EVOLUTIONARY NEURAL NETWORKS

by

Kenneth D. Landry

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science and Applications

APPROVED:

D. P. Miller, Chairman

J. W. Roach

D. G. Kafura

19 February 1988

Blacksburg, Virginia

EVOLUTIONARY NEURAL NETWORKS

Kenneth D. Landry

Committee Chairman: David P. Miller

Computer Science

(Abstract)

To create neural networks that work, one needs to specify a structure and the interconnection weights between each pair of connected computing elements. The structure of a network can be selected by the designer depending on the application, although the selection of interconnection weights is a much larger problem. Algorithms have been developed to alter the weights slightly in order to produce the desired results. Learning algorithms such as Hebb's rule, the Delta rule and error propagation have been used, with success, to learn the appropriate weights. The major objection to this class of algorithms is that one cannot specify what is not desired in the network in addition to what is desired. An alternate method to learning the correct interconnection weights is to evolve a network in an environment that rewards "good" behavior and punishes "bad" behavior. This technique allows interesting networks to appear which otherwise may not be discovered by other methods of learning. In order to teach a network the correct weights, this approach simply needs a direction where an acceptable solution can be obtained rather than a complete answer to the problem.

Acknowledgements

I would like to thank my major professor, David Miller, for all his encouragement and direction throughout my masters program.

I would also like to thank . and for their invaluable comments and support. Thanks also go to for his Mac Plus and for his patience with me while I was writing this thesis.

Lastly I would like to thank my fiancee, for being there for me when I needed her.

Table of Contents

List of Figures

Chapter 1

1.1 Introduction to neural networks

Neural networks are a form of parallel distributed processing. They consist of a number of computing elements (CE) that perform local computation. These computing elements are connected to a number of other elements that in turn are connected to others and so forth. Generally, the connections are unidirectional. The information that is passed along the connection is an activation or energy value which is scaled according to the strength of the connection. This strength is called the connection weight. The input to a computing element is a function of the input from those elements that are connected to it. Most often the input function is the weighted sum of these inputs. Figure 1 shows a portion of a neural network, showing the connections of a computing element from other elements. It also shows the computing elements that it is connected to.

Some computing elements have connections from an outside environment. These computing elements are called input units. Other CEs have connections to an outside environment. These are called output units.

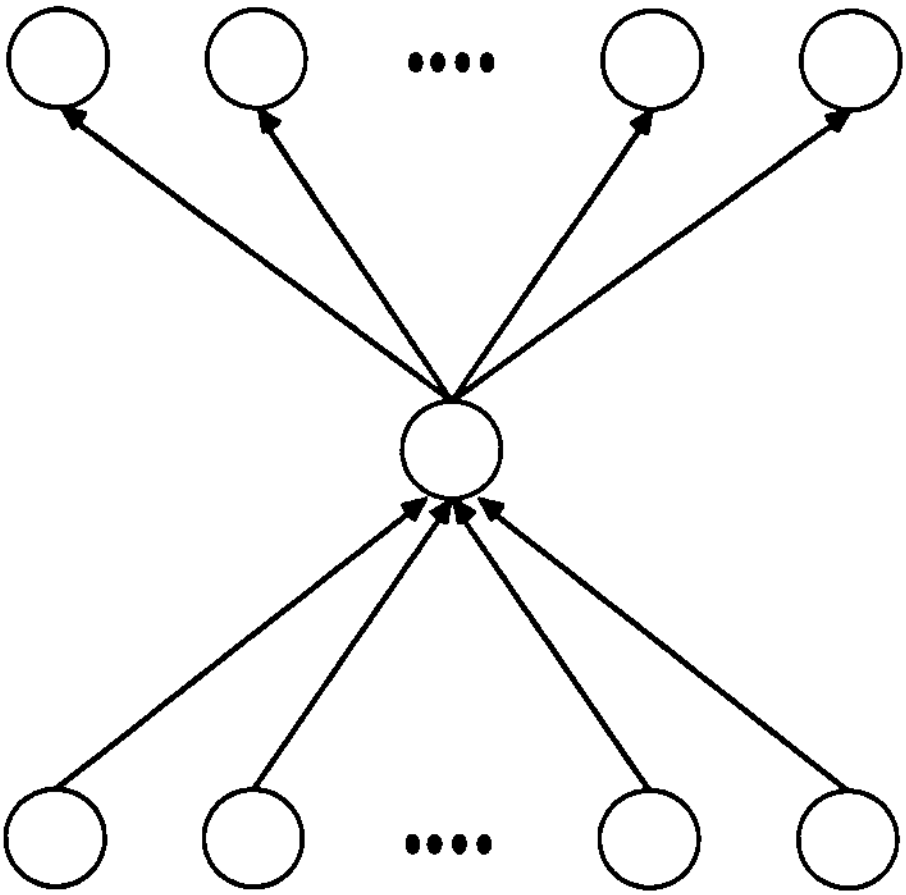

Figure 1 Computing Element Connections

The rest of the elements are referred to as hidden units. A neural network can be considered to be a black box (Figure 2) which takes some input pattern distributed along the input units and produces an output pattern which is distributed along the output units. A neural network's task could then be considered as one that associates output patterns with specific input patterns.

There are two major tasks associated with creating a neural network: the first is deciding on the network organization and the second is deciding on the appropriate set of interconnection weights.

1.1.1 Selecting a network organization

A decision that must be made when designing a neural network is what its structure should be. The structural organization has a definite effect on the computing power of the network. For example, if only two layers of linear threshold computing elements are used with connections made from one layer to the other, then certain input/output pattern mappings cannot be made [Rummelhart86]. The classic example is the XOR function which cannot be modeled in such a network because similar input patterns do not map to similar output patterns (which is a restriction of many network structures). Typically, the organization of a network has been a layered structure where activation flows from one layer to another starting at the input unit layer, continuing through the hidden unit layers and finally reaching the output unit layer.

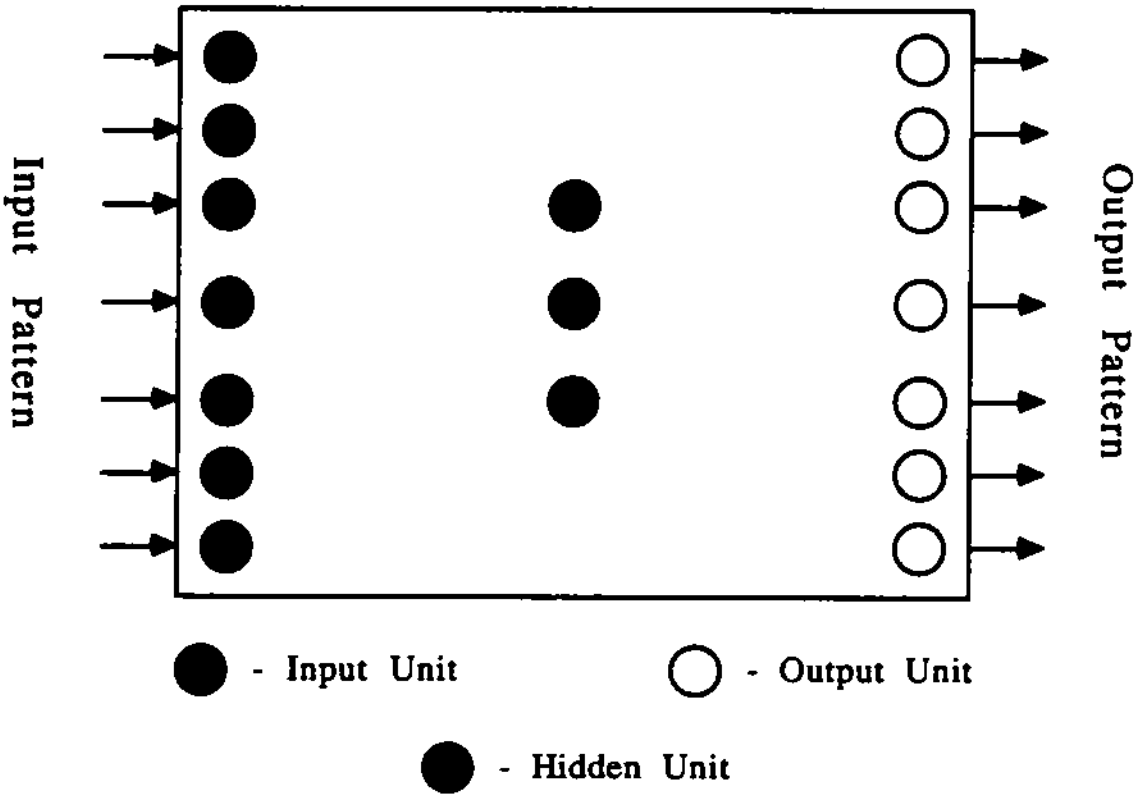

Figure 2 Neural Network Black Box

While the physical structure of the network is an important factor affecting its power, the type of computing element also has an effect on what the network can compute. Computing elements range from linear units (the output value of the unit is equal to a constant times its current activation) to linear threshold units (the output is some minimum until the activation is above a threshold where upon it is set to a maximum) to Sigma-Pi units. The type of computing element used is important is because certain learning rules only work for a particular type of unit and only guarantee a certain level of computing power [Rumelhart86].

It may seem that it would be hard to choose the appropriate structure and type of computing units but, as we will see in Chapter 3, certain organizations are more appropriate for particular classes of applications. Also, some types of organizations may be made to mimic other organizations simply by setting some interconnection strengths equal to zero. The difficult task in designing a network is knowing what the interconnection weights should be in order to model a particular set of input/output pattern pairs.

1.1.2 Selecting a set of connection weights

The network organization can be designed by hand but what about the interconnection weights? This is a slightly more difficult problem because distributed weights do not offer a clear idea of what information is stored in

the network. It we take a particular network and change the set of weights slightly, then the mapping of input to output patterns changes. What we have changed is the information collectively stored in the network. Deciding on the correct set of weights is a difficult task since changing a single weight may possibly change a large number of mappings. The task then is to specify the correct weights in order to achieve the desired mappings. But if the network is even of moderate size then trying to specify the correct weights manually becomes almost impossible.

The solution to this problem is to use an algorithm that will learn the correct weights through a series of tests. A number of learning algorithms have been suggested and some show promise for certain areas of computation, although all have deficiencies that make them unable to model common problems. An example of this type of problem is given below.

1.1.3 Hazard avoidance for robots

A domain in which the current learning rule deficiencies are seen is the domain where a robot attempts to avoid hazardous areas. The function of the robot's network is to decide where to move the robot depending on what it sees around itself. A domain such as this requires that the robot know that certain movements may be dangerous for certain sensory input. For example, Figure 3 shows an environment in which a robot must move around a nuclear reactor and pick up trash. In this case, it is not feasible to specify a particular

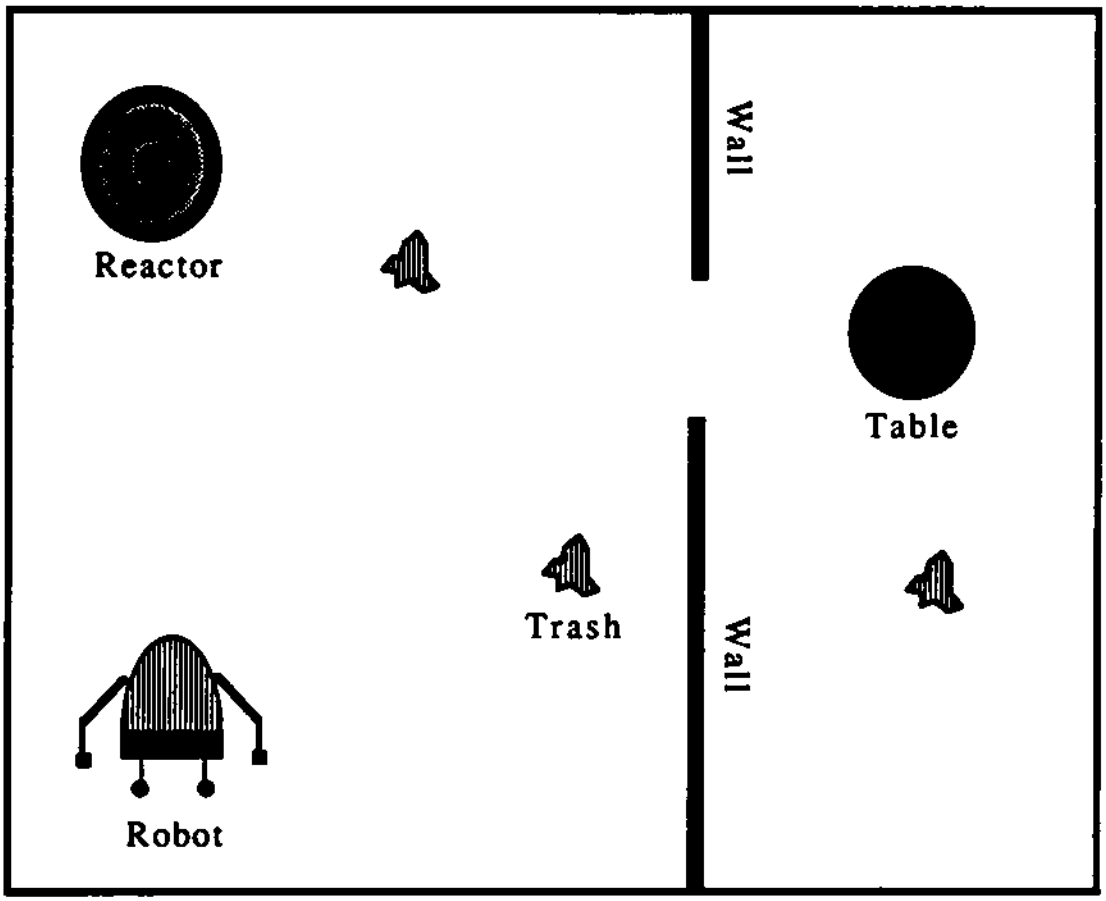

Figure 3 Hazard Avoidance for Robots

movement for each sensory input. It is better to specify what movements are detrimental for certain sensory inputs. For example, the robot could have a network which allows it to move around the room in no particular order except that it avoids walls, tables, nuclear reactors, chairs etc. This example will be used as the domain for the discussion of the current learning rules in this section.

1.2 Problem statement

Most methods for allowing a neural network to learn appropriate connection weights require that the desired output of the network be known during the teaching process. It does not allow you to specify those outputs you do not want instead of those that you do want. This is a significant drawback if the desired output is not known or if it is easier to say what the output should not be. A new method for leaning the appropriate connection weights called the evolutionary learning algorithm will be presented which takes care of these problems of previous methods.

1.3 Previous solutions

A number of algorithms have been proposed which allow a network to learn an appropriate set of connection weights (if a set does exist for the network). Most of these learning algorithms are restricted to a small number

of network organizations and some will not find a solution even in one exists. Despite the many differences between the algorithms, one feature is common among most. This similarity is the need to have a teaching input to the network during the learning process. This teaching input is generally the desired output pattern for the given input pattern. Using the desired output pattern and sometimes the actual output pattern, the connection strengths can be changed gradually, in order to get the desired mapping after the learning phase is over.

1.3.1 Use of teaching inputs in learning rules

The Widrow-Hoff rule [Sutton81] (also known as the Delta rule [Rumelhart86]) and the Generalized Delta rule [Rumelhart86] are two learning rules for networks which require teaching inputs in order to compute the change (delta) in the connection strengths. Because these rules need to have a teaching input (the desired output), the change in connection strength either cannot be computed locally without some external input or they cannot be computed in parallel. These rules will be discussed in more detail in Chapter 3.

1.3.2 Deficiencies of previous methods

The problem in having to use a teaching input is that we have to specify the exact mapping that we want the network to learn. There may be situations

in which the possible mappings are so large that specifying all mappings becomes a tremendous task. Also, it is not possible with these learning rules to state what mappings are undesirable in the network as opposed to what is desirable. In the case of a robot avoiding hazardous areas while moving around the room collecting trash, we would like to specify that it should avoid certain objects in the room (disallow certain sensory input to motor output mappings) instead of specifying every move the robot should make in the room.

1.4 Evolutionary neural networks

The deficiencies discussed above prevent some common problems from being modeled by a neural network. Because of this deficiency with the current learning methods, an algorithm will be presented for using evolutionary pressures and natural selection to produce a network which exhibits desirable mappings (or lack of undesirable mappings) for a particular domain.

The evolutionary learning algorithm works with a group of neural networks of the same structure (although this could be relaxed to a certain extent) each network modelling the same function (in the case of hazard avoidance with robots, the function is sensory input to motor output). The networks are presented with a number of input patterns and the nearness of

the output patterns to the desired answer is evaluated. The actual answer does not have to be known, but only a relative idea as to how close the output patterns are to the answer. It is also possible to specify, in nearness to the answer, when network makes an undesireable response (such as running into the nuclear reactor). This evaluation of network responses is used to select the candidate networks for the creation of new and slightly different ones. This evolutionary process over a period of time, will converge towards the correct solution or a network that is sufficient to do the job. This technique will learn the correct set of weights without having to specify all the possible mappings from input to output. It also allows one to specify what mappings are undesirable instead of what are desired. Chapter 4 will discuss in detail the process of the evolutionary learning algorithm.

1.5 Outline of Thesis

Chapter 2 discusses the domain used in this thesis to provide an environment for evolutionary neural networks. The domain is called FIREFLY WORLD where fireflies, having neural networks, reproduce to form new and hopefully better neural networks.

Chapter 3 presents a discussion of neural networks. The basic structures will be discussed along with a number of learning algorithms that have been developed to teach a neural network some task or function. The

problems associated with these leaning rules will be discussed in detail.

Chapter 4 discusses evolution as it relates to learning in neural networks. The evolutionary learning algorithm will be discussed in detail as will other work on the topic of evolutionary programming.

Chapter 5 presents an implementation of FIREFLY WORLD. A description of the environment, the cycle of a simulation, the object-oriented programming approach taken, and some simulation results will be presented.

Chapter 6 will present some concluding remarks and discuss future work.

Chapter 2

The Firefly World Domain

The domain for testing evolutionary pressures on neural networks as a learning mechanism will be discussed in this chapter. Firefly World is a domain in which simple functions can be modeled in a neural network and "taught" the right connection weights without explicitly stating the mapping of input values to output values. This domain also provides an easy environment for reproduction and evolution. Through evolution of a group of neural networks, each trying to model the same function, the desired network will eventually be produced.

2.1 Motivation

In order to test the method of evolving neural networks with the correct weight configuration, we need some type of environment where multiple networks can be tested. These networks, using the same structure but with different weights, will be tested with many different input patterns from the environment. After a number of network tests, the nearness of the

networks overall response to the actual network solution is evaluated. Those networks with the best overall response will be the candidates for generation of new networks. Consequently, a group of networks evolve toward a solution. This environment is able to specify, via the evaluation function, what outputs it does not want associated with certain input patterns.

An obvious domain for an environment such as this is one found in nature. The ability of a species to adapt to its environment over time is an example of an environment where good behavior is rewarded by staying alive to reproduce and hopefully create better adapted members. It seems reasonable then to use a domain which requires a function of moderate complexity to be performed such as mapping sensory input to motor output. The agent which grades network responses in such an environment would be a predator of the species so that good movement responses of the network based on sensory perception will be rewarded by staying alive. FireFly World is such an environment. The desired network would be one which never associates any input pattern with an output pattern which would force contact with the predator.

2.2 FireFly World Environment

The environment described in this section is a world which encourages favorable behavior and discourages other behavior. It allows generations of

neural networks to be produced and tested to determine the ability to survive in such an environment. The following sections discuss the structure of the world and then describe the agents which function in this environment.

2.2.1 Overview of Environment

The terrain of FireFly World is a grid of fixed size cells as seen in Figure 1. FireFly World has the shape of a torus. The torus shape permits bidirectional wrap around from top to bottom and from left to right. Each discrete cell in this world is a square which allows eight other cells to border it at 45 degree angles. The environment will permit movement in all eight directions from any given cell.

The world is populated by three types of objects: food which is stationary, fireflies and their predator which are both mobile. These three objects are discussed in detail in the following sections.

2.2.2 The FireFly

The firefly is one of the two mobile agents in this world. The function modeled in the firefly is the mapping of sensory input to motor output. The sensory capability is circular so that in any given cell in the world, the firefly is capable of seeing in all eight directions. It can sense every agent that can

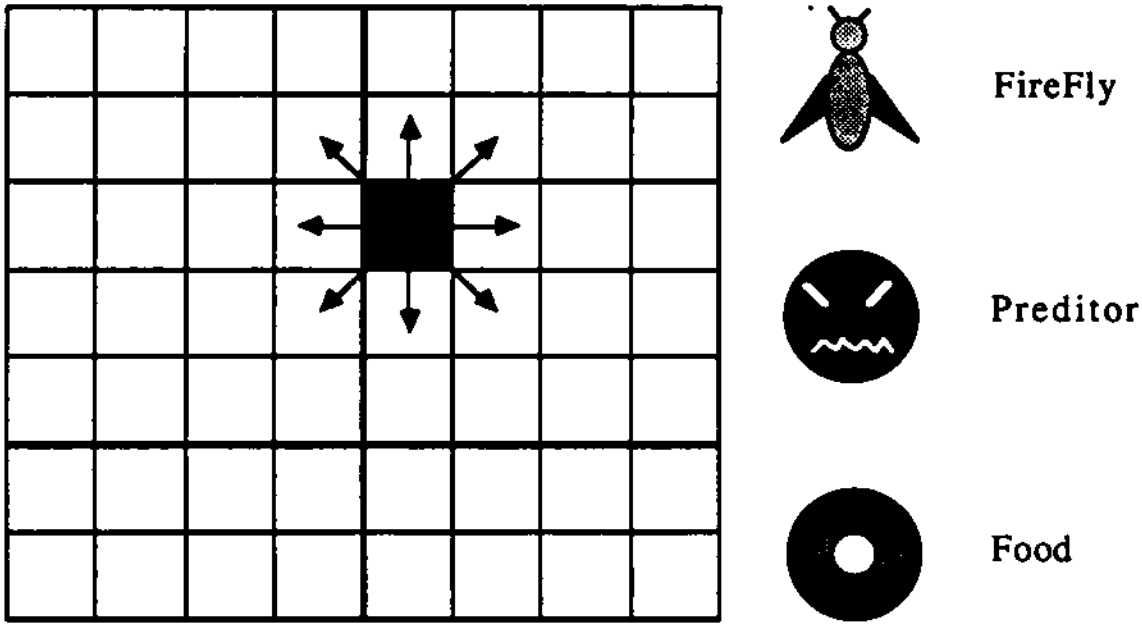

Figure 4 Firefly World and Its Entities

appear in the world - another firefly, food, the predator and an empty cell. The sensory capability can be extended to any number of levels out from the firefly. Giving the firefly more than one level of sight has some drawbacks as well as benefits. It is good in that it can give the firefly better reaction time to objects that it sees but there is no association of the action in response to an object at level 1 to level 2. In other words, a firefly may chase after another firefly if it sees it two levels away but avoid it if it sees it at one level. The firefly also has the ability to move in any of the eight directions surrounding it or, if desired, it may stay stationary.

The sensory/motor mapping function in the firefly is implemented using a neural network. The exact structure of the network will be discussed in Chapter 5 although many different network organizations could be used. There are other structures contained in a firefly but it is the only one which will change through time. Also, there is no structure in the firefly which possesses the knowledge that food is associated with survival and the predator is associated with death. It may happen that fireflies will be produced that are "paranoid" and avoid contact with everything. In which case they will die off without reproducing. Others may seem to be "Kamikaze Fireflies" who seek out the predator for a one time encounter at which time it dies. The desired firefly is one which seeks out food and avoids the predator.

2.2.3 The Predator

The predator in this world does not contain a neural network as does the firefly. It uses a deterministic algorithm to seek out fireflies. The predator serves as a sort of benchmark for fireflies, destroying those that do not perform good while not being able to catch those that do. It is necessary to keep the predator's algorithm for movement constant so that it may act as a guideline for natural selection to take place. This guideline, if changed, would change the kind of fireflies which are capable of avoiding the predator. For example, if the predator's algorithm now decided to chase fireflies which are only moving then it would be better to stay stationary when it sees the predator instead of running from it.

The predator is provided with limited vision; he can only see in the direction he is moving and to the left and right at forty five degree angles. His movement is also limited to the three directions in which he can see. Although the goal of the predator is to destroy fireflies, if none are in its sight it is able to move in any unoccupied cell in its sight. The predator is equipped with the same sight range as the firefly in order to put them on equal terms.

2.2.4 Food

The food in this world does not move. Energy is transferred from the food unit to the firefly when one is consumed. This energy will allow the firefly to continue moving and possibly mating for a period of time. If a

firefly does not seek out food, then it will eventually die from lack of energy.

When a unit of food is consumed, another one is not produced immediately. The supply of food will decrease over time until a firefly dies, where upon a number of food units will be produced. This causes competition among fireflies for food in order to stay alive. If no more food exists then those fireflies that have eaten a lot will survive the drought of food.

2.3 Reproduction and Evolution

In order to produce fireflies that perform well in this environment, new generations are created through reproduction. Through the process of reproduction, we hope to produce a firefly which exhibits more desirable characteristics. In this environment, good characteristics of fireflies are the desire to eat food and the ability to avoid contact with the predator. There are a number of ways in which the process of reproduction can proceed. The first and less effective way is to choose a firefly pair at random to mate. This has the effect of allowing evolution to take place for fireflies who can avoid the predator but it does not necessarily select those who eat food. A second method is to use what is referred to as "fast time evolution." This approach will select the "best" fireflies in the world, regardless of their current position, and use these for producing the next generation of fireflies. In other words, global simulation control is needed over the environment to decide which fireflies

should reproduce and which ones should not. The method used in this domain has no global control over reproduction. This method permits fireflies to reproduce if they are attracted to food (at least to some extent) and they avoid their predator, while allowing the decision to reproduce to be made by the candidate fireflies. This imposes another requisite quality on fireflies - the desire to reproduce sexually. In order for evolution to take place, reproduction is necessary and if a firefly is reluctant to reproduce then it of does nothing to further the process of evolution. Asexual reproduction can be useful in a domain such as this if a firefly shows good qualities yet is reluctant to reproduce sexually. Fireflies such as this may still form offspring but using only its network to form the new one.

The process of evolution requires that two conditions be met - natural selection and hereditary variations. Natural selection associates reproduction with attractive qualities of the subjects. Those who do not possess such qualities will not reproduce as much as those who do. In FireFly World, natural selection has been implemented by requiring certain firefly attributes to be present before reproduction can take place. The hereditary variation creates new and possibly better members of a species. This is discussed in detail in Chapter 4.

Chapter 3

Discussion of Neural Networks

In order to adequately describe the evolutionary learning algorithm in Chapter 4, the structures of neural networks and some current learning rules are discussed. All of the learning rules presented have deficiencies that limit their modeling capabilities. The evolutionary learning algorithm discussed in Chapter five overcomes the problems of the present learning rules.

3.1 Appeal of Neural Networks

There are many examples in our world which exhibit distributed processing capabilities - human motor functions, vision and memory just to name a few. Among those named, memory is the most interesting to many but the least understood. It seems reasonable that human memory is not designed like that of a computer. Human memory is content addressable, that is it can be accessed by its contents rather than its location. Conventional computer memory is location addressable.

Content addressable memory is important in that we as humans can remember a certain situation by a partial description. This partial description may activate not just a single memory but a collection of memories. From this collection of memories, a number can be ruled out due to conflicting information and usually just a few memories are left. Further scrutiny of available information may eventually lead to the selection of a single memory that "best fits" the partial description.

Content addressable memory is just one example of the need for parallel distributed processing as opposed to conventional serial processing. The question is now - where do we look to get answers about how to build such a parallel model. The obvious answer is the human brain.

There are many theories about the processes of the brain. Among these is the theory of neural networks. The basic structure of a neural network is the computing element (CE) which is described by:

1. A list of connections from other CE's
2. A collection function - $C_i(inputs)$
3. An activation level - $A_i(t)$
4. An activation function - $F(C_i(inputs), A_i(t))$ for combining inputs to CE and the previous activation to produce the current activation
5. An output function - $O_i(A_i)$

The connection from CE_j to CE_i is the weight (W_{ji}) or strength which is

used in the collection function to determine the total input to CE_i at a given point in time. The collection function (C_i) in most network models is a linear function which sums the product of the output function O_j and the interconnection weight W_{ji}.

$$C_i = \sum_j W_{ji} * O_i$$

The activation level A_i receives its value from the activation function that takes its input from the collection function and the previous level of activation of CE_i.

$$A_{i(t+1)} = F(C_i , A_i) = C_i + A_i(t)$$

In some cases the activation function F will be independent of the activation value at time (t-1) so that F_i will be identically equal to C_i. In other cases the previous activation will be graded so that there is a decay of the activation level over time. It is often desirable to set a range for the activation value. This range may be -1 to +1 or from 0 to 1.

Once the activation level of the CE is known, it is necessary to compute the output value in order to spread activation to other computing elements. This function (O_i) can be written in many different ways. In the simplest form the output O_i is equal to the activation value (the identity function).

Alternatively, this function is a threshold function in which the output has no effect (the value 0) until the activation reaches a certain level. Any activation value above this level will produce the same output value (many times the value 1). Some computing elements have connections to other elements which are called inhibitory links. Figure 5 shows a simple network structure with eight computing elements which are fully connected.

3.2 Previous Work

There are two distinct components to a neural network - the structure and the weights. The organization of a network has a definite effect on its computational power [Rumelhart86]. Rumelhart suggests a number of network hierarchies, among those are bottom-up and interactive. In the bottom-up organization, the propagation of activation occurs from bottom to top. This organization allows a single level of CE's to be connected to the same level and the level immediately above it. The interactive model allows a level to have connections with the same level and one level above and below.

Choosing a network organization is not a difficult problem since a particular task or function could be modeled in a number of different structures, each structure being sufficiently complex to contain a solution. Some network structures subsume others because connections can be made to disappear by making the connection strength zero. The more difficult

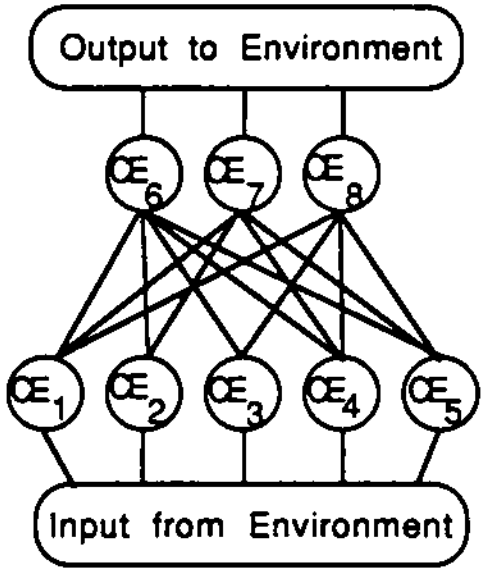

Figure 5 Simple Neural Network

problem is to choose the appropriate interconnection weights between computing elements. This can be a large task if the number of computing units is large. To try to do this manually for a network of even moderate size would be impractical.

If we take a network and change the weights even slightly this produces a different mapping of input patterns to output patterns. What we have done is changed the information that is collectively stored in the weights. In order to give a network a particular set of information (if it is possible to learn it for that structure), the correct weights must be assigned in some manner other than by hand. The method generally used is to have the CE's "learn" the appropriate weights on their connections through a learning algorithm. These learning algorithms have been the topic of much research and a few of them will be discussed shortly.

A certain model is linear if the output function (O_i) returns the value equal to a constant times the activation. The most common type of linear networks, the *linear associator* [Anderson70], is one in which output patterns are associated with input patterns. A linear associator has the capability to associate a number of different input patterns with output patterns only if the input patterns are orthogonal[1] to each other. Depending on the learning rule

---

[1]Two input patterns (or vectors) are orthogonal if the inner product is equal to zero.

used, more patterns may be learned in the same network [Rumelhart86]. A linear system has most of its success centered around pattern associators and has not been used successfully to accomplish many other tasks. For example, it cannot compute the Exclusive OR or Parity functions because the inputs patterns are not orthogonal.

Another limitation of linear networks is that it can be proven that multiple layers of connections can be reduced to a single layer of connections. This means that there is no advantage to having feedback or having multiple layers of connections in a linear system [Rummelhart86].

A more interesting network is one called the linear threshold model. In it the activation function (F_i) is a threshold function which assigns a value of zero to the activation value, A, if the value of the function C (weighted sum of the inputs) is below some level. The activation takes on the value one if the weighted sum of the input is above the threshold level. The output function is simply the activation value, A. The outputs of each computing element may either be excitatory or inhibitory.

A classic example of the type of functions that a linear threshold system can model is the Exclusive OR (XOR) function. This function will return a one if it sees a one and a zero in any order on its two input lines and will return zero otherwise. Figure 6 shows a linear threshold network which computes the XOR function.

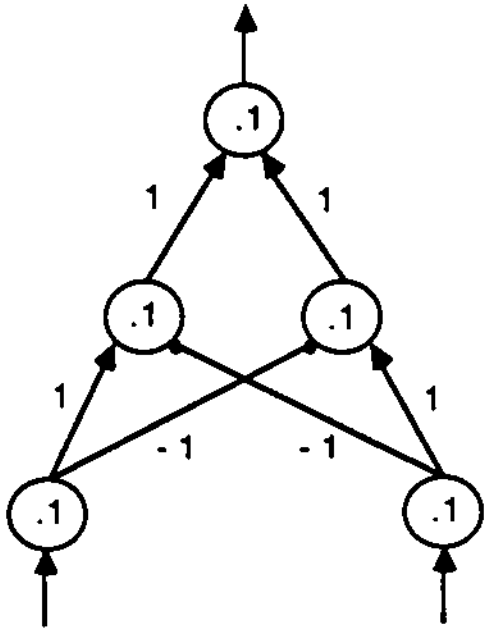

Figure 6 XOR Neural Network

The example network shown is a bottom-up organization with two layers of connections. The first set of CE's at the bottom are the input units; the middle set are called hidden units; the top set of CE's are called output units. If one input unit gets activated then it will send its signal to its hidden unit while at the same time inhibit the other input's hidden unit. The threshold values are shown in the circles and all have the value of 0.1. This structure will allow activation to reach the output unit only if one input unit is activated. The output will be zero if neither input unit is active or if both are active.

The XOR function cannot be computed in a linear threshold network with a single layer of connections. This presents a problem because the learning algorithms for this type of network rely on teaching inputs which are not readily available for networks with more than one layer of connections. There is however a proven learning rule for a single level linear threshold model without feedback. This type of network is called a *Perceptron* [Rosenblatt62]. The perceptron convergence theorem guarantees that if a set of patterns is learnable by a perceptron then the perceptron learning rule will find the appropriate weights for the network in finite time. The perceptron has come under harsh criticism [Minsky69] because its learning capabilities are no greater than a linear associator. Learning rules, however, have been developed for the class of networks whose activation functions are differentiable.

Learning rules can be traced back to Hebb [Hebb49] who offered insight as to how the connection weights in a network should be changed. One of the rules which can be traced back to Hebb is stated as follows:

> strengthen the connection between CE_1 and CE_2 proportional to the product of their activation.

This rule says that if CE_1 is active then strengthen the connections to CE_1 that are active. An important point about this rule is that it can be implemented local to each computing element connection; all that is needed is the activation of the two CE's it connects. Although this learning rule has been used successfully for models such as the linear associator, Hebb's rule has some serious limitations. One limitation is that that the patterns that the associator learns must be independent of each other.

A more sophisticated and powerful learning rule is called the Delta Rule [Rumelhart86] (also known as the Widrow-Hoff rule [Sutton81]). This rule requires a teaching input which is compared to the actual activation of the unit. The rule can be stated as follows:

$$\Delta W_{ji} = \Omega * (T_i(t) - A_i(t)) * O_j(t)$$

The name of the rule is taken from the fact that the difference between the activation of CE_i and its teaching input is used in the equation. Ω is defined as the constant of proportionality which determines the rate of learning.

The perceptron, as mentioned earlier, has been widely criticized because as a single layered linear threshold network, it can only learn a group of patterns that for similar inputs produce similar outputs - unlike that of the XOR function. The XOR network needs a level of hidden units to contain a higher level of representation of the input. This internal representation cannot be contained in a single level of connections without adding more input units. The advantage that a perceptron has over multi-level linear threshold networks is that it has a learning algorithm which can be proved to converge on a solution if the set of patterns is indeed learnable by a perceptron. Multi-level linear threshold networks have no such learning rule to derive their connection strengths. There is a class of networks whose CE's have output functions that are differentiable and there is a learning rule for this class of networks [Rumelhart86].

The generalized delta rule (error propagation) developed by the PDP Research Group [Rumelhart86] provides such a learning capability. The learning rule works on the basis of calculating an error signal. This error signal is the product of the difference between the actual output and the

desired output and of the first derivative of the activation function. In mathematical terms, it appears as follows for input/output pattern pair p :

$$\delta_{pj} = (T_{pj} - O_{pj}) * F_j'(\sum_i (W_{ji} * O_{pj}))$$

This equation is for the output units where T_{pj} is the j^{th} element of the desired output pattern p. The error signal for the hidden units cannot be calculated using a teaching input. It is, however, calculated using the weighted sum of the error signals from the previous level. This equation is as follows :

$$\delta_{pj} = F_j'(\sum_i (W_{ij} * O_{pj})) * \sum_k (\delta_{pk} * W_{jk})$$

The change in weight for the connection between CE_i and CE_j is the product of the learning rate Ω, the error signal for unit j, and the output signal from unit i. In mathematical terms,

$$\Delta_p W_{ij} = \Omega * \delta_{pj} * O_{pi}$$

This rule has been proven successful in simulations where the network had to learn the XOR function with only one hidden unit, but it does so at the cost of having to use a large number of pattern presentations (in the tens of thousands) in order for it to learn a correct set of weight.

All of these rules with the exception of Hebb's rule, require the network to know the "answer" to the problem during the training of the network. That is, a teaching input is needed to change the connection strengths. This restriction can be cumbersome and in some situations simply impossible. There is also the limitation that one cannot specify the mappings that are not desired instead of the ones that are. What would be desirable then would be a learning rule which does not need to know the answer. The evolutionary learning algorithm described later provides such capabilities.

Chapter 4

Discussion of the Evolutionary Learning Algorithm

The major motivation for the evolutionary learning algorithm is that it overcomes the deficiencies of the present learning rules in teaching a neural network an appropriate set of interconnection weights. Another motivation for an evolutionary approach is that neural networks attempt to model functions of the brain; it therefore seems natural to involve the process of evolution in the learning algorithm. This chapter will begin with some background in the area of evolutionary programming as it is related to the evolutionary learning algorithm. The steps of the algorithm will then be discussed in detail and an analysis of this approach will be given.

4.1 Evolutionary Programming

Some work has been done used evolution as a means of selecting structures exhibiting desirable characteristics or traits. Fogel used evolutionary programming [Fogel67] to select finite state machines which could predict the next symbol in an input stream. Fogel used a number of

different types of mutations each with different probabilities of occurring. A cost matrix was used to decide which members of a species of machines best predicted the next symbol with the lowest cost. Depending the goal, the cost matrix assigned an error value for a wrong prediction. Fogel and his colleagues had success with their evolutionary finite state machines in such areas as symbol prediction, diagnosis (measuring an unknown parameter on the basis of other concurrent parameters), pattern classification and competitive goal seeking.

Although Fogel's results were significant, the structure that was evolved was a finite state machine and not a neural network. What is important from Fogel's work is the evolutionary learning process.

Genetic algorithms is another form of evolutionary programming. Evolutionary pressures are used to select which rules in a system of rules offer the best "payoff" for an environment. Genetic Algorithms have been used to simulate adaptive environments [Holland84] where goals change over time. Both genetic algorithms and Fogel's adaptive finite state machines use a form of evolutionary programming which will be used by the evolutionary learning algorithms in the domain of neural networks.

4.2 The Evolutionary Learning Algorithm

The evolutionary learning algorithm is comprised of an environment

which has multiple neural networks, each having the same structure and trying to model the same function. The goal is to produce the effects of evolution by simulating its necessary conditions - natural selection and hereditary variation. Natural selection will select those networks exhibiting desirable qualities as the candidates for the generation of new networks. Hereditary variation takes two forms - genetic and crossover mutation. By mutating the new networks slightly, different input/output mappings will be produced which over a period of time will approach the desired network.

4.2.1 The Environment

The environment must contain one or more ways in which it can evaluate the overall network response for each of the networks. In Firefly World, the evaluation may be in the form of the number of food units eaten by a firefly and the fact that it has avoided the predator. This evaluation of overall network response is used to determine candidate networks to participate in the generation of new networks. It can also be used to determine which networks do not have good responses. These networks should be removed from the environment.

4.2.2 The Algorithm

Once the environment is specified, the network structures defined and

the process of evaluating networks is determined then the process of the evolutionary learning algorithm can be used to produce networks which perform well in the environment. The algorithm is stated below in 7 steps:

1. Present all networks in the environment with input patterns for a period of time.
2. Evaluate the overall response of each network.
3. Select candidate networks for generation of new networks.
4. Combine the candidate networks to form a new network.
5. Mutate the new network to create variant performance.
6. Remove those networks with low overall ratings.
7. Continue steps 1-6 until a network achieves a certain level of response rating.

This algorithm decides how networks are reacting to input stimuli and based on this information creates new networks from those networks that have the best overall rating. These newly generated networks are changed slightly so as to generate variances in the network responses. Those networks not achieving a certain level of acceptable performance will be removed from the environment. Over time, network structures will converge to the area of network solutions where the correct solution exists (whatever that network may be).

Evaluation of the output pattern can occur after each presentation of input patterns or it may be done at the end of all presentations. In Firefly World the evaluation is the number of units of food that gets consumed and the

fact that the firefly still exists. Those fireflies that eat a lot of food and avoid the predator will become candidates for reproduction of new fireflies. In this case, a partial evaluation is done each time a firefly moves. It decides whether the move has made contact with the predator or food. If contact with the predator was made then it will produce a very poor evaluation (low enough so that the firefly is removed from the environment). If it makes contact with food then it gets a favorable evaluation.

Networks are selected as candidates for the generation of new networks based upon their overall performance evaluation. This selection may take place at certain predetermined intervals of time or it may be a function of some aspect of the environment. In Firefly World, fireflies must have eaten enough units of food and be next to another firefly in order to reproduce. The new network is a combination of one or more of the candidate networks. The combination can be based on their performance rating or it can be an equal contribution. This form of sexual recombination is a powerful way to create networks that are divergent from its parents. It also adds the capability to inherit the characteristics from two networks possessing desirable qualities. If the network is large enough then simply using sexual recombination without mutation can be sufficient to create a network that has an appropriate set of weights.

In order to produce variant networks (not just from combinations of candidate networks) it is necessary to perform mutations on the the newly

created networks. Mutations can occur in two forms - genetic and crossover. The mutations for neural networks will be discussed in detail shortly.

The environment would very quickly over-populate with networks if new ones were constantly being created and none removed. If a network has a low overall performance then the probability is high that it will perform poorly in the future since its structure does not change once it is created. For this reason a level of performance evaluation should be set in order to rid the environment of networks with poor evaluations.

This process of testing, evaluating, creating and removing networks from the environment should be repeated until networks are evaluated and found to have a high enough level of performance to be an acceptable solution. What also can be done at this point is to refine the network evaluation process so that the evolutionary learning algorithm will even further narrow its search around the network with the correct solution.

4.2.3 Natural Selection

Since the evolutionary learning algorithm is an evolutionary process, it seems reasonable to adhere to the necessary conditions of evolution in everyday natural settings. The first condition selects members from a group that exhibit qualities which better its chance for survival and hence allows it to reproduce and create new members with similar qualities. The other

condition is hereditary variation (mutation).

The conditions of natural selection are embodied in the environment in the form of the evaluation process. The evaluation process rates the "survivability" of the neural networks. This evaluation is the criteria by which the networks are compared when candidates are selected for reproduction. Therefore, the first condition of evolution, natural selection, is ensured by the fact that the network evaluation process is used as the criteria for reproduction.

4.2.4 Hereditary Variation

To insure that an acceptable solution is possible to find, mutation is necessary. Mutation can appear in two forms - genetic and crossover. Both forms affect the weights in a network, each to a different degree. Mutation allows us to diverge from current networks in the hopes of producing better networks.

It is appropriate to make an analogy between genes and chromosomes and their counterparts in neural networks before the two forms of mutation are described. Genes are located on the chromosome within a nucleus of a cell. Chromosomes come in pairs in which one of the gene's alleles appears on one chromosome and the other allele appears on the other chromosome. In neural network terms, a gene is a single weight in the network and a chromosome is

the set of weights for those connections from one computing element to all others. Figure 7 shows what this means for a simple network. Any weight in this network is analogous to a gene. Groups of genes are collected on chromosomes. For example CE_1's connections to CE_3 - CE_6 can be considered a group of weights which are analogous to a chromosome.

4.2.4.1 Genetic mutation

The first mutation is one which affects only a single weight in the network. Assume that the environment contains m networks labeled N_1 - N_m. If network N_k is mutated then we must select at random a connection strength in N_k to modify. The change in weight is based on the old connection strength for some weight W_{ij}.

$$\Delta W_{ij} = x * \Omega * W_{ij}$$

Where X is a random variable between -1 and 1 and Ω is the learning rate which can range between 0 and 1. A learning rate of 0.20 will produce a change in weight from 0 to 20% above or below the current weight value. Figure 8 shows an example of genetic mutation for the network in figure 7. W_{14} is the weight that was changed. The learning rate was 0.2 and x was 0.1.

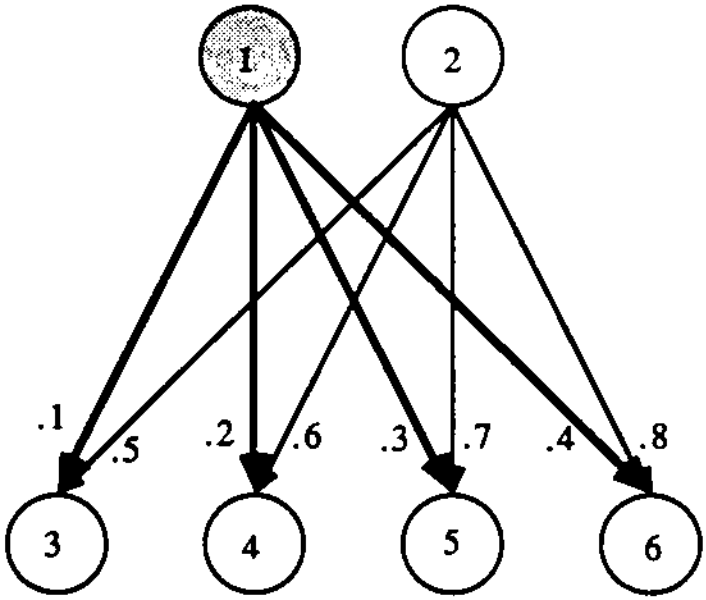

Figure 7 Simple Network for Genetic Analogies

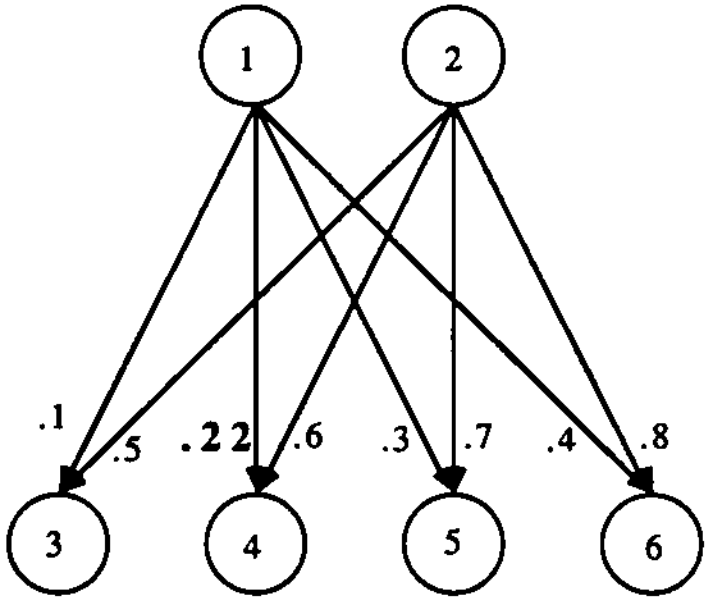

Figure 8 Network after Genetic Mutation

The resulting change in weight was 0.02.

4.2.4.2 Crossover mutation

The second form of mutation, crossover, can be considered an expedient form of genetic mutation. Crossover mutation selects two computing units (two that have the same connections, usually from the same level) at random. Their list of connections to other computing units will be split at some point in the list and crossed over to form the new connection lists. The crossover procedure can be described mathematically using the following terms.

For a given computing element i the set C_i is the set of connection weights (W_{ij}) from unit i to all other elements (j). Let us also define n_i to be equal to the cardinality of set C_i. If we impose an arbitrary ordering on the set C_i and rename the weights in the set a_1 through a_{n_i}, then we can define the set C_i and n_i as follows:

$$C_i = \{ a_1, a_2, \ldots\ldots, a_{n_i} \}$$
$$n_i = |C_i|$$

Then for two random computing elements i and j their connection sets are as follows:

$$C_i = \{ a_1, a_2, \ldots, a_{n_i} \} \qquad C_j = \{ b_1, b_2, \ldots, b_{n_j} \}$$
$$n_i = |C_i| \qquad n_j = |C_j|$$

Let K be a random variable from 1 to n_i. K is the point in the connection list where the split is going to be made. The new connection lists, after crossover at the k^{th} weight (gene), are as follows:

$$C_i = \{a_h \mid 1 <= h < k\} \cup \{b_g \mid k <= g <= n_j\}$$
$$C_j = \{b_h \mid 1 <= h < k\} \cup \{a_g \mid k <= g <= n_i\}$$

An example of crossover mutation is given in figure 9 using the network in figure 7. Computing elements 1 and 2 were chosen for the crossover with k=3.

Before Crossover	After Crossover
$C_1 = \{ .1, .2, .3, .4 \}$	$C_1 = \{ .1, .2, .7, .8 \}$
$C_2 = \{ .5, .6, .7, .8 \}$	$C_2 = \{ .5, .6, .3, .4 \}$

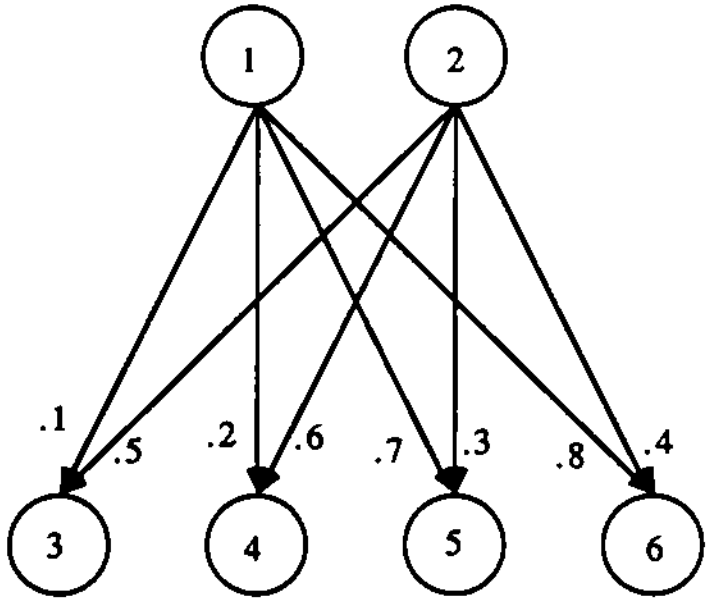

Figure 9 Network after Crossover Mutation

4.3 Analysis of The Evolutionary Learning Algorithm

The evolutionary learning algorithm has advantages over the present learning rules for neural networks. The first advantage is that it specifies in general terms the goal of the environment without having to specify exact input/output mappings. The second advantage is that one can specify in environmental terms what input/output mappings are not desired instead of specifying which ones are desired.

The specification for the goal of a perfect (or acceptable) network is built into the evaluation process. The evaluation process rates the overall performance of the networks and this rating is used to decide which networks will take part in the generation of new networks. The goal of the environment can be changed by simply changing the ratings of the evaluation process. This will lead to the selection of networks with different qualities which will be used to produce new networks.

The goal may also be specified by determining what network mappings are undesirable. This will force the evaluation process to rate poorly those networks that exhibited these mappings. These networks will be removed from the environment because of the poor ratings so that new and better networks can take their place.

Chapter 5

Firefly World Implementation

The domain of Firefly World, as discussed in Chapter 2, was implemented in order to test the theory of the evolutionary learning algorithm. The goal of this environment is to develop fireflies with neural networks which direct them to eat food and to avoid the predator. The following sections will discuss the details of the implementation and present the results of the simulations.

5.1 The System

Firefly World was implemented on a Macintosh II and the language used was Coral Common Lisp. Coral common lisp was chosen because of the speed and ease of programming. It also offered the use of objects so that an object-oriented approach could be taken.

5.2 An Object-Oriented Approach

An object can be considered an entity in the world that possesses identifiable characteristics. For example a robot may be considered an object with such things as position, size, shape, speed of movement as characteristics. When objects interact with each other, requests or messages are sent to one another to communicate information. Objects may request some action to be performed by another object. For example a robot may be requested to calculate its distance to a particular landmark. A message does not tell the object how the request is to be performed but assumes that the object will perform it in the manner it sees fit. This approach allows for objects to be changed later without affecting other objects.

Two parts of the simulation were designed using the object-oriented approach, the neural network's computing elements and the entities of the world. The computing elements were designed as objects because the behavior of each CE is completely local and needs no external control over its operation. The computing element is given information such as its activation level, the list of computing elements that are connected to it, the list of CE's that it is connected to and the functions to collect the activation from its input connections, calculate its own activation and produce an output to other CEs.

When a CE outputs its activation and to other computing elements, it sends a "message" to accept a certain activation. In Firefly World, the messages are to collect the incoming activation in a weighted sum fashion for

all inputs to that particular CE. When the activation spreads down to the output units of the network, the messages from these units are sent to the environment. These messages will then be interpreted by the environment in the fashion it sees fit.

The entities in the world (fireflies, the predator and food) are all designed to act as objects in this simulation. Therefore, if any action should need to be done to the entities, such as moving the predator, consuming food or reproduction of fireflies, then those entities that are involved are sent a message asking it to be done. This message may be in the form of a function to be performed of simply a request to return some information about that entity. For example, when the firefly is sensing what is around it, it must ask the environment what is in each cell. The environment in turn will ask each surrounding entity to identify itself so that this information can be returned to the firefly.

5.3 The Environment

The environment is much like the world that was described in Chapter 2. The shape of the world is a torus which allows wrap-around movement in all four directions. The environment is made up of fixed size cells which can hold only one entity at a time. The entities which can appear in this environment are fireflies, predators and food.

Most communication, for example between fireflies and predators, must first go through the environment. This ensures correct behavior of the entities and minimizes the chance for miscommunication.

5.4 The Firefly

Fireflies are equipped with neural networks that perform a mapping from sensory input of the environment to motor output. Sensory perception is allowed in all eight directions around the firefly and at a depth of two levels. Likewise, movement is permitted in all eight direction but at a depth of only one level per move. The organization of the network is in a layered fashion with activation flowing from the sensory layer to the motor layer. No feedback is allowed in the network. Each layer, with the exception of the sensory layer, can be set up with laterally inhibitory connections. If this is done then the layer of units is grouped together in clusters. Each member of a cluster will laterally inhibit all others within that cluster. After a settling time, only one unit from each cluster will still be activated. This unit will be the only one which will spread activation to the next level. In our implementation, the network contains the sensory layer, an intermediate layer and a motor layer. Figure 10 shows an overview of the firefly neural network.

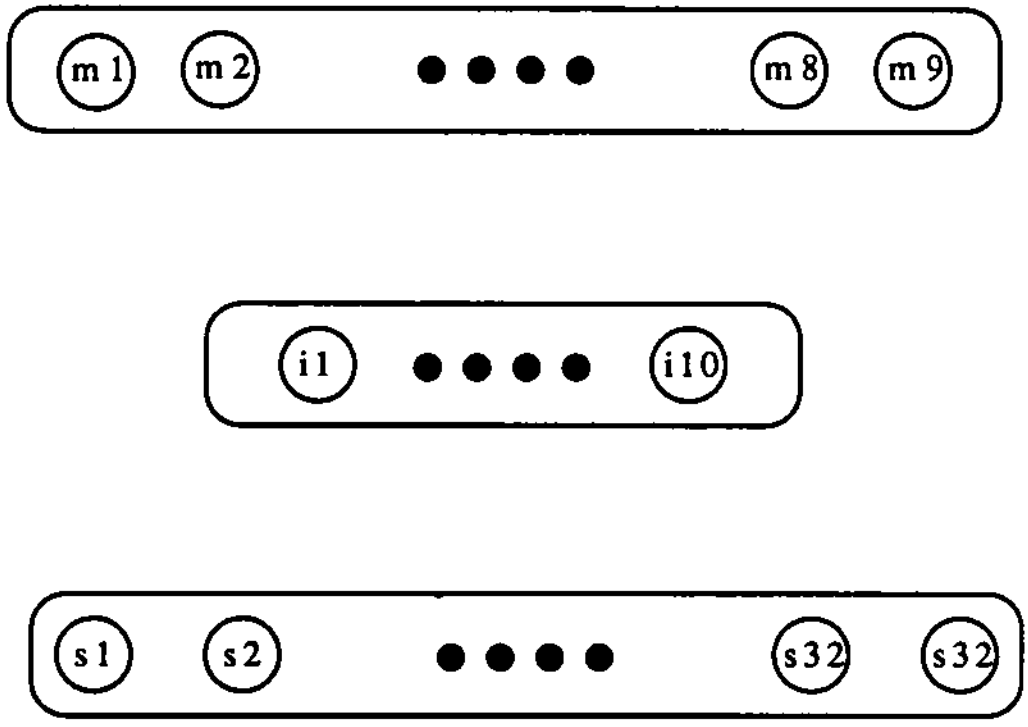

Figure 10 Overall View of Firefly Network

The sensory layer, which is not laterally inhibitory, contains 32 computing units. There are four sensory units for the eight different directions. The first three units for each direction encode the entity which is seen (this could be another firefly, a predator, a food unit or an empty cell) while the fourth unit encodes the level of sight. There is one internal level of non-laterally inhibitory computing units. This level is necessary in order to encode an internal representation of the sensory input units. The motor computing elements are grouped into one cluster within the motor layer. Each movement (including staying stationary) is represented by one computing element. This allows the firefly to move in only one direction.

The firefly also contains a level of energy. This energy decreases over time whether the firefly is stationary or not. If a firefly does not eat food periodically then it will eventually die. Consumed food will add a certain amount of energy to the firefly which will allow it to continue to live. The level of energy is also connected to the ability to reproduce with another firefly which will be discussed in detail shortly.

5.5 The Predator

The predator in this environment can see in the direction of his current movement and to the left and right at forty-five degrees. This has the effect of giving the predator a front and a back which prevents him from having an

unfair advantage over fireflies. The depth of perception is the same as fireflies (two levels). The movement of the predator is not determined by a neural network but by a algorithm. The algorithm will look for any fireflies in its field of vision and randomly moves towards one of them. If none are seen then it will randomly move to an unoccupied cell in its range. If contact is made with a firefly then the firefly will die and be removed from the environment. Contact can be made by either the predator moving into a firefly's cell or a firefly moving into a predator's cell.

5.6 The Food

The food in this environment provides the firefly with one unit of energy which can be used to move a given number of times. The food is only consumable by fireflies and is only replenished when one dies. The exact number of replenished food units can be set by a parameter. Since the supply of food is not constant, there is competition among fireflies for food. This thins out the fireflies leaving the more eager ones to continue their search for food.

5.7 Reproduction

The necessary conditions for sexual reproduction between two fireflies

are that they have the appropriate level of energy and have moved since their last reproduction. When reproduction occurs between two fireflies, a new firefly is produced with a network that is a combination of the two mating firefly networks. For genetic purposes each firefly has three networks - two for reproduction and one for movement in the environment. The new firefly's reproduction network #1 will be produced by combining the reproduction network #1 from each of the mating flies. The same is done for reproduction network #2. The new network for firefly movement is produced by combining its two reproduction networks. The mutation of networks occurs for the reproduction networks and not for the network used for movement. This ensures that the effects of the mutations will not be lost when reproduction occurs for the mutated firefly.

The two kinds of mutation that are used are genetic and crossover mutation. These mutations affect a single weight in the network and a collection of weights respectively. The details of the mutation process was discussed in Chapter 4.

5.8 Simulation Results

Before the results of the simulations are presented, it is necessary to discuss the approaches taken to collect the data. The first simulation approach is a divide and conquer method. It takes the overall goal of the network and

divides it up into smaller sub-goals. Simulation runs were made to achieve these sub-goals and after networks were developed, they were put together in the same simulation to achieve the overall goal. This approach allowed a complex goal to be achieved piece by piece. The second simulation appoach continuously ran simulations for the overall goal. When only N networks remained in the environment, the simulation would save these networks for the next simulation.

Each simulation run consisted of 10 fireflies. A total of about 1500 fireflies were used in continuous runs during a 15 hour span. The top 4 fireflies were carried over from run to run and the environment was repopulated with these and 6 random flies. After the entire simulation was finished, the top 10 fireflies were chosen by the number of simulation runs they survived for and the amount of food that they ate. Seventy percent of the best flies were the result of either sexual or asexual reproduction.

These top ten fireflies were compared to random fireflies. The evolved fireflies exhibited abilities to avoid the predator from a couple of directions whereas the random flies seemed to want to move in a single direction regardless of what is in its field of vision. Fireflies are not given the ability a priori to associate objects from one direction to another. So if a firefly learns to avoid the predator from the left, it cannot take this information and apply to seeing the predator on its right. A firefly must learn seperately for each direction what it must avoid and what it should be attracted to. The results of

the simulations show that the evolutionary algorithm works in selecting those networks that perform well so that over generations of fireflies ones with survivable behavior will be evolved.

Chapter 6

6.1 Concluding Remarks

The information modeled in a network is collectively stored in the connection weights. So in order to model some information or function, the correct set of connection weights needs to be determined. This is a difficult problem if the size if the network is large. To overcome this problem, learning rules have been developed which successfully learn the correct weights. However, these learning rules require the designer to know what the output of the network should be for every input presented. This can be a problem if there is no one correct output pattern for an input pattern. In many cases it is much easier to specify the input to output mappings which are not desired instead of the ones that are desired. This information cannot be used in the previously developed learning rules.

The evolutionary learning algorithm overcomes the deficiencies of the previous methods by using evolutionary pressures to find a network with a correct set of connection weights. The algorithms allows the designer to evaluate the responses of networks and select those with the best overall

response. The evaluation process can be one that looks for bad responses by networks and assigns an appropriate response rating to that network. By using this approach, one can specify what is not desired by the network as well as what is desired. The simulation results show that this approach does work in evolving networks which perform relatively well in the environment in which they are set in. This approach to learning connection weights is best suited for domains (which are capable of being simulated) where it is easier to specify what not to do as opposed to specifying what to do.

6.2 Future Work

An implementation of Firefly World was designed with a specific goal in mind. It would be desirable to design a generic environment where any function can be modeled in a network using the evolutionary learning algorithm. The function to be modeled along with the network evaluation process can be specified and therefore mold the generic environment into one for a specific domain.

Another addition to this approach would be to merge the evolutionary learning algorithm and the previous learning rules into one learning process. The learning process could possibly benefit from both approaches and hence speed up the time it takes to learn.

Bibliography

[Ackley85] Ackley, D. H., Hinton, G. E., & Sejnowski, T. J. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science*, 9, 147-169.

[Amari77a] Amari, S. A. (1977a). A mathematical approach to neural systems. In J. Metzler (Ed.), *Systems neuroscience* (pp. 67-117). New York: Academic Press.

[Amari77b] Amari, S. A. (1977b). Neural theory of association and concept formation. *Biological Cybernetics*, 26, 175-185.

[Anderson70] Anderson, J. A. (1970). Two models for memory organization using interacting traces. *Mathematical Biosciences*, 8, 137-160.

[Bienenstock82] Bienenstock, E. L., Cooper, L. N., & Munro, P. W. (1982). Thheory for the development of neuron activity: Orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience*, 2, 32-48.

[Dillon70] Dillon, L. S. (1970). *Animal Variety*. Dubuque, Iowa: Wm. C. Brown Company Publishers.

[Feldman82] Feldman, J. A. (1982). Dynamic connections in neural networks. *Biological Cybernetics*, 46, 27-39.

[Feldman85] Feldman, J. A. (1985). Connectionist models and their applications: Introduction. *Cognitive Science*, 9, 1-2.

[FeldmanBallard82] Feldman, J. A., & Ballard, D. H. (1982). Connectionist models and their properties. *Cognitive Science*, 6, 205-254.

[Flew84] Flew, A. (1984). *Darwinian Evolution*, London, England: Paladin Press.

[Fogel69] Fogel, L. J., Owens A. J., & Walsh, M. J. (1967). *Artificial Intelligence Through Simulated Evolution*, New York, NY: John Wiley & Sons, Inc.

[Fukushima75] Fukushima, K. (1975). Cognitron: A self-organizing multilayered neural network. *Biological Cybernetics*, 20, 121-136.

[Fukushima80] Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36, 193-202.

[Grossberg76] Grossberg, S. (1976). Adaptive pattern classification and universal recoding: Part I. Parallel development and coding of neural feature detectors. *Biological Cybernetics*, 23, 121-134.

[Holland84] Holland, J. (1984). "Genetic Algorithms and Adaptation" in O. Selfridge, et al., *Adaptive Control of Ill-Defined Systems*, Plenus Press: New York.

[Hopfield82] Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, USA, 79, 2554-2558.

[Hopfield84] Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences*, USA, 81, 3088-3092.

[Marr76] Marr, D., & Poggio, T. (1976). Cooperative computation of stereo disparity. *Science*, 194, 283-287.

[Minsky69] Minsky, M., & Papert, S. (1969). *Perceptrons*. Cambridge, MA: MIT Press.

[Poggio78] Poggio, T., & Torre, V. (1978). A new approach to synaptic interconnections. In R. Heim & G. Palm (Eds.), *Approaches to complex systems*. Berlin: Springer-Verlag.

[Rosenblatt62] Rosenblatt, F. (1962). *Principles of neurodynamics*,Spartan: Spain.

[Rumelhart86] Rumelhart, D. E., McClelland, J. L., et al., (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*, Cambridge, MA: The MIT Press.

[Sutton81] Sutton, R. S., Barto, A. G., (1981). Toward a Modern Theory of Adaptive Networks: Expectation and Prediction, *Psychological Review*, 88, p. 135-170.

[Watson77] Watson, J. D. (1977). *Molecular Biology of the Gene*, Menlo Park, CA: W. A. Benjamin, Inc.

The three page vita has been removed from the scanned document. Page 1 of 3

The three page vita has been removed from the scanned document. Page 2 of 3

The three page vita has been removed from the scanned document. Page 3 of 3