An Incremental Learning Algorithm That Optimizes Network Size and Sample Size in One Trial

Byoung-Tak Zhang
German National Research Center for Computer Science (GMD)
Schloss Birlinghoven, D-53757 Sankt Augustin, Germany
E-mail: zhang@gmd.de

Abstract— **A constructive learning algorithm is described that builds a feedforward neural network with an optimal number of hidden units to balance convergence and generalization. The method starts with a small training set and a small network, and expands the training set incrementally after training. If the training does not converge, the network grows incrementally to increase its learning capacity. This process, called selective learning with flexible neural architectures (SELF), results in a construction of an optimal size network for learning all the given data using only a minimal subset of them. We show that the network size optimization combined with active example selection generalizes significantly better and converges faster than conventional methods.**

I. Introduction

Feedforward neural networks with a hidden layer of sigmoid units are capable of approximating any continuous multivariate function 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 a large extent by the network architecture or size [1, 9, 22]. If the network contains too small a number of hidden 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 basis 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 hyperplanes [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 [27] 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. The algorithm is described in section II. The performance of the algorithm is studied in Section III in the context of a benchmark problem and digit recognition. Section IV discusses the implications of the work.

II. Algorithm Description

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 other weight modification rule.

0-7803-1901-X/94 $4.00 ©1994 IEEE

If the training converges, we expand the training set by selecting λ candidate examples. Otherwise, we expand the network by introducing ν 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 [26].

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 from C. We use a superscript s, as in $D^{(s)}$, to denote the sth 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 gth growing step.

The weights of the network are initialized randomly with values from the interval $-\omega \leq w_{ij} \leq +\omega$. 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 $D^{(s)}$ 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

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

and each network growing step satisfies the condition

$$A^{(g)} \subset A^{(g+1)} \tag{2}$$

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

$$\mathbf{w}^{(s,g,t+1)} = \mathbf{w}^{(s,g,t)} + \Delta\mathbf{w}^{(s,g,t)} \tag{3}$$

$$\Delta\mathbf{w}^{(s,g,t)} = -\epsilon \nabla E_s|_{\mathbf{w}=\mathbf{w}^{(s,g,t)}} + \eta \Delta\mathbf{w}^{(s,g,t-1)} \tag{4}$$

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

$$\begin{aligned} E_s &= E(D^{(s)}|\mathbf{w}^{(s,g,t)}, A^{(g)}) \\ &= \sum_{m=1}^{N_s} \left(\mathbf{y}_m - f(\mathbf{x}_m; \mathbf{w}^{(s,g,t)}, A^{(g)})\right)^2 \end{aligned} \tag{5}$$

and the error gradient $\nabla E_s|_{\mathbf{w}=\mathbf{w}^{(s,g,t)}}$ is approximated by a back-propagation procedure [20]. In equation (4), ϵ and η are the step size and the momentum factor, respectively.

At every Δt 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,

$$E(D^{(s)}|\mathbf{w}^{(s,g,t)}, A^{(g)}) \leq \varepsilon_g \tag{6}$$

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

$$\varepsilon_g = \frac{1}{\tau}\{(I+1)\cdot H_g + (H_g+1)\cdot O\} \tag{7}$$

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

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

$$G_s = \frac{1}{N} \sum_{(\mathbf{x}_q, \mathbf{y}_q) \in B} \Theta\left(\mathbf{y}_q, f\left(\mathbf{x}_q; \mathbf{w}^{(s,g,t)}, A^{(g)}\right)\right) \tag{8}$$

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

$$\begin{aligned} &\Theta\left(\mathbf{y}_q, f\left(\mathbf{x}_q; \mathbf{w}^{(s,g,t)}, A^{(g)}\right)\right) \\ &= \begin{cases} 1 & \text{if } y_{qi} = f_i\left(\mathbf{x}_q; \mathbf{w}^{(s,g,t)}, A^{(g)}\right) \text{ for } \forall i \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{9}$$

If G_s exceeds the desired performance level ℓ, say 99%, then the entire algorithm stops. If $C^{(s)}$ is empty, the algorithm 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 $\mathbf{w}$ is computed.

The criticality $(\mathbf{x}_c, \mathbf{y}_c)$ of an example is defined as

$$e_{\mathbf{w}}(\mathbf{x}_c) = \frac{1}{\dim(\mathbf{y}_c)} \left(\mathbf{y}_m - f(\mathbf{x}_m; \mathbf{w}^{(s,g,t)}, A^{(g)})\right)^2 \tag{10}$$

which has a value $0 \leq e_{\mathbf{w}}(\mathbf{x}_c) \leq 1$ if the sigmoid activation function is used at the output layer. Then the training set is increased by selecting λ candidate examples, $(\mathbf{x}_c, \mathbf{y}_c)$, which are most critical:

$$\begin{aligned} D^{(s+1)} &= D^{(s)} \cup \{(\mathbf{x}_c, \mathbf{y}_c)\} \\ C^{(s+1)} &= C^{(s)} - \{(\mathbf{x}_c, \mathbf{y}_c)\} \end{aligned} \tag{11}$$

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. (6) 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 Δt epochs

$$\begin{aligned} \Delta E(t) = &E\left(D^{(s)}|\mathbf{w}^{(s,g,t-\Delta t), A^{(g)}}\right) \\ &- E\left(D^{(s)}|\mathbf{w}^{(s,g,t), A^{(g)}}\right) \end{aligned} \tag{12}$$

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

$$\Delta E_{sum}^{(s,g)}(t) = \Delta E(t) + \frac{1}{2}\Delta E_{sum}^{(s,g)}(t - \Delta t) \tag{13}$$

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

$$\Delta E_{avg}^{(s,g)}(t) = \frac{\Delta E_{sum}^{(s,g)}(t)}{N_s \cdot O} \tag{14}$$

Then the network grows if the total error is larger than the error tolerance ε_g defined in (6) and the average error change is smaller than the specified threshold ρ, i.e.

$$\left[E(D^{(s)}|\mathbf{w}^{(s,g,t)}, A^{(g)}) > \varepsilon_g\right] \wedge \left[\Delta E_{avg}^{(s,g)}(t) < \rho\right] \tag{15}$$

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

$$\begin{aligned} \mathcal{H}^{(g+1)} &= \mathcal{H}^{(g)} \cup \{H_g{+}1, \ ..., \ H_g{+}u\} \\ H_{g+1} &= H_g + u. \end{aligned} \tag{16}$$

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. This strategy gurantees 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.

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. In this case, the weights from the new hidden units j' to the output units i have

$$w_{ij'}^{(s,g+1,0)} = \gamma \cdot \left(\frac{\sum_{i\in\mathcal{O}}\sum_{j\in\mathcal{H}^{(g)}} w_{ij}^{(s,g,t)}}{O \cdot H_g}\right) + \omega_{ij'} \tag{17}$$

where $0 \leq \gamma \leq 1$ is a discount factor and the ω_{ij} terms are random values from the interval $[-\omega, +\omega]$, used to break the symmetries [20]. Likewise, the weights of the connections from the input k to the new hidden units j' are initialized by

$$w_{j'k}^{(s,g+1,0)} = \gamma \cdot \left(\frac{\sum_{j\in\mathcal{H}^{(g)}}\sum_{k\in\mathcal{I}} w_{jk}^{(s,g,t)}}{H_g \cdot I}\right) + \omega_{j'k} \tag{18}$$

The biases of the output and hidden units are initialized by the averages of existing bias values:

$$w_{j'0}^{(s,g+1,0)} = \gamma \cdot \left(\frac{\sum_{j\in\mathcal{H}^{(g)}} w_{j0}^{(s,g,t)}}{H_g}\right) + \omega_{j'0} \tag{19}$$

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 [26].

III. Application Results

The performance of the SELF algorithm in its learning speed and generalization accuracy is studied and compared with plain back-propagation (BP) networks in the context of two representative problems. The first task is a benchmark problem for which the optimal number of hidden units can be validated. The second is a more realistic application which contains some noisy data.

A. Autoassociation Problem

To show our method can find optimal network size we need a problem with known optimal architecture. One problem that serves this purpose is the autoassociation problem between n dimensional binary vectors:

$$\mathbf{x} = f(\mathbf{x}), \quad f : \{0,1\}^n \rightarrow \{0,1\}^n \tag{20}$$

Notice that the problem has a total number of 2^n examples and thus differs from the usual encoder/decoder problem [20] for which only n examples are possible. Using a fully connected feedforward network of H hidden units and $I = n$ inputs and outputs (I-H-I architecture), the minimal number of hidden units for solving this problem is $H = I$; between each pair of kth ($k = 1, ..., I$) input/output units a unique hidden unit must be allocated. We experimented with $I = 15$.

The optimality of the method was tested by varying the hidden layer growth size u. For each u, the network was initialized with u hidden units and in each network growing step the hidden layer was extended by u new units. A total of 300 randomly generated examples are given ($N = 300$), and the learning was started with a training set consisting of two seed examples ($N_0 = 2$). In each selection phase the training set was expanded by 10 new examples ($\lambda = 10$).

For $u = 1, 3, 5$ the minimal network size $H = 15$ was always found. For $u = 2, 4$ the network converged to $H = 16$. In the latter case, the algorithm could not find the minimal size because the algorithm searches only the integer multiples of u. However the algorithm finds always the smallest size greater than the minimum, which is optimal in terms of the specified u.

The efficiency of the network size optimization is affected by the network growth parameter. Figure 1 depicts the learning time T as a function of the network growth size u, averaged over 30 runs. T was measured as the total number of weight modifications. For this problem the learning time decreases exponentially as u increases. Increasing u means, however, increasing difference of the optimized network size from the minimal size unless $\mathrm{mod}\,(H_{min}, u) = 0$.

The performance of SELF nets is compared with that of BP nets, varying the initially given learning set size $N = 100, 200, 300, 400$. Figures 2, 3, and 4 show the average

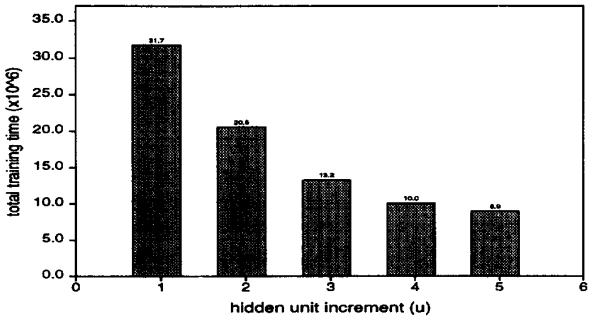

Figure 1: *Total learning time T vs. network growth size u. Increasing the growth size reduces the total learning time at the cost of minimality of the network size found.*

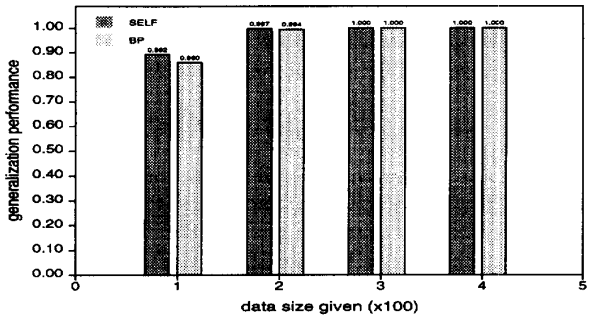

Figure 2: *Comparison of generalization performance. The smaller the sample size, the better the generalization performance of* SELF *nets compared with that of* BP *nets.*

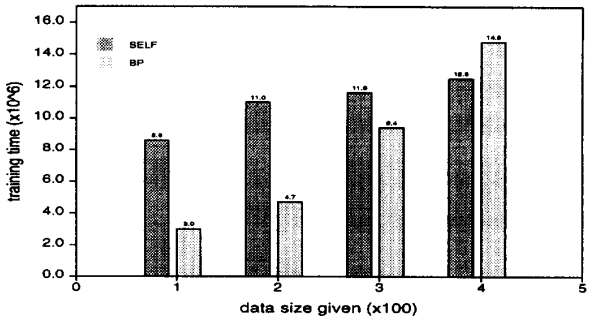

Figure 3: *Comparison of total learning time. The learning time for* SELF *nets remains constant for large sample size, while that of* BP *nets increases with the sample size.*

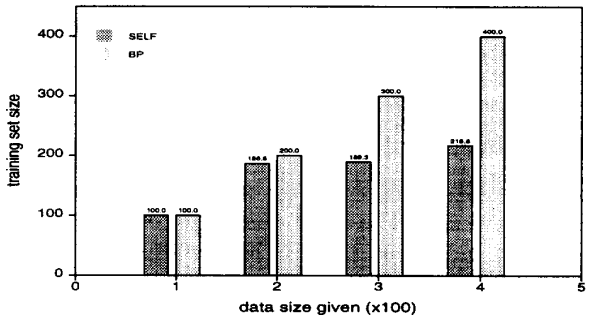

Figure 4: *Comparison of training set size used. The amount of training data for* SELF *nets remains almost constant for large sample size.*

results of 30 learning trials. Each SELF net was initialized with $H_0 = 3$ which was converged to $H_4 = 15$ using the growth size $u = 3$. The BP nets were initialized $H = 15$, the optimal size, which was fixed during learning. Weight modification step sizes were the same for both algorithms.

The results can be summarized as follows. (1) Given a training set whose size was short of achieving 100% generalization, SELF nets achieved better generalization performance than conventional BP nets, but with more learning time. (2) Given a training set whose size was enough to achieve 100% generalization, SELF nets converged, without loss of generalization performance, faster than BP nets.

B. Handwritten Digit Recognition

The algorithm was applied to the recognition of handwritten digits. We collected 6800 digit patterns written by 10 persons. Each pattern consists of 15×10 bitmap. Some of the bitmap patterns are shown in Figure 5. One half of the examples were used for training the network and the other half for testing the generalization performance of the trained network.

We used network size 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 $\varepsilon_g = \frac{1}{150} K_g$, where K_g is the total number of adjustable weights in the network of gth 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 Figures 6 and 7 as a function of the training set size. The performance was measured at each initial and final training epoch for each training set. In Figure 7, the points where the accuracy goes to almost zero indicates the network growth step. This is because we reini-

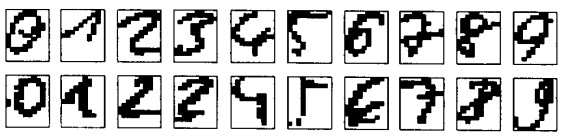

Figure 5: *Some of the digit patterns*

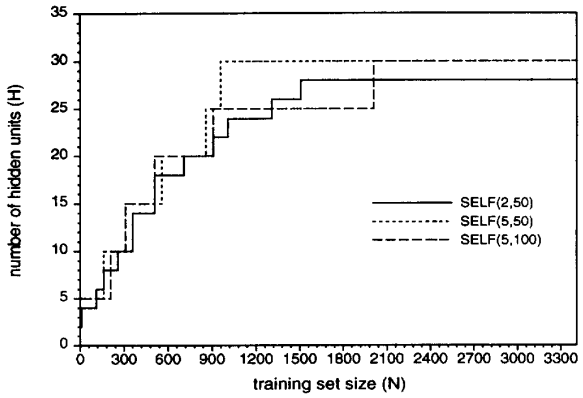

Figure 6: *Network size growth as a function of training set size. Optimization of network size is relatively robust against the parameters u and λ.*

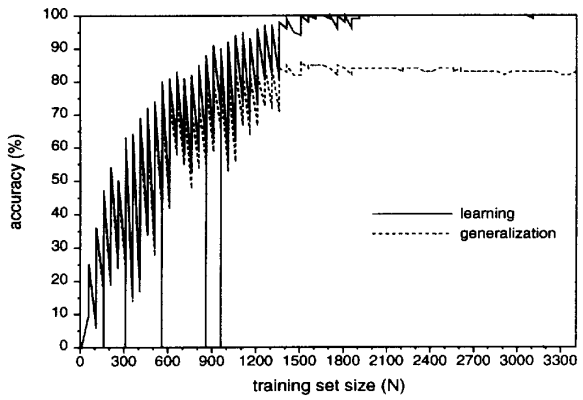

Figure 7: *Learning and generalization performance for the* SELF(5,50) *algorithm. The traning set size for which the accuracy is almost zero indicates the time when network growing takes place. Optimal size network is constructed using about 1000 examples out of 3400.*

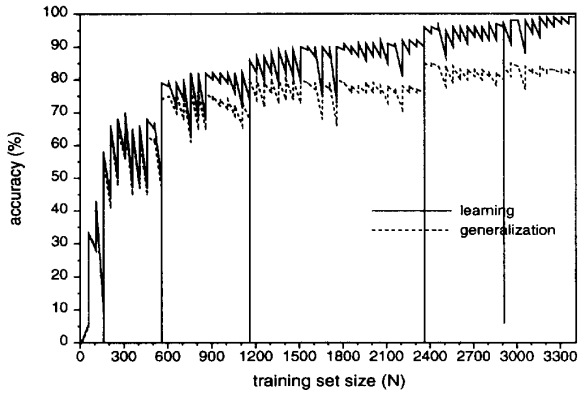

Figure 8: *Learning and generalization performance of the* SELF(5,50RANDOM) *algorithm in which examples are chosen randomly. Until an optimal network size is found, this algorithm uses about 3000 examples. This training set size is approximately three times larger than that of* SELF(5,50).

tialized 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 λ, resulting in two variations SELF(2,50) and SELF(5,100) of SELF(5,50). The results are shown in Figure 6, 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, λ) to denote a selective learning with H hidden units and λ 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 measure (Eqn. 10). Comparing Figures 7 and 8 we see that network size optimization combined with active example selection generalizes better and converges much faster than with random selection.

IV. Concluding Remarks

We show that an optimal number of hidden units can be found efficiently by selecting examples intelligently, instead of randomly or without any selection. One of the most useful characteristics of the SELF algorithm 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 optmization 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.

A final remark on noise is in order. The selection mechanism may prefer noisy data to regular examples if a large noise is injected in the given data set. It should, however, be noticed that the SELF method uses several interleaving stages of example selection and network training. Thus if at some stage the selected data is dominanted by noise, then the subsequent selection steps will choose dominantly typical data because they would reduce the total error. It is expected that the repetition of these would converge to a training set which has the same noise distribution as 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 Newton'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. Lebiere, "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, Sankt Augustin: Infix-Verlag, 1992. Also available as Informatik Berichte No. 93, Institut für Informatik I, Universität Bonn, D-53117 Bonn, 1992.

[25] B. T. Zhang, "Accelerated learning by active example selection," to appear in *International Journal of Neural Systems*, 1993.

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

[27] B. T. Zhang and G. Veenker, "Focused incremental learning for improved generalization with reduced training sets," in *Artificial Neural Networks: Proc. Int. Conf. Artificial Neural Networks (ICANN-91)*, T. Kohonen *et al.*, Ed. North-Holland, vol. I, pp. 227–232, 1991.

[28] B. T. Zhang and G. Veenker, "Neural networks that teach themselves through genetic discovery of novel examples," in *Proc. Int. Joint Conf. Neural Networks (IJCNN-91)*, IEEE, vol. I, pp. 690–695, 1991.