Neural Network architectures design by Cellular Automata evolution

Inés M. Galván
Pedro Isasi
José Manuel Molina
Araceli Sanchis

Departamento de Informática
Universidad CarlosIII de Madrid
28911, Leganés, Madrid.
E-mail: igalvan@inf.uc3m.es

ABSTRACT

The design of the architecture is a crucial step in the successful application of a neural network. However, the architecture design is basically, in most cases, a human experts job. The design depends heavily on both, the expert experience and on a tedious trial-and-error process. Therefore, the development of automatic methods to determine the architecture of feed-forward neural networks is a field of interest in the neural network community. These methods are generally based on search techniques, as genetic algorithms, simulated annealing or evolutionary strategies. Most of the designed methods are based on direct representation of the parameters of the network. This representation does not allow scalability, so to represent large architectures very large structures are required. In this work, an indirect constructive encoding scheme is proposed to find optimal architectures of feed-forward neural networks. This scheme is based on cellular automata representations in order to increase the scalability of the method.

Keywords: Neural Networks, Cellular Automata, Machine Learning, Evolutionary Computation.

1. INTRODUCTION

The design of neural network architectures is crucial in the successful application because the architecture may strongly drive the neural network's information processing abilities. In most of the cases exist a large number of architectures of feed-forward neural networks set suitable to solve an approximation problem. Unfortunately, the architecture design is still a human experts job. It depends heavily on the expert experience and on a tedious trial-and-error process. There is no systematic way to design a near optimal architecture automatically for a given task.

Designing the optimal architecture can be formulated as a search problem in the architectures space, where each point represents an architecture. The search space of all possible architectures is very large, and the task of finding the simplest architecture may be an arduous and mostly a random task.

In last years, many works have been centered toward automatic resolution of the design of neural networks architecture [1, 2, 3, 4, 5]. Two representation approaches exist to find the optimum net architecture using Genetic Algorithm (GA): one based on the complete representation of all the possible connections and other based on an indirect representation of the architecture. The first one is called Direct Encoding Method, and is based on the codification of the complete network (connections matrix) into the chromosome of the GA, and starts with Ash's work [6] and continues with other works, including [7, 8, 9, 10]. The direct representation is relatively simple and straightforward to implement. However, it does not scale well since large architectures require very large chromosomes to be represented. It is specially suitable for small and problem dependent particular architectures. In these cases, some unpredictable designs could be reached [9]. However, the capabilities of direct encoding for larger architectures are limited, remaining to be proven whether it can scale up to more complex tasks, because large architectures requires much larger chromosomes. This could end in a too huge space search that could make the method impossible in practice.

In order to reduce the length of the genotype, the search space, and to make the problem more scalable, indirect encoding scheme has been proposed in last years. An indirect scheme under a constructive algorithm, by the other hand, starts with a minimal architecture and new levels, neurons and connections are added, step by step, using some sets of rules. The rules and/or some initial conditions are codified into a chromosome of a GA. In this way, an evolution towards good constructive rules is achieved, and there is no evolution of architectures by themselves. The first indirect encoding consisted in the codification of different connectivity patterns as a whole, instead of each connection particularly [1]. This reduces the search space, limiting the possible pattern connections to those which follows some predetermined rules. These limitations in the connections do not affect to the power of the architecture, because is well known that many connectivity patterns have similar learning abilities. Although the parametric encoding can reduce the length of the binary codification of neural networks architectures, still has similar scalability problems as the direct encoding. One of the works to avoid the scalability problems was from Kitano [11], introducing a constructive scheme based on grammars. The solution proposed by Kitano was to encode networks as grammars, and let the GA to evolve grammars instead of networks architectures. Other works have considered some variations of Kitano's rule descriptions using recursive equations to model the growth of connection matrix

[12]. In this case, the coefficients of some fixed equations were codified as chromosomes and evolved by a GA.

The grammatical approach is not the only proposed representation method. Merril and Port [13] introduce a fractal representation for encoding the architectures, arguing that it is more related with biological ideas than constructive algorithms. They used fast simulated annealing for architectures evolution.

In this work, an indirect constructive encoding scheme, based on Cellular Automata (CA), is proposed to find the optimal architecture of a feed-forward neural network. In this scheme, positions of several seeds in a bidimensional grid are represented in the chromosome. These seeds are used as the initial configuration of a cellular automata which is translated into a feed-forward neural network through a growing procedure as the cellular automata is evolved. Thus, the chromosome length is significantly reduced and the scalability of the method is increased.

This method has been proven to automatically design the optimal architecture of a neural network to forecast one step ahead in a time series. The well known logistic chaotic time series has been used for prediction. The special features of this series makes the problem a good test bed in order to prove the ability of the method to generate good architectures.

2. CELLULAR AUTOMATA REPRESENTING FEED-FORWARD NEURAL NETWORKS

Cellular Automata consist of an n-dimensional grid of M machines, called cells, that could be in different states. Usually, the dimension of the lattice is one or two, and the states of the cells are binary. The cells can change their state depending on the state of the cells in their neighbourhood. This modification is specified through rules. All the cells change their state following same rules. The state of each cell in the lattice in some instant, determines what is called configuration. The evolution of the CA is determined by the application of the rules over a configuration to obtain a new configuration and so on. CA have been mainly studied as mathematical models for many physical systems [14].

The features of a CA depend on the rules that govern its behaviour. Some works have been done to automatically evolve the rules of cellular automata. In 1988, Packard [15] used a GA to evolve CA for simple tasks. More recently other techniques as Genetic Programming have been used for the same problem [16]. More difficult problems, as the density-classification problem [17,18], and the synchronisation problem [19] have proven the capabilities of GA to find CA's able to solve complex problems.

2.1 Cellular automata description

For generating neural networks architectures, a bi-dimensonal CA has been proposed. The size of the bi-dimensional grid of the automata depends on the number of input and output neurons given by the problem and on the maximum number of hidden neurons to be considered. Let Dim_x be the number of rows in the grid, its value is equal to the number of input neurons plus the number of output neurons in the problem. On the other hand, Dim_y is the number of columns in the grid and corresponds with the maximum number of hidden neurons to be considered. Each cell in the grid could be in different states, active, inactive, or occupied by a seed. Two different kinds of seeds have been introduced: growing seeds and decreasing seeds. The growing seeds could be different $(s_1, s_2, ..., s_m)$, while the decreasing seeds are all the same (d). The number of seeds of each type is a parameter of the method. Each seed type corresponds with a different type of rule, so there are two rules called growing rule and decreasing rule respectively.

Growing rule: This type of rule reproduces a particular growing seed when there are at least three identical growing seeds in its neighbourhood. There are different configurations, growing seeds located in: rows, columns, or in a corner of the neighbourhood (see table 1). In tables 1 and 2 some of those rules are shown (the others are symmetrical). Here *s* is a specific growing seed, *d* is a decreasing seed, *i* is an inactive state for the cell and *a* means that the cell could be in any state or contains any type of seed (even a decreasing seed).

s	s	s		s	s	s
a	i	a	⇒	a	s	a
a	a	a		a	a	a

s	s	a		s	s	a
s	i	a	⇒	s	s	a
a	a	a		a	a	a

Table 1. Growing rules for some configuration of seeds in the neighbourhood of a particular cell.

Decreasing rules: The growing rules previously specified allow to obtain feed-forward neural networks with a large number of connections. In order to remove connections in the network, decreasing rules are included in the system. These rules deactivate a cell in the grid when the cell has a decreasing seed and two cells in a row of its neighbourhood contain also a decreasing seed. One situation in which the decreasing rules can be applied is showed in table 2, others can be obtained symmetrically.

s	d	a		s	d	a
s	s	a	⇒	s	d	a
a	a	a		a	a	a

Table 2. Decreasing rules for one configuration of seeds in the neighbourhood of a particular cell.

2.2 Cellular automata conversion

The cellular automata evolves the seeds following the previously described rules. A special procedure of activating cells has also been designed in this work. This procedure allows the convergence of the automata toward a final configuration depending on the initial configuration chosen. This final configuration has to be translated into a neural network architecture.

To relate the grid of the automata with an architecture of a neural network, the following meaning for a cell in the grid *(x,y)* is defined: if $x \leq n$, -with *n* the number of input neurones- *(x,y)* represents a connection between the *x-th* input neuron and the *y-th* hidden neuron; if $x>n$, *(x,y)* represents a connection between the *y-th* hidden neuron and the *(x-n)-th* output neuron. The conversion procedure is then as follows:

1) Obtaining a binary matrix *M*.- The places of the grid in which a growing seed appear, are set to 1 and the places corresponding with inactive cells or decreasing seeds are set to 0. Thus, a *Dim1xDim2* binary matrix, $M=(m_{ij})$, is obtained.
2) Processing the binary matrix *M*.- In the above binary matrix the one value is interpreted as a connection, and the zero value as the absence of connection, the rows and columns in the matrix with values 0 are removed. A new and shorter matrix is obtained that has been called processed matrix (*PM*).
3) Obtaining the feed-forward neural network.- For the *PM* matrix, if $pm_{ij} =1$ then a connection between the *i-th* input neuron and the *j-th* hidden neuron is created, or between the *j-th* hidden neuron and the *(i-n)-th* output neuron, as is previously described. If $pm_{ij}=0$, there do not exist connection between that neurones.

2.3 Codification of initial configurations

The rules governing the evolution of the automata are fixed, therefore the final configuration obtained by the automata is related only with its initial configuration. The initial configuration determines the number and position of the different seeds in the grid. In order to evolve initial configurations, a GA has been used. The size of the chromosomes in the GA corresponds with the number of seeds, and it codifies all the possible locations of seeds in the grid. Chromosomes have been codified in base *b*, where *b* is the number of rows in the grid: the number of inputs plus the number of outputs. Each seed is determined by a coordinate *(x,y)*:

1) The first coordinate could be represented by an unique gen, indicating the row in which the seed is located.
2) The second coordinate will need more than one gen, if as usual the maximal number of hidden neurons is bigger than *b*.

In this case, two genes have been used to codify the *y* coordinate, what allows a maximum of $(b-1)xb+b$ hidden neurons. For instance, if there are 3 inputs and 2 outputs, the maximum size of the hidden layer is: 4x5+5 = 25. This could be a good estimation of the maximum number of neurons in the hidden layer, but any other consideration could be taken into account without modifying the proposed method.

x	y	
x_1	y_{11}	y_{12}

Figure 1. Codification of each seed location in the GA chromosome structure.

The chromosome will have 3 genes (Fig. 1) for each seed to be placed in the grid, firstly the growing seeds are represented and finally the decreasing seeds. The number of seeds of each type has to be determined previously, and is a parameter of the method. If, for instance, there are five growing seeds, and five decreasing seeds, the size of the chromosome is 30, divided in ten trios of gens, the first five trios representing the growing seeds, and the final five trios representing the decreasing seeds. It is important to notice that the growing seeds belong to different types of seeds, while the decreasing seeds are all of the same type, as has been shown previously in the description of the automata rules.

2.4 Dynamic of the System

The global system is composed by three different modules: the GA, the CA and the module responsible of neural network training, as is shown in Fig. 2.

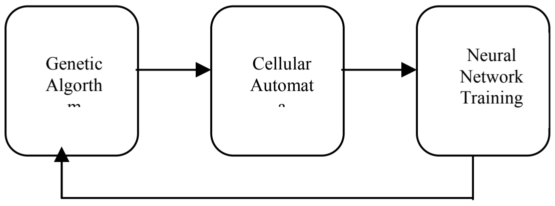

Figure 2. System's description.

All the modules are related to make a general procedure of generating and optimising neural networks architectures. The GA module takes charge of generating initial configurations of the cellular automata, and to optimize these configurations from the information obtained from the training module. The cellular automata takes the initial configuration and generates a final configuration corresponding to a neural network architecture. A special procedure of activating the rules in the cellular automata has been defined in order to generate better, faster and meaningful final configurations. The procedure has to be complete to generate all possible final configurations. This procedure is as follows:

1) The growing seeds are located in the grid.
2) An expansion of the growing seeds takes place. This expansion consists on replicating each seed in turns, on its quadratic neighbourhood, in such a way that if a new seed has to be placed in a position previously occupied by another seed, the first one is replaced.
3) The growing rules are applied until no more rules could be fired.
4) The decreasing seeds are placed in the grid. If there are some other seeds in those places, they are replaced.
5) The decreasing rules are applied until the final configuration is reached.

Finally, the generated architecture has to be trained and evaluated for a particular problem and the error is used as the fitness value for the GA. The complete dynamic of the system is then as follows:

1) Individuals of the population of the GA are randomly generated.
2) Each chromosome is decodified and converted in the grid locations of a number of seeds previously specified.
3) All the cells in the grid are deactivated.
4) The CA is evolved following the previous procedure of rule activation.
5) The final configuration of the CA is translated in a network topology.
6) The neural network obtained is trained to solve the problem.
7) Weights of the neural network are randomly initialised, and learned using the backpropagation learning method. A value is computed after the learning phase of the network, measuring the efficiency of the architecture. This value is composed by not only the error of the network, but some

other considerations could be taken into account to evaluate the efficiency of the network as the generalisation capability, the size of the network, the convergence velocity, etc.... This value is used as the fitness function of the chromosome.

8) Steps 2 to 7 are repeated for all individuals in the population. New populations are generated by GA.

The architectures optimization is carried out by the GA. However, the cellular automata is used as a constructive way of generating the architectures, avoiding direct encoding.

3. EXPERIMENTAL RESULTS

The method has been applied to determine the simplest feed-forward neural network being able to approximate the logistic time series. The logistic map is given by Eq. 1.

$$x(k+1)=\lambda x(k)(1-x(k)) \qquad (1)$$

When $\lambda=3.97$ and $x(0)=0.5$, the map describes a strongly chaotic time series. The use of the logistic map has two main advantages:

1) The chaotic behaviour of the time series makes its prediction a non trivial task.
2) The optimal network to predict the map is known. As the value of the map at instant t depends only on the value of the map at instant t-1, the optimal network has to consider only the input carrying the t-1 signal.

The goal is to approximate the logistic map with a feed-forward neural network. In this case, one output neuron, and one input neuron must be enough to obtain suitable approximations. However, in order to increase the complexity of the problem and to test the ability of the method to generate good architectures, five inputs have been taken into account: x(k-4),...,x(k). Thus, the system must find, in addition to the minimum number of hidden neurons to solve the problem, the most relevant input variables in the dynamic behaviour of the logistic time series, that is, consider only the x(k) input. Some experiments have been carried out varying randomly the location of the inputs, to probe the ability of the method to determine the most useful input (in this case the only useful input) independently of its position in the input layer.

The problem to find the best architecture of neural network has been formulated in three different ways:

1) First, the goal was to find the neural network architecture that provides the best approximation of the logistic time series when the training period is fixed to 1000 learning cycles. In that case, the fitness function maximised by the GA had the expression given in Eq. 2.

$$f_1 = \frac{1}{E} \qquad (2)$$

where E is the mean square error produced by the network in the training phase.

2) In order to find the optimal neural network architecture is necessary to establish an arrangement between the training error produced by the network and the number of connections (number of neurones) in the network. In the context of neural networks is interesting to obtain the simplest network being able to achieve adequate training errors. Hence, the number of connections has been introduced in the fitness function. In the second case, the goal has been to find the simplest architecture of neural network producing the best approximations of the logistic time series when the network is trained during 1000 learning cycles. The fitness function has adopted the expression given en Eq. 3.

$$f_2 = \frac{1}{E \cdot 2^{nc}} \qquad (3)$$

where E is the mean square error produced by the network in the training phase and nc is the number of connections in the network.

3) The number of connections in the network must be measured in order to find the simplest architecture. However, the complexity of the network depends also on the number of learning cycles. A neural network architecture with few connections may required a long number of learning cycles to reach appropriate approximations depending on the placement of the connections. In this third case the goal has been to find the neural network architecture requiring the least computational effort to reach a level of the mean square error. That computational effort depends on the number of connections and on the number of learning cycles. Hence, both are indirectly incorporated in the fitness function. That function is defined as follows: a maximum number of learning cycles is defined as 15.000 and a level for the training error is also set to $1.5\text{x}10^{-4}$. If the network reaches the above level of error, the procedure is then stopped and the fitness function is evaluated as described in Eq. 4.

$$f_3 = \frac{1}{n} \qquad (4)$$

where n is a natural number computed by the sum of the number of connections that have to be changed for each learning cycle. That value depends on the number of connections and on the learning cycles. If the network ends with all the learning cycles without reaching the minimal allowed error, a maximum value of n is associated to that network.

For the above fitness functions, two different approaches have been tested: a classical direct codification and the new method proposed in this work, which are described in the next.

3.1 Results with direct encoding

Five inputs and a maximum of 25 hidden neurons have been considered. The codification consists of a binary matrix representing the existence of connection between an input neuron and a hidden neuron. In this case de dimension of the matrix is *5x25*. This binary matrix constitutes the chromosome of the individual that represents the architecture, therefore the size of chromosomes is 120. The results of the evolution of a GA with this codification are shown in table 3.

Fitness	Architec.	Connec.	Error
f_1	Figure 3	93	0.000878
f_2	Figure 5	4	0.001248
f_3	Figure 4	19	0.001489

Table 3. Results for direct encoding.

In the first fitness function (Eq. 1) only the error of the network has been taken into account. In this case, the complexity of the architecture generated is great, 24 neurons in the hidden layer, because there is no selective pressure toward less complex architectures (Fig. 3). However, architectures with a better error value ($9.8x10^{-4}$ toward $1.4x10^{-3}$) have been achieved. The second fitness function (Eq. 2) gives much more importance to the number of the connections in the architecture, in this case the direct encoding is able to generate the optimal architecture (Fig. 5) because for the logistic map, all the architectures considered could reach an acceptable error value. If the error has few importance in the fitness function, the pressure is drive only toward simple architectures. However, in general, this fitness function will not be able to solve the problem efficiently because in most of the cases short architectures are unable to solve the problems, and bigger architectures are not being generated.

Input	Hidden
t-5	0 0 0 0 1 0 0 1 1 0 0 0 0 1 0 1 1 1 0 0 0 1 0 1
t-4	0 0 0 0 0 0 1 0 0 1 1 0 1 1 1 0 0 0 1 0 0 1 0 1
t-3	0 1 1 1 1 1 1 0 1 1 0 1 0 1 1 1 1 0 1 1 1 1 1 0
t-2	1 1 0 0 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 0 0 1 1
t-1	1 0 0 0 1 1 0 1 1 0 1 1 1 1 1 1 0 1 0 1 0 0 1 1

Figure 3. Best neural topology obtained with direct encoding using f_1 as fitness function.

The last fitness function (Eq. 3) has been introduced to balance both complexity and efficiency, giving more importance to the efficiency than to the complexity and error. In this case the direct encoding produces a complex architecture (Fig. 4).

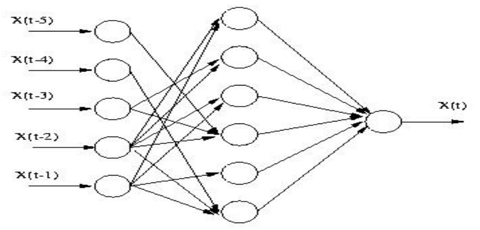

Figure 4. Best neural topology obtained with direct encoding using f_3 as fitness function.

3.2 Results with cellular automata evolution

Since five inputs and one output have been considered, the seeds have to be codified in base 5. For simplicity, all the hidden neurons are considered connected with the only output neuron, this reduces the size of the chromosome in one unit without loosing generality. This assumption can be done only if there is a unique output neuron. The size of the grid for the automata is *5x25*, following the previous descriptions.

Two sets of experiments have been realised, with three growing seeds and three decreasing seeds; and five growing seeds and five decreasing seeds respectively. In the first case the size of the chromosomes was *18=3x3+3x3* and in the second case *30=3x5+3x5*.

The GA is composed of a population of 100 individuals, which are initially randomly generated. This population is iterated for 100 generations in order to maximise the fitness functions previously described.

In table 4 are shown the results of all the experiments for CA evolution. They show the number of seeds used, the best architecture founded, the number of connections of the architecture and the error achieved by the network. Lets notice that error values for the experiments are slightly different, in spite that they represent the same architecture. This is because error values are computed as the average of the error over a set of randomly initialised networks, and the final error depends on the initial weights assignations.

Fitness	Seeds	Architec.	Connec.	Error
f_1	3 and 3	Figure 6.a	59	0.001006
f_1	5 and 5	Figure 6.b	67	0.000986
f_2	3 and 3	Figure 7	5	0.001255
f_2	5 and 5	Figure 7	5	0.001375
f_3	3 and 3	Figure 5	4	0.001429
f_3	5 and 5	Figure 5	4	0.001440

Table 4. Results for cellular automata evolution.

In the experiments for f_1 function the best networks have 59 and 67 connections respectively (Fig. 6). This means that the CA encoding has been able to find significant less complex networks, with similar error values, that with direct encoding. In any case, this fitness function is no able to reduce the complexity of the network, because the generated networks behave enough well in terms of error.

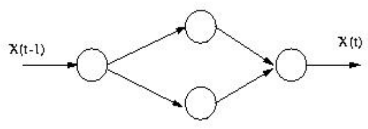

Figure 5. Best neural topology obtained with CA encoding

Inp.	Hidden
t-5	1 1 1 1 1 1 1 1 1 1 1 1
t-4	0 0 1 1 1 1 1 1 1 1 1 1
t-3	0 1 1 1 1 1 1 1 1 1 1 1
t-2	0 0 1 1 1 1 1 1 1 1 1 1
t-1	0 0 1 1 1 1 1 1 1 1 1 1

(a) Seeds 3-3

Inp.	Hidden
t-5	1 1 1 1 1 1 1 1 1 1 1 1
t-4	1 1 1 1 1 1 1 1 1 1 1 1
t-3	1 1 1 1 1 1 1 1 1 1 1 1
t-2	1 1 1 1 1 1 1 1 1 1 1 1
t-1	1 1 1 1 1 1 1 1 0 0 1 1

(b) Seeds 5-5

Figure 6. Best neural topology obtained with CA encoding, using f_1.

In Fig. 5 is represented the architecture found for experiments with f_3 fitness function. In all these experiments the proposed method converges to the same architecture. This neural network is the best possible architecture to solve the problem. It has the less computational cost able to solve the problem efficiently, and an error lower than $1.5x10^{-3}$. In addition, this architecture takes into account the only relevant input to the problem (x(t-1)). Here, the reduction in complexity, compared with direct encoding is also significative, not only because of the less number of connections, but because the CA encoding has been able to detect just the significative input, and to consider it in the solution. Lets notice that to find the significant inputs is, in many cases, an important problem to be solved.

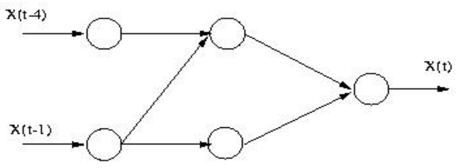

Figure 7. Best neural topology obtained with CA encoding using f_2.

In the case of the second fitness function, the best architecture is not achieved, however a very similar one is found (Fig. 7). The irrelevant x(t-4) input is considered, and the final architecture has only one additional connection.

4. DISCUSSION AND CONCLUSIONS

The election of good neural networks architectures is an important step in many problems where there is few knowledge about the problem itself. Evolutionary computation techniques are good approaches for automatically generate those good architectures. However the codification of the network is a crucial point in the success of the method. Direct codification's become inefficient from a practical point of view, making bigger and redundant the search space. To solve this problem an indirect encoding has to be used.

Indirect encoding is driven to reduce the search space in such a way that similar solutions are eliminated and represented by only one representative. In these cases, the codification makes the method able to find better architectures.

Cellular Automata are good candidates for non-direct codification's. The constructive representation introduced in this work solves some of the problems for non-direct codification's. The final representations has a reduced size and could be controlled by the number of seeds used. The results with the logistic map show how the complexity of the achieved network could be reduced in spite of the fitness function used. This method has also been able to find the appropriate network for the logistic map, identifying the only useful input. This result could have a great practical interest because some times the identification or importance of the inputs is a primordial step in the solution of a problem.

5. REFERENCES

[1] S. Harp, Samad T. and Guha A. Towards the Genetic Synthesis of Neural Networks. Proceedings of the Third International Conference on Genetic Algorithms and their applications, pp 360-369, San Mateo, CA, USA, 1989.

[2] G.F. Miller, P.M. Todd and S.U. Hegde. Designing neural networks using genetic algorithms. In Proc. of the third international conference on genetic algorithms and their applications, pp 379-384, San Mateo, CA, USA, 1989.

[3] S. Harp, Samad T. and Guha A. Designing Application-Specific Neural Networks using the Genetic Algorithm, Advances in Neural Information Processing Systems, vol2, 447-454, 1990.

[4] F. Gruau. Genetic Synthesis of Boolean Neural Networks with a Cell Rewriting Developmental Process. Proc. of COGANN-92 International Workshop on Combinations of Genetic Algorithms and Neural Networks, pp. 55-74, IEEE Computer Society Press, 1990.

[5] F. Gruau. Automatic Definition of Modular Neural Networks. Adaptive Behaviour, vol. 2, 3, 151-183, 1995.

[6] T. Ash. Dynamic Node Creation in Backpropagation Networks ICS Report 8901, The Institute for Cognitive Science, University of California, San Diego (Saiensu-sh, 1988), 1988.

[7] D.B. Fogel, Fogel L.J. and Porto V.W. Evolving Neural Network, Biological Cybernetics, 63, 487-493, 1990.

[8] T.P. Caudell and Dolan C.P. Parametric Connectivity: Training of Constrained Networks using Genetic Algorithms, Proc. of the third International Conference on Genetic Algorithms and their Applications, 370-374. Morgan Kaufman, 1989.

[9] J.D. Schaffer, R.A. Caruana and L.J. Eshelman. Using genetic search to exploit the emergent behaviour of neural networks. Physica D, 42, pp 244-248, 1990.

[10] E. Alba, J.F. Aldana and J.M. Troya. Fully automatic ANN design: A genetic approach. In Proc. of International workshop on artificial neural networks, pp 179-184, 1993.

[11] H. Kitano. Designing Neural Networks using Genetic Algorithms with Graph Generation System, Complex Systems, 4, 461-476, 1990.

[12] E. Mjolsness, D.H. Sharp and B.K. Alpert. Scaling machine learning and genetic neural nets. Advances in applied mathematic, 10, pp 137-163, 1989.

[13] J.W.L. Merril and R.F. Port. Fractally configured Neural Networks. Neural Networks, 4, 53-60, 1991.

[14] S. Wolfram. Theory and applications of cellular automata. World Scientific, Singapore, 1988.

[15] 15 N.H. Packard. Dynamic patterns in complex systems. Chapter Adaptation toward the edge of chaos, pp 293-301. World Scientific, Singapore, 1988.

[16] J.R. Koza. Genetic programming: On the programming of computers by means of natural selection. MIT Press, Cambridge, MA, 1992.

[17] R. Das, M. Mitchell and J.P. Crutchfield. A genetic algorithm discovers particle based computation in cellular automata. In Parallel problem Solving from Nature, Vol 866 of Lecture Notes in computer Science, pp 244-353, Berlin, 1994.

[18] J.P. Crutchfield, M. Mitchell. The evolution of emergent computation. In Proc. Of the national academy of sciences, vol 92, 23, 1995.

[19] R. Das, J.P. Crutchfield, M. Mitchell and J.E. Hanson. Evolving globally synchronised cellular automata. In Proc. of the Sixth International Conference on genetic Algorithms, pp 336-343, San Francisco, CA, 1995.