Arabian Journal for Science and Engineering (2020) 45:11129–11146
https://doi.org/10.1007/s13369-020-05004-5

RESEARCH ARTICLE-COMPUTER ENGINEERING AND COMPUTER SCIENCE

Time Series Forecasting Using Differential Evolution-Based ANN Modelling Scheme

Sibarama Panigrahi[1] · **H. S. Behera**[1]

Received: 2 May 2020 / Accepted: 4 October 2020 / Published online: 20 October 2020

Abstract
Over the past few decades, time series forecasting (TSF) has been predominantly performed using different artificial neural network (ANN) models. However, the performance of ANN models in TSF has not yet been fully explored due to several issues like the determination of near-optimal ANN architecture for a time series and the efficiency of training algorithm used to determine the near-optimal weights of ANN. Motivated by this, we have proposed an adaptive differential evolution (DE)-based modelling scheme to automatically determine the near-optimal architecture of ANN for a time series under study. Additionally, we have proposed an adaptive differential evolution-based ANN training algorithm (ADE-ANNT) to determine the near-optimal weights of ANN. To make the adaptive modelling scheme consistently effective, several comparisons are made between different alternatives in the treatment of trend component and normalization techniques. Twenty-one benchmark time series datasets are being considered to assess the comparative performance of the proposed method with the established forecasting models, namely autoregressive integrated moving average, exponential smoothening with error, trend and seasonality, deep belief network and multilayer perceptron + Levenberg–Marquardt (LM) method. To assess the efficiency of the proposed ADE-ANNT training algorithm, comparisons are made with the ANN training algorithms based on recently developed evolutionary algorithms, such as TLBO-ANNT, DE-CRO-HONNT and DE-ANNT+; and the most popular LM training algorithm. Extensive statistical analysis on simulation results reveal the statistical superiority of the proposed training algorithm and proposed method when compared with their counterparts for the datasets used.

Keywords Artificial neural network · Multilayer perceptron · Differential evolution · Evolutionary neural network · Time series forecasting

1 Introduction

Future of most of the phenomena is a consequence of its past. Therefore, future values of a phenomenon can be extrapolated by systematically analysing its past values. Such a process of predicting the future values of a phenomenon by analysing the past observations is known as time series forecasting (TSF). Accurate TSF plays a vital role in almost every area of study including finance, economics, engineering and management science. Conventionally, statistical models like exponential smoothing, autoregressive integrated moving average (ARIMA) have been widely used in TSF [1]. These models work under the assumption of the linear correlation structure of time series data and often fail to provide a satisfactory result when the time series possess nonlinear patterns. To add to this owe, most of the real-world time series is nonlinear and often suffered from the issues of temporal as well as spatial variability. Hence, traditional statistical models cannot handle the nonlinear patterns equally well as linear patterns. This is because linear models can't handle nonlinear patterns and vice versa [2]. Therefore, nonlinear statistical models like autoregressive heteroskedastic models (ARCH and GARCH family) have been applied in TSF to capture the nonlinear patterns. However, the existence of different variations of these models [3, 4] imposes a burden in choosing the suitable model, and thus limits its application to a generalized forecasting problem.

In the past two decades, ANN models have gained popularity in forecasting theory due to several advantages features. First, unlike traditional models, ANNs are data-driven models, i.e. instead of modelling data by a pre-defined model;

✉ H. S. Behera
hsbehera_it@vssut.ac.in

[1] Department of InformationTechnology, Veer Surendra Sai University of Technology, Burla, Odisha 768018, India

ANN employs the data to determine the parameters of the model. Also, ANNs neither require any a priori assumption on model nor on the characteristics of data to be modelled. Second, ANNs are universal approximators, i.e. ANNs can model any nonlinear or linear function to any desired level of accuracy [5]. Finally, ANNs possess nonlinear modelling ability, i.e. can learn and generalize any function [6]. Considering all these advantages of ANN, thousands of research papers using different ANN models have been published to solve various problems in TSF. Despite more than two decades of research on ANN for TSF and some studies [7, 8] indicating the superior performance of ANN models than statistical models, some studies [9, 10] have also indicated inferior performance under certain datasets. This inconsistent performance of ANN models across various studies was due to several factors such as heuristic and ad hoc modelling process [3], pre-processing (e.g. detrending and normalization) of time series data [11–13] and the effectiveness of the training algorithms used [3]. If these factors were suitably optimized, ANNs would be established as a reliable tool for TSF.

All the ANN-based forecasting methods proposed to forecast time series have used two approaches of ANN model design. In the first approach [14–18], the architecture of ANN (number of inputs, number of hidden layers, number of neurons in each hidden layer) is given, and the remaining parameters (weights, learning rate, etc.) of ANN are computed in the later part of the modelling process. In this approach, neither optimal nor acceptable results can be guaranteed. Contrasting to the first approach, the second one [19–36] determines the architecture and other parameters automatically using different techniques such as pruning algorithm [19, 20], growing algorithm [29], cellular automata [30] and evolutionary algorithms [21–28, 31–36]. The application of evolutionary algorithms to construct the architecture of ANN dominated the literature. It is mostly based upon genetic algorithm (GA) [21, 25, 26, 32], estimation distribution algorithm (EDA) [24, 33], particle swarm optimization (PSO) [27], grasshopper optimization algorithm (GOA) [28] and differential evolution (DE) [24, 33]. The evolutionary algorithms used either direct encoding schemes (DES) [24–28, 31–36] or indirect encoding scheme (IES) [21–23]. In DES, the chromosomes contain information about the learning parameters, architecture and parameters of the topology. In contrast, in IES, the chromosomes contain necessary information so that a constructive method can be used to delineate the architecture of ANN. Although no study has been made relating to the comparative performance of DES and IES, the DES has been used predominantly due to its simplicity in implementation.

In this paper, an adaptive modelling scheme based on DES for time series forecasting is developed. To make a robust evaluation of the proposed modelling scheme, the choice of the ANN model plays a vital role. To address this issue, Makridakis et al. [37] conducted a comparative study on different machine learning (ML) models for TSF and suggested that multilayer perceptron (MLP) provides better result than other ML models including long short-term memory (LSTM). Therefore, in this paper, a single hidden layer feedforward MLP is considered as the forecasting model. The number of neurons in the output layer is fixed to one, whereas the number of inputs and hidden neurons is automatically determined using an adaptive evolutionary algorithm. The DES is used to encode the architecture of ANN into chromosomes; meanwhile, the fitness of each chromosome is obtained after training with the proposed training algorithm (ADE-ANNT). The major contributions of the paper are as follows:

(i) An adaptive DE-based modelling scheme (DEMS) has been developed to determine the near-optimal architecture of ANN for a time series under study.
(ii) An adaptive DE-based ANN training algorithm (ADE-ANNT) has been proposed to determine the near-optimal weights of ANN.
(iii) To make the DEMS consistently effective, several comparisons are made between different alternatives in pre-processing techniques such as treatment of trend component and normalization techniques.

The rest of this paper is organized as follows. Section 2 provides an overview of ANN and DE algorithm. The proposed ADE-ANNT training algorithm is explained in Sect. 3. Section 4 presents a detailed description of the proposed methodology. In Sect. 5, the experimental set up is described, and simulation results are analysed. Finally, the conclusions are drawn in Sect. 6.

2 Preliminaries

2.1 Artificial Neural Network (ANN)

ANNs are data-driven, self-adaptive, flexible models that can approximate an enormous class of nonlinear functions to any preferred degree of precision. Thus, different ANN models (MLP [14], radial basis function (RBF) neural network [15], LSTM [16], convolutional neural network (CNN) [17], functional link ANN (FLANN) [18], deep belief network [27], etc.) have been applied in a variety of TSF applications. Out of different ANN models, feedforward MLP with a single hidden layer is most popular in TSF [3, 37]. This is because of its complex nonlinear mapping capability and universal approximation capability [3]. The architecture (as in Fig. 1) of such MLP consists of an input, hidden and output layer. The neurons of adjacent layers are symmetrically connected

by directed acyclic links. In TSF, the number of inputs and hidden neurons is flexible, whereas the number of output neurons is usually kept to one. Solving the TSF problem using MLP can be considered as identifying the underlying relationship (as in Eq. 1) existing between the output y_t and the past k values of the series $y = [y_1, y_2, \ldots, y_n]$. Therefore, the time series data are first transformed to n-k patterns, with each pattern has k inputs $y_{t-1}, y_{t-2}, \ldots, y_{t-k}$ and one target y_t. Generally, the patterns are partitioned into train, validation and test sets. The training algorithms like evolutionary training algorithms [18, 38–44], Levenberg–Marquardt (LM) algorithm, etc. use the train and validation sets to determine the weights and other parameters of ANN. Then, the future values on the test set are computed and used to evaluate the performance of the model.

$$y_t = f\left(\alpha_0 + \sum_{j=1}^{q} \alpha_j g\left(\beta_{0j} + \sum_{i=1}^{k} \beta_{ij} y_{t-1}\right)\right) \tag{1}$$

where α_j and β_{ij} are the set of weights between input–hidden and hidden–output layers. β_{0j} and α_0 are the bias unit of hidden and output neurons. Usually, binary sigmoid activation function (as in Eq. 2) has been widely used in the neurons of the hidden layer. In contrast, the linear activation function has been used in the neurons of the output layer.

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

2.2 Differential Evolution

In 1997, Storn et al. [45] proposed a simple yet efficient evolutionary algorithm called differential evolution (DE). Compared to other evolutionary algorithms, DE is straightforward and easier to implement. Even though PSO and its variants are easy to code, DE variants outperformed the PSO variants and other evolutionary algorithms [46–48]. The recent review paper [49] summarizes the superior performance of DE than different evolutionary algorithms in CEC competitions organized from 2005 to 2018. DE is a stochastic search method, which starts with a set of chromosomes (decision vectors), with each chromosome consisting of a set of genes (decision variables). The chromosomes of a generation use mutation, crossover and greedy selection operations repeatedly to move the population towards the next generations until a near-optimal solution is achieved.

All DE algorithm variants operate in the following steps:

1. Initialization of problem parameters, algorithm parameters and initial population (consisting of a set of chromosomes).
2. Calculate the fitness of each chromosome/individual.
3. For each chromosome (target vector) of the current generation, apply step: 3.1 to step: 3.3 to generate the chromosomes of next generation.
   - 3.1 Apply mutation operator (anyone from Eqs. 3 to 7) to generate the mutant (donor) vector
   - 3.2 Generate the trial vector by applying crossover (either Eqs. 8 or 9) between the target vector and mutant vector.
   - 3.3 Perform selection (Eq. 10) between target vector and trial vector
4. If the termination criteria are satisfied, go to step-5 or else go to step-3
5. Use the fittest individual of the final generation as the solution to the problem.

Let the population of gth generation is $p_g = \left(C_g^1, C_g^2, \ldots C_g^i \ldots C_g^{PopSize}\right)$, with $C_g^i = \left(W_g^{i1}, W_g^{i2}, \ldots W_g^{ij} \ldots W_g^{iD}\right)$ for i = 1,2,3…..PopSize, D = length of each chromosome, W_g^{ij} = jth gene of the ith individual in gth generation. The mutation for ith chromosome produces a mutant vector $M_g^i = \left(M_g^{i1}, M_g^{i2}, \ldots M_g^{ij} \ldots M_g^{iD}\right)$, which is used in the crossover with the current chromosome $C_g^i = \left(W_g^{i1}, W_g^{i2}, \ldots W_g^{ij} \ldots W_g^{iD}\right)$ to produce a trial vector $T_g^i = \left(T_g^{i1}, T_g^{i2}, \ldots T_g^{ij} \ldots T_g^{iD}\right)$

Since the inception of DE, it has been modified and upgraded rigorously in recent years [49, 50]. However, the variants of the DE algorithm mostly vary in the nature of mutation and crossover scheme being used. Storn and Price [51, 52] suggested different mutation schemes (as in Eqs. 3 to 7) for differential evolution to generate the mutant vector. Two crossover methods such as binomial (as in Eq. 8) and exponential (as in Eq. 9) can be applied on any of the mentioned mutation strategies to generate the trial vector. Thus, a total of $2 \times 5 = 10$ different DE mutation strategies can be used to generate a trial vector. Then, selection (as in Eq. 10) operation is applied to select the better chromosome between trial and target vector for the next generation.

$$\mathbf{DE/best}/1 : M_g^i = C_g^{best} + F \times \left(C_g^{r1} - C_g^{r2}\right) \tag{3}$$

$$\mathbf{DE/rand}/1 : M_g^i = C_g^{r1} + F \times \left(C_g^{r2} - C_g^{r3}\right) \tag{4}$$

$$\mathbf{DE/rand}/2 : M_g^i = C_g^{r1} + F \times \left(C_g^{r2} - C_g^{r3}\right) + F \times \left(C_g^{r4} - C_g^{r5}\right) \tag{5}$$

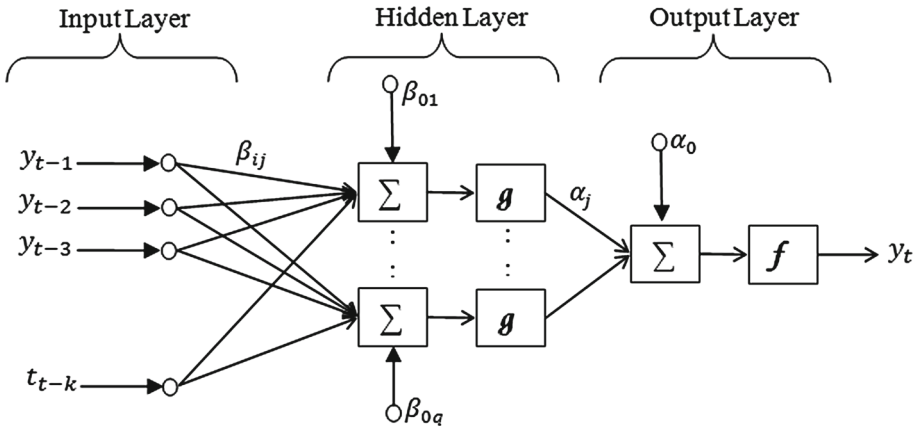

Fig. 1 Architecture of three-layer ANN

$$\mathbf{DE/best/2}: M_g^i = C_g^{best} + F \times \left(C_g^{r1} - C_g^{r2}\right) + F \times \left(C_g^{r3} - C_g^{r4}\right) \tag{6}$$

$$\mathbf{DE/target-to-best/1}: M_g^i = C_g^i + F \times \left(C_g^{best} - C_g^{r1}\right) + F \times \left(C_g^{best} - C_g^{r2}\right) \tag{7}$$

$$T_g^{ik} = \begin{cases} M_g^{ik} & if rand(0,1) < Cr \\ W_g^{ik} & otherwise \end{cases} \text{ for } k = 1, 2, \ldots \mathrm{D} \tag{8}$$

$$T_g^{ik} = \begin{cases} M_g^{ik} & l \le k \le (l+t) \\ W_g^{ik} & otherwise \end{cases} \text{ for } k = 1, 2, \ldots \mathrm{D} \tag{9}$$

$$C_{g+1}^i = \begin{cases} T_g^i & Fitness\left(T_g^i\right) \text{ is better than } Fitness\left(C_g^i\right) \\ C_g^i & otherwise \end{cases} \tag{10}$$

The notations used above are in the form *DE/a/b*, where *DE* means differential evolution, *a* indicates the choice of base vector used in mutation (it may be a randomly chosen vector, target vector or the best vector), *b* indicates the number of difference vectors applied in mutation; M_g^i stands for *i*th mutant vector in *g*th generation; C_g^{best} for the best vector of the population in *g*th generation; C_g^r for a randomly chosen chromosome from the population of generation *g*; C_g^i is the target (donor) vector; *Cr* is the crossover probability; *F* is the scale factor; T_g^{ik} *k*th gene of *i*th chromosome in *g*th generation; rand (0,1) represents a random number between 0 and 1; and the value *l* and *k* are chosen based on the crossover probability in such manner that: $1 \le l \le (l+k) \le D$ with *D* is the dimension of the chromosome. Note that, all the chromosomes which are selected for any mutation scheme must be distinct to each other.

3 Proposed ANN Training Algorithm (ADE-ANNT)

In the proposed method, a large number of different ANN architectures need to be trained to obtain the best architecture for TSF. This demands a robust training algorithm that reaches a near-optimal solution in quick time. Conventionally, gradient-based algorithms are widely used for training ANN. However, the gradient-based methods have several shortcomings such as unreliable [53], slow convergence speed [54], chances to trapping to local optima is high [55], performance is subjected to initial values of algorithm parameters [54], etc. Therefore, recently, several evolutionary ANN training algorithms [18, 38–44] have been developed. Out of several evolutionary algorithms, the applications of DE to ANN training [38–41, 44] have dominated the literature. Some authors [38–40, 44] have used traditional DE variants for ANN training, whereas Slowik [41] proposed DE-ANNT + training algorithm with multiple trial vectors concept. In DE-ANNT + algorithm, for each target vector, multiple mutant vectors are generated. The mutant vector having the best fitness is used in crossover to generate a trial vector. Then, selection operation is carried out between the trial vector and target vector to generate the chromosomes for the next generation. The control parameters *F* and *Cr* are obtained based on the rate of change of the best solution (Eq. 11), where *TheBest* represents the fitness of the best chromosome. In Slowik [41], the population is said to be stagnated when the value of *R* (Eq. 11) of the population does not change by a certain percentage. DE-ANNT + has shown better performance than extended backpropagation (EBP), EA-ANNT and DE-ANNT. Also, the results are as good as the Levenberg–Marquardt algorithm. However, the major drawback of DE-ANNT + is that it suffers from exploitation due to the generation of multiple mutant vectors for each individual in every generation. Therefore,

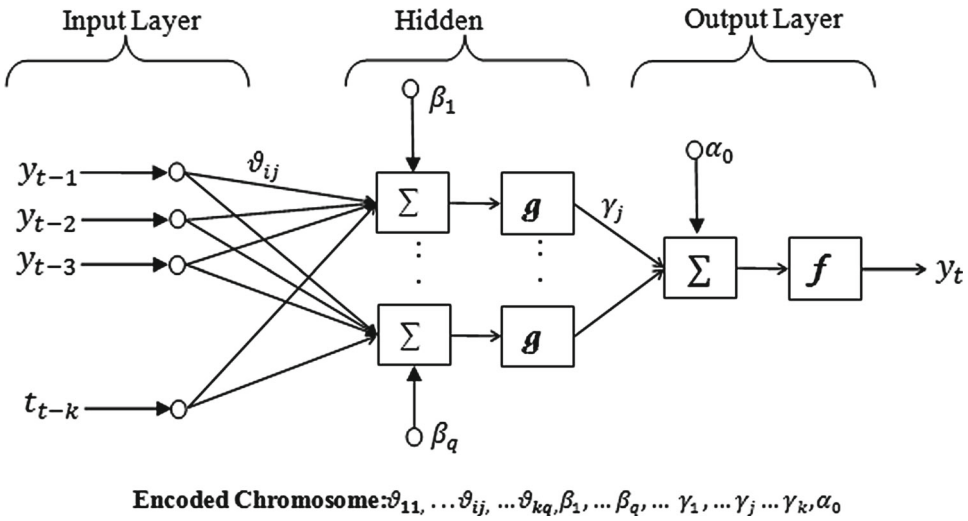

Fig. 2 Encoding weights of ANN on a Chromosome

DE-ANNT + is not so suitable for the proposed self-adaptive modelling scheme.

$$Rate\ of\ change\ of\ best\ solution\,(R) = TheBest_i \div TheBest_{i-1} \tag{11}$$

To develop a robust DE-based training algorithm, the choice of DE strategy plays a vital role. To address this issue, Mezura-Montes et al. [56] made a comparative study on eight different DE schemes using 13 benchmark problems and suggested that the DE/best/1/bin scheme provides robust result irrespective of the problem in hand. However, the DE/best/1 scheme has the tendency of premature convergence, which can be effectively avoided by generating multiple trial vectors upon stagnation. Therefore, by integrating DE/best/1/bin strategy with the concept of multiple trial vectors upon stagnation, an adaptive differential evolution with conditional multiple trial vectors for ANN training (ADE-ANNT) is proposed. In addition, the values of control parameters F and Cr are determined self-adaptively based on the problem in hand.

Algorithm 1 presents the pseudocode of ADE-ANNT. In this algorithm, the trainable weight set of an ANN is encoded into the chromosomes as a list of real numbers (as in Fig. 2) with each gene representing a weight of ANN. Hence, the length of each chromosome for an ANN with k inputs, q hidden neurons and one output neuron with bias component is $q \times (k+2) + 1$. The first generation of the population consists of *PopSize* number of randomly selected chromosomes, with each chromosome denoting the weights of ANN. The fitness of each chromosome (weight set) is obtained by using Eq. 12. Note that, when a chromosome has a lower root mean square error (RMSE) (as in Eq. 13) than another chromosome, the chromosome having lower RMSE is considered as better. Then, the next generations of the population are repeatedly generated until the termination criteria are satisfied. The next generation of the population is generated by applying mutation, crossover and greedy selection operators on each chromosome of the current generation. DE/best/1 mutation scheme is applied to generate the mutant vector(s) using the proposed parameter adaptation scheme. The number of mutant vectors depends on the stagnation criteria, i.e. upon stagnation of the population, multiple mutant vectors are generated by using multiple scale factors. Unlike DE-ANNT + [41], all the mutant vectors are crossed over with the target vector to generate multiple trial vectors ($v > 1$) and then the best trial vector is used for selection. If stagnation does not occur, one scale factor is used to generate one mutant vector, which consequently generates one trial vector. Moreover, motivated by [57, 58], the scale factor and crossover probability are self-adaptively, and dynamically determined considering the problem in hand.

$$\text{Fitness} = -1 \times \text{RMSE} \tag{12}$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - T_i)^2} \tag{13}$$

where n represents the number of observations, T_i and Y_i are computed and actual values of ith observation.

Algorithm 1 Adaptive DE based ANN training algorithm (ADE-ANNT)

Input: Number of inputs (k), Number of neurons in the hidden layer (q), Pattern set (n-k patterns), number of train (l_{tr}) and validation (l_{tv}) patterns.

Output: Predicted values on the test set

Step 1 Initialization Step

1.1: Initialize the generation counter g=0, $F_m = 0.5$, $Cr_m = 0.5$, v (greater than 1)

1.2: Randomly Initialize the population of P_{size} individuals: $p_g = (C_g^1, \ldots, C_g^{P_{size}})$, with $C_g^i = (W_g^{i,1}, \ldots, W_g^{i,L})$ for i=1,..... P_{size}, L=number of genes in each chromosome, $W_g^{i,j}$ = jth gene of ith chromosome in gth generation representing a weight of ANN.

1.3: Calculate the fitness of each chromosome

Step 2 Iteration Step

2.1: While (termination criteria are not satisfied) do begin

2.2: for i=1 to P_{size}

2.3: Select the fittest individual (C_g^{best}) and generate 2 random numbers (r_2, r_3) such that $r_1 \neq r_2 \neq i \neq best$

2.4: Generate v scale factors $F = (F_1, \ldots F_j \ldots F_v)$by using the parameter adaption scheme

2.5: %Using v scale factors generate v mutant vectors for ith chromosome

2.6: $M_i = (M_i^1, M_i^2, \ldots M_i^j \ldots M_i^v)$

2.7: for j=1 to v

2.8: $M_i^j = C_g^{best} + F_j \times (C_g^{r1} - C_g^{r2})$

2.9: end of for

2.10: Generate the crossover probability Cr using parameter adaption scheme

2.11: %Generate v trial vectors for ith chromosome $T_i = (T_i^1, T_i^2, \ldots T_i^j \ldots T_i^v)$

2.12: for j=1 to v

2.13: for x= 1to L

2.14: if rand(0,1)< Cr

2.15: $T_i^{j,x} = M_i^{j,x}$

2.16: else

2.17: $T_i^{j,x} = W_g^{j,x}$

2.18: end of if

2.19: end of for

2.20: end of for

2.21: Select the trial vector having the least RMSE as best trial vector T_i^{best}

2.22: if fitness(T_i^{best}) > fitness(C_g^i)

2.23: set $C_{g+1}^i = T_i^{best}$, $F_{Success} = F_{best}$, $Cr_{Success} = Cr$

2.24: else

2.25: $C_{g+1}^i = C_g^i$

2.26: end of if

2.27: end of for

2.28: Update F_m and Cr_m

2.29: Set g=g+1 and value of v based on the stagnation criteria

2.30: end of while

Step 3 Final Step

Use the chromosome having the best fitness as the optimal weight set of ANN and using which the values for test set are predicted.

3.1 Parameter Adaption in ADE-ANNT

Algorithm 2 presents the pseudocode of the parameter adaptation used in the proposed ADE-ANNT training algorithm. In every generation, the scale factor F_i and crossover probability Cr_i for each target vector is generated. When stagnation occurs, more than one trial vectors (v) are generated. To achieve this, v (>1) number of scale factors are used to generate v mutant vectors which after crossover generates v trial vectors. If stagnation does not occur, one scale factor is used to generate one mutant vector, which consequently generates one trial vector.

Algorithm 2 Pseudo_code of parameter adaptation

If Stagnation

Generate v (>1) number of scale factors $F = (F_1, F_2, F_3, \ldots F_v)$

$F = 2 \times rand(1, v) F_i \in [0,2]$ *with* $i = 1,2,3, \ldots v$

Else

Generate v (=1) scale factor (F_1) $F = (F_1)$

$F_1 = Cauchyrnd(F_m, 0.1)$ *with* $F_1 \in [0; 1]$

End of if

Generate crossover probability

$Cr = Gaussianrnd(Cr_m, 0.1)$ *with* $Cr \in [0; 1]$

Where rand(1,v) stands for v randomly generated numbers from a uniform distribution within the range (0, 1), Cauchyrnd(F_m, 0.1) is a randomly generated number from a Cauchy distribution having scale parameter 0.1 and location parameter F_m, Gaussianrnd(Cr_m, 0.1) is a randomly generated number from a Gaussian distribution with standard deviation 0.1 and mean Cr_m.

Initially, the location parameter F_m is set to 0.5 and is then updated after each generation (using Eq. 14).

$$W_f = 0.7 + 0.3 \times \text{rand}(0, 1) \tag{14}$$

$$F_m = W_f \times F_m + \left(1 - W_f\right) \times mean(F_{success})$$

where W_F is the weight factor which varies between 0.8 and 1. $F_{success}$ is the set of successful scale factors using which better trial vectors for the next generation are generated.

The mean of the Gaussian distribution Cr_m is initially set to 0.5 and is then updated after each generation (using Eq. 15).

$$W_{cr} = 0.7 + 0.3 \times \text{rand}(0, 1) \tag{15}$$

$$Cr_m = W_{cr} \times Cr_m + (1 - W_{cr}) \times \text{mean}(Cr_{success})$$

where W_{Cr} is the weight factor varying within 0.8 and 1. $\text{Cr}_{\text{success}}$ is the set of the successful crossover probabilities generating better trial vectors in the current generation.

3.2 Explanation of Parameter Adaption

Upon stagnation, multiple scale factors are used to generate multiple mutant vectors. The multiple mutant vectors are crossed over with the target vector to generate multiple trial vectors. The generation of multiple trial vectors around the existing solution assists in getting out of stagnation. This is because (1) If the difference vector $C_g^{r1} - C_g^{r2}$ becomes small, i.e. when the vectors converge to a small domain, the larger values of scale factor assists in getting out of the suboptimal valleys/peaks causing stagnation; (2) If the difference vector $C_g^{r1} - C_g^{r2}$ is large, the population may jump the global minima due to larger difference vectors, but the use of scale factors having smaller values assists in overcoming this problem. Thus, the generation of multiple scale factors from a uniform distribution between (0–2) helps in performing both local and global search.

When stagnation does not occur, only one scale factor is randomly generated from a Cauchy distribution with scale factor 0.1 and location parameter F_m. This is because the location parameter F_m of Cauchy distribution varies the value of F more than the conventional normal distribution. Moreover, during the adjustment of F_m, the use of the arithmetic mean guides F_m towards a higher value which aids larger perturbation to the target vectors; hence, premature convergence to local minima is avoided. Irrespective of stagnation criteria, $F_{success}$ records the successful scale factors that generate better trial vectors than target vector in the current generation. Thus, it increases the chances of generating better trial vectors as the generations proceed further.

The crossover probability generated is independent of stagnation the criterion, i.e. for both the cases, the crossover

probability is randomly generated from a Gaussian distribution with standard deviation 0.1 and mean Cr_m. The adaptation of Cr_m is also based on the memorization of thriving crossover probabilities generating better trial vectors. The use of arithmetic mean while adjusting the value of Cr_m avoids the bias of Cr towards smaller values [41]. Here, we have chosen Gaussian distribution because it glorifies the chances to generate the majority of Cr values within unity [41].

4 Proposed Methodology

This section presents the proposed DE-based modelling scheme (DEMS) to automatically determine the most parsimonious ANN architecture for a time series under study. The DEMS method (as in Fig. 3) follows the wrapper approach of model selection. It uses the proposed ADE-ANNT training algorithm to determine the near-optimal weight set and fitness of ANN architectures. Algorithm 3 presents the steps of the proposed DEMS method which operates in four steps, namely (a) Initialization, (b) Pre-processing, (c) ANN modelling and (d) Forecast.

Algorithm 3: Proposed DE based Modeling Scheme (DEMS)

Input: Time Series(Y)and Forecast horizon(h)

Output: Forecasted values and forecast accuracy

Step 1 Initialization

1.1: length of train set (l_{tr}), length of validation set(l_v), calculate the length of the test set(l_{te}), maximum number of inputs(k_{max}), maximum number of hidden neurons(q_{max}), population size(P_{size})

Step 2 Pre-processing

2.1: Normalize the time series

Step 3 ANN Modeling

3.1: Randomly initialize P_{size} number of chromosomes from a uniform distribution (0-2.5) with each chromosome representing an ANN architecture.

3.2: Calculate the fitness of each chromosome

3.3: for i=1 to P_{size}

3.4: Calculate the number of inputs k

3.5: Calculate the number of hidden neurons q

3.6: Transform the time series into $n-k$ patterns.

3.7: Calculate the fitness $k-q-1$ ANN architecture using ADE-ANNT

3.8: end of for

3.9: While (termination criteria are not satisfied)

3.10: for j=1 to P_{size}

3.11: Generate two distinct random numbers r_1, r_2 from an interval $(1-P_{size})$

3.12: Generate the mutant vector using ADE-ANNT mutation scheme

3.13: Obtain the trial vector(s) using ADE-ANNT crossover scheme

3.14: Perform selection between the best trial vector T^{best} and target vector C^i

3.15: end of for

3.16: i=i+1

3.17: Calculate the fitness of the best chromosome on the validation set**,** If the fitness on validation set increases than that of the previous generation then termination criteria are satisfied.

3.18: end of while

Step 4 Forecast

4.1: Obtain the forecasted values using the architecture having the best fitness.

4.2: De-normalize the forecasted values to get true forecasts.

4.3: Compute the forecast accuracy.

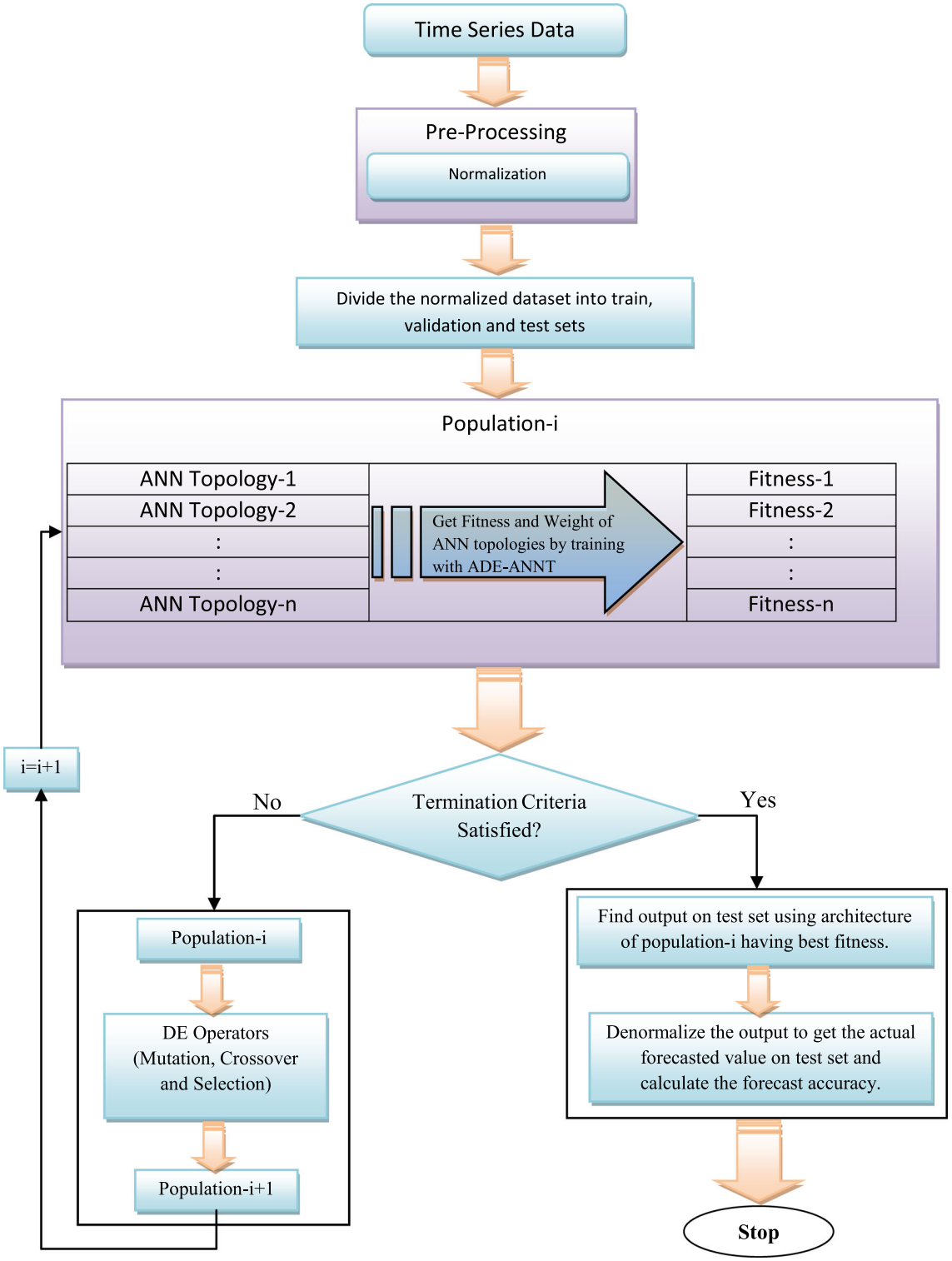

Fig. 3 Graphical abstract of the proposed method

In the initialization step, the problem parameters like the length of the train (l_{tr}), validation (l_v) and test (l_{te}) are initialized. The number of inputs of ANN is restricted to 1 to k_{max} (computed using Eq. 16), and the number hidden layer neurons of ANN is restricted to 1 to q_{max} (calculated using Eq. 17).

$$k_{max} = 10 \times \log_{10}(l_{tr} + l_v) \tag{16}$$

$$q_{max} = 2 \times k_{max} \tag{17}$$

In the pre-processing step, the time series is normalized to a range between (0–1). To assess the sensitivity to different normalization techniques, min–max (as in Eq. 18), decimal scaling (as in Eq. 19) and vector (as in Eq. 20) normalization techniques are considered. Also, the effect of treatment of trend by taking the first difference of the series is tested. But differencing is avoided due to deterioration in performance. Let $y = \left[y_1, y_2, \ldots\ldots\ldots\ldots, y_n\right]^T$ be the n-dimensional time series vector and $y' = \left[y_1^{'}, y_2^{'}, \ldots, y_n^{'}\right]^T$ be the normalized series vector with $y_i^{'}$ is the ith normalized data of y_i.

$$y_i' = \frac{y_i - \min_y}{\max_y - \min_y} \tag{18}$$

$$y_i' = \frac{y_i}{10^{\log_{10}(\max_y)}} \tag{19}$$

$$y_i' = \frac{y_i}{\sqrt{\sum_{j=1}^{n} y_j^2}} \tag{20}$$

In the ANN modelling step, the proposed adaptive DE algorithm is used to search the most parsimonious ANN architecture in the space of all possible architecture combinations. The DES is used to encode the ANN architectures into the chromosomes. Initially, P_{size} number of chromosomes are randomly chosen between 0 and 2.5, with each chromosome $C_j = \{g_1, g_2, g_3, g_4, g_5, g_6, g_7, g_8\}$ consisting of 8 genes (4 are used for the number of inputs and four are used for the number of hidden neurons). Then, the fitness (i.e. the RMSE measure on train set) of each chromosome is calculated. To calculate the fitness, the genotypes are first converted to phenotypes (i.e. ANN architectures). The number of inputs *(k)* of ANN is calculated using Eq. 21 and the number of neurons in the hidden layer *(q)* are computed using Eq. 22.

$$k = \frac{k_{max} \times (g_1 + g_2 + g_3 + g_4)}{10} \tag{21}$$

$$q = \frac{q_{max} \times (g_5 + g_6 + g_7 + g_8)}{10} \tag{22}$$

Once the number of inputs of ANN is determined, the normalized time series is converted to $n - k$ patterns with each pattern consisting of k inputs, one target. After the determination of $k - q - 1$ ANN architecture and patterns, the fitness of ANN architecture is obtained using the proposed ADE-ANNT training algorithm. Then, the DE mutation, crossover and selection operators are repeatedly applied till the termination criterion is satisfied, i.e. the error on the validation set increases. When the termination criterion is satisfied, the architecture having the best fitness is chosen as the near-optimal ANN architecture for the time series under study.

In the Forecast step, the values for the test set are predicted using the obtained *k-q-1* architecture and near-optimal weight set. The predicted values are de-normalized to obtain the actual forecasts, and then forecast accuracy is measured.

5 Experimental Setup and Results

In this section, the simulation results, along with the analysis and discussions, are presented. ARIMA and ETS models are a linear form of most popular neural network MLP and hence considered as an excellent benchmark to compare with ML models [37]. Therefore, in this study, ARIMA and ETS are considered as comparative models. The appropriate ARIMA and ETS model for a time series is determined by using the Forecast package of R [59]. In addition, to evaluate the effectiveness of the proposed architecture selection DEMS method, another architecture selection method using DBN [27] is considered. Additionally, the MLP model trained using the Levenberg–Marquardt (LM) algorithm (MLP + LM) is used for comparison. The MLP + LM method is implemented using the neural network toolbox of MATLAB. In the MLP + LM method, the number of output is fixed to one, whereas the number of input is determined by analysing the autocorrelation and partial autocorrelation function of the time series. To avoid the inconsistencies arising due to improper ANN architecture and to make a fair comparison with the proposed DEMS method, the number of hidden neurons in MLP + LM method is determined by altering the neurons between 1 and 20 (as in [2, 60]) and the architecture having the smallest mean SMAPE over 50 independent simulations is considered. In all the MLP models, the binary sigmoid activation function is used in the neurons of the hidden layer, and linear activation is used in the neuron of the output layer.

Twenty-one time series datasets (described in Table 1) from time series data library [61] are considered to evaluate the methods using root mean square error (RMSE as in Eq. 12) and symmetric mean absolute percentage error (SMAPE as in Eq. 23). SMAPE is a scale-free measure which can be used to evaluate the forecasting methods across one or more time series datasets and hence used in the NN3 forecasting competition to assess the forecasting methods [13]. Therefore, in this paper, SMAPE measure is considered to evaluate the forecasting methods. Additionally, since RMSE is used to assess the fitness of the proposed ADE-ANNT training algorithm, the RMSE measure is also considered to evaluate the forecasting methods. In the proposed DEMS method, the population size P_{size} of DE is fixed to 10, the number of genes in each chromosome is fixed to 8. The genes are initialized and bound in a range (0–2.5). To make a fair

Table 1 Description of time series datasets

Time series	Description	Interval	Duration
Accidental Death	Accidental deaths in USA	Monthly	1973–1978
Acres Burned	Acres burned in forest fires in Canada	Yearly	1918–1988
Car Sells	Car sales in Quebec	Monthly	1960–1968
Chickenpox	Number of chickenpox reported in New York city	Monthly	1931–1972
Colorado River	Flows in Colorado River Lees Ferry	Monthly	1911–1972
Gas Usage	Residential gas usage in Iowa in cubic feet	Monthly	1/1971–10/1979
Internet Traffic	Internet traffic data (in bits) in the United Kingdom academic network	Yearly	19/11/2004–26/01/2005
ITDUK	Aggregate internet traffic data (in bits) of the United Kingdom academic network	Hourly	19/11/2004, 09:30 to 27/01/2005, 11:11
Lake	Lake Erie Levels	Monthly	1921–1970
Lynx	Number of lynx trapped in the Mackenzie River of Northern Canada district	Yearly	1821-1934
Milk	Milk production: pounds per cow	Monthly	1962–1975
Mumps	Number mumps cases reported in New York city	Monthly	1928–1972
Passenger	Airline passengers in thousands	Monthly	01/1949–12/1960
Pollution	Shipment of pollution equipment (in thousands) of French francs	Monthly	01/1966–10/1976
Rainfall	Rainfall in inches at London	Yearly	1813–1912
Stock	Common stock price, US	Yearly	1871–1970
Sun Spot	Wolf's Sunspot Numbers	Yearly	1700–1987
Tasty Cola	Tasty Cola sales	Monthly	2001–2003
Temperature	Average monthly temperature of Nottingham Castle city in Fahrenheit	Monthly	1920-1939
Traffic	Traffic fatalities in Ontario	Monthly	1960–1974
Unemployment	Canadian total unemployment in thousands	Monthly	1956–1975

comparison of the proposed DEMS method with DBN [27], the number of particles in PSO is set to 10, early stopping is used in backpropagation algorithm, and other experimental parameters are kept same to that of DBN [27]. The parameters used in the proposed ADE-ANNT training algorithm are presented in Table 2. Table 3 presents the most parsimonious ARIMA model, ETS model and ANN architecture obtained and used in the simulations. With this experimental setup, for each method, 50 independent simulations are carried out on each time series dataset (as in Table 1). The analysis on obtained results are presented in two sections: (a) Analysing results using RMSE measure, (b) Analysing results using SMAPE measure.

$$SMAPE = \frac{1}{n}\sum_{j=1}^{n}\frac{|Y_j - T_j|}{(|Y_j| + |T_j|)/2} \tag{23}$$

where n represents the number of observations, T_i and Y_i are computed and actual values of ith observation.

Table 2 Simulated parameters in ADE-ANNT

Parameters	Value
Population Size (P_{size})	50
Bound of genes of chromosomes	[0–1]
Stagnation criterion	$R \leq 0.95$
Number of trial vectors (v)	5 if stagnation
	1 otherwise

5.1 Analysing Results Using RMSE Measure

Table 4 presents the mean RMSE over 50 independent simulations for each time series. However, to make a comparative statistical analysis of the proposed DEMS method with other methods, the Wilcoxon signed-rank test [62] is applied. Table 5 presents the individual hypothesis test results. It can be observed from Tables 4 and 5 that the DEMS method provides the best RMSE on 13 time series datasets (Acres Burned, Accidental Death, Internet Traffic, Gas Usage, Lake, Lynx, Milk, Mumps, Tasty Cola, Rainfall, Passenger, Temperature, Unemployment) with 11 statistical significant (as

Table 3 Most parsimonious models identified and used in the simulation

Time series	ARIMA	ETS	MLP architecture obtained using proposed DEMS
Accidental death	ARIMA(2,0,2)	ETS(A,N,N)	7–4–1
Acres burned	ARIMA(0,0,0)	ETS(A,N,N)	2–1–1
Car sells	ARIMA(2,1,3)	ETS(A,N,N)	12–1–1
Chickenpox	ARIMA(4,0,2)	ETS(A,Ad,N)	2–4–1
Colorado river	ARIMA(3,1,3)	ETS(A,N,N)	5–1–1
Gas usage	ARIMA(1,0,1)	ETS(A,N,N)	3–5–1
Internet traffic	ARIMA(2,1,0)	ETS(A,N,N)	6–4–1
Internet traffic UK	ARIMA(1,1,2)	ETS(A,Ad,N)	21–8–1
Lake	ARIMA(5,1,1)	ETS(A,N,N)	25–1–1
Lynx	ARIMA(2,0,2)	ETS(A,N,N)	2–3–1
Milk	ARIMA(1,1,3)	ETS(A,N,N)	19–1–1
Mumps	ARIMA(2,0,2)	ETS(A,Ad,N)	23–6–1
Passenger	ARIMA(2,1,3)	ETS(A,N,N)	12–3–1
Pollution	ARIMA(2,1,0)	ETS(A,A,N)	13–1–1
Rainfall	ARIMA(0,0,0)	ETS(A,N,N)	5–1–1
Stock	ARIMA(2,1,1)	ETS(A,N,N)	12–6–1
Sun Spot	ARIMA(2,0,1)	ETS(A,N,N)	4–8–1
Tasty Cola	ARIMA(2,0,1)	ETS(A,N,N)	13–1–1
Temperature	ARIMA(5,0,1)	ETS(A,N,N)	12–1–1
Traffic	ARIMA(0,1,0)	ETS(A,N,N)	10–3–1
Unemployment	ARIMA(2,0,3)	ETS(A,N,N)	12–3–1

Table 4 RMSE for each dataset (best values are presented in bold face)

Time Series	ETS	ARIMA	MLP + LM	DBN [27]	Proposed DEMS
Accidental Death	685.21	584.06	569.76	579.37	**447.45**
Acres Burned	2,891,782.4	2,993,543.9	3,210,620.97	2,879,300	**2,840,956.0**
Car Sells	3636.61	2851.18	**2172.89**	4142.9	2892.41
Chickenpox	171.39	233.75	**164.37**	182.75	181.43
Colorado River	0.50	**0.47**	0.59	0.59	0.53
Gas Usage	45.94	39.36	19.00	20.09	**15.75**
Internet Traffic	184,862.83	148,542.44	156,735.15	221,330.02	**127,679.82**
ITDUK	4030.9	3933.60	**1976.22**	7902.7	2011.53
Lake	0.51	0.36	0.39	0.82	**0.35**
Lynx	967.75	806.64	1259.20	1039.3	**727.37**
Milk	26.78	23.27	15.44	113.42	**11.73**
Mumps	108.15	109.99	104.59	90.53	**84.11**
Passenger	52.43	37.38	50.42	162.59	**30.05**
Pollution	872.96	**771.91**	1917.67	1810.9	1094.80
Rainfall	4.09	4.22	4.38	4.22	**3.88**
Stock	**6.97**	8.6	44.42	52.63	20.83
Sun Spot	31.88	**21.56**	31.71	31.269	23.94
Tasty Cola	277.96	209.25	267.17	559.12	**171.67**
Temperature	5.14	2.69	2.71	2.74	**2.67**
Traffic	31.74	31.36	**28.71**	34.23	32.99
Unemployment	51.29	54.27	46.70	186.03	**36.08**

in Table 5). Additionally, it provides better RMSE than ETS in 16 cases, ARIMA in 15 cases (14 statistically significant), MLP + LM in 16 cases and DBN [27] in 21 cases (18 statistically significant). It provides inferior RMSE than

Table 5 Individual hypothesis test results using RMSE indicating the superior (+), inferior (−) or equivalent (≈) method with respect to the proposed DEMS method

Time Series	ETS	ARIMA	MLP + LM	DBN [27]
Accidental Death	−	−	−	−
Acres Burned	−	−	−	≈
Car Sells	−	≈	+	−
Chickenpox	+	−	+	≈
Colorado River	+	+	−	−
Gas Usage	−	−	−	−
Internet Traffic	−	−	−	−
ITDUK	−	−	+	−
Lake	−	−	−	−
Lynx	−	−	−	−
Milk	−	−	−	−
Mumps	−	−	−	−
Passenger	−	−	−	−
Pollution	+	+	−	−
Rainfall	−	−	−	−
Stock	+	+	−	−
Sun Spot	−	+	−	−
Tasty Cola	−	−	−	−
Temperature	−	≈	≈	≈
Traffic	≈	≈	+	−
Unemployment	−	−	−	−

ETS, ARIMA and MLP + LM in only 4, 5 and 4 time series datasets, respectively, and for other cases, it provides statistically equivalent RMSE.

5.2 Analysing Results Using SMAPE Measure

In this subsection, the results are analysed using the scale-free SMAPE performance measure. Therefore, the results are analysed considering each time series individually and all time series at a time. For evaluating the methods dataset wise, the mean SMAPE over 50 independent simulations is calculated and presented in Table 6. To statistically compare the proposed DEMS method with other methods, Wilcoxon signed-rank test [62] is applied, and the test results are presented in Table 7. It can be observed from Table 6 that the DEMS method provides the best SMAPE in 11 time series datasets (Accidental Death, Chickenpox, Internet Traffic, ITDUK, Lake, Milk, Passenger, Sun Spot, Tasty Cola, Temperature, Unemployment) with statistical significance in 8 cases (as in Table 7). In addition to this, it provides better SMAPE than ETS in 15 time series datasets, ARIMA in 15 time series datasets, MLP + LM in 16 time series datasets and DBN [27] in 17 (16 statistically significant) time series datasets. It provides inferior SMAPE than ETS in 4, ARIMA in 5, MLP + LM in 2 and DBN [27] in 1 time series datasets and for other cases, it provides statistically equivalent SMAPE.

Table 6 SMAPE for each dataset (best values are presented in bold face)

Time Series	ETS	ARIMA	MLP + LM	DBN [27]	Proposed DEMS
Accidental Death	7.04	5.42	5.52	5.61	**4.07**
Acres Burned	62.45	53.77	58.48	**51.56**	52.33
Car Sells	15.84	12.66	**10.79**	17.65	13.58
Chickenpox	54.30	54.43	39.65	43.14	**35.61**
Colorado River	**28.82**	37.02	41.94	42.45	38.91
Gas Usage	31.56	26.99	22.44	**16.53**	18.73
Internet Traffic	10.55	10.16	10.87	16.70	**8.54**
ITDUK	4.91	4.84	3.94	11.89	**3.02**
Lake	2.81	1.78	1.87	4.01	**1.71**
Lynx	**48.78**	50.71	68.15	63.06	49.13
Milk	2.08	1.96	1.45	12.37	**1.20**
Mumps	29.11	31.33	27.53	**22.83**	23.38
Passenger	9.99	6.88	9.30	37.12	**5.12**
Pollution	21.33	**19.30**	47.09	57.82	30.15
Rainfall	**10.97**	12.68	13.31	11.19	11.21
Stock	**9.92**	11.81	69.99	113.52	34.08
Sun Spot	49.04	37.14	46.25	40.55	**34.16**
Tasty Cola	26.69	20.32	32.97	68.41	**18.15**
Temperature	8.37	4.81	4.59	4.88	**4.49**
Traffic	18.14	17.67	**14.58**	18.62	18.71
Unemployment	6.59	7.31	6.15	29.84	**4.66**

Table 7 Individual hypothesis test results using SMAPE indicating the superior (+), inferior (−) or equivalent (≈) method with respect to the proposed DEMS method

Time Series	ETS	ARIMA	MLP + LM	DBN [27]
Accidental Death	−	−	−	−
Acres Burned	−	−	−	≈
Car Sells	−	+	+	−
Chickenpox	−	−	−	−
Colorado River	+	+	−	−
Gas Usage	−	−	−	+
Internet Traffic	−	−	−	−
ITDUK	−	−	≈	−
Lake	−	−	−	−
Lynx	≈	≈	−	−
Milk	−	−	≈	−
Mumps	−	−	−	≈
Passenger	−	−	−	−
Pollution	+	+	−	−
Rainfall	≈	−	−	≈
Stock	+	+	−	−
Sun Spot	−	−	−	−
Tasty Cola	−	−	−	−
Temperature	−	−	≈	−
Traffic	+	+	+	≈
Unemployment	−	−	−	−

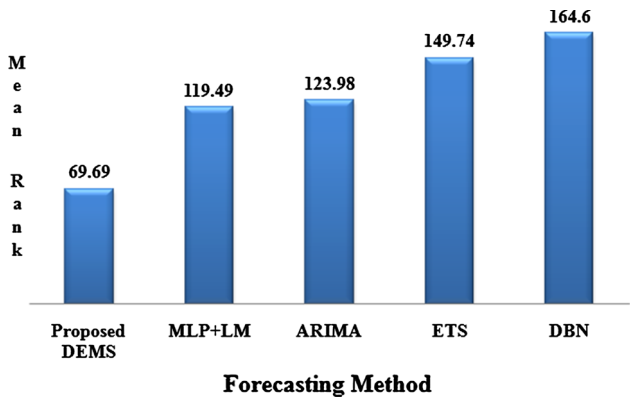

Fig. 4 Ranks of forecasting methods using SMAPE ($p = 0.0$ and Critical Distance = 1.3)

To determine the best method considering all the time series datasets, Friedman and Nemenyi hypothesis test [62, 63] is applied on the SMAPEs of all time series datasets. The test results presented in Fig. 4 indicates the superiority of the DEMS method as it has the lowest mean rank (69.69) among all the methods employed in this study. In addition, since the mean rank of DEMS method is lower than other methods with at least an absolute difference of critical distance (1.3), the DEMS method is statistically superior to other methods.

To make the proposed DEMS method consistently effective, several comparisons between different designs are performed. The design strategies are intended to evaluate the effect of treatment of trend component, sensitivity to different normalization techniques and effectiveness of the proposed ADE-ANNT training algorithm.

For this, the design configuration (i.e. the full design parameter) used in the proposed method using min–max normalization technique and ADE-ANNT training algorithm is treated as the baseline. To evaluate the effect of treatment of trend component, a detrending step by taking the first difference of time series data is added before the normalization step. Here, the first difference is considered because detrending data by making the first difference is the most suitable method for building the ANN model for nonlinear and stochastic time series [12]. To evaluate the sensitivity to different normalization techniques, min–max (as in Eq. 18), decimal scaling (as in Eq. 19) and vector (as in Eq. 20) normalization techniques are considered. To evaluate the sensitivity to different training algorithms, the proposed ADE-ANNT training algorithm, TLBO [64] for ANN training (TLBO-ANNT), DE-ANNT + [41] (with five trial vectors), DE-CRO-HONNT [43] and Levenberg–Marquardt (LM) algorithm are considered. Additionally, to evaluate the effectiveness of architecture selection using DEMS, results from MLP + LM (architecture selection using ACF & PACF and trained with LM) method is compared with DEMS + LM (architecture obtained using DEMS method and trained with LM algorithm). In the comparison of architecture selection methods, the DBN [27] method is not considered because of its inferior statistical performance than the MLP + LM method (as in Fig. 4). In addition, to avoid the bias of DEMS method towards ADE-ANNT training algorithm (since the architectures are selected using ADE-ANNT training algorithm), the LM algorithm is used to train the models in DEMS + LM method. By making these modifications, 50 independent simulations are conducted for each time series dataset. To evaluate the sensitivity of the DEMS method considering different strategies, a one-way analysis of variance (ANOVA) test is conducted on the obtained SMAPEs. The test results are presented in Table 8. It indicates that the architectures selected using the DEMS method are statistically insensitive (p-value > 0.05) to different normalization techniques. However, the chosen architectures using the DEMS method are sensitive to detrending using the first difference of the series (p-value < 0.05) and training algorithms (p-value < 0.05). Additionally, the performance of the ANN model is sensitive to the architecture selection method (p-value < 0.05).

To identify the effect of treating the trend component by using the first difference of the series, Friedman and Nemenyi hypothesis test [62, 63] is applied on the obtained SMAPEs, and the test results are presented in Fig. 5. It can be observed

Table 8 ANOVA results considering different strategies

	df	Sum of squares	Mean square	*F* value	*p* value
Direct versus detrending using first difference	1	3,550,883.05	3,550,883.05	1271.45	0.00
Normalization technique	2	274.32	137.16	0.70	0.35
Training algorithm considering SMAPE	4	104,266.00	26,066.50	51.78	0.00
MLP + LM versus DEMS + LM	1	14,635.13	14,635.1	29.06	0.00

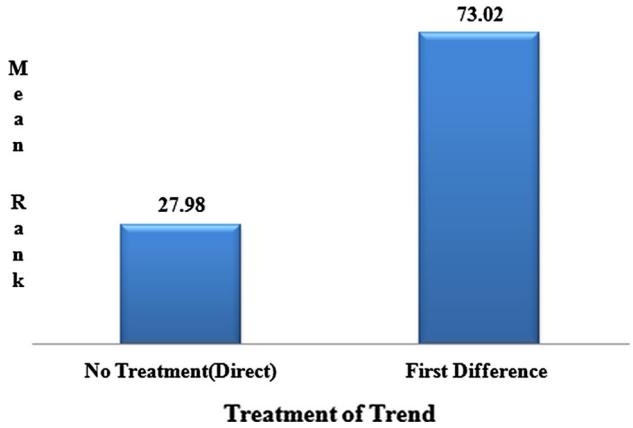

Fig. 5 Ranks of direct versus detrending using the first difference of the series using SMAPE ($p = 0.0$ and Critical Distance $= 0.4$)

that the proposed method performs statistically better when detrending using the first difference is avoided. Contrasting to the claim of the superior performance of ANN models with detrending using the first difference by Qi and Zhang [12], in this study, the performance of ANN model deteriorates significantly when detrending using the first difference is used. However, in Qi and Zhang [12] method, the architecture of the ANN model is chosen in an ad hoc manner. In addition, they suggested that the conclusions may change with different parameters. From simulation results, it is claimed that ANN with an appropriate architecture can effectively handle the trend component of the time series. And in such a case, detrending using the first difference deteriorates the performance of the model significantly.

To identify the best training algorithm among all the algorithms considered in this study, Friedman and Nemenyi hypothesis test [62, 63] is applied to the obtained SMAPEs. The test results shown in Fig. 6 indicate the clear superiority of the proposed ADE-ANNT training algorithm. The most competing algorithm with the proposed ADE-ANNT algorithm is LM. However, the absolute difference in mean rank (1.51) is more than the critical distance (CD) (1.3). Hence, their difference is statistically significant. Moreover, the memory consumption by the proposed ADE-ANNT algorithm is always less than or equal to that of DE-ANNT + [41], since, in ADE-ANNT algorithm, multiple trial vectors are generated only upon stagnation. In the worst-case (i.e. stagnation occurs at every generation, which is a rare case) both the algorithms have the same memory consumption. The DE-ANNT + [41] algorithm has lower memory consumption than the LM algorithm. Thus, the ADE-ANNT algorithm has lower memory consumption than LM algorithm as well. In the proposed wrapper approach-based DEMS method, a variety of ANN architectures need to be trained. Thus, considering the memory consumption and forecasting accuracy, ADE-ANNT algorithm is more suited in the proposed modelling scheme.

Table 8 indicates that the DEMS + LM and MLP + LM methods provide statistically different overall forecasting accuracy considering SMAPE measure. In both methods, the LM training algorithm is used but on different architectures.

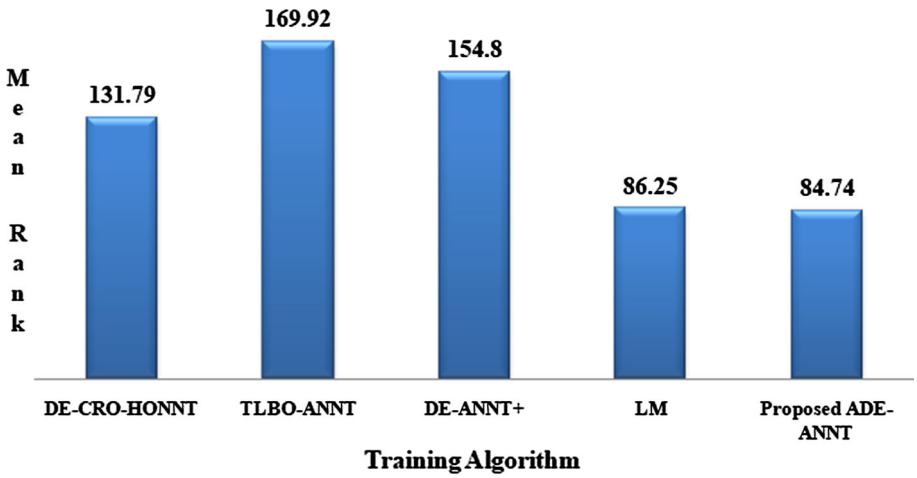

Fig. 6 Ranks of training algorithms using SMAPE ($p = 0.0$ and Critical Distance $= 1.3$)

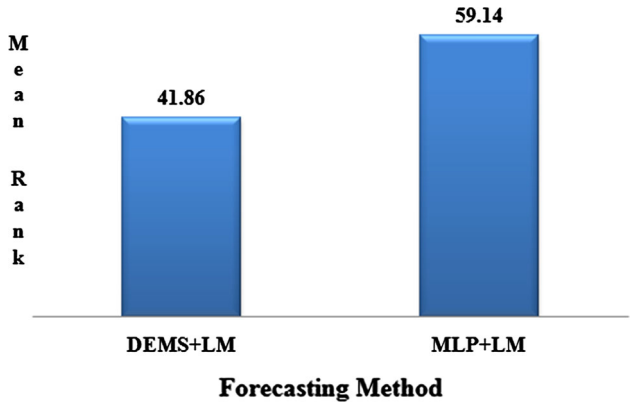

Fig. 7 Ranks of DEMS + LM and MLP + LM method using SMAPE ($p = 0.0$ and Critical Distance $= 0.4$)

In DEMS + LM method, the architectures are obtained using the proposed DEMS (as in Table 3) method whereas, in MLP + LM method, the architectures are obtained using ACF and PACF function of the series. To identify the better method, Friedman and Nemenyi hypothesis test [62, 63] is applied to the obtained SMAPEs, and the test results are shown in Fig. 7. It indicates the superiority of the proposed architecture selection DEMS method.

6 Conclusion

In this paper, a systematic investigation is made to explore the full potential of multilayer feedforward ANN, especially single hidden layer MLP in TSF. For this, a DE-based modelling scheme (DEMS) is developed to determine the most parsimonious architecture of MLP for a time series under study. In addition, an adaptive DE-based ANN training algorithm (ADE-ANNT) is proposed to determine the near-optimal weight set of ANN. Extensive statistical analyses on obtained results indicate the statistical superiority of the proposed DEMS method than the established statistical models ARIMA and ETS. The architectures selected using the proposed DEMS method provide statistically superior result than the MLP + LM and DBN [27] methods. The results also indicate the statistical superiority of ADE-ANNT algorithm when compared with LM, DE-ANNT + [41], DE-CRO-HONNT [43] and TLBO-ANNT [64]. In addition to statically superior SMAPE, the ADE-ANNT algorithm has lower memory consumption than DE-ANNT + [41] and the most popular LM algorithm.

The experiments reveal that the ANN models are quite capable of handling the trend component if the near-optimal architecture for a time series is obtained and in such a case detrending by taking the first difference of the series deteriorates the forecasting accuracy. Therefore, in the proposed DEMS method, treatment of trend component is avoided. It is also observed that in TSF, the ANN models are insensitive to different normalization techniques.

The proposed DEMS method is fully automatic and can be applied in various real-world time series with least human intervention. The DEMS method is computationally expensive because of the use of hierarchical DE algorithms for simultaneous optimization of architecture and weight set of ANN. However, it supports parallel implementation to reduce computational time. In addition, the proposed DEMS method can also be applied to optimize the architecture of other ANN models like FLANN, Pi-Sigma neural network, RBF neural network, etc.

References

1. Makridakis, S.G.; Steven, C.; Wheelwright, R.J.H.: Forecasting Methods and Applications. Wiley, New York (2008)
2. de Oliveira, J.F.L.; Ludermir, T.B.: A hybrid evolutionary decomposition system for time series forecasting. Neurocomputing **180**, 27–34 (2016). https://doi.org/10.1016/j.neucom.2015.07.113
3. Zhang, G.; Eddy Patuwo, B.; Hu, Y.M.: Forecasting with artificial neural networks: the state of the art. Int. J. Forecast. **14**, 35–62 (1998). https://doi.org/10.1016/S0169-2070(97)00044-7
4. Enders, W.: Applied Econometric Time Series. Wiley, New York (2014)
5. Wei, W.W.S.: Time Series Analysis. Oxford University Press, Oxford (2013)
6. Zhang, G.P.: Neural networks for time-series forecasting. In: Rozenberg, G., Bäck, T., Kok, J.N. (eds.) Handbook of Natural Computing, pp. 461–477. Springer, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-540-92910-9_14
7. Swanson, N.R.; White, H.: Forecasting economic time series using flexible versus fixed specification and linear versus nonlinear econometric models. Int. J. Forecast. **13**, 439–461 (1997). https://doi.org/10.1016/S0169-2070(97)00030-7
8. Hill, T.; O'Connor, M.; Remus, W.: Neural network models for time series forecasts. Manag. Sci. **42**, 1082–1092 (1996). https://doi.org/10.1287/mnsc.42.7.1082
9. Heravi, S.; Osborn, D.R.; Birchenhall, C.R.: Linear versus neural network forecasts for European industrial production series. Int. J. Forecast. **20**, 435–446 (2004). https://doi.org/10.1016/S0169-2070(03)00062-1
10. Callen, J.L.; Kwan, C.C.; Yip, P.C.; Yuan, Y.: Neural network forecasting of quarterly accounting earnings. Int. J. Forecast. **12**, 475–482 (1996). https://doi.org/10.1016/S0169-2070(96)00706-6
11. Panigrahi, S.; Behera, H.S.: Effect of normalization techniques on univariate time series forecasting using evolutionary higher order neural network. Int. J. Eng. Adv. Technol. **3**, 280–285 (2013)
12. Min, Q.; Zhang, G.P.: Trend time-series modeling and forecasting with neural networks. IEEE Trans. Neural Netw. **19**, 808–816 (2008). https://doi.org/10.1109/TNN.2007.912308
13. Crone, S.F.; Hibon, M.; Nikolopoulos, K.: Advances in forecasting with neural networks? Empirical evidence from the NN3 competition on time series prediction. Int. J. Forecast. **27**, 635–660 (2011). https://doi.org/10.1016/j.ijforecast.2011.04.001
14. Panigrahi, S.; Karali, Y.; Behera, S.H.: Time series forecasting using evolutionary neural network. Int. J. Comput. Appl. **75**, 13–17 (2013). https://doi.org/10.5120/13146-0553
15. Yang, Z.; Mourshed, M.; Liu, K.; Xu, X.; Feng, S.: A novel competitive swarm optimized RBF neural network model for short-

term solar power generation forecasting. Neurocomputing. **397**, 415–421 (2020). https://doi.org/10.1016/j.neucom.2019.09.110
16. Xiangxue, W.; Lunhui, X.; Kaixun, C.: Data-driven short-term forecasting for urban road network traffic based on data processing and LSTM-RNN. Arab. J. Sci. Eng. **44**, 3043–3060 (2019). https://doi.org/10.1007/s13369-018-3390-0
17. Poorzaker Arabani, S.; Ebrahimpour Komleh, H.: The improvement of forecasting ATMS cash demand of Iran banking network using convolutional neural network. Arab. J. Sci. Eng. **44**, 3733–3743 (2019). https://doi.org/10.1007/s13369-018-3647-7
18. Panigrahi, S.; Behera, H.S.: Nonlinear time series forecasting using a novel self-adaptive TLBO-MFLANN model. Int. J. Comput. Intell. Stud. **8**, 4 (2019). https://doi.org/10.1504/IJCISTUDIES.2019.10019170
19. Minku, F.L.; Ludermir, T.B.: Clustering and co-evolution to construct neural network ensembles: an experimental study. Neural Netw. **21**, 1363–1379 (2008). https://doi.org/10.1016/j.neunet.2008.02.001
20. Stepniewski, S.W.; Keane, A.J.: Pruning backpropagation neural networks using modern stochastic optimization techniques. Neural Comput. Appl. **5**, 76–98 (1997). https://doi.org/10.1007/BF01501173
21. Miller, G.F.; Todd, P.M.; Hegde, S.U.: Designing Neural Networks using Genetic Algorithms. In: Proceedings of the 3rd International Conference on Genetic Algorithms, George Mason University, Fairfax, Virginia, USA, pp. 379–384 (1989)
22. Kitano, H.: Designing neural networks using genetic algorithms with graph generation system. Complex Syst. **4**, 461–476 (1990)
23. Gruau, F.: Genetic synthesis of Boolean neural networks with a cell rewriting developmental process. In: [Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks, pp. 55–74. IEEE Computer Society Press (1992)
24. Donate, J.P.; Li, X.; Sánchez, G.G.; de Miguel, A.S.: Time series forecasting by evolving artificial neural networks with genetic algorithms, differential evolution and estimation of distribution algorithm. Neural Comput. Appl. **22**, 11–20 (2013). https://doi.org/10.1007/s00521-011-0741-0
25. Saboo, A., Sharma, A., Dash, T.: GASOM: Genetic Algorithm Assisted Architecture Learning in Self Organizing Maps. Presented at the (2017)
26. Sun, Y.; Xue, B.; Zhang, M.; Yen, G.G.; Lv, J.: Automatically designing CNN architectures using the genetic algorithm for image classification. IEEE Trans. Cybern. **10**, 1–15 (2020). https://doi.org/10.1109/TCYB.2020.2983860
27. Kuremoto, T.; Kimura, S.; Kobayashi, K.; Obayashi, M.: Time series forecasting using a deep belief network with restricted Boltzmann machines. Neurocomputing **137**, 47–56 (2014). https://doi.org/10.1016/j.neucom.2013.03.047
28. Ewees, A.A.; Elaziz, M.A.; Alameer, Z.; Ye, H.; Jianhua, Z.: Improving multilayer perceptron neural network using chaotic grasshopper optimization algorithm to forecast iron ore price volatility. Resour. Policy **65**, 101555 (2020). https://doi.org/10.1016/j.resourpol.2019.101555
29. Baffes, P.T., Zelle, J.M.: Growing layers of perceptrons: introducing the Extentron algorithm. In: [Proceedings 1992] IJCNN International Joint Conference on Neural Networks. pp. 392–397. IEEE
30. Gutierrez, G.; Sanchis, A.; Isasi, P.; Molina, J.M.; Galvan, I.M.: Non-direct encoding method based on cellular automata to design neural network architectures. Comput. Inf. **24**, 225–247 (2005)
31. Yao, Xin; Liu, Yong: Making use of population information in evolutionary artificial neural networks. IEEE Trans. Syst. Man Cybern. Part B **28**, 417–425 (1998). https://doi.org/10.1109/3477.678637
32. Peralta, J., Gutierrez, G., Sanchis, A.: ADANN: Automatic design of artificial neural networks. In: Proceedings of the 2008 GECCO Conference Companion on Genetic and Evolutionary Computation—GECCO'08, pp. 1863–1870. ACM Press, New York (2008)
33. Peralta, J., Gutierrez, G., Sanchis, A.: Time series forecasting by evolving artificial neural networks using genetic algorithms and estimation of distribution algorithms. In: The 2010 International Joint Conference on Neural Networks (IJCNN), pp. 1–8. IEEE (2010)
34. Xu, X., Li, Y.: Comparison between particle swarm optimization, differential evolution and multi-parents crossover. In: 2007 International Conference on Computational Intelligence and Security (CIS 2007), pp. 124–127. IEEE (2007)
35. Ash, T.: Dynamic node creation in backpropagation networks. In: International Joint Conference on Neural Networks, vol. 2, p 623. IEEE (1989)
36. Balkin, S.D.; Ord, J.K.: Automatic neural network modeling for univariate time series. Int. J. Forecast. **16**, 509–515 (2000). https://doi.org/10.1016/S0169-2070(00)00072-8
37. Makridakis, S.; Spiliotis, E.; Assimakopoulos, V.: Statistical and machine learning forecasting methods: concerns and ways forward. PLoS ONE **13**, e0194889 (2018). https://doi.org/10.1371/journal.pone.0194889
38. Choi, T.J., Cheong, Y.-G., Ahn, CW: A Performance Comparison of Crossover Variations in Differential Evolution for Training Multilayer Perceptron Neural Networks. Presented at the (2018)
39. Pattanayak, R.M., Behera, H.S., Panigrahi, S.: A Novel Hybrid Differential Evolution-PSNN for Fuzzy Time Series Forecasting. Presented at the (2020)
40. Slowik, A., Bialko, M.: Training of artificial neural networks using differential evolution algorithm. In: 2008 Conference on Human System Interactions, pp. 60–65. IEEE (2008)
41. Slowik, A.: Application of an adaptive differential evolution algorithm with multiple trial vectors to artificial neural network training. IEEE Trans. Ind. Electron. **58**, 3160–3167 (2011). https://doi.org/10.1109/TIE.2010.2062474
42. Sahu, K.K.; Panigrahi, S.; Behera, H.S.: A novel chemical reaction optimization algorithm for higher order neural network training. J. Theor. Appl. Inf. Technol. **53**, 402–409 (2013)
43. Panigrahi, S.: A novel hybrid chemical reaction optimization algorithm with adaptive differential evolution mutation strategies for higher order neural network training. Int. Arab J. Inf. Technol. **14**, 18–25 (2017)
44. Karali, Y.; Panigrahi, S.: Behera, HS: A novel differential evolution based algorithm for higher order neural network training. J. Theor. Appl. Inf. Technol. **56**, 355–361 (2013)
45. Storn, R.; Price, K.: Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. **11**, 341–359 (1997). https://doi.org/10.1023/A:1008202821328
46. Awad, N.H., Ali, M.Z., Suganthan, P.N., Reynolds, R.G.: An ensemble sinusoidal parameter adaptation incorporated with L-SHADE for solving CEC2014 benchmark problems. In: 2016 IEEE Congress on Evolutionary Computation (CEC), pp. 2958–2965. IEEE (2016)
47. Awad, N.H., Ali, M.Z., Suganthan, P.N.: Ensemble sinusoidal differential covariance matrix adaptation with Euclidean neighborhood for solving CEC2017 benchmark problems. In: 2017 IEEE Congress on Evolutionary Computation (CEC). pp. 372–379. IEEE (2017)
48. Akhmedova, S., Stanovov, V., Semenkin, E.: LSHADE algorithm with a rank-based selective pressure strategy for the circular antenna array design problem. In: Proceedings of the 15th International Conference on Informatics in Control, Automation and Robotics. pp. 159–165. SCITEPRESS—Science and Technology Publications (2018)
49. Pant, M.B.; Zaheer, H.; Garcia-Hernandez, L.; Abraham, A.: Differential evolution: a review of more than two decades of research.

Eng. Appl. Artif. Intell. **90**, 103479 (2020). https://doi.org/10.1016/j.engappai.2020.103479
50. Das, S.; Mullick, S.S.; Suganthan, P.N.: Recent advances in differential evolution—an updated survey. Swarm Evol. Comput. **27**, 1–30 (2016). https://doi.org/10.1016/j.swevo.2016.01.004
51. Price, K.; Storn, R.M.; Lampinen, J.A.: Differential Evolution: A practical approach to global optimization. Springer, Berlin (2005)
52. Price, K.V.: An introduction to differential evolution. In: Corne, D., Dorigo, M., Glover, F., Dasgupta, D., Moscato, P., Poli, R., Price, K.V. (eds.) New Ideas in Optimization, pp. 79–108 (1999)
53. Grippo, L.: A class of unconstrained minimization methods for neural network training. Optim. Methods Softw. **4**, 135–150 (1994). https://doi.org/10.1080/10556789408805583
54. Jacobs, R.A.: Increased rates of convergence through learning rate adaptation. Neural Networks. **1**, 295–307 (1988). https://doi.org/10.1016/0893-6080(88)90003-2
55. Gori, M.; Tesi, A.: On the problem of local minima in backpropagation. IEEE Trans. Pattern Anal. Mach. Intell. **14**, 76–86 (1992). https://doi.org/10.1109/34.107014
56. Mezura-Montes, E., Velázquez-Reyes, J., Coello Coello, C.A.: A comparative study of differential evolution variants for global optimization. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation—GECCO'06, p. 485. ACM Press, New York (2006)
57. Weber, M.; Tirronen, V.; Neri, F.: Scale factor inheritance mechanism in distributed differential evolution. Soft. Comput. **14**, 1187–1207 (2010). https://doi.org/10.1007/s00500-009-0510-5
58. Islam, S.M.; Das, S.; Ghosh, S.; Roy, S.; Suganthan, P.N.: An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization. IEEE Trans. Syst. Man Cybern. Part B **42**, 482–500 (2012). https://doi.org/10.1109/TSMCB.2011.2167966
59. Hyndman, R.J.; Khandakar, Y.: Automatic time series forecasting: the forecast package for R. J. Stat. Softw. (2008). https://doi.org/10.18637/jss.v027.i03
60. Panigrahi, S.; Behera, H.S.: A hybrid ETS–ANN model for time series forecasting. Eng. Appl. Artif. Intell. **66**, 49–59 (2017). https://doi.org/10.1016/j.engappai.2017.07.007
61. Hyndman, R.: YY: tsdl: Time Series Data Library. v0.1.0., https://pkg.yangzhuoranyang.com/tsdl/articles/tsdl.html. Accessed 2 June 2019
62. Hollander, M.; Wolfe, A.D.; Chicken, E.: Nonparametric Statistical Methods. Wiley, New York (2015)
63. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (2006)
64. Rao, R.V.; Savsani, V.J.; Vakharia, D.P.: Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems. Comput. Des. **43**, 303–315 (2011). https://doi.org/10.1016/j.cad.2010.12.015