Arbeitspapiere der GMD 805

Byoung-Tak Zhang

Teaching Neural Networks by Genetic Exploration

November 1993

GESELLSCHAFT FÜR MATHEMATIK
UND DATENVERARBEITUNG MBH

e Arbeitspapiere der GMD enthalten vornehmlich rab-Veröffentlichungen, spezialisierte Einzelergeb- se und ergänzende Materialien. Im Interesse einer teren Veröffentlichung wird gebeten, die Arbeits- iere nicht weiter zu vervielfältigen. Die Autoren l für kritische Hinweise dankbar.

The "Arbeitspapiere der GMD" primarily comprise preliminary publications, specific partial results and complementary material. In the interest of a subsequent final publication the "Arbeitspapiere" should not be copied. Critical comments would be appreciated by the authors.

chrift des Verfassers/
ress of the author:

. Byoung-Tak Zhang
itut für Angewandte Informationstechnik
Gesellschaft für Mathematik
Datenverarbeitung mbH
fach 1316
53731 Sankt Augustin

ail: zhang@gmd.de

GMD-Arbeitspapiere
Herausgegeben von / Edited by:

Peter Behr
Raul Camposano
Thomas Christaller
Wolfgang K. Giloi
Peter Hoschka
Stefan Jähnichen
Thomas Lengauer
Erich J. Neuhold
Radu Popescu-Zeletin
Eckart Raubold
Ronald Tost
Ulrich Trottenberg
Dionysios Tsichritzis
Friedrich Winkelhage
Peter Wißkirchen

Verantwortlich für dieses Heft:

Prof. Dr. rer. nat. Thomas Christaller
Institut für Angewandte Informationstechnik

N 0723-0508

ellschaft für Mathematik und Datenverarbeitung mbH

8 Birlinghoven
53757 Sankt Augustin
fach 13 16
53731 Sankt Augustin

Telefon (02241)14-0
Telex 8 89 469 gmd d
Telefax (02241) 14 26 18
Teletex 2627-2241135 = GMDVV

Abstract

A fundamental difference in artificial neural networks from symbolic artificial intelligence systems is that the knowledge does not need to be given explicitly as symbolic rules. Instead, neural networks are able to learn the knowledge from examples. However, coventional learning methods for neural networks have limited applicability since training examples have to be provided by the teacher. In this paper we investigate an active learning method which incrementally acquires knowledge from the environment by generating training examples autonomously. We use genetic algorithm to explore the example space and present a fitness measure for guiding the search for useful examples. The effectiveness of the method is demonstrated on a robot control problem. The simulation results show that teaching neural nets through autonomously created examples not only extends the application domains of conventional learning algorithms but also improves convergence speed and generalization performance of the network. We discuss the implications of this self-teaching approach to constructing intelligent systems.

1 Introduction

Artificial intelligence (AI) research has made great advancements for last two decades. Many heuristic search algorithms, inference methods, learning paradigms, knowledge representation schemes, and system architectures have been developed in the AI framework (Nilsson, 1980; Winston, 1992). Traditionally, problem solving in AI has been considered as a sequential symbol manipulation. Symbolic systems are appropriate and useful for tasks whose solutions can be represented as symbolic rules. However, in some domains it is very difficult or impossible to find such rules. Examples include speech recognition, machine vision, and robot control (Arbib, 1987; Fischler and Firschein, 1987).

An artificial neural network consists of a large number of relatively simple processing elements connected densely and computing in parallel. There are several reasons why neural networks or connectionist systems provide an attractive alternative to the sequential symbolic approach to constructing intelligent systems.

One reason is the assumption on the structure of intelligent systems. Since its birth, the science of artificial intelligence has aimed to understand the intelligent behaviour of humans by simulation. Neural networks are, in their structure and function, more similar to human brain than conventional systems and seem to provide more sound substrate for building intelligent machines. The studies of formal neural networks may shed light on a deeper understanding of the principles on which human brain is based (Boden, 1990).

Another reason is the processing speed of neural networks. Intelligent systems require not only good heuristics but also a powerful computation ability (Fahlman *et al.*, 1983). Neural computation is inherently parallel (Zhang and Veenker, 1990) and can be easily implemented in VLSI (Mead, 1989). Neural hardware technology will allow the physical size of intelligent systems to decrease and the processing speed to increase significantly.

The third reason is that neural networks can effectively handle uncertain information. Information in neural nets are stored in a large number of connection weights and a connection weight takes part in a large number of information units. This distributed representation combined with parallel processing in a large number of small computing elements is robust against fault, noise, and incomplete information.

One of the most interesting properties of neural networks is the learning ability. In neural systems, the knowledge does not need to be given explicitly as rules or procedures. Instead, networks learn the knowledge from examples. Therefore, learning is one of the central issues in building artificial neural systems.

Among various kinds of network architecture, multilayer networks has been used most extensively due to their generality (Hornik *et al.*, 1989). Learning in these networks is done supervised, i.e. on the basis of a training set of input-output pairs $(\mathbf{x}_p, \mathbf{y}_p)$. Although the actual goal of learning is to approximate the mapping $\mathbf{y}_p = F(\mathbf{x}_p)$, most supervised learning algorithms assume the training to contain sufficient information. In this passive learning scheme the network can not learn much more across the training set provided by the environment (Figure 1). Thus the application domain and efficiency of such learning systems are limited.

To be more useful, a learning system should be able to actively interact with its environment to acquire new knowledge by generating hypotheses based on its own knowledge and testing them in its environment (Figure 2). Such ability, often referred to as exploration, is a fundamental feature of intelligent systems. Research on exploration has been done in symbolic machine learning under the AI framework (Michalski *et al.*, 1986), but remains relatively unexplored in neural networks.

The aim of this paper is to show that the learning in neural networks can be made more autonomous and thus more "intelligent" by active exploration of the environment. We describe an incremental learning algorithm that learns from the environment by generating training examples during learning. The method uses genetic algorithm (Holland, 1975; Goldberg, 1989) to find new useful examples.

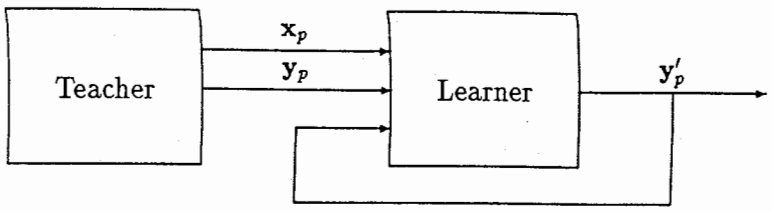

Figure 1: *The conventional passive learning paradigm*

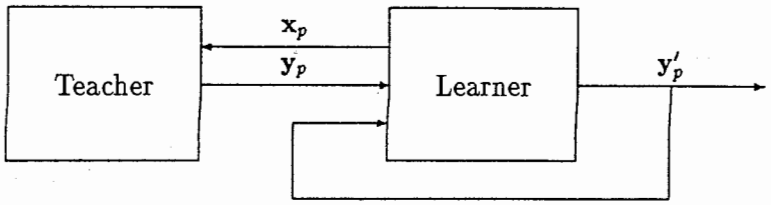

Figure 2: *The active learning paradigm used in this paper*

The organization of the paper is as follows. In Section 2 we give a brief introduction to neural networks to motivate the proposed method. The learning algorithm is described in Section 3. Section 4 demonstrates the applicability on robot arm control. In Section 5 we discuss the relationship with previous works and the implications of this work from the traditional AI perspective.

2 Backpropagation Networks

Neural networks learn an unknown relation F based on a training set:

$$D_N = \{(\mathbf{x}_p, \mathbf{y}_p)\}_{p=1}^{N} \tag{1}$$

where $\mathbf{x}_p \in X \subset \Re^I$, and $\mathbf{y}_p \in Y \subset \Re^O$. The pair of an input vector $\mathbf{x}_p$ and an output vector $\mathbf{y}_p$ is called an example of F. The relation F can be described by a probability density function defined over the space $X \times Y \subset \Re^{I+O}$:

$$P_F(\mathbf{x}, \mathbf{y}) = P_F(\mathbf{x}) P_F(\mathbf{y}|\mathbf{x}) \tag{2}$$

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.

Multilayer networks consist of units organized in layers (Figure 3). 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 j-th to i-th unit. The net input of unit i is computed by

$$net_i = \sum_{j \in R(i)} w_{ij} a_j + \theta_i \tag{3}$$

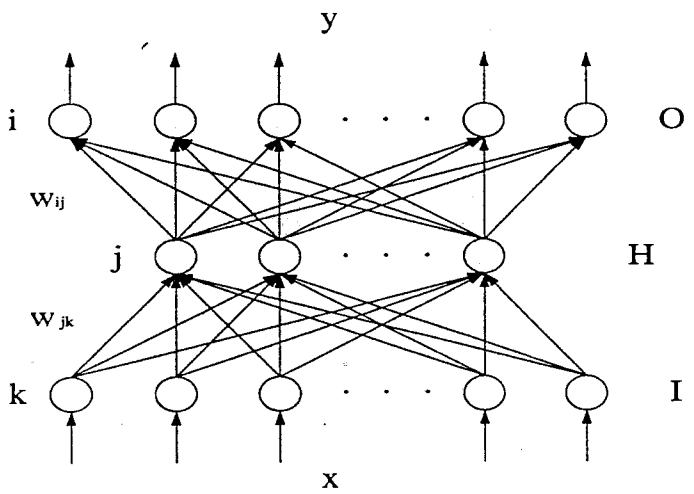

Figure 3: *A feedforward neural network*

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

$$a_i = f(net_i) = \frac{1}{1 + e^{-net_i}}. \tag{4}$$

For the case of a two layer (one hidden layer) perceptron as shown in in Figure 3, the i-th output of the network, f_i, $i = 1, ..., O$, is a nonlinear function of inputs x_k:

$$f_i(\mathbf{x}; \mathbf{w}) = f_i \left(\sum_{j=0}^{H} w_{ij} f_j \left(\sum_{k=0}^{I} w_{jk} x_k \right) \right), \tag{5}$$

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.

Learning algorithms change the weights of the network so that the performance of the network improves. Backpropagation (Rumelhart *et al.*, 1986) is a gradient descent method that uses a training set repeatedly to modify the weights. Each time an input vector $\mathbf{x}_p$ is presented to the network, the network computes an actual output vector $\mathbf{y}'_p = f(\mathbf{x}_p; \mathbf{w})$ which is compared with the desired output $\mathbf{y}_p$. A commonly used error measure is defined by the sum of squared errors $e(\mathbf{y}_p | \mathbf{x}_p, \mathbf{w})$:

$$e(\mathbf{y}_p | \mathbf{x}_p, \mathbf{w}) = \sum_{i=1}^{O} (y_{pi} - f_i(\mathbf{x}_p; \mathbf{w}))^2 \tag{6}$$

where O is the number of output units and y_{pi} denotes the i-th component of vector $\mathbf{y}_p$. Now the error $e(\mathbf{y}_p | \mathbf{x}_p, \mathbf{w})$ is propagated backwards to change the weights $\mathbf{w}$ in

the direction to reduce the errors:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \epsilon\nabla(\mathbf{w}_t) \quad (7)$$

where $\nabla(\mathbf{w}_t)$ is the t-th estimate of the error gradient with respect to the weights, and the parameter ϵ is the step size. The effectiveness of this process is measured by the generalization performance, i.e. the ability to produce correct answers to new inputs.

Most learning algorithms have considered the learning problem mainly as a problem of weight modification. However, there are other factors that affect the learning performance. One is the choice of training examples. The training set should be representative enough to achieve good results. It is obvious that too small a training set cannot train the network to a sufficient accuracy. On the other hand, a large training set will slow down the convergence. The redundant examples may not contribute to increasing the generalization performance of the network. The problem is how to select a small yet representative training set for the given network architecture. This has usually been considered as the task of human teacher, but the high nonlinearity of neural networks makes it difficult or impossible to determine the goodness of examples in advance.

Zhang and Veenker (1991a) and Zhang (1993) presents methods for selecting critical examples incrementally during one learning trial. Although proved useful for applications where a large data is known, the selection alone can not be applied in problems where examples are unknown at the outset or changing during learning. The method described in Zhang and Veenker (1991b) extends the selection method further by incorporating an example generation component.

In both of these methods, the network size (number of hidden units) remains unchanged during learning. Thus, a continuous performance improvement is not guaranteed unless the given network size is appropriate. The present work improves this further by employing another learning component that grows network size as needed to make sure that the increased training set improves the performance. The problem of network size optimization, given a fixed training set, is discussed in Zhang (1993). We will focus on active exploration of example space to teach the networks.

3 Teaching Backprop Nets by Genetic Search

3.1 The Self-teaching Approach

The proposed system consists of two learning modules: a neural learning (NL) module and a genetic learning (GL) module. The neural module is divided into two learning components of adaptation and development. The adaptation component is responsible for changing the weights of the network and the development component constructs new network architecture. The genetic module also consists of two components: selection and creation. The selection component filters useful examples

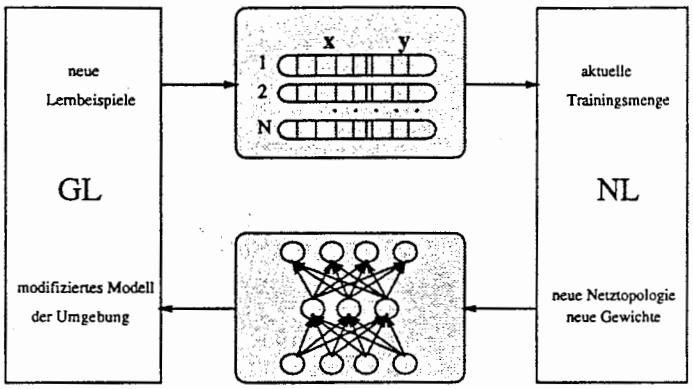

Figure 4: *The self-teaching approach*

from a large set of candidates. The creation component generates novel examples by applying genetic operators on the existing training set.

The interaction between these two modules are illustrated in Figure 4. The training set and the neural network play the role of communication channel between two modules. The neural module learns the examples which the genetic module provided and the result is stored in the neural network. The genetic module makes use of the knowledge in the neural network to generate more informative examples to train the network in the next stage. This is what we mean by self-teaching. The following section describes the four learning components and their relationship in more detail.

3.2 Algorithm Description

Learning starts with a feedforward neural network of h_0 hidden units whose weights $\mathbf{w}$ are initialized randomly. The initial training set D_0 contains a small number of seed examples chosen at random from the given data set. The rest of the data set is referred to as the candidate set and denoted by C_0.

In the adaptation phase, the network is trained by the current training set D:

$$w_{ij}(t+1) = w_{ij}(t) + \epsilon_p(t)\frac{\partial E_p}{\partial w_{ij}(t)} + \eta_p(t)\Delta w_{ij}(t-1) \tag{8}$$

where $E_p = e(\mathbf{y}_p|\mathbf{x}_p, \mathbf{w}) = \sum_{i=1}^{O}(y_{pi} - f_i(\mathbf{x}_p; \mathbf{w}))^2$. The parameters ϵ and η denote learning rate and momentum factor. Training continues until the precision of the network converges to the desired accuracy. This results in an improved knowledge base $\mathbf{w}$ and the learning continues with the selection phase. Otherwise, the learning continues with the development phase.

In the development phase, the necessity of network growing is first tested. We calculate at each time interval of Δt the change of errors $\Delta E(t) = E(t - \Delta t) - E(t)$ and its discounted sum:

$$\Delta E_{sum}(t) = \Delta E(t) + \frac{1}{2}\Delta E_{sum}(t - \Delta t). \tag{9}$$

Then the network grows if the average value $\Delta E_{avg}(t) = \frac{1}{N \cdot O} \Delta E_{sum}(t)$ exceeds some tolerance value. Otherwise, learning continues with the adaptation phase. The development process itself introduces new ν units into the hidden layer and adaptation of the network is tried again. Increasing the network size can learn the training examples eventually since in the worst case every training example can be memorized to the same number of hidden units.

In the selection phase, the training set D is increased by choosing examples from the candidate set C. If there are enough candidate examples, the most critical λ of them are included in the training set. The criticality e_p of the examples is measured by the trained network and defined as proportional to the error:

$$e_p = \frac{1}{\dim(\mathbf{y}_p)} \|\mathbf{y}_p - f(\mathbf{x}_p; \mathbf{w})\|^2. \tag{10}$$

This ensures the nework to gain a maximal information by successive training. An information-theoretical discussion of this issue is given in (Zhang, 1993). If there remains less than λ examples in the candidate set and the performance of the network is still not satisfactory, the learning continues with the creation phase which then returns to the selection phase.

In the creation phase, the new examples are generated and added to the candidate set. New examples are created by genetic recombination of two parent examples of the already existing training set D. The parent examples are chosen on the basis of the reproductivity $r_p(g)$ of the examples, i.e. the capability of examples to mate and generate child examples. $r_p(g)$ is defined as

$$r_p(g) = \frac{e_p(g)}{\sum_{q \in C_{\mathbf{y}_p}} e_q(g)} \cdot \frac{1}{M} \left(1 - \frac{N_{\mathbf{y}_p}}{N}\right) \tag{11}$$

where M is the number of possible categories and N denotes the current training set size. $N_{\mathbf{y}_p}$ is the number of examples belonging to the category of $\mathbf{y}_p$ in the current training set. The derivation of Eqn. (11) is given in Appendix. By this definition, the examples of categories which have fewer training examples have a larger probability of mating. So the novel examples of classes containing fewer examples will be generated more frequently than those classes which containing more examples. If two parent examples are determined, genetic operators are applied to generate offspring examples. Many operators are possible. For the experiments we have used the crossover and mutation operators described in the following section.

3.3 Genetic Operators for Example Generation

The crossover operation is used to exchange information between two parent examples. Let the parent chromosomes $\mathbf{x}_p$ and $\mathbf{x}_q$ be

$$\mathbf{x}_p = (x_1^p, ..., x_n^p) \text{ und } \mathbf{x}_q = (x_1^q, ..., x_n^q).$$

In two-point crossover $\otimes(\mathbf{x}_p, \mathbf{x}_q)$, two crossover sites are chosen in random (Figure 5). This divides the chromosome into three parts. The second part of each chromosome

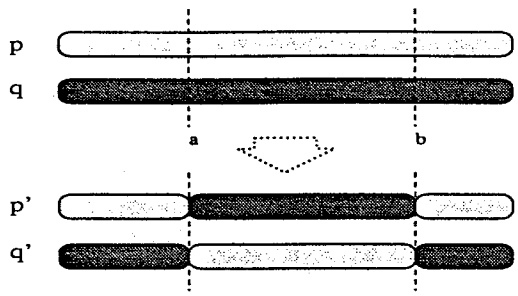

Figure 5: *Two-point crossover operation*

are exchanged, resuting in two offspring examples $\mathbf{x}'_p$ and $\mathbf{x}'_q$:

$$\otimes(\mathbf{x}_p, \mathbf{x}_q) = \begin{cases} \mathbf{x}'_p = (x_1^p, ..., x_{a-1}^p, x_a^q, ..., x_b^q, x_{b+1}^p, ..., x_n^p) \text{ and} \\ \mathbf{x}'_q = (x_1^q, ..., x_{a-1}^q, x_a^p, ..., x_b^p, x_{b+1}^q, ..., x_n^q), \end{cases} \tag{12}$$

where a and b, $1 \leq a \leq b \leq n$, are crossover sites and n is the chromosome length.

In the case of bitstrings, consider the following two chromosomes

$$\begin{aligned} \mathbf{x}_1 &= 1\ 1\ 1\ 1\ 1\ |\ 1\ 1\ |\ 1\ 1\ 1 \\ \mathbf{x}_2 &= 0\ 0\ 0\ 0\ 0\ |\ 0\ 0\ |\ 0\ 0\ 0 \end{aligned}$$

where the cross sites are as indicated. The information exchange by crossover results in two new examples

$$\begin{aligned} \mathbf{x}'_1 &= 1\ 1\ 1\ 1\ 1\ |\ 0\ 0\ |\ 1\ 1\ 1 \\ \mathbf{x}'_2 &= 0\ 0\ 0\ 0\ 0\ |\ 1\ 1\ |\ 0\ 0\ 0 \end{aligned}$$

which are similar but different from the parents.

After crossover, mutation operator is used. Mutation is useful for introducing new alleles which are nonexistent for the current training set. The mutation operator $\odot$ takes the input part of the example

$$\mathbf{x}_p = (x_1^p, ..., x_n^p)$$

and generates a new string of the same length,

$$\odot(\mathbf{x}_p) = \odot(x_1^p, ..., x_n^p) = (x_1^{p'}, ..., x_n^{p'}) = \mathbf{x}'_p \tag{13}$$

where each component of the vector is modified with a mutation rate $\mu > 0$. For instance, consider a binary pattern of length 10:

$$\mathbf{x}_1 = 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1$$

After mutation with a mutation rate of 0.2 (20%) one may achieve a bit vector, say

$$\mathbf{x}'_1 = 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1$$

where two of ten bits are toggled. This method can be generalized to mutate a vector of real-valued components (Mühlenbein *et al.*, 1993).

4 Application in Robot Control

A camera is set up to observe a ball which rolls through the work space of a robot arm. The task of the robot is to grasp the ball before the ball rolles out of its work space. This seemingly very easy task turned out to be difficult for a learning machine to work in real-time (Möller, 1991). First, the ball should be traced visually and its position have to be determined (image recognition). Second, the movement of the ball can be predicted because the ball rolls while the arm is moving (prediction of ball position). Now the robot should be able to move in the work space to the position that gets to the ball using the minimum energy. This means that the position and the time must be chosen for the arm to reach the ball with a shortest trajectory (planning). Given a desired position, the corresponding joint angles of the arm have to be computed (inverse kinematics) and the motions should be produced (motion generation).

The inverse kinematic problem was chosen to demonstrate the effectiveness of the self-teaching method. We used the robot arm RV M1 of Mitsubishi with five degrees of freedom (DOF), of which three DOF were used to solve the problems above. The rotation of the hand was not considered since it was irrelevant to the task. Figure 6 shows the kinematic parameters of the robot arm, where the angle θ_1 denotes the rotation of the basis.

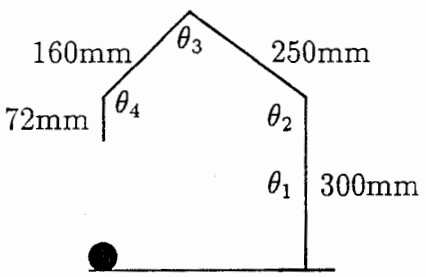

Figure 6: *Kinematic parameters of RV M1*

The inverse kinematics problem for this work consists of determining the joint angles of the robot arm to reach the desired position of the ball. This task can be described as a transformation

$$IK: \quad (p_x, p_y, p_z) \rightarrow (\theta_1, \theta_2, \theta_3, \theta_4) \tag{14}$$

where $(p_x, p_y, p_z) \in I\!R^3$ is a point in the work space of the robot arm. The vector $(\theta_1, \theta_2, \theta_3, \theta_4) \in I\!R^4$ describes a point in the configuration space of the joint angles. A total of 30 input units are used to encode the spatial position (p_x, p_y, p_z) of the ball in the work space of the robot arm. The four joint angles $\theta_1, \theta_2, \theta_3, \theta_4$ are represented on 24 output units of the network. Thus, each training example consists of an input vector of 30 bits and an output vector of 24 bits.

Instead of giving all training examples to the learning algorithm, we give the algorithm just one training example and let the genetic algorithm generate the input vectors, i.e. the training points. The corresponding joint angles for the training

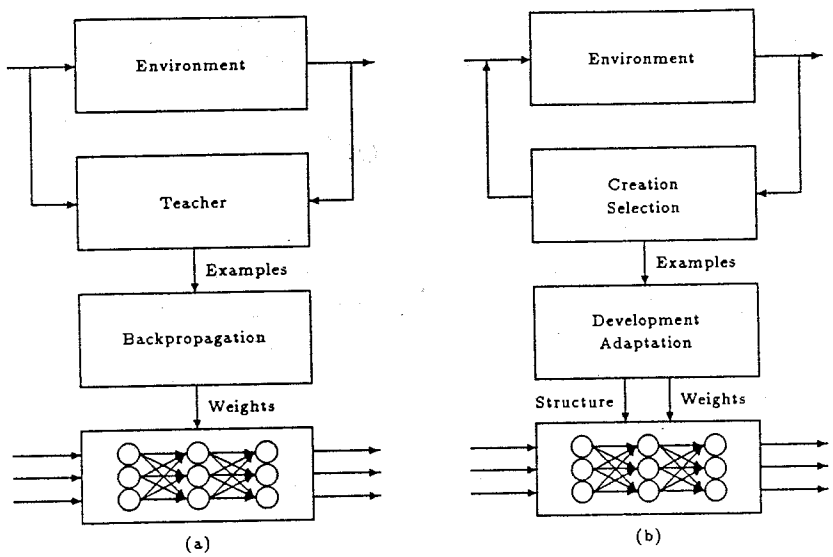

Figure 7: *Comparison of the conventional backprop (a) and the self-teaching approach (b)*

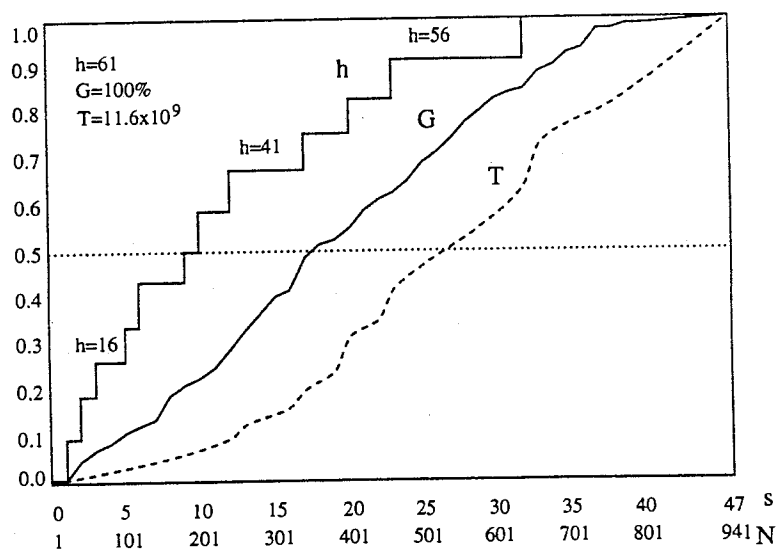

Figure 8: *Learning curves of a self-teaching network for robot arm control*

points were produced by a simulated teacher which led the arm to the desired position and measured the joint angles. Initially the network has one hidden unit, i.e. 30-1-24 architecuture. An appropriate number of hidden units for solving the problem must be found by the algorithm.

The structure of the self-teaching method is illustrated in Figure 7 in comparison to the conventional method. Whereas the backpropagation procedure just adjusts the weights of the given network structure using the entire set of training examples, the self-teaching generates and selects the examples, network architecture, and weight values.

The learning curves for solving the inverse kinematics problem are shown in Figure 8 as a function of the selection step s and the training set size N. The h curve shows the growing of network size in the number of hidden units. The T curve shows the relative time measured in total number of connection modifications for learning various size of examples. The G curve is the generalization performance of the network during learning. A continuous improvement of generalization performance is observed, suggesting the examples created are very useful. The learning finished with a network with 61 hidden units which corresponds to 12 development stages, i.e. $h_0 = 1$, $h_1 = 6$, $h_2 = 11$, ..., $h_{12} = 61$. Notice that only two thirds of all possible examples are used to find this network structure. The total number of weight modifications was 11.6×10^9 which took approximately 24 CPU hours on a SUN-2 Sparcstation.

To see the effectiveness of the example generation and selection mechanism we studied the sequence of examples used during the learning. Figure 9 shows the learning points that were discovered and used to train the network on the xy-plane. The brightness of the field indicates in which generation the corresponding example was introduced to the training set. Notice the tendency of the algorithm to search for good examples first in the vicinity of the starting points for the training to be economic, but sometimes it makes some jumps to distant regions to learn more. Using about a quarter (250) of the all possible examples, the genetic search explored the work space of the robot arm very well.

The six pictures in Figure 10 show in which generation the various joint angle was first used to learn the inverse kinematics. Notice that many of the angle combinations are used already in early generations. This implies the proposed reproductivity measure guide the search to generate the examples which is involved with the less used joints. In general, these examples will improve the performance of network maximally since they contain more information than others.

In summary, while the creation component searches in the work space of the robot arm, the selection component seeks critical examples in the configuration space of the arm. The iteration of creation and selection of examples leads to an automatic correction of the correlation between the input space and the output space of the desired mapping. This results in fast improvement of network performance.

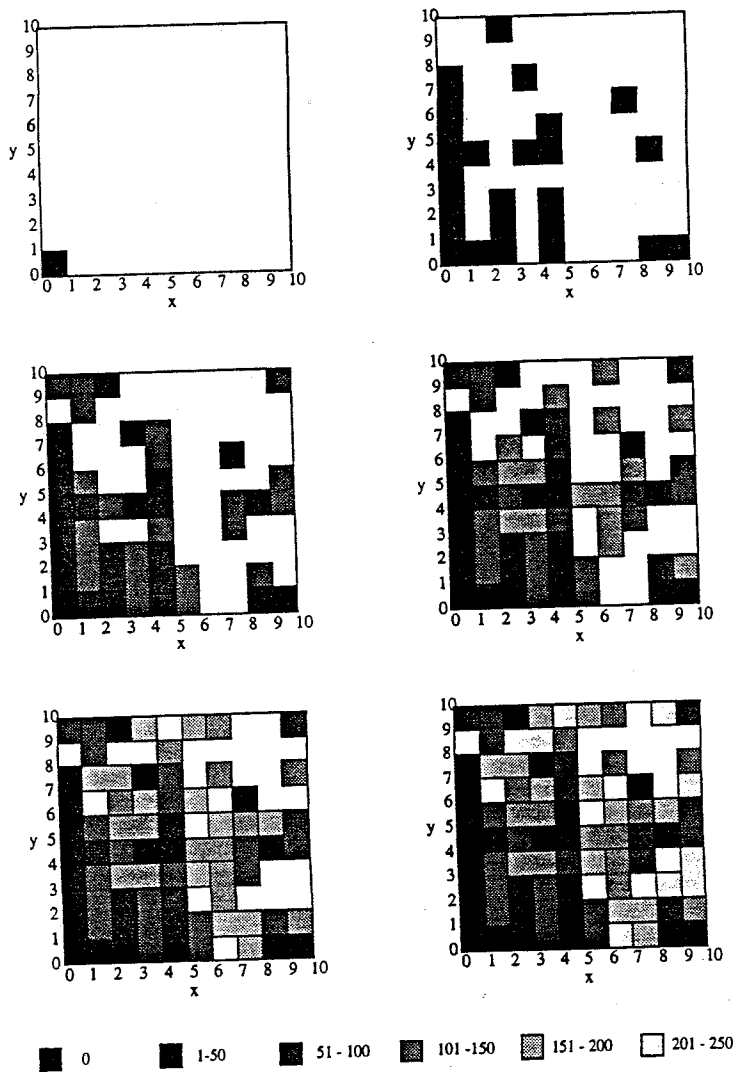

Figure 9: *Training points in work space*

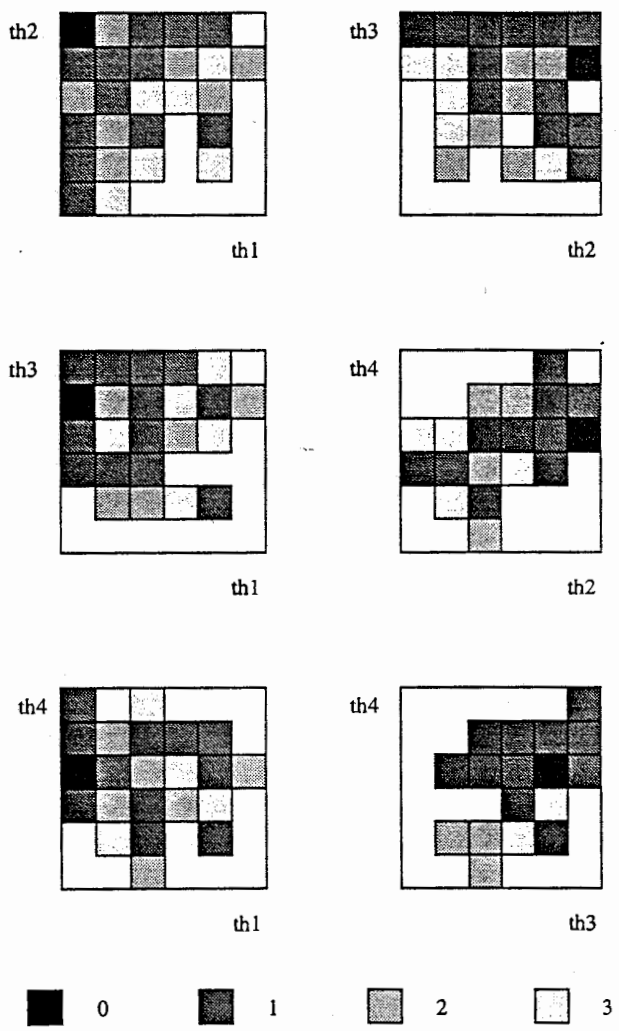

Figure 10: *Learning points in configuration space*

5 Discussion

Hinton (1989) classifies learning methods for neural networks into 3 groups: supervised learning, reinforcement learning and unsupervised learning. This taxonomy is based on the type of output signals provided by the teacher. In all of them, the input patterns have to be given by the environment and thus they can not learn new information without external stimuli.

Another toxonomy of learning methods for neural networks is introduced in (Zhang, 1992), depending on the exploration capability of the algorithms. This classification includes 6 learning types shown in Table 1. The first three passive algorithms corresponds to those of Hinton. In the last three types, the examples are determined by the learner itself, not by the teacher, and called active learning algorithms. The subclassification is again done by how the output patterns are generated. Each active algorithm may again be subdivided into selective algorithms and creative algorithms, depending on the examples are only selected from a given large data set, or created by the learner itself. The method we discussed in this paper belongs to the type 4 of above. It is straightforward to extend the method to apply to type 5 and type 6 learning.

class	type	environment	learner	name
passive learning	type 1	$\mathbf{x}$, $\mathbf{y}$	—	passive supervised learning
	type 2	$\mathbf{x}$, $c(\mathbf{x},\mathbf{y})$	$\mathbf{y}$	passive semisupervised learning
	type 3	$\mathbf{x}$	$\mathbf{y}$	passive unsupervised learning
active learning	type 4	$\mathbf{y}$	$\mathbf{x}$	active supervised learning
	type 5	$c(\mathbf{x},\mathbf{y})$	$\mathbf{x}$, $\mathbf{y}$	active semisupervise learning
	type 6	—	$\mathbf{x}$, $\mathbf{y}$	active unsupervised learning

Table 1: The 6 learning types

Recently, Beyer and Śmieja (1993) studied exploration in learning continuous functions by k-nearest neighbor methods. They distinguish density-based exploration and error-based exploration. Density-based exploration gives an equally distributed grid-like structure of exploration points in the input space. The density of points is dependent on how many $\mathbf{x}$ values may be chosen. In the error-bsased exploration, the learner calculate the error before $\mathbf{x}$ is learned. The next $\mathbf{x}$ to be explored is chosen in the neighborhood of known $\mathbf{x}$-values where the error is highest. The size of the neighborhood and the method for choosing a candidate out of this range are parameters of the exploration algorithm. Notice that our definition of reproductivity considers not only the error of the examples but also their distribution. Furthermore the examples are generated incrementally by training the network before the next exploration stage starts. Thus our method combines, in a natural way, the error-based and density-based exploration. The simulation experiments suggest that this combination results in an automated correction of correlation between the input space and the output space of the target relation.

Exploration has been, in one or the other way, applied to problems involving adaptation, in robot control (Thrun and Möller, 1992; Weber and Linden, 1992), in autonomous vehicles with a changing environment (Bessiére *et al.*, 1993). The role of exploration in learning control has been discussed in (Thrun, 1992). Our method differs from the earlier approaches in that we use genetic search. The basic idea in using genetic algoirthms in teaching neural networks was to combine the exploration capability of genetic algorithm with the exploitation capability of neural networks. In defining the genetic search we considered training examples as individuals, training set as population, and the trained neural network for fitness evaluation. We note that genetic algorithms are especially efficient in high dimensional search space.

The differences of our genetic algorithm from conventional ones should be made clear. In conventional genetic algorithms, the population size is fixed over generations. In contrast, the population size of our genetic algorithm increases monotonically. Usual genetic algorithms search for a single string in the population, whereas ours considers the whole population as a solution. In conventional genetic algorithms the fitness value of an individual is fixed, while the fitness of examples in our case varies during learning. Fitness functions are defined in usual genetic algorithms so that more adapted individuals have larger fitness, while we defined the fitness of less fitted examples to have larger fitness values.

Note also that this use of genetic algorithms is distinguished from earlier work using genetic algorithms for optimization of weights (Whitley *et al.*, 1990) and topology (Zhang and Mühlenbein, 1993) of neural networks. They encode the weights or architecture of a network as chromosomes which are modified by genetic operators. In contrast, we consider training examples as individuals while weights and architectures are modified in a standard way. As was shown in the robot control problem, the proposed application of genetic algorithms complements the weaknesses of connectionist learning methods in exploration.

To summarize, we argued in this paper that the conventional learning methods for neural networks are insufficient for automatic knowledge acquisition. We introduced an active learning method that teaches neural networks incrementally by exploring the knowledge source. For exploration we used a genetic algorithm with the fitness function defined to find out an informative subset of all possible examples for training the network. The simulation results on robot arm control show that the active exploration in the example space combined with the network architecture optimization not only extends the application domains of neural networks but also improves the convergence speed and the generalization performance of existing learning algorithms. We believe that the genetic neural learning approach builds a good starting point for constructing intelligent systems based on neural networks.

Appendix: Derivation of Reproductivity

Exchanging information between every combination of parents would be too expensive to generate new examples. To choose parents, we use a measure, called reproductivity, which should represent the importance of the examples for training the network. The reproductivity, r_p, is defined as

$$r_p(g) = \frac{e_p(g)}{\sum_{q=1}^{N} e_q(g)} \tag{15}$$

where $e_p(g)$ is the criticality of the p-th example in the g-th generation (Eqn. 10). N is the size of the current training set. According to this measure, the examples which have larger criticality value have higher probability of the critical parents to be selected. Because critical examples are defined as those still having high errors, this reproduction scheme let the search for new examples focus on the regions which are not approximated well, yet.

The reproductivity measure (15) is very general and can be used for many kinds of problems. However, if one has to solve a classification problem and the distribution over the categories is known *a priori*, then this may be incorporated to the reproductivity. Let $N_{\mathbf{y}_p}$ be the number of examples belong to the category $\mathbf{y}_p$ in the current training set:

$$N_{\mathbf{y}_p} = |S_{\mathbf{y}_p}|. \tag{16}$$

If one chooses an arbitrary example from the current training set, the probability of the example to be in the category $\mathbf{y}_p$ is given

$$P_{\mathbf{y}_p} = \frac{N_{\mathbf{y}_p}}{N} \tag{17}$$

where N is the training set size. In order to train the network effectively, new training examples should come from those categories that have still less training examples. Let $P^*_{\mathbf{y}_p}$ the prior probability of the category $\mathbf{y}_p$ and set

$$\bar{P}_{\mathbf{y}_p} = 1 - P_{\mathbf{y}_p}. \tag{18}$$

Now the reproductivity of the category $\mathbf{y}_p$ is defined to be

$$R_{\mathbf{y}_p} = P^*_{\mathbf{y}_p} \cdot \bar{P}_{\mathbf{y}_p} = P^*_{\mathbf{y}_p}(1 - P_{\mathbf{y}_p}) = P^*_{\mathbf{y}_p}\left(1 - \frac{N_{\mathbf{y}_p}}{N}\right) \tag{19}$$

where $P^*_{\mathbf{y}_p}$ and $P_{\mathbf{y}_p}$ are *a priori* probability and the frequency of examples belonging to the category of example p, respectively. In the case of uniform distribution, (19) reduces to

$$R_{\mathbf{y}_p} = \frac{1}{M}\left(1 - \frac{N_{\mathbf{y}_p}}{N}\right) \tag{20}$$

where M denotes the number of categories.

Now, the relative reproductivity of an example in a category, e'_p, is defined by considering the criticality of the example as

$$e'_p(g) = \frac{e_p(g)}{\sum_{q \in S_{\mathbf{y}_p}} e_q(g)} \tag{21}$$

where $S_{\mathbf{y}_p}$ and $e_p(g)$ are the category and the criticality of the example p in the g-th generation.

Now the effective reproductivity of an example is expressed as a product of (19) and (21):

$$r_p(g) = e'_p(g) \cdot R_{\mathbf{y}_p}(g) \tag{22}$$

In the case of uniform distribution, (22) reduces to Eqn. (11)

$$r_p(g) = \frac{e_p(g)}{\sum_{q \in S_{\mathbf{y}_p}} e_q(g)} \cdot \frac{1}{M} \left(1 - \frac{N_{\mathbf{y}_p}}{N}\right) \tag{23}$$

where M is the number of different categories and N denotes the current training set size.

Acknowledgements

We wish to thank G. Veenker and H. Mühlenbein for stimulating and supporting the work. Thanks are also to K. Möller, P. H. Chu, F. Śmieja, U. Beyer for their discussions and helpful comments. This work was supported in part by the POSCO Scholarship Society.

References

[1] Arbib, M. A. (1987) *Brains, Machines, and Mathematics* (New York: Springer-Verlag).

[2] Bessiére, P., Ahuactzin, J. M., Albi, E.G., and Mazer, E. (1993) The Ariadne's clew algorighm: Global planning with local methods. In *Proc. IEEE Conf. on Intelligent Robots and Systems.*

[3] Beyer, U. and Śmieja, F. (1993) Learning from examples using reflective exploration. Tech. Rep., German National Research Center for Computer Center (Sankt Augustin, GMD).

[4] Boden, M. A. (ed.) (1980) *The Philosophy of Artificial Intelligence.* (Oxford University Press: Oxford).

[5] Fahlman, S., Hinton, G. E., and Sejnowski, T. J (1983) Massively parallel architectures for AI: NETL, Thistle, and Boltzmann machines. In *Proc. AAAI-83.*

[6] Fischler, M. A. and Firschein, O. (1987) *Intelligence: The Eye, The Brain, and The Computer* (Addison-Wesely).

[7] Hinton, J. E. (1989) Connectionist learning procedures. *Artificial Intelligence*, 40:185–234.

[8] Holland, J. H. (1975) *Adaptation in Natural and Artificial Systems* (University of Michigan Press: Ann Arbor).

[9] Goldberg, D. E. (1989) *Genetic Algorithms in Search, Optimization, & Machine Learning* (Addison-Wesely).

[10] Hornik, K., Stinchcombe, M. and White, H. (1989) Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359-366.

[11] Mead, C. (1989) *Analog VLSI and Neural Systems* (Addison-Wesely).

[12] Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (eds.) (1986) *Machine Learning: An Artificial Intelligence Approach*, Vol. II. (Morgan Kaufmann).

[13] Möller, K. (1991) *ARC: Adaptive Roboterkontrolle mit Konnektionistischen Systemen*, Ph.D. thesis, Informatik Berichte No. 86 (Institute for Computer Science, University of Bonn).

[14] Mühlenbein, H. and Schlierkamp-Voosen, D. (1993) Predictive models of the breeder genetic algorithm I: Continuous parameter optimization. *Evolutionary Computation*, 1:25–49 (MIT Press).

[15] Nilsson, N. J. (1980) *Principles of Artificial Intelligence* (Springer-Verlag).

[16] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986) Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L. (eds.) *Parallel Distributed Processing*, Vol. I (MIT Press), 318–362.

[17] Śmieja, F. and Mühlenbein, H. (1992) Reflective modular neural network systems. Tech. Rep. No 633, German National Research Center for Computer Science (Sankt Augustin: GMD).

[18] Thrun, S. and Möller, K. (1992) Active exploration in dynamic environments. In Moody, J. E., Hanson, S. J., and Lippmann, R. P. (eds.) *Proc. Neural Information Processing Systems 4*.

[19] S. Thrun (1992) The role of exploration in learning control with neural networks. In White, D. A. and Sofge, D. A. (ed.) *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches* (New York: Van Nostrand Reinhold), 527–558.

[20] Weber, F. and Linden, A. (1992) Neural networks for reflective exploration. In *Proc. 2nd Int. Conf. Automation, Robotics and Computer Vision*, INV-8.1.1–INV-8.1.4.

[21] Winston, P. H. (1992) *Artificial Intelligence* (Addison-Wesley).

[22] Whitley, D., Starkweather, T., and Bogart, C. (1990) Genetic algorithms and neural networks: optimizing connections and connectivity. *Parallel Computing*, 14:347–361.

[23] Zhang, B. T. (1992) *Learning by Genetic Neural Evolution: Active Adaptation to Unknown Environments by Self-developing Parallel Networks*, Ph.D. thesis in German, ISBN 3-929037-16-5 (St Augustin, Infix-Verlag). Also available as Informatik Berichte No. 93, Institute for Computer Science, University of Bonn.

[24] Zhang, B. T. (1993) Accelerated learning by active example selection. To appear in *International Journal of Neural Systems.*

[25] Zhang, B. T. (1993) Self-development Learning: Constructing optimal size neural networks via incremental data selection. Tech. Rep. No. 768, German National Research Center for Computer Science (St Augustin: GMD). Submitted to *IEEE Transactions on Neural Networks.*

[26] Zhang, B. T. and Kim, Y. T. (1990) Morphological analysis and synthesis by automated discovery and acquisition of linguistic rules. In Karlgren, H. (ed.) *Proc. 13th Int. Conf. Computational Linguistics*, Vol. I, 431–436.

[27] Zhang, B. T. and Mühlenbein, H. (1993) Genetic programming of minimal neural nets using Occam's razor. In Forrest, S. (ed.) *Proc. Fifth Int. Conf. Genetic Algorithms* (San Mateo: Morgan Kaufmann), 342–349.

[28] Zhang, B. T. and Veenker, G. (1990) Distributed parallel cooperative problem-solving with a voting and election system of neural learning networks. In Eckmiller, E. *et al.* (eds.) *Parallel Processing in Neural Systems and Computers* (North-Holland), 513–516.

[29] Zhang, B. T. and Veenker, G. (1991a) Focused incremental learning for improved generalization with reduced training sets. In Kohonen, T. *et al.* (eds.) *Artificial Neural Networks.* (Elsevier), Vol. I, 227-232.

[30] Zhang, B. T. and Veenker, G. (1991b) Neural networks that teach themselves through genetic discovery of novel examples. In *Proc. Int. Joint Conf. Neural Networks.* (New York: IEEE), Vol. I, 690-695.