HIOPGA: A New Hybrid Metaheuristic Algorithm to Train Feedforward Neural Networks for Prediction

Masoud Yaghini[1], Mohammad M. Khoshraftar[2], Mehdi Fallahi[3]
[1]School of Railway Engineering, Iran University of Science and Technology, Tehran, Iran
[2] School of Railway Engineering, Company / University of Science and Technology, Tehran, Iran
[3] School of Railway Engineering, Company / University of Science and Technology, Tehran, Iran

Abstract - *Most of neural network training algorithms make use of gradient-based search and because of their disadvantages, researchers always interested in using alternative methods. In this paper to train feedforward, neural network for prediction problems a new Hybrid Improved Opposition-based Particle swarm optimization and Genetic Algorithm (HIOPGA) is proposed. The opposition-based PSO is utilized to search better in solution space. In addition, to restrain model overfit with training pattern, a new cross validation method is proposed. Several benchmark problems with varying dimensions are chosen to investigate the capabilities of the proposed algorithm as a training algorithm. The result of HIOPGA is compared with standard backpropagation algorithm with momentum term.*

Keywords: PSO, GA, Prediction, Hybrid Algorithm

1 Introduction

Neural network (NN) is one of the most important data mining techniques. It is used with both supervised and unsupervised learning [1]. Training NN is a complex task of great importance in problems of supervised learning. Most of NN training algorithms make use of gradient-based search. These methods have the advantage of the directed search, in that weights are always updated in such a way that minimizes the error, which called NN learning process. However, there are several negative aspects with these algorithm such as dependency to a learning rate parameter, network paralysis, slowing down by an order of magnitude for every extra (hidden) layer added and complex and multi-modal error space, Therefore, these algorithms most likely gets trapped into a local minimum, making them entirely dependent on initial (weight) settings which make the algorithms not guaranteed to be universally useful [2]. Metaheuristic global search strategy makes them able to avoid being trapped into secondary peak of performance and can therefore provide effective and robust solution to the problem of NN and training [3]. Metaheuristics have the advantage of being applicable to any type of NN, feedforward or not, with any activation function, differentiable or not [2]. Metaheuristics provide acceptable solutions in a reasonable time for solving hard and complex problems; they are particularly useful for dealing with large complex problems, which generate many local optima. They are less likely to be trapped in local minima than traditional gradient-based search algorithms. They do not depend on gradient information and thus are quite suitable for problems where such information is unavailable or very costly to obtain or estimate [4]. The outline of this paper is as follows. Section 2 presents literature review about metaheuristic algorithm for training NN. In section 3, the proposed particle and chromosome, criterion for accuracy evaluation, component and operator of the proposed algorithm, the proposed cross validation, steps of the algorithm, and the termination criterions is completely described. In section 4, experimental results, the value of parameter and convergence graph is presented. In section 5 summery, conclusion and some hints for the future research is given.

2 Literature Review

Metaheuristic algorithms for training NN could divide into single-solution based and population-based algorithms (S-Metaheuristic and P-Metaheuristic). In training NN with S-Metaheuristic [5], [6] used tabu search approach and [7], [8] used simulated annealing approach. One could divide NN training with P-Metaheuristic into two main groups, which are train with Evolutionary Algorithms (EA) and train with swarm intelligence algorithms, respectively. Learning and evolution are two fundamental forms of adaptation. There has been a great interest in combining learning and evolution with NN and combinations between NN's and EA's can lead to significantly better intelligent systems than relying on NN's or EA's alone [9]. In Training NN with EA [10] and [11] make a comparison among proposed EA and a gradient-based algorithm, [12] and [13] combine EA with gradient-based local search algorithm to obtain better result. Another class of P-Metaheuristic, which is used as training algorithm, is swarm intelligence. They originated from the social behavior of those species that has a common target (e.g. compete for foods) [4]. Among swarm intelligence inspired optimization algorithms Particle Swarm Optimization (PSO) is one the most successful one. Unlike Genetic Algorithm (GA), PSO has no complicated evolutionary operators such as crossover, selection, and mutation and it is highly dependent on stochastic processes [2]. The PSO was introduced by [14] for the first time. [15] proposed a method to employ PSO in a cooperative configuration which is achieved by splitting the input vector into several sub-vectors, each which is optimized cooperatively in its own swarm.[16] and

[17] make use of PSO to train neural network. In these research authors just use a very simple problem that did not reveal outperformance of their method.[18] presents a modified PSO which adjust the trajectories (positions and velocities) of the particle based on the best positions visited earlier by themselves and other particles, and also incorporates population diversity method to avoid premature convergence. [19] analyzes the use of the PSO algorithm and two variants with a local search operator. [20] use multi-phase PSO algorithm (MPPSO) which simultaneously evolves multiple groups of particles that change their search criterion when changing the phases, and also incorporates hill-climbing.

In addition to the modifications made to basic PSO algorithm, a variety of other PSO variations have also been developed. Among these variations are those which incorporate opposition-based learning into PSO is capable of delivering better performance as compared to the standard PSO. Opposition-based learning was first introduced by [21] later applied to PSO. Opposition-based learning is based on the concept of opposite points and opposite numbers. [22] proposed a modified PSO algorithm for noisy problems which utilized opposition-based learning. [23] proposed an opposition-based comprehensive learning PSO which utilized opposition-based learning for swarm initialization and for exemplar selection. [24] Presented the improved PSO which utilized a simplified form of opposition-based learning. In this approach, the particle having worst fitness in each iteration is replaced by its opposite particle. Opposition-based learning was only applied to one particle instead of the whole swarm and was also not used at the time initialization. Apart from PSO researcher employed other swarm intelligence but none of the is successful as PSO. [25] presented a continuous version of ACO algorithm (i.e., ACO_R) also [26] proposed a novel hybrid algorithm based on Artificial Fish Swarm Algorithm and PSO both compare their proposed algorithm with specialized gradient based algorithms for NN training.

3 The Proposed Algorithm

3.1 Proposed Particle and Chromosome

A good detail on basic version PSO algorithm is in [27] and for GA is in [4]. In this research we employ fully connected layered feedforward networks. All units have a bias except for input units. In the proposed algorithm (HIOPGA) to utilize a combination of PSO and GA a structure as Fig. 1 is employed. For simplification in this figure a NN with one hidden layer, three input units, one hidden units and two output units is considered.

3.2 Criterion for Accuracy Evaluation

For classification problems, *classification error percentage* (CEP) is utilized as shown in (1) and (2) to evaluate the accuracy. op and tp are predicted value and target value, p is input pattern and P is the number of pattern.

$$\psi(\vec{p}) = \begin{cases} 1 & if \quad \vec{o}_p \neq \vec{t}_p \\ 0 & otherwise \end{cases} \tag{1}$$

$$CEP = 100 \times \left(\sum_{p=1}^{P} \psi(p) \Big/ P \right) \tag{2}$$

For approximation problem *Normalized Root Mean Squared Error* (NRMSE) is utilized as shown in (3) and (4).where N is number of the output units, P number of pattern, opi and tpi are predicted value and target value of ith output unit for pattern p.

$$RMSE = \sqrt{\sum_{p=1}^{P} \sum_{i=1}^{N} \left(t_{pi} - o_{pi}\right)^2 \Big/ P \times N} \tag{3}$$

$$NRMSE = 100 \times \frac{RMSE}{\sum_{p=1}^{P} \sum_{i=1}^{N} t_{pi} \Big/ P \times N} \tag{4}$$

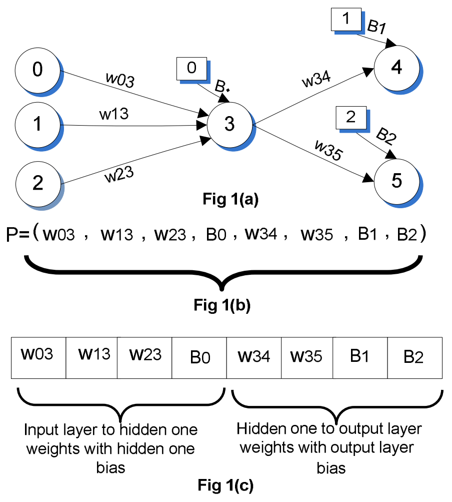
0
1
2
3
4
5
w03
w13
w23
w34
w35
0
B
1
B1
2
B2
Fig 1(a)
P=(w03, w13, w23, B0, w34, w35, B1, B2)
Fig 1(b)
W03
W13
W23
B0
W34
W35
B1
B2
Input layer to hidden one weights with hidden one bias
Hidden one to output layer weights with output layer bias
Fig 1(c)

Fig. 1. (a) a NN structure (b) the particle for PSO (c) the chromosome for GA.

3.3 Improved PSO

Although PSO is capable of locating a good solution at a significantly fast rate, but its ability to fine tune the optimum solution is comparatively weak, mainly due to the lack of diversity at the end of the evolutionary process. To improve the search ability of standard PSO time-varying parameter is utilized. Suppose that *t* and *m* is current and final iteration number and $C_1(t)$, $C_2(t)$, $C_1(m)$ and $C_2(m)$ are cognitive and social component of current and final iteration then time-varying parameter is calculated as (5) and (6). If each of parameter reaches to final values, it set to initial value again. By using the time-varying parameter, we can implement large cognitive component and small social component at the beginning of the search to guarantee particles' moving around the search space and to avoid particles moving toward the population best position. On the other hand, a small cognitive component and a large social component allow the particles to converge to the global optima in the latter of the search [28]. *K* is another parameter that utilized along with these parameter and called *constriction coefficient* with the hope that it can insure a PSO to converge [29]. *K* is calculated as (6), $\phi(t) = C_1(t) + C_2(t)$ and $\phi(t) \geq 4$.

$$C_1(t+1) = \frac{t}{m} \times \left(C_{1m} - C_1(t)\right) + C_1(t) \qquad (4)$$

$$C_2(t+1) = \frac{t}{m} \times \left(C_{2m} - C_2(t)\right) + C_2(t) \qquad (5)$$

$$K(t) = 2 \Big/ \left| 2 - \phi(t) - \sqrt{\phi^2(t) - 4\phi(t)} \right| \qquad (6)$$

3.4 Opposition-based Learning Components

The proposed *HIOPGA* implement this method two ways. First, after population initialization, to start with a better population, the algorithm calculate the opposite position and velocity of each particle, then for each particle the better one (current particle or its opposite) is inserted into the population. Second, during the iteration, when the algorithm finds a new velocity and position for a particle, the opposite position and velocity of each particle is calculated and the better one is inserted into the current generation. When creating opposite particles an important question that arises is, what should be the velocity of these particles? Either we can have the same velocity as that of the original particle or we can randomly reinitialize the velocity. Alternatively, we can calculate the opposite of the velocity of the original particle. We cannot use the velocity of the original particle because that velocity was calculated using the current position of the original particle which would be invalid for the opposite particle. Reinitializing the opposite particles velocity randomly is not such an inviting option because we would not be taking advantage of the experience gained by original particle. Other researchers have not investigated this question and use random initialization of velocity. We have decided to use the opposite velocity of the original particle. We believe that by using opposite velocity we would be able to achieve better performance as we do with utilizing opposite positions. The opposite velocity is calculated in exactly the same way as we calculate the opposite particles. The pseudocode of opposite particle calculation is illustrated in Fig 2. [x_{min}, x_{max}] is the initial interval of the particle position (initial weight of NN) and [v_{min}, v_{max}] is the velocity interval. The poisons and velocity of *i*th particle at iteration *t* are $X_i(t)=(x_{i1}(t),\ldots,x_{id}(t))$ and $V_i(t)=(v_{i1}(t),\ldots,v_{id}(t))$.

```
For all particles i
 For all dimensions d
   opposite position: $\breve{x}_{id}(t) = x_{\max} + x_{\min} - x_{id}(t)$
   opposite velocity: $\breve{v}_{id}(t) = v_{\max} + v_{\min} - v_{id}(t)$
   if( $\breve{v}_{id}(t) >= v_{\max}$ ) then $\breve{v}_{id}(t) = v_{\max}$
   if( $\breve{v}_{id}(t) <= v_{\min}$ ) then $\breve{v}_{id}(t) = v_{\min}$
   if ( $E(\breve{X}_i(t)) < E(X_i(t))$ ) then $\breve{x}_{id}(t) = \breve{x}_{id}(t)$ and $v_{id}(t) = \breve{v}_{id}(t)$ ;
  Update ($x_i$(t),$v_i$(t));
 End for d
End for i
```

Fig. 2. Pseudocode of opposite particle calculation

3.5 Random Perturbation

PSO can quickly find a good local solution but it sometimes suffers from stagnation without an improvement [28]. Therefore, to avoid this drawback of basic PSO, the velocity of particles is reset in order to enable particles to have a new momentum. Under this new strategy, when the global best position is not improving with the increasing number of generations, each particle *i* will be selected by a predefined probability (0.5 in this study) from the population, and then a random perturbation is added to each dimension v_{id} (selected by a predefined probability 0.5 in this study) of the velocity vector v_i of the selected particle *i*. The velocity resetting is presented as follows: where r_1, r_2 and r_3 are separately generated, uniformly distributed random numbers in range (0, 1), and v_{max} is the maximum magnitude of the random perturbation to each dimension of the selected particle.

```
For all particles i
Generate a random number( $r_1 \in [0,1]$ )
   If ($r_1$>0.5) then select particle i;
    For all dimensions of selected particle i
      Generate a random number ( $r_2 \in [0,1]$ )
      If ($r_2$>0.5) then $vi_d = v_{\max} \times (2r_3 - 1) + v_{id}$
       If ( $vi_d > V_{\max}$ ) then $vi_d = V_{\max}$
       If( $\breve{v}_{id}(t) <= v_{\min}$ ) then $\breve{v}_{id}(t) = v_{\min}$
  End for d
End for i
```

Fig. 3. Pseudocode of opposite particle calculation

3.6 Evolutionary Operators

All illustrations, on some evolutionary schemes of GA, several effective mutation and crossover operators have been proposed for PSO. [30] proposed a crossover operator, and [31] proposed a Gaussian mutation operator to improve the performance of PSO. Utilizations of these operators in PSO have potential to achieve faster convergence and to find better solutions. During iteration of HIOPGA, if the best personal position for each particle *i* (*Pbest$_i$*) is not improved for *maxPbestPersistence* successive iteration, we suppose that these particles are get stuck in local minima of the problem. Therefore, crossover and mutation operator is utilized to improve the performance of algorithm and obviate the aforementioned problem. These trapped particles establish a sub-population of particle. Then according to Stochastic Universal Sampling (SUS) two individual are selected. In addition, a crossover point is selected randomly. For example, for a NN with one hidden layer, crossover point is a number between 1 and 2, for a NN with two hidden layers crossover point is a number between 1 and 3, and for a NN with three hidden layer crossover point is a number between 1 and 4. To make new offspring if 1 is considered as crossover point, then connections between input layer and hidden layer one is used for crossover operator. Suppose *parent*$_1$(x_i) and *parent*$_2$(x_i) is the *i*th component of selected individuals. The crossover operator is conducted by the (7) for position crossover and the (8) for velocity crossover ($r_i \in (0,1)$).

$$\begin{aligned} Child_1(x_i) &= r_i \times Parent_1(x_i) + (1-r_i) \times (Parent_2(x_i)) \\ Child_2(x_i) &= (1-r_i) \times Parent_1(x_i) + r_i \times (Parent_2(x_i)) \end{aligned} \quad (7)$$

$$\begin{aligned} Child_1(v_i) &= \left|Parent_1(v_i)\right| \times \frac{(Parent_1(v_i) + Parent_2(v_i))}{\left|Parent_1(v_i) + Parent_2(v_i)\right|} \\ Child_1(v_i) &= \left|Parent_2(v_i)\right| \times \frac{(Parent_1(v_i) + Parent_2(v_i))}{\left|Parent_1(v_i) + Parent_2(v_i)\right|} \end{aligned} \quad (8)$$

In each generation, the mutation operator is conducted by the (9), $r_i \in (-1,1)$. (*m-t*)/*m* makes algorithm to have great jump at the early phase of algorithm and small jump at the latter phase.

$$\begin{aligned} Child_1(x_i) &= \frac{m-t}{m} \times r_i \times x_{\max} + Parent(x_i) \\ Child_1(v_i) &= \frac{m-t}{m} \times r_i \times v_{\max} + Parent(v_i) \end{aligned} \quad (9)$$

3.7 The Proposed Cross Validation Method

The training error of an NN may reduce as its training process progresses. However, at some point, usually in the later stages of training, the NN may start to take advantage of idiosyncrasies in the training data. Consequently, its generalization performance may start to deteriorate even though the training error continues to decrease. Early stopping in cross validation [32] is one common approach to avoid overfitting. In this method, the training data is divided into training and validation sets. The training process will not terminate when the training error is minimized instead it stopped when the validation error starts to increase. This termination criterion is deceptive because the validation set may contain several local minima. In HIOPGA to decrease negative effect of multimodal validation space on model generalization ability, we use a simple criterion that terminates the training process of the NN. At the end of each *L* training iterations, the validation error is evaluated and the process repeated, when this error increases for *T* successive times in comparison to first *L* training iterations (independent of how large the increases actually are), training process is terminated. The idea behind the termination criterion is to stop the training process of the NN when its validation error increases not just once but during *T* consecutive times. It can be assumed that such increases indicate the beginning of the final overfitting not just the intermittent.

3.8 The Steps of HIOPGA

The steps of HIOPGA are explained as follows.

Step1) according to initial value of parameters, specify starting position and velocity of particles. Set iteration counter to zero (*iter* =0).

Step 2) to establish a better population, calculate opposite position and velocity of particles, and insert better particle into population (Pseudocode in Fig. 2).

Step 3) for each particle specify best personal position and calculate the number of no improvements in it ($pbest_i$ and $pbest_iCounter$). Also, specify best global position and calculate the number of no improvements in it (*gbest* and *gbsetCounter*).

Step 4) if ($E_{train}(gbest(iter)) < \varepsilon$) then go to step 28, otherwise go to Step 5.

Step 5) calculate the best global position of particle validation error ($E_{val}(gbest(iter))$).

Step 6) according to equation (4), (5), and (6) calculate $C_1(iter+1)$, $C_1(iter+1)$ and $K(iter+1)$.

Step 7) calculate new position and velocity of particles (training by PSO).

Step 8) for each particle, specify best personal position and calculate the number of no improvements in it ($pbest_i$ and $pbest_iCounter$). Also, specify best global position and calculate the number of no improvements in it (*gbest* and *gbsetCounter*).

Step 9) if ($E_{train}(gbest(iter)) < \varepsilon$) then go to Step 28 otherwise go to Step 10.

Step 10) do the proposed cross validation.

Step 11) identify a sub-population for genetic algorithm by specifying the particles that number of no improvements in the best personal position is greater than maximum allowed number ($pbest_iCounter$>max*Pbest*) and go to Step12.

Step 12) if the number of individuals in sub-population is grater than 1+[*m*/*iter*] then go to Step 13, otherwise go to Step 23.

Step 13) set genetic counter to zero (*GeneticCounter*=0) **Step 14)** select two individuals by using SUS methods.

Step 15) call crossover operator using equation (7) and (8), and the parents are replaced with their better offsprings in main population.

Step 16) call mutation operator using equation (9) and the new better offspring take the place of its parents in main population.

Step 17) if number of individual in the new generation is equal to number of individual in the current generation go to Step 18, otherwise, go to Step 14.

Step 18) establish next generation with best individual in the current and new generation and add one to genetic algorithm counter (*GeneticCounter*++)

Step 19) if genetic algorithm counter is equal to number of individual in the sub-population go to Step 20, otherwise go to Step 14.

Step 20) for each particle, specify best personal position and calculate the number of no improvements in it ($pbest_i$ and $pbest_iCounter$). Also, specify best global position and calculate the number of no improvements in it (*gbest* and *gbsetCounter*).

Step 21) if ($E_{train}(gbest(iter)) < \varepsilon$) then go to Step 28, otherwise, go to Step 22.

Step 22) if the number of no improvements in the best global position is greater than maximum allowed number (*gebestCounter*>*Maxgbest*) then call random perturbation (Pseudocode in Fig. 3)

Step 23) increase iteration counter (*iter*++).

Step 24) if iteration counter is greater than maximum allowed number (*iter*>*m*) then go to Step 28, otherwise, go to Step 25.

Step 25) if the remaining for number of iteration divided by the proposed cross validation strip length (*iter*/*L*) become zero go to Step 27, otherwise, go to Step 2.

Step 26) if validation error of global position for the current iteration is grater than pervious iteration ($E_{val}(gbest(iter))$> E_{val}

(*gbest*(*iter*-1)) then increase overtraining counter (*Tcounter*++), otherwise set it to zero (*Tcounter* =0).

Step 27) if overtraining counter is greater than maximum allowed number (*Tcounter*>*T*) then go to step 28 (termination because of overfitting), otherwise go to Step 2. **Step 28)** stop the training.

3.9 Termination Criterion

The algorithm simultaneous uses three criterions as termination conditions. First termination condition is based on training error. In this approach, at the end of each iteration *t* if the error on the training pattern is less than ε the training process will terminate (for the classification problems $\varepsilon = 10^{-2}$ for the approximation problems $\varepsilon = 10^{-6}$). Second, if number of iteration becomes greater than a predefined number, the training process will be terminated. Third, according to the proposed cross validation method, if the algorithm meets the over training condition, the training process will be terminated.

4 Experimental Studies

In this section, a comparison between the performance of HIOPGA and backpropagation algorithm (BP) with momentum term on several well-known benchmark problems is presented. The characteristics of these problems are summarized in Table 1, which show a considerable diversity in the number of examples, attributes, and classes. The detailed description of these problems can be obtained from the University of California Irvine, Machine Learning Repository. For each benchmark problem, entropy is calculated according to (10). Where $P(C_i)$ is the probability of class C_i in the data set, determined by dividing the number of pattern of class C_i by the total number of pattern in data set. Entropy of a data set is the average amount of information needed to identify the class label of a pattern in data set. In fact, entropy explores class distribution information in its calculation and show impurity of data set. It could considered as a criterion for difficulty of the problems.

$$E = -\sum_i P(C_i) \times \log_2 (P(C_i)) \qquad (10)$$

TABLE 1
CHARACTERISTICS OF BENCHMARK PROBLEMS

problem	Number of Input attributes	Output classes	Training pattern	Validation pattern	Testing pattern	Entropy
Cancer	9	2	350	175	174	0.93
Card	51	2	345	173	172	0.99
Diabetes	8	2	348	192	192	0.93
Glass	9	6	107	54	53	2.18
Heart	35	2	460	230	230	0.99
Horse	58	3	182	91	91	1.32
Iris	4	3	75	38	37	1.58
Mushroom	125	2	4062	2031	2031	1
Thyroids	21	3	3600	1800	1800	0.45

4.1 Experimental Methodology

The proposed algorithm is implemented with Java programming language and a personal computer with Intel(R) Pentiume (R) CPU 2.66 GHz 2.68GHz, 32 Bits Windows 7 Ultimate operating system and 1.50 GB installed memory (RAM) is used to achieve all of the results. In both algorithm (HIOPGA and BP) we consider 150 as maximum number of iteration and observably both algorithm has less iteration to converge. The metaheuristic and gradient-based algorithms are sensitive to the value of their parameters. The parameters are the configurable components of HIOPGA and BP. Parameter tuning may allow a larger flexibility and robustness to the algorithm, but requires a careful initialization. Those parameters may have a great influence on the efficiency and effectiveness of the search. It is not obvious to define a priori which parameter setting should be used. The optimal values for the parameters depend mainly on the problem and even the instance to deal with and on the search time that the user wants to spend in solving the problem. One main step of this research is fine parameter tuning of the algorithm. To tune the parameters of each algorithm, by random we selected three benchmark problems with different sizes and only one parameter is modified at a time for each algorithm, while the other are not changed. Then proper values of the parameters determined through running the algorithm 15 times over different values of the parameters and calculating average of the objective function for these 15 runs. The criteria for modifying parameters are the quality of solutions and CPU time to find them. The final value for the parameters is as follows. Initial NN weights (initial position of particles or x_{min} and x_{max}) is [-1,1], the velocity interval is [-3,3], number of particles (n_p) is 15, initial and final value for cognitive component are 5 and 1, respectively. Initial and final value for social component are 1 and 5. GA mutation probability (P_m) is 0.02 and crossover probability (P_c) is 0.7. For the proposed cross validation method, *L* and *T* are 5 and 3. In addition, the learning rate and momentum term for BP are 0.1 and 0.9, respectively. The max number of training epochs, i.e., *m*, for both algorithms is set to 150. One bias neuron with a fixed input +1 was connected to the neurons of the hidden layers and output layers. The logistic sigmoid function was used for the neurons in the hidden layers and output layer.

4.2 Experimental Results

To reduce the effect of random parameter initialization on prediction ability of the models, we run each model 100 times, independently and take the average and standard deviation of results in table 2. The BP needs much more time and iteration to converge but less disperses solution. The reason for this matter is the random essence of metaheuristic algorithm.

4.3 The Convergence Graph

Fig. 4(a)-4(i) represent number of iteration-testing error for the best so far network from the beginning of the algorithms. As these figures reveal, the proposed HIOPGA with doing big jump in the testing space to find better solutions converge faster than BP.

5 Conclusion

In this research, a hybrid algorithm based on particle swarm optimization and genetic algorithm was presented. During iteration of the proposed algorithm, when some NN (or particles position) in the *d*-dimensional space cloud not be improved through the IOPSO, a sub-population of such NNs is established and be sent to the GA, to with utilizing the GA crossover and mutation operators, the HIOPGA finds better NN for replacing in the population. The comparison between the proposed algorithm and standard BP with momentum term reveals the superiority of the algorithm, however HIOPGA in the final latter steps and tuning the final solution need more iteration than BP. For example, in Cancer problem, the testing error of the proposed algorithm in iteration 40 was 4.02 and 10 iterations later i.e. at the iteration 50 error decreased to 2.99, but in the BP the testing error in iteration 79 is 3.21 and in the iteration 80 is 3.

TABLE 2
PERFORMANCE OF HOIPGA ON NINE BENCHMARK PROBLEMS FOR DIFFERENT PARAMETER VALUES. ALL RESULTS WERE AVERAGED OVER 50 INDEPENDENT RUNS

Name of		Model Accuracy on						Number of Iteration		Average of Training Time(second)
Problem	Algorithm	Training Set		Validation Set		Test Set		Mean	SD	
		Mean	SD	Mean	SD	Mean	SD			
Cancer	HOIPGA	98.23	1.13	97.21	1.26	97.01	1.22	50.06	7.23	1.43
	BP	98.11	1.01	97.23	1.00	97.00	1.02	79.11	6.54	2.25
Card	HOIPGA	87.02	2.24	86.09	2.21	86.92	1.42	25.32	2.45	6.94
	BP	87.12	1.89	87.09	1.64	86.85	1.33	45.75	2.32	12.38
Diabetes	HOIPGA	69.19	1.23	69.11	1.14	68.87	1.13	26.59	1.45	1.68
	BP	68.99	0.98	68.87	1.09	68.54	0.95	45.11	1.40	2.81
Glass	HOIPGA	68.42	1.33	68.42	1.17	67.71	1.27	51.54	9.66	0.28
	BP	68.11	1.24	68.09	1.12	67.11	1.14	84.23	7.78	0.46
Heart	HOIPGA	79.34	1.41	79.28	1.45	77.39	1.42	45.21	5.21	2.20
	BP	78.98	1.20	78.89	1.07	77.02	1.15	59.75	3.98	2.91
Horse	HOIPGA	67.69	1.17	66.76	1.31	65.44	1.03	58.01	8.34	3.15
	BP	67.06	0.97	66.80	0.93	64.99	0.80	67.56	6.11	3.69
Iris	HOIPGA	98.02	0.18	97.01	0.32	97.02	0.39	10.18	1.02	0.12
	BP	98.12	0.02	97.21	0.19	97.01	0.18	30.51	0.87	0.34
Mushroom	HOIPGA	96.10	0.22	96.08	0.14	96.02	0.17	28.22	3.71	6.39
	BP	96.15	0.05	96.07	0.09	96.00	0.08	42.14	2.65	9.45
Thyroids	HOIPGA	93.68	0.13	93.61	0.17	92.69	0.22	17.12	1.83	48.87
	BP	93.48	0.21	93.52	0.24	92.46	0.21	25.37	1.02	69.97

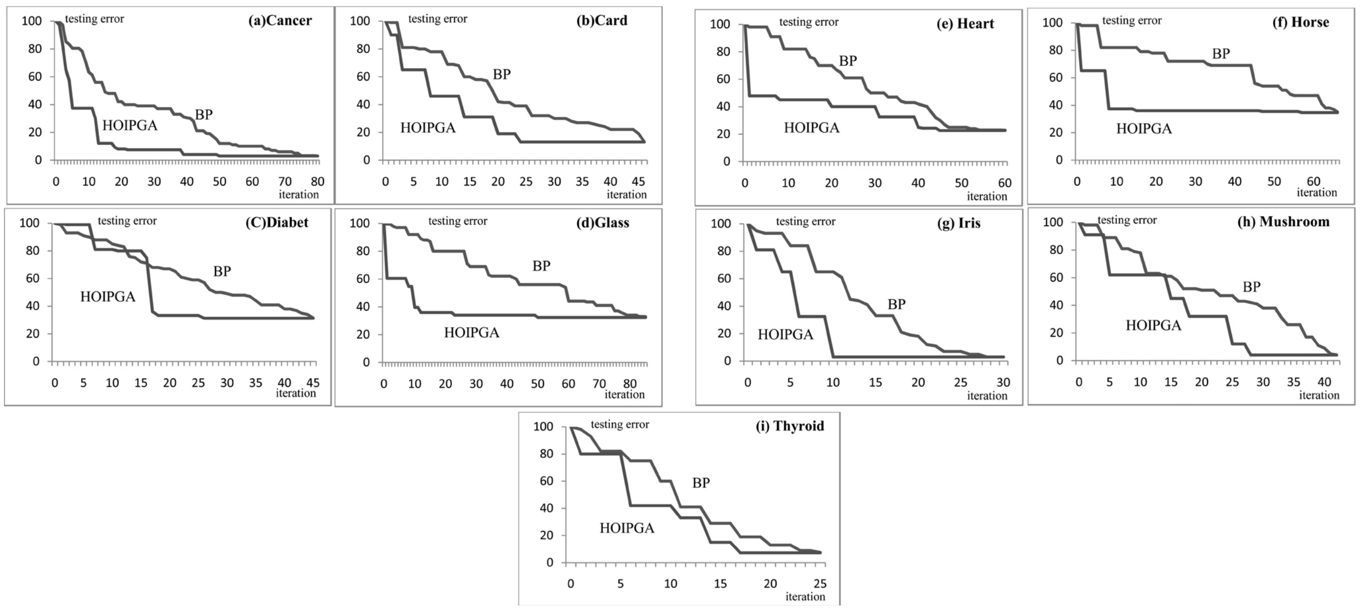

Fig. 4. The best so far network iteration- testing error graph for the Benchmarking problems

Future research will progress in two directions including improving training time and prediction accuracy of the proposed algorithm by modifying the final iteration of algorithm or the architecture of NN. The model training time or accuracy may be improved through incorporation gradient-based local search methods with the proposed algorithm especially at the final training iteration of the algorithm; also, prediction accuracy of the algorithm could be improved by using another metaheuristic to optimize NN architecture.

6 References

[1] M. Yaghini, M. M. Khoshraftar, M. Seyedabadi, "Predicting Passenger Train Delays Using Neural Network," The 4th International Seminar on Railway Operations Modelling and Analysis (*RAILROME* 2011), Feb, 2011,vol.10, no.3, Juan, 1995,16–22.

[2] S. Kiranyaz, T. Ince, A. Yildirim, M. Gabbouj, "Evolutionary artificial neural networks by multi-dimensional particle swarm optimization, "*Neural Networks*, vol. 22, no. 10, December, 2009, pp. 1448-1462.

[3] M. Castellani, H. Rowlands, "Evolutionary Artificial Neural Network Design and Training for woodveneer classification", *Engineering Applications of Artificial Intelligence*, vol. 22, 2009, pp. 732–741.

[4] E-G. Talbi, *Metaheuristic: From Design to Implementation*, University of Lille – CNRS – INRIA, a John Wiley & sons, Inc., 2009

[5] R. Battiti and G. Tecchiolli, "Training neural nets with the reactive tabu search", *IEEE Transaction on Neural Network*, vol. 6, no. 5, Sept. 1995, pp. 1185–1200.

[6] R. S. Sexton, B. Alidaee, R. E. Dorsey, and J. D. Johnson, "Global optimization for artificial neural networks: A tabu search application," *European Journal of Operational Research*, vol. 106, no. 2–3, April, 1998, pp. 570–584.

[7] N. K. Treadgold and T. D. Gedeon, "Simulated annealing and weight decay in adaptive learning: The SARPROP algorithm," *IEEE Transaction on Neural Network.*, vol. 9, no. 4, Jul. 1998, pp. 662–668.

[8] S. Chalup and F. Maire, "A study on hill climbing algorithms for neural network training," in Proc. Congress on Evolutionary Computation, vol. 3, 1999, pp. 2014–2021.

[9] X. Yao, "Evolving Artificial Neural Network", *Proceedings of the IEEE*, vol. 87, no. 9, 1999,pp. 1423–1447.

[10]V. W. Porto, D. B. Fogel, L. J. Fogel, "Alternative neural network training methods", *IEEE Expert* ,vol. 10, no. 3, Juan, 1995, pp.16–22.

[11]M. Mandischer, "A comparison of evolution strategies and backpropagation for neural network training," *Neurocomputing*, vol.42, Jan, 2002, pp. 87–117.

[12] E. Alba, J. F. Chicano, "Training Neural Networks with GA Hybrid Algorithms", K. Deb (ed.), *Proceedings of GECCO'04*, Seattle, Washington, LNCS 3102, 2004, pp. 852-863.

[13] P. Malinak, R. Jaksa, "Simultaneous Gradient and Evolutionary Neural Network Weights Adaptation Methods," *IEEE Congress on Evolutionary Computation (CEC)*, Sept, 2007, pp. 2665-2671.

[14] J. Kennedy, R. Eberhart, "Particle swarm optimization," *In Proc. IEEE international conference on neural networks*. vol. 4, Nov, 1995, pp. 1942-1948.

[15] A. P. Engelbrecht, F. V.D. Bergh, "Cooperative Learning in Neural Networks using Particle Swarm Optimizers," *South African Computer Journal*, vol. 26, 2000, pp. 84-90.

[16] Mendes R., Cortez P., Rocha M., Neves J., "Particle Swarm for Feedforward Neural Network Training," *IEEE, in Proc. International Joint Conference on Neural Networks*, 2002, pp. 1895-1899.

[17] V. G. Gudise, G. K. Venayagamoorthy, "Comparison of Particle Swarm Optimization and Backpropagation as Training Algorithms for Neural Networks, "*IEEE Swarm Intelligence Symposium*, April, 2003, pp.110-117.

[18] F. Zaho, Z. Ren, D. Yu, Y. Yang, "Application of An Improved Particle Swarm Optimization Algorithm for Neural Network Training," *International Conference on Neural Networks and Brain (ICNN&B '05)*, Oct, 2005, pp.1693-1698.

[19] M. Carvalho, T. B. Ludermir, "Particle swarm optimization of neural network architectures and weights," *In Proc. of the 7th international conference on hybrid intelligent systems*, Sept, 2007, pp. 336-339.

[20] B. Al-Kazemi , C. K. Mohan, "Training Feedforward Neural Networks using Multi-Phase Particle Swarm Optimization", *in Proc. Ninth International Conference on Neural Information Processing*, vol. 5, 2002, pp. 2615-2619.

[21] H. R. Tizhoosh, "Opposition-based learning: A new scheme for machine intelligence," *in Proc. International Conference Computational Intelligence Modeling Control and Automation,* Vienna, Austria, vol. 1, Nov, 2005, pp. 695–701.

[22] L., Han, X. He, "A novel Opposition-based Particle Swarm Optimization for Noisy Problems," *in Proc. Third International Conference on Natural Computation* (*ICNC*), *IEEE* Press, vol. 3,Aug, 2007, pp. 624 – 629.

[23] Z. Wu, Z. Ni, C. Zhang, L. Gu, "Opposition based comprehensive learning particle swarm optimization", *in Proc. 3rd International Conference on Intelligent System and Knowledge Engineering* (*ISKE*), Nov, 2008, pp. 1013-1019.

[24] M. G. H. Omran, "Using Opposition-based Learning with Particle Swarm Optimization and Barebones Differential Evolution," *Particle Swarm Optimization, InTech Education and Publishing*, 2009.

[25] C. Blum, K. Socha, "Training feed-forward neural networks with ant colony optimization: An application to pattern classification", *Fifth International Conference on Hybrid Intelligent Systems* (*HIS'*05), 2005, pp. 233-238.

[26] X. Chen, J. Wang, D. Sun, J. Liang,"A Novel Hybrid Evolutionary Algorithm Based on PSO and AFSA for Feedforward Neural Network Training", *IEEE 4th International Conference on Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08.*, Oct, 2008, pp.1-8.

[27] J. Yu, S.Wang, L. Xi, "Evolving artificial neural networks using an improved PSO and DPSO," *Neurocomputing*, vol.71, January, 2008, pp. 1054–106.

[28] A. Ratnaweera, K.Saman, H.C. Watson,"Self–organizing hierarchical particle swarm optimizer with time–varing acceleration coefficients," *IEEE Trans Evol Comput* vol.8 (3), June, 2004, pp.240–255.

[29] M. Clerc, J. Kennedy, "The particle swarm: explosion, stability, and convergence in a multi-dimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, 2002, pp. 58-73.

[30] M. Lovbjerg, T. K. Rasmussen, T. Krink, "Hybrid particle swarm optimiser with breeding and subpopulations," *In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO). San Francisco*, CA, July, 2001.

[31] N. Higashi, H. Iba, "Particle swarm optimization with Gaussian mutation," *In: Proc. of the IEEE Swarm Intelligence Symp.* Indianapolis,April, 2003, pp. 72–79.

[32] L. Prechelt, "Automatic early stopping using cross validation: Quantifying the criteria," *Neural Network*, vol. 11, no. 4, Jun. 1998, pp. 761–767