Department of Information Technology

technical report NUIG-IT-111002

Applying Evolutionary Computation to Designing Neural Networks: A Study of the State of the Art

D. Curran (NUI, Galway)
C. O'Riordan (NUI, Galway)

Applying Evolutionary Computation to Designing Neural Networks: A Study of the State of the Art

Dara Curran
Department of IT
NUI, Galway

Colm O'Riordan
Department of IT
NUI, Galway
colmor@geminga.nuigalway.ie

October 14, 2002

Abstract

In recent times, much research has been undertaken in the combination of two important and distinct areas: genetic algorithms and neural networks. Genetic algorithms attempt to apply evolutionary concepts to the field of problem solving while neural networks represent a simplified model of the brain capable of solving classification problems.

Within the field of neural networks, problems arise in attempting to define the optimal network (weights, architecture and learning functions etc.). In this paper, we review approaches that have been adopted to evolve, using genetic algorithms, suitable neural networks.

1 Introduction

In recent times, much research has been undertaken in the combination of two important and distinct areas: genetic algorithms and neural networks. Genetic algorithms attempt to apply evolutionary concepts to the field of problem solving, notably function optimisation, and have proven to be valuable in searching large, complex problem spaces.

Neural networks are highly simplified models of the working of the brain. These consist of a combination of neurons and synaptic connections, which are capable of passing data through multiple layers. The end result is a system which is capable of pattern recognition and classification.

In the past, algorithms such as back propagation have been developed which refine one of the principal components of neural networks: the connection weights. The system has worked well, but is prone to becoming trapped in local maxima and is incapable of optimisation where problems lie in a multi-modal or non-differentiable problem space.

Genetic algorithms and neural networks can be combined such that a population of neural networks compete with each other in a Darwinian 'survival of the fittest' setting. Networks which are deemed to be fit are combined and passed onto the next generation producing an increasingly fit population, so that after a number of iterations an optimised neural network can be obtained without resorting to a design by hand method.

The remainder of the paper will outline some of the characteristics of neural networks which have been evolved and some of the many possible solutions for representing a neural network in a format suitable to the genetic algorithm, i.e. transforming a network to a gene code or chromosome. In the concluding re-

marks some suggestions for further research are discussed.

2 Layers of Evolution

The evolution of neural networks can be classed according to the goals behind such evolution. Some schemes have proposed the evolution of the weights, starting with a fixed architecture. Others have suggested that the architecture is more important. Other approaches have included transfer functions and learning rules. Perhaps the most potentially interesting area for new research directions lies in the combination of such evolutionary techniques. In the following sections, some of the most popular evolutionary frameworks are discussed, including some which combine more than one aspect.

2.1 Evolution of weights

The evolution of weights assumes that the architecture of the network remains static. This implies some degree of pre-processing by the human designers and is generally employed when some ideas exist pertaining to the structure of the neural network.

The primary motivation for using evolutionary techniques to establish the weighting values rather than traditional gradient descent techniques such as back propagation, lies in the inherent problems associated with gradient descent approaches. Back propagation in particular can become easily trapped in local maxima. Furthermore, it is impossible for a back propagation algorithm to find the optimal solution if the function being optimised is multimodal or non-differentiable[1]. In addition, it has been shown that back propagation is sensitive to the initial condition of the neural network causing additional problems[2].

Evolutionary approaches such as genetic algorithms however, are able to optimise functions in such environments and, furthermore, do not even require the function to be continuous. This is because genetic algorithms employ fitness functions which can be tailored to suit the problem at hand and are not restricted in any way.

Much research has been completed in the evolution of connection weights [3, 4, 5, 6, 7, 8, 9]. Another approach uses the global searching capability of genetic algorithms to search the broad area of the weight problem space, and then uses back propagation as a local search to refine the weights[5]. This approach ensures that the networks evolved are very accurate, more accurate than they would have been had genetic algorithms been used in isolation.

2.2 Evolution of architectures

Work which addresses the evolution of architectures views the structure of the neural network as the defining characteristic. The view is that once a suitable structure is found, an algorithm such as back propagation can be used to find the correct weights. Techniques prior to the evolutionary approach consisted of two basic operations: constructive and destructive.

Broadly speaking, the constructive method begins with a minimal network and successively adds nodes and connections until the network is capable of solving the desired problem with sufficient accuracy. The destructive method takes the mirror approach. It begins with an already functioning network and successively removes connections and neurons until the network is no longer able to solve the problem, at which point the last move is undone. There are clearly problems with this kind of approach, which is in fact a form of hill-climbing and is likely to become trapped in local maxima[10].

The fact that the search space for neural network architectures is infinitely large and non-differentiable[11] makes the genetic algorithm approach a good candidate for success. Indeed, research into the evolution of neural network architectures has been largely successful[12, 13, 14, 15, 16, 17, 10].

2.3 Transfer Functions

The transfer function for all neurons of a neural networks is generally taken to be fixed, although some attempts have been made to allow its adaptation over generations [18, 19]. These schemes typically begin with a fixed proportion of transfer functions, such as sigmoidal or Gaussian, and allow the genetic algorithm to adapt to a useful combination according to the situation. Often, transfer function evolution is combined with another type, such as architecture or weight evolution which produces more interesting results.

2.4 Learning Rules

Neural networks have a number of learning rules which govern the speed and accuracy with which the network will train. Examples of these are the learning rate and momentum. These parameters can be difficult to assign by hand, and therefore make good candidates for evolutionary adaptation. Typically the parameters are encoded into the gene code of each network and allowed to evolve [20, 15].

2.5 Simultaneous Evolution

One of the most interesting areas of evolutionary neural networks is the combination of several schemes which simultaneously evolve different aspects of the networks. One of the most important is the combination of architecture and weight evolution [11, 7, 12, 21, 22, 18, 23, 24, 25]. The advantage of combining these two basic elements of a neural network is that a completely functioning network can be evolved without any human interaction.

Clearly, it might be advantageous to simultaneously evolve more neural network features, thus leading to more efficient and accurate results. However, it is not clear whether the increase in complexity of the resulting encoding scheme would be offset by a marked improvement in performance. Typical convergence times for combined approaches are large in comparison to approaches dealing with a sginle feature, and the addition of more complexity might make the problem intractable.

3 Encoding Strategies

Crucial to the successful evolution of a neural network is the way its structure and/or weights are encoded into the chromosome used by the genetic algorithm. A great deal of research has been carried out in this area and as a consequence many systems exist. These can however be divided into two main camps: direct and indirect encoding.

3.1 Direct Encoding

Direct encoding is a strategy where some or all of a neural network's defining parameters, such as weight values, number of nodes, connectivity etc., is encoded into the gene code. Thus it is possible to recreate the exact neural network from the underlying genotype. [26].

3.1.1 Connectionist Encoding

Connectionist encoding is concerned with mapping a neural network's connectivity to the gene code. An early implementation of such an encoding scheme is that of Miller's Innervator [13]. Innervator has a fixed number of nodes and connectivity is denoted by a single bit. A matrix is then derived containing a full connectivity map of the neural network. The networks are selected according to their performance after several epochs of training.

Another approach to connectivity encoding is to encode the weights for each connection in the neural network. In the canonical genetic algorithm[27, 28],

chromosomes are defined as a binary string - so early approaches used a binary string to encode the real number weights [4, 5].

This is the approach of GENITOR[4]. Wherein, each connection and its corresponding weight are encoded into the gene code. Each weight in GENITOR is encoded as an 8-bit binary string, indexed by a single bit which is used to denote the presence or absence of the connection. The weight values are first evolved to an optimum, and then a pruning algorithm streamlines the neural network's architecture, by changing the index bit as required.

It has been widely asserted that weights associated with a specific node should be placed in the same region of the chromosome[5, 11] as this reduces the probability that they will become separated during crossover, thereby potentially losing an evolved characteristic of the network.

There are obvious advantages to preserving the traditional binary approach. It is both very simple and extremely general. In addition, no new genetic operators need to be devised. However, converting a real number to a binary representation invariably means a loss of accuracy, unless very large strings are created. Large chromosomes are very often detrimental to the genetic algorithm's performance in term of processing time[27].

Montana [6] devised an encoding scheme which represents weights as real numbers. He also created a number of tailored genetic operators which are able to deal with the change. Another approach has been the use of integer fractions to denote the real number value [29, 20], as opposed to approximating the value by direct encoding.

An encoding scheme using a variable length binary string has also been developed [23]. A granularity parameter is added to the chromosome which governs how many bits are allocated to each weight. Typically, the algorithm is allowed to evolve globally using a small number of bits, and is then fine-tuned locally by increasing the number of bits per weight.

3.1.2 Node Based Encoding

Node based encoding strategies concentrate on the number of neurons which should be used. Whilst weight encoding schemes assume that an architecture has already been designed, the construction of an efficient network structure is just as difficult and is therefore well suited to a genetic algorithm approach.

In Schiffman's [30] approach, a blueprint is used to describe the neural network's structure. Starting with the input node, each node is numbered in sequence and placed in a list. Then, each node in the list is taken in turn and the numbers of the connecting nodes from the previous network layers are placed before its entry in the list, forming a complete node mapping of the network. Crossover is implemented between nodes only, and several mutation operators are used to add or delete weak connections.

A similar system, GANNet, has been devised[18], but with a few restrictions notably on the number of input nodes and the fact that only connections between adjacent layers are allowed. The mutation and crossover operators have also been altered slightly.

3.1.3 Graph Based Encoding

Graph based encoding views the network as a grid of functions and terminals. This is an alternative to the usual parse tree used in other genetic programming approaches. The scheme also provides a linear chromosome which can be converted back and forth into the grid for efficiency purposes [31].

The chromosome consists of an ordered list of the nodes in the network, with an index indicating their position within the gene code. Each node contains information regarding its bias, and all its connections to other nodes, including weights. The nodes can be seen as either functions or terminals, where functions are the neurons of the network and the terminals are the input variables.

The model proposed by Pujol *et al* [31] allows for multiple activations functions and any type of network, though the number of nodes is made to be the same for each member of the population. Specialised crossover and mutation operators are specified according to whether the node in question is a function or terminal, which is made easier thanks to a dual representation mechanism, allowing the linear chromosome to be readily mapped to a grid representing connectivity.

3.1.4 S-Expressions

The use of LISP Symbolic Expressions has been adopted in the creation of an alternate encoding strategy. Each network is represented by a number of functions, representing nodes, and terminals. Rather than encoding the neural network structure as a list, Koza and Rice [12] represent this as a parameter tree.

A number of operators can be used to define the network: arithmetic and weighting functions can be combined to create the weights of the network and the bias of a node can be altered via special processing nodes in the grammar tree.

The S-expressions approach to encoding neural networks results in quite streamlined gene codes which do not suffer from the same scalability problems as direct encoding. Crossover takes place at a sub-tree level, ensuring that learned portions of the network are not entirely disrupted.

3.1.5 Layer Based Encoding

Layer based encoding uses a chromosome which is subdivided into areas corresponding to the neural networks's layers. In the GENESYS system [15], each area has an identifying index, the number of nodes within it and a number of projector fields which specify its connectivity to the next layer. Mandischer[32] modified the system and specifies a radius and density of connections for each layer. The radius describes the spread of connections to nodes in the given area, while the density indicates how many nodes in the layer are connected. Crossover is applied between layers and mutation alters the learning rate, momentum, radius, density and the number of nodes in a layer.

3.1.6 Marker Based

Marker based encoding is inspired by the structure of DNA in living organisms [33]. In DNA, structures known as nucleotide triplets specify amino acids which make up a protein. Some triplets are given special 'marker' status, which allows them to denote the start and end of the protein definition.

Marker based chromosomes are said to be circular, in that one end of the chromosome can be wrapped around to join the other. The start marker does not necessarily have to be placed at the start of the gene code because the algorithm reads the chromosome until such a marker is found. The scheme allows for both recurrent and feed-forward networks and can be said to be complete, in that any gene code can be correctly converted into a functioning network. Marker based encoding certainly allows more freedom in the definition of a neural network and the mutation and crossover operators can operate without restrictions.

3.2 Indirect Encoding

The indirect encoding strategies attempt to describe a neural network in terms of assembly instructions or recipes[30]. Whilst in the direct encoding approach, parameters of the network were explicitly present in the genetic code, with indirect encoding only a method of assembling the network is present.

The main motivations for such a shift are size and modularity. While direct encoding schemes are, in general, quite straight-forward to implement, they suffer from a lack of scalability - the more complex the network the more computationally intractable they

become due to the size of the gene code in the genetic algorithm.

3.2.1 Matrix Re-writing

One of the first indirect encoding schemes proposed is that of Kitano's matrix rewriting[14]. The scheme is based around the connectivity matrix seen in direct encoding schemes[13]. It begins with a base 2x2 matrix and repeatedly applies generation rules to each non-terminal element in the matrix until all elements are terminals. For instance, if the starting matrix is:

$$\begin{matrix} A & B \\ B & A \end{matrix}$$

and if the A and B are replaced by $\begin{matrix} aa \\ aa \end{matrix}$ and $\begin{matrix} aa \\ ab \end{matrix}$,

$$\begin{matrix} a & a & a & a \\ a & a & a & b \\ a & a & a & a \\ a & b & a & a \end{matrix}$$

and if a and b are $\begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix}$ and $\begin{matrix} 1 & 1 \\ 0 & 0 \end{matrix}$,

The final matrix showing the connectivity of the neural network would be:

$$\begin{matrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{matrix}$$

Generally a fixed number of rewriting steps is used to create the final matrix. These rewriting rules can be encoded into a gene code. In this case each rule can correspond to 4 alleles on the chromosome corresponding to the first start rule matrix, containing A and B. Typically the rules defining the final rewriting step (i.e. to the binary stage) are predefined and do not play a part in the evolution of the rules. For instance the set of rules above could be defined by the gene code:

ABBAaaaaaaabaaaaaaab

Some good results have been reported for this scheme [14]. However, recent work [34] has shown that direct encoding can be at least as good as the matrix rewriting proposed here.

3.2.2 Cellular Encoding

Cellular encoding, created by Gruau[29, 21], represents neural networks as grammar trees, i.e. the grammar describing the network is encoded as a tree. The building block of cellular encoding is the cell, which represents a node in an ordered graph. Each cell has a reading head which reads the cellular code and acts upon the instructions therein. The cell manages internal variables which can govern its development or regulate neural network related parameters such as weights or thresholds. The cell can be viewed as a Turing machine, only reading sections of the grammar tree instead of tape.

The development begins with a single cell known as the ancestor cell, which is connected to an input and an output cell. The cell's reading head is placed at the start of the cellular code (itself in the beginning) and executes the operator located there. Various operators exist:

- **Sequential Division**
  In sequential division, the cell splits into two cells, the first of which inherits all the input links and the second the output links.

- **Parallel Division**
  In parallel division, the cell splits as above, but

both children inherit the input and output links of the parent cell.

- **End Program**
  This command causes the cell to lose its reading head and therefore cease to be active.

In addition to these, several value-modifying operators exist for the parameters stored in the cells:

- **Increase and Decrease Bias**
  This command increases or decreases the threshold within the cell that reads it.
- **Increase or Decrease Link Register**
  The link register contained in each cell points to a specific connections to or from the cell. Changing its value causes it to point to a different link. Once the register has been set to a specific connection, other operations can be performed on it, such as VAL+ or VAL- which alter the weight value of the connection pointed to by the link register.

The sequence of execution of commands is achieved through means of a FIFO queue. Once a cell executes a command, it enters a FIFO queue. If the cell divides, the child reading on the left of the tree enters the queue first. The idea is to emulate parallel execution of commands by the cells. The system ensures that no cell will execute two consecutive commands if there exists a cell which has not executed any command. To this end, the scheme also includes a WAIT command, should one portion of the tree be executing too quickly.

Cellular encoding also allows for recursion to occur, where a sub-tree of commands can be executed in a loop with a specified number of iterations. It also implements a pruning mechanism through the use of the CLIP and CUT commands which removes a connection pointed to by the link register.

Cellular encoding can be used to evolve both weights and architecture of a neural network [35]. It compares quite favorably towards direct encoding, in that while the cellular encoding takes longer to compute, the relative amount of effort required to achieve efficient neural networks make it attractive.

3.2.3 Edge Encoding

A scheme similar to cellular encoding, edge encoding[16] grows network graphs using edges instead of nodes. While the cellular encoding approach evaluates each grammar tree node in a breadth first search manner, thus ensuring parallel execution, edge encoding is designed to work using depth first search.

The cellular encoding approach tends to create graphs with a large number of connections which must then be pruned using the CUT or CLIP operators. Edge encoding, using its depth first search approach, favours graphs with fewer connections. However, the relative lack of connections does not necessarily imply a smaller genetic code - on the contrary, edge encodings often have larger gene codes than those produced by cellular encoding. This is because there is implicitly more information required to store edges in a network graph than there is about the nodes.

Luke and Spector[16] argue that this is not too significant and that the real benefit of edge encoding is that modularity can be created through the development of building blocks resulting from their depth first search approach.

3.2.4 L-Systems

The set of Lindermayer-systems are an encoding scheme based on the work of Lindermayer. They are based on a biological model where cells exchange information with their neighbours. They use a specialised grammar in which production rules are applied in parallel rather than sequentially, as seen in previous examples. Boers and Kuiper [36] have used

this model to generate neural networks. The rewriting rules take in account the relative position of each cell to determine whether the production rule should apply. All possible production rules which are applicable are applied immediately, rather than waiting for other portions of the graph to catch up.
The encoding method generates strings which are not always guaranteed to produce correct production rules. Therefore, error recovery operations were implemented to address this. The system was successful with the XOR problem and simple letter recognition tasks. A modified version, the Building Blocks in Cascades (BBC) algorithm has been developed based on this work[37].

3.2.5 Growth Encoding

Cangelosi *et al* [38] have argued for a more biological approach to the evolution of neural networks. They criticise the direct encoding mechanism for its lack of scalability, but also for its lack of biological plausibility as it is unlikely that the entire nervous system of an organism is mapped out in detail in its genetic code.

Their work is based on an earlier project which was concerned with simulating the growth of synaptic connections between neurons in a neural network[39]. In the earlier work, the gene code contained information on the manner in which connections were allowed to grow from neurons. The network was mapped in 2-dimensional space, and those connections which had reached other nodes in the given time frame were considered valid - others were discarded.

Cangelosi takes the work further by taking this previous growing principle but also adding the possibility of nodes dividing and migrating in their 2-dimensional environment. The set of rules employed is similar to the rule rewriting schemes seen earlier but the application of the rules is radically different from previous approaches. The genetic code specifies the type of cell which is being represented, the number of divisions allowed per cell, the connection growing rate and then angle of growth.

The problem domain studied reflects the biological premises in the work: the neural networks learn to move around their environment as well as obtaining food and water located in certain areas. The scheme produced interesting results.

3.3 Encoding Difficulties: The Permutation Problem

The biggest potential problem associated with the encoding of a neural network onto a gene code is known as the permutation or competing conventions problem. Consider two neural networks which are computationally equivalent, but contain hidden units in different order. With most of the encoding schemes examined, these networks will be phenotypically identical, but will look different from a genotype point of view.

If we take a very simple example, where the genotype portion concerned with the hidden units for networks X and Y are AB and BA the problem can be illustrated. If these two networks are selected for mating, the offspring produced will be unlikely to succeed in the population, as it will be missing essential portions of its network - its genotype would become AA or BB after crossover occurs. The permutation problem is thought to rise at a rate of n!, where n is the number of networks[5, 40]. Such a situation is not conducive to the convergence of a genetic algorithm and indeed has been the reason why some researchers have abandoned the use of crossover altogether[11, 19].

However, it has been found that the permutation problem does not seem to have much of an impact in the performance of genetic algorithms. In addition, several schemes have been designed to address it. Hancock[40] and Whitley[4] suggest assigning roles to each of the hidden units in the network so as to break the symmetry between otherwise identical networks. A system tracking the historical provenance of genes is proposed by Stanley and Miikkulainen[17]. With this system, it is possible to ascertain the an-

cestry of a particular inherited neuron or connection, again breaking the symmetry which causes the permutation problem.

4 Conclusion

It has been shown repeatedly that the evolution of neural networks is by far superior, both in terms of development time and performance, to hand fabricated methods of development. This paper has outlined the main aspects of neural network evolution which have been researched, as well as the possible encoding schemes for transforming the network into a gene code used by the genetic algorithm

In terms of future directions, it is important to note that while many evolutionary combinations have been tried, many have not yet been combined. The most important development has been the combination of architecture and weight evolution, but this could be built upon by the addition of learning rule or transfer function evolution.

There is also a need for an objective analysis and comparison of the encoding schemes presented, particularly the indirect variety. Many of these lack a theoretical basis for their existence and while many have proven to be superior to direct schemes, some have been found to be equivalent. It is also interesting to note that the permutation problem, while it has raised concerns, has not hindered the use of schemes which fall prey to it and that but for a few exceptions, many rely on the crossover operator to power their genetic algorithm.

Finally, whilst some research has been done on the collaborative aspects of genetic algorithm populations, it would be interesting to ascertain if the addition of communication could yield better results to the evolutionary process.

References

[1] R. S. Sutton. Two problems with backpropagation and other steepest-descent learning procedures for networks. In *Proc. of 8th Annual Conf. of the Cognitive Science Society*, pages 823–831, 1986.

[2] John F. Kolen and Jordan B. Pollack. Back propagation is sensitive to initial conditions. In Richard P. Lippmann, John E. Moody, and David S. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 3, pages 860–867. Morgan Kaufmann Publishers, Inc., 1991.

[3] T. Sasaki and M. Tokoro. Evolving learnable neural networks under changing environments with various rates of inheritance of acquired characters: Comparison between darwinian and lamarckian evolution. *Artificial Life*, 5(3):203–223, 1999.

[4] D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Computing*, 14:347–361, 1990.

[5] Richard K. Belew, John McInerney, and Nicol N. Schraudolph. Evolving networks: Using the genetic algorithm with connectionist learning. In Christopher G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen, editors, *Artificial Life II*, pages 511–547. Addison-Wesley, Redwood City, CA, 1992.

[6] D. J. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 762–767. San Mateo, CA: Morgan Kaufmann., 1989.

[7] H. de Garis. Genetic programming: building artificial nervous systems using genetically programmed neural network modules. In B. W. Porter and R. J. Mooney, editors, *Machine*

Learning: Proceedings of the Seventh International Conference, pages 132–139, Austin, TX, 21-23 1990. Morgan Kaufmann, Palo Alto, CA.

[8] J. Branke. Evolutionary algorithms for neural network design and training. *Technical Report No. 322, University of Karlsruhe, Institute AIFB*, 1995.

[9] K. Chellapilla and D. B. Fogel. Evolving neural networks to play checkers without relying on expert knowledge. *IEEE Trans. Neural Networks*, 10(6):1382–1391, 1990.

[10] Peter J. Angeline, Gregory M. Saunders, and Jordan P. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(1):54–65, January 1994.

[11] Xin Yao. Evolving artificial neural networks. In *Proceedings of the IEEE*, pages 1423–1447, 1999.

[12] John R. Koza and James P. Rice. Genetic generation of both the weights and architecture for a neural network. In *International Joint Conference on Neural Networks, IJCNN-91*, volume II, pages 397–404, Washington State Convention and Trade Center, Seattle, WA, USA, 8-12 1991. IEEE Computer Society Press.

[13] P. M. Todd G. F. Miller and S. U. Hedge. Designing neural networks using genetic algorithms. In *Proceedings of the Third International Conference onGenetic Algorithms and Their Applications*, pages 379–384, 1989.

[14] H. Kitano. Designing neural networks using genetic algorithm with graph generation system. *Complex Systems*, 4:461–476, 1990.

[15] Steven Harp and Tariq Samad. Genetic synthesis of neural network architecture. In L. Davis, editor, *Handbook of Genetic Algorithms*, pages 202–221. Van Nostrand Reinhold, 1991.

[16] Sean Luke and Lee Spector. Evolving graphs and networks with edge encoding: Preliminary report. In John R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1996 Conference Stanford University July 28-31, 1996*, pages 117–124, Stanford University, CA, USA, 28–31 1996. Stanford Bookstore.

[17] K. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. technical report ai01-290. Technical report, Department of Computer Science, University of Texas as Austin, 2001.

[18] David W. White. *GANNet: A genetic algorithm for searching topology and weight spaces in neural network design.* PhD thesis, University of Maryland College Park, 1994.

[19] Xin Yao and Yong Liu. Evolving artificial neural networks through evolutionary programming. In *Evolutionary Programming*, pages 257–266, 1996.

[20] Edward B. Allen Robert Hochman, Taghi M. Khosgoftaar and John P. Hudepohl. Using the genetic algorithm to build optimal neural networks for fault-prone module detection.

[21] F. Gruau. *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm.* PhD thesis, Centre d'etude nucleaire de Grenoble, Ecole Normale Superieure de Lyon, France, 1994.

[22] R.A. Browse, T.S. Hussain, and M.B. Smillie. Using attribute grammars for the genetic selection of backpropagation networks for character recognition. *Proceedings of Applications of Artificial Neural Networks in Image Processing IV (January 25-28, San Jose, CA)*, 1999.

[23] V. Maniezzo. Searching among search spaces: Hastening the genetic evolution of feedforward neural networks. In R. F. Albrecht, C. R. Reeves, and N. C. Steele, editors, *Artificial Neural Nets and Genetic Algorithms*, pages 635–643. Springer-Verlag, 1993.

[24] B. Zhang and H. Muhlenbein. Evolving optimal neural networks using genetic algorithms with

occam's razor. *Complex Systems 7(3)*, pages 199–220, 1993.

[25] N. Richards, D.E. Moriarty, and R. Miikkulainen. Evolving neural networks to play go. *Applied Intelligence*, 8:85–96, 1997.

[26] P. Koehn. *Combining genetic algorithms and neural networks: The encoding problem.* PhD thesis, University of Erlangen and The University of Tennessee, Knoxville, 1994.

[27] J. H. Holland. *Adaptation in Natural and Artificial Systems.* Ann Arbor MI: The University of Michigan Press, 1975.

[28] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading, MA, Addison-Wesley, 1989.

[29] Frederic Gruau. Automatic definition of modular neural networks. *Adaptive Behaviour*, 3(2):151–183, 1995.

[30] W. Schiffmann, M. Joost, and R. Werner. Application of genetic algorithms to the construction of topologies for multilayer perceptrons. In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 675–682, 1993.

[31] João Carlos Figueira Pujol and Riccardo Poli. Efficient evolution of asymmetric recurrent neural networks using a PDGP-inspired two-dimensional representation. *Lecture Notes in Computer Science*, 1391, 1998.

[32] M. Mandischer. Representation and evolution of neural networks. In R. F. Albrecht, C. R. Reeves, and N. C. Steele, editors, *Artificial Neural Nets and Genetic Algorithms Proceedings of the International Conference at Innsbruck, Austria*, pages 643–649. Springer, Wien and New York, 1993.

[33] David E. Moriarty and Risto Miikkulainen. Discovering complex othello strategies through evolutionary neural networks, 1995.

[34] A. Siddiqi and S. Lucas. A comparison of matrix rewriting versus direct encoding for evolving neural networks. *Proc. of the 1998 IEEE International Conference on Evolutionary Computation*, pages 392–397, 1998.

[35] F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 81–89. MIT Press.

[36] E.J.W. Boers and H. Kuiper. Biological Metaphors and the Design of Artificial Neural Networks. Master's thesis, Niels Bohrweg 1, 2333 CA, Leiden, The Netherlands, aug 1992.

[37] H.-M. Voigt, J. Born, and I. Santibáñez-Koref. Evolutionary structuring of artificial neural networks. Technical Report TR-02-93, TU Berlin, 1993.

[38] D. Parisi A. Cangelosi and S. Nolfi. Cell division and migration in a 'genotype' for neural networks. *Network: computation in neural systems*, 5(4), 1994.

[39] S. Nolfi and D. Parisi. Growing neural networks. *Technical Report, Institute of Psychology, CNR Rome*, 1992.

[40] P. J. B. Hancock. Pruning neural nets by genetic algorithm. In I. Aleksander and J.G. Taylor, editors, *Proceedings of the International Conference on Artificial Neural Networks, Brighton*, pages 991–994. Elsevier, 1992.