ESANN'1996 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-25-26 April 1996, D-Facto public., ISBN 2-9600049-6-5, pp. 291-296

Evolving Neural Network Learning Behaviours with Set-Based Chromosomes

S.M. Lucas
Department of Electronic Systems Engineering,
University of Essex,
Wivenhoe Park, Colchester CO4 3SQ, UK

Abstract

This paper describes a set-based chromosome for describing neural networks. The chromosome specifies sets of neurons with particular functions, and the interconnections between sets. Each set is updated in order, as are the neurons in that set, in accordance with a simple pre-specified algorithm. This allows all details of a neural architecture, including its learning behaviour to be specified in a simple and purely declarative manner. To evolve a learning behaviour for a particular network architecture, certain details of the architecture are pre-specified by defining a chromosome template, with some of the genes fixed, and others allowed to vary. In this paper, a learning perceptron is evolved, by fixing the feedforward and error-computation parts of the chromosome, then evolving the feedback part responsible for computing weight updates. Using this methodology, learning behaviours with similar performance to the delta rule have been evolved.

1 Introduction

Evolutionary programming methods or genetic algorithms have been used extensively to optimise the details of a particular network architecture, for example, the weights, number of units, and number of layers in a multi-layer perceptron. Many researchers in this area have used a direct encoding, where there exists a one-to-one correspondence between the chromosome and the features of the network. For example, it is possible in a direct encoding to identify which part of the chromosome corresponds to a particular weight. This direct encoding method has the virtue of simplicity but scales poorly, and hence is only suited to evolving relatively small networks.

For this reason, there has been some interesting work done on indirect encoding, where the chromosome is used to drive some kind of network construction algorithm. Such methods have the property that small chromosomes can grow massive neural networks.

Some of the earliest work in this area was reported by Kitano in 1990 [1], and has since been followed up with superior network generation languages and grammars designed by Gruau [2], Boers and Kuiper [3], Muhlenbein [4] and Sharman *et al* [5]. All of these however, either use the GA (operating on strings or graphs in the neural description language or chromosome) to evolve a hard-wired neural network, or use the GA to evolve a good topology which is then trained by error backpropagation or simulated annealing.

In contrast to this, the Author [6, 7] has shown how it might be possible to evolve the learning algorithm within the same unified framework in which the

ESANN'1996 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-25-26 April 1996, D-Facto public., ISBN 2-9600049-6-5, pp. 291-296

other network details are evolved. The work reported here goes one step further than the previous work by actually demonstrating the evolution of learning behaviour for a perceptron-type network.

There are two distinct features of the evolutionary framework that make this possible and practical. Evolving a learning behaviour is made possible by using an active weight model, as discussed elsewhere [6, 7]. This sees the weights as active summation cells that are updated just like any other neuron in the network, but can behave just like weights if their outputs are fed into product units, together with the signal to be *weight*ed. Then, we can model any neural network, including its learning behaviour, with the simple algorithm that states: *update each neuron in the network by applying its function to its set of inputs.*

2 Chromosome Structure

The chromosome is split into four distinct parts. For a given problem, some of the genes within a chromosome must be fixed. In effect, this allows the search for a fit individual to be concentrated within a particular 'species' of network. By fixing certain genes we can ensure that all chromosomes generated will give rise to networks that at least have a structure which is *viable* for solving the problem. It is a waste of CPU time to bother looking at networks that have the wrong number of outputs, for example.

The first part consists of a list of size declarations. The value of each size expression is allocated to successive integers.

The second part consists of a list of set declarations. Each set is declared as being of a given type, and of a given size. The size is specified by an integer that refers to a size declaration from the first part of the chromosome. The types allowed at the moment allow summation, product, difference, read, sigmoid, constant and sumstore neurons (that sum all their inputs together with their previous value. Also, there is a mergeset, whose purpose is simply to amalgamate two other sets.

The third part is a list of connections between sets, whose nature is described below.

The fourth part of the chromosome is a list of input/output connections for any phenotypes grown from this chromosome. Without this part, the created networks would be isolated from any interaction with the system in which they are evolved.

2.1 Set Interconnections

At present, set connections may be one of two types, called *Div* and *Mod*, after the operators used to calculate which node in set A to connect to which node in set B. Figure 1 illustrates this. Set **a** is a sumset of size 3, set **b** is a sumset of size 2, set **c** is a prodset of size 6 and set **c** is a sumset of size 2. The interconnections are as follows: (**c a** Mod)(**c b** Div)(**d c** Div).

By appropriate use of *Div* and *Mod* connection specifiers in conjunction with sets of arithmetic operators, any of the standard vector/matrix operations are

ESANN'1996 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-25-26 April 1996, D-Facto public., ISBN 2-9600049-6-5, pp. 291-296

possible, plus many other non-standard ones. For example, the set **c** represents $\mathbf{c} = \mathbf{ab}^{-1}$ in standard vector notation.

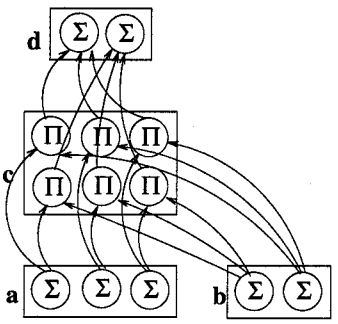

Figure 1: Illustration of how the set connection operators work.

2.2 Example Chromosome

We now look at an example chromosome template for evolving a single layer perceptron with learning behaviour (see Table 1). Most of this the chromosome for this network has to be fixed to ensure that we search a viable space of networks.

We have a set of I inputs which are augmented with a constant unit that always outputs 1.0, to allow each output to have a different bias depending on the weight connecting it with the constant unit. Therefore, if we have O outputs, we need a set of W weights where $W = O \times (1 + I)$. This is shown in the first part of the chromosome.

Hence, any viable solution network must include sets of these sizes. For the next part we must declare some necessary sets, and allow other sets (in fact just one in this case) to be chosen randomly. The following set declarations are commented to aid understanding

The only set definition that is subject to evolutionary change is that of set 8. This was chosen since a solution was known to exist using this number of sets. For future work, more open-ended experiments are intended. The third part of the chromosome shows the groups of connections between sets. At the moment, each set has an arity of zero, one or two. READSETs have an arity of zero, indicating that they must not take inputs from any other sets in the network – this would destroy their purpose, which is to take inputs from sources external to the network. ACCSETs (accumulator set) and SUMSTOREs each have an arity of one. Accsets are used to define a set of neurons whose purpose is each to accumulate a sum of their inputs. Sumstores are identical except they also include in the summation their previous output. The set of connection is also mostly fixed, with the only variability allowed in connections 8,9 and 10. Finally, the connection between the neural network and the system that calls it is set up. All supervised networks can be seen in black box terms as

ESANN'1996 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-25-26 April 1996, D-Facto public., ISBN 2-9600049-6-5, pp. 291-296

SizeDef	Parameter Name
0	Inputs
1	1
2	Outputs
3	1+Inputs
4	(1+Inputs) * Outputs

Set	Function	SizeVar	Comment
0	readset	0	The set of inputs
1	constant	1	A set with a single constant unit
2	mergeset	3	The augmented input set
3	readset	2	Target Vector
4	prodset	4	Connections
5	accset	2	Synaptic Activity at Output
6	sigset	2	Sigmoid of Synaptic Activity
7	diffset	2	Difference between Output and Target
8	ANYTYPE	$\{0,1,2,3,4\}$	Randomly Chosen Set Type and Size
9	sumstore	4	The Weight Stores

Connection	To	From	Type
0	2	0	Mod
1	2	1	Mod
2	4	2	Mod
3	4	9	Mod
4	5	4	Div
5	6	5	Mod
6	7	3	Mod
7	7	6	Mod
8	8	ANYSET	{Mod, Div}
9	8	ANYSET	{Mod, Div}
10	9	ANYSET	{Mod, Div}

External Name	Set number
Inputs	0
Targets	3
Outputs	6

Table 1: The four segments of the chromosome. The first part declares the possible sizes of each set, the second part declares the composition of each set, the third part declares the groups of connections between neurons of each set, and the fourth and final part declares the external connections.

ESANN'1996 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-25-26 April 1996, D-Facto public., ISBN 2-9600049-6-5, pp. 291-296

taking information from the outside world via their inputs and their targets and returning information via their outputs.

3 Results

Initial results demonstrate that learning behaviour can be evolved within this framework. The chromosome template discussed above was used to generate a random chromosomes. These were then adapted using a random hill climbing procedure, or just used in a pure random search. Future work will also involve experimentation with conventional population GAs, though it should be noted that these do not necessarily outperform random hill-climbing methods.

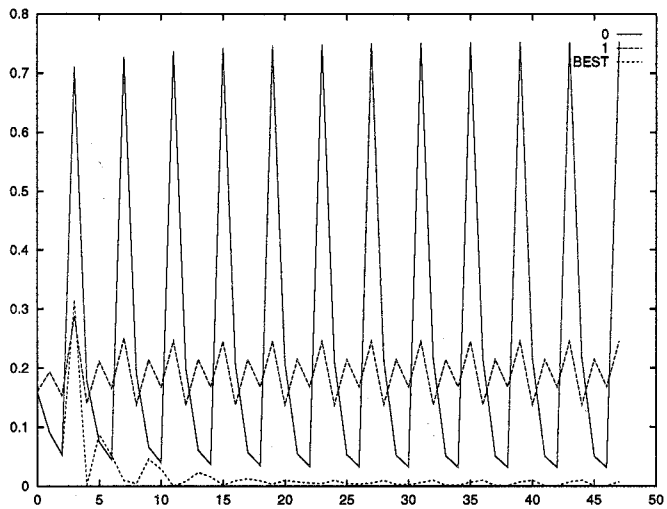

Figure 2: Plots of error with respect to cycle number for some random chosen networks, together with the best one that was evolved.

The fitness function was simply the mean square error averaged over 1 complete pass through all the data (i.e. one epoch), measured just once after 12 epochs. Hence, the fittest possible individual would have a score of zero. The fitness of the delta rule was found to be 0.009. The best individual found by the random hill climber had a fitness of 0.004. Figure 2 shows the behaviour of some individuals chosen from one run of the algorithm, including the best one that was found in this particular case after 9,718 steps. A network was considered to have learnt the problem if its average squared error fell below 0.01 for the final epoch.

So far, the results are interesting, in that the pure random search has outperformed random hill climbing. Given 10 runs of the experiment, allowing a maximum of 10,000 fitness evaluations, the random hill climber only found a solution 2 times out of 10 (but did so with a mean of 289 steps). Pure random search found a solution every time within 10,000 evaluations, with a mean

of 2152 fitness evaluations. This is probably due to the highly discontinuous nature of the search space that follows from using set-based chromosomes, in that changing the function of a set or the source set of one of its inputs typically has a dramatic effect on network behaviour.

4 Conclusions

Given the structure of a single layer perceptron as a starting point, a learning behaviour has been evolved. Future work will concentrate on evolving learning behaviours for more complicated, more useful networks, and also in evolving new classes of neural networks, together with their learning behaviours. The use of set based chromosomes allows a parameterised description of a neural network, with the effect that once an architecture has been evolved for a particular problem, it could then be applied to related problems of a different size.

Acknowledgement: This work was supported by UK EPSRC grant GR/J86209.

References

[1] H. Kitano, "Designing neural networks using genetic algorithm with graph generation system," *Complex Systems*, vol. 4, pp. 461 – 476, (1990).

[2] F. Gruau, "Cellular encoding of genetic neural networks," *Laboratoire de l'Informatique du Parallelisme Technical Report 92-21, Ecole Normale Superieure de Lyon*, (1992).

[3] E. Boers and H. Kuiper, "Biological metaphors and the design of modular artificial neural networks," *Masters thesis, Department of Computer Science and Experimental and Theoretical Psychology, Leiden University, the Netherlands*, (1993).

[4] H. Muhlenbein and B. Zhang, "Synthesis of sigma-pi neural networks by the breeder genetic programming," in *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 318 – 323, Orlando: IEEE, (1994).

[5] K. Sharman, A. Esparcia-Alcazar, and Y. Li, "Evolving signal processing algorithms by genetic programming," in *Proceedings of IEE 1st International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pp. 473 – 480, London: IEE, (1995).

[6] S. Lucas, "Growing adaptive neural networks with graph grammars," in *Proceedings of European Symposium on Artificial Neural Networks (ESANN '95)*, pp. 235 – 240, Brussels: D facto, (1995).

[7] S. Lucas, "Towards the open-ended evolution of neural networks," in *Proceedings of IEE 1st International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pp. 388 – 393, London: IEE, (1995).