A Genetic Breeding Algorithm Which Exhibits Self-Organizing in Neural Networks

H. B. Penfold† O. F. Diessel† M. W. Bentink‡

† Department of Electrical Engineering and Computer Science,
The University of Newcastle, N.S.W. 2308 Australia
‡ Honeywell Limited, Newcastle, Australia

Abstract

Genetic algorithms have been extensively assessed for application to the training of neural networks. The great majority of the genetic algorithm approaches have thus far exploited a technique inspired by the process of mutation where populations of neural networks experience random - and usually small - perturbation in the connecting weights and neuron biases. Those which perform best in a test are retained for future generations.

In this paper we introduce a genetic algorithm which incorporates the notion of meiosis. That is, the process through which the genetic code of offspring acquires equal contributions from each parent. We are thus able to provide a breeding model which can incorporate many of the features of evolutionary processes, especially the retention of genetic information in the correct proportions, from generation to generation. Results obtained from an algorithm based on these ideas when tested on two well-known problems are reported and display good convergence properties.

KEYWORDS: Neural Networks, Self-organizing, Evolution, Genetic Algorithms

1 Introduction

The dominant techniques in the training of neural networks tend to be variants of the backpropagation method due to Rummelhart *et al* [1]. The feature shared by all such methods is that they apply only to forward-propagating layered networks in which all the interconnecting weights are accessible to modification in the light of some objective function based on the performance of the network at a training set. There is implied the requirement that *all* negative feedback (or learning) be external to the network, and have a form arbitrarily prescribed by the learning algorithm. It is well known that these methods suffer from convergence problems in that they can be 'trapped' by local minima, and that convergence rates fall dramatically for large networks [2].

Optimisation techniques other than backpropagation have been evaluated in an attempt to add to the gradient-descent properties of that method desirable stochastic attributes. This work, of which simulated annealing is an example, has a long history [3] and employs random perturbations to the weights of a network in an attempt to span the space of the network in search of the optimum solution. Such methods are reminiscent of the biological phenomenon of mutation in which small random changes are presumed in the describing gene of an organism to allow it an opportunity to better 'fit' its environment. In a like manner to stochastic gradient descent such organisms are thought to retain variations found advantageous.

We extend this biological metaphor in this paper and observe several motivating issues which are thought to add impetus to this line of investigation.

- It is desirable to extend our capability in the training of neural networks to include networks with a 'general' connection. That is, feedback may be *internal* to the network.
- There is the potential to *learn about learning*. If we could apply a genetic algorithm to the above general network where the objective function, instead of being performance at a prescribed task, is the ability to *learn* a class of tasks, then the structure disclosed by the network may hold useful insights.
- The parallel with biology may in fact be reversible so that theories of evolution could be quantified by using neural networks as example populations of elementary species.

We proceed in a manner analogous to biology by defining a 'gene', in this case an ordered set of numbers representing the weights of connections into a neuron from others in the net, and the bias of that neuron. An ordered set of genes representing the connectivity of the whole network constitutes a 'chromosome'. The process of breeding requires that genes be selected from the chromosomes for each parent network. We also generate the complementary 'sibling' network to ensure that

the law of genetic retention is observed [4]. During the breeding process, mutations are permitted to occur, their incidence and severity being governed by a measure of the genetic diversity in the population.

The algorithm typically begins with a specified number of randomly generated nets (generation 0), and produces generation 1 according to rules. The members of this generation are tested against the required task and culled on the basis of performance to derive the parents of generation 2. If the population performance improvement stagnates, the probability of mutation is increased.

In tests the algorithm has successfully derived nets which, from a random initial condition, have organised into a layered network which can perform tasks such as the 2-input XOR function, an incomplete parity problem –both of which are reported here – and several real-valued continuous algebraic functions.

2 Methodology

We shall adopt as our neural network a conventionally structured network comprising layers of neurons with each neuron generating an output which is a function of the weighted sum of the inputs. A bias input or offset is also represented. That is, the output of neuron j, which has inputs i for $i = 1, \ldots, n$ is given by -

$$\text{output}_j = \mathcal{F}_j(\sum^{i} \text{input}_i \times \text{weight}_i + \text{bias}_j)$$

where $\mathcal{F}_j$ is taken in this work to be a sigmoid function.

For a genetic algorithm to be definable, we need to resolve the structure of a data string which we regard as analogous to a gene, a method for combination of sections of this data string which is analogous to breeding, and the mechanism for variations to the data string which is analogous to mutation.

2.1 Gene Definition

There is considerable latitude in the selection of a suitable data string of real numbers (weights and biases) such that it will represent an arbitrary network. We have chosen to represent the connectivity of a network as a matrix of weights where the real number elements (in the range -1 to +1) represent the degree of interconnection of the inputs, biases, and other neurons. We propose that this representation is general in the sense that it permits other than a layer-to-layer forward propagating network to be represented. That is, inputs may bypass a layer, and equally, the output of a neuron may feed back to an 'earlier' layer. Such possibilities for internal feedback mark a departure from the model in which feedback occurs in a mechanism external to the network.

An example of this connection matrix is given in Figure 1, which represents a 2-input, 2-layer network having 2 neurons per layer. The choice of the characteristics of this matrix which we consider to be 'genes' which we assemble into a 'string to represent the 'chromosome' for this network is open, and the possibilities are indicated on the diagram. Structure Q represents all inputs to a neuron from a layer; structure R represents all outputs from a neuron. Structure S represents all inputs to a neuron; T represents all inputs to a layer from a layer; U represents all outputs from a neuron to a layer.

We have chosen structures of the form Q, the inputs to a neuron from layers as representing the network. The string of 'genes' then is a vector of input weights to a neuron taken in turn, together with its bias, and repeated for each neuron in the network.

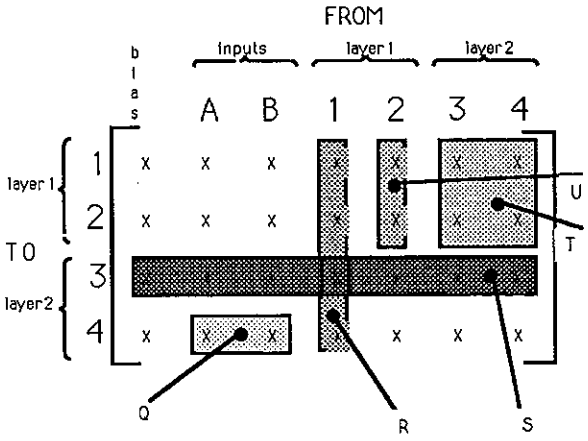

Figure 1: Connection Matrix

2.2 Breeding

Most work which represents a 'breeding' function between networks uses the crossover operator [5] in which all bits up to a randomly chosen position in the gene of one parent are copied to the offspring, with the remaining bits being copied from the other parent. Whilst this is well founded in optimisation theory (and represents the intersection of two hyperplanes in the optimisation space) we have chosen to depart from this practice. Instead each gene' in the offspring – representing the weight or bias for one input to a neuron – is selected at random with equal probability from the two parents. A second offspring (or sibling) is simultaneously generated from the complementary genes. The weights for the initial population are generated at random.

Whist our justification for this method is at present heuristic, substantial experience with the procedure leads us to believe that the method produces offspring which

are 'topologically similar' to the parent generation. The quantification for this assertion is under current consideration. The procedure is depicted in Figure 2.

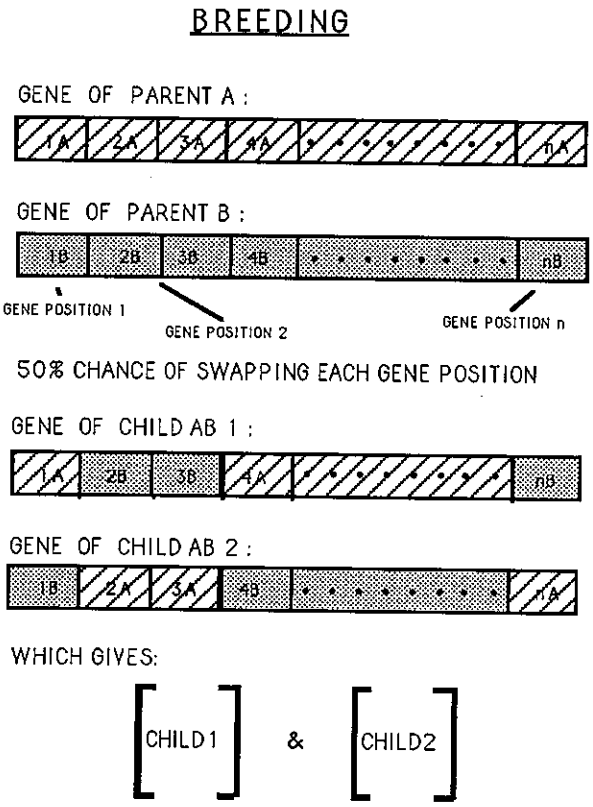

Figure 2: Breeding Rule

2.3 Mutation

The other genetic operator usually incorporated is mutation. Indeed, many algorithm rely solely on mutation for their search of the optimisation space [3] and do not thus incorporate potentially useful structural information from one generation to another. It is our experience that neither a breeding strategy nor a mutation rule alone can ensure rapid convergence of a genetic algorithm and guarantee escape from local minima.

Mutation is treated in the work presented here as a method to reintroduce genetic diversity into a population on the occasions when the preservation of structural similarity imposed by a breeding rule loses that diversity without having reached optimum performance. That is, a local minimum has been reached.

Mutation is accounted for in our model by adding to a selected weight (gene) a zero mean random variable of gaussian distribution. The gene selection criterion is a uniformly distributed random variable which ranges up to the number of genes in the chromosome; the number of such mutated genes increases linearly with the stability of the network from generation to generation. That is, the mutation rate is 'driven' by the failure of the population to show continued improvement.

3 Test Results

Two example problems are presented here and have been selected on the basis that they have been widely documented [5] and are thus useful for comparison with other methods. Although not reported here, other problems have been considered, especially those concerning real-valued functions which change over time.

3.1 XOR Problem

This problem requires the network to learn the exclusive-or problem shown in the following table.

x_1	x_2	z
0	0	0
0	1	1
1	0	1
1	1	0

Table 1: Exclusive-or problem

For this task an initial population of two-input, two-output networks each of two layers was specified with one of the outputs being nominated as a 'don't care' state. An initial population size of 20 was used, with selection of subsequent generations reducing the population to that size at every generation. The mutation variance was 0.2 and each time population stability was reached, a further 10 percent of the total genes in the chromosome were mutated.

The results for 10 runs of the algorithm are shown below and are a considerable improvement over comparable methods. For example Bartlett and Downs [5] report convergence in between 200 and 1900 presentations of the learning set, and Rummelhart *at al* [1] report 250 presentations for the back-propagation algorithm. The average number of generations required for convergence of the algorithm reported here was 121.

Each run yielded a discernably different network, and for one case the elimination of neurons whose input weights were all zero, or whose output was connected via a zero weight resulted in a two-neuron network in which each input was connected to each neuron, and the neurons were connected to each other. We are accustomed to expect that the solution to this problem will require at least three neurons when constrained to a strict layer-to-layer connection.

Run	Generations
1	129
2	508
3	4
4	5
5	329
6	14
7	56
8	27
9	30
10	103

Table 2: Exclusive-or Problem Results

3.2 Parity Problem

The training set for the incompletely specified seven input parity problem is shown below. It was learned by a seven input network with three layers and one output neuron.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	z
0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0
0	0	0	0	0	1	0	0
1	0	1	0	0	0	0	1
0	0	0	0	1	0	1	1
0	1	0	1	0	0	0	1
0	0	0	1	0	1	0	1
1	0	1	0	1	0	0	0
0	0	1	0	1	0	1	0
1	1	1	1	1	1	1	0

Table 3: Seven input Parity Problem

An initial population size of 10 was used in this problem, rather a small number for a problem with so many degrees of freedom. The mutation increment was 0.04 (a little more than one gene), and the mutation variance was 0.01. Of ten attempts the algorithm had not converged on four occasions within 2000 generations and the run was abandoned. Of the remaining six (shown below), the average number of generations for convergence was 475.

Run	Generations
1	> 2000
2	> 2000
3	> 2000
4	> 2000
5	982
6	467
7	7
8	549
9	479
10	370

Again by way of comparison we mention the results of Bartlett and Downs [5]. They report that the Matyas' optimisation technique frequently failed to converge and that their genetic algorithm converged in an average of 930 generations using a starting population size of 50 (considerable greater than used here). Back propagation required an order of magnitude longer.

4 Conclusions

There are indications that the method may be a fruitful approach to the derivation of a neural network with specific capacities, especially those in slowly changing environments. Some of the observed results are also suggestive of the usefulness of the method in exploring experimentally some of the issues in evolution theory, such as the rate of environment change versus group survival threshold and individual capability. The long-term goal of the research is to determine whether the method can derive a network which displays some capacity to learn; that is to select not for performance at a task, but for ability to learn a task. This could illuminate some of the issues in the structure of nets with feedback which theory has not yet made clear.

References

[1] D. E. Rummelhart, G. E. Hinton and R. J. Williams, "Learning Internal Representations by Error Propagation", in *Parallel Distributed Processing, Vol.1*, D. E. Rummelhart and J. L. McClelland Eds., MIT Press, pp. 318-362, 1986.

[2] J. M. McInerney, K. G. Haines, S. Biafore, R. Hect-Nielsen, "Back propagation error surfaces have local minima", Int. Joint Confce. on Neural Networks, Washington, D.C., June 18-22, 1989.

[3] L. J. Fogel, A. J. Owens, M. J. Walsh, *Artificial Intelligence Through Simulated Evolution*, John Wiley and Sons, 1967.

[4] J. C. Schank and W. C. Wimsatt, "Generative Entrenchment and Evolution", PSA 1986 Volume 2 , pp 33-60.

[5] P. Bartlett and T. Downs, "Training a Neural Network with a Genetic Algorithm", First Australian Conference on Neural Networks, Sydney, January, 1990.