# 점진적 패턴 선택에 의한 다층 퍼셉트론의 효율적 구성 및 학습

장 병 탁†


## 요 약

본 논문에서는 주어진 문제를 해결하기 위해 사용될 최적의 다층 퍼셉트론을 구성하기 위한 하나의 점진적 학습 방법을 제시한다. 고정된 크기의 트레이닝 패턴 집합을 반복적으로 사용하는 기존의 알고리즘들과는 달리, 제시되는 방법에서는 학습 패턴의 수를 점차 증가시키면서 전체 데이터를 학습하기 위해 필요하고도 충분한 은닉뉴런의 수를 찾는다. 이와 같이 신경망 크기의 최적화에 학습 패턴을 점차적으로 선택하여 늘려나감으로써 일반화 능력과 학습 속도가 기존의 방법에서보다 향상됨을 필기체 숫자인식 문제에 있어서 실험적으로 보여준다.


# Efficient Construction and Training of Multilayer Perceptrons by Incremental Pattern Selection

Byoung-Tak Zhang†


## ABSTRACT

An incremental learning algorithm is presented that constructs a multilayer perceptron whose size is optimal for solving a given problem. Unlike conventional algorithms in which a fixed size training set is processed repeatedly, the method uses an increasing number of critical examples to find a necessary and sufficient number of hidden units for learning the entire data. Experimental results in hand-writtern digit recognition shows that the network size optimization combined with incremental pattern selection generalizes significantly better and converges faster than conventional methods.


## 1. Introduction

Any continuous multivariate function on a bound domain can be approximated by a multilayer feedforward neural network with a hidden layer of sigmoid units to any desired degree of accuracy [11, 13]. However, this existence theorem does not provide any hint on how many hidden units are necessary for solving a particular problem. The method of error back-propagation [20], one of the standard techniques for training multilayer perceptrons, is limited to optimizing weights only and suffers from slow convergence.

The training speed and generalization performance of back-propagation networks are affected to large extent by the network architecture or size [1, 9, 22]. If the network contains too small a number of hidden

*본 논문은 한국전자통신연구소 위탁연구에 의하여 일부 지원되었음(과제번호:96144)

† 정 회 원:건국대학교 컴퓨터공학과

논문접수:1996년 5월 14일, 심사완료:1996년 5월 23일

units, the training will never converge. On the other hand, if the nework is too large, the generalization performance of the trained network will be generally poor. Some theoretical works give bounds on network size for learning a class of problems [6]. However, most of these studies are worst case analysis based on one or more unrealistic assumptions, and not very helpful in practice.

Recently, several learning algorithms have been proposed to construct optimal feedforward networks for specific applications [21]. All constructive methods begin with a small network and introduce new hidden units and/or connections on demand. Some of them try to find a compact distributed representation, where the optimization is done with respect to the number of units in the hidden layer [4, 12]. Others construct more or less localized representations by building a deep or flat-but-wide network [8, 19]. Both approaches have their own strengths and weaknesses. Each of these methods uses all the given data for network construction and training.

In this paper we present a constructive learning algorithm that uses a small subset of given examples to construct a feedforward network with an optimal number of hidden units. The solution obtained has the feature of local approximation [22], similar in spirit to that of radial hasis function networks [18] and $k$-nearest neighbor classifiers [10], without giving up the beneficial global approximation ability of the multilayer networks of sigmoid units. Furthermore the algorithm does not need a second data set for testing the generalization performance of the networks to decide the stopping of the training, because the generalization accuracy of the network increases almost monotonically once an optimally sized architecture is found.

The method is based on the observation that not all available examples are critical to realize the desired mapping. In classification tasks, for instance, the critical examples are those that lie closest to the separating hyperplances [23]. Previous studies have shown that a network trained only on these "border" patterns generalizes substantially better than one trained on the same number of random patterns [2, 14].

This observation was used in [28] to devise an incremental learning procedure that starts with a small set of seed examples and expands the training set selectively after training. In the previsous work [24, 25] we have shown that this selective learning, called SEL can find a minimal training set that is sufficient for learning the entire data. We have also shown that the SEL learning algorithm achieves better convergence and generalization performance than non-incremental learning with the original data, provided the data set is large enough to have all the border patterns.

The current algorithm is an extension of the selective incremental learning that also optimizes the network size during training.

After a brief discussion of our constructive approach in Section 2, the algorithm is described in Section 3. The performance of the algorithm is studied in Section 4 in the context of digit recognition. Section 5 discusses the implications of this work.

## 2. Learning in Layered Networks

Multilayer feedforward neural networks, or multilayer perceptrons are networks of units organized in layers. The external inputs are presented in the input layer which is fed forward via one or more layers of hiden units to the output layer. There is no direct connection between units in the same layer. The activation value of unit $i$ is influenced by the activations $a_j$ of incoming units $j$ and the real-valued weights $w_{ij}$ from $j$th to $i$th unit. The net input of unit $i$ is computed by

$$net_i = \sum_{j \in R(i)} w_{ij} a_j + \theta_i, \tag{1}$$

where $R(i)$ is the receptive field of unit $i$. The bias $\theta_i$ is usually considered as a weight $w_{i0}$ connected to an extra unit whose activation value is always 1. The

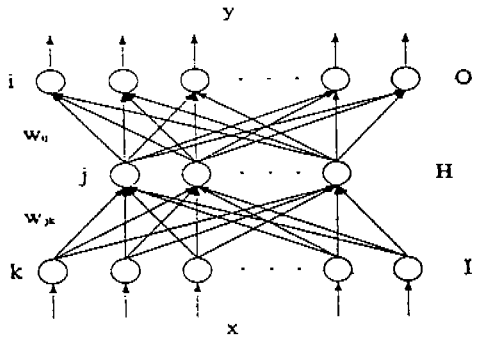
output value of unit $i$ is determined by a nonlinear transfer function $f$. A commonly used output function is the sigmoid nonlinearity

$$f(net_i)=\frac{1}{1+e^{-net_i}} \tag{2}$$

For the case of a two layer perceptron (see Fig. 1), the $i$th output of network, $f_i$, $i=1,\cdots,O$, is a nonlinear function of inputs $x_k$:

$$f_i(x;w)=f_i\left(\sum_{j=0}^{H} w_{ij}\, f_j\left(\sum_{k=0}^{I} w_{jk}x_k\right)\right) \tag{3}$$

where $I$, $H$ and $O$ are the number of input, hidden, and output units, respectively. Each network configuration $w$ implements a mapping from an input $x \in X\subset R^I$ to an $y\in Y\subset R^O$. We denote this mapping by $y=f(x;w)$, $f: R^I\times W\rightarrow R^O$.




(Fig. 1) A fully connected feedforward network consisting of $I$ input, $O$ output and $H$ hidden units. For simplicity, bias weights are not shown.

The set of all possible weight vectors $w$ constitutes the configuration space $W\subset R^d$, where $d$ is the total number of weights of the network architecture.

The networks are used to learn an unknown relation $F$. A set of $N$ input-output pairs is given as a training set:

$$D_N=\{(x_c, y_c)\}_{c=1}^{N} \tag{4}$$

where $x_c\in X\subset R^I$, and $y_c\in Y\subset R^O$. The relation $F$ can be generally described by the probability density function defined over the space of input-output pairs $X\times Y\subset R^{I+O}$:

$$P_F(x, y)=P_F(x)P_F(y\mid x) \tag{5}$$

where $P_F(x)$ defines the region of interest in the input space and $P_F(y\mid x)$ describes the functional or statistical relation between the inputs and the outputs.

Learning the training set by a network is formulated as an optimization problem. One defines a quality measure of the approximation of the desired relation $F$ by the mapping $f(x:w)$ realized by the network.

A commonly used measusre is the additive error functional

$$E(D_N\mid w)=\sum_{c=1}^{N} e(y_c\mid x_c, w) \tag{6}$$

where $e(y_c\mid x_c, w)$ is the squared error between the desired output $y_c$ and the actual output

$$e(y_c\mid x_c, w)=\sum_{i=1}^{O}\left(y_{ci}-f_i(x_c;w)\right)^2 \tag{7}$$

where $y_{ci}$ denotes the $i$th component of vector $y_c$.

Usually the problem is formulated as finding a set of parameters $w$ that minimizes the total error on the training set $D_N$ of $N$ independent examples:

$$w^*=\arg\min_{w\in W} E(D_N\mid w) \tag{8}$$

Much research has been done to increase the speed of this optimization problme (see [5] for a recent review of various methods).

Note, however, that the eventual goal of learning is generalization on unseen data; that is, the performance of a network should be measured on the whole input space by generalization error, defined as

$$E(w)=\int e(y\mid x, w)\,dp(x, y) \tag{9}$$

where the joint probability distribution $p(x, y)=p(y\mid$

$x)p(x)$ is unknown and the only avaiable information is contained in the training set. The minimization of the training error does not necessarily imply optimal generalization. In general, the network size should also be optimized to be sufficiently large to realize the target mapping and, at the same time, small enough to avoid overfitting to the training examples [9, 22]. Usually one uses a second data set $C$, $C \cap D_N = \phi$, for the evaluation of generalization performance. The new problem is formulated as finding an architecture and its weights that minimize the error on $D_N$:

$$w^*, A^* = \arg\min_{w, A} E(D_N \mid w, A) \tag{10}$$

where the test set $C$ is used to determine the termination of the optimization process.

## 3. The Algorithm

The learning proceeds as follows. We choose a small initial training set from a given data. The rest of the data is called candidates. The network is initialized with a small number of hidden units. One hidden unit and two seed examples are typical choice in the experiments. The network is trained on the training set by the back-propagation algorithm or possibly by any ther weight modification rule. If the training converges, we expand the training set by selecting $\lambda$ candidate examples. Otherwise, we expand the network by introducing $\nu$ hidden units and reinitializing the connection weights. Training of weights, selection of examples, and introduction of hidden units with retraining is repeated until an acceptable generalization is achieved on the candidate data set.

A more detailed description follows. The algorithm is called SElective Learning with Flexible neural architectures, or simply SELF. A theoretical discussion of the algorithm can be found in [27].

Given a data set $B$, the training set $D$ is initialized to contain a small number of seed examples chosen from $B$. The rest of $B$ is used as a candidate set $C$. During learning, $D$ is increased by selecting examples form $C$ We use a superscript $s$, as in $D^{(s)}$, to denote the $s$th training set. The network architecture is initially small and grows during learning. The symbol $A^{(g)}$ is used to denote the architecture of a network after the $g$th growing step.

The weights of the network are initialized randomly with values from the interval $-\omega \le w_{ij} \le +\omega$. The initial network $A^{(0)}$ is trained with the training set $D^{(0)}$. The trained network is used to expand the training set for the next generation, i.e. $D^{(1)}$, which are again used to find and train the network $A^{(g)}$, where $A^{(g)}$ is of the size same or larger than $A^{(0)}$. In this way, the trained network $A^{(g)}$ and the training set $A^{(s)}$ at time $s$ cooperate with each other, where the indices $g$ and $s$ do not necessarily correspond. For each $s$, the following conditions are always satisfied

$$\begin{aligned} &D^{(s)} \cup C^{(s)} = B \\ &D^{(s)} \cap C^{(s)} = \phi \\ &D^{(s)} \subset D^{(s+1)} \end{aligned} \tag{11}$$

and each network growing step satisfies the condition

$$A^{(g)} \subset A^{(g+1)} \tag{12}$$

In the training phase, the connection weights of the network are updated using the examples in the training set. If we denote by $w^{(s, g, t)}$ the weight vector of the network $A^{(g)}$ for the $t$-th sweep through the training set $D^{(s)}$, the weights are modified by

$$w^{(s, g, t+1)} = w^{(s, g, t)} + \Delta w^{(s, g, t)} \tag{13}$$

$$\begin{aligned} \Delta w^{(s, g, t)} = &-\varepsilon \nabla E_s|_{w = w^{(s, g, t)}} \\ &+ \Delta w^{(s, g, t-1)} \end{aligned} \tag{14}$$

where $E_s$ is the total sum of the errors for $D^{(s)}$

$$E_s = E(D^{(s)} \mid w^{(i, g, t)}, A^{(g)})$$

$$= \sum_{m=1}^{N_s} (y_m - f(x_m; w^{(s,g,t)}, A^{(g)}))^2 \tag{15}$$

and the error gradient $\nabla E_s|_{w=w^{(s,g,t)}}$ is approximated by a back-propagation procedure [20]. In equation (14), $\varepsilon$ and $\gamma_l$ are the step size and the momentum factor, respectively.

At every $\Delta l$ epochs we check the convergence of the error minimization. If the total error for the current training set is reduced to a specified error tolerance level,

$$E(D^{(s)} \mid w^{(s,g,t)}, A^{(g)}) \leq \varepsilon_g \tag{16}$$

the training process terminates and the training set is expanded. We define the error tolerance value as

$$\varepsilon_g = \frac{1}{\tau}\{(I+1) \cdot H_g + (H_g+1) \cdot O\} \tag{17}$$

where $I$, $O$ and $H_g$ are the number of input, output and hidden units of network $A^{(g)}$. The constant tau $\tau$ determines the error sensitivity per connection.

In the selection phase, the generalization accuracy of the current network is tested on the original data, $B = D^{(s)} \cup C^{(s)}$:

$$G_s = \frac{1}{N} \sum_{(x_q, y_q) \in B} \Theta(y_q, f(x_q; w^{(s,g,t)} A^{(g)})) \tag{18}$$

where the function $\Theta(\cdot,\cdot)$ is some measure of correctness. For classification problems, such as digit recognition, it is an indicator function:

$$\Theta(y_q, f(x_q; w^{(s,g,t)}, A^{(g)})) = \begin{cases} 1 & \text{if } y_{qi} = f_i(x_q; w^{(s,g,t)}, A^{(g)}) \text{ for } \forall i \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

If $G_s$ exceeds the desired performance level $\ell$, say 99%, then the entire algorithm stops. If $C^{(s)}$ is empty, the alorithm also stops. Notice that halting with a nonempty $C^{(s)}$ means the network has generalized correctly to the candidate data set. Otherwise, the criticality with respect to the current model $w$ is computed.

The criticality $(x_c, y_c)$ of an example is defined as

$$e_w(x_c) = \frac{1}{\dim(y_c)} (y_m - f(x_m; w^{(s,g,t)}, A^{(g)}))^2 \tag{20}$$

which has a value $0 \leq e_w(x_c) \leq 1$ if the sigmoid activation function is used at the output layer. Then the training set is increased by selecting $\lambda$ candidate examples, $(x_c, y_c)$, which are most critical:

$$D^{(s+1)} = D^{(s)} \cup \{(x_c, y_c)\}$$
$$C^{(s+1)} = C^{(s)} - \{(x_c, y_c)\} \tag{21}$$

In case of $|C^{(s)}| < \lambda$, all the remaining candidate examples are selected into $D^{(s+1)}$ Using the expanded training set, the next cycle of training and selection is done.

If Eq. (16) is not satisfied, then check if the model trapped in a local minimum. For the detection of local minima, a time window is used to consider the change in errors during the last $\Delta l$ epochs

$$\Delta E(t) = E(D^{(s)} \mid w^{(s,g,t-\Delta t)}, A^{(g)}) - E(D^{(s)} \mid w^{(s,g,t)}, A^{(g)}) \tag{22}$$

For a robust detection we extend the time window to the entire training time from the start by having a temporally discounted influence of earlier error changes

$$\Delta E_{sum}^{(s,g)}(t) = \Delta E^{(t)} + \frac{1}{2} \Delta E_{sum}^{(s,g)}(t - \Delta t) \tag{23}$$

This quantity is normalized to an average error $\Delta E_{sum}^{(s,g)}(t)$ for a training example in each epoch:

$$\Delta E_{avg}^{(s,g)}(t) = \frac{\Delta E_{sum}^{(s,g)}(t)}{N_s \cdot O} \tag{24}$$

Then the network grows if the total error is larger than the error tolerance $\varepsilon_g$ defined in (16) and the average error change is smaller than the specified threshold $\rho$, i.e.

$$[E(D^{(s)} \mid w^{(s,g,t)}, A^{(g)}) > \varepsilon_g] \wedge [\Delta E_{avg}^{(s,g)}(t) < \rho] \tag{25}$$

Network growing is performed by introducing $u$ new units to the hidden layer:

$$\mathcal{H}^{(g+1)} = \mathcal{H}^{(g)} \cup \{H_g + 1, \ldots, H_g + u\} \qquad H_{g+1} = H_g + u. \tag{26}$$

where $\mathcal{H}^{(g)}$ denotes the index set of hidden units in $A^{(g)}$. The new hidden units have full connectivity with all input and output units.

The values of new connections can be initialized in several ways.

Two strategies are studied in the simulations. The first one is to reinitialize all the weights, including the existing ones.

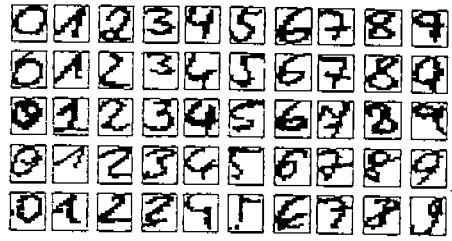
An alternative approach is to keep the trained weights unchanged and to initialize new connections with values proportional to the average of the weights in the existing connections of the same weight layer. The first strategy guarantess an escape from a local minimum and hence leads to a minimal network size. We will use this strategy for classification problems where the input and output space are discrete.

The latter strategy will ensure an effective escape from the local minimum without loss of information learned up to the growing point. This is used for continuous-valued problems.

## 4. Hand-Written Digit Recognition

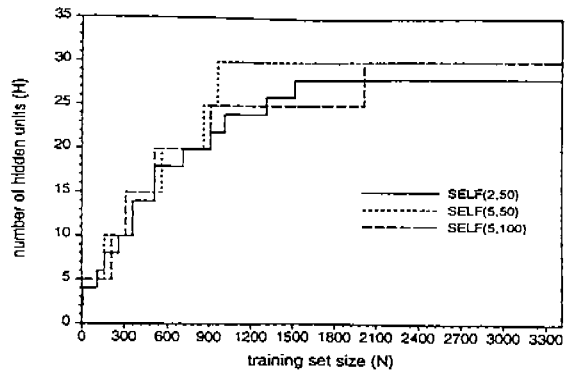
The performance of the SELF algorithm in its learning speed and generalization accuracy is studied in the context of hand-written digit recognition.

We collected 6800 digit patterns written by 10 persons. Each pattern consists of $15 \times 10$ bitmap. Some of the bitmap patterns are shown in Fig. 2. One half of the examples were used for training the network and the other half for testing the generalization performance of the trained network.

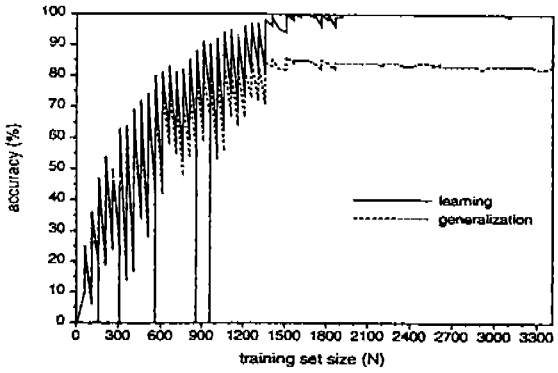


(Fig. 2) Some digit patterns for training and testing.

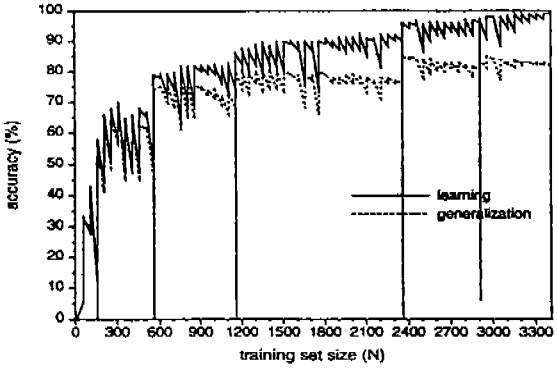
We used networksize increment $u = 5$ and training set size increment $\lambda = 50$. i.e. the algorithm SELF(5, 50). The initial network contained five hidden units and the weights were initialized with random values from the interval $[-0.1, +0.1]$. 10 digits of 0 to 9 were randomly chosen from the data set to initialize the training set. In each adaptation phase the network was trained for each training set until the total sum of errors for the training set dropped below $\varepsilon_g = \frac{1}{50} K_g$, where $K_g$ is the total number of adjustable weights in




(Fig. 3) Network size growth as a function of training set size. Optimization of network size is relatively robust against the parameters $u$ and $\lambda$.




(Fig. 4) Learning and generalization performance for the SELF(5, 50) algorithm. The training set size for which the accuracy is almost zero indicates the time when network growing takes place. Optimal size network is constructed using about 1000 examples out of 3400.




(Fig. 5) Learning and generalization performance of SELF (5, 50RANDOM) algorithm in which examples are chosen randomly. Until an optimal network size is found, this algorithm uses about 3000 examples. This training set size is approximately three times larger than that of SELF(5, 50)

the network of $g$th growing stage.

The learning trial converged to a network with 30 hidden units, i.e. 150-30-10 architecture, achieving approximately 85% generalization. The evolution of network size and performance are shown in Figs. 3 and 4 as a function of the training set size. The performance was measusred at each initial and final training epoch for each training set.

In Fig. 4, the points where the accuracy goes to almost zero indicates the netwrok growth step. This is because we reinitialized the entire weights at each growing step to ensure the minimum network size. Notice that after reaching the optimal size network there is no significant improvement in the generalization performance. This indicates the method has already found a critical subset of the given data.

The effectiveness the algorithm was tested by performing control experiments. We varied the parameter values $u$ and $\lambda$, resulting in two variations SELF(2, 50) and SELF(5, 100) of SELF(5, 50). The results are shown in Fig. 3, where the network size is depicted as a function of the training set size.

The algorithm SELF(2, 50) converged to a network structure with 28 hidden units, two less than the other variants. This suggests using a small $u$ parameter allows a finer-grained optimization of network size than a large $u$. However, SELF(2, 50) was the slowest of the three algorithms in terms of training time. Algorithm SELF(5, 100) was slightly faster than SELF(5, 50), but the difference was smaller than that between SELF(5, 50) and SELF(2, 50). There was no significant difference in the final generalization performance.

The SELF(5, 50) algorithm was compared with two non-constructive algorithms: the plain back-propagation (BP) and SEL. The algorithm SEL is the same as SELF, except that SEL uses a fixed number of hidden units [25]. We use SEL($H$, $\lambda$) to denote a selective learning with $H$ hidden units and $\lambda$ examples chosen in each selection step. A 150-30-10 architecture was used for BP and SEL.

As can be expected, SELF was the most expensive algorithm of the three due to the additional costs for network size optimization. However, SELF converges faster and more robust than BP and SEL once it finds an optimal network size. On the other hand, SELF is the most robust algorithm of the three. SELF was always able to converge, while SEL did

not always converge and BP usually did not. Among the non-constructive algorithms, the selective incremental learning SEL was generally superior to BP both in convergence and generalization performance.

The effect of example selection during network growing was studied by selecting examples randomly, instead of using the criticality measusre (Eqn. 20). Comparing Figs. 4 and 5 we see that network size optimization combined with active example selection generalizes better and converges much faster than with random selection.

## 5. Conclusion

We have shown that an optimal size of multilayer perceptron can be constructed efficiently by selecting examples intelligently, instead of randomly or without any selection. One of the most useful characteristics of the SELF alogorithm is that there is no need for the user to decide in advance on the exact complexities of the network and the training set. The incremental data selection method allows the user to use all the available data without having to worry about the size of the data. Indeed, the superiority of this algorithm to other methods becomes clearer as available data becomes more abundant.

Simulation results show that for a small data set, the selective construction algorithm maximizes the generalization performance by finding a minimal network size for learning the data. For a large redundant data set, the method converges faster than the back-propagation network with an optimal number of hidden units, without sacrificing the final generalization accuracy. The enhancement of learning speed is proportional to the rate of data reduction. In general, the larger the given data set, the better the relative performance of the SELF algorithm compared with the plain back-propagation or other constructive algorithms.

The final number of hidden units is a function of the network growth parameter $u$ which affects the total learning time. Using a small $u$ leads to a fine-grained network size with high optimization costs, while a large $u$ finds a rough size fast. Thus the balance between granularity and costs of optimization can be done by varying $u$. We observe, however, no significant difference in the final generalization performance unless the difference of $u$ values are big. It can be recommended to try first with a large $u$ to find a rough size and then perform a second run to find a smaller size, provided one-shot optimization is not compelling.

This is however different from the usual trial-and-error methods, where one trial does not give much insight into the minimal network size for learning the given examples. In contrast, after the first run of the SELF algorithm one can be statistically sure that the minimum lies within the interval $[H_g - u + 1,\ H_g]$, where $H_g$ is the number of hidden units for the final growing step in the run.

Another advantage of the SELF learning algorithm is that it helps to decide how good the given data is. If the generalization performance of the trained network is poor, one can conclude that the data is lacking in critical information since the algorithm used an optimal network for the given data set.

## REFERENCES

[1] Y. S. Abu-Mostafa, "The Vapnik-Chervonenkis dimension: information versus complexity in learning," *Neural Computation*, vol. 1, pp. 312-317, 1989.

[2] S. Ahmad and G. Tesauro, "Scaling and generalization in neural networks: a case study," in *Proc. 1988 Connectionist Models Summer School*, Morgan Kaufmann, 1989, pp. 3-10.

[3] S. Amari, N. Fujita, and S. Shinomoto, "Four types of learning curves," *Neural Computation*, vol. 4, pp. 605-618, 1992.

[4] T. Ash, "Dynamic node creation in back-propagation networks," *Connection Science*, vol. 1, pp. 365-375, 1989.

[5] R. Battiti, "First-and second-order methods for learning between steepest descent and Netwton's method," *Neural Computation*, vol. 4, pp. 141-166, 1992.

[6] E. B. Baum and D. Haussler, "What size net gives valid generalization?," in *Advances in Neural Information Processing 1*, D. S. Touretzky, Ed. Morgan Kaufmann, 1989, pp. 81-90.

[7] J. Denker *et al.*, "Large automatic learning, rule extraction, and generalization," *Complex Systems*, vol. 1, pp. 877-922, 1987.

[8] S. E. Fahlman and C. Hebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing 2*, D. S. Touretzky, Ed. Morgan Kaufmann, 1990, pp. 524-532.

[9] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Computation*, vol. 4, pp. 1-58, 1992.

[10] P. E. Hart, "The condensed nearest neighbor rule," *IEEE Trans. Inform. Theory*, vol. 14, pp. 515-516, 1968.

[11] K. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, vol. 2, pp. 183-192, 1989.

[12] Y. Hirose, K. Yamashita, and S. Hijiya, "Back-propagation algorithm which varies the number of hidden units," *Neural Networks*, vol. 4, pp. 61-66, 1991.

[13] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359-366, 1989.

[14] K. A. Huyser and M. A. Horowitz, "Generalization in connectionist networks that realize Boolean functions," in *Proc. 1988 Connectionist Models Summer School*, Morgan Kaufmann, 1989, pp. 191-200.

[15] J. N. Hwang *et al.*, "Query-based learning applied to partially trained multilayer perceptrons," *IEEE Trans. Neural Networks*, vol. 2, pp. 131-136, 1991.

[16] Y. LeCun, "Generalization and network design strategies," CRG-TR-89-4, Dept. of Computer Science, University of Toronto, 1989.

[17] M. Plutowski and H. White, "Selecting concise training sets from clean data," *IEEE Trans. Neural Networks*, vol. 4, no. 2, pp. 305-318, 1993.

[18] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proceedings of the IEEE*, vol. 78, pp. 1481-1497, 1990.

[19] A. S. Refenes and S. Vithlani, "Constructive learning by specialization," in *Artificial Neural Networks: Proc. Int. Conf. on Artificial Neural Networks*(ICANN-91), Vol. I, T. Kohonen *et al.*, Ed. North-Holland, 1991, pp. 923-929.

[20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*, Vol. I, D. E. Rumelhart and J. L. McClelland, Eds. MIT Press, 1986, pp. 318-362.

[21] F. Śmieja, "Neural network constructive algorithms : Trading generalization for learning efficiency?," *Circuits, Systems and Signal Processing*, vol. 12, pp. 331-374, 1993.

[22] V. Vapnik, "Principles of risk minimization for learning theory," in *Advances in Neural Information Processing 4*, J. E. Moody *et al.*, Ed. Morgan Kaufmann, 1992. pp. 831-838.

[23] D. J. Volper and S. E. Hampson, "Learning and using specific instances," *Biological Cybernetics*, vol. 57, pp. 57-71, 1987.

[24] B. T. Zhang, *Learning by Genetic Neural Evolution*. Ph.D. Thesis in German, ISBN 3-929037-16-5, Bonn/Sankt Augustin : Infix-Verlag, 1992. Also available as Informatik Berichte No. 93, Institut für Infromatik I, Universität Bonn, D-53117 Bonn, 1992.

[25] B. T. Zhang, "Selecting a critical subset of given examples during learning," in *Artificial Neural Networks: Proc. Int. Conf. Artificial Neural Networks(ICANN-94)*, Springer-Verlag, pp. 517-520, 1994.

[26] B. T. Zhang, "Accelerated learning by active example selection," *International Journal of Neural Systems*, 5(1):67-75, 1994.

[27] B. T. Zhang, "Self-development learning: Constructing optimal size neural networks via incremental data selection," *GMD-Arbeitspapiers*, No. 768, German National Research Center for Computer Science (GMD) July 1993.

[28] B. T. Zhang, "An incremental learning algorithm that optimizes network size and sample size in one trial," in *Proc. IEEE Int. Conf. on Neural Networks(ICNN'94)*, IEEE Computer Society Press, pp. 215-220, 1994.

[29] B. T. Zhang, "Balancing accuracy and parsimony in genetic programming," *Evolutionary Computation*, 3(1):17-38, 1995.

장 병 탁

1986년 서울대학교 컴퓨터공학과 졸업(학사)
1988년 서울대학교 대학원 컴퓨터공학과 졸업(석사)
1992년 독일 Bonn 대학교 전산학과 졸업(박사)
1988년~1992년 Bonn 대학교 AI Lab. 연구원
1992년~1995년 독일국립전산학연구소(GMD) 연구원
1995년~현재 건국대학교 컴퓨터공학과 조교수
관심분야: 신경망, 유전알고리즘, 기계학습, 인공지능