A MODEL FOR GOAL ORIENTED LEARNING IN A NEURAL NETWORK

by

Bryan Aucoin

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

MASTERS OF SCIENCE

in

Electrical Engineering

APPROVED:

F. J. Ricci, Chairman

C. J. Egyhazy

E. Haddad

R. M. Wnek

May, 1989

Blacksburg, Virginia

A Model for Goal Oriented Learning in a Neural Network

by

Bryan Jennings Aucoin

Committee Chairman: Fred J. Ricci
Electrical Engineering

(ABSTRACT)

A mathematical model for goal oriented learning in a network of neuron-like elements was developed. Using a mouse/goal box analogy, a simulation of a network with four elements was programmed in Turbo Pascal, Version 4.0 (Borland International) to test the model. Each location in the network corresponded to a particular network input. The output of the network consisted of one of four behaviors: forward, backward, left or right. The network successfully learned sequences of up to six movements in increasingly complex mazes.

Acknowledgements

To my beautiful wife, who despite many hours of neglect, gave me the patience and encouragement which allowed me to complete this paper.

I would like to thank the members of my thesis committee, Dr. Ricci, Dr. Egyhazy, Dr. Haddad, and Dr. Wnek for their guidance and support.

Table of Contents

1.0 INTRODUCTION:

In the last thirty years we have seen an astounding increase in the power of digital computers. One only needs to read the "antiquated" computer texts of the 1950's to recognized the extent of the progress. Consider that 640 kilobytes of Random Access Memory (RAM), now considered woefully inadequate even for microcomputer applications was at that time an obscene dream. Computers were buildings filled floor to ceiling with racks of vacuum tubes. Programming consisted of flipping switches or running patch cords behind a console the size of a large desk. Now more computing power than the most sophisticated computers built in the 1950's fits tidily on a desk. The trend is continuing, with a complete generation of technology every five to ten years [1].

I believe this phenomenal increase in technology has produced, and in turn has been driven by economics. With the advent of electronic digital computer we have entered a new "information" age, in which generation, manipulation and dissemination of information has taken unprecedented prominence.

Information is rapidly beginning to be viewed as a product, much like agricultural and manufactured goods. In our extremely competitive world, the generation of information is becoming synonymous with the generation of wealth. As a result, we are building increasing more powerful and easily available computers to satisfy our needs to create, analyze and process information, and as we learn, our requirements for these capabilities increase.

I consider that in a broad sense we are beginning to face the limits of a paradigm. The original architectural foundations for modern stored program digital computers were proposed by John Von Neumann in 1946. Despite the increase of computational power we have seen since then, machines today share the same fundamental architecture, with a few extensions [2]. However, Von Neumann machines have limitations, and as our need for capability increases these limitations have become increasingly restrictive.

Von Neumann machines are serial machines (instructions are executed in sequence), and therefore, their computational capacity is limited by the speed at which they can execute their programs. Unfortunately, the cost of building a machine increases rapidly with its speed. Even if one is willing to pay the price, at the current level of technology, physical

laws are beginning to appear as limits[1].

Another restriction results from how information is represented and processed in these machines. Each datum must be stored in its own memory element, as well as its relation to other data (e.g. in data structures such as trees). The processing of the data is accomplished by the application of a set of rules which are implemented in a list of instructions, or program. For an individual to write a program to accomplish a given task, he must first know explicitly all the rules and all the relations of his data. The definition of rules can be an involved, arduous process, and for many classes of problem cannot be practically done. Once the rules are defined, the programming process itself is not particularly easy, as evidenced by major software companies' consistent inability to ship software on time.

As always, necessity is the mother of invention. Our necessity is increased computational power, and a means by which we can resolve the problems mentioned above. The

---

[1] Consider that in today's supercomputers, the speed of computation is limited primarily by the requirements for heat dissipation, which limits how closely components can be packed, and the speed of light, which limits how fast information can travel between points. The intent here is not to say that we will not make future technological breakthroughs that will greatly increase our computational capability, but that this extension of technology will come at a price.

"invention", as it always has been, is a shift in perspective, a revisiting of basic assumptions, and finally a shift in paradigm. We have begun to revise how we think about computational processes, and how we can develop machines to implement these processes.

One of the primary directions this effort has taken is the development of machines that perform processing in parallel[2]. In many cases a computational process can be broken down into child processes, each of which can be performed in parallel by relatively simple, slow processors. Although the individual processors in a parallel machine may be much slower than a serial processor used to do the same job, the overall time to complete the task can be shorter. A large number of small, slow, simple processors may be much less expensive than a single extremely powerful processor.

Some of the more interesting developments in the past several years have resulted from taking the idea of parallel processing to its extreme: using extremely simple processing elements in highly interconnected networks. The study of these networks has a variety of titles, including parallel distributed processing, massively parallel processing, etc.

---

[2] For a discussion of the relative merits of and motivations for parallel processing, see Hwang and Briggs [1].

In most cases the processing elements advanced are either proposed models of neurons or some neural process, hence the networks are often to as neural networks[3].

These networks have a number of characteristics that make them interesting [4]:

1. Existing neural networks are not programmed in the conventional sense, but are trained[4]. Examples are placed on the input of the network, and the resulting response is compared to a desired network response. Using any of a variety of learning algorithms, the network interconnections are modified until the network response is always the desired response. (Typically, a weight associated with each interconnection is modified). The rules for differentiating various input patterns need not be explicitly defined, but over the training period become inherent in the network

---

[3] For the record, the idea of using networks of neuron-like elements to perform calculations is not a new idea when measured against the field of computer science. For some interesting history and perspectives the reader is directed to the prologue and epilogue of Minsky and Papert [3] and Volume 1, Chapter 1 of Rumelhart and McClelland [4].

[4] Some neural networks are not trained, but are self organizing. Kohonen [5] proposes a model of a neural network as an adaptive filter. Each element provides extensive feedback to other elements and receives the same. Hopfield, in a variety of publications, approaches the problem of organization in networks as a thermodynamic process. Organization occurs in a relaxation process, like annealing.

interconnections. Hence, neural networks do not require programming in the conventional sense.

2. Since the information is stored in interconnection weights, the loss of any particular processing element is inconsequential. Even if the element was of primary importance for a particular computation, the network can be retrained using the remaining elements. Hence a neural network is extremely fault tolerant, and even if a critical fault occurs, the network can easily recover.

3. As stated earlier, neural networks perform processing in a highly parallel fashion, and the network elements are extremely simple. If we appropriately define network elements and learning algorithms, we may be able to build devices with a capacity which is beyond our current computational capability. There is also the potential of an extremely low cost.

The most promising area of neural networks in the long term, in my opinion, is hardly addressed in the literature.[5] That is the area of goal oriented learning. With few exceptions all the machines we have developed must either have

---

[5] Rumelhart and McClelland [4] touch on the subject briefly in Chapter 14.

a programmer or a teacher. In a sense these machines do not solve problems, but merely execute the solutions using the algorithms we develop. The purpose of this thesis is to present both a different perspective on machine learning and the beginnings of a model for a machine which can develop its own algorithms.

2.0 REVIEW OF LITERATURE

The literature search began with a catalog search (both books and periodicals) of both the Virginia Tech and the George Mason University Libraries. The subject areas searched included neural networks, parallel distributed processing, massive parallel processing, learning, operant conditioning and goal oriented learning. Using the bibliographies of the few publications I had (mostly periodicals), I performed author searches for D. Hebb, F. Rosenblatt, M. Minsky, S. Papert, T. Kohonen, J. Hopfield, J. McClelland and D. Rumelhart. Although some useful information was found, in general the holdings in both libraries are limited. I suspect this results from the relatively recent publication of most of the books and other publications that I wished to obtain.

After the marginal success at the libraries, I began to discuss the problem of obtaining relevant literature with a number of friends and associates who do research frequently. One of these people directed me to Reiter's Scientific and Professional Books, 2021 K. St, NW, Washington D.C. 20006. To my surprise I found that Reiter's had an extensive collection of books on neural networks and related subjects. Based upon what I had previously learned from the publications I had

already obtained, I purchased a number of books by key authors.

Unfortunately, the search did not reveal much information directly related to goal oriented learning in neural networks. There are, however, many publications that contain pertinent information on neural network models and their biological foundations. The major publications which were reviewed in the preparation of this thesis, in order of importance are:

Principles of Neural Science, edited by E. R. Kandel and J. H. Schwartz, 1981 (obtained from George Mason). As the title implies, this is an introductory text in neurology. The primary advantage of this text is that it contains a thorough but relatively simple presentation of neural structure, chemistry, and function. The text also develops, in the space of five chapters, an electrical model describing the functioning of a neuron which proved useful in the development of the learning model presented herein.

Self-Organization and Associative Memory, T. Kohonen, 1984 (obtained from Reiter's although George Mason has a copy). Kohonen presents a neural network model for associative memory. His model for the network element is more refined than most of the models proposed in the rests of the

literature and resembles in the broadest aspects the model presented in Kandel and Schwartz. He contends that the output for neurons is graded and corresponds to variations in firing frequency, as opposed to the an "on" or "off" output characteristic of some neural models. In general, his mathematics are rigorous.

Perceptrons, Expanded Edition, M. L. Minsky and S. A. Papert, 1988 (originally in published 1969, this book was also obtained from Reiter's, although George Mason has a copy). Marvin Minsky and Seymour Papert performed a rigorous mathematical analysis of the Perceptron in this text, revealing some interesting capabilities and limitations of this simple model. In the initial work which lead up to this thesis, the Perceptron was used as the network element. As the work progressed it became evident that the Perceptron was too limited a model to be useful.

Parallel Distributed Processing, edited by D. E. Rumelhart and J. E. McClelland, Volumes 1 and 2, 1986 (Reiter's). This book, generally called PDP in the literature, contains a series of articles on various aspects of neural computing and presents some implementations for natural language processing, pattern and speech recognition, etc. The learning algorithm of choice in this text is "back-

propagation" or the generalized delta rule.

Explorations in Parallel Distributed Processing, A Handbook of Models, Programs, and Exercises, edited by Rumelhart and McClelland, 1988 (Reiter's). This book contains a synopsis of the various network models presented in PDP. It also comes with several diskettes containing programs implementing the models presented in PDP.

Neural Networks and Natural Intelligence, S. Grossberg, 1988 (Reiter's). This volume contains some of Steven Grossberg's major articles (with several coauthors) on neural processing. One of the major articles presents his theory of Adaptive Resonance, a powerful algorithm for pattern recognition.

Cognizers, Neural Networks and Machines that Think, R. C. Johnson and C. Brown, 1988 (Reiter's). This book is an overview of the history of neural networks and the theories that are currently prevalent.

3.0 MATERIALS AND METHODS

A mathematical model of goal oriented learning in a neural network was developed and tested using simulations written in Turbo Pascal, Version 4.0 (Borland International). An analogy of a mouse in a maze was used to enhance an intuitive grasp of the learning process. A graphical interface was used to display the network input and output, and the state of each network element. Also, the corresponding movements of the mouse through the maze were displayed.

In order to reduce the complexity of the program code, the various parameters used in the simulation were coded as Pascal constants. Given the rapid speed in which Turbo Pascal compiles source code, this was not a restriction. Output from each simulation trial, such as elapse time and the number of moves required to complete the maze, was written to a text file. Text files from subsequent simulations were imported into a Lotus 1-2-3 spreadsheet (Lotus Development Corp., Version 2.01).

The spreadsheet was then used to generate summary statistics and produce graphical output.

Each simulation consisted of approximately 1000 lines of commented source code. A sample simulation is enclosed as an appendix to this paper.

An 8 MHz IBM PC AT with an Enhanced Graphics Adapter (EGA) and 512 kilobytes of Random Access Memory (RAM) was used to run the simulations.

4.0 RESULTS

A detailed presentation of the results is contained in Section 5.5 of this thesis. In summary a mathematical model for goal oriented learning in a neural network was developed and successfully tested.

5.0 DEVELOPMENT AND PRESENTATION OF THE THESIS

5.1 Introduction

This portion of the thesis consists of five major sections: this introduction and four others. Section 5.2 presents an overall view of intelligence and how it relates to the functioning of an organism in an environment. Section 5.3 describes some of the biological aspects of learning, beginning with the functioning of the neuron and examining some of the known mechanisms for learning. This section provides a basis for the model presented in Section 5.4. In addition to the presentation of the model, Section 5.4 discusses some of the insights which are gained from the model and some areas for further study. Section 5.5 presents the results of computer simulations of the network model.

5.2 An Overview of Intelligence and the Organism

5.2.1 Intelligence and Artificial Intelligence

In my opinion, the term Artificial Intelligence as it is used today does not refer to true intelligence exhibited by a machine, but an aide to it.

Consider a designer developing an expert system. The design process basically involves analyzing the thought processes of knowledgeable people and then formalizing the rules that they use to make decisions. Similarly, in work involving natural language processing, pattern recognition, and similar fields of study the concentration is on the development of algorithms that can be implemented by machines to perform the desired differentiation. The intelligence is not in the algorithms, nor in the processes that implement them, but in their definition[6].

I consider that intelligence is not a process that arises from an algorithm, but an algorithm that arises from a process. Hence the Holy Grail in the implementation of true machine intelligence is the definition of a process that develops algorithms. My primary goal in this thesis is to present some ideas on the development of such machines. This will obviously require a shift in how we normally think about intelligence.

A neural network approach was taken for the following

---

[6] There is a fine but important distinction between an algorithm and a process. An algorithm is a list of rules or steps that must be followed to achieve a desired end. A process is the actual implementation of an algorithm. This terminology has been used in the discussion of parallel processing techniques. See Hwang and Briggs [1].

reasons:

a. We know that organisms, in order to survive in a constantly changing and often hostile environment, develop new sets of behaviors. In effect they develop algorithms on an ad-hoc basis to fulfill their needs. Hence, a good starting point for the investigation of the process we are trying to define is a review of some of the biological foundations of learning and behavior. This includes how neurons in the brain function and how they change as a result of learning.

b. One thing that conventional Artificial Intelligence has taught us is that many seemingly simple processes, especially in the area of perception, require an immense amount of processing power[7]. As stated in the Introduction to this thesis, neural networks, because of their highly parallel architecture, appear to be a promising avenue for obtaining the required processing power.

The next question which arises in the context of this work is which is more important: understanding the biological processes which result in intelligence or building a machine

---

[7] The converse is also true. Many seemingly complex processes, such as playing chess and other games, requires a surprisingly small amount of processing power.

that implements intelligent behavior. The answer is that attempts to confine artificial intelligence strictly to the realm of the machine and algorithm have met with limited success. Although it may be possible to build a machine in isolation of biological data, it seems a better approach to learn everything one can about the biological underpinnings of intelligence and then extrapolate upon that knowledge.

5.2.2 On Mice, Mazes and Goal Boxes

The primary tenet of this thesis is that artificial intelligence should be viewed from the context of an organism functioning in an environment. Intelligence did not appear for no reason. It is an evolutionary adaptation that enhances an organism's ability to face the myriad challenges inherent in day-to-day existence in an environment that is constantly changing and often hostile. Those creatures endowed with a greater ability to acquire and apply knowledge of their environment have a selective advantage.

When an animal is learning to contend with a new environment it is in effect developing algorithms. An animal has a certain repertoire of behaviors which it develops from simpler behaviors. It develops these behaviors in response

to the pressures of its environment and its own needs[8].

For the purposes of this paper, there are five traits that characterize an organism:

1. The organism functions in an environment.

2. The organism has some mechanism for assessing the state of its environment (i.e. perceptions).

3. The organism has a set of behaviors through which it can modify its relationship to the environment and its own internal state.

4. An organism has needs, which can be thought of as motivations to behavior. The organism functions to gratify these needs.

5. Lastly, the organism has within it a decision maker

---

[8] Although I had arrived at these conclusions independently, I later found that similar ideas are expressed elsewhere [4], [6] and [7], though less concisely. Reference [6] has the very promising title "Neural Mechanisms of Goal Directed Behavior and Learning". When I first found it, I thought I had struck the "Mother Load". A bit more reading revealed that it contained a great deal of information on the areas of the brain associated with goal oriented learning, and some broad hypotheses on function, but very little information on neural mechanisms at the level of the synapse. Hence, it was of limited use in the development of the model to follow.

to choose behaviors which are appropriate to the environment and need. A block diagram of such an organism is shown in Figure 1.

An organism can be described as a system with a feedback loop through the environment. The environment results in perceptions. Based upon these perceptions and the organism's needs, a behavior is performed which changes the organism's relationship to its environment. The behavior may reduce, exacerbate or have no effect on the organism's needs. Learning can be defined as the modification of the choice of behaviors to maximize the probability of gratifying a need based upon experience.

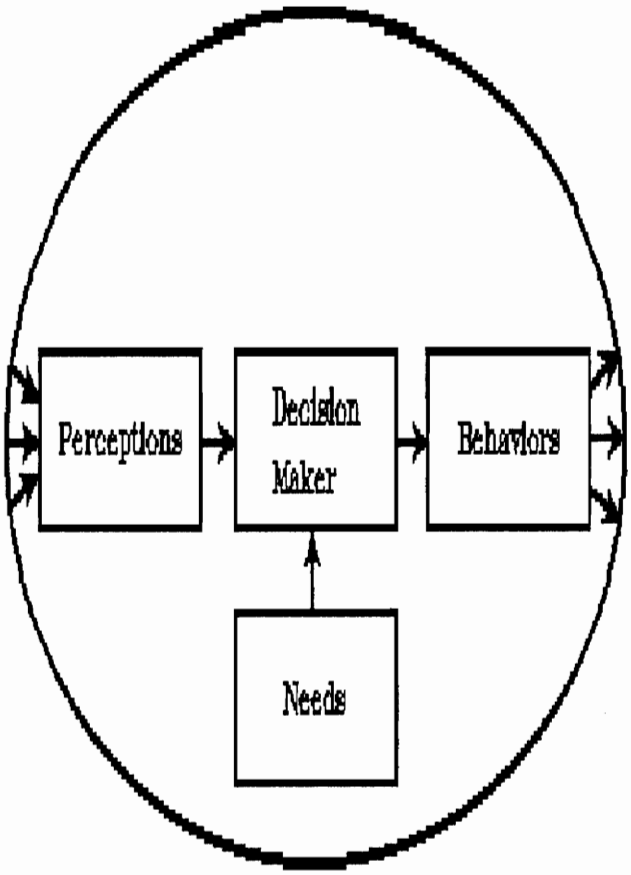

Figure 1 - The Simple Model for an Organism to be used in this Thesis.

Consider a mouse[9] in a maze with a goal box at the end. Our mouse, AKA Scooter, has a set of perceptions; he can identify, through some process, the section of the maze in which he is currently located. He[10] has a set of behaviors; he can move forward, backwards, turn left or turn right, and if he finds food he can eat. The need is hunger, and it motivates Scooter to search the maze for food. At each point in the maze, Scooter makes a decision as to which of his set of behaviors to execute. When he finds the goal box his hunger will be gratified, and he will no longer behave to gratify that particular need. When Scooter is first placed in the maze he will perform a certain sequence of behaviors. If he has no prior experience of the maze, for all practical purposes the movements will be random. If the sequence is successful (i.e. he finds the goal box), his hunger will be gratified and the sequence will be reinforced (i.e., the sequence is more likely to be repeated). If the sequence is not successful, it is not as likely to reoccur. Over a series of trials, a sequence of behaviors will be reinforced until it is the only one performed.

---

[9] My wife, after seeing the simulation I was doing on my computer has christened him "Scooter".

[10] Scooter is asexual. As yet I have not put procreation in his repertoire.

5.2.3 The Limitations of Existing Neural Network Models

The next step in the search for a plausible neural network model for goal oriented learning is obviously to review existing network models. However, such a review did not reveal any promising candidates.

The problem that stands out the most in the learning algorithms discussed in most of the literature is the requirement to have a teacher[11]. The typical training of a neural network consists of placing a set of patterns on the input of the network. The resulting output is compared with the desired output and the network interconnection weights are adjusted based upon the comparison. This implies that there is an active teacher, separate from the network, who knows the correct network response, evaluates each response and then informs the network of its performance.

Although such networks can be taught sequences of patterns, which is essential for goal oriented learning, the network must be trained in each individual pattern. Such a model for learning cannot accurately reflect the processes which occur in a living organism, nor provide a means for a

---

[11] This general problem is pointed out by both Kohonen [5] and Grossberg [7].

network to develop algorithms. If there is an external teacher, where does it reside? How does it function? How does it know whether the network response is correct or incorrect? How are the individual cells in the network informed of their performance so that they can adjust their input weights accordingly? It appears more likely that learning is a localized event, where each neuron makes the decision as to whether its output was correct[12].

Models that rely on the feedback characteristics to do adaptive filtering (Kohonen, et al) or organize themselves by some relaxation process (Hopfield, et al) may describe some of the physical processes in the brain. These models are extremely useful in the areas of associative memory, perception, and other self-organizing processes [5] [7] [8]. They are inadequate to describe goal orient learning, because there is no mechanism by which they can dynamically learn sequences of output in response to a set of stimuli.

Probably the most applicable work to date has been Grossberg's Adaptive Resonance Theory [7]. However, while

---

[12] In this discussion, we can construe the environment as a teacher. Recall, however, that in this context we are referring to the output of individual neurons, and the modification of individual synaptic weights. There are no known or even probable mechanisms by which the environment can directly accomplish such localized and minuscule changes.

Grossberg's model appears to be a plausible model for learning and pattern recognition, it is not immediately apparent it can be directly applied to goal oriented learning.

5.3 Biological Aspects of Learning[13]

5.3.1 A Overview of the Neuron

Given the lack of pertinent information in the literature, we must begin to develop our own model. The logical place to start is with a description of the functional elements that form the network, the neuron. Unfortunately this is not a particularly easy task. The human brain has approximately 1000 different types of neurons. These cells are morphologically as well as functionally different, but there are common attributes. We can begin to develop a model based upon our understanding of these attributes. A thorough description of what is known about neurology would constitute many volumes. The following descriptions, therefore, do not constitute a complete summary or even a thorough discussion. It is intended to provide the reader with sufficient background to be able to understand the development of the model which is to be presented herein, and provide the reader

---

[13] The discussion to follow is a synopsis of some of the discussion in Kandel and Schwartz [9]. Other references are called out where pertinent.

with an appreciation of the complexity of the problem. The reader is directed to the references for more complete discussions.

Neurons vary a great deal in shape and size, but all share a similar morphology. The typical nerve cell consists of four regions, the soma or cell body, the dendrites, the axon and the presynaptic terminals of the axon (Figure 2).

The soma serves the same function as the cell body of all animal cells. It metabolizes food, generates proteins and other macromolecules and contains the genetic information required for reproduction within its nucleus.

The cell body forms into the axon, a tubular extension of the cell's membrane which may be over a meter long. The function of the axon is the transmission of electrical signals to other neurons or active receptors on other cells controlled by the nervous system, such as muscles. The neuron transmits information by means of an action potential, an electrical pulse that traverses down the length of the neuron starting at the cell body at the base of the axon (the axon hillock).

Near its end the axons divides into many small branches with highly specialized ends called presynaptic terminals.

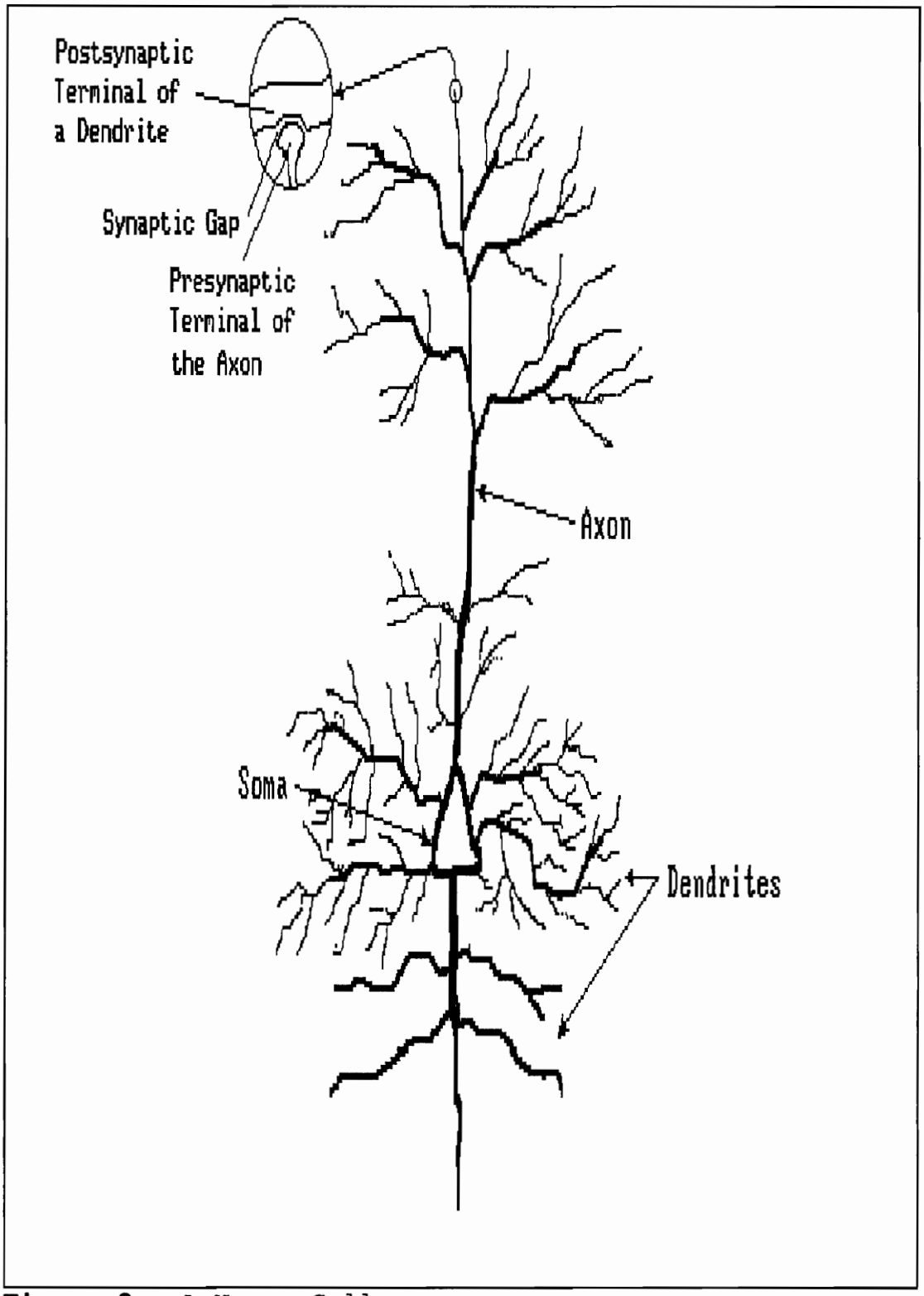

Figure 2 - A Nerve Cell

The terminals contact the receptive surfaces of other cells and transmit, either chemically of electrically, signals across the synaptic gap.

Information is received in the cell via its membrane or by other extensions of the cell membrane called dendrites. Like the axon the dendrites divide into many branches. They also terminate in highly specialized ends called postsynaptic terminals.

5.3.2 Signal Processing in the Neuron

5.3.2.1 The Resting Membrane Potential

The neuron, like other cells of the body, sustains an electrical potential across the cell membrane, with the outside of the membrane being positive. This potential results from the selective diffusion of ions (primarily sodium, potassium, and chloride) across the membrane. When the neuron is unperturbed by any external stimuli, the potential from the separation of charge remains in the range of 40 to 75 mV, depending on the cell, with a typical value of 60 mV. This resting potential is important: signaling involves the disturbance of the cell's potential from the resting potential. An increase in the magnitude of membrane

potential is referred to *hyperpolarization*, while a decrease in the potential is referred to as *depolarization*.

5.3.2.2 Input Potentials

Synaptic Potentials are the means by which one neuron perturbs the membrane potential of another cell. Input to the cell is via the dendrites at the post synaptic terminals. The signal is transmitted either electrically or chemically, depending on the type of cell, to these localized sites and results in a change in the membrane potential. In general, electrical transmission is used in motor neurons transmitting signals over long distances. Electrical transmission is extremely rapid compared to chemical transmission[14], but the synapses that control electrical transmission are not as plastic (they do not change their efficacy for signal transmission). Hence, chemical transmission is of primary interest in this paper.

In chemical transmission, the initiating cell releases a chemical known as a neural transmitter into the synaptic gap. The neural transmitter interacts with receptors in the

[14] This is one of the reasons why one does not seem to feel pain immediately when one touches a hot piece of metal. However, once the pain is perceived, pulling one's hand away happens very quickly.

postsynaptic cell to generate a corresponding change in the membrane potential of the post synaptic cell.

One type of input potential, related specifically to sensation, is generated by a *receptor*, and is referred to as a *receptor, or generator, potential*. The receptor in a sense is an energy transducer, translating sensory input such as sound, heat, or light into electrical energy to be used by the neuron for performance of calculations. For example, at the receptor organ of a sensory neuron in a stretch reflex, the mechanical energy of the stretching muscle is converted into electrical energy, which changes the potential across the cell membrane. These inputs are graded, and may either be depolarizing or hyperpolarizing. The change in membrane potential follows the magnitude of the sensation.

5.3.2.3 Signal Integration

The membrane of a neuron is a leaky integrator. The local signals produced by the synapses or receptors are weighted and summed. The strength of a particular input depends on the strength of the signal received (the amount of neural transmitter received at the post synaptic terminal), the synaptic size and type (i.e. synapses that mediate hyperpolarization and depolarization of the cell membrane have

different morphologies and efficacies) and the passive properties of the cell membrane (transmission strength decays as the distance from the axon hillock increases). The integration performed is both temporal and spatial. In both cases, the integration appears linear[15], but with a saturation level [5].

5.3.2.4 The Action Potential

A trigger zone at the axon hillock adds up the excitatory and inhibitive potentials generated by the receptor and synaptic excitations, and if the potential is reduced to a threshold, usually about 45 mV, an action potential will be generated. The action potential is a large depolarizing signal, up to 110 mV in amplitude, and typically about 1 msec in duration. It results from a rapid influx of positive ions, primarily calcium, beginning at the axon hillock. The infused ions change the permeability of the adjacent section of the axon, and an action potential propagates down the axon. The axons of some larger cells are coated with a substance called myelin which increases the rate of propagation. The myelin coating is responsible for the characteristic white color of the tissue in the interior of the brain.

---

[15] The change in the potential across the membrane is proportional to the excitation received within a linear region.

5.3.2.5 Output or Secretory Potential

At the terminal region of the neuron the action potential serves as a stimulus to the secretion of neural transmitters into the synaptic gap. This release is graded as it depends on a local potential which is mediated by calcium ions. The transmitter diffuses to the next cell where it causes a local change in cell membrane potential. The efficacy of the transmission is affected by a variety of circumstances, including the presence of certain neural transmitters, and the amount of special chemicals called receptors (different from the receptors dealing with sensation discussed above) embedded in the post synaptic membrane.

5.3.3 Learning

Although this thesis is an investigation of a model for goal oriented learning in a neural network, it is important to gain a perspective of the various types of learning that are exhibited. This may provide insight into some of the mechanisms which are involved and potentially applicable to goal oriented learning.

The descriptions below are based upon investigations

performed on the Aplysia California, a marine snail with a relative simple nervous system consisting of approximately 20,000 neurons[16]. Prior to the discussion it is necessary to provide some background into some of the experiments performed on the Aplysia. To date, the work (Kandel and Schwartz, et al) has focus primarily on the gill and syphon withdrawal reflex of this organism. The gill, the Aplysia's respiratory organ, is covered by a mantle shelf, a flap of skin which terminates in a small fleshy spout known as a syphon. Touching the syphon causes is to withdraw and the gill to contract. This response is controlled by a relatively simple neural network and is modified by various forms of learning.

A simplified representation of the neural network associated with the response is shown in Figure 3.

5.3.3.1 Habituation

Habituation is the decrease in a behavioral reflex when a stimulus is repeated and there is no noxious effect. In the Aplysia, if the syphon is touched repeatedly, a smaller and smaller response results. If only a few stimuli are given, the response will mostly recover. However, if the

---

[16] This work is presented in Kandel and Schwartz [9] and in greater detail in Davis, Newburgh and Wegman [10].

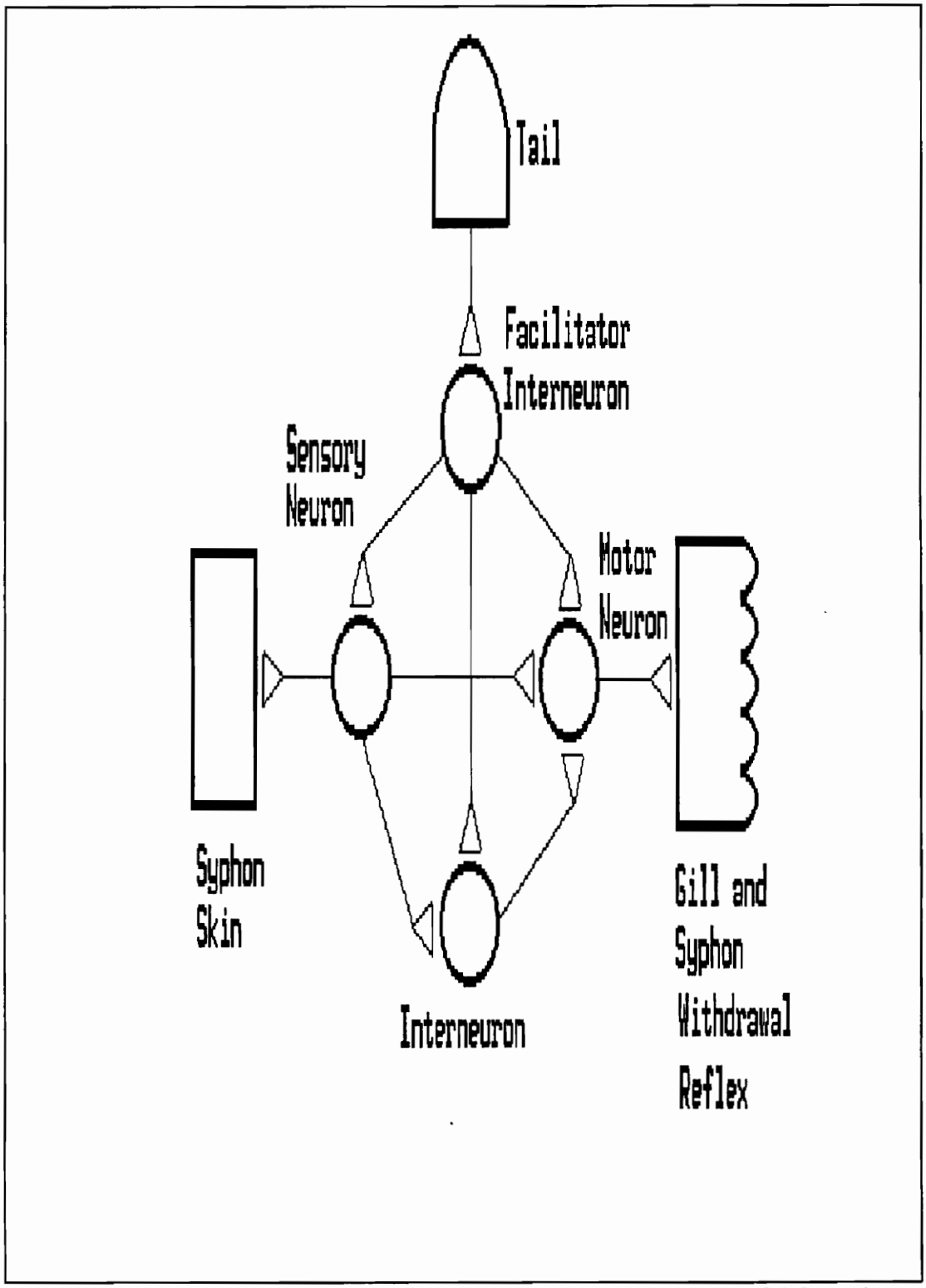

Figure 3 - Neural Network Mediating Aplysis California's Gill and Syphon Withdrawal Reflex

tactile stimulation is repeated over a number of training sessions the habituation will be mostly permanent.

Investigation into the habituation response revealed that repeated stimulation caused the post synaptic potential in the gill motor neuron to decease. This reduction was found to result from a decrease in the amount of neuron transmitter released into the synaptic gap. This in turn, appears to result from a decrease in the amount of calcium ion entry into the cell during the action potential, weakening it. Training appears to temporarily reduce the number of ion channels available for ion transport during the action potential. If training continues the reduction becomes permanent.

5.3.3.3 Sensitization

Sensitization is the increase of a behavioral reflex to a stimulus that follows an intense or noxious stimulus. In the Aplysia, a strong tail shock given to the tail will enhance the gill and syphon withdrawal reflex, despite previous habituation.

Application of such a noxious stimulus activates a number of facilitator interneurons that synapse on the sensory neurons. Activation of these neurons, and the subsequent

release of certain neural transmitters, results in the closure of specific ion channels in the cell membrane associated with the repolarization of the membrane after an action potential. The overall effect is that the action potentials are lengthened, and a greater release of neural transmitter results.

5.3.3.4 Classical Conditioning

Classical conditioning involves the association of a Conditioned Stimulus (CS) with an Unconditioned Stimulus (US) in the generation of a response. The classical example of classical conditioning is Pavlov's dog. Pavlov observed that when the ringing of a bell was repeatedly paired with the presentation of meat, the dog began to salivate at the sound of the bell alone. Like sensitization, classical conditioning involves the effect of one stimulus on the response to another. However in classical conditioning the two stimuli must be temporally paired. Also, in classical conditioning, there is a greater specificity the effects of various stimuli on the response [10].

No specific neural mechanism for classical conditioning was found, but I suspect that the key, at least in the Aplysia, is in the interneurons connecting the sensory neurons

and the facilitator interneurons to the motor cells. Simultaneous excitation of the interneurons by the facilitator interneurons and the sensory neurons may lead to a selective increase in the synaptic weights associated with the facilitator interneurons. It should be noted that the sensory neurons must provide excitation first (The dog will not be conditioned if the bell is rung after he receives food).

5.3.3.5 Operant Conditioning

Unlike the previously described mechanisms, operant conditioning involves associating a response with a reward (or the avoidance of noxious stimuli). Consider a mouse in his cage with a lever that releases a food pellet each time it is depressed. The mouse due to random activity, or previous experience will press the lever and receive a food pellet. After several occasions the mouse will begin to associate the pressing of the lever with receipt of food. Then each time the animal is hungry it will press the lever.

The running of a maze, and other complex goal oriented behaviors, are an extension of the above case of simple operant conditioning.

Although potential cellular mechanisms for habituation,

sensitization, and classical conditioning have been proposed, operant conditioning, the key to goal oriented learning, is not understood. We must therefore make logical extrapolations from the information we have.

5.3.3.6 Facilitation

Another effect which does not directly relate to the learning mechanisms described above, but will play a part in the model to be described later, is facilitation. It has been determined experimentally that in many cases once a neuron fires it requires a lower excitation to fire it again [9]. There are several possible mechanisms for this effect, including modification of synaptic efficacies and long term potentiation of the cell membrane [10] (a slight but long lasting depolarization of the cell membrane). This effect may last several minutes, hours, or days.

5.3.3.7 Key Points on Learning in Neural Networks

Based upon our review of the learning mechanisms above we can begin to identify the key points associated with learning in neural networks:

1. Information storage in the neuron is analog, not

digital [10]. Transmission of information from cell to cell may be quantized, but is not binary. The quantization may occur on the molecular level, when specific molecules interact. However, on the level of he cell, inputs and outputs may be thought of as graded and analog. Note that this rules out a number of prevalent models of network elements as a realistic models of the neuron.

2. Different types of learning have different cellular mechanisms [10]. Each type of learning involves a different type of plasticity. General learning (i.e. habituation, sensitization, facilitation) affects the transmission of a cell on the cellular level, mostly by modulating the duration and strength of the action potential or polarization of the cell membrane. Specific forms of learning, since they involve specific associations between stimuli, must affect the cells on the synaptic level.

5.3.4 Characteristics of Goal Oriented Behavior

Each organism has a finite, through potentially very large set of simple behaviors. Complex behaviors arise from simple behaviors much like music arises from simple, single musical notes. These complex behaviors occur in response to needs of the organism.

Based upon these points, and the empirical data for the functioning of the neuron delineated above, we now have a framework for the development of some ideas for the functioning of a neural model capable of exhibiting goal oriented behavior.

1. Sensory perceptions correspond to the transmission of information via neural signals. Some authors have equated firing rates to the strengths of the signalling[17]. However, as demonstrated by habituation and sensitization, this is not always the case. In fact, signalling strengths are modulated by a variety of mechanisms of which firing rate is only one.

2. Need acts as a modulation to neural activity[18]. A mouse will only search a maze if it has some form of motivation to do so.

3. Several different types of learning (plasticity), probably associated with modification of synaptic efficacies, are necessary to adequately describe goal oriented learning:

---

[17] Such an assertion is made by Kohonen [5] and in some of the discussion in Rumelhart and McClelland [4].

[18] This is may own assertion, but there is a lot of evidence to support such a supposition. The reader is directed to the discussions of attention mechanisms in Kandel and Schwartz [9] and Grossberg [7].

a. Short term plasticity which allows separate behaviors to be rapidly tried in some sequence. Consider our mouse roaming through his maze. At a particular juncture he has to make a decision to go either left or right. He decides to go left and immediately realizes that he has gone down the wrong path. He then returns to the juncture and is faced again with the decision to go left or right. Based on his recent experience with the dead end, he decides to go right, and continues his search. This corresponds to the recording and activation of a short-term memory trace.

b. Long-term plasticity which allows the recording of long-term memory. Short term memory tends to decay rapidly with time. Our mouse, on the second running of his maze, may not recall his previous experience with the juncture, and he may again make the wrong decision. However, if he runs the maze several more times, he will get to the point where he no longer err at the juncture. This corresponds to the recording of a long-term memory trace.

c. Sensitivity to Reward Both long-term and short-term memory tend to decay with time, but at different rates. However, those memory traces associated with the gratification of a need are reinforced. Our mouse, when he

enters the maze, will execute a series of behaviors which depend on the functioning of his long-term and short-term memory. Sequences that result in the discovery of the goal box are more likely to be remembered. On a microscopic level, behaviors are associated with perceptions by the synaptic efficacies. Each synaptic weight associated with a particular sequence of behaviors becomes sensitive to the whether a need is later gratified[19].

One important aside to this discussion is that forgetfulness has a useful biological function. Even within a brain as complex as our own, only a finite though large amount of information can be stored. Forgetfulness tends to clear the slate of information which is not particularly useful. This allows an organism to use resources that would otherwise be committed.

4. In the discussion above, it was proposed that the output of a neuron is a graded response, somewhere between zero and some saturation value. This leads to an interesting problem from the standpoint of designing a neural network to

---

[19] The existence of two different types of memory, long-term and short-term, is widely accepted. Typically in Grossberg's developments (see Grossberg [7], Adaptive Resonance) he models both long-term and short-term modification of synaptic weights. He has not, to my knowledge, modeled anything like the sensitivity effect I am describing here.

implement goal oriented behavior. The neurons in the network all have nonzero inputs, and as a result, all have nonzero outputs. The question is how does an organism decide on which behavior to try given that the output of several neurons may be active. A mouse cannot go forward and backward at the same time. Therefore there must be mechanism for arbitration between neurons[20].

5.4 A Model for Goal Orient Learning in a Neural Network

5.4.1 Scooter's World View

One of the problems that we must face in the design of our hypothetical mouse is how he perceives his environment, specifically, how the inputs are provided to the organism's decision making box. Our mouse is a fairly sophisticated fellow, so he will be able to pick up on a wide variety of sensory perceptions that he can feed into his decision box. If the sensory information received is different for each section of the maze, then the mouse will have a means of differentiating between the various maze sections. (We should note here that perceptions in a mammalian brain are the result

---

[20] I would like to say I anticipated this problem and took it into account in my original analysis and simulations. The truth is that I discovered the problem only after I had developed a simulation which produced nonsense.

of a great deal of processing, and that we are giving a broad brush to a great deal of this complexity in order to arrive at a simple model [9]). Fortunately, in the investigation of this thesis, perception can be handled in a relatively simple manner.

In keeping with our mouse/maze analogy, consider the simple maze of Figure 4. The maze has five different sections numbered P1 though P5. The mouse's starting position is section P1, while the goal box is in P4. Our mouse has a variety of behaviors. He can move forward (F), backward (B), to the left (L) or to the right (R). When he behaves he changes his position in the maze and therefore his perceptions. Our mouse's relationship to his environment can be thought of in terms of a Moore or state machine. Each position in the maze can be thought of as a state, and each behavior results in a state transition. The state transition diagram for the maze of Figure 4 is shown in Figure 5. To see how this works consider that the mouse is in his starting position at P1. The decision box decides that the mouse should move backwards. When the mouse attempts this maneuver he will go nowhere and hence behavior B will result

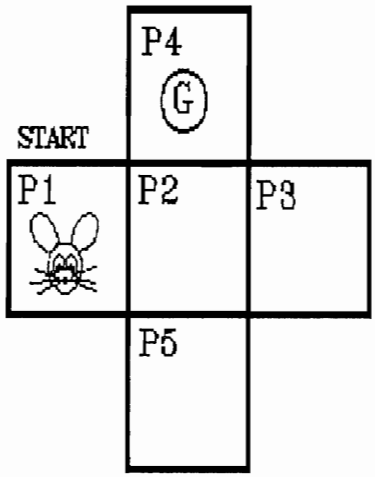

Figure 4 - Scooter's Maze

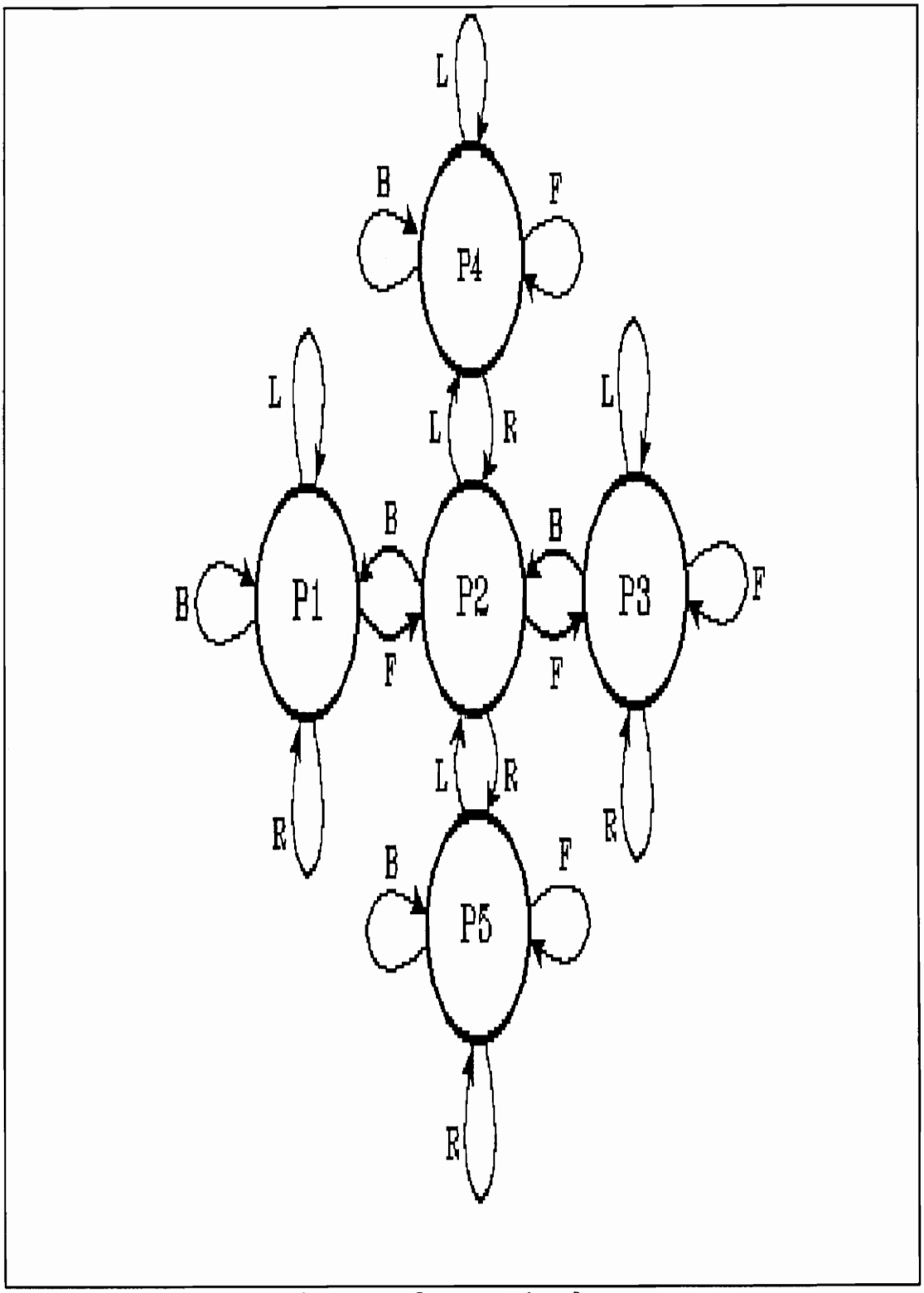

Figure 5 - State Diagram for a Simple Maze

in a transition back to state P1[21].

This is obviously a simplistic approach, but it will provide a necessary resource to test the network model to follow.

5.4.2 A Network Model

5.4.2.1 The Network Transfer Function

The structure of the cerebral cortex (the location where actual implementation of goal oriented learning occurs) is extremely complex[22]. Like most neural networks in the brain, the structures in the cerebral cortex are essentially two dimensional layers of processing units (cells or cellular modules) in which the units are densely interconnected through lateral feedback [5]. A simplified representation of such a structure is shown in Figure 6.

The rules that govern the interaction of neurons are also extremely complex. We are only beginning to understand the

---

[21] This sounds simple, but it took me awhile to figure out.

[22] Volume 2, Chapter 20 of Rumelhart and McClelland provides a good overview of the basic functioning of the neuron and the structure of the cerebral cortex.

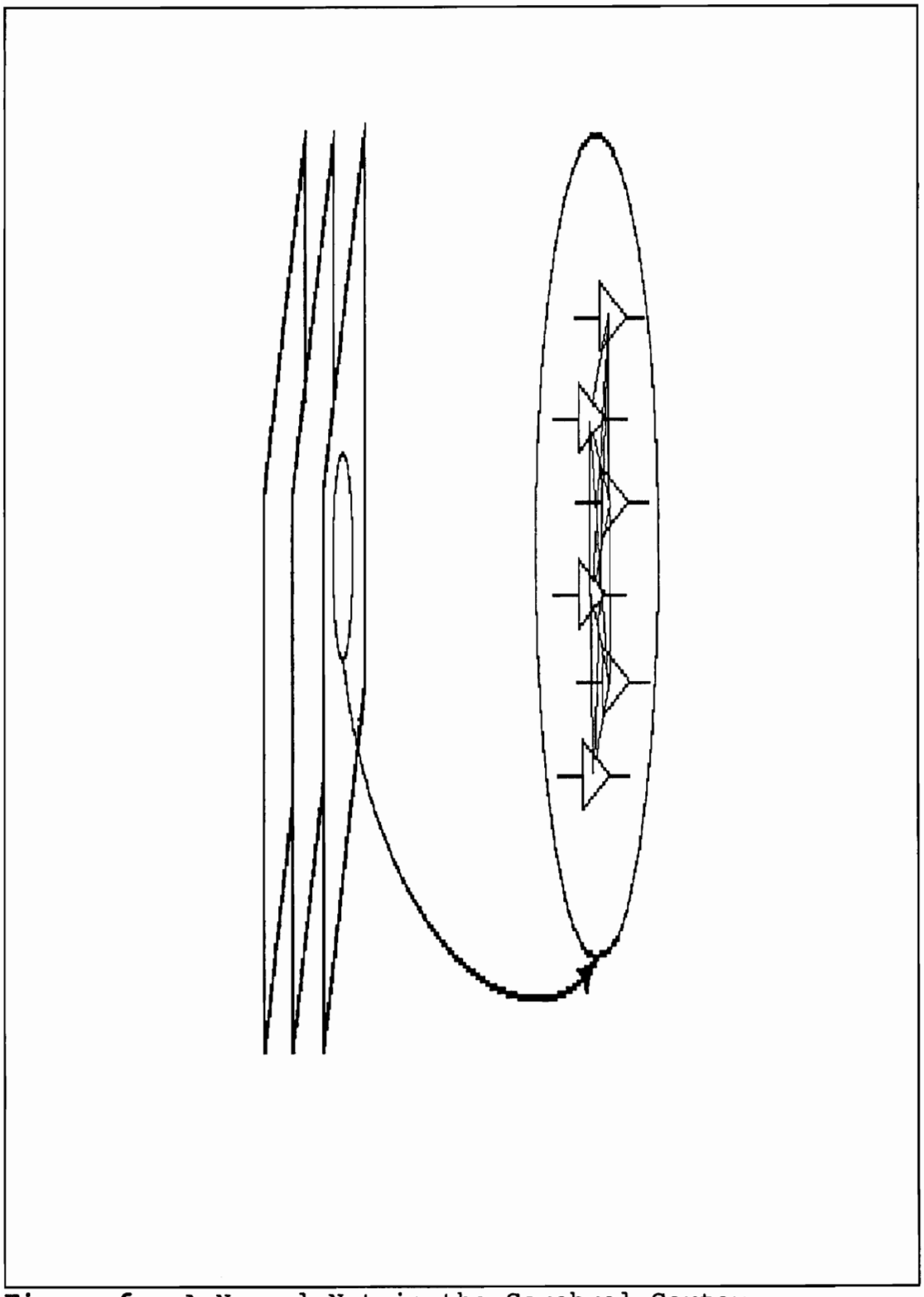

Figure 6 - A Neural Net in the Cerebral Cortex

key features of how they operate. There is a huge amount of information about the chemistry of how neurons actually operate, but much is not known. The equation below describes the output of a neuron, in location (i, j), in layer L.

This equation represents a plausible guess[23].

$$O_{L,i,j} = g\left[\sum_{i=1}^{N}\sum_{j=1}^{M} w_{i,j}O_{L-1,i,j}\right] + h\left[\sum_{i=i}^{N}\sum_{j=1}^{M} f_{i,j}O_{L,i,j}\right] \quad (1)$$

$O_{L,i,j}$ is the output of neuron i, j in layer L.

The first term describes the input of the neuron from the previous layer.

$w_{i,j}$ is the weight assigned to the output of neuron $O_{L-1,i,j}$.

---

[23] In a neurological sense this equation states that the output of a particular neuron is a graded response dependent on the weighted integration of the inputs from the dendrites. The groundwork for this assumption was laid in the previous section. This is a typical assumption prevalent in artificial neural network research. The reader is directed to the references for further information. Although not explicitly stated, the factors in the equation below are all functions of time.

The weighted outputs are summed and the function g[] is used to determine the previous layer's contribution to the output of the neuron (L, i, j).

The second term describes the input to the neuron from feedback from the neurons in the same layer.

$f_{i,j}$ is the weight associated with the output of neuron (L, i, j).

Note that this model is basically feed-forward between layers. There is no feedback, or back-propagation between layers[24]. Note also that there is feedback between the input and output of the neurons in a layer. This will allow us to model the facilitation effect. (Actually such an affect can result from positive feedback among a small local neighborhood of neurons. See Kohonen [5]).

Equation (1) is somewhat ungainly, and it can be greatly simplified, at least notationally, if we restrict ourselves to one layer and limit this layer to a single dimension.

---

[24] One of the models currently prevalent is the back-propagation or generalized delta rule. In this model a great deal of error information must propagate backwards through the network. From a physical perspective in a biological network, such propagation seems implausible.

Equation (1) becomes:

$$O_i = g[\sum_{j=1}^{N} w_j X_j] + h[\sum_{k=1}^{M} f_k O_k] \qquad (2)$$

Again the first and second terms represent the input from the previous layer and the feedback from the neurons in the same layer. We can add further constraints of the network function (as a result of simulation). Let's assume the following:

1. g[ ] and h[ ] are direct transfer functions (i.e. g[x] = h[x] = x). This really more general an assumption than it appears. In the neural network models presented in the references, the output of a neuron is presumed to be proportional to the weighted sum of the inputs, at least over some linear region. We can state that g[x] = x vice g[x] = ax, where a is some positive real constant, because it would merely represent the scaling of the input weights w_j, and f_k.

2. w_j and X_j are restricted to positive real numbers. X_j would correspond to $O_{L-1,j}$ in a multilayer network. Based upon studies of mammalian brains, there is empirical evidence to support an assumption that input weights are typically positive [5].

3. f_k, $k <> i$, is equal to some α where α is a negative real number greater than -1. This implies that the adjacent cells in a layer tend to inhibit each other. Again, there is some empirical evidence to support such an assumption based on studies of mammalian brains [5]. The assumption that α is less than one is also based upon empirical evidence. Interconnections between cells in a layer tend to be weakly inhibitive, at least over some minimal distance [5].

4. f_k, $k = i$, is equal to some β where β is a positive real number less than 1. As stated earlier, this allows us to model a facilitation effect. The requirement that β is less than one is a necessary requirement for stability. If β approached one, any net positive excitation would cause the neuron's output to tend toward infinity.

Equation (2) becomes:

$$O_i = \sum_{j=1}^{N} w_{i,j}X_j + \alpha \sum_{\substack{k=1 \\ k<>i}}^{M} O_k + \beta O_i \qquad (3)$$

In short, the output from a cell tends to inhibit adjacent cells. A particular cell, when it begins firing tends to reinforce itself. This is still a seemingly complex

network model, but it is fairly easy to implement, and it has some extremely interesting properties. All the inputs, outputs and weights, with the exception of the feedback weights are functions of time.

Expressing equation (3) as a function of discrete time, n, we obtain:

$$O_i(n+1) = \sum_{j=1}^{N} w(n)_{i,j} X_j(n) + \alpha \sum_{\substack{k=1 \\ k<>i}}^{M} O_k(n) + \beta O_i(n) \qquad (4)$$

If the parameters of the above equation are chosen within some fairly board constraints, the output of the network layers will converge such that only one neuron in the layer is active. The remaining active neuron will be that neuron with the largest sum of weighted inputs resulting from the X_js. This is an extremely important result. It describes how a network can arbitrate between the outputs of a number of neurons. Here is a mechanism by which an organism can give a clear, unambiguous response to a particular set of stimuli.

The Figure 7 shows the results of a simulation of a five neuron neural net, with $\alpha = -0.1$, and $\beta = 0.9$. The sums of

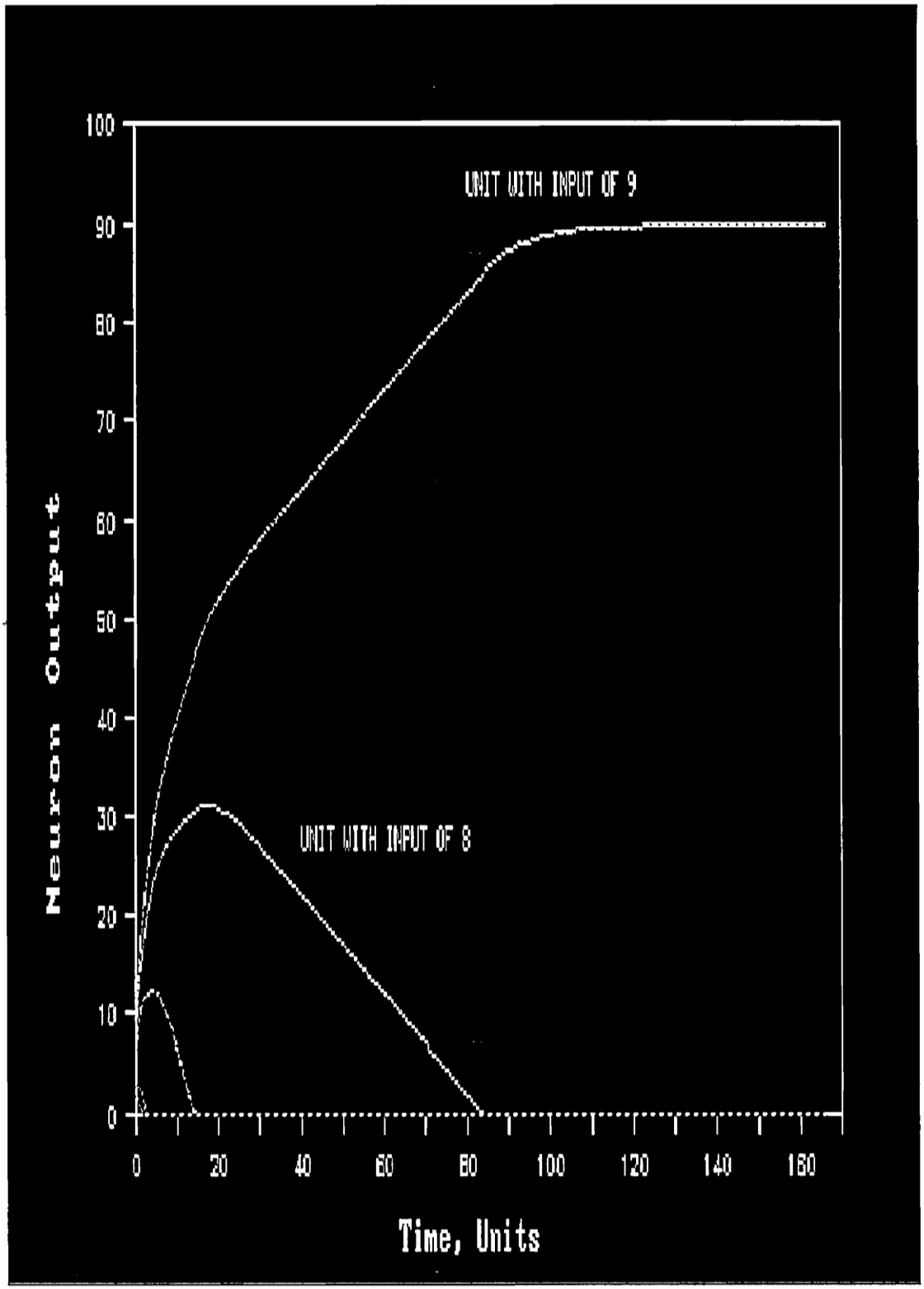

Figure 7 - Output of a Network to Arbitrary Input

the weighted inputs for each neuron are 8, 3, 6, 2, and 9 (a random selection). As the reader can see, the cell with the input of nine eventually drives the output of the other cells to zero.

Kohonen [5] proposes a similar, but somewhat more complex two-dimensional scheme that leads to basically leads to the same result. However, Kohonen, to my knowledge has not applied his scheme to a model of goal oriented learning.

5.4.2.2 Network Convergence

Since the function describing the output is nonlinear[25], it is not easy to identify a tidy closed form solution for the network output. We can, however, find some useful information by doing some quick analysis.

Recall Equation (3):

$$O_i(n+1) = \sum_{j=1}^{N} w(n)_{i,j} X_j(n) + \alpha \sum_{\substack{k=1 \\ k<>i}}^{M} O_k(n) + \beta O_i(n) \qquad (4)$$

[25] By the assumptions made above, the output of a particular neuron must be a positive real number. The output cannot go negative independent of the negative input the cell receives.

$$\text{Let } y(n) = \sum_{j=1}^{N} w_j(n)\, X_j(n)$$

then

$$O_i(n+1) = y(n) + \alpha \sum_{\substack{k=1 \\ k<>i}}^{M} O_k(n) + \beta O_i(n) \qquad (5)$$

Note that as a condition for convergence the second term

$$\alpha \sum_{\substack{k=1 \\ k<>i}}^{M} O_k(n)$$

must go to zero. (The output of all other neurons in the layer will be driven to zero).

Assuming y(n) is some constant, Equation (5) becomes:

$$O_i(n+1) = y + \beta O_i(n) \qquad (6)$$

Using standard Z-transform techniques, $O_i(n)$ can be expressed as follows:

$$O_i(n) = y \, \frac{[1 - \beta^n]}{1 - \beta} \qquad (7)$$

Equation (7) obviously does not hold while the network is starting to converge, but it does apply when all other outputs in the network have become inactive. The output of the network after convergence is given by:

$$O_i = \lim_{n \to \infty} y \frac{[1 - \beta^n]}{1 - \beta}$$

$$= \frac{y}{1 - \beta} \qquad (8)$$

Using the same parameters used in the simulation that generated Figure 13, we obtain:

$$O_i = \frac{9}{1 - 0.9} = 90$$

This corresponds to what is shown in the graph[26].

Another requirement for convergence is that the one neuron that remains active must have a sufficiently active output to drive the other neurons inactive.

---

[26] Equation 8 can also be obtained from Equation (3) and some simple algebra, but I believe Equation 8 gives us a little more information on how the output of the neuron will converge.

Hence:

$$\|\alpha O_i\| >= y_k, \; k <> i \tag{9}$$

In other words, the feedback from the active neuron must be greater than or equal to the sum of the weighted inputs to neuron k. Carrying through the calculation:

$$O_i >= \frac{y_k}{\|\alpha\|} \tag{10}$$

Combining Equations (8) and (10):

$$\frac{y_i}{1 - \beta} >= \frac{y_k}{\|\alpha\|} \tag{11}$$

$$\frac{y_i}{y_k} >= \frac{1 - \beta}{\|\alpha\|}$$

Equation (11) effectively establishes the resolution with which the outputs can be differentiated. For convergence, the left side of the equation must be greater than or equal to the right side. For example, if we wished to differentiate between a $y_i = 11$ and a $y_k = 10$, then the right side of the equation must be less than or equal to 1.1. In the type of network we want to implement, we always want to differentiate

between a y_k and a larger y_i, so we must take the limiting case of $y_i/y_k = 1$. We then obtain the relation below:

$$1 >= \frac{1 - \beta}{\|\alpha\|} \quad \text{or}$$

$$\|\alpha\| >= 1 - \beta \tag{12}$$

This is not a particularly rigid constraint.

The next question that must be answered is "What happens if the inputs change?" The answer, as shown in Figure 8, is that the network will dynamically change its output. It will again converge such that the neuron with the most excitation will be the only one remaining active. What we have, in a sense, is a device that dynamically selects the neuron with the maximum input. In a mathematical sense, the network selects the local maximum of the input function.

5.4.2.3 Learning in the Network

Again, after some simulations, there are three effects that must be modeled in a neural network:

a. Short Term change in input weights.

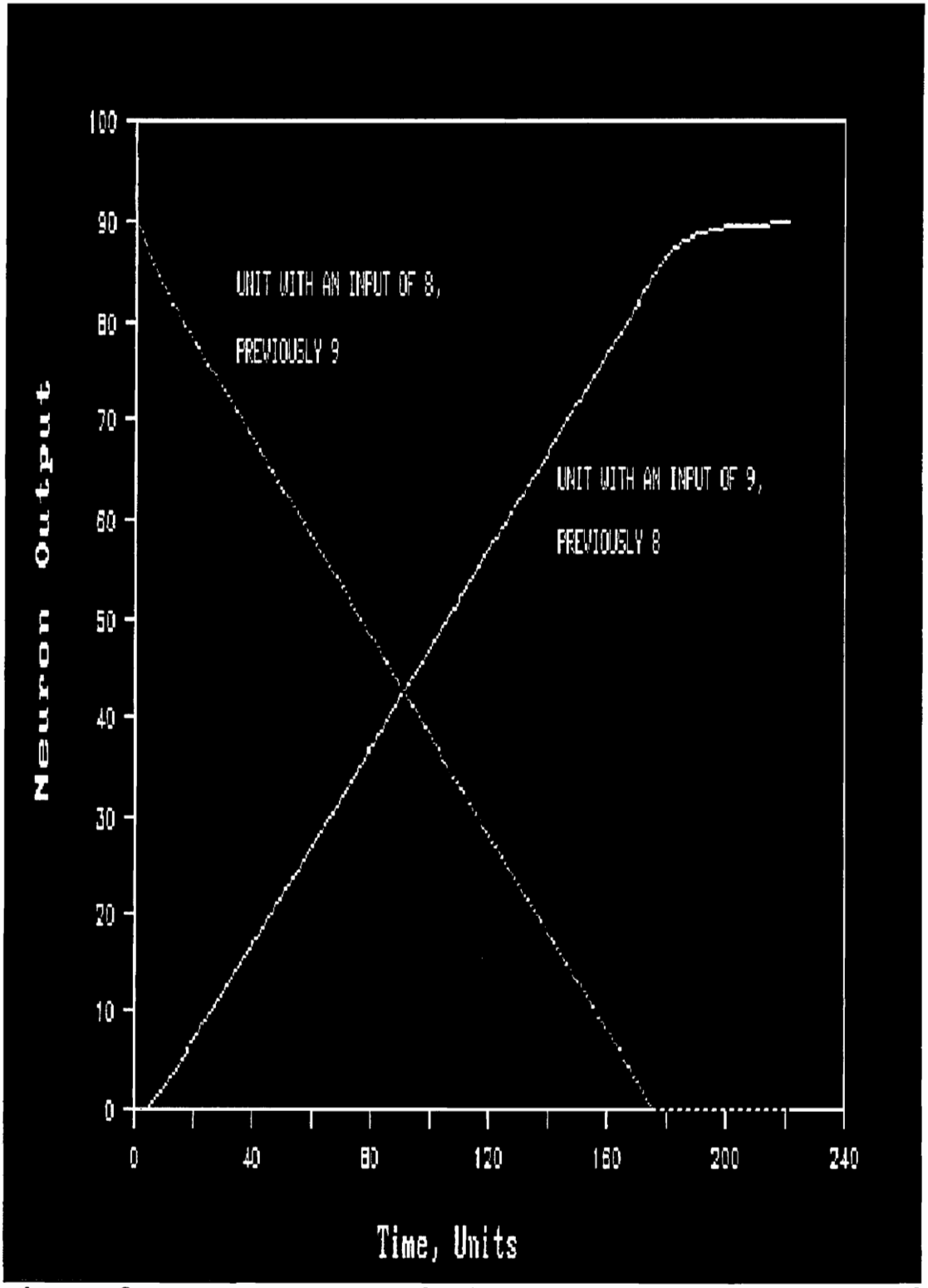

Figure 8 - Convergence of the Network after Change of Input

b. Long Term Change in input weights.

c. Reward - Increase of both short term and long term input weights as a result of a successful output.

The best way to explain how these effects are interrelated is by example. As in the previous discussion, let us limit the network to one layer. This discussion, as above will revolve around the hypothetical mouse, Scooter. The network will represent Scooter's brain.

Let each location in the maze correspond to one of Scooter's perceptions. All the perceptions are interconnected with each of the neurons in Scooter's one layer brain, and an active perception is represented by a value of 1. Scooter's maze is shown in Figure 9.

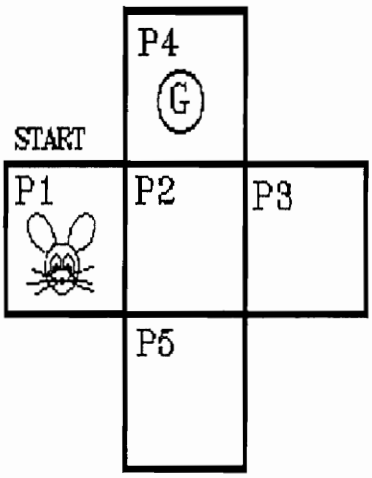

Figure 9 - Scooter's Maze

Scooter, being hungry, wants to find the Goal Box. We put him at P1[27]. The neuron with the greatest weight associated with P1 will become active and suppress all other output, and one of Scooter's behaviors will be illicited. If the behavior is not immediately successful, the short term weight associated with the active input will be reduced. At some point, some other neuron will have a greater input and it will take control of the network. A new behavior will be illicited. Eventually, some behavior will be successful in moving Scooter into the next section of the Maze.

The process will continue, step by step. Scooter will investigate his maze until he finds the Goal Box, at which point he will be rewarded. Modification of short term weights allow rapid selection of one behavior after another. After a period of inactivity the short term weight recover, approaching the long term weight. However, a long term weight associated with an unsuccessful behavior will decay as well. This process is somewhat complex, but it follows how some nerve cells appear to operate in situ. Slow decay of long term weights prevent unnecessary inhibition of behaviors that will later be rewarded.

---

[27] Modelling need as motivation will be described later in this section.

Reward is even a bit more complicated. Active inputs, successful in eliciting a behavior, will be strengthened (their weights will be increased) as a function of how long ago the behavior occurred. Behaviors which occurred a short time ago will be rewarded more, those occurring a long time ago will be rewarded much less.

The equations for learning are as follows:

Define a correlation function for neuron i, input j.

$$c_{i,j}(n) = x_{i,j}(n)O_i(n) \qquad (13)$$

or simply the input times the input.

Modification in Short Term Weights

If $c_{i,j}(n) <> 0$ (i.e. either the input is not active, the output is not active, or both)

$$w_{ST\ i,j}(n+1) = w_{ST\ i,j}(n) * (1 - \Gamma)$$

Note that this is a first order Taylor approximation

of $w_{ST\ i,j}(t) = w_{ST\ i,j}(0)\ e^{-\Gamma t}$.

If $c_{i,j}(n) = 0$

$$w_{ST\ i,j}(n+1) = w_{ST\ i,j}(n) + [w_{LT\ i,j}(n) - w_{ST\ i,j}(n)] * (1 - \Gamma)$$

Over a period of inactivity the short term weight will return to the long term weight.

Modification of Long Term Weights

If $c_{i,j}(n) = 0$

$$w_{LT\ i,j}(n+1) = w_{LT\ i,j}(n) * (1 - \delta)$$

Reward

For this aspect, we have to define a function describing the sensitivity of each input to reward:

if $c_{i,j}(n) <> 0$

$$s_{i,j}(n+1) = 1$$

The maximum sensitivity of an input occurs when both

the that input and the neurons output is active.

otherwise

$$s_{i,j}(n+1) = s_{i,j}(n) * (1 - \sigma)$$

The sensitivity of the inactive inputs decay with time.

If Scooter enters the Goal Box, all long term weights are modified as follows:

$$w_{LT\ i,j} = w_{LT\ i,j} + w_{LT\ i,j} * s_{i,j} * R$$

where R is the value of the reward.

5.4.2.4 Model Extensions

There several facets of goal oriented learning that are not directly addressed in this network model. They are, however, important to understand and consider in future work.

The first item concerns how the network responds to need. If a mouse is thoroughly feed prior to being placed in the maze, he may be more likely to sleep than hunt for a goal box.

gratified, obviously no behavior to gratify that need will be forthcoming. We can therefore assume that the activity of a group of neurons related to a certain set of behaviors will be modulated by a level of need. We can model this effect by adding a term, (MaxNeed - Need), to the network transfer function and limiting the output of the network to positive, real numbers:

$$O_{l,i,j} = g\left[\sum_{i=1}^{N} \sum_{j=1}^{M} w_{i,j} O_{l-1,i,j} \right] + h\left[\sum_{i=i}^{N} \sum_{j=1}^{M} f_{i,j} O_{l,i,j} \right] - (\text{MaxNeed} - \text{Need}) \qquad (14)$$

MaxNeed is a maximum saturation value. We can assume that Need rises as organism's needs become more pronounced with passing time (e.g. it becomes increasingly hungry), and falls to zero if the needs are gratified. We can also assume that MaxNeed, in the condition of zero need is sufficient to suppress all neural output.

A second item concerns scaling of output in multilayer layer networks. One can easily see the concern if one recalls the converge example shown in Figure 8. With a maximum value

of 9 on one of the network inputs the output will converge at a value of 90. In the context of the simple model presented above, one only need to scale the output prior to feeding into the input of the next layer. In a living neural network, I suspect that there is a dynamic ranging effect. The strengths of perceptions activating a living network probably vary over a considerable magnitude, yet the behaviors that result are typically well controlled. (If we darken the light shining into the maze, it does not cause the mouse to slow his search). Hence, there must be some network-wide control mechanism that normalizes network response with changing perception strengths.

A some point, two-dimensional, multilayer networks will be investigated. We have to change the development of the model somewhat. In a large two dimensional array of neurons, we cannot assume that each neuron equally inhibits each and every one of its neighbors. The overall affect would be that the overall network response would be driven to zero. We must therefore assume that any particular neuron provides inhibitory feedback to a limited number of its neighbors. What results from this supposition is very interesting. For a large two-dimension network input, we do not obtain a single output, but a series of activity bubbles on the two-dimensional surface of the network output layer [5]. These

bubbles would correspond to the local maxima of the network input function. As perceptions change, or learning occurs, the bubbles would move over the surface as a result of these changes.

The last item is merely a conclusion drawn from the network structure proposed herein. In a multilayer network the top most layer controls the most rudimentary behaviors, such as the control of individual muscles in an animal. Subsequent layers, as we go deeper in the network, control increasing complex composite behaviors. Learning of complex rules and the generalization of learning must occur in the deep layers of the network. Therefore future investigation and subsequent attempts to put this model to practical use should concentrate on relatively deep multilayer networks.

5.5 Results of Simulation

The model for goal oriented learning in a neural network described above was tested using simulations written in Pascal. Five increasing complex mazes were used to test the learning capacity of the simulated network. Ten experiments consisting of ten trials each were run in each maze. The results of the first three trials for the experiment are shown in Tables 1 through 5. (The results of subsequent trials are identical to the third). For each experiment, the tables list the elapse time to complete the maze and the number of moves required. The mazes in which each group of experiments was run are shown in Figures 10 through 15. Table 6 provides the summary statistics for the experiments. These are shown graphically in Figure 16.

The results can be summarized as follows:

1. In all cases, the network was able to learn the maze completely after two trials. By the third trial the mouse always goes directly to the goal box.

2. As shown in Figure 16, the average time required to complete the maze on the first trial of an experiment increases monotonically with maze complexity (the number of

moves to complete the maze). This is an expected result.

Although the mazes presented are relatively simple, by the nature of the simulation[28], it appears that the model can be extended to arbitrarily complex mazes. Hence the primary goal of this thesis, to demonstrate goal oriented learning in a neural network has been achieved.

[28] The input weights during learning decay exponentially with time, but never reach zero. The implication is that the decay, along with the corresponding selection of behaviors, could continue at infinitum, or until a reward is received.

Table 1 Results of Maze 1 Experiments (Two Moves to Complete Maze)

Experiment	Trial 1 Moves	Time	Trial 2 Moves	Time	Trial 3 Moves	Time
1	34	113	2	4	2	2
2	44	192	2	2	2	2
3	2	2	2	2	2	2
4	4	4	2	2	2	2
5	42	90	2	6	2	2
6	4	19	2	2	2	2
7	4	54	2	6	2	2
8	4	31	2	2	2	2
9	52	137	2	2	2	2
10	4	20	2	2	2	2

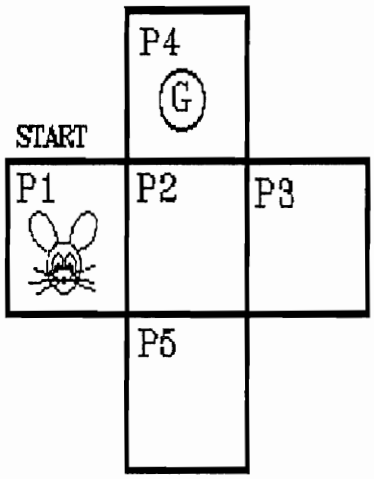

Figure 10 - Maze 1

Table 2 Results of Maze 2 Experiments (Three Moves to Complete Maze)

Experiment	Trial 1 Moves	Time	Trial 2 Moves	Time	Trial 3 Moves	Time
1	57	164	3	3	3	3
2	29	70	3	3	3	3
3	15	52	3	4	3	3
4	19	93	3	3	3	3
5	15	46	3	3	3	3
6	3	6	3	3	3	3
7	21	43	3	3	3	3
8	41	119	3	3	3	3
9	25	66	3	3	3	3
10	17	61	5	12	3	3

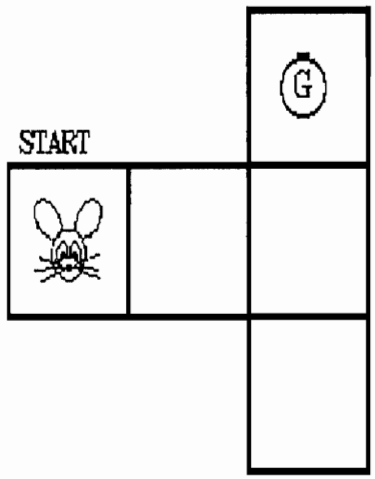

Figure 11 - Maze 2

Table 3 Results of Maze 3 Experiments (Four Moves to Complete Maze)

Experiment	Trial 1 Moves	Time	Trial 2 Moves	Time	Trial 3 Moves	Time
1	42	98	4	4	4	4
2	88	190	4	4	4	4
3	72	175	4	4	4	4
4	10	55	6	16	4	4
5	84	215	4	4	4	4
6	22	81	4	4	4	4
7	20	40	4	8	4	4
8	26	63	6	12	4	4
9	12	40	4	4	4	4
10	20	35	4	4	4	4

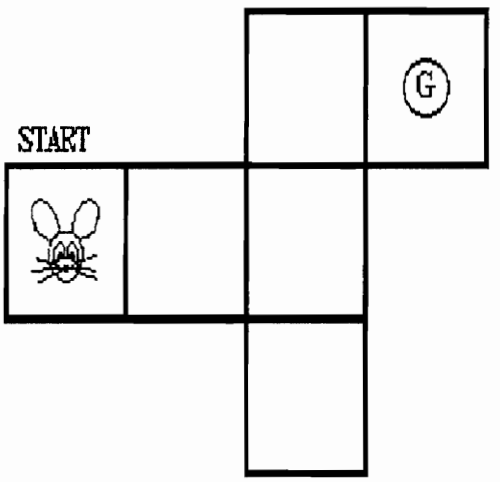

Figure 12 - Maze 3

Table 4 Results of Maze 4 Experiments (Five Moves to Complete Maze)

Experiment	Trial 1 Moves	Time	Trial 2 Moves	Time	Trial 3 Moves	Time
1	7	14	5	5	5	5
2	69	130	5	10	5	5
3	117	275	5	6	5	5
4	15	30	7	8	5	5
5	115	265	5	5	5	5
6	9	26	5	6	5	5
7	41	106	5	6	5	5
8	7	21	5	6	5	5
9	23	73	5	13	5	5
10	75	119	5	6	5	5

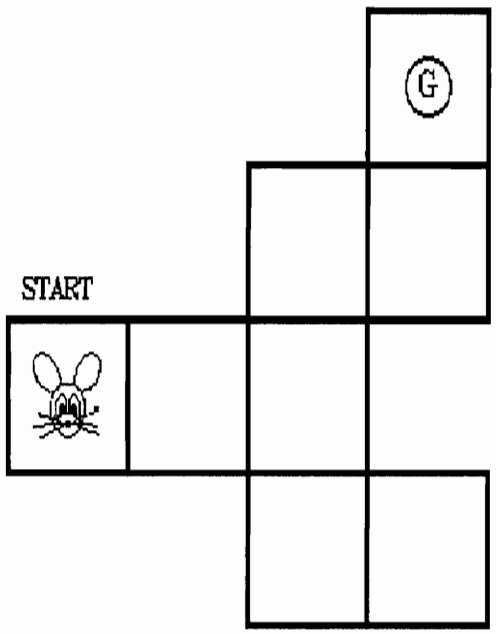

Figure 13 - Maze 4

Table 5 Results of Maze 5 Experiments (Six Moves to Complete Maze)

Experiment	Trial 1 Moves	Time	Trial 2 Moves	Time	Trial 3 Moves	Time
1	158	340	6	6	6	6
2	80	149	6	10	6	6
3	58	89	6	6	6	6
4	232	497	8	14	6	6
5	178	338	6	6	6	6
6	8	18	6	7	6	6
7	10	12	6	6	6	6
8	6	8	6	6	6	6
9	380	793	6	6	6	6
10	132	316	6	8	6	6

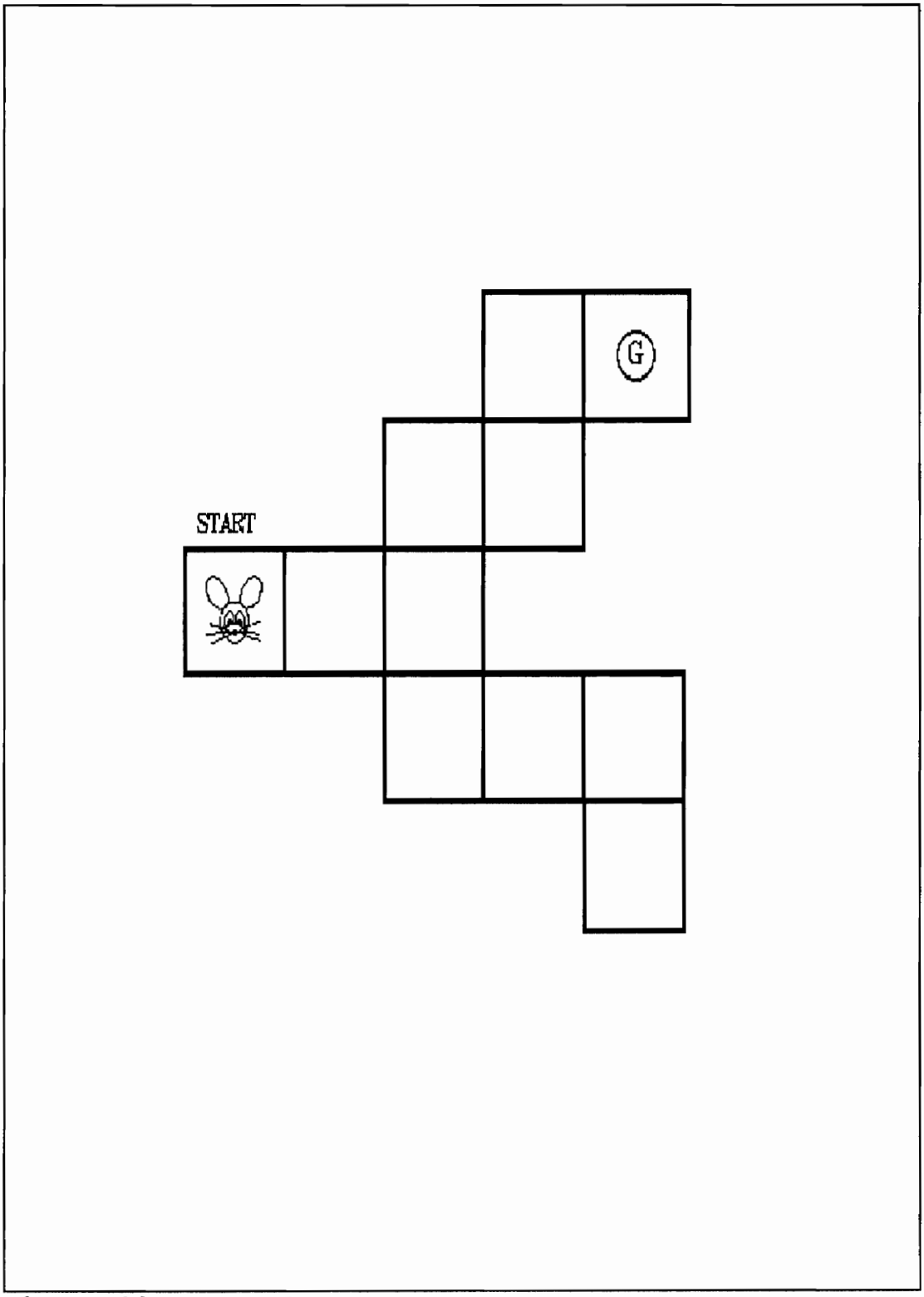

Figure 14 - Maze 5

Table 6 Summary Statistics

Maze Complexity (Number of Moves)	Average Number of Moves to Complete (First Trial)	Average Time to Complete Maze (First Trial)
2	19.4	66.2
3	24.2	72.0
4	39.6	99.2
5	47.8	104.0
6	124.2	256.0

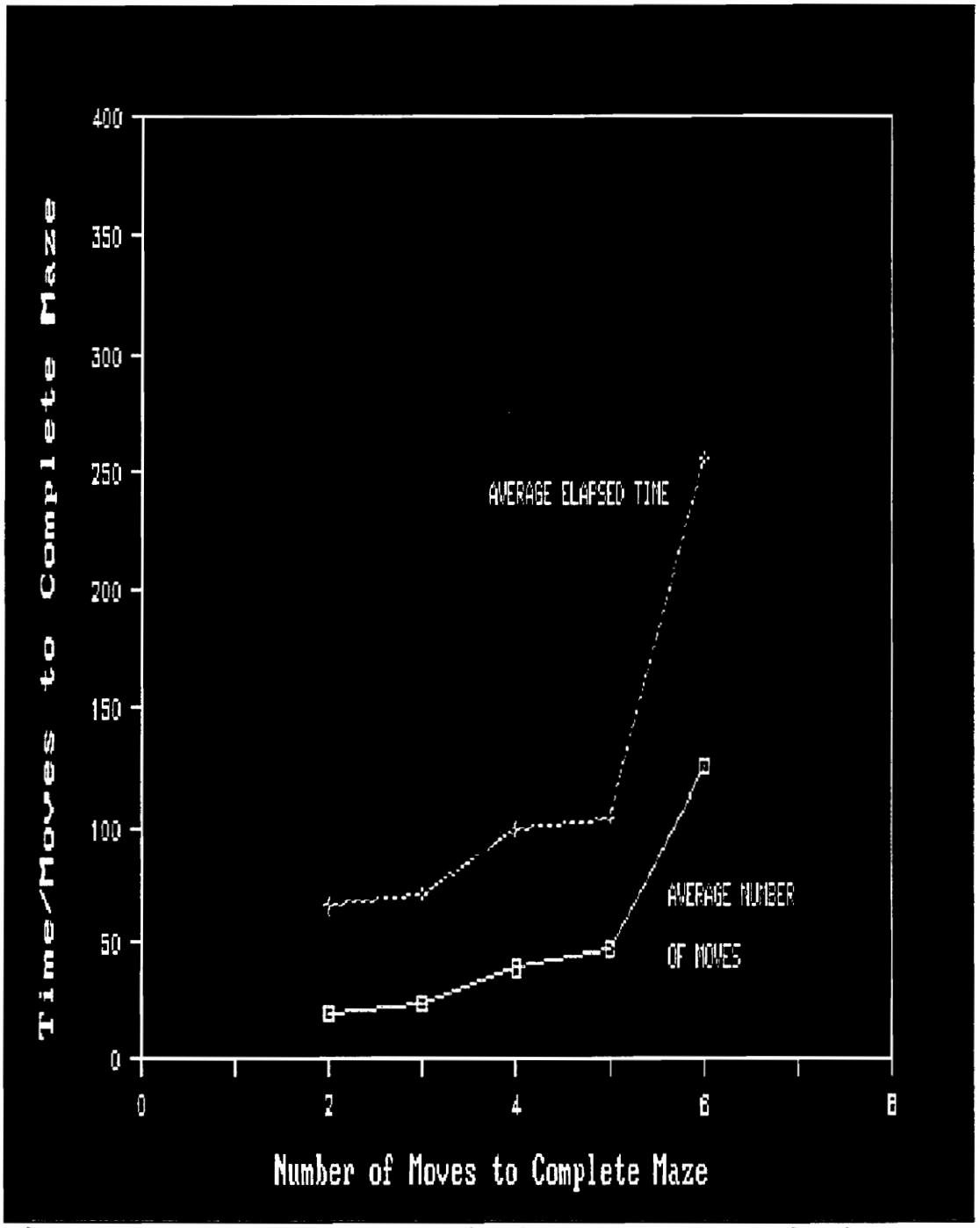

Figure 15 - Summary Statistics, Maze Complexity vs. Completion Time

6. Conclusions

The importance of this work is not that it was shown that a simulated mouse is able to run through a simulated maze. There are much simpler ways to do such a simulation without resorting to a neural network implementation. The one thing that is important about this thesis is that it represents a shift of paradigm in artificial intelligence from machines that run algorithms to those that create them. This is the real goal of artificial intelligence work.

As to the model presented herein, it is simple, consistent with the biological processes known to occur, and powerful. If one studies the model in some depth, one can see that it can be extended to arbitrarily complex mazes. In simulation the only restriction is that the short term and long term weights will become increasing small, but these problem can be eliminated if these weights are normalized each time the network output is calculated[26].

---

[26] Such a process occurs in the receptor cells in the eye. Its dynamic range (the levels of light intensity it can differentiate) is relatively small and centered at the mean light level. This ability to shift its dynamic range allows the eye to operate over huge range of mean light intensity.

This is pretty impressive for a device composed of four simple computational elements.

There are a number of areas for future work:

1. Extension of the model to multiple layers. The one layer model presented in this paper is capable of learning only one maze at a time, and most information regarding previous mazes is lost. Multiple layers are necessary to allow learning of multiple mazes and the generalization of that learning.

2. Composite Behaviors. Complex behaviors are made up of simpler behaviors which are made up of still simpler behaviors. The model presented herein shows that such composite behaviors can develop from simpler behaviors, but to a limited degree. It may be that this is also a multilayer problem. One can envision a neural network in which simple behaviors are mediated by the neurons closest to the network output. Increasingly complex behaviors would be mediated by the neurons in deeper layers.

3. Perception. This is a problem that was treated only in passing in this paper, but represents one of the

major problems associated with true artificial intelligence.

Last but not least is the development of practical applications. There are a number that come quickly to mind: machines that learn to separate parts in an assembly line; machines that independently search out and detect valuable mineral deposits on the sea bottom; machines that separate waste and hence make recycling practicable and profitable. The reader can do doubt think of others.

This paper obviously does not represent the discovery of the Holy Grail, but it outlines a possible area of search. Overall the prospects, I believe, are exciting.

Summary

A mathematical model for goal oriented learning in a network of neuron-like elements was developed. Using a mouse/goal box analogy, a simulation of a network with four elements was programmed in Turbo Pascal, Version 4.0 (Borland International) to test the model. Each location in the network corresponded to a particular network input. The output of the network consisted of one of four behaviors: forward, backward, left or right. The network successfully learned sequences of up to six movements in increasingly complex mazes.

Bibliography

1. Kai Hwang and Faye A. Briggs. Computer Architecture and Parallel Processing, New York, New York, McGraw-Hill, 1984.

2. R.E. Matick, B.D. Moldow, C.E. Watson. "Electronic Data Processing" in Electrical Engineers Handbook, 2'nd Edition, New York, New York, McGraw-Hill, 1982.

3. M. Minsky and S. Papert. Perceptrons: An Introduction to Computational Geometry, Expanded Edition, Cambridge, Massachusetts, MIT Press, 1988.

4. D. Rumelhart and J. McClelland. Parallel Distributed Processing, Cambridge, Massachusetts, MIT Press, 1988.

5. T. Kohonen. Self-Organization and Associative Memory, 2'nd Edition, New York, New York, Springer-Verlag, 1988.

6. R. Thompson, L. Hicks, and V. Shvyrkov. Neural Mechanisms of Goal Directed Behavior and Learning, New York, New York, Academic Press, 1980.

7. S. Grossberg. Neural Networks and Natural Intelligence, Cambridge, Massachusetts, MIT Press, 1988.

8. R. Johnson and C. Brown. Cognizers, New York, New York, J. Wiley and Sons, 1988.

9. E. Kandel and J. Schwartz. Principles of Neural Science, New York, New York, Elsevier/North-Holland, 1981.

10. Davis, Newburg and Wegman. Brain Structure, Learning, and Memory, Boulder Colorado, Westview Press, 1988.

11. M. Minsky. The Society of the Mind, New York, New York, Simon & Schuster, 1988.

12. E. Rietman. Experiments in Artificial Neural Networks, Blue Ridge Summit, Pennsylvania, Tab Books, 1988.

13. J. McClelland and D. Rumelhart. Explorations in Parallel Distributed Processing, Cambridge, Massachusetts, 1988.

14. W. P. Jones and J. Hoskins. "Back-Propagation, A Generalized Delta Learning Rule", Byte, October 1987, Vol. 12, No. 11, pages 155 - 162.

15. G. M. Josin. "Neural-Network Heuristics, Three Heuristic Algorithms that Learn from Experience", Byte, October 1987, Vol. 12, No. 11, pages 183 - 192.

APPENDIX

A Program for Simulation of Goal Oriented Learning in a Neural Network

```
{
*******************************************************************************
*                                                                             *
*                                                                             *
*                          NETWORK TRIAL PROGRAM                              *
*                               (MAZE 5)                                      *
*                                                                             *
*******************************************************************************
}

program network;

uses dos, crt, printer, graph;


{** Program Definitions ************************************************}

const

   max_neurons = 4;             { Maximum number of Neurons in a layer.    }
   max_layers  = 1;             { Maximum number of Layers.                }
   max_connections = 25;        { Maximum number of Connections per Neuron.}
   max_perceptions = 11;        { Maximum number of Perceptions            }
   max_behaviors = 4;           { Maximum number of Behaviors              }

  I = -0.1;                 { Inhibition feedback constant             }
  F = 0.9;                  { Self feedback or facilitation constant   }

  ST = 0.1;                 { Short Term Learning Constant           }
  LT = 0.05;                  { Long Term Learning Constant            }
  decay = 0.2;                { Decay Constant for Sensitivity           }
  reward = 2;               { Reward of Getting in Goal Box            }
```

```
type

   { Neuron Type Structures }

   connection_type = record
                        neuron: integer;  { The location of the neuron      }
                                  { in the layer.                   }
                        layer:  integer;  { The layer in which the neuron   }
                                  { resides.                        }
                        weight: real;     { The connection weight.          }
                                          { (Short term memory)             }
                     LT_weight: real;  { Long term connection weight     }
                                          { (Long term memory)              }
                     sensitivity: real; { The sensitivity of the         }
                                    { connection to changes in need. }

                      end;

  input_array = array [1..max_connections] of connection_type;

   neuron_type = record
                    input:       input_array;
                    output:      real;
                 end;

   network_type = array [1..max_neurons, 1..max_layers] of neuron_type;

   perception_type = array [1..max_perceptions] of integer;

   behavior_type = array [1..max_behaviors] of integer;

   { Maze definitions }

   maze_type = (P1, P2, P3, P4, P5, P6, P7, P8, P9, P10,
              P11, P12, P13, P14, P15);
                { These are all positions in the maze }
```

```
{ Graphics definitions }

rect = record
        xmin, ymin, xmax, ymax: integer;
       end;

circ = record
          x, y, rad: integer;
        end;

position = record
             x, y: integer;
           end;

net_drawing_type = array [1..max_neurons, 1..max_layers] of position;

 perception_drawing_type = array [1.. max_perceptions] of position;

behavior_drawing_type = array [1..max_behaviors] of position;

maze_drawing_type = array [1..max_perceptions] of position;

buffer = ^byte;
```

```
var

   { Main Program Variables }

  maze_data: text;

  mouse_brain: network_type;
  perception:  perception_type;
  behavior:    behavior_type;
  goalbox:     boolean;
  maze:        maze_type;
  trial:      integer;
  elapse_time: integer;
  move:       boolean;
  moves:      integer;

  { Graphics Definitions }

  GraphDriver, GraphMode, ErrorCode: integer;
  p_rect, net_rect, b_rect, maze_rect: rect;
  n_drawing: net_drawing_type;
  p_drawing: perception_drawing_type;
  b_drawing: behavior_drawing_type;
  m_drawing: maze_drawing_type;

  on_p, off_p, on_n, off_n, on_b, off_b, on_m, off_m: buffer;
```

```
{** Initialize Graphics ***************************************************}


procedure initialize_graphics;

  var
     GraphDriver, GraphMode: integer;
     ErrorCode: integer;

  begin
     GraphDriver:= Detect;
     InitGraph (GraphDriver, GraphMode, '');
     ErrorCode:= GraphResult;
     if ErrorCode <> grOk then
        begin
           writeln ('Graphics Error: ', GraphErrorMsg(ErrorCode));
           writeln ('Program Aborted.');
           halt(1);
        end;
  end;
```

```
{** Layout the Display ***************************************************}

procedure Set_Up_Screen (var p_rect, net_rect, b_rect, maze_rect: rect);

  var
    MaxX, MaxY: integer;

  begin
    MaxX:= GetMaxX;
    MaxY:= GetMaxy;

    p_rect.xmin:= round(4*MaxX/80);
    p_rect.xmax:= round(8*MaxX/80);
    p_rect.ymin:= round(4*MaxY/25);
    p_rect.ymax:= round(20*MaxY/25);

    net_rect.xmin:= round(8*MaxX/80);
    net_rect.xmax:= round(32*MaxX/80);
    net_rect.ymin:= round(4*MaxY/25);
    net_rect.ymax:= round(20*MaxY/25);

    b_rect.xmin:= round(32*MaxX/80);
    b_rect.xmax:= round(36*MaxX/80);
    b_rect.ymin:= round(4*MaxY/25);
    b_rect.ymax:= round(20*MaxY/25);

    maze_rect.xmin:= round(44/80*MaxX);
    maze_rect.xmax:= round(76/80*MaxX);
    maze_rect.ymin:= round(4*MaxY/25);
    maze_rect.ymax:= round(20*MaxY/25);

  end;
```

```
{** Generate Graphics Symbols and Locations *********************************}


procedure Generate_Symbols (p_rect, net_rect, b_rect, maze_rect: rect;
                        var n_drawing: net_drawing_type;
                        var p_drawing: perception_drawing_type;
                        var b_drawing: behavior_drawing_type;
                        var m_drawing: maze_drawing_type;
                        var on_p, off_p, on_n, off_n, on_b, off_b,
                            on_m, off_m: buffer);


  var
     size, height, width, radius, I, J: integer;
     triangle: array [1..4] of PointType;

  begin

     SetColor (white);
     SetFillStyle (SolidFill, yellow);

     { Generate and store images for perceptions }

     width:=  p_rect.xmax - p_rect.xmin - 4;
     height:= (p_rect.ymax - p_rect.ymin) div max_perceptions - 4;
     radius:= round(width/3);

     size:=  imagesize (0, 0, width, height);
     GetMem (off_p, size);
     GetMem (on_p, size);

     ClearDevice;
     SetColor (white);
     circle (width div 2, height div 2, radius);
     GetImage (0, 0, width, height, off_p^);
     FloodFill (width div 2, height div 2, white);
     GetImage (0, 0, width, height, on_p^);

     for I:= 1 to max_perceptions do
        begin
           p_drawing [I].x:= p_rect.xmin + 2;
           p_drawing [I].y:= p_rect.ymin + (I-1)*height + 2;
        end;

     { Generate and store images for neurons }

     width:=  (net_rect.xmax - net_rect.xmin) div max_layers - 4;
     height:= (net_rect.ymax - net_rect.ymin) div max_neurons - 4;

     size:=  imagesize (0, 0, width, height);
     GetMem (off_n, size);
     GetMem (on_n, size);

     triangle [1].x:= width div 5;
     triangle [1].y:= height div 5;
     triangle [2].x:= width div 5;
     triangle [2].y:= 4 * height div 5;
     triangle [3].x:= 4 * width div 5;
     triangle [3].y:= height div 2;
     triangle [4].x:= width div 5;
     triangle [4].y:= height div 5;

     ClearDevice;
     DrawPoly (4, triangle);
     GetImage (0, 0, width, height, off_n^);
     FloodFill (width div 2, height div 2, white);
     GetImage (0, 0, width, height, on_n^);
```

```
for I:= 1 to max_layers do
   for J:= 1 to max_neurons do
      begin
         n_drawing [J, I].x:= net_rect.xmin + (I-1)*width + 2;
         n_drawing [J, I].y:= net_rect.ymin + (J-1)*height + 2;
      end;


{ Generate and store images for behaviors }

width:=  b_rect.xmax - b_rect.xmin - 4;
height:= (b_rect.ymax - b_rect.ymin) div max_behaviors - 4;
radius:= round(width/3);

size:=  imagesize (0, 0, width, height);
GetMem (off_b, size);
GetMem (on_b, size);

ClearDevice;
SetColor (white);
circle (width div 2, height div 2, radius);
GetImage (0, 0, width, height, off_b^);
FloodFill (width div 2, height div 2, white);
GetImage (0, 0, width, height, on_b^);

for I:= 1 to max_behaviors do
   begin
      b_drawing [I].x:= b_rect.xmin + 2;
      b_drawing [I].y:= b_rect.ymin + (I-1)*height + 2;
   end;

{ Generate and store images for the maze }

width:=  (maze_rect.xmax - maze_rect.xmin) * 3 div 15;
height:= (maze_rect.ymax - maze_rect.ymin) * 3 div 15;

size:=  imagesize (0, 0, width, height);
GetMem (off_m, size);
GetMem (on_m, size);

ClearDevice;
SetColor (white);
rectangle (0, 0, width, height);
GetImage (0, 0, width, height, off_m^);
FloodFill (width div 2, height div 2, white);
GetImage (0, 0, width, height, on_m^);

m_drawing [4].x:= maze_rect.xmin + (maze_rect.xmax - maze_rect.xmin)
                                 * 6 div 15;
m_drawing [4].y:= maze_rect.ymin + (maze_rect.ymax - maze_rect.ymin)
                                 * 3 div 15;
m_drawing [1].x:= maze_rect.xmin + (maze_rect.xmax - maze_rect.xmin)
                                 * 0 div 15;
m_drawing [1].y:= maze_rect.ymin + (maze_rect.ymax - maze_rect.ymin)
                                 * 6 div 15;
m_drawing [2].x:= maze_rect.xmin + (maze_rect.xmax - maze_rect.xmin)
                                 * 3 div 15;
m_drawing [2].y:= maze_rect.ymin + (maze_rect.ymax - maze_rect.ymin)
                                 * 6 div 15;
m_drawing [3].x:= maze_rect.xmin + (maze_rect.xmax - maze_rect.xmin)
                                 * 6 div 15;
m_drawing [3].y:= maze_rect.ymin + (maze_rect.ymax - maze_rect.ymin)
                                 * 6 div 15;
m_drawing [5].x:= maze_rect.xmin + (maze_rect.xmax - maze_rect.xmin)
                                 * 6 div 15;
m_drawing [5].y:= maze_rect.ymin + (maze_rect.ymax - maze_rect.ymin)
                                 * 9 div 15;
```

```
  m_drawing [6].x:= maze_rect.xmin + (maze_rect.xmax - maze_rect.xmin)
                                * 9 div 15;
  m_drawing [6].y:= maze_rect.ymin + (maze_rect.ymax - maze_rect.ymin)
                                * 3 div 15;
  m_drawing [7].x:= maze_rect.xmin + (maze_rect.xmax - maze_rect.xmin)
                                * 9 div 15;
  m_drawing [7].y:= maze_rect.ymin + (maze_rect.ymax - maze_rect.ymin)
                                * 0 div 15;
  m_drawing [8].x:= maze_rect.xmin + (maze_rect.xmax - maze_rect.xmin)
                                * 9 div 15;
  m_drawing [8].y:= maze_rect.ymin + (maze_rect.ymax - maze_rect.ymin)
                                * 9 div 15;
  m_drawing [9].x:= maze_rect.xmin + (maze_rect.xmax - maze_rect.xmin)
                                * 12 div 15;
  m_drawing [9].y:= maze_rect.ymin + (maze_rect.ymax - maze_rect.ymin)
                                * 0 div 15;
  m_drawing [10].x:= maze_rect.xmin + (maze_rect.xmax - maze_rect.xmin)
                                * 12 div 15;
  m_drawing [10].y:= maze_rect.ymin + (maze_rect.ymax - maze_rect.ymin)
                                * 9 div 15;
  m_drawing [11].x:= maze_rect.xmin + (maze_rect.xmax - maze_rect.xmin)
                                * 12 div 15;
  m_drawing [11].y:= maze_rect.ymin + (maze_rect.ymax - maze_rect.ymin)
                                * 12 div 15;
end;
```

```
{** Draw Screen ************************************************************}

procedure Draw_Screen (p_rect, net_rect, b_rect, maze_rect: rect);


  begin
     ClearDevice;
     setcolor (white);
     outtextxy (p_rect.xmin, p_rect.ymin - 10, 'P');
     rectangle (p_rect.xmin,   p_rect.ymin, p_rect.xmax, p_rect.ymax);
     outtextxy (net_rect.xmin, net_rect.ymin - 10, 'Scooter''s Brain');
     rectangle (net_rect.xmin, net_rect.ymin, net_rect.xmax, net_rect.ymax);
     outtextxy (b_rect.xmin,   b_rect.ymin - 10, 'B');
     rectangle (b_rect.xmin,   b_rect.ymin, b_rect.xmax, b_rect.ymax);
     outtextxy (maze_rect.xmin,  maze_rect.ymin - 10, 'Maze');
     rectangle (maze_rect.xmin,  maze_rect.ymin, maze_rect.xmax, maze_rect.ymax);
  end;
```

```
{** Initialize Connections ************************************************}

procedure initialize_connections (var mouse_brain: network_type);

  var
     layer_count, neuron_count, connection: integer;

  begin

     { Make network connections }

     randomize;

     for layer_count:= 1 to max_layers do
        for neuron_count:= 1 to max_neurons do
           begin
              mouse_brain [neuron_count, layer_count].output:= 0;

              { Clear all Connections }

              for connection:= 1 to max_connections do
                 with mouse_brain [neuron_count, layer_count].input [connection] do
                    begin
                       neuron:= 0;
                       layer:= 0;
                       weight:= 0;
                       sensitivity:= 0;
                    end;

              { Now Make Up Input Connections }

              if layer_count = 1
                 then
                    begin
                       for connection:= 1 to max_perceptions do
                          with mouse_brain [neuron_count, layer_count].input [connection] do
                             begin
                                sensitivity:= 0;
                                neuron:= connection;
                                layer:= layer_count - 1; { Zeroth layer is the perception array.}
                                weight:= 5 + 5 * random;
                                LT_weight:= weight;
                             end;
                    end
                 else
                    begin
                       for connection:= 1 to max_neurons do
                          with mouse_brain [neuron_count, layer_count].input [connection] do
                             begin
                                sensitivity:= 0;
                                neuron:= connection;
                                layer:= layer_count - 1; { Zeroth layer is the perception array.}
                                weight:= 5 + 5 * random;
                                LT_weight:= weight;
                             end;
                    end;

              { Feedback Connections }
              { These are not required - Closed form solution available }

              {

              for connection:= (max_connections - max_neurons) to max_connections do
                 with mouse_brain [neuron_count, layer_count].input [connection] do
                    begin
                       sensitivity:= 0;
```

```
                neuron:= connection - max_neurons;
                layer:= layer_count;
                if neuron = neuron_count
                  then weight:=   f
                  else weight:=   1;
              end;

          }

      end;
end;
```

```
{** Initialize Perceptions ************************************************}

procedure initialize_perceptions (var perception: perception_type);

  var
    perception_number: integer;

  begin
    perception [1]:= 1; { This is the perception that corresponds to  }
                        { position P1 of the maze.                    }

    for perception_number:= 2 to max_perceptions do
      begin
        perception [perception_number]:= 0;
      end;
  end;
```

```
{** Calculate Net ************************************************************}


procedure calculate_net (var perception: perception_type;
                         var mouse_brain: network_type);

  var
     neuron_count, layer_count, I: integer;
     sum: real;
     done: boolean;
     test: integer;
     out: string [10];
     MaxOut: real;
     MaxSum: real;
     MaxNeuron: integer;
     NumInputs: integer;


  function input (neuron, layer: integer;
                  var mouse_brain: network_type;
                  var perception: perception_type): real;

     begin
        if (layer = 0) and (neuron = 0)
           then input:= 0
           else if (layer = 0) and (neuron <> 0)
                   then input:= perception [neuron]
                   else if (layer <> 0) and (neuron <> 0)
                           then input:= mouse_brain [neuron, layer].output;
     end; { function input }



  begin { calculate_net }

     SetFillStyle (SolidFill, blue);

     for layer_count:= 1 to max_layers do

        { Calculate output of layer }

        begin
           MaxSum:= 0;

           for neuron_count:= 1 to max_neurons do
              begin
                 sum:= 0;
                 for I:= 1 to max_connections do
                   begin
                      with mouse_brain [neuron_count, layer_count].input[I] do
                      begin
                         sum:= sum + weight * input (neuron, layer,
                                mouse_brain,
                                perception);

                      end;
                   end;

                 if sum > MaxSum then
                   begin
                      MaxSum:= sum;
                      MaxNeuron:= neuron_count;
                   end;
              end;

           for neuron_count:= 1 to max_neurons do
```

```
          begin
             if neuron_count = MaxNeuron
               then mouse_brain [neuron_count, layer_count].output:=
                       MaxSum / (1 - F)
               else mouse_brain [neuron_count, layer_count].output:= 0;
          end;


        bar (24 * GetMaxX div 80, 23 * GetMaxY div 25,
             40 * GetMaxX div 80, 24 * GetMaxY div 25);
        MoveTo (10 * GetMaxX div 80, 23 * GetMaxY div 25);
        str ((MaxSum / (1-F)):7:1,out);
        OutText ('Max Output:     ' + out);

      end; { Layer Calculation }

end;  { calculate net }
```

```
{** Calculate Behaviors ***************************************************}

procedure calculate_behaviors (mouse_brain: network_type;
                               var behavior: behavior_type);

  var
     neuron: integer;

  begin
     for neuron:= 1 to max_neurons do
        begin
           if neuron <= max_perceptions
              then
                 begin
                    if mouse_brain [neuron, max_layers].output > 0
                       then behavior [neuron]:= 1
                       else behavior [neuron]:= 0;
                 end;
        end;
  end;
```

```
{** Move in Maze ************************************************************}

function move_in_maze (behavior: behavior_type;
                       var maze: maze_type;
                       var perception: perception_type;
                   var move: boolean): boolean;

  var
     I: integer;
     start: maze_type;

  begin

     start:= maze;

     case maze of

        P1:
           begin
              if (behavior [1] = 1)
                 then maze:= P2;
           end;
        P2:
           begin
              if (behavior [1] = 1)
                 then maze:= P3;
              if (behavior [2] = 1)
                 then maze:= P1;

           end;
        P3:
           begin
              if (behavior [2] = 1)
                 then maze:= P2;
              if (behavior [3] = 1)
                 then maze:= P4;
              if (behavior [4] = 1)
                 then maze:= P5;
           end;
        P4:
           begin
              if (behavior [1] = 1)
                 then maze:= P6;
              if (behavior [4] = 1)
                 then maze:= P3;
           end;
        P5:
           begin
              if (behavior [1] = 1)
                 then maze:= P8;
              if (behavior [3] = 1)
                 then maze:= P3;
           end;
        P6:
           begin
              if (behavior [2] = 1)
                 then maze:= P4;
              if (behavior [3] = 1)
                 then maze:= P7;
           end;
        P7:
           begin
              if (behavior [1] = 1)
                 then maze:= P9;
              if (behavior [4] = 1)
                 then maze:= P6;
           end;
```

```
      P8:
        begin
          if (behavior [1] = 1)
            then maze:= P10;
          if (behavior [2] = 1)
            then maze:= P5;
        end;
      P9:
        begin
          if (behavior [2] = 1)
            then maze:= P7;
        end;
      P10:
        begin
          if (behavior [2] = 1)
            then maze:= P8;
          if (behavior [4] = 1)
            then maze:= P11;
        end;
      P11:
        begin
          if (behavior [3] = 1)
            then maze:= P10;
        end;

    end;  { case }

    for I:= 1 to max_perceptions do
      perception [I]:= 0;

    case maze of
      P1: perception [1]:= 1;
      P2: perception [2]:= 1;
      P3: perception [3]:= 1;
      P4: perception [4]:= 1;
      P5: perception [5]:= 1;
      P6: perception [6]:= 1;
      P7: perception [7]:= 1;
      P8: perception [8]:= 1;
      P9: perception [9]:= 1;
      P10: perception [10]:= 1;
      P11: perception [11]:= 1;

    end;

    if maze = P9
      then move_in_maze:= true
      else move_in_maze:= false;

    if start = maze
      then move:= false
      else move:= true;

end;
```

```
{** Modify Weights ***********************************************************}

procedure modify_weights (var mouse_brain: network_type);

  var
     neuron_count, layer_count, I: integer;
     correlation: real;

  begin

     for layer_count:= 1 to max_layers do
        for neuron_count:= 1 to max_neurons do
           for I:= 1 to max_connections do

              begin

                 with mouse_brain [neuron_count, layer_count] do
                   begin

                   { First calculate if both the input and output }
                   { are active at the same time.                 }

                      if output <> 0
                        then
                          begin

                             { This is for multilayer networks }

                             if input [I].layer <> 0
                                then
                                   correlation:= output *
                                     mouse_brain [input[I].neuron,input[I].layer].output;

                             { This is for the first layer }

                             if (input [I].layer = 0) and
                                (input [I].neuron <= max_perceptions) and
                                (input [I].neuron > 0)

                                   then correlation:= output *
                                          perception [input[I].neuron];
                          end
                        else correlation:= 0;

                      { Now calculate the sensitivity to changes in need. }

                      if correlation <> 0
                        then input [I].sensitivity:= 1
                        else input [I].sensitivity:= input[I].sensitivity *
                                                     (1 - decay);

                      { Calculate changes in input weights. }

                      { "Punish" unsuccessful behaviors. }

                      if correlation <> 0
                        then
                          begin
                             with input [I] do
                                begin
                                   LT_weight:= LT_weight * (1 - LT);
                                   weight:= weight * (1 - ST)
                                end;
                          end;

                      { Let inactive inputs recover. }
```

```
                if (correlation = 0)
                   then
                      begin
                         with input [1] do
                            begin
                               weight:= weight + (LT_weight - weight) *
                                       (1-ST);
                            end;
                      end;


             end;

          end;
end;
```

```
{** Food Pellet ************************************************************}


procedure food_pellet (goalbox: boolean;
                  var mouse_brain: network_type);

  var
    neuron_count, layer_count, I: integer;

  begin
    if not goalbox then exit;
    for layer_count:= 1 to max_layers do
      for neuron_count:= 1 to max_neurons do
        for I:= 1 to max_connections do

          begin
            with mouse_brain [neuron_count, layer_count] do
              begin
                input [I].LT_weight:= input [I].LT_weight +
                                      input [I].LT_weight *
                                      input [I].sensitivity * reward;
                if input [I].LT_weight > 500
                  then input [I].LT_weight:= 500;
                    input [I].weight:= input[I].LT_weight;
              end;
          end;
  end;
```

```
{** Show Net ***********************************************************}

procedure Show_Net (mouse_brain: network_type;
                    n_drawing: net_drawing_type;
                    on_n, off_n: buffer);

  var
    I, J: integer;

  begin
    for I:= 1 to max_layers do
      for J:= 1 to max_neurons do
        with mouse_brain [J, I] do
          begin
            if output = 0
              then
                begin
                  PutImage (n_drawing [J, I].x, n_drawing [J, I].y,
                            off_n^, NormalPut);
                end
              else
                begin
                  PutImage (n_drawing [J, I].x, n_drawing [J, I].y,
                            on_n^, NormalPut);
                end;
          end;
  end;
```

```
{** Show Perceptions ******************************************************}

procedure Show_Perceptions (perception: perception_type;
                        p_drawing: perception_drawing_type;
                        on_p, off_p: buffer);

  var
    I: integer;

  begin
    for I:= 1 to max_perceptions do
      begin
        if perception [I] = 0
          then
            begin
              PutImage (p_drawing [I].x, p_drawing [I].y,
                        off_p^, NormalPut);
            end
          else
            begin
              PutImage (p_drawing [I].x, p_drawing [I].y,
                        on_p^, NormalPut);
            end;
      end;
  end;
```

```
{** Show Behaviors ***********************************************************}

procedure Show_Behaviors (behavior: behavior_type;
                          b_drawing: behavior_drawing_type;
                          on_b, off_b: buffer);

  var
    I: integer;

  begin
    for I:= 1 to max_behaviors do
      begin
        if behavior [I] = 0
          then
            begin
              PutImage (b_drawing [I].x, b_drawing [I].y,
                        off_b^, NormalPut);
            end
          else
            begin
              PutImage (b_drawing [I].x, b_drawing [I].y,
                        on_b^, NormalPut);
            end;
      end;
  end;
```

```
{** Show Maze ************************************************************}

procedure Show_Maze        (perception: perception_type;
                        m_drawing: maze_drawing_type;
                        on_m, off_m: buffer);

  var
    I: integer;

  begin
    for I:= 1 to max_perceptions do
      begin
        if perception [I] = 0
          then
            begin
              PutImage (m_drawing [I].x, m_drawing [I].y,
                        off_m^, NormalPut);
            end
          else
            begin
              PutImage (m_drawing [I].x, m_drawing [I].y,
                        on_m^, NormalPut);
            end;
      end;
  end;
```

```
{** Main Routine ************************************************************}

begin
  assign (maze_data, 'C:\THESIS\NETWORK\MAZE.PRN');
  rewrite (maze_data);

  initialize_graphics;
  setbkcolor (blue);
  Set_Up_Screen (p_rect, net_rect, b_rect, maze_rect);
  Generate_Symbols (p_rect, net_rect, b_rect, maze_rect,
                n_drawing, p_drawing, b_drawing, m_drawing, on_p,
                off_p, on_n, off_n, on_b, off_b, on_m, off_m);
  Draw_Screen (p_rect, net_rect, b_rect, maze_rect);

  initialize_connections (mouse_brain); { Set up the network connections  }

  trial:= 0;
  repeat

     trial:= trial + 1;                    { Increment the number of trials }
     moves:= 0;
     elapse_time:= 0;                    { Reset timer }

     initialize_perceptions (perception);  { Place mouse at a start }
     goalbox:= false;                    { The mouse does not start off }
                                         { in the goal box. }
     maze:= P1;                          { Start the mouse off at P1. }

     Show_Perceptions (perception, p_drawing, on_p, off_p);
     Show_Net (mouse_brain, n_drawing, on_n, off_n);
     Show_Behaviors (behavior, b_drawing, on_b, off_b);
     Show_Maze (perception, m_drawing, on_m, off_m);

      repeat

       { Update the clock }

       elapse_time:= elapse_time + 1;

       { Show what the mouse sees }

       Show_Perceptions (perception, p_drawing, on_p, off_p);

       { Calculate the output of the network }

       calculate_net (perception, mouse_brain); { Calculate the output }
       Show_Net (mouse_brain, n_drawing, on_n, off_n);
       modify_weights (mouse_brain);

       { Based on the network output calculate the behaviors }

       calculate_behaviors (mouse_brain, behavior);
       Show_Behaviors (behavior, b_drawing, on_b, off_b);

       { Move the mouse in the maze }

       goalbox:= move_in_maze (behavior, maze, perception, move);
       Show_Maze (perception, m_drawing, on_m, off_m);
       if move then moves:= moves + 1;

       { If it is in the goal box, give ot a reward }

       food_pellet (goalbox, mouse_brain);

       if keypressed then exit;
     until goalbox;
```

```
    writeln (maze_data, trial, '  ', moves, '  ', elapse_time);
  until trial = 10;
  close (maze_data);
  CloseGraph;
end.
```

The vita has been removed from the scanned document