

Genetic Synthesis of Modular Neural Networks.

Frédéric Granau*

Centre d'étude nucléaire de Grenoble
DRFMC SP2M PSC, BP85X 38041 Grenoble France

Abstract

Cellular Encoding is a method for encoding families of boolean neural networks having the same structure, that can compute scalable boolean functions. The current study describes how to incorporate modularly into Cellular Encoding. A Genetic Algorithm is used to find part of a modular code that yields both architecture and ± 1 weights specifying a particular neural network for solving the decoder boolean function of 10 inputs, 1024 outputs. This result suggests that the GA can exploit modularly in order to find architectures within a more complex range.

In any cases, making cross-over directly on the hardware of the connections can produce non-functional offsprings. The solution is to encode on the chromosome a collection of abstract features more likely to provide building blocks.

Parameterized encoding A list of parameters is encoded. In [7] the parameters describe the number of layers, the size of the layers, and how layers are interconnected. This method is more abstract, it could provide big nets with small chromosomes, but only within a restricted range of architectures: layered architectures.

Grammatical Encoding A rewriting grammar is encoded. The grammar is interpreted in a recursive manner, and allows to generate a family of related neural networks. Grammatical encoding allow to develop big neural nets with small code without restricting the range of architectures. This method clearly improves upon the two previous.

Kitano introduced grammatical encoding. He uses a grammar that rewrites integer matrices. He develops a family of integer matrices from which a family of neural nets can be computed, using some conversions [8]. There are certain problems with Kitano's representation scheme. An $m \times m$ matrix must be developed for a network of n neurons, where m is the smallest power of 2 bigger than n . In order to get an acyclic graph for a feed forward neural network, one must consider only the upper right triangle, which also decreases the efficiency of the encoding.

Mjolness [11] defines a recursive equation for a matrix from which he computes a family of integer matrices and then a family of weighted neural nets; in this sense, the approach is similar to that used by Kitano. Unlike Kitano, Mjolness is not restricted to only using matrices of size 2^n ; it is not clear, however, what is the range of possible sizes. Also, the size of the neural networks has to be determined in advance.

Granau [3] directly develops a family of neural nets and avoids the need to go through the matrix representation. Instead of a matrix, the object on which the grammatical rules are applied is the cell of a network. Each cell has a copy of the chromosome. Each cell reads the chromosome at a different position. Depending on what it reads, a cell can divide, change some internal parameters, and finally become a neuron. It is both more natural and more efficient to act development at the level of cells rather than on elements of a connection matrix. The resulting language can describe networks in a clear and compact way, and the genetic algorithm. Various properties of cellular encoding have been formalized and proved in [6]. We used a genetic algorithm to recombine grammar trees representing cellular encodings and showed that neural networks for the parity problem and symmetry problem of arbitrary large size could be found.

Definition: Consider a network N_1 that includes at many different places, a copy of the same subnetwork N_2 . The code of N_1 is modular if it includes the code of N_2 , a single time.

In this paper we incorporate modularly into the Cellular Encoding. Modularity allows to tackle more difficult problem. We report an experiment where a family of neural-networks for the decoder boolean function of 10 inputs, 1024 outputs could be found by the Genetic Algorithm.

2 Review of cellular development.

We now present the details of cellular encoding. The chromosome is represented as a tree called *grammar tree* with ordered branches whose nodes are labeled with characters called program symbols. A cell is a node of an oriented *network graph* with ordered connections. Each cell carries a duplicate copy of the chromosome (i.e., the grammar tree) and has an internal *reading head* that reads from the grammar tree: typically, each cell reads from the grammar tree at a different position. The program symbols represent programs or instructions for cell development that act on the cell or on connections that fan-in to the cell. During a step of the development process, a cell executes the instruction referenced by the symbol it reads, and moves its reading head down in the tree. One can draw an analogy between a cell and a Turing machine. The cell reads from a tree instead of a tape and the cell is capable of duplicating itself; but both execute instructions by moving the reading head in a manner consistent with the symbol that is read.

A cell also manages a set of internal registers, some of which are used during development, while others

determine the weights and thresholds of the final neural net. The link register is used to refer to one of possibly several fan-in connections (i.e., links) into a cell. Figure 1 shows the development of a network for the decoder. This figure is complex, we refer the reader to [3] for a more simple example of development. We present here a more complex encoding for three purposes. It shows the power of cellular encoding. It presents a non-trivial boolean function, the decoder, for which a modular decomposition is needed to encode the corresponding network family. It allows to better understand how the GA can find a solution to the decoder problem.

Development starts with a single cell called the *ancestor* cell connected to an input pointer cells and an output pointer cell. Consider the starting network named *start* and the grammar-tree depicted in figure 1 (b) or (c). At the starting step the ancestor cell possesses a reading head positioned on the root of the tree. Its registers are initialized with default values. For example, its threshold is set to 0. As this cell repeatedly divides it gives birth to all the other cells that will eventually make up the neural network. The 2 input/output pointer cells to which the ancestor is linked do not execute any program-symbol. Rather, at the end of the development process, the upper pointer cell is connected to the set of input units, while the lower pointer cell is connected to the set of output units. These input and output units are created during the development, they are not added independently at the end. After development is complete the pointer cells can be deleted. For example, in figure 1 (a), the final decoded neural net has two input units labeled 1 and 2, and four output units labeled 1, 2, 3 and 4. It is possible to start the development from a more complex network-graph, as in figure 1 (a). There are different kinds of programs and program-symbols.

1 Introduction

Most classical learning algorithms aim at finding weights for a neural network whose architecture is frozen. On the other hand, a GA generates a number of different architectures. It is appealing to combine both approaches in a single type of algorithm. A usual GA handles strings of characters called chromosomes. In order to apply a GA to the problem of finding good neural network architectures, there is nothing but one problem to solve: how to code an architecture on a chromosome. Three class of methods exist:

Direct encoding. A graph data structure is encoded. In [10] the connectivity matrix of the network is encoded. This gives very long chromosomes of size $O(n^2)$ for a network including n neurons. Because of this scalability problem, only small networks (four hidden units) have been found. An improvement can be to encode only a list of links ([1]), the size of the chromosome is $O(l)$ for a network with l links.

*Laboratoire de l'informatique du parallélisme, Ecole Normale Supérieure de Lyon 47 Allée d'Italie 69007 Lyon granau@linfo.cnrs.fr

166A-93

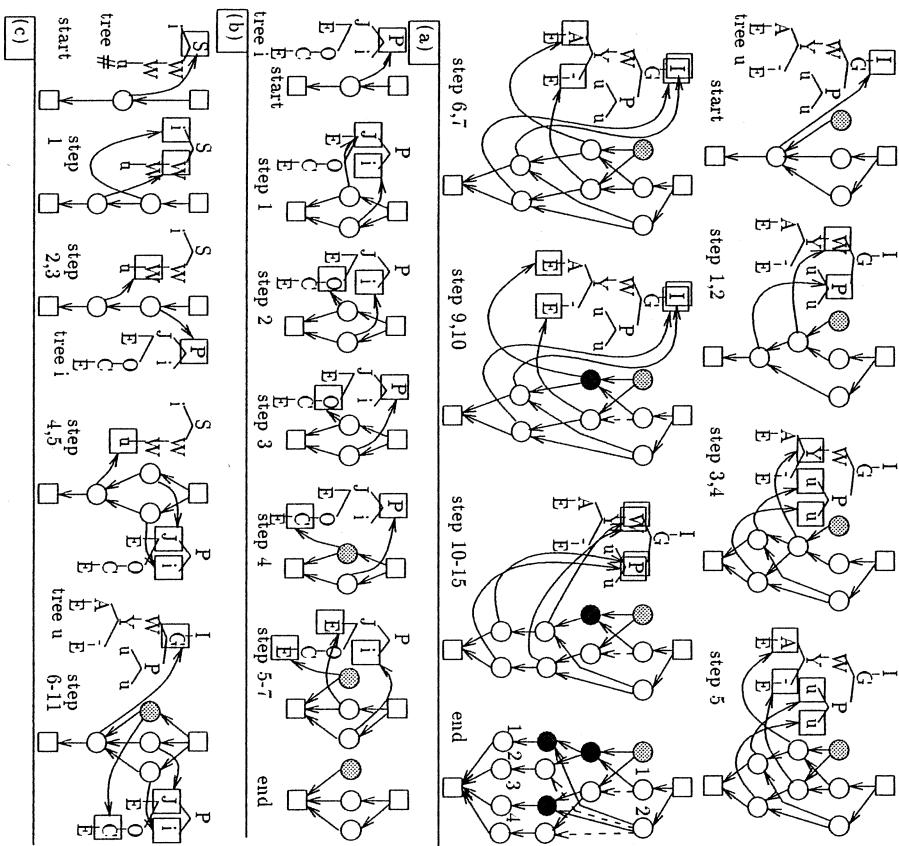


Figure 1: The 2-input decoder network. (a) Starting from an initial network graph already containing subnetwork 1. (b) Development of sub-network 1. (c) First 11 steps of the development of the whole 2-input decoder network. At step 11, the network-graph is similar to the initial network-graph of (a), the last steps can therefore be deduced from (a). Circles represent active cells or neurons. Cells are connected to their reading head with a curved arrow. Reading head are represented by boxes around letters. If the threshold is 0, the circle is empty, while if the threshold is -1 the circle is filled with gray. Squares represent input/output pointer cells. If the threshold is 1 the circle is filled with black. A continuous line indicates a weight 1, and a dashed line a weight -1.

• A *value program-symbol* modifies the value of an internal register of the cell. The program-symbol **I** increments (and **D** decrements) the value of the link register, which points to a specific fan-in link or connection (fig. 1 (a) step 1). Changing the value of the link register causes it to point to a different fan-in-connection. The link register has a default initial value of 1, thus pointing to the leftmost fan-in link. Operations on other connections can be accomplished by first resetting the value of the link register. An unary program-symbol **C** cuts the link pointed by the link register (fig. 1 (b) step 6). The program-symbol denoted **+** sets the weight of the input link pointed by the link register to 1, while **-** sets the weight to -1 (see fig. 1 (a) step 10). The program-symbols **C**, **+**, and **-** do not explicitly indicate to which fan-in-connection the corresponding instructions are applied. When **C** or **+** or **-** is executed it is applied to the link pointed to by the link register. The program-symbol denoted **A** increments (and **O** decrements) the threshold (fig. 1 (a) step 9, (b) step 4).

• A *division program-symbol* creates two cells from one. In a *sequential division* (denoted by **S**) the first child inherits the input links, the second inherits the output links of the parent cell and the first child connects to the second with weight 1. The link is oriented from the first child to the second child. This is illustrated in step 1 fig. 1 (c). In a *parallel division* (denoted by **P**) both child cells inherit the input and output links from the parent cell (in fig. 1 (a) step 4 or (b) step 1 and step 5). In a *gradual division* (denoted by **G**) the first child inherits the first *r* input links, where *r* is the value of the link register, the second child inherits the other input links and all the output links: the first child connects to the second with weight 1 (in step 2, 14 and 15 fig. 1 (a)). In a *separation division* (denoted by **Y**), both child cells inherit all the input links, the first child inherits the first half of the output links, the second child inherits the second half (step 5 fig. 1 (a)). Since there are two child cells, a division program-symbol must label nodes of arity two. The first child moves its reading head to the left subtree and the second child moves its reading head to the right subtree. Finally, when a cell divides, the values of the internal registers of the parent cell are recycled in the child cells.

• The *waiting program-symbol* denoted **W** makes the cell wait for its next rewriting step. In some cases, the final configuration of the network depends on the order in which cells execute their corresponding instructions. **W** is necessary for those cases where the development process must

be controlled by generating appropriate delays.

- The *ending program-symbol* denoted **E** causes a cell to loose its reading head and become a neuron. Since the cell does not read any subtrees of the current node, there need not be any subtrees to the node labeled by **E**. Therefore **E** labels the leaves of the grammar tree (i.e., nodes of arity 0).

The sequence in which cells execute program-symbols is determined as follows: once a cell has executed its program-symbol, it enters a First In First Out (FIFO) queue. The next cell to execute is the head of the FIFO queue. If the cell divides, the child which reads the left subtree enters the FIFO queue first. This order of execution tries to model what would happen if cells were active in parallel. It ensures that a cell cannot be active twice while another cell has not been active at all. Because of this particular rewriting order, the grammar-tree is read in *breadth-first* manner. Nodes of the tree of equal depth are scanned from left to right.

The signato *s* of the neuron is defined by $s(x) = 1$ if $x > 0$, and $s(x) = 0$ if $x \leq 0$. Neurons execute a program that removes them from the network if they are not useful. A neuron is useless if its output is always 0, or if it computes the identity. If the neuron has some links to certain upstream neurons, deleting means that these links disappear and the neuron identifies itself with the first upstream neuron.

3 Adding modularity to the Cellular Encoding

We now consider a system of grammar-trees, each one has its own name which for convenience will be a single lower-case letter, except for the *axiom grammar tree*, whose name is **#**. At the beginning of the rewriting, the ancestor cell is positioned on the root of the axiom grammar tree. For each grammar-tree *x*, there exists a program-symbol denoted by *x* which includes the sub-network encoded by that grammar tree. When a cell reads the program-symbol *x*, it positions its reading head on the root of the tree which name is *x*. If a grammar tree includes itself, the development enters an infinite loop! In this particular case, the cell executes the following algorithm before moving its reading head:

```

1- life := life - 1
2- If (life > 0)
   reading-head := root of current tree
3- Else loose reading-head
   become a neuron.

```

¹It amounts to use a program symbol of recursion R defined in [3]

The variable *lfr* is a register of the cell. It is initialized with *L* in the ancestor cell. Thus a grammar develops a family of neural networks parametrized by *L*. This implementation of the recurrence allows precise control of the growth process. The development is not stopped when the network size reaches a predetermined limit as in [9], but when the code has been read exactly *L* times through. The number *L* parametrizes the structure of the neural network. We also introduce a branching program-symbol denoted *J* which labels nodes of arity 2. A cell that reads *J* executes the following algorithm:

```

1- If (Life = L)
    reading-head := right subtree
                    of current node
2- Else
    reading-head := left subtree
                    of current node
    
```

Figure 1 (c) shows that the decoder-network consists of 2 sub-networks sequentially connected. The first one is encoded by *i* and develops the input units, as well as a unit which constantly outputs 1. It includes itself once, therefore its size is $O(L)$. The second one is encoded by *u*. It uses one by one the neurons generated by sub-network *i*. It includes itself twice, therefore its size is $O(2L)$. The decoder network is developed for $L = 2$ in figure 1. The development can be done for an arbitrary large integer *L*. Moreover, the code is modular according to the definition given in the introduction. In general, a cellular encoding will always be modular, by using one grammar-tree per sub-network.

4 Genetic Programming and Cellular Development

Genetic Programming has been defined by Koza as a way of evolving computer programs with a genetic algorithm [10]. In Koza's Genetic Programming paradigm the individuals in the population are LISP *s*-expressions which can be depicted graphically as rooted, point-labeled trees with ordered branches. Since grammar trees have the same structure, the same approach is used for generating and recombining grammar trees. The set of alleles is the set of cell program-symbols that label the tree, the search space is the hyperspace of all possible labeled trees. During crossover, a subtree is cut from one parent tree and replaces a subtree from the other parent tree; the result is an offspring tree. The sub-trees which are exchanged during recombination are randomly selected. An individual is an ordered set of grammar-trees. Koza has also implemented the cross over between two ordered sets of grammar-trees by recombining together grammar-trees that have the same rank, like in the human genome. We look the same approach.

We have designed a sequential genetic algorithm that uses local selection and mating based on a geographical lay out of individuals. Individuals reside on a 2-D grid and mate with the best chromosome found during a random walk in the neighborhood. This implements isolation by distance [2], such that different local regions on the grid may display local trends in terms of evolutionary development. Using this sequential GA, we designed a parallel GA. The same sequential GA runs on 64 processors 1860 of an IPSC parallel machine. Is is a MIMD machine. These processors are arranged on a 2-D torus. The GA is steady-state. On a given processor, each time the GA wants to create a new genome, with a probability .99 it performs a cross-over, with a probability 0.01 it exchanges a genome chosen during a random walk near the border of its 2-D grid, with the processor neighbor with respect to this border.

A genome develops a family of neural-nets (N_L). The neural-net N_L should compute a parametrized function f_L , for $L_{min} < L < L_{max}$. The GA does not typically develop all the neural networks N_L for $L_{min} < L < L_{max}$ in the early generations. The following rule decides exactly when it is necessary to develop the neural network $N_{L_{min}+1}$, and then $N_{L_{min}+2}$ and so on, until $N_{L_{max}}$.

```

Developing Rule: If  $N_{L_{min}}, N_{L_{min}+1}, \dots, N_L$  have already been developed, then,
continue to develop  $N_{L+1}$  if  $N_L$  correctly
processes more than a fraction r of the
training set, where r is a given threshold.
If  $L > L_{max}$  stop the development.
    
```

We use $L_{max} = 10$ and $r = .7$ in our experiments. In the parallel GA, L_{min} varies regularly among the 64 processors between 1 and 3. The processors for which $L_{min} = 3$ directly start to develop N_3 whereas those for which $L_{min} = 1$ start by N_1 . There is three different fitness functions, each one represents a different feature of the problem. So each processor develops a population with a view to find the building block corresponding to a particular feature, and the final solution can be found through the recombination of exchanged genomes. Genetic diversity is maintained until a code that develops N_1, N_2, N_3 right is found. But such a code is likely to develop N_L right for $L > 3$, because the recurrence is learned. Hence genetic diversity can be kept until a general solution is found.

Mutation and the generation of random genomes are implemented so as to generate a hierarchy between the grammar-tree of the genome. A grammar-tree *x* cannot include a grammar-tree *y* if $x < y$. This prevents looping between grammar-trees.

5 Experiments

We try to find a network family N_L such that N_L computes the decoder of size *L*. N_L must have *L* input units and $2L$ output units. If the binary code of an integer $i < 2^L$ is clamped on the input units, the *i*th output must be set to 1, all the others to 0. This problem is more complex than the parity or the symmetry treated in [3], with respect to Cellular Encoding. In section 3, we have presented an encoding for a decoder network that needs 3 grammar trees, whereas parity or symmetry networks can be encoded with a single grammar-tree.

In order to help the GA, Mühlenbein [12] promotes the use of individual hill-climbing. Each individual generated by the GA should try to enhance itself with hill-climbing. In the case of neural network optimization, a learning procedure can be used as a hill-climber. The learning procedure defined in [4] was implemented. This learning is applied for each genome produced by the GA, on the first neural network of the family. The learned information is then coded back on the grammar-trees and can be used to develop the other networks of the family. The method called Developmental Learning is studied in [3].

The genome recombined by the GA encompassed two grammar-trees named *i* and *u*. The axon grammar-tree was predefined as $S(i)(W(W(u)))$. This tree indicates the general structure of the network. It is made of two sub-networks *i* and *u*, sequentially connected. The development of *u* is delayed 2 times, because *u* needs neurons produced by *i*. We predefined another grammar-tree named *b* which is $J(E)(O(-(C(E))))$. It encodes a sub-network that consists of a single unit, which outputs a constant 1, or performs the identity, depending on the value of the life register. The set of alleles is { P, S, G, Y, W, E, I, D, A, O, -, +, u, i, b }.

Figures 2 shows the genome found by the GA. The corresponding $L=3$ neural network solves the decoder of *L* inputs with $4 * 2^L - 2$ hidden units and $15 * 2^L - 6$ links. It is clearly a combination of $2^L - 1$ sub-networks. We tested N_L until $L=10$. Although there is no formal proof, it seems obvious that N_L solves the decoder of *L* inputs for an arbitrary large *L*, because the GA learns the recurrence. The GA used was quite powerful. The sub-population size on each processor was 128, the total population size on the 64 processors was 8192. The number of generations averaged over the processors was about 130. So the total number of genomes processed was around 1 000 000. The total cpu time taken was one hour, it amounts roughly to four days of sun4, sparc2 workstation.

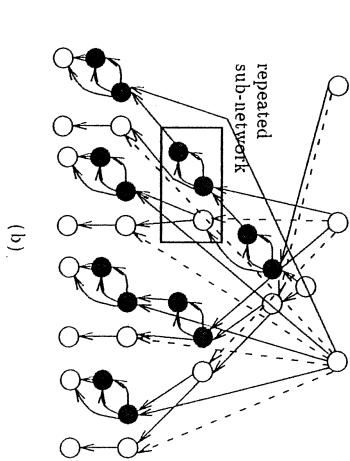
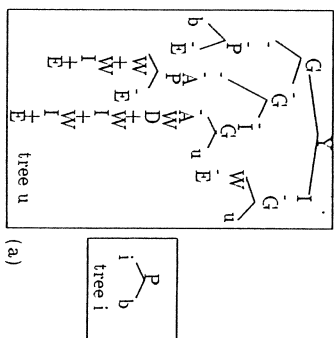


Figure 2: (a) The genome found by the GA. (b) the neural net for $L=3$.

6 Conclusion

In this paper an encoding method of neural networks called Cellular Encoding is enhanced. The improvement allows to encode neural networks in a modular fashion. We show how to develop a boolean neural network for the decoder boolean function, using two sub-networks. Experiments report that the GA could simultaneously evolve the two sub-networks and generate a neural network that computes the decoder boolean function of 10 inputs, 1024 outputs. However, the GA had to be helped with two pieces of code provided by hand. The first one describes how to include the sub-networks searched by the GA, the other one encodes a sub-network which can be then used by the GA as a building block. This way of adding information is a general technique, interesting in itself. The decoder is a boolean function which

belongs to a more complex class compared to the parity and the symmetry studied in [3], with respect to Cellular Encoding. The corresponding architecture although regular, is more elaborate. Both type of architectures are compared in the appendix. Because the decoder was so hard, we had to design an improved GA that efficiently preserves genetic diversity, in order to solve the decoder problem.

Acknowledgement

Support from the Centre d'étude nucléaire de Grenoble and the project C3 is acknowledged. I am grateful to John Koza for a discussion at Baltimore, where he gave me the idea of recombining sets of trees with the GA.

References

- [1] R. Collins and D. Jefferson. Antfarm: toward simulated evolution. In *Artificial life II*, 1990.
- [2] R. Collins and D. Jefferson. Selection in massively parallel genetic algorithm. In *4th Intern. Conf. on Genetic Algorithms*, 1991.
- [3] F. Grunau. Genetic synthesis of boolean neural networks with a cell rewriting developmental process. In *Combination of Genetic Algorithms and Neural Networks*, 1992.
- [4] F. Grunau. A learning and pruning algorithm for genetic neural networks. In *European Symposium on Artificial Neural Network*, 1993.
- [5] F. Grunau and D. Whitley. Adding learning to the cellular developmental process: a comparative study. Research report r93-04, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, 1993.
- [6] Frédéric Grunau. Cellular encoding of genetic neural network. Technical report 92.21, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, 1992.
- [7] S. Harp, T. Samad, and A. Gulha. Toward the genetic synthesis of neural networks. In D. J. Schaffer, editor, *3rd Intern. Conf. on Genetic Algorithms*, pages 360–369, 1989.
- [8] H. Kitano. Designing neural network using genetic algorithm with graph generation system. *Complex Systems*, 4:461–476, 1992.
- [9] H. Kitano. Neoungenic learning an integrated method of designing and training neural networks using genetic algorithms. *Conn-conn-92-134*, Carnegie Mellon university, 1992.
- [10] John R. Koza. *Genetic programming: A paradigm for genetically breeding computer programs for solving problems*. MIT press, 1992.
- [11] Eric Mylness, David Sharp, and Bradley Alpert. Scaling machine learning and genetic neural nets. La-ur-88-142, Los Alamos National Laboratory, 1988.
- [12] H. Muehlenbein, M. Schomish, and J. Born. The parallel genetic algorithm as function optimizer. *Parallel Computing*, 17:619–632, 1991.
- [13] D. Whitley, T. Stakweather, and C. Bogart. Genetic algorithms and neural networks: optimizing connection and connectivity. *Parallel Computing*, 14:347–361, 1990.

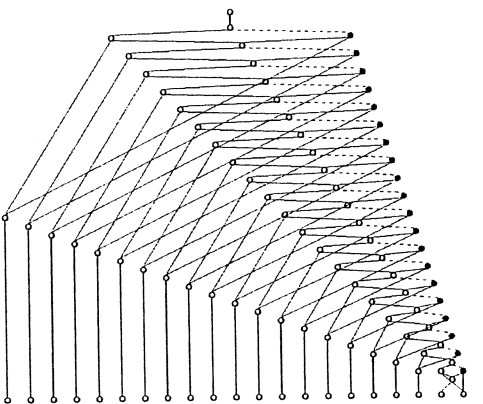


Figure 3: A neural network found using the GA for the parity of 21 input units, 1 output unit. The architecture is layered, the fan-out is bounded by 2, the complexity is linear.

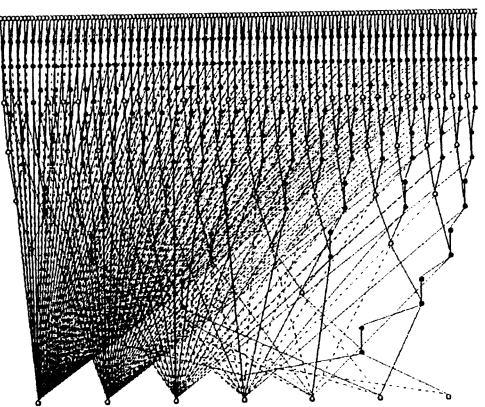


Figure 4: The neural network found byt the GA for the decoder of 7 input unit, 128 output units. The links are arranged in a structure of trees, the fan-out is unbounded, the complexity is exponential.