Journal of information and communication convergence engineering
http://jicce.org

J. Inf. Commun. Converg. Eng. 13(2): 123-131, Jun. 2015 Regular paper

Learning an Artificial Neural Network Using Dynamic Particle Swarm Optimization–Backpropagation: Empirical Evaluation and Comparison

Swagatika Devi, Alok Kumar Jagadev, and Srikanta Patnaik*, *Member*, *KIICE*

Department of Computer Science and Engineering, Siksha'O'Anusandhan University, Bhubaneswar-751030, India

Abstract

Training neural networks is a complex task with great importance in the field of supervised learning. In the training process, a set of input–output patterns is repeated to an artificial neural network (ANN). From those patterns weights of all the interconnections between neurons are adjusted until the specified input yields the desired output. In this paper, a new hybrid algorithm is proposed for global optimization of connection weights in an ANN. Dynamic swarms are shown to converge rapidly during the initial stages of a global search, but around the global optimum, the search process becomes very slow. In contrast, the gradient descent method can achieve faster convergence speed around the global optimum, and at the same time, the convergence accuracy can be relatively high. Therefore, the proposed hybrid algorithm combines the dynamic particle swarm optimization (DPSO) algorithm with the backpropagation (BP) algorithm, also referred to as the DPSO-BP algorithm, to train the weights of an ANN. In this paper, we intend to show the superiority (time performance and quality of solution) of the proposed hybrid algorithm (DPSO-BP) over other more standard algorithms in neural network training. The algorithms are compared using two different datasets, and the results are simulated.

Index Terms: ANN, BP algorithm, DPSO, Global optimization, Gradient descent technique

I. INTRODUCTION

Artificial neural networks (ANNs) are nonlinear mapping structures based on the function of the human brain. They are powerful tools for modeling, particularly because the underlying data relationship is unknown. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. An ANN is configured for a specific application, through a learning process. Learning in a biological system involves adjustment to the synaptic connections that exist between the neurons. ANNs can identify and learn the correlated patterns between input datasets and the corresponding target values. After training, ANNs can be used to predict the input data. ANNs imitate the learning process of the human brain and can process problems involving nonlinear and complex data even if the data are imprecise and noisy. Thus, they are ideal for the modeling of complex and often nonlinear data. ANNs have great capacity for predictive modeling; i.e., all the characters describing the unknown situation can be presented to the trained ANNs, and then, the prediction of systems is guaranteed. ANNs are capable of performing many classification, learning, and function approximation tasks, yet in practice, they sometimes deliver only marginal

Received 23 October 2014, **Revised** 20 November 2014, **Accepted** 17 December 2014
***Corresponding Author** SrikantaPatnaik (E-mail:patnaik_srikanta@yahoo.co.in. Tel: +91-9937167777)
Department of Computer Science and Engineering, Siksha 'O' Anusandhan University, Bhubaneswar-751030, India.

Open Access http://dx.doi.org/10.6109/jicce.2015.13.2.123 print ISSN: 2234-8255 online ISSN: 2234-8883

performance. Inappropriate topology selection and weight training are frequently blamed for this. Neural networks are adjusted, or trained, such that a particular input leads to a specific target output. The ANN weights are adjusted on the basis of a comparison of the output and the target, until the network output matches the target. Typically, many such input/target pairs are needed to train a network. Increasing the number of hidden layer neurons helps to improve the network performance, yet many problems can be solved with very few neurons if only the network takes its optimal configuration. Unfortunately, the inherent nonlinearity of an ANN [1] results in the existence of many suboptimal networks, and a considerable majority of training algorithms converge to these suboptimal configurations. The problem of multiple local minima in neural networks has been widely addressed. The proposed solutions include multiple starts from randomly chosen initial points, simulated annealing, random perturbation, diffusion techniques, and evolutionary computing. Most of these methods are probabilistic in nature: They can find the globally optimal solution with a certain probability, which depends on the number of iterations of the algorithm. In this study, an ANN is trained using the proposed hybrid approach. The rest of this paper is organized as follows: Section II presents a brief introduction to different existing ANN learning algorithms with their pros and cons. The proposed hybrid DPSO-BP approach is introduced in Section III. Simulation results and comparisons are provided in Section IV to demonstrate the effectiveness and potential of the proposed hybrid algorithm. Finally, several conclusions are presented in Section V.

II. NEURAL NETWORK LEARNING ALGORITHMS

Training neural networks is a complex task of great importance in the field of supervised learning. ANNs have been shown to have the potential to perform well for classification problems in many different environments, including business, science, and engineering. A majority of the studies on this topic rely on a gradient algorithm, typically a variation of backpropagation (BP), to obtain the weights of the model. Various learning techniques are introduced to optimize the weights of an ANN. Although the limitations of gradient search techniques applied to complex nonlinear optimization problems, such as the ANN, are well known, many researchers still choose to use these methods for network optimization. ANNs can identify and learn the correlated patterns between input datasets and the corresponding target values. After training, ANNs can be used to predict input data. ANNs imitate the learning process of the human brain and can process problems involving nonlinear and complex data even if the data are imprecise and noisy. Different learning algorithms are described below.

A. BP Algorithm

In an ANN, activation functions of the output units become differentiable functions of the input variables and of the weights and biases, as shown in Fig. 1. If we define an error function (E), such as a sum of squares function, which is a differentiable function of the network outputs, then this error function is itself a differentiable function of the weights. We can therefore evaluate the derivatives of the error with respect to the weights, and these derivatives can then be used to find the weight values that minimize the error function, by using any of the learning algorithms such as the BP, conjugate gradient, quasi-Newton, and Levenberg-Marquardt (LM) algorithms [2]. From the perspective of mathematical programming, supervised batch training of a neural network is a classical nonlinear optimization problem: find the minimum of the error function given some set of training data. Traditionally, this is accomplished by a suitable local descent technique, such as BP. The independent variables are the weights $\boldsymbol{w}$, and the objective function is usually the sum of squared errors (although other measures of error are also used). The objective function is formulated mathematically in Eq. (1).

$$\underset{w}{min} E(w_o, w_h) = \sum_{k=1}^{K} (f(w_o^T z_k) - y_k)^2, where z_k = f(w_h^T x_k). \tag{1}$$

Here, f denotes the transfer function; $\boldsymbol{w}_o$, the output weights; $\boldsymbol{w}_h$, the hidden layer weights; x_k, the input training data; y_k, the desired output; and z_k, the activations of hidden neurons. Despite its popularity, BP has been widely criticized for its inefficiency, and more advanced minimization techniques, such as conjugate gradient and LM methods, are available. Yet, all these techniques converge to the closest local minimum of the error function, which is very unlikely to be the global one. As a consequence, a network trained with a local algorithm may exhibit marginal performance. In this connection, the primitive BP may result in a better solution than more sophisticated methods, because its disadvantages

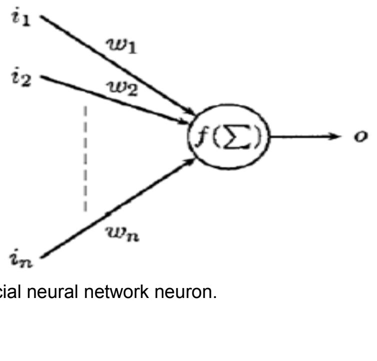

Fig. 1. Artificial neural network neuron.

```
Initialize weights
While not stop − criteria do
    For all i, j do
        $w_{ij} = w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$
    End For
End While
```

```
Initialize weights
While not stop − criteria do
    For all i, j do
        $w_{ij} = w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$
    End For
End While
```

Fig. 2. Pseudo-code of the backpropagation algorithm.

turn into the benefits of avoiding some shallow local minima. The problem of many local minima has been widely addressed in the past. It was shown that training even a simple perceptron with a nonlinear transfer function may result in multiple minima. The remedies include starting the local descent from several random points, using tabu search, simulated annealing, and genetic algorithms [3]. The new stochastic optimization algorithms significantly outperform the local methods, yet they do not provide any guarantee that their solution is the global minimum indeed. Moreover, the number of local minima of the error function grows exponentially with the number of neurons, and the likelihood that these stochastic methods will find the global minimum is not very high.

The BP algorithm is a classical domain-dependent technique for supervised training. It works by measuring the output error, calculating the gradient of this error, and adjusting the ANN weights [4] (and biases) in the descending gradient direction.

Hence, BP is a gradient descent local search procedure (expected to stagnate in the local optima in complex landscapes). The squared error of the ANN [5] for a set of patterns is calculated using Eq. (2).

$$E = \sum_{p=1}^{m} \sum_{i=1}^{n} \left(t_i^p - o_i^p\right)^2. \tag{2}$$

The actual value of the previous expression depends on the weights of the network. The basic BP algorithm calculates the gradient of E (for all the patterns) and updates the weights by moving them along the direction of the gradient descent. This can be summarized with the expression $\Delta w = -\eta \nabla E$, where the parameter $\eta > 0$ is the learning rate that controls the learning speed. The pseudo-code of the BP algorithm is shown in Fig. 2.

B. Particle Swarm Optimization Algorithm

The particle swarm optimization (PSO) algorithm was first introduced by Kennedy and Eberhart [6]. Instead of using evolutionary operators to manipulate the individuals, as in the case of other evolutionary computational algorithms, each individual in the PSO flies in the search space with a velocity that is dynamically adjusted according to its own flying experience and its companions' flying experience. Each individual is treated as a volume-less particle (a point) in the D-dimensional search space. The i-th particle is represented as $X_i = (x_{i1}, x_{i2}, ..., x_{iD})$. The best previous position (the position giving the best fitness value) of the i-th particle is recorded and represented as $P_i = (p_{i1}, p_{i2}, ..., p_{iD})$. The index of the best particle among all the particles in the population is denoted by the symbol g_b representing the global best value. The index of the best position for each particle in the population is denoted by the symbol i_b representing the individual's best value. The rate of the position change (velocity) for particle i is represented as V_i in the following equation: $(v_{i1}, v_{i2}, ..., v_{iD})$. The particles are manipulated according to the following equation.

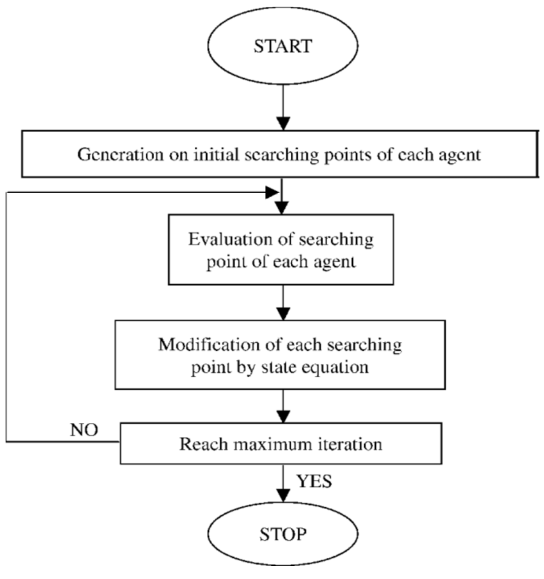
START
Generation on initial searching points of each agent
Evaluation of searching point of each agent
Modification of each searching point by state equation
NO
Reach maximum iteration
YES
STOP

Fig. 3. Particle swarm optimization flowchart.

$$v_{id} = v_{id} + c_1 * rand(\ ) * (p_{ib} - x_{id}) + c_2 * rand(\ ) * (p_{gb} - x_{id})$$
$$x_{id} = x_{id} + v_{id}\ . \tag{3}$$

The algorithm can be summarized as follows (in Fig. 3):

1. Initialize the position and velocity of all the particles randomly in the N-dimensional space.
2. Evaluate the fitness value of each particle, and update the global optimum position.
3. According to the changing gathering degree and the steady degree of the particle swarm, determine whether all the particles are re-initialized or not.
4. Determine the individual's best fitness value. Compare the p_i value of every individual with its current fitness value. If the current fitness value is better, assign the current fitness value to p_i.

5. Determine the current best fitness value in the entire population. If the current best fitness value is better than p_g, assign the current best fitness value to p_g.
6. For each particle, update the particle velocity.
7. Update the particle position.
8. Repeat Steps 2–7 until a stop criterion is satisfied or a predefined number of iterations are completed.

C. PSO-BP Algorithm

PSO-BP is an optimization algorithm combining PSO with BP [7, 8].The PSO algorithm [9] is a global algorithm that has a strong ability to find the global optimistic result. However, this algorithm has a disadvantage that the search around the global optimum is very slow. In contrast, the BP algorithm has a strong ability to find the local optimistic result, but its ability to find the global optimistic result is weak. The fundamental idea for this hybrid algorithm is that at the beginning stage of searching for the optimum, PSO [10] is employed to accelerate the training speed. When the fitness function value has not changed for some generations, or the change in value is smaller than a predefined number, the search process is switched to the gradient descent search according to this heuristic knowledge. The PSO-BP [11] algorithm's search process also starts by initializing a group of random particles. First, all the particles are updated according to Eq. (4).

$$\begin{gathered} v_{id}(t+1) = w * v_{id}(t) + c_1 * rand(\ \ ) * \\ [p_{id}(t) - x_{id}(t)] + c_2 * rand(\ \ ) * [p_{gd}(t) - x_{id}(t)], \\ x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \\ 1 \le i \le n \quad 1 \le d \le D, \end{gathered} \tag{4}$$

where c_1 and c_2 are the acceleration constants with positive values and rand() is a random number between 0 and 1. Further, w is the inertia weight [12], until a new-generation set of particles is generated; then, these new particles are used to search the global best position in the solution space. Finally, the BP algorithm is used to search around the global optimum. Thus, this hybrid algorithm may find an optimum more quickly.

The procedure for this PSO-BP algorithm can be summarized as follows:

1. Initialize the positions and velocities of a group of particles randomly in the range of [0, 1].
2. Evaluate each initialized particle's fitness value, and set P_b as the positions of the current particles and P_g as the best position of the initialized particles.
3. If the maximal iterative generations are arrived, go to Step 8; else, go to Step 4.
4. The best particle of the current particles is stored. The positions and velocities of all the particles are updated according to Eq. (4), and then, a group of new particles is generated. If a new particle flies beyond the boundary [*Xmin*, *Xmax*], the new position will be set as *Xmin* or *Xmax*; if a new velocity is beyond the boundary [*Vmin*, *Vmax*], the new velocity will be set as *Vmin* or *Vmax*.
5. Evaluate each new particle's fitness value, and replace the worst particle by the stored best particle. If the *i*-th particle's new position is better than P_{ib}, P_{ib} is set as the new position of the *i*-th particle. If the best position of all new particles is better than P_g, then P_g is updated.
6. Reduce the inertia weights w according to the selection strategy.
7. If the current P_g is unchanged for ten generations, then go to Step 8; else, go to Step 3.
8. Use the BP algorithm to search around P_g for some epochs. If the search result is better than P_g, output the current search result; else, output P_g. This is only the first type of condition; we can also use Steps 9–11 to replace the above Steps 6–8, and then, obtain the second type of condition.
9. Use the BP algorithm to search around P_g for some generations. If the search result is better than P_g, P_g is set for the current search result. Else, compare it with the worst particle of the current particles; if it is better than the best particle, use it to replace the worst particle; else, go to Step 7.
10. Reduce the inertia weights *w*.
11. Output the global optimum P_g. The parameter *w* in the PSO-BP algorithm also reduces gradually as the iterative generation increases.

This algorithm has a parameter called the learning rate [13] that controls the convergence of the algorithm to an optimal local solution; however, obtaining a good value for this parameter is difficult.

III. PROPOSED ALGORITHM (DYNAMIC PSO-BP)

PSO combined with BP gives good result in learning but due to some cons of basic PSO, we modified it with some dynamic constraints where during the learning process of the ANN objective space can be compressed or expanded. In PSO, each particle should be kept in a confined space corresponding to the parameter limitations. This decreases the diversity of the particle. If the global best particle does not change its g_{best} position after one iteration, then stagnation occurs in the population. Then, the solution may be a local optimal solution. That is, due to its stochastic behavior, it is not possible to find a way to the global optimum. The drawback of PSO is that the swarm may converge prematurely. The underlying principle behind this problem is that for the global best PSO, particles converge to a single point, which is on the boundary between the global best and the personal best positions. This point is not guaranteed for a local optimum. Another reason for this problem is the fast rate of information flow between particles, resulting in the creation of similar particles with a loss in diversity that increases the possibility of being trapped in the local optima. A further drawback is that stochastic approaches have a problem-dependent performance. This dependency usually results from the parameter settings in each algorithm. The different parameter settings for a stochastic search algorithm result in high performance variances. In general, no single parameter setting can be applied to all problems. Increasing the inertia weight (w) will increase the particle speed, resulting in more exploration (global search) and less exploitation (local search). On the other hand, reducing the inertia weight will decrease the particle speed, resulting in more exploitation and less exploration. Thus, finding the best value for the parameter is not an easy task and may differ from one problem to another. Therefore, from the above, it can be concluded that the PSO performance is problem dependent. The problem-dependent performance can be addressed through a hybrid mechanism that combines different approaches in order to benefit from the advantages of each approach.

To overcome such limitations, a multiple-swarm PSO algorithm called dynamic multiple swarms in PSO is proposed in which the number of swarms is adaptively adjusted throughout the search process via a dynamic swarm strategy. The strategy allocates an appropriate number of swarms required to support the convergence and diversity criteria among the swarms. The main objective of this is to develop a multiple-swarm PSO that eliminates the need to estimate an initial number of swarms to improve the computational efficiency without compromising the performance of the algorithm. Once the swarm template (x_{new}) is generated, the template is perturbed to generate a swarm of particles. In order for perturbation to happen, the perturbation region centered around x_{new} needs to be defined; in addition to x_{new}, a particle from every swarm is randomly selected. For example, Fig. 4 shows eight randomly selected particles from eight different swarms in addition to x_{new}. The black circle in Fig. 4 represents the coordinate of x_{new}, i.e., (x_{new}, j_a, x_{new}, j_b). Since there are more than two corners around the coordinate of x_{new} (i.e., represented by the 'x' symbol and labeled as Z1–Z6 in Fig. 4), a corner is randomly selected. In Fig. 4, the selected corner is Z2 and is denoted as v. The distance between the center and Z2, i.e., D, is computed to form the perturbation region of x_{new}. The proposed algorithm, DPSO, involves two key strategies: a swarm growing strategy to allocate more swarms if necessary and justified, and a swarm declining strategy to eliminate swarms that fail to contribute in the search of a Pareto front. Additional designs are included to support the aforementioned two strategies. These designs include the following:

1) cell-based rank density estimation scheme to keep track of the rank and density values of the particles;
2) objective space compression and expansion strategy to adjust the size of the objective space whenever needed to progressively search for a high-precision true Pareto front;
3) PSO updating equation modified to exploit its usefulness and to accommodate the multiple swarm concept; and
4) local best archive of swarms updated on the basis of the progression of its swarm representatives, the swarm leaders.

In DPSO adding and removing the swarms throughout the search process will directly affect the swarm population distribution. Instead of applying the Pareto ranking method to update the Pareto rank of the particles and applying a niching strategy to estimate the density of the particles when the swarm population progresses at every iteration.

1. If the local best archive of swarms is empty or the reinitialized parameter is triggered, record rank values of all swarm leaders, including their corresponding positions and the positions of their respective swarm members.
2. If the local best archive of swarms is nonempty, compare the rank values of the swarm leaders in the current iteration with those recorded in the local best archive of swarms. Any current swarm leader that has a relatively low rank value is identified, and its rank value, its position, and the positions of its corresponding swarm members will replace the recorded values. For this purpose, BP is used.

In the DPSO-BP algorithm, to know when the search process is transited from the particle swarm search to the gradient descent search, a heuristic way was introduced.

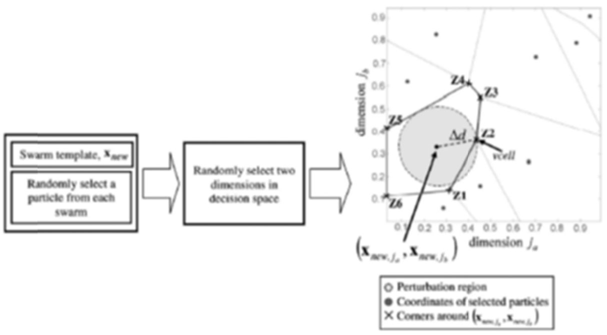
Swarm template, $\mathbf{x}_{new}$
Randomly select a particle from each swarm
Randomly select two dimensions in decision space
dimension j_b
dimension j_a
Z1
Z2
Z3
Z4
Z5
Z6
Δd
vcell
$(\mathbf{x}_{new,j_a}, \mathbf{x}_{new,j_b})$
○ Perturbation region
● Coordinates of selected particles
× Corners around $(\mathbf{x}_{new,j_a}, \mathbf{x}_{new,j_b})$

Fig. 4. Generation of swarm template x_{new}.

That is, when the best fitness value in the history of all particles does not change for some generations (i.e., ten generations), the search process is transferred to the gradient descent search. When the best fitness does not change for some generations, all the particles may lose the ability to find a better solution; at this time, a gradient descent search can be used to obtain better results. If the rank values of a current swarm and its recorded swarm leader have the same rank value, then the pure Pareto ranking method is applied to both the swarm leaders. If the current swarm dominates the recorded swarm leader, then the current one will replace the recorded one. If both do not dominate each other, one of them is randomly chosen to update the local best archive of swarms. The new velocity and position are given in Eq. (5).

$$\begin{gathered} v_{i,j}^{n}(t+1) = w \times v_{i,j}^{n}(t) + c_1 + r_1 \times \left(P_{i,j}^{n}(t) - v_{i,j}^{n}(t)\right) \\ c_2 \times r_2 \times \left(P_{g,j}(t) - x_{i,j}^{n}(t)\right) + c_3 \times (1 - r_2) \\ \times \left[r_3 \times \left(P_{L,j}^{n}(t) - x_{i,j}^{n}(t)\right) + (1 - r_3) \times (P_{L,j}^{n}(t) - x_{i,j}^{n}(t))\right] \\ x_{i,j}^{n}(t+1) = x_{i,j}^{n}(t) + V_{i,j}^{n}(t+1) \ , \end{gathered} \tag{5}$$

where $v_{i,j}^{n}(t)$ is the *j*-dimensional velocity of swarm member *i* of swarm *n* in iteration *t*; $x_{i,j}^{n}(t)$, the *j*-dimensional position of swarm member *i* of swarm *n* in iteration *t*; $P_{i,j}^{n}(t)$, the *j*-dimensional personal best (p_{best}) position of the swarm members *i* of swarm *n* in iteration *t*; $P_{g,j}^{n}(t)$, the j-dimensional global best (g_{best}) selected from the archive in iteration *t*; $P_{L,j}^{n}(t)$, the *j*-dimensional local best position of the swarm leader of swarm *n* in iteration *t*; and $P_{L,j}^{\tilde{n}}(t)$ (with a superscript $\tilde{n}$), the *j*-dimensional local best position of any swarm leaders other than their own swarm leader in iteration *t*. Further, r_1, r_2, and r_3 are random numbers within [0, 1] that are regenerated every time they occur; *w* is the inertial weight that varies between 0.1 and 0.5 at every iteration; and c_1, c_2, and c_3 are the acceleration constants. Each of the acceleration constants is randomly varied between 1.5 and 2 to place different emphasis on the components. When the objective space compression strategy [14] is applied several times in the early iterations, there is a possibility that the objective space is overly compressed and can cause the boundaries of the objective to not cover the true Pareto front.

IV. EXPERIMENTAL RESULT AND DISCUSSION

A. Standard Datasets

We choose two different datasets of small and large dimensions for the experiment. It is assumed that the proposed hybrid learning algorithm works in both these environments. The datasets are the e-learning dataset (number of patterns = 90) and the thyroid dataset (number of patterns = 7,200).

B. Results and Algorithm Settings

In the following experiments, two datasets are chosen for comparing the performances of the BP, PSO, PSO-BP, and DPSO-BP algorithms in evolving the weights of the ANN. Suppose that every weight in the network was initially set in the range of [-50, 50], and all thresholds in the network were 0 s. Further, suppose that every initial particle was a set of weights generated at random in the range of [0, 1]. Let the initial inertial weight w be 1.8, the acceleration constants, both c_1 and c_2 be 2.0, r_1 and r_2 be two random numbers in the range of [0, 1]. The maximum velocity was assumed as 10 and the minimum velocity as -10. The initial velocities of the initial particles were generated randomly in the range of [0, 1]. After each iteration, if the calculated velocity was larger or smaller than the maximum velocity or the minimum velocity, velocity would be reset to 10 or -10. The population size was a variable that was set according to the dataset. In this example, we trained an ANN with the structure of 1–S1–1. The training algorithms used were the DPSO-BP, DPSO, and BP algorithms. Assuming that S1 = 3, 4, 5, 6, 7, *x* was obtained from the range of [0, *p*] and the training set was obtained at an identical sampling interval of 0.03 from [0, *p*]; further, the test set was obtained at an identical sampling interval of 0.1 from 0.02 to p. For every fixed hidden unit number, we ran the training algorithms for each dataset. We set the maximum number of training iterations at 7,000 times for the algorithms.

Algorithms for training ANNs were compared. Tests were conducted on gradient descent algorithms such as BP, and population-based heuristics such as PSO. Experimental results showed that DPSO–BP outperformed all other algorithms in training neural networks. In this study, the DPSO–BP algorithm, which is a new, simple, and robust optimization algorithm, was used to train the standard and e-learning datasets for classification purposes. Training procedures involved the selection of the optimal values of the parameters, such as the weights between the hidden layer and the output layer, spread parameters of the hidden layer base function,

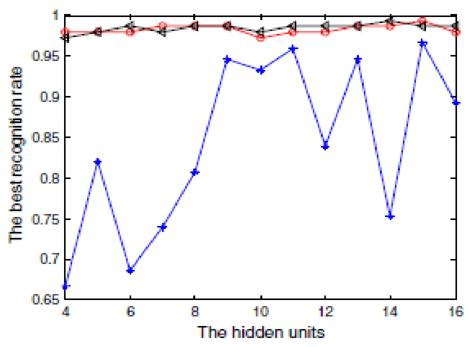

Fig. 5. Recognition rate of PSO, PSO-BP, and DPSO-BP. PSO: particle swarm optimization, BP: backpropagation, DPSO: dynamic PSO.

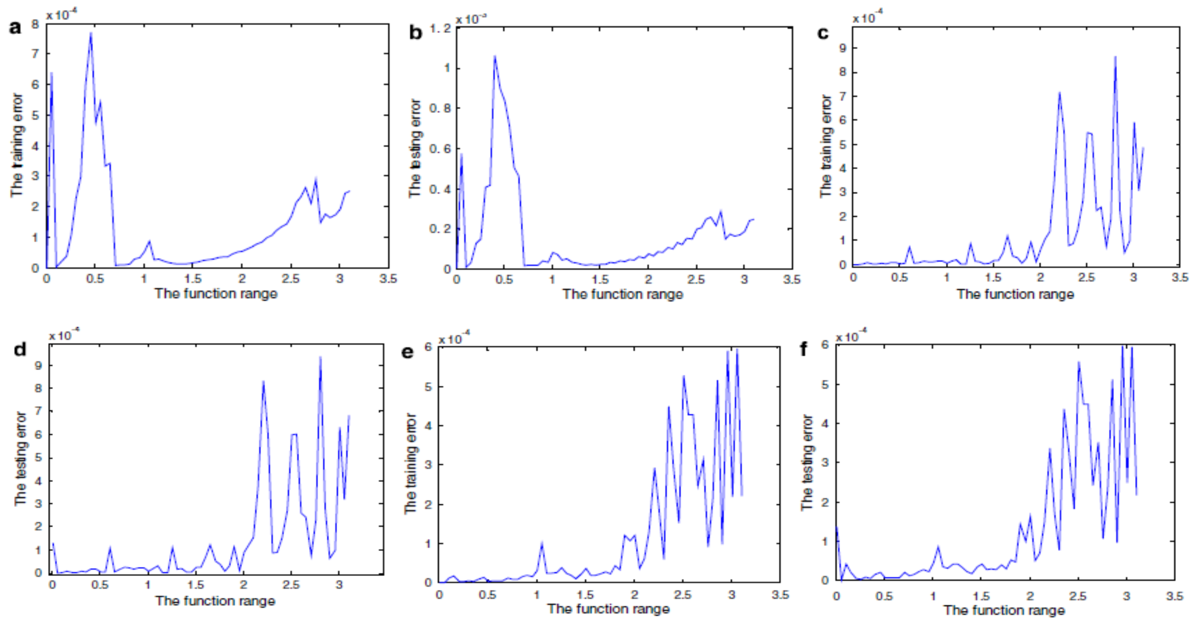

Fig. 6. Training and testing error curves for e-learning dataset using PSO, PSO-BP, and DPSO-BP. PSO: particle swarm optimization, BP: back-propagation, DPSO: dynamic PSO.

center vectors of the hidden layer, and bias parameters of the neurons of the output layer.

PSO and PSO-BP algorithms showed better performance than derivative-based methods; however, these algorithms had the disadvantage of a slow convergence rate. Trapping a local minimum was a disadvantage for these algorithms. When the learning performances were compared, experimental results showed that the performance of the proposed algorithm was better than that of the others.

The success of the classification results of the test problems was superior and correlated with the results of many research [9, 10, 12]. In real-time applications, the number of neurons might affect the time complexity of the system. The results of the e-learning classification problem were reasonable and might help training algorithms in other e-learning applications.

Fig. 5 shows the recognition rate of different algorithms while training an ANN in the thyroid dataset. Here, the red line denotes the proposed algorithm, while the blue and black lines denote the results of PSO and PSO-BP, respectively. It can be inferred from Fig. 5 that PSO-BP has a better recognition rate than PSO. Apparently, the PSO algorithm has a very low recognition rate while learning the ANN, but when it is combined with the BP algorithm, the mean recognition rate increases, as shown in Fig. 5. However, the proposed algorithm again increases the rate of recognition due to its dynamic nature. This shows that the DPSO-BP algorithm is more stable, while in the training process, the DPSO-BP algorithm uses less CPU time than the PSO-BP algorithm and the PSO algorithm.

Fig. 6 illustrates the curves of the training errors and the testing errors for the three training algorithms using the e-learning dataset. Fig. 6(a), (c), and (e) show the training error curves of the PSO, PSO-BP, and DPSO-BPA algorithms, respectively. Fig. 6(b), (d), and (f) illustrate the testing error curves of the PSO, PSO-BP, and DPSO-BPA algorithms, respectively. When the value of x is smaller, there are fewer training samples and testing samples. Further, it can be seen from Fig. 6 that the DPSO algorithm has better training and testing errors than the BP algorithm. When the value of x is larger, there are more training samples and testing samples. Moreover, it can be seen that the BP algorithm has better training and testing errors. The hybrid algorithm combines the advantages of the DPSO algorithm and the BP algorithm; therefore, it can be observed that the DPSO-BP algorithm has better training and testing errors.

V. CONCLUSIONS

In this paper, a hybrid DPSO-BP algorithm is proposed, which combines the PSO algorithm's strong ability of global learning and the BP algorithm's strong ability of local learning. Hence, we can obtain better training results by using this hybrid algorithm. Some heuristic knowledge is adapted to transit from the DPSO algorithm search to the BP algorithm search. That is, when the best fitness value in the history of all particles does not change for some generations (i.e., ten generations), the search process is transferred to the gradient descent search. The heuristic way is used to avoid wasting too much CPU time on a vain search (as used in the other compared algorithms); therefore, the training efficiency of the DPSO-BP algorithm is improved considerably. A different selection strategy is introduced for updating the inertial weight w. In the initial searching stage, the searching inertial weight is reduced rapidly in order to rapidly achieve the global optima. Then, around the global optimum, we reduce the inertial weight more smoothly by using BP so that a higher accuracy can be achieved.

From the conducted experiments, we conclude that for the same goal, the DPSO-BP algorithm uses less CPU time and provides higher training accuracy than the PSO algorithm and the BP algorithm. A comparative study shows that the performance of the variant is competitive in comparison with the selected algorithms on standard benchmark problems. It is concluded that the DPSO-BP algorithm is more stable than the BP algorithm and the PSO algorithm. In future research works, we shall focus on how to apply this hybrid PSO algorithm to solve more practical problems.

REFERENCES

[1] J. Salerno, "Using the particle swarm optimization technique to train a recurrent neural model," in *Proceedings of IEEE 9th International Conference on Tools with Artificial Intelligence*, Newport Beach, CA, pp. 45-49, 1997.

[2] O. L. Mangasarian, "Mathematical programming in neural networks," *ORSA Journal on Computing*, vol. 5, no. 4, pp. 349-360, 1993.

[3] C. M. Kuan and K. Hornik, "Convergence of learning algorithms with constant learning rates," *IEEE Transactions on Neural Networks*, vol. 2, no. 5, pp. 484-489, 1991.

[4] S. Ergezinger and E. Thomsen, "An accelerated learning algorithm for multilayer perceptrons: optimization layer by layer," *IEEE Transactions on Neural Networks*, vol. 6, no. 1, pp. 31-42, 1995.

[5] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 54-65, 1994.

[6] J. Kennedy and R. C. Eberhart, *Swarm Intelligence*. San Francisco, CA: Morgan Kaufmann, 2001.

[7] M. Gori and A. Tesi, "On the problem of local minima in backpropagation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 1, pp. 76-86, 1992.

[8] M. K. Weir, "A method for self-determination of adaptive learning rates in back propagation," *Neural Networks*, vol. 4, no. 3, pp. 371-379, 1991.

[9] F. Van den Bergh, and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 225-239, 2004.

[10] Y. Shi and R. C. Eberhart, "A modified particle swarm optimizer," in *Proceedings of IEEE World Congress on Computational Intelligence*, Anchorage, AK, pp. 69-73, 1998.

[11] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, Orlando, FL, pp. 4104-4108, 1997.

[12] A. Abraham and B. Nath, "ALEC: an adaptive learning framework for optimizing artificial neural networks," in *Computational Science-ICCS 2001*. Heidelberg: Springer, pp. 171-180, 2001.

[13] H. V. Gupta, K. L. Hsu, and S. Sorooshian, "Superior training of artificial neural networks using weight-space partitioning," in *Proceedings of International Conference on Neural Networks*, Houston, TX, pp. 1919-1923, 1997.

[14] K. S. Tang, C. Y. Chan, K. F. Man, and S. Kwong, "Genetic structure for NN topology and weights optimization," in *Proceedings of the 1st International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA)*, Sheffield, UK, pp. 250-255, 1995.

Swagatika Devi
received her B.Tech and M.Tech degrees in Computer Science and Engineering. She is working as an assistance professor in the department of Computer Science and Engineering in SOA University, Bhubaneswar, Orissa. She got gold medal during her M.Tech career. She contributed many international journals and presented research papers in different international conferences and also has tremendous contribution towards research book chapters. Her research interests include soft computing, data mining and bio-informatics.

Alok Kumar Jagadev
is working as an Associate Professor in the Department of Computer Science & Engineering. He has obtained his Ph.D. degree for his work in the field of Wireless Ad-hoc Networks from Siksha 'O' Anusandhan University in 2011. He has contributed more than 30 papers in various journals and conferences of international repute. He has contributed three book chapters. He has authored/co-authored four text books in the field of computer science. He has also edited two books for different international publications like IGI global, Springer. He has involved in organizing many international conferences. His research interest includes soft computing, data mining, bio-informatics, etc.

Srikanta Patnaik
is a Professor in the Department of Computer Science and Engineering, SOA University, Bhubaneswar, India. He has received his Ph.D. (Engineering) on Machine Intelligence from Jadavpur University in 1999 and supervised 12 Ph.D. theses and more than 30 M. Tech theses in the area of machine intelligence, soft computing applications and re-engineering. He has published around 100 technical papers in international journals and conference proceedings. He is author of 2 textbooks and edited 12 books and few invited book chapters, published by leading international publisher like Springer-Verlag, Kluwer Academic, etc. He was the Principal Investigator of AICTE sponsored TAPTEC project “Building Cognition for Intelligent Robot” and UGC sponsored Major Research Project “Machine Learning and Perception using Cognition Methods”. He is the Editors-in-Chief of International Journal of Information and Communication Technology and International Journal of Computational Vision and Robotics published from Inderscience Publishing House, England and also Editors-in-chief of Book Series on “Modeling and Optimization in Science and Technology” published from Springer, Germany.