http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 7 (4): 639-646, 2008
ISSN 1812-5638

Research on Structure Learning of Product Unit Neural Networks by Particle Swarm Optimization

[1]Xian-Hui Wang, [1,2]Zheng Qin, [1]Xing-Chen Heng and [2]Yu Liu
[1]Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, China
[2]School of Information Science and Technology, Tsinghua University, Beijing, China

Abstract: In this study, we put forward a new method to learn structure of Product Unit Neutral Network (PUNN). The technique used in our research is based on Particle Swarm Optimization (PSO) algorithm. The technique can optimized collocate network structure and weight of the PUNN at the same time using PSO algorithm through standard data set. Moreover, the number of Hidden Layer units of PUNN is decided by training set, not prefixed by the designer's prior knowledge. Particles encoding scheme is simple and effective. The design of fitness function considers not only the mean square error between networks output and desired output, but also the number of hidden layer units. Therefore, the resulting network can alleviate the problem of over-fitting. The results of the experiment indicate that PSPUNN algorithm can achieve rational architecture for PUNN relying on standard data set and the resulting networks hence obtain strong generalization abilities.

Key words: Product unit neural network, particle swarm optimization, structure learning, cascade correlation

INTRODUCTION

It is often needs the designer prefixes structure of network according to his prior knowledge before using neural network to solve practical issues. For the designer, it is extremely tough because the optimal network structure can be set solely relying on the designer's deep understanding of the problem. The structure of neural network embodies its degree of complexity, on the one hand, over simple network is hard to fully fitting training data dues to its limited flexibility; on the other hand, over complex network products weak generalization ability due to its feature of over flexibility, which fittings training the noises of data. Therefore, how to learn and optimize the structure of neural network according to training data set has been on the international hotspot. The methods which learn and optimize structure of neural network include: growth, pruning, evolutionary algorithm etc. Fahlman and Lebiere (1990) presented famous cascade correlation algorithm (CC). It is a supervision-training algorithm to construct close to the smallest network in the training process.

Neurons receive process signals from other ones and yield output signals and then send the output signals to other neurons again. General neurons which also called Summation Units (SUs) sum input signals; while Product Units (PUs) make product for input signals. For input signals, product units can form high-level combination and increase information capability. Product Unit Neural Networks (PUNN) (Durbin and Rumelhart, 1990; Schmitt, 2001; Ismail and Engelbrecht, 2002) has the virtue of strong ability of information storage which denotes that it can use simple network structure to render complex information. Compared to Summation Unit Neural Networks (SUNN), such as BP network, its network has the advantage of simple network topology, scarce amount of connective weight; at the meantime, it shows superior performance in the application fields such as pattern classification etc. (Fischer, 2002; Martinez-Estudillo *et al*., 2006).

Learning algorithm of PUNN is rather complicated relative to SUNN. Leerink *et al*. (1995) studied PUNN training algorithm widely. When using BP algorithm to train PUNN, on account of local minimum, the training would encounter many obstacles. Different researchers adopt different algorithm to train PUNN: Janson and Frenzel (1993) suggested that using Genetic Algorithm (GA), Ismail and Engelbrecht (2000) trained it through testing various global optimization algorithm, for instance, Particle Swarm Optimization (PSO) algorithm, Genetic Algorithm (GA), Van Den Bergh and Engelbrecht (2001) proposed Cooperative Particle Swarm Optimization (CPSO) algorithm. Among those different types of global optimization algorithm, PSO algorithm has relatively better

Corresponding Author: Xian-Hui Wang, Department of Computer Science and Technology, Xi'an Jiaotong University, P.O. Box 2287, 710049, Xi'an, Shaanxi, People's Republic of China

effect in training PUNN (Engelbrecht and Ismail, 1999). Therefore, in this study, we depend on the PSO algorithm to learn and optimize PUNN structure, concurrently train PUNN.

PRODUCT UNIT NEURAL NETWORKS

Durbin and Rumelhart (1990) firstly use Product Unit (PU) to extend multilayer perceptron. Product Unit is different from common neuron which is called Summation Unit (SU). All the neurons receive input signals from other neurons and make output signals, then send the output signals to other neurons. A PU combines input signals according to bellowing formula: $\prod_{i=1}^{N} x_i^{w_i}$; while the form of SU is $\sum_{i=1}^{N} w_i x_i$. Here, N represents the total amount of neurons connect to it; x_i is the ith input signal connected neuron and w_i is the ith weight vector connected neuron.

All layers of Product Unit Neural Networks (PUNN) can be constructed by PU or alternation between PU and SU. A PUNN may have several hidden layers. A common PUNN network's topology is a three-layer feed-forward network. Figure 1 is a typical a three-layer feed-forward PUNN network, hidden layer consists of PU, output layer consists of SU.

Suppose PUN's input layer has D nodes and hidden layer has M nodes, while output layer has C nodes; suppose the connective weight vectors between input and output layers is u, the connective weight vectors between nodes of hidden and output layer is w; if the p-th input pattern applied to the input nodes, the k-th output node will output $y_{k,p}$ (the value with a bias θ_k). We can use the bellowing formula to calculate $y_{k,p}$:

$$y_{k,p} = \sum_{j=1}^{M} w_{kj} \prod_{i=1}^{D} x_{i,p}^{u_{ji}} + \theta_k, \quad k = 1,\ldots, C. \tag{1}$$

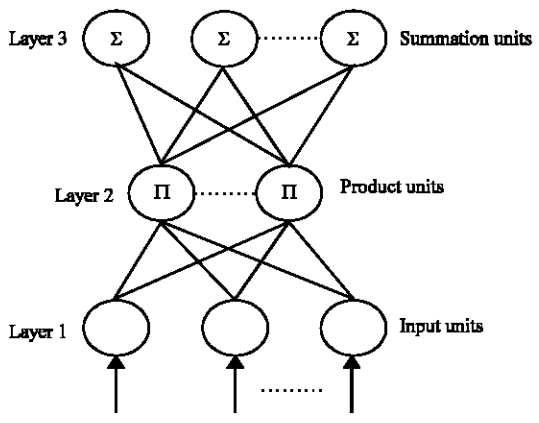

Fig. 1: The architecture of product unit neural networks

PARTICLE SWARM OPTIMIZATION

Particle Swarm Optimization (PSO), a new population-based evolutionary computation technique inspired by social behavior simulation, was first introduced in 1995 by Kennedy and Eberhart (1995). Since it is simple and effective, it has been successfully applied to many fields such as non-linear optimization, neural network and pattern recognition. PSO is a population based iterative search algorithm. A swarm consists of N particles moving around in a D-dimensional search space. Each particle adjusts its flying to a promising area according to its own flying experience and sharing social information among particles. The position of the i-th particle at the iteration is represented by $X_i^{(t)} = (x_{i1}, x_{i2},\ldots,x_{iD})$ that are used to evaluate the quality of the particle. During the searching process the particle successively adjusts its position to the global optimum according to two factors: the best position encountered by itself (pbest) denoted as $P_i = (p_{i1}, p_{i2},\ldots,p_{iD})$ and the best position encountered by the whole swarm (gbest) denoted as $P_g = (p_{g1}, p_{g2},\ldots,p_{gD})$. Its velocity at t iteration is represented by $V_i^{(t)} = (v_{i1}, v_{i2},\ldots,v_{iD})$. The position at next iteration is calculated according to the following equations:

$$V_i^{(t+1)} = \lambda(w * V_i^{(t)} + C_1 * rand_1() * (P_i - X_i^{(t)}) + C_2 * rand_2() * (P_g - X_i^{(t)})) \tag{2}$$

$$X_i^{(t+1)} = X_i^{(t)} + V_i^{(t+1)} \tag{3}$$

where, c_1 and c_2 are two positive constants, called cognitive learning rate and social learning rate, respectively; rand() is a random function in the range [0,1]; w is inertia factor (Shi and Eberhart, 1998) and λ is constriction factor (Clerc, 1999). In addition, the velocities of the particles are confined within $[V_{min}, V_{max}]^D$. If an element of velocities exceeds threshold Vmin or Vmax, it is set equal to corresponding threshold. The pseudo code of PSO is as follows:

Begin PSO

- Initialize();
- for t =1 to the limit of iterations
  for i = 1:N
  $Fitness_i^{(t)}$ = Evaluation Fitness ($X_i^{(t)}$);
  Update Velocity($V_i^{(t+1)}$) according to formula(2);
  Limit Velocity($V_i^{(t+1)}$);
  Update Position($X_i^{(t+1)}$) according to formula(3);
  if needed, update P_i and P_g;
  Endfor
- Terminate if P_g meets problem requirements;
  Endfor

End PSO

PSPUNN ALGORITHM

It is proved by Hornik *et al.* (1989) that any measurable function can be approximate to any desired degree of accuracy by three-layer network with only one hidden layer, provided sufficient hidden units are available. Therefore, PUNN's structure adopt three-layer feed-forward network as showed in Fig. 1. The hidden layer is consist of PU, the output layer is consist of SU. Suppose all the layers make use of linear activation functions, from $\prod_{i=1}^{N} x_i^{w_i}$ we know that if x_i and w_i are negative, there will have a part of imaginary number. In order to avoid it, the input x_i can normalize by the bellowing formula.

$$\text{Normalized value} = \frac{\text{original value}}{\max(|\text{original values of all instances}|)} + 2 \quad (4)$$

Because $0^{-1/2}$ is meaningless, so plus a bias more than or equal to 2, we adopt 2, after preconditioning, x_i's normalized value among (1, 3).

In the neural network, the quantity of hidden layer units decides its computation complexity degree of complexity. So the duty of training is to determine 3 groups of parameters: the weight connected input and hidden layers (denoted as W_IH), the amount of hidden layer units (denoted as Num_H) and the weight connected hidden and output layers (denoted as W_HO). Since the output layer is constituted by SUs and the activation function is linear function, so the determination of W_HO can be formulized a problem of solving linear algebraic equations when the outputs of hidden layers are given. Thus, just Num_H and weights W_IH are encoded into a particle for stochastic searching. In each iteration, each particle determines the weights W_IH and Num_H. Then the outputs of hidden layer are figured out, according to which, as well as expected target patterns, W_HO can be obtained by Singular Value Decomposition (SVD). In this study, we formulate the neural network training as an optimization problem and then employ PSO algorithm to resolve it. When adopting PSO, we must tackle two issues: the representation of a particle and designing fitness function.

The encoding representation of a single particle: $HUnit_i$ denotes the weight vector that connects the i-th hidden unit to input layer, where the number of element of this vector is equal to that of input units. $Flag_i$ indicates whether or not the i-th hidden unit is involved into the network, i.e., whether or not $HUnit_i$ is included in calculation. If $Flag_i >= 0$, the i-th hidden unit is included in the network. Otherwise the i-th hidden unit is removed from the network (Fig. 2). In such a way, particles can be interpreted to networks with various number of hidden units though they have a fixed length for a particle. Here, different particles correspond to networks with different number of hidden units.

Hidden-existence-array			Input-hidden-weight-array		
$Flag_1$	$Flag_2$	...	$HUnit_1$	$HUnit_2$	...

Fig. 2: Structure of a particle

This is hybrid representation that the first part consists of discrete variables and the second part continuous ones. It is well known that PSO performances very well on continuous variables. In 1997, Kennedy and Eberhart (1997) proposed a method to handle discrete variables with a little modification of the original PSO. The only difference is in the position update equation, which is modified to:

$$\begin{cases} x_{id} = 1 & \text{if } \text{rand}() < S(v_{id}) \\ x_{id} = 0 & \text{otherwise} \end{cases} \quad (5)$$

where, S is a sigmoid function and rand() is a random function in the range (0,1); Vmin, Vmax are usually set to -6, 6. Here, the velocity is a new meaning. Each v_{id} represents the probability of bit x_{id} taking the value 1. Suppose the previous best position p_{id} is 1 in any bit, $(p_{id}–x_{id})$ may be 1, 0. When x_{id} is same to p_{id} ($x_{id} = p_{id} = 1$), $(p_{id}–x_{id})$ is equal to 0, which has no contribution to the change of velocity according to formula 2. When x_{id} is different from p_{id} ($x_{id}=0$; $p_{id}=1$), $(p_{id} –x_{id})$ is equal to 1, which increases the velocity. On the other hand, the previous best position p_{id} is 0 and x_{id} is different from p_{id}, the velocity will be decreased, which increases the probability of x_{id} at the next step to be 0. In a word, formulas 2, 5 increase the probability of any bit to be corresponding the best position. Therefore, the essence of discrete PSO is same as that of continuous one. They both promote particles gathering toward the best particle. If we incorporate above discrete PSO into neural network training, a particle must be updated by two different formulas at each iteration. Regarding to a particle, the first part is updated by formula 5 and the second part by formula 3, which will complicate the algorithm.

Usually, evolutionary algorithm uses consistent encoding method, that is, either all discrete variables or continuous variables. Now that the discrete and continuous PSO have the same essence, we still use formulas 2, 3 as velocity update equation and position update equation respectively to handle discrete problem. When the fitness function is evaluated, x_{id} is given the different explanation.

m_{id} = hardlim (x_{id})

hardlim function is defined as follows:

$$\text{hardlim}(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \quad (6)$$

m_{id} is used to calculate the fitness according to the problem to be solved, x_{id} is still a continuous variables. The following experiment will proof the validity of our proposal. The discrete test problem (Kennedy and Eberhart, 1997) is used to compare the performances of our proposal and Eberhart's discrete PSO algorithm.

$$\text{Max } f_1(u) = 78.60 - \sum_{i=1}^{3}\left(u_i^2\right) \quad (7)$$

u_i is represented by 10 binary bits. Since there are three u_i, the dimension of particles is 30. Every 10 binary bits will be transformed to a real value. For example, a 10-bit string is s = 1010001111 where u_i is 1.4300.

The transform function is as follows:

$$u_i = (\text{decimal}(s) - 2^9)/2 \quad (8)$$

where, decimal function converts binary number into decimal number. In fitness evaluation, binary bits m_{id} are first formed from x_{id} according to formula 6 and then are transformed according to formula 8. The resulting values are taken into formula 7 to calculate the fitness.

As shown in Fig. 3, our proposal can effectively handle discrete problem. Furthermore, we conducted many experiments with different values of Vmin, Vmax. The experimental results show that Vmin and Vmax have no special requirement and their values do not influence the performance. So their values of the first part can be assign the same value as that of the second part. Therefore, with our proposal the two parts of a particle are consistent. The training algorithm can use the same velocity and position update equations and parameters, which is simple and effective.

The design of fitness function: In our training algorithm, $Flag_i$ has new meaning. $Flag_i \geq 0$ means that the i-th hidden unit is included in the network, otherwise means that the i-th hidden unit is remove from the network. According to the following formula, the resulting network is measured on the training set where PN input-target patterns are given.

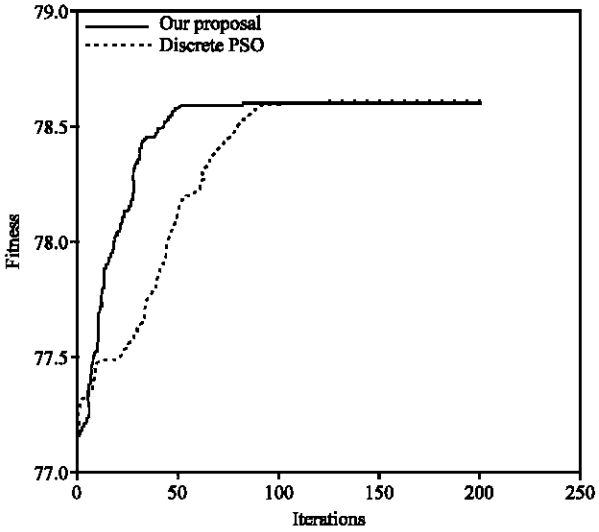

Fig. 3: The fitness against iterations

$$\text{Min fitness} = \frac{1}{PN}\sum_{p=1}^{PN}\left\|t_p - o_p\right\|^2 + k \cdot \frac{N\,\text{hidden}}{N\,\text{max hidden}} \quad (9)$$

where, t_p and o_p are the desired output and network output for pattern p respectively, k is a constant; Nhidden is the number of hidden units involved in networks; Nmaxhidden is predefined the max possible number of hidden units. The above formula takes into account both approximate accuracy and scale of networks, thus the resulting PUNNs can alleviate over-fitting. Parameter k is used to balance approximate accuracy and structure complexity. Value of k is decided during the course of the experiments according to problem's prior knowledge. In the experiments of the paper, it is suitable that k = 2.

EXPERIMENTAL SET-UP

The data sets used in this section were obtained from the UCI repository of machine learning databases (Blake *et al*., 1998). Based on every single standard data set, we take three algorithm experiments; due to all of the three algorithms betake three layers of neural network, so we use the bellowing quality index to evaluate them: (a) the unit number of hidden layer, which embodies the complex grade of neural network structure and (b) the generalization ability of neural network, in other words, the correct classification rate of testing sets of standard data sets. Its generalization ability is an important technical index of judging quality of neural network.

Three different algorithms:

Algorithm 1: PSPUNN: It is a kind of algorithm which is proposed by authors. It is a type of training algorithm which uses PSO allocating the structure and weight of

PUNN. PSPUNN is a sort of neural network structure optimization algorithm; here the Nmaxhidden represents the maximum possible number of hidden layer units; Nhidden represents the obtained ultimate number of hidden layer units through PSPUNN.

Algorithm 2: PSO-fixation: It is a sort of training algorithm which uses PSO optimizing the weight of PUNN. Here, the structure of PUNN is prefixed by the designer's prior knowledge. Owing to the PUNN structure used here is the three-layer feed-forward network of Fig. 1, fixing the number of the hidden layer unit is enough. We take two experimentations about fixed structure: PSO-FHmax and PSO-FHmin. PSO-FHmax represents making use of PUNN which have a number of Nmaxhidden hidden units; PSO-FHmin represents making use of PUNN which have a number of Nhidden hidden units.

Algorithm 3: Cascade correlation algorithm: CC algorithm (Cascade correlation algorithm) is a famous classic neural network structure optimization algorithm. In this paper, it is used to allocate structure and weight of PUNN depending on standard data set. The neural network which inferred from CC algorithm is called CC network. The CC network can be seemed as changed three-layer feed-forward neural network comparing with common three-layer feed-forward neural network. It has connections among the hidden layer nodes. CC hidden represents ultimate number of hidden layer unit which is obtained from CC algorithm.

In the PUNN, from $\prod_{i=1}^{N} x_i^{w_i}$ we know: if x_i and w_i are minus, it will have an imaginary number section. In order to avoid it, all the experimental datasets in the paper are standardized by formula 4. The experimentation adopts 4 standard data sets (Table 1).

Parameter set of PSO like bellow: inertia factor w is linearity descending between 0.9 and 0.4, learning rate $c_1 = c_2 = 2$, Vmin = -2, Vmax = 2, the maximum iterative degree is 2000; the population size is 10. Nmaxhidden: represents the number of the maximum possible hidden layer neuron; Nhidden: represents the unit number of ultimate hidden layer which gained by PSPUNN.

CC algorithm's parameter set like bellow: the activation function of input layer using linearity function, hidden layer using SIGMOID function, output layer using linearity function. The maximum hidden layer unit number MaxHUnit_CC = 40. The condition which terminates the increasing of CC algorithm hidden layer unit number: aiming at relative data sets' training set, if CC corrective classification rate more than or equal to PSPUNN's or CC hidden = MaxHUnit_CC, CC algorithm will stop increasing the unit number of hidden layer. Other experimental parameter uses CC's default setting (Fahlman and Lebiere, 1990).

In the study, we use 3-fold cross-validation to do all experiments. All experimentations will be circulated 3 times. Firstly, every data set should be divided into three parts. Then, in each of them, one part as testing set, others' two parts as training set. Each experiments should select different one-part from data set. Train Accuracy and Test Accuracy point out separately that on the training set and testing set, the experiment working 3 times will get average correct classification rate; while on the PSPUNN and CC, the number of hidden layer unit can be obtained by running 3 times to get average value, then round down integer. Computer resources as follows: CPU is intel Celeron 2.4 G and 512 M RAM.

Table 1: UCI experiment data sets information

Data set name	Iris	Wine	New-thyroid	Page-blocks
Patterns	150	178	215	5473
Input units	4	13	5	10
Output units	3	3	3	5

EXPERIMENTAL RESULTS

Comparison of PSPUNN and PSO-fixation: Through Table 2 and 3 we know that PSPUNN can allocate logical network structure automatically and its resulted network has quite higher predicted precision degree on the testing set. Figure 4 shows the results comparing of PSPUNN and PSO-Fixation on different dataset. On reverse, PSO-FHmax trained PUNN, because the higher complex degree of network, owns poor accuracy on the testing set. PSO-FHmin trained PUNN, due to its number of hidden layer neuron are decided by number (which called Nhidden) which gained by PSPUNN, so its network's generalization ability is better than PSO-FHmax's. It demonstrates that PSPUNN has the automatic ability to realize PUNN's

Table 2: The result of PSPUNN algorithm

Data set name	Iris	Wine	New-thyroid	Page-blocks
Nmaxhidden	10.0000	30.0000	15.0000	20.0000
Nhidden	2.0000	8.0000	4.0000	5.0000
Train accuracy	0.9811	0.9998	0.9757	0.9527
Test accuracy	0.9629	0.9525	0.9257	0.9478

Table 3: The result of PSO-Fixation algorithm (Including PSO-FHmax and PSO-FHmin)

Data set name	Iris	Wine	New-thyroid	Page-blocks
The result of PSO-FHmax				
Hidden units	10.0000	30.0000	15.0000	20.0000
Train accuracy	0.9920	1.0000	0.9893	0.9800
Test accuracy	0.9618	0.9016	0.8911	0.9216
The result of PSO-FHmin				
Hidden units	2.0000	8.0000	4.0000	5.0000
Train accuracy	0.9824	0.9986	0.9774	0.9601
Test accuracy	0.9635	0.9414	0.9307	0.9411

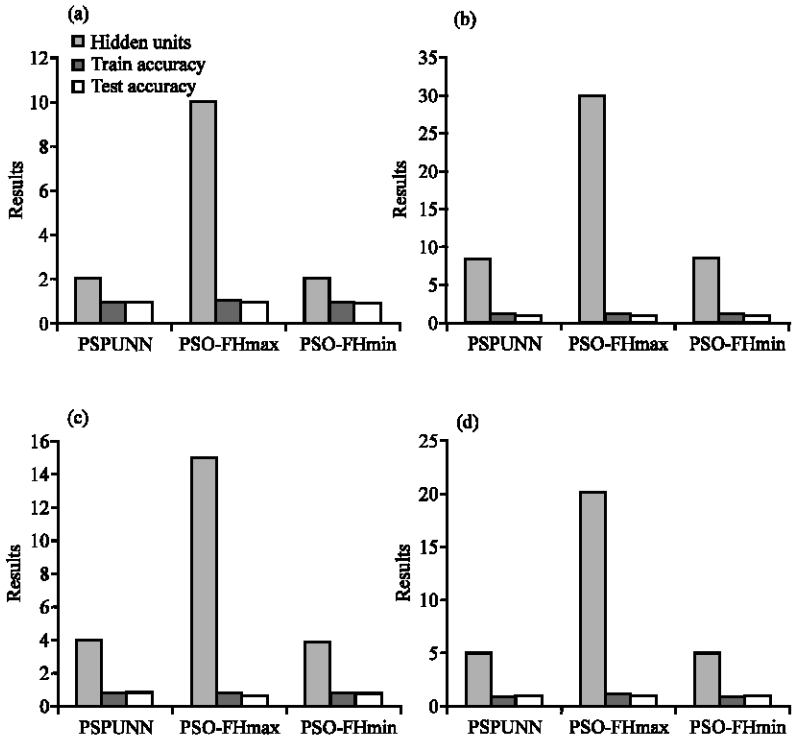

Fig. 4: Results comparing of PSPUNN and PSO-Fixation on different algorithm (a) Iris, (b) Wine, (c) New-thyroid and (d) Page-blocks

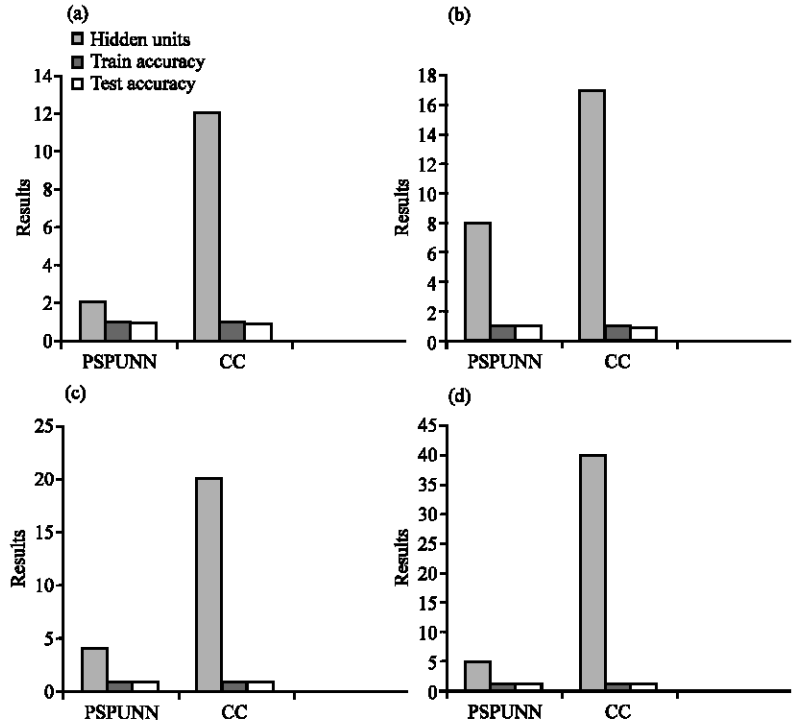

Fig. 5: Results comparing of PSPUNN and CC on different algorithm (a) Iris, (b) Wine, (c) New-thyroid and (d) Page-blocks

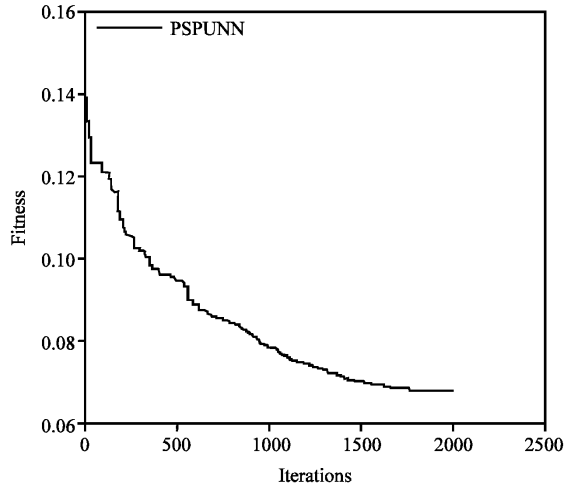

Fig. 6: The fitness values against iterations

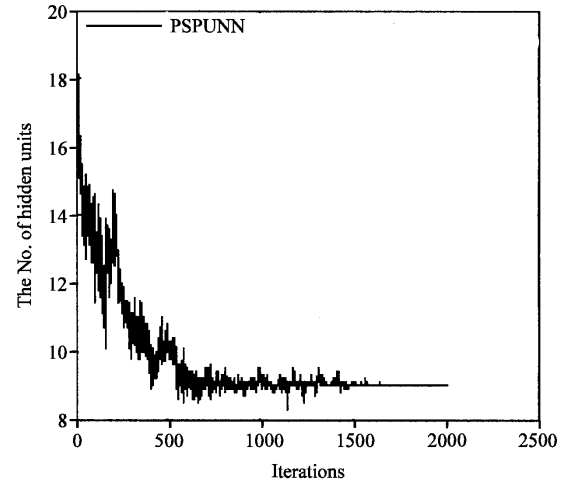

Fig. 7: The No. of hidden units against iterations

rational structure allocation dynamically. Upwards experiments show that PSPUNN is a kind of simple and effective algorithm and it is inferred network has stronger generalization ability.

Figure 6 shows the fitness evolution curve when PSO-Var trained PUNNs on Wine data set. Figure 7 shows the corresponding average number of hidden units of PUNNs that were formed according to particles at each iteration. As shown in Fig. 7, in early iterations, the structures of PUNNs were intensively explored. The structures gradually became stable along the iterations. In later iterations, the algorithm focused on adjusting the weights.

Comparison of PSPUNN and CC: PSPUNN is a sort of mentioned algorithm in this study, which learns and optimizes structure of neural network, while CC (Cascade correlation algorithm) is a famous classical algorithm

Table 4: The result of CC algorithm

Data set name	Iris	Wine	New-thyroid	Page-blocks
CC hidden	12.0000	17.0000	20.0000	40.0000
Train accuracy	0.9930	0.9983	0.9799	0.9317
Test accuracy	0.9480	0.9525	0.9225	0.9327

which optimizes structure of neural network. Table 2 and 4 show that PSPUNN has better optimized ability of neural network structure than CC after experiences tests of four data sets. Figure 5 shows the results comparing of PSPUNN and CC on different dataset Depending on standard data sets, PSPUNN can figurate smaller network structure (through comparing the unit number of hidden layer) automatically compared with CC, besides it also demonstrates that the making out network has better ability of generalization.

CONCLUSIONS

In this research, a approach (PSPUNN) to learn the structure of PUNN is proposed, it can configure the architecture and weight of PUNN simultaneously, depending on training sets and using PSO. During the process of iteration of PSO, every particle determined the weights of connecting input and hidden layer and the number of the hidden layer units and then the weights that connect the hidden and output layers are obtained by Singular Value Decomposition (SVD). Consequently, a PUNN network was constructed. Fitness function takes into account not only mean square error between networks output and desired output, but also the number of hidden units. Therefore, the resulting network can alleviate over-fitting problems. Through experimental comparing among PSPUNN, PSO-Fixation and CC algorithms, the results show that PSPUNN algorithm has ability of allocating smaller network structure automatically, compared to CC algorithm and the resulting networks obtain strong generalization abilities.

ACKNOWLEDGMENTS

The research is funded by National Basic Research 973 Program of China (No. 2004CB719401) and also supported by the National Defense 11th-Five-Year Preliminary Research fund.

REFERENCES

Blake, C., E. Keogh and C.J. Merz, 1998. UCI Repository of Machine Learning Databases. http://mlearn.ics.uci.edu/MLRepository.html.

Clerc, M., 1999. The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization. Proceeding of the Congress on Evolutionary Computation, pp: 1951-1957.

Durbin, R. and D.E. Rumelhart, 1990. Product Units: A Computationally Powerful and Biologically Plausible Extension to Backpropagation Networks. Advances in Neural Information Processing Systems, San Mateo, CA: Morgan Kaufmann Publishers, 1: 133-142.

Engelbrecht, A.P. and A. Ismail, 1999. Training product unit neural networks. Stability and Control. Theor. Appl., 2 (1/2): 59-74.

Fahlman, S.E. and C. Lebiere, 1990. The Cascade-Correlation Learning Architecture. In: Advances in Neural Information Systems II, Touretzky, D.S. (Ed.). Morgan Kaufmann Publishers, pp: 524-532.

Fischer, M.M., 2002.A Novel Modular product unit neural network for modeling constrained spatial interaction flows. Proceedings of the Congress on Evolutionary Computation (CEC '02), Vol 2. Piscataway, NJ, USA: IEEE Press, pp: 1215-1220.

Hornik, K., M. Stinchcombe and H. White, 1989. Multilayer feedforward networks are universal approximators. Neural Networks, 2: 359-366.

Ismail, A. and A.P. Engelbrecht, 2000. Global optimization algorithms for training product unit neural networks. IEEE. International Joint Conference Neural Networks, pp: 132-137.

Ismail, A. and A.P. Engelbrecht, 2002. Pruning Product Unit Neural Networks. Proceedings of the International Joint Conference on Neural Networks, Vol 1. Piscataway, NJ, USA: IEEE. Press, pp: 257-262.

Janson, D.J. and J.F. Frenzel, 1993. Training product unit neural networks with genetic algorithm. IEEE. Expert Mag., 8: 26-33.

Kennedy, J. and R. Eberhart, 1995. Particle swarm optimization. Proceedings of IEEE International Conference on Neural Networks, Piscataway, NJ, pp: 1942-1948.

Kennedy, J. and R. Eberhart, 1997. A discrete binary version of the particle swarm algorithm. IEEE International Conference on Systems, Man and Cybernetics, 5: 4104-4108.

Leerink, L.R., C.L. Giles, B.G. Horne and M.A. Jabri, 1995. Learning with product units. Adv. Neural Proc. Syst., 7: 537-544.

Martinez-Estudillo, F.J. and C. Hervas-Martinez *et al*., 2006. Evolutionary product-unit neural networks for classification. LNCS., 4224: 1320-1328.

Schmitt, M., 2001. Product unit neural networks with constant depth and superlinear VC dimension. LNCS., 21 (3): 253-258.

Shi, Y. and R. Eberhart, 1998.A Modified particle swarm optimizer. Proceedings of the IEEE Congress on Evolutionary Computation (CEC1998), Piscataway, NJ, pp: 69-73.

Van Den Bergh, F. and A.P. Engelbrecht, 2001. Training product unit networks using cooperative particle swarm optimisers. IEEE International Joint Conference on Neural Networks, pp: 126-131.