Proc. Int. Conf. on Neural Information Processing Systems (ICONIP'94), Seoul, October 17-20, 1994.

Active Learning Algorithms for Neural Networks

Byoung-Tak Zhang
German National Research Center for Computer Science (GMD)
Schloss Birlinghoven, D-53754 Sankt Augustin, Germany
E-mail: zhang@gmd.de

Abstract— Five neural algorithms are described that have been derived from an incremental learning framework, called GENIAL. The GENIAL learning employs four learning elements to adapt the network structure and weights while exploring its environment to acquire novel information. The common feature of all the algorithms is the active use of training data. This includes scheduling the presentation order of given examples, selecting a critical subset of a large data set, and generating new examples. The algorithms are described in the order of increasing activeness or autonomy and compared against four different tasks. The results show improved generalization performance as the autonomy of the algorithm increases.

1 Introduction

In supervised learning applications of neural networks, the learning algorithm changes the weights $\mathbf{w}$ of the network so that the performance of the network improves. In doing this, a fixed set of training examples is presented repeatedly to the network of fixed size. Each time an input vector $\mathbf{x}_m$ is presented to the network, the network computes an output vector $f(\mathbf{x}_m;\mathbf{w})$ which is compared with the desired output $\mathbf{y}_m$. The resulting error $e(\mathbf{y}_m|\mathbf{x}_m,\mathbf{w})$ is propagated backwards to change the weights in the direction to reduce the errors. The effectiveness of this process is usually measured by three factors:

- Training accuracy: A learning method must be able to achieve the desired accuracy for the given training data. This means the error of the network should be reduced to some arbitrarily small value.
- Learning speed: Even with a guarantee of some performance the algorithm may not be useful if the convergence is impractically slow. In some situations learning speed may be more critical than learning accuracy.
- Generalization: One of the most important properties of learning systems is the ability to generalize from previous experiences. The network should be able to produce correct answers to unseen inputs.

Most existing algorithms consider learning in multilayer perceptrons as a problem of weight modification. However the complexity of the network architecture, including the number of layers, units and weights, influences learning performance as well. The dilemma is as follows [2, 5]. If the network is too small, then it may fail to represent the complex relationship between the inputs and outputs in the training set. On the other hand, if the network is too large, the learning can lead to overfitting which results in a look-up-table and achieves poor generalization on unobserved data. Therefore, the network size should be large enough to make sure the learning converges to a reasonable performance level on the training set, but as small as possible to ensure good generalization. Recently, several learning algorithms have been proposed to construct optimal size feedforward networks for specific applications [4]. Each of these methods uses all of the given data for network construction and training.

Another factor that affects the efficiency and effectiveness of learning is the quality of training set. The training set should be representative enough to achieve good results. It is also obvious that too small a training set or lack of examples cannot train the network to a sufficient accuracy. On the other hand, just a large training set will slow down the learning process. Redundant examples may not contribute to increasing generalization performance of the network. This observation motivated several researchers to search for learning algorithms that find a small yet representative training set [1, 3, 10].

The difficulty in determining approriate network size and sample size is that both are interdependent on each other. That is, the necessary training set size depends on the network architecture, and the optimal or minimal network size can be affected by the nature and size of the training set. The GENIAL approach provides a resolution to this dilemma [6]. GENIAL was originally designed to solve the knowlege acquisition problem in unknown or changing environments by neural networks. The basic idea in the GENIAL approach is to start with a small size network and a small training set and let them grow on demand. It determines the architecture and training data actively, instead of passively accepting them as given by the user. A further feature of GENIAL is that it can generate new examples for itself to get new information.

Because of its generality in problem formulation, the GENIAL approach provides novel algorithms for solving conventional problems in neural networks. The purpose of this paper is to identify some subalgorithms of GENIAL which are useful in practice to improve learning speed and generalization performance of multilayer perceptrons. After giving an overview of the GENIAL approach in the next section, we describe five incremental algorithms and their application results.

2 Overview of GENIAL and GENIE

The GENIAL system consists of two learning modules: a neural learning module and a genetic learning module. The neural module is divided into two components of adaptation and development. The adaptation component is responsible for changing the weights of the network and the development component constructs new network architectures. The genetic module is again composed of two components: selection and creation. The selection component filters useful examples from a large set of candidates, while the creation component generates novel examples by applying a genetic algorithm on the existing training set. 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 provides and the results are 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 genetic neural evolutionary learning [6]. The GENIAL system has been implemented in a computer program known as genetic neural intelligence engine (GENIE). Figure 1 shows the architecture of GENIE and the top-level control algorithm of GENIAL.

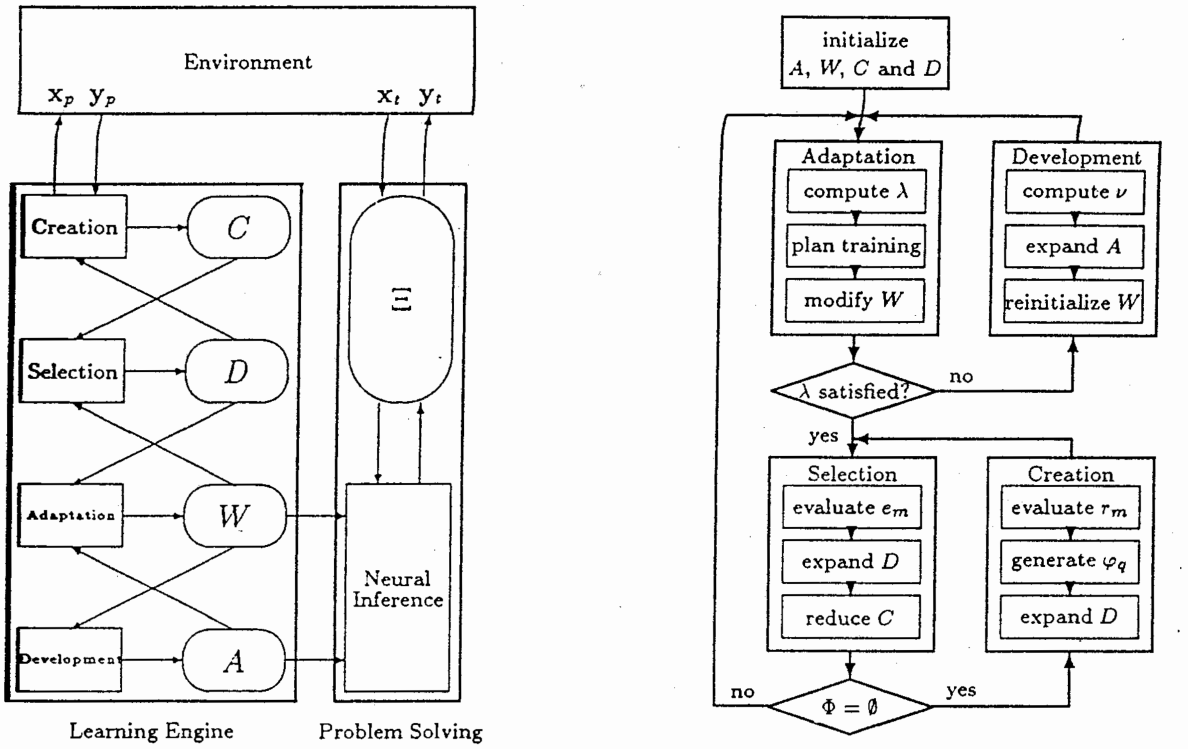

Figure 1: The GENIE architecture and the GENIAL algorithm. See text for meaning of the symbols.

The genetic learning subsystem makes use of four data structures (C, D, A, W) and two operators (F_C and F_S). D contains examples that are used for training the network, while C contains candidate data reserved for being included in D later. A and W are the sets of neurons and weights defining the network. F_C and F_S are operators that transform the example sets. The creation operator F_C generates new candidates C from existing examples in D:

$$C_{s+1} = F_C(D_s, A, W). \tag{1}$$

New examples are created through recombination of existing examples using genetic operators (described below). The selection operator takes as input the candidate set C and expands the training set D:

$$D_{s+1} = F_S(C_s, A, W). \tag{2}$$

The knowledge in the trained network is used to select useful examples. The selection operator F_S is an artificial version of natural selection, the Darwinian survival of the fittest.

The neural submodule of GENIAL makes use of D, A, W, F_D and F_A. D, A and W are as above and F_D and F_A are operators for modifying the network architecture A and weights W. The development operator constructs a new architecture

$$A_{t+1} = F_D(D, A_t, W_t), \tag{3}$$

while the adaptation operator F_A modifies the weights

$$W_{t+1} = F_A(D, A_t, W_t). \tag{4}$$

In the adaptation phase, the network is trained. Training continues until the performance of the network reaches the desired level λ. This results in an improved knowledge base W and the learning continues with the selection phase. Otherwise, the learning continues with the development phase (Figure 1).

3 Five Variants of the GENIAL Learning Procedure

The GENIAL scheme is general enough so that it can be used in various learning situations. In case that the given data set contains sufficient information for learning the task, the creation component may be unnecessary. The example generation ability, however, opens the possibility for neural networks to be applied to unknown or changing environments where new examples have to be sampled during learning. Likewise, the development component may be omitted in applications for which the initially given network size or number of hidden units are sufficient for learning the desired task. Thus the entire GENIAL algorithm can be divided into several subalgorithms which may be used for different application requirements. Table 1 summarizes five of them supported by GENIE.

learning mode	name	development	creation	selection	adaptation
non-incremental learning	FP				x
selective incremental learning	SEL			x	x
creative incremental learning	CSEL		x	x	x
selective developmental learning	SELF	x		x	x
creative developmental learning	CSELF	x	x	x	x

Table 1: Five learning modes of GENIE

In addition to the five learning modes GENIE also supports a performance mode. In the performance or problem solving mode, the input vectors (problems) are presented to the network and output vectors (solutions) are produced at the topmost layer of the network. This is the usual user mode and differs from the learning mode, in that no weights or network architectures are changed in problem solving.

The simplest learning mode of GENIE is called focused propagation (FP) and uses only the adaptation component of the GENIAL learning algorithm. The network size and training set are assumed to be appropriate. This mode is similar to the usual back-propagation procedure:

$$w_{ij}(t+1) = w_{ij}(t) + \epsilon_m(t)\frac{\partial E_m}{\partial w_{ij}(t)} + \eta_m(t)\Delta w_{ij}(t-1) \tag{5}$$

where $E_m = \sum_{i=1}^{O}(y_{mi} - f_i(\mathbf{x}_m; \mathbf{w}, A))^2$ is the total sum of errors for the training set D. A difference is that the learning rate ϵ and momentum factor η are adapted during learning:

$$\epsilon_m(t) = \epsilon_{min} + e_m(t-1), \tag{6}$$

$$\eta_m(t) = \eta_{min} + e_m(t-1), \tag{7}$$

where ϵ_{min} and η_{min} are small constants. The quantity $e_m(t)$ is called criticality of the mth example at epoch t and defined as:

$$e_m(t) = \frac{1}{\dim(\mathbf{y}_m)}\|\mathbf{y}_m - f(\mathbf{x}_m; \mathbf{w}_t, A_t)\|^2. \tag{8}$$

The use of criticality encourages the examples with large error be learned more strongly than the examples with small error. A further extension of this idea is to rearrange the presentation order of training examples according to this measure.

The second algorithm, called selective incremental learning (SEL), includes the selection component into FP. In SEL the network architecture is fixed but the training set size starts small and increases during further learning. This is useful when a large amount of data is known at the outset of learning. The training set D is increased by choosing the most critical example from C:

$$m^* = \arg\max_{m\in C}\{\, e_m \,\}, \tag{9}$$

where the index m runs over all candidate examples. If there are enough candidate examples, λ of them are selected. Choosing examples this way samples the border patterns, as shown in the next section, and thus improves learning speed without reduction of generalization performance.

The third one, creative incremental learning (CSEL), includes the creation component to the SEL algorithm and is useful for applications where the examples are unknown in advance. In the creation phase, the new examples are generated using the existing ones and added to the candidate set. New examples are generated through genetic search in the example space. To create new examples, two parent examples are selected on the basis of their reproductivity $r_m(t)$:

$$r_m(t) = \frac{e_m(t)}{\sum_{q\in C_{\mathbf{y}_m}} e_q(t)} \cdot \frac{1}{M}\left(1 - \frac{N_{\mathbf{y}_m}}{N}\right), \tag{10}$$

where M is the number of possible categories and N denotes the current training set size. $C_{\mathbf{y}_m}$ is the set of examples belonging to the category of $\mathbf{y}_m$ in the current training set and $N_{\mathbf{y}_m}$ is the size of $C_{\mathbf{y}_m}$. The derivation of this equation is given in [9]. In this definition, the examples that are more critical or have produced large error are assigned greater reproductivity than others. This promotes more intensive exploration of the input space where the network shows less performance. The next section shows an application of this method to robot control.

If two parent examples are determined, the genetic operators are applied to generate offspring examples. Many operators are possible. For the experiments we have used the crossover and mutation operators. The mutation operator toggles each input value with some probablity. Mutation is useful for introducing new input values which are nonexistent for the current training set. The crossover operation is used to exchange information between two parent examples. In two-point crossover $\otimes(\mathbf{x}_p, \mathbf{x}_q)$, two crossover sites are chosen at random and the middle segments of each chromosome are exchanged, resulting in two offspring examples $\mathbf{x}'_p$ and $\mathbf{x}'_q$:

$$\otimes(\mathbf{x}_p, \mathbf{x}_q) \quad = \quad \begin{cases} \mathbf{x}'_p = (x^p_1, \ldots, x^p_{a-1}, x^q_a, \ldots, x^q_b, x^p_{b+1}, \ldots, x^p_n) \text{ and} \\ \mathbf{x}'_q = (x^q_1, \ldots, x^q_{a-1}, x^p_a, \ldots, x^p_b, x^q_{b+1}, \ldots, x^q_n), \end{cases} \tag{11}$$

where a and b, $1 \leq a \leq b \leq n$, are crossover sites and n is the number of input elements.

The other two algorithms, SELF and CSELF, are variations of SEL and CSEL by adding the development component to each of them. The test for network expansion or development is done at each time interval Δt by calculating the discounted change of errors

$$\Delta E_{sum}(t) \quad = \quad E(t - \Delta t) - E(t) + \frac{1}{2}\Delta E_{sum}(t - \Delta t) \tag{12}$$

and taking its average

$$\Delta E_{avg}(t) \quad = \quad \frac{\Delta E_{sum}(t)}{N \cdot O} \tag{13}$$

where N is the training set size and O is the number of output units. The development process introduces ν new units into the hidden layer and initiates a new cycle of weight adaptation. The increased network size enhances the learning capacity of the network so that the increased training set can be learned.

While the selective developmental learning (SELF) algorithm only selects examples from the given data set, the creative developmental learning (CSELF) algorithm creates novel examples, thus realizing the complete GENIAL learning procedure. These algorithms try to find a minimal network A using a parsimonious training set D that generalizes well to the entire data space. Specifically, the objective of SELF is formulated as:

$$D_{s^*}, \mathrm{w}^*, A^* \quad = \quad \arg \min_{D_s, \mathrm{W}, A} \sum_{s=0}^{sm} E\left(D_s \cup C_s | \mathrm{w}, A\right). \tag{14}$$

Here sm is the maximum possible number of selection steps given by $sm = \left\lceil \frac{N - N_0}{\lambda} \right\rceil$, where N is the size of the given data set, N_0 denotes the number of seed examples, and λ is the training set increment parameter. When data is abundant, SELF has proven to be very efficient to find an optimal network size for the given data set [8]:

4 Application Results

The algorithms have been applied to a wide class of problems. Here we show their effectiveness in four selected application domains.

Descrete Function Approximation. The SEL was used to solve the four-quadrant problem which is a real-input variant of the XOR problem. The goal function is shown in Figure 2 (a). We used a total of 400 (20 × 20 resolution) examples. The training set was initialized with four seed examples at the four corners. In each selection step, additional four examples ($\lambda = 4$) were added to the existing training set. Figure 2 shows the graphes of the approximated function (c) and the corresponding training points (b) at the 10th selection step. Notice that selected examples lie on the separating lines of output 0 and 1. These are the border patterns and known as critical to solving this problem. Notice also that 44 selected examples, corresponding to 11% of the whole data, are almost sufficient for learning the function.

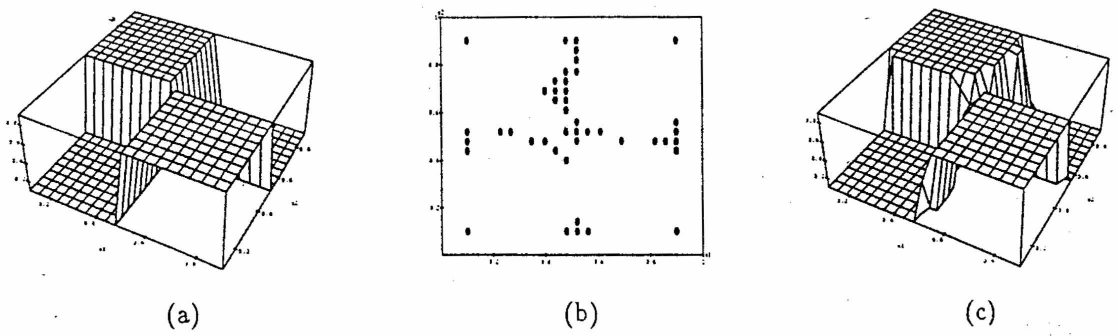

Figure 2: The four-quadrant problem. (a) goal function, (b) data points selected, (c) learned function.

Digit Recognition. The algorithm SELF was applied to the recognition of handwritten digits. We collected 6800 digit patterns written by 10 persons. Each pattern consists of 15 × 10 bitmap. One

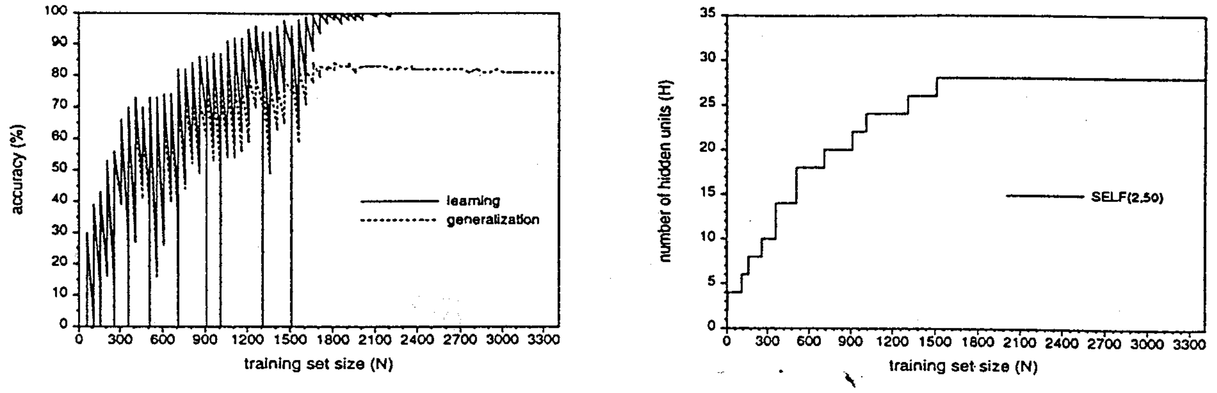

Figure 3: The learning and generalization performance and the network size for handwritten digit recognition, as a function of the training set size. Ther parameters were $\nu = 2$ and $\lambda = 50$.

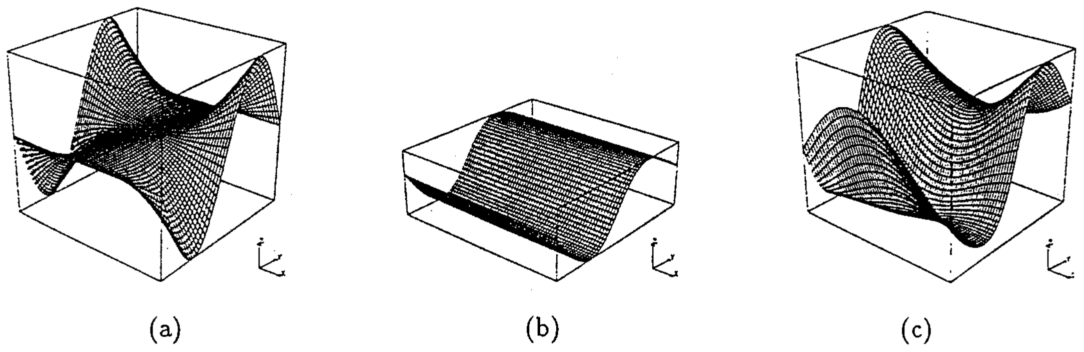
(a) (b) (c)

Figure 4: Comparison of SELF and BP nets. (a) goal function, (b) results of BP, (c) results of SELF learning at the time of $T = 8.1 \times 10^6$ weight modifications.

half of the examples were used for training the network and the other half for testing the generalization performance of the trained network. Figure 3 shows the learning and generalization accuracy as a function of the training set size. Also shown is the growth of network size during incremental learning. Notice that after reaching about the half of the given data size, there is no significant improvement in the generalization performance. This indicates the method has already found a critical subset of the given data. Note also that the network size does not increase any more from this time, either. We also studied the effect of example selection during network expansion by selecting examples randomly and found that network size optimization combined with active example selection generalizes better and converges much faster than with random selection [8].

Continuous Function Approximation. The algorithm SELF was applied to approximate a continuous function. The graph of the function to be learned is shown in Figure 4. Because of its multimodality, this task 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. A total of $11 \times 11 = 121$ examples were 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 ($H_0 = 2$). In each selection step, 10 new examples were chosen to expand the training set ($\lambda = 10$). Each network-expansion step introduced three new hidden units ($\nu = 3$) with a complete connectivity to input and output units. 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 too. The learning rate and the momentum factor of the BP net were the same as the SELF net. Figure 4 shows the performance at $T = 8.1 \times 10^6$ for both methods. This shows 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, resulting in slow convergence. On the other hand, the SELF 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 has the advantage that a good solution can be found quickly.

Robotics. The algorithms with the example creation component, CSEL and CSELF, were used in robot control problems. A robot arm is to learn to follow a rolling ball. The arm has five degrees of freedom (DOF), four of which were used to solve this problem (Figure 5). The rotation of the hand was not considered since it was irrelevant to the task. We used neural networks for learning the inverse kinematics of the arm, i.e. to compute the joint angles $(\theta_1, \theta_2, \theta_3, \theta_4) \in \mathbb{R}^4$ of the arm given the position $(p_x, p_y, p_z) \in \mathbb{R}^3$ of the ball in the work space. A total of 30 input units are used to encode the spatial

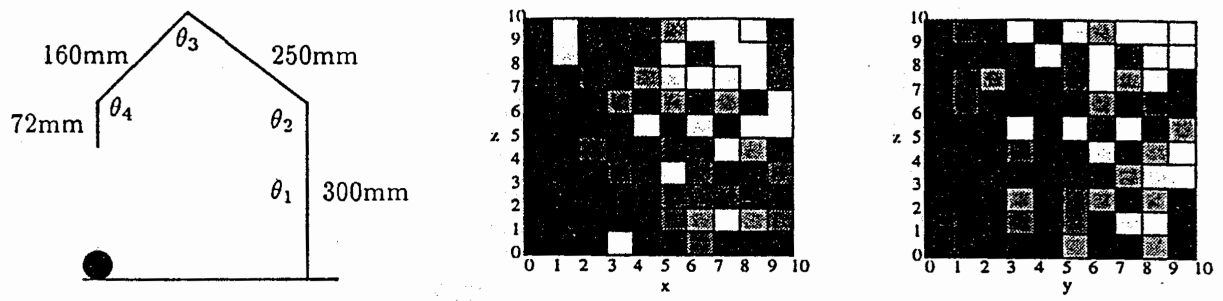

Figure 5: Robot arm and the learning points projected on the xz und yz space

position of the ball. The four joint angles 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 points were produced by a simulated teacher which led the arm to the desired position and measured the joint angles. Figure 5 shows the learning points that were discovered and used to train the network on the yz and xz planes. 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.

5 Conclusion

We described algorithms that actively determine the training examples to adapt the architecture and weights of neural networks. The activeness of the algorithms ranges from the rearrangement of the presentation order of training examples of fixed size, through the incremental selection of given examples, to the creation of new examples by exploring its environment. We showed that active example selection during learning can significantly improve the learning speed and generalization performance for a wide class of problems. Furthermore, active creation of novel examples enables neural networks to be used to solve problems for which the exploration of the environment is inevitable or data collection is expensive. The results also showed that automatic optimization of network size through active data selection improves the efficiency of constructive learning algorithms. On the other hand, the network expansion combined with selective learning allowed a fast escape from local minima, thus increasing the robustness of the data selection algorithms.

References

[1] L. Atlas, D. Cohn, and R. Ladner, "Training connectionist networks with queries and selective sampling," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. Morgan Kaufmann, 1990, pp. 566-573.

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

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

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

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

[6] B. T. Zhang, *Learning by Genetic Neural Evolution*, Ph.D. Thesis in German, Institut für Informatik, Universität Bonn, ISBN 3-929037-16-5, Infix-Verlag, Sankt Augustin, 1992.

[7] B. T. Zhang, "Accelerated learning by active example selection," forthcoming in *International Journal of Neural Systems*, 1994.

[8] B. T. Zhang, "An incremental learning algorithm that optimizes network size and sample size in one trial," in *Proc. IEEE World Congress on Computational Intelligence*, IEEE, 1994.

[9] B. T. Zhang, "Teaching neural networks by genetic exploration," Tech. Rep. No. 805, German National Research Center for Computer Science (GMD), November 1993.

[10] 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.