A Novel Hybrid Chemical Reaction Optimization Algorithm with Adaptive Differential Evolution Mutation Strategies for Higher Order Neural Network Training

Sibarama Panigrahi
School of Computer Science, National Institute of Science and Technology, India

Abstract: *In this paper, an application of a hybrid Chemical Reaction Optimization (CRO) algorithm with adaptive Differential Evolution (DE) mutation strategies for training Higher Order Neural Networks (HONNs), especially the Pi-Sigma Network (PSN) is presented. Contrasting to traditional CRO algorithms, the reactant size (population size) remains fixed throughout all iterations, which makes it easier to implement. In addition, four DE mutation strategies (DE/rand/1, DE/best/1, DE/rand/2 and DE/best/2) with adaptive selection of control parameters as inter-molecular reactions and one intra-molecular reaction have been used. The proposed algorithm combines the diversification property of inter-molecular reactions following DE/rand mutation strategies and intensification property of intra-molecular reaction as well as inter-molecular reactions following DE/best mutation strategies, thereby glorifying the chances of reaching the global optima in less iteration. The performance of the proposed algorithm for HONN training is evaluated through a well-known neural network training benchmark i.e., to classify the parity-p problems. The results obtained from the proposed algorithm to train HONN have been compared with results from the following algorithms: Basic CRO algorithm, CRO-HONN Training (HONNT) and the most popular variants of DE algorithm (DE/rand/1/bin, DE/best/1/bin). It is observed that the application of the proposed hybridized algorithm to (DE-CRO-HONNT) performs statistically better than that of other algorithms considering both classification accuracy and number of generation taken to attain the solutions.*

Keywords: *CRO, DE, HONN, training algorithm, PSN.*

Received August 18, 2013; accepted September 21, 2014

1. Introduction

Over the past few decades, Artificial Neural Network (ANN) models have been widely used for pattern reorganization, pattern classification and mathematical function approximation. However, now-a-days instead of traditional neural networks, Higher Order Neural Networks (HONNs) have found in increasing consideration in forecasting, classification and regression problems due to several unique characteristics, including: Stronger approximation with faster convergence property; greater storage capacity; and higher fault tolerance capability. On the other hand, the major drawback of most of the HONN models is that the number of weights of the network grows exponentially with the increase in dimensionality of input patterns. But, Pi-Sigma Networks (PSNs) are a special class of HONN which are not only computationally much more efficient than other HONN models but also manages to incorporate the capability of first order HONN indirectly. The PSNs were introduced by Shin and Ghosh [24] and have addressed several difficult tasks such as zeroing polynomials [9] and polynomial factorization [18] more effectively than traditional Feed-forward Neural Networks (FNNs). Shin and Ghosh [26] have formulated Ridge Polynomial Neural Networks (RPNN) by adding gradually more complex PSNs. RPNNs have shown competitive performance in various tasks such as pattern recognition [29], image prediction [17], time series prediction [10], data classification [26], and intelligent control [12]. Since, the RPNNs are a generalization of PSNs, its effectiveness directly depends on the effectiveness of PSNs. Therefore, a better learning algorithm for PSNs will also improve the efficiency of RPNN. Despite of better performance of PSN and RPNN across various application domains, a few papers were devoted to develop an efficient training algorithm for PSNs [5, 23, 24]. This motivates towards the development of an efficient training algorithm for training PSNs.

The PSNs are supervised networks and efficiency of any supervised neural network depends on the algorithm used for its training. The objective of any supervised training algorithm is to minimize the approximation error by obtaining the optimal weight set. The optimal weight set of PSNs can be obtained by using either gradient or evolutionary learning algorithm. Since the training of PSN is a multimodal search problem, the gradient based training algorithms

often suffer from several shortcomings, including: Easily getting trapped to local minima; have slow convergence properties; and training performance is sensitive to initial values of its parameters. Due to these disadvantages, research on different optimization techniques that are dedicated to PSN training is still needed. There are many optimization techniques such as Differential Evolution (DE) [2, 3, 4, 11, 20, 21, 28], Genetic Algorithm (GA) [7], Particle Swarm Optimization (PSO) [13], Ant Colony Optimization (ACO) [27], a Bee Colony Optimization (BCO) [19], an Evolutionary Strategy (ES) [1], Quantum Inspired Algorithms (QEA) [8], Chemical Reaction Optimization (CRO) [14, 15, 16] etc., which can be used to train PSN. In this paper an attempt has been made to hybridize the self adaptive DE mutation operators with a CRO algorithm to obtain an efficient training algorithm for PSN.

The remainder of this paper is organized as follows: Section 2 briefly describes the mathematical model of PSN, differential evolution algorithm and chemical reaction optimization algorithm. The proposed training algorithm for PSN has been explained in section 3. In section 4 experimental results are presented. And finally, conclusions are drawn in section 5.

2. Related Work

2.1. Pi-Sigma Neural Network

PSN is a special type higher order feed forward neural network that calculates the product of the sum of the input components and passes it to a nonlinear function. The network architecture of PSN as shown in Figure 1 consists of a single hidden layer of summing units and an output layer of product units. The weights connecting the input neurons to the hidden neurons are trainable whereas those connecting the hidden neurons to the output neurons are fixed to one. Such a network topology with only one layer of trainable weights drastically reduces the training time [6, 24, 25]. Moreover, the product units provide the higher order capabilities of HONN models.

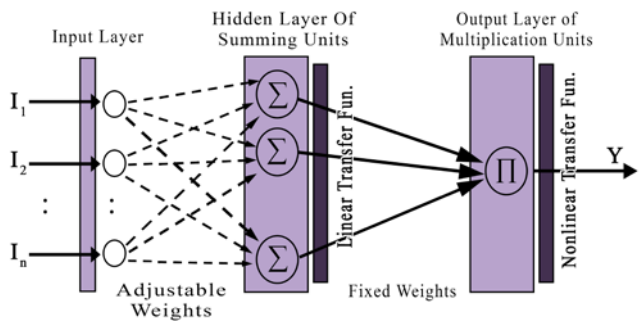

Figure 1. Architecture of a typical PSN.

Consider a PSN with n inputs, k hidden neurons and m output neurons. The number of hidden neurons in the hidden layer defines the order of a PSN. For a k^{th} order PSN the number of trainable weights is $n \times k$ considering each summing unit is associated with n weights (bias components are not considered). The output of the PSN is computed by making product of the output of k hidden units and handing it to a nonlinear function, which is defined as:

$$Y = \sigma\left(\prod_{j=1}^{k} h_j\right) \quad (1)$$

Where σ is a nonlinear transfer function and h_j is the output of the j^{th} hidden unit which is computed by making the sum of the products of each input (I_i) with the corresponding weight (w_{ij}) between i^{th} input and j^{th} hidden unit. The output of hidden unit is computed as:

$$h_j = \sum_{i=1}^{n} (w_{ij} I_i) \quad (2)$$

2.2. Differential Evolution

The DE algorithm was introduced by Storn and Price [28]. It is a simple yet efficient stochastic direct search method for global optimization of multimodal function. Compared to most other evolutionary algorithms, DE is much simpler and straightforward to implement. Although, PSO is also very easy to code, the performance of DE and its variants outperforms the PSO variants over a wide variety of problems [3, 22] and the Congress on Evolutionary Computation (CEC) competition series. Since, the inception DE, it has been upgraded intensively in recent years [2]. The variants of DE algorithm differ from each other by the type of mutation and crossover scheme being used. The crossover may be binary or exponential. For both the crossover different mutation schemes, suggested by Price *et al.* [20, 21] are summarized as follows:

- DE/rand/1: $MV = C_{r1} + F*(C_{r2}-C_{r3})$.
- DE/best/1: $MV = C_{best} + F*(C_{r1}-C_{r2})$.
- DE/rand/2: $MV = C_{r1} + F*(C_{r2}-C_{r3}) + F*(C_{r4}-C_{r5})$.
- DE/best/2: $MV = C_{best} + F*(C_{r1}-C_{r2}) + F*(C_{r3}-C_{r4})$.
- DE/target-to-best/1: $MV = C_i + F*(C_{best}-C_{r1}) + F*(C_{best}-C_{r2})$.

The conventions used above are *DE/a/b*, where *DE* stands for differential evolution, *a* represents the base vector to be perturbed (it may be best vector or target vector or a randomly chosen vector), *b* represents the number of difference vectors used for perturbation of *a*; *MV* stands for mutant vector; C_{best} for the best vector of a population and C_r for a randomly chosen vector from the population. Note that, all the vectors chosen for any mutation strategy must be from the same generation and should be distinct to each other.

2.3. Chemical Reaction Optimization

CRO algorithm was proposed recently by Lam and Li [15], is a metaheuristic optimization technique. It is

inspired by the nature of chemical reactions which loosely couple chemical reactions with optimization. A chemical reactant system consists of a set of chemical substances (reactants/molecules) and its surrounding. Each molecule consists of some atoms and is associated with enthalpy (minimization problem) or entropy (maximization problem). A chemical change of a molecule is triggered by a collision and the corresponding subtle change is called ineffective elementary reaction. There are two types of collision: Uni-molecular/intra-molecular/monomolecular collision (occurs when a molecule hits on some external substance like wall of a container) and inter-molecular collision (occurs when molecules collide with each other). Basing on the number of molecules take part in a reaction, the reaction may be: Uni-molecular or bi-molecular or tri-molecular and so on.

Most of the reactions are reversible in nature i.e., they can go in forward or backward direction. Chemical reactions transform one set of chemical substances to another in order to make the system stable. The CRO can be thought of as a new evolutionary technique with molecules as chromosomes; atoms as genes; enthalpy/entropy as fitness function; reactions as crossover and mutation strategies; and reversible reaction as a selection process. However, unlike other evolutionary algorithms in CRO, the reactant size (similar to population size) may vary from one generation to the other. Few authors also have proposed fixed population sized CRO algorithms and shown that fixed population sized CRO not only performs better but also easier to implement [23]. To have an elaborated description regarding CRO algorithm, interested readers may go through the tutorial of CRO [14].

3. DE-CRO-HONNT Method

Algorithm 1 presents the pseudo-code of the proposed method. In this proposed method an attempt has been made to use adaptive DE mutation strategies as inter-molecular reactions of a CRO algorithm and use it for training PSN. Like other evolutionary algorithms, the proposed DE-CRO-HONNT operates in three phases: Initialization phase, iteration phase and final phase. The initial phase assigns the value to initial parameters like termination criteria, total number of reactants/molecules in a generation represented by ReacNum and generates initial set of reactants. The iteration phase simulates the reaction processes. Five different reactions are considered comprising of one intra-molecular (uni-molecular) and four inter-molecular reactions. Every elementary reaction is followed by a greedy reversible reaction to update the reactants.

Algorithm 1: DE-CRO-HONNT.

Set the iteration-counter i=0

*/*Randomly Initialize the ReacNum of Reactants from a uniform distribution*[*U*(*upper bound*); *L*(*lower bound*)]: $P^i=\{R_1^i, R_2^i, R_3^i, ..., R_{ReacNum}^i\}$, *with* $R_j^i=\{W_{j,1}^i, ..., W_{j,D}^i\}$ *for* j=1, 2, 3, ..., *ReacNum, D=length of each reactant* (*NOIN*×*NOHN*), $W_{j,k}^i=k^{th}$ *atom of* j^{th} *reactant in* i^{th} *iteration representing a weight of PSN.*

```
for j=1 to ReacNum
    Calculate the enthalpy e(R_j)
end of for
While (termination criteria is not satisfied) do begin
  for j=1 to ReacNum
  // Perform reactions over all the reactants of P^i
    Generate rand_1 randomly in an interval [0, 1]
    if rand_1 ≤ 0.2
      Decomposition (R_j^i); //Uni-molecular Reaction
    else if rand_1>0.2 && rand_1≤0.6
      Perform tri-molecular reactions
      Generate rand_2 randomly in an interval [0, 1]
       if rand_2 ≤ 0.5
       //Use DE/rand/1 mutation strategy
          Select three random numbers R_1^i, R_2^i, R_3^i ∈
          ReacNum such that R_j^i≠R_1^i≠R_2^i≠R_3^i
          DErand1(R_1^i, R_2^i, R_3^i)
       else
       // Use DE/best/1 mutation strategy
          Select the best reactant R_best^i and two random
          numbers R_1^i, R_2^i ∈ ReacNum such that
          R_j^i≠R_best^i≠ R_1^i≠R_2^i
          DEbest1(R_1^i, R_2^i, R_best^i)
       end of if
    else
          Perform penta-molecular reactions
          Generate rand_3 randomly in an interval [0, 1]
          if rand_3 ≤ 0.5
          // Use DE/rand/2 mutation strategy
          Select five random numbers
          R_1^i, R_2^i, R_3^i, R_4^i, R_5^i ∈ ReacNum
          such that R_j^i≠R_1^i≠R_2^i≠ R_3^i ≠ R_4^i≠R_5^i
            DErand2(R_1^i, R_2^i, R_3^i, R_4^i, R_5^i)
          else
          // Use DE/best/2 mutation strategy
          Select the best reactant R_best^i and four
          random numbers R_1^i, R_2^i, R_3^i, R_4^i ∈ ReacNum
          such that R_j^i≠R_best^i≠ R_1^i≠R_2^i≠R_3^i≠R_4^i
            DEbest2(R_1^i, R_2^i, R_3^i, R_4^i, R_best^i)
       end of if
    end of if
   Apply greedy Reversible Reaction for increased
       enthalpy to update reactants
  end of for
  Set the iteration counter i=i+1
end of while
```

Use the reactant having best enthalpy as the optimal weight set of PSN.

All the reactions are elaborated in the following subsequent subsections. In the final phase the reactant having best enthalpy is used as the optimal solution (i.e., optimal weight set of a PSN).

3.1. Reactant Encoding

A set of real numbers is used to represent one reactant, with each real number corresponding to a weight of the PSN. Thus, a reactant represents a weight set of the PSN. The length of a reactant depends on the

number of inputs (n) and hidden neurons (k) of the PSN and which is equal to $n \times k$ (not considering bias units).

3.2. Enthalpy of Reactant

Each reactant is associated with some enthalpy (fitness value). As each reactant represents a weight set of the PSN, the Mean Square Error (MSE) on the train set is considered to be its enthalpy. The lower the value of enthalpy represents better the reactant. The *MSE* is defined as follows:

$$MSE = \frac{\sum_{i=1}^{NOP} (Y_i - T_i)^2}{NOP} \quad (3)$$

Where Y_i and T_i are the output of PSN and target for i^{th} train pattern.

3.3. Elementary Chemical Reactions

One uni-molecular, two tri-molecular and two penta-molecular elementary reactions are considered. The tri-molecular and penta-molecular reactions use different DE mutation strategies. The scale parameter (F) used by the DE mutation strategies are dynamically and self adaptively determined depending on the problem. Note that the parameter adaption is somewhat inspired by MDE_pBX algorithm [11] but the former one is distinct due to its own characteristics. The five reactions are chosen considering both intensification and diversification. All the five reactions considered have equal chance to occur. Therefore, uni-molecular reactions occur with 20%, tri-molecular with 40% and penta-molecular with 40% probability.

3.3.1. Uni-Molecular Reactions

In uni-molecular reactions only one reactant takes part in the reaction and one product is produced by modifying one atom of the reactant. These reactions assist in intensification of the solution by making local search. One uni-molecular reaction is considered called as decomposition reaction which is explained below.

3.3.1.1. Decomposition Reaction

In this reaction a randomly selected atom of the reactant undergoes sudden change to bring a new reactant.

Consider a reactant $R_j = \{W_{j,1}, W_{j,2}, ..., W_{j,D}\}$ with $W_{j,x}$ ($x \in [1, n]$) be an atom of the reactant-j. The pseudo-code of the decomposition reaction is described in Algorithm 2.

Algorithm 2: Decomposition(R_j).

Input: *A reactant* R_j.
Duplicate R_j *to produce* R_{new}.
Select an atom x ($x \in [1, D]$) *randomly.*
$W_{new,x} = L + \lambda \times (U - L)$.
Where the rate of reaction (λ) *is a random number generated randomly from a uniform distribution between* [0, 1].
Output: *A new reactant* R_{new}.

3.3.2. Tri-Molecular Reactions

In tri-molecular reactions three reactants take part in the reaction to produce one product. Two tri-molecular reactions are considered using two different DE mutation strategies DE/rand/1 and DE/best/1 with the intension to combine the diversification property of the former one and intensification property of later one. Each of the reactions as a whole has a probability of 20% to occur.

3.3.2.1. DErand1 Reaction

Here, DE/rand/1 mutation strategy is used to generate new reactants. In addition, the scale factor (F) used in DE mutation strategy (equivalent to reactant rate λ) is dynamically and self adaptively determined based on the problem. The rate of reaction (λ) is generated randomly from a Cauchy distribution with location parameter M and scale parameter 0.1. The value of M is initially set to 0.6 and self adaptively determined in the following manner:

$$S_f = 0.8 + 0.2 \times \text{rand}(0, 1)$$
$$M_{t+1} = S_f \times M_t + (1 - S_f) \times \text{mean}(\lambda_{success}) \quad (4)$$

Where t= Number of times the reaction occurs, $\lambda_{success}$ stores the successful rate of reactions that generates better reactants, thereby improving the chances of generating better reaction rates consequently better reactants as more and more this reaction occurs. Here, instead of traditional normal or uniform distribution Cauchy distribution is used because it diversifies the solution more. The pseudo-code of the DErand1 reaction is described in Algorithm 3.

Algorithm 3: *DErand*1 (R_1, R_2, R_3).

Input: *Three reactants* R_1, R_2, R_3.
$R_{new} = R_1 + \lambda \times (R_2 - R_3)$.
Where $\lambda = Cauchyrnd(M_t, 0.1)$, *is a random number generated from a Cauchy distribution with location parameter* M_t *and scale parameter* 0.1. *It is regenerated if the random number falls out of the range* [0, 2].
Output: A new reactant R_{new}.

3.3.2.2. DEbest1 Reaction

This reaction is almost similar to that of DErand1 but, here DE/best/1 mutation strategy is used to generate new reactants. The reaction rate is self adaptively determined similar to that of previous reaction. The pseudo-code of the DEbest1 reaction is described in Algorithm 4.

Algorithm 4: DEbest1 (R_1, R_2, R_{best}).

Input: *Three reactants* R_1, R_2, R_3.
$R_{new} = R_{best} + \lambda \times (R_1 - R_2)$.
Where $\lambda = Cauchyrnd$ (M_t, 0.1), *is a random number generated from a Cauchy distribution with location parameter* M_t *and*

scale parameter 0.1. *It is regenerated if the random number falls out of the range* [0, 2].
Output: *A new reactant* R_{new}.

3.3.3. Penta-Molecular Reactions

In penta-molecular reactions five reactants take part in the reaction to produce one product. Two penta-molecular reactions are considered using two other DE mutation strategies DE/rand/2 and DE/best/2 to combine the diversification property of the former one and intensification property of later one.

3.3.3.1. DErand2 Reaction

Here, DE/rand/2 mutation strategy is used to generate new reactants. In addition, the scale factor (F) used in DE mutation strategy (equivalent to reactant rate λ) is dynamically and self adaptively determined based on the problem. The rate of reaction (λ) is generated randomly from a Gaussian distribution with mean M and standard deviation 0.1. The value of M is initially set to 0.5 and self adaptively determined in the following manner.

$$S_f=0.9+0.1\times rand(0, 1)$$
$$M_{t+1}=S_f\times M_t+(1-S_f)\times mean\ (\lambda_{success}) \quad (5)$$

Where t= Number of times the reaction occurs, $\lambda_{success}$ stores the successful rate of reactions that generates better reactants, thereby glorifying the chances of generating better reaction rates consequently better reactants as more and more this reaction occurs. Here, instead of Cauchy distribution, Gaussian distribution is used because it generates most of the values within unity due to its short tail property [20]. The pseudo-code of the DErand2 reaction is described in Algorithm 5.

Algorithm 5: DErand2 (R_1, R_2, R_3, R_4, R_5).

Input: *Five reactants* R_1, R_2, R_3, R_4, R_5.
$R_{new}=R_1+\lambda\times(R_2-R_3)+\lambda\times(R_4-R_5)$.
Where λ=*Gaussianrnd*(M_t, 0.1), *is a random number generated from a Gaussian distribution with mean* M_t *and standard deviation* 0.1. *It is regenerated if the random number falls out of the range* [0, 1].
Output: *A new reactant* R_{new}.

3.3.3.2. DEbest2 Reaction

Here, DE/best/2 mutation strategy is used to generate new reactants. In addition, the scale factor (F) used in DE mutation strategy (equivalent to reactant rate λ) is dynamically and self adaptively determined similar to that of in DErand2 reaction. The pseudo-code of the DEbest2 reaction is described in Algorithm 6.

Algorithm 6: DEbest2 (R_1, R_2, R_3, R_4, R_{best}).

Input: *Five reactants* R_1, R_2, R_3, R_4, R_{best}.
$R_{new}=R_{best}+\lambda\times(R_1-R_2)+\lambda\times(R_3-R_4)$.
Where λ=*Gaussianrnd* (M_t, 0.1), *is a random number generated from a Gaussian distribution with mean* M_t *and standard deviation* 0.1. *It is regenerated if the random number falls out of the range* [0, 1].
Output: *A new reactant* R_{new}.

3.3.4. Greedy Reversible Reaction

In order to keep the number of reactants fixed throughout all iterations, a greedy reversible reaction between target reactant (R_j) and newly generated reactant (R_{new}) is carried out to select the better reactant. By keeping the reactant size fixed it makes the algorithm easier to implement. The pseudo-code of the greedy reversible reaction is elaborated in Algorithm 7.

Algorithm 7: Reversible (R_j, R_{new}).

Input: *Two reactants* R_j, R_{new}.
If enthalpy(R_{new})<*enthalpy*(R_j)
  Set R_j= R_{new}
end of if
Output: *The reactant* R_j.

4. Experimental Results

All simulations were carried out on a system with Intel ® core (TM) 2Duo E7500 CPU, 2.93GHz with 2GB RAM and implemented using SCILAB. Parity-p problems ($p\in$[3; 7]) are considered for comparative performance analysis. These problems are widely used and regarded as benchmarks for testing the generalization capability of training algorithms. To classify parity-p ($p\in$[3; 7]) problem, PSNs having structure p-p-1, threshold activation function at output layer and linear transfer function at hidden layer are considered. The population size is fixed to 10 for all the problems and algorithms. The results obtained from proposed method are compared with DE/rand//1/bin, DE/best/1/bin, CRO algorithm used for ANN training [30] and CRO-HONNT [23]. For DE algorithms the crossover probability C_r and scale factor F are fixed to 0.7 and 0.5 respectively.

The termination criterion applied to the training algorithms for parity-p ($p\in$[3; 4]) was the *MSE* on train set (0.025,0.0125 respectively); and for parity-p($p\in$[5; 7]) the networks were trained up to a maximum of 1000 generations or *MSE* not less than 0.125. The upper and lower bound of initial weight sets for parity-p problem is set to 2^p to -2^p. By making above experimental set up 1000 independent simulations using each method for each parity-p problem were conducted. To have a better comparison among the methods, Post Hoc analyses were performed on the results obtained from 1000 independent simulations for each problem using each method. Note that in each simulation the initial weight set for all the methods were kept same.

4.1. Performance Measure

To evaluate the effectiveness of the proposed training method with the other methods two performance and percentage of correct classification. The correct classification percentage is computed as follows:

$$CorrectClassification(\%) = \frac{\sum_{i=1}^{NOP} C_i}{NOP} \quad (6)$$

Where *NOP* is the number of testing patterns (was equal to the training set and each contain 2^p patterns); C_i-the coefficient representing the correctness of the classification of the i^{th} testing pattern which is determined as follows:

$$C_i = \begin{cases} 1, when\, Y_i = 1 and\, T_i = 1 \\ 1, when\, Y_i = -1 and\, T_i = -1 \\ 0, Otherwise \end{cases} \quad (7)$$

Where Y_i is the output of PSN and T_i is the target for i^{th} test pattern.

4.2. Discussions

One can see from Tables 1 and 2 that all the methods gave perfect generalization (100% correct classification) capability for parity 3 and 4 problems respectively. For both the problems the proposed method takes less number of generations (though statistically insignificant) to obtain the optimal solutions than CRO-HONNT, DE/rand/1/bin and DE/best/1/bin methods whereas takes statistically less number of generations than CRO method.

Table 1. Simulation results on parity 3 problem (best results in bold).

Algorithms	Generations			Correct Classification (%)		
	Mean ± St.D.	Min	Max	Mean ± St.D.	Min	Max
DE-CRO-HONNT	1.72 ± 1.45[a]	1	12	100 ± 0	100	100
CRO-HONNT	1.86 ± 1.64[a]	1	12	100 ± 0	100	100
CRO	2.65 ± 4.03[c]	1	65	100 ± 0	100	100
DE/rand/1	2.12 ± 1.52[b]	1	17	100 ± 0	100	100
DE/best/1	2.11 ± 1.46[b]	1	9	100 ± 0	100	100
Means within a column the same letter(s) are not statistically significant (p=0.05) according to duncan's multiple range test (SPSS V.16.0.1).						

Table 2. Simulation results on parity 4 problem (best results in bold).

Algorithms	Generations			Correct Classification (%)		
	Mean ± St.D.	Min	Max	Mean ± St.D.	Min	Max
DE-CRO-HONNT	16.98 ± 15.92[a]	1	102	100 ± 0	100	100
CRO-HONNT	17.41 ± 15.27[a]	1	187	100 ± 0	100	100
CRO	23.04 ± 40.49[b]	1	920	100 ± 0	100	100
DE/rand/1	18.21 ± 15.38[a]	1	193	100 ± 0	100	100
DE/best/1	18.79 ± 15.74[a]	1	163	100 ± 0	100	100
Means within a column the same letter(s) are not statistically significant (p=0.05) according to duncan's multiple range test (SPSS V.16.0.1).						

Table 3 shows the simulation results obtain on parity-5 problem. It can be observed that all methods gave 100% generalization most of the time but none of the methods gave 100% correct classification for all the 1000 independent simulations. The percentage of correct classification by proposed method is not statistical significant to that of DE/rand/1 and CRO-HONNT whereas statistically significant to that of DE/best/1/bin and traditional CRO method. However, the proposed method takes statistically less number of generations than other methods (except CRO-HONNT) to obtain the optimal solutions.

Table 3. Simulation results on parity 5 problem (best results in bold).

Algorithms	Generations			Correct Classification (%)		
	Mean ± St.D.	Min	Max	Mean ± St.D.	Min	Max
DE-CRO-HONNT	165.83 ± 158.63[a]	5	1000	99.92 ± 0.71[c]	93.75	100
CRO-HONNT	173.61 ± 160.95[a]	2	1000	99.87 ± 0.87[bc]	93.75	100
CRO	194.45 ± 235.14[b]	6	1000	99.67 ± 1.43[a]	87.50	100
DE/rand/1	245.30 ± 227.84[c]	10	1000	99.82 ± 1.03[bc]	93.75	100
DE/best/1	248.62 ± 224.79[c]	5	1000	99.79 ± 1.15[b]	87.50	100
Means within a column the same letter(s) are not statistically significant (p=0.05) according to duncan's multiple range test (SPSS V.16.0.1).						

Tables 4 and 5 show the experimental results for parity 6 and 7 problems respectively. It is clearly observed that none of the methods gave perfect generalization capability for both problems throughout all 1000 simulations. The proposed method not only provides statistically better generalization capability (correct classification percentage) but also takes statistically significantly less number of generations to attain the solutions than the other methods considered.

To have a better idea regarding the performance (Convergence) of the proposed training algorithm with respect to other four algorithms, a comparative performance was plotted as shown in Figure 2 for parity 7 problem showing *MSE* error on train set, which again evidenced the superiority of proposed training algorithm (DE-CRO-HONNT).

Table 4. Simulation results on parity 6 problem (best results in bold).

Algorithms	Generations			Correct Classification (%)		
	Mean ± St.D.	Min	Max	Mean ± St.D.	Min	Max
DE-CRO-HONNT	360.24 ± 258.08[a]	15	1000	99.13 ± 3.56[d]	81.250	100
CRO-HONNT	783.49 ± 275.93[d]	28	1000	97.58 ± 3.20[c]	81.250	100
CRO	728.97 ± 340.57[c]	23	1000	94.02 ± 3.69[a]	78.125	100
DE/rand/1	535.43 ± 332.98[b]	29	1000	95.12 ± 5.52[b]	78.125	100
DE/best/1	547.46 ± 336.36[b]	30	1000	95.21 ± 5.30[b]	78.125	100
Means within a column the same letter(s) are not statistically significant (p=0.05) according to duncan's multiple range test (SPSS V.16.0.1).						

Table 5. Simulation results on parity 7 problem (best results in bold).

Algorithms	Generations			Correct Classification (%)		
	Mean ± St.D.	Min	Max	Mean ± St.D.	Min	Max
DE-CRO-HONNT	487.02 ± 281.45[a]	35	1000	97.23 ± 6.37[d]	75.00	100
CRO-HONNT	991.48 ± 64.87[c]	203	1000	90.42 ± 3.41[c]	71.87	100
CRO	995.42 ± 36.47[c]	620	1000	81.56 ± 4.79[a]	70.31	100
DE/rand/1	714.35 ± 300.42[b]	150	1000	88.08 ± 11.61[b]	71.87	100
DE/best/1	707.16± 357.50[b]	152	1000	86.46± 11.17[b]	71.87	100
Means within a column the same letter(s) are not statistically significant (p=0.05) according to duncan's multiple range test (SPSS V.16.0.1).						

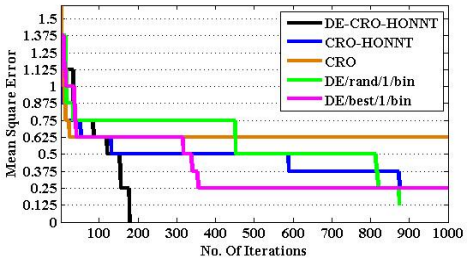

Figure 2. DE-CRO-HONNT, CRO-HONNT, CRO, DE/rand/1/bin and DE/best/1/bin algorithms *MSE* value on train set of parity 7 problem up to termination criteria.

5. Conclusions

In this paper, a hybrid DE-CRO-HONNT training algorithm for PSN is developed. In this algorithm adaptive DE mutation strategies are hybridized as inter-molecular reactions in CRO algorithm. The proposed algorithm enjoys both the intensification property of DE/best mutation strategies along with uni-molecular reaction and diversification property of DE/rand mutation strategies. Thus, a trade between intensification and diversification is implicitly maintained. In addition, control parameters for DE mutation strategies are dynamically and self adaptively determined based on the problem in hand. The simulation results demonstrate that the proposed training algorithm has superior performance in terms of correct classification percentage and generations taken to attain the solutions when compared with most popular DE variants, traditional CRO method and CRO-HONNT method. It is also observed that the performance of other methods drops off sharply with the increase in number of parity bits whereas the same is not the case for DE-CRO-HONNT. Therefore, it can be concluded that, the use of DE-CRO-HONNT method incorporates efficient and effective searching mechanisms, such that it has less chance to trap to local minima and thus enhance the higher order neural network training procedure.

References

[1] Beyer H. and Schwefel H., "Evolutionary Strategies: A Comprehensive Introduction," *Natural Computing*, vol. 1, no. 1, pp. 3-52, 2002.

[2] Das S. and Suganthanam P., "Differential Evolution: A Survey of the Sate-of-the-Art," *IEEE Transaction on Evolutionary Computation*, vol. 15, no.1, pp. 4-31, 2011.

[3] Das S., Abraham A., Chakraborty U., and Konar A., "Differential Evolution Using a Neighbourhood Based Mutation Operator," *IEEE Transaction on Evolutionary Computation*, vol. 13, no. 3, pp. 526-553, 2009.

[4] Draa A., Meshoul S., Talbi H., and Batouche M., "A Quantum-Inspired Differential Evolution Algorithm for Solving the N-Queens Problem," *The International Arab Journal of Information Technology*, vol. 7, no. 1, pp. 21-27, 2010.

[5] Epitropakis M., Plagianakos V., and Vrahatis M., "Hardware-friendly Higher-Order Neural Network Training Using Distributed Evolutionary Algorithms," *Applied Soft Computing*, vol. 10, no. 2, pp. 398-408, 2010.

[6] Ghosh J. and Shin Y., "Efficient Higher-Order Neural Networks for Classification and Function Approximation," *International Journal of Neural Systems*, vol. 3, no. 4, pp. 323-350, 1992.

[7] Goldberg D., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publisher, 1989.

[8] Han K. and Kim J., "Quantum-Inspired Evolutionary Algorithm for a Class of Combinatorial Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 6, pp. 580-593, 2002.

[9] Huang D., Ip H., Law K., and Chi Z., "Zeroing Polynomials Using Modified Constrained Neural Network Approach," *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 721-732, 2005.

[10] Hussain A., Knowles A., Lisboa P., El-Deredy W., and Al-Jumeily D., "Polynomial Pipelined Neural Network and its Application to Financial Time Series Prediction," *in Processing of AI 2006: Advances in Artifcial Intelligence*, pp. 597-606, 2006.

[11] Islam S., Das S., Ghosh S., Roy S., and Suganthan P., "an Adaptive Differential Evolution Algorithm with Novel Mutation and Crossover Strategy for Global Numerical Optimization," *IEEE Transaction on System, Man, and Cybernetics-PART B: Cybernetics*, vol. 42, no. 2, pp.482-500, 2012.

[12] Karnavas Y. and Papadopoulos D., "Excitation Control of a Synchronous Machine using Polynomial Neural Networks," *Journal of Electrical Engineering*, vol. 55, no. 7, pp. 169-179, 2004.

[13] Kennedy J. and Eberhart R., *Swarm Intelligence*, Morgan Kaufmann Publisher Inc., 2001.

[14] Lam A. and Li V., "Chemical Reaction Optimization: a Tutorial," *Memetic Computing*, vol. 4, no. 1, pp. 3-17, 2012.

[15] Lam A. and Li V., "Chemical-Reaction-Inspired Metaheuristic for Optimization," *IEEE Transaction on Evolutionary Computation*, vol. 14, no. 3, pp. 381-399, 2010.

[16] Lam A., Li V., and Yu J., "Real-Coded Chemical Reaction Optimization," *IEEE Transaction on Evolutionary Computation*, vol. 16, no. 3, pp. 339-353, 2012.

[17] Liatsis P. and Hussain A., "Nonlinear One-Dimensional DPCM Image Prediction Using Polynomial Neural Networks," *in Proceeding of*

SPIE: Applications of Artificial Neural Networks in Image Processing IV San Jose, CA, pp. 58-68, 1999.

[18] Perantonis S., Ampazis N., Varoufakis S., and Antoniou G., "Constrained Learning in Neural Networks: Application to Stable Factorization of 2-d Polynomials," *Neural Processing Letter*, vol. 7, no. 1, pp. 5-14, 1998.

[19] Pham D., Ghanbarzadeh A., Koc E., Otri S., Rahim S., and Zaidi M., "The Bees Algorithm-A Novel Tool for Complex Optimization Problems," *in Processing of 2nd International Virtual Conference on Intelligent Production Machines and Systems*, UK, pp.454-459, 2006.

[20] Price K., *New Ideas in Optimization*, McGraw-Hill Ltd., pp. 79-108, 1999.

[21] Price K., Storn R., and Lampinen J., *Differential Evolution-A Practical Approach to Global Optimization*, Springer, 2005.

[22] Rahnamayan S., Tizhoosh H., and Salama M., "Opposition Based Differential Evolution," *IEEE Transaction on Evolutionary Computation*, vol. 12, no. 1, pp. 64-79, 2008.

[23] Sahu K., Panigrahi S., and Behera H., "A Novel Chemical Reaction Optimization Algorithm for Higher Order Neural Network Training," *Journal of Theoretical and Applied Information Technology*, vol. 53, no. 3, pp. 402-409, 2013.

[24] Shin Y. and Ghosh J., "The Pi-Sigma Network: An Efficient Higher-Order Neural Network for Pattern Classification and Function Approximation," *in Proceedings of Neural Networks, IJCNN-91-Seattle International Joint Conference on Neural Networks*, Seattle, pp. 13-18, 1991.

[25] Shin Y. and Ghosh J., "Realization of Boolean Functions Using Binary Pi-Sigma Networks," *in Proceedings of Conference on Artificial Neural Networks in Engineering*, pp. 205-210, 1991.

[26] Shin Y. and Ghosh J., "Ridge Polynomial Networks," *IEEE Transactions on Neural Networks*, vol. 6, no. 3, pp. 610-622, 1995.

[27] Socha K. and Doringo M., "Ant Colony Optimization for Continuous Domains," *European Journal of Operation Research*, vol. 185, no. 3, pp. 1155-1173, 2008.

[28] Storn R. and Price K., "Differential Evolution-A Simple and Efficient Heuristic for Global Optimization Over Continuous Spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341-359, 1997.

[29] Voutriaridis C., Boutalis Y., and Mertzios G., "Ridge Polynomial Networks in Pattern Recognition," *in Proceedings of 4th EURASIP Conference Focused on Video/Image Processing and Multimedia Communications*, pp. 519-524, 2003.

[30] Yu J., Lam A., and Li V., "Evolutionary Artificial Neural Network Based on Chemical Reaction Optimization", *in Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, LA, pp. 2083-2090, 2011.

Sibarama Panigrahi received the B.Tech degree in Computer Science and Engineering from Biju Patnaik University of Technology, M.Tech Degree in Computer Science and Engineering from Veer Surendra Sai University of Technology (VSSUT) Burla respectively in 2009 and 2013. He is currently an Assistant Professor in the School of Computer Science in National Institute of Science and Technology, Odisha, India. He has received "University Silver Medal for Best Computer Science and Engineering Post Graduate for the year 2013" from VSSUT Burla. Author has published some research papers in reputed Journals. His research interests are focused on Time Series Forecasting, Pattern Recognition, Machine Learning, Evolutionary Computation and Neural Networks.