July 1993 HD–THEP–93–26
LBL–34384

General Asymmetric Neural Networks and Structure Design by Genetic Algorithms: A Learning Rule for Temporal Patterns *†‖

Stefan Bornholdt‡
Institut für Theoretische Physik
Universität Heidelberg
Philosophenweg 16, 69120 Heidelberg, Germany

Dirk Graudenz ‖§
Theoretical Physics Group, LBL
University of California
1 Cyclotron Rd., Berkeley, CA 94720, USA

Abstract

A learning algorithm based on genetic algorithms for asymmetric neural networks with an arbitrary structure is presented. It is suited for the learning of temporal patterns and leads to stable neural networks with feedback.

**This work was supported by the Director, Office of Energy Research, Office of High Energy and Nuclear Physics, Division of High Energy Physics of the U.S. Department of Energy under Contract DE-AC03-76SF00098.*

†This work was supported in part by the Studienstiftung des deutschen Volkes.

‖published in: Proceedings of the IEEE Conference on Systems, Man, and Cybernetics, Le Touquet, France, 1993; neuroprose INDEX bornholdt.garnn.ps.Z, bornholdt@theo-physik.uni-kiel.de, 8 pages, genetic training algorithm for diluted recurrent neural nets with asynchronous update

‡Address after August 1, 1994: Institut fuer theoretische Physik, Universitaet Kiel, Olshausenstr. 40, D-24118 Kiel, Germany. email: bornholdt@theo-physik.uni-kiel.de

‖supported by Max Kade Foundation, New York

§Address after January 1, 1994: CERN, Theoretical Physics Division, CH-1211 Geneva, Switzerland. email: graudenz@surya11.cern.ch, graudenz@theorm.lbl.gov

Disclaimer

1 Introduction

During the last decade, a highly active research field has developed dealing with artificial neural networks (ANN), which aims at developing new computational techniques as well as trying to get an understanding of brain functioning. In recent years, neural network concepts have been successfully combined with genetic algorithms [1]. From the engineering point of view, combining two powerful tools is a promising challenge, whereas in a more biologically inspired approach one is naturally led to evolutionary methods, since in nature brains and hard-wired neural structures are to a great extent determined by genetic encoding created through evolution. Most work on combining artificial neural networks and genetic algorithms has been done under the "engineering paradigm" of competitiveness with the standard training algorithms for artificial neural networks, like back propagation [2], which, in our view, is a constraint on the potential of genetic algorithms too strong in the early stage of the art. Many attempts have been made to assist standard learning algorithms by genetic algorithms, either by preparing the initial data for the neural net [3], or by letting the genetic algorithm choose the initial weights ("synapses") for learning through back-propagating errors [4]. A more general concept is to use a genetic algorithm to determine the topology of a neural net. Most work in this direction applies the idea to the architecture of feed forward nets with some local learning rule choosing the weights (while the genetic algorithm decides which weights are supposed to be different from zero) [5]. Although combining genetic algorithms with standard local learning rules might be more efficient than using genetic algorithms alone to choose the weights, one has to pay a price: The standard learning methods are in general restricted to fixed types of nets, like a strict feed forward architecture. In our view, one of the main advantages of the use of genetic algorithms is that they can provide learning rules for neural nets with arbitrary structures where conventional learning rules are not available. In particular this applies to networks with feedback as required for the training of temporal patterns. More sophisticated systems of this genetic mechanism for the evolution of the network topology could be useful for the solution of complex problems related to the detection of pattern sequences in time (e.g. speech recognition, analysis of picture sequences, etc.).

In this article we take a more biologically motivated view and allow for arbitrary connections between the neurons of an in general diluted model-brain (the terms neuron, brain, etc. are used in the following for the constituents of the simulated models which are understood to be only very crude approximations to the natural equivalents). Especially this includes asymmetric connections, backwards directed connections, and feedback loops. These kinds of connections are clearly observed in nature and we see no reason to exclude them from our model. However, there is no basic training concept for such general neural nets. In particular for the learning of temporal patterns one has to introduce some form of feedback into the neural net which makes standard learning rules inapplicable. Extended versions of standard learning rules for feedback nets are plagued by severe dynamical problems [6]. However, one can show [7] that a genetic algorithm can provide a useful learning rule for general asymmetric neural networks.

In this work we extend the genetic algorithm based learning concept to neural networks with feedback. It is applied to the training of temporal patterns with an emphasis of the network architecture and the learning dynamics. To allow for an arbitrary network structure, we will divide the neurons of the model into three groups: input neurons, "cortex" neurons, and output neurons. The connections from the input to the cortex and from the cortex to the output neurons are of the feed forward type, while connections inbetween the cortex neurons are arbitrary, i.e. they may be asymmetric or backwards directed thus creating loops, etc. In conventional terms this includes within-layer-links and feedback-links, however, note that the present cortex architecture is not grouped into layers but that the topology within the cortex is completely arbitrary. Since we allow for an arbitrary network structure there is no simple linear representation of its architecture. In our present simulations we therefore use a one to one mapping from the genome onto the network and do not use crossing over (It is certainly desirable to find a representation that allows for crossing over in order to take advantage of the combination of building blocks which would yield a very fast increase of the fitness in every recombination step. In nature, however, evolution did not start with sexual reproduction from the very beginning; this feature was implemented when individuals became too complicated in order to allow for a large mutation rate). A genetic encoding of every neuron with all its connections is not implemented in the ontogeny of biological brains [8]. However, our present approach is motivated by small hard wired neuron structures in nature like those for heart rhythm or oscillatory processes in primitive animals.

Asexual reproduction is best suited for the evolution of small systems. For the following simulations we chose to train Boolean functions and time sequences of Boolean functions which are simple enough but nevertheless include the most general data structure and can be applied to any "real world" problem. The next section describes the program used in the simulations followed by a section presenting our results.

2 The Simulation Program

The simulations described in this paper have been done with the program `GARFIELD` (which is an acronym for `General Asymmetric ReFIned Evolutionary Learning Device`). This program consists of two basic parts: the genetic algorithm `CREATOR`, and the neural network management part `DEEP_THOUGHT`.

`CREATOR` sets up the initial population, determines the fitness and controls the mutation and selection step. In the current version of the program the genome of a network is the network structure itself, so that there is no mechanism for crossing over of genomes. The algorithm therefore solely uses mutations as a driving force. The fitness of a member of the population is determined by applying to its inputs the input Bits of the Boolean function or time sequence that is to be learned. Then the update of the network is performed until it either reaches a stable state or the number of updates exceeds a certain limit. The output status of the network is then compared with the required output, and the fraction of correct Bits is determined. This procedure is performed for every possible input combination in the case of Boolean functions or every sequence of inputs in the case of time sequences. The fitness is calculated as a function of the fraction of correct Bits and other quantities like the size of the cortex (in analogy to the power consumption being a constraint in biological systems) and the average number of updates that was required to reach a stable state (being a measure for speed and stability). The fitnesses of all members are ranked in increasing order. The selection step picks out a certain number of the networks with a large fitness that replace some networks of lower fitness. Finally, in the mutation step, the networks are modified in a random way by removing neurons from the network or inserting neurons with a specified average number of synapses (with a random coupling between -1 and 1).

`DEEP_THOUGHT` is a package of functions that allows the management of a very flexible structure of neural networks. The network data structure is based on linked lists. This representation has several advantages: it is memory efficient, the networks are easy to maintain, the update of the network structure is quite fast, the mutation operators have a simple implementation, and the network structure is reflected directly in the data structure. The approach described in this paper is based on the assumption that every individual neuron has only a small number of synapses that are connected to the outputs of other neurons. The topology of these connections is not fixed by an initial assumption about the architecture of the network. Certainly a representation by a matrix whose entries are the coupling strengths between the neurons would be possible, but most of the entries in such a matrix would be zero (under the assumption of a sparse network), and therefore the update algorithm would be too slow to be effective. In the approach based on linked lists, a network is described by a linked list of neurons, each of which in turn possesses a linked list representing its synapses. These synapses have a pointer to the neuron from which they get their input. To update the state of a neuron the algorithm simply has to traverse the linked lists of the synapses and add up the weights associated with the synapses multiplied by the output of the neuron they are pointing to.

3 Genetic Training and Temporal Patterns

The simulations that are described in the following deal with supervised learning of Boolean functions and time sequences of Boolean functions. No additional local learning rules are implemented besides the genetic algorithm. The weights and the topology of the neural net are contained in the genotype. Therefore one is free to choose a very general architecture of the net. As stated above, we will group our neurons only in three regions of input, cortex, and output neurons. Binary threshold neurons are used while the weights take on continuous values in the range from -1 to 1.

Asymmetric neural networks do not in general settle down to a stable state [9]. Problems of this type are avoided by defining a maximum allowed number of sweeps. Then the stable nets are sorted out by the genetic algorithm through asynchronous update of the net in randomly chosen order.

We now describe the genetic training of two small tasks, firstly a five Bit parity function to demonstrate the ability of the present algorithm to genetically train stable Boolean functions, and secondly a small counting problem to demonstrate the learning of a temporal pattern.

For the training of a five Bit parity problem we choose a population of 10 individuals. Each individual starts with a cortex of one single neuron, 5 input synapses and one output synapse. In the selection step, a number of $2n_r$ individuals are removed from the population and replaced by n_r copies of the two best individuals each with $n_r = 2$ in this example. Then, the complete population is mutated (whereas in the second simulation described below we will only mutate the new individuals). In the mutation step new genotypes are produced by inserting neurons with a limited but random number of randomly chosen synapses with randomly chosen weights in the range from -1 to 1, and by removing complete neurons with all their synaptic connections. The numbers of neurons which are added or removed are also limited and randomly chosen. The processes of adding or deleting neurons are put in action with probability of 0.5 each. The numbers of added or deleted neurons are limited to 3 resp. 2. Also the numbers of new synapses per neuron are limited (5 'input' to 'neuron', 3 'cortex' to 'neuron' and 'neuron' to 'cortex' and 1 'neuron' to 'output'). The creation process for each type of synapses is started with probability 0.5. The cortex size is limited to 100 neurons while the age of an individual is not constrained. For the fitness function we choose

$$\text{fitness} = \frac{f_c}{(1 + 0.01 * n_s + 0.0001 * N_c)}, \tag{1}$$

where f_c denotes the fraction of correct output Bits, n_s the average number of sweeps until the network reaches a stable state, and N_c is the number of neurons in the cortex. That means, for two individuals with equal numbers of correct Bits, the faster one wins, and of those with similar score and speed, the smaller one is considered to be better.

The results of four simulations with different initializations of the random number generator are presented in Figs. 1 – 5. The parity function is usually learned in less than 10000 generations. In Table 1 we give the properties of the emerging neural nets at the point where it first solves the given task where N denotes the total number of neurons in the cortex, S the number of synapses and D the degree of dilution defined by $D = S/N^2$. In Fig. 1 the fitness

generation	N	S	sweeps	D
700	100	457	7.5	0.0457
800	57	268	6.7	0.0825
1100	96	425	7.7	0.0461
4600	101	425	7.4	0.0417

Table 1: NEURAL NETS THAT SOLVE THE 5 BIT PARITY PROBLEM

of the four different nets for the first 2000 generations is plotted. One realizes a continuous increase during the first few hundred generations. Due to the asynchronous update one arrives at totally different net architectures in different runs, however the learning behavior is astonishingly uniform. Also the number of neurons (Fig. 2) and synapses (Fig. 3) shows a similar evolution. The fraction of correct Bits (Fig. 4) increases quickly during the first few hundred generations, while the last steps of improvement take longer. This is sometimes called the "hare strategy" as opposed to the the "tortoise strategy" of slow and steady increase. These learning behaviors are determined by the choice of the parameters of the genetic algorithm. The number of sweeps until the nets are stable (Fig. 5) is increasing in the phase of growth, and then slightly decreases in the more stable phase due to the pressure of the fitness function that prefers fast individuals. A picture of one of the neural nets is shown in Fig. 11, where the squares are input neurons and the discs are cortex neurons. Weights greater than zero are denoted by lines, weights smaller than zero by dotted lines. One clearly sees that this net is diluted, thus the encoding by conventional matrices would be very unhandy.

The second task under investigation is a counting problem for the number of ones in an input sequence applied to a single input neuron. The population is now chosen to contain 20 individuals with the starting configuration of one cortex neuron with only one input and one output synapse. The probabilities for creating or deleting new neurons are set to 0.7, the probabilities for creating new input synapses to 0.4 and for all other types of synapses to 0.8. The fitness function is here

$$\text{fitness} = \frac{f_c}{(1 + 0.001 * n_s + 0.000001 * N_c)}, \tag{2}$$

and $n_r = 5$. All other parameters are equal to the previous case. The sequence of Boolean functions applied to the net is given in Table 2. This set of sequences is successfully learned by

input	desired output
0	00
1	01
1	01
0	01
1	10
1	01
0	01
1	10
0	10
1	11

Table 2: SEQUENCES OF INPUT FUNCTIONS FOR THE COUNTING PROBLEM

the genetically trained neural net. Remarkably, the required learning time can be reduced considerably by checking this sequence several times (three times in our simulations) to determine the fitness. The reason is that the asynchronous update is different every time and requiring the results to be reproducible under asynchronous update leads to more robust networks. This selection process works faster when testing this "robustness" already at the individual level during one generation. In Fig. 6 the fitness curve of the learning of this temporal sequence is shown and in Fig. 7 the number of correct output Bits. The latter increases rapidly during the first few hundred generations and after 2000 generations networks emerge that perfectly fulfill the given task. However, due to the asynchronous update and the requirement that the network gives correct results three times in a row, still single errors occur, which are straightened out after about 3000 more generations. In Table 3 the parameters of the net after learning the counting problem are shown. The average number of sweeps that are required to stabilize the neural net after a pattern is presented to it is shown in Fig. 8. Again, it increases with the complexity of the emerging neural net. The average cortex size (Fig. 9) increases to about 20

generation	N	S	sweeps	D
1600	20	169	8.5	0.4225

Table 3: NEURAL NET THAT SOLVES THE COUNTING PROBLEM

neurons and after the task is fulfilled, it slightly decreases since this is again favored by the fitness function. The number of synapses (Fig. 10) shows a similar behavior. The emerging brain structure of this example is plotted in Fig. 12. One realizes the diluted nature and the statistical origin of the architecture.

In conclusion: It has been demonstrated that genetic algorithms provide a learning rule for general asymmetric neural nets which is suited for learning temporal patterns without any assumptions on the topology of the emerging network. The learning capability has been demonstrated with a 5 Bit parity problem for a static diluted neural net. The successful training of a counting neural network shows the ability of the learning rule to choose the weights in a way that lead to feedback loops in the neural network that store the information necessary for counting. This mechanism turns out to be robust against asymmetric update of the neural network. Genetic algorithms prove to be useful for training and choosing the weights of neural networks where standard learning rules fail.

4 Acknowledgments

We wish to thank the computer centers of Heidelberg University and LBL for support and computer time.

References

[1] J.D. Schaffer, D. Whitley, and L.J. Eshelman, "Combinations of genetic algorithms and neural networks: A survey of the state of the art", in: COGANN-92: International workshop on combinations of genetic algorithms and neural networks (1992: Baltimore, MD), Los Alamitos, CA: IEEE Computer society press, 1992, and references therein.

[2] D.E. Rumelhard, G.E. Hinton, and R.J. Williams, "Learning Representations by Back-propagating Errors", Nature **323** (1986) 533. Y. Le Cun, "Learning Process in an Asymmetric Threshold Network", in: NATO ASI Series, **F 20**, New York: Springer-Verlag (1986). D.B. Parker, "Learning-Logic: Casting the Cortex of the Human Brain in Silicon", MIT Technical Report, **TR 47** (1985).

[3] J.D. Kelly and L. Davis, "Hybridizing the genetic algorithm and the K nearest neighbor classification algorithm", in: R.K. Belew and L.B. Booker (eds.), "Fourth international conference on genetic algorithms", San Mateo, CA: Morgan Kaufmann (1991).

[4] R.K. Belew, J. McInerney and N.N. Schraudolph, "Evolving networks: Using genetic algorithms with connectionists learning", CSE technical report CS90-174, La Jolla, CA: IEEE (1990).

[5] J.D. Schaffer, R.A. Caruana and L.J. Eshelman, "Using genetic search to exploit the emergent behavior of neural networks", in: S. Forest (ed.), "Emergent computation", Amsterdam: North Holland (1990).

[6] H.-U. Bauer and T. Geisel, "Nonlinear dynamics of feedback multilayer-perceptrons", Phys. Rev. **A42** (1990) 2401.

[7] S. Bornholdt and D. Graudenz, "General asymmetric neural networks and structure design by genetic algorithms", Neural Networks **5**, (1992) 327.

[8] G.M. Edelman, "Topobiology : an introduction to molecular embryology". New York : Basic Books (1988).

[9] B. Derrida and R. Meir, "Chaotic Behavior of a Layered Neural Network", Physical Review, **A 38** (1988) 3116–3119.

Figures

- Figure 1: Fitness (5 Bit parity)
- Figure 2: Number of neurons (5 Bit parity)
- Figure 3: Number of synapses (5 Bit parity)
- Figure 4: Fraction of correct Bits (5 Bit parity)

- Figure 5: Number of sweeps (5 Bit parity)
- Figure 6: Fitness (Counting problem)
- Figure 7: Fraction of correct Bits (Counting problem)
- Figure 8: Number of Sweeps (Counting problem)
- Figure 9: Average cortex size (Counting problem)
- Figure 10: Number of synapses (Counting problem)
- Figure 11: Brain architecture (5 Bit parity)
- Figure 12: Brain architecture (Counting problem)