Applied Mathematics and Computation 218 (2012) 11125–11137

Contents lists available at SciVerse ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Training feedforward neural networks using hybrid particle swarm optimization and gravitational search algorithm

SeyedAli Mirjalili *, Siti Zaiton Mohd Hashim, Hossein Moradian Sardroudi

Soft Computing Research Group, Faculty of Computer Science and Information Systems, Universiti Teknologi Malaysia, 81310 UTM Skudai, Johor, Malaysia

ARTICLE INFO

Keywords:
FNN
Neural network
Learning neural network
Gravitational search algorithm
Particle swarm optimization
PSO
Evolutionary algorithm
Multilayer perceptron

ABSTRACT

The Gravitational Search Algorithm (GSA) is a novel heuristic optimization method based on the law of gravity and mass interactions. It has been proven that this algorithm has good ability to search for the global optimum, but it suffers from slow searching speed in the last iterations. This work proposes a hybrid of Particle Swarm Optimization (PSO) and GSA to resolve the aforementioned problem. In this paper, GSA and PSOGSA are employed as new training methods for Feedforward Neural Networks (FNNs) in order to investigate the efficiencies of these algorithms in reducing the problems of trapping in local minima and the slow convergence rate of current evolutionary learning algorithms. The results are compared with a standard PSO-based learning algorithm for FNNs. The resulting accuracy of FNNs trained with PSO, GSA, and PSOGSA is also investigated. The experimental results show that PSOGSA outperforms both PSO and GSA for training FNNs in terms of converging speed and avoiding local minima. It is also proven that an FNN trained with PSOGSA has better accuracy than one trained with GSA.

1. Introduction

Recently, Feedforward Neural Networks (FNNs), especially FNNs with two layers [1], have been widely used. In fact, FNNs with two layers are the most popular neural network in practical applications. Generally, an FNN with two layers is suitable for classifications of nonlinearly separable patterns [2,3] and to approximate functions [4,5]. It has been proven that two-layer FNNs can approximate any continuous and discontinuous function [4]. Learning is an essential part of any neural network and becomes attractive for many researchers. For FNNs, most applications have used the standard [6] or improved [7–9] Back-Propagation (BP) algorithm as their training method. The BP algorithm is a gradient-based algorithm which has some drawbacks, such as slow convergence [10] and the tendency to become trapped in local minima [11].

In FNNs, during the learning process, the goal is to find the best combination of connection weights and biases in order to achieve the minimum error. However, most of the time FNNs converge to points which are the best solution locally but not globally. In other words, learning algorithms lead FNNs to local minima rather than the global minimum. According to [10], the convergence of the BP algorithm is highly dependent on the initial values of weights, biases, and its parameters. These parameters include learning rate and momentum. In the literature, using novel heuristic optimization methods or evolutionary algorithms is a popular solution to enhance the problems of BP-based learning algorithms.

Various heuristic optimization methods have been used to train FNNs, such as Simulated Annealing (SA) [12,13], Genetic Algorithms (GAs) [14], Particle Swarm Optimization (PSO) algorithms [15–19], Magnetic Optimization Algorithm (MOA) [20], and Differential Evolution (DE) [21]. According to [10], some, such as SA and GA, could reduce the probability of trapping in local minima, but they still suffer from slow convergence rates. According to [16,22], PSO is one of the most

* Corresponding author.
E-mail addresses: ali.mirjalili@gmail.com (S. Mirjalili), sitizaiton@utm.my (S.Z. Mohd Hashim), hosseinmoradian@gmail.com (H. Moradian Sardroudi).

0096-3003/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.amc.2012.04.069

efficient and practical optimization algorithms in terms of reducing both of the aforementioned drawbacks. In this paper, the efficiencies of GSA and PSOGSA are investigated for training FNNs in comparison with PSO [23].

The rest of the paper is organized as follows: Sections 2–4 present a brief introduction to the concepts of PSO, GSA, and PSOGSA, respectively. Section 5 discusses the method of applying PSO, GSA, and PSOGSA to FNNs as evolutionary training algorithms. The experimental results are demonstrated in Section 6. Finally, Section 7 concludes the paper.

2. Particle swarm optimization

PSO is an evolutionary computation technique which is proposed by Kennedy and Eberhart [24]. The PSO was inspired by the social behaviour of bird flocking. It uses a number of particles (candidate solutions) which fly around in the search space to find the best solution. Meanwhile, the particles all look at the best particle (best solution) in their paths. In other words, particles consider their own best solutions as well as the best solution found so far.

Each particle in PSO should consider the current position, the current velocity, the distance to pbest, and the distance to gbest in order to modify its position. PSO was mathematically modelled as follows:

$$v_i^{t+1} = w v_i^t + c_1 \times rand \times (pbest_i - x_i^t) + c_2 \times rand \times (gbest - x_i^t), \tag{2.1}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1}, \tag{2.2}$$

where v_i^t is the velocity of particle i at iteration t, w is a weighting function, c_j is an acceleration coefficient, rand is a random number between 0 and 1, x_i^t is the current position of particle i at iteration t, $pbest_i$ is the pbest of agent i at iteration t, and $gbest$ is the best solution so far.

The first part of (2.1), $w v_i^t$, provides exploration ability for PSO. The second and third parts, $c_1 * rand * (pbest_i - x_i^t)$ and $c_2 * rand * (gbest - x_i^t)$, represent private thinking and collaboration of particles respectively. The PSO starts by randomly placing the particles in a problem space. In each iteration, the velocities of particles are calculated using (2.1). After defining the velocities, the positions of particles can be calculated as (2.2). The process of changing particles' positions will continue until an end criterion is met.

3. Gravitational search algorithm

In 2009, Rashedi et al. [25] proposed a new heuristic optimization algorithm called the Gravitational Search Algorithm (GSA) for finding the best solution in problem search spaces using physical rules. The basic physical theory from which GSA is inspired is Newton theory, which says: "*Every particle in the universe attracts every other particle with a force that is directly proportional to the product of their masses and inversely proportional to the square of the distance between them*" [26]. GSA can be considered as a collection of agents (candidate solutions) which have masses proportional to their value of fitness function. During generations all masses attract each other by the gravity forces between them. The heavier the mass, the bigger the attraction force. Therefore, the heaviest masses which are probably close to the global minimum attract the other masses in proportion to their distances.

According to [25,27], suppose there is a system with N agents. The position of each agent (masses) which is a candidate solution for the problem is defined as follows:

$$X_i = (x_i^1, \ldots, x_i^d, \ldots, x_i^n), \quad i = 1, 2, \ldots, N, \tag{3.1}$$

where N is the dimension of the problem and x_i^d is the position of the ith agent in the dth dimension.

The algorithm starts by randomly placing all agents in a search space. During all epochs, the gravitational forces from agent j on agent i at a specific time t are defined as follows:

$$F_{ij}^d(t) = G(t) \frac{M_{pi}(t) \times M_{aj}(t)}{R_{ij}(t) + \varepsilon} \left(x_j^d(t) - x_i^d(t) \right), \tag{3.2}$$

where M_{aj} is the active gravitational mass related to agent j, M_{pi} is the passive gravitational mass related to agent i, $G(t)$ is a gravitational constant at time t, ε is a small constant, and $R_{ij}(t)$ is the Euclidian distance between two agents i and j.

The gravitational constant G and the Euclidian distance between two agents i and j are calculated as follows:

$$G(t) = G_0 \times exp(-\alpha \times iter / maxiter), \tag{3.3}$$

$$R_{ij}(t) = \|X_i(t), X_j(t)\|_2, \tag{3.4}$$

where α is the descending coefficient, G_0 is the initial gravitational constant, $iter$ is the current iteration, and $maxiter$ is the maximum number of iterations.

In a problem space with the dimension d, the total force that acts on agent i is calculated by the following equation:

$$F_i^d(t) = \sum_{j=1, j \neq i}^{N} rand_j F_{ij}^d(t), \tag{3.5}$$

where $rand_j$ is a random number in the interval [0,1].

According to the law of motion, the acceleration of an agent is proportional to the resultant force and inverse of its mass, so the accelerations of all agents are calculated as follows:

$$a_i^d(t) = \frac{F_i^d(t)}{M_{ii}(t)}, \tag{3.6}$$

where d is the dimension of the problem, t is a specific time, and M_i is the mass of object i.

The velocity and position of agents are calculated as follows:

$$v_i^d(t+1) = rand_i \times v_i^d(t) + a_i^d(t), \tag{3.7}$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1), \tag{3.8}$$

where d is the problem dimension and $rand_i$ is a random number in the interval [0,1].

As can be inferred from (3.7) and (3.8), the current velocity of an agent is defined as a fraction of its last velocity ($0 \leqslant rand_i \leqslant 1$) added to its acceleration. Furthermore, the current position of an agent is equal to its last position added to its current velocity.

Agents' masses are defined using fitness evaluation. This means that an agent with the heaviest mass is the most efficient agent. According to the above equations, the heavier the agent, the higher the attraction force and the slower the movement. The higher attraction is based on the law of gravity (3.2), and the slower movement is because of the law of motion (3.6) [25].

The masses of all agents are updated using the following equations:

$$m_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)}, \tag{3.9}$$

where $fit_i(t)$ is the fitness value of the agent i at time t; $best(t)$ is the strongest agent at time t, and $worst(t)$ is the weakest agent at time t.

$best(t)$ and $worst(t)$ for a minimization problem are calculated as follows:

$$best(t) = \min_{j\in\{1,\ldots,N\}} fit_j(t), \tag{3.10}$$

$$\text{worst(t)} = \max_{j\in\{1,\ldots,N\}} fit_j(t). \tag{3.11}$$

$best(t)$ and $worst(t)$ for a maximization problem are calculated as follows:

$$\text{best(t)} = \max_{j\in\{1,\ldots,N\}} fit_j(t) \tag{3.12}$$

$$worst(t) = \min_{j\in\{1,\ldots,N\}} fit_j(t). \tag{3.13}$$

The normalization of the calculated masses (3.9) is defined by the following equation:

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)}. \tag{3.14}$$

In the GSA, at first all agents are initialized with random values. Each agent is a candidate solution. After initialization, the velocity and position of all agents will be defined using (3.7) and (3.8). Meanwhile, the other parameters such as the gravitational constant and masses will be calculated by (3.3) and (3.9). Finally, the GSA will be stopped by meeting an end criterion. The steps of GSA are represented in Fig. 1.

In all population-based algorithms which have social behaviour like PSO and GSA, two intrinsic characteristics should be considered: the ability of the algorithm to explore whole parts of search spaces and its ability to exploit the best solution. Searching through the whole problem space is called exploration whereas converging to the best solution near a good solution is called exploitation. A population-based algorithm should have these two vital characteristics to guarantee finding the best solution. In PSO, the exploration ability has been implemented using *Pbest* and the exploitation ability has been implemented using *Gbest*. In GSA, by choosing proper values for the random parameters (G_0 and α), the exploration can be guaranteed and slow movement of heavier agents can guarantee the exploitation ability [25,28].

Rashedi et al. [25] provided a comparative study between GSA and some well-known heuristic optimization algorithms like PSO. The results proved that GSA has merit in the field of optimization. However, GSA suffers from slow searching speed in the last iterations [29]. In this paper a hybrid of this algorithm with PSO, called PSOGSA, is proposed in order to improve this weakness.

4. The hybrid PSOGSA algorithm

The basic idea of PSOGSA is to combine the ability for social thinking (gbest) in PSO with the local search capability of GSA. In order to combine these algorithms, (4.1) is proposed as follows:

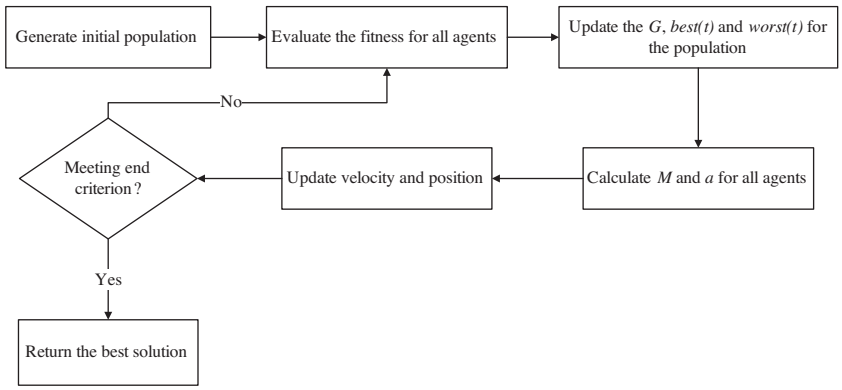

Fig. 1. General steps of the gravitational search algorithm [25].

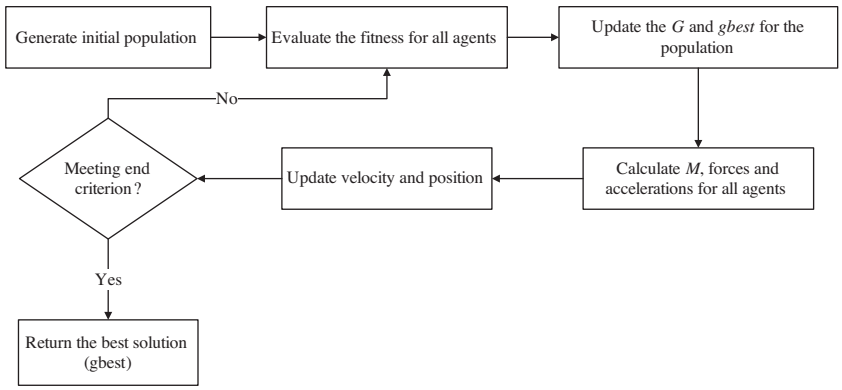

Fig. 2. Steps of PSOGSA [30].

$$V_i(t+1) = w \times V_i(t) + c_1' \times rand \times ac_i(t) + c_2' \times rand \times (gbest - X_i(t)), \tag{4.1}$$

where $V_i(t)$ is the velocity of agent i at iteration t, c_j' is an acceleration coefficient, w is a weighting function, rand is a random number between 0 and 1, $ac_i(t)$ is the acceleration of agent i at iteration t, and $gbest$ is the best solution so far.

In each iteration, the positions of agents are updated as follows:

$$X_i(t+1) = X_i(t) + V_i(t+1). \tag{4.2}$$

In PSOGSA, at first, all agents are randomly initialized. Each agent is considered as a candidate solution. After initialization, the gravitational force, gravitational constant, and resultant forces among agents are calculated using (3.2), (3.3) and (3.5) respectively. After that, the accelerations of particles are defined as (3.6). In each iteration, the best solution so far should be updated. After calculating the accelerations and updating the best solution so far, the velocities of all agents can be calculated using (4.1). Finally, the positions of agents are updated by (4.2). The process of updating velocities and positions will be stopped by meeting an end criterion. The steps of PSOGSA are represented in Fig. 2.

To see how PSOGSA is efficient, the following remarks are noted:

- In PSOGSA, the quality of solutions (fitness) is considered in the updating procedure.
- The agents near good solutions try to attract the other agents which are exploring different parts of the search space.
- When all agents are near a good solution, they move very slowly. In this case, gbest helps them to exploit the global best.
- PSOGSA uses a memory (gbest) to save the best solution found so far, so it is accessible at any time.
- Each agent can observe the best solution (gbest) and tend toward it.
- By adjusting c_1' and c_2', the abilities of global searching and local searching can be balanced.

The above-mentioned remarks make PSOGSA powerful enough to solve a wide range of optimization problems [30]. In the following subsections, mechanisms for training FNNs using PSO, GSA, and PSOGSA, called FNNPSO, FNNGSA, and FNNPSOGSA, respectively, are introduced and evaluated.

5. PSO, GSA, and PSOGSA for training FNNs

Generally, there are three methods of using a heuristic algorithm for training FNNs. First, heuristic algorithms are used for finding a combination of weights and biases which provide the minimum error for an FNN. Second, heuristic algorithms are employed as a way to find a proper structure for an FNN in a particular problem. The last method is to use an evolutionary algorithm to tune the parameters of a gradient-based learning algorithm, such as the learning rate and momentum.

In the first case, the structure is fixed before training FNNs. The duty of a training algorithm is to find a proper value for all connection weights and biases in order to minimize the overall FNNs' error. In the second case, the structures of FNNs vary. A training algorithm is applied to an FNN to determine the best structure for a certain problem. Changing the structure can be accomplished by manipulating the connections between neurons, the number of hidden layers, and the number of hidden nodes in each layer of the FNN.

In this paper, PSO, GSA, and PSOGSA are applied to an FNN using the first method; these mechanisms are called FNNPSO, FNNGSA, and FNNPSOGSA, respectively. This means that the structure of the FNN is fixed; PSO, GSA, and PSOGSA find a combination of weights and biases which yield the minimum error for the FNN. In order to design FNNPSO, FNNGSA, and FNNPSOGSA, the following basic elements need to be defined.

First, a fitness function using the error of the FNN should be defined to evaluate agents' fitness in FNNPSO, FNNGSA, and FNNPSOGSA. Second, an encoding strategy should be defined to encode the weights and biases of the FNN for the agents of FNNPSO, FNNGSA, and FNNPSOGSA. These elements are described below.

5.1. Fitness function

The fitness function which is used in this article is calculated as follows [10]:

Fig. 3 shows an FNN with two layers (one input, one hidden, and one output layer), where the number of input nodes is equal to n, the number of hidden nodes is equal to h, and the number of output nodes is m. In each epoch of learning, the output of each hidden node is calculated as follows:

$$f(s_j) = 1/\left(1 + \exp\left(-\left(\sum_{i=1}^{n} w_{ij} \cdot x_i - \theta_j\right)\right)\right), \quad j = 1, 2, \ldots, h \tag{5.1}$$

where $s_j = \sum_{i=1}^{n} w_{ij} \cdot x_i - \theta_j$, n is the number of the input nodes, w_{ij} is the connection weight from the ith node in the input layer to the jth node in the hidden layer, θ_j is the bias (threshold) of the jth hidden node, and x_i is the ith input.

After calculating outputs of the hidden nodes, the final output can be defined as follows:

$$o_k = \sum_{i=1}^{h} w_{kj} \cdot f(s_j) - \theta_k, \quad k = 1, 2, \ldots, m, \tag{5.2}$$

where w_{kj} is the connection weight from the jth hidden node to the kth output node and θ_k is the bias (threshold) of the kth output node.

Finally, the learning error E (fitness function) is calculated as follows:

$$E_k = \sum_{i=1}^{m} \left(o_i^k - d_i^k\right)^2 \tag{5.3}$$

$$E = \sum_{k=1}^{q} \frac{E_k}{q} \tag{5.4}$$

where q is the number of training samples, d_i^k is the desired output of the ith input unit when the kth training sample is used, and y_i^k is the actual output of the ith input unit when the kth training sample is used.

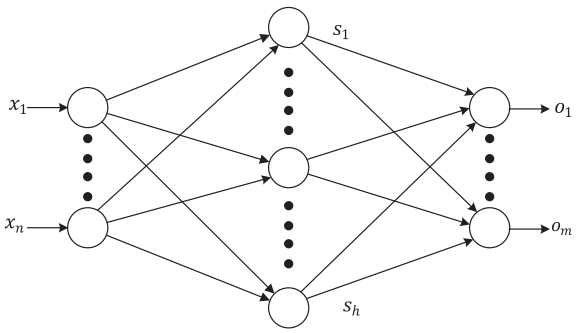

Fig. 3. Structure of two-layer FNNs.

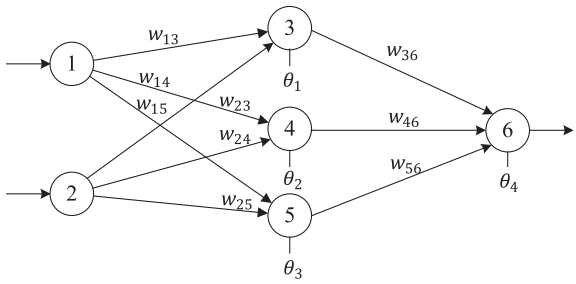

Fig. 4. FNN with a 2-3-1 structure.

Therefore, the fitness function of the ith training sample can be defined as follows:

$$Fitness(X_i) = E(X_i) \tag{5.5}$$

5.2. Encoding strategy

After defining the fitness function for FNNPSO, FNNGSA, and FNNPSOGSA, the next stage is to choose an encoding strategy in order to represent the weights and biases of the FNN for every agent in FNNPSO, FNNGSA, and FNNPSOGSA. According to [10], there are three methods of encoding and representing the weights and biases of FNNs for every agent in evolutionary algorithms. These are the vector, matrix, and binary encoding methods. In vector encoding, every agent is encoded as a vector. To train an FNN, each agent represents all the weights and biases of the FNNs structure. In matrix encoding, every agent is encoded as a matrix. In binary encoding, agents are encoded as strings of binary bits. Each of these strategies has its own advantages and disadvantages that can be useful in a particular problem.

According to [10], in the first strategy, the encoding phase is easy, but the decoding process (decoding particles' vectors to a weights and biases matrix) is complicated. This method is often used in the function optimization field. In the second strategy, the decoding stage is easy but the encoding is difficult for neural networks with complex structures. This method is very suitable for the training processes of neural networks because the encoding strategy makes it easy to execute decoding for neural networks. In the third strategy, we need to represent particles' variables in binary form. The length of each particle will increase when the structure becomes more complex. Therefore, the process of decoding and encoding becomes very complicated.

In this article, the matrix encoding strategy has been used because we are dealing with training FNNs. An example of this encoding strategy for the FNN of Fig. 4 is provided as follows:

$$particle(:,:,i) = \left[W_1, B_1, W_2', B_2\right], \tag{5.6}$$

$$W_1 = \begin{bmatrix} w_{13} & w_{23} \\ w_{14} & w_{24} \\ w_{15} & w_{25} \end{bmatrix}, \quad B_1 = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}, \quad W_2' = \begin{bmatrix} w_{36} \\ w_{46} \\ w_{56} \end{bmatrix}, \quad B_2 = [\theta_4], \tag{5.7}$$

where W_1 is the hidden layer weight matrix, B_1 is the hidden layer bias matrix, W_2 is the output layer weight matrix, W_2' is the transpose of W_2, and B_2 is the hidden layer bias matrix.

6. Results and discussion

In this section, three benchmark problems are used to compare the abilities of FNNPSO, FNNGSA, and FNNPSOGSA in training FNNs. They are three-bit parity, function approximation, and the Iris classification problem, which are presented in Sections 6.1, 6.2 and 6.3, respectively. In these problems, it is assumed that every particle is randomly initialized in the range of [0,1]. The other assumptions are as follows:

For FNNPSO, C_1 and C_2 are set to 2, r_1 and r_2 are two random numbers in the interval [0,1], w decreases linearly from 0.9 to 0.4, and the initial velocities of particles are randomly generated in the interval [0,1].

For FNNGSA, α is set to 20, the gravitational constant (G_0) is set to 1, the initial velocities of particles are randomly generated in the interval [0,1], and initial values of acceleration and mass are set to 0 for each particle.

For FNNPSOGSA, c_1' and c_2' are set to 1, w decreases linearly from 0.9 to 0.4, and the initial velocities of agents are randomly generated in the interval [0,1].

The population sizes of FNNPSO, FNNGSA, and FNNPSOGSA for the first, second, and third problems are equal to 50, 200, and 200, respectively.

Table 1
Three bits parity problem (3-bit XOR).

Input	Output
0 0 0	0
0 0 1	1
0 1 0	1
0 1 1	0
1 0 0	1
1 0 1	0
1 1 0	0
1 1 1	1

Table 2
Average, median, standard deviation, and best of MSE for all training samples over 30 independent runs for FNNPSO, FNNGSA, and FNNPSOGSA in a 3-bit XOR problem.

Hidden nodes (S)	Algorithm	Average MSE	Median MSE	Std dev MSE	Best MSE
5	*FNNPSOGSA*	**1.3175e−02**	**1.5760e−06**	**2.6085e−02**	**3.1772e−10**
	FNNPSO	2.2400e−02	2.2461e−03	3.6093e−02	8.9964e−09
	FNNGSA	1.7953e−01	1.9648e−01	5.5945e−02	4.3492e−02
6	*FNNPSOGSA*	**4.5125e−03**	**4.0238e−06**	**1.5887e−02**	1.2319e−09
	FNNPSO	8.8953e−03	8.5529e−04	2.0499e−02	**4.1943e−12**
	FNNGSA	1.4230e−01	1.6426e−01	6.4535e−02	3.7346e−02
7	*FNNPSOGSA*	**2.7122e−03**	**3.0954e−07**	**1.2818e−02**	1.4208e−11
	FNNPSO	1.0325e−02	5.3893e−04	2.7711e−02	**3.8274e−24**
	FNNGSA	1.2565e−01	1.4032e−01	6.5325e−02	1.2422e−02
8	*FNNPSOGSA*	**2.0399e−05**	**1.8349e−07**	**6.2884e−05**	1.2591e−11
	FNNPSO	5.1294e−03	1.3468e−05	2.2972e−02	**3.3166e−13**
	FNNGSA	1.1464e−01	1.0255e−01	7.6374e−02	7.1257e−03
9	*FNNPSOGSA*	**7.7205e−06**	**1.0106e−07**	**2.6532e−05**	5.5311e−12
	FNNPSO	5.0104e−03	3.6779e−05	2.2776e−02	**1.3058e−24**
	FNNGSA	9.4026e−02	6.9627e−02	5.8491e−02	2.1135e−02
10	*FNNPSOGSA*	**6.1340e−06**	**8.4584e−08**	**2.8868e−05**	**1.5503e−10**
	FNNPSO	1.2894e−02	4.9998e−05	3.8024e−02	4.4242e−09
	FNNGSA	8.0449e−02	7.0787e−02	5.4407e−02	1.0597e−02
11	*FNNPSOGSA*	**1.8259e−05**	**6.4874e−08**	**7.6978e−05**	4.6565e−10
	FNNPSO	4.3063e−03	2.3309e−06	2.2798e−02	**2.0963e−23**
	FNNGSA	7.7604e−02	2.2714e−02	4.3129e−02	1.2058e−02
13	*FNNPSOGSA*	**4.1675e−02**	**7.7728e−08**	**2.2822e−02**	1.6510e−10
	FNNPSO	9.2572e−03	9.9122e−05	3.1521e−02	**3.5649e−50**
	FNNGSA	6.5790e−02	6.4600e−02	3.1967e−02	1.0404e−02
15	*FNNPSOGSA*	**4.1675e−03**	**2.3626e−08**	2.2822e−02	4.6680e−11
	FNNPSO	5.0059e−03	3.1100e−05	**2.2812e−02**	**2.2852e−15**
	FNNGSA	6.9776e−02	6.1100e−02	4.1229e−02	9.2803e−03
20	*FNNPSOGSA*	**1.6813e−02**	**1.6077e−08**	**4.3167e−02**	**6.2804e−13**
	FNNPSO	3.3754e−02	3.9900E − 05	5.6000E − 02	1.2035E − 12
	FNNGSA	7.5058e−02	6.0155e−02	4.7876e−02	1.2029e−02
30	*FNNPSOGSA*	**4.1671e−03**	**9.5787e−10**	**2.2822e−02**	**8.9801e−14**
	FNNPSO	2.5016e−02	6.4170e−06	5.0838e−02	1.7768e−11
	FNNGSA	6.2372e−02	4.9286e−02	3.9109e−02	1.4439e−02

6.1. The N bits parity (XOR) problem

The N bit parity problem is a famous nonlinear benchmark problem. The problem is to recognize the number of "1s" in the input vector. The XOR result of the input vector should be returned. In other words, if the input vector contains an odd number of "1s", the output is "1". If the input vector contains an even number of "1s", the output is "0". Table 1 shows the inputs and desirable outputs of this problem for three bits:

The XOR problem is not linearly separable, and we cannot solve it using an FNN without hidden layers (Perceptron). In this section, we use an FNN with the structure 3-S-1 to solve this problem, where S is the number of hidden nodes, and we compare the performance of FNNs with $S = 5, 6, 7, 8, 9, 10, 11, 13, 15, 20$, and 30.

We compare FNNPSO, FNNGSA, and FNNPSOGSA based on the average, median, standard deviation, and best of the Mean Square Error (MSE) for all training samples (5.4) over 30 independent runs. The criterion for finishing the training process is

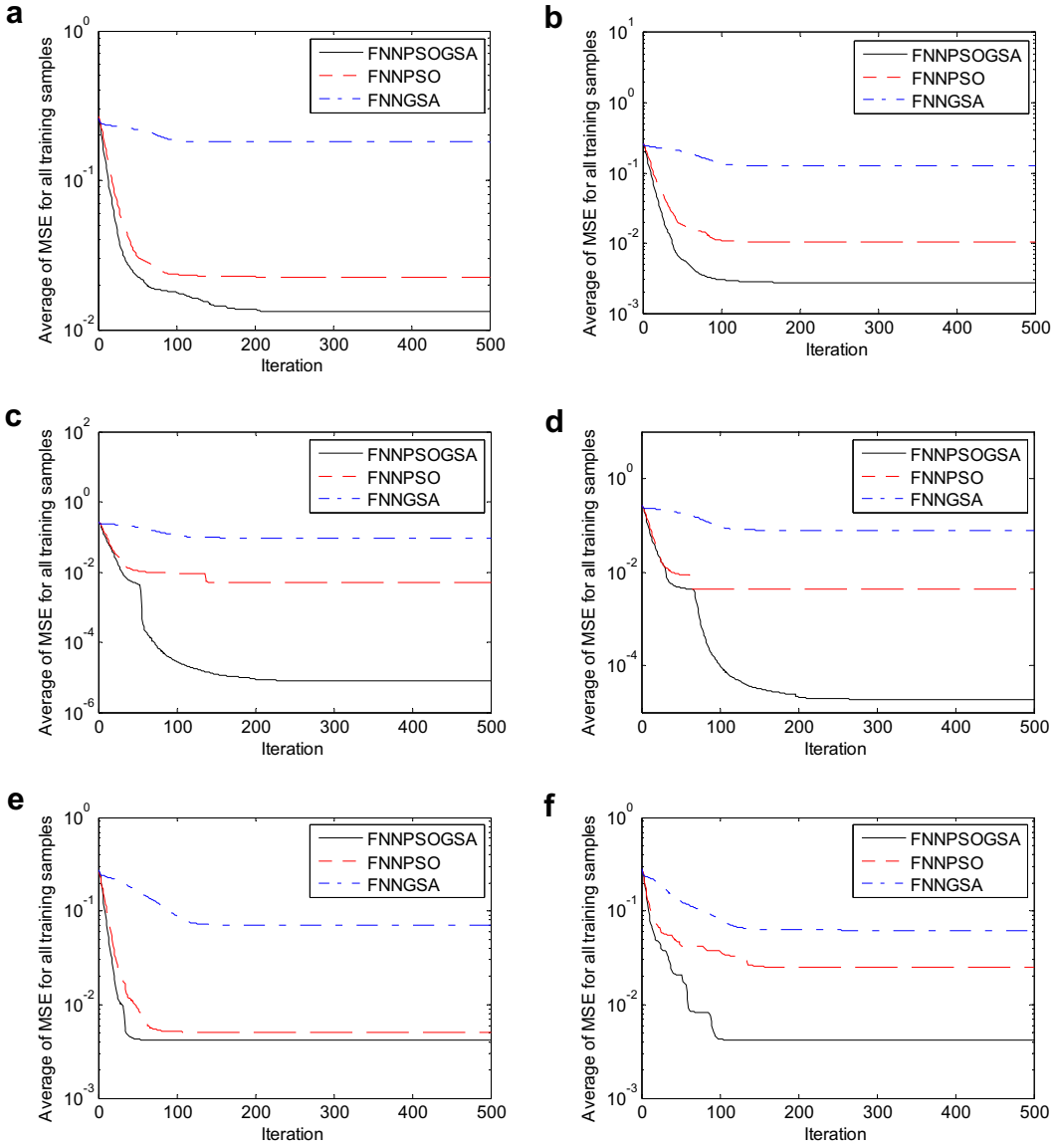

Fig. 5. Convergence curves of FNNPSO, FNNGSA, and FNNPSOGSA based on averages of MSE for all training samples over 30 independent runs in a 3-bit XOR problem. (a), (b), (c), (d), (e), and (f) are the convergence curves for FNNs with S = 5, 7, 9, 11, 15, and 30, respectively.

to complete the maximum number of iterations (equal to 500 in this paper). The experimental results for this problem are shown in Table 2. The best results are indicated in bold type.

From Table 2, it can be seen that the FNNPSOGSA apparently performs better than the FNNPSO and FNNGSA for the average, median, and standard deviation of MSE. The results of these statistical variables prove that FNNPSOGSA has the best ability to avoid local minima. For the best MSE, FNNPSO has better results, so the results show that FNNPSO is more accurate than FNNGSA and FNNPSOGSA.

Fig. 5 shows the convergence curves of FNNPSO, FNNGSA, and FNNPSOGSA based on averages of the MSE for all training samples over 30 independent runs. Parts (a), (b), (c), (d), (e), and (f) of Fig. 5 are the convergence curves for FNN with S = 5, 7, 9, 11, 15, and 30, respectively. These figures confirm that FNNPSOGSA apparently has the best convergence rate for all FNNs.

6.2. Function approximation problem

In this section, FNNs with the structure 1-S-1 are trained to approximate the function $y = \sin(2x)e^{-x}$, where S is the number of hidden nodes, and we compare the performance of FNNs with S = 3, 4, 5, 6, and 7. Fig. 6 illustrates this function. In this study, we use a dataset in the interval $[0,\pi]$ with increments of 0.03, so the number of training data is 105. The comparison

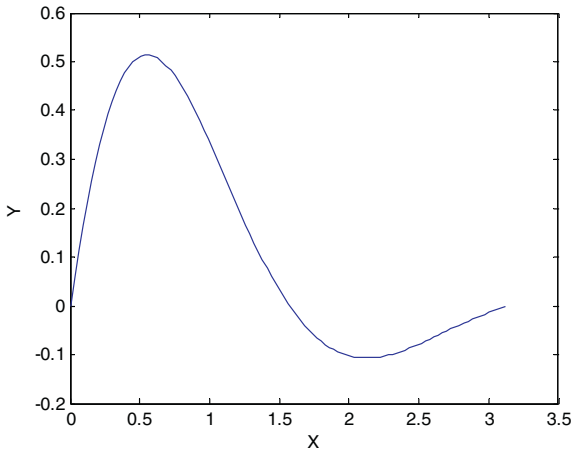

Fig. 6. Function $y = \sin(2x)e^{-x}$.

Table 3
Average, median, standard deviation, and best of MSE for all training samples over 30 independent runs for FNNPSO, FNNGSA, and FNNPSOGSA in the function approximation problem.

Hidden nodes (S)	Algorithm	Average MSE	Median MSE	Std dev MSE	Best MSE
3	*FNNPSOGSA*	**9.6113e−03**	**9.9619e−03**	**2.7267e−03**	5.5411e−03
	FNNPSO	1.7118e−02	1.1945e−02	1.7241e−02	**2.5563e−03**
	FNNGSA	6.882e−02	7.2232e−02	1.1630e−02	4.5684e−02
4	*FNNPSOGSA*	**7.4686e−03**	6.0540e−02	**3.4799e−03**	**1.7598e−03**
	FNNPSO	1.0292e−02	**9.7328e−03**	3.8812e−03	3.0212e−03
	FNNGSA	6.0834e−02	6.8970e−02	1.5084e−02	3.6454e−02
5	*FNNPSOGSA*	**7.2789e−03**	5.2391e−02	**3.6151e−03**	**1.4629e−03**
	FNNPSO	7.3565e−03	**7.1217e−03**	3.6697e−03	1.4705e−03
	FNNGSA	5.6646e−02	5.9142e−02	1.6858e−02	3.2538e−02
6	*FNNPSOGSA*	**6.0633e−03**	5.0239e−02	**3.5212e−03**	1.5045e−03
	FNNPSO	7.4143e−03	**7.1026e−03**	4.3036e−03	**1.4328e−03**
	FNNGSA	6.4594e−02	6.6303e−02	1.3256e−02	4.6777e−02
7	*FNNPSOGSA*	**6.7104e−03**	**5.4945e−03**	5.4070e−03	**9.5455e−04**
	FNNPSO	6.9547e−03	6.1180e−03	**3.9005e−03**	2.3657e−03
	FNNGSA	5.9256e−02	6.0936e−02	1.4731e−02	3.6341e−02

between FNNPSO, FNNGSA, and FNNPSOGSA for training FNNs in order to solve this benchmark problem is presented in Table 3 and Fig. 7. The best results are indicated in bold type.

The results in Table 3 show that FNNPSOGSA has better results for the average, median, and standard deviation of the MSE over 30 independent runs. These results prove that FNNPPSOGSA also has better ability than FNNPSO and FNNGSA to avoid local minima in this benchmark problem. For the best MSE over 30 runs, FNNPSOGSA and FNNPSO have very close results, but FNNPSO still has the most accurate results. As can be inferred from Table 3, the best performance is that of FNNPSOGSA for the FNN with $S = 7$.

Fig. 7 shows the convergence curves of FNNPSO, FNNGSA, and FNNPSOGSA based on averages of MSE for all training samples over 30 independent runs. Parts (a), (b), (c), (d), and (e) of Fig. 7 are the convergence curves for FNN with $S = 3$, 4, 5, 6, and 7, respectively. These figures confirm that FNNPSOGSA achieves the best convergence rate for all values of hidden numbers in this benchmark problem as well. FNNGSA has the worst convergence speed because of the slow searching process of its GSA.

6.3. Iris classification problem

The Iris classification problem has been widely used in the FNN field. The Iris dataset has 150 samples which can be divided into three classes: Setosa, Versicolor, and Virginica. All samples have four features: sepal length, sepal width, petal length, and petal width. Consequently, we use FNNs with the structure 4-S-3 to solve this classification problem, where S in the number of hidden nodes, and we compare the performance of FNNs with $S = 4$, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, and 15. Left-one cross-validation is used for training FNNs. In this method, in each generation, 149 different samples from the

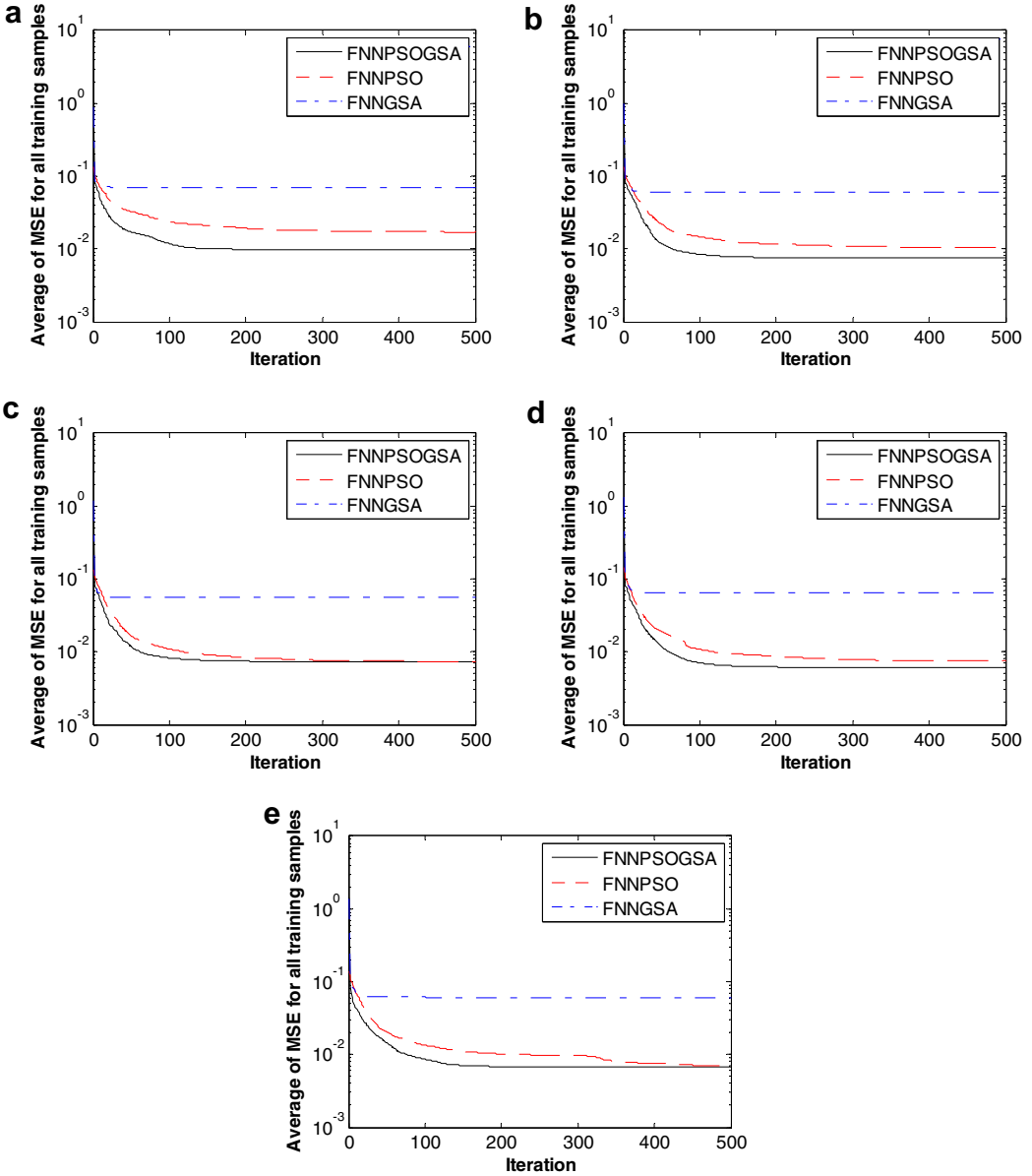

Fig. 7. Convergence curves of FNNPSO, FNNGSA, and FNNPSOGSA based on averages of MSE for all training samples over 30 independent runs in a function optimization problem. (a), (b), (c), (d), and (e) are the convergence curves for FNNs with $S = 3$, 4, 5, 6, and 7, respectively.

Iris dataset are selected and used to train the FNN, and the remaining sample is used to test the FNN. The results of training FNNs to solve this problem are presented in Table 4.

As shown in Table 4, for all hidden nodes, FNNPSOGSA has more than two of the best values for three statistical variables (average, median, and standard deviation) over 30 independent runs. These results prove that FNNPSOGSA also improves the capability of the FNN to avoid local minima in this benchmark problem. For the best MSE over 30 runs, FNNPSOGSA and FNNPSO have very close results, but FNNPSO still has the most accurate results in most of the cases.

Fig. 8 shows the convergence curves of FNNPSO, FNNGSA, and FNNPSOGSA based on averages of the MSE for all training samples over 30 independent runs. Parts (a), (b), (c), (d), and (e) of Fig. 8 are the convergence curves for FNNs with $S = 5$, 7, 9, 11, 13, and 15, respectively. These figures confirm that FNNPSOGSA has the best convergence rate for all values of hidden numbers.

From Fig. 9, it can be inferred that FNNPSOGSA has a better recognition rate than FNNPSO, and FNNPSO has a better recognition rate than FNNGSA. With regard to the mean recognition rate, FNNPSOGSA shows a more stable rate than FNNPSO and FNNGSA. With regard to the best recognition rate, PSOGSA also shows the best results for all FNNs. The best recognition

Table 4
Average, median, standard deviation, and best of MSE for all training samples over 30 independent runs for FNNPSO, FNNGSA, and FNNPSOGSA in the Iris classification problem.

Hidden nodes (S)	Algorithm	Average MSE	Median MSE	Std dev MSE	Best MSE
4	*FNNPSOGSA*	**2.0926e−02**	**2.0540e−02**	**2.8907e−03**	1.6538e−02
	FNNPSO	2.7648e−02	2.0880e−02	2.3825e−02	**9.0291e−03**
	FNNGSA	5.1936e−02	4.8319e−02	1.0393e−02	4.0493e−02
5	*FNNPSOGSA*	**1.9027e−02**	**1.7768e−02**	**2.7267e−03**	1.6285e−02
	FNNPSO	2.6816e−02	2.1881e−02	1.7528e−02	**1.3419e−02**
	FNNGSA	5.4355e−02	5.4115e−02	1.1371e−02	4.1670e−02
6	*FNNPSOGSA*	**1.6593e−02**	**1.5815e−02**	**2.4582e−03**	1.3613e−02
	FNNPSO	1.9053e−02	1.9382e−02	9.2553e−03	**9.3087e−03**
	FNNGSA	1.1541e−01	5.1797e−02	1.4374e−01	4.4200e−02
7	*FNNPSOGSA*	**1.6082e−02**	1.5533e−02	**1.8746e−03**	1.4428e−02
	FNNPSO	1.7181e−02	**1.4900e−02**	4.7723e−03	**1.3333e−02**
	FNNGSA	4.8626e−02	5.0297e−02	8.6222e−03	3.8858e−02
8	*FNNPSOGSA*	**1.9142e−02**	**1.5357e−02**	9.2745e−03	**1.4611e−02**
	FNNPSO	2.0554e−02	1.8288e−02	**5.2498e−03**	1.5378e−02
	FNNGSA	4.7041e−02	4.4749e−02	7.2863e−03	3.9507e−02
9	*FNNPSOGSA*	**1.6394e−02**	**1.6585e−02**	**4.5993e−03**	**1.0470e−02**
	FNNPSO	2.4615e−02	1.8431e−02	1.2549e−02	1.4001e−02
	FNNGSA	5.8214e−02	5.8671e−02	1.3757e−02	4.4793e−02
10	*FNNPSOGSA*	**1.6021e−02**	**1.6208e−02**	**1.1821e−03**	1.4421e−02
	FNNPSO	2.9265e−02	3.4564e−02	1.3342e−02	**1.3427e−02**
	FNNGSA	5.4245e−02	5.5153e−02	1.0514e−02	4.3230e−02
11	*FNNPSOGSA*	**1.6146e−02**	**1.5686e−02**	**2.0309e−03**	1.3840e−02
	FNNPSO	1.6203e−02	1.6036e−02	1.1120e−02	**1.0293e−03**
	FNNGSA	5.1535e−02	5.2502e−02	8.7323e−03	3.8938e−02
12	*FNNPSOGSA*	**1.6806e−02**	1.6272e−02	**2.9391e−03**	1.3849e−02
	FNNPSO	1.7889e−02	**1.4919e−02**	6.9515e−03	**1.3409e−02**
	FNNGSA	1.2098e−01	6.3867e−02	1.3702e−01	4.2516e−02
13	*FNNPSOGSA*	**1.6453e−02**	1.5672e−02	**1.8690e−03**	1.5421e−02
	FNNPSO	1.7012e−02	**1.5593e−02**	5.9498e−03	**9.8958e−03**
	FNNGSA	1.1424e−01	5.2206e−02	1.3744e−01	4.3250e−02
14	*FNNPSOGSA*	1.5770e−02	1.5502e−02	**1.1286e−03**	1.4592e−02
	FNNPSO	**1.4341e−02**	**1.3779e−02**	2.9371e−03	**1.0897e−02**
	FNNGSA	6.4177e−02	6.3476e−02	1.7704e−02	4.2140e−02
15	*FNNPSOGSA*	**1.4689e−02**	**1.5220e−02**	**1.8930e−03**	1.1709e−02
	FNNPSO	1.6457e−02	1.6382e−02	4.4336e−03	**1.0284e−02**
	FNNGSA	9.4039e−02	5.3571e−02	8.2463e−02	4.8570e−02

rate for FNNPSOGSA and FNNPSO for all hidden nodes is 99.33%, while the best recognition rate for FNNGSA is 98%. FNNPSOGSA reaches 99.33% for all hidden nodes. These results prove that FNNPSOGSA is capable of solving the Iris classification problem more reliably and accurately than FNNPSO and FNNGSA. FNNPSOGSA achieves this reliability because it reduces the probability of trapping in local minima.

Generally, in all results produced, it could be observed that FNNGSA does not give a good performance because of the slow searching process of the GSA, which affects FNNGSAs exploitation ability. However, GSA has strong exploration ability among all evolutionary algorithms [25]. Learning algorithms for FNNs need not only strong exploration ability but also precise exploitation ability. Referring to the results of the classification accuracy obtained by FNN based algorithms, it is shown that FNNPSO performs better than FNNGSA due to the more precise exploitation ability of PSO, but it still suffers from the problem of trapping in local minima. This weakness means that FNNPSO has unstable performance. The results obtained by FNNPSOGSA prove that it has both strong exploitation and good exploration abilities. In other words, the strength of PSO and GSA has been successfully utilized and gives outstanding performance in FNN training. This means that FNNPSOGSA is capable of solving the problem of trapping in local minima and gives fast convergence speed.

7. Conclusion

In this paper, two new training algorithms called FNNGSA and FNNPSOGSA are introduced and investigated utilizing GSA and PSOGSA. Three benchmark problems: 3-bit XOR, function approximation, and Iris classification, are employed to evaluate the efficiencies of these new learning algorithms. The results are compared with FNNPSO. For all benchmark problems, FNNPSOGSA shows better performance in terms of convergence rate and avoidance of local minima. It is observed that

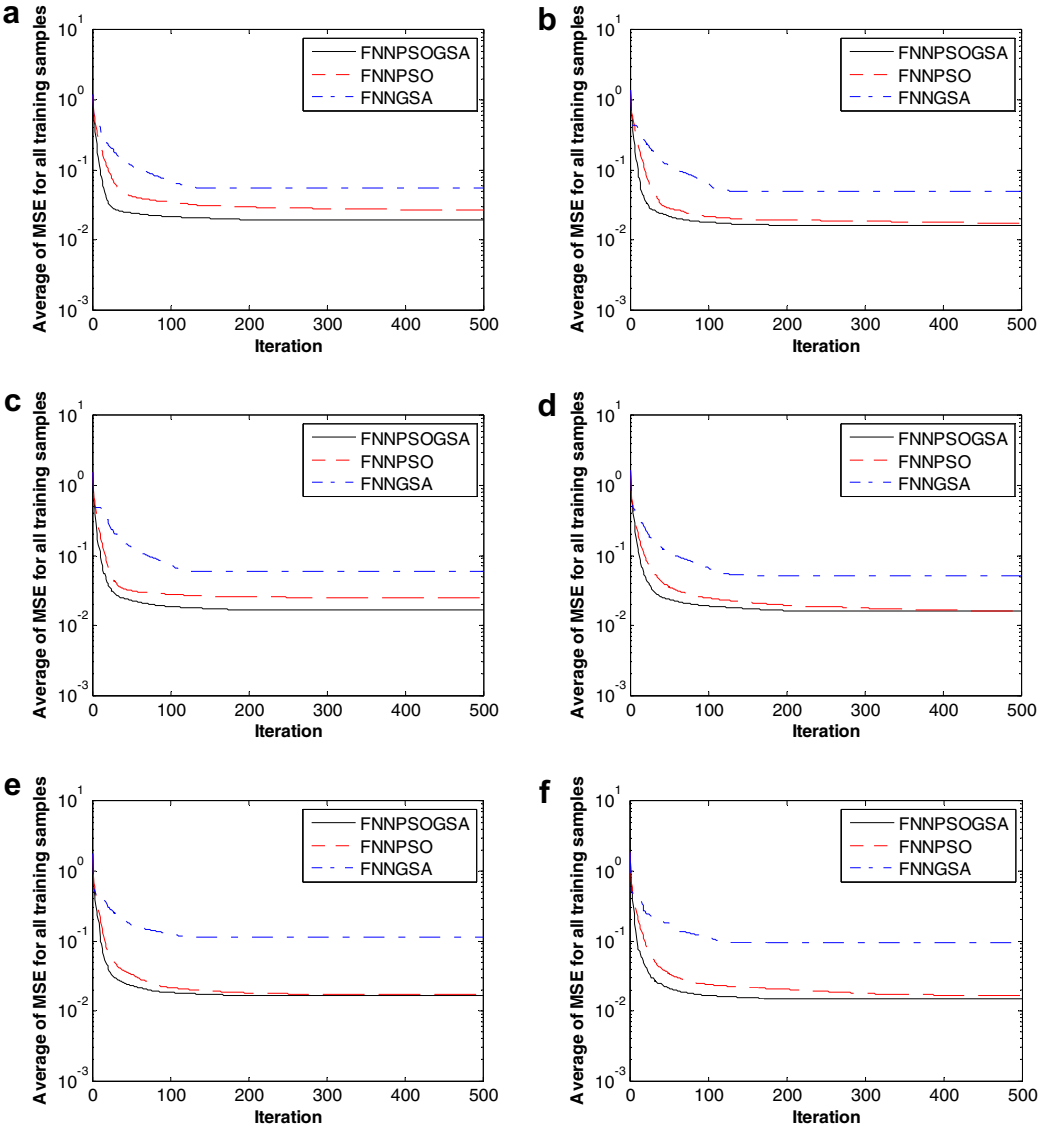

Fig. 8. Convergence curves of FNNPSO, FNNGSA, and FNNPSOGSA based on averages of MSE for all training samples over 30 independent runs in the Iris classification problem. (a), (b), (c), (d), (e), and (f) are the convergence curves for FNNs with $S = 5, 7, 9, 11, 13$, and 15, respectively.

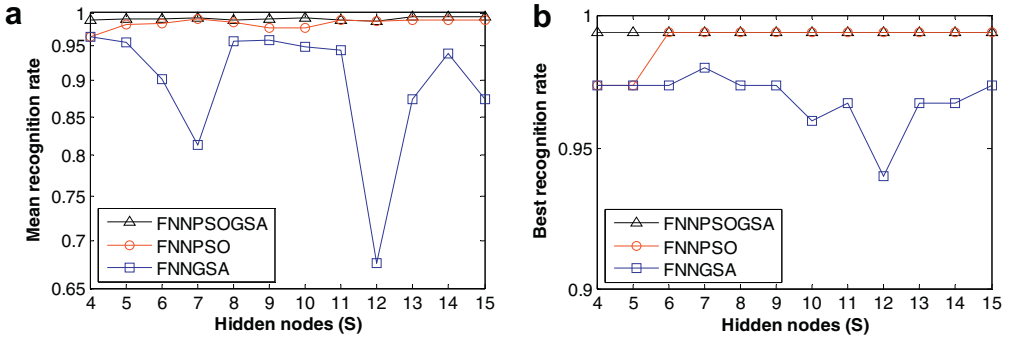

Fig. 9. Correct recognition rate of FNNPSO, FNNGSA, and FNNPSOGSA for the Iris benchmark problem. (a) best recognition rate and (b) average correct recognition rate.

FNNPSO gives the highest accuracy while FNNGSA shows the worst. Therefore, it can be concluded that the proposed FNNPSOGSA improves the problem of trapping in local minima with very good convergence speed. The results for FNNGSA also prove that GSA is not good for training FNNs because of its slow searching speed. In summary, the results prove that FNNPSOGSA boosts the problem of trapping in local minima and enhances the convergence speed compared to the existing learning algorithms for FNNs.

References

[1] B. Irie, S. Miyake, Capability of three-layered perceptrons, in: Proceedings of IEEE International Conference on Neural Networks, San Diego, USA, 1998, pp. 641–648.
[2] C. Lin, Cheng-Hung, C. Lee, A self-adaptive quantum radial basis function network for classification applications, in: IEEE International Joint Conference on Neural Networks, 2004, pp. 3263–3268.
[3] N. Mat Isa, Clustered-hybrid multilayer perceptron network for pattern recognition application, Applied Soft Computing 11 (1) (2011).
[4] K. Homik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Networks 2 (1989) 359–366.
[5] B. Malakooti, Y. Zhou, Approximating polynomial functions by feedforward artificial neural network: capacity analysis and design, Appl. Math. Comput. 90 (1998) 27–52.
[6] D.R. Hush, N.G. Horne, Progress in supervised neural networks, IEEE Signal Process Mag. 10 (1993) 8–39.
[7] M.T. Hagar, M.B. Menhaj, Training feedforward networks with the Marquardt algorithm, IEEE Trans. Network 5 (6) (1994) 989–993.
[8] H. Adeli, S.L. Hung, An adaptive conjugate gradient learning algorithm for efficient training of neural networks, Appl. Math. Comput. 62 (1994) 81–102.
[9] N. Zhang, An online gradient method with momentum for two-layer feedforward neural networks, Appl. Math. Comput. 212 (2009) 488–498.
[10] J.R. Zhang, J. Zhang, T.M. Lock, M.R. Lyu, A hybrid particle swarm optimization–back-propagation algorithm for feedforward neural network training, Appl. Math. Comput. 128 (2007) 1026–1037.
[11] M. Gori, A. Tesi, On the problem of local minima in back-propagation, IEEE Trans. Pattern Anal. Mach. Intell. 14 (1) (1992) 76–86.
[12] S. Shaw, W. Kinsner, Chaotic simulated annealing in multilayer feedforward networks, in: Canadian Conference on Electrical and Computer Engineering, 1996, pp. 265–269.
[13] S.K. Chang, O.A. Mohammed, S. Y Hahn, Detection of magnetic body using article neural network with modified simulated annealing, IEEE Trans. Magn. 30 (1994) 3644–3647.
[14] D.J. Monata, L. Davis, Training feedforward neural networks using genetic algorithms, in: 11th International Joint Conference on Artificial Intelligence, 1989, pp. 762–767.
[15] S. Kiranyaz, T. Ince, A. Yildirim, M. Gabbouj, Evolutionary artificial neural networks by multi-dimensional particle swarm, Neural Networks 22 (10) (2009) 1448–1462.
[16] M. Cells, B. Rylander, Neural network learning using particle swarm optimization, Advances in Information Science and Soft Computing, 2002, pp. 224–226.
[17] C. Zhang, Y. Li, H. Shao, A new evolved artificial neural network and its application, in: 3rd World Congress on intelligent Control and application, Hefei, China, 2000, pp. 1065–1068.
[18] F. Van den Bergli, A.P. Engelbrecht, Cooperative learning in neural network using particle swarm optimization, South African Computer Journal 26 (2000) 84–90.
[19] C. Zhang, H. Shao, Y. Li, Particle swarm optimization for evolving artificial neural network, in: IEEE international Conference on System, Man, and Cybemetics, 2000, pp. 2487–2490.
[20] S. Mirjalili, A. Safa Sadiq, Magnetic optimization algorithm for training multi layer perceptron, in: IEEE International Conference on Industrial and Intelligent Information (ICIII 2011), vol. 2, Indonesia, 2011, pp. 42–46.
[21] T. Si, S. Hazra, N. Jana, Artificial neural network training using differential evolutionary algorithm for classification, in: S. Satapathy, P. Avadhani, A. Abraham (Eds.), Proceedings of the International Conference on Information Systems Design and Intelligent Applications 2012 (INDIA 2012) held in Visakhapatnam, India, January 2012, Springer, Berlin/Heidelberg, AISC 132, pp. 769–778.
[22] M. Settles, B. Rodebaugh, T. Soule, Comparison of genetic algorithm and particle swarm optimizer when evolving a recurrent neural network, Genetic and Evolutionary Computation – GECCO'2003, vol. 2723, Springer, Berlin/Heidelberg, 2003, pp. 148–149.
[23] S. Mirjalili, Hybrid particle swarm optimization and gravitational search algorithm for multilayer perceptron learning, Universiti Teknologi Malaysia (UTM), Johor Bahru, Malaysia, M.Sc. Thesis 2011.
[24] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, vol. 4, 1995, pp. 1942–1948.
[25] E. Rashedi, S. Nezamabadi, S. Saryazdi, GSA: a gravitational search algorithm, Information Sciences 179 (13) (2009) 2232–2248.
[26] I. Newton, In experimental philosophy particular propositions are inferred from the phenomena and afterwards rendered general by induction, 3rd ed.: Andrew Motte's English translation published, 1729, vol. 2.
[27] A.A. Atapour, A. Ghanizadeh, S.M. Shamsuddin, Advances of Soft Computing Methods in Edge Detection, Int. J. Advance. Soft Comput. Appl. 1 (2) (2009) 162–202.
[28] E. Rashedi, H. Nezamabadi-pour, S. Saryazdi, BGSA: binary gravitational search algorithm, Natural Computing 9 (3) (2009) 727–745.
[29] S. Sinaie, Solving shortest path problem using Gravitational Search Algorithm and Neural Networks, Universiti Teknologi Malaysia (UTM), Johor Bahru, Malaysia, M.Sc. Thesis 2010.
[30] S. Mirjalili and S.Z. Mohd Hashim, A New Hybrid PSOGSA Algorithm for Function Optimization, in: International Conference on Computer and Information Application(ICCIA 2010), 2010, pp. 374-377.