Evolutionary Learning of Sigma-Pi Neural Trees and Its Application to Classification and Prediction

시그마파이 신경 트리의 진화적 학습 및 이의 분류 예측에의 응용

Byoung-Tak Zhang*
장 병 탁*

이 논문은 한국과학재단 핵심전문 연구과제 연구비에 의해 일부 지원되었음(과제번호 961-0901-001-2)

ABSTRACT

The necessity and usefulness of higher-order neural networks have been well-known since early days of neurocomputing. However the explosive number of terms has hampered the design and training of such networks. In this paper we present an evolutionary learning method for efficiently constructing problem-specific higher-order neural models. The crux of the method is the neural tree representation employing both sigma and pi units, in combination with the use of an MDL-based fitness function for learning minimal models. We provide experimental results in classification and prediction problems which demonstrate the effectiveness of the method.

요 약

하이오더 신경망에 대한 필요성과 유용성에 대해서는 신경망 연구의 초기부터 잘 알려져 있다. 그러나 오더가 늘어남에 따라 항의 수가 급격히 증가하는 문제로 인하여 이러한 망을 설계하고 학습하는데 많은 어려움이 있었다. 본 논문에서는 문제에 적합한 하이오더 신경망 모델을 효율적으로 구성하기 위한 진화적 학습 방법을 제시한다. 이 방법에서는 시그마유닛과 파이유닛을 융합한 신경트리 표현을 사용한다. 또한 MDL기반의 적합도 분류 및 예측 문제에 있어서 제시된 방법의 유용성을 검증한다.

I. Introduction

One of the most popular neural network models used for supervised learning applications has been the mutilayer feedforward network. A commonly adopted topology employs one hidden layer with full connectivity between neighboring layers. This structure has been very successful for many applications. However, they have some weaknesses. For instance, the fully connected structure is not necessarily a good topology unless the task contains a good predictor for the full input space.

*건국대학교 컴퓨터공학과

Most network architectures consist of neural units which compute the weighted sum of inputs. These summation units are especially appropriate to approximating additive functions, since they employ linear combinations of inputs. However, the multilayer perceptrons cannot approximate efficiently if there are high order interactions between the inputs. Adding additional hidden layers may help to extend the representational capacity of the network, but the training becomes more difficult. A solution may be to use higher-order units. The necessity and usefulness of higher-order neural networks have been well-known. However the explosive number of terms hampers the design and training of such networks.

In this paper we present a method for the construction of higher-order neural networks with partial connectivity. The method uses a genetic algorithm. Genetic algorithms are search methods based on a population of individuals, each of which represents a search point in the space of potential solutions to a given problem [6]. The population is arbitrarily initialized, and it evolves toward better and better regions of the search space by means of randomized processes of selection, mutation and recombination. The environment delivers the fitness value of the search points, and the selection process favors those individuals of higher fitness to reproduce more often than those of lower fitness. The recombination mechanism allows the mixing of parental information while passing it to their descendants, and mutation introduces innovation into the population.

The presented method uses a tree representation of the network, called neural trees, on which genetic operators are applied to modify and find fitter architectures. Another feature of the method is the use of complexity factor in its fitness function. It makes an optimal trade-off between the error fitting ability and the parsimony of the network. In section 2 we describe this representation scheme in more detail. Section 3 describes the evolutionary algorithm for learning problem-specific neural trees. Section 4 analyzes the complexity of building polynomial networks using sigma-pi neural trees to motivate the genetic approach. Section 5 shows the experimental results. Conclusions are provided in Section 6.

Ⅱ. Genetic Algorithms for Neural Networks

Various schemes for combining genetic algorithms and neural networks have been proposed and tested in recent years [13, 15]. One possibility is to use genetic algorithms for selecting features of training patterns. This combination has already achieved some success on real world tasks. A second possibility is to use evolutionary algorithms to determin neural network weights. In weight optimization, the set of weights is represented as a chromosome and a genetic search is applied on the encoded representation to find a set of weights that best fits the training data. Some encouraging results have been reported which are comparable with conventional learning algorithms [10]. Where gradient or error information is not available, genetic algorithms may be a promising training method. A third possibility of combination is to use genetic search techniques to optimize the network topology. Here the topology of the networks is encoded as a chromosome and some genetic operators are applied to find an architecture which best fits the specified task according to some explicit design criteria. Many methods have been proposed for evolving network topologies.

A general way of genetically evolving neural networks was suggested by M hlenbein and Kindermann in [12]. Recent works, however, have focused on using genetic algorithms separately in each optimization problem, mainly in optimizing the network topology. Miller *et al.* [8] has described representation schemes in which the anatomical properties of the network structure are encoded as bit-strings. Similar representation has also been used by Whitley *et al.* [15] to prune unnecessary connections. All these methods use the backpropagation algorithm, a gradient-descent

method, to train the weights of the network.

We represent a feedforward network as a set of m trees, each corresponding to one output unit. For example, the genotype of a feedforward network consisting of $n=6$ inputs and $m=1$ output unit is shown in Figure 1. In this tree representation, what we call neural tree, a node consists of one or more elements. For hidden and output units, each node contains an activation function type U_i, a threshold value θ_i, and an arbitrary number of weight values w_{ij}. The node may point recursively to other hidden units U_i or an external input unit x_k $k \in \{1,\cdots,n\}$.

This encoding scheme can represent any feedforward network with local receptive fields and direct connections between non-neighboring layers (see [16] for more details) and thus extends the tree representation used in [7].

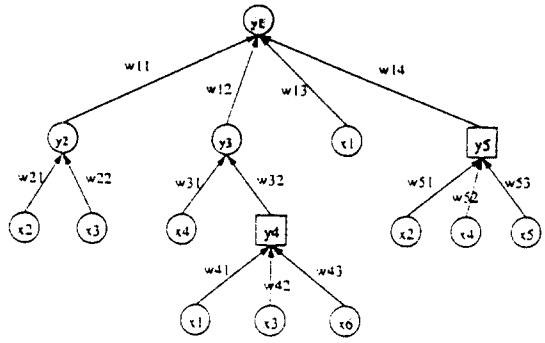

Fig. 1 An example of sigmpa-pi neural tree.

This is contrasted with the more commonly used perceptron architecture of fully connected feedforward networks.

Ⅲ. Evolving Sigma-Pi Neural Trees

The neural tree representation allows any type of activation functions to be defined. In particular, we may use pi units as well as the usual sigma units. While a sigma unit calculates a sum of weighted inputs,

$$u_i = \sum_{j \in R(i)} w_{ij} x_j \tag{1}$$

a pi-unit builds a {\it product} of weighted inputs:

$$u_i = \sum_{j \in R(i)} w_{ij} x_j \tag{2}$$

Here w_{ij} is the connection weight from unit j to unit i and $R(i)$ denotes the receptive field of unit i. By using pi units we can, for example, directly build k th-order terms

$$T^{(k)} = f_k \left(\prod_{j=1}^{j_k} w_{kj} x_j \right) \tag{3}$$

which in conventional neural networks require a number of layers consisting of summation units. These terms can be again used as basis functions in the upper layers:

$$y_i = f_i \left(\sum_k w_{ik} f_k \left(\prod_{j=1}^{j_k} w_{kj} x_j \right) \right) \tag{4}$$

For the construction of problem-specific neural models we maintain a population A consisting of M individuals A_i of variable size. Each individual is a neural network represented as a set of sigma-pi neural trees. The algorithm is summarized in Figure 2:

The initial population $A(0)$ is generated with M neural networks created at random. The random initialization includes the type and receptive field of units, the depth of the network, and the values of weights. Then, the fitness values (defined below) of the individual networks are evaluated using a training set of size N. According to their fitness value, the best τ% members of gth generation are selected into the mating pool $B(g)$. The $(g+1)$th generation of size M is produced by applying crossover and mutation operators to the parent networks in the mating pool $B(g)$. New populations are generated until the variance of fitness values falls below a specified limit or the generation number reaches g_{max}.

The crossover operator selects two parents, B_i and B_j and exchanges their subtrees to generate two off-

spring B_i' and B_j'. This is the way how the size, depth and receptive field shape of the network architecture is adapted. The weights are adapted by repeatedly applying a mutation operator to each individual. The mutation operator also changes the index for the input units and the neuron type. For instance, a sigma unit is mutated to a pi unit and vice versa. This flexibility gives the chance of evolving conventional networks as well as networks consisting of any combinations of sigma and pi units.

The fitness of a network is defined as

$$F=\beta F_E+\alpha F_C \tag{5}$$

The F_E term expresses the accuracy penalty caused by the error for the training set. As usual the total sum of errors has been used:

1. Generate initial population $A(0)$ of M networks at random. Set current generation $g \leftarrow 0$.
2. Evaluate fitness values $F_i(g)$ of networks using the training set of N examples.
3. If the termination condition is satisfied, then return the population; otherwise continue with step 4.
4. Select upper τM networks of gth population into the mating pool $B(g)$.
5. Each network in $B(g)$ undergoes a local hillclimbing, resulting in revised mating pool $B(g)$.
6. Create $(g+1)$th population $A(t+1)$ of size M by applying genetic operators to randomly chosen parent networks in $B(g)$.
7. Replace the worst fit network in $A(t+1)$ by the best in $A(t)$.
8. Set $g \leftarrow g+1$ and return to step 2.

Fig. 2 Summary of the procedure for evolving problem-specific sigma-pi neural tree.

$$F_E=\sum_{i=1}^{N}(y_i-f(x_i;w, A_i))^2 \tag{6}$$

The F_C term in the fitness function expresses the complexity penalty of the network.

$$\begin{aligned}
SPNN &\rightarrow (Y_1\ Y_2\ \ldots\ Y_m)\\
Y &\rightarrow (U\ r\ \Theta\ W_1\ W_2\ \ldots\ W_r)\\
U &\rightarrow 'S' \mid 'P'\\
W &\rightarrow (W\ w\ \{Y|X\})\\
X &\rightarrow 'X'\ i\\
\Theta &\rightarrow \Theta_{bin} \mid \Theta_{int} \mid \Theta_{real}\\
\Theta_{bin} &\rightarrow -1 \mid +1\\
\Theta_{int} &\rightarrow -r \mid \ldots \mid 0 \mid \ldots \mid +r\\
\Theta_{real} &\rightarrow R\\
w &\rightarrow \Omega_{bin} \mid \Omega_{int} \mid \Omega_{real}\\
\Omega_{bin} &\rightarrow -1 \mid +1\\
\Omega_{int} &\rightarrow 0 \mid \pm 1 \mid \pm 2 \mid \pm 3 \mid \ldots\\
\Omega_{real} &\rightarrow R\\
r &\rightarrow 1 \mid 2 \mid 3 \mid \ldots\\
i &\rightarrow 1 \mid 2 \mid 3 \mid \ldots \mid n
\end{aligned}$$

Fig. 3 Grammar for generating the sigma-pi neural trees.

The complexity of the network is expressed as the sum of the numbers of weights $W(A)$, units $U(A)$, and layers $L(A)$, i.e.

$$F_C=W(A)+U(A)+L(A) \tag{7}$$

This definition prefers smaller (and shallow) networks to larger (and deep) structures and in this way implements the principle of minimum description length (MDL) [17, 19]. In the experiments we used the following values for parameters β and α:

$$\beta=\frac{1}{m\cdot N},\ \alpha=\frac{1}{N\cdot C_{\max}} \tag{8}$$

where m is the number of outputs. Notice that the complexity term F_C is divided by the number of training examples N multiplied by the possible maximum network size $C_{\max}$. This ensures a small network survives selection only if it achieves a competitive performance to a large network. Otherwise, the evolution may not lead to a desired accuracy by preferring smaller networks which lack the capacity to learn the training set.

Ⅳ. Theoretical Analysis

A higher-order neuron of order k has an input-output relation given by

$$\begin{aligned} y &= f(u), \\ u &= w_0 + \sum_i w_i^{(1)} x_i \\ &\quad + \sum_{i_1}\sum_{i_2} w_{i_1 i_2}^{(2)} x_{i_1} x_{i_2} + \cdots \\ &\quad + \sum_{i_1}\sum_{i_2}\cdots\sum_{i_k} w_{i_1 \cdots i_k}^{(k)} x_{i_1}\cdots x_{i_k} \end{aligned} \tag{9}$$

where all indices $i_1,\ldots,i_m$ in $w_{i_1\cdots i_m}^{(m)}$ are assumed to take different values satisfying $i_1 < i_2 < \cdots < i_m$.

Since the k-th order term consists of a linear weighted sum over k-th order products of inputs, we can rewrite it using pi-units:

$$\begin{aligned} T^{(k)} &= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_k} w_{i_1\cdots i_k}^{(k)} x_{i_1}\cdots x_{i_k} \\ &= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_k} w_{i_1,\ldots,i_k}^{(k)}\ g\left(\prod_{i=i_1}^{i_k} x_i\right) \\ &= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_k} w_{i_1,\ldots,i_k}^{(k)}\ g\left(\prod_{i=i_1}^{i_k} v_i x_i\right) \\ &= \sum_{(i_1, i_2,\ldots,i_k)} w_{i_1,\ldots,i_k}^{(k)} P^{(k)} \end{aligned} \tag{10}$$

where

$$P^{(k)} = P_{i_1, i_2,\ldots,i_k}^{(k)} = g\left(\prod_{i=i_1}^{i_k} v_i x_i\right) \tag{11}$$

assuming

$$g(u) = u \text{ and } v_i = 1 \tag{12}$$

The higher-order terms can be again used as building blocks which are able to capture a high-order correlational structure of the data.

In particular, by building a sigma unit which has as input various higher-order terms, we can construct a higher-order network of sigma-pi units:

$$y_i = f_i(u_i) = f_i\left(\sum_k w_k T^{(k)}\right) \tag{13}$$

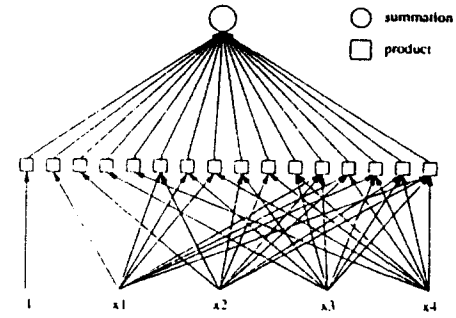

Fig. 4 An order 4 neuron realized as a sigma-pi neural tree.

The problem in using higher-order networks is that the number of terms explodes with the problem size; the number of parameters necessary for specifying an order k neuron is

$$r_k = \sum_{i=0}^{k} {}_nC_i \tag{14}$$

because $w_{i_1\cdots i_m}^{(m)}$ have n Cm components. Here n is the total number of inputs and n Cm are the binomial coefficients. As an example, an order 4 neuron has $2^4 = 16$ parameters as shown in Figure 4.

The experimental results in the following section shows how the genetic search "intelligently" discovers and combines useful terms and eliminates non-essential terms, without exhaustively searching through the entire space.

Ⅴ. Experimental Results

5.1 Benchmark Tests

The effectiveness of the method was studied on parity problems with input size $n = 2, 4, 6, 8$. The parity problem is difficult to solve by perceptrons since the sigma units can not effectively represent the higher order interactions necessary for solving the problem. We attempted to solve the problem using both sigma and pi units, i.e. let the genetic algorithm find a suitable architecture starting with a population of structures initialized with 50% sigma units and 50% pi units.

As generation goes on, sigma units died out and finally most of the best solutions consisted of pi units. This indicates that the method can evolve problem-specific neuron types and topologies starting from random structures. This fact was confirmed by additional control experiments in which only sigma units were allowed. As results in Tables 1 and 2 show, if pi units (P) are used, the performance in accuracy as well as in complexity reduction was consistently improved as to the case when sigma units (S) alone are used. Throughout all the experiments, training data consisted of 2^n examples for problem size n. For each n, the population size was $M = 100n/2$ and the maximum generation was set $g_{max} = 10n$.

Table 1. Comparison of network size

n type	units	weights
2 S	4.0 ± 0.4	12.9 ± 0.9
S/P	2.2 ± 0.1	6.5 ± 0.2
4 S	17.9 ± 0.7	55.0 ± 1.0
S/P	3.6 ± 0.4	14.8 ± 1.7
6 S	30.2 ± 3.5	161.4 ± 17.9
S/P	8.9 ± 0.5	40.2 ± 2.2
8 S	65.6 ± 5.4	414.1 ± 31.5
S/P	21.0 ± 0.9	106.8 ± 2.2

Table 2. Performance comparison

n type	accuracy	generations
2 S	95.0 ± 0.9	12.5 ± 0.8
S/P	100.0 ± 0.0	2.9 ± 0.2
4 S	81.9 ± 0.1	40.0 ± 0.0
S/P	100.0 ± 0.0	8.5 ± 1.3
6 S	89.5 ± 3.4	46.2 ± 1.1
S/P	98.4 ± 0.8	35.8 ± 1.5
8 S	86.4 ± 2.1	63.4 ± 3.0
S/P	98.1 ± 1.8	56.2 ± 4.2

5.2 Water Pollution Forecasting

The task involves an environmental system in the Sangamon River, Illinois. Our objective is to predict nitrate levels a week ahead in the watersheds of the river from the previous values. The predicted values can be used for monitoring the progress of water pollution. The original study in the literature also aims at giving some indication of the biochemical and physical relationships among the variables and of the controllability of the system.

The training data is based on the nitrate-nitrogen levels during the period from January 1, 1970, to December 31, 1971. The sampling interval is one week. The training set is generated from this series using a time lag of four. The initial population was created by randomly generating neural trees with a branching factor up to four and maximum depth of four also. 50% of the units were randomly chosen as sigma units and the rest as pi units. During evolution, however, sigma units usually survived more often than the pi units did. Using the weight interval of [−10, +10], the best solutions after 300 generations contained on average 10 hidden units in three layers.

The mean square error for the training data was 0104. To test the predictive accuracy of the evolved models, unseen data for the same watershed for 1972 was used. The measured and predicted outputs for this test data are plotted in Figures 5 and 6. As can be seen in the figure, the nitrate levels for the following week was predicted relatively well, considering the

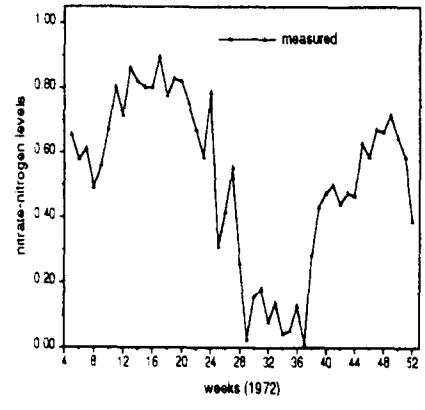

Fig. 5 The test data for the water pollution problem.

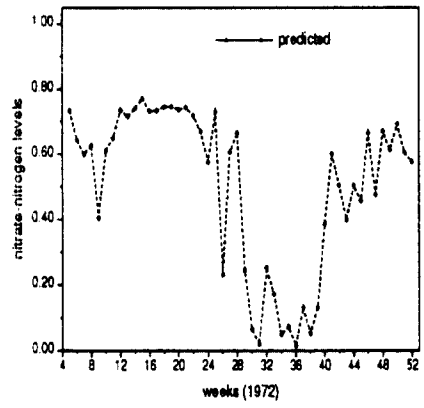

Fig. 6 One-step ahead prediction for the water pollution problem.

sparseness of the training data. This result is comparable to that obtained by the well-engineered GMDH algorithm [3, 18].

5.3 Laser Intensity Prediction

Evolutionary learning of sigma-pi trees has also been successfully applied to a real-world time-series prediction problem. The data came from a far-infrared laser [14]. The training examples were formed from the time series $\{x_t\}$ by specifying groups with respect to a time index t. The input pattern was assigned $(x_{t-3}, x_{t-2}, x_{t-1})$, the desired output was x_t.

The next 500 steps are shown in Figure 7 and its

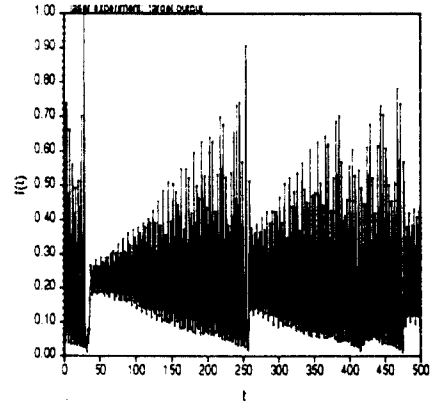

Fig. 7 The test data for the laser intensity

one-step ahead prediction result is depicted in Figure 8. It is remarkable that though some peaks of the time series were not fit completely during the learning phase, the generalization performance on the unseen time series is relatively good. This is the desired effect when using the complexity penalty during fitness evaluation.

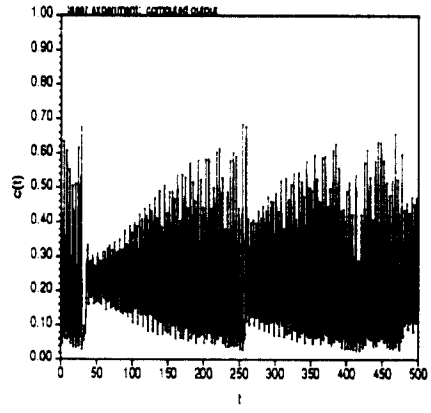

Fig. 8 One-step ahead perdiction performance for the laser data.

VI. Conclusion

From the engineering point of view, various schemes for combining genetic algorithms and neural networks have been proposed and tested in recent years, including preprocessing of data for neural network application, determination of neural network weights, and optimization of the network topology.

In this paper we have presented an evolutionary method for constructing higher-order neural networks. The method uses a tree encoding scheme in which the node type, weight, size and topology of the network are dynamically adapted by genetic operators. We demonstrate the effectiveness of the genetic algorithm on the synthesis of sigma-pi neural networks which are useful for building higher-order terms. In particular, we have shown that the method is effective for learning to solve classification and prediction problems.

In contrast to conventional learning algorithms for neural networks, the presented method makes relatively few assumptions on the architecture space in which the search is performed. Thus it may be used to design and train other types of neural network models. The potential for evolving novel neural networks that are customized for specific applications is one of the most interesting properties of evolutionary learning algorithms.

References

1. S. Amari. Dualistic geometry of the manifold of higher-order neurons, *Neural Networks*, vol. 4, pp. 443-451, 1991.
2. T. B ck and H. -P. Schwefel. An overview of evolutionary algorithms for parameter optimization, *Evolutionary Computation*, vol. 1, pp. 1-23, 1993.
3. J. J. Duffy and M. A. Franklin, A learning identification algorithm and its application to an environmental system, *IEEE Trans. on Sys. Man and Cyb.*, 5(2):226-240, 1975.
4. R. Durbin and D. E. Rumelhart. Product units:a computationally powerful and biologically plausible extension to backpropagation networks, *Neural Computation*, vol. 1, pp. 133-142, 1989.
5. C. L. Giles and T. Maxwell. Learning, invariance, and generalization in high-order neural networks, *Applied Optics*, vol. 26, no. 23, pp. 4972-4978, 1987.
6. D. E. Goldberg. *Genetic algorithms in search, optimization machine learning.* Addison Wesley, 1989.
7. J. R. Koza, *Genetic programming:on the programming of computers by means of natural selection.* MIT Press, 1992.
8. G. F. Miller, P. M. Todd and S. U. Hegde. Designing neural networks using genetic algorithms, in *Proc. Third Int. Conf. on Genetic Algorithms (ICGA-89)*, Morgan Kaufmann, 1989, pp. 379-384.
9. M. Minsky and S. Papert. *Perceptrons:an introduction to computational geometry.* MIT Press, 1969 and 1988.
10. D. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," *in Proc. Int. Joint Conf. Artificial Intelligence*, 1989.
11. H. M hlenbein. Limitations of muti-layer perceptron networks-steps towards genetic neural networks, *Parallel Computing*, vol. 14, pp. 249-260, 1990.
12. H. M lenbein and J. indermann, "The dynamics of evolution and learning-Towards genetic neural networks," in *Connectionism in Perspective*, R. Pfeifer *et al.* (eds.) Elsevier, 1989, pp. 173-197.
13. J. DSchaffer, D. Whitley, and L. J. Eshelman, "Combinations of genetic algorithms and neural networks:A survey of the state of the art," in *Proc. Int. Workshop on Combinations of Genetic Algorithms and Neural Networks*, IEEE, 1992, pp. 1-37.
14. A. S. Weigend, N. A. Gershenfeld, Eds. *Time Series Prediction*, Addison-Wesley, 1993.
15. D. Whitley, T. Starkweather and C. Bogart. Genetic algorithms and neural networks:optimizing connections and connectivity, *Parallel Computing*, vol. 14, pp. 347-361, 1990.
16. B. T. Zhang and H. M hlenbein, Evolving optimal neural networks using genetic algorithms with Occam's razor, *Complex Systems*, 7(3):199-220, 1993.
17. B. T. Zhang and H. Mühlenbein, Balancing accuracy and parsimony in genetic programming, *Evolutionary Computation*, 3(1):17-38, 1995.
18. B. T. Zhang and H. M hlenbein, Water pollution prediction with evolutionary neural trees, in *Proc. IJCAI-95 Workshop on AI and the Environment*, Montreal, Canada, August 1995.
19. B. T. Zhang and H. M hlenbein, MDL based fitness functions for learning parsimonious programs, in *Proc. AAAI 1995 Fall Symposium on Genetic Programming*, AAAI Press, 1995, pp. 122-126.

장 병 탁(Byoung-Tak Zhang) 정회원
1986년 : 서울대학교 컴퓨터공학과 졸업(학사)
1988년 : 서울대학교 대학원 컴퓨터공학과 졸업(석사)
1992년 : 독일 Bonn 대학교 전산학과 졸업(박사)
1988년~1992년 : Bonn대학교 AI Lab. 연구원
1992년~1995년 : 독일국립전산학연구소(GMD) 연구원
1995년~현재 : 건국대학교 컴퓨터공학과 조교수
※주관심분야 : 신경망, 유전알고리즘, 기계학습, 인공지능