ESANN'1995 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 19-20-21 April 1995, D-Facto public., ISBN 2-9600049-3-0, pp. 235-240

Growing Adaptive Neural Networks with Graph Grammars

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

Abstract

This paper describes how graph grammars may be used to grow neural networks. The grammar facilitates a very compact and declarative description of every aspect of a neural architecture; this is important from a software/neural engineering point of view, since the descriptions are much easier to write and maintain than programs written in a high-level language, such as C++, and do not require programming ability.

The output of the growth process is a neural network that can be transformed into a Postscript representation for display purposes, or simulated using a separate neural network simulation program, or mapped directly into hardware in some cases.

In this approach, there is no separate learning algorithm; learning proceeds (if at all) as an intrinsic part of the network behaviour. This has interesting application in the evolution of neural nets, since now it is possible to evolve all aspects of a network (including the learning 'algorithm') within a single unified paradigm. As an example, a grammar is given for growing a multi-layer perceptron with active weights that has the error back-propagation learning algorithm embedded in its structure.

1 Introduction

Neural Description Languages have been investigated in some depth by the software and systems engineering communities, and more recently, by genetic algorithm researchers. All the environments/languages developed so far [1] aim to make the job of designing neural networks easier, more reliable and more efficient, by giving the programmer high-level data and control structures for describing the network and the associated learning algorithm. None of them force the neural systems designer to implement the learning algorithm as an intrinsic part of the network, and indeed, most of them encourage the separation between network and learning algorithm.

Some very interesting developments have come from the genetic algorithms community regarding the specification of neural networks. This area was probably started by Kitano in 1990 [2], and has since been followed up with superior network generation languages and grammars designed by Gruau [3], Boers and Kuiper [4] and Muhlenbein [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. In the approach presented here, there is no separate learning algorithm; learning proceeds as an intrinsic part of the network behaviour. Hence, it is now possible (in principle) to evolve all

ESANN'1995 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 19-20-21 April 1995, D-Facto public., ISBN 2-9600049-3-0, pp. 235-240

aspects of a network, including the learning 'algorithm', within a single unified paradigm.

2 The Rewriting Operations

A set of basic graph rewriting operations is listed in Table 1 and depicted in Figure 1. Table 3 shows a grammar for growing the complete structure of an adaptive MLP. Figure 2 shows the network that grows when we rewrite `S(2 2 1)` according to this grammar, and wire it up according to the connection rules shown in Table 4.

Action	Symbol
:modwrite	Γ
:change	Δ
:connect	Θ
:multiwrite	Φ
:parwrite	Ξ
:toparwrite	Π
:seqwrite	Ψ

Table 1: Graph rewriting operators

Symbol	Description
ι	Reads a value from input stream
π	Computes $o_j = \prod_{i \in I_j} o_i$
Σ	Computes $o_j = \sum_{i \in I_j} o_i$
σ	Computes $o_j = \sigma(\sum_{i \in I_j} o_i)$
γ	Outputs constant 1.0
δ	Computes $o_j = o_{I_1} - \sum_{i \in I_j \forall i \neq I_1} o_i$

Table 2: Neural cell definitions

3 Simulating Neural Networks

Many, if not most neural network simulations divide the simulation in two phases: the training phase and the operational phase.

For the operational phase, the simulation can generally be simplified to the following form: we have a set N of neurons, where each neuron n_i has a set of input connections denoted I_i, and computes a function $f_i : \mathcal{R}^{|\mathcal{I}_i|} \rightarrow \mathcal{R}$ that maps from the set of input values to the output activation o_i. The following algorithm is suitable for simulating all the popular neural network models irrespective of whether they are feedforward or recurrent.

$$\boxed{\texttt{repeat forever } \forall i \in N \quad o_i := f_i(I_i) \ ;}$$

Given this fundamental simplicity, it is quite remarkable how complicated many implementations of neural networks become in practice. Interestingly, as demonstrated in below, this algorithm is also sufficient for simulating the learning phase of a neural network. The key concept that makes this possible is the *active weight*. Normally, neurons are seen to be connected by *passive* weights that are incapable of computation or self adjustment – they simply act as multipliers on synaptic inputs to neurons.

Via a simple transformation, we can instead view all the connections as having a weight of unity. Then, differing connection strengths are modelled not by passive numbers, but by feeding the input of a weight-store neuron into a

ESANN'1995 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 19-20-21 April 1995, D-Facto public., ISBN 2-9600049-3-0, pp. 235-240

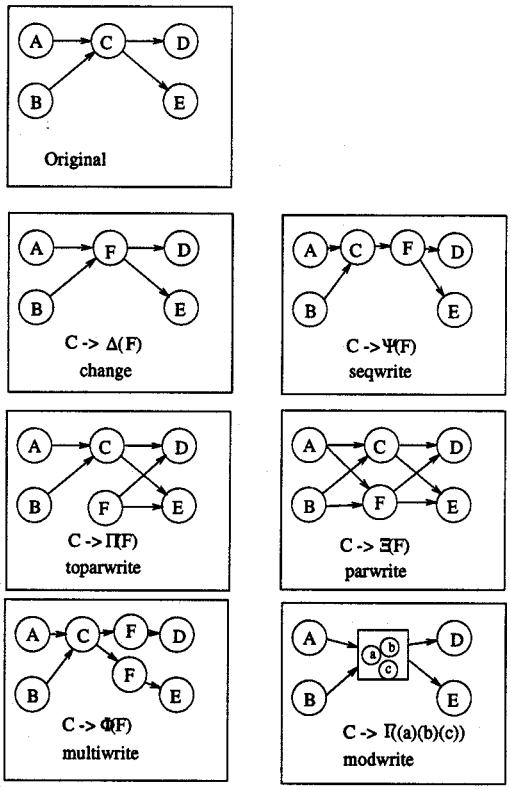

Figure 1: An illustration of the action of the fundamental graph rewriting operators listed in Table 1.

product neuron – together with the signal to be multiplied. When we do this, the end result is a network with many more neurons than previously, but with the strong advantage that now the learning behaviour can become *intrinsic* to the normal network operation.

3.1 Ordering the Neuron Updates

If the network grown is purely feedforward, then the digraph of the network forms a partially ordered set (POSET), which can then be sorted into some arbitrary but correct total order before being submitted to the simulator. However, when the connections are recurrent, a POSET is not formed, and hence this creates a problem. The order in which the neurons are evaluated is important, and in the case of conflict (as will inevitably arise in the case of networks with recurrent connections), we have the choice of making the decision in the rewriting process (i.e. embedding some decision logic in the grammar), or making the choice at network build time (e.g. making decisions at random, or according to some heuristic). The solution adopted here is that, as a network grows, the 'scaffolding' connections between cells do not imply any order. When the network has finished

S(i h o)	$\xrightarrow{T}$	Δ(S I i)Υ(HS H o)Π(CM 3)	(1)
CM(n)	$\xrightarrow{(n>0)}$	Δ(CM $(n-1)$)	(2)
CM(n)	$\xrightarrow{T}$	Φ(BX)Γ((mtf γ)) Δ(M)	(3)
S(F n)	$\xrightarrow{(n>1)}$	Ξ(S $F(n-1)$)Δ(F)	(4)
S(F n)	$\xrightarrow{T}$	Δ(F)	(5)
HS(h o)	$\xrightarrow{T}$	Υ(S O o)Π(CM 3)Δ(S H h)	(6)
I()	$\xrightarrow{T}$	Φ(BX)Δ(IM)	(7)
IM()	$\xrightarrow{T}$	Γ((mtf ι) (mub Σ)) Δ(M)	(8)
H()	$\xrightarrow{T}$	Φ(BX)Δ(HM)	(9)
HM()	$\xrightarrow{T}$	Γ((mtf σ) (mub Σ) (const γ) (diff δ) (mlb π)) Θ(mlb mtf) Θ(mlb diff) Θ(mlb mub) Θ(diff mtf) Δ(M)	(10)
BX()	$\xrightarrow{T}$	Γ((w Σ) (ctb π) (ctfb π) (ctf π)) Θ(w ctfb) Θ(w w) Θ(ctb w) Θ(ctf w) Δ(C)	(11)
O()	$\xrightarrow{T}$	Γ((mtf σ) (drv δ) (err δ) (tgt ι) (const γ) (mlb π)) Θ(err mtf) Θ(err tgt) Θ(mlb err) Θ(mlb drv) Θ(mlb mtf) Θ(drv mtf) Θ(drv const) Δ(M)	(12)

Table 3: A grammar for growing the structure of an adaptive MLP

growing a set of connection rules are applied to wire up some actual connections along the scaffolding. Some of these connections are designated as feedforward, and others as feedback. When establishing the POSET, the algorithm accounts

(M mtf)	$\rightarrow$	(C ctf)
(C ctfb)	$\rightarrow$	(M mtf)
(M mtf)	$\rightarrow$	(C ctbf)
(C ctb)	$\leftarrow$	(M mlb)
(C ctfb)	$\leftarrow$	(M mlb)
(M mub)	$\leftarrow$	(C ctb)

Table 4: Connection rules for wiring up the network. A forward ($\rightarrow$) or backward ($\leftarrow$) connection is made between any units (lowercase) within any modules (uppercase) with the appropriate labels.

ESANN'1995 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 19-20-21 April 1995, D-Facto public., ISBN 2-9600049-3-0, pp. 235-240

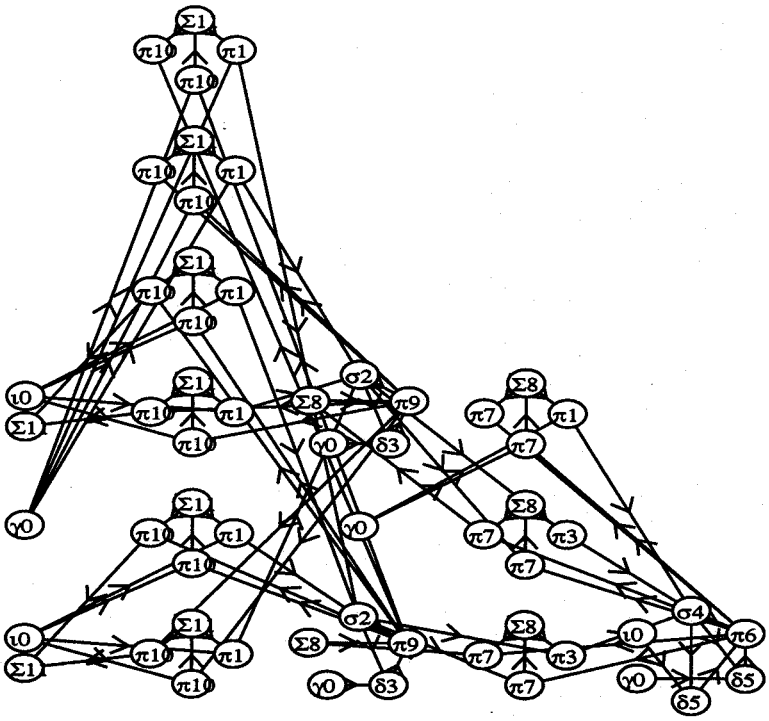

Figure 2: The adaptive MLP, grown from rewriting the symbol **S(2 2 1)** according to the grammar in Table 3. When simulated using the simple yet general purpose neural network simulator described in the text, this network performs the error back-propagation algorithm. Each cell is labelled by its type (see Table 2) and its partial order (all cells ordered 0 are evaluated, then those labelled 1, and so on. The inputs are at the left of the diagram, and the outputs at the right.

for the nature of each connection, and providing no cycles exist either in the set of feedforward connections, or the set of feedback connections, then no conflict arises. Note that the distinction between feedforward and feedback connections is made *only during the partial ordering process.* All connections are treated identically at the simulation phase.

The network in Figure 2 was simulated using the simple algorithm given above, and tested on the XOR problem. Results were compared with the operation of a standard MLP simulator, and were broadly similar, though not identical, perhaps due to the updates to the weights and the deltas being done in a different order.

4 Conclusions

The use of a graph grammar for growing adaptive networks has been demonstrated. In this paradigm, there is no separate learning algorithm – the network simply behaves as prescribed by the simple simulator – which can simulate any

artificial neural network.

Growth of the network proceeds according to cell divisions and modifications. As the network grows in this way, so a trail of connection scaffolding is left behind, connecting the groups or modules of neurons in each region. Once the growth process is essentially finished, actual connections are established between linked modules, according to a very simple connection grammar.

The main result of the paper is to demonstrate the growth of modular networks that are self-adaptive. Now, only one program is needed to simulate *any* static neural architecture, allowing neural network researchers to concentrate on the important details of a neural architecture, and not waste time implementing simulation programs or libraries for each new combination of architecture and learning algorithm.

Finally, the author has argued elsewhere [6, 7] that the relationship between neural networks and formal grammars is an interesting, informative and useful one; this paper offers further evidence in support of that case.

Acknowledgements

This work was supported by SERC grant GR/J86209.

References

[1] M. Recce, R. P.V., and P. Treleaven, "Neural network programming environments," in *Artificial Neural Networks, 2: Proceedings of ICANN-92*, pp. 1237 – 1244, Amsterdam: Elsevier, (1992).

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

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

[4] 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).

[5] 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).

[6] S. Lucas, "Connectionist architectures for syntactic pattern recognition," *PhD Thesis, University of Southampton*, (1991).

[7] S. Lucas and R. Damper, "Syntactic neural networks," *Connection Science*, vol. 2, pp. 199 – 225, (1990).