USING MARKER-BASED GENETIC ENCODING OF NEURAL NETWORKS TO EVOLVE FINITE-STATE BEHAVIOUR *†

Brad Fullmer and Risto Miikkulainen
Department of Computer Sciences
The University of Texas at Austin, Austin, TX 78712-1188
email fullmer,risto@cs.utexas.edu

Abstract

A new mechanism for genetic encoding of neural networks is proposed, which is loosely based on the marker structure of biological DNA. The mechanism allows all aspects of the network structure, including the number of nodes and their connectivity, to be evolved through genetic algorithms. The effectiveness of the encoding scheme is demonstrated in an object recognition task that requires artificial creatures (whose behaviour is driven by a neural network) to develop high-level finite-state exploration and discrimination strategies. The task requires solving the sensory-motor grounding problem, i.e. developing a functional understanding of the effects that a creature's movement has on its sensory input.

1 Introduction

The behaviour of a particular biological organism is driven by its neural circuitry. In modeling artificial life forms it is therefore natural to represent the organism as an artificial neural network (ANN) with a set of sensory inputs and motor outputs. ANNs have been shown to be capable of very complex processing, and in most cases they can learn the processing task from examples (Siegelman and Sontag, 1991; McClelland et al., 1986).

However, the usual neural network learning algorithms are not always useful in artificial life problems. Many algorithms, such as backpropagation (Rumelhart et al., 1986), require that the correct output is known at each input situation. This requirement is relaxed in reinforcement learning, where only an estimate of the goodness of the action (or sequence of actions) is needed for learning (Barto et al., 1983). In artificial life, even this feedback may not be immediately available. For example, if the artificial creatures are supposed to learn cooperation in a complicated task, there is no easy way to specify what the correct actions at each point are, or even whether a particular sequence of actions is good or bad.

For this reason, genetic algorithms (Holland, 1975; Goldberg, 1988) are naturally well-suited for developing neural networks in artificial life. It is only necessary to specify a fitness function that estimates how well the creature performs in the task over its lifetime. The best creatures are then genetically combined to produce offspring, thereby increasing the density of successful traits in the population. Over many generations, the average fitness of the population improves until a sufficient proficiency level is attained. As in biological evolution, the population adjusts to evolutionary pressures by developing advantageous attributes including high-level behavioural strategies and low-level sensory processing capabilities. Genetic algorithms have been used previously to evolve various types of behaviour in artificial creatures such as following a broken trail (Jefferson et al., 1991), foraging for food (Collins and Jefferson, 1991) and communicating instructions (Werner and Dyer, 1991).

A central question in the neuro-evolution approach is how the network structure can be represented in terms of genetic information so that genetic algorithms are maximally effective. In this paper, a new representation mechanism that is loosely based on the marker structure of DNA is proposed. Unlike previous approaches, this mechanism allows all aspects of the network structure, including the number of nodes and their connectivity, to be controlled by evolution.

The effectiveness of the encoding scheme is demonstrated in an object discrimination task. Successful completion of this task requires that the creatures develop high-level finite-state exploration strategies. The creatures in our experiments possess a primitive visual apparatus, i.e. their input consists of a coarse

*This research was supported in part by a grant from the University of Texas Research Institute to the second author. Majority of simulations were run on a Cray Y–MP8/864 at the University of Texas Center for High-Performance Computing.

†To appear in *Proceedings of the First European Conference on Artificial Life (ECAL-91)*, Paris, 1991.

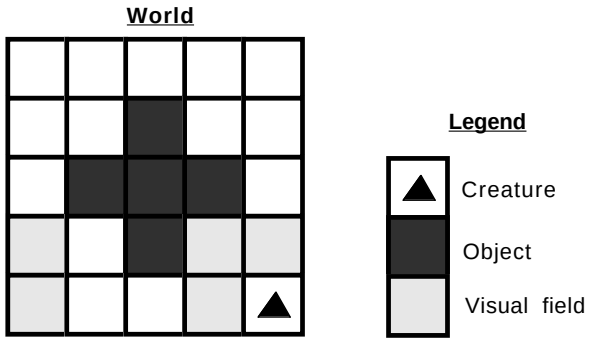

Figure 1: **Object discrimination task.** The object occupies the center of the world and the edges of the world wrap around. A creature is initially placed on a random empty square facing a random direction and it sees the squares in front of it and beside it.

visual image. In order to recognize the object, the creatures first need to evolve a capability to make sense out of their raw sensory information, and to relate it to their own actions (movements). We will call this task the "sensory-motor grounding problem".

2 The object discrimination task

In the experiments described below, creatures are evolved in the task of discriminating between a "good" and a "bad" object in an artificial world. An unprocessed "digitized" representation of the visual field is given as input, and the creatures have to evolve high-level search and recognition strategies.

The basic test scenario involves placing a creature and an object together in the simulated world and allowing the creature to explore the world (figure 1). The world is a toroidal five-by-five grid. The edges of the grid wrap around, i.e. the square immediately to the right of the rightmost square on the grid is the leftmost square on the same row, and similarly in the vertical direction. The objects vary in size and shape but generally occupy five to nine squares and are placed in the center of the grid. A creature occupies one square and can "see" five squares around it: the square directly in front of it, directly to its left and right and diagonally to its front left and right.

The creature can perform the following actions:

1. `Turn left`
2. `Turn right`
3. `Move forward`
4. `Do nothing`

The fourth option is included so that the creature can change the internal state of its neural net without changing its position (Jefferson et al., 1991).

The creature is initially placed on a random square (one not occupied by the object) and faces a random direction. The creature is given a lifespan of 35 cycles, where each cycle consists of:

1. Evaluating the creature's neural network given the network's current state and the creatures visual field as input.
2. Adjusting the creature's physical position according to the network's output.

There are two possible outcomes for each creature's lifespan:

1. The creature succeeds if:
   - The object was 'good' and the creature hit it, i.e. moved onto a square occupied by the object **-or-**
   - The object was 'bad' and the creature avoided it for all 35 cycles.
2. The creature fails if:
   - The object was 'good' and the creature did not hit it in 35 cycles **-or-**
   - The object was 'bad' and the creature hit it.

If the creature succeeds, it is given a new lifespan and the test is repeated with a new object. This process continues until the creature fails, at which point the creature's fitness level is determined by adding up the number of consecutive tests that the creature has successfully completed. After each creature has been tested and scored in this manner, the population of creatures is evolved using genetic algorithms (section 4). If a creature evolves that performs correctly for sufficiently many runs, it is said to have perfected the task and the evolution terminates.

3 Genetic representation of neural networks

3.1 Motivation

Genetic algorithms require that a creature's neural network be represented in a chromosome, that is, as a homogeneous string of e.g. integer values. It is these strings that the algorithm manipulates in order to improve the fitness of the population.

Node definition:

<start> <key> <initial_value> <k_1> <w_1> <k_2> <w_2> .... <k_n> <w_n> <end>

where:

`start`	- Start marker.
`key`	- Node identification value used in the connections (see below).
`initial_value`	- Output value of this node prior to first evaluation.
<k_i> <w_i>	- Input connection specifications: - k_i specifies the source of the connection. The node whose key value (see above) is closest to this value is taken as the source. - w_i specifies the weight of the connection.
`<end>`	- End marker.

Figure 2: **Structure of a node definition.** Every node in the network is defined by this sequence, which may appear anywhere on the chromosome.

Previous approaches to genetic representation of neural networks have restricted the number of neurons (nodes) or the connectivity of the network (Dress, 1987; Mjolsness et al., 1988; Hancock, 1990; Collins and Jefferson, 1991; Jefferson et al., 1991; Werner and Dyer, 1991). These encoding schemes simplify the work of the genetic algorithm by reducing the number of parameters that must be optimized. However, any constraints placed on the network structure can result in a network that is either inefficient or incapable of performing the desired task. In order to maximize chances of evolving an optimal network, the search space of the genetic algorithm should be as large as possible. The marker-based encoding scheme, proposed below, allows *every* aspect of the network architecture to be controlled by evolution.

3.2 Marker-based encoding

The key feature of the marker-based encoding scheme is to use marker values to section off the working areas of the genetic material. This approach is inspired by the structure of biological DNA.

In DNA, the genetic information is contained in a sequence of nucleotide triplets. These triplets specify strings of amino-acids that make up a protein. Typically, a single strand of DNA specifies multiple proteins in this fashion. To separate the specification of different proteins, certain nucleotide triplets serve as markers rather than being part of amino-acid definitions. Each protein specification consists of a start marker and an end marker with the triplets in between defining the composition of the protein (Rothwell, 1988).

In a similar manner, we use markers to separate individual node definitions. Each definition contains all information that the node needs in order to carry out its computations. Instead of encoding the network structure in global terms such as number of layers or degree of connectivity, we let these features emerge from individual node definitions. The number of nodes in the network depends solely on the number of start/end marker pairs found in the chromosome.

Each node definition contains the identification of the node, its initial activation value, and a list specifying its input sources and weights (figure 2). The neuron may receive input from other nodes, from the sensors, and from its own output. The number of connections is determined by the distance between the start and end markers, allowing each node to use as many or as few inputs as it requires.

The chromosome in our experiments is a list of 800 integers ranging between -100 and 100. The start and end markers are identified by their absolute values: if this value MOD 15 equals 1, the integer is a start marker; if the value MOD 15 equals 2, the integer is an end marker. The interpretation of other chromosome integers depends on their position relative to the start and end markers. This scheme gives each value approximately 13% chance of being some type of a marker. By making the MOD constant larger or smaller, the density of node definitions can be adjusted.

The chromosome is implemented as a linear list but is treated as a continuous circular entity, that is, a node definition may begin near the end of the list and continue at the beginning of the list (figure 3). Node definitions are not allowed to overlap. If a start marker is encountered in the middle of a node definition it is treated like any other value (as a weight, or key, etc.). A node definition that 'wraps around' to the start of the chromosome is terminated by the start marker of the first node definition if an end marker has not yet

Figure 3: **Sample marker-based chromosome-to-neural-network mapping.** The chromosome contains two node definitions. The 'a' node definition begins at the fifth chromosome position and ends at the 12th. The 'b' definition begins at the 17th position and wraps around to the beginning of the chromosome, finally terminating at the second position.

been encountered. Integers between an end marker and a start marker are considered inactive, i.e. they do not take part in defining any part of the network.

3.3 Evaluating the network

During evaluation (execution) of the network, each node computes the weighted sum of its input and thresholds at zero:

$$o_k = F\left(\Sigma_{i=1}^{n} w_i o_{k_i}\right) \tag{1}$$

where o_k is the output of node identified by key k and $F(x)$ is a binary threshold function at 0.

The nodes are evaluated in the order in which they are read off the chromosome. Before each node's initial evaluation, its output value is set to its initial value (specified in the node definition) MOD 2. There are five binary inputs to the network, each representing whether a square within the creature's field of vision is empty or occupied. These inputs are referenced in the node's input connection list by mapping them to a certain range of key values. We define any connection key whose absolute value is less than 20 to be a reference to an input. The actual input is then identified as the key's absolute value MOD 5. This gives each connection a 20% chance of coming from a sensory input. The output of the network, specifying one of the four possible actions, is taken as the output values of the last two nodes read off of the chromosome. If a network contains fewer than two nodes it is not evaluated.

3.4 Properties

Most genetic representations of neural networks fix each position on the chromosome to a particular net characteristic (Dress, 1987; Mjolsness et al., 1988; Hancock, 1990; Collins and Jefferson, 1991; Jefferson et al., 1991; Werner and Dyer, 1991). Marker-based representation allows each position to be used in the way that produces the maximum benefit for the creature. In some cases many nodes with a small number of connections may be ideal, in other cases fewer nodes with a larger number of connections may be required. If a small, efficient network topology is desired, network size or execution speed can be incorporated into the fitness function.

An interesting phenomenom which consistently emerges when using this encoding scheme is the occurrence of nodes with no input connections, or 'constant nodes'. Since they receive no input, the output value of these nodes will never deviate from the initial value. Other nodes can reference these nodes as inputs, effectively establishing a non-zero threshold for that node. In other words, the constant nodes act as bias nodes, which are commonly used in place of threshold parameters in e.g. backpropagation learning (Rumelhart et al., 1986).

4 The Genetic Algorithm

4.1 Overall Strategy

The genetic algorithm used in our experiments is based on standard techniques (Goldberg, 1988). After all

One Iteration of GA for a Population of 50
1. Combine the best 15 chromosomes to form 30 new chromosomes (pairing each with another chromosome whose score is at least as good).
2. Replace the worst 30 chromosomes with the new offspring.
3. Mutate chromosomes (except the top scorer's).
4. Sort chromosomes by score.

Table 1: **Summary of the Genetic Algorithm.**

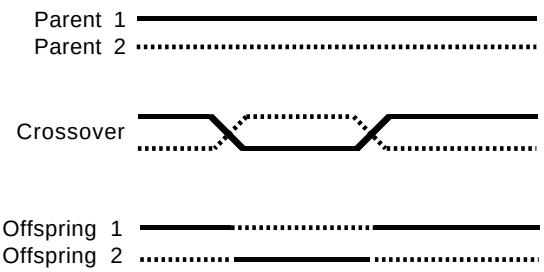

Figure 4: **Two-point crossover operation.** Two offspring are generated by recombining the genetic material of the parents.

creatures have been tested and assigned a score, a mate is assigned to each elite creature (elite = the highest scoring 30% of the population) by randomly selecting another creature whose score is *at least as good*. This strategy ensures that the best creature in the entire population is always duplicated in the offspring, and the higher-scoring creatures have better chances of propagating their genetic information. The new creatures replace the worst creatures in the population, while the original elite remain in the population. Finally, every creature except the top-scorer undergoes mutation. A population of 50 creatures was used in the experiments. The genetic algorithm is summarized in table 1.

It is common in genetic algorithm experiments to allow the genetic operators to manipulate the chromosome at the bit level (Goldberg, 1988). Our scheme, however, treats the integer as the basic genetic unit. This approach was adopted mainly to reduce processing overhead, thereby allowing larger chromosomes.

4.2 Crossover

The standard two-point crossover approach is used to generate offspring (Goldberg, 1988). The parent chromosomes are partitioned at two randomly chosen points (figure 4). Since the chromosome is treated as a circular entity, this effectively breaks the chromosome into two continuous chunks. An offspring chromosome is constructed by taking one chunk from each parent. This way two new offspring are generated at each crossover operation. The idea behind crossover is that different beneficial traits, previously encoded on different parent chromosomes, will have a chance of ending up on the same offspring chromosome, resulting in an offspring superior to either parent.

The marker-based representation scheme interacts with the crossover process in an interesting way. Since much of the space in the chromosome is unused (the space between the end of one node definition and the start of the next), the crossover points have a chance of falling in the unused sections, in which case the node definitions are transferred to the offspring without being disrupted. Additionally, because the connections are specified with key values (rather than e.g. with positions on the chromosome), a whole group of nodes can be passed from a parent to an offspring with their connections intact, perhaps preserving a useful trait. On the other hand, it is also possible to break node definitions during crossover, and some keys may take on different meanings in the new context. The likelihood of preservation vs. variation can be adjusted by changing the density of the start and end markers (section 3.2).

If the crossover operation was performed at the bit level, some chromosome values could change if a split broke up the bits of a chromosome integer. This would introduce new variability in the gene pool. However, much of the same variability can also be achieved through mutation.

4.3 Mutation

The standard mutation operation works by flipping a bit in a chromosome (Goldberg, 1988). To simulate the natural variability of this scheme at the integer level, the following approach is used: individual integer elements are mutated by randomly selecting a delta value within the legal range and adding the delta to the existing integer value. If the new value falls outside of the allowable range, it will "wrap around". Each element in the chromosome has a 0.4% change of undergoing mutation during each evolutionary cycle.

Three types of changes can occur, depending on where the mutation takes place. Most of the mutations occur in connection weights, and result in minor, smooth changes in the creature's behaviour. Mutation in a connection source is as frequent but has no

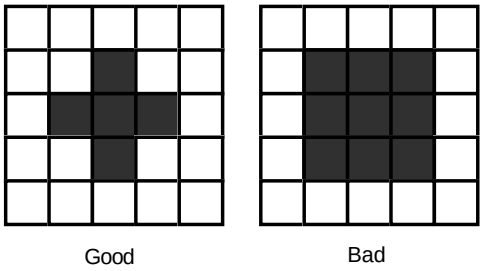

Figure 5: **Data set 1.** The objects can always be identified before hitting them.

Seed	Generations	Nodes	Connects/Node
1	304	15	11.20
2	4	11	9.00
3	77	16	7.56
4	7	15	10.07
5	296	15	9.73

Table 2: **Test results for data set 1.** Listed are: random number generator seed, number of nodes in the perfect creature's neural network, and average number of connections per node for the same network.

effect on behaviour until the change is large enough so that a different source is identified. This change is discrete, and may result in more significant changes in functionality. The third type of mutation occurs in start and stop markers. These are relatively rare but result in very significant changes. Nodes may be created and deleted, or large groups of connections may be created or deleted. In other words, mutation in the marker-based genetic representation can account for both smooth and discrete evolutionary steps.

5 The Experiments

In each experiment, a population of creatures is evolved until a creature completely mastering the task emerges. This creature must be able to always seek out the good object and avoid the bad object from an arbitrary starting position and orientation. The difficulty of this task depends on the choice of objects. Three different data sets (i.e. object selections) are used. The initial chromosome values for all creatures are generated randomly. Each data set was tested five times using five different random number generator seeds.

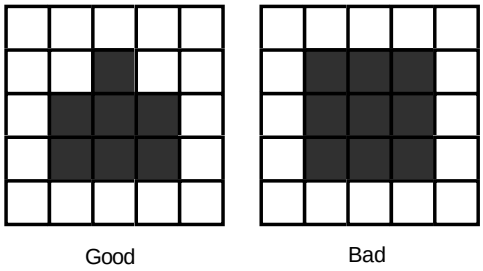

Figure 6: **Data set 2.** Several ambiguous views exist. Recognizing the good object requires traversing around it looking for a characteristic view.

Seed	Generations	Nodes	Connects/Node
1	116	17	6.94
2	211	15	10.80
3	36	16	9.25
4	414	14	8.29
5	15	9	20.11

Table 3: **Test results for data set 2.**

5.1 Data set 1: Straightforward pattern recognition

The first data set (figure 5) is relatively simple because each object can be easily identified. The creature only needs to go directly towards the object. When the creature sees an object square directly in front of it, it can always tell whether the object is good or bad. For example, if the creature sees "■■■" , it can simply stop, since this pattern cannot be found in the good object. Likewise, if the creature sees a "■" directly in front of it and a "■" directly to one side it should know to move forward, since this pattern only appears in the good object.

The creatures learned to master this task in an average of 138 generations (iterations of the genetic algorithm, table 2). Two types of behaviour evolved. In one case the creatures would search for a distinguishing view of the object and once found, either go into a wait sequence if the object was bad, or hit the object if it was good. The other type of behaviour (developed in one of the five experiments) had the creature *circle* the bad object indefinitely and quickly hit the good object.

5.2 Data set two: Surveying required

The objects in the second data set (figure 6) present a greater challenge as there is no longer any unique view that identifies the bad object. Every view found with

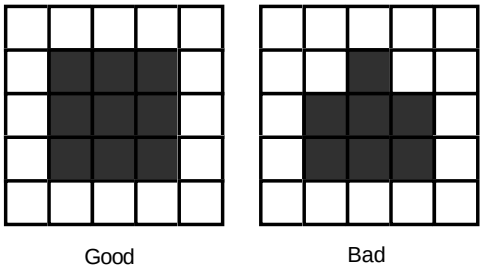

Figure 7: **Data set 3.** Objects are reversed from data set two. Hitting the good object requires using information from at least two different views.

the bad object is also possible with the good object. The creature must survey different parts of the object and hit it only if it sees the unique characteristics of the good object.

The average number of generations taken to evolve a perfect creature in this test was 158 (table 3). The behaviour of all these creatures followed the same basic pattern: circle the bad object indefinitely and hit the object if a missing corner is seen. Note that this is identical to the behaviour of one of the creatures in the first data set. In fact when this creature was run on data set two, it achieved a perfect score.

An interesting phenomenom was observed in the circling creatures. They seem to have evolved an initialization phase, where they will first usually move forward one square and rotate 360 degrees before they start circling. Occasionally, though, they will forgo this phase and start circling immediately. It seems that for certain initial positions and orientations, they cannot determine exactly where they are from the visual field input and must look around to get oriented. Other starting situations, however, leave no doubt as they have a unique visual field associated with them.

5.3 Data set 3: Memory required

The third data set consists of the same two objects as the second data set. The difficulty of the problem is increased, however, by making the object with the distinguishing characteristic bad. There is no *single* view that indicates that an object should be hit. Now, in order to act correctly in the presence of the good object, the creature must survey at least two sides of the object, determine that it does not have the undesirable characteristic and then hit it.

This task is significantly more complex than the previous ones, because it requires that the creature must internally "remember" what it has seen previously and use that data along with the current input to determine the appropriate action. In other words, a simple reflex response to its immediate input is not sufficient. It has to develop a finite-state strategy.

Seed	Generations	Nodes	Connects/Node
1	820	13	10.30
2	1302	13	5.92
3	1643	16	5.94
4	832	14	10.93
5	978	11	13.00

Table 4: **Test results for data set 3.**

The average number of generations taken to evolve a perfect creature for this data set was 1115 (table 4). The behaviour of the five creatures was very similar. When encountering a good object, they would survey two sides and if both were found to be flat, then hit the object. If a bad object was recognized, the creatures would go into a waiting pattern of a few cyclic moves.

6 Discussion

Given the creatures' high-level behaviour, can we infer that they have solved the sensory-motor grounding problem? Let's consider their task in more detail. Every time a creature moves, certain predictable changes occur. For example, if the creature sees a "■" directly in front of it and executes a right turn, the creature will always see the "■" to its left. In the case of right turns the left view will never contain any new information. Consider now the effect of a *left* turn. The left view now reveals previously unseen information that may be important in determining the identity of an object. The same visual input which was previously redundant, now contains valuable data. For each movement that the creature makes, there is a predictable change that will occur in the creature's visual field. Without the ability to relate a particular movement to a predictable change, the visual input would make no sense. In order to perform complicated recognition tasks, the creature must first develop a functional understanding of what effects its own movements have on its visual inputs.

What would happen if instead of giving a creature a five-element visual field, a (more sophisticated) creature was equipped with a 10,000 element retina? Could it evolve the capacity to use this data in a meaningful way? Processing large amounts of 'real world' data such as visual images using traditional AI techniques has proven to be somewhat problematic. Perhaps, noting the success achieved through natural evolution of biological neural networks, using a neuro-

evolution approach would prove promising in this area.

The behaviour exhibited by the creatures in the experiments is at the complexity of finite-state automata. A creature takes an action based on its current visual input and its internal state, and updates its internal state. Similar behaviour also evolves in other neuro-evolution systems such as Genesys (Jefferson et al., 1991) and AntFarm (Collins and Jefferson, 1991). An interesting direction for future work is to determine how far this approach can be carried on. ANNs even with finite number of nodes are Turing-equivalent (Siegelman and Sontag, 1991). Would it be possible to evolve creatures that recognize context-free or context-sensitive languages?

But how far can the behavioural strategies be pushed? Could the creatures learn to hunt, or to play chess if it was necessary for survival? Unfortunately, the computational complexity of evolving such creatures grows very fast with the complexity of the task. The free-form genetic encoding scheme introduced in this paper is well-suited for more complicated applications since it can develop nets of arbitrary complexity and capacity. Exactly how efficient it is in highly complex tasks remains to be seen.

7 Conclusion

We have shown that marker-based genetic encoding of neural networks can evolve high-level behaviour similar to that of finite-state automata. In addition, the networks evolve an understanding of their sensory inputs and actions, i.e. they develop an internal world model. The main direction for future research is to see exactly how far this approach can take us in developing sophisticated visual processing capabilities and behaviour in highly complex tasks.

References

Barto, A. G., Sutton, R. S., and Anderson, C. W. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:834–846.

Collins, R. J. and Jefferson, D. R. 1991. AntFarm: Towards simulated evolution. In Farmer, J. D., Langton, C., Rasmussen, S., and Taylor, C., editors, *Artificial Life II*. Reading, MA: Addison-Wesley.

Dress, W. B. 1987. Darwinian optimization of synthetic neural systems. In *Proceedings of the IEEE First International Conference on Neural Networks*. Piscataway, NJ: IEEE.

Goldberg, D. E. 1988. *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley.

Hancock, P. J. B. 1990. GANNET: Design of a neural net for face recognition by genetic algorithm. Unpublished Research Report.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Ann Arbor, MI: University of Michigan Press.

Jefferson, D., Collins, R., Cooper, C., Dyer, M., Flowers, M., Korf, R., Taylor, C., and Wang, A. 1991. Evolution as a theme in artificial life: The genesys/tracker system. In Farmer, J. D., Langton, C., Rasmussen, S., and Taylor, C., editors, *Artificial Life II*. Reading, MA: Addison-Wesley.

McClelland, J. L., Rumelhart, D. E., and Hinton, G. E. 1986. The appeal of parallel distributed processing. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*. Cambridge, MA: MIT Press.

Mjolsness, E., Sharp, D. H., and Alpert, B. K. 1988. Scaling, machine learning and genetic neural nets. Technical Report YALEU/DCS/TR-613: Department of Computer Science, Yale University.

Rothwell, N. V. 1988. *Understanding Genetics*. New York: Oxford University Press, Inc. Fourth edition.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. 1986. Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*. Cambridge, MA: MIT Press.

Siegelman, H. and Sontag, E. D. 1991. Neural nets are universal computing devices. Technical Report SYCON-91-08: Rutgers Center for Systems and Control, Rutgers University.

Werner, G. M. and Dyer, M. G. 1991. Evolution of communication in artificial organisms. In Farmer, J. D., Langton, C., Rasmussen, S., and Taylor, C., editors, *Artificial Life II*. Reading, MA: Addison-Wesley.