Nonlinear System Identification using Opposition Based Learning Differential Evolution and Neural Network Techniques

Bidyadhar Subudhi, *Senior Member, IEEE*, and Debashisha Jena
Center for Industrial Electronics and Robotics, Dept. of Electrical Engineering
National Institute of Technology, Rourkela, India-769008
E-mail: bidyadharnitrkl@gmail.com

Abstract

The slow convergence and local minima problems associated with neural networks (NN) used for non-linear system identification have been resolved by evolutionary techniques such as differential evolution (DE) combined with Levenberg Marquardt (LM) algorithm. In this work the authors attempted further to employ an opposition based learning in DE, known as opposition based differential evolution (OBDE) for training neural networks in order to achieve better convergence of DE. The proposed OBDE together with DE and neuro-fuzzy (NF) approaches to non-linear system identification has been applied for identification of two non-linear system benchmark problems. Results presented clearly demonstrate that the OBDE-NN method of non-linear system identification provides excellent identification performance in comparison to both the DE and NF approaches.

Keywords: *Back Propagation, Differential evolution, Evolutionary computation, Nonlinear System Identification, Neuro-fuzzy, Opposition based differential evolution*

1. Introduction

System identification is widely used in a number of applications such as in control system [1], communication [2], signal processing [3], chemical process control [4] and biological processes [5] etc. In the strict sense, all the real-world problems are nonlinear in nature. It is pertinent that there is less computational difficulty encountered for identification of a linear system. However, in the nonlinear system identification, the scenario is not straightforward. There are some classical parameterized models such as Voltera series [6], Winner-Hammerstein model [7] and polynomial identification methods [8, 9] which provide a reasonable degree of accuracy but these methods involve computational complexities. Subsequently, neural networks have been extensively used for modeling complex system dynamics in a systematic approach especially those which are hard to be described mathematically. It has been proved that any continuous function can be approximated by a feed-forward neural network trained with back-propagation learning algorithm to a reasonable degree of accuracy [10]. This function approximation property can be exploited to model a number of complex systems. Narendra and Parthasarathy [11] have shown that multilayer neural networks trained with standard back-propagation algorithm can be used effectively for the identification of nonlinear dynamical systems. Subsequently the local minima problems encountered in these have been overcome to a reasonable extent exploiting evolutionary approaches to train neural networks and these networks are known as evolving neural networks.

Research on designing evolutionary neural networks has progressed significantly by combing several variants of evolutionary algorithms such as genetic algorithm (GA), genetic programming (GP) and particle swarm optimization (PSO) with neural networks [12]. Differential evolution is an effective, efficient and robust optimization method capable of handling nonlinear and multimodal objective functions. The advantages of DE are many such as it is simple and has compact structure which uses a stochastic direct search approach and utilizes common concepts of EAs. Furthermore, DE uses few easily chosen parameters and provides excellent results for a wide set of benchmark and real-world problems [13, 14]. DE+LM+NN nonlinear system identification has been developed earlier by the authors [15] where, DE and LM have been used in a combined framework to train a neural network for achieving faster convergence of neural network weight optimization. To improve the performance of DE trained NN further, the authors propose in this paper a new training method of NN by using the concept of opposition based learning (OBL) in DE. This new nonlinear system identification scheme is called OBDE-NN.

The concept of *opposition-based learning* (OBL) was first introduced by Tizhoosh [17]. It usually applied to accelerate reinforcement learning [18] and back-propagation learning in neural networks [19]. The main idea behind OBL is the simultaneous consideration of an estimate and its corresponding opposite estimate (i.e., guess and opposite guess) in order to achieve a better approximation for the current candidate solution. In this paper, OBL has been exploited to accelerate the convergence rate of DE. Hence, the proposed approach is called opposition-based differential evolution (OBDE). OBDE uses opposite numbers during population initialization and also for generating new populations during the evolutionary process. Here opposite numbers have been utilized to speed up the convergence rate of an optimization algorithm. It may be noted that selection of solutions based on purely random initialization gives rise to problem of visiting or even revisiting unproductive search regions. The chance of the above problem occurring can be resolved by using opposite numbers instead of using purely random numbers for initialization of populations. In fact, a mathematical proof has already been proposed to show that, opposite numbers are more likely to be closer to the optimal solution than purely random ones [20]. In [21], the usefulness of opposite numbers is investigated by replacing them with random numbers and it is applied for population initialization and generation jumping for different versions of DE.

However, a little work has been reported on applying OBDE to system identification and its use in training neural network for nonlinear system identification. Therefore, we present in this paper an opposition based differential evolution for training a feed-forward neural network used as a nonlinear system identifier.

Nonlinear system as considered in [22, 23] has been chosen in this work for demonstrating the effectiveness of the proposed OBDE-NN system identification approach compared to DE-NN and NF approach. In this work, an opposition based differential evolution method combined with LM has been applied as a global optimization method for training a feed-forward neural network. In the proposed scheme, the OBDE is used to train the neural network that is chosen as a suitable candidate for nonlinear system identification. After observing the trends of training towards minimum through OBDE, the network is then trained by LM. The role of the OBDE here is to approach towards global minimum point and then LM is used to move forward achieving fast convergence. According to the proposed algorithm, after reaching the basin of global minimum, the algorithm is switched from global search of the evolutionary algorithm (OBDE) to local search, LM. In opposition based differential evolution, at the moment of starting, the differential term is very high. As the solution approaches to global minimum, the differential term automatically changes to a low value. So during the initial period, the convergence speed is faster and the search space is very large but in latter stages nearer to the optimum as the differential term is small, the algorithm becomes slower which will take more time to converge. As LM is a gradient based algorithm it can be exploited to increase the convergence speed of the neural network training algorithm for reaching the global minimum.

The main contributions of the paper are as follows:

- Designing of OBDE-NN scheme to accelerate the convergence of DE used for non-linear system identification.
- LM has been integrated with the OBDE to enable faster search process for achieving faster convergence.
- The identification performance of the proposed OBDE-NN scheme has been compared with the DE-NN and NF approaches to nonlinear system identification and found to be better than the later.

The paper is organized as follows. Section 2 reviews the NNarchitecture. Section 3 presents the neuro-fuzzy technique in system identification followed by a brief review on the differential evolution technique in section 4. Subsequently in section 5 and 6 we describe the proposed differential evolution combined with neural network approach to nonlinear system identification. Finally the paper discusses the results of all the aforesaid techniques in section 7 to arrive at conclusions.

2. The NN Architecture

In case of MLPNN architecture, one hidden layer is sufficient to guarantee the universal approximation feature. Fig 2 illustrates this kind of network.

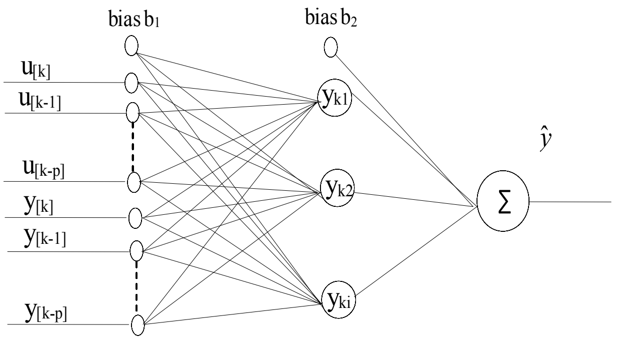


Fig.1 NN identifier with external dynamics

The input vector dimension depends on the system order. The following two equations describe the signal propagation from input layer to output layer.

$$y_h = \psi\,(W_1 v + b_1) \quad (1)$$

$$\hat{y} = \psi\,(W_2^T y_h + b_2) \quad (2)$$

where y_h is the output from the hidden units, W_1 is the weight matrix of the first layer, W_2 is the weight of the output layer v is the regression vector of inputs, b_1 is the bias for hidden units, b_2 is the bias for output units and ψ is the nonlinear function which we have taken as a sigmoid function as these are easier to train compared to many other activation functions such as hard limiter etc. Moreover, with sigmoid units, a small change in the weights will usually produce a change in the outputs, which makes it possible to tell whether that change in the weights is good or bad.

3. A Review on Neuro-Fuzzy Technique for Nonlinear System Identification

Neuro-fuzzy [NF] technique is a very popular system modeling paradigm used for obtaining models of many complex nonlinear systems. Although this technique is described in many books and papers [24, 25] but we describe it here in brief for completeness. Fig.3 shows a well known neuro-fuzzy modeling structure [24, 25]. In neuro-fuzzy modeling technique (Fig.2) there are five layers with the following characteristics.

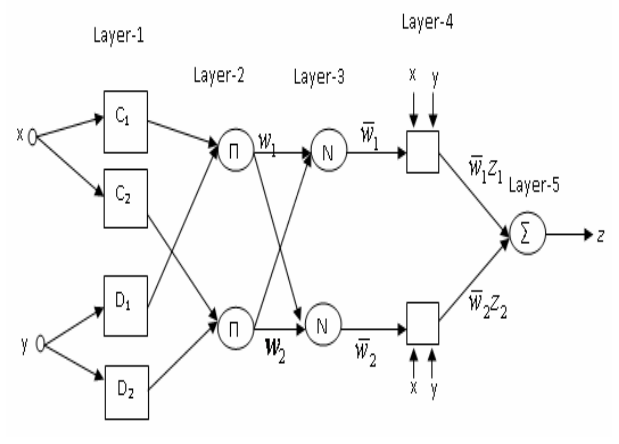


Fig.2 Neuro-fuzzy technique

In layer 1, each adaptive node *i has* node function defined as:

$$O_{1,i} = \mu_{C_i i}(x) \qquad \text{for } i = 1, 2, \text{or}$$

$$O_{1,i} = \mu_{D_i i-2}(y) \qquad \text{for } i = 3, 4$$

where (*x* or *y*) is the input to node *i* and C_i (or D_{i-2}) is a linguistic label associated with this node. Here, the node function of the *i*-th node is considered as a generalized bell shaped membership function given by

$$\mu_{C_i}(x) = \frac{1}{1 + \left| \frac{x - c_i}{a_i} \right|^{2b_i}} \qquad (3)$$

where *{a_i, b_i, c_i}* are the parameter set that changes the shapes of the membership function. Parameters in this layer are the *premise parameter*s.

In layer 2, represents a fixed node and each of the node calculates the firing strength of a fuzzy as follows

$$O_{2,i} = w_i = \mu_{C_i}(x).\mu_{D_i}(y), \qquad i = 1, 2 \qquad (4)$$

In layer 3, the fixed node *i* calculates the ratio of the *i*-th rule's firing strength (w_i) to the total of all firing strength as:

$$O_{3,i} = \overline{w}_i = \frac{w_i}{w_1 + w_2}, \qquad i = 1,2 \qquad (5)$$

The output of layer-3 given in equation (5) denotes the *normalized firing strength* ($\overline{w}_i$)

In layer 4, the adaptive node *I,* computes the contribution of *i*-th rule towards the overall output with the following node function:

$$O_{4,i} = \overline{w}_i z_i = \overline{w}_i (p_i x + q_i y + r_i) \qquad (6)$$

where (p_i, q_i, r_i) are Known as the consequent parameters of the rule.

The single fixed node in layer 5 computes the overall output as the summation of signal contributions from each rule:

$$O_{5,i} = \sum_i \overline{w}_i z_i \frac{\sum_i w_i z_i}{\sum_i z_i} \qquad (7)$$

The basic learning rule adopted in this neuro-fuzzy modeling technique is the back propagation gradient descent, which calculates error signals (the derivative of the squared error with respect to each node's output) recursively from the output layer backward to the input nodes. This learning rule is exactly the same as the back propagation learning rule used in the feed-forward neural networks.

The overall output *z* can be expressed as linear combinations of the consequent parameters:

$$z = \overline{w}_i z_1 + \overline{w}_i z_2 = (\overline{w}_i x) p_1 + (\overline{w}_i y) q_1 (\overline{w}_i) r_1$$
$$+ (\overline{w}_i x) p_2 + (\overline{w}_i y) q_2 + (\overline{w}_i) r_2 \qquad (8)$$

4. Differential Evolution Technique

The differential evolution technique is capable of handling non-differentiable, non-linear and multimodal objective functions [13, 14]. It has been used to train neural networks having real and constrained integer weights. In a population of potential solutions within a d-dimensional search space, a fixed number of vectors are randomly initialized, then evolved over time to explore the search space and to locate the minima of the objective function.

At each generation new vectors are generated by the combination of vectors randomly chosen from the current population (mutation). The out coming vectors are then mixed with a predetermined target vector. This operation is called recombination and produces the trial vector. Finally, the trial vector is accepted for the next generation if and only if it yields a reduction in the value of the objective function. This last operator is referred to as a selection. There are many different variants of DE [13], the variants are

- **DE/best/1/exp**
- **DE/rand/1/exp**
- **DE/rand-to-best/1/exp**
- **DE/best/2/exp**
- **DE/rand/2/exp**

Now we explain the working steps involved in employing a DE cycle.

Step 1: Parameter setup

Choose the parameters of population size, the boundary constraints of optimization variables, the mutation factor *(F)*, the crossover rate *(C)*, and the stopping criterion of the maximum number of generations *(g).*

Step 2: Initialization of the population

Set generation *g=0*. Initialize a population of *i*=1, *P* individuals (real-valued *d*-dimensional solution vectors) with random values generated according to a uniform probability distribution in the *d* dimensional problem space. These initial values are chosen randomly within user's defined bounds.

Step 3: Evaluation of the population

Evaluate the fitness value of each individual of the population. If the fitness satisfies predefined criteria save the result and stop, otherwise go to step 4.

Step 4: Mutation operation (or differential operation)

Mutation is an operation that adds a vector differential to a population vector of individuals. For each target vector $x_{i,g}$ a mutant vector is produced using the following relation,

$$v_{i,g} = x_{r_1,g} + F(x_{r_2,g} - x_{r_3,g}) \quad (9)$$

In Eqn. (10), F is the mutation factor, which provides the amplification to the difference between two individuals $(x_{r_2,g} - x_{r_3,g})$ so as to avoid search stagnation and it is usually taken in the range of [0, 1], where $i, r_1, r_2, r_3 \in \{1, 2, ...P\}$ are randomly chosen numbers but they must be different from each other. P is the number of population.

Step 5: Recombination operation

Following the mutation operation, recombination is applied to the population. Recombination is employed to generate a trial vector by replacing certain parameters of the target vector with the corresponding parameters of a randomly generated donor (mutant) vector. There are two methods of recombination in DE, namely, binomial recombination and exponential recombination.

In binomial recombination, a series of binomial experiments are conducted to determine which parent contributes which parameter to the offspring. Each experiment is mediated by a crossover constant, *C*, ($0 \le C \le 1$). Starting at a randomly selected parameter, the source of each parameter is determined by comparing *C* to a uniformly distributed random number from the interval [0, 1) which indicates the value of *C* can exceed the value 1. If the random number is greater than *C*, the offspring gets its parameter from the target individual; otherwise, the parameter comes from the mutant individual. In exponential recombination, a single contiguous block of parameters of random size and location is copied from the mutant individual to a copy of the target individual to produce an offspring. A vector of solutions are selected randomly from the mutant individuals when $rand_j$ ($rand_j \in [0,1]$, is a random number) is less than *C*.

$$t_{j,i,g} = \begin{cases} v_{j,i,g} & \text{if } (rand_j \le C) \text{ or } j = j_{rand} \\ x_{j,i,g} & \text{otherwise} \end{cases} \quad (10)$$

$j = 1, 2 ......d$, where d is the number of parameters to be optimized.

Step 6: Selection operation

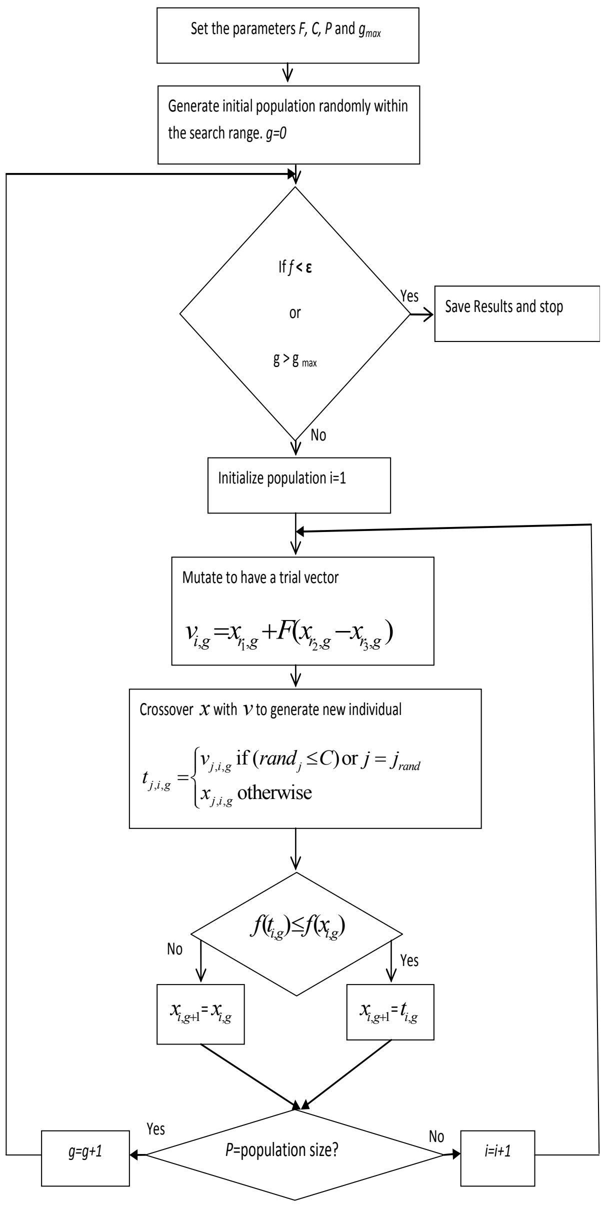
Selection is the procedure of producing better offspring. If the trial vector, $t_{i,g}$ has an equal or lower value than that of its target vector, $x_{i,g}$ it replaces the target vector in the next generation; otherwise the target retains its place in the population for at least one more generation.

$$x_{i,g+1} = \begin{cases} t_{i,g}, & if\ f(t_{i,g}) \le f(x_{i,g}) \\ x_{i,g}, & otherwise \end{cases} \quad (11)$$

Once new population is installed, the process of mutation, recombination and selection is replaced until the optimum is located, or a specified termination criterion is satisfied, e.g., the number of generations reaches a predefined maximum $g_{\max}$.

At each generation, new vectors are generated by the combination of vectors randomly chosen from the current population (mutation). The upcoming vectors are then mixed with a predetermined target vector. This operation is called recombination and produces the trial vector. Finally, the trial vector is accepted for the next generation if it yields a reduction in the value of the objective function. This last operator is referred to as a selection. The most commonly used method for validation is to utilize the root mean-squared error (RMSE) between the actual output *y(n)* of the system and the predicted output $\hat{y}(n)$. In this work we have taken the cost function as root mean squared error (RMSE)

i.e. $\mathbf{E} = \sqrt{\frac{1}{N}\sum_{k=1}^{N}[\mathbf{y} - f(\mathbf{x}, \mathbf{w})]^2}$, where *N* is the number of data considered. The block diagram and pseudo code for DE are given in Fg.4 and Fig.5 respectively.



Set the parameters F, C, P and g_{max}
Generate initial population randomly within the search range. $g=0$
If $f<\varepsilon$ or $g>g_{max}$
Yes
Save Results and stop
No
Initialize population i=1
Mutate to have a trial vector $v_{i,g}=x_{r_1,g}+F(x_{r_2,g}-x_{r_3,g})$
Crossover x with v to generate new individual $t_{j,i,g}=\begin{cases}v_{j,i,g} & \text{if } (rand_j\le C) \text{ or } j=j_{rand}\\ x_{j,i,g} & \text{otherwise}\end{cases}$
$f(t_{i,g})\le f(x_{i,g})$
No
Yes
$x_{i,g+1}=x_{i,g}$
$x_{i,g+1}=t_{i,g}$
Yes
P=population size?
No
$g=g+1$
$i=i+1$

Fig. 3 Block diagram for DE algorithm

5. Opposition based Differential Evolution

Generally in all evolutionary algorithm approaches, a uniform random guess is considered for initial population. As the search process progresses, the solutions obtained move towards the optimal value. It may be noted that in the case of random guess, when the distance between the initial guess and final optimal solution is more the algorithm takes more time to reach the optimal value and vice-versa. However Opposition based learning improves the chance of starting with better initial population by checking the opposite solutions. As the initial guess is always random, by looking into the opposite direction and starting with the closer of the two guesses (as judged by the fitness value) the algorithm accelerates towards convergence. The same approach can be applied not only to initial solutions but also applied continuously to each solution in the current population. However, before concentrating on OBL, we need to define the concept of opposite numbers [17].

- ***Definition of opposite number:***

Let $x \in [a,b]$ be a real number. The opposite number is $\tilde{x}$ which is defined by $\tilde{x} = a + b - x$

- *Definition of opposite point:*

Let $p = (x_1, x_2......x_d)$ be a point in the D dimensional space where $x_1, x_2......x_d \in R$ and $x_i \in [a_i, b_i]$. The opposite point $\tilde{p} = (\tilde{x}_1, \tilde{x}_2......\tilde{x}_d)$ where $\tilde{x}_i = a_i + b_i - x_i$

- ***Opposition based optimization:***

Let $p = (x_1, x_2......x_d)$ be a point in the D dimensional i.e. a candidate solution. Assume $f(.)$ is the fitness function which is used to measure the candidate's fitness. According to the definition of the opposite point $\tilde{p} = (\tilde{x}_1, \tilde{x}_2......\tilde{x}_d)$ is the opposite of $p = (x_1, x_2......x_d)$. Now if $f(\tilde{p}) \ge f(p)$ then point p can be replaced by $\tilde{p}$ otherwise we will continue with p. Hence the point and its opposite point are evaluated simultaneously in order to continue with the fit one.

Development of Proposed OBDE Algorithm

Similar to all other variants of evolutionary computing techniques DE also employs two main steps namely population initialization and application of evolutionary operations (mutation, crossover and selection) for producing new generations of populations. We will enhance these two steps using the OBL scheme. The original DE is chosen as a parent algorithm and the proposed opposition-based ideas are embedded in DE to accelerate its convergence speed. In OBDE the first step follows an opposition based population initialization where as the second step considers an opposition based generation jumping. Corresponding block diagram and pseudo code for the proposed approach (OBDE) is given in Fig. 6 and Fig.7 respectively.

Opposition-Based Population Initialization

In all population based algorithms owing to the absence of a-priori knowledge on solutions, a set of candidate solutions (populations) are usually initialized as random numbers. But this evolution process takes more time as said earlier. Hence by utilizing OBL, we can obtain fitter starting candidate solutions even when there is no *a priori* knowledge about the solution(s). To implement OBL based population initialization we create a population *pop* and then applying the formula given in equation (14), we get the opposite population.

$$opop_{i,j} = a_j + b_j - pop_{i,j} \qquad (12)$$
$$i = 1, 2, ...P \qquad j = 1, 2, ...D$$

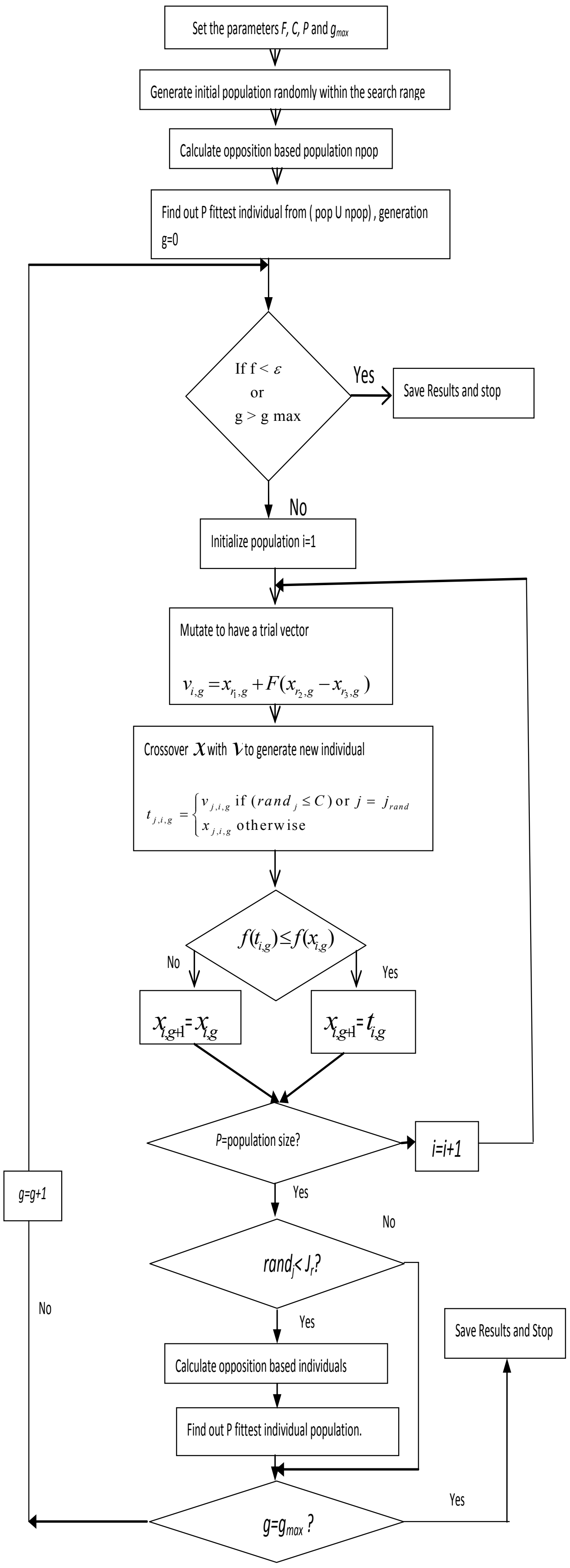
where $pop_{i,j}$ and $opop_{i,j}$ denote the j^{th} variable of the ith vector of the population and opposite population respectively. Select P fittest individual from the total population of *pop* and *opop* i.e. (*pop* U *opop*) as initial population, where U stands for union.

Opposition-Based Generation Jumping

Generation jumping decides whether the current population will follow all the steps as in opposition based population initialization procedure or directly jumps to the next generation. This can be implemented by using a jumping probability (jumping rate), J_r. If the jumping rate exceeds a random number that is generated in the current generation then the population will jump to the next generation directly as shown in Fig.6. Thus, the computation can be saved and algorithm becomes faster. Based on this jumping rate, after generating new population by mutation, crossover, and selection, the opposite population is calculated and P fittest individuals are selected from the union of the current population and the opposite population. Instead of using variables' predefined interval boundaries generation jumping calculates the opposite of each variable based on minimum and maximum values of that variable in the current population which is given in equation (13).

$$onpop_{i,j} = \min\,(npop_j) + \max\,(npop_j) - npop_{i,j} \quad (13)$$

Thus, generation jumping calculates the opposite population dynamically.



Set the parameters F, C, P and g_{max}
Generate initial population randomly within the search range
Calculate opposition based population npop
Find out P fittest individual from (pop U npop) , generation g=0
If f < ε or g > g max
Yes
Save Results and stop
No
Initialize population i=1
Mutate to have a trial vector
$v_{i,g} = x_{r_1,g} + F(x_{r_2,g} - x_{r_3,g})$
Crossover x with v to generate new individual
$t_{j,i,g} = \begin{cases} v_{j,i,g} & \text{if } (rand_j \le C) \text{ or } j = j_{rand} \\ x_{j,i,g} & \text{otherwise} \end{cases}$
$f(t_{i,g}) \le f(x_{i,g})$
No
Yes
$x_{i,g+1} = x_{i,g}$
$x_{i,g+1} = t_{i,g}$
P=population size?
i=i+1
g=g+1
Yes
No
$rand_j < J_r$?
No
Yes
Save Results and Stop
Calculate opposition based individuals
Find out P fittest individual population.
Yes
$g = g_{max}$?

Fig. 4 Block diagram for DE algorithm

6. A Combined OBDE-NN Approach to System Identification

Here, we describe how an OBDE is applied for training a neural network in the framework of system identification (see pseudo code in Fig.5). According to the proposed algorithm, the value of the cost function after reaching a particular value of ε , the algorithm is switched from global search such of the evolutionary algorithm (OBDE) to local search, LM. In opposition based differential evolution, at the moment of starting, the differential term is very high. As the solution approaches to global minimum the differential term automatically changes to a low value. So at initial period the convergence speed is faster and search space is very large but in latter stages nearer to the optimum due to small differential term the algorithm becomes slower which will take more time to converge. As LM is a gradient based algorithm at that point the role of LM is to increase the convergence speed for reaching the global minimum. OBDE can be applied to global searches within the weight space of a typical feed-forward neural network. Output of a feed-forward neural network is a function of synaptic weights $\mathbf{w}$ and input values $\mathbf{x}$, i.e. $\mathbf{y} = f(\mathbf{x},\mathbf{w})$. The role of LM in the proposed algorithm has been described in section I. In the training processes, both the input vector $\mathbf{x}$ and the output vector $\mathbf{y}$ are known and the synaptic weights in $\mathbf{w}$ are adapted to obtain appropriate functional mappings from the input $\mathbf{x}$ to the output $\mathbf{y}$. Generally, the adaptation can be carried out by minimizing the network error function $\mathbf{E}$ which is of the form $\mathbf{E}(\mathbf{y}, f(\mathbf{x},\mathbf{w}))$. In this work we have taken $\mathbf{E}$ as root mean squared error i.e. $\mathbf{E}=\sqrt{\frac{1}{N}\sum_{k=1}^{N}[\mathbf{y}-f(\mathbf{x},\mathbf{w})]^2}$

where N is the number of data considered. The optimization goal is to minimize the objective function $\mathbf{E}$ by optimizing the values of the network weights, $\mathbf{w}$, where, $\mathbf{w}=(w_1,\cdots\cdots,w_d)$

Pseudo code (OBDE-NN Identification Algorithm)

```
/*Initialization of opposition based population*/
Generate a initial random population ipop
for (i=0; i<P; i++)
  for (j=0; j<D; i++)
    $iopop_{i,j} = a_j + b_j - pop_{i,j}$
Select P fittest individual from the set (ipop,iopop) as initial population
While (convergence criterion not yet met)
{
  // $x_{i,g}$ defines a vector of the current vector population
  // $x_{i,g+1}$ defines a vector of the new vector population in the next generation
  for (i=0; i<P; i++)
  {
    $r_1$ = rand (P); //select a random index from 1, 2, P
    $r_2$ = rand (P); //select a random index from 1, 2, P
    $r_3$ = rand (P); //select a random index from 1, 2, P
    $v_{i,g} = x_{r_1,g} + F\,(x_{r_2,g} - x_{r_3,g})$
    /* opposition based mutation starts*/
    Store in mutation population mpop
    $ompop_{i,j}$=min($mpop_j$)+ max($mpop_j$) – $mpop_{i,j}$
    Select P fittest individual from the set ($mpop_{i,j}$, $ompop_{i,j}$) which is denoted by ov
    /* opposition based mutation ends*/
    $t_{j,i,g} = \begin{cases} ov_{j,i,g} & \text{if } (rand_j \le C) \text{ or } j = j_{rand} \\ x_{j,i,g} & \text{otherwise} \end{cases}$
    if $f(t_{i,g}) \le f(x_{i,g})$
      $x_{i,g+1} = t_{i,g}$
    }
    else
    {
      $x_{i,g+1} = x_{i,g}$
    }
  }
}//end  % while
Initialize the weight matrix of Levenberg-Marquardt algorithm taking the values of weights obtained after the fixed number of iterations.
Initialize Weights;
while not meet the Stop Criterion do
  Calculates $e^p(w)$ for each pattern;
  $e_1 = \sum_{p=1}^{P} e^p(w)^T e^p(w)$
  Calculates $J^p(w)$ for each pattern;
  Repeat
    Calculate $\Delta w$ ;
    $e_2 = \sum_{p=1}^{P} e^p(w+\Delta w)^T e^p(w+\Delta w)$ ;
    if $e_1 \le e_2$ then
      $\mu = \mu\beta$
    End %if
  until $(e_2 < e_1)$ ;
  $\mu = \mu / \beta$
  $w = w + \Delta w$
end %while
```

Fig. 5 Pseudo code of OBDE-NN identification scheme

7. Problem Formulation

Example: 1 (A Benchmark Nonlinear System)

We will consider the non-linear system [22, 23] to verify our system identification algorithm given by

$$y_p(t+1) = \frac{y_p(t)[y_p(t-1)+2][y_p(t)+2.5]}{8.5+[y_p(t)]^2+[y_p(t-1)]^2} + u(t) \tag{14}$$

where $y_p(t)$ is the output of the system at the t^{th} time step and *u(t)* is the plant input which is uniformly bounded function of time. The plant is stable at $u(t) \in [-2\ 2]$ i.e. the input signal *u(t)* is bounded in the region [-2 2]. The identification model be in the form of

$$y_{pi}(t+1) = f(y_p(t), y_p(t-1)) + u(t) \tag{15}$$

where $f(y_p(t), y_p(t-1))$ is the nonlinear function of $y_p(t)$ and $y_p(t-1)$ which will be the inputs for DE-NN and OBDE-NN neural system identifier. The output from neural network will be $y_{pi}(t+1)$. The goal is to train the neural networks used to identify the above system such that when an input *u(t)* is presented to the network and to the nonlinear system, the network outputs $y_{pi}(t+1)$ and the actual system output $y_p(t+1)$ should be as close as possible i.e. the identified system output should follow the actual system output.

Example 2 (Box and Jenkins' Gas Furnace System) [23]:

Box and Jenkins' gas furnace data are frequently used in performance evaluation of system identification methods. Given the recorded input–output samples i.e. the input variables as the gas flow *u(t)*, and one output variable, the concentration of carbon dioxide (CO_2), *y (t)*., the objective is to obtain a model of this gas combustion process using neural network identification approaches.

8. Results and Discussion

We present in the system identification results obtained with different approaches such as DE-NN, OBDE-NN and NF applied to the system given in equation (16) and Box Jenkins, Gas furnace problem [23]. In all our simulation studies we have used MATLAB software for coding. The parameters considered for simulation of OBDE-NN and DE-NN schemes are given in Table-1

Table-1 Parameters for DE-NN and OBDE-NN

Population size, P	50
Upper and lower bound of weights	[0 1]
Mutation constant factor , *F*	0.6
Crossover constant, *C*	0.5
Random number, J_r	0.3

Results for Example 1:

We have chosen 11 numbers of neurons in the hidden layer by trial and error so as to obtain the best results. This is explained as follows. We have tried for more no of neurons for the same problem which takes large computational time without achieving appreciable amount of accuracy. After 500 epochs, the training of the neural identifier has been stopped. After the training is over, the identifier's prediction ability has been tested for the input given as follows.

$$u(t) = \begin{cases} 2\cos\left(\frac{2\pi t}{100}\right) & \text{if} \quad t \le 200 \\ 1.2\sin\left(\frac{2\pi t}{20}\right) & \text{if} \quad 200 < t \le 500 \end{cases} \tag{16}$$

Figures 6, 7 give the identification performances of the three identification schemes discussed before namely, DE-NN, OBDE-NN and NF schemes.

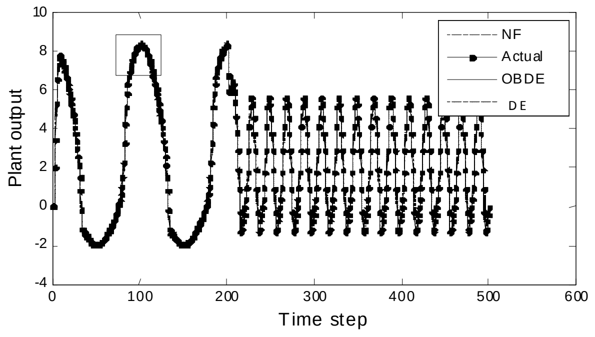


Fig. 6 Comparison of identification performance of OBDE-NN, DE-NN and NF approaches

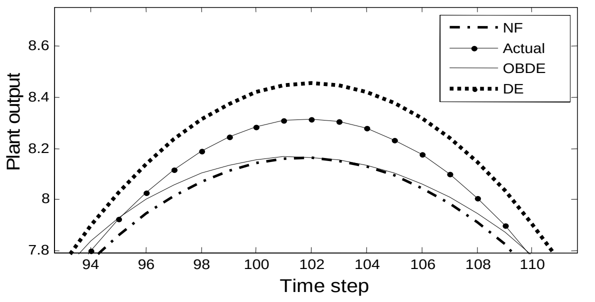


Fig. 7 Comparison of identification performance of OBDE-NN, DE-NN and NF approaches (for the time step 92-112)

Figure 6 compares the actual output , $y(t)$ and identified plant output $\hat{y}(t)$ within the time step of 0 to 500. As the identification performances shown in Figure 6 are overlapping each other, in Figure 7 we have shown the results within the time step of 92 to112. From this it is clear that the proposed OBDE-NN exhibits better identification ability compared to DE-NN and NF approaches.

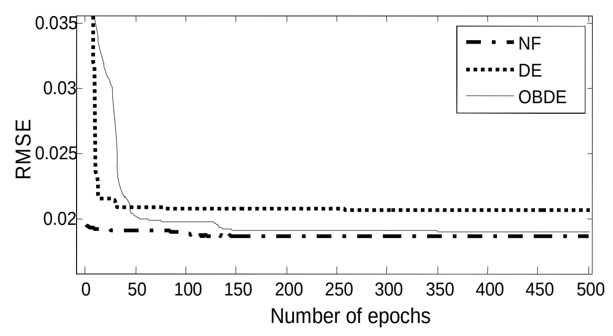


Fig. 8 Comparison of training RMSE for OBDE-NN, DE-NN and NF approaches

Figure 8 shows that in NF based identification, the RMSE is less than DE-NNand OBDE-NN cases. Due to the over fitting of the weights of the neural network the the testing RMSE for NF scheme is not minimum even if the training RMSE is minimum. This can be analysed from the numerical values given in Table-2 i.e. the value of tarining RMSE for NF is minimum i.e. 0.0187 whereas the testing RMSE is minimum for OBDE-NN i.e. 0.1137. The comparion of RMSE between DE-NN and OBDE-NN indicates that OBDE-NN is having faster convergence than DE-NN. Hence, the proposed OBDE-NN has the advantage of less testing RMSE and faster convergence compared to the previous reported DE-NN and NF appproaches to system identification.

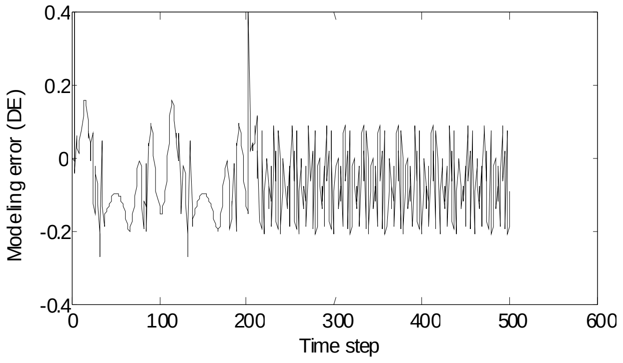


Fig. 9 DE-NN Identification error

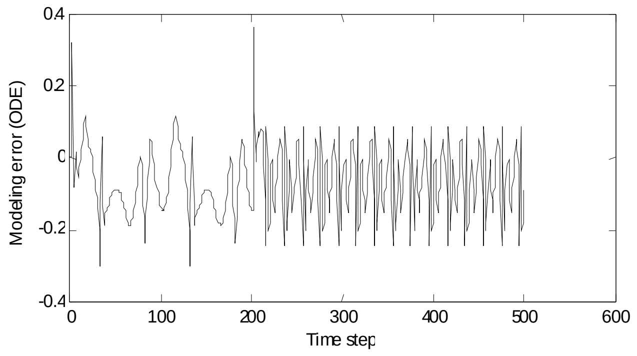


Fig. 10 OBDE-NN Identification error

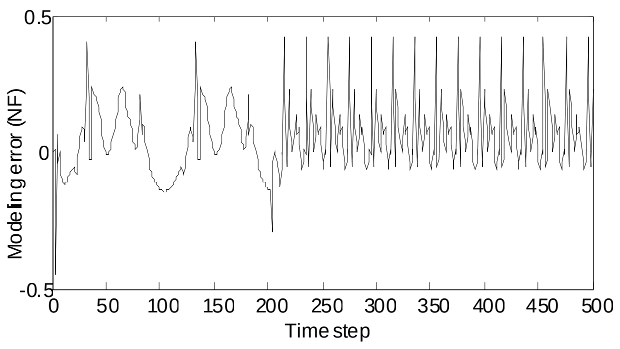


Fig. 11 NF Identification error

Figures 9, 10 and 11 show the identification errors for all the three approaches i.e. OBDE-NN, DE-NN and NF. It is seen that the error is the least in the case of OBDE-NN amongst the three cases.

Example: 2 (Box Jenkin's Gas Furnace Modeling):

The time series data set for a gas furnace consists of 296 input–output samples recorded with a sampling period of 9 seconds. The instantaneous values of output $y(t)$ have been regarded as being influenced by ten variables mainly the past values of *y(t) for past four* sampling times and *u(t)* for past six sampling times i.e. $y(t-1)$, $y(t-2)$, $y(t-3)$, $y(t-4)$, $u(t-1)$, $u(t-2)$, $u(t-3)$, $u(t-4)$,$u(t-5)$, $u(t-6)$.

The original data set contains 296 [*u(t),y(t)*] data pairs. But, by converting the data set to previous sampling instants so that each training data consists of [*y(t − 1)…………y(t-4),u(t − 1)………u(t − 6)*] reduces the number of data points to effectively 290 data pairs.

The number of training data was taken as 100 for the three identification schemes (DE-NN, OBDE-NN and NF) and the rest 190 data pairs were considered as the test data.

It may be noted that, for dynamic system modeling, the inputs selected must contain elements from both set of historical furnace outputs {*y(t-1)…………..y(t-4)*} and the set of historical furnace inputs {*u(t-1)…………..u(t-6)*}. For simplicity, we assumed two inputs are fed to the neural networks namely, one from outputs and the other from inputs. In other words, we aim to build 24 [$=4\times6$; because we consider past four *y(t)* and past six *u(t)*] models with various input-output combinations.

Table 2 gives the training and testing performances of these 24 models. During these experiments, we observed the pattern of estimation errors corresponding to the number of hidden nodes taken. By the process we end up with choice of eleven numbers of hidden units leading lowest estimation error. For all the methods eleven number of hidden layer neurons were taken and the results obtained after 100 epochs. We have tried for more no of neurons for the same problem which took more computational time without achieving appreciable amount of accuracy.

As it is not possible to show the identification performance and error curve for all the 24 cases given in Table 2, we have shown three cases only [*y(t-1),u(t-3)*) ; (*y(t-4),u(t-5)* and (*y(t-4),u(t-4)*)] to analyze the RMSE and their performances.

In Fig.12 we can see that the NF with *y(t-1)* and *u(t-3)* as inputs has the smallest training error but for the same inputs OBDE-NN is having least testing error. Figure 13 display the identification performances curves, for time step 0 to 300. Figure 14 shows the same identification performances within the time step 110 to 116, the actual and OBDE-NN model output are matching approximately but for the other two i.e. DE-NN and NF model the identification error is more.

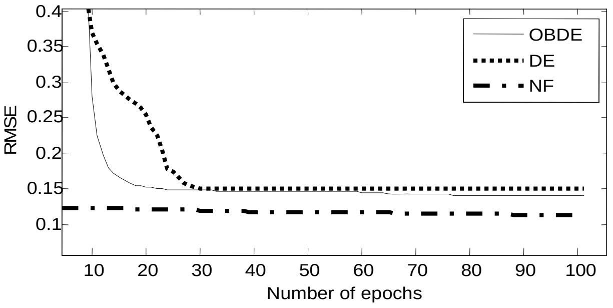


Fig. 12 Comparison of training RMSE for DE-NN,OBDE-NN and NF for the input (*y(t-1),u(t-3)*)

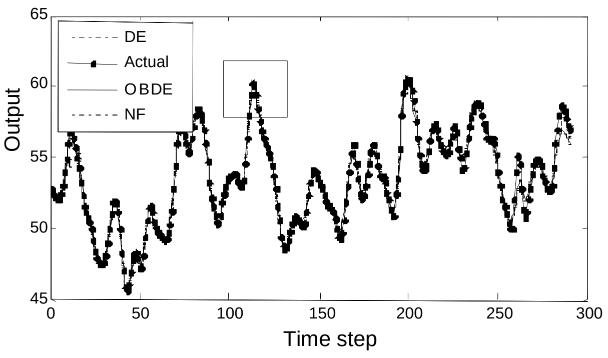


Fig. 13 Comparison of identification performance for the input (*y(t-1),u(t-3)*

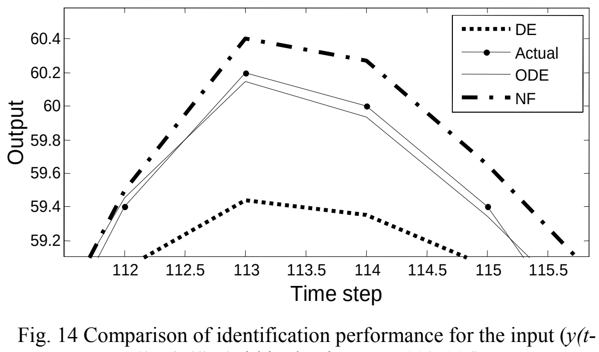


Fig. 14 Comparison of identification performance for the input (*y(t-1),u(t-3)*) (within the time step 110-116)

Figure 15 gives the comparison of identification performance of all the three system identifcation approaches for the input *y(t-4), u(t-5)* within the time step 0 to 300. Figure 16 shows the same identification performance within the time step 110 to 116 for better comparison. From these figures it observed that NF is not identifying the system properly at the same time the training RMSE is minimum, whose numerical value is given in Table-2.

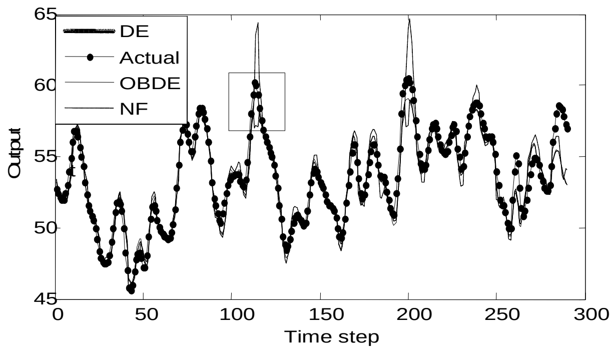


Fig. 15 Comparison of identification performance for the input (*y(t-4),u(t-5)*

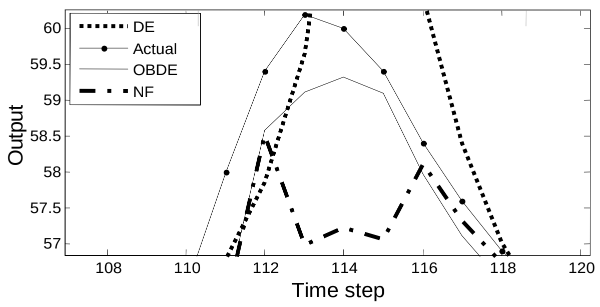


Fig. 16 Comparison of identification performance for the input (*y(t-4),u(t-5)* (within the time step 110-120)

Figure 17 gives the RMSE for the input *y(t-4)* and *u(t-5)*. Here the we have considered 20 epochs because trere was no change in RMSE for DE-NN approach after 20 epochs. In this case even if for OBDE-NN the value of RMSE starts from a higher vale but it is converging to a lower value finally.

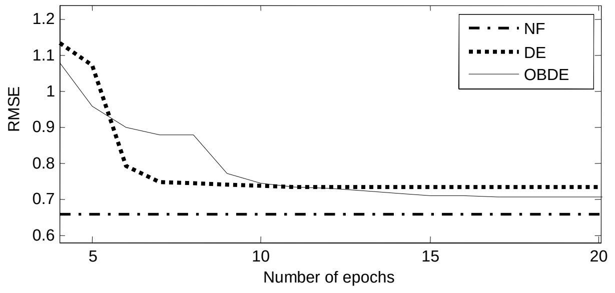


Fig. 17 Comparison of training RMSE for the input (*y(t-4),u(t-5)*)

From Figures 16 and 17 it is observed that even if the training RMSE for neuro-fuzzy approach is less than that DE-NN and OBDE-NN approach but due to the more testing error it is not identifying the nonlinear system properly.

Figure 18 shows the RMSE for the input *y(t-4)* and *u(t-4)*. From the figure it is clear that the RMSE for OBDE-NN is having higher convergence speed and attending a lower value in comparison to DE-NN approach the numerical values are mentioned in Table-2. Figure 19 shows the identification performance for the input *y(t-4)* and *u(t-4)* from which it is found even if the training error for OBDE-NN approach is slightly higher than the DE-NN approach but having much better identification capability in comparison to DE-NN approach interms of faster convergence and less identification error.

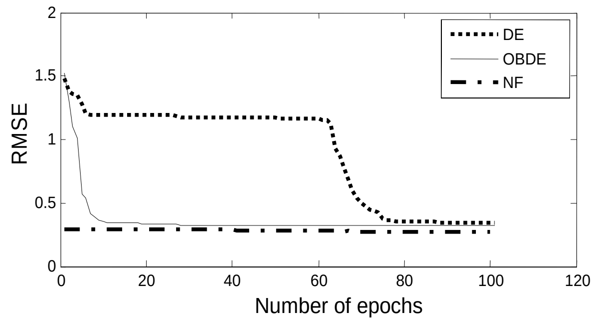


Fig. 18 Comparison of training RMSE for DE-NN,OBDE-NN and NF for the input (*y(t-4),u(t-4)*)

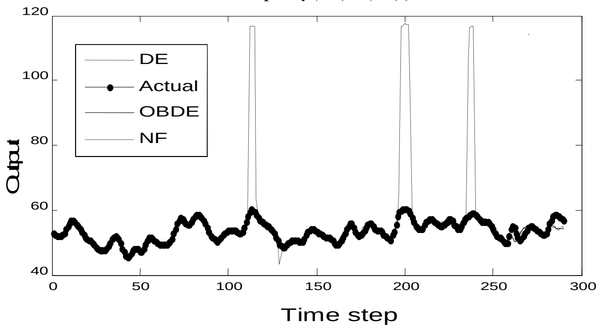


Fig. 19 Comparison of identification performance for the input (*y(t-4),u(t-4)*

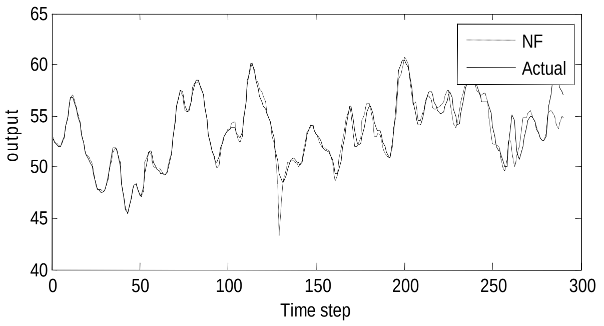


Fig. 20 Comparison of NF identification performance for the input (*y(t-4),u(t-5)*

Figure 20 shows the NF identification performance for the input *y(t-4)* and *u(t-4)*. For this input, the testing RMSE was found to be maximum for DE-NN approach which is clear from the Fig.18. Comparing the values given in Table-2 it was found that the training RMSE is minimum for NF approach but it is having more testing RMSE in comparison with OBDE-NN approach. So neither the DE-NN nor the NF approach is identifying the nonlinear system accurately. Even if the training RMSE is less for NF than OBDE-NN method but due to the overfitting of the neural network the identification error is more.

Table-2 shows the numerical values of training and testing RMSE for 24 different combinations of inputs and outputs. The training RMSE was found to be less for 19 cases in NF approach compared to OBDE-NN but in testing OBDE-NN outperformed for 18 cases. This occurs due to the over fitting of the neural networks trained with NF which can be avoided by using the proposed OBDE-NN approach. Also OBDE-NN was found to be better in terms of identification performance and faster convergence compared to DE-NN almost all cases. We found that model with *y(t-1)* and *u(t-3)* as input has the smallest training and testing RMSE for DE-NN, OBDE-NN and NF identification schemes and considered as best model input output combination compared to all the 24 combinations given in Table-2.

8. Conclusions

The paper has presented a new differential evolution approach based on opposition based learning applied for training neural network used for non-linear system identification. This new evolutionary approach is found to exhibit better system identification performances compared to DE+LM+NN[15] approach presented earlier. This approach is also outperforms over the existing neuro-fuzzy approach in terms of better identification capability.

Table-2 Comparison of Errors for DE-NN, OBDE-NN and NF

		Testing Error (RMSE)			Training Error (RMSE)		
	Example 1						
	Input	DE	OBDE	NF	DE	OBDE	NF
	y(k), y(k-1) u(k)	0.1186	0.1137	0.1342	0.0207	0.0190	**0.0187**
	Example 2						
Case Studies	Input	DE	OBDE	NF	DE	OBDE	NF
1	*y(t-1),u(t-3)*	0.4400	0.4194	0.4296	0.1501	0.1411	**0.1134**
2	*y(t-3),u(t-4)*	0.7838	0.7773	**0.7643**	0.3402	0.2850	**0.2562**
3	*y(t-2),u(t-4)*	0.6733	0.6602	0.6854	0.3256	0.2898	**0.2570**
4	*y(t-1),u(t-2)*	0.4906	0.6801	**0.5054**	0.2909	0.2924	**0.2413**
5	*y(t-1),u(t-4)*	0.5430	0.5132	0.5829	0.2991	0.2926	**0.2441**
6	*y(t-4),u(t-4)*	12.259	0.8894	0.9648	0.3428	0.3274	**0.2674**
7	*y(t-2),u(t-3)*	1.1340	0.7199	0.9789	0.2968	0.3051	**0.2137**
8	*y(t-1),u(t-1)*	0.6183	0.6056	0.6363	0.4638	0.4151	**0.3688**
9	*y(t-4),u(t-3)*	1.2405	1.2771	1.5953	0.7266	0.4301	0.5676
10	*y(t-1),u(t-6)*	0.8469	0.8410	0.9687	0.6012	0.5661	**0.5117**
11	*y(t-3),u(t-3)*	1.0067	1.0347	**0.9883**	0.5172	0.5176	**0.4172**
12	*y(t-2),u(t-2)*	0.9889	0.9753	**0.9517**	0.6314	0.6261	**0.5187**
13	*y(t-1),u(t-5)*	0.6873	0.6518	1.7212	0.6220	0.6303	**0.5409**
14	*y(t-4),u(t-5)*	1.0149	0.9698	1.0981	0.7333	0.7038	**0.6393**
15	*y(t-2),u(t-1)*	1.8368	1.2726	1.7128	0.8934	0.6844	**0.6728**
16	*y(t-2),u(t-5)*	0.9176	1.1808	1.5024	0.7222	0.6804	**0.6147**
17	*y(t-3),u(t-5)*	0.9536	1.0470	1.1396	0.7138	0.7338	**0.6046**
18	*y(t-3),u(t-2)*	1.8184	1.4138	1.9161	0.8766	0.8600	**0.7266**
19	*y(t-4),u(t-6)*	1.7628	1.4677	1.6660	1.3988	1.1126	1.1224
20	*y(t-2),u(t-6)*	1.3352	1.2639	2.5044	1.6264	1.1945	**1.0835**
21	*y(t-4),u(t-2)*	1.6725	1.6377	2.3285	1.1799	1.1963	**0.8697**
22	*y(t-3),u(t-6)*	27.468	1.4641	2.2827	1.2063	1.2424	**1.0782**
23	*y(t-3),u(t-1)*	1.7123	1.6475	2.0182	1.5725	1.2702	**0.8776**
24	*y(t-4),u(t-1)*	2.0821	2.0217	2.3595	1.4250	1.4352	**1.1995**

References

[1] S. Chen, S. A. Billings (1989). Representation of non-linear systems: the NARMAX model, *International Journal of Control.* 49, 1013-1032.

[2] H. Hujiberts, H. Nijmeijer, R.willems (2000). System identification in communication with chaotic Systems, *IEEE Trans on Circuits and Systems-I.* 47(6), 800-808.

[3] M. Adjrad, A. Belouchrani, (2007). Estimation of multi component polynomial phase signals impinging on a multi sensor array using state-space modeling, *IEEE Trans Signal Processing.* 55(1), 32-45.

[4] K.Watanbe, I.Matsuura, M.Abe, M.Kebota, D.M.Himelblau. Incipient Fault Diagnosis of Chemical Processes via Artificial Neural Networks, *AICHE Journal*. 35(11) 1803-1812.

[5] Yao Xie, Bin Guo, Luzhou Xu, Jian Li, Peter Stoica (2006). Multistatic adaptive microwave imaging for early breast cancer detection, *IEEE Trans Biomedical Eng*. 53(8), 1647-1657.

[6] M. Schetzmen, The Voltera and Winner Theories on Nonlinear Systems, Newyork, Wiley.

[7] Feng Dinga, Tongwen Chenb (2005) Identification of Hammerstein nonlinear ARMAX systems, *Automatica*. 41, 1479-1489.

[8] E. Hernandez, Y. Arkun. (1993). Control of nonlinear systems using Polynomial ARMA models, *AICHE Journal*. 39(3), 446–460.

[9] T. T. Lee, J. T. Jeng (1998). The Chebyshev polynomial-based unified model neural networks for functional approximation, *IEEE Trans. Syst., Man Cybern.-B*. 28, 925–935.

[10] N. Sadegh, (1993). A perceptron based neural network for identification and control of nonlinear systems, *IEEE Trans. Neural Networks*.4, 82–988.

[11] K. S. Narendra, and K. Parthaasarathy (1990). Identification and Control of Dynamical Systems Using Neural Networks, *IEEE Trans Neural Networks*. 1, 4-27.

[12] X. Yao (1993). Evolutionary artificial neural networks, . *Int. Journal of Neural Systems*, .4, 203-222,

[13] Storn, R (1999). System Design by Constraint Adaptation and Differential Evolution, *IEEE Trans. Evol. Comput.* 3, 22-34.

[14] J. Ilonen, J. K Kamarainen, J. Lampinen (2003). Differential Evolution Training Algorithm for Feed Forward Neural Networks, *Neural Processing Letters,* 17, 93-105.

[15] B. Subudhi, D. Jena (2008). Differential Evolution and Levenberg Marquardt Trained Neural Network Scheme for Nonlinear System Identification, *Neural Processing Letters*, 27(3), 285-296.

[16] B. Subudhi , D. Jena (2008). A State of the Art on Hybrid Soft Computing Techniques Towards Identification of Nonlinear Dynamical System *Proc. IEEE Sponsored Conf. on Computational Intelligence, Control and Computer Vision in Robotics & Automation*,119-124, N.I.T Rourkela

[17] H. R. Tizhoosh (2005). Opposition-based learning: A new scheme for machine intelligence, *in Proc. Int. Conf. Comput. Intell. Modeling* Control*and Autom.*, Vienna, Austria, vol. I, pp. 695–701.

[18] M. Shokri, H. R. Tizhoosh, and M. Kamel (2006). Opposition-based Q (λ) algorithm, in *Proc. IEEE World Congr. Comput. Intell.*, Vancouver,BC, Canada, 646–653.

[19] M. Ventresca and H. R. Tizhoosh (2006). Improving the convergence of back-propagation by opposite transfer functions, in *Proc. EEE WorldCongr. Comput. Intell.* Vancouver, BC, Canada, 9527–9534.

[20] S. Rahnamayan, H. R. Tizhoosh, and M. M. A Salama (2008). Opposition versus randomness in soft computing techniques, *Journal of Applied Soft Comput.*, 8 (2), 906-918.

[21] S Rahnamayan, H. R. Tizhoosh, and Magdy M. A. Salam (2008). Opposition-Based Differential Evolution, *IEEE Trans. Evolutionary Computation,* 12 (1), 64-79

[22] C.-T. Lin, C.S. George Lee (1996). Neural *Fuzzy Systems:* A Neuro-fuzzy Synergism to Intelligent Systems, Prentice Hall International, Inc., New Jersey:

[23] G.E.P. Box and G.M. Jenkins (1970), Time Series Analysis, Forecasting and Control, Holden Day, San Francisco.

[24] J.-S. Jang (1993). ANFIS: Adaptive Network based Fuzzy Inference Systems, *IEEE Transactions on Systems, Man, and Cybernetics*. 23(3), 665-685.

[25] J.-S. Roger Jang, C.-T. Sun and E. Mizutani (2003). Neuro-fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence, Eastern Economy Edition, New Delhi: PHI.

[26] L. Jung (1999). System identification: theory for the user. (2nd ed). Englewood Cliffs, New Jersey, Prentice-Hall.

Author Biographies

DR. BIDYADHAR SUBUDHI has received a Bachelor Degree in Electrical Engineering from Regional Engineering College Rourkela (presently National Institute of Technology Rourkela), India, Master of Technology in Control & Instrumentation from Indian Institute of Technology, Delhi in 1994 and PhD degree in Control System Engineering from University of Sheffield, United Kingdom in 2003. He worked as a Post Doctoral Research Fellow in the Department of Electrical & Computer Engineering, NUS, Singapore during May-Nov 2005. Currently he is a Professor in the Department of Electrical Engineering in the National Institute of Technology, Rourkela, India. His research interests include System Identification, Intelligent Control, Control of Mobile and Flexible Robot Manipulators, Estimation of Signals & Systems. He is an author of 25 journal papers and 30 conference papers, a chapter in a research monograph. He is a Fellow of the Institution of Engineers (India), Life Member of Systems Society of India and Senior Member of IEEE. He is serving as a Technical Committee Member, IEEE Intelligent Control Society.

DEBASHISHA JENA has received a Bachelor of Electrical Engineering degree from University College of Engineering, Burla, India, in 1996 and Master of Technology in Electrical Engineering in 2004. He is currently working toward the award of the PhD degree in the Department of Electrical Engineering, National Institute of Technology, Rourkela, India. He worked as a Faculty Member in the National Institute of Science & Technology, Berhmapur, Orissa during 2004-2007. His research interests include Evolutionary Computation and System Identification with application to Non-linear Systems. He has been awarded a GSEP Fellowship in 2008 from Canada for research in Control and Automation.