Neural Process Lett (2007) 26:217–231
DOI 10.1007/s11063-007-9053-x

An Improved Particle Swarm Optimization for Evolving Feedforward Artificial Neural Networks

Jianbo Yu · Lifeng Xi · Shijin Wang

Published online: 10 October 2007

Abstract This paper presents a new evolutionary artificial neural network (ANN) algorithm named IPSONet that is based on an improved particle swarm optimization (PSO). The improved PSO employs parameter automation strategy, velocity resetting, and crossover and mutations to significantly improve the performance of the original PSO algorithm in global search and fine-tuning of the solutions. IPSONet uses the improved PSO to address the design problem of feedforward ANN. Unlike most previous studies on only using PSO to evolve weights of ANNs, this study puts its emphasis on using the improved PSO to evolve simultaneously structure and weights of ANNs by a specific individual representation and evolutionary scheme. The performance of IPSONet has been evaluated on several benchmarks. The results demonstrate that IPSONet can produce compact ANNs with good generalization ability.

Keywords Artificial neural network · Evolutionary algorithm · Particle swarm optimization · Generalization

1 Introduction

Artificial neural networks (ANNs) have been applied widely in many application domains in recent years. Most applications use feedforward ANNs that use the standard back-propogation (BP) learning algorithm or some improved BP algorithms [1]. The BP algorithm is a gradient-based method, hence some inherent problems are frequently encountered in the use of this algorithm, e.g., very slow convergence speed in training, easily to get stuck in a local minimum, etc. In addition, the BP learning needs to predetermine some important learning parameters such as learning rate, momentum and a predetermined structure. Moreover, the BP algorithm assumes a fixed ANN structure and only trains its weights in the fixed ANN

J. Yu (✉) · L. Xi · S. Wang
School of Mechanical Engineering, Shanghai Jiaotong University, Shanghai 200240, P.R. China
e-mail: yjb168@sjtu.edu.cn

structure. Thus, the problem of designing a near optimal ANN structure for an application remains unsolved.

Evolutionary algorithms (EAs) are non-gradient approaches, and are very promising approaches for training ANNs. Since EAs are heuristic and stochastic based on populations made up of individuals with a specified behavior similar to biological phenomenon, they are robust and efficient at exploring an entire solution space of optimization problems. Thus, EAs are less likely to get stuck in local optima. These characteristics result in the development of evolutionary computation as an increasing important field. EAs have been successfully used to evolve weights, structure, and learning parameters of ANNs in recent years [2–8]. EAs are considered to be capable to reduce the ill effect of the BP algorithm, because they do not require gradient and differentiable information. In general, evolutionary methodologies can be categorized as genetic algorithm (GA), evolution strategies (ES), evolutionary programming (EP), etc. Among existing EAs, the most well-known branch is GA. One attractive feature of GA is its support for generic implementation of some major operators such as crossover, mutation or selection operators. However, GA still suffers from some drawbacks such as competing conventions, premature convergence problem, and lengthy local searches near a local optima.

Recently, a new evolutionary computation technique, the particle swarm optimization (PSO), is proposed by Kennedy and Eberhart [9] as an alternative to GA. The PSO was inspired by insect swarms and has proven to be a competitor to GA when it comes to optimization problems. In comparison with GA, PSO has some attractive characteristics. It retains the good knowledge studied by all particles; whereas GA destroys the previous knowledge of the problems once the population changes. PSO encourages constructive cooperation and information sharing between particles, which enhance the search for a global optimal solution. Successful applications of PSO to some optimization problems such as function minimization [9] and ANN design, have demonstrated its potential. Salerno [10] used a PSO to evolve weights of ANNs for solving the XOR problem and parsing natural language. Juang [11] proposed a hybrid of GA and PSO (HGAPSO) for training recurrent networks. HGAPSO used PSO to enhance the elites generated by GA to generate higher quality individuals. The performance of HGAPSO is compared to both GA and PSO in recurrent network design problems, demonstrating its superiority. Da and Ge [12] proposed an improved PSO-based ANN with simulated annealing (SA) technique for solving a rock engineering problem. Their results showed that SAPSO-based ANN has a better training and generalization performance than PSO-based ANN. Comparisons between PSO and GA for evolving recurrent networks were done analytically by Settles et al. [13]. Their comparisons indicated that the GA is more successful on larger networks and the PSO is more successful on smaller networks. However, in comparison with the wide applications of GA in evolutionary ANNs, the applications of PSO for evolving ANNs are sparse.

These reported work indicated that PSO-based ANN algorithms were successful in evolving ANNs and achieved the generalization performance comparable to or better than those of standard BPNs or GA-based ANNs. However, they used PSO to evolve the weights of ANNs without considering the optimization of structure of the ANNs. Thus, the problem of designing a near optimal ANN structure by only using PSO for an application remains unsolved. However, this is an important issue because the information processing capability of an ANN is determined by its structure.

Although these researches have shown that PSO performs well for global search because it is capable of quickly finding and exploring promising regions in the search space, they take relative inefficiency in fine tuning solutions. Moreover, a potentially dangerous property in PSO still exists: stagnation due to the lack of momentum, which makes it impossible to arrive at the global optimum. To avoid these drawbacks of the basic PSO, some improvements such as the time-varying parameters and random perturbation (e.g., velocity resetting) [14] have been proposed. These improvements can enhance convergence of PSO toward the global optimum, to find the optimum solution efficiently.

Therefore, our approach proposes an improved PSO for the evolutionary design of three-layer feedforward ANNs, and is thus named IPSONet. Unlike some previous work that only used PSO to evolve weights under the condition of fixed-structure ANNs, IPSONet evolves simultaneously the structure and weights of ANNs by using the improved PSO, which could improve the probability to find ANNs with good generalization ability and compact structure. In order to evaluate the performance of IPSONet, it has been tested on some benchmark classification problems.

The rest of this paper is organized as follows: an improved PSO algorithm is developed in Sect. 2. Section 3 describes IPSONet in detail and gives motivations and ideas behind various design schemes. The experimental results on IPSONet and some discussions are presented in Sect. 4. Finally, Sect. 5 provides the conclusion and a few remarks.

2 An Improved Particle Swarm Optimization Algorithm

2.1 Basic PSO

The PSO algorithm [9] is an adaptive algorithm based on a social-psychological metaphor. It has proven to be a useful global optimization algorithm and competitor to the standard GA when it comes to function optimization. The traditional PSO conducts searches using a population of particles that correspond to individuals in GA. Each particle i represents a possible solution and has a position vector x_i, a velocity vector v_i, and the best personal position p_i (i.e., cognitive component) encountered so far by the particle. In each iteration, each particle moves in the direction of its own personal best position p_i, as well as in the direction of the global best position p_g (i.e., social component) discovered so far by any of the particles in the population. As a result, p_i and p_g can be used to adjust their own velocities and positions. This means that if a particle discovers a promising new solution, all other particles will move closer to it, exploring the problem region more roughly in the process. At each time step t, the velocity of a particle i is updated using Eq. 1:

$$v_i(t+1) = wv_i(t) + c_1r_1(p_i(t) - x_i(t)) + c_2r_2(p_g(t) - x_i(t)) \tag{1}$$

where w is the inertia weight and typically setup to vary linearly from 1 to near 0 during the course of an iteration run; c_1 and c_2 are acceleration coefficients; $r_1 \sim U(0,1)$ and $r_2 \sim U(0,1)$ are uniformly distributed random numbers in the range (0,1). The velocity v_i is limited to the range $[v_{\min}, v_{\max}]$. Updating velocity in this way enables the particle i to search around its individual best position p_i, and the global best position p_g. Based on the updated velocities, the new position of the particle i is calculated using

$$x_i(t+1) = x_i(t) + v_i(t+1). \tag{2}$$

Based on (1) and (2), the population of particles tend to cluster together with each particle moving randomly in a random direction.

2.2 Improved PSO

2.2.1 Parameter Automation Strategy

In population-based optimization approaches, proper control of global exploration and local exploitation is crucial in searching the optimum solution efficiently [14,15]. Shi and Eberhart [16] introduced the concept of inertia weight to the original version of PSO, in order to balance the local and global search during the evolutionary process. For EAs, high diversity of individuals is necessary during the early search to allow use of the full range of the search space. In contrast, during the latter part of the search, when the algorithm is converging to the optimal solution, fine-tuning of the solutions is important to find the global optima efficiently. Shi and Eberhart [17] proposed a linearly varying inertia weight (w) over the course of generations, which significantly improves the performance of PSO.

$$w = (w_1 - w_2) \times \frac{MAXITER - iter}{MAXITER} + w_2 \tag{3}$$

where w_1 and w_2 are the initial and final values of the inertia weight, respectively, $iter$ is the current iteration number and $MAXITER$ is the maximum number of allowable iterations. The empirical studies in [17] indicated that the optimal solution can be improved by varying the value of w from 0.9 at the beginning of the evolutionary process to 0.4 at the end of the evolutionary process for most problems.

Although the version of PSO based on the time-varying inertia weight is capable of locating a good solution at a significantly faster rate, its ability to fine tune the optimum solution is comparatively weak, mainly due to the lack of diversity at the end of the evolutionary process [14,15]. It can be observed that, in PSO, the search toward the optimum solution is guided by the two stochastic components (the cognitive component and the social component). Thus, proper control of these two components is effective for finding optimum solution. Ratnaweera et al. [14] proposed a version of PSO based on time-varying acceleration coefficients, which reduces the cognitive component and increases the social component by changing the acceleration coefficients c_1 and c_2 with time. It encourages a large cognitive component and small social component at the beginning of the search to guarantee particles' moving around the search space and to avoid particles' moving toward the population best position. On the other hand, a small cognitive component and a large social component allow the particles to converge to the global optima in the latter of the search. The varying scheme of c_1 and c_2 is given as follows:

$$\begin{aligned} c_1 &= (c_{1f} - c_{1i})\frac{iter}{MAXITER} + c_{1i} \\ c_2 &= (c_{2f} - c_{2i})\frac{iter}{MAXITER} + c_{2i} \end{aligned} \tag{4}$$

where c_{1i}, c_{1f}, c_{2i} and c_{2f} are constants, *iter* is the current iteration number and *MAXITER* is the maximum number of allowable iterations. Through empirical studies, Ratnaweera et al. [14] have observed that the optimal solutions on most of the benchmarks can be improved by changing c_1 from 2.5 to 0.5 and changing c_2 from 0.5 to 2.5 over the full range of the search.

2.2.2 Velocity Resetting

PSO can quickly find a good local solution but it sometimes suffers from stagnation without an improvement [14]. Therefore, to avoid this drawback of basic PSO, the velocity of particles is reset in order to enable particles to have a new momentum. Under this new strategy, when the global best position is not improving with the increasing number of generations, each particle i will be selected by a predefined probability (0.5 in this study) from the population, and then a random perturbation is added to each dimension v_{id} (selected by a predefined probability 0.5 in this study) of the velocity vector v_i of the selected particle i. The velocity resetting is presented as follows: where r_1, r_2 and r_3 are separately generated, uniformly distrib-

```
For ($i$=1 to number of particles)
    If $r\_1 > 0.5$
        Particle $i$ is selected for velocity resetting
        For $d$=1 to number of dimensions of velocity vector $v_i$
            If $r\_2 > 0.5$
              $v_{id} = v_{id} + (2 * r\_3 - 1) * v_{\max}$
            End if
    End if
```

uted random numbers in range (0, 1), and $v_{\max}$ is the maximum magnitude of the random perturbation to each dimension of the selected particle.

2.2.3 Crossover and Mutation

Based on some evolutionary schemes of GA, several effective mutation and crossover operators have been proposed for PSO. Lovbjerg et al. [18] proposed a crossover operator, and Higashi and Iba [19] proposed a Gaussian mutation operator to improve the performance of PSO. Utilizations of these operators in PSO have potential to achieve faster convergence and to find better solutions.

The crossover operator is conducted by the Eq. 5 for position crossover and the Eq. 6 for velocity crossover in terms of a certain crossover rate (α). The position of the offspring is generated for each dimension by arithmetic crossover on the position of two parents selected randomly from the population.

$$\left.\begin{aligned} child_1(x_i) &= r_i * parent_1(x_i) + (1 - r_i) * parent_2(x_i) \\ child_2(x_i) &= r_i * parent_2(x_i) + (1 - r_i) * parent_1(x_i) \end{aligned}\right\} \tag{5}$$

where r_i is a uniformly distributed random value in the range (0,1) and is separately generated for each dimension (x_i) of the selected particles, $parent_1$ and $parent_2$ are the two parents selected randomly from the population, and $child_1$ and $child_2$ are the offspring.

The velocity of the offspring is calculated as the sum of the velocity vectors of the two parents normalized to the original length of each parent velocity vector.

$$\left.\begin{aligned} child_1(v_i) &= \tfrac{parent_1(v_i)+parent_2(v_i)}{|parent_1(v_i)+parent_2(v_i)|}\,|parent_1(v_i)| \\ child_2(v_i) &= \tfrac{parent_1(v_i)+parent_2(v_i)}{|parent_1(v_i)+parent_2(v_i)|}\,|parent_2(v_i)| \end{aligned}\right\} \tag{6}$$

In each generation, the mutation operator is conducted by the Eq. 7 in terms of a certain mutation rate (β), which is realized by Gaussian perturbation with $N(0, \sigma)$.

$$child(x_i) = parent(x_i) + \frac{MAXITER - iter}{MAXITER} * N(0, \sigma) \tag{7}$$

where $parent(x_i)$ is the parent selected randomly from the population, $child(x_i)$ **is** the offspring, $iter$ is the current iteration number and $MAXITER$ is the maximum number of allowable iterations.

In this study, the varying schemes of inertia weight (w) and acceleration coefficients $(c_1$ and $c_2)$, velocity resetting, and crossover and mutation operators have been used in the original version of PSO, and thus the improved PSO is named PVMCPSO. Then, it is used to evolve simultaneously structure and weights of three-layer feedforward ANNs.

3 IPSONet Algorithm

A typical feedforward ANN topology that takes the form of three-layer perceptrons is depicted in Fig. 1(a), where $X(x_1, x_2, \ldots, x_l)$ and $Y(y_1, y_2, \ldots, y_m)$ are the inputs with l elements and outputs with m nodes, respectively. The neurons are connected by weights and output signals. The output values of the nodes in the hidden layer and in the output layer can be formulated as

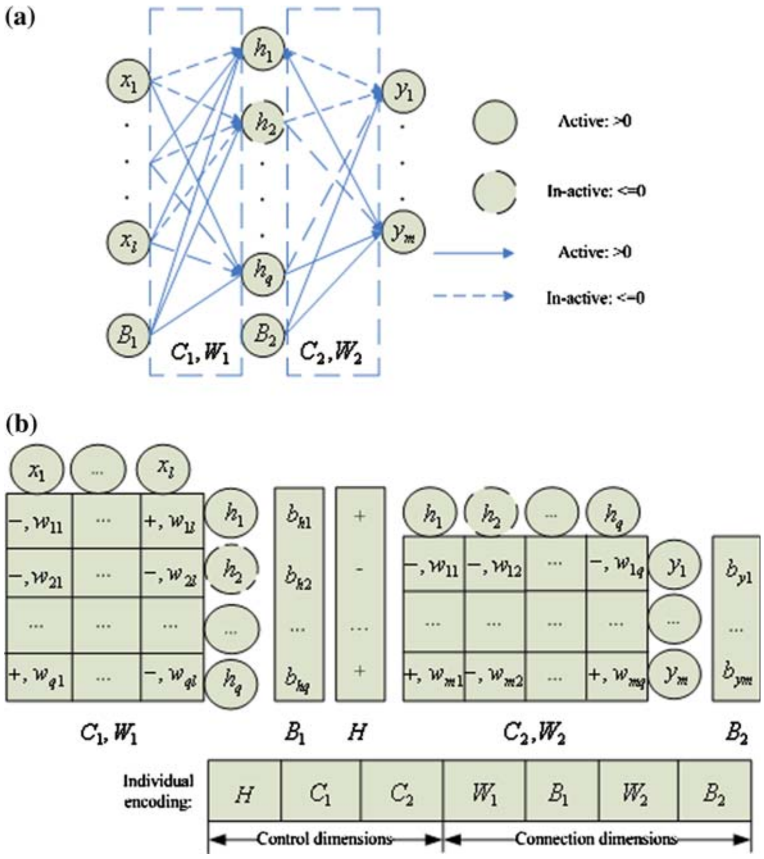

Fig. 1 (**a**) Three-layer feedforward ANN and (**b**) Encoding scheme

$$h_j = f\left(\sum_{i=1}^{l} w_{ji}x_i + b_j\right), \quad 1 \le j \le q \tag{8}$$

and

$$y_k = f\left(\sum_{j=1}^{q} w_{kj}h_j + b_k\right), \quad 1 \le k \le m \tag{9}$$

respectively, where f is the following sigmoid function:

$$f(x) = (1 + \exp(-x))^{-1} \tag{10}$$

where w_{ji} denotes the connection weights between the input nodes and hidden nodes, w_{kj} denotes the connection weights between the hidden nodes and output nodes, b_j and b_k denote the bias of the hidden node and the outputs node, respectively, and q is the number of hidden nodes in the hidden-layer.

The detailed design algorithm of three-layer feedforward ANNs by IPSONet is described as follows:

Coding scheme: When constructing an ANN, the core problem lies upon the optimization procedures to obtain an optimal ANN structure. In this study, IPSONet evolves simultaneously the structure and weights of ANNs and needs information about each connection in the ANN. Thus, particles are represented by its hierarchy structure, in which each particle consists of multilevel dimensions. Figure 1(b) shows the individual representation in the trained ANN. Each particle consists of two types of dimensions, i.e., the control dimensions in the form of real numbers, are the hidden neurons and connections for activation, and the connection dimensions, a real-value representation, are the values for connection weightings and neuron bias. Thus, there are three vectors H, B_1 and B_2, and four matrices C_1, C_2, W_1 and W_2, which are used to specify an ANN in IPSONet. The dimensions of the vector H are equal to the maximum number of hidden neurons allowable in the ANN. In the vector H, the value of an element that is smaller than or equal to 0 denotes inactive state (i.e. the hidden node does not exist) and while the value of an element that is greater than 0 denotes active state (i.e. the hidden node exists). The connections from the input layer to the hidden layer are encoded in the connection matrix C_1 while the connection matrix C_2 contains the connections from the hidden layer to the output layer, whose entries take a form of real numbers (i.e. the value of an element in the matrix that is smaller than or equal to 0 denotes the inactivation of a connection, while the value of an element in the matrix that is larger than 0 denotes the activation of a connection). It should be noted that from Fig. 1(b) "–" represents that the value of an element in control dimensions is smaller than or equal to 0, while "+" represents that the value of an element in control dimensions is larger than 0. The other two matrices W_1 and W_2 are weight matrices corresponding to the connection matrices C_1 and C_2, whose entries are real numbers. The vectors B_1 and B_2 represent the bias of the hidden neurons and output neurons, respectively. Within such a specific treatment, structure and weights of an ANN evolution can be implemented easily by PVMCPSO.

Fitness function: The fitness function, which measures the performance of individuals, is defined as:

$$fitness(g) = \frac{1}{mn}\sum_{j=1}^{m}\sum_{i=1}^{n}(T_i - O_i)^2 \tag{11}$$

where T_i is the expected output, O_i is the calculated output of individual network g, n is the number of training set examples, and m is the number of output nodes. Thus, $fitness(g)$ is the normalized mean squared error (MSE) of the individual g on the training set.
Population initialization: The algorithm begins with the random generation of a number of networks bigger than the number of networks used during the evolutionary process. We randomly generate $10M$ networks, and then select the best M among them based on the fitness function (11).
Evolution by PVMCPSO along with velocity resetting: Evolving the population by using PVMCPSO until the global best position P_g cannot be improved for some successive generations predetermined by users, and then the velocities of the particles are reset in order to enable particles to have a new momentum (i.e., new velocities) for finding better networks (discussed in Sect. 2).
Stopping scheme: The stop criterion is reached whenever one of the following two conditions is fulfilled: (1) a given fitness value is achieved; (2) the total number of generations specified by users is reached.
Selection of the best individual network: Once the stop criterion is reached, IPSONet uses a validation set to select the best individual ANN (with the lowest classification error rate) from all the personal best individuals $P_i(i = 1, 2, \ldots M$, where M is the number of individuals in the population) in the last generation, and then uses the testing set to evaluate the performance of the best ANN.

4 Experimental Studies

To evaluate the performance of IPSONet, several experiments were conducted on 8 data sets listed in Table 1. All data sets come from the UCI machine learning benchmark repository [20]. These problems have been the subjects of many studies in ANNs and machine learning. Experimental details, results and comparisons with other work are presented in the following sub-sections.

4.1 Experimental Setup

In this study, all data sets are. partitioned into three sets: a training set, a validation set and a testing set. The validation set is used to select the best one (with the lowest classification error rate) from the personal best individuals of PVMCPSO, while the testing set is used to test the generalization performance of the best ANN and is not seen by any individual ANNs during the whole training and validating process. It is known that the experimental results may vary significantly for different partitions of the same data set [21]. It is necessary to know precise

Table 1 Description of data sets used in the experiments

Data set	Attributes	Classes	Instances
Breast cancer	10	2	699
Credit card (A)	14	2	690
Diabetes	8	2	768
German	20	2	1000
Heart	13	2	270
Iris	4	3	150
Pima	8	2	768
Vehicle	18	4	846

specification of the partition in order to reproduce an experiment or conduct fair comparisons with other work. In the experiment, we firstly analyze the evolutionary process of IPSONet and evaluate the performance of IPSONet on the breast cancer, diabetes and heart data sets. To do so, we partitioned the three data sets as follows:

- For the breast cancer data set, the first 349 examples were used for the training set, the following 175 examples for the validation set, and the rest (i.e., 175 examples) for the testing set.
- For the diabetes data set, the first 384 examples were used for the training set, the following 192 examples for the validation set, and the rest (i.e., 192 examples) for the testing set.
- For the heart data set, the first 134 examples were used for the training set, the following 68 examples for the validation set, and the rest (i.e., 68 examples) for the testing set.

The input attributes of all the data sets were rescaled to between 0.0 and 1.0 by a linear function. The outputs were encoded by the 1-of-c representation for c classes. The winner-takes-all method was used in the proposed approach, i.e., the output with the highest activation designates the class.

There are some parameters in IPSONet which need to be specified by the user. It is, however, unnecessary to tune all these parameters for each data set because IPSONet is not very sensitive to them. Therefore, these parameters were set to the same for all these problems: the population size $M(50)$, the number of generations (1,500), the initial position and velocity range of particles ($[-1, +1]$), the inertia weights $w_{\max}$ and $w_{\min}$ of PVMCPSO (0.9 and 0.4), the initial acceleration coefficients c_1 and c_2 of PVMCPSO (2.5 and 0.5), and the global best position P_g in PVMCPSO is not improved for successive generations (50) to conduct the velocity resetting, the crossover rate α (0.7) and the mutation rate β (0.1), and the standard deviation σ (0.7) of Gaussian mutation in PVMCPSO. The used ANNs in the population are three-layer feedforward ANNs with the maximum number of hidden nodes (7). It should be noted that we used full connections (i.e., all hidden nodes and connections are active in the initial particles) in the individual networks of the initial population.

4.2 Experimental Results

Table 2 presents the results of IPSONet over 30 independent runs on the breast cancer, diabetes and heart data sets (the Mean, SD, Max and Min indicate the mean value, standard deviation, maximum value and minimum value, respectively). The error rate in the table refers to the percentage of wrong classification produced by the trained ANNs on the training, validation and testing set. We report the results in terms of average error rates over 30 independent runs on the three data sets. It should be noted that IPSONet does not consider any structural optimization in the fitness function to construct a compact ANN, but only evolves the structure by using PVMCPSO based on the specific individual representation scheme. The results demonstrate that IPSONet has the capability to evolve compact ANNs which generalize well.

To observe the evolutionary process in IPSONet, Figs. 2–4 show the evolution of mean of average numbers of connections and the mean of average classification error rate of ANNs over 30 runs for the three problems. It should be noted that these results are calculated based on the personal best positions (i.e., $P_i (i = 1, 2, \ldots M)$, where M is the number of individuals in the population), namely, the classification error rates of all $P_i (i = 1, 2, \ldots M)$ on the training set are calculated to obtain the average classification error rate of the population in each generation. The evolutionary processes are quite interesting. It can be observed that the number of connections in ANNs decreases rapidly in the beginning of the evolution. After

Table 2 Results of IPSONet for the breast cancer, diabetes and heart data sets

Data set		Training set	Validation set	Testing set	Number of hidden nodes	Number of connections
		Error rate	Error rate	Error rate		
Breast cancer	Mean	0.0181	0.0286	0.0127	4.7	39.7
	SD	0.0060	0.0047	0.0057	1.0	10.7
	Min	0.0086	0.0229	0.0057	4	23
	Max	0.0287	0.0343	0.0286	7	50
Diabetes	Mean	0.2217	0.1946	0.2102	4.9	40.5
	SD	0.0139	0.0142	0.0123	1.2	11.1
	Min	0.1875	0.1667	0.1875	2	17
	Max	0.2448	0.2240	0.2292	7	63
Heart	Mean	0.0582	0.1945	0.1814	5.0	56.6
	SD	0.0123	0.0199	0.0342	1.1	12.5
	Min	0.0448	0.1471	0.1176	3	33
	Max	0.0821	0.2353	0.2500	7	96

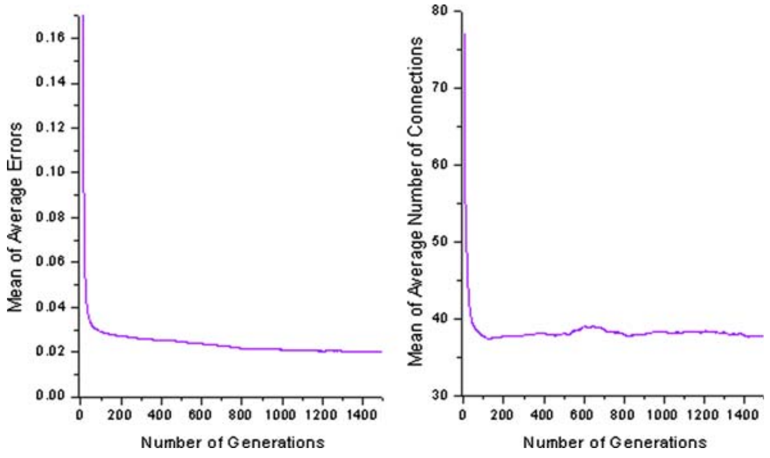

Fig. 2 Evolution of ANN connections and error rates for the breast cancer data set

certain number of generations, the number of connections starts decreasing slowly. During the latter part of the evolution, the number of connections keeps stability. Meanwhile, it can be observed that IPSONet can reduce the error of the trained ANNs quickly during the early part of the evolution. Moreover, IPSONet is capable of continuously reducing the error of the ANNs on the training set during the latter part of the evolution. This demonstrates that PVMCPSO can converge quickly toward the global optima, and fine tune the solutions very efficiently. The phenomenon illustrates the effectiveness of IPSONet in simultaneously evolving the weights and structure of individual networks through using the specific individual representation and evolutionary scheme.

In the beginning stage of the evolution, IPSONet deletes many connections in the individual network to reduce its error on the training data set. After certain number of generations, networks in the population will have fewer connections and lower errors than before. They have reached such a level that further large deletion of connections will increase their errors.

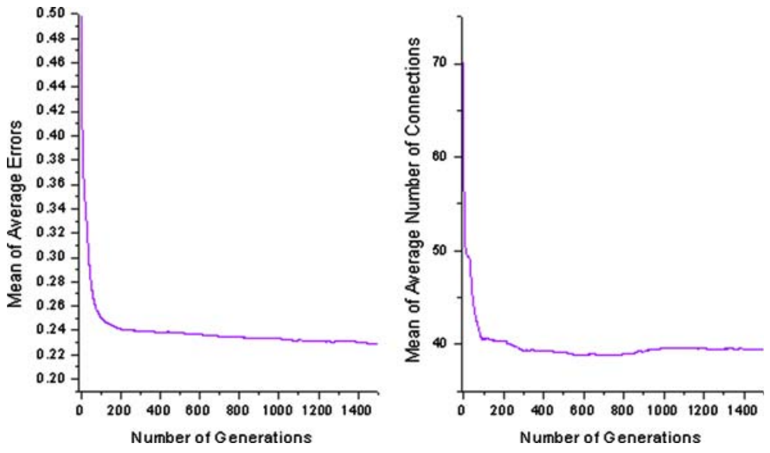

Fig. 3 Evolution of ANN connections and error rates for the diabetes data set

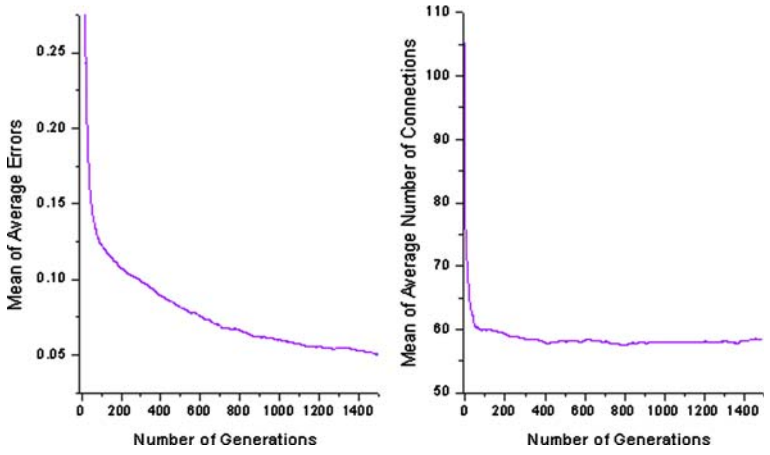

Fig. 4 Evolution of ANN connections and error rate for the heart data set

Hence large deletion is likely to fail, and small addition or deletion is likely to be attempted in the latter stage of the evolution. Based on the relative stable structure of the individual network, further fine tune the weights of the network can reduce continuously its error during the latter part of the evolution. Therefore, these results demonstrate that IPSONet is capable of evolving compact ANNs with good training results.

Table 3 compares the results of IPSONet with those of EPNet proposed by Yao and Liu [4]. Due to IPSONet and EPNet used the same training set, validation set and testing set for the three problems, it is appropriate to compare the performance of IPSONet with that of EPNet. The results of IPSONet and EPNet are the average testing error rates over 30 independent runs on the three data sets. This comparison shows that IPSONet achieves the generalization performance better than EPNet for both the breast cancer and diabetes data sets. It should be noted that EPNet spends much additional computation time to evolve ANNs due to its complicated computation and large search. In comparison with EPNet, IPSONet uses more simple evolutionary method and spends less computation cost through using PVMCPSO.

Table 3 Comparison between IPSONet and EPNet in terms of the average testing error rate for the breast cancer, diabetes and heart data sets

Data set	IPSONet	EPNet
Breast cancer	0.0127	0.0137
Diabetes	0.2102	0.2238
Heart	0.1814	0.1677

Table 4 Comparison with other algorithms in terms of the average testing accuracy rate for all data sets

Data set	NB	BN	TAN	CL	C4.5	SNB	IPSONet
Card	86.23 ± 1.10	86.23 ± 1.76	84.20 ± 1.24	85.07 ± 1.31	85.65 ± 1.82	86.67 ± 1.81	86.83 ± 2.77
Cancer	97.36 ± 0.50	96.92 ± 0.63	96.92 ± 0.67	92.40 ± 0.81	94.73 ± 0.59	96.19 ± 0.63	97.07 ± 0.51
Diabetes	74.48 ± 0.89	75.39 ± 0.29	75.52 ± 1.11	74.74 ± 1.19	76.04 ± 0.85	76.04 ± 0.83	77.63 ± 0.78
German	74.70 ± 1.33	72.30 ± 1.57	73.10 ± 1.54	73.90 ± 1.85	72.20 ± 1.23	73.70 ± 2.02	75.52 ± 3.63
Heart	81.48 ± 3.26	82.22 ± 2.46	83.33 ± 2.48	82.22 ± 2.96	81.11 ± 3.77	81.85 ± 2.83	82.62 ± 5.25
Iris	93.33 ± 1.05	94.00 ± 1.25	94.00 ± 1.25	93.33 ± 1.05	94.00 ± 1.25	94.00 ± 1.25	96.00 ± 3.51
Pima	75.51 ± 1.63	75.00 ± 1.22	75.52 ± 1.27	75.39 ± 1.51	75.13 ± 1.52	74.86 ± 2.61	76.68 ± 2.09
Vehicle	58.28 ± 1.79	61.00 ± 2.02	69.63 ± 2.11	67.15 ± 2.06	69.74 ± 1.52	61.36 ± 2.33	70.02 ± 5.34

4.3 Comparison with Other Work

In order to further evaluate the performance of IPSONet under cross-validation method, the following comparisons were carried out by using 5-fold cross-validation [22] on all data sets, and other cross-validation on the Australia credit card and diabetes data sets. In order to conduct a fair comparison with other work, the training set was used as the validation set in IPSONet for the selection of the best individual network from the personal best positions of the population.

4.3.1 Comparison with Other Work using 5-fold Cross-validation

In order to compare IPSONet with previous work, the experimental setup that is the same as the previous experimental setup described in [23], namely, 5-fold cross-validation was used for all data sets in this study. Table 4 compares results of IPSONet against those of five Bayesian network classifiers and C4.5 [24] tested by Friedman [23]. The accuracy rate in the table refers to the percentage of correct classifications produced by the trained ANNs on the testing set. It is clear that IPSONet outperforms other algorithms on major data sets.

4.3.2 Comparison with Other Work using Other Cross-validation

This section compared the results of IPSONet with those of other learning algorithms. The experimental setup is the same as the previous experimental setup described in [25]. Namely, the 10-fold cross-validation [26] was used for the Australian credit card data set, and 12-fold cross-validation was used for the diabetes data set, respectively.

Table 5 Comparisons among IPSONet, EENCL and MPANN in terms of the average testing error rate for the Australian credit card data set and the diabetes data set

Data set	IPSONet	EENCL	MPANN
Credit card	0.132	0.135, 0.132	0.136
Diabetes	0.230	0.221, 0.223	0.251

Table 5 compares the results (i.e., average testing error rates) of IPSONet against those of other evolutionary ANN algorithms proposed by Liu et al. [27] and Abbass [28]. Liu et al. [27] proposed an evolutionary ensemble algorithm with negative correlation learning (EENCL) to address the issues of automatic determination of the number of individual ANNs in an ensemble and the exploitation of the interaction between individual ANN design and combination. Their results indicated that EENCL showed good generalization performance in the Australian credit card and diabetes data set. In general, ensemble ANN can obtain better generalization performance than those of single ANNs. It should be noted that results of EENCL have two error rates which are obtained from two combination approaches, namely, ensemble all individuals in the population and select representation from the population, respectively. For the ensemble using the whole population and the winner-takes-all combination method, the average error rates of EENCL are respectively 0.135 and 0.221 for the Australian credit card data set and the diabetes data set, respectively. For the ensemble using the representatives from species, the average testing error rates of EENCL are 0.132 and 0.223 for the Australian credit card data set and the diabetes data set, respectively. The reason for choosing EENCL algorithm is to show good generalization performance of IPSONet when compared with an evolutionary ANN ensemble algorithm. Abbass [28] presented an evolutionary ANN approach (MPANN) based on pareto multi-objective optimization and differential evolution augmented with local search. From Table 5, it is evident that IPSONet have been able to achieve the generalization performance comparable to EENCL and better than the MPANN. It should be noted that EENCL used the evolutionary ensemble approach, which helps EENCL obtain better generalization performance in comparison with using the best individual network in the population. In contrast, IPSONet used the best individual network to test its performance.

Tables 6 and 7 compare the results (i.e., average testing error rates) of IPSONet against those of 23 algorithms tested by Michie et al. [25]. These algorithms can be categorized into

Table 6 Comparison with other algorithms in terms of the average testing error rate for the Australian credit card data set

Algorithm	Error rate	Algorithm	Error rate
IPSONet	0.132	Baytree	0.171
Discrim	0.141	NaiveBay	0.151
Quadisc	0.207	CN2	0.204
Logdisc	0.141	C4.5	0.155
SMART	0.158	ITrule	0.137
ALLOC80	0.201	Cal5	0.131
k-NN	0.181	Kohonen	Fail
CASTLE	0.148	DIPOL92	0.141
CART	0.145	Backprop	0.154
IndCART	0.152	RBF	0.145
NewID	0.181	LVQ	0.197
AC^2	0.181	Default	0.440

Table 7 Comparison with other algorithms in terms of the average testing error rate for the diabetes data set

Algorithm	Error rate	Algorithm	Error rate
IPSONet	0.230	Baytree	0.271
Discrim	0.225	NaiveBay	0.262
Quadisc	0.262	CN2	0.289
Logdisc	0.223	C4.5	0.270
SMART	0.232	ITrule	0.245
ALLOC80	0.301	Cal5	0.250
k-NN	0.324	Kohonen	0.273
CASTLE	0.258	DIPOL92	0.224
CART	0.255	Backprop	0.248
IndCART	0.271	RBF	0.243
NewID	0.289	LVQ	0.272
AC^2	0.276		

statistical algorithms (Discrim, Quadisc, Logdisc, SMART, ALLOC80, k-NN, CASTLE, NaiveBay, and Default), decision trees (CART, IndCART, NewID, AC^2, Baytree, Cal5, and C4.5), rule-based methods (CN2, and ITrule), and ANNs (Backprob, Kohonen, LVQ, RBF, and DIPOL92). IPSONet used the same experimental setup as in [25] (i.e., 10-fold cross-validation for the Australian credit card data set and 12-fold cross-validation for the diabetes data set). The comparisons indicate that IPSONet achieves the generalization performance comparable to or better than the best of 23 algorithms for both the Australian credit card and diabetes data sets.

5 Conclusions

In this paper, a new evolutionary ANN algorithm, i.e., IPSONet, is proposed. IPSONet is based on an improved PSO algorithm PVMCPSO for evolving three-layer feedforward ANNs. PVMCPSO enables ANNs to dynamically evolve its structure and adapt its weights by using a specific individual representation and evolutionary scheme. To improve efficiency of PSO in global search and fine-tuning of the solutions, parameter automation strategy, velocity resetting, and crossover and mutations are used in PVMCPSO. The performance of IPSONet has been evaluated on seven benchmark problems. Very competitive results have been produced by IPSONet in comparison with other algorithms. Moreover, the results demonstrate that IPSONet can evolve compact ANNs by only using PVMCPSO without considering any structural information of ANNs in the fitness function. One of the future improvements to IPSONet would be to form the ensemble output from combination of individual ANN outputs. Another future improvement would be to further improve the performance of PVMCPSO by combing other EAs such as ES and EP.

Acknowledgments This work was supported by the National Science Foundation of China under Grant 50675137, and by the Program of Introducing Talents of Discipline to Universities Grant B06012. The authors would like to express sincere appreciation to the anonymous referees for their detailed and helpful comments to improve the quality of the paper.

References

1. Hush DR, Horne NG (1993) Progress in supervised neural networks. IEEE Signal Process Mag 10:8–39
2. Angeline PJ, Saunders GM, Pollack JB (1994) An evolutionary algorithm that constructs recurrent neural networks. IEEE Trans Neural Netw 5(1):54–65

3. Maniezzo V (1994) Genetic evolution of the topology and weight distribution of neural networks. IEEE Trans Neural Netw 5(1):39–53
4. Yao X, Liu Y (1997) A new evolutionary system for evolving artificial neural networks. IEEE Trans Neural Netw 8(3):694–713
5. Yao X (1999) Evolving artificial neural networks. Proc IEEE 87(9):1423–1447
6. Schindler KH, Fischer MM (2000) An incremental algorithm for parallel training of the size and the weights in a feedforward neural network. Neural Process Lett 11:131–138
7. Castillo PA, Carpio J, Merelo JJ, Prieto A, Rivas V (2000) Evolving multilayer perceptrons. Neural Process Lett 12:115–127
8. Yang JM, Kao CY (2001) A robust evolutionary algorithm for training neural networks. Neural Comput Appl 10:214–230
9. Kennedy J, Eberhart RC (1995) Particle swarm optimization. IEEE IntConf. Neural Networks. Piscataway, pp 1942–1948
10. Salerno J (1997) Using the particle swarm optimization technique to train a recurrent neural model. 9th International Conference on Tools With Artificial Intelligence (ICTAI97). IEEE Press, pp 45–49
11. Juang CF (2004) A hybrid genetic algorithm and particle swarm optimization for recurrent network design. IEEE Trans Syst Man Cybern 32:997–1006
12. Da Y, Ge XR (2005) An improved PSO–based ANN with simulated annealing technique. Neurocomput Lett 63:527–533
13. Settles M, Rodebaugh B, Soule T (2003) Comparison of genetic algorithm and particle swarm optimizer when evolving a recurrent neural network. In: Cantú–Paz E et al. (eds), Genetic and Evolutionary Computation————GECCO–2003. Chicago, 12–16 July, Springer–Verlag. 2723 pp 148–149
14. Ratnaweera A, Saman K, Watson HC (2004) Self–organizing hierarchical particle swarm optimizer with time–varing acceleration coefficients. IEEE Trans Evol Comput 8(3):240–255
15. Angeline PJ (1998) Evolutionary optimization verses particle swarm optimization: philosophy and performance difference. In: Lecture notes Computer Science, vol. 1447, Proc. 7th Int. Conf. Evolutionary Programming–Evolutionary Programming VII, Mar. 1998, pp 600–610
16. Shi Y, Eberhart RC (1998) A modified particle swarm optimizer. In: Proc. IEEE Int. Conf. Evolutionary Computation. pp 69–73
17. Shi Y, Eberhart RC (1999) Empirical study of particle swarm Opimization. In: Proc. IEEE Int. Conf. Evolutionary Computation 3 pp 101–106
18. Lovbjerg M, Rasmussen TK, Krink T (2001) Hybrid particle swarm optimiser with breeding and subpopulations. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO). San Francisco, CA, July 2001
19. Higashi N, Iba H (2003) Particle swarm optimization with Gaussian mutation. In: Proc. of the IEEE Swarm Intelligence Symp. Indianapolis: IEEE Inc pp 72–79
20. Murphy PM, Aha DW (1994) UCI repository of machine learning databases. Dept. Inf. Comput. Sci., Univ. California, Irvine, CA
21. Prechelt L (1994) Proben1—A set of benchmarks and benchmarking rules for neural network training algorithms. Univ. Karlsruhe, Karlsruhe, Germany
22. Kohavi R (1995) A study of cross–validation and bootstrap for accuracy estimation and model selection. In: proceeding of the fourteenth international joint conference on artificial intelligence. Morgan Kaufmann, San Francisco, CA, pp 1137–1143
23. Friedman N (1997) Bayesian network classifiers. Mach Learn 29:131–163
24. Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann, San Francisco
25. Michie D, Spiegelhalter DJ, Taylor CC (1994) Machine learning, neural and statistical classification. Ellis Horwood Limited, London, UK
26. Stone M (1974) Cross–validation choice and assessment of statistical predictions. J Royal Stat Soc 36:111–147
27. Liu Y, Yao X, Tetsuya HC (2000) Evolutionary ensemble with negative correlation learning. IEEE Transaction on Evolutionary Computation 4(4):380–387
28. Abbass HA (2001) A memetic pareto evolutionary approach to artificial neural networks. Proceedings of the 14th Australian Joint Conference on Artificial Intelligence: Advances in Artificial Intelligence. Lect Notes Comput Sci 2256:1–12