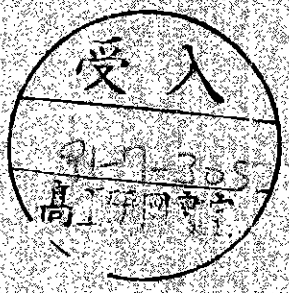
DEUTSCHES ELEKTRONEN – SYNCHROTRON

DESY 91-046
May 1991



General Asymmetric Neural Networks and Structure Design by Genetic Algorithms

S. Bornholdt

Deutsches Elektronen-Synchrotron DESY, Hamburg

D. Graudenz

Inst. f. Theor. Physik, Lehrstuhl E, RWTH Aaachen

ISSN 0418-9833

NOTKESTRASSE 85 · D-2000 HAMBURG 52

To be sure that your preprints are promptly included in the
HIGH ENERGY PHYSICS INDEX,
send them to the following (if possible by air mail):

DESY
Bibliothek
Notkestrasse 85
D–2000 Hamburg 52
Germany

DESY 91-046
May 1991

ISSN 0418-9833

General Asymmetric Neural Networks and Structure Design by Genetic Algorithms

Stefan Bornholdt
Deutsches Elektronen-Synchrotron DESY
Notkestr.85, 2000 Hamburg 52, FRG

Dirk Graudenz[1]
Institut für Theoretische Physik, Lehrstuhl E
RWTH Aachen, 5100 Aachen, FRG

April 29, 1991

[1]Supported by Bundesministerium für Forschung und Technologie under contract number 05 5AC 91P.

Abstract

A learning algorithm for neural networks based on genetic algorithms is proposed. The concept leads in a natural way to a model for the explanation of inherited behavior. Explicitly we study a simplified model for a brain with sensory and motor neurons. We use a general asymmetric network whose structure is solely determined by an evolutionary process. This system is simulated numerically. It turns out that the network obtained by the algorithm reaches a stable state after a small number of sweeps. Some results illustrating the learning capabilities are presented.

1 Introduction

Recently, there has been a lot of interest in the fields of neural networks. In the future, these devices are expected to solve problems that could not be dealt with by conventional von Neumann computers, such as pattern recognition or associative memory problems, which call for highly parallel and error tolerable machines.

Neural networks are basically simplified models of brains. They consist of a number of small computational units imitating neurons and couplings or "synapses" between them. Several architectures specifying these models have been proposed in the past. Among them are the Hopfield model [1] which is closely related to the Ising spin model and different types of multilayer networks [2]. On the cellular level it is rather easy to build a model that imitates a biological neuron and its connection to a neighbor neuron. And indeed the present models do a good job in performing these so-called local learning rules, like the simple Hebb rule [3] or more sophisticated concepts with couplings that are asymmetric or depend on parameters like time or neural activity [4,5].

The problem in building a neural network is not so much to define the local learning rule, but to find out how to arrange the neurons in the net and how to choose their couplings in order to obtain a desired learning behavior. We refer to this latter set of instructions as global learning rules. This can be illustrated with the Hopfield model, a fully connected neural net using the local learning rule proposed by Hebb. This rule assumes that correlations between the states of two neurons determine the coupling between them. Without specifying any global learning recipe, this net already performs like a simple associative memory. It can be improved cosiderably if the couplings are computed by means of the Moore-Penrose pseudoinverse, a global learning rule for the Hopfield network that stores patterns as states of minimal energy [6]. For multi-layer networks error back-propagation [7,8,9] is a powerful tool that is well adapted to this specific network type.

However, it is hard to derive these global concepts from biological observations. The multi-layer structure of networks may well be modeled after some regions of the cortex but already the back-calculation of couplings appears to be unnatural. Also the simple observation that brains do not

entirely consist of layer structures indicates that this may not be the ultimate architecture for any kind of computational problems. The question is whether there is a general principle that can be used for these purposes and that is not tied to a certain network type.

In nature the brain has evolved by trial and error, and it is widely believed that the coarse structure of the brain is determined genetically. Also simple behavioral patterns that are not learned during the lifetime of an individual are determined this way. Genetic algorithms that simulate the process of evolution are capable of solving problems whose complexity does not allow for a direct solution [10].

There have been attempts to apply genetic algorithms to neural networks [11]. However, they still used a predefined architecture of the network, like a two layer feed forward type, and used the genetic algorithm in order to alter the couplings in the fixed neural net. The new concept in the present approach is the application of evolutionary methods to the structure of neural networks itself. The structure of the neural net will be determined by the algorithm and no global learning rule has to be specified for a given problem, except the parameters of the genetic algorithm.

In this paper we propose the application of genetic algorithms to the problem of learning algorithms for neural networks. In section 2 we describe a very simple model that we use to demonstrate our ideas. In section 3 the genetic algorithm is described that we used to train our toy model network. Numerical simulations are presented in section 4. In section 5 we summarize our results and propose possible extensions of our work.

2 The Model

This section will specify the neural network model that is used in our simulations. Section 2.1 gives a brief review of presently used network models and serves as a motivation for our specific model which is described in section 2.2.

2.1 Neural Network Architectures

Let us first have a look at the smallest unit of a neural net, a neuron i with its couplings to other neurons. A neuron in the state S_j sends a signal with strength $J_{ij}S_j$ to neuron i, where J_{ij} denotes the coupling between neuron i and neuron j. Summing all incoming signals at neuron i leads to the so-called membrane potential m_i of neuron i. This potential determines the new state of neuron i, in the simplest case the new state is just defined as $S_i^{new} = sgn(m_i)$. For binary neurons with states $S_i = \pm 1$ which are fully connected by symmetric couplings $J_{ij} = J_{ji}$ this is called a Hopfield-type neural network [1]. Such a simple net of model neurons stabilizes itself via the mutual couplings towards local minima of a global energy function of the system. By cleverly choosing the coupling strengths between the neurons, one can store binary patterns in the system, even more than one at a time, each pattern corresponding to a local minimum in the overall energy function. If the system is put close to a stored pattern then its dynamics will recover the complete stored pattern. This is applied to pattern recognition with great success.

However, the immediate problem that arises is how to choose the synaptic couplings in order to obtain this dynamic behavior. One simple prescription that works surprisingly well has its origin in a work of neurophysiologist Donald Hebb [3]. According to this rule, a synaptic connection that is very active will grow in strength, whereas synapses that are less active become weaker. Hebb's rule allows to store sets of different patterns at the same time in a single neural net and works well in pattern recognition and for associative memory purposes. Indeed it represents a simplified model for memory in biological brains. It is defined on the local level of single synapses and is biologically motivated. Being the simplest of all learning rules it is not necessarily the best of all, and indeed it serves as a basis for a number of more advanced rules. One popular way of improving the storage capacity and the separability of similar patterns is to calculate the synaptic couplings via the so-called pseudoinverse method [12]. However, one has to pay the price that the learning rule is not locally defined anymore, but calculated by some global technique "from the outside" instead. At this point one leaves the terrain of immediate biological motivation for the learning concept.

The second task of the brain besides memory is of course information processing of various kinds. The Hopfield type networks perform rather poorly in this domain. Nets with distinct layers of neurons are more suited to solve these problems. The information to be processed enters the network through an input layer of neurons which influences an output layer in a way that is given by the couplings inbetween the neurons. In these systems, the synapses are in general unidirectional. There may be additional layers of neurons between input and output layer where the information runs through and is being processed. The unidirectional or feed-forward networks of this type are usually called perceptrons [2,13]. It turns out that a two layered perceptron fails to reproduce even such a basic operation like the logical exclusive-or function and one needs to introduce at least one intermediate or hidden layer of neurons to overcome this problem. Again the main task in building such a network is to choose the synaptic couplings in the right way, and once there are more than two layers of neurons, a local prescription like the Hebb rule won't do. The most successful method to train these networks is known as error back-propagation [7,8,9]. The error of the network output is used to calculate corrections to the synaptic couplings and the network is improved in subsequent iterative steps. This works very well in numerous applications. However, no such mechanisms have been found in biological systems and one may ask whether there are any basic biological principles that could be used instead. Our present approach is to train a neural network using a genetic algorithm without specifying any details about the "hidden" architecture. Of course this is not the way our brain learns during its lifetime, but it is plausible that at least those neural structures that are passed on genetically have been developed on an evolutionary timescale. This includes e.g. "hard wired" neural circuitry for heart rhythm or breathing, as well as behavioral patterns that are genetically determined.

2.2 The Model for a Genetically Altered Neural Net

The neural network that will be processed by a genetic algorithm was chosen to be as general as possible and to not constrain the evolutionary process by any intrinsic structures or parameters. The simplest neural net that fulfills these requirement consists of three regions: a set of sensory neurons

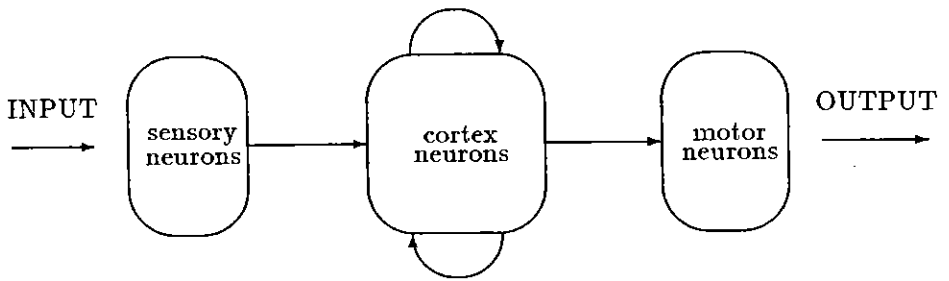


Figure 1: Basic architecture

and one of motor neurons that process input and output information, and an arbitrary set of neurons as a central "cortex" region. The synaptic connections between sensory neurons and cortex neurons are of feed forward type, as are those between cortex neurons and motor neurons. However, in the cortex region we allow arbitrary connections between the neurons. Of course, in a more general approach, one could also introduce direct couplings of feed forward type between input and output neurons. This would correspond to an additional two-layer perceptron type network in the system, or, from a more biological point of view, would allow reflexes between sensory and motor neurons, without any further processing of information through intermediate neurons. Since this is not our main interest at the moment, we will concentrate on the pure cortex version as described above.

Hardly anything else has to be said about the model, since we want to leave it to the genetic algorithm to generate the structure of the neural net. The further features are kept as general as possible. We allow asymmetric couplings inbetween the neurons of the cortex, and the number of synapses to be diluted. This does not mean that the neural net that eventually emerges the genetic algorithm has to be asymmetric or diluted. However, this is the most general concept and the algorithm will be free to choose any degree of asymmetry or dilution. Biological brains usually are highly

diluted and their synapses between two neurons are mostly asymmetric. This is a strong motivation to allow for these features in our model, as well.

A problem that one might expect to encounter in an asymmetric network is their instability [14,15]. They do not necessarily reach a stable state of lowest energy after a finite number of update cycles, but may enter a periodic cycle of different states. Within a genetic algorithm, however, they hardly cause any serious problems since they simply do not survive. To ensure this, the update will be asynchronous. Then the possible periodic cycles vary and it becomes very unlikely that a nonstable asynchronous network remains undetected over a longer period of time until, eventually, it will be singled out by the genetic algorithm.

As for the type of neuron that is used in the model, one is free to choose the one that is suited best for a given problem. In our simulations teaching a Boolean function to a neural network, we took binary, discrete neurons for convenience. However, there is no strong biological motivation for strictly binary neurons, since biological neurons, once they fire, still show some sort of variation of their output through the firing frequency. Since this makes a difference also from the point of view of the genetic algorithm, we will further comment on this point below.

3 The Program

In this section we describe the program that we used for the simulations. In section 3.1 the basic principles of genetic algorithms are reviewed. In 3.2 we describe how the structure for a model brain is implemented in our program. Finally, in 3.3 it is described how mutation and selection act in our example.

3.1 The Genetic Algorithm

Genetic algorithms are capable of solving difficult optimization problems that do not allow for a straightforward solution [10]. Frequently, the results of these algorithms are not the optimal solution, but only an approximation. This can be tolerated if the approximation is not too bad. Genetic algorithms are modelled after the process of evolution in nature [16,17], which

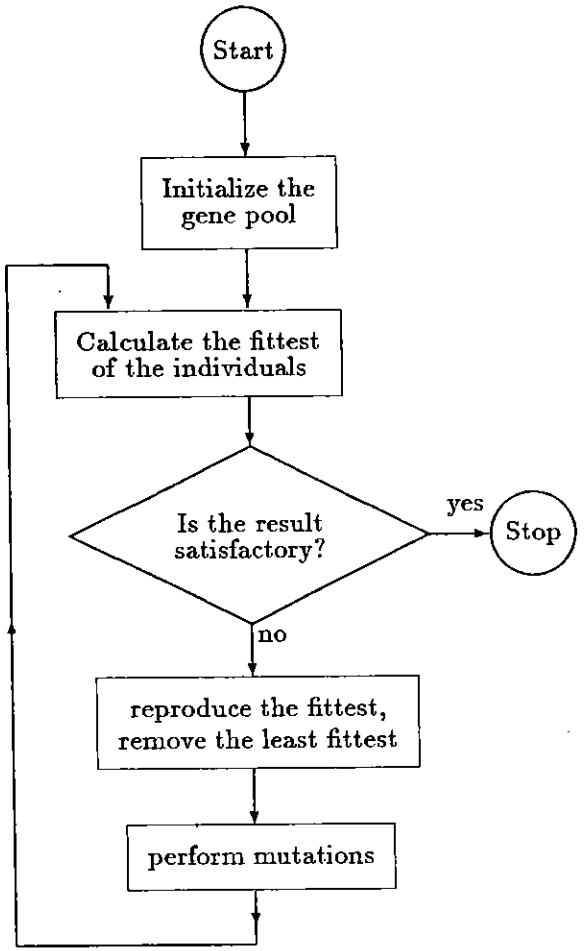


Figure 2: Structure of genetic algorithms

is, loosely speaking, the interplay of small mutations and selection, resulting in a stepwise optimization of organisms. The algorithms apply these principles to technical problems. The main structure of genetic algorithms ist depicted in fig. 3.1 [10].

Individual "organisms" are described by a set of genes which constitutes the genome. The set of all genomes of a population constitutes the gene pool. The genome of an individual determines its phenotype, i.e. all properties of the organism having consequences in the real world [18]. In order to solve a given problem via genetic algorithms, one is interested in

the phenotype of the optimal individuals for a given environment.

A genetic algorithm works as follows. First, the gene pool is initialized, that means, all genes are set to their starting values. This could be a random choice in the allowed parameter range. Then, in every evolutionary step, the fitness of each individual is calculated. If there is an individual that solves the problem, the algorithm stops. If this is not the case, a selection step follows. The fittest individuals are reproduced, the least fittest are removed from the population, and the others simply "survive". Then, in the next step, mutations are performed, that is, the genome of each individual is slightly perturbed with a certain probability. Then the next evolutionary step follows. It is clear that such an algorithm can be implemented on a computer in a very general way, and we think it is not necessary to describe this part of the program in detail.

3.2 Implementation of the Network Model

Now we describe the implementation of the model that has been described in section 2. The network consists of three parts: the input neurons, the "cortex", and the output neurons. Input and output neurons are represented by an array of variables, whereas the cortex neurons are organized as linked lists, a dynamical structure more suitable than static arrays. This keeps memory requirements low. Each node of these linked lists represents a "synapse" that is connected to another neuron. A synapse is then specified by the neuron to which it is connected (by a pointer), and by the coupling strength (a real number). It is obvious that for diluted networks a lot of memory can be saved (in contrast to a matrix of couplings J_{ij}). However, one has to pay a price for this, since a certain amount of computer code is needed for the organization of this more complicated data structure. A benefit is that this dynamical structure is suitable for modification by genetic algorithms, this will be discussed in detail later.

The dynamical evolution of the network is performed by an asynchronous algorithm. The model neurons can be in states $S_i \in \{\pm 1\}$. In each step, an arbitrary neuron i is chosen (at time t). Then the membrane potential m_i is calculated,

$$m_i = \sum_j J_{ij} S_j, \tag{1}$$

and the new state at time $t+1$ is defined by

$$S_i' := sgn(m_i). \tag{2}$$

This corresponds to the dynamics at temperature $T = 0$ of the Hamiltonian

$$H = -\frac{1}{2}\sum_{ij} J_{ij} S_i S_j. \tag{3}$$

Since our goal is to train the network to learn binary functions, we have to test whether the network has learned a specified function. If the function has n input bits and r output bits, we then have to test 2^n possibilities, resulting in $2^n \cdot r$ bits that can be right or wrong. We apply each of the 2^n possibilities to the inputs of the network and set the state of all other neurons to -1, in order to have a well defined initial state. Then the update algorithm is applied until the state of the network doesn't change any more for a certain number of sweeps (three in our program), such that with a certain probability a stable state is reached. Of course, we have to restrict the number of sweeps to an upper limit, since asymmetric networks do not in general settle down to a stable state. In any case the state of the output neurons after the described procedure is the final output state of the network, which can then be compared with the output of the given Boolean function for the specific input. Applying all 2^n input values sequentially, the number of correct output bits can thus be determined.

The number of correct output bits is a measure for the efficiency of the network. In addition, there are other quantities of the network that should be optimized, like the time the network needs to reach a stable state and the number of neurons. In our program the "fitness" of a given network is given by

$$f = \frac{n_c}{1 + 10^{-3} a}, \tag{4}$$

where n_c is the number of correct bits of the Boolean function and a ist the average number of sweeps until the network reaches a stable state. Therefore, a large number of correct bits is favoured, and among the best of them those are favoured that reach the final state fast. The genome of our networks is given by the complete structure of the network and the couplings J_{ij},and the phenotypic expression is given by the Boolean function that is simulated by the network.

3.3 Mutation and Selection

Now that we have described the genetic algorithm and the implementation of the network, we still have to specify the interface between the two parts. This is done by fixing a prescription for mutations and for the selection step. We begin by describing the selection step for a population consisting of P individuals I_i, $i = 1, ..., P$. First their individual fitness $f(I_i)$ is calculated. These numbers are then ordered such that

$$f(I_{\sigma(i)}) \geq f(I_{\sigma(i+1)}). \tag{5}$$

The individuals $I_{\sigma(P-r+1)}, ..., I_{\sigma(P)}$ are removed from the population. Individual $I_{\sigma(1)}$ is copied r_1 times into the population, and individual $I_{\sigma(2)}$ r_2 times, where $r = r_1 + r_2$ (therefore the population size is constant). After this selection step, the mutation step follows. There are, of course, a lot of possibilities for the mutation of a given network. We used the simplest possibility one can conceive of. With a certain probability, we remove a given number n_r of neurons completely from the brain, and add a given number n_a of neurons with numbers $S_1, ..., S_{n_a}$ of synapses with randomly chosen couplings to the network. The choice of the numbers n_r, n_a and $S_1, ..., S_{n_a}$ is described in section 4. The couplings are distributed randomly in the range of -1 to 1 and the synapses are coupled to randomly chosen neurons that are already present in the brain.

One might object that removing entire neurons with all their couplings is not the most subtle kind of mutation one can think of. In fact other prescriptions with additional careful adjustment of single couplings and insertion of single new synapses might be more suited to solve a given problem much faster. The point is, however, that even this crude concept of mutation is suited for improving a neural net through genetic algorithms and that no complicated structure information or global learning rule has to be specified. There are even biological brains that use insertion of new neurons during the learning process, which might serve as a distant biological motivation for our system. The canary is known to learn his songs during early summer with the process of neurogenesis in his brain [19]. A number of new neurons penetrates the existing brain and is built into the brain during the process of memorizing the new songs. At the end of the summer the canary "forgets" the melodies by deleting a number of neurons that are

not used any more. The reason for this is simply that the small bird cannot carry all the neurons that he needs during his rather long life. This analogy should not be stressed too much of course, since in our computational model the process of inserting and deleting neurons happens on an evolutionary timescale, and resembles much more the biological mechanisms of inherited reflexes and instincts. What in fact is similar is the mechanism of learning by adding neurons.

4 Numerical Simulations

In this section we will present a few numerical simulations with the model described above. We trained a population of ten neural nets consisting of binary discrete neurons with Boolean functions. Binary neurons are well suited for Boolean functions and make it easy to decide whether a network has settled down to a stable state. The backdraw however is that part of the fitness function f becomes discrete and undermines the basic principle of evolution, the accumulation of infinitesimally small survival benefits. For instance if we want to teach the system a simple exclusive-or (XOR) function with two input bits and one output bit, the five different final states of 0 to 4 correct bits are far apart. A mutation from a bad network with e.g. 3 correct bits to the next better one with 4 correct bits is rather unlikely.

We first present the teaching of a simple XOR function to a neural net using a genetic algorithm, and move on to more complicated logical functions with finer spaced fitness parameters later. The genetic algorithm generated the corresponding neural networks representing the logical functions in a finite amount of time for a wide range of parameters in the algorithm. The only critical adjustment is the rate of mutation, it has to be chosen as to supply enough new mutations but not devastate previous achievements in the network. The probability for starting to add or starting to delete neurons within the genetic algorithm was chosen to be $p = 0.5$. Then there is always a sufficiently large group of individuals preserving the successful genes. In the case of teaching a simple XOR function, the parameters n_a, n_r were chosen by a random number generator to be in the range $0 - 3$, and the number of synapses per neuron S_i to range from 0 to 7. Using a random

number generator has the advantage that one does not need to introduce new arbitrary parameters into the model. Instead, one offers various kinds of mutations to the algorithm that (hopefully wisely) chooses the ones that are best suited to solve a given problem. Some examples of teaching a simple XOR function to the genetically trained network are presented in the first table.

# of generations	N	S	# of sweeps	D
1200	22	50	4.3	0.10
2400	7	12	2.5	0.25
5000	8	15	1.4	0.23
6000	12	19	2.3	0.13
5500	13	24	2.9	0.14

Table 1: Evolutionary teaching of XOR function

The parameters listed were taken at the end of the evolutionary process when the network completely solved the given problem. The different runs only differ in the starting value of the random number generator for the asynchronous update of the neurons. The number of sweeps gives the average number of complete asynchronous update cycles until the network settles down. The selection rule of the mutation process used in the simulations is given by $r_1 = 4$ and $r_2 = 3$. That is, the two best of the individuals are reproduced four resp. three times and the offspring is mutated. From the old population the three best ones remain unchanged while the seven last ones are removed. The genetic process starts with a minimal brain of one neuron and five randomly chosen synapses. Subsequently, new neurons are inserted and in case they improve the brain, the brain survives, otherwise it is removed. One observes that the brains steadily grow until they reach an average brainsize that is mainly determined by mutation parameters like the number of synapses, but also by the speed of performance, since larger nets become too slow and are removed in the selection step. In general the resulting brains are diluted. Their degree of dilution is defined

by

$$D = \frac{S}{N^2} \tag{6}$$

where S denotes the total number of synapses and N is the size of the cortex region. It is given in the table for the resulting neural nets.

A network with still discrete neurons but several output bits is even better suited for a genetically trained neural net, since the fitness function is much finer spaced and the probability that the fitness function is improved by a single mutation is larger than in the previous case. Therefore Boolean functions with several output bits have been used for another set of simulations. The functions that were chosen have two input bits and six output bits in the following configurations:

- a Boolean function corresponding to six parallel XOR functions,
- six parallel OR functions,
- four parallel OR combined with two parallel XOR functions,
- a mixed Boolean function with randomly chosen bits, and
- six parallel XOR functions with one false bit.

For the simulations of these functions the same parameters for the genetic algorithm have been used as before, except for the parameters n_o, n_r which were now chosen by a random number generator to be in the range $0-7$. Some typical runs are presented in the table.

Boolean function	# of generations	# of sweeps	N	D
6fold XOR	2400	7.3	42	0.053
6fold OR	600	4.8	30	0.079
4fold OR, 2fold XOR	4500	4.9	44	0.046
MIXED	2700	4.0	40	0.078
6fold XOR, 1 false bit	5400	8.2	49	0.052

Table 2: Evolution of Boolean functions

Again the parameters listed in the table have been taken at the end of a teaching cycle, when the given function was learned by the network without error.

The Boolean functions with two input bits considered so far correspond to four simultaneously stored patterns in the neural net. An example with a higher number of patterns is the parity function that gives a one in the output for an odd number in the input and a zero for an even number. It is a generalization of the XOR function. For the case of five input bits the neural net has to store 32 seperate activation patterns. Results of a simulation of a 5 bit parity function with the genetically trained network are shown in the last table.

Correct bits	N	S	# of sweeps	n_s	r_1	r_2
29	43	84	6.5	7	3	2
30	62	124	6.5	7	3	2
30	51	139	11.4	9	4	3
32	61	134	7.5	9	4	3

Table 3: Simulation of a 5 bit parity function after 10.000 generations

Here, the number of correct bits of the 32 tested patterns is listed after 10.000 generations. The parameters of the genetic algorithm have been chosen as before except for the three listed on the right. n_s gives the maximum number of synapses per neuron. 3000 generations was the minimum number we observed for a network to learn the 32 bits of a parity function correctly during the simulations. The generation of the hidden cortex unit is not controlled by any other prescription besides mutation and selection.

5 Summary and Outlook

It has been shown that it is possible to train neural networks using genetic algorithms without the need to specify any global learning rule for the synaptic couplings. The hidden region of the network does not have to be specified in detail since its architecture is generated by the genetic algorithm.

This model is biologically well motivated by genetically determined neural structures like brains that are found in nature in a large variety. The simulations presented in this paper provide a simple model for genetically determined behavior in biological organisms.

Numerical studies have been successfully performed, teaching Boolean functions to a small network consisting of asymmetric neurons and using asynchronous update.

This technique is well suited for training truly parallel neural network computers, since the couplings are altered locally and there will be no need for a global daemon that calculates the couplings, as in systems that use error back-propagation.

Future improvements concentrate on refined mutation concepts, to alter single couplings or the number of synapses of a single neuron, additionally to the here used mutation by adding and removing complete neurons. Furthermore, the fitness-function of a system could be chosen to be finer spaced, in order to take better advantage of small differential survival benefits, the key ingredient of evolutionary processes, rather than rely too much on chance. This can be achieved by a different measure or even introducing continuous neurons during the learning phase. The latter would not be too far fetched from the biological point of view, since biological neurons are able to continuously adjust their firing rate in a certain range.

Genetically generated neural networks are neither tied to any specific global learning rule nor to any special class of computational problems. Therefore, genetic networks may well be a promising concept in the future.

Acknowledgements
The authors thank H. Joos and C. Wetterich for critical comments on the manuscript and Ray D. Koluvek from the DESY computer center for the opportunity to use an IBM RISC station and for great support.

References

[1] J.J. Hopfield: Neural Networks and Physical Systems with Emergent Computational Abilities, Pro. Natl. Acad. Sci. USA 79, 2554 (1982)

[2] H.D. Block: The Perceptron: A Model for Brain Functioning, Rev. Mod. Phys. 34, 123 (1962)

[3] D.O. Hebb: The Organization of Behavior: A Neurophysiological Theory, Wiley, New York (1949)

[4] D. Kleinfeld and H. Sompolinsky: Associative Neural Network Model for the Generation of Temporal Patterns, Biophys. J. 54, 1039 (1988)

[5] D. Horn and M. Usher: Neural Networks with Dynamical Thresholds, Phys. Rev. A 40, 1036 (1989)

[6] T. Kohonen: Self-Organization and Associative Memory, Springer (1984)

[7] Y. Le Cun: Learning Process in an Asymmetric Threshold Network, in: NATO ASI Ser. F 20, Springer (1986)

[8] D.B. Parker: Learning-Logic: Casting the Cortex of the Human Brain in Silicon, MIT Techn. Rep. TR 47, (1985)

[9] D.E. Rummelhard, G.E. Hinton, and R.J. Williams: Learning Representations by Back-propagating Errors, Nature 323, 533 (1986)

[10] D.E. Goldberg: Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley (1989), and references therein

[11] D.J. Montana and L. Davis: Training Feed Forward Neural Networks Using Genetic Algorithms, BBN Systems and Technologies, Cambridge, Mass. (1989)

[12] L. Personnaz, I. Guyon, and J. Dreyfus: Collective Computational Properties of Neural Networks: New Learning Mechanisms, Phys. Rev. A34, 4217 (1986)

[13] H.D. Block B.W. Knight Jr., and F. Rosenblatt: Analysis of a Four-Layer Series-Coupled Perceptron, Rev. Mod. Phys. 34, 135 (1962)

[14] B. Derrida and R. Meir: Chaotic Behavior of an Layered Neural Network, Phys. Rev. A 28, 3116 (1988)

[15] B. Müller and J. Reinhard: Neural Networks: An Introduction, Springer (1990)

[16] C.R. Darwin: The Origin of Species, Harmondsworth (1859)

[17] R. Dawkins: The Blind Watchmaker, Longman (1986)

[18] R. Dawkins: The Extended Phenotype, Oxford University Press (1982)

[19] F. Nottebohm: From Bird Song To Neurogenesis, Scientific American 2, 56 (1989)