Forecasting financial time series using a low complexity recurrent neural network and evolutionary learning approach

Ajit Kumar Rout a, P.K. Dash b,*, Rajashree Dash b, Ranjeeta Bisoib

a G.M.R. Institute of Technology, Rajam, Andhra Pradesh, India
b S.O.A. University, Bhubaneswar, India

Received 14 March 2015; revised 24 May 2015; accepted 3 June 2015

KEYWORDS
Low complexity FLANN models; Recurrent computationally efficient FLANN; Differential Evolution; Hybrid Moderate Random Search PSO

Abstract The paper presents a low complexity recurrent Functional Link Artificial Neural Network for predicting the financial time series data like the stock market indices over a time frame varying from 1 day ahead to 1 month ahead. Although different types of basis functions have been used for low complexity neural networks earlier for stock market prediction, a comparative study is needed to choose the optimal combinations of these for a reasonably accurate forecast. Further several evolutionary learning methods like the Particle Swarm Optimization (PSO) and modified version of its new variant (HMRPSO), and the Differential Evolution (DE) are adopted here to find the optimal weights for the recurrent computationally efficient functional link neural network (RCEFLANN) using a combination of linear and hyperbolic tangent basis functions. The performance of the recurrent computationally efficient FLANN model is compared with that of low complexity neural networks using the Trigonometric, Chebyshev, Laguerre, Legendre, and tangent hyperbolic basis functions in predicting stock prices of Bombay Stock Exchange data and Standard & Poor’s 500 data sets using different evolutionary methods and has been presented in this paper and the results clearly reveal that the recurrent FLANN model trained with the DE outperforms all other FLANN models similarly trained.

1. Introduction

Financial time series data are more complicated than other statistical data due to the long term trends, cyclical variations, seasonal variations and irregular movements. Predicting such highly fluctuating and irregular data is usually subject to large errors. So developing more realistic models for predicting financial time series data to extract meaningful statistics from it, more effectively and accurately is a great interest of research...
in financial data mining. The traditional statistical models used for financial forecasting were simple, but suffered from several shortcomings due to the nonlinearity of data. Hence researchers have developed more efficient and accurate soft computing methods like Artificial Neural Network (ANN); Fuzzy Information Systems (FIS), Support Vector Machine (SVM), Rough Set theory etc. for financial forecasting. Various ANN based methods like Multi Layer Perceptron (MLP) Network, Radial Basis Function Neural Network (RBFNN), Wavelet Neural Network (WNN), Local Linear Wavelet Neural Network (LLWNN), Recurrent Neural Network (RNN) and Functional Link Artificial Neural Network (FLANN) are extensively used for stock market prediction due to their inherent capabilities to identify complex nonlinear relationship present in the time series data based on historical data and to approximate any nonlinear function to a high degree of accuracy. The use of ANN to predict the behavior and tendencies of stocks has demonstrated itself to be a viable alternative to existing conventional techniques (Andrade de Oliveira and Nobre, 2011; Naeni et al., 2010; Song et al., 2007; Lee and Chen, 2007; Ma et al., 2010).

A system of time series data analysis has been proposed in Kozarzewski (2010) for predicting the future values, based on wavelets preprocessing and neural networks clustering that has been tested as a tool for supporting stock market investment decisions and shows good prediction accuracy of the method. MLP neural networks are mostly used by the researchers for its inherent capabilities to approximate any non-linear function to a high degree of accuracy (Lin and Feng, 2010; Tahersima et al., 2011). But these models suffer from slow convergence, local minimum, over fitting, have high computational cost and need large number of iterations for its training due to the availability of hidden layer. To overcome these limitations, a different kind of ANN i.e. Functional Link ANN (Proposed by Pao (1989)) having a single layer architecture with no hidden layers has been developed. The mathematical expression and computational calculation of a FLANN structure is same as MLP. But it possesses a higher rate of convergence and lesser computational load than those of a MLP structure (Majhi et al., 2005; Chakravarty and Dash, 2009). A wide variety of FLANNS with functional expansion using orthogonal trigonometric functions (Dehuri et al., 2012; Mili and Hamdi, 2012; Patra et al., 2009), using Chebyshev polynomial (Mishra et al., 2009; Jiang et al., 2012; Li et al., 2012), using Laguerre polynomial (Chandra et al., 2009) and using Legendre orthogonal polynomial (Nanda et al., 2011; George and Panda, 2012; Rodriguez, 2009; Das and Satapathy, 2011; Patra and Bornand, 2010) has been discussed in the literature. The well known Back Propagation algorithm is commonly used to update the weights of FLANN. In Yogi et al. (2010), a novel method using PSO for training trigonometric FLANN has been discussed for equalization of digital communication channels.

In this paper, the detailed architecture and mathematical modeling of various polynomial and trigonometric FLANNS have been described along with a new computationally efficient and robust FLANN, and its recurrent version. It is well known that the recurrent neural networks (RNNs) usually provide a smaller architecture than most of the nonrecursive neural networks like MLP, RBFNN, etc. Also their feedback properties make them dynamic and more efficient to model nonlinear systems accurately which are imperative for nonlinear prediction and time series forecasting. Many of the Autoregressive Moving Average (ARMA) processes have been accurately modeled by RNNs for nonlinear dynamic system identification. One of the familiar approaches of training the RNNs is the Real-Time Recurrent Learning (RTRL) (Amprolucci et al., 1999), which has problems of stability and slow convergence. In nonlinear time series forecasting problems it gets trapped in local minima and cannot guarantee to find global minima. On the other hand, evolutionary learning techniques such as Differential Evolution, particle swarm optimization, genetic algorithm, bacteria foraging, etc. have been applied to time series forecasting successively. DE is found to be efficient among them and outperforms other evolutionary algorithms since it is simpler to apply and involves less computation with less function parameters to be optimized as compared to other algorithms. DE is chosen because it is a simple but powerful global optimization method and converges faster than PSO. A comparative study between Differential Evolution (DE) and Particle Swarm Optimization (PSO) in the training and testing of feed-forward neural network for the prediction of daily stock market prices has shown that DE provides a faster convergence speed and better accuracy than PSO algorithm in the prediction of fluctuated time series (Abdual-Salam et al., 2010). Differential Evolution based FLANN has also shown its superiority over Back Propagation based Trigonometric FLANN in Indian Stock Market prediction (Hatem and Mustafa, 2012). The convergence speed is also faster to find a best global solution by escaping from local minima even for multiple optimal solutions.

Thus, in this paper various evolutionary learning methods like PSO, HMRPSO, DE for improving the performance of different types of FLANN models have been discussed. Comparing the performance of various FLANN models for predicting stock prices of Bombay Stock Exchange data and Standard & Poor’s 500 data set, it has been tried to find out the best FLANN among them. The rest of the paper is organized as follows. In Sections 2 and 3, the detailed architecture of various FLANNS and various evolutionary learning algorithms for training has been described. The simulation study for demonstrating the prediction performance of different FLANNS has been carried out in Section 4. This section also provides a comparative result of training and testing of different FLANNS using PSO, HMRPSO, and DE (Mohapatra et al., 2012; Qin et al., 2008; Wang et al., 2011) based learning for predicting financial time series data. Finally conclusions are drawn in Section 5.

2. Architecture of low complexity neural network models

The FLANN originally proposed by Pao in 1992 is a single layer single neuron architecture, having two components: Functional expansion component and Learning component. The functional block helps to introduce nonlinearity by expanding the input space to a higher dimensional space through a basis function without using any hidden layers like MLP structure. The mathematical expression and computational calculation of a FLANN structure is same as MLP. But it possesses a higher rate of convergence and lesser computational cost than those of a MLP structure. A wider application of FLANN models for solving non linear problems like channel equalization, non linear dynamic system identification, electric load forecasting, prediction of earthquake, and financial forecasting has demonstrated its viability, robustness and ease of computation. The functional expansion block comprises either a trigonometric block or a polynomial
block. Further, the polynomial block can be expressed in terms of Chebyshev, Laguerre, or Legendre basis functions. Trigonometric FLANN (TRFLANN) is a single layer neural network in which the original input in a lower dimensional space is expanded to a higher dimensional space using orthogonal trigonometric functions (Chakravarty and Dash, 2009; Dehuri et al., 2012; Mili and Hamdi, 2012). With the order \( m \) any \( n \) dimensional input pattern \( X = [x_1, x_2, \ldots, x_n]^T \) expanded to a \( p \) dimensional pattern \( TX = [1, T\text{F}_1(x_1), T\text{F}_2(x_1) \ldots T\text{F}_m(x_1), T\text{F}_1(x_2), T\text{F}_2(x_2) \ldots T\text{F}_m(x_2), \ldots T\text{F}_1(x_n), T\text{F}_2(x_n) \ldots T\text{F}_m(x_n)]^T \) where \( p = m \times n + 1 \). Each \( x_i \) in input pattern is expanded using trigonometric functions with order \( m \) as \( \{1\sin(\pi x_1)\cos(\pi x_1)\sin(2\pi x_1)\cos(2\pi x_1) \ldots \sin(m\pi x_1)\cos(m\pi x_1)\} \).

Trigonometric function:

\[
T\text{F}_0(x) = 1, \quad T\text{F}_1 = \sin \pi x, \quad T\text{F}_2 = \sin \pi x, \quad T\text{F}_3 = \cos 3\pi x, \quad T\text{F}_4 = \sin 3\pi x
\] (1)

Chebyshev polynomial:

The recursive formula to generate higher order Chebyshev polynomials is given by

\[
C_{p+1}(x) = 2xC_p(x) - C_{p-1}(x), \quad C_0(x) = 1, \quad C_1(x) = x,
\]

\[
C_2(x) = 2x^2 - 1, \quad C_3(x) = 4x^3 - 3x,
\]

\[
C_4(x) = 8x^4 - 8x^2 + 1, \quad C_5(x) = 16x^5 - 20x^3 + 5x,
\]

\[
C_6(x) = 32x^6 - 48x^4 + 18x^2 - 1.
\] (2)

Laguerre polynomials:

The recursive formula to generate higher order Laguerre polynomials is given by

\[
L_{p+1}(x) = \frac{1}{p+1}[(2p+1)L_p(x) - pL_{p-1}(x)]
\]

\[
L_0(x) = 1, \quad L_1(x) = 1 - x, \quad L_2 = 0.5x^2 - 2x + 1,
\]

\[
L_3 = -x^3/6 + 3x^2/2 - 3x + 1,
\]

\[
L_4(x) = -x^4/24 + 2x^3/3 + 3x^2 - 4x + 1,
\]

\[
L_5(x) = -x^5/120 + 5x^4/24 - 5x^3/3 + 5x^2 - 5x + 1.
\] (3)

Legendre polynomials:

The Legendre polynomials are denoted by \( L_p(X) \), where \( p \) is the order and \(-1 < x < 1\) is the argument of the polynomial. It constitutes a set of orthogonal polynomials as solutions to the differential equation:

\[
\frac{d}{dx} \left[ (1-x^2) \frac{dy}{dx} \right] + p(p+1)y = 0
\]

The zeroth and the first order Legendre polynomials are respectively given by, \( L_0(x) = 1 \) and \( L_1(x) = x \). The higher order polynomials are

\[
L_2(x) = 3x^2/2 - 1/2, \quad L_3(x) = 5x^3/2 - 3x/2,
\]

\[
L_4(x) = 35x^4/8 - 15x^2/4 + 3/8
\]

\[
L_5(x) = 63x^5/8 - 35x^3/4 + 15x/8,
\]

\[
L_6(x) = 231x^6/16 - 315x^4/16 + 105x^2/16 - 5/16
\] (4)

The predicted sample \( x_{(i+k)} \) can be represented as a weighted sum of nonlinear polynomial arrays, \( P_i(x_j) \). The inherent nonlinearities in the polynomials attempt to accommodate the nonlinear causal relation of the future sample with the samples prior to it.

Using trigonometric expansion blocks and a sample index \( k \), the following relations are obtained:

\[
\begin{align*}
\delta_{(1)}(k) & \quad T\text{F}_1(x_1(k)) \\
\delta_{(2,1)}(k) & \quad T\text{F}_2(x_1(k)) \\
& \vdots \\
\delta_{(m-1,1)}(k) & \quad T\text{F}_{m-1}(x_1(k)) \\
\delta_{(m,1)}(k) & \quad T\text{F}_m(x_1(k))
\end{align*}
\] **(5)**

\[
\begin{align*}
\delta_{(1,0)}(k) & \quad T\text{F}_1(x_0(k)) \\
\delta_{(2,0)}(k) & \quad T\text{F}_2(x_0(k)) \\
& \vdots \\
\delta_{(m-1,0)}(k) & \quad T\text{F}_{m-1}(x_0(k)) \\
\delta_{(m,0)}(k) & \quad T\text{F}_m(x_0(k))
\end{align*}
\]

and this FLANN is named as TRFLANN.

Using Chebyshev polynomials, the CHFLANN output is obtained as

\[
\begin{align*}
\beta_{(1)}(k) & \quad C_1(x_1(k)) \\
\beta_{(2,1)}(k) & \quad C_2(x_1(k)) \\
& \vdots \\
\beta_{(m-1,1)}(k) & \quad C_{m-1}(x_1(k)) \\
\beta_{(m,1)}(k) & \quad C_m(x_1(k))
\end{align*}
\] **(6)**

Using Laguerre polynomials, the LAGFLANN output is found as

\[
\begin{align*}
\beta_{(1)}(k) & \quad L\text{a}_1(x_1(k)) \\
\beta_{(2,1)}(k) & \quad L\text{a}_2(x_1(k)) \\
& \vdots \\
\beta_{(m-1,1)}(k) & \quad L\text{a}_{m-1}(x_1(k)) \\
\beta_{(m,1)}(k) & \quad L\text{a}_m(x_1(k))
\end{align*}
\] **(7)**
Using Legendre polynomials, Eq. (8) gives the LEG-FLANN output

\[
\begin{align*}
\gamma(k)_{\text{Legendre}} &= \gamma(k)_0 + \\
&+ \begin{bmatrix}
\gamma_{(1,1)}(k) \\
\gamma_{(2,1)}(k) \\
\vdots \\
\gamma_{(m,1)}(k)
\end{bmatrix}^T \\
&\times \\
\begin{bmatrix}
L_{c1}(x_1(k)) \\
L_{c2}(x_2(k)) \\
\vdots \\
L_{cm}(x_m(k))
\end{bmatrix} + \ldots
\end{align*}
\]

Finally, the output from each of the FLANN model is passed through an activation block to give the output as

\[
y(k) = \rho S(u(k))_{\text{Trigonometric}}
\]

or

\[
y(k) = \rho S(u(k))_{\text{Chebyshev}},
\]

or

\[
y(k) = \rho S(u(k))_{\text{Laguerre}},
\]

or

\[
y(k) = \rho S(u(k))_{\text{Legendre}}
\]

where \( \rho \) controls the output magnitude and \( S \) is a nonlinear function given by

\[
S(u(k)) = \frac{2}{1 + e^{-2u(k)}} - 1
\]

To obtain the optimal \( \delta, \alpha, \beta \) and \( \gamma \) values, an error minimization algorithm can be used.

### 2.1. Computationally efficient FLANN (CEFLANN)

Computationally efficient FLANN is a single layer ANN that uses trigonometric basis functions for functional expansion. Unlike earlier FLANNS, where each input in the input pattern is expanded through a set of nonlinear functions, here all the inputs of the input pattern passes through a few set of nonlinear functions to produce the expanded input pattern; the new FLANN comprises only a few functional blocks of nonlinear functions for the inputs and thereby result in a high-dimensional input space for the neural network. This new architecture of FLANN has much less computational requirement and possesses high convergence speed. Fig. 1 depicts the single layer computationally efficient FLANN architecture.

In this architecture a cascaded FIR element and a functional expansion block are used for the neural network. The output of this network is obtained as

\[
y(k)^{\text{off}} = \rho S(u(k))\{W_1(k)X(k) + W_2(k)FE(k)\},
\]

and

\[
X(k) = [x_1(k), x_2(k), \ldots, x_n(k)]^T
\]

\[
FE(k) = [FE_1(k), FE_2(k), FE_3(k), \ldots, FE_p(k)]^T,
\]

and

\[
FE_i = \tanh \left( a_0 + \sum_{j=1}^{n} a_j x_j \right), \; i = 1, 2, \ldots, p
\]

where \( W_1 \) and \( W_2 \) are weight vectors for the linear part and functional expansion part, and \( p \) is the total number of expansions with \( n \) number of inputs; \( S \) is the derivative of the activation function \( S \).

### 2.2. Adaptation of weights of the FLANN models

The weights and associated parameters of the four FLANN models and the CEFLANN model are updated at the end of each experiment by computing the error between the desired output and the estimated output. The error at the \( k \)th time step of the \( L \)th experiment is expressed as \( e(k) = y_{\text{des}}(k) - y(k) \), and the \( y_{\text{des}} \) is the desired FLANN output. The cost function is

\[
E(k) = \frac{1}{2} e^2(k)
\]

Using gradient descent algorithm, the weights \( W_{Ti}, W_{Che}, W_{Lag}, W_{Leg} \) of Trigonometric, Chebyshev, Laguerre, and Legendre FLANN model, respectively are updated as

![Figure 1](https://example.com)  
(a) Computationally Efficient FLANN

![Figure 2](https://example.com)  
(b) Computationally Efficient Recurrent FLANN

Please cite this article in press as: Rout, A.K. et al., Forecasting financial time series using a low complexity recurrent neural network and evolutionary learning approach. Journal of King Saud University – Computer and Information Sciences (2015), http://dx.doi.org/10.1016/j.jksuci.2015.06.002
$W^{fl}(k) = W^{fl}(k-1) + \eta p S(u(k))X^{fl}(k)$
$$+ \delta(W^{fl}(k-1) - W^{fl}(k-2))$$
where $\eta$ and $\delta$ are the learning rate and momentum terms, and $W^{fl}$ is the weight vector for the corresponding FLANN model; and $X^{fl}$ stands for the functional expansion terms of the input. With trigonometric expansions $X^{fl}$ becomes
$$X^{fl} = [1\phi_1 \phi_2 \phi_3 \ldots \phi_{m-2} \phi_{m-1} \phi_{m}]^T$$
$$= [1x_1 \cos \pi x_1 \ldots x_n \cos \pi x_n \sin \pi x_n]^T$$
with total number of terms being equal to $mn + 1$. For the first four FLANN models, the weight vector $W^{fl}$ is given by
$$W_{FL}^{RT} = [\phi_{0}, \delta_{(1,1)}, \ldots, \delta_{(m,n)}]^T, W_{FL}^{uCEF} = [\phi_{0}, x_{(1,1)}, \ldots, x_{(m,n)}]^T, W_{FL}^{exp} = [\gamma_{(0)}, \gamma_{(1,1)}, \ldots, \gamma_{(m,n)}]^T$$
(16)

For the CEFANN model
$$W_1(k) = W_1(k-1) - \rho S(u(k))e(k)X^T$$
$$W_2(k) = W_2(k-1) - \rho S(u(k))e(k)FET$$
(17)
and the coefficients of the $i$th functional block are obtained as
$$a_0(k) = a_0(k-1) + \rho W_2, S(u(k)) \sec^2 h \left( a_0 + \sum_{j=1}^{n} a_0 x_j \right)$$
(18)
$$a_0(k) = a_0(k-1) + \rho W_2, S(u(k)) \sec^2 h \left( a_0 + \sum_{j=1}^{n} a_0 x_j \right) x_j$$
(19)

2.3. Computationally efficient recurrent FLANN (RCEFLANN)

In conventional FLANN models since no output lagging terms are used there is no correlation between the training samples. However, when lagged output samples are used as inputs, a correlation exists between the training samples and hence an incremental learning procedure is required for the adjustment of weights. A recurrent version of this FLANN is shown in Fig. 1, where one step delayed output samples are fed to the input to provide a more accurate forecasting scenario. As shown in Fig. 1 the input vector contains a total number $m + n + p$ inputs comprising $n$ inputs from the delayed outputs, $m$ inputs from the stock closing price indices, $p$ inputs from the functional expansion. Thus the input vector is obtained as
$$V(k) = [R^T(k), X^T(k), FET(k)]$$
(20)
which is written in an expanded form as
$$V(k) = [y_1(k-1), y_2(k-2), \ldots, y_1(k-n), x_1, x_2, \ldots, x_m, \phi_1, \phi_2, \ldots, \phi_p]$$
(21)
and for a low complexity expansion $\phi_i$ takes the form
$$\phi_i = \tanh(w_{0i} + w_{1i} x_1 + w_{2i} x_2 + \ldots + w_{ni}), i = 1, 2, \ldots, p$$
Thus the output is obtained as
$$y(k) = y S(u^{ref}(k))$$
$$= y S(W_1(k)R(k)) + W_2(k)X(k) + W_3(k), FE(k)$$
(22)
where
$$W_1 = [a_1, a_2, \ldots, a_n]^T, W_2 = [b_1, b_2, \ldots, b_m]^T, W_3 = [c_1, c_2, \ldots, c_p]^T$$
(23)

The most common gradient based algorithms used for online training of recurrent neural networks are BP algorithms and real-time recurrent learning (RTRL) (Amplouci et al., 1999).

The RTRL algorithm is shown in the following steps:

Using the same cost function for the recurrent FLANN model, for a particular weight $w(k)$, the change of weight is obtained as
$$\Delta w(k) = -\eta \frac{\partial e(k)}{\partial w(k)} = \eta e(k) S(u^{ref}(k)) \frac{\partial y(k)}{\partial w(k)}$$
(24)
where $\eta$ is the learning rate.

The partial derivative of the above Eq. (24) is obtained in a modified form for the RTRL algorithm as
$$\frac{\partial y(k)}{\partial w_{kq}} = y(i-k) + \sum_{j=1}^{n} a_{j} \frac{\partial y(i-k)}{\partial w_{kq}} \cdot k, i = 1, 2, \ldots, n$$
$$\frac{\partial y(k)}{\partial w_{kq}} = x_k + \sum_{j=1}^{n} a_{j} \frac{\partial y(i-k)}{\partial w_{kq}} \cdot k, i = 1, 2, \ldots, m$$
$$\frac{\partial y(k)}{\partial w_{kq}} = \phi_k + \sum_{j=1}^{n} a_{j} \frac{\partial y(i-k)}{\partial w_{kq}} \cdot k, i = 1, 2, \ldots, p$$
$$\frac{\partial y(k)}{\partial w_{kq}} = \phi_k x_k \sec^2 \phi_k + \sum_{j=1}^{n} a_{j} \frac{\partial y(i-k)}{\partial w_{kq}} \cdot k, i = 1, 2, \ldots, p$$
(25)

The weight adjustment formulas are, therefore, obtained as (with $k$ taking values $n, m,$ and $p$ for the weights of the recurrent, input, functional expansion parts, respectively):
$$a_{k}(i) = a_{k}(i - 1) + \eta e(k) \frac{\partial y(i)}{\partial w_{kq}}$$
$$b_{k}(i) = b_{k}(i - 1) + \eta e(k) \frac{\partial y(i)}{\partial w_{kq}}$$
$$c_{k}(i) = c_{k}(i - 1) + \eta e(k) \frac{\partial y(i)}{\partial w_{kq}}$$
$$w_{k}(i) = w_{k}(i - 1) + \eta e(k) \frac{\partial y(i)}{\partial w_{kq}}$$
(26)
where $u_{k}(k) = \sum_{j=1}^{n} u_{j}(k) v_{j}$, and $\delta_{ij}$ is a Kronecker delta equal to 1 when $j = k$ and 0 otherwise.

Further if the learning rate is kept small, the weights do not change rapidly and hence
$$\frac{\partial y(i - 1)}{\partial \alpha_{k}} = \frac{\partial y(i)}{\partial \alpha_{k}}$$
$$\frac{\partial y(i - 1)}{\partial \alpha_{k}}$$
(27)
Denoting the gradient
$$\pi_{k}(i) = \frac{\partial y(i)}{\partial w_{k}}$$
and the gradient values in successive iterations are generated assuming
$$\pi_{k}(0) = 0$$
(28)

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. To avoid the common drawbacks of back propagation algorithm and to increase accuracy, different methods have been proposed that include additional momentum method, self-adaptive learning rate adjustment method, and various search algorithms like GA, PSO, DE algorithm in the training step of the neural network to optimize the parameters of the network like the network weights and the number of hidden units in the hidden layer. In Section 3, the various evolutionary learning methods like Particle Swarm Optimization, Differential Evolution and Hybrid Moderate Random Search PSO have been described for training various FLANN models. Evolutionary algorithms act as excellent global optimizers for real parameter problems.

3. Evolutionary learning methods

The problem described can be formulated as an objective function for error minimization using the equation below.

\[
ERROR = \frac{1}{N} \sum_{k=1}^{N} \left( \frac{d(k) - y(k)}{ \sqrt{N} } \right)^2
\]

where prediction is done \( k \) days ahead and \( y(k) \) is represented as a function of weights and the prior values of the time series:

\[
y^{\text{Thompson}}(k) = f \left( \delta(0), \delta(1), \ldots, \delta(m,n), x(t), \ldots, x(k-n+1) \right)
\]

\[
y^{\text{Chebyshev}}(k) = f \left( x(0), x(1), \ldots, x(m,n), x(t), \ldots, x(k-n+1) \right)
\]

\[
y^{\text{Legendre}}(k) = f \left( \beta(0), \beta(1), \ldots, \beta(m,n), x(t), \ldots, x(k-n+1) \right)
\]

\[
y^{\text{Legendre}}(k) = f \left( \gamma(0), \gamma(1), \ldots, \gamma(m,n), x(t), \ldots, x(k-n+1) \right)
\]

The dimension of the problem to be solved by the evolutionary algorithm would be \((m + n + 1)\).

The objective function for optimization is chosen as

\[
J = \frac{1}{1 + ERROR^2}
\]

3.1. DE based learning

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. 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. Here, a self-adaptive strategy for the control parameters of DE like \( F \) and \( Cr \) is adopted to improve the robustness of the DE algorithm. The pseudo code for DE implementation is given below:

3.1.1. Pseudo code for DE implementation

Input: population size \( Np \), No. of variables to be optimized (Dimension \( D \)), initial scaling and mutation parameters \( F, F_1, F_2 \), and \( Cr, G \) = total number of generations, \( N \) = target vector, Strategy candidate pool: “DE/rand/2”

\[
\begin{align*}
P_K & = \{X^g(0), \ldots, X^g(Np)\} = \{X^g(1), X^g(2), \ldots, X^g(N_p)\}
\end{align*}
\]

1. While stopping criterion is not satisfied do

2. for \( i = 1 \) to \( Np \)

   \[
   F_1 = F_{L_1} + (F_{L_1} - F_{L_2}) \times \text{rand}(0,1)
   \]
   \[
   F_2 = F_{L_2} + (F_{U_2} - F_{L_2}) \times \text{rand}(0,1)
   \]

3. Generate the mutant vector

\[
\begin{align*}
U^g_i & = \{v^g_i(1), v^g_i(2), \ldots, v^g_i(D)\}
V^g_i & = X^g_i + F_1 \times (X^g_i - X^g_{\text{best}}) + F_2 \times (X^g_i - X^g_{\text{best}})
\end{align*}
\]

4. for \( j = 1 \) to \( D \)

   The crossover rate is adapted as

\[
Cr = Cr_L + (Cr_U - Cr_L) \times \text{rand}3
\]

where \( \text{rand}3 \) is a random number between \( [0,1] \).

Generate trial vector \( U^g_i \) using the target vector

\[
\begin{align*}
U^g_i & = \{u^g_i(1), u^g_i(2), \ldots, u^g_i(D)\}
\end{align*}
\]

\[
\begin{align*}
u^g_{i,j} & = \begin{cases} v^g_{i,j} & \text{if (rand) \( \leq Cr \) or } j = j_{\text{rand}} \\ x^g_{i,j} & \text{otherwise} \end{cases}
\end{align*}
\]

5. end while

3.2. Adaptive HMPSO based learning

In the conventional PSO algorithm the particles are initialized randomly and updated afterward according to:
\[ x_k^{i+1} = x_k^i + V_k^{i+1} \]

\[ V_k^{i+1} = wV_k^i + c_1r_1(p_{best}^i - x_k^i) + c_2r_2(g_{best} - x_k^i) \]  

(41)

where \( w, c_1, c_2 \) are inertia, cognitive and social acceleration constants respectively, \( r_1 \) and \( r_2 \) are random numbers within \([0,1]\). \( p_{best} \) is the best solution of the particle achieved so far and indicates the tendency of the individual particles to replicate their corresponding past behaviors that have been successful. \( g_{best} \) is the global best solution so far, which indicates the tendency of the particles to follow the success of others (Lu, 2011; Sanjeevi et al., 2011; Ampolucci et al., 1999). HMRPSO is a new PSO technique using the moderate random search strategy to enhance the ability of particles to explore the solution spaces more effectively and a new mutation strategy to find the global optimum (Gao and Xu, 2011). MRS strategy the position update equation is as follows:

\[ x_k^{i+1} = p_j + r_2\left(m_{best} - x_k^i\right) \]  

(42)

where

\[ m_{best} = \frac{\sum_{j=1}^{np} p_{best,j}}{np} \]  

(43)

\[ p_j = r_1 \times p_{best,j} + (1-r_1)g_{best,j}, \gamma = (r_2 - r_3)/r_4 \]  

(44)

The attractor \( p \) is the main moving directions of particles and \( r_1, r_2, r_3 \) are the uniformly distributed random variables within \([0,1]\), where as \( r_4 \) is a random variable within \([-1,1]\).
The \( p_{best} \) term gives step size for the movement of particles and also makes the contribution of all \( p_{Best} \) to the evolution of particles. The inertia factor \( x \) controls the convergence rate of the method. If the absolute value of the difference between \( x_{ij}^{k+1} \) and \( x_{ij}^{k} \) is smaller than a threshold value \( T_j \), then the current particle can use a mutation operator on dimension \( j \). A suitable value is chosen for threshold, such that neither the algorithm spends more time in mutation operator and less time to conduct MRS in solution space, nor it gets a little chance to do mutation such that the particles will need a relatively long time to escape from the local optima. Again a gradient descent method is used to control the value of \( T_j \) using the following formula:

If \( (F_j = k) \) then \( F_j = 0 \) and \( T_j = T_j/m \)  

Parameter \( m \) controls the decline rate of \( T_j \) and \( k \) controls the mutation frequency of particles. \( F_j \) denotes the number of particles that has used the mutation operator on dimension \( j \). \( F_j \) is updated by the following equation:

\[
F_j(k) = F_j(k - 1) + \sum_{i=1}^{n} b_{ij}^k
\]

where

\[
b_{ij}^k = \begin{cases} 
0 & \text{if } \left( \frac{\text{abs} x_{ij}^k - x_{ij}^{k-1}}{C_0} \right) > T_j \\
1 & \text{if } \left( \frac{\text{abs} x_{ij}^k - x_{ij}^{k-1}}{C_0} \right) \leq T_j
\end{cases}
\]
$F_j$ is initially set to zero and $T_j$ is initially set to the maximum value taken for the domain of particles position. To enhance the global search ability of the HMRPSO, a global mutation operator is applied using the following formula:

$$\text{mutate}(x_j) = (r_5 \times \text{range})$$

(48)

$r_5$ is a random variable within $[-1,1]$ and range is the maximum value set for the domain of particles position. Again the weaker local searching ability caused using the global mutation operator is compensated using a local mutation operator as follows:

$$p_{best,i} = p_{best,i}(1 \pm \text{mut}(\sigma))$$

(49)

where $i$ is the index of the $p_{best}$ that gives best achievement among all other $p_{best}$ positions and $\text{mut}(\sigma)$ returns a value within $(0, 1)$, which is drawn from a standard Gaussian distribution. Finding the variance of the population fitness as

Figure 4  Comparison of actual and predicted S&P500 stock indices with different FLANN models.
\[ \zeta = \sqrt{\frac{1}{N_p} \sum_{i=1}^{N_p} \left( \frac{J_i - J_{avg}}{F_0} \right)^2} \]  

(50)

where \( J_{avg} \) is the average fitness of the population of particles in a given generation, \( F_i \) is the fitness of the \( i \)th particle in the population, \( M \) is the total number of particles.

\[ F_0 = \left\{ \max(|F_i - F_{avg}|), i = 1, 2, 3 \ldots, M \right\} \]  

(51)

In the equation given above \( F_0 \) is the normalizing factor, which is used to limit \( \zeta \). If \( \zeta \) is large, the population will be in a random searching mode, while for small \( \zeta \) or \( \zeta = 0 \), the solution tends toward a premature convergence and will give the local best position of the particles. To circumvent this phenomenon and to obtain gbest solution, the factor \( \alpha \) in (42) controls the convergence rate of the HMRPSO, which is similar with inertia weight used in the PSO. On the one hand, a larger \( \alpha \) value enables particles to have a stronger exploration ability but a less exploitation ability, while on the other hand, smaller \( \alpha \) allows particles a more precise exploitation ability.

The factor \( \alpha \) in Eq. (42) is updated using a fuzzy rule base and fuzzy membership values of the change in standard deviation \( \Delta \zeta \) in the following way:

\[ \mu_{\text{large}}(\zeta) = e^{-|\zeta|^2}, \quad \mu_{\text{small}}(\zeta) = 1 - e^{-|\zeta|^2} \]  

(52)

Figure 5  Comparison of actual and predicted BSE stock indices with different FLANN models.

Please cite this article in press as: Rout, A.K. et al., Forecasting financial time series using a low complexity recurrent neural network and evolutionary learning approach. Journal of King Saud University – Computer and Information Sciences (2015), http://dx.doi.org/10.1016/j.jksuci.2015.06.002
Change in standard deviation

\[ \Delta z(k) = \xi(k) - \xi(k-1) \]  

The fuzzy rule base for arriving at a weight change is expressed as

R1. If \(|\Delta z|\) is Large Then \(\Delta z = \text{rand}_{1}(\Delta z)\)
R2. If \(|\Delta z|\) is Small Then \(\Delta z = \text{rand}_{2}(\Delta z)\)

where the membership functions for Large and Small are given by

\[ \mu_{\text{Large}}(\xi) = |\Delta z|; \quad \mu_{\text{Small}}(\xi) = 1 - |\Delta z| \]  

where \(\text{rand}_{1}(\cdot)\) and \(\text{rand}_{2}(\cdot)\) are random numbers between 0 and 1, and

\[ 0 \leq |\Delta z| \leq 1 \]  

\[ \Delta z = |\Delta z| \cdot \text{rand}_{1}(\Delta z) + \text{rand}_{2}(\Delta z) \cdot \Delta z(1 - |\Delta z|). \]  

Thus the value of the new weight is obtained as

\[ z(k) = z(k-1) + \Delta z \]  

3.2.1. Steps of adaptive HMRPSO based learning

Step 1: Expand the input pattern using functional expansion block
Step 2: Initialize the position of each individual within a given maximum value
Step 3: Find the fitness function value of each particle i.e. the error obtained by applying the weights specified in each particle to the expanded input and applying the nonlinear tanh () function at the output unit
Step 4: Initialize the \(pbest\), \(gbest\), \(pbestt\) positions of the particles
Step 5: Update the particle’s position using the updated value of \(x\) using Eq. (58)
Step 6: For each dimension \(j\)
Step 7: if \(\text{abs}(x^j_i - x^j_i) < T_j\)
Step 8: if \((F_j \leq k/2)\)
Step 9: Apply global mutation operator on \(x_j\) using Eq. (48)
Step 10: else
Step 11: Apply local mutation operator on best \(pbest\) position i.e. \(pbest_j\)
Step 12: end if
Step 13: end if
Step 14: Update the \(F_j\) and \(T_j\) using Eqs. (47) and (46)
Step 15: end for
Step 16: Update the \(pbest\) and \(gbest\) positions by comparing the fitness value of new mutant particles obtained using global mutation with the fitness value of particles of last iteration
Step 17: Update the position of \(pbest\), and accordingly the \(gbest\) position by comparing the fitness value of \(pbest\), obtained using location mutation with its previous fitness value
Step 18: Repeat step 5–17 for each particle in the population until some termination condition is reached, such as predefined number of iterations is reached or the error has satisfied the default precision value
Step 19: Fixed the weight equal to the \(gbest\) position and use the network for testing

4. Experimental result analysis

In this study the sample data set of Bombay Stock Exchange (BSE) and Standard’s &Poor’s 500(S&P500) collected from Google finance comprising the daily closing prices has been taken for experiment to test the performance of different FLANN models trained using evolutionary methods. The total no. of samples for BSE is 1200 from 1st January 2004 to 31st December 2008 and for S&P500 are 653 from 4th January 2010 to 12th December 2012. Both the data sets are divided into training and testing sets. For BSE data set; the training set consists of initial 800 patterns and the following 400 data are set for testing and for S&P500 dataset the training set

![Graph showing comparison of actual and predicted S&P500 and BSE stock indices with RBF.](image)
consists of initial 400 patterns leaving the rest for testing. To improve the performance initially all the inputs are scaled between 0 and 1 using the min–max normalization as follows:

\[ y = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \]  

(59)

where \( y \) = normalized value.

\( x \) = value to be normalized
\( x_{\min} \) = minimum value of the series to be normalized
\( x_{\max} \) = maximum value of the series to be normalized

The Root Mean Square Error (RMSE) and Mean Absolute Percentage Error (MAPE) has been used to compare the performance of different evolutionary FLANNs for predicting

Figure 7  Comparison of actual and predicted S&P500 and BSE stock indices with the recurrent CEFLANN model (3–15 days ahead).
In order to avoid the adverse effect of very small Stock indices over a long period of time, the AMAPE defined in (62) is adopted and compared for all the learning models:

$$AMAPE = \frac{1}{N} \sum_{j=1}^{N} \left| \frac{y_j - \hat{y}_j}{\sum_j y_j} \right| \times 100$$  \hspace{1cm} (62)

Before using a single model or a combination of models to predict the future stock market indices, it is assumed that there is a true model or a combination of models for a given time series. In this paper, the variance of forecast errors is used to measure this uncertainty of the neural network model. The smaller the variance, the less uncertain is the model or more accurate is the forecast results. The variance of the forecast errors over a certain span of time is needed and this parameter is obtained as

$$\sigma^2 = \frac{1}{ND} \sum_{i=1}^{ND} \left\{ \frac{|y_i - \hat{y}_i|}{(1/ND) \sum_j y_j} - \frac{1}{N} \sum_{j=1}^{N} \left( \frac{y_j - \hat{y}_j}{\sum_j y_j} \right) \right\}^2$$ \hspace{1cm} (63)

Further the Technical indicators seem to predict the future price value by looking at past patterns. A brief explanation of each indicator is mentioned here:

**Technical Indicators**

(i) Five days Moving Average(MA) Moving average (Eq. (64)) is used to emphasize the direction of a trend and smooth out price and volume fluctuations that can confuse interpretation.

$$MA(n) = \frac{\sum_{i=n-5}^{n-1} y_i}{n}$$  \hspace{1cm} (64)

MA of the n days and y$_i$ the closed price of the $i^{th}$-day.  

(ii) Five days bias (BIAS) The difference between the closing value and moving average line, which uses the stock price nature of returning back to average price to analyze the stock market in (Eq. (65)).

$$BIAS \text{ for } n \text{ days, } BIAS(n) = \frac{y_i - MA_5(n)}{MA_5(n)}.$$  \hspace{1cm} (65)

(iii) Standard Deviation(SD)

$$SD(\sigma) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \mu)^2} \text{ where } \mu = \frac{1}{n} \sum_{i=1}^{n} y_i$$ \hspace{1cm} (66)

The no. of input layer node for the FLANN models has been set to 8 to express the closing index of 5 days ago and the technical indicators mentioned above and the number of output node to 1 for expressing the closing index of 6th day. The 8 dimensional input patterns have been expanded to a pattern of dimension 33 using order 4 and the corresponding basis functions for TRFLANN, LAGFLANN, CHFLANN, and LEFLANN. For CEFLANN using two nonlinear tanh() functions for expansion, the input pattern has been expanded to pattern of 10 dimensions. The same model also is used in the case of the recurrent CEFLANN model with three lagged output terms fed into the input. Initially the performance of each FLANN model having the same network structure and same data sets trained using PSO, HMRPSO, and DE with different mutation strategy has been observed. The same population having size 20 trained using PSO, HMRPSO, and DE with different mutation strategy has been observed. The same population having size 20 trained using PSO, HMRPSO, and DE with different mutation strategy has been observed. The same population having size 20 trained using PSO, HMRPSO, and DE with different mutation strategy has been observed. The same population having size 20 trained using PSO, HMRPSO, and DE with different mutation strategy has been observed.

Figure 8  Computed variances of the S&P 500 and BSE stock indices.
structure using evolutionary methods. But the individual has width 28 i.e. equal to the total no. of associated parameters and the weights when used for training the CEFLANN network. For PSO the values of c1 and c2 are set at 1.9 and the inertia factor has linearly decreased from 0.45 to 0.35. For DE the mutation scale has been fixed to \(F_{u1} = 0.95, F_{u2} = 0.4\) \(F_{r1} = 0.8, F_{r2} = 0.3\), and the crossover rate to \(Cr_{u1} = 0.9, Cr_{r1} = 0.6\). For HMRPSO the inertia factor has linearly decreased from 0.4 to 0.1. The RMSE error has taken as the fitness function for all the evolutionary learning algorithms.

![Table 1](image1)

<table>
<thead>
<tr>
<th>Type of FLANN</th>
<th>Evolutionary learning method</th>
<th>S&amp;P500</th>
<th>BSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RMSE test</td>
<td>MAPE test</td>
</tr>
<tr>
<td>Recurrent CEFLANN</td>
<td>PSO</td>
<td>0.0411</td>
<td>1.3876</td>
</tr>
<tr>
<td></td>
<td>HMRPSO</td>
<td>0.0332</td>
<td>1.0895</td>
</tr>
<tr>
<td></td>
<td>DE current to best</td>
<td><strong>0.0308</strong></td>
<td><strong>1.0197</strong></td>
</tr>
<tr>
<td>CEFLANN</td>
<td>PSO</td>
<td>0.0441</td>
<td>1.5158</td>
</tr>
<tr>
<td></td>
<td>HMRPSO</td>
<td>0.0343</td>
<td>1.1362</td>
</tr>
<tr>
<td></td>
<td>DE current to best</td>
<td><strong>0.0328</strong></td>
<td><strong>1.0849</strong></td>
</tr>
<tr>
<td>LEFLANN</td>
<td>PSO</td>
<td>0.0479</td>
<td>1.7231</td>
</tr>
<tr>
<td></td>
<td>HMRPSO</td>
<td>0.0438</td>
<td>1.5243</td>
</tr>
<tr>
<td></td>
<td>DE current to best</td>
<td><strong>0.0340</strong></td>
<td><strong>1.1594</strong></td>
</tr>
<tr>
<td>LAGFLANN</td>
<td>PSO</td>
<td>0.0567</td>
<td>2.0437</td>
</tr>
<tr>
<td></td>
<td>HMRPSO</td>
<td>0.0527</td>
<td>1.8557</td>
</tr>
<tr>
<td></td>
<td>DE current to best</td>
<td><strong>0.0506</strong></td>
<td><strong>1.7729</strong></td>
</tr>
<tr>
<td>CHFLANN</td>
<td>PSO</td>
<td>0.0545</td>
<td>1.9178</td>
</tr>
<tr>
<td></td>
<td>HMRPSO</td>
<td>0.04200</td>
<td>1.4166</td>
</tr>
<tr>
<td></td>
<td>DE current to best</td>
<td><strong>0.0371</strong></td>
<td><strong>1.3099</strong></td>
</tr>
<tr>
<td>TRFLANN</td>
<td>PSO</td>
<td>0.1041</td>
<td>3.4783</td>
</tr>
<tr>
<td></td>
<td>HMRPSO</td>
<td>0.0883</td>
<td>3.0075</td>
</tr>
<tr>
<td></td>
<td>DE current to best</td>
<td><strong>0.0738</strong></td>
<td><strong>2.5679</strong></td>
</tr>
<tr>
<td>Recurrent CEFLANN</td>
<td>RTRL</td>
<td>0.0563</td>
<td>2.579</td>
</tr>
<tr>
<td></td>
<td>Gradient Descent</td>
<td>0.0513</td>
<td>2.456</td>
</tr>
</tbody>
</table>

Table 2  Comparison of convergence speed of CEFLANN models with different evolutionary learning methods.

<table>
<thead>
<tr>
<th>Type of FLANN</th>
<th>Evolutionary learning method</th>
<th>S&amp;P500</th>
<th>BSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time elapsed in sec during training</td>
<td>Time elapsed in sec during training</td>
</tr>
<tr>
<td>Recurrent CEFLANN</td>
<td>PSO</td>
<td>66.28</td>
<td>154.26</td>
</tr>
<tr>
<td></td>
<td>HMRPSO</td>
<td>155.85</td>
<td>270.65</td>
</tr>
<tr>
<td></td>
<td>DE current to best</td>
<td><strong>49.94</strong></td>
<td><strong>148.31</strong></td>
</tr>
<tr>
<td>CEFLANN</td>
<td>PSO</td>
<td>40.07</td>
<td>96.57</td>
</tr>
<tr>
<td></td>
<td>HMRPSO</td>
<td>108.17</td>
<td>229.43</td>
</tr>
<tr>
<td></td>
<td>DE current to best</td>
<td><strong>35.09</strong></td>
<td><strong>92.65</strong></td>
</tr>
</tbody>
</table>

Table 3  MAPE errors during testing of S&P500 data set using different evolutionary FLANN models.

<table>
<thead>
<tr>
<th>Days ahead</th>
<th>TRFLANN</th>
<th>LAGFLANN</th>
<th>CHFLANN</th>
<th>LEFLANN</th>
<th>CEFLANN</th>
<th>RCEFLANN</th>
<th>RBFNN</th>
<th>WNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.8264</td>
<td>2.1865</td>
<td>1.7922</td>
<td>1.4662</td>
<td>1.1212</td>
<td>0.9958</td>
<td>1.8565</td>
<td>1.9432</td>
</tr>
<tr>
<td>3</td>
<td>3.4751</td>
<td>2.3116</td>
<td>2.6464</td>
<td>2.3718</td>
<td>1.7768</td>
<td>1.7672</td>
<td>2.9070</td>
<td>3.112</td>
</tr>
<tr>
<td>5</td>
<td>3.0419</td>
<td>2.5902</td>
<td>3.8075</td>
<td>3.1018</td>
<td>2.1164</td>
<td>2.1064</td>
<td>3.1682</td>
<td>3.323</td>
</tr>
<tr>
<td>10</td>
<td>5.3751</td>
<td>3.0774</td>
<td>6.1739</td>
<td>5.2114</td>
<td>2.6965</td>
<td>2.6951</td>
<td>4.1729</td>
<td>4.0867</td>
</tr>
</tbody>
</table>

Bold values are the outputs from the proposed method.
trained with DE current to best technique. Again to compare that the recurrent CEFLANN produce the least errors when evolutionary approach are shown in Fig. 3. From this figure it is seen that each FLANN trained using DE/current to best provides good result than other learning methods. Observed that each FLANN trained using DE/current to best and HMRPSO, DE with different mutation strategies, it has been using various evolutionary learning methods like PSO, HMRPSO, and HMRPSO. Comparing the performance of each FLANN in comparison to other evolutionary methods like the PSO, using DE current to best variant during the training period converges to the lowest value for the two FLANN models. From these figures it is quite clear that the RMSE and MAPE values of BSE and S&P500 data sets during testing with different evolutionary FLANN models.

Table 4 MAPE errors during testing of BSE data set using different evolutionary FLANN models.

<table>
<thead>
<tr>
<th>Days ahead</th>
<th>TRFLANN</th>
<th>LAGFLANN</th>
<th>CHFANN</th>
<th>LEFLANN</th>
<th>CEFLANN</th>
<th>RCEFLANN</th>
<th>RBFNN</th>
<th>WNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.1153</td>
<td>5.9825</td>
<td>4.2276</td>
<td>3.8741</td>
<td>2.2441</td>
<td>1.8180</td>
<td>2.8752</td>
<td>2.924</td>
</tr>
<tr>
<td>5</td>
<td>8.3101</td>
<td>6.9390</td>
<td>5.4759</td>
<td>5.5157</td>
<td>4.8933</td>
<td>4.6082</td>
<td>5.3918</td>
<td>5.281</td>
</tr>
<tr>
<td>7</td>
<td>8.7211</td>
<td>7.2503</td>
<td>7.4708</td>
<td>6.4177</td>
<td>5.2592</td>
<td>5.2263</td>
<td>7.7466</td>
<td>7.612</td>
</tr>
</tbody>
</table>

Table 5 Variances of the predicted S&P 500 stock indices on specific days.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S&amp;P 500 23-07-2010</td>
<td>dvo(400:415) 1.0e–07 0.0399</td>
<td>29-10-2010</td>
<td>dvo(470:485) 1.0e–07 0.2379</td>
<td>09-02-2011</td>
<td>dvo(540:555) 1.0e–07 0.0245</td>
</tr>
<tr>
<td>26-07-2010</td>
<td>0.0430</td>
<td>01-11-2010</td>
<td>0.3299</td>
<td>10-02-2011</td>
<td>0.0293</td>
</tr>
<tr>
<td>27-07-2010</td>
<td>0.1077</td>
<td>02-11-2010</td>
<td>0.2383</td>
<td>11-02-2011</td>
<td>0.0270</td>
</tr>
<tr>
<td>28-07-2010</td>
<td>0.1302</td>
<td>03-11-2010</td>
<td>0.1290</td>
<td>14-02-2011</td>
<td>0.0372</td>
</tr>
<tr>
<td>29-07-2010</td>
<td>0.1674</td>
<td>04-11-2010</td>
<td>0.0229</td>
<td>15-02-2011</td>
<td>0.1269</td>
</tr>
<tr>
<td>30-07-2010</td>
<td>0.1168</td>
<td>05-11-2010</td>
<td>0.0321</td>
<td>16-02-2011</td>
<td>0.2620</td>
</tr>
<tr>
<td>02-08-2010</td>
<td>0.0089</td>
<td>08-11-2010</td>
<td>0.0830</td>
<td>17-02-2011</td>
<td>0.3279</td>
</tr>
<tr>
<td>03-08-2010</td>
<td>0.0113</td>
<td>09-11-2010</td>
<td>0.0904</td>
<td>18-02-2011</td>
<td>0.2261</td>
</tr>
<tr>
<td>04-08-2010</td>
<td>0.0101</td>
<td>10-11-2010</td>
<td>0.2536</td>
<td>22-02-2011</td>
<td>0.0789</td>
</tr>
<tr>
<td>05-08-2010</td>
<td>0.2518</td>
<td>11-11-2010</td>
<td>0.2301</td>
<td>23-02-2011</td>
<td>0.0947</td>
</tr>
<tr>
<td>06-08-2010</td>
<td>0.4284</td>
<td>12-11-2010</td>
<td>0.1159</td>
<td>24-02-2011</td>
<td>0.0917</td>
</tr>
<tr>
<td>09-08-2010</td>
<td>0.3153</td>
<td>15-11-2010</td>
<td>0.1184</td>
<td>25-02-2011</td>
<td>0.1228</td>
</tr>
<tr>
<td>10-08-2010</td>
<td>0.3136</td>
<td>16-11-2010</td>
<td>0.1187</td>
<td>28-02-2011</td>
<td>0.1240</td>
</tr>
<tr>
<td>11-08-2010</td>
<td>0.0365</td>
<td>17-11-2010</td>
<td>0.1054</td>
<td>01-03-2011</td>
<td>0.1104</td>
</tr>
<tr>
<td>12-08-2010</td>
<td>0.0520</td>
<td>18-11-2010</td>
<td>0.0630</td>
<td>02-03-2011</td>
<td>0.0877</td>
</tr>
<tr>
<td>13-08-2010</td>
<td>0.0738</td>
<td>19-11-2010</td>
<td>0.0663</td>
<td>03-03-2011</td>
<td>0.0557</td>
</tr>
</tbody>
</table>

Fig. 2 shows a comparison of RMSE errors of S&P500 and BSE data sets during training of the two better performing FLANN models like the LEFLANN and CEFLANN using PSO, HMRPSO (adaptive version), DE/current to best methods, etc. From these figures it is quite clear that the RMSE converges to the lowest value for the two FLANN models using DE current to best variant during the training period in comparison to other evolutionary methods like the PSO, and HMRPSO. Comparing the performance of each FLANN using various evolutionary learning methods like PSO, HMRPSO, DE with different mutation strategies, it has been observed that each FLANN trained using DE/current to best mutation provides good result than other learning methods. Thus for producing the final forecast of the stock indices the DE current to best is used to reduce the prediction errors of the stock market indices. The RMSE and MAPE values for the two stock indices like the S&P500 and BSE obtained during training with different models and DE current to best evolutionary approach are shown in Fig. 3. From this figure it is seen that the both the CEFLANN and its recurrent version produce the least RMSE and MAPE errors during training in just 20 iterations. The RMSE and MAPE values for all the FLANN models using different evolutionary training paradigms are shown in Table 1, from which it can be concluded that the recurrent CEFLANN produce the least errors when trained with DE current to best technique. Again to compare the convergence speed of the evolutionary learning algorithms on the CEFLANN models the time elapsed during training is also specified in Table 2. From the table it is also clear that DE method has a faster convergence speed compared to the other two methods.

Figs. 4 and 5 show the actual and predicted closing price values of BSE and S&P500 data sets during testing with different FLANN models, and from this figure it is quite clear that the CEFLANN and recurrent adaptation produce accurate forecasts in comparison to all other FLANN models like the TRFLANN, LAGFLANN, CHFLANN, and LEGFLANN, etc. Also the LAG FLANN and TRFLANN show the worst forecast results for one day ahead stock closing price for both S&P500 and BSE stocks. A comparison of the evolutionary approaches with the well known gradient descent algorithm is also shown in Table 1 showing clearly the superior forecasting performance of the former in comparison to the gradient descent algorithms. Further Tables 3 and 4 show the forecast results from one day ahead to 15 days ahead, in which the RCEFLANN has a MAPE varies for nearly 1–3.15% in comparison with CEFLANN and LEGFLANN and LAG-FLANN from nearly 1.15–6%, respectively (Fig. 7).

Tables 3 and 4 provide a comparison of MAPE values with other neural network models like the RBF neural network (Fig. 6), and Wavelet neural network (WNN), when trained with DE/current to best variant. From these tables it is clearly...
apparent that the RCFLANN is the simplest neural model which is quite robust and provides superior forecasting performance when trained with DE algorithm in comparison to the well established neural networks like the RBF and the wavelet neural network. After it is observed that the RCEFLANN performs the best in predicting the future stock indices, it is important to calculate the day to day variances and variances over a given time frame. Fig. 8 depicts the variances showing clearly the accuracy of the forecasting model, since their magnitudes are quite small over a large time frame (more than one year). Further the day to day variances shown in Tables 5 and 6 completely validate this argument.

### 5. Conclusion

Functional Link Artificial Neural Network is a single layer ANN structure, in which the hidden layers are eliminated by transforming the input pattern to a high dimensional space using a set of polynomial or trigonometric basis functions giving rise to a low complexity neural model. Different variants of FLANN can be modeled depending on the type of basis functions used in functional expansion. In this paper, the detailed architecture and mathematical modeling of different FLANNs with polynomial and trigonometric basis functions have been described to predict the stock market return. Further for improving the performance of different FLANNS, a number of evolutionary methods have been discussed to optimize their parameters. Comparing the performance of each FLANN using various evolutionary learning methods like PSO, HMRRPSO, DE, etc. with different mutation strategies, it has been observed that each FLANN trained using DE/current to best mutation provides good result in comparison to other learning methods. Again the performance comparison of various evolutionary FLANN models to predict stock prices of Bombay Stock Exchange and Standard & Poor’s 500 data set for 1 day, 7 days, 15 days and 30 days ahead, shows that performance of stock price prediction can significantly be enhanced using CEFLANN and recurrent CEFLANN models trained with DE/current to best method in comparison with the well known RBF and WNN models. The recurrent CEFLANN exhibits very low variances and can also be used to predict the volatility of stock indices and the trends for providing trading decisions.

### References


