] (2.85)

\[
\sigma^2 = \frac{1}{ND} \sum_{i=1}^{ND} \left\{ \frac{|y_i - \hat{y}_i|}{\frac{1}{ND} \sum_{i=1}^{N} y_i} - \frac{1}{ND} \sum_{i=1}^{N} \left| y_i - \hat{y}_i \right| \right\}^2
\] (2.86)

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.

2.6. 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 Bombay Stock Exchange (BSE), Oracle Stock Market, Reliance Stock Market (RIL), IBM Stock Market, and, 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. (2.87), and the normalized values along with technical indicators are fed into DNN as inputs for future stock price prediction. The scaling is done as:

\[
\tilde{X}^i = (X^i - X_{\min}) / (X_{\max} - X_{\min})
\] (2.87)

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 \( N_p = 10 \); 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.(2.62). The normal distribution function (represented by \( \phi^s \) and \( \phi^g \)) generates positive and negative values in continuous domain with mean as zero. For DE learning procedure the \( N_p = 30 \); \( D = 12 \) for one dynamic neuron; \( G = 500 \). A flow chart for the implementation of the hybrid learning and prediction scheme is shown in Figs.2.3 (a), and 2.3(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 1000 \( I \); \( I \) 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 2.3.2.

The prediction results for BSE, Oracle, RIL, and IBM stock indices using the proposed dynamic neural model trained with DE and DEUKF learning approaches are shown in Figs. 2.4 (a) – 2.4 (h), 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 -2.1 to 2.4 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.

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 to 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 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.
Start

Initialize UKF parameters
\[XX]=\{\alpha, \beta, q, R, \Theta, \sigma\]

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.(2.63)
and trial vector \(U_i^g\) using Eq. (2.65)

\(i = i + 1\)

\(i \leq N_p\)

Yes

Go to UKF loop. Obtain
\(f(U_i^g) = (1/N)\sum_{k=1}^{N_N} e_i^2\) using Eq. (2.67)

\(XX_{i+1} = XX_i\)

\(f(U_i^g) < f(XX_i^g)\)

\(g = g + 1\)

\(g > G\)

Yes

Stop

No

(a) Flow chart for training DEUKF
Start

Use optimized UKF parameters $\alpha, \beta, R, q, \theta, \sigma$

Read the final trained weights of DNN $W, A, B$ after 100 iterations

Test data samples $j=1$

Compute sigma points using Eq. (2.16)

Estimate model and measurement covariances, $\hat{p}, S, U$ from Eqs. (2.21), (2.26), (2.27).

Determine UKF gain (Eq. 2.27)

Set tuning parameters $\epsilon, \gamma, \xi, \mu, q$, and compute $S_k$

Check $S_k > \epsilon S_k$

Update error covariance

Update error covariance

Evaluate $G_k, \hat{G}_k, V_k$ Eqs. (2.35-2.37)

Determine $q_k$ and $Q$ Eq. (2.38)

Determine $r_k, R$ Eq. (2.41)

Find MAPE, MSE, Variance

Stop

Yes $j > 200$

No

(b) Flow chart for testing DEUKF

Fig. 2.3. Flow chart for training and testing of the proposed DEUKF model
(a) BSE Stock Prices (DE)

(b) BSE stock Prices (DEUKF)
(c) Oracle Stock Prices (DE)

(d) Oracle Stock Prices (DEUKF)
(e) RIL Stock Prices (DE)

(f) RIL Stock Prices (DEUKF)
Fig. 2.4. Prediction Results of BSE, Oracle, RIL and IBM using DE and DEUKF
Table-2.1: Comparison amongst DE, UKF, and DE-UKF using BSE stock data

<table>
<thead>
<tr>
<th>Stock Market</th>
<th>MAPE</th>
<th>AMAPE</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSE (400 data)</td>
<td>3.60</td>
<td>3.52</td>
<td>0.0387</td>
</tr>
<tr>
<td>(504-903)</td>
<td>3.42</td>
<td>3.32</td>
<td>0.0276</td>
</tr>
<tr>
<td></td>
<td>2.97</td>
<td>2.92</td>
<td>0.0315</td>
</tr>
<tr>
<td></td>
<td>2.55</td>
<td>2.54</td>
<td>0.0200</td>
</tr>
<tr>
<td></td>
<td>2.55</td>
<td>2.51</td>
<td>0.0243</td>
</tr>
<tr>
<td></td>
<td>2.0450</td>
<td>2.0424</td>
<td>0.0143</td>
</tr>
<tr>
<td></td>
<td>4.36</td>
<td>4.38</td>
<td>0.092</td>
</tr>
<tr>
<td></td>
<td>4.13</td>
<td>4.08</td>
<td>0.013</td>
</tr>
</tbody>
</table>

Table-2.2: Comparison amongst DE, UKF, and DE-UKF using Oracle stock data

<table>
<thead>
<tr>
<th>Stock Market</th>
<th>MAPE</th>
<th>AMAPE</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle (400 data)</td>
<td>4.58</td>
<td>4.62</td>
<td>0.0624</td>
</tr>
<tr>
<td>(504-903)</td>
<td>3.80</td>
<td>3.82</td>
<td>0.04898</td>
</tr>
<tr>
<td></td>
<td>3.79</td>
<td>3.78</td>
<td>0.0346</td>
</tr>
<tr>
<td></td>
<td>3.41</td>
<td>3.36</td>
<td>0.0360</td>
</tr>
<tr>
<td></td>
<td>3.62</td>
<td>3.70</td>
<td>0.0300</td>
</tr>
<tr>
<td></td>
<td>3.192</td>
<td>3.248</td>
<td>0.0297</td>
</tr>
<tr>
<td></td>
<td>5.32</td>
<td>5.362</td>
<td>0.0905</td>
</tr>
<tr>
<td></td>
<td>4.39</td>
<td>4.421</td>
<td>0.0818</td>
</tr>
</tbody>
</table>

Table-2.3: Comparison amongst DE, UKF, and DE-UKF using RIL stock data

<table>
<thead>
<tr>
<th>Stock Market</th>
<th>MAPE</th>
<th>AMAPE</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIL (400 data)</td>
<td>4.68</td>
<td>4.74</td>
<td>0.0529</td>
</tr>
<tr>
<td>(504-903)</td>
<td>3.54</td>
<td>3.69</td>
<td>0.0213</td>
</tr>
<tr>
<td></td>
<td>3.21</td>
<td>3.33</td>
<td>0.0278</td>
</tr>
<tr>
<td></td>
<td>2.66</td>
<td>2.75</td>
<td>0.0191</td>
</tr>
<tr>
<td></td>
<td>2.70</td>
<td>2.73</td>
<td>0.0226</td>
</tr>
<tr>
<td></td>
<td>2.205</td>
<td>2.43</td>
<td>0.0138</td>
</tr>
<tr>
<td></td>
<td>5.45</td>
<td>5.48</td>
<td>0.0700</td>
</tr>
<tr>
<td></td>
<td>4.36</td>
<td>4.41</td>
<td>0.0616</td>
</tr>
</tbody>
</table>
Table-2.4: Comparison amongst DE, UKF, and DE-UKF using IBM stock data

<table>
<thead>
<tr>
<th>Stock Market</th>
<th>MAPE</th>
<th>AMAPE</th>
<th>RMSE</th>
<th>MAPE</th>
<th>AMAPE</th>
<th>RMSE</th>
<th>MAPE</th>
<th>AMAPE</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UKF (400 data)</td>
<td>DE (200 data)</td>
<td>DEUKF (400 data)</td>
<td>RTRL (200 data)</td>
<td>(504-903)</td>
<td>(504-703)</td>
<td>(504-903)</td>
<td>(504-703)</td>
<td>(504-903)</td>
</tr>
<tr>
<td>IBM</td>
<td>4.62</td>
<td>4.16</td>
<td>3.96</td>
<td>3.55</td>
<td>2.758</td>
<td>5.78</td>
<td>5.36</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.34</td>
<td>4.09</td>
<td>3.72</td>
<td>3.52</td>
<td>2.648</td>
<td>5.653</td>
<td>5.302</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0458</td>
<td>0.0412</td>
<td>0.0331</td>
<td>0.0261</td>
<td>0.0263</td>
<td>0.0787</td>
<td>0.0700</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The corresponding values between DEUKF and UKF are 9.88, 3.08e-05, and 4.81, and .0015, respectively. Since the value of \( p \) is less than the .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-2.5 shows the error variance for a period of 30 days and computed on different days for both DE and DEUKF learning models, using BSE and RIL stock markets, which clearly reveal the prediction accuracy of DEUKF over DE.

Table -2.5: Error Variance of BSE and RIL stock data using different ANN models using DE

<table>
<thead>
<tr>
<th>Time Period</th>
<th>Error Variance (DEUKF) (BSE)</th>
<th>Error Variance (DE) (BSE)</th>
<th>Error Variance (DEUKF) (RIL)</th>
<th>Error Variance (DE) (RIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30 days</td>
<td>0.00028614</td>
<td>0.0004182</td>
<td>0.00010225</td>
<td>0.000402</td>
</tr>
<tr>
<td>Jan. 08</td>
<td>0.0015</td>
<td>0.0024</td>
<td>0.0021</td>
<td>0.0034</td>
</tr>
<tr>
<td>March, 08</td>
<td>0.00084366</td>
<td>0.0013</td>
<td>0.00042020</td>
<td>0.0056</td>
</tr>
<tr>
<td>May, 08</td>
<td>0.00029185</td>
<td>0.000396</td>
<td>0.00038887</td>
<td>0.00062</td>
</tr>
<tr>
<td>July, 08</td>
<td>0.00052033</td>
<td>0.000807</td>
<td>0.00047518</td>
<td>0.0071</td>
</tr>
</tbody>
</table>
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 measures are listed in Table-2.6 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 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-2.7. 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-2.7, 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-2.6: Effect of UKF training

<table>
<thead>
<tr>
<th>Stock Market</th>
<th>Window size</th>
<th>1 Day ahead</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSE</td>
<td>25(DE) + 250 (UKF)</td>
<td>2.15</td>
</tr>
<tr>
<td></td>
<td>50(DE) + 250 (UKF)</td>
<td>1.92</td>
</tr>
<tr>
<td></td>
<td>100(DE) + 250 (UKF)</td>
<td>2.46</td>
</tr>
<tr>
<td>RIL</td>
<td>25(DE) + 250 (UKF)</td>
<td>2.83</td>
</tr>
<tr>
<td></td>
<td>50(DE) + 250 (UKF)</td>
<td>2.29</td>
</tr>
<tr>
<td></td>
<td>100(DE) + 250 (UKF)</td>
<td>3.15</td>
</tr>
</tbody>
</table>
Table-2.7: Effect of window size

<table>
<thead>
<tr>
<th>Stock</th>
<th>Window size</th>
<th>1 Day ahead</th>
<th>7 Days ahead</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSE</td>
<td>8</td>
<td>1.92</td>
<td>5.03</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2.34</td>
<td>4.87</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>2.83</td>
<td>4.66</td>
</tr>
<tr>
<td>RIL</td>
<td>8</td>
<td>2.2</td>
<td>5.84</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2.74</td>
<td>5.32</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>2.91</td>
<td>5.14</td>
</tr>
</tbody>
</table>

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.2.2 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.2.5 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-2.8. 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 centre 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 centre 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.
(a) BSE Stock Prices (LLWNN)

(b) BSE Stock Prices (LLRBFNN)
To illustrate the superiority of DEUKF over DE learning algorithms, 3-days ahead and seven days ahead prediction results of the BSE stock index are shown in Fig.2.6. Also from the results shown in Table-2.8, 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.2.1 (a).
(a) 3 days ahead prediction results of BSE in DE

(b) 3 days ahead prediction results of BSE in DEUKF
(c) 7 days ahead prediction results of BSE in DE

(d) 7 days ahead prediction results of BSE in DEUKF

Fig. 2.6 3 days and 7 days Prediction results of BSE stock using DE and DEUKF
Table-2.8: MAPE errors during testing of BSE Data set using different ANN models using DE

<table>
<thead>
<tr>
<th>Days ahead</th>
<th>MLP</th>
<th>RBFNN</th>
<th>ANFIS</th>
<th>FLANN</th>
<th>DNN (DEUKF)</th>
<th>LLRBFNN</th>
<th>LLWNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.9825</td>
<td>3.2276</td>
<td>3.0741</td>
<td>2.3441</td>
<td>1.9188</td>
<td>2.8752</td>
<td>2.924</td>
</tr>
<tr>
<td>5</td>
<td>5.4390</td>
<td>5.4759</td>
<td>5.2157</td>
<td>4.8933</td>
<td>4.2082</td>
<td>5.3918</td>
<td>5.281</td>
</tr>
</tbody>
</table>

2.7.1 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 [76] that by using different Sigma point calculation strategy, its computational complexity can be reduced significantly. In this chapter the 250 steps of UKF with 10 variables take nearly 11.6 secs. in a dual core processor, while combining it with DE in an alternate fashion, it takes nearly 18 secs for stock indices prediction. In [76], the computational complexity of UKF alone has been given, which is similar to the one in this chapter. Implementation of DE alone takes more than 15 secs. for 50 generations. The computation will increase substantially with DE alone for multilayered dynamic neural networks.

2.8. 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 chapter for extracting relevant information from the stock time series to predict the stock market trend. From these technical indicators, a rule based system is developed for classifying the trend into three categories such as uptrend, downtrend, and no trend depending [15] on the direction of movement of the stock price on a day ahead basis. Price and trading volume [26-29] are the two important variables to study the market
action. The following technical indicators are considered for the prediction of stock price trends:

**Simple Moving Average:**

Simple moving average (SMA) is a simple statistical mean and the formulas given in eqns.(2.88) to (2.90) are used to identify trend such as long term, medium term or short term by smoothing out price and volume fluctuations.

\[
SMA(i) = \frac{1}{n} \sum_{i=1}^{n} CP(i)
\]  

(2.88)

SMA (5-day SMA), 6SMA (6-day SMA), 10SMA (10-day SMA), 20SMA (20-day SMA), 25SMA (25-day SMA), 65SMA (65-day SMA) are the statistical indicators that can be used for price trend prediction. Out of these two indicators are chosen here as

\[
SMA_{25} = \frac{CP(i) + CP(i-1) + CP(i-2) + \ldots + CP(i-25)}{25}
\]  

(2.89)

\[
SMA_{65} = \frac{CP(i) + CP(i-1) + CP(i-2) + \ldots + CP(i-65)}{65}
\]  

(2.90)

**Bias:** The difference between the closing value and moving average line (exp: 5 Bias, 10 Bias), is used to analyze the stock market behaviour.

**Exponential Moving Average (EMA):**

It is a type of moving average and it gives more weightage to the recent data. It is very much useful in trend prediction. EMA calculation is given below:

\[
EMA (current) = ( (Price(current) – EMA(prev.) ) \times Multiplier ) + EMA(prev.)
\]  

(2.91)

where Multiplier= \( \frac{2}{\text{no. of days to be considered} + 1} \)  

(2.92)

**Rule base Formulation:**

For stock market trend analysis the following rule is used:

**Rules using MA indicator:**

The stock market is classified to move upward (Uptrend) or downward (Down trend) using the rule given below. However, if neither of the rules are satisfied then the stock market is said to have No trend.
1. If \( C_t > SMA_{25} \), and \( MA_{25} > SMA_{65} \),
and \( \Delta SMA_{25} > 0 \), for at least 5 days
and \( \Delta SMA_{65} > 0 \) for at least 1 day
Then Uptrend

\[ (2.93) \]

2. If \( C_t < SMA_{25} \), and \( MA_{25} < SMA_{65} \),
and \( \Delta SMA_{25} < 0 \), for at least 5 days
and \( \Delta SMA_{65} < 0 \) for at least 1 day
Then Downtrend

\[ (2.94) \]

where \( C_t \) is the closing price of the \( t^{\text{th}} \) trading day; \( SMA_{25} = 25 \text{ day moving average} \);
\( SMA_{65} = 65 \text{ day moving average}, \Delta \) represents the change.

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 2.9 and 2.10 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 (2.68) and (2.59). On the other hand, the Oracle stock closing price data for the same period showed an uptrend.
Table 2.9  Technical Indicators (MA_25, MA_65, K-Index, D-Index, R-Index) for IBM Data

<table>
<thead>
<tr>
<th>Stock market</th>
<th>Technical Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MA_25 (15 data) (750-765)</td>
</tr>
<tr>
<td>IBM</td>
<td>0.5528</td>
</tr>
<tr>
<td></td>
<td>0.5496</td>
</tr>
<tr>
<td></td>
<td>0.5479</td>
</tr>
<tr>
<td></td>
<td>0.5468</td>
</tr>
<tr>
<td></td>
<td>0.5433</td>
</tr>
<tr>
<td></td>
<td>0.5402</td>
</tr>
<tr>
<td></td>
<td>0.5385</td>
</tr>
<tr>
<td></td>
<td>0.5374</td>
</tr>
<tr>
<td></td>
<td>0.5402</td>
</tr>
<tr>
<td></td>
<td>0.5447</td>
</tr>
<tr>
<td></td>
<td>0.5521</td>
</tr>
<tr>
<td></td>
<td>0.5575</td>
</tr>
<tr>
<td></td>
<td>0.5618</td>
</tr>
<tr>
<td></td>
<td>0.5669</td>
</tr>
<tr>
<td></td>
<td>0.5703</td>
</tr>
<tr>
<td></td>
<td>0.5738</td>
</tr>
</tbody>
</table>
Table 2.10  Technical Indicators (MA_25, MA_65, K-Index, D-Index, R-Index) for Oracle Data

<table>
<thead>
<tr>
<th>Stock market</th>
<th>Technical Indicators</th>
<th>Predicted Prices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle</td>
<td>0.8412</td>
<td>0.7328</td>
</tr>
<tr>
<td></td>
<td>0.8339</td>
<td>0.7287</td>
</tr>
<tr>
<td></td>
<td>0.8261</td>
<td>0.7254</td>
</tr>
<tr>
<td></td>
<td>0.8167</td>
<td>0.7222</td>
</tr>
<tr>
<td></td>
<td>0.8043</td>
<td>0.7192</td>
</tr>
<tr>
<td></td>
<td>0.7946</td>
<td>0.7164</td>
</tr>
<tr>
<td></td>
<td>0.7837</td>
<td>0.7135</td>
</tr>
<tr>
<td></td>
<td>0.7706</td>
<td>0.7094</td>
</tr>
<tr>
<td></td>
<td>0.7620</td>
<td>0.7071</td>
</tr>
<tr>
<td></td>
<td>0.7525</td>
<td>0.7047</td>
</tr>
<tr>
<td></td>
<td>0.7475</td>
<td>0.7031</td>
</tr>
<tr>
<td></td>
<td>0.7391</td>
<td>0.7008</td>
</tr>
<tr>
<td></td>
<td>0.7305</td>
<td>0.6989</td>
</tr>
<tr>
<td></td>
<td>0.7236</td>
<td>0.6985</td>
</tr>
<tr>
<td></td>
<td>0.7148</td>
<td>0.6974</td>
</tr>
<tr>
<td></td>
<td>0.7067</td>
<td>0.6980</td>
</tr>
</tbody>
</table>

2.9 Trading Rule Generation

After obtaining the direction of movement of stock indices the trading rules can be formulated in a straightforward manner considering a specified time horizon ranging from one week to several weeks.

1. If the next day trend= Uptrend then the decision is Buy
2. If Buy decision exists then Hold
3. If the next day trend= Downtrend then the decision is Sell
4. If Sell decision exists then Hold
5. If the next day trend = No Trend then the decision is Hold

Apart from obtaining the direction of movement of the stocks, other technical indicators are used to provide Buy or Sell decisions for the stock indices in the following manner:

1. **Stochastic lines (KD):**

   The KD indicator comprises two smooth moving average lines, K line and D line. Both these indicators range between 0 and 100; and this indicator is relied by the investors in buying or selling the stocks. When the KD indicators are $\geq 80$, 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 $\leq 20$, 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. The formulas for %K and %D are given below in eqns. (2.95) and (2.96), respectively as

   The n-day %K of the ith trading day is
   \[
   \% K = 100 \times \frac{\text{CP}(i) - \text{LL}(n)}{\text{HH}(n) - \text{LL}(n)}
   \]  \hspace{1cm} (2.95)

   where CP= is the current closing price
   LL(n)=Lowest low of n period
   HH(n)=Highest high of n period

   \% D = 3 Period Moving average of %K

   \[
   = \frac{\text{MA}(\% K(i) + \% K(i - 1) + \% K(i - 2))}{3}
   \] \hspace{1cm} (2.96)

2. **WMS %R (Williams R indicator (%R)):**

   The William %R is a technical indicator developed by Larry Williams shows how the current price is related to the high and low prices of n period of past values. It is useful in measuring the overbought/oversold levels of the price in a time period. The formula for calculating the WMS%R is given below:
The n-day %R of the ith trading day is calculated as below:

\[
%R = \frac{CP(i) - LL(n)}{HH(n) - LL(n)} \times -100
\]  

(2.97)

where

- HH(n) = the highest high stock price during the latest n trading days,
- LL(n) = the lowest low stock price during the latest n trading days,
- CP(i) = the closing price of the ith trading day.

3. Relative strength index (RSI):

RSI, a momentum indicator, is most useful in determining the overbought and oversold conditions of an asset. The formula for calculating the RSI is:

\[
RSI = 100 - \frac{100}{1 + RS}
\]  

(2.98)

where

\[
RS = \frac{\text{Average of n days' up closes}}{\text{Average of n days' down closes}}
\]

4. Moving average convergence and divergence (MACD):

The MACD shows the relationship between two exponential moving averages of prices. It is calculated as below:

MACD : 12-day EMA – 26-day EMA

(2.99)

5. Signal line:

Signal line is calculated as 9-day EMA of the MACD.

2.9.1 Trading Rules from Technical Indicators

After obtaining the technical indicators buying and selling rules can be obtained using combinations of technical indicators or simply from any one of them.

A. Rules using %K, %D, and WMS%R

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

The WMS%Ri or simply the R-index will be adopted to assist the KD intersection for judging the points of buying/selling.
Rule-3: When K and D indicators intersect with each other at the low level i.e. less than 20 and the smooth value of WMS%R is 100%, it is a buying point.

Rule-4: When K and D indicators intersect with each other at the high level i.e. greater than 80 and the smooth value of WMS%R is 0%, it is a selling point. Otherwise hold.

B. Rules using RSI:

RSI ranges from 0 to 100. Generally, if the RSI rises above overbought level (usually considered as 80), it indicates a selling signal; if the RSI falls below oversold level (usually 20), it indicates a buying signal.

6-RSI and 12-RSI are the generally used RSIs for stock trend detection.

After obtaining the stock price trend, the next step will be to obtain the stock turning points and the technical indicators K, D, and 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.

Description

The NIKKEI 225 dataset from 15th April 2009 to 7th August 2009, are taken for finding the trend and trading signals. The SMA 25 and SMA 65 calculated for the selected dataset, are shown Table 2.11. The last column (F) shows the up and down trends using MA indicators (Eqns. 2.93 and 2.94) where as the changes in the MA 25 and MA 65 are presented in columns D and E, respectively.

Table 2.12 shows the increase and decrease directions of initial ten data from the datasets chosen for study. From Table 2.13, it is clear that, out of 10 samples the time series shows a total no. of up trends are 3 and down trends are 1, therefore in these points the trading decision can be taken. The trading signals are shown in the last column of Table 2.13.
Table 2.11  Trends of NIKKEI 225 using MA indicators

<table>
<thead>
<tr>
<th>Stock market</th>
<th>Closing prices ($C_t$)</th>
<th>MA_25</th>
<th>MA_65</th>
<th>ΔMA_25</th>
<th>ΔMA_65</th>
<th>Uptrend (1) / Down trend (0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NIKKEI 225</td>
<td>0.3066 0.3088 0.3365 0.3396 0.3008 0.3037 0.3255 0.3002 0.3036 0.2613 0.3221 0.3491 0.3491 0.4233 0.4319 0.4353 0.4075 0.4151 0.3703 0.4014 0.3603 0.406 0.4159 0.4012 0.3943 0.4163 0.4097 0.4329 0.4352 0.4482 0.4764</td>
<td>0.2248 0.2371 0.2471 0.2595 0.2693 0.2767 0.2823 0.2887 0.2942 0.2979 0.2981 0.2994 0.302 0.3074 0.3137 0.3182 0.3231 0.328 0.3323 0.3357 0.3411 0.3424 0.3446 0.3455 0.3487 0.3531 0.3568 0.3591 0.3622 0.3665 0.3717 0.3761 0.382</td>
<td>0.169 0.1688 0.1697 0.171 0.1736 0.1749 0.1762 0.1784 0.1807 0.1825 0.1846 0.189 0.1915 0.1935 0.1963 0.1993 0.2037 0.2076 0.2118 0.2152 0.219 0.2222 0.2266 0.2301 0.2344 0.2392 0.244 0.2487 0.2541 0.2595 0.2655 0.2711 0.2769</td>
<td>0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248 0.2248</td>
<td>0.169 0.0123 0.0124 0.0098 0.0073 0.0057 0.0064 0.0055 0.0037 0.0025 0.0001 0.0014 0.0025 0.0054 0.0045 0.0063 0.0045 0.0008 0.0043 0.0034 0.0054 0.0013 0.0013 0.0025 0.0003 0.0013 0.0013 0.0022 0.0023 0.0019 0.0021 0.0023 0.0021 0.0024 0.0001 0.0003 0.0003 0.0003 0.0003 0.0003 0.0003 0.0003 0.0003</td>
<td>1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>
### Table 2.12 Increase / decrease directions of NIKKEI 225

<table>
<thead>
<tr>
<th>Time series</th>
<th>Closing Prices (Actual)</th>
<th>Closing prices (scaled) ($C_t$)</th>
<th>Increase (1)/ Decrease (0)</th>
<th>Total Increase</th>
<th>Total Decrease</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8742.96</td>
<td>0.3066</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>8755.26</td>
<td>0.3088</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8907.58</td>
<td>0.3365</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8924.75</td>
<td>0.3396</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>8711.33</td>
<td>0.3008</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>8727.30</td>
<td>0.3037</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8847.01</td>
<td>0.3255</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8707.99</td>
<td>0.3002</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>8726.34</td>
<td>0.3036</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>8840.77</td>
<td>0.2613</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 2.13 Trading session of NIKKEI 225

<table>
<thead>
<tr>
<th>Time series</th>
<th>Closing Prices (Actual)</th>
<th>Closing prices (scaled) ($C_t$)</th>
<th>Increase (1)/ Decrease (0)</th>
<th>Trends</th>
<th>Trading Points</th>
<th>Trading signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8742.96</td>
<td>0.3066</td>
<td>0</td>
<td>Up</td>
<td>Buy</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>8755.26</td>
<td>0.3088</td>
<td>1</td>
<td>Up</td>
<td>Buy</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>8907.58</td>
<td>0.3365</td>
<td>1</td>
<td>Up</td>
<td>Buy</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>8924.75</td>
<td>0.3396</td>
<td>1</td>
<td>Up</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>8711.33</td>
<td>0.3008</td>
<td>0</td>
<td>Down</td>
<td>Sell</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>8727.30</td>
<td>0.3037</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8847.01</td>
<td>0.3255</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8707.99</td>
<td>0.3002</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>8726.34</td>
<td>0.3036</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>8840.77</td>
<td>0.2613</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.10 Conclusion

In this chapter, 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), Oracle Stock market, Reliance Industries Limited (RIL) and IBM stock market (IBM) 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 performance metrics such as the lowest MAPE of around 2% in comparison to 3.42% with UKF, 2.55% with DE and 4.13% with RTRL algorithms and also provides the lowest AMAPE around 2% in comparison to 3.32% with UKF, 2.54% with DE and 4.08% with RTRL algorithms. It is clear that the other three learning methods also perform more or less accurately in predating stock market trends. The chapter 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. Apart from obtaining the prediction of stock market indices over a certain time horizon, its direction of movement commonly known as trend (Uptrend or Downtrend) has been also analysed in this chapter. Further, various strategies have been indicated for trading the stocks (Buy or Sell or Hold) have been outlined with some relevant technical indicators.