Self-development Learning: Constructing Optimal Size Neural Networks via Incremental Data Selection

Abstract—An incremental learning scheme is presented that constructs optimal size neural networks for solving particular problems. Unlike conventional constructive 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. The algorithm is derived from a statistical analysis of network learning. The effectiveness of this approach in convergence and generalization is demonstrated on several benchmark problems and a real-world data for digit recognition. While the method is described in the context of fully connected two-layer perceptrons, the underlying principle of network size optimization through incremental data selection can be applied to a wider class of neural network architectures.

1 INTRODUCTION

Any continuous multivariate function can be approximated by a multilayer feedforward 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 [23], one of the most popular 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 depends on the network architecture and size [9]. If the network contains too small a number of hidden units, the training will never converge because of lack of learning capacity. On the other hand, if the nework is too large, the generalization performance of the trained network will be poor due to possible overfitting. Some theoretical works give bounds on network size for learning a class of problems [6, 29]. Most of these studies are based on worst case analysis and not very practical.

Recently, several learning algorithms have been proposed to construct optimal size feedforward networks for specific applications [26]. These 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, 21]. Each of these methods uses all of the given data for network construction and training. The training time generally increases as the number of examples increases, while there is no guarantee that the generalization performance is improved by increasing the training set size [1]. Hence, one should use those examples which are most likely to help the network solve the problem.

The smallest set of training examples that is sufficient for a perfect generalization is referred to as minimal training set. For classification problems, the minimal training set consists of the border patterns, i.e., the patterns that lie closest to the separating hyperplanes. Some studies [2, 14] have shown that a network trained on border patterns generalizes better than a network trained on the same number of examples chosen at random. However, these studies have been limited to binary problems whose example space is small enough to be examined exhaustively, or the problem is simple enough to analyze. More recently, information-based objective functions for data selection have been addressed by MacKay [17]. He approaches the problem from the Bayesian framework. Plutowski and White [24] described a similar approach to finding concise training sets from clean data sets. The integrated squared bias (ISB) criterion requires calculation or approximation of Hessian which could be quite expensive.

In [34, 30] we have proposed criteria for selecting

informative examples, called interestingness and criticalness, which do not need to calculate second derivatives. We also presented in [31] a computationally efficient method for selecting critical examples and scheduling their presentation in order to maximize the training speed and improve generalization performance of the back-propagation based algorithms. The algorithm was shown to find a small subset of the given data which is sufficient to achieve a generalization performance as good as the original data.

In this paper we present an algorithm which constructs an optimal size feedforward network, and simultaneously selects a critical subset of a given data set. The algorithm learns fast because a small yet critical training set is used. It has a good generalization performance since a minimal size network is used. Unlike the usual back-propagation training procedure, this constructive learning procedure also has a robust convergence property. The solution obtained has the feature of local approximation, similar to k-nearest neighbor classifiers [10] or radial basis function networks [20], without giving up the global approximation ability of the multilayer networks.

The organization of the paper is as follows. In Section 2 the learning problem in neural networks is formulated as statistical inference. In Section 3 we derive the learning method from a Bayesian analysis of network learning and illustrate the rationale behind our approach. In Section 4 the algorithm is described and its performance is analyzed on a continuous function approximation problem. The general convergence and generalization properties of the method are investigated in Section 5 with some experimental results on benchmark problems. Section 6 discusses the results of application in digit recognition, followed by conclusion in Section 7.

2 Problem Formulation

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 hidden 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 jth to ith unit. The net input of unit i is computed by

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

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

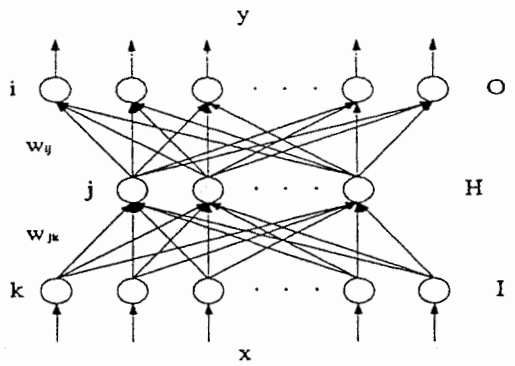


Figure 1: Fully connected feedforward network with one hidden layer

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

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

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

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

where I, H and O are the number of input, hidden, and output units, respectively. Each network configuration $\mathbf{w}$ implements a mapping from an input $\mathbf{x} \in X \subset \Re^I$ to an output $\mathbf{y} \in Y \subset \Re^O$. We denote this mapping by $\mathbf{y} = f(\mathbf{x}; \mathbf{w}), f : \Re^I \times \mathcal{W} \rightarrow \Re^O$. The set of all possible weight vectors $\mathbf{w}$ constitutes the confinguration space $\mathcal{W} \subset \Re^d$, 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:

$$D_N = \{(\mathbf{x}_m, \mathbf{y}_m)\}_{m=1}^{N} \qquad (4)$$

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

$$P_F(\mathbf{x}, \mathbf{y}) = P_F(\mathbf{x}) P_F(\mathbf{y}|\mathbf{x}) \qquad (5)$$

where $P_F(\mathbf{x})$ defines the region of interest in the input space and $P_F(\mathbf{y}|\mathbf{x})$ 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 $f(\mathbf{x};\mathbf{w})$ realized by the network. A commony used measure is the additive error functional

$$E(D_N|\mathbf{w}) = \sum_{m=1}^{N} e(\mathbf{y}_m|\mathbf{x}_m,\mathbf{w}) \quad (6)$$

where $e(\mathbf{y}_m|\mathbf{x}_m,\mathbf{w})$ is the sum of squared errors between the desired output $\mathbf{y}_m$ and the actual output $f(\mathbf{x}_m;\mathbf{w})$ of the network:

$$e(\mathbf{y}_m|\mathbf{x}_m,\mathbf{w}) = \sum_{i=1}^{O} (y_{mi} - f_i(\mathbf{x}_m;\mathbf{w}))^2 \quad (7)$$

where O is the number of output units and y_{mi} denotes the ith component of vector $\mathbf{y}_m$.

When the network configuration $\mathbf{w}$ is given, the network can be viewed as a model of the data [16, 17] and we can assign the probability of the data as a function of $\mathbf{w}$

$$P(D_N|\mathbf{w}) = \prod_{m=1}^{N} p(\mathbf{y}_m|\mathbf{x}_m,\mathbf{w})p(\mathbf{x}_m) \quad (8)$$

where the probability, $p(\mathbf{y}|\mathbf{x},\mathbf{w})$, that a network specified by $\mathbf{w}$ generates output $\mathbf{y}$ for input $\mathbf{x}$ is expressed as

$$p(\mathbf{y}|\mathbf{x},\mathbf{w}) = \frac{\exp(-\beta e(\mathbf{y}|\mathbf{x},\mathbf{w}))}{z(\beta)} \quad (9)$$

$$z(\beta) = \int_Y \exp(-\beta e(\mathbf{y}|\mathbf{x},\mathbf{w}))dy \quad (10)$$

Here β is a positive constant which determines the sensitivity of the probability to the error value and $z(\beta)$ is a normalizing constant.

The problem is to find a set of parameters $\mathbf{w}$ that maximizes the likelihood of the training set D_N of N independent examples:

$$\begin{aligned}
\mathbf{w}^* &= \arg\max_{\mathbf{w}\in\mathcal{W}} P(D_N|\mathbf{w}) \\
&= \arg\max_{\mathbf{w}\in\mathcal{W}} \prod_{m=1}^{N} p(\mathbf{y}_m|\mathbf{x}_m,\mathbf{w})p(\mathbf{x}_m) \\
&= \prod_{m=1}^{N} p(\mathbf{x}_m) \arg\max_{\mathbf{w}\in\mathcal{W}} \prod_{m=1}^{N} p(\mathbf{y}_m|\mathbf{x}_m,\mathbf{w}) \\
&= \arg\max_{\mathbf{w}\in\mathcal{W}} \prod_{m=1}^{N} p(\mathbf{y}_m|\mathbf{x}_m,\mathbf{w}) \\
&= \arg\max_{\mathbf{w}\in\mathcal{W}} \frac{1}{z_N(\beta)} \exp\left(-\beta \sum_{m=1}^{N} e(\mathbf{y}_m|\mathbf{x}_m,\mathbf{w})\right) \\
&= \arg\min_{\mathbf{w}\in\mathcal{W}} \sum_{m=1}^{N} e(\mathbf{y}_m|\mathbf{x}_m,\mathbf{w}) \\
&= \arg\min_{\mathbf{w}\in\mathcal{W}} E(D_N|\mathbf{w}) \quad (11)
\end{aligned}$$

using the relation (9) between the probability and the error.

3 Deriving the Method

We consider an ensemble of networks and the probability that the trained network generates the right output for a randomly chosen input on average. For a network with *fixed* architecture, which has been studied by Tishby *et al.* and Schwartz *et al.* [16, 25, 27], the total available volume Z_0 of the weight space is

$$Z_0 = \int_{\mathcal{W}} p(\mathbf{w})\, d\mathbf{w} \quad (12)$$

where $p(\mathbf{w})$ is some a priori density factor. Every vector $\mathbf{w}$ in the weight space represents a network implementing a function $f(\mathbf{x};\mathbf{w})$. Let $Z_0(\mathbf{w})$ be the effective volume of $\mathbf{w}$ when no examples are observed or with training set D_0, which is equivalent to $p(\mathbf{w})$. The prior distribution on the configuration space is

$$\begin{aligned}
P_0(\mathbf{w}) &= P(\mathbf{w}|D_0) = \frac{Z_0(\mathbf{w})}{Z_0} \\
&= \frac{1}{Z_0} p(\mathbf{w}) \quad (13)
\end{aligned}$$

The entropy of the prior distribution

$$S_0 = -\int_{\mathcal{W}} P_0(\mathbf{w}) \log P_0(\mathbf{w}) d\mathbf{w} \quad (14)$$

is a measure of the functional diversity of the chosen architecture. If S_0 is too small, there will be no method that solves the problem efficiently. On the other hand, if S_0 is large, then a large amount of information or data is needed to learn a particular function. This suggests that

- the network architecture should be complex enough to find a solution but simple enough to find the solution efficiently.

This is the first goal that our algorithm tries to optimize.

After successful learning N training examples, the weight vector $\mathbf{w}$ eventually lies within the region of the configuration space that is compatible with the training examples. Thus the effective volume of the configuration space is reduced to

$$Z_N = \int_{\mathcal{W}} p(\mathbf{w}) \prod_{m=1}^{N} p(\mathbf{y}_m|\mathbf{x}_m,\mathbf{w}) d\mathbf{w} \quad (15)$$

The volume of the configuration space consistent with the training set is given by

$$Z_N(\mathbf{w}) = p(\mathbf{w}) \prod_{m=1}^{N} p(\mathbf{y}_m|\mathbf{x}_m,\mathbf{w}) \quad (16)$$

Therefore, the fraction of the remaining weight space consistent with the training set D_N, i.e. the posterior

of the parameter $\mathbf{w}$ is

$$P_N(\mathbf{w}) = P(\mathbf{w}|D_N) = \frac{Z_N(\mathbf{w})}{Z_N} = \frac{1}{Z_N} p(\mathbf{w}) \prod_{m=1}^{N} p(\mathbf{y}_m|\mathbf{x}_m, \mathbf{w}) \quad (17)$$

terms of the training error $E(D_N|\mathbf{w}, A)$ the posterior probability of $\mathbf{w}$ is written as

$$P_N(\mathbf{w}) = \frac{1}{Z_N(\beta)} p(\mathbf{w}) \exp(-\beta E(D_N|\mathbf{w}))d\mathbf{w} \quad (18)$$

$$Z_N(\beta) = \int_{\mathcal{W}} p(\mathbf{w}) \exp(-\beta E(D_N|\mathbf{w}))d\mathbf{w} \quad (19)$$

notice that the normalization integral Z_N includes the region belonging to the desired function F, plus regions corresponding to other functions that agree with F on the training set. As we train the network we expect fewer such alternative functions, and we can think of learning as a continual reduction of the volume of allowed weight space

$$Z_0 \geq Z_1 \geq Z_2 \geq \cdots \geq Z_N. \quad (20)$$

This suggests another requirement on the learning algorithm:

- learning can be speeded up by increasing the reduction rate of the volume, or accelerating the error minimization process.

Much research has been done in this direction (see [5] for a recent review of various methods). In this work we use a simple gradient decent error correction rule, but other error correction method could be used as well.

The entropy corresponding to the posterior probability of the weight

$$S_N = -\int_{\mathcal{W}} P_N(\mathbf{w}) \log P_N(\mathbf{w})d\mathbf{w} \quad (21)$$

is a measure of the amount of implementable functions compatible with the training set D_N. As training proceeds it decreases steadily, and would go to zero if we ever reached the stage where only the desired function F was possible. Since S_N is actually a measure of the information required to specify a particular function, the difference $S_{N-1} - S_N$ tells how much information is gained by training on the Nth example. A common way of quantifying this information gain is the mean information for discrimination between $P_N(\mathbf{w})$ and $P_{N-1}(\mathbf{w})$ [15]:

$$I(P_N, P_{N-1}) = \int_{\mathcal{W}} P_N(\mathbf{w}) \log \frac{P_N(\mathbf{w})}{P_{N-1}(\mathbf{w})} d\mathbf{w} \quad (22)$$

The greater the value of $I(P_N, P_{N-1})$, the less resemblance there is between the two distributions and the more information we gain about $\mathbf{w}$. Given a fixed distribution $P_N(\mathbf{w})$, the maximum information gain is thus achieved by maximizing the difference of $P_N(\mathbf{w})$ from $P_{N-1}(\mathbf{w})$. Because of the relation (9), this can be done by selecting the example whose addition to D_{N-1} leads to the greatest $E(D_N|\mathbf{w})$ with the current parameters $\mathbf{w}$, i.e. the example that maximizes

$$\Delta E_N = E(D_N|\mathbf{w}) - E(D_{N-1}|\mathbf{w}) \quad (23)$$

and training the network on this example to reduce the error. Thus we have derived another method for maximizing information gain, that is

- information gain is maximized by selecting examples that cause the largest error and by training the network on these examples to reduce the error.

In general we define the criticality of an example $(\mathbf{x}_c, \mathbf{y}_c)$ as

$$e_{\mathbf{w}}(\mathbf{x}_c) = \frac{1}{\dim(\mathbf{y}_c)} e(\mathbf{y}_c|\mathbf{x}_c, \mathbf{w}) \quad (24)$$

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.

Another important property of learning networks is the generalization ability, the probability that networks trained on N examples, given an input $\mathbf{x}$, will correctly predict the output $\mathbf{y}$, where the pair $\mathbf{d} = (\mathbf{x}, \mathbf{y})$ is an independent sample from P_F:

$$P(\mathbf{y}|\mathbf{x}, D_N) = \int_{\mathcal{W}} P_N(\mathbf{w}) p(\mathbf{y}|\mathbf{x}, \mathbf{w})d\mathbf{w} = \frac{Z_{N+1}(\mathbf{d})}{Z_N \cdot z} \quad (25)$$

where z and Z_N are defined in (10) and (19). The ratio $\frac{Z_{N+1}}{Z_N}$ describes the relative volume of networks that are compatible with all $N+1$ examples among those that are compatible with the N training examples. This implies that

- maximally reducing the relative volume leads to maximization of generalization performance of the network.

We will present an algorithm which satisfies these four requirements in the next section. First we introduce three principles which the algorithm is based on. The first one is the principle of incremental data selection. Although a large amount of data is available, the training of network starts with a small data set and additional data are selected incrementally:

$$D_0 \subseteq D_1 \subseteq D_2 \subseteq \cdots \subseteq D_N. \quad (26)$$

As discussed above, the maximal information gain is achieved by selecting the example that causes the greatest error for the current network (Eq. 23). This

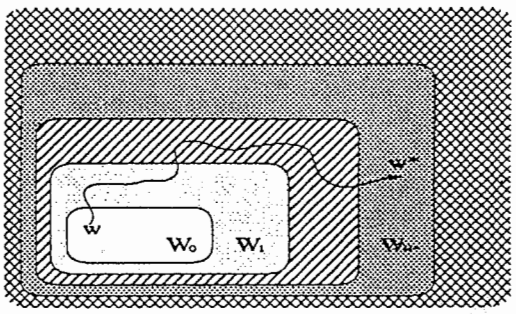


Figure 2: Configuration space of two-layer perceptrons with variable number of hidden units. The whole space has an embedded structure in which a larger network contains smaller ones. Learning is viewed as finding a path from a point w in the subspace $\mathcal{W}_0$ to a point w* in the subspace $\mathcal{W}_{H^*}$.

example will also increase the generalization performance because it maximally reduces the relative volume of the network (Eq. 25).

The second principle is to use a minimal size network for each training set during learning. This is based on the consideration that the functional diversity of the architecture should be minimal in order to achieve a good generalization (Eq. 14). We realize this requirement by considering an embedded structure of configuration space

$$\mathcal{W}_0 \subseteq \mathcal{W}_1 \subseteq \mathcal{W}_2 \subseteq \cdots \subseteq \mathcal{W}_H \tag{27}$$

whose effective volume increases monotonically

$$Z_0^{(0)} \leq Z_0^{(1)} \leq Z_0^{(2)} \leq \cdots \leq Z_0^{(H)}. \tag{28}$$

Figure 2 illustrates the structure of configuration space for the two-layer perceptron whose number of hidden units is small at the beginning and grows during learning. This constructive algorithm is thought to search the space for a point $\mathbf{w}^*$ in a region $\mathcal{W}_{H^*}$ starting from w in $\mathcal{W}_0$ via $\mathcal{W}_H$, $0 \leq H \leq H^*$, where H is the number of hidden units. The convergence speed and generalization performance depends on the lengh and route of the path.

The third principle is the integration of incremental data selection with constructive network growing, which we call self-development. As will be shown, the self-development process results in a synergetic effect—The minimal architecture size leads to a good generalization performance for the current training set, which in turn affects the selection of next example. On the other hand, the use of a small set of good training examples results in fast optimization of the complexity for learning the given data.

4 The Self-development Algorithm

4.1 Algorithm Description

The goal of the algorithm is to find a minimal network and a minimal training set, yet is large enough to achieve perfect generalization to the given data set. The top level of the algorithm is an iteration of three main steps, training the network, selecting examples, and growing the network (Figure 3). The algorithm will be referred to as SELF (SElective Learning with Flexible architectures).

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 of 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

$$D^{(s)} \cup C^{(s)} = B$$
$$D^{(s)} \cap C^{(s)} = \emptyset$$
$$D^{(s)} \subset D^{(s+1)}$$

and each network growing step satisfies the condition

$$A^{(g)} \subset A^{(g+1)} \tag{29}$$

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 tth sweep through the training set $D^{(s)}$, the weights are modified by

$$\mathbf{w}^{(s,g,t+1)} = \mathbf{w}^{(s,g,t)} + \Delta\mathbf{w}^{(s,g,t)} \tag{30}$$

$$\Delta\mathbf{w}^{(s,g,t)} = -\epsilon\nabla E_s|_{\mathbf{w}=\mathbf{w}^{(s,g,t)}} + \eta\Delta\mathbf{w}^{(s,g,t-1)} \tag{31}$$

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

$$E_s = E(D^{(s)}|\mathbf{w}^{(s,g,t)}, A^{(g)}) = \sum_{m=1}^{N_s} \left(\mathrm{y}_m - f(\mathbf{x}_m; \mathbf{w}^{(s,g,t)}, A^{(g)})\right)^2 \tag{32}$$

and the error gradient $\nabla E_s|_{\mathbf{w}=\mathbf{w}^{(s,g,t)}}$ is approximated by a back-propagation procedure [23]. In equation (31), 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,

$$E(D^{(s)}|\mathbf{w}^{(s,g,t)}, A^{(g)}) \leq \varepsilon_g \tag{33}$$

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

$$\varepsilon_g = \frac{1}{\tau}(I+1)\cdot H_g + (H_g+1)\cdot O \tag{34}$$

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 and plays the role of β in equation (9).

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

$$G_s = \frac{1}{N} \sum_{(\mathbf{x}_q, \mathbf{y}_q)\in B} \Theta\left(\mathbf{y}_q, f\left(\mathbf{x}_q; \mathbf{w}^{(s,g,t)}, A^{(g)}\right)\right) \tag{35}$$

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

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

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 non-empty $C^{(s)}$ means the network has generalized correctly to the candidate data set. Otherwise, the criticality $e_{\mathbf{w}}(\mathbf{x}_c)$ (see Eq. 24) with respect to the current model $\mathbf{w}$ is computed and the training set is increased by selecting λ candidate examples, $(\mathbf{x}_c, \mathbf{y}_c)$, which are most critical:

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

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. (34) 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

$$\begin{aligned} \Delta E(t) &= E\left(D^{(s)}|\mathbf{w}^{(s,g,t-\Delta t),A^{(g)}}\right) \\ &\quad - E\left(D^{(s)}|\mathbf{w}^{(s,g,t),A^{(g)}}\right) \end{aligned} \tag{36}$$

1. Initialize the training set $D^{(0)}$ and the candidate set $C^{(0)}$. Initialize the network architecture $A^{(0)}$ and weights $\mathbf{w}^{(0,0,0)}$. Set $s \leftarrow 0$, $g \leftarrow 0$, $t \leftarrow 0$.
2. Train the network $A^{(g)}$ to reduce the error $E(D^{(s)}|\mathbf{w}^{(s,g,t)}, A^{(g)})$ by using $D^{(s)}$ and adjusting $\mathbf{w}^{(s,g,t)}$. Set $t \leftarrow t+1$.
3. If the error smaller than the error tolerance ε_g, jump to step 6. Otherwise continue with the next step.
4. Check getting stuck in a local minimum. If yes, go to next step, otherwise return to step 2.
5. Increase the network size to create $A^{(g+1)}$ and set weight values $\mathbf{w}^{(s,g+1,0)}$. Set $g \leftarrow g+1$, and $t \leftarrow 0$. Return to step 2 to train the new network.
6. Test the generalization G of the network on the original data $D^{(s)} \cup C^{(s)}$. If G exeeds the desired performance level ℓ, halt the entire learning process.
7. Generate $D^{(s+1)}$ and $C^{(s+1)}$ by extracting λ most critical candidates from $C^{(s)}$ and appending them to $D^{(s)}$. Set $s \leftarrow s+1$. Return to step 2 to learn the increased training set.

Figure 3: Summary of the algorithm SELF

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

$$\Delta E^{(s,g)}_{sum}(t) = \Delta E(t) + \frac{1}{2}\Delta E^{(s,g)}_{sum}(t-\Delta t) \tag{37}$$

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

$$\Delta E^{(s,g)}_{avg}(t) = \frac{\Delta E^{(s,g)}_{sum}(t)}{N_s \cdot O} \tag{38}$$

Then the network grows if the total error is larger than the error tolerance ε_g defined in (34) and the average error change is smaller than the specified threshold ρ, i.e.

$$\left[E(D^{(s)}|\mathbf{w}^{(s,g,t)}, A^{(g)}) > \varepsilon_g\right] \wedge \left[\Delta E^{(s,g)}_{avg}(t) < \rho\right] \tag{39}$$

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

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

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 is 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

$$w_{ij'}^{(s,g+1,0)} = \gamma \cdot \left(\frac{\sum_{i \in \mathcal{O}} \sum_{j \in \mathcal{H}(g)} w_{ij}^{(s,g,t)}}{O \cdot H_g} \right) + \omega_{ij'} \quad (40)$$

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 [23]. Likewise, the weights of the connections from the input k to the new hidden units j' are initialized by

$$w_{j'k}^{(s,g+1,0)} = \gamma \cdot \left(\frac{\sum_{j \in \mathcal{H}(g)} \sum_{k \in \mathcal{I}} w_{jk}^{(s,g,t)}}{H_g \cdot I} \right) + \omega_{j'k} \quad (41)$$

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

$$w_{j'0}^{(s,g+1,0)} = \gamma \cdot \left(\frac{\sum_{j \in \mathcal{H}(g)} w_{j0}^{(s,g,t)}}{H_g} \right) + \omega_{j'0} \quad (42)$$

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.

4.2 Continuous Function Approximation

Now we demonstrate the performance of the method on an example problem. Let us consider the function:

$$z(x, y) = \frac{1}{2} \sin(\pi x^2) \sin(2\pi y). \quad (43)$$

For the input domain $-\frac{1}{2} \leq x, y \leq +\frac{1}{2}$, this function has the range of $-\frac{1}{2} \leq z \leq +\frac{1}{2}$. Figure 4 depicts the graph of the function in this domain. Because of its multimodality, the function is not trivial to learn.

A network consisting of two input units and one output unit is used. The necessary number of hidden units is to be found during learning. The x- and y-values were coded in the interval $[0.1, 0.9]$ over an input unit of the network. Likewise, the z-values were represented in the output unit using the activation interval of $[0.1, 0.9]$. A total of $11 \times 11 = 121$ examples

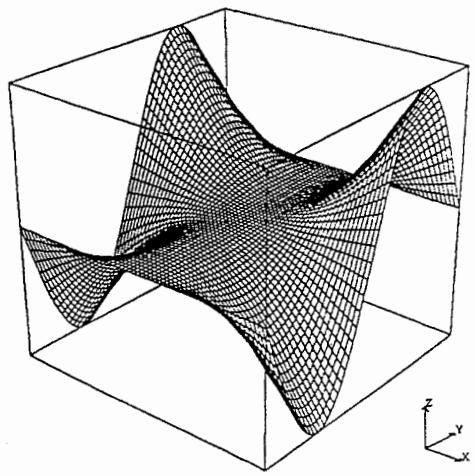


Figure 4: The desired function. The (x, y, z) coordinate of bottom left has the value $(-\frac{1}{2}, -\frac{1}{2}, -\frac{1}{2})$ and that of top right is $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$.

are given as the data set. The performance was tested on a set of $61 \times 61 = 3721$ data points. The learning started with a training set consisting of two seed examples ($N_0 = 2$) and a network with two hidden units ($H_0 = 2$). The weights were initialized with randomly chosen values from the interval $[-0.1, +0.1]$. The learning rate and the momentum factor were $\epsilon = 0.1$ and $\eta = 0.5$. In each selection step, 10 new examples were chosen to expand the training set ($\lambda = 10$). Each network growing step introduced three new hidden units ($u = 3$) with a complete connectivity to input and output units. The averaging strategy was used to initialized the newly introduced weights.

The intermediate approximation results during learning are depicted in Figure 5. In the left column are shown the results for SELF networks. Below each picture are indicated the learning time (T) in the total number of weight modifications, the number of hidden units (H_g), and the size of training set (N_s). The constructive method converged with 8 hidden units. For comparison, the performance of a back-propagation (BP) network with 8 hidden units for the nearest time point to each SELF learning is shown on the right column of Figure 5. Note that the network size and training set size are fixed for the BP learning. The learning rate and the momentum factor of the BP net were the same as the SELF net.

In the pictures it is easy to recognize the difference in the learning strategies of each method. The BP net attempts from the outset "ambitiously" to learn all the data points in the large training set. One sees the typical effect of "dancing together" [8] in the BP network to get a role for each hidden unit, resulting in a slow convergence. On the other hand, the SELF

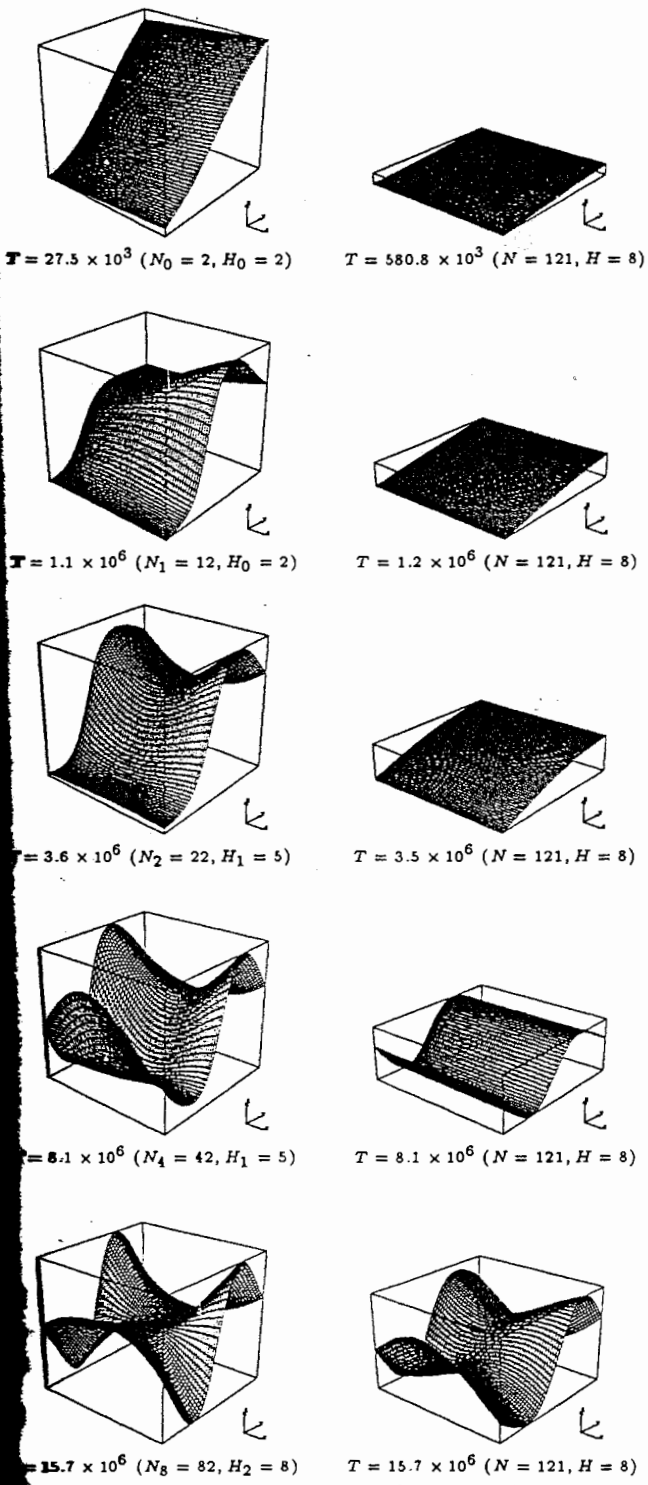


Figure 5: Comparison of SELF and BP nets. The figures in the left column shows the approximation results for a SELF net during a learning trial. The performance of a BP net for the nearest learning time was shown on the right column to compare the results.

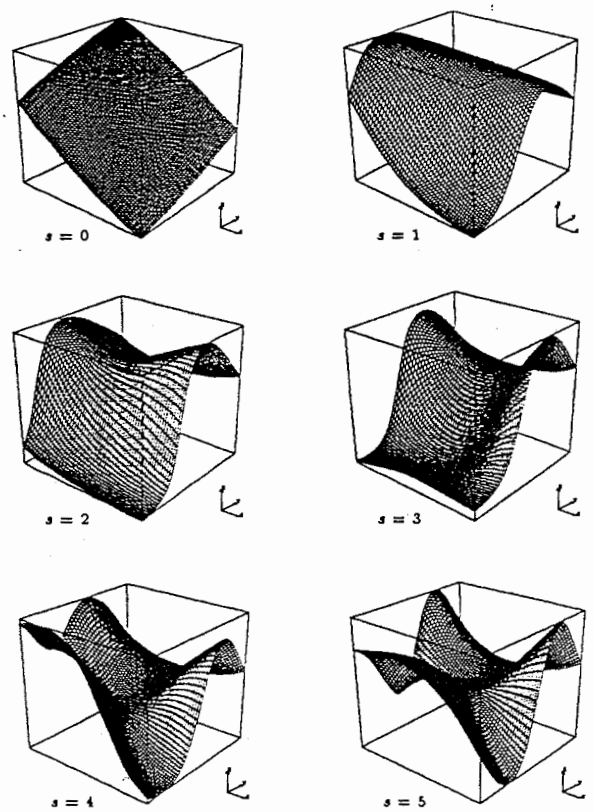


Figure 6: Self-developing approximation using different seed examples

net makes use of a "modest" strategy; it learns first a part of the input space, then attempts incrementally to learn the regions in which the desired function is still not well approximated. This incremental process by focusing on poorly approximated regions has the advantage that it finds a good solution quickly.

The results can be influenced by seed examples. Figure 6 shows a learning trial using a different set of seed examples. The results shows that inspite of much difference at earlier stages the approximation converges to a similar result in a few selection steps. In general it is expected that the initial training set has insignificant effect on the final approximation result if its size is small compared to the training set growth λ. The seed examples serve as an initialization of network knowledge to "ignite" a focused exploration of the example space. Thus we shall denote a SELF algorithm by SELF(u, λ), using only the growth parameter for network and training set and ignoring their initial sizes.

5 Optimality, Convergence, and Generalization

The SELF algorithm tries to find a smallest training set, $D^{(s^*)}$, and a smallest network for learning the

original data set $D^{(s^*)} \cup C^{(s^*)}$ by using an increasing size of training sets and by finding a smallest network which is able to learn each training set:

$$D^{(s^*)}, A^{(g^*)}, \mathbf{w}^{(s^*,g^*,t^*)} =$$

$$\arg \min_{D^{(s)}, \mathbf{w}^{(s,g,t)}, A^{(g)}} \sum_{s=0}^{sm} E\left(D^{(s)} \cup C^{(s)} | \mathbf{w}^{(s,g,t)}, A^{(g)}\right)$$

Here sm is the maximum possible number of selection steps given by

$$sm = \left\lceil \frac{N - N_0}{\lambda} \right\rceil \quad (44)$$

where N is the size of given data set, N_0 denotes the number of seed examples, and λ is the training set increment parameter. Note that the size of sth training set, $N_s = |D^{(s)}|$, satisfies the relation

$$N_0 < N_1 < \cdots < N_{sm} = N \quad (45)$$

and can be computed by

$$N_s = N_0 + s \cdot \lambda \quad (46)$$

Likewise, the size H_g of gth network in the number of hidden units increases monotonically

$$H_0 < H_1 < \cdots < H_{gm} = H \quad (47)$$

and can be computed as a function of growing step

$$H_g = H_0 + g \cdot u \quad (48)$$

Now we are interested in the following questions:

- Is the network size found by the algorithm optimal or minimal for learning the given data set?
- How efficient is the architecture optimization process?
- What is the effect of the parameters u and λ on the efficiency?
- How efficient is the convergence and generalization behavior of SELF as a function of the training set size.
- How does the performance of SELF nets compare to plain back-propagation nets.

These questions will be discussed in the context of a benchmark problem in this section and on a real-world data in the next section.

5.1 Optimality and Efficiency in Network Sizing

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:

$$\mathbf{x} = f(\mathbf{x}), \quad f : [0,1]^n \rightarrow [0,1]^n \quad (49)$$

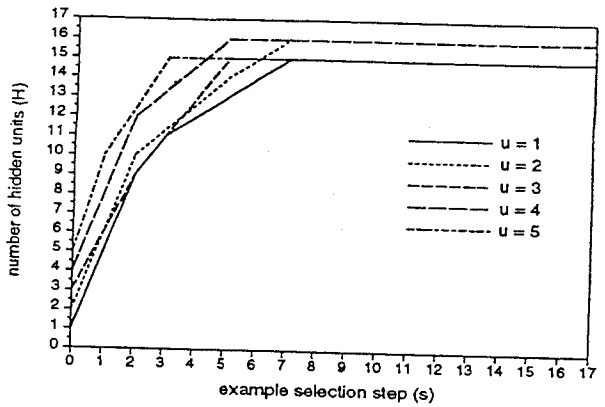


Figure 7: Network growth vs. training set size for the 15-H-15 autoassoziation problem. Five curves correspond to parameter values $u = 1, 2, 3, 4, 5$, respectively. Each network size found is optimal with respect to the parameter u (see text for explanation).

Notice that the problem has a total number of 2^n examples and thus differs from the usual encoder/decoder problem [23] 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 ($\lambda = 10$).

The results for $u = 1, 2, 3, 4, 5$ are shown in Figure 7. 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 as can be seen in Eq. (48). However the algorithm finds always the smallest size greater than the minimum, which is optimal in terms of the specified u. Notice that the network size grows rapidly at the early stage and the optimal network size was found, in the worst case, using only 72 examples out of 300 given examples.

The efficiency of the network size optimization is affected by the network growth parameter. Figure 8 depicts the total learning time T as a function of the

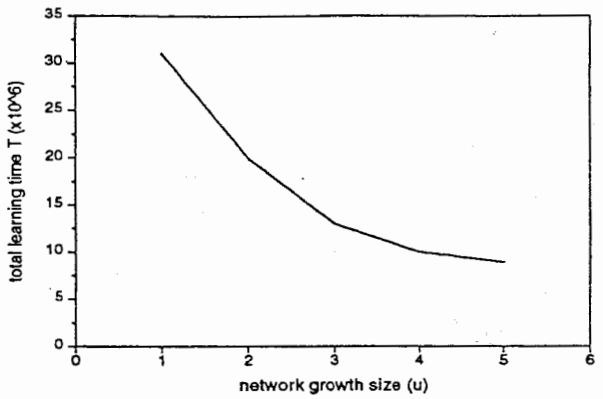


Figure 8: Total learning time T vs. network growth size u. Increasing the growth size reduces the total learning time at the cost of minimality of the network size found.

network growth size u, averaged over 30 runs. For the autoassociation problem the learning time decreases exponentially as u increases. Increasing u means, however, as was shown above, increasing difference of the optimized network size from the minimal size unless $\mathrm{mod}\,(H_{min}, u) = 0$.

5.2 Comparison of Convergence and Generalization Performance

We want to compare the learning curves of SELF and plain BP nets, i.e. the change of learning and generalization performance as a function of learning time.

The cost function which is tried to be minimized by the self-development process is

$$E_{SELF}(D_N|\mathbf{w}, A) = \sum_{s=0}^{sm} E(D_{N_s}|\mathbf{w}, A) \quad (50)$$

If we know the number of network growings for the sth training set, g_s, and the number of example presentations for each architecture and training set, $t_{s,g}$, then the computational requirements of SELF method in the total number of weight modifications is

$$\begin{aligned} T_{SELF}(D_N) &= \sum_{s=0}^{sm} T(D_{N_s}) \\ &= \sum_{s=0}^{sm} \sum_{g_s}^{gm_s} \sum_{t=1}^{t_{s,g}} \sum_{m=1}^{N_s} K_g \end{aligned} \quad (51)$$

where g_s and gm_s denote the first and final network growing step for the sth training set. The number of free parameters of the gth network, K_g, can be computed from the number H_g of hidden units:

$$K_g = (I+1) \cdot H_g + (H_g + 1) \cdot O \quad (52)$$

The costs for computing the criticality is small since the criticality test is carried out only once in each selection step and the total number of selection steps, sm, is much smaller than the total number of training epochs, i.e.

$$sm \ll \sum_{s=0}^{sm} t_{s,g} \quad (53)$$

Furthermore the criticality computation involves only a forward pass.

To describe learning curves let us consider the growth rate, r_s, of the sth training set:

$$r_s = \frac{N_s}{N_{s-1}} = \frac{N_0 + s \cdot \lambda}{N_0 + (s-1) \cdot \lambda} \quad (54)$$

The rate r_s is a monotonically decreasing function of s ($r_s \to 1$ as $s \to sm$). For example, in case of $N = 400$, $N_0 = 2$, and $\lambda = 10$, we have

$$N_0 = 2, \quad N_1 = 12, \quad N_2 = 22,$$
$$N_3 = 32, ..., N_{39} = 392, \quad N_{40} = 400$$

and the training set growth rates are

$$r_1 = \frac{N_1}{N_0} = 6.0, \quad r_2 = \frac{N_2}{N_1} \approx 1.8,$$
$$r_3 = \frac{N_3}{N_2} \approx 1.5, ..., r_{40} = \frac{N_{40}}{N_{39}} \approx 1.0$$

Similarly, the growth rate h_g of the gth network size is a monotonically decreasing function of g

$$h_g = \frac{H_g}{H_{g-1}} = \frac{H_0 + g \cdot u}{H_0 + (g-1) \cdot u}. \quad (55)$$

or monotonically non-increasing function of N_s.

Observe that the large growth rate of training set at the early stages means that the probability of any additional examples for changing the current model is high, and a large number of training sweeps $t_{s,g}$ and/or network growth steps g is required. In this case, inspite of a large $t_{s,g}$ the learning time will remain relatively small, because N_s and H_g are small. As learning proceeds r_s approaches 1, meaning the relative information gain decreases. This implies no growth of network size. In this case, inspite that N_s and H_g are large, the training time will also not be very long, since $t_{s,g}$ is small.

On the other hand, consider a plain backpropagation procedure which tries to minimize the cost function

$$E_{BP}(D_N|\mathbf{w}, A) = \sum_{m=1}^{N} e(\mathbf{y}_m|\mathbf{x}_m, \mathbf{w}, A) \quad (56)$$

Comparing this objective function with Eq. (50) we can see that the SELF algorithm minimizes the usual objective function eventually, since $N_{sm} = N$. The

N	Method	G	N_s	T
100	SELF	0.892	100.0	8.6
	BP	0.860	100.0	3.0
200	SELF	0.997	186.6	11.0
	BP	0.994	200.0	4.7
300	SELF	1.000	189.3	11.6
	BP	1.000	300.0	9.4
400	SELF	1.000	216.8	12.5
	BP	1.000	400.0	14.8

Table 1: Results for 15-H-15 autoassociation problem with $u = 3$, averaged over 30 runs. The symbol N denotes the size of initially given data set. The generalization performance G was measured on an independent test set of 1000 examples. N_s denotes the size of the training set actually used. The learning time T is in millions of weight modifications.

total training time is given by

$$T_{BP}(D_N) = \sum_{t=1}^{t_{max}} \sum_{m=1}^{N} K = \sum_{s=0}^{0} \sum_{g=0}^{0} \sum_{t=1}^{t_{s,g}} \sum_{m=1}^{N_s} K_g \qquad (57)$$

where we set $N_s = N$ and $K_g = K$, and $t_{s,g} = t_{max}$ in the second equality. Comparing (51) and (57) we see that the self-development learning reduces to the usual back-propagation procedure, if the entire data set is used as the traning set from the start and the network size is fixed during learning. Hence the constructive incremental learning procedure is a generalization of the plain back-propagation procedure.

The performances of self-developing nets are compared with those of back-propagation nets, varying the initially given learning set size $N = 100, 200, 300, 400$. Table 1 summarizes the average 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. For $N = 100$, i.e. given 100 examples, the SELF nets converged three times slower that the BP nets. Both nets could not achieve 100% generalization, but the generalization performance of the SELF nets was better than that of the BP nets. For $N = 400$, both nets achieved 100% gerneralization but the SELF nets converged faster than the BP nets. Between $N = 100$ and 400 as the given data size increases the convergence speed of SELF becomes increasingly superior to back-propagation networks. As

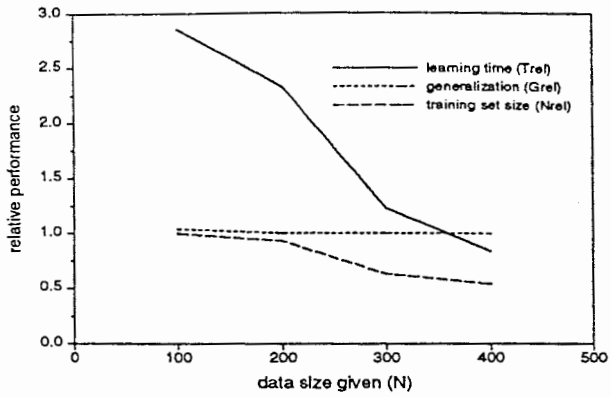


Figure 9: Relative performance of SELF to BP learning, averaged over 30 runs. The relative generalization performance $G_{rel} = \frac{G_{SELF}}{G_{BP}}$ is measured by the ratio of generalization accuracy of the SELF network to that of the BP network. Similarly, T_{rel} und N_{rel} represents the relative performance in the total learning time and the reduction in training set size. It is observed that the learning time of SELF is closely related with the reduction rate of training sets.

expected the learning time has a close relation to the actually used training set size (see Figure 9).

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.

6 Application in Digit Recognition

Now we apply the algorithm to the recognition of handwritten digits. The data was collected from 10 persons. Each person wrote two sheets of paper, each sheet containing 340 digits. The digits were read by a scanner to be converted in 15 × 10 bitmaps. One sheet of each writer was chosen randomly to build the data set (3400 digits). The other sheets of them were collected to build the test set (3400 digits). Some examples for the bitmap patterns are shown in Figure 10.

We first tried to find an optimal network size by SELF(2,50), i.e. using network size increment $u = 2$ and training set size increment $\lambda = 50$. The initial network containd two 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 first run converged to a network with 28 hidden units, i.e. 150-28-10 architecture, which was constructed using about a half of the given data set. The learning trial achieved approximately 85% generalization. Figure 12 shows the learning and generalization curves of the network until convergence as a function of the training set size. The performance was measured at each initial and final training epoch for each training set. The network growing step can be recognized at the point where the accuracy is almost zero. This is because we reinitialized the entire weights at each growing step to enforce the network size minimization. Notice that after reaching the optimal size network there is no significant improvement in the generalization performance, indicating that the method has already found a critical subset of the given data. The same performance is shown again in Figure 13 as a function of the training time.

The effectiveness of the method was tested by making further control experiments. We varied the parameter values u and λ, resulting in two variations SELF(5,50) and SELF(5,100) of SELF(2,50). The robustness of the algorithm in optimizing the network size is shown in Figure 11, where the network size is depicted as a function of the training set size (top) and the learning time (bottom). The learning and generalization curves for the three runs are shown in Figure 12 and Figure 13. The two variations of the basic SELF algorithm did not find the minimal size $H = 28$, but a subminimal size $H = 30$ because they used $u = 5$ and the number 28 is not a multiple of 5. The algorithm SELF(5,50) was the fastest one of the three, but there was no significant difference in generalization performance. Hence all the networks constructed are optimal with respect to the specified u.

The self-development algorithm was compared with two non-constructive algorithms: the plain backpropagation (BP) and SEL. The algorithm SEL is the same as SELF, except that SEL uses a fixed number of hidden units [30]. We use SEL(H, λ) to denote a selective learning with H hidden units and λ examples are chosen in each selection step. A 150-30-10 architecture was used for BP and SEL. Figure 14 compares the results. As expected, SELF is the most expensive algorithm of the three because of 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.

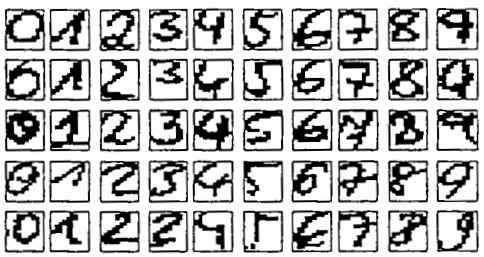


Figure 10: Some of the test digits

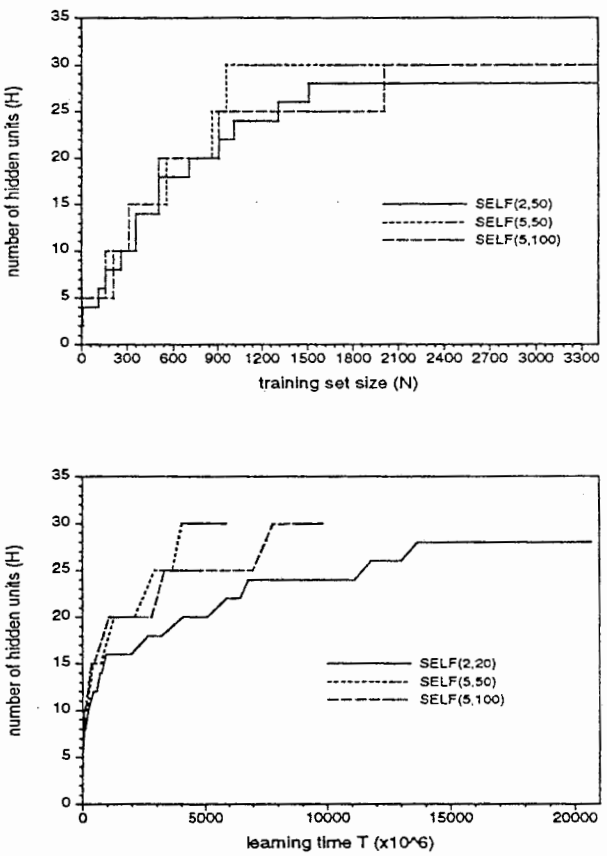
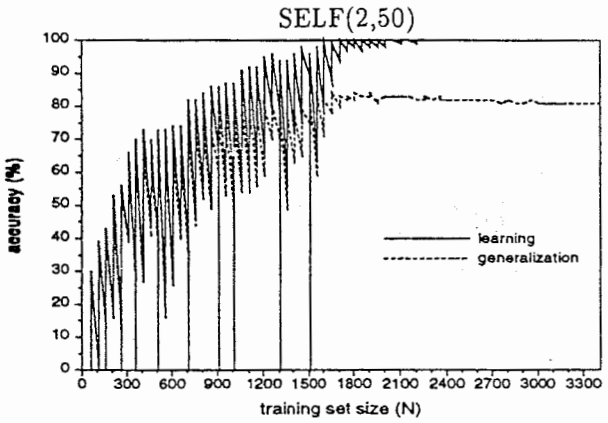
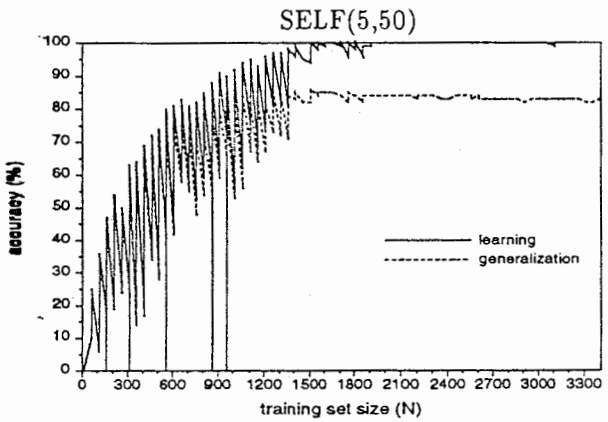


Figure 11: Network size growth as a function of training set size and learning time





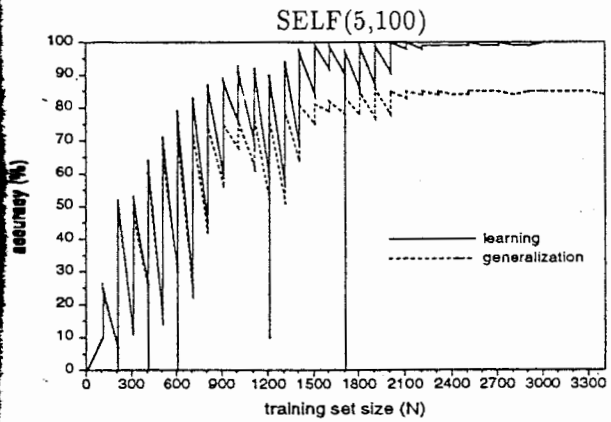
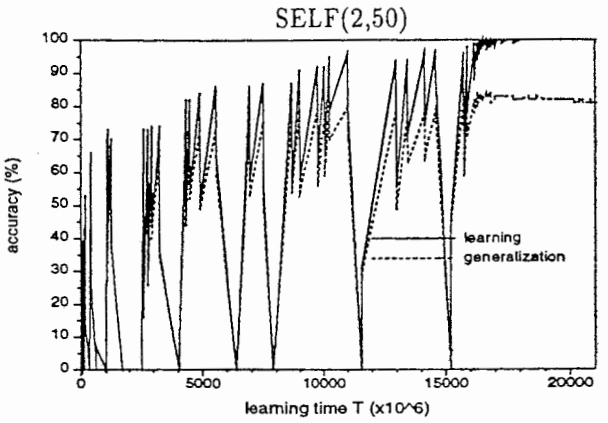
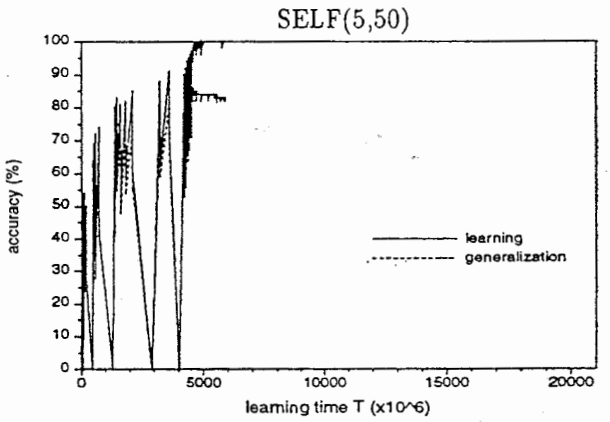


Figure 12: Learning and generalization accuracy as a function of the training set size for handwritten digit recognition.





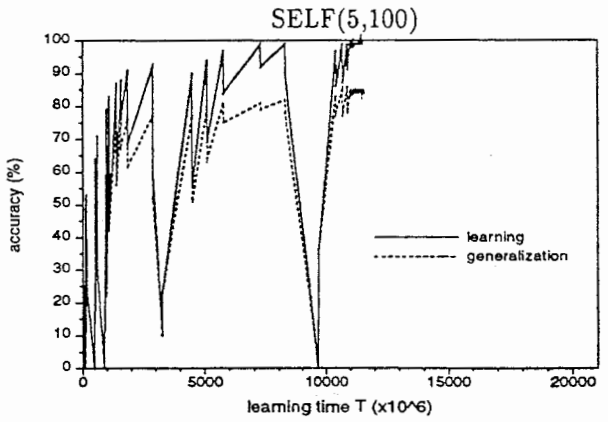
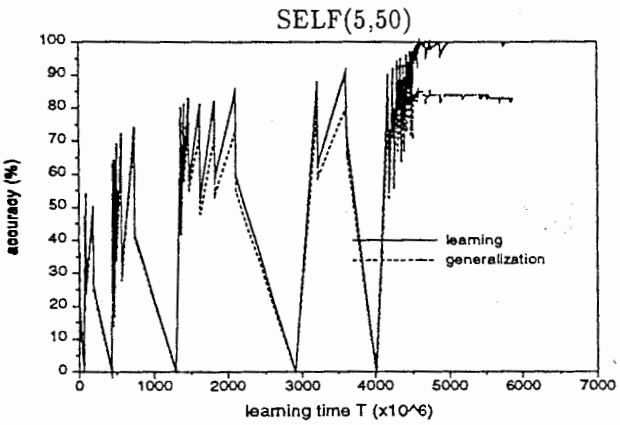
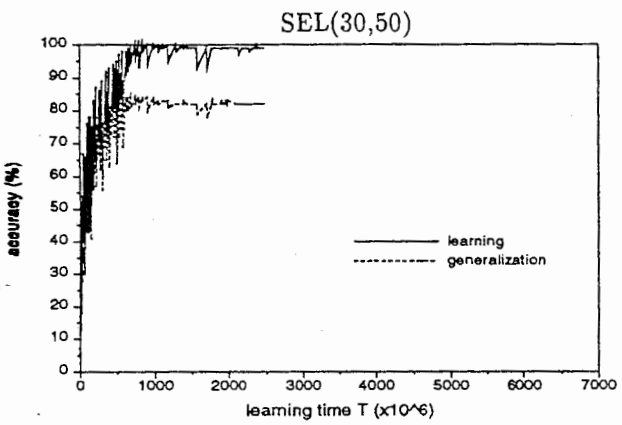


Figure 13: Learning and generalization accuracy as a function of constructive training time for handwritten digit recognition.





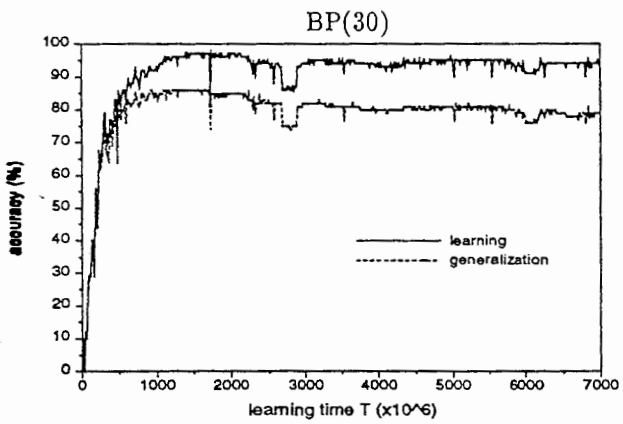
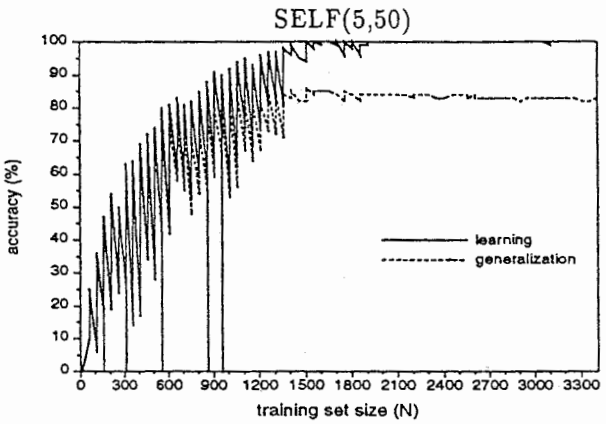
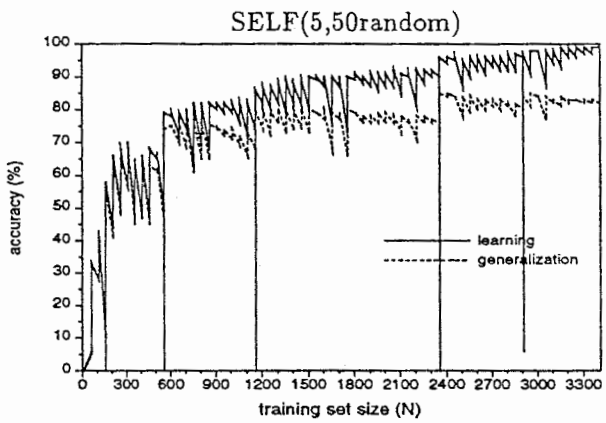


Figure 14: Comparison of learning and generalization performance as a function of learning time. The algorithm SELF used network growing plus example selection, while SEL used example selection only, and BP is the plain back-propagation.





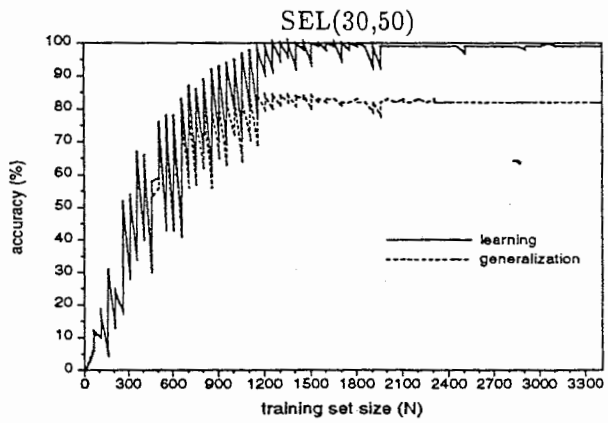


Figure 15: Effect of example selection and network growing in self-development learning. SELF(5,50random) used network growing with random example selection. SEL(30,50) used active example selection without network growing.

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 in convergence and generalization performance.

The effect of example selection during network growing was studied by by selecting examples randomly, instead of using the criticality measure. The results in Figure 15 show that the optimal network size is found much quicker if critical examples are selected. We also studied the effect of network growing during incremental learning. The results of SELF(5,50) and SEL(30,50) in Figure 15 suggests that growing the network size during learning will converge more robust than without.

7 Conclusion

In this paper we have presented a method that uses active data selection in sizing and training feedforward neural networks and thus overcomes some of the problems in earlier algorithms based on multilayer perceptrons. The method constructs a minimal size network for particular applications by searching the architecture space systematically in an increasing order of complexity. The network size optimization is efficient because only a critical subset of given training data is used. The algorithm generalizes quite well, since a minimal size network is used to learn the training set at any given time. We show that the combination of network growing and incremental data selection achieves better performance in convergence and generalization than any other algorithms using only one of both or none.

One of the most useful characteristics of self-development learning is that there is no need to decide in advance on the optimal 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, as the simulation results show. Another advantage of the self-development learning 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 a minimal network for the given data set.

Acknowledgement

The author would like to thank H. Mühlenbein and T. Christaller for stimulating the work. Thanks are also to P. H. Chu and F. Śmieja for careful reading and useful comments on the manuscript. This work was also benefited from the discussions with the members of the Neural Networks Research Groups of the University of Bonn and the Learning Systems Research Group of the GMD.

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 backpropagation 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 Inform. Pro. 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 delemma," *Neural Computation*, vol. 4, pp. 1–58, 1992.

[10] P. E. Hart, "The condensed nearest neighbor rule," *IEEE Trans. Inform. Theory*, vol. IT-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, "Backpropagation 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. of 1988 Connectionist Models Summer School*, Morgan Kaufmann, 1989, pp. 191-200.

[15] S. Kullback, *Information Theory and Statistics*, Wiley, New York, 1959.

[16] E. Levin, N. Tishby, and S. A. Solla, "A statistical approach to learning and generalization in layered neural networks," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1568–1574, 1990.

[17] D. J. C. MacKay, *Bayesian Methods for Adaptive Models*, Ph.D. thesis, Caltech, Pasadena, CA., 1992.

[18] H. Mühlenbein and J. Kindermann, "The dynamics of evolution and learning—Towards genetic neural networks," in *Connectionism in Perspective*, R. Pfeifer *et al.*, Ed. Elsevier, 1989, pp.173–197.

[19] M. Plutowski and H. White, "Selecting Concise Training Sets from Clean Data," *IEEE Trans. Neural Networks*, vol. 4, no. 2, pp. 305–318, 1993.

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

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

[22] M. D. Richard and R. P. Lippmann, "Neural network classifiers estimate Bayesian a posteriori probabilities," *Neural Computation*, vol. 3, pp. 461–483, 1991.

[23] 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.

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

[25] D. B. Schwartz *et al.*, "Exhaustive learning," *Neural Computation*, vol. 2, pp. 374–385, 1990.

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

[27] N. Tishby, E. Levin and S. A. Solla, "Consistent inference of probabilities in layered networks: predictions and generaliation," *Proc. Int. Joint Conf. Neural Networks*, Vol. II, IEEE, 1989, pp. 403–409.

[28] V. Vapnik, *Estimation of Dependencies Based on Empirical Data*, Springer-Verlag, New York.

[29] V. Vapnik, "Principles of risk minimization for learning theory,", in *Advances in Neural Information Processing 4*, Morgan Kaufmann, 1992, pp. 831–838.

[30] B. T. Zhang, "Learning by incremental selection of critical examples," Tech. Rep. No. 735, German National Research Center for Computer Science (GMD), Sankt Augustin, 1993, submitted to *Neural Computation*.

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

[32] B. T. Zhang, *Learning by Genetic Neural Evolution*, (in German), ISBN 3-929037-16-5, Infix-Verlag, Sankt Augustin, 1992. Also available as Informatik Berichte No. 93, Institut für Informatik I, Universität Bonn.

[33] B. T. Zhang and H. Mühlenbein, "Genetic programming of minimal neural nets using Occam's razor," in *Proc. Fifth Int. Conf. Genetic Algorithms*, S. Forrest, Ed. Morgan Kaufmann, 1993.

[34] B. T. Zhang and G. Veenker, "Focused incremental learning for improved generalization with reduced training sets," in *Artificial Neural Networks*, Vol. I, T. Kohonen *et al.*, Eds. Elsevier, 1991, pp. 227-232.

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