한국퍼지 및 지능 시스템학회 논문지 1997, Vol. 7, No. 4, 74~81

투영신경회로망의 훈련을 위한 진화학습기법

Evolutionary Learning Algorithm for Projection Neural Networks

황민웅 · 최진영
Min Woong Hwang and Jin Young Choi

서울대학교 전기공학부, ERC-ACI, ASRI
School of Electrical Engineering, ERC-ACI, ASRI, Seoul National University

요 약

본 논문에서는 시그모이드 함수와 방사형 기저 함수 모두를 생성시킬 수 있는 특별한 은닉층 노드를 갖는 투영신경회로망에 대하여 알아보고 그것을 훈련시키기 위한 진화 학습 기법을 제시한다. 제시된 기법은 신경회로망의 매개변수와 연결 가중치뿐만 아니라, 어떤 목적함수를 나타내기위한 최적의 은닉층 노드개수 또한 구조 최적화를 위한 진화연산자를 통해 찾아낸다. 각각의 은닉층 노드의 역할은 진화를 거듭하면서 방사형 기저 함수를 나타낼지 시그모이드 함수를 나타낼지 결정된다. 알고리즘을 구현하기위해서 투영신경회로망은 연결 고리 리스트 자료구조로 나타내었다. 모의실험에서 기존의 오차역전파에 의한 학습과 구조 성장 방식보다 적은 노드로 투영신경회로망을 훈련시킬 수 있음을 볼 수 있다.

ABSTRACT

This paper proposes an evolutionary learning algorithm to discipline the projection neural networks (PNNs) with special type of hidden nodes which can activate radial basis functions as well as sigmoid functions. The proposed algorithm not only trains the parameters and the connection weights but also optimizes the network structure. Through the structure optimization, the number of hidden nodes necessary to represent a given target function is determined and the role of each hidden node is decided whether it activates a radial basis function or a sigmoid function. To apply the algorithm, PNN is realized by a self-organizing genotype representation with a linked list data structure. Simulations show that the algorithm can build the PNN with less hidden nodes than the existing learning algorithm using error back propagation(EBP) and network growing strategy.

1. Introduction

The performance of neural network depends mainly on the structure of the networks which comprises the number of hidden layer, the number of nodes in a hidden layer, the weights between layers and the activation function, etc. But when a neural network approximates a function, there is no general method to determine the things enumerated above. A very simple method we can do easily is as follows: build a neural network, discipline it, test it for a desired function, modify its structure and repeat this routine until the result is satisfactory. However, this method takes too much time and is very boring procedure. In addition, we cannot find what relation between stucture and performance exists. In according to recent study, finding the optimal network structure and disciplining the network are NP-complete problems. And we can enhance the efficiency of learning by constraining the structure intuitively [1]. Many approaches to optimizing the structures of neural networks by means of evolutionary algorithms have been studied [1, 2].

Neural networks, if they have different activate functions respectively, show their own characteristics when they approximate a certain function. It can be shown that multilayer perceptron with sigmoid func-

※이 논문은 서울대학교 "공대 교육 연구재단, 대학 발전 기금" 연구비로 연구되었음(95-6004)

tions as activation function approximates any functions. However when we use it to represent a locally salient function, the size of network comes to be big, which is an inefficient strategy. On the other hands, many neural networks with radial basis functions as activation function are also available for function approximation. Although these models can easily represent a locally salient function, many nodes in hidden layer is required to represent a globally spreaded function such as plane. To combine the two characteristics efficiently, there have been many efforts [3, 4, 5, 6]. Most of them is to combine simply sigmoid network and radial basis network. Recently Wilensky and Manukian suggested the projection neural networks (PNNs) which can represent both radial basis function and sigmoid function by same structures with different parameter values [11]. The PNN has been extended to a general form so that it can be applied to function approximation problem [7]. They have also proposed network growing learning algorithm for PNN to deal with local minima problem in EBP learning. The learning algorithm shows good performance but can't optimize the network and results in over creation of hidden nodes.

To optimize and train the PNN, this paper introduces an evolutionary learning algorithm. We investigate how the projection neural networks can activate such functions which have different characteristics and how evolutionary algorithm is modified and applied for the projection neural networks. We first represent a PNN (its structure and parameters) by a genotype form. Parameters of hidden node and connection weights between layers are coded by binary string and structure is realized by linked list data structure. Individuals which are created at the stage of initialization of population. The number of the hidden nodes of each individual is somewhat random in a certain predefined range. Thereafter as normal evolutionary algorithm does, we apply genetic operators to each individual, evaluate fitness values and reproduce new generation, and so on. Here we adopt new operators to optimize the network structure. They are node appending and node deleting operators. These are also probabilistic operators which can increase or decrease the number of hidden nodes. Besides, as generation goes, we change the probability with which genetic operators are applied and the genetic parameters such as scaling factor in fitness functions. And we apply this algorithm to PNN to approximate given target functions and discuss its performance compared to the existing algorithm.

2. The structure of projection neural network

The main difference between PNN and other NN is that PNN projects the original n-dimensional input vector onto the n+1-dimensional vector and utilizes it as a new input vector.

The details of this scheme are as follows : 1) project the original input vector $\mathrm{x}_{ori}=[x_{ori_1}\, x_{ori_2}\cdots x_{ori_n}]^T$ onto the n+1-dimensional vector $\mathrm{x}=[x_1 x_2 \cdots x_n x_{n+1}]^T$ by appending a new coordinate n which is orthogonal to the original n-dimensional space; 2) normalize the projected vector to have magnitude R in the n+1-dimensional space. Then the projected and normalized vector is expressed as follows.

$$\mathrm{x}=\frac{R}{\sqrt{\| \mathrm{x}_{ori} \|^2+h^2}}\,[\mathrm{x}_{ori}\ h]^T \tag{1}$$

This vector(1) is used as an input vector to the network.

In addition to projection and normalization of the input vector, the weight vector between nodes of input layer and nodes of hidden layer also should be normalized to have magnitude R. Hence,

$$\mathrm{w}_i=[w_{i1} w_{i2} \cdots w_{in+1}]^T \quad (i=1, 2, \cdots, m) \tag{2}$$

$$\| \mathrm{w}_i \| = R \tag{3}$$

where n, m are the number of nodes in input layer and hidden layer respectively.

The output of each hidden layer node (4) obtained through activation function (5) whose input is the result of subtracting the bias B_i from the inner product of $\boldsymbol{x}$ and $\boldsymbol{w}_i$

$$\phi_i=\frac{1}{1+\exp(-\mu_i(\mathrm{w}_i{}^T\,\mathrm{x}-\beta_i))} \tag{4}$$

where μ_i is shaping parameter.

$$\phi(\xi)=\frac{1}{1+\exp(-\xi)} \tag{5}$$

The total output of neural network which has m hidden nodes can be represented in (6).

$$\hat{f}(x)=\sum_{i=1}^{m} v_i \phi(\mu_i(\sum_{j=1}^{n+1} w_{ij}x_j-\beta_i)) \quad (6)$$

where v_i is weight from ith hidden node to output node.

Now we look into how PNN can activate both sigmoid function and radial basis function. In equation (4), since $w_i^T x=\|w_i\|\,\|x\|\cos\theta=R^2\cos\theta$, $w_i^T x$ depends only on the angle θ between input vector and weight vector. To simplify analysis, we set R and μ_i arbitrarily to be 1. Then ϕ_i becomes

$$\phi_i=\frac{1}{1+\exp(-(\cos\theta-\beta_i))} \quad (7)$$

In Fig. 1, we plot the ϕ_i with respect to several β_i. It can be noted that if θ includes $\theta=0$, radial basis function will be activated and if θ doesn't includes $\theta=0$, sigmoid function will be activated. For example, for a certain range of input vector if w_i is included in the range (w_1 in Fig. 2), θ will have surely 0 value at a input vector, and this case results in activating a radial basis function. This case is plotted in Fig. 3. on the contrary, if w_i is not included in the range of input vector (w_2 in Fig. 2), θ never has 0 value for any input vector, this case results in activating a sigmoid function. This case is plotted in Fig. 4.

In addition, we can note β_i takes part in varying the peak value and we can also guess that μ_i will be used in adjusting the width of range where θ_i ctivates.

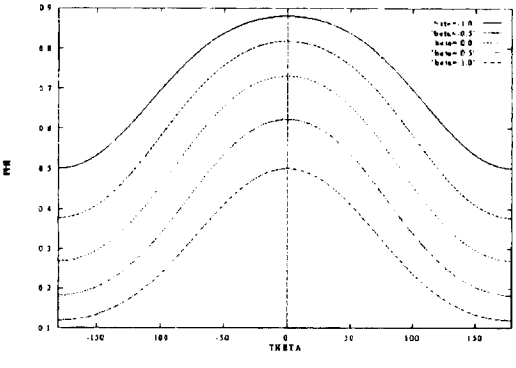

Fig. 1. θ_i with respect to various β_i

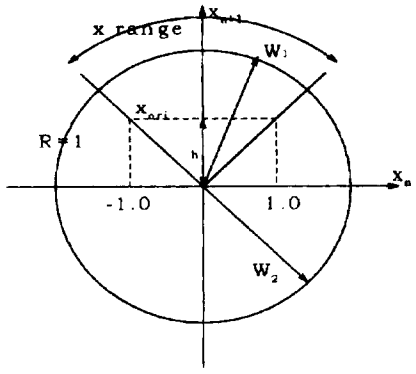

Fig. 2. Projection from n-dimensional space to n+1-dimensional space

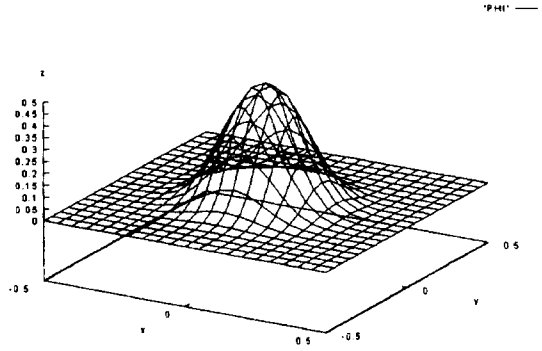

Fig. 3. Redial basis function activated when β_i=1, μ_i=100, w_i=[0 0 1]^T

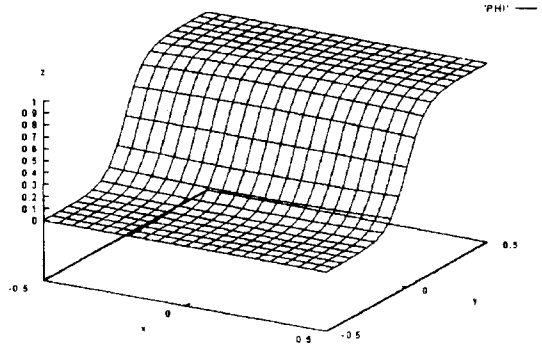

Fig. 4. Sigmoid function activated when β_i=0, μ_i=25, w_i=[0 1 0]^T

3. Evolutionary learning algorithm for PNN

Using evolutionary learning algorithms for neural network can be summarized as follows : 1) represent a neural network (its structure and parameters) by a

genotype; 2) compose individuals which are genotypic neural networks into a population; 3) apply genetic operators;

4) assign a fitness value to each individual; 5) reproduce a new population on the basis of fitness values; 6) and iterate this procedure, which is called as a generation, until criteria are satisfied.

But, evolution of neural network is slightly different from optimizing regular multivariable functions. Because the role of parameters in the neural network is not uniform. So to obtain a solution fast and precisely, modifying operators and constraining environment according to given problem are needed [9, 10].

3.1 Representation of PNN

To discipline PNN by using evolutionary algorithm, structure and all parameters of network must be represented by a genotype. In the case of PNN, we use only one hidden layer, and input nodes and output nodes are determined by given problem. Hidden layer nodes, their parameters and connection weights are included in a genotype which describes the PNN.

The PNN may have many nodes in hidden layer and this structure can be coded by the linked list data structure as shown in Fig. 5. The order of nodes and the information about which nodes are connected to each other are not important. We concern only how many hidden nodes there are and what values their parameters have. Each field in a node is coded by a certain length of binary string which can be transformed to a real value. If a field is too short, the parameter cannot have minute value, and if a field is too long, the solution space is so huge that we cannot find solution conveniently. In the Fig. 5, we must note that role of parameter in PNN is different to each other. Some are weights and others are shaping parameters, etc. In addition, they have difference in the range in which they have value.

In other words, the coded genotype is not uniform through fields. So determining the range of each field and the probability with which we apply genetic operators to each field is very important.

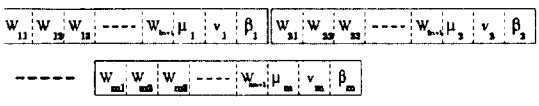

Fig. 5. Genotype for a neural network

In the Fig. 5, w_{mn} is a weight between mth input node and nth hidden node, μ_m, β_m are shaping factor, bias value of mth hidden node, respectively, and v_m is a weight between mth hidden node and output node. The field representing β_m is treated somewhat tricky. We do not use the β value itself. We transform it to zero or one when constructing PNN from genotype form. The reason we do such a thing is for PNN to approximate target function more efficiently, because PNN activates sigmoid function more explicitly when β is close to zero and PNN represents radial basis function well when β is almost one.

3.2 Application of mutation operator

Basically mutation is toggling a bit. With a certain probability mutation operator is applied to each bit in a field of each individual. And the probabilities which are applied to different fields are different. The reason is stated at the previous subsection.

3.3 Application of crossover operator

There are many methods to apply crossover operator. Since several nodes are connected to each other in neural network, dependency among nodes is so strong. Therefore, crossover operation is very destructive in evolution of neural network. Here, we adopt node level change. In PNN the order of nodes in linked list data structure is not important. So, one node in individual A is exchanged with one node in individual B. Surely this operation is probablistic and it is also impossible to know which node in each individual will join this procedure. We cannot know whether the selected node goes well with other nodes or not. It is desirable to apply this operator with as small probability as possible.

3.4 Appending and deleting hidden nodes

According to a given problem, there may exist optimal hidden nodes. Because we are willing to find not only optimal parameters but also optimal number of hidden nodes, there must be node appending and node deleting operator. To do this, we calculate the error of node itself when evaluating RMSE of the whole network. Among many nodes which compose a PNN, some are very helpful to network and others do not contribute or even de-

grade the performance of network. We want to prune the worst node in an individual and share the best node with another individual.

Node appending operator is similar to crossover operator. This operator is also applied with a certain probability. It operates as follows : find a minimal error node, which we can think to approximate target function well in some region, in an individual and append the node to its mate. Similarly, node deleting operator deletes a maximal error node, which we can think to be not helpful to network, in an individual if the invidual has more nodes than a certain predefined minimum nodes.

3.5 Selection and reproduction

Evaluating the fitness values of individuals, selecting excellent individuals and reproducing new generation are most important in evolutionary algorithms. In order to enhance evolution speed toward global optimum and not to fall into local optimum, following methods are required.

First, after obtaining objective function values we must scale the values to use them as fitnesses. Without scaling, in the early generation there is a tendency for a few super individual to dominate the whole population. Therefore, before searching through enough solution space, entire individuals converge to a local optimum. In this case fitness values must be scaled down to prevent takeover of the population by these super individuals.

And in the later generation, when the population is largely converged, competition among individuals is less strong and evolution tends to wander. In this case fitness values must be scaled up to accentuate differences between individuals.

Here, we adopt σ-truncation method as described in equation (8).

$$F' = F - \bar{F} + \alpha\sigma \tag{8}$$

In this equation, $\bar{F}$ is mean fitness value of population, σ is standard deviation of population and scale factor α is constant. We can set negative results (F') arbitrarily to be 0.

Second, in order to preserve the succession to preceeding generation, we must prevent operators from destroying the population. Hence elitism is indispensable. This plan excludes the best individual generated upto the current generation from the application of genetic operators in order to make it exist unchanged in the next generation. In convergence analysis of genetic algorithms it is proved that without elitism genetic algorithm is not convergent[12].

4. Simulations

4.1 Implementation of evolutionary algorithms

Population of a generation is composed of 50 individuals. One individual may have many nodes which are composed of fields of parameters, respectively. Here, one field is 8-bit binary string. Hence, the resolution is 256. And one node has 6 fields (w_1, w_2, w_3, μ, β, υ), because functions we will approximate here are all two-dimensional.

As generation increases, we vary parameters of algorithm such as probabilities of operators, scale factor, etc. Usually, mutation probability and crossover probability should decrease as generation goes. In our simulation, mutation probability and crossover probability start at 0.005, 0.05 respectively and end at both 0. And node appending and deleting probabilties increase from 0.1 to 0.2 linearly.

As decribed in section 3.5 scaling of fitness is done by using σ-truncation method(8). At (8) scale factor α also can be varied. In our simulation α decreases from 3.0 to 1.0 linearly.

Finally at selection and reproduction we adopt elitism.

4.2 Result 1

Function to be approximated by PNN is described in (9) and illustrated in Fig. 6. It is noted that this function is the mixture of a radial basis function and a sigmoid function. We can easily guess that about 2 hidden nodes are optimal.

$$f_1(x,y) = \exp\left(-\frac{(x+0.5)^2+y^2}{0.9}\right) + \frac{1}{1+\exp(-15(x-0.5))} \tag{9}$$

In our simulation the algorithm found the optimal PNN which had 2 hidden nodes. And the average node values are described in Table 1. Fig. 7 is the

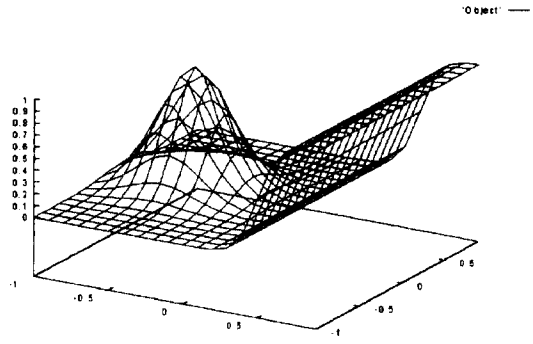

Fig. 6. Target function $f_1(x, y)$

output function $\hat{f}(x, y)$.

The constraints we take are as follows : At initialization of population the individual has nodes between 1 and 10. The ranges of parameter values are w_1, w_2, w_3=[-1,1], μ=[0,100], υ=[0,2], β=[0,1]. And RMSE(Root Mean Squared Error) is calculated at 900(30×30) points. If RMSE of the elite in any generation goes below 0.03, then iteration stops.

All simulation was done under UNIX environment of Sparc 20 system. C++ language was used.

Table 1. Parameter values of hidden nodes

node	w_1	w_2	w_3	μ	υ	β
1	0.898	-0.011	-0.442	25.1	-0.980	0.388
2	-0.422	0.003	0.906	45.5	1.92	-0.976

Table 2. comparision between the evolutionary algorithm and the newtork growing algorithm with EBP

method	RMSE	structure(i-h-o)	time
EBP	0.028	3-11-1	7 min
Evo.	0.021	3-2-1	20. min

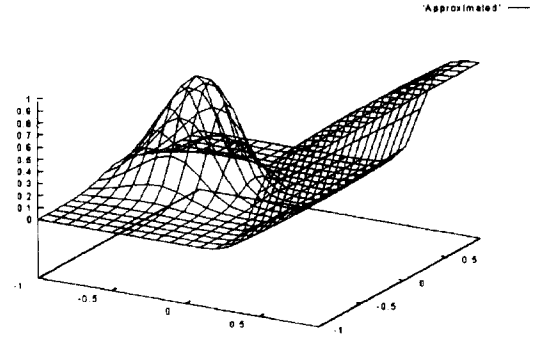

Fig. 7. Output function $\hat{f}(x, y)$

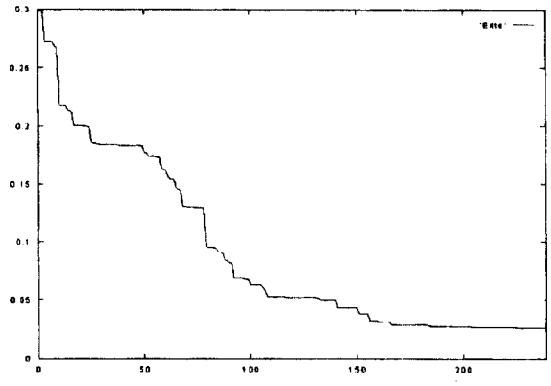

Fig. 8. Elite's learning curve(RMSE)

4.3 Result 2

Function to be approximated by PNN secondly is described in (10) and illustrated in Fig. 9. This time it can be noted that only radial basis functions compose the whole target function and about 4 hidden nodes are optimal.

$$f_2(x,y)=e-\frac{(x+0.5)^2+(y+0.5)^2}{0.9}$$

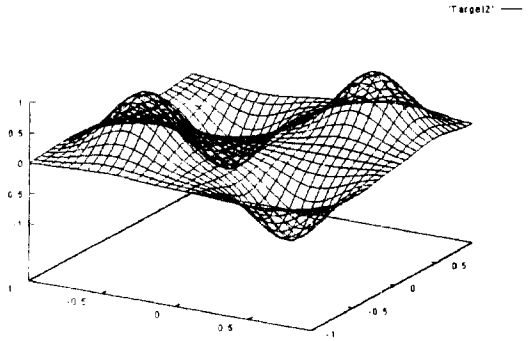

Fig. 9. Target function $f_1(x, y)$

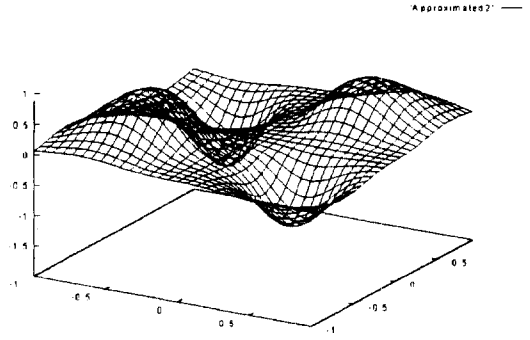

Fig. 10. Output function $\hat{f}(x, y)$

Table 3. parameter values of hidden nodes

node	w_1	w_2	w_3	μ	υ	β
1	0.407	0.433	0.805	27.8	0.996	0.945
2	-0.410	0.391	0.823	49.8	-1.655	0.881
3	-0.367	-0.388	0.845	55.3	1.89	0.962
4	0.389	-0.415	0.821	65.88	-1.93	0.727

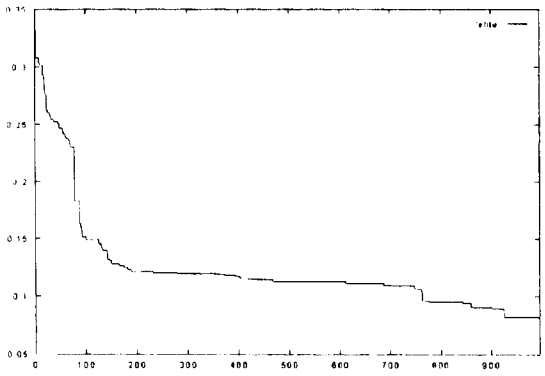

Fig. 11. Elite's learning curve(RMSE)

$$-e^{-\frac{(x-0.5)^2+(y+0.5)^2}{0.9}} -e^{-\frac{(x+0.5)^2+(y-0.5)^2}{0.9}} +e^{-\frac{(x-0.5)^2+(y-0.5)^2}{0.9}} \quad (10)$$

In our simulation the algorithm found the network which has 4 hidden nodes. And the result is described in Fig. 10 and the value of nodes in Table 3.

5. Conclusions

In this paper, we have suggested the evolutionary learning algorithm for the projection neural network which can be used in approximating continuous functions. The proposed algorithm is not liable to indulge in local minimum and uses only objective function information without derivatives or other auxiliary knowledge unlike the conventional EBP algorithm. Moreover the algorithm gives the construction of optimal PNN with much less hidden nodes than the existing network growing algorithm with EBP.

But, this algorithm needs much resources such as time and memory space because many individuals occupy much memory space and iterations take much time. And the result is somewhat probabilistic. Hence, on every trial the result may be different. In addition, this algorithm can not tune the parameters very precisely. To a certain degree, this algorithm can find the almost optimal parameters. For the more fine tuning, EBP algorithm can be additionally applied to the optimized PNN by the proposed evolutionary algorithm.

References

[1] J.S.Judd, Neural Network Design and the Complexity of Learning, MIT Press-Bradford Book, Cambridge, MA, 1990.

[2] Martin Mandischer, Representation and Evolution of Neural Networks, Department of Computer Science, University of Dotmund, Germany.

[3] M.Hirahara and N.Oka, A hybrid model composed of a multilayer perceptron and a radial basis function network, Procs. of IJCNN, Nagoya, vol.1, pp. 1353-1356, 1993.

[4] R.M.Kil and J.Y.Choi, Time series prediction based on global and local estimation models, WCNN, vol. 4, pp.617-621, July 1993.

[5] S.Park,L.J.Park and C.H.Park, Structural hybrid of multilayer neural networks and Gaussian potential function networks, Procs. of ICONIP, vol.3, pp. 1393-1397, Seoul, Oct 1994.

[6] S.Park,L.J.Park and C.H.Park, Efficient learning algorithm using structural hybrid of multilayer neural networks and Gaussian potential function networks, Journal of Korean Institute of Communication Sciences, vol.18, no.12, pp.2418-2425.

[7] Byeong-Kap Choi, Chong-Ho Choi, Jin-Young Choi, Projection Neural Networks for Function Approximation, KIEE, Vol. 44, pp.961-968, 1995.7.

[8] D.E.Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, 1989.

[9] Z.Michalewicz, T.D.Logan, S.Swaminathan, Evolutionary Operators for Continuous Convex Parameter Spaces, Proceedings of the Third Annual Conference on Evolutionary Programming, pp.84-97, San Diego California, 1994.

[10] Mukesh J.Patel, Heuristical Constraints on Search Complexity for Multi-modal Non-optimal Models, IEEE Press, pp.164-180, 1995.

[11] G.D.Wilensky and N.Manukian, The Projection neural networks, Procs. of IJCNN, Baltimore, vol.2, pp.II-358~II-367, 1992.

[12] Gunter Rudolph, Convergence analysis of canonical genetic algorithms, IEEE Transactions on Neural Networks, Vol.5, No.1, January, 1994.

Jin Young CHOI

Jin Young CHOI received the B. S., M. S., and Ph.D. degrees in Control and Instrumentation Engineering from Seoul National University, Seoul, Korea, in 1982, 1984, and 1993, respectively. >From 1984 to 1989, he joined the project of TDX switching system at the Electronics and Telecommunication Research Institute (ETRI). From 1992 to 1994, he was with the Basic Research Department of ETRI, where he was a senior member of technical staff working on the neural information processing system. Since 1994, He has been with Seoul National University, where he is currently a Assistant Professor in the School of Electrical Engineering. His research interests are brain-based computing and control technologies including neural network, fuzzy logic, evolutionary computing, and adaptive and learning control.

Min Woong HWANG

Min Woong HWANG received the B. S. degree in School of Eletrical Engineering from Seoul National University, Seoul, Korea, in 1996. He is currently working towards the M. S. degree in Seoul National University. His research interests are evolutionay computing, neural networks.