Available online at www.sciencedirect.com
ScienceDirect
Applied Mathematics and Computation 185 (2007) 1026–1037
APPLIED MATHEMATICS AND COMPUTATION
www.elsevier.com/locate/amc

A hybrid particle swarm optimization–back-propagation algorithm for feedforward neural network training

Jing-Ru Zhang [a,b,*], Jun Zhang [a], Tat-Ming Lok [c], Michael R. Lyu [d]

[a] *Intelligent Computing Lab, Hefei Institute of Intelligent Machines, Chinese Academy of Sciences, P.O. Box 1130, Hefei, Anhui 230031, China*
[b] *Department of Automation, University of Science and Technology of China, Hefei 230027, China*
[c] *Information Engineering Department, The Chinese University of Hong Kong, Shatin, Hong kong*
[d] *Computer Science and Engineering Department, The Chinese University of Hong Kong, Shatin, Hong kong*

Abstract

The particle swarm optimization algorithm was showed to converge rapidly during the initial stages of a global search, but around global optimum, the search process will become very slow. On the contrary, the gradient descending method can achieve faster convergent speed around global optimum, and at the same time, the convergent accuracy can be higher. So in this paper, a hybrid algorithm combining particle swarm optimization (PSO) algorithm with back-propagation (BP) algorithm, also referred to as PSO–BP algorithm, is proposed to train the weights of feedforward neural network (FNN), the hybrid algorithm can make use of not only strong global searching ability of the PSOA, but also strong local searching ability of the BP algorithm. In this paper, a novel selection strategy of the inertial weight is introduced to the PSO algorithm. In the proposed PSO–BP algorithm, we adopt a heuristic way to give a transition from particle swarm search to gradient descending search. In this paper, we also give three kind of encoding strategy of particles, and give the different problem area in which every encoding strategy is used. The experimental results show that the proposed hybrid PSO–BP algorithm is better than the Adaptive Particle swarm optimization algorithm (APSOA) and BP algorithm in convergent speed and convergent accuracy.

Keywords: Particle swarm optimization algorithm; Feedforward neural network; BP algorithm; Adaptive particle swarm optimization

1. Introduction

In recent years, feedforward neural networks (FNN), in particular, two layered FNNs [12] have been widely used to classify nonlinearly separable patterns [31,26,22] and approximate arbitrary continuous functions

* Corresponding author. Address: Intelligent Computing Lab, Hefei Institute of Intelligent Machines, Chinese Academy of Sciences, P.O. Box 1130, Hefei, Anhui 230031, China.
E-mail addresses: jrzhang@iim.ac.cn (J.-R. Zhang), zhangjun@iim.ac.cn (J. Zhang), tmlok@ie.cuhk.edu.hk (T.-M. Lok), lyu@cse.cuhk.edu.hk (M.R. Lyu).

0096-3003/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2006.07.025

[20,25]. Currently, there have been many algorithms used to train the FNN, such as back-propagation algorithm (BPA), genetic algorithm (GA) [5,6], simulating annealing algorithm (SAA) [13,14], particle swarm optimization algorithm (PSO) [16,18], and so on.

The particle swarm optimization (PSO) is an evolutionary computation technique developed by Eberhart and Kennedy in 1995 [1,2], inspired by social behavior of bird flocking. And we can also say it to be a kind of algorithm based on social psychology. Similar to the genetic algorithm (GA), the PSO algorithm is an optimization tool based on population, and the system is initialized with a population of random solutions and can search for optima by the updating of generations. In 1998, Shi and Eberhart firstly introduced the inertia weights w into the previous PSO algorithm [3,17]. Through adjusting w, the performances of the PSO algorithm can be improved significantly. Researchers often use a kind of Adaptive particle swarm optimization (APSO) algorithm. The adaptive particle swarm optimization can be described as following, in different searching stages, the inertial weight w is changed adaptively.

Unlike the GA, the PSO algorithm has no complicated evolutionary operators such as crossover and mutation [21]. In the PSO algorithm, the potential solutions, called as particles, are obtained by ''flowing'' through the problem space by following the current optimum particles. Generally speaking, the PSO algorithm has a strong ability to find the most optimistic result, but it has a disadvantage of easily getting into a local optimum .After suitably modulating the parameters for the PSO algorithm, the rate of convergence can be speeded up and the ability to find the global optimistic result can be enhanced. The PSO algorithm's search is based on the orientation by tracing P_b that is each particle's best position in its history, and tracing P_g that is all particles' best position in their history, it can rapidly arrive around the global optimum. However, because the PSO algorithm has several parameters to be adjusted by empirical approach, if these parameters are not appropriately set, the search will become very slow near the global optimum.

Regarding the FNNs training, the mostly used training algorithm is the back-propagation (BP) algorithm, which is a gradient-based method. Hence some inherent problems existing in BP algorithm are also frequently encountered in the use of this algorithm. Firstly, the BP algorithm will easily get trapped in local minima especially for those non-linearly separable pattern classification problems or complex function approximation problem [7], so that back-propagation may lead to failure in finding a global optimal solution. Second, the convergent speed of the BP algorithm is too slow even if the learning goal, a given termination error, can be achieved. The important problem to be stressed is that the convergent behavior of the BP algorithm depends very much on the choices of initial values of the network connection weights as well as the parameters in the algorithm such as the learning rate and the momentum. To improve the performance of the original BP algorithm, researchers have concentrated on the following two factors: (1) selection of better energy function [8,9]; (2) selection of dynamic learning rate and momentum [10,11]. However, these improvements haven't removed the disadvantages of the BP algorithm getting trapped into local optima in essence. In particular, with FNN's structure becoming more complex, its convergent speed will be even slower. But if the search for the BP algorithm starts from near the optimum, and if the learning rate is adjusted small enough, how will the searching results be? The experiments in the sequel will give further analyses.

Genetic algorithm (GA) has been also used in training FNNs recently, but in training process, this algorithm needs encoding operator and decoding operator. Usually there are three kinds of complicated evolutionary operators with this algorithm, i.e., selection, crossover and mutation, it was found in experiments that when the structure of FNNs is simple, its training results may be better than the ones using the BP algorithm to train, when the structure of FNNs becomes complex and there are large training samples, the GA's convergent speed will become very slow, so that the convergent accuracy may be influenced by the slow convergent speed.

In this paper, we combined the adaptive particle swarm optimization (APSO) algorithm with the BP algorithm to form a hybrid learning algorithm for training FNNs. This hybrid uses the APSO algorithm to do global search in the beginning of stage, and then uses the BP algorithm to do local search around the global optimum P_g. In particular, this hybrid algorithm will be used to train the FNN weights for function approximation and classification problems, respectively compared with the APSO algorithm and the BP algorithm in convergent speed and generalization performance.

This paper is organized as follows. Section 2 presents a briefly introduction to adaptive particle swarm optimization, and the selection strategy for inertia weights w. The proposed new hybrid PSO–BP algorithm is introduced in Section 3, where three kinds of encoding strategies will be presented for the PSO algorithm. Simulation results are provided in Section 4 to demonstrate the effectiveness and potential of the new proposed hybrid algorithm. Finally, several conclusions are included in Section 5.

2. Adaptive particle swarm optimization

Particle swarm optimization (PSO) is a kind of algorithm to search for the best solution by simulating the movement and flocking of birds. The algorithm works by initializing a flock of birds randomly over the searching space, where every bird is called as a ''particle''. These ''particles'' fly with a certain velocity and find the global best position after some iteration. At each iteration, each particle can adjust its velocity vector, based on its momentum and the influence of its best position (P_b) as well as the best position of its neighbors (P_g), and then compute a new position that the ''particle'' is to fly to. Supposing the dimension for a searching space is D, the total number of particles is n, the position of the ith particle can be expressed as vector $X_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$; the best position of the ith particle being searching until now is denoted as $P_{ib} = (p_{i1}, p_{i2}, \ldots, p_{iD})$, and the best position of the total particle swarm being searching until now is denoted as vector $P_g = (p_{g1}, p_{g2}, \ldots, p_{gD})$;the velocity of the ith particle is represented as vector $V_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$. Then the original PSOA [1,2] is described as:

$$v_{id}(t+1) = v_{id}(t) + c_1 * \text{ rand}() * [p_{id}(t) - x_{id}(t)] + c_2 * \text{ rand}() * [p_{gd}(t) - x_{id}(t)], \tag{1a}$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \quad 1 \leqslant i \leqslant n \quad 1 \leqslant d \leqslant D, \tag{1b}$$

where c_1, c_2 are the acceleration constants with positive values; rand() is a random number between 0 and 1; w is the inertia weight. In addition to the parameters c_1, and c_2 parameters, the implementation of the original algorithm also requires placing a limit on the velocity (v_{max}). After adjusting the parameters w and v_{max}, the PSO can achieve the best search ability.

The adaptive particle swarm optimization (APSO) algorithm is based on the original PSO algorithm, firstly proposed by Shi and Eberhart in 1998 [3]. The APSO can be described as follows:

$$v_{id}(t+1) = w * v_{id}(t) + c_1 * \text{ rand}() * [p_{id}(t) - x_{id}(t)] + c_2 * \text{ rand}() * [p_{gd}(t) - x_{id}(t)], \tag{2a}$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \quad 1 \leqslant i \leqslant n \quad 1 \leqslant d \leqslant D, \tag{2b}$$

where w is a new inertial weight. This algorithm by adjusting the parameter w can make w reduce gradually as the generation increases. In the searching process of the PSO algorithm, the searching space will reduce gradually as the generation increases. So the APSO algorithm is more effective, because the searching space reduces step by step nonlinearly, so the searching step length for the parameter w here also reduces correspondingly. Similar to GA, after each generation, the best particle of particles in last generation will replace the worst particle of particles in current generation, thus better result can be achieved.

In the literature [3,4,19], several selection strategies of inertial weight w have been given. Generally, in the beginning stages of algorithm, the inertial weight w should be reduced rapidly, when around optimum, the inertial weight w should be reduced slowly. So in this paper, we adopted the following selection strategy:

$$w = \begin{cases} w_0 - (w_1/\max gen1) * generation, & 1 \leqslant generation \leqslant \max gen1 \\ (w_0 - w_1) * e^{(\max gen1 - generation)/k}, & \max gen1 \leqslant generation \leqslant \max gen2, \end{cases} \tag{3}$$

where w_0 is the initial inertial weight, w_1 is the inertial weight of linear section ending, max$gen2$ is the total searching generations, max$gen1$ is the used generations that inertial weight is reduced linearly, *generation*, is a variable whose range is [1, max$gen2$]. Through adjusting k, we can achieve different ending values of inertial weight. Fig. 1 illustrates the reduction scheme for the inertial weight. In particular, the value of max$gen2$ is selected according to empirical knowledge [3].

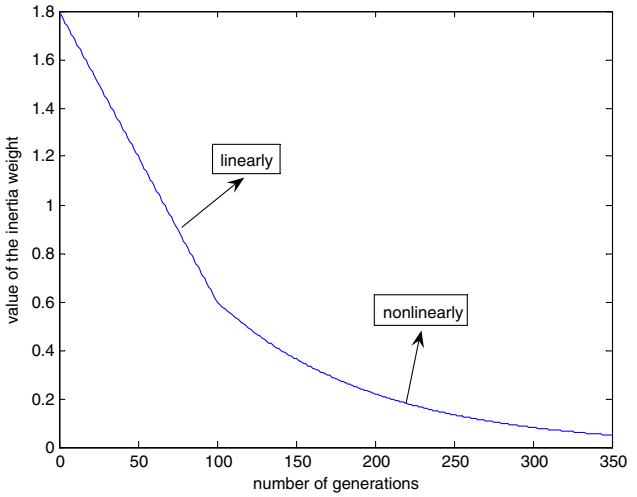

Fig. 1. The reduction scheme for value of the inertia weight w.

3. Hybrid PSO–BP algorithm and feedforward neural network training

3.1. Hybrid PSO–BP algorithm

The PSO–BP is an optimization algorithm combining the PSO with the BP. Similar to the GA, the PSO algorithm is a global algorithm, which has a strong ability to find global optimistic result, this PSO algorithm, however, has a disadvantage that the search around global optimum is very slow. The BP algorithm, on the contrary, has a strong ability to find local optimistic result, but its ability to find the global optimistic result is weak. By combining the PSO with the BP, a new algorithm referred to as PSO–BP hybrid algorithm is formulated in this paper. The fundamental idea for this hybrid algorithm is that at the beginning stage of searching for the optimum, the PSO is employed to accelerate the training speed. When the fitness function value has not changed for some generations, or value changed is smaller than a predefined number, the searching process is switched to gradient descending searching according to this heuristic knowledge.

Similar to the APSO algorithm, the PSO–BP algorithm's searching process is also started from initializing a group of random particles. First, all the particles are updated according to the Eqs. (2a) and (2b), until a new generation set of particles are generated, and then those 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. In this way, this hybrid algorithm may find an optimum more quickly.

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

Step 1: Initialize the positions and velocities of a group of particles randomly in the range of $[0, 1]$.
Step 2: Evaluate each initialized particle's fitness value, and P_b is set as the positions of the current particles, while P_g is set as the best position of the initialized particles.
Step 3: If the maximal iterative generations are arrived, go to *Step 8*, else, go to *Step 4*.
Step 4: The best particle of the current particles is stored. The positions and velocities of all the particles are updated according to Eqs. (1) and (2), then a group of new particles are generated, If a new particle flies beyond the boundary $[X_{min}, X_{max}]$, the new position will be set as X_{min} or X_{max}; if a new velocity is beyond the boundary $[V_{min}, V_{max}]$, the new velocity will be set as V_{min} or V_{max}.
Step 5: Evaluate each new particle's fitness value, and the worst particle is replaced by the stored best particle. If the ith particle's new position is better than P_{ib}, P_{ib} is set as the new position of the ith particle. If the best position of all new particles is better than P_g, then P_g is updated.
Step 6: Reduce the inertia weights w according to the selection strategy described in Section 3.
Step 7: If the current P_g is unchanged for ten generations, then go to *Step 8*; else, go to *Step 3*.

Step 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; or else, output P_g. This is only the first kind of condition, we can also use the following steps to replace the above *Steps 6–8*, then get the second kind of condition.

Step 6: Use the BP algorithm to search around P_g for some generations, if search result is better than P_g, P_g is set for the current search result; or else, comparing it with the worst particle of current particles, if it is better than the best particle, using it to replace the worst particle, or else, go to *Step 7*.

Step 7: Reducing the inertia weights w according to the strategy in Section 3.

Step 8: Output the global optimum P_g.

The parameter w, in the above PSO–BP algorithm also reduces gradually as the iterative generation increases, just like the APSO algorithm. The selection strategy for the inertial weight w is the same as the one described in Section 2, i.e., firstly reduce w linearly then reduce it nonlinearly. But the parameter max*gen*1 generally is adjusted to an appropriate value by many repeated experiments, then an adaptive gradient descending method is used to search around the global optimum P_g.

The BP algorithm based on gradient descending was firstly proposed by Werbos [32] in his Ph.D. thesis in 1974, in 1986, Rumelhart et al. further formulated the standard back-propagation algorithm (BPA) for multi-layered perceptrons [28]. This algorithm has a parameter called learning rate that controls the convergence of the algorithm to an optimal local solution. Rumelhart et al. did not show how to get a good value for this parameter. In practical applications, users usually employed theoretical, empirical or heuristic methods to set a good value for this learning rate. Kuan and Hornic [29] investigated the convergence for a constant learning rate condition. Baldi [30] gave an overview about main learning algorithm based on gradient method, but the convergent rate was not studied. In the literatures [10,11,15,27], several adaptive back-propagation algorithms were proposed, respectively. In this paper, we adopted the following strategy for learning rate:

$$\eta = k * e^{-\eta_0 * \text{epoch}}, \tag{4}$$

where η is learning rate, k, η_0 are constants, epoch is a variable that represents iterative times, through adjusting k and η_0, we can control the reducing speed of learning rate.

3.2. The PSO–BP algorithm for feedforward neural network training

3.2.1. The two-layered feedforward neural network

We will use three kinds of algorithm to evolve the weights of the feedforward neural network with two layered structures. Supposed that the input layer has n nodes; the hidden layer has H hidden nodes; output layer has O output nodes. Fig. 2 shows the structure of a two layered feedforward neural network. According to the figure, a corresponding fitness function was given.

Assuming that the hidden transfer function is sigmoid function, and the output transfer function is a linear activation function. From Fig. 2, it can be seen that the output of the jth hidden node is:

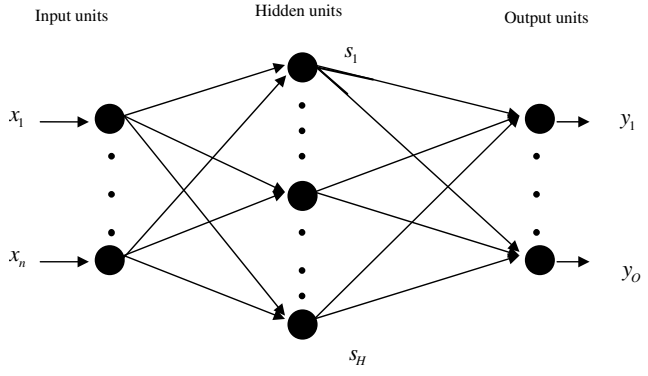

Fig. 2. A two-layered feedforward neural network structure.

$$f(s_j) = 1 \Big/ \left(1 + \exp\left(-\left(\sum_{i=1}^{n} w_{ij} \cdot x_i - \theta_j\right)\right)\right), \quad j = 1, 2, \ldots, H, \tag{5}$$

where n is the number of the input node, w_{ij} is the connection weight from the ith node of input layer to the jth node of hidden layer, θ_i is the threshold of the jth hidden layer unit; x_i is the ith input. s_j is the weight input sum in hidden, and $s_j = \sum_{i=1}^{n} w_{ij} \cdot x_i - \theta_j$. The output of the kth output layer is:

$$y_k = \sum_{j=1}^{H} w_{kj} \cdot f(s_j) - \theta_k \quad k = 1, 2, \ldots, O, \tag{6}$$

where w_{kj} is the connection weight from the jth hidden node to the kth output node, θ_k is the threshold of the kth output unit.

The learning error E can be calculated by the following formulation:

$$E = \sum_{k=1}^{q} E_k/(q * O) \quad \text{where} \quad E_k = \sum_{i=1}^{O} (y_i^k - C_i^k)^2, \tag{7}$$

where q is the number of total training samples, $y_i^k - C_i^k$ is the error of the actual output and desired output of the ith output unit when the kth training sample is used for training. We defined the fitness function of the ith training sample as follows:

$$\text{fitness}(X_i) = E(X_i) \tag{8}$$

When the PSO algorithm is used in evolving weights of feedforward neural network, every particle represent a set of weights, there are three encoding strategy for every particle, the following section give detailed description.

3.2.2. Encoding strategy

3.2.2.1. Vector encoding strategy. In this encoding strategy, every particle is encoded for a vector. For feedforward neural network (FNN) involved, each particle represents all weights of a FNN's structure. For example, for the FNN with the structure of 2–5–1, the corresponding encoding style for each particle can be represented as:

$$\text{particle}(i) = [w_{31} w_{32} w_{41} w_{42} w_{51} w_{52} w_{61} w_{62} w_{71} w_{72} w_{83} w_{84} w_{85} w_{86} w_{87}], \tag{9a}$$

$$\text{particles matrix} = [\text{particle}(1); \text{particle}(2); \ldots; \text{particle}(M)], \tag{9b}$$

where M is the number of the total particles, $i = 1, \ldots, M$.

However, when calculating the output of the FNN, we need to decode each particle into weights matrix, thus the decoding process becomes a little complicated. Nevertheless, this kind of encoding strategy is often used in function optimization field.

3.2.2.2. Matrix encoding strategy. In this encoding strategy, every particle is encoded for a matrix. We also take the FNN with the structure of 3–4–2 for an example, the encoding strategy can be written as:

$$\text{particles}(:, :, i) = [W_1, W_2'] \tag{10a}$$

$$W_1 = \begin{bmatrix} w_{41} & w_{42} & w_{43} \\ w_{51} & w_{52} & w_{53} \\ w_{61} & w_{62} & w_{63} \\ w_{71} & w_{72} & w_{73} \end{bmatrix}, \quad W_2' = \begin{bmatrix} w_{84} & w_{94} \\ w_{85} & w_{95} \\ w_{86} & w_{96} \\ w_{87} & w_{97} \end{bmatrix}, \tag{10b}$$

where W_1 is the hidden layer weight matrix, while W_2 is the output layer weight matrix, and W_2' is the transpose of W_2.

In practical application, we can use a three dimensional matrix $\text{particles}_{5\times3\times M}$ to store all particle matrices, where M is the number of all particles, $i = 1, \ldots, M$.

With this encoding strategy for FNN, decoding is easy to execute. The point that should be stress is that in the training process of the FNN, this kind of encoding strategy is often adopted.

3.2.2.3. Binary encoding strategy. In 1997, Kenney and Eberhart proposed the binary particle swarm method [23,24], in which, similar to chromosome in genetic algorithm (GA), every particle is encoded for string bits including the values of zero and one value. In training the FNN, every particle represents a series of weights. When the structure become more complex, the length of every binary encoding particle is much longer, so that the encoding and decoding process becomes very complicated. So this kind of encoding strategy is not often adopted in training the FNN.

4. Experimental result and discussion

In the following experiments, by using three examples we compared the performances of BP algorithm, APSO algorithm and PSO–BP algorithm in evolving the weights of the FNN. Supposed that every weight in the network was initially set in the range of $[-50, 50]$, and all thresholds in the network were 0 s. Supposed 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 $c1$ and $c2$ be 2.0, r_1 and r_2 be two random numbers in the range of $[0, 1]$, respectively. The maximum velocity 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, it would be reset to 10 or -10. The population size is 200.

4.1. Example 1. Three bits parity problem

In this problem, there are three input units and one output result, so we adopt the FNN with the structure of 3–$S1$–1, where $S1$ is the number of hidden nodes, in the following experiment, we shall compare the performance of $S1 = 4, 5, 6, 7, 8, 9, 10, 11, 13, 15, 20, 30$, respectively. For the BP algorithm, we adopted adaptive learning rate as described in Section 3, the maximal iteration number is assumed as 10 000. For the APSO algorithm, the maximal generation is assumed as 500. For the PSO–BP algorithm, assumed that we used the algorithm under the first kind condition. At first, the maximal generation of particles' search is assumed as 100, then the gradient method is used to search for 2000 iteration.

About three bits parity problem, the feedforward neural network is designed to recognize the number of "1" in the input vector composing of three bits parity problem. When the input vector has odd number of "1", the output is "1"; when the input vector has even number of "1", the output is "0".

We compared the two performances indices of mean square error (MSE) and CPU Time used in training. When the MSE is less than 0.001, or when the maximal iterative times is arrived, the current algorithm training ends. Running every algorithm successively for five times under the same hidden unit number, then taking the average result among five results to take part in comparisons. The results are shown in Table 1.

From Table 1, we can see that the PSO–BP algorithm is apparently better than the PSO algorithm and the BP algorithm. When achieving the same MSE, the PSO–BP algorithm spends less CPU than the PSO algorithm and the BP algorithm. In this example, the performances of the PSO algorithm and the BP algorithm are very close. For these three algorithms, with the hidden unit increasing, the calculating time firstly decreases then increases.

In experiment, we keep on an eye on the phenomenon that when the best P_g of all the particles in searching history has not changed for ten generations, the PSO–BP algorithm will transfer to use.

In the PSO algorithm, we trace the global optimum P_g, so we see its MSE is stepwise, its real MSE is just like presentation in Table 1.

When the best result P_g of all the particles in searching history has not changed for ten generations, the PSO–BP algorithm transfers to use the gradient descending method to search around the P_g, by a heuristic way to transit from the particle swarm search to gradient descending search, the searching efficiency of the algorithm was improved greatly. The experimental results also showed this point.

Table 1
The comparison of the performances of the PSO–BPA, the PSOA and the BPA for three bits parity problem

Hidden number	The PSO–BPA		The PSOA		The BPA	
$S1$	MSE	Time (s)	MSE	Time (s)	MSE	Time (s)
4	9.9894e−004	3.084000	9.3174e−004	9.003000	9.9956e−004	9.946000
5	9.9924e−004	3.205000	8.7849e−004	8.392000	9.9914e−004	8.282000
6	9.8920e−004	3.395000	9.7871e−004	8.672000	9.9590e−004	9.674000
7	9.9547e−004	2.703000	9.7578e−004	8.573000	9.9702e−004	9.794000
8	9.8970e−004	3.785000	8.6146e−004	8.242000	9.9998e−004	8.622000
9	9.9609e−004	3.064000	9.4730e−004	8.952000	9.9946e−004	7.882000
10	9.9905e−004	3.755000	7.5435e−004	8.483000	9.9678e−004	7.491000
11	9.9850e−004	3.926000	9.5960e−004	7.391000	9.9779e−004	7.852000
13	9.8661e−004	3.175000	9.5993e−004	8.933000	9.9799e−004	8.533000
15	9.9262e−004	3.515000	8.2313e−004	9.624000	9.9801e−004	8.923000
20	9.8239e−004	4.246000	9.8359e−004	11.20600	9.9877e−004	9.784000
30	9.7767e−004	6.620000	9.2008e−004	12.32890	9.9273e−004	11.18700

4.2. Example 2. Function approximate problem

In this example, we trained an FNN with the structure of 1–$S1$–1 to approximate the function $f = \sin(2x)e^{-x}$. The used training algorithms are still the PSOBP algorithm, the PSO algorithm and the BP algorithm. Assuming that $S1 = 3, 4, 5, 6, 7$, x is obtained from the range of $[0, \pi]$, and the training set was obtained at an identical sampling interval of 0.03 from $[0, \pi]$; while the test set was obtained at an identical sampling interval of 0.1 from 0.02 to π.

For every fixed hidden unit number, we ran the three training algorithms for five times respectively. We set the maximal training iteration for 7000 times for the ABP algorithm, set the maximal training generation for 500 for the APSO algorithm, in the PSO–BP algorithm, set maximal generation for 200 for global search with the PSO algorithm, and set maximal iteration for 1500 times for the gradient descending method based on BP algorithm. Table 2 gives the performance comparisons for the three algorithms. From Table 2, we can see that the hybrid algorithm consumed less CPU time, but achieved higher accuracy than the ABP algorithm and APSO algorithm.

Table 2
The comparison of the performances of the PSO–BPA, the PSOA and the BPA for function approximate problem of $f = \sin(2x)e^{-x}$

Hidden unit number	The BPA training error			Testing error	CPU time
$S1$	The best	The worst	The average	Average error	The average
3	6.8432e−004	7.4183e−004	7.0747e−004	6.8969e−004	27.656000s
4	3.3664e−004	8.7374e−004	6.199e−004	6.0843e−004	33.348000s
5	2.4683e−004	7.3552e−004	4.4248e−004	4.3799e−004	38.655000s
6	2.721e−004	3.4994e−004	3.0151e−004	2.9642e−004	44.064000s
7	2.106e−004	6.9003e−004	3.7318e−004	3.5172e−004	49.501000s
	The PSOA training error				
3	2.1861e−004	5.7089e−004	4.0011e−004	4.1945e−004	18.777000s
4	2.6181e−004	5.5377e−004	4.0266e−004	4.2373e−004	19.749000s
5	1.5801e−004	5.1031e−004	3.76905e−004	3.8876e−004	20.689000s
6	8.642e−005	5.7121e−004	2.7945e−004	2.7607e−004	22.782000s
7	8.5208e−005	4.9845e−004	2.7404e−004	2.7416e−004	25.186000s
	The BP–PSOA training error				
3	3.2781e−004	7.266e−004	4.8984e−004	4.8327e−004	13.526000
4	5.6104e−005	5.4891e−004	2.7888e−004	2.9297e−004	15.069000
5	7.6982e−005	6.3747e−004	2.9032e−004	2.8262e−004	17.172000
6	6.476e−005	4.2293e−004	2.0032e−004	2.0743e−004	18.925000
7	4.2074e−005	2.7363e−004	1.4333e−004	1.4788e−004	20.167000

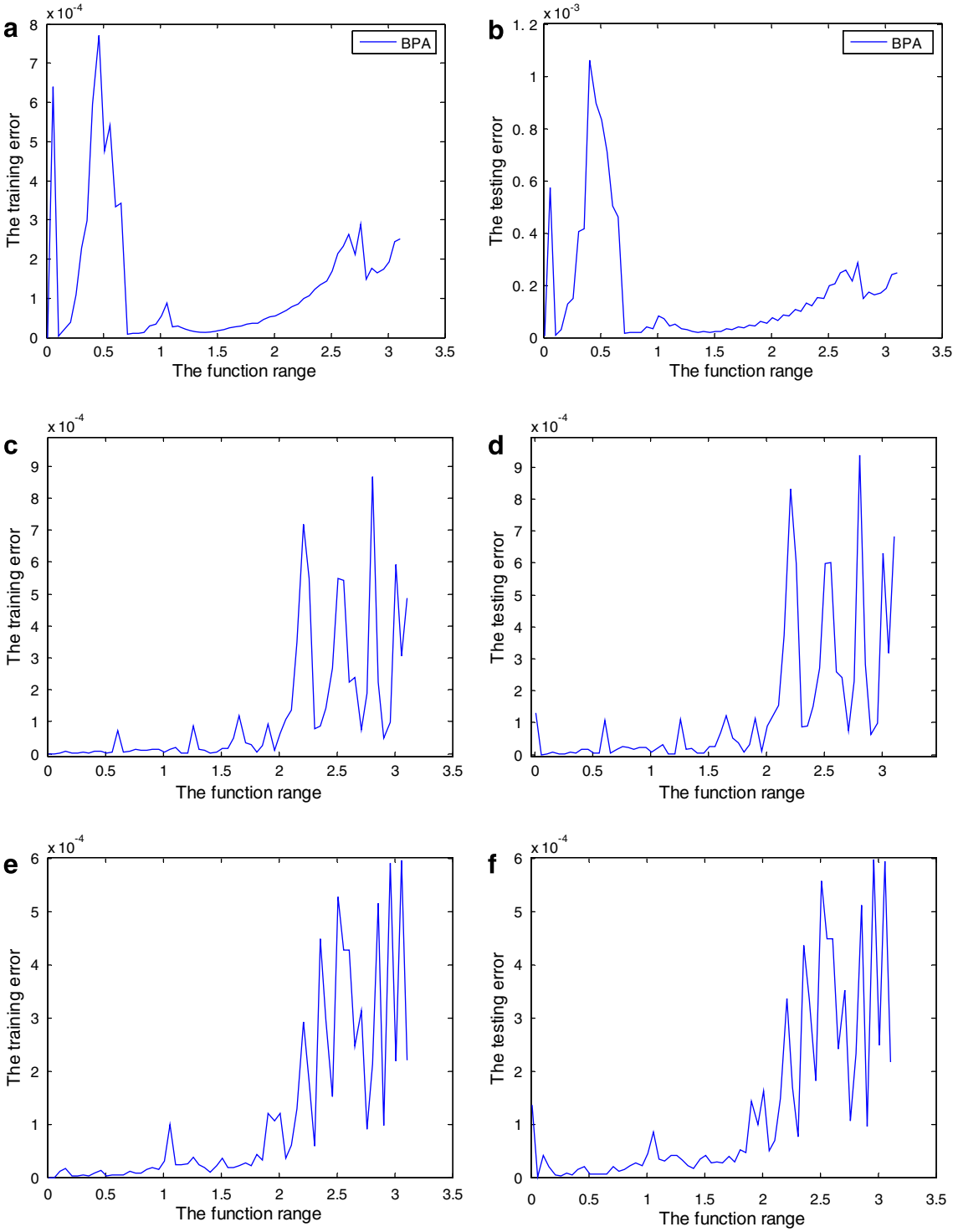

Fig. 3. The training error curves of the training samples and the testing error curves of the testing samples for the function $f = \sin(2x)e^{-x}$ for the three training algorithms. (a), (c), (e) are the training error curves of the ABPA, the APSOA and the PSO–BPA, respectively. (b), (d), (f) are the testing error curves of the ABPA, the APSOA and the PSO–BPA, respectively.

In the light of the above Table 2, we selected the hidden unit number that is corresponding to the better performance, i.e., $S1 = 7$. For the approximated function $f = \sin(2x)e^{-x}$, assuming that the range of variable

x is $[0.01, \pi]$, the corresponding identical sampling interval is 0.05. Assuming that the training sample was generated from the range of $[0.001, x]$ at the identical sampling interval of 0.03; and the test sample was generated from the range of $[0.002, x]$ at the identical sampling interval of 0.1. Fig. 3 illustrates the curves of the training errors and the testing errors for the three training algorithms.

When the value of x is smaller, there are less training samples and testing samples, it can be seen from Fig. 3 that the PSO algorithm has better training error and testing error than the BP algorithm. When the value of x is larger, there are more training samples and testing samples, it can be seen that the BP algorithm has better training error and testing error. The PSO–BP algorithm combines the PSO algorithm with the BP algorithm, in this example, the hybrid algorithm combines the advantages of both the PSO algorithm and the BP algorithm, so from Fig. 3, it can be seen that the PSO–BP algorithm has better training error and testing error.

4.3. Example 3. Iris classification problem

Iris classification problem is used as benchmark data widely in the ANN field, it is a for dimensional pattern recognition problem with three classes. So we use an FNN with the structure of 4–$S1$–3 to address this problem, where $S1 = 4, 5, \ldots, 16$. A training way, referred to as left-one cross validation, is adopted. Supposed the number of samples is N. For left-one cross validation, $N - 1$ samples are used to training the FNN, while one sample is used to test generalization ability. After the next generation, another $N - 1$ samples are used to train the FNN, the left one sample is used to give a test. This process is continued to cycle N times until every sample is made sure to have been tested for its generalization capability.

We used the second kind condition of PSO–BP algorithm, supposing that the maximal generation is 5, while the PSO algorithm was finished, the algorithm was switched to the gradient method to search for 500 times. For the PSO algorithm, the maximal generation is assumed as 500. For the BP algorithm, the maximal iteration is assumed as 3000 times. Assumed that under fixed hidden nodes $S1 = 4, 5, \ldots, 16$, every procedure was run successively for five times, the best recognition rate was selected among these five results, then the mean recognition rate was calculated for these five results.

From Fig. 4, it can be found that the PSO–BP algorithm and the PSO algorithm has similar maximal recognition rates, the PSO–BP algorithm has better recognition rate than the BP algorithm; but the PSO–BP algorithm has better mean recognition rate than the BP algorithm and the PSO algorithm. This shows that the PSO–BP algorithm is more stable, while in training process, the PSO–BP algorithm uses less CPU time than the BP algorithm and the PSO algorithm.

Apparently, the PSO algorithm has better recognition rate than the BP algorithm, although there are enough hidden nodes. The best recognition rate for the BP algorithm can achieve 96.67%, while the best recognition rate of the PSO algorithm can reach around 98% under every fixed hidden node. When the number of hidden nodes is 15, the best recognition rate can reach 99.33%. The proposed PSO–BP algorithm in this paper

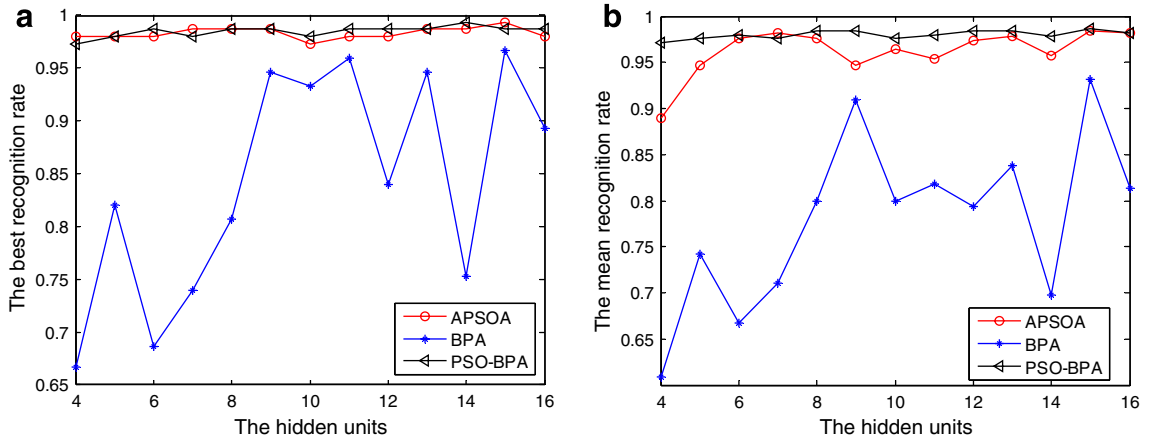

Fig. 4. The correct recognition rate for IRIS classification problem. (a) The best recognition rate. (b) The average correct recognition rate.

also has better recognition rate, its best recognition rate can also achieved around 98% under every fixed hidden nodes. When the number of hidden nodes is 13, the best recognition rate can achieve 99.33%. So the best recognition rate for the PSO–BP algorithm is similar to the best recognition rate for the PSO algorithm. The mean recognition rates of the PSO and the PSO–BP algorithms are better than the BP algorithm apparently, but the mean recognition rate of the PSO–BP algorithm is better than the PSO algorithm. The reason is that a local gradient descending method is used to search around global optimum, the searching efficiency is improved significantly.

5. Conclusion

In this paper, we have proposed a hybrid PSO algorithm, the hybrid algorithm combining the adaptive PSO algorithm with adaptive back-propagation algorithm, which is to combine the particle swarm optimization algorithm's strong ability in global search and the back-propagation algorithm's strong ability in local search. We can get better search result using this hybrid algorithm. In this hybrid algorithm, the initial particles are distributed randomly in the problem space, a global searching is assured around the global optimum. But due to the PSO algorithm having a poor capability in searching for global optimum, we adopt some heuristic knowledge to transit from the PSO algorithm searching to the gradient descending based on BP algorithm searching. In adaptive particle swarm optimization (PSO) algorithm, we introduced a different selection strategy for inertial weight w. In the initial searching stage, we wish that the searching inertial weight could reduce rapidly, so that the global optima can be achieved rapidly. And then around global optimum, we reduced the inertial weight more smoothly, so that a higher accuracy can be achieved. We introduced this kind of the selection strategy of inertial weight w into the PSO–BP algorithm. In the PSO–BP algorithm, to know when the searching process is transited from the particle swarm search to the gradient descending search, a heuristic way was introduced. That is when the best fitness value in the history of all particles has not changed for some generations (i.e., ten generations), the search process would be transferred to the gradient descending search. When the best fitness has not changed for some generations, all the particles may lose the ability of finding a better solution, at this time, gradient descending search used can get better results. The heuristic way is used to avoid wasting too much CPU time for vain search using particle swarm, so the searching efficiency of the PSO–BP algorithm is improved greatly.

From the conducted experiments, we can get conclusions that for the same goal, the PSO–BP algorithm spends less CPU time than the PSO algorithm and the BP algorithm, in other words, the PSO–BP algorithm uses less CPU time to get higher training accuracy than the PSO algorithm and the BP algorithm. From example 3, we can also see that the PSO–BP algorithm has more smooth mean recognition rate. This shows that the PSO–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] R.C. Eberhart, J. Kennedy, A new optimizer using particles swarm theory, in: Proc. of Sixth Int. Symp. on Micro Machine and Human Science, Nagoya, Japan (1995) 39–43.
[2] R.C. Eberhart, J. Kennedy, Particle swarm optimization, in: Proc. of IEEE Int. Conf. on Neural Network, Perth, Australia (1995) 1942–1948.
[3] Y. Shi, R.C. Eberhart, A modified particle swarm optimizer, in: Proc. of IEEE World Conf. on Computation Intelligence (1998) 69–73.
[4] Y. Shi, R.C. Eberhart, Empirical study of Particle Swarm Optimization, in: Proc. of IEEE World Conference on Evolutionary Computation (1999) 6–9.
[5] X. Yao, A review of evolutionary artificial neural networks, Int. J. Intell. Syst., 8(4) (1993) 539–567.
[6] P.J. Angeline, G.M. Sauders, J.B. Pollack, An evolutionary algorithm that constructs recurrent neural networks, IEEE Trans. Neural Networks 5 (1) (1994) 54–65.
[7] Marco Gori, Alberto Tesi, On the problem of local minima in back-propagation, IEEE Trans. Pattern Anal. Mach. Intell. 14 (1) (1992) 76–86.
[8] A. van Ooyen, B. Nienhuis, Improving the convergence of the back-propagation algorithm, Neural Network 5 (4) (1992) 465–471.
[9] M. Ahmad, F.M.A. Salam, supervised learning using the Cauchy energy function, in: Proc. of International Conference ON Fuzzy logic and Neural Networks, 1992.

[10] R.A. Jacobs, Increased rates of convergence through learning rate adaptation, Neural Networks 1 (1988) 295–307.
[11] M.K. Weirs, A method for self-determination of adaptive learning rates in back propagation, Neural Networks 4 (1991) 371–379.
[12] B. Irie, S. Miyake, Capability of three-layered perceptrons, in: Proc. of IEEE Int. Conf. On Neural Networks, San Diego, USA (1998) 641–648.
[13] S. Shaw, W. Kinsner, Chaotic simulated annealing in multilayer feedforward networks, in: Proc. of Canadian Conf. on Electrical and Computer Engineering, vol. 1 (1996) 265–269.
[14] Seop Koh Chang, O.A. Mohammed, Song-Yop Hahn, Detection of magnetic body using article neural network with modified simulated annealing, IEEE Trans. Magn. 30 (1994) 3644–3647.
[15] U. Bhattacharya, S.K. Parui, The Use of Self-adaptive learning rates to improve the convergence of backpropagation algorithm, Tech. Rep. GVPR-1/95, CVPR Unit, Indian Statistical Institute, Calcutta, India, 1995.
[16] Chunkai Zhang, Huihe Shao, Yu Li, Particle swarm optimization for evolving artificial neural network, in: Proc. of IEEE Int. Conf. on System, Man, and Cybernetics, vol. 4 (2000) 2487–2490.
[17] Y.H. Shi, R.C. Eberhart, Parameter selection in particle swarm optimization, in: Proc. of 1998 Annual conference on Evolutionary Programming, San Diego, 1998.
[18] J. Salerno, Using the particle swarm optimization technique to train a recurrent neural model, in: Proc. of Ninth IEEE Int. Conf. on Tools with Artificial Intelligence (1997) 45–49.
[19] R.C. Eberhart, Y. Shi, Comparing Inertia Weights and Constriction Factors in Particle swarm Optimization, in: Proc. of 2000 congress on Evolutionary Computing, vol. 1 (2000) 84–88.
[20] K. Homik, Multilayer feedforward networks are universal approximators, Neural Networks 2 (1989) 359–366.
[21] D.W. Boeringer, D.H. Werner, Particle swarm optimization versus genetic algorithms for phased array synthesis, IEEE Trans. Antennas Propagation 52 (3) (2004) 771–779.
[22] Cui Zhihua, Zeng Jianchao, Nonlinear particle swarm optimizer: Framework and the implementation of optimization, control, in: Proc. of Automation, Robotics and Vision Conference, 2004. ICARCV 2004 8th, vol. 1 (2004) 238–241.
[23] J. Kennedy, R.C. Eberhart, A discrete binary version of the particle swarm algorithm, System, Man, and Cybernetics, in: Proc. of Computational Cybernetics and Simulation, 1997 IEEE international conference on, vol. 5, 12–15, October 1997, vol. 2, 980–984.
[24] L. Schoofs, B. Naudts, Swarm Intelligence on the binary constraint satisfaction problem, Evolutionary Computation 2002. CEC 02, in: Proc. of the 2002 Congress, vol. 2, 12–17 May (2002) 1444–1449.
[25] D.S. Chen, R.C. Jain, A robust Back-propagation Algorithm for Function Approximation, IEEE Trans. Neural Network 5 (1994) 467–479.
[26] Cheng-Jian Lin, Cheng-Hung, Chi-Yung Lee, A self-adaptive quantum radial basis function network for classification applications, in: Proc. of 2004 IEEE International Joint Conference on Neural Networks, vol. 4, 25–29 July (2004) 3263–3268.
[27] J.J.F. Cerqueira, A.G.B. Palhares, M.K. Madrid, A simple adaptive back-propagation algorithm for multilayered feedforward perceptrons, in: Proc. of IEEE International Conference on Systems, Man and Cybernetics, volume 3, 6–9 October (2002) 6.
[28] D.E. Rumelhart, G.E. Hinton, R.J. Willianms, Learning internal representations by error propagating, in: D.E. Rumelhart, J.L. McClelland (Eds.), Parallel Distributed Processing: Exploration in the Microstructure of Cognition, MIT Press, Cambridge, MA, 1986 (Chapter 8).
[29] C.M. Kuan, K. Hornik, Convergence of learning algorithms with constant learning rates, IEEE Trans. Neural Networks 2 (5) (1991) 484–489.
[30] P. Baldi, Gradient descent learning algorithms overview: A general dynamical systems perspective, IEEE Trans. Neural Networks 6 (1) (1995) 82–195.
[31] D.S. Huang, Systematic Theory of Neural Networks for Pattern Recognition, Publishing House of Electronic Industry of China, Beijing, 1996.
[32] P.J. Werbos, Beyond regression: New tools for predictions and analysis in the behavioral science, Ph.D. Thesis, Harvard University, 1974.