AN ALTERNATIVE COMPUTATIONAL MODEL FOR ARTIFICIAL INTELLIGENCE

Brian Louis Stuart

May 1992

Abstract

Stuart, Brian Louis. Ph.D., Purdue University, May 1992. An Alternative Computational Model for Artificial Intelligence. Major Professor: Chia-Hong Lee.

One objective of artificial intelligence is to mimic the behavior of natural intelligences, and learning is one of the most inherently intelligent activities. Unfortunately, there has not been much success in emulating the low-level learning of classical conditioning and instrumental learning. A novel model of computation, called the cybernetic automaton, which possesses a number of desirable properties, is presented here. First there exists a learning theorem which states that cybernetic automaton learning is in some sense complete. Then through a series of experiments, it is shown that cybernetic automata exhibit many of the properties of classical conditioning and instrumental learning. Finally, the cybernetic automaton model can be implemented using networks of model neurons which are somewhat biologically plausible. Cybernetic automata represent a new and substantial step toward artificially emulating natural behavior.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The search for an artificial intelligence is one of the oldest endeavors in computer science. In notes expanding her translation of Menabrea's paper about Babbage's Analytical Engine, Ada Lovlace (Babbage, 1889/1984) discussed the possibility that the engine might be able to play chess and to compose music. Both of these activities exhibit significant intelligence.

Work on early computers quickly turned to the problems of chess and music. At that time, it was felt that such problems represented the most intelligent type of activities and that if computers could be made to solve them, then they would be intelligent. However, as progress began to be made in these areas, it became evident that there was more to intelligence than solving hard problems. Indeed, even though computers are now encroaching on grandmaster level chess play, they are still not viewed as intelligent entities.

The realization that intelligence was more elusive than first thought led many to believe that a rigorous definition of intelligence was needed. Unfortunately, no definition has generally been agreed upon. Rather than address the myriad of philosophical and psychological issues inherent in a definition of intelligence, for this work it will suffice to present a number of characteristics of intelligence that are necessary in an artificial intelligence.

1.1 Goals for Artificial Intelligence

The fundamental goal for artificial intelligence is to produce man-made systems which exhibit behavior that is judged to be intelligent by human observers. As alluded to in the previous section, this goal has often been manifested in the attempt to solve difficult problems with the computer. In addition to game playing, one of AI's successes in solving difficult problems is the expert system. Expert systems attempt to emulate a human expert in facilitating the decision making process. In these and other areas, computers have indeed been fairly successful in addressing problems that were previously handled only by intelligent beings. However, the computers solving these problems are still not generally considered intelligent. Some of the reasons why are addressed in the next section.

Another goal for artificial intelligence is to contribute to cognitive science. There is a potential for the models of intelligence developed in AI to contribute to an understanding of natural intelligence.

The opportunity is greatest for those models created to mimic behavior in naturally intelligent systems and for those created to be as biologically plausible as possible.

1.2 Requirements of Artificially Intelligent Systems

One can infer from many of the results of AI that the degree to which a behavior is deemed intelligent often depends as much on how a problem is solved as on what problem is solved. Even though computers play good chess, the fact that their human programmers must be the ones to learn from the computer's mistakes leads most people to view the computer's play as unintelligent.

In order to help address the issue of what behavior is addressed in contrast to what problems can be solved, this section presents a set of characteristics which artificially intelligent systems must possess. These characteristics form a specification for the model of computation which is presented in the next chapter. As are discussed in the following subsections, systems which are viewed as intelligent must exhibit both adaptive and probabilistic behavior.

1.2.1 Learning

The idea that an intelligent system must learn from its mistakes is often fundamental to the very idea of intelligence. The related idea of adapting to changes in the environment is also frequently expected of intelligences.

Psychology has long recognized the importance of learning and adaptive behavior in intelligence. A substantial body of data has been collected on learning in lower animals and in humans. These data have led to a number of classifications of learning and to identifying properties of those classifications. Of particular interest here are classical conditioning and instrumental learning (often called operant conditioning). These types of learning are among the most extensively studied. They are found in a wide variety of organisms from humans to molluscs. Consequently, conditioning is often viewed as one of the most basic types of learning. It follows that any system which mimics natural intelligence (at whatever level) must exhibit the properties of classical and operant conditioning.

The model of computation presented in the next chapter exhibits many of the properties of classical and operant conditioning, as shown in Chapters 4 and 5.

1.2.2 Probabilistic Behavior

The concept of learning from mistakes implies some degree of trial-and-error searching for the correct behavior. In order to have a number of behaviors to select from, there must be a variability in responses. The converse type of behavior is to produce the same response every time to a particular situation. This type of behavior is often called mechanistic or machine-like and is viewed as being the opposite of intelligent in many ways.

In naturally intelligent systems, varied responses are viewed as being stochastic in nature. Most of the results in psychology are reported in terms of their statistical significance. It is often the unpredictability of a response itself that is perceived as natural and intelligent by human observers. Consequently, any system which mimics naturally intelligent behavior must appear to have a statistical component to its response selection.

1.3 Prior Art

The importance of learning and probabilistic behavior was recognized early in the history of computer science, and much research has gone into these issues. This section reviews some of the highlights of previous learning research. It is divided into two subsections, one on learning in symbolic AI and one on learning in neural networks. Because of its close connection to the present work, previous work on probabilistic automata is discussed in the next chapter.

1.3.1 Symbolic Learning

As with much of AI, several of the early experiments in machine learning centered on game playing. One of the simplest learning "machines" was Gardner's (1962) HER (Hexapawn Educable Robot). This system learned how to win games of hexapawn by pruning the branches that led to prior losses from its search tree. Initially, all moves were available to the "machine," which was implemented with a set of matchboxes, each containing colored beads. When it was the machine's turn to make a move, it selected at random a bead from the box which corresponded to the current board configuration. This bead then determined one of the legal moves in that position. All of these selections were remembered. If the machine won the game, then all of the played beads were restored to their boxes. If the machine lost the game, then the last bead played was discarded to prevent making that move again. In addition, if the box from which the last bead was drawn became empty with the discard, then the previous bead was also discarded, and the discard process was continued recursively. (Gardner originally waited until the next game with the position whose box was empty and took the previous bead then.) Gardner reported that in a typical 50-game tournament, the machine lost 11 games out of the first 36, but played perfectly after that.

A more advanced application of learning was developed by Samuel (1959) on an IBM 704 for playing checkers. His system played the game of checkers by using an α-β search on the move search tree. The search was carried to a maximum of 20 plies and a minimum of 3 plies. The first approach he took to learning was to have the system remember the values for nodes seen in the past. This saved search time, since in an α-β search, the value of a node depends on the sub-tree rooted at that node. When a position was encountered that had been seen before, its sub-tree was not searched again to determine the value of the node. After accumulating 53,000 moves on tape from games with a variety of opponents, the computer was able to play "as a rather better-than-average novice, but definitely not as an expert" (Samuel, 1959).

Samuel next sought to generalize the knowledge gained through experience so that the storage requirements would be lessened. In particular, he applied learning to the evaluation heuristic used by the program. The evaluation function was a 16 term polynomial with weighting coefficients on each term. The terms were chosen from a set of 38 parameters. Learning consisted of modifying the coefficients and selecting the set of 16 terms. His most successful learning occurred when two versions of the program, called Alpha and Beta, were pitted against each other. Alpha updated its set of coefficients and term selections at the end of each move, while Beta kept its evaluation function throughout the game. At the end of each game, if Alpha had won, then Beta took on its evaluation function. On the other hand, if Beta won, then Alpha's loss was recorded. If Alpha lost three games in a row then, to remove it from a local maximum, the system would reduce the coefficient of the leading term to zero. The evaluation function learning approach was successful in improving

the middle game playing of the system, but it did not develop the strong openings that the rote learning version did. It also had some minor weaknesses in the endgame. Overall, though, it did do much better than the rote learning approach to checkers. In his 1959 paper, Samuel discusses the possibility of combining the two approaches to gain the benefits of each, but there is no indication that this work was ever carried out.

Hexapawn and Checkers both have the property that each player (in particular, the learning computer) has complete knowledge of the game state at all times. In contrast, poker does not have this property and, consequently, a program that learns poker faces special challenges. Nevertheless, Donald Waterman developed a system that learned to place bets in poker (Cohen & Feigenbaum, 1982). Waterman's poker program chose its bets according to a set of rules that were learned. An example of such a rule would be the following:

$$(FAIR,\ LARGE,\ LARGE,\ *,\ *,\ *,\ *) \Rightarrow$$
$$(*,\ POT + (2 \times LASTBET),\ 0,\ *,\ *,\ *,\ *)\ CALL.$$

which means that if the computer's current hand is $FAIR$, the pot is currently $LARGE$ and the last bet was $LARGE$, it should $CALL$ (place a bet equal to the last bet). The stars in the antecedent of the rule indicate that the last four state variables are to be ignored, and those in the consequent indicate that the state variables should retain their previous values. The rules were searched in the database in order, and the first one encountered whose antecedent was a subset of the current state was executed. These rules could be learned either by being told or by an analysis of previous games. Also, there were two options for being told new rules: the user could give them or they could be obtained from a computer opponent. The most interesting of these options was the analytic learning. Here Waterman provided the system with a set of Predicate Calculus axioms which described the rules of poker. Using these, the program assigned blame to bad bets in a lost game. It did this by first using the state variables of the game to determine a set of true predicates for each bet placed. Then it tried to prove the predicate $MAXIMIZE(YOURSCORE)$ by backward chaining. If it found that a better bet could have been made than the one that was, it would then modify its betting rules to change its behavior. If a rule prior to the rule that caused the bad bet had the correct consequent and had an antecedent that was close to the game state at the time of the bad bet, then it was generalized to cover that state. Otherwise, if there was one after the offending rule, then the offending rule was specialized to prevent its being activated in that state, and the following close rule generalized (if necessary) to be activated. If no such close rule was found in the database, a rule was created with the game state as its antecedent and the correct action as its consequent, and the rule was placed in the database before the offending rule.

The technique of specializing and generalizing rules to produce the correct results for novel situations forms a general paradigm for rule-based learning systems. A good introduction to these techniques can be found in the *Introduction to Artificial Intelligence* by Charniak and McDermott (1987). Some of the influential programs that have used this learning approach are HACKER, by Sussman, which learns how to solve planning problems and LEX, which learns to do symbolic integration (Cohen & Feigenbaum, 1982).

Another interesting learning system is AM, developed by Lenat (Cohen & Feigenbaum, 1982). The function of this program is to find interesting conjectures in the domain of mathematics. It maintains a database of mathematical concepts known to it. This database is initially seeded with

some knowledge that establishes the specific area of mathematics which AM will examine. As the program investigates mathematical concepts, it applies a set of fixed rules to the concepts in its database to generate new concepts from variations on and relationships between existing ones. The concepts themselves have rules governing their properties. These rules can be generalized and specialized as new supporting examples and counter-examples are found. There are also rules which are used to evaluate the "interestingness" of the new concepts. When AM begins with some basic axioms of set theory, it learns (discovers) a number of well-known properties of sets. After running for long enough, AM generates a concept of numbers and from there develops many of the fundamental concepts of arithmetic. Among these are prime numbers and the Fundamental Theorem of Arithmetic (that any number has a unique prime factorization).

1.3.2 Neural Networks

Recently, another approach to learning has become popular. This approach is called variously neural networks, parallel distributed processing and connectionist processing. However, this area of research is not really new; it dates back at least to the 1950's. Neural networks are generally organized as a large collection of highly interconnected processing elements. Each processing element is capable of only very limited functions. Typically, these functions are to perform a weighted sum of the inputs to the element and perhaps to apply a threshold or some other shaping function to the resulting sum. In most systems, learning is performed by updating the weights between processing elements. This organization is inspired by current knowledge of the organization of the brain.

From today's perspective, the most influential learning model to come out of the early work was the Perceptron. This learning machine uses a very simple model of a neuron whose output is simply a step function of the weighted sum of its inputs (Rumelhart, Hinton & McClelland, 1986). A complete Perceptron has two sets of neurons, an input layer and an output layer. Each element in the output layer takes input from all elements of the input layer. The elements of the input layer take input from the environment. While several variations exist on what outputs are allowed from an element, the discussions here will assume that the outputs will either be 0 or 1. The output of a particular output element is given by the following equation:

$$O_i = \begin{cases} 0, & \text{if } \sum_{j=1}^{N} w_{ij} I_j < \Theta_i; \\ 1, & \text{otherwise} \end{cases}$$

where O_i is the output of the i^{th} output element, $0 < \Theta_i < 1$ is the threshold for output unit i, I_j is the output of the j^{th} input element and w_{ij} is the weight applied to the connection from the j^{th} input unit to the i^{th} output unit. When training a Perceptron, one presents input patterns that should map to target output patterns T_i. The weights are then updated according to the equation:

$$w'_{ij} = w_{ij} + \eta(T_i - O_i)I_j$$

where w'_{ij} is the new value for w_{ij} and η is the speed with which the weight is moved toward its trained value. This update rule means that if an output is not right, then the weights entering that output element should be adjusted. The weight is only adjusted if the input element feeding that connection was active in the training instance. If the output was a 1 and should have been a 0, then the weight is decreased, and vice versa.

There is a well-known theorem, first stated by Rosenblatt, proving that any set of input/output patterns that can be mapped by a Perceptron can be learned using the learning rule described above (Nilsson 1965). This is a very powerful theorem. However, Minsky and Papert have shown that many useful functions, among them the exclusive-or (XOR), cannot be solved by a Perceptron (Minsky & Papert, 1969). Therefore, in the absence of much more complicated neuron models, connectionist solutions to many problems require more than the two layers of the Perceptron. Unfortunately, as of yet, there is no learning rule for multiple layer neural nets that has the properties of the Perceptron learning rule.

For multiple layer nets, the most popular learning rule is the Generalized Delta Rule (the Perceptron learning rule is also known as the Delta Rule), otherwise known as Backpropagation (Rumelhart, Hinton & Williams, 1986). This rule also allows the network to have feedback from elements in higher layers to elements in lower layers. When using this learning rule, the neural elements cannot have a step threshold function. Instead, a sigmoid is generally used. With this threshold function, a neural element's output is given by:

$$O_i = \frac{1}{1 + \mathrm{e}^{-(\sum_j w_{ij} O_j + \Theta_i)}}$$

where the summation is taken over all elements in the net that have connections to inputs of element i. Also using the sigmoid threshold function, the learning rule is defined by the following equations:

$$\delta_i = \begin{cases} (T_i - O_i) O_i (1 - O_i), & \text{if element } i \text{ is an output unit} \\ O_i (1 - O_i) \sum_j \delta_j w_{ji}, & \text{otherwise} \end{cases}$$

$$w'_{ij} = w_{ij} + \eta \delta_i O_j$$

Using this learning rule, many of the problems which were impossible for the Perceptron have been solved, among them XOR, parity and symmetry.

Using other models of neural nets, many other researchers have succeeded in training nets to many other functions. In particular, neural nets have been applied extensively to pattern recognition. Fukushima (1987) has developed a system for recognizing letters and digits. This system is trained by presenting it with repeated patterns from the set to be learned. The system identifies repeated applications of the same pattern vs. applications of new patterns for itself. It is capable of recognizing the learned patterns with a fair amount of distortion and with a shift of position. The system also acts as an associative memory so that incomplete patterns are recognized and reconstructed. In the system's most recent incarnation, multiple patterns presented in the same field of reception are recognized in turn, demonstrating a primitive focus of attention.

1.4 Summary

Two fundamental goals of artificial intelligence are to produce man-made intelligent systems, and to contribute to an understanding of natural intelligence. Any system which is judged to be intelligent must exhibit adaptive and probabilistic behavior. Substantial work has been done on producing systems which exhibit some degree of learning and/or probabilistic behavior. None of these systems, however, shows a significant degree of emulation of natural learning.

The following chapter presents a new model of computation which mimics much of natural learning and which may provide some insight into natural intelligence. Chapters 3 through 6 provide some theoretical and experimental support for the utility of the model.

Chapter 2

The Cybernetic Automaton Model

The problem of modeling intelligence has occupied the minds of many of computer science's pioneers. Indeed, both Turing and von Neumann have addressed the problem. After John von Neumann's death, his wife, Klara, published his manuscript for the undelivered Silliman Lectures, (von Neumann, 1958). In this treatise, he considers parallels between the computer technology he helped to pioneer and the operation of the brain. Among the issues he addresses are adaptability and statistical operation within the brain. Turing (1970) also addressed these issues in his consideration of intelligent machinery. He proposed the idea that adaptive, probabilistic automata could exhibit intelligent behavior. While it appears that he never followed up on this line of thought, the paper summarizing these ideas should be viewed as a forerunner of the present work.

In further developing the idea of adaptive, probabilistic machinery as artificial intelligences, this chapter presents a new model of computation. This model is an adaptive, probabilistic automaton model which, in later chapters, is shown to exhibit some intelligent behavior. The material in this chapter builds on and assumes a knowledge of the basic principles of deterministic and nondeterministic finite automata (DFAs and NFAs). Introductions to and results of these computational models can be found in Hopcroft and Ullman (1979), Cohen (1986) and Lewis and Papademtriu (1981).

2.1 The Probabilistic Finite Automaton

In an attempt to generalize the deterministic finite automaton (DFA) and the nondeterministic finite automaton (NFA), Rabin (1963) proposed a probabilistic finite automaton (PFA). (The PFA sometimes goes under the name stochastic automaton.) In order to formally define the PFA, some preliminary notation will be helpful.

DEFINITION 1 *Let P^S be a probability distribution over the set S. Let $\sigma\left(P^S\right)$ be a selection from the set chosen at random according to the probability distribution P^S. Hence, the probability that an element $s \in S$ is selected is given by*

$$P\left(\sigma\left(P^S\right) = s\right) = P^S(s).$$

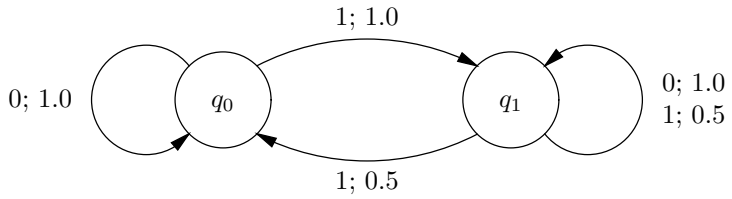

Figure 2.1: An Example PFA

Furthermore, let P_R^S be the set of probability distributions over the set S indexed by the set R, and let P_r^S be the $r^{\text{th}}(r \in R)$ probability distribution over S. □

The PFA can then be defined in a manner analogous to the DFA.

DEFINITION 2 *A PFA, A, is defined by*

$$A \equiv \langle Q, \Sigma, \delta, q_0, F \rangle$$

where Q is the set of states of A, Σ is the input alphabet for A, δ is a transition function for A, $q_0 \in Q$ is the initial state for A, and $F \subseteq Q$ is the set of final states for A. For the PFA, the function, $\delta : Q \times \Sigma \mapsto Q$ where $\delta(q, a) = \sigma\left(P_{(q,a)}^Q\right)$, selects the next state q_{t+1} given the current state q_t and current input symbol a_t. This selection is made probabilistically based on the probability distribution determined by q_t and a_t.

Just as for the DFA and NFA, the PFA is said to accept an input string x if the automaton is left in a state $q_f \in F$ upon the reading of the last symbol in x. However, because of the probabilistic nature of the PFA, this is true only for a particular execution of the PFA on the string x. In a more general sense, a PFA is said to accept a string x with probability p if the probability that the PFA will accept x on a single run is greater than or equal to p. Similarly, it is said to accept a language L with probability p if for every $x \in L$ it accepts x with probability p. □

Figure 2.1 illustrates a PFA. When the automaton is in state q_1, and receives input 1, it will take the transition to state q_0 50% of the time and will stay in state q_1 the other 50%.

It is a well-known result from automata theory that the DFA and NFA are equivalent in the sense that any DFA has an equivalent NFA which accepts the same language and vice versa. It should also be clear that any DFA is also a PFA where $P_{(q,a)}^Q(q') = 1$ and $P_{(q,a)}^Q(\hat{q}) = 0$ for $q', \hat{q} \in Q, \hat{q} \neq q'$ and $q' = \delta(q, a)$ in the DFA. Paz (1966) has shown, however, that the converse is not true. In particular, he showed that there exists a PFA which accepts a language that is not regular. Therefore, the set of languages accepted by PFAs is strictly a superset of those accepted by DFAs.

2.2 PFAs with Output

For DFAs and NFAs, two forms of automata with output have been described. These are the Moore machine and the Mealy machine. For the Moore machine, output symbols are associated with states

whereas for the Mealy machine, output symbols are associated with transitions. The same output production schemes can be applied to probabilistic automata leading to the Moore PFA and the Mealy PFA.

In a Moore or Mealy PFA, the question arises as to whether the output symbols are generated in a deterministic or a probabilistic fashion. This leads to two types of Moore and Mealy PFAs, which will be designated as Type I and Type II. More precisely, Type I and Type II Moore and Mealy PFAs are defined as follows:

DEFINITION 3 *A Moore or Mealy PFA, A, is defined by*

$$A \equiv \langle Q, \Sigma, \Delta, \delta, \lambda, q_0 \rangle$$

where Q, Σ, δ, and q_0 are defined as in Definition 2, Δ is the output alphabet of A, and λ is the output function for A. The form of the output function λ is given as follows for each of these models:

$\lambda : Q \mapsto \Delta$	*for Type I Moore PFA*
$\lambda : Q \mapsto \Delta$, *where* $\lambda(q) = \sigma\left(P_q^{\Delta}\right)$	*for Type II Moore PFA*
$\lambda : Q \times \Sigma \mapsto \Delta$	*for Type I Mealy PFA*
$\lambda : Q \times \Sigma \mapsto \Delta$, *where* $\lambda(q,a) = \sigma\left(P_{(q,a)}^{\Delta}\right)$	*for Type II Mealy PFA*

For a Type I Moore PFA, λ deterministically selects an output symbol for the current state. For a Type II Moore PFA, λ selects an output symbol randomly subject to the probability distribution for output symbols associated with the current state. For a Type I Mealy PFA, λ selects an output symbol deterministically for the transition being taken. For a Type II Mealy PFA, λ randomly selects an output symbol subject to the probability distribution for output symbols associated with the transition being taken. □

Another well-known result from automata theory is that Moore and Mealy machines are equivalent. For each Moore machine, there is a Mealy machine which has the same mapping between input and output strings, and vice versa. The following group of theorems establishes equivalence among the four PFA output models.

THEOREM 1 *Type I and Type II Moore PFAs are equivalent.*

PROOF: Part 1: It is clear that every Type I Moore PFA can be directly transformed to a Type II PFA with the same states and transitions and where the output probability distribution is composed of a single unity value for one output symbol and zero values for all others.
Part 2: Given a Type II PFA, an equivalent Type I PFA can be constructed as follows: let q be a state with n transitions entering with probabilities $p_1, p_2, p_3, \ldots, p_n$. Furthermore, let the output probability distribution of q have k non-zero values for output symbols $b_1, b_2, b_3, \ldots, b_k$. Now replace q with k states, $q_1, q_2, q_3, \ldots, q_k$ with each state q_l having the single output symbol b_l. Each of the states q_l will have the same set of transition probability distributions as did q. For each transition m entering q, a transition is created entering each q_l with probabilities given by $p_m P_q^{\Delta}(b_l)$. This construction is illustrated in Figure 2.2. The machine created by transforming the single state is equivalent to its predecessor. By repeating this transformation for each state in a Type II PFA, an equivalent Type I PFA is constructed.

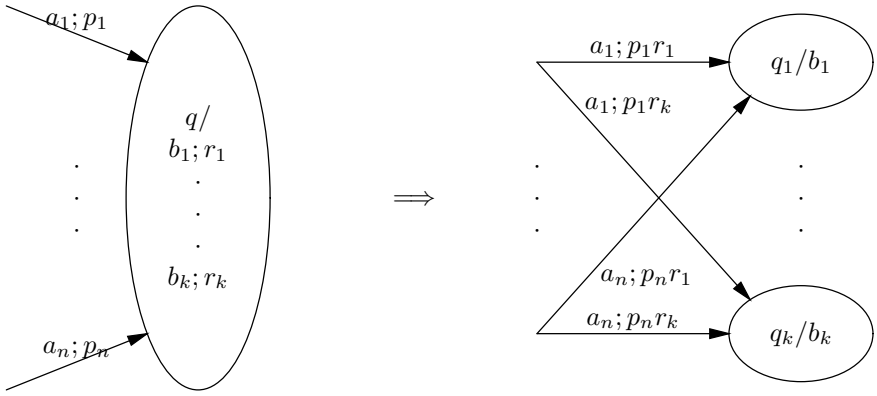

Figure 2.2: Transformation from Type I to Type II

Since a Type I PFA can be transformed into a Type II PFA and vice versa, the Type I and Type II Moore PFAs are equivalent models of computation. Q.E.D. □

By a similar argument based on transitions rather than states, the following is established.

THEOREM 2 *(Stated without proof) Type I and Type II Mealy PFAs are equivalent.*

The equivalence between the Type I Moore PFA and the Type I Mealy PFA is established in the following theorem. However, because the Moore PFA outputs a symbol from state q_0 prior to receiving its first input symbol, its output string will always be one symbol longer than that of the Mealy PFA. So the equivalence is defined as follows: if for all input strings x and output strings y, a Type I Moore PFA produces output string ay with probability p_{xy}, and if the Type I Mealy PFA produces output string y with probability p_{xy}, the PFAs are said to be equivalent.

THEOREM 3 *The Type I Moore PFA and the Type I Mealy PFA are equivalent models.*

PROOF: This theorem can be proven in a manner exactly analogous to that for DFAs. Namely, the Moore PFA may be reduced to a Mealy PFA as follows: for each state q with output symbol a in the Moore PFA, all transitions entering q in the Mealy PFA have output symbol a. Likewise, the Mealy PFA $\langle Q, \Sigma, \Delta, \delta, \lambda, q_0 \rangle$ can be reduced to a Moore PFA $\langle Q', \Sigma', \Delta', \delta', \lambda', q_0' \rangle$. Here let $Q' = Q \times \Delta$, $\Sigma' = \Sigma$, $\Delta' = \Delta$, $P^{Q}_{((q_1,b),a)}((q_2, c)) = P^{Q'}_{(q_1,a)}(q_2)$, $\lambda'((q, b)) = b$, and $q_0' = (q_0, b_0)$ where $b_0 \in \Delta$ is selected arbitrarily. Because the Type I Moore PFA is reducible to the Type I Mealy PFA and vice versa, they are equivalent models. Q.E.D. □

COROLLARY 1 *All four of the models, Types I and II Moore PFAs and Types I and II Mealy PFAs are equivalent.*

PROOF: Because the nature of the proofs of equivalence given in this section are constructive, the equivalences are transitive. Consequently, any pair of them are equivalent and the whole set is equivalent. Q.E.D. □

2.3 Probabilistic Output Automata

In this section, probabilistic output automata (POAs) are described. The POA forms the basis of the cybernetic automaton defined in a later section which addresses many of the objectives of artificial intelligence discussed in Chapter 2. Fundamentally, the POA is an automaton with deterministic state transitions and probabilistic output symbol selection.

DEFINITION 4 *A Probabilistic Output Automaton, A, is given by*

$$A \equiv \langle Q, \Sigma, \Delta, \delta, \lambda, q_0 \rangle$$

where Q is the set of states in A, Σ is the input alphabet for A, Δ is the output alphabet for A, $\delta : Q \times \Sigma \mapsto Q$ is the transition function for A and $q_0 \in Q$ is the initial state for A. As for other automata with output, the output symbols may be associated with the states or with the transitions. The output function, λ can be defined then as

$$\begin{array}{ll} \lambda : Q \mapsto \Delta, \textit{where } \lambda(q) = \sigma\left(P_q^{\Delta}\right) & \textit{for Moore POA} \\ \lambda : Q \times \Sigma \mapsto \Delta, \textit{where } \lambda(q,a) = \sigma\left(P_{(q,a)}^{\Delta}\right) & \textit{for Mealy POA.} \end{array}$$

□

As might be expected from the equivalence of the Moore and Mealy machine in deterministic, nondeterministic and probabilistic automata, the Moore POA and Mealy POA are also equivalent. This is demonstrated in the following theorem.

THEOREM 4 *The Moore POA and the Mealy POA are equivalent models.*

PROOF: Let a Moore POA be given by $A \equiv \langle Q, \Sigma, \Delta, \delta, \lambda, q_0 \rangle$ and let a Mealy POA be given by $A' \equiv \langle Q', \Sigma', \Delta', \delta', \lambda', q_0' \rangle$.
Part 1: A Moore POA, A, can be converted to a Mealy POA, A', as follows. Let $Q' = Q$, $\Sigma' = \Sigma$, $\Delta' = \Delta$, $\delta' = \delta$ and $q_0' = q_0$. The output distribution function for the Mealy POA is given by $P_{(q,a)}^{\Delta'} = P_{\delta(q,a)}^{\Delta}$.
Part 2: A Mealy POA, A', can be converted to a Moore POA, A, as follows. Let $Q = Q' \times Q' \times \Sigma'$, $\Sigma = \Sigma'$, $\Delta = \Delta'$ and $q_0 = (q_0', q_0', a)$ where $a \in \Sigma$ is chosen arbitrarily. The transition function is given by $\delta((q_1, q_2, a), b) = (\delta'(q_1, b), q_1, b), \forall a \in \Sigma, q_2 \in Q$. The output probability distributions are given by $P_{((q_1,q_2,a))}^{\Delta} = P_{(q_2,a)}^{\Delta'}, \forall q_1 \in Q$.

These transformations demonstrate that there is an equivalent Moore POA for every Mealy POA and vice versa. Therefore, the Moore POA and the Mealy POA are equivalent models. Q.E.D. □

2.4 Adaptive Probabilistic Automata

One method of creating probabilistic automata that learn is to specify rules for modifying the probabilities on transitions. Several cybernetics and control systems researchers have examined models of this type. Some of these are reported in Glushkov (1964/1966) and more recently Lee, et al (1990) and Qian, et al (1990) have applied such learning to stochastic cellular automata.

2.4.1 Reinforcement Probability Learning

In all of the adaptive probabilistic automata referenced above, learning is governed by an external punishment signal. As each transition is taken, the environment will activate the punishment signal if the transition is taken incorrectly. The environment will not activate the punishment signal (and implicitly deliver a reward) if the transition is taken correctly.

2.5 The Need for a New Model

Existing probabilistic automaton models are not sufficient to meet the objectives of artificial intelligence discussed in Chapter 1. Similarly, neither are existing neural network models. This section will examine some of the shortcomings of existing models which will be addressed by the new model defined in the following section.

2.5.1 Timing Issues for Input Symbols

Systems which mimic naturally intelligent behavior must operate in real-time. This means that they must take into account a variety of concerns regarding the timing of input symbols. However, existing automaton models abstract away the timing relationships among input symbols and treat the input sequence as an untimed stream of input symbols.

As will be seen in Chapter 4, there are circumstances in which the behavior of a natural intelligence differs based on whether two stimuli occur simultaneously or sequentially. Consequently, it is necessary for artificially intelligent systems to account for such contingencies in the input. In existing automaton models, the only way to do this is to define new input symbols which represent subsets of the original alphabet. Unfortunately, this approach would lose the connection between the simultaneous symbol and the original constituent symbols. Some neural network models do account for simultaneous input symbols. For instance, the neocognitron (Fukushima 1975, 1984, 1986, 1987; Fukushima & Miyake 1982) uses an attention mechanism to control recognition of more than one pattern found simultaneously in the receptive field.

Another important timing consideration is the existence of long delays between input symbols. If two input symbols occur in close temporal proximity, then natural systems will tend to associate them. On the other hand, if they are widely separated in time, then they will not be associated. Again, existing automaton models do not account for this phenomenon directly. As for simultaneous input symbols, it is possible to define a symbol to represent a period of delay. Such a symbol would have to be treated as a special symbol to prevent two stimuli separated by a delay from being indirectly associated through the delay symbol. The more biologically faithful neural network models often allow for the behavioral effects of delays in the input. In the model of mollusc behavior described in the appendix, the details of neuron operation specify that the degree to which inputs are associated is a decreasing function of their temporal separation.

Some neural network models also do a better job of dealing with the duration of a stimulus. As long as a stimulus is active, the network's corresponding input neuron(s) will be firing. As before, the automaton models do not have any mechanism for representing the duration of stimuli or the case where stimuli overlap without their onsets or offsets being simultaneous. It is, however, a relatively simple matter to define the input alphabet to be symbols representing onset and offset of

stimuli. Namely, each stimulus results in two input symbols, one at its onset and one at its offset. This approach does not require any special treatment of symbols or other changes to the general nature of automata. This approach is also used in the new model defined in the next section and is discussed in more detail in Chapter 4.

In general, some neural network models account for time relationships among inputs. However, existing automaton models do not have any concept of time defined beyond the relative times at which inputs symbol arrive. Some of the temporal issues have easy solutions within the framework of automata theory and others require fundamental changes to the notions of automata theory itself. The new model defined in the next section represents one approach to solving these problems.

2.5.2 Strengths of Input Symbols

As in the temporal issues related to input symbols, some neural network models directly support inputs of varying strength whereas the concept of input strength is not defined for existing automaton models. In naturally intelligent systems, stimulus strength is an important aspect of the stimulus. As discussed in Chapter 4, the strength of a stimulus affects how strongly and how fast learning takes place. Consequently, any artificially intelligent system which successfully emulates natural intelligence must account for the strength of input symbols.

Many neural network models support analog inputs. In these, the strength of an input is indicated either by a signal level or, in more biologically plausible models, the firing rate of the input neurons. An automata theoretic model would also require some mechanism for representing and utilizing the strength of an input symbol. The most obvious method is to treat each input as a pair, $\langle x, s \rangle$, where $x \in \Sigma$ is an input symbol and $s \in \Re$ is a real number representing the strength of the input symbol. The strength, s, could then be used by the automaton to produce the desired behavior.

2.5.3 Learning

The form of learning which was discussed above as part of adaptive probabilistic automata relied on the existence of reward and punishment signals. Such learning is termed supervised learning. While some natural learning is supervised, much of what is known about learning in natural systems is based on unsupervised learning. Therefore, a successful computational model for artificial intelligence must exhibit both correct supervised learning and correct unsupervised learning.

No existing automaton models include both supervised and unsupervised learning. Consequently, mechanisms for supervised and unsupervised learning which correctly mimic natural behavior would need to be developed and incorporated into any automata theoretic model for artificial intelligence. There do exist both supervised and unsupervised neural network models though rarely both in the same model. Furthermore, none of the unsupervised models emulate natural learning very accurately. For example, Gelperin, Hopfield and Tank (1985) exhibited a model for simulating learning in the limax mollusc. This model simulates some of the characteristics of classical conditioning, but a large subset of its properties are not accounted for.

In many ways, learning is the most central and most demanding of the issues in developing a computational model for artificial intelligence. Historically, computational models have not even attempted to incorporate learning in any way. Systems designed to exhibit adaptive behavior have generally been directed toward solving optimization problems and consequently have shown the

greatest success in control systems applications. The learning models studied for neural networks are quite varied. Rummelhart and McClelland (1986) have provided an encyclopedic survey of the techniques which are in most common use. These learning algorithms generally train networks to classify or recognize inputs, but again they do not display the psychologically observed properties of natural learning. Consequently, they are not entirely appropriate for artificial intelligence and another model is needed.

2.5.4 Structural Adaptation

It is important to note that in the automaton models discussed thus far the structure of the automaton is set a priori. This means that all possible situations must be accounted for in the initial automaton before any learning takes place. Such a requirement is theoretically unsatisfying. To understand why, consider the following. If the input symbol α is to produce the output symbol β after reading the input string x and symbol γ after reading the string y, then the automaton must be in different states after reading strings x and y. However, by the pumping lemma, for any automaton with a finite number of states, there exist strings x and y which cannot be distinguished. Therefore, such an automaton could not be trained to produce different output symbols on the input symbol α after following x and y if x and y were long enough. The basic implication of these observations is that there must be a mechanism which allows the structure of an artificially intelligent system to adapt.

A similar situation exists in work on neural networks. Virtually all network models establish the number and set of allowable connections a priori. One exception is dynamic node creation (DNC) in backpropagation networks as described by Ash (1989). DNC adds one hidden node at a time until the output converges sufficiently close to the desired output. Because it relies on knowing what the output should be, such a technique would not be applicable to unsupervised learning. Any technique which creates new nodes is also unsatisfactory from a cognative science viewpoint. In the maturation process, natural brains develop virtually all of their neurons early relative to their periods of learning. In other words, natural systems continue to learn after they have grown virtually all of the neurons they will ever have. Consequently, the set of connections is the only place where learning can be expected to take place in natural systems, and learning models which are not based on this will not lend much insight into natural processes.

Clearly, there is a need for a new model of learning in which the structure of the learning system is modified by the inputs. It is important that this new model do so in a controlled way and not simply record all past inputs. Such a simple record would be undesirable for several reasons. First, the volume of information would quickly become unmanageable. Also, a naïve record of inputs would not encode the probabilities of one symbol being followed by another. In classical conditioning (discussed in Chapter 4), this probability determines the degree of partial reinforcement which, in turn, affects the behavior of the system.

2.5.5 Completeness

In 1962, Rosenblatt gave the perceptron learning theorem for the perceptron neural network model. This theorem states that with the perceptron learning rule, a two layer perceptron can learn the

input-output mapping which is performed by any two-layer perceptron. Later, however, Minsky (1969) showed that the set of mappings which were possible in a two layer network was unsatisfactorily limited. Since then, networks with three and more layers have been shown to possess many of the mappings that were not possible in two layer networks. This has been achieved by using more sophisticated learning rules, but no theorem similar to the perceptron learning theorem has been found for these more complex networks.

The history of automata theory has been rich with theorems of completeness and computability, but the same is not true for neural networks or for artificial intelligence in general. From a theoretical standpoint, it is desirable that for any model of intelligence or learning, some theorems of completeness be provable. For instance, it would be useful to have a proof that a minimal initial configuration of the model can learn to solve any problem solvable by any implementation of the model. It would also be useful to identify the class of problems which can be solved by systems in the model and to position that class within the hierarchy of other known computational models.

2.5.6 Summary

In general, there are a number of characteristics which are necessary for a model to successfully emulate intelligent systems, but which are not possessed as a complete set by any existing model. Some of these, such as dealing with input timing and stimulus strength are found in some neural network models. Others, such as exhibiting adaptive probabilistic behavior and being susceptible to proofs regarding computational capabilities, are found in the work on adaptive probabilistic automata. Most importantly, however, none of the existing models show significant approximations to the varied and subtle properties of natural learning. This is especially true at the lowest level of learning, that of classical conditioning and instrumental learning. The remainder of this chapter is devoted to presenting a new computational model which does possess the desired characteristics. It is called the cybernetic automaton model, and subsequent chapters illustrate its capabilities.

2.6 Formal Definition of the Cybernetic Automaton

This section presents a formal definition of the new model, cybernetic automata. A more general and higher-level discussion of the new model can be found in the next section.

DEFINITION 5 *A Cybernetic Automaton, A, is defined by*

$$A \equiv \langle Q, \Sigma, \Delta, \delta, \lambda, q_0, R, P, C, E \rangle$$

where Q is the set of states, Σ is the input alphabet, Δ is the output alphabet, $\delta : Q \cup \{\epsilon\} \times \Sigma \mapsto Q$ is the state transition function, $\lambda : Q \times \Sigma \mapsto \Delta$ where $\lambda(q,a) = \sigma\left(P^{\Delta}_{(q,a)}\right)$ is the output selection function, q_0 is the initial state, $R \subseteq Q$ is the set of reward states, $P \subseteq Q$ is the set of punishment states, $C : Q \times \Sigma \mapsto \Re$ is a confidence function over the set of transitions, and $E : Q \times \Sigma \times Q \times \Sigma \mapsto \Re$ is an expectation function between pairs of transitions.

The operation of the cybernetic automaton is described by the following procedures.

Cybernetic Automaton Operation:

1. *Let* $c = q_0$, $q_l = q_0$, $a_l = \epsilon$ *and* $o_l = \epsilon$.

2. *If the time since the last input symbol arrived is greater than* τ *then:*

   If $\delta(c, \epsilon)$ *is defined:*

   If $\delta(c, \epsilon)$ *is marked as temporary:*

   Mark $\delta(c, \epsilon)$ *as permanent.*

   Let $q_l = c$.
   Let $c = \delta(c, \epsilon)$.

   Let the anchor state $q_a = c$.
   Let $a_l = \epsilon$ *and* $o_l = \epsilon$.
   Unmark all symbols o *and distributions* $P^{\Delta}_{(q,\alpha)}$.
   Go to step 2.

3. *Input a set of symbol-strength pairs,* $I = \{\langle a_1, s_1\rangle, \langle a_2, s_2\rangle, \ldots, \langle a_n, s_n\rangle\}$ *and let* I_l *be the previous set* I.

4. *Let* $\langle a_d, s_d\rangle$ *be the dominant input symbol-strength pair, where* s_d *is a maximum over all* s_i, $1 \leq i \leq n$.

5. *Perform Create New Transitions.*

6. *Let* $q_l = c$, $a_l = a_d$ *and* $o_l = o$.

7. *Output the symbol* $o = \sigma\left(P^{\Delta}_{c,a_d}\right)$ *with strength* $\frac{s_d C(c,a_d)}{1+C(c,a_d)}$.

8. *Mark* P^{Δ}_{c,a_d} *and its element for* o *for later modification.*

9. *Let* $c = \delta(c, a)$.

10. *Perform Update Expectations.*

11. *If* $c \in R$, *then perform Apply Reward else if* $c \in P$, *then perform Apply Punishment else perform Apply Conditioning.*

12. *Go to step 2.*

Create New Transitions:

If $\delta(c, \epsilon)$ *is defined and marked temporary,*

Remove the transition $\delta(c, \epsilon)$.

For each input pair $\langle a_i, s_i\rangle$ *in* I*:*

If $\delta(c, a_i)$ *does not exist, then*

Add a unique state q_N *to* Q.
Let $\delta(c, a) = q_N$.
Let $\delta(c, \epsilon) = q_a$ *and mark the transition as temporary.*
If there exists a state q' *for which* $\delta(q', a_i)$ *exists, then*
Let $P^{\Delta}_{(c,a)} = P^{\Delta}_{(q',a)}$.
Let $C(c, a_i) = C(q', a_i)$.
else
Let $P^{\Delta}_{(c,a)}(\epsilon) = \eta$ *and* $P^{\Delta}_{(c,a)}(\beta) = \frac{(1-\eta)}{|\Delta|-1}, \beta \neq \epsilon$.
Let $C(c, a_i) = 0.1$.

Update Expectations:

If $E(q_l, a_l, c, a_d)$ *exists, then*

Let $\Delta E(q_l, a_l, c, a_d) = \ \alpha(1 - E(q_l, a_l, c, a_d))$.
Let $C(q_l, a_l) = C(q_l, a_l)(1 - \beta|\Delta E(q_l, a_l, c, a_d)|)$.
Let $\Delta E(c, a_d, q_l, a_l) = \ \alpha(1 - E(c, a_d, q_l, a_l))$.
Let $C(c, a_d) = C(c, a_d)(1 - \beta|\Delta E(c, a_d, q_l, a_l)|)$.

else

Create and let $E(q_l, a_l, c, a_d) = \alpha$.
Create and let $E(c, a_d, q_l, a_l) = \alpha$.

For each symbol $a \in \Sigma$*:*

If $E(q_l, a_l, c, a)$ *exists and* $\langle a, \cdot \rangle \notin I$*, then*
Let $\Delta E(q_l, a_l, c, a) = -\alpha E(q_l, a_l, c, a)$.
Let $C(q_l, a_l) = C(q_l, a_l)(1 - \beta|\Delta E(q_l, a_l, c, a)|)$.

For each $q \in Q$, $a \in \Sigma$, $q \neq q_l$ *or* $a \neq a_l$*:*

If $E(c, a_d, q, a)$ *exists, then*
Let $\Delta E(q, a, c, a_d) = -\alpha E(q, a, c, a_d)$.
Let $C(c, a_d) = C(c, a_d)(1 - \beta|\Delta E(q, a, c, a_d)|)$.

For each $a \in \Sigma$ *and* $b \in \Sigma, a \neq b$*:*

If $\langle a, \cdot \rangle \in I$ *and* $\langle b, \cdot \rangle \in I$*, then*
Let $\Delta E(c, a, c, b) = \alpha(1 - E(c, a, c, b))$.
Let $C(c, a) = C(c, a)(1 - \beta|\Delta E(c, a, c, b)|)$.
else if $\langle a, \cdot \rangle \in I$ *or* $\langle b, \cdot \rangle \in I$*, then*
Let $\Delta E(c, a, c, b) = -\alpha E(c, a, c, b)$.
Let $C(c, a) = C(c, a)(1 - \beta|\Delta E(c, a, c, b)|)$.

Apply Reward:

Let $t = 1$.

For each $P^{\Delta}_{(q,\alpha)}$ *which is marked, starting at the most recent and moving toward the oldest:*

For each marked element, β*, in the distribution* $P^{\Delta}_{(q,\alpha)}$*:*

Let $P^{\Delta}_{(q_l,a)}(\beta) = \frac{P^{\Delta}_{(q_l,a)}(\beta)+\zeta ts_d(1/C(q_l,a))}{1+\zeta ts_d(1/C(q_l,a))}$.
Let $P^{\Delta}_{(q_l,a)}(b) = \frac{P^{\Delta}_{(q_l,a)}(b)}{1+\zeta ts_d(1/C(q_l,a))}$ *for all* $b \in \Delta, b \neq \beta$.
Let $\Delta C(c, a_d) = \zeta ts_d$.
Unmark β *from pending update.*
For each state, q'*:*

Let $P^{\Delta}_{(q',a)}(\beta) = \frac{P^{\Delta}_{(q',a)}(\beta)+\nu\zeta ts_d\left(1/C(q',a)\right)}{1+\nu\zeta ts_d(1/C(q',a))}$.
Let $P^{\Delta}_{(q',a)}(b) = \frac{P^{\Delta}_{(q',a)}(b)}{1+\nu\zeta ts_d(1/C(q',a))}$ *for all* $b \in \Delta, b \neq \beta$.
Let $C(q', a_d) = \nu\zeta ts_d$.

Unmark $P^{\Delta}_{(q,\alpha)}$ *from pending update.*
Let $t = \kappa t$.

Apply Punishment:

Let $t = 1$.

For each $P^{\Delta}_{(q,\alpha)}$ *which is marked, starting at the most recent and moving toward the oldest:*

For each marked element, β*, in the distribution* $P^{\Delta}_{(q,\alpha)}$*:*

Let $P^{\Delta}_{(q_l,a)}(\beta) = \frac{P^{\Delta}_{(q_l,a)}(\beta)}{1+\zeta ts_d(1/C(q_l,a))}$.
Let $P^{\Delta}_{(q_l,a)}(b) = \frac{P^{\Delta}_{(q_l,a)}(b)+\left(\frac{1}{|\Delta|-1}\zeta ts_d(1/C(q_l,a))\right)}{1+\zeta ts_d(1/C(q_l,a))}$ *for all* $b \in \Delta, b \neq \beta$.
Let $\Delta C(c, a_d) = \zeta ts_d$.
Unmark β *from pending update.*
For each state, q'*:*

Let $P^{\Delta}_{(q',a)}(\beta) = \frac{P^{\Delta}_{(q',a)}(\beta)}{1+\nu\zeta ts_d(1/C(q',a))}$.
Let $P^{\Delta}_{(q',a)}(b) = \frac{P^{\Delta}_{(q',a)}(b)+\left(\frac{1}{|\Delta|-1}\nu\zeta ts_d(1/C(q_l,a))\right)}{1+\nu\zeta ts_d(1/C(q',a))}$ *for all* $b \in \Delta, b \neq \beta$.
Let $\Delta C(q', a_d) = \nu\zeta ts_d$.

Unmark $P^{\Delta}_{(q,\alpha)}$ *from pending update.*
Let $t = \kappa t$.

Apply Conditioning:

If $o_l \neq \epsilon$ *and* $o \neq o_l$, *then*

For each $a \in \Sigma$:

If $E(q_l, a_l, q_l, a)$ *exists and* $\langle a, \cdot \rangle \in I_l$, *then*

Let $P^{\Delta}_{q_l,a}(o_l) = \frac{P^{\Delta}_{q_l,a}(o_l)+\gamma s_d(1/C(q_l,a))}{1+\gamma s_d(1/C(q_l,a))}$.

Let $P^{\Delta}_{q_l,a}(b) = \frac{P^{\Delta}_{q_l,a}(b)}{1+\gamma s_d(1/C(q_l,a))}$ *for all* $b \in \Delta, b \neq o_l$.

For each $q \in Q$ *and* $a \in \Sigma$ *such that* $\delta(q, a) = q_l$:

If $E(q_l, a_l, q, a)$ *exists, then*

Let $P^{\Delta}_{q,a_l}(o_l) = \frac{P^{\Delta}_{q,a_l}(o_l)+\gamma s_d(1/C(q,a_l))}{1+\gamma s_d(1/C(q,a_l))}$.

Let $P^{\Delta}_{q,a_l}(b) = \frac{P^{\Delta}_{q,a_l}(b)}{1+\gamma s_d(1/C(q,a_l))}$ *for all* $b \in \Delta, b \neq o_l$.

For each $a \in \Sigma$:

If $E(q_l, a_l, q_l, a)$ *exists,* $\langle a, \cdot \rangle \in I_l$ *and* $P^{\Delta}_{q_l,a}$ *has not been conditioned, then*

Let $\Delta C(q_l, a) = \gamma s_d$.

Perform Update Conditioning $(q_l, a, s_d(1/C(q_l, a)))$.

For each $q \in Q$ *and* $a \in \Sigma$ *such that* $\delta(q, a) = q_l$:

If $E(q_l, a_l, q, a)$ *exists and* $P^{\Delta}_{q,a}$ *has not been conditioned, then*

Let $\Delta C(q, a) = \gamma s_d$.

Perform Update Conditioning $(q, a, s_d(1/C(q, a)))$.

Update Conditioning: (q', a', s)

If $s > 0$, *then*

For each $a \in \Sigma$:

If $E(q', a', q', a)$ *exists and* $P^{\Delta}_{q',a}$ *has not been conditioned, then*

Let $P^{\Delta}_{q',a}(o_l) = \frac{P^{\Delta}_{q',a}(o_l)+\gamma s(1/C(q',a))}{1+\gamma s(1/C(q',a))}$.

Let $P^{\Delta}_{q',a}(b) = \frac{P^{\Delta}_{q',a}(b)}{1+\gamma s(1/C(q',a))}$ *for all* $b \in \Delta, b \neq o_l$.

For each $q \in Q$ *and* $a \in \Sigma$ *such that* $\delta(q, a) = q'$:

If $E(q', a', q, a)$ *exists, then*

Let $P^{\Delta}_{q',a'}(o_l) = \frac{P^{\Delta}_{q,a}(o_l)+\gamma s(1/C(q,a))}{1+\gamma s(1/C(q,a))}$.

Let $P^{\Delta}_{q,a}(b) = \frac{P^{\Delta}_{q,a}(b)}{1+\gamma s(1/C(q,a))}$ *for all* $b \in \Delta, b \neq o_l$.

For each $a \in \Sigma$:

If $E(q', a', q', a)$ *exists,* $\langle a, \cdot \rangle \in I_l$ *and* $P^{\Delta}_{q',a'}$ *has not been conditioned, then*

Let $\Delta C(q', a) = \gamma s$.

Perform Update Conditioning $(q', a, s(1/C(q', a)))$.

For each $q \in Q$ and $a \in \Sigma$ such that $\delta(q,a) = q_l$:
If $E(q', a', q, a)$ exists and $P^{\Delta}_{q,a}$ has not been conditioned, then
Let $\Delta C(q,a) = \gamma s$.
Perform Update Conditioning $(q, a, s(1/C(q,a)))$.

where α, β, γ, ζ, ν, κ and η are constants less than unity. The parameter α controls how fast expectations are acquired and lost, β control how fast confidence is lost and γ controls how fast learning takes place during conditioning. For supervised learning, the ζ parameter controls the speed of learning, ν determines how much the probability changes are propagated to other transitions with the same input symbol and κ ensures that more recent action get more reinforcement than earlier ones. The parameter η is the amount of probability that is assigned to the ϵ responses for a newly created transition. It must be less than unity in order for the Cybernetic Automaton Learning Theorem, described in the next chapter, to be valid. All of these parameters will be discussed in more detail in the next section. □

It should be noted that SMPFA Operation is not strictly an algorithm since it does not terminate. Similarly, no set of final states is defined for the SMPFA. Both of these characteristics reflect the artificial intelligence aspects of this model. Naturally intelligent systems do not terminate their operation until their biological supporting systems die. This type of termination of operation is outside the operational characteristics of the cybernetic automaton model. Consequently, there is no allowance made for death here.

2.7 General Principles of the Cybernetic Automaton Model

The preceding section presented a precise and detailed description of what a cybernetic automaton is and how it operates. Many of the details are not likely to be intuitively obvious, however. For that reason, this section presents a higher level description of the cybernetic automaton model.

2.7.1 Underlying POA Structure

It is clear from the formal definition in the preceding section that the underlying model for the cybernetic automaton is the POA. This would seem to be an unexpected choice. The PFA and stochastic cellular automaton models are computationally more capable than the DFA which forms the structure of the POA. However, as is shown in later chapters, the cybernetic automaton based on the POA model is sufficient to achieve the goals set for it. Furthermore, it is not at all clear how structural adaptation should work in a PFA based model. If new transitions and states are created only for novel situations, then how are multiple transitions created that leave the same state on the same input symbol? Otherwise, what mechanism controls when new transitions and states are created, and is it desirable that they be created in familiar situations? These become unnecessary complications to using the PFA as the underlying computational model.

Existing neural network models already possess more of the desired characteristics than automaton models. Consequently, they would seem to be the best starting place, and indeed early work on this model was directed toward a neural network based model. It was quickly found that the

complexity of the set of properties to be emulated contradicted the principle that processing at a single neuron should be very simple. Rather than account for all learning properties at the level of a single neuron, it was apparent that they should be accounted for at an organizational level. Seeking a good way of characterizing the operation of the network as a whole has led to an automaton based model.

2.7.2 Mealy vs. Moore POA

Given that the model is built on an automaton structure, it must next be determined whether it will be a Mealy machine or a Moore machine regarding its output. Here, the formal definition makes it clear that outputs follow the Mealy model. It has already been shown that Mealy and Moore POAs are equivalent, so it makes no computational difference which model is used. The Mealy model, however, is closer to the psychological characteristics of natural systems. They produce an output in response to an input (stimulus), and which output is often mediated by the state of the system. This is one way of saying that the outputs are associated with transitions and hence follow the Mealy model.

2.7.3 Timing Issues

The cybernetic automaton model extends the POA to account for input symbol timing. Provision for simultaneous inputs is made by defining an input set I rather than a single input symbol. Having defined the input to be a set rather than a single symbol, some mechanism must be employed to determine which transition to take. One possibility is to simultaneously take all transitions and to maintain a set of current states. This approach is much like the way NFAs operate. It does, however, make the task of selecting an output symbol somewhat complicated. The transition selection mechanism employed here is to determine a dominant input. Dominance here is a function of the input symbol's strength and will be discussed further in the next sub-section.

While cybernetic automata make no provision for representing exact amounts of time, they do detect the passage of an amount of time taken to be a long time between inputs. The precise length of time which is defined to be long is a parameter of the model denoted by τ. The passage of a length of time longer than τ triggers the taking of a transition labeled ϵ which does not take any input. This is similar to ϵ-transitions found in NFAs, and this similarity is discussed in further detail in the next chapter.

As suggested earlier, stimulus duration in this model is represented by defining two symbols for each stimulus, one representing its onset and the other its offset. This will be discussed in more detail in Chapter 4.

2.7.4 Symbol Strength Issues

In the cybernetic automaton model, each input in the set I has a real-valued strength on the interval $[0, 1]$ associated with it. These strengths are used in a variety of ways. First they are used in determining the dominant input. The dominant input is the one with the greatest input strength.

Input strengths are also used in determining the output symbol strength. Each output symbol also has a strength in the interval $[0, 1]$. The strength of an output symbol is taken to be proportional

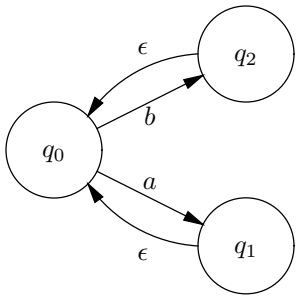

Figure 2.3: Automaton Before Growth

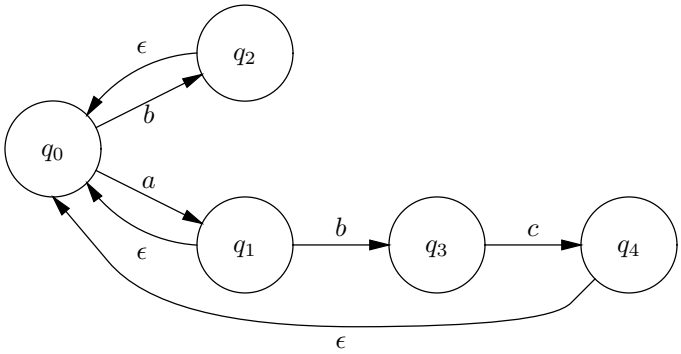

Figure 2.4: Automaton After Receiving a, b, c

to the product of the strength of the dominant input symbol and a monotonically increasing function of the confidence factor on the transition producing that output symbol.

Finally, learning speed is moderated by the strength of input symbols. For symbols leading to reward or punishment states, the strength directly modulates learning. Similarly learning during conditioning is modulated by the strength of the input symbol which triggered conditioning. This will be seen in more detail in the discussion of learning below.

2.7.5 Structural Adaptation

The structure of a cybernetic automaton (its set of states and transitions) is not static. Novel experiences cause the automaton to "grow" new states and transitions to represent them. The growth process is best illustrated by example. If the automaton shown in Figure 2.3 is in state q_0 and receives the input string a, b, c followed by a delay, then the automaton shown in Figure 2.4 results reflecting the creation of two new states. State q_3 represents the symbol b following a and q_4 represents c following a, b. Note that state q_4 has an ϵ-transition to q_0, but q_3 has none. This is because the b was not followed by a delay whereas the c was.

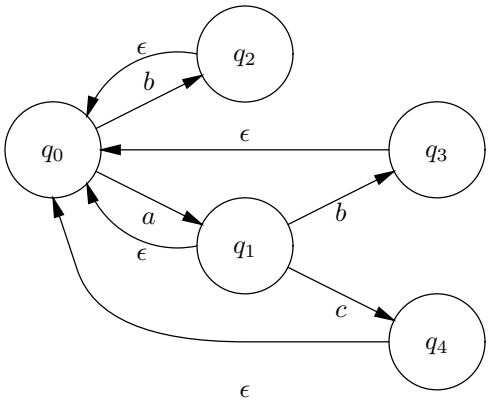

Figure 2.5: Automaton After Receiving a, bc

Simultaneous input symbols create a family of transitions and states. This is illustrated in Figure 2.5 where the automaton has received the input sequence a, bc.

Structural adaptation as defined for cybernetic automata encodes the set of all unique input sequences which have been received by the automaton. The structure of such an automaton can be viewed as a graph where the states are nodes and the transitions are edges. One can then see that growth in a cybernetic automaton leads to a tree structure where the initial automaton (prior to growth) is taken to be the root and ϵ-transitions are back edges. This property is used in the next chapter to prove the completeness of structural adaptation in cybernetic automata.

2.7.6 Learning Mechanisms

Clearly the most extensive part of the definition of the cybernetic automaton model is its learning mechanisms. The mechanisms defined here account for both supervised and unsupervised learning.

Supervised Learning

The form of supervised learning used here is a generalization of that found in adaptive PFAs. By defining states to be reward or punishment rather than a single global signal, several additional capabilities are gained. First, it allows for points in time when the automaton is being neither rewarded nor punished. It also allows some input symbols to be rewarding sometimes, punishing sometimes and neutral at others. Finally, it allows for more than one input to be rewarding or punishing.

As for adaptive PFAs, supervised learning takes the form of probability modification triggered by reward or punishment signals. When a reward or punishment state is entered, the probabilities of recently selected output symbols are increased or decreased respectively. For the reward of the

symbol o^*, the change in probabilities is given by:

$$P^{\Delta}_{q,a}(o) = \begin{cases} \frac{P^{\Delta}_{q,a}(o)+\zeta ts_d(1/C(q,a))}{1+\zeta ts_d(1/C(q,a))} & \text{if } o = o^* \\ \frac{P^{\Delta}_{q,a}(o)}{1+\zeta ts_d(1/C(q,a))} & \text{otherwise.} \end{cases}$$

For punishment the effect is reversed as follows:

$$P^{\Delta}_{q,a}(o) = \begin{cases} \frac{P^{\Delta}_{q,a}(o)}{1+\zeta ts_d(1/C(q,a))} & \text{if } o = o^* \\ \frac{P^{\Delta}_{q,a}(o)+\left(\frac{1}{|\Delta|-1}\zeta ts_d(1/C(q,a))\right)}{1+\zeta ts_d(1/C(q,a))} & \text{otherwise.} \end{cases}$$

The definition of recently selected output symbols is all those symbols which have been selected since the last application of reward or punishment or the last input delay.

Because natural learning of a response tends to generalize across situations, it is desirable that changes in probability be propagated to other transitions on the same input symbol. This is done in the cybernetic automaton model. The degree to which these changes are propagated is controlled by the parameter ν. Similarly, more recent actions have a closer association with reward and punishment reinforcements than do older outputs. Consequently, the more recent ones should receive a greater change in probability. The factor t in the equations controls this. It starts at unity and is decreased by a factor of κ for each marked output reinforced starting at the most recent and moving progressively farther back in time.

Unsupervised Learning

While supervised learning in cybernetic automata is similar to that in other models, the form of unsupervised learning is somewhat unusual. The basic rule for unsupervised learning in cybernetic automata is

> When the output of the system changes (including the inactivation of output upon an input delay), probability modifications are propagated from the previous transition through expectation links. The degree of probability change and propagation is inversely related to the confidence factor of a transition.

This basic rule specifies the existence of expectations and confidence factors. These are the E and C functions in the formal definition.

Expectations encode the tendency for one input to be followed by another or for two inputs to occur simultaneously. The expectation itself is a function of two transitions whose range is the interval $[0, 1]$. When one of a pair of transitions t_1 is taken, the expectation between them is modified according to the following equation:

$$E(t_1, t_2) = \alpha x + (1 - \alpha)E(t_1, t_2)$$

where

$$x = \begin{cases} 1, & \text{if } t_2 \text{ is taken} \\ 0, & \text{otherwise.} \end{cases}$$

In a sense, expectations are similar to an estimation of the conditional probability that the t_2 will be taken if t_1 is taken weighted toward more recent occurrences of t_1. This is not a completely accurate analogy since the expectations are symmetric (i.e. $E(t_1, t_2) = E(t_2, t_1)$) and the same is not in general true of conditional probabilities.

There is another type of "expectation" link which has not yet been mentioned. That is the copy link. When a transition output probability set is copied in the creation of a new transition, a copy link is created between the old and new transition. Here the value of the E function is fixed and is setonly so that the function will exist for those transitions. Copy links are used in the same way as other expectation links in propagating probability changes. However, they are not used for changes to confidence factors discussed below.

The confidence factor for a transition represents how strongly the transition "believes" that its set of output probabilities is correct. This factor is given by the function C which is a real-valued function of a transition and has a range which is the interval $[0, \infty)$. It is increased by changes to the transition's probabilities (including unsupervised changes) and decreased by changes to expectations attached to the transition. These changes are summarized by the following equation:

$$\Delta C(t) = \begin{cases} -C(t)\beta|\Delta E(t,\cdot)| & \text{for changes in } E(t,\cdot) \\ \gamma s & \text{for changes in probability of strength } s. \end{cases}$$

As expressed in the basic rule of unsupervised learning, changes in the output probabilities associated with a transition are triggered by changes in the output of the automaton. The changes are made so as to increase the probability of producing the most recent (prior to the triggering change) output symbol o_l and are specified by the following equations:

$$P^{\Delta}_{q,a}(o) = \begin{cases} \frac{P^{\Delta}_{q,a}(o)+\gamma s(1/C(q,a))}{1+\gamma s(1/C(q,a))} & \text{if } o = o_l \\ \frac{P^{\Delta}_{q,a}(o)}{1+\gamma s(1/C(q,a))} & \text{otherwise} \end{cases}$$

where $s = s_d$ for the transition from which the changes propagate and $s = s'(1/C(q', a'))$ for changes which propagate from transition (q', a') with change strength s'.

In order to illustrate how the unsupervised mechanisms operate, Figure 2.6 shows an automaton in which learning will take place. The ϵ-transitions have been omitted from Figure 2.6 for clarity. The dashed arcs in the figure indicate the expectation links. Suppose that the automaton in initially in state q_0 and then receives the input sequence a, b, c. Further suppose that it produces output symbol d for both inputs a and b and that it produces output symbol e for input c. Because the symbol a does not occur simultaneously with c, the expectation $E(t_1, t_2)$ is reduced after the first input symbol causing a decrease in $C(t_1)$ and $C(t_2)$. After the second input symbol, the expectation $E(t_1, t_4)$ is increased since a is followed by b. Similarly, $C(t_1)$ and $C(t_4)$ are decreased here too. The copy link $E(t_2, t_3)$ is not changed. After the c is received, the expectation $E(t_4, t_5)$ is increased again decreasing $C(t_4)$ and $C(t_5)$. The receipt of c also triggers learning to take place since it causes the output to change. Probability changes are first propagated along the links $E(t_1, t_4)$ and $E(t_3, t_4)$. These changes are made with strength s_d and cause increases in $C(t_1)$ and $C(t_3)$. Probability changes are then propagated along the $E(t_1, t_2)$ link with strength dependent on the confidence $C(t_1)$. If $C(t_1)$ was already high, then little learning will take place in t_2, but if $C(t_1)$ was low, then a high degree of learning will take place in t_2.

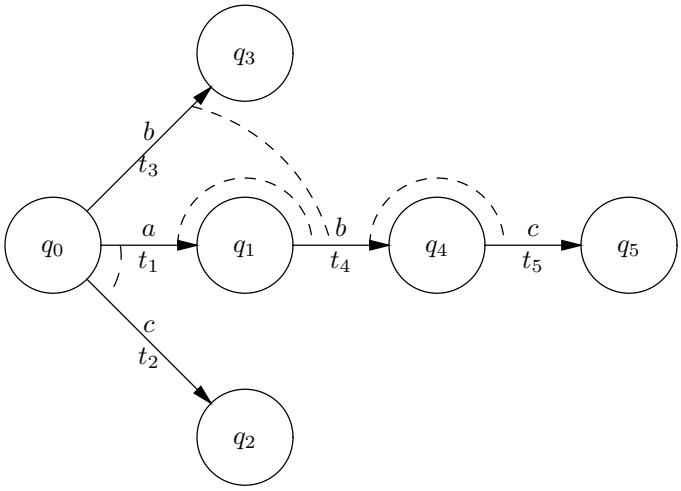

Figure 2.6: Example of Unsupervised Learning

Chapter 3

Learning Properties of Cybernetic Automata

It is natural to ask what capabilities are possessed by the cybernetic automaton model presented in the previous chapter. Since, in the absence of learning, the model has the same characteristics as probabilistic output automata, the question becomes "what can be learned by the model?" This chapter addresses that question. It presents two theorems which, taken together, demonstrate that the cybernetic automaton can develop into an equivalent of any desired POA. The first of these theorems states that an equivalent structure of states and transitions can be created, and the second establishes that the correct probabilities on outputs can be learned.

3.1 Some Preliminaries

Prior to presenting these two theorems, some intermediate definitions and results will be useful. The first of these is the definition of a pseudo-deterministic finite automaton. The pseudo-DFA is created by adding a non-deterministic element into the DFA structure, but one which is restricted to be used deterministically. The following definition gives the precise formulation.

DEFINITION 6 *A pseudo-deterministic finite automaton (pseudo-DFA), A, is given by*

$$A \equiv \langle Q, Q_\epsilon, \Sigma, \delta, q_0, F \rangle$$

where Q, Σ, q_0, and F are the same as for a DFA. The set of states, Q_ϵ, are additional states which have unique transitions. The transition function, $\delta : (Q \times \Sigma) \cup (Q_\epsilon \times \{\epsilon\}) \mapsto Q \cup Q_\epsilon$, admits the possibility of spontaneous transitions. It should be noted that these spontaneous transitions are only found leaving states in Q_ϵ and that no other transitions leave those states. □

The real utility of the pseudo-DFA is seen to arise when placed in a restricted form which will then be shown to be equivalent to the DFA. This restricted form, called the afferent pseudo-DFA, is defined as follows.

DEFINITION 7 *An afferent pseudo-DFA, A, is a pseudo-DFA with the following properties:*

1. *All transitions on input symbols leaving states of distance k lead to states of distance $k+1$, where a state's distance is given by the minimum number of transitions which must be taken to reach it from q_0.*

2. *Each ϵ-transition leaving a state, q, leads to a state from which q can be reached without taking an ϵ-transition.*

3. *State q_0 has no input symbol labeled transitions entering it. All other states are entered by exactly one transition on an input symbol.* □

A DFA can be shown in the form of a graph where states appear as nodes and transitions as edges. If an afferent pseudo-DFA, A, is presented in this way, then $A' = \langle Q, \Sigma, \delta, q_0, F \rangle$ is a tree with q_0 as its root. In the graph for A, the ϵ-transitions form back edges in the tree of A'.

Now the equivalence of afferent pseudo-DFAs and DFAs can be established.

LEMMA 1 *Let A be a DFA accepting language L. Then there exists an afferent pseudo-DFA, A', which also accepts L.*

PROOF: Apply the following algorithm to transform A to A'.

Construct a spanning tree, T, of the graph representing A rooted at q_0.

Initialize the set, P, to include all transitions in A.

Repeat until the set P is empty:

1. Select a transition from state p to state q on input symbol x from the set P.
2. If the transition is represented by an edge in T then remove it from P.
3. Else if q or a q-equivalent state is an ancestor of p in T, then

   Remove the transition from P and δ.
   Create a unique state $q_{\epsilon i}$ and add it in the set Q_ϵ and to T.
   Add to δ and T the transition from p to $q_{\epsilon i}$ on x.
   Add to δ an ϵ-transition from state $q_{\epsilon i}$ to p's q-equivalent ancestor.
4. Else

   Remove the transition from P and δ.
   Create a new q-equivalent state, q', adding it to Q and to T.
   Add a new transition from p to q' on x to δ and as an edge to T.
   For each transition originally in A from q to r on y, add to P and δ a transition from q' to r on y.

Three observations about this algorithm will establish the proof. The first of these is that the algorithm terminates. In order to show this, one first observes that in each iteration of the main loop one transition is removed from P. So it suffices to show that only a finite number of transitions are ever added to P. Clearly, P initially has at most $|Q||\Sigma|$ transitions. Transitions are only added to P in step 4 when new states are created. For each state, there can be no more than $|\Sigma|$ transitions added. That there are only a finite number of new states created is clear from the tree structure of T. There can be no path from q_0 to any leaf node longer than $|Q|$ transitions without there being more than one q-equivalent state for some q on the path and the algorithm will not create any path with more than one q-equivalent state in it. Therefore, the tree will have maximum depth of $|Q|$ and consequently have, at most, $\mathcal{O}(|\Sigma|^{|Q|})$ states.

The second observation is that the resulting automaton is an afferent pseudo-DFA. This too is made apparent by the tree structure of T and the fact that all ϵ-transitions are created to be represented by back edges in T.

The third observation is that the resulting afferent pseudo-DFA is equivalent to A. Here the important consideration is that the automaton is only changed in steps 3 and 4. In step 3, an innocuous state is added to an existing transition. In step 4, the added state mimics the function of an existing state not changing the overall function of the automaton. Consequently, each iteration of the main loop changes the automaton only in ways which do not affect the language accepted by A.

It is clear from these three observations that the algorithm given here transforms any DFA A into an equivalent afferent pseudo-DFA A', so for any DFA there exists an equivalent afferent pseudo-DFA. Q.E.D. □

At this point, it is important to note that there exists an inconsistency between the usage of ϵ-transitions in the afferent pseudo-DFA and in cybernetic automata. In both cases, they are taken spontaneously without using an input symbol. The difference is a function of timing. If no input symbol arrives within the time parameter τ, the cybernetic automaton behaves like the afferent pseudo-DFA, but if one does, the cybernetic automaton extends its structure. Because time is not defined in any significant way for the afferent pseudo-DFA, the ϵ-transitions in cybernetic automata are best viewed as generalizations of those in DFAs.

3.2 Learning of the Structure of a POA

This section presents the first of the two main results in this chapter. Here it is shown that for any POA, the cybernetic automaton can, given a suitable input sequence, develop itself into an automaton with a structure equivalent to the desired POA.

THEOREM 5 *Let A be a POA. There exists an input string x which, when provided as input to a cybernetic automaton with a single state, develops the cybernetic automaton into an automaton A' with a structure equivalent to A.*

PROOF: Let the structure of $A \equiv \langle Q, \Sigma, \Delta, \delta, \lambda, q_0, F\rangle$ be given by $A_S \equiv \langle Q, \Sigma, \delta, q_0\rangle$. By Lemma 1, there exists an afferent pseudo-DFA, A'_S, which is equivalent to A_S. The following algorithm exhibits the construction of an input sequence, x, which will build an automaton which is equivalent to A'_S under the input alphabet Σ.

1. Let A' initially be given by $\langle \{q_0\}, \Sigma \cup \{Reset\}, \emptyset, \emptyset, \emptyset, q_0, \emptyset, \emptyset \rangle$.
2. Let L be a list of the states Q'_S of A'_S (excluding q_0) sorted in ascending order of their distance from q_0.
3. For each state q in L

   Let the unique transition entering q be from state p on input symbol a.
   Let y be the input string which takes the automaton from state q_0 to state p.
   Append y to the end of the string x.
   Append a to the end of the string x.
   Append $Reset$ to the end of the string x.
   Append a time delay to the end of the string x.
4. For each state q_ϵ in $Q'_{\epsilon S}$

   Let the unique transition entering q_ϵ be from state p on input symbol a.
   Let the ϵ-transition from q_ϵ terminate in state r.
   Let y be the input string which takes the automaton from state q_0 to state r.
   Let z be the input string which takes the automaton from state r to state p.
   Append y to the end of the string x.
   Append a time delay to the end of the string x.
   Append z to the end of the string x.
   Append a to the end of the string x.
   Append a time delay to the end of the string x.

When x is applied to an A' initially be given by $\langle \{q_0\}, \Sigma \cup \{Reset\}, \emptyset, \emptyset, \emptyset, q_0, \emptyset, \emptyset \rangle$, the cybernetic automaton, A', develops into an automaton in which A'_S is embedded. All of the states in A' not in A'_S are entered on the input symbol $Reset$ which is not in Σ. Consequently, under the input alphabet Σ, A' is equivalent to A. Q.E.D. □

The inconsistency of ϵ-transition usage arises here again. In order for the ϵ-transitions to properly emulate those in the afferent pseudo-DFA, the time parameter τ must be smaller than the interval of time between symbols arriving to the DFA. However, in order for training to proceed properly, τ must be larger than the interval of time between symbols arriving during training. Consequently, either the training sequence must be fed to the automaton faster than inputs after training, or the value of τ must be larger during training than during operation after training.

3.3 Learning of the Output Probabilities of a POA

There are two elements of a POA, the sets of states and transition and the set of probability distributions, both of which must be learned in order to learn the complete POA. The previous section showed that cybernetic automata can learn the structure (the sets of states and transitions) of a POA, and this section shows that it can learn the set of probability distributions.

3.3.1 Definition of the Teacher T

Unlike the learning of an automaton's structure, the learning of its output probability distributions requires an interactive teacher. This sub-section defines one such teacher which will be used in the lemma in the next sub-section.

DEFINITION 8 *A teacher T takes three parameters. The first, A, is a description of a reset and R–P augmented POA which represents the current configuration of the cybernetic automaton being trained. The second argument is the set of target output probability distributions $\hat{P}^{\Delta}_{Q\times\Sigma}$. The third argument is an error measure ϵ which determines how close to the target probability distribution the cybernetic automaton is required to get. T takes inputs which are symbols from the output set of A and produces outputs which are symbols from input set of A. Finally, T operates according to the following procedure, during which it simulates the operation of the cybernetic automaton $\tilde{A}$ equivalent to A.*

Let X be a string which takes $\tilde{A}$ through all of its transitions starting in and finishing in state q_0.

Repeat until $|\hat{P}^{\Delta}_{(q,x)}(z) - \tilde{P}^{\Delta}_{(q,x)}(z)| < \epsilon$ for all q, x and z

For each symbol x in X

Output symbol x with strength 1.0 to the automaton being trained.

Input symbol z from the automaton being trained.

If $\hat{P}^{\Delta}_{(q,x)}(z) \geq \tilde{P}^{\Delta}_{(q,x)}(z)$ then

Let $\delta = \hat{P}^{\Delta}_{(q,x)}(z) - \tilde{P}^{\Delta}_{(q,x)}(z)$ and let y be a symbol such that $\tilde{P}^{\Delta}_{(q,x)}(y) - \hat{P}^{\Delta}_{(q,x)}(y) \geq \frac{\delta}{|\Delta|-1}$.

Let $\alpha = \min\left[\frac{\delta}{1-\hat{P}^{\Delta}_{(q,x)}(z)-\delta}, \frac{\delta}{(|\Delta|-1)\hat{P}^{\Delta}_{(q,x)}(y)-\delta}\right]$.

Output symbol Reward with strength $\min\left[1, \frac{\alpha}{\zeta(1/C(q,x))}\right]$.

else

Let $\delta = \tilde{P}^{\Delta}_{(q,x)}(z) - \hat{P}^{\Delta}_{(q,x)}(z)$ and let y be a symbol such that $\hat{P}^{\Delta}_{(q,x)}(y) - \tilde{P}^{\Delta}_{(q,x)}(y) \geq \frac{\delta}{|\Delta|-1}$.

Let $\alpha = \min\left[\frac{\delta}{\tilde{P}^{\Delta}_{(q,x)}(z)-\delta}, \frac{\delta}{1-(|\Delta|-1)\tilde{P}^{\Delta}_{(q,x)}(y)-\delta}\right]$.

Output symbol Punish with strength $\min\left[1, \frac{\alpha}{\zeta(1/C(q,x))}\right]$. □

3.3.2 Errors in the Probability Distributions

The discussion of learning probability distributions below requires a definition of the error in a probability distribution. This sub-section presents that as well as a useful lemma regarding the error in an output probability distribution.

DEFINITION 9 *Let e_y denote the absolute error in one component of the probability distribution for a particular transition. Specifically,*

$$e_y = \left| \hat{P}^{\Delta}_{(q,x)}(y) - P^{\Delta}_{(q,x)}(y) \right|.$$

Then let E denote the total absolute error over all of the probability distribution given by

$$E = \sum_{y \in \Delta} e_y = \sum_{y \in \Delta} \left| \hat{P}^{\Delta}_{(q,x)}(y) - P^{\Delta}_{(q,x)}(y) \right|.$$

When a change is made to the probabilities during learning, the change in absolute error is denoted by

$$\Delta e_y = \left[\left| (\hat{P}^{\Delta}_{(q,x)}(y) - P^{\Delta}_{(q,x)}(y) - \Delta P^{\Delta}_{(q,x)}(y) \right| - \left| \hat{P}^{\Delta}_{(q,x)}(y) - P^{\Delta}_{(q,x)}(y) \right| \right]$$

and the change in total error is then the sum of error changes in individual probabilities:

$$\Delta E = \sum_{y \in \Delta} \Delta e_y.$$

□

LEMMA 2 *With teacher, T, each input symbol will cause the total error in probabilities to be reduced unless the probability for the selected output symbol is correct, in which case the total error will not change.*

PROOF: There are three cases to be considered. First, if $\hat{P}^{\Delta}_{(q,x)}(z) = P^{\Delta}_{(q,x}(z)$, then the strength, $s = 0$ and no change in probabilities will take place.

Next, if $\hat{P}^{\Delta}_{(q,x)}(z) > P^{\Delta}_{(q,x)}(z)$, then let $\delta = \hat{P}^{\Delta}_{(q,x)}(z) - P^{\Delta}_{(q,x)}(z)$. Because $\hat{P}$ and P are probability measures and must sum to 1,

$$\sum_{y \in \Delta} \left| \Delta P^{\Delta}_{(q,x)}(y) \right| = 2 \left| \Delta P^{\Delta}_{(q,x)}(z) \right|$$

and there must exist a $y \in \Delta$ for which

$$P^{\Delta}_{(q,x)}(y) - \hat{P}^{\Delta}_{(q,x)}(y) \geq \frac{\delta}{|\Delta| - 1}.$$

The equation determining the reward symbol strength in T leads to the following two properties:

$$0 < s \leq \frac{\delta}{\zeta \left(1/C(q,x)\right) \left(1 - P^{\Delta}_{(q,x)}(z) - \delta\right)}$$

$$0 < \Delta P^{\Delta}_{(q,x)}(z) = \frac{\zeta s (1/C(q,x)) \left(1 - P^{\Delta}_{(q,x)}(z)\right)}{1 + \zeta s (1/C(q,x))} \leq \delta$$

$$\Delta e_z = -\Delta P^{\Delta}_{(q,x)}(z)$$

and

$$0 < s \leq \frac{\delta}{\zeta(1/C(q,x))\left[(|\Delta|-1)P^{\Delta}_{(q,x)}(y) - \delta\right]}$$

$$0 > \Delta P^{\Delta}_{(q,x)}(y) = -\frac{\zeta s(1/C(q,x))P^{\Delta}_{(q,x)}(y)}{1+\zeta s(1/C(q,x))} \geq -\frac{\delta}{|\Delta|-1}$$

$$\Delta e_y = \Delta P^{\Delta}_{(q,x)}(y).$$

Finally, these two values for Δe_z and Δe_y lead to the following bound on ΔE:

$$\begin{aligned}
\Delta E &= \Delta e_z + \Delta e_y + \sum_{\substack{w\in\Delta \\ w\neq z \\ w\neq y}} \Delta e_w \\
&\leq \Delta e_z + \Delta e_y + \sum_{\substack{w\in\Delta \\ w\neq z \\ w\neq y}} |\Delta e_w| \\
&\leq \Delta e_z + \Delta e_y + \sum_{\substack{w\in\Delta \\ w\neq z \\ w\neq y}} \left|\Delta P^{\Delta}_{(q,x)}(w)\right| \\
&\leq -\Delta P^{\Delta}_{(q,x)}(z) + \Delta P^{\Delta}_{(q,x)}(y) + 2\left|\Delta P^{\Delta}_{(q,x)}(z)\right| - \left|\Delta P^{\Delta}_{(q,x)}(z)\right| - \left|\Delta P^{\Delta}_{(q,x)}(y)\right| \\
&\leq 2\Delta P^{\Delta}_{(q,x)}(y).
\end{aligned}$$

Since $\Delta P^{\Delta}_{(q,x)}(y)$ is always negative for $\hat{P}^{\Delta}_{(q,x)}(z) > P^{\Delta}_{(q,x)}(z)$, then ΔE will be negative as well.

The following similar argument establishes the result for $P^{\Delta}_{(q,x)}(z) > \hat{P}^{\Delta}_{(q,x)}(z)$. Let $\delta = P^{\Delta}_{(q,x)}(z) - \hat{P}^{\Delta}_{(q,x)}(z)$. Because $\hat{P}$ and P are probability measures and must sum to 1, there must exist a $y \in \Delta$ for which

$$\hat{P}^{\Delta}_{(q,x)}(y) - P^{\Delta}_{(q,x)}(y) \geq \frac{\delta}{|\Delta|-1}.$$

The equation determining the punishment symbol strength in T leads to the following two properties:

$$0 < s \leq \frac{\delta}{\zeta\left(1/C(q,x)\right)\left(P^{\Delta}_{(q,x)}(z) - \delta\right)}$$

$$0 > \Delta P^{\Delta}_{(q,x)}(z) = -\frac{\zeta s(1/C(q,x))P^{\Delta}_{(q,x)}(z)}{1+\zeta s(1/C(q,x))} \geq -\delta$$

$$\Delta e_z = \Delta P^{\Delta}_{(q,x)}(z)$$

and

$$0 < s \leq \frac{\delta}{\zeta(1/C(q,x))\left[1-(|\Delta|-1)P^{\Delta}_{(q,x)}(y) - \delta\right]}$$

$$0 < \Delta P^{\Delta}_{(q,x)}(y) = \frac{\zeta s(1/C(q,x)) \left(\frac{1}{|\Delta|-1} - P^{\Delta}_{(q,x)}(y) \right)}{1 - \zeta s(1/C(q,x))} \leq \frac{\delta}{|\Delta| - 1}$$

$$\Delta e_y = -\Delta P^{\Delta}_{(q,x)}(y).$$

Finally, these two values for Δe_z and Δe_y lead to the following bound on ΔE:

$$\begin{aligned}
\Delta E &= \Delta e_z + \Delta e_y + \sum_{\substack{w \in \Delta \\ w \neq z \\ w \neq y}} \Delta e_w \\
&\leq \Delta e_z + \Delta e_y + \sum_{\substack{w \in \Delta \\ w \neq z \\ w \neq y}} |\Delta e_w| \\
&\leq \Delta e_z + \Delta e_y + \sum_{\substack{w \in \Delta \\ w \neq z \\ w \neq y}} \left| \Delta P^{\Delta}_{(q,x)}(w) \right| \\
&\leq \Delta P^{\Delta}_{(q,x)}(z) - \Delta P^{\Delta}_{(q,x)}(y) + 2 \left| \Delta P^{\Delta}_{(q,x)}(z) \right| - \left| \Delta P^{\Delta}_{(q,x)}(z) \right| - \left| \Delta P^{\Delta}_{(q,x)}(y) \right| \\
&\leq -2\Delta P^{\Delta}_{(q,x)}(y).
\end{aligned}$$

Since $\Delta P^{\Delta}_{(q,x)}(y)$ is always positive for $\hat{P}^{\Delta}_{(q,x)}(z) < P^{\Delta}_{(q,x)}(z)$, then ΔE will be positive.

Summarizing the three cases, if the probability of producing the selected output symbol is correct, then there is no change in the total error. If the target probability is either greater than or less than the actual probability, then the corresponding reward or punishment causes the change in total error to be negative. Q.E.D. □

At this point it should be noted that this proof makes implicit assumptions of the model. In their strongest form, these assumptions assert that no changes are made to a probability distribution unless made by the reward or punishment signals of the model. This in turn implies that the model parameter $\nu = 0$. The consideration for this assumption are the same as for the requirements on the parameter τ.

3.3.3 Convergence of Output Probability Distributions

This sub-section presents the second main result of this chapter. In particular, if the teacher, T, is allowed to run long enough the set of probability distributions will converge to the desired set with probability 1. In the same way as the afferent pseudo-DFA was augmented with a *Reset* input symbol and corresponding states, the learning of correct output probabilities can be guaranteed if one begins with an $R - P$ augmented POA.

THEOREM 6 *Let A be a POA and A' be the Reset and R-P augmented POA of A. Furthermore, let $P^{\Delta}_{Q \times \Sigma}$ be the output probability distribution of A, and let $\hat{P}^{\Delta}_{Q \times \Sigma}$ be the desired output probability distribution. Then a teacher T with parameter A' will teach a cybernetic automaton initially given by A' the output probability distribution $\hat{P}^{\Delta}_{Q \times \Sigma}$ with probability 1.*

PROOF: From Definition 8 of the teacher T, Definition 5 of the cybernetic automaton and from Definition 9 of the error measure, it is clear that for each output symbol Y produced during training, $\Delta e_y \leq 0$. Furthermore, it is only equal to zero when $\hat{P}^{\Delta}_{(q,a)}(y) = P^{\Delta}_{(q,a)}(y)$. By Lemma 2, the change in error over all output symbols is $\Delta E_y \leq 0$. In other words, each time a symbol is output by the cybernetic automaton, the teacher causes the total error for that transition to decrease if the probability of that symbol is not correct. Lastly, if the teacher continues long enough, each output symbol with a non-zero probability will repeatedly be selected for output and therefore its probability will converge to the desired probability. Consequently, since all of the individual probabilities converge, the probability distribution of the cybernetic automaton converges to $\hat{P}^{\Delta}_{Q,\Sigma}$. Q.E.D. □

3.4 The Cybernetic Automaton Learning Theorem

Taken together, the two theorems presented in this chapter lead to a general learning theorem for cybernetic automata. Specifically, the functionality of any given POA may be learned by a cybernetic automaton with a simple initial structure. A structural equivalent of the desired POA is first learned according to Theorem 5 and then the set of output probability distributions is learned according to Theorem 6. This principle is stated more formally in the following theorem.

THEOREM 7 (The Cybernetic Automaton Learning Theorem) *A functional equivalent of any POA, $A \equiv \langle Q, \Sigma, \Delta, \delta, \lambda, q_0 \rangle$ may be learned by an initial cybernetic automaton given by:*

$$A' \equiv \langle \{q_0, P, R\}, \Sigma \cup \{Reset, Reward, Punish\}, \Delta, \delta', \emptyset, q_0, \{R\}, \{P\}, \emptyset, \emptyset \rangle.$$

PROOF: Following the construction given in the proof of Theorem 5, A' can be trained to possess the structure of an afferent pseudo-DFA equivalent to A. As a second step, this structurally equivalent automaton can then be trained by the teacher T to possess a set of output probability distributions arbitrarily close to those of A. Since the training process of T does not affect the structure of A', the two training techniques may be applied in the order described, treating the structure and output probabilities as independent problems. The resulting A', then, has both a structural equivalence to A and a set of output probability distributions which are arbitrarily close to those of A, so it is functionally equivalent to an arbitrary degree of precision. Q.E.D. □

3.5 Summary

In this chapter, the first main result of cybernetic automata theory is developed. There exists a Cybernetic Automaton Learning Theorem which states that a cybernetic automaton can learn to emulate the operation of any automaton which can be constructed within the framework of cybernetic automata. This theorem is similar in spirit to Rosenblatt's Perceptron Learning Theorem, but applies to a stronger learning model.

Chapter 4

Conditioning in Cybernetic Automata

Psychologists have, for many years, studied learning in natural systems and have cataloged a number of properties of natural learning. Hall (1976) provides an extensive description of many of these. In this chapter, the cybernetic automaton model is subjected to experiments which are based on those used by psychologists. But first it is necessary to examine the relationship between the psychological concepts of stimulus and response and the automata theoretic concepts of input and output symbols.

4.1 Symbolizing Stimuli and Responses

Because stimuli in psychological experiments are generally continuous with onset and offset times, there is an essential mismatch between them and the discrete event input symbols in automata theory. A similar mismatch exists between experimental responses and the output symbols. These mismatches can easily be resolved by defining input symbols which indicate the onset and offset of a particular stimulus and similarly for responses.

Figure 4.1 illustrates a period of time and the actions of two stimuli during that period. Let $A+$ and $B+$ be symbols which indicate the onset of the stimuli A and B respectively. Similarly, let $A-$ and $B-$ be input symbols which indicate the offset of the stimuli. The sequence of input symbols for Figure 4.1 would then be $A+$ $B+$ $B-$ $A-$ at times t_1, t_2, t_3 and t_4 respectively.

During the period of time between the arrival of an $A+$ input and an $A-$ input, all transitions taken, states visited and output symbols selected reflect the continuing active level of the A stimulus. No continued input signaling the fact that A continues to be active is necessary.

Output symbols are treated similarly. For an experimental response which is continuous and has onset and offset times, there are output symbols which indicate the onset and offset of the response. If an offset output symbol is selected for a response which is not currently active, then it is taken to be a null operation. Similarly, if an output is still active when a delay occurs in the input and when an ϵ-transition occurs, then it becomes inactive.

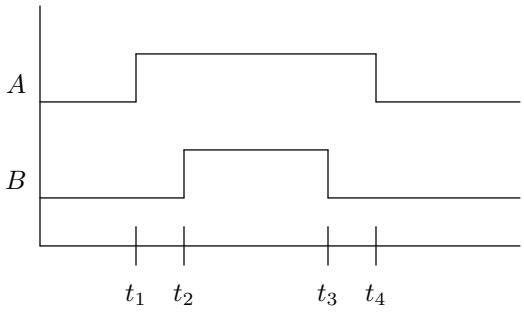

Figure 4.1: Example of Stimuli

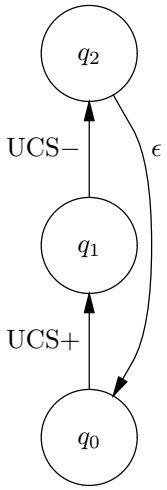

Figure 4.2: Automaton Prior to Classical Conditioning

4.2 First-Order Classical Conditioning

Classical conditioning is one of the simplest forms of learning which has been identified by researchers. It is a learned association between two stimuli such that one learns to produce the response of the other. The most famous example of classical conditioning in the literature is the experiment of Pavlov (Misiak & Sexton, 1966). In this experiment, he repeatedly presented a dog with a sound stimulus followed by a food stimulus. After several such training cycles, the dog began to salivate upon hearing the bell even without food being presented. This type of conditioning is called delayed conditioning because the sound stimulus (the conditioned stimulus or CS) was active at the onset of the food stimulus (the unconditioned stimulus or UCS).

Figure 4.2 shows a cybernetic automaton suitable for being classically conditioned. In particular, the onset of a UCS takes the automaton from state q_0 to q_1 and produces the unconditioned response. In this automaton, there is no transition on the CS, so initially it produces no response.

Table 4.1: Model Parameters for Classical Conditioning

Parameter	α	β	γ	η	ζ	ν	κ
Value	0.05	0.05	0.2	1.0	0.0	0.0	0.0

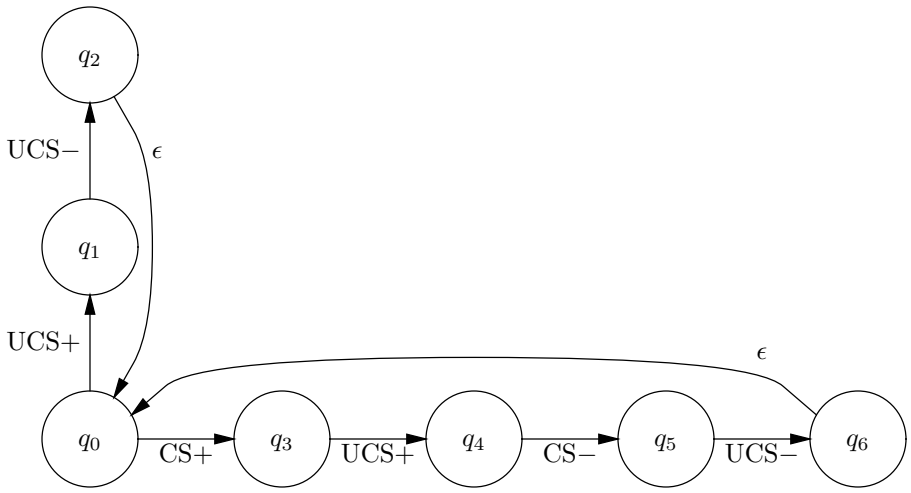

Figure 4.3: Automaton After Delayed Conditioning

This initial automaton is used for all of the experiments reported in this chapter. The confidence values for the UCS+ and UCS− transitions are set to 1,000,000 to represent responses that are innate or are otherwise very difficult to change. The parameters of the model used here are given in Table 4.1.

After presentation of the training sequence, the automaton includes the states q_0–q_6 in Figure 4.3. As the sequence is repeatedly presented, the expectation values for the transitions shown move toward 1 and the output probabilities on the transition from q_0 on CS+ move toward those for UCS+ from q_3 to q_4.

Figure 4.4 shows the results of an experimental test of delayed conditioning in a cybernetic automaton. Each batch of trials consists of ten trials. The point for each batch indicates the percentage of its ten trials in which the automaton produced the UCR upon receipt of the CS and prior to the onset of the UCS. This graph clearly indicates that the automaton did learn to produce the UCR upon receipt of the CS.

4.3 Second-Order Conditioning

For most sets of stimuli, researchers have shown that organisms exhibit second-order conditioning. In this type of leaning, a stimulus, CS2, is conditioned on a CS in the same way as the CS was

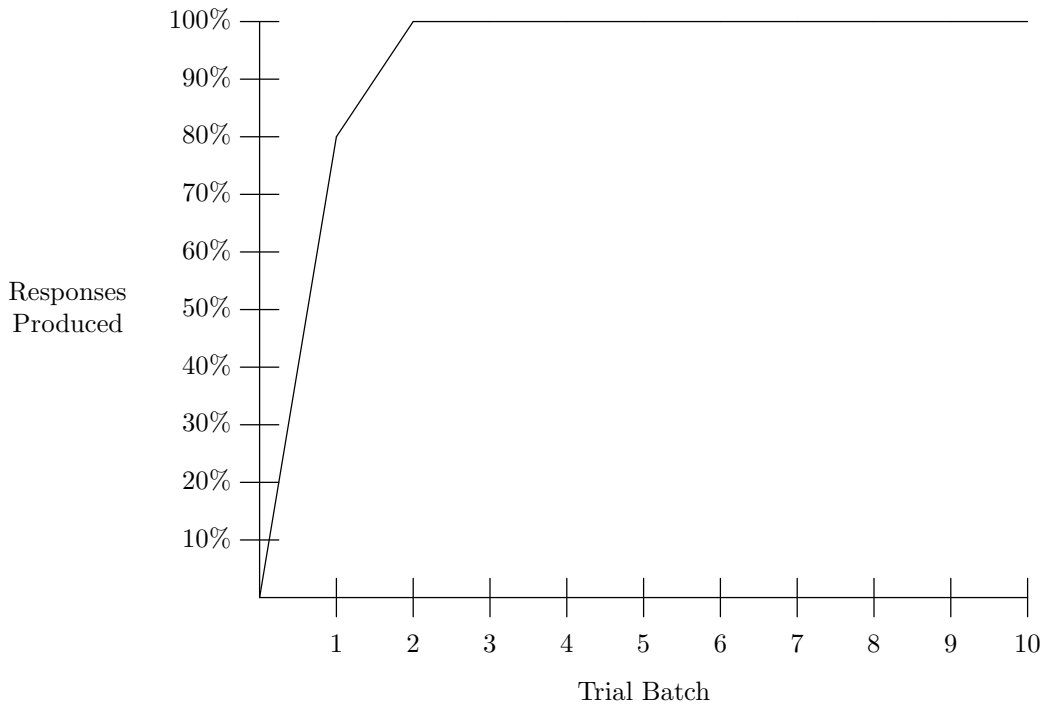

Figure 4.4: Results of Delayed Conditioning

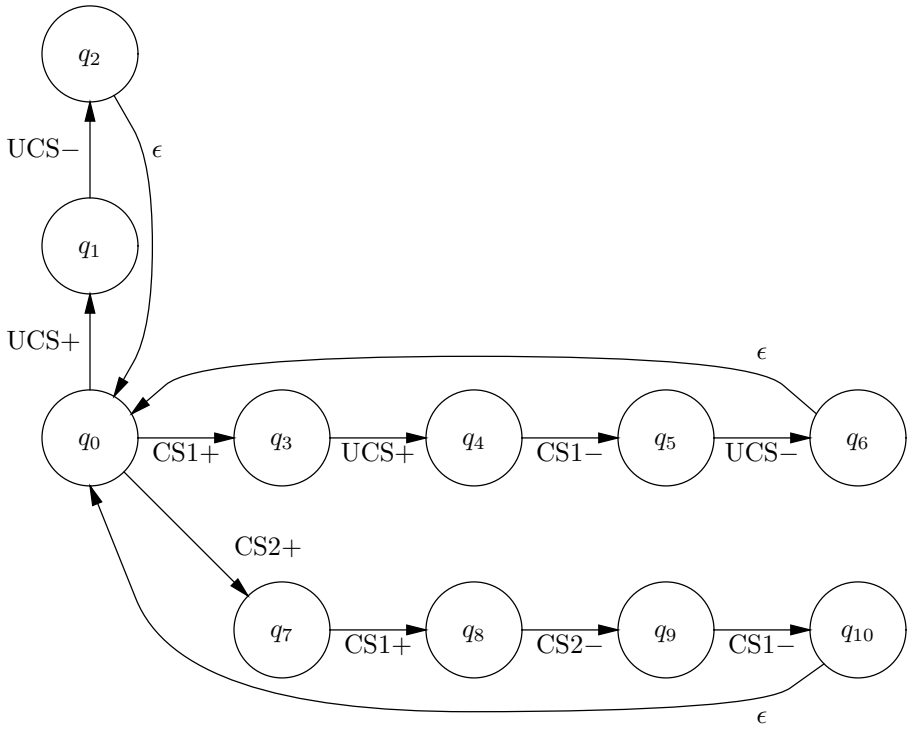

Figure 4.5: Automaton After Delayed Conditioning of CS2

conditioned on the UCS in first-order conditioning. Prior to the conditioning of CS2, the automaton is as in Figure 4.3. After conditioning of CS2, it is as in Figure 4.5.

Figure 4.6 shows the experimental results of delayed second-order conditioning in cybernetic automata. The curve shows the performance of CS2 during its conditioning on CS1 after CS1 had been conditioned until it produced 40 consecutive conditioned responses (CRs).

4.4 Third and Higher-Order Conditioning

Most researchers have found that third and higher-order conditioning is difficult at best. The basic conclusion appears to be that third-order conditioning is possible in some cases, depending upon what stimuli are being conditioned, but that fourth-order conditioning is rarely if ever possible.

Cybernetic automata theory gives some insight into why this phenomenon might occur. Consider the conditioning of CS1 on UCS in Figure 4.3. As conditioning proceeds, $P^{\Delta}_{(q_0,CS1+)}(UCR)$ increases, approaching $P^{\Delta}_{(q_3,UCS+)}(UCR)$ until some criterion of conditioning is reached. At this point, $P^{\Delta}_{(q_0,CS1+)}(UCR)$ is still less than $P^{\Delta}_{(q_3,UCS+)}(UCR)$, but will probably be greater than the

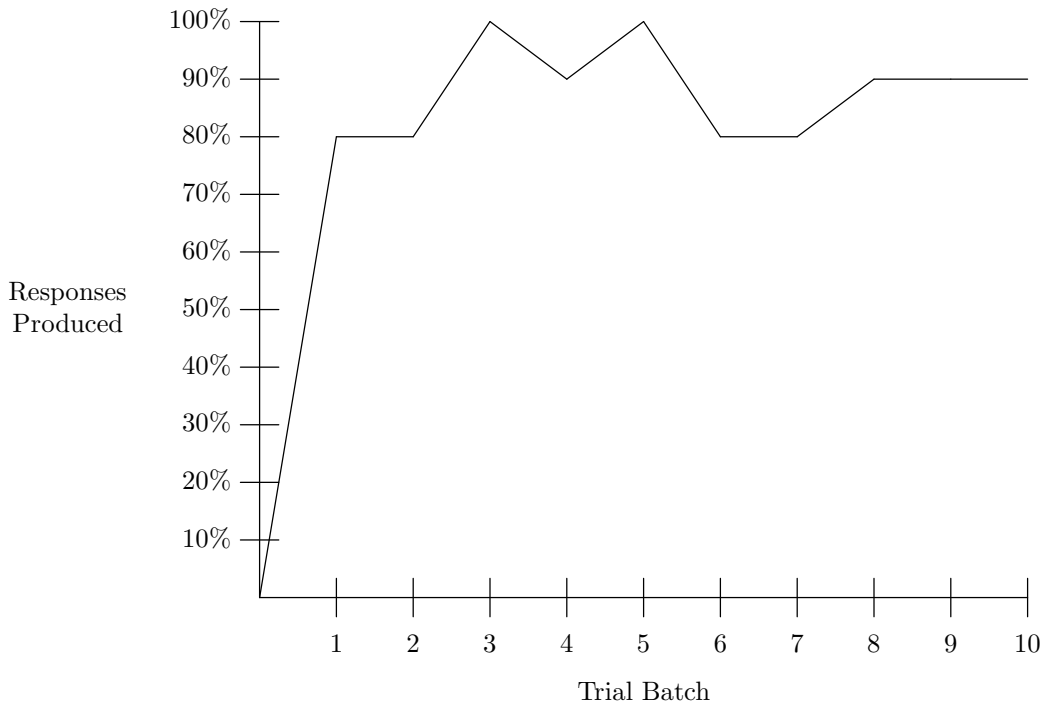

Figure 4.6: Results of Delayed Second-Order Conditioning

probability necessary to satisfy the experimental conditioning criterion. To see this, one must recognize that on the average $P^{\Delta}_{(q_0,CS1+)}(UCR)$ reaches and passes the necessary probability in the middle of the set of trials which establish satisfaction of the conditioning criterion. Therefore, during half of those trials, $P^{\Delta}_{(q_0,CS1+)}(UCR)$ is expected to move between the necessary probability and $P^{\Delta}_{(q_3,UCS+)}(UCR)$.

The same process occurs during second-order conditioning where $P^{\Delta}_{(q_0,CS2+)}(UCR)$ moves toward $P^{\Delta}_{(q_7,CS1+)}(UCR)$. Since $P^{\Delta}_{(q_7,CS1+)}(UCR) < P^{\Delta}_{(q_3,UCS+)}(UCR)$ second-order conditioning takes place more slowly than first-order conditioning does. Furthermore, because the probabilities move more slowly, $P^{\Delta}_{(q_0,CS2+)}(UCR)$ will be much closer to the necessary probability at the time that the conditioning criterion is satisfied.

In regards to third-order conditioning, if the first and second-order conditioning take place fast enough, then $P^{\Delta}_{(q_{11},CS2+)}(UCR)$ will be high enough that CS3 will yield responses to satisfy the conditioning criterion before the experiment is terminated. Otherwise, $P^{\Delta}_{(q_0,CS3+)}(UCR)$ will not reach the necessary probability before the experiment is terminated in enough subjects to be statistically significant. This is in keeping with the findings of Pavlov who reported that third-order conditioning only took place when a strongly aversive stimulus (like shock) was the UCS.

4.5 Latent Inhibition

Experiments have shown that biological organisms learn in ways other than strictly the choice of response to stimulus. One of the phenomena which illustrate this is latent inhibition. Two others, sensory preconditioning and silent extinction, are discussed in later sections. In principle, latent inhibition occurs when a stimulus resists being conditioned because of an expectation that it will occur alone. In a typical latent inhibition experiment, the subject is presented with a number of trials of the CS alone, followed by a number of trials conditioning the CS on the UCS. The control subject has only the same number of trials conditioning the CS on the UCS. In such experiments, experimental subjects show a lower degree of conditioning than control subjects.

Figure 4.7 shows an automaton on which a latent inhibition experiment has been run. After the presentation of CS only trials and before conditioning of CS on UCS, a high degree of confidence exists on the transition from state q_0 on CS+. As the conditioning of the CS on the UCS proceeds, this high degree of confidence resists change, causing the CS to be conditioned more slowly to the UCS. Consequently, if an automaton exhibits substantial conditioning after n conditioning trials without latent inhibition, it will exhibit a much lower level of conditioning after n trials with latent inhibition.

The results of experimental simulations of latent inhibition in cybernetic automata can be seen in Figure 4.8. The control results are taken from Figure 4.4. In this experiment, the automaton receives the same conditioning training set as the control automaton, but after receiving 50 trials of CS-only stimulus. These results clearly show that conditioning in the system was substantially impaired by the latent inhibition.

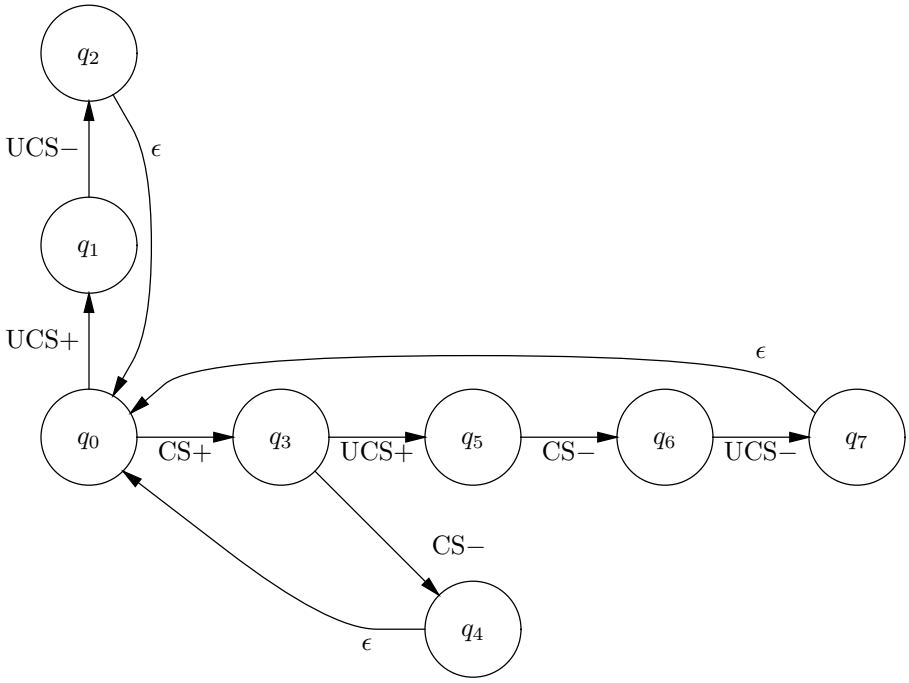

Figure 4.7: Automaton After CS-UCS Conditioning

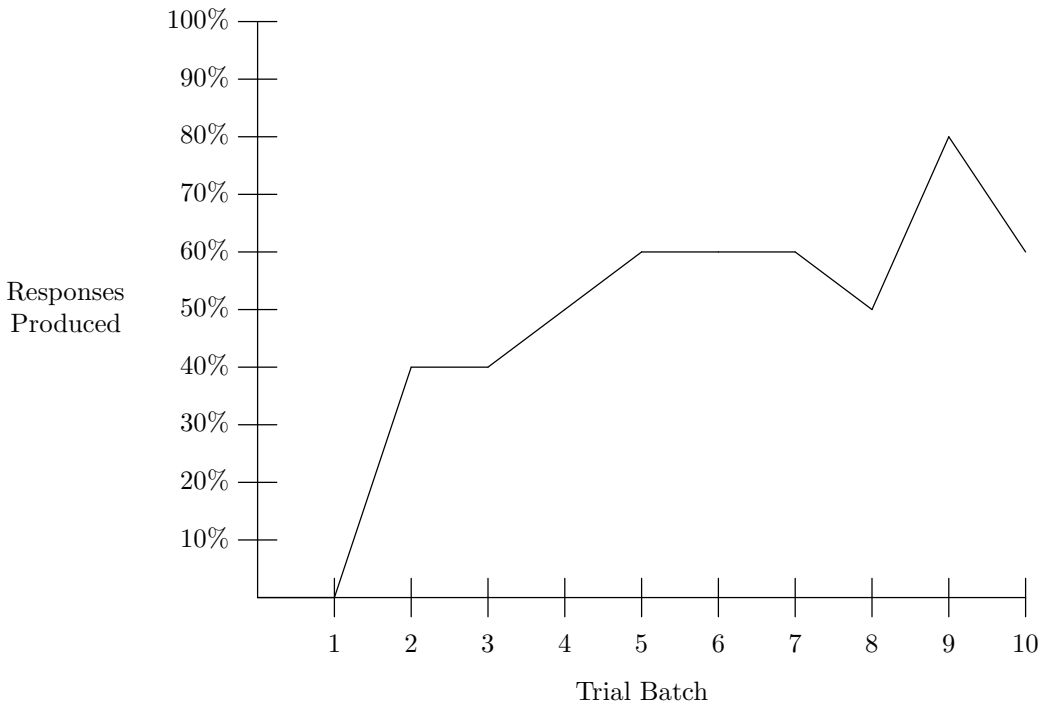

Figure 4.8: Results of Latent Inhibition

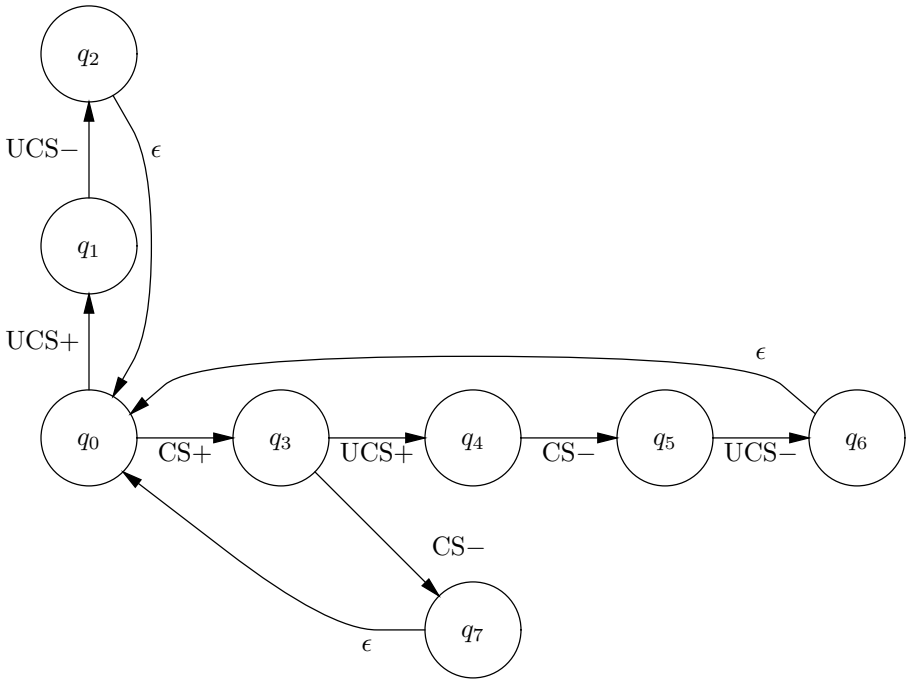

Figure 4.9: Automaton During Extinction

4.6 Extinction

All of the learning described until now has been of the form that the system accumulates knowledge of its environment which then affects its responses. Organisms can also, in effect, "un-learn." If a CS is conditioned on a UCS to produce a CR and the CS is presented alone for many trials, then the system will no longer respond with the CR to the CS. This is called extinction. In extinction, previously learned responses are extinguished. Extinction has many subtle and diverse properties, some of which are discussed in the following sections.

In cybernetic automata, extinction arises due to violated expectations which are established during conditioning. Figure 4.9 illustrates the delay conditioned automaton during extinction. In the early stages of extinction, the transition from state q_3 on input symbol UCS+ has a high degree of expectation. However, since it is not taken during extinction, the probability that CS+ will produce UCR is decreased.

Results of an experimental simulation are shown in Figure 4.10. An automaton structurally identical to that in Figure 4.3 was presented with a number of trials of CS only. As in the other simulations, each batch of trials consists of ten trials and the percentage of those trials, which resulted in UCR being produced as output, is plotted. Conditioning continued until 20 consecutive conditioned responses were produced prior to the presentation of CS-only trials.

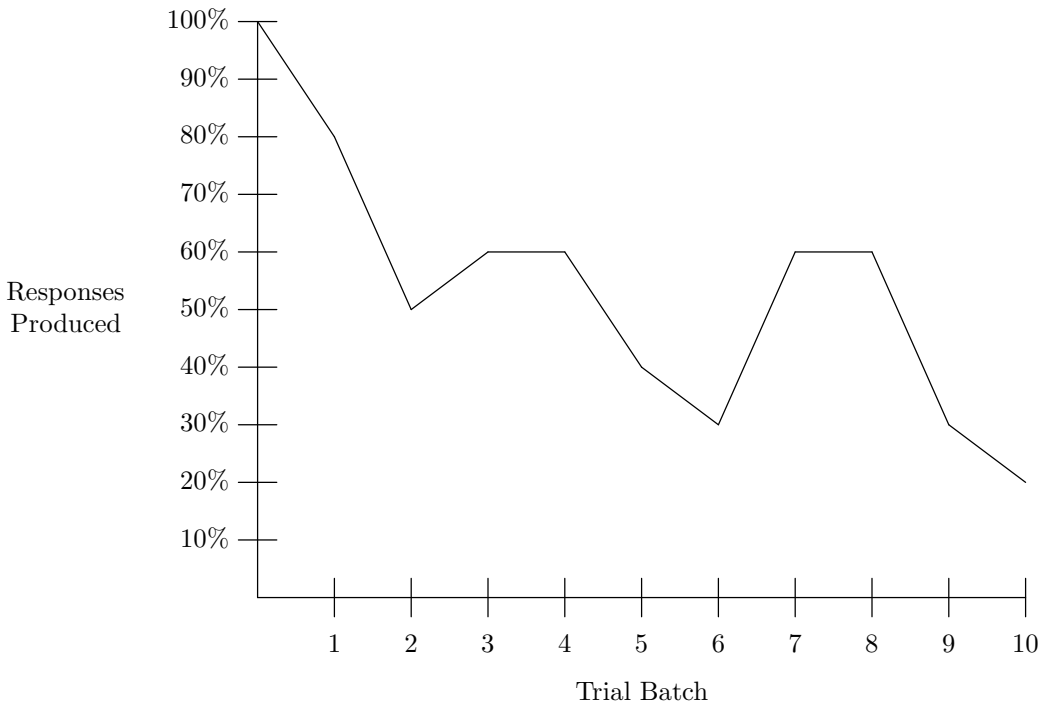

Figure 4.10: Results of Extinction on Delay Conditioning

4.7 Silent Extinction

Another property of extinction which deserves examination is silent extinction, the tendency for additional presentations of the CS alone to "strengthen" the extinction. Suppose that n extinction trials are sufficient to achieve some level of extinction and m conditioning trials are necessary before the conditioned response can be reestablished. Then if more than n extinction trials are presented, then more than m conditioning trials will be required to reestablish the conditioned response.

It is clear the cybernetic automata also exhibit this property. Indeed, the circumstances of silent extinction can be described as extinction followed by latent inhibition. In other words, the automaton first decreases the probability on the dominant output symbol as expectation is violated. Then as expectation that the CS is presented alone increases, the CS becomes resistant to conditioning.

4.8 Partial Reinforcement

In each of the experimental situations described thus far, for each presentation of the CS, there was also a presentation of the UCS. Consequently, there was an unequivocal indication, or reinforcement, of the conditioned stimulus. Some important properties emerge when the UCS is presented only part of the time and the CS is conditioned with paritial reinforcement. The two most fundamental properties of partial reinforcement are that 1) conditioning is weaker and slower and 2) conditioning is more resistant to extinction.

That conditioning in a cybernetic automaton would be weaker and slower in partial reinforcement is readily apparent from the conditioning function. During trials when reinforcement occurs, the probability of the CS producing the CR is increased, and the expectation of CS followed by UCS is also increased. Trials without reinforcement, however, decrease these parameters. Consequently, the probabilities for a transition on CS+ move more slowly toward those for the transition on UCS+. Similarly, for a given number of trials, even when only counting those trials with reinforcement, partially reinforced conditioning leads to lower probabilities of conditioned response.

The key to understanding why conditioning is more resistant to extinction under partial conditioning is held in the expectation paramenters mentioned above. Because partial conditioning leads to smaller expectation values of the CS+ symbol being followed by the UCS+ symbol, extinction in CS-only trials is slower. Therefore, more extinction trials are required to achieve the same degree of extinction.

These theoretical observations are illustrated in Figures 4.11 and 4.12. The first of these shows conditioning for 40 reinforcements at several different probabilities of reinforcement. The results confirm that conditioning is slower and weaker for lower probabilities of reinforcement. Figure 4.12 shows the results of 100 extinction trials for each of the partial reinforcement cases. These curves, too, confirm the theoretical considerations which predict that extinction is slower for lower reinforcement probabilities.

4.9 Simultaneous Conditioning

None of the experiments discussed thus far utilize the capability of cybernetic automata to handle simultaneous inputs. Furthermore, first-order conditioning can be accomplished by other timing

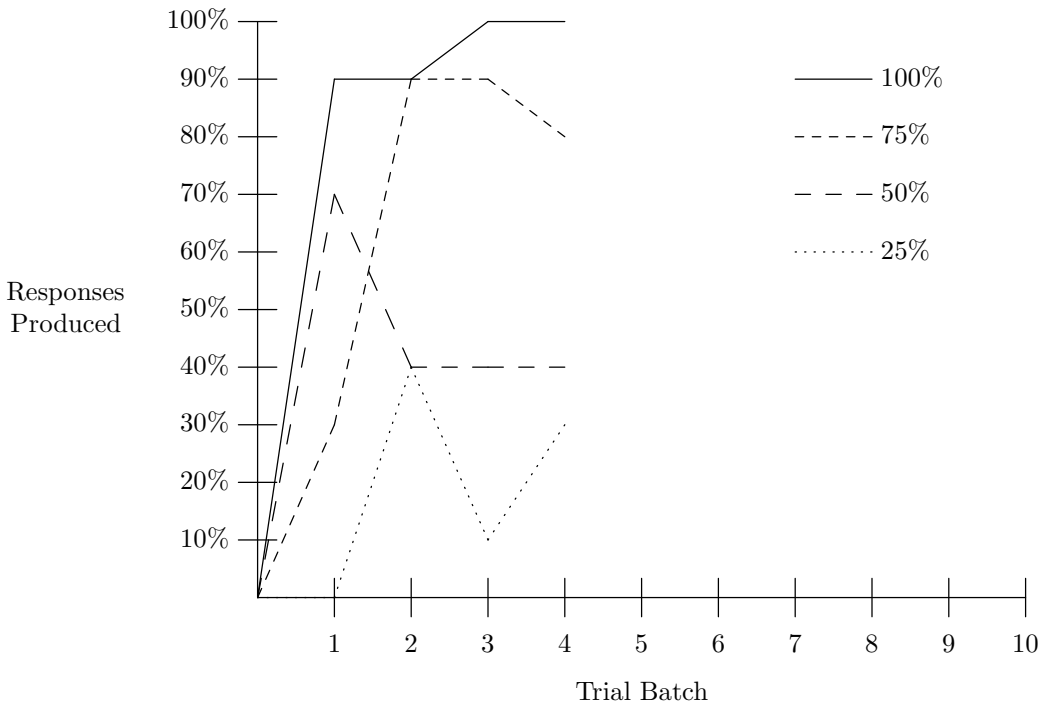

Figure 4.11: Conditioning with Partial Reinforcement

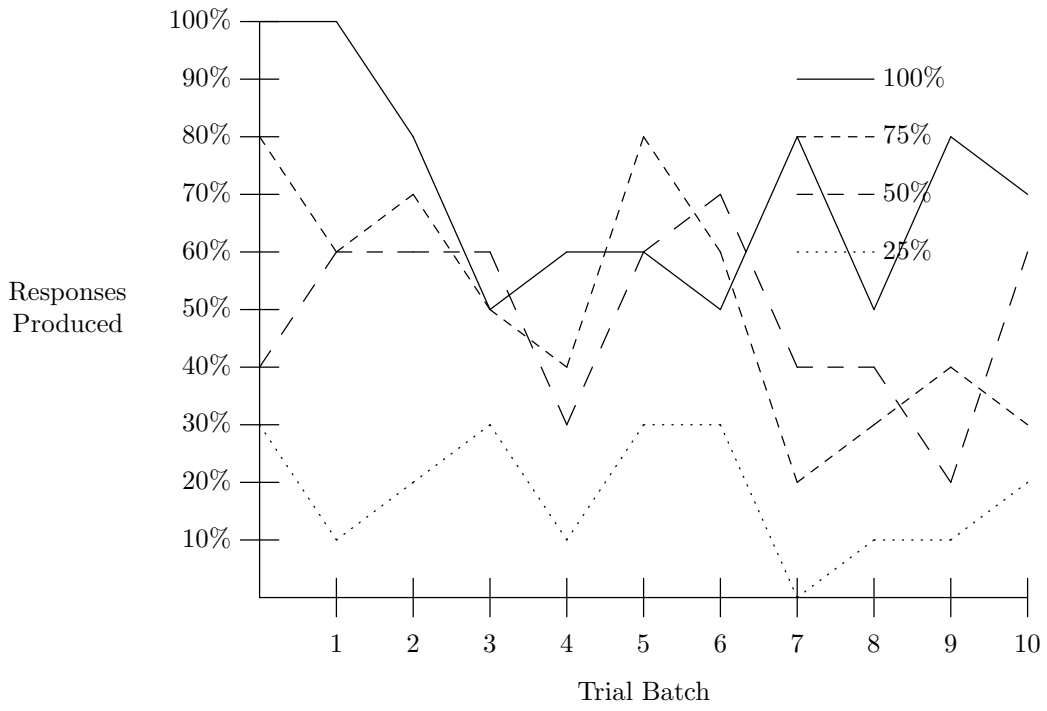

Figure 4.12: Extinction of Partially Reinforced Conditioning

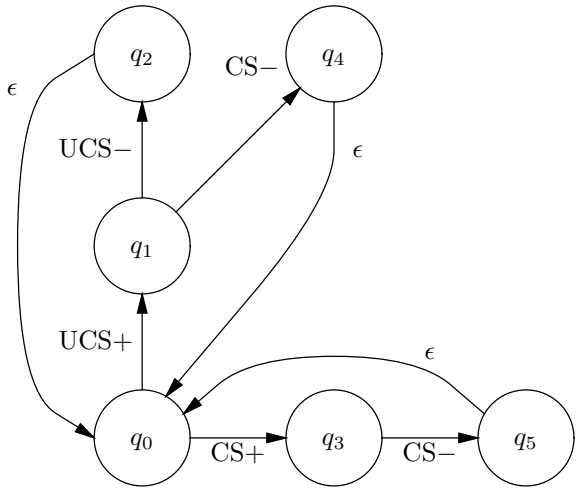

Figure 4.13: Automaton After Simultaneous Conditioning

relationships than the delayed one examined in an earlier section. This section examines simultaneous first-order conditioning in which the onset of the CS and the UCS occur simultaneously.

After presentation of the training sequence, the automaton includes the states q_0–q_4 in Figure 4.13. As the sequence is repeatedly presented, the confidence values for the transitions shown increase and the output probabilities on the transition from q_0 on CS+ move toward those for UCS+. After a testing trial where the CS is presented without the UCS, the automaton is shown in Figure 4.13.

Figure 4.14 shows the results of an experimental test of simultaneous conditioning in a cybernetic automaton. Because the UCS produces the UCR directly, it is uninformative to track the percentage of responses elicited during conditioning as was done for delayed conditioning. Consequently, the experimental results show the course of extinction of the conditioned response after 40 conditioning trials. Each batch of trials consists of ten trials. The point for each batch indicates the percentage of its ten trials in which the automaton produced the UCR upon receipt of the CS. This graph indicates that the automaton did learn to produce the UCR upon receipt of the CS.

4.10 Compound Conditioning

In compound conditioning, the system is conditioned on two conditioned stimuli simultaneously. After the conditioning, each of the conditioned stimuli produce the conditioned response alone.

Figure 4.15 shows a cybernetic automaton which has been compound conditioned. Similar to simultaneous conditioning, either CS1+ or CS2+ produces the conditioned response on the transition from q_0 to q_3.

Figure 4.16 shows the results of experimental verification of compound conditioning in cybernetic automata. Because it is not possible to isolate the responses due to CS2 vs. CS1 during conditioning,

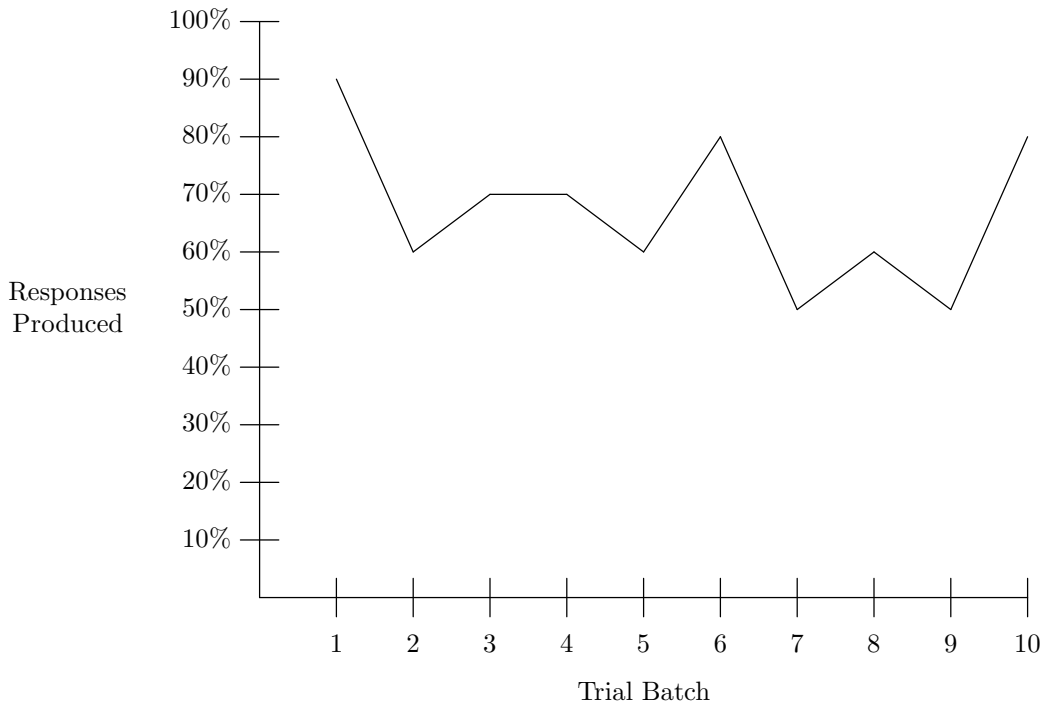

Figure 4.14: Results of Simultaneous Conditioning

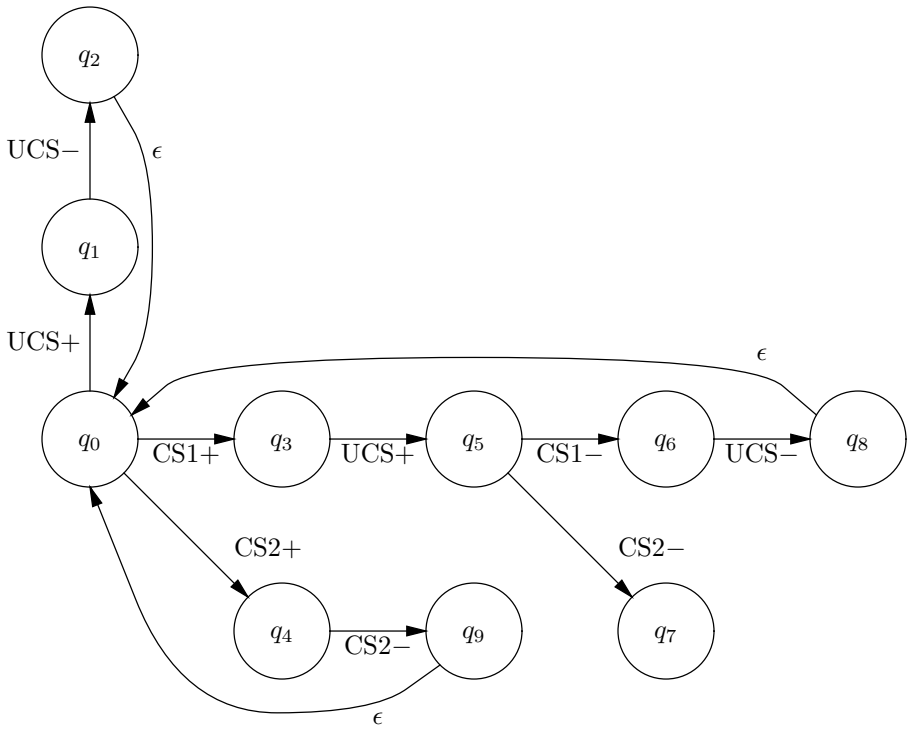

Figure 4.15: Automaton After Compound Conditioning

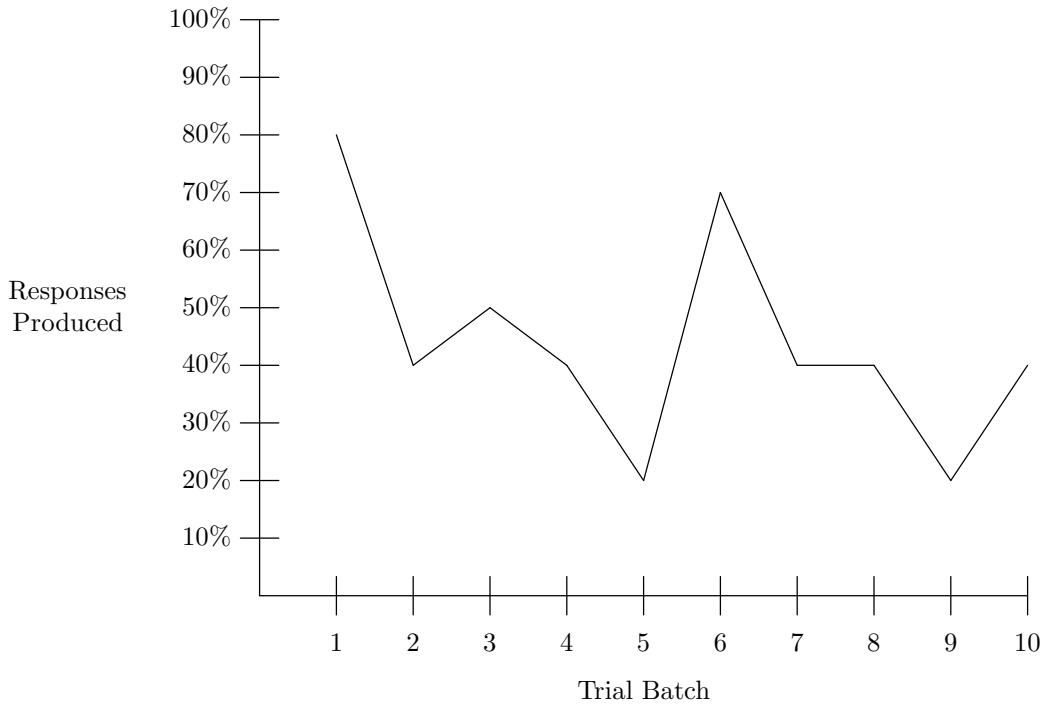

Figure 4.16: Results of Compound Conditioning

the graph shows the results of extinction of the CS2 response. This extinction was performed after compound conditioning of CS1 and CS2 on UCS for 80 delayed conditioning trials.

4.11 Sensory Preconditioning

In sensory preconditioning, the conditioning of one stimulus upon another affects the future learning of the two stimuli. More specifically, if CS1 and CS2 are simultaneously conditioned and CS1 is later delay conditioned on UCS to produce CR, then CS2 will also produce the CR response. In other words, the conditioning of CS1 and CS2 creates some type of link between them such that when one of them is later conditioned, the other follows along.

The subset of Figure 4.17 consisting of states q_0–q_6 represents an automaton after simultaneous preconditioning of CS1 and CS2. Upon conditioning CS1 on UCS, the automaton is configured as in Figure 4.17. Since preconditioning builds an expectation between transitions leaving q_0 on CS1+ and CS2+, changes in probability on the CS1+ transition get propagated to the CS2+ transition. Consequently, CS2 has been preconditioned on CS1 and then follows it as it is conditioned on UCS.

As in compound conditioning, sensory preconditioning is best illustrated experimentally by ex-

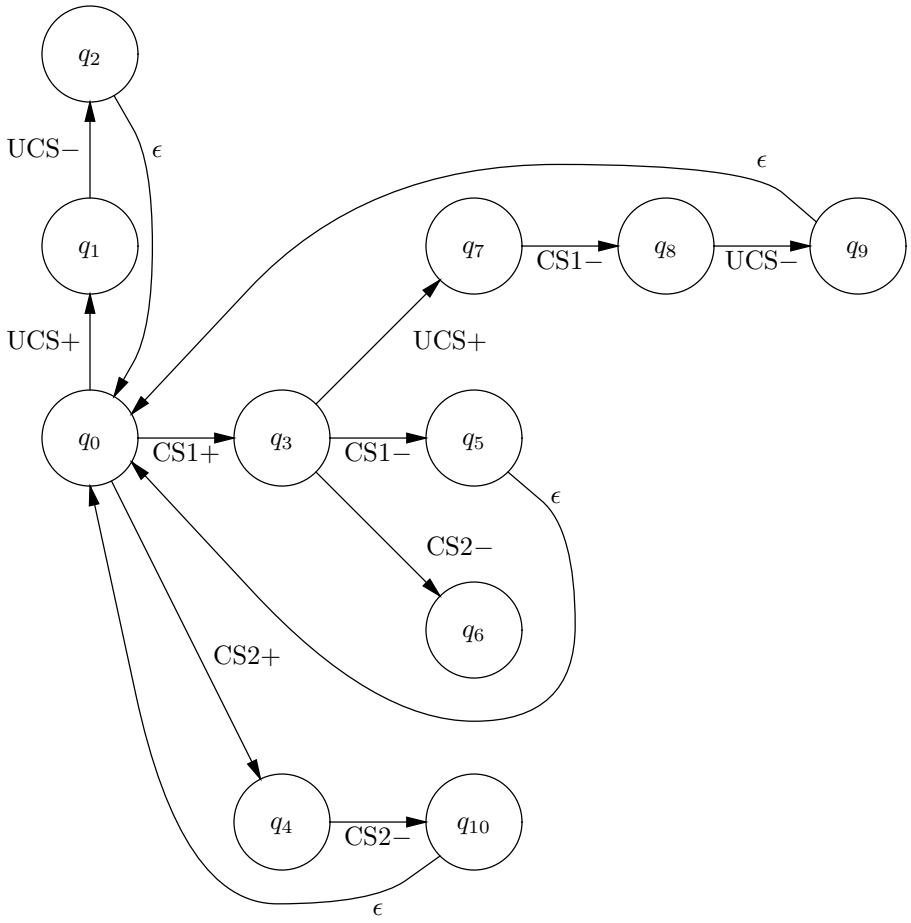

Figure 4.17: Automaton After Conditioning CS1 on UCS

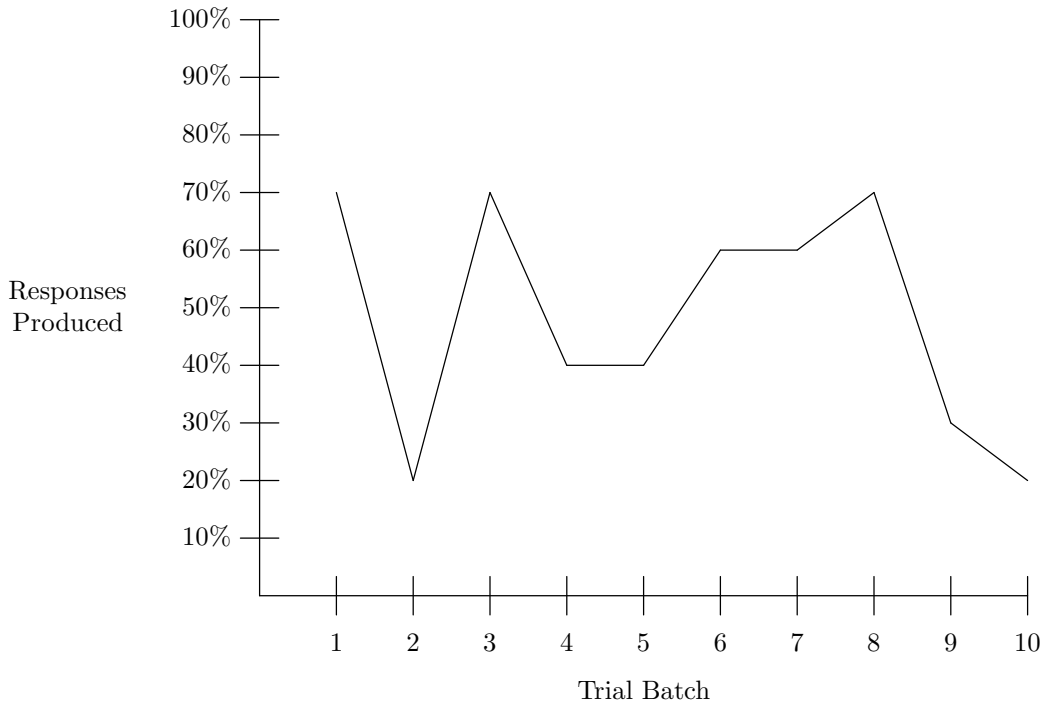

Figure 4.18: Sensory Preconditioning Results

hibiting the extinction of CS2. Figure 4.18 shows this extinction. The extinction phase followed 20 presentations of CS1 and CS2 simultaneously and 100 trials delay conditioning CS1 on UCS. The extinction curve in Figure 4.18 clearly shows that CS2 had indeed been conditioned indirectly through CS1 as a result of the preconditioning.

Thompson (1972) has summarized much work done on sensory preconditioning and has reported that it is quite sensitive to a number of parameters. Of particular interest is that preconditioning is most effective with a relatively few number of preconditioning trials. The experiments he describes indicate that the best results are usually obtained with fewer than 20 preconditioning trials. Here, too, cybernetic automata theory gives some potential insight into this phenomenon. If there is a relatively large number of preconditioning trials, then CS1 and CS2 effectively become simultaneously conditioned, and the conditioning of CS1 on UCS becomes like extinction and re-conditioning. After the preconditioning, the transitions on both CS1+ and CS2+ have high degrees of confidence. During the conditioning of CS1 on UCS, however, they are subject to different expectation changes. The difference in these changes causes the decreases in confidence to be unsynchronized which, in turn, severely reduces the change in probability in the CS2+ transition. Consequently, a large number of preconditioning trials reduces the effectiveness of the preconditioning.

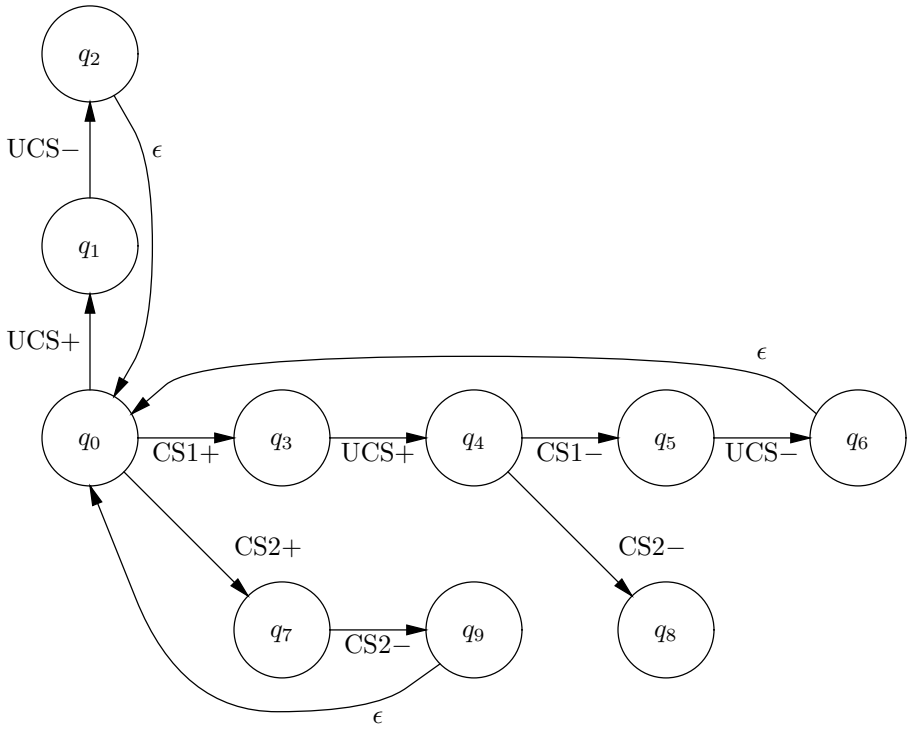

Figure 4.19: Automaton After Blocking Experiment

4.12 Blocking

If CS1 is conditioned on UCS and then an attempt to compound CS1 and CS2 on UCS is made, a phenomenon called blocking emerges. In such an experiment, CS2 is not significantly conditioned. Figure 4.19 shows a cybernetic automaton after a blocking experiment.

It is important to notice the similarity between this scenario and that of simultaneously conditioning CS2 on CS1 after CS1 was conditioned on UCS. (Such a case of simultaneous second-order conditioning is examined in the next section.) Indeed, until the arrival of the UCS during the compound phase of a blocking experiment, there is no difference between the two. Yet in one case, CS2 is conditioned and in the other it is not. Clearly this indicates that either the "decision" regarding how learning takes place is made some time after the arrival of CS1 and CS2 or learning takes place either way and is somehow undone if blocking takes place. The former more accurately describes what takes place in cybernetic automata. Here no learning takes place until the CS1− input is received, causing a change in output and triggering probability changes to take place. However, the confidence on the CS1+ transition is high due to the prior conditioning and thus little change in probability takes place. Because little change takes place on the CS1+ transition, little is propagated to the CS2+ transition, and little conditioning of CS2 takes place.

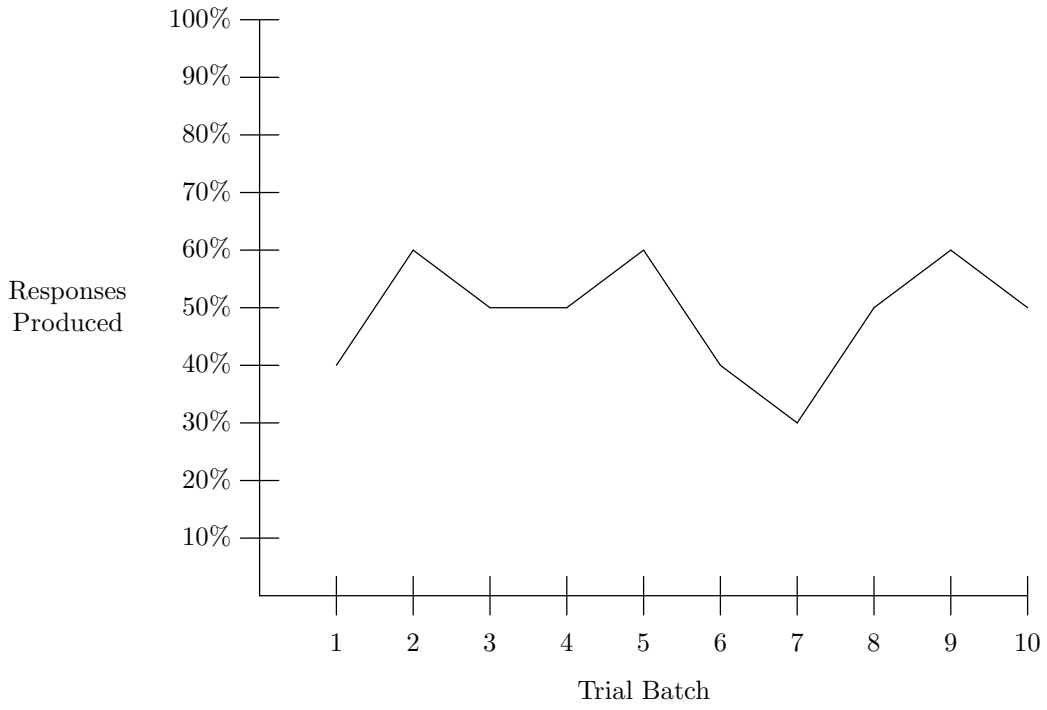

Figure 4.20: Results of a Blocking Experiment

Figure 4.20 shows the results of blocking experiment on a cybernetic automaton. In this experiment, CS1 was conditioned on UCS until 60 consecutive conditioned responses were produced, then an attempt was made to compound condition CS1 and CS2 on UCS with 40 trials. The graph shows the subsequent 100 extinction trials of CS2.

4.13 Extinction in Second-Order Conditioning

One of the most interesting phenomena of conditioning arises when the response of CS1 is extinguished after CS2 is second-order conditioned on CS1. If the second-order conditioning was done with delayed timing, then CS2 will retain its conditioned response. However, if the second-order conditioning was done with simultaneous timing, then the conditioned response of CS2 will be extinguished also. This section examines three experiments which establish that cybernetic automata exhibit this phenomenon.

Before this property can be interesting, it is important to verify that second-order conditioning does indeed work when CS2 is simultaneously conditioned on CS1. Figure 4.21 shows an automaton on which simultaneous second-order conditioning has taken place. Figure 4.22 shows a graph of the extinction of CS2 after simultaneous second-order conditioning. Experimentally, CS1 was first

delay conditioned on UCS until 80 consecutive positive responses were received. Then CS2 was simultaneously conditioned on CS1 for 20 trials, and finally 100 extinction trials of CS2 were run and plotted.

The first part of this property says that if CS1 is conditioned on UCS, then CS2 is delay conditioned on CS1 and CS1's response is then extinguished, CS2 will retain its conditioning. Figure 4.23 shows a cybernetic automaton which has undergone such an experiment, and Figure 4.24 shows the results of extinction on CS2 after the experiment. Here CS1 was conditioned on UCS until 80 consecutive conditioned responses were achieved. Then CS2 was simultaneously conditioned on CS1 for 20 trials, and CS1 was presented alone for 100 extinction trials. Figure 4.24 clearly shows that CS2 retained a substantial degree of conditioning even though CS1's response was extinguished.

Figure 4.25 shows the extinction of CS2 in the automaton in Figure 4.21 after CS1's response has been extinguished. Here CS1 was conditioned on UCS until 80 consecutive responses were achieved, then CS1 was simultaneous conditioned on CS1 for 20 trials and CS1 was then extinguished for 100 trials. Figure 4.25 demonstrates that CS2 has a lower level of conditioning after CS1's extinction if it was conditioned on CS1 simultaneously as opposed to delayed.

The intuition behind this phenomenon in cybernetic automata comes from the manner in which probability changes are propagated. In the case of simultaneous second-order conditioning, the changes to the CS1+ transition are directly propagated to the CS2+ transition. Because the CS2+ transition suffers the same loss of confidence during CS1s early extinction, (following simultaneous conditioning of CS2 on CS1), it is susceptible to those changes and is extinguished. On the other hand, if CS2 is delay conditioned on CS1, then extinction propagation from the CS1+ transition from q_0 are mediated by the CS1+ transition from q_7 while being propagated to the CS2+ transition. During the extinction of CS1, however, the CS1+ transition from q_7 does not receive the same drop in confidence, so it does not pass on a significant change in probability. Thus the decrease in conditioning on the CS2+ transition is far less if CS2 were delay conditioned on CS1.

4.14 Effects of Stimulus Strength and Stimulus Generalization

In regards to the strength of the stimuli, the general trend is that stronger stimuli (both CS and UCS) lead to faster and stronger conditioning. The cybernetic automaton model clearly possesses this property as a result of the form of its conditioning function.

Another consideration regarding the conditioned stimulus is stimulus generalization. Often stimuli similar to those conditioned yield the conditioned response as well. It is reasonable to assume then that an organism being conditioned has had a rich history of experience prior to the conditioning. Furthermore, if it is assumed that similar stimuli have been encountered together frequently, then stimulus generalization becomes a special case of sensory preconditioning. Since the cybernetic automaton model has already been shown to exhibit sensory preconditioning, it will also exhibit stimulus generalization for stimuli which have been encountered together.

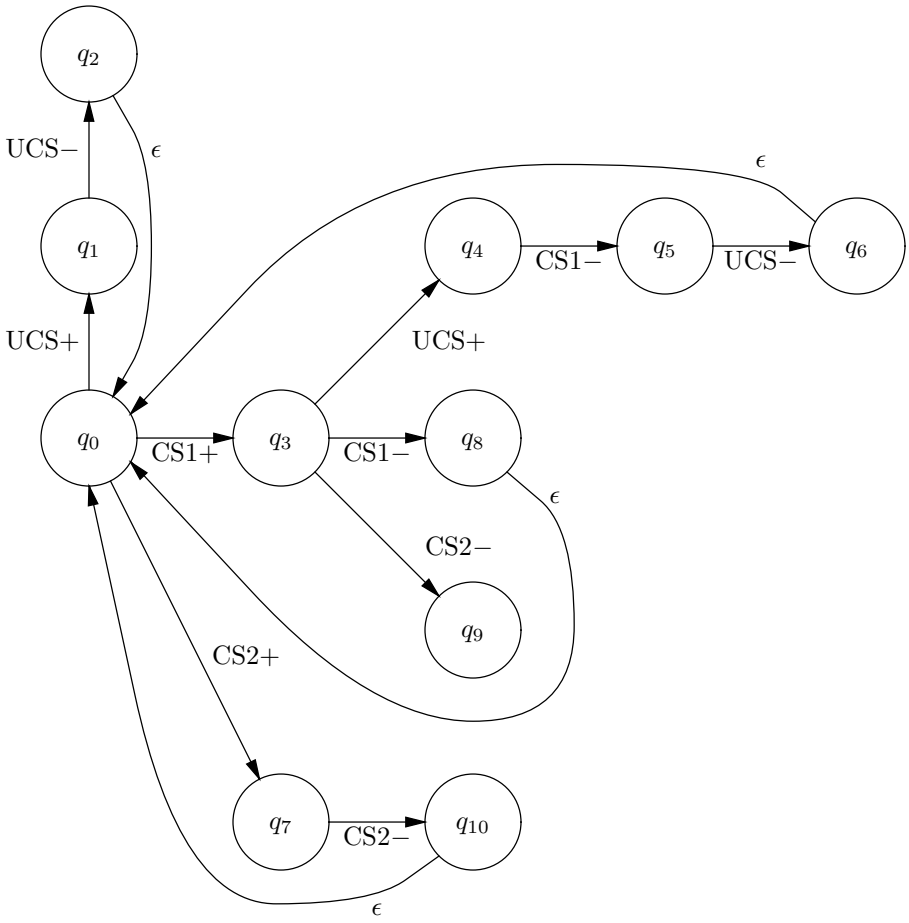

Figure 4.21: Automaton After Simultaneous Second-Order Conditioning

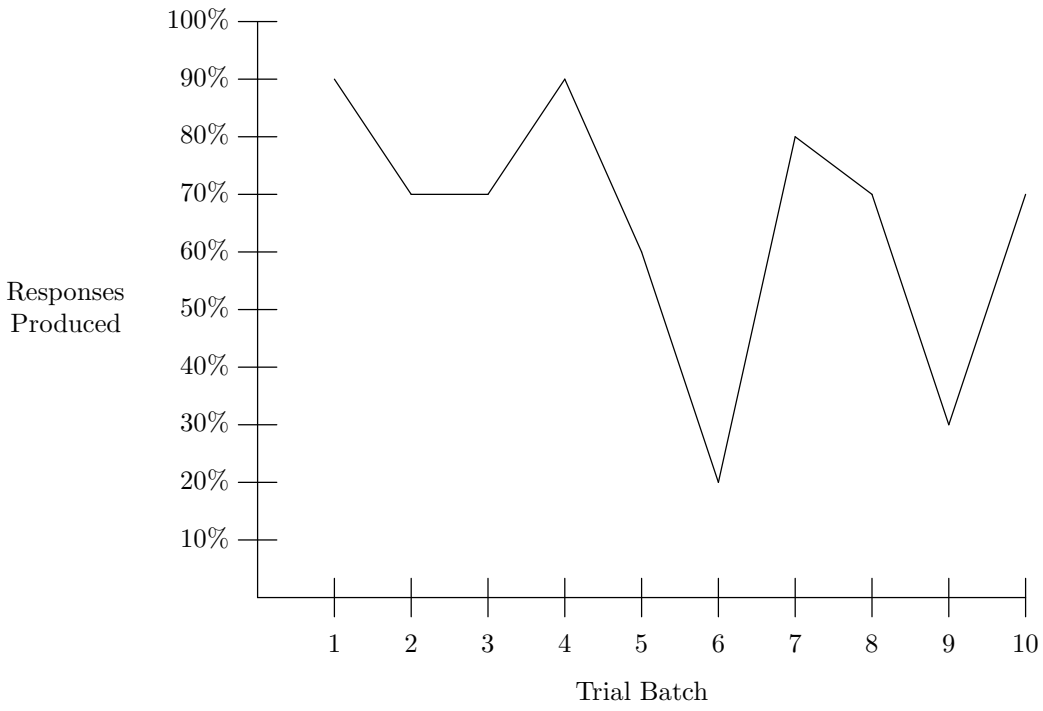

Figure 4.22: Results of Simultaneous Second-Order Conditioning

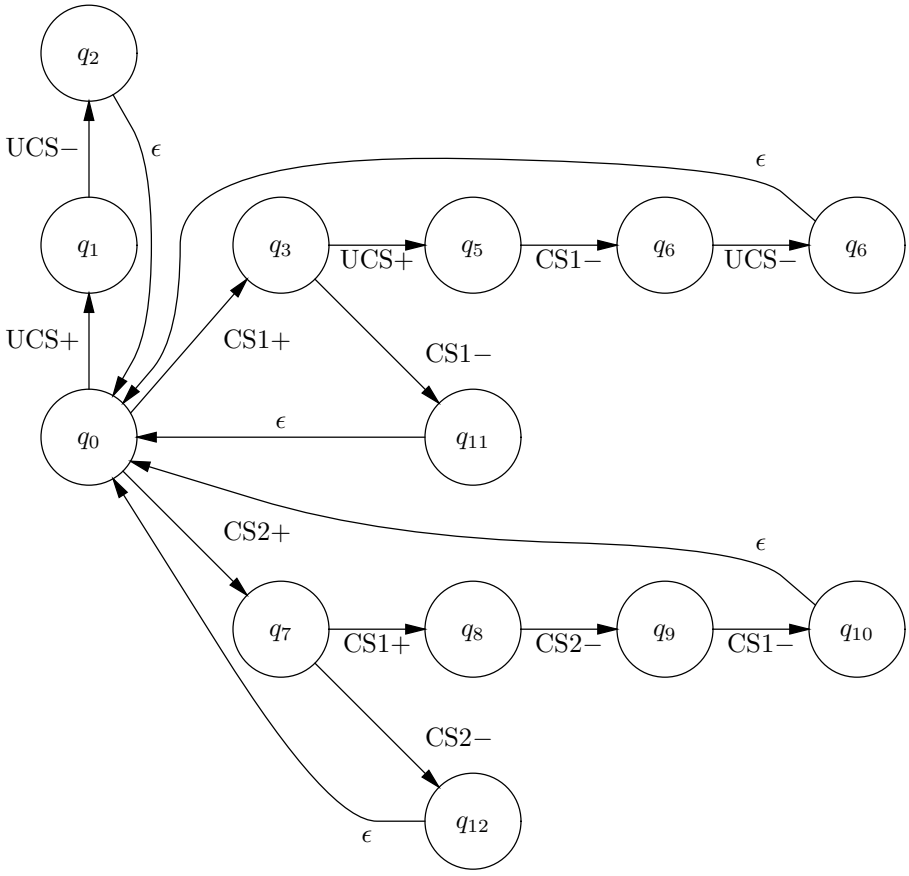

Figure 4.23: Automaton After Delayed Second-Order Conditioning and Extinction

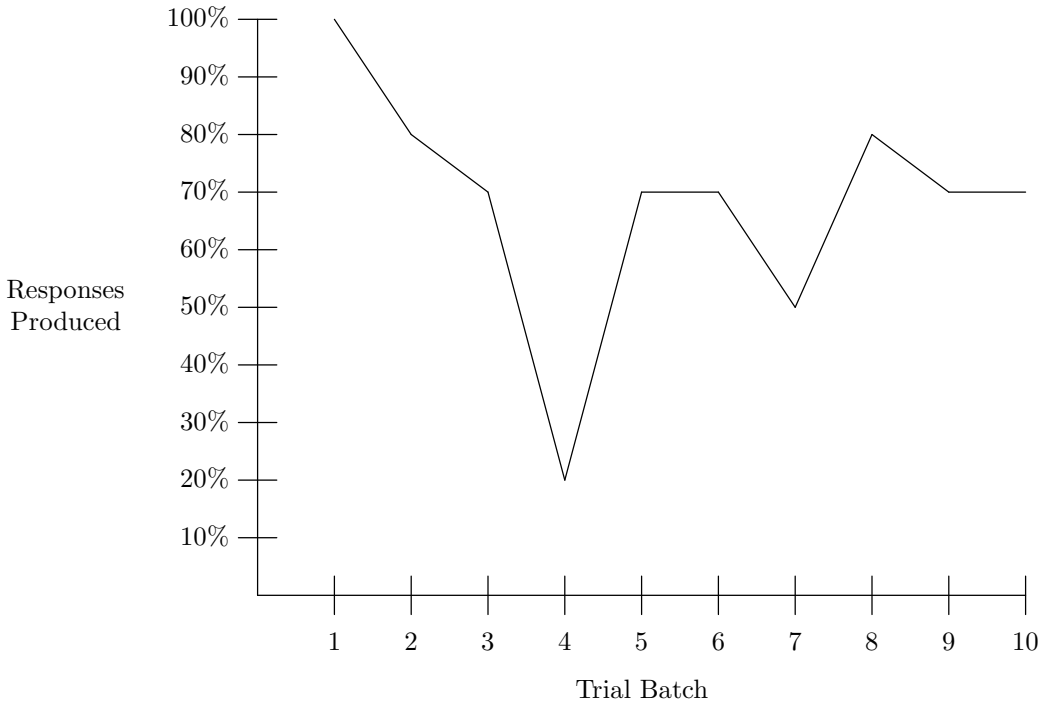

Figure 4.24: Results of CS2 Extinction After Delayed 2nd-Order Conditioning

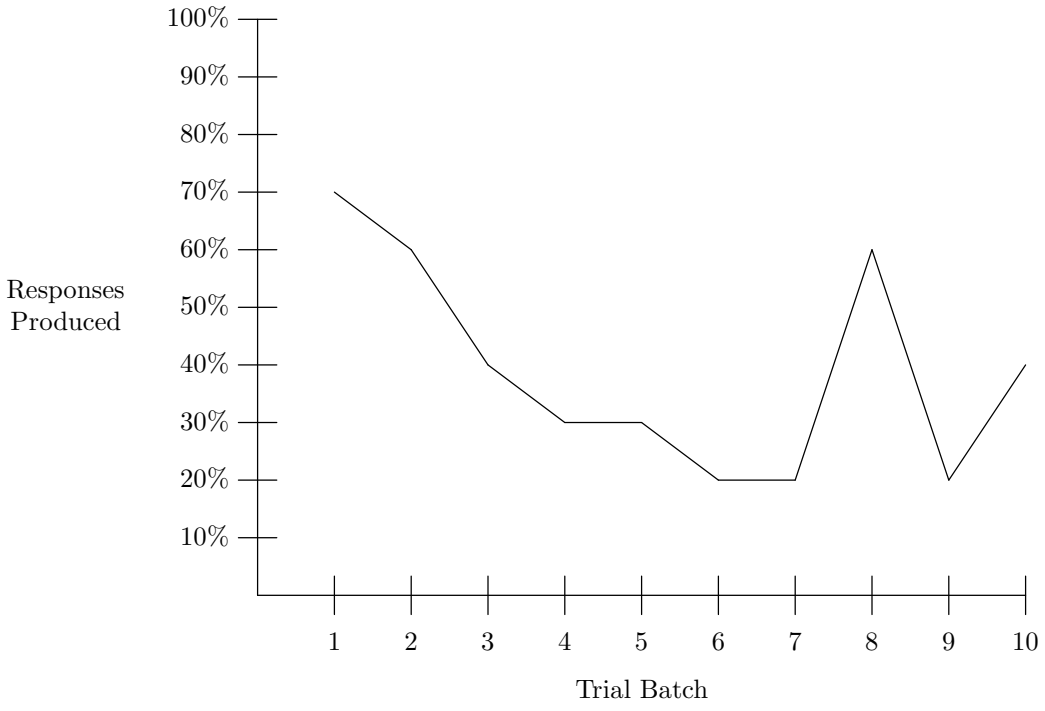

Figure 4.25: Results of CS2 Extinction After Simultaneous 2nd-Order Conditioning

4.15 Other Conditioning Properties

Cybernetic automata theory is not intended to be a psychological theory in the sense that it should explain how natural learning operates. Consequently, it should not be expected to faithfully emulate all aspects of natural learning. Three known properties of natural learning which are not adequately emulated by cybernetic automata are trace conditioning, temporal conditioning and spontaneous recovery.

In trace conditioning, both the onset and offset of the conditioned stimulus occur prior to the onset of the unconditioned stimulus. If a cybernetic automaton were trained according to a trace conditioning paradigm, CS− would be conditioned and by propagation so would CS+. However, at no point would the CS be conditioned to terminate the response. Consequently, such conditioning could not be extinguished. It is highly unlikely that natural systems learn trace conditioning in this way.

While the representation of time in cybernetic automata is sufficient to emulate most properties of classical conditioning, it is not sufficient for the model to exhibit temporal conditioning. In temporal conditioning, the UCS is presented repeatedly at a fixed interval of time. After a number of such trials, the system begins to produce the response spontaneously at intervals approximately equal to the interval at which the UCS was presented. Again, cybernetic automata have no mechanism to emulate temporal conditioning.

Similarly, there is no mechanism to account for spontaneous recovery. If a natural system is allowed an extended period of rest after extinction, the extinguished response returns. This is called spontaneous recovery. In cybernetic automata, no changes in probabilities take place without some input sequences to trigger them.

4.16 Summary

Cybernetic Automata successfully emulate a large variety of psychological learning phenomena. This provides strong evidence that the cybernetic automata model is a good one for artificial intelligence. The extent of accuracy is provided by a relatively small number of learning mechanisms which cooperate to exhibit these properties. The model is not, however, perfect. There do exist some properties of natural learning which are not accounted for by cybernetic automata. Nevertheless, the model does provide more naturally accurate learning than do other models.

Chapter 5

Reinforced Learning in Cybernetic Automata

In Chapter 2, the cybernetic automaton model is defined as having both reinforced and unreinforced learning components. All of the psychological properties discussed in the previous chapter utilize only the unsupervised learning mechanisms. Even though the supervised mechanisms are used in Chapter 3 for proving the completeness of cybernetic automaton learning, no experimental use has been made of these mechanisms. This chapter presents experiments which use the supervised learning in cybernetic automata. First, the classic Skinner box experiment is performed on a cybernetic automaton. Next, a cybernetic automaton learns to maintain balance in a two-dimensional world. These experiments illustrate that in addition to classical conditioning cybernetic automata also mimic supervised psychological learning and are capable of solving useful problems.

5.1 Instrumental Learning

Instrumental learning (also known as operant conditioning) differs from classical conditioning in that the stimuli which are presented to the subject are affected by the responses which the subject gives. In other words, the subject's responses are instrumental in the learning process. Because the instrumental learning process involves stimuli which depend on subject responses, it is inherently reinforced. The stimuli, which serve as reinforcements in instrumental learning, must be either rewarding or punishing in nature for the cybernetic automaton to treat them as reinforcing. This characteristic is generally true of reinforcing stimuli in instrumental learning.

The remainder of this section presents the Skinner box experiment and a simulation of it by a cybernetic automaton. The experimental simulation demonstrates that cybernetic automata exhibit instrumental learning as well as classical conditioning.

5.1.1 The Skinner Box

One of the most common experimental set-ups for instrumental learning is the Skinner box (Misiak & Sexton, 1966). A Skinner box is generally a cage with a bar at one end. The bar may be pressed by the experimental subject to receive some type of reward. There is a large variety of experiments performed in Skinner boxes to demonstrate instrumental learning. One of the more basic is discussed here.

In the Skinner box experiment simulated here, pressing the bar delivers a portion of food to the subject. The only stimuli presented to the subject are the visual patterns seen from various parts of the cage and the taste of food. If the subject is hungry, then the food presented will be a direct reward.

During the initial phase of the experiment, the subject behaves with random, spontaneous actions. Once the random actions lead the subject to the bar and cause it to be pressed, then the reward strengthens the responses which led it to that state. Assuming that the subject remains hungry as the experiment continues, the time which is taken in random actions leading to bar pressing decreases dramatically as more rewards are delivered. Soon the subject directly moves to the bar and repeatedly presses the bar until its hunger is satiated.

Some common variations on this basic Skinner box experiment involve cue stimuli such as sounds which might, for instance, indicate immediate and temporary food availability. In this case, the subject learns to quickly respond to the cue stimulus by moving directly to the bar and pressing it. Another common variation involves a punishment stimulus such as electric shock applied to the bottom of the cage (either with or without a cue stimulus). In these cases, the bar has the effect of terminating or preventing the punishment from occurring. The subject, in such experiments, again learns to press the bar either upon the cue stimulus or upon onset of the punishment stimulus.

5.1.2 Simulation of the Skinner Box

In simulating the basic Skinner box experiment, the automaton shown in Figure 5.1 is used as an initial automaton. For it, the state q_R is in the set R, and $P = \emptyset$. Furthermore, the initial probability distribution of output symbols on all transitions from q_0 has equal probability for all symbols. In addition to the input symbol for food consumption (F), there are input symbols which represent the view of the subject from several points within the cage. These are four symbols, one when facing each direction, for each of the points shown in Figure 5.2. Finally, the output symbols include two symbols, one for turning left and one for right, a symbol to move forward, a symbol to press the bar and the epsilon symbol. The action of eating food when it is presented is assumed to be instinctive and independent of this experiment. Consequently, it will be taken to be automatic and supplied by the physical system in which the automaton operates. The model parameters used for this experiment are given in Table 5.1.

Because the sequence of input symbols that a subject receives during such experiments depends on its sequence of randomly selected output symbols, it is not possible to analyze the subject's behavior as precisely as for classical conditioning experiments. Furthermore, the fact that the experimental trials last for potentially many stimulus presentations can cause the automaton to grow quite large. For purposes of illustration, consider the automaton shown in Figure 5.3. Here, the automaton started in position 1 facing north and proceeded to turn right, to go forward and to depress the bar.

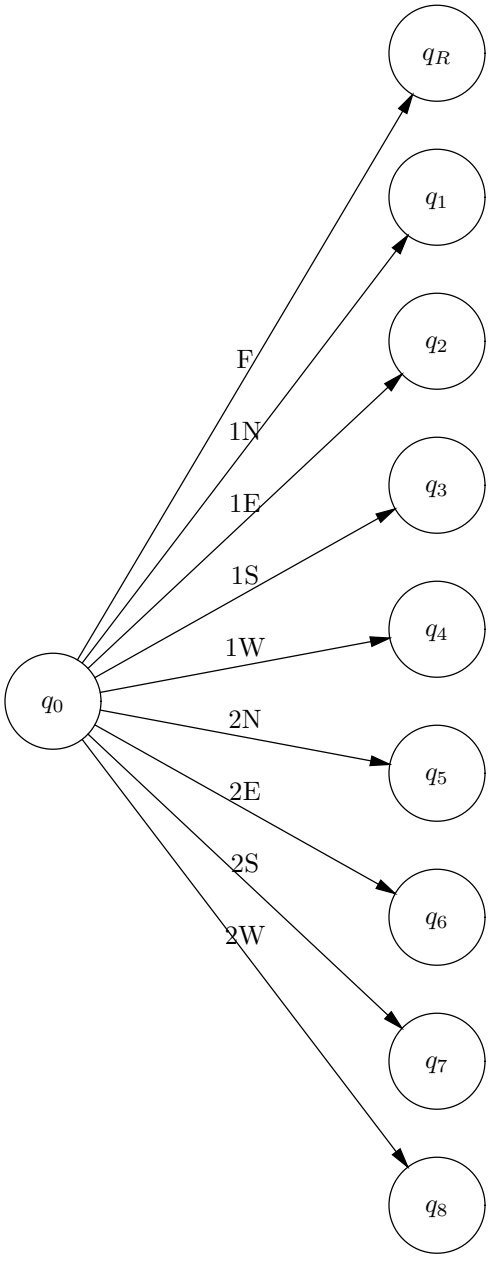

Figure 5.1: Initial Automaton for Skinner Box Experiment

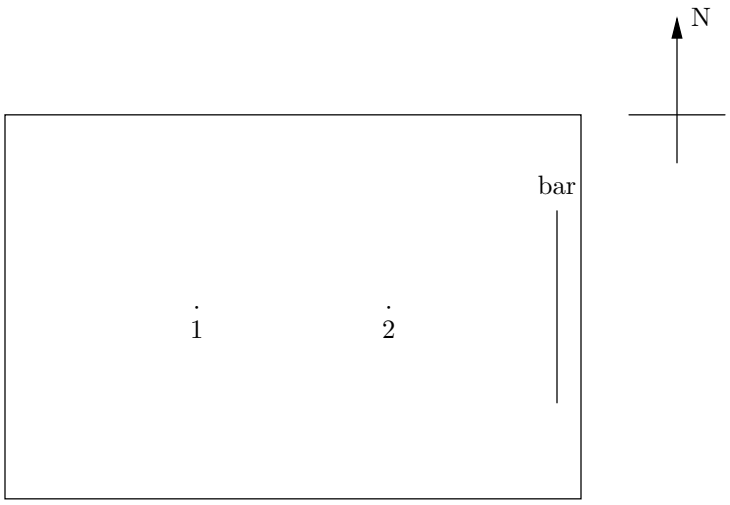

Figure 5.2: Subject Positions Within the Skinner Box

Table 5.1: Model Parameters for Skinner Box Learning

Parameter	α	β	γ	η	ζ	ν	κ
Value	0.05	0.5	0.001	1.0	0.1	0.5	0.9

At the completion of this sequence of events, the automaton received the food consumption input symbol which takes it to a reward state. This reward then, in turn, increases the probability that those output symbols are selected on their corresponding transitions.

In practice, it is very unlikely that an automaton would initially take such a direct route to the bar. Instead, experimental subjects generally exhibit a certain amount of random "trial and error" behavior. A cybernetic automaton exhibiting such behavior clearly grows a much larger tree than the one shown here. A large number of responses prior to reward tends to deepen the tree and diverse response sequences among trials tends to broaden the tree.

Learning in a more realistic experimental subject is still based on rewarding the actions which lead up to pressing the bar. However, there are two elements of cybernetic automaton learning which apply directly to such situations. First, the effect of supervised learning is greater for recent output selections and less for older selections. Second is the propagation of probability changes through the conditioning mechanisms. Together, these mechanisms tend to shorten the time taken to reach the reward by decreasing the probability that a position and orientation is repeated while seeking the reward.

Suppose input symbol A is encountered twice before reward during some trial, and that the first time A is encountered, output symbol X was selected but that symbol Y was selected the second time. Clearly, if all other selections are the same then, the time to reach the reward would be shortened if Y were selected upon the first receipt of A. Cybernetic automaton learning will encourage this selection as follows:

1. Because of the temporal effects of supervised learning, the selection of X the first time A is encountered will be rewarded less than the selection of Y the second time A is encountered.

2. Because of the temporal effects and the propagation of probability changes, the probability of selecting Y the first time A is encountered will be rewarded more than the probability of selecting X the second time A is encountered.

3. During other trials where A is encountered, only one selection of Y will lead to faster reward and hence stronger reward.

The result of these items is that the trend will be toward higher probability for the selection of Y when A is encountered.

Figure 5.4 shows the results of a series of experimental trials of a cybernetic automaton in a Skinner box experiment. For each trial, the number of responses given prior to reward is plotted. This graph clearly shows that the performance of the automaton improves with experience and, hence, is learning.

5.2 Balance

The task of learning to balance an object has been extensively studied in machine learning and control systems research. Barto, Sutton and Anderson (1983) examined pole balancing in relation to their neural computational model. Quin, et al (1990) examined it as an application of stochastic cellular automata. In both of these cases, the problem is stated as that of balancing a stick atop a mobile cart. The cart moves in order to maintain the stick's balance.

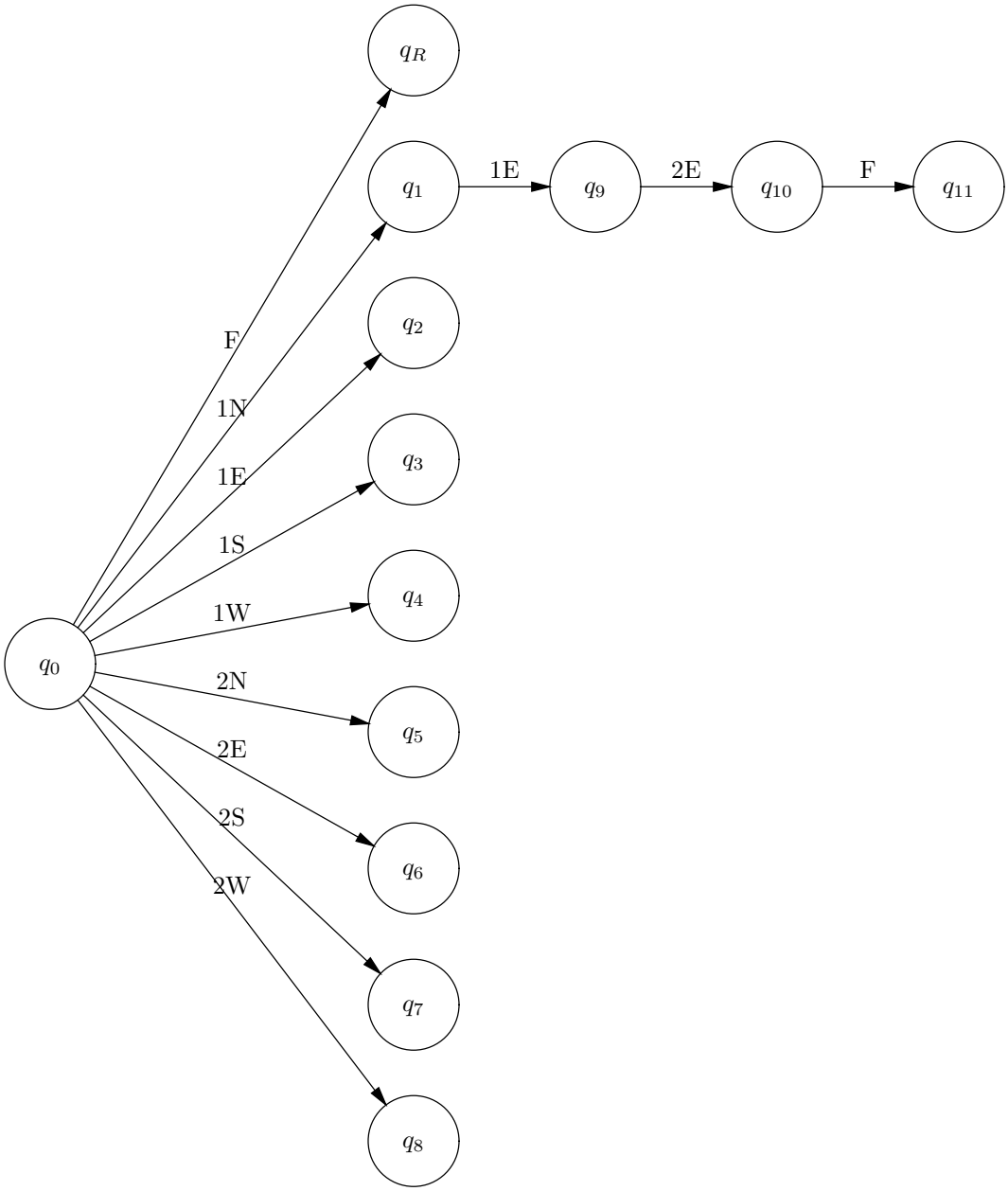

Figure 5.3: Example Automaton After Skinner Box Experiment

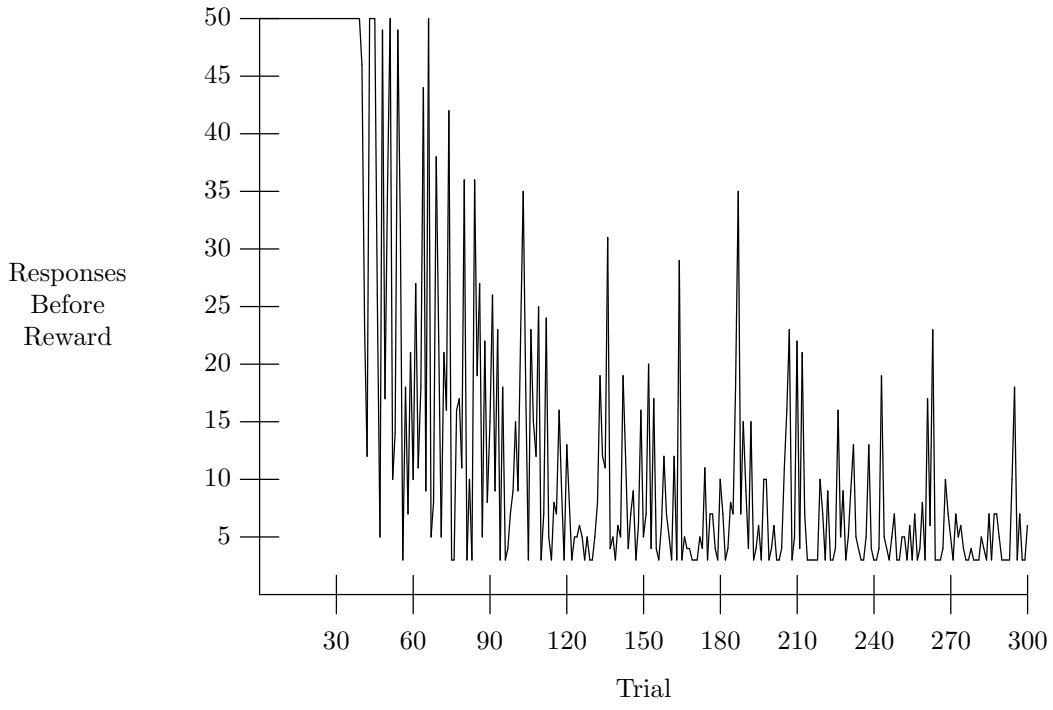

Figure 5.4: Results of a Skinner Box Experiment

Table 5.2: Model Parameters for Balance Learning

Parameter	α	β	γ	η	ζ	ν	κ
Value	0.05	0.1	0.0001	1.0	0.3	0.5	0.8

This section presents the balance problem as one which is solvable by cybernetic automata. Here the problem is stated in a slightly more biologically realistic fashion. A stick organism is anchored to the ground. It has two muscles which may pull it to rotate about its anchor point; one muscle pulls to the left, and one to the right. Achieving an angle from vertical less than 0.01 radians and an absolute angular velocity less than 0.01 rad/sec rewards the system, causing it to learn how to stay balanced. Falling to the ground yields a punishment proportional to the magnitude of the angular velocity at the time of impact.

Cybernetic automaton learning in the balance problem is, in many ways, very similar learning in the Skinner box. In the balance problem, there is an additional consideration of the punishment which decreases the probability of unfavorable behavior. In both cases the application of reward upon achieving the goal reinforces and increases the probability of the behavior that led to that goal.

5.2.1 Simulation of Balance Learning

The subject has five input stimuli representing the following:

1. angle not less than -0.01 rad, positive angular velocity (PP)
2. angle not less than -0.01 rad, negative angular velocity (PN)
3. angle not greater than 0.01 rad, positive angular velocity (NP)
4. angle not greater than 0.01 rad, negative angular velocity (NN)
5. angular magnitude less than 0.01 rad, angular velocity less than 0.01 rad/sec. (B)
6. angular magnitude greater than or equal to $\frac{\pi}{2}$ rad (F)

The system has three responses available to it. There is the null response where the system does nothing. The other two responses are the activation of the left muscle and the right muscle. Figure 5.5 shows the initial cybernetic automaton used for simulation of balance learning. The model parameters used for this experiment are given in Table 5.2.

Simulating the balancing of an object involves simulating the physics of the object's motion and of the effect of gravity on the object. Figure 5.6 shows the two-dimensional system anchored to the ground. The gravitational force between the subject and the earth tends to add to the angular velocity an increment which is of the same sign as the angular position. Activation of the muscles by suitable responses also adds to the angular velocity. The angular position and velocity are governed by the differential equation:

$$\frac{d^2\theta}{dt^2} = \frac{3(\tau_g + \tau_m)}{ml^2}$$

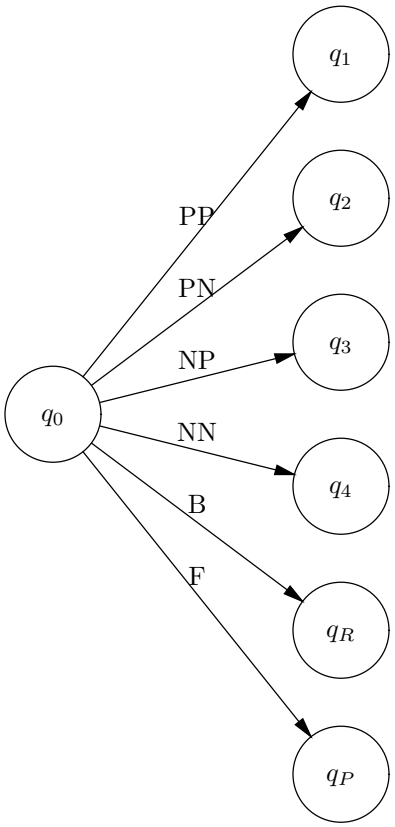

Figure 5.5: Initial Automaton for Balance Simulations

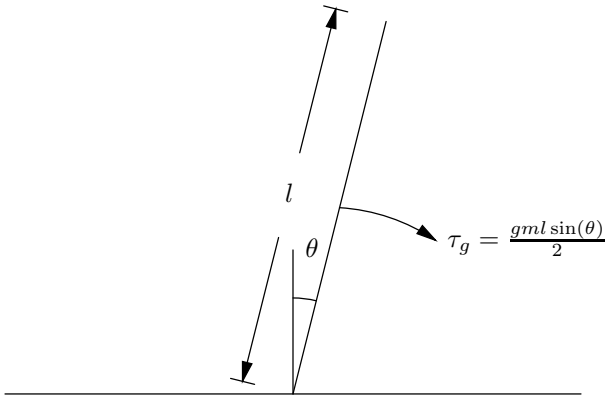

Figure 5.6: Stick Balancing in a Two-Dimensional World

where τ_m is the torque due to muscle action. This is an idealized environment without any air resistance or damping at the anchor point.

The learning results shown in Figures 5.7–5.9 were obtained with a discrete-time simulation of the initial cybernetic automaton in the physical "world" defined above. Figure 5.7 shows the learning that takes place with only the reward reinforcement for achieving balance. In Figure 5.8, the learning shown is for an experiment with only the punishment reinforcement on falling over. The results for both forms of reinforcement are shown in Figure 5.9. For each trial the amount of time before falling is plotted. Trials longer than 10 seconds were terminated.

From these three figures it is clear that neither punishment nor reward alone is sufficient to achieve satisfactory learning. When only punishment is presented, the performance does improve, but without the guidance of a reward for success, there is no direction to the search for correct probabilities. Conversely, with only reward reinforcement, performance stays poor until random chance yields balance. After that learning is swift. The combined effects of reward and punishment are better than either alone. Progress is steadier and goes farther than punishment alone and it comes sooner than reward alone.

5.3 Conclusions

All of the results shown in this chapter share some common traits. One of the most noticeable is that the learning is not perfect. Namely, the subject in the Skinner box does not learn to go to the bar directly every time and the balance subject does not stay up for ever. These imperfections illustrate the difficulty in the credit-assignment problem when applied to learning systems where reinforcement is not presented upon every subject output. If a teacher knows exactly what type of reinforcement needs to be applied at every step as in the Cybernetic Automaton Learning Theorem, then it is not difficult to achieve perfect learning. However, such is not the case in the learning facing most intelligent systems.

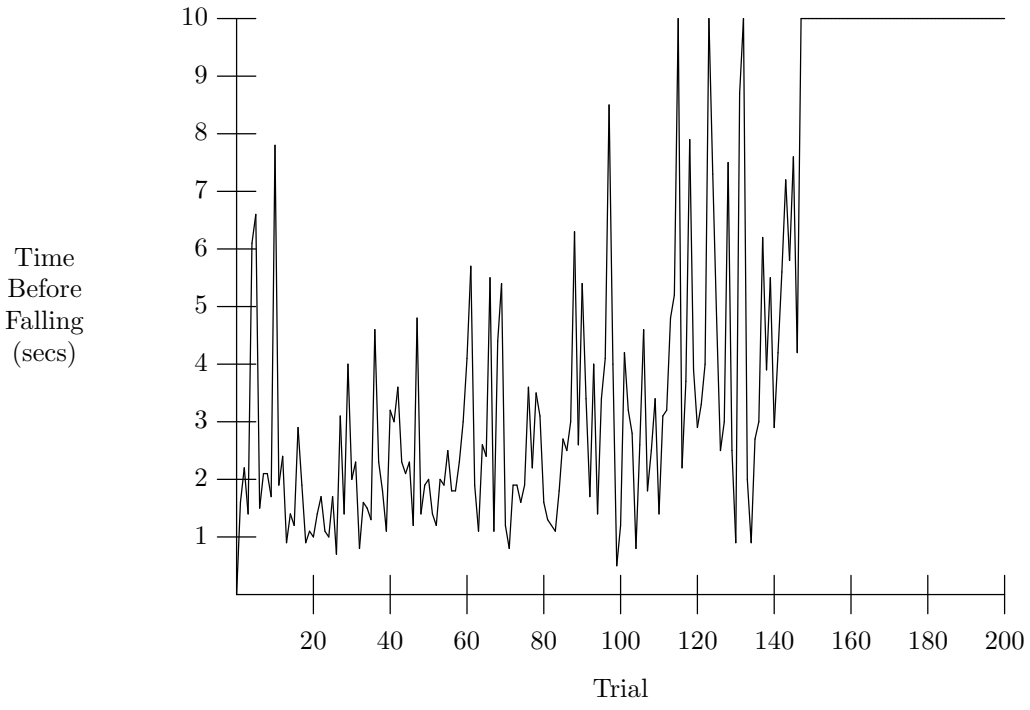

Figure 5.7: Results of Reward-only Balance Leaning

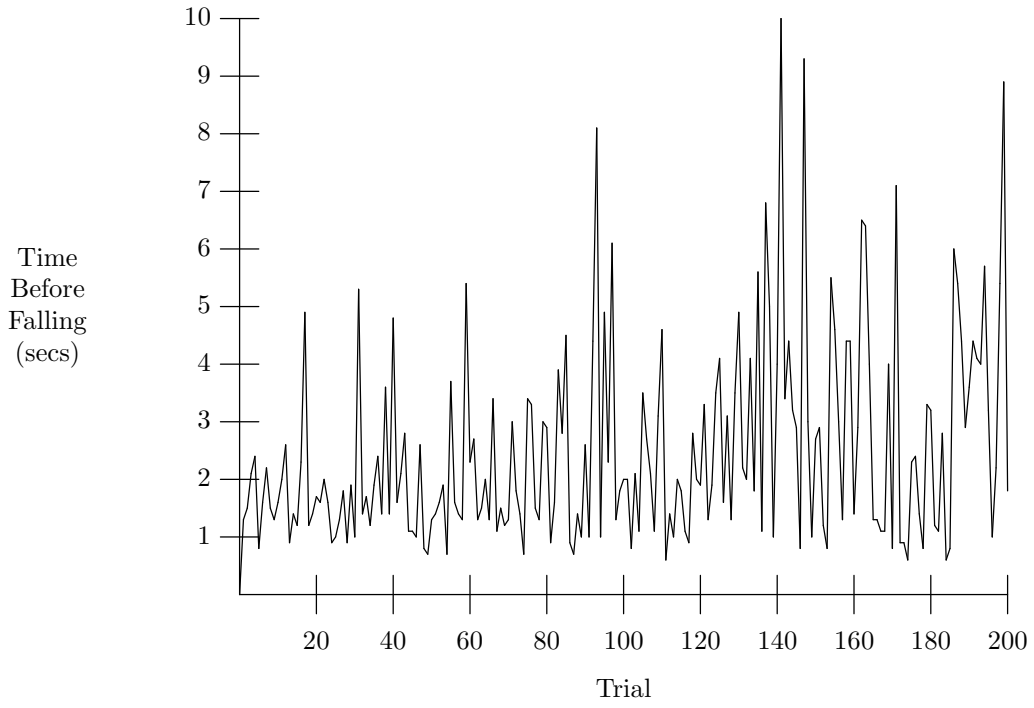

Figure 5.8: Results of Punishment-only Balance Leaning

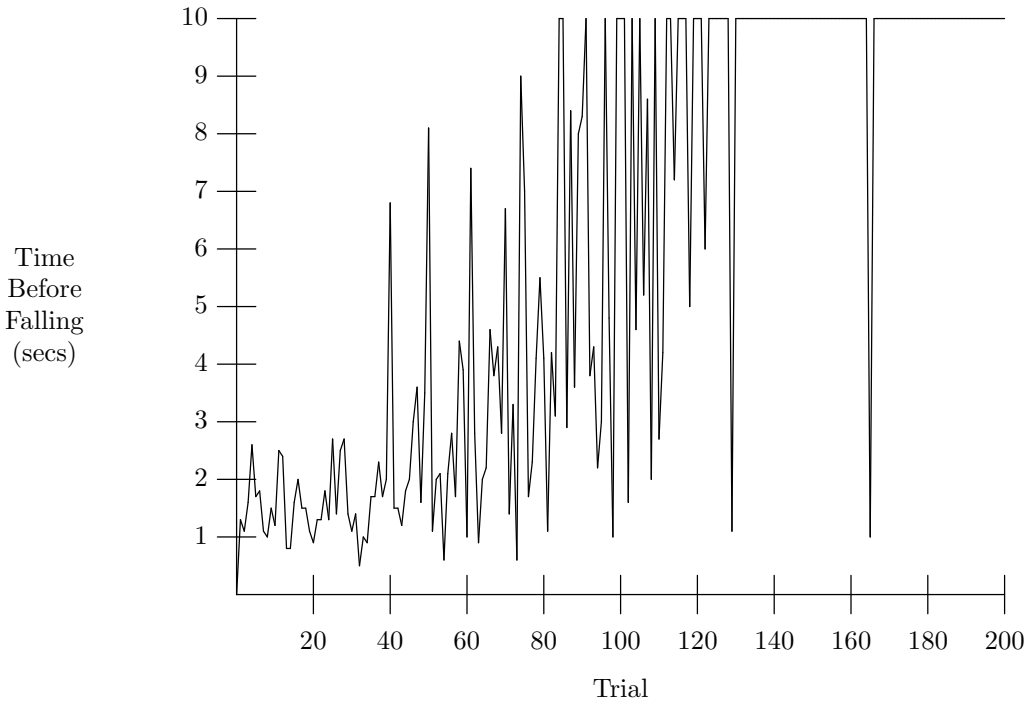

Figure 5.9: Results of Reward and Punishment in Balance Leaning

Chapter 6

Neural Cybernetic Automata

The cybernetic automaton computational model is clearly realizable as a program implemented on a conventional stored program computer. Indeed, such a program exists and was used for the simulations described in Chapters 4 and 5. It is, however, also interesting to examine a more direct realization of the model.

The stored program computer is often considered an implementation of a universal Turing machine. Together, the main memory and I/O devices may be taken as a tape. For no real machine, though, can the this tape be unbounded. Consequently, the modern computer is a finite approximation to a universal Turing machine. Likewise, the implementation of cybernetic automata developed here is a finite approximation to the model. The bound in this case is placed on the depth to which tree may grow during learning.

This section presents one possible implementation of cybernetic automata. The technology used for this implementation is biologically plausible neurons. Two caveats apply here, though. First, no claim is made that the designs presented here are the most efficient possible. They have been selected for clarity of implementation rather than efficiency. Second, even though model neurons are used, the designs presented here are not meant to represent what actually exists in any natural brains. Any similarity to real brains is a fortunate coincidence.

6.1 High-Level Description

The basic functions performed in the cybernetic automaton algorithm can be easily broken down into a set of sub-systems. These include:

1. Input and state transition (Short-Term Memory)
2. Maintenance of confidence (C) and expectation (E) functions (Long-Term Memory)
3. Probabilistic output selection (Output Selection)
4. Control over probability modifications (Learning Control)

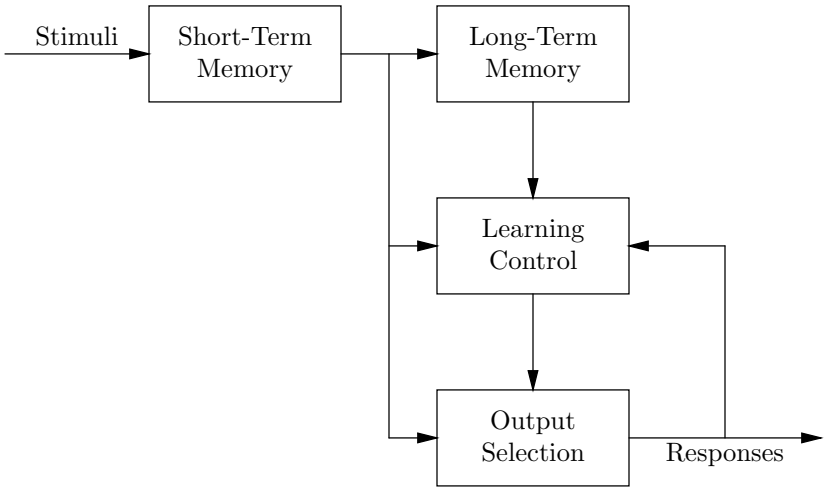

Figure 6.1: Organization of Sub-Systems

These sub-systems fit together as shown in Figure 6.1.

The next four subsections discuss the functional characteristics of these four modules.

6.1.1 Short-Term Memory

Processing an input symbol to make a state transition is essentially a function of short-term memory. This is because such a process maintains the state information, and the current state encodes the recent input symbols.

It is here that the finite nature of the approximate realization comes into play. If the depth of the tree were to be unbounded, then the number of states would also be unbounded and any realization would require an unbounded amount of information to encode the current state. Consequently, for an implementation to be physically realizable, there must be a finite number of states. The easiest means of ensuring the finiteness of Q is to bound the depth of the tree to be built.

Boundedness of the growth also simplifies the implementational details of structural adaptation. By having a predefined size on the automaton tree, all possible states can be "pre-wired" initially, but many of them will be unactivated. This means that initial values for C, E and P^{Δ} will be maintained by other modules in the system for use when a new state is first activated.

6.1.2 Expectation Memory

The expectation (or long-term) memory serves to maintain the expectation (E) functions. It maintains them both as a repository for the E values and by effecting their modification as part of the learning process.

Similar to the short-term memory, values of E for non-existent transitions can be initially set to 0. All changes applied to them will be of 0 value until the corresponding transition is first activated.

6.1.3 Learning Control

The primary function of the learning control module is to identify when and what type of learning should take place and to signal the learning mechanisms in the output selection network accordingly. Learning in output probabilities is triggered by any of the following three events:

1. Entering a reward state,
2. Entering a punishment state or
3. A change in the output symbol.

Upon detecting any of these events, the learning control module must signal the appropriate type of learning in the output selection module.

6.1.4 Output Selection Module

The output selection module is responsible for maintaining the output probability distributions and for using them to randomly select output symbols. Like the expectation memory module, maintenance of the distribution implies both storage and learning.

Handling the unactivated transitions is more difficult in the output selection module than in other modules. In this case, the module must maintain a suitable initial probability distribution for unused transitions. This initial probability distribution may be simply a copy of the distribution on another transition or may be some combination of the distributions on other transitions.

6.2 Implementation Details

The details of implementation are intimately tied to the implementation technology. It would be a relatively simple matter to implement this model using conventional digital logic. However, it would not be very interesting from an artificial intelligence or cognitive science point of view. Instead, this section addresses the question: "How can cybernetic automata be implemented using model neurons?" Each subsection outlines a design of one of the modules using model neurons as design components. The implementational considerations given here do not make up a complete neuron-by-neuron design. Instead, for each of the key functional components of the model, the characteristics of a neural implementation are considered.

6.2.1 Short-Term Memory

There are four functions performed by the short-term memory network:

1. Edge Detection
2. Dominance Detection
3. State Machine
4. τ Time-Out Detection

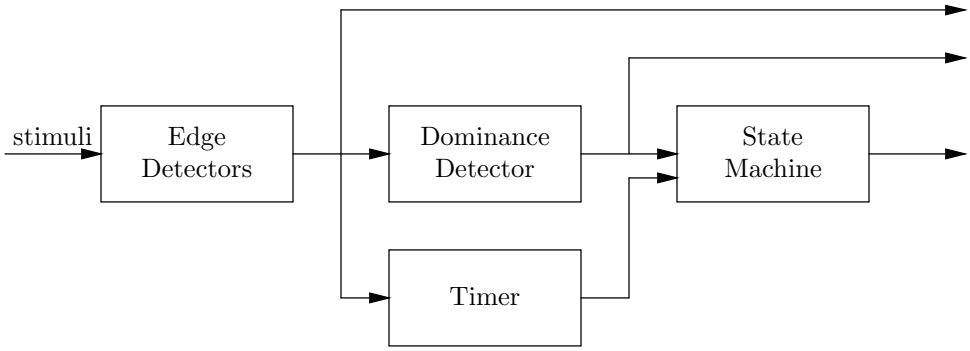

Figure 6.2: Organization of Short-Term Memory

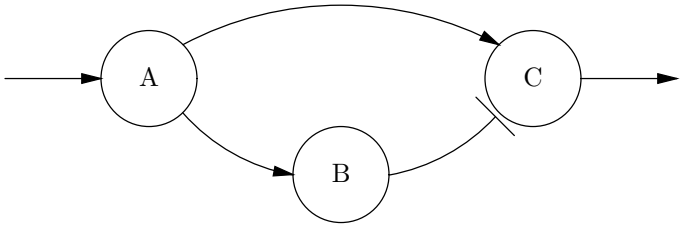

Figure 6.3: Onset Edge Detector

These four functional components fit together as shown in Figure 6.2.

As discussed in Chapters 2 and 4, the input symbol set for a cybernetic automaton is made up of symbols which denote the onset and offset of external stimuli. Few, if any, neuron models account for this need directly, but this implementation will not require a neuron model to do so. Instead, the neuron model is required to have two biologically accurate characteristics. First, it must exhibit some delay in activating the output after receiving an input signal. Second, it must provide for neurons that can fire spontaneously; that is, which produce an output without any input. With these two characteristics, an edge detector which outputs a temporary burst of activity upon the onset of its input can be constructed as shown in Figure 6.3. Here the A neuron will fire whenever the input is active. The B neuron serves as a delay which inhibits C only after A has been active for a short period of time. Consequently, when A changes from inactive to active, C will be active for the duration of B's delay. Similarly, the network shown in Figure 6.4 will detect the offset of a stimulus. Here the B and C neurons must fire spontaneously in the absence of input. The duration of B's delay is given by τ_B and is the shortest period of time which can be resolved by the system.

The task of determining the dominant input is easily accomplished by a competitive, winner-take-all network (Rumelhart & McClelland, 1986). Here the network consists of a single layer of mutually inhibited neurons, one for each input symbol. In the absence of inhibition, each neuron fires at a rate which is proportional to the input symbol strength. The mutual inhibition ensures that only the neuron corresponding to the strongest input symbol will fire after a short settling time.

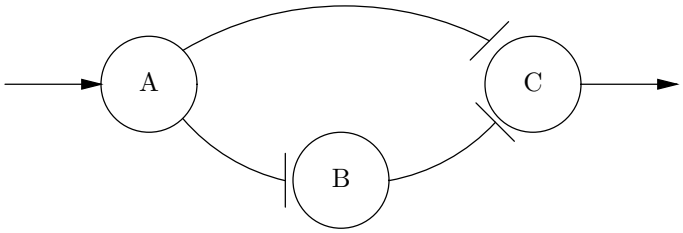

Figure 6.4: Offset Edge Detector

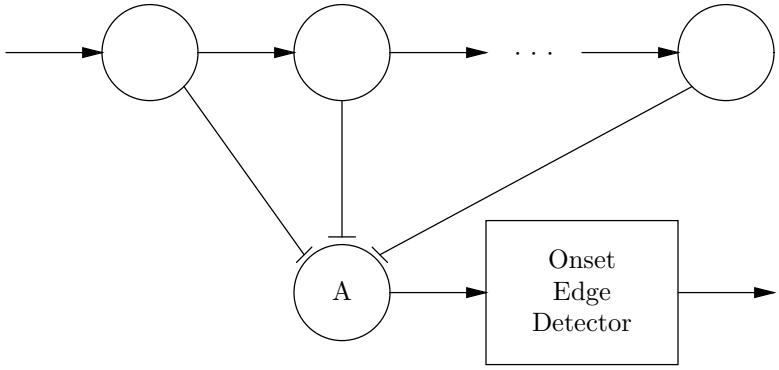

Figure 6.5: Timer for Short-Term Memory

Minsky (1967) has presented a constructive theorem showing that any finite state machine can be simulated by a neural network. With slight modification, this construction can be used here to implement the deterministic automaton underlying the cybernetic automaton model. The necessary modification is to latch the current state until a new input symbol arrives. This is needed since Minsky's construction assumes that input symbols will remain active until the next one arrives. It is here that the finite approximation in this implementation is most crucial. Only with a finite approximation is the underlying deterministic (non-output) automaton static in structure and consequently is susceptible to Minsky's construction.

The last functional component of the short term memory is the timer which determines if an ϵ-transition is to be taken. The most straightforward approach to implementing this timer is to use $\frac{\tau}{\tau_B}$ neurons each with delay τ_B in a delay line, where the input to the first delay element is the OR of all input symbols. Then if any of the delay line neurons are firing there is no signal sent to take an ϵ-transition. If none are active then, the A neuron will spontaneously fire. This network is shown in Figure 6.5.

6.2.2 Expectation Memory

The only function of the expectation memory is to maintain the expectation function and its only external effect is decreasing the confidence values in the output selection module. This is accomplished

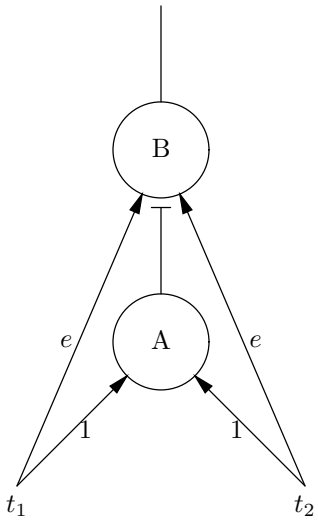

Figure 6.6: Expectation Memory Exclusive-Or Network

by providing one exclusive-or network (shown in Figure 6.6) for each pair of transitions which may have an expectation link. There will be $|\Sigma|$ such networks for each transition, one for the preceding transition and $|\Sigma| - 1$ for those input symbols which might occur simultaneously. The weights, e, are equal to the expectation between the transitions t_1 and t_2. Neuron A detects the case where both t_1 and t_2 are active and neuron B detects the case where exactly one of t_1 and t_2 are active. When neuron A fires, the e's are increased by $\alpha(1 - e)$. When neuron B fires, the e's are decreased by αe. Furthermore, global mediators in the amount of the absolute value of Δe are emitted into the output selection network to decrease $C(\cdot, \cdot)$.

6.2.3 Learning Control

The learning control module is perhaps the simplest of the four modules. It decodes the sets of reward (R) and punishment (P) states. This is simplified since the state output of the short-term memory has exactly one active output neuron corresponding to the current state. Consequently, the learning control module must have two neurons which act as ORs with inputs coming from those state outputs which correspond to states in the sets R and P, respectively. These neurons activate global mediators which trigger changes in synaptic weights for those synapses which have been marked for reinforced learning. The details of the marking are discussed in the next sub-section.

6.2.4 Output Selection Module

There are two basic elements of the design of the output selection module, the random selection of an output symbol and the modification of probabilities in learning. Figure 6.7 shows the neural organization of an implementation of the output selection module, where $n = |Q||\Sigma|$ and $m = |\Delta|$.

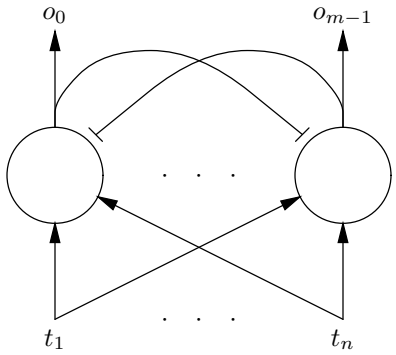

Figure 6.7: Output Selection Module

The nature of the cybernetic automaton model requires that output symbols be chosen probabilistically. This can be easily accomplished neurally using a modification of the neuron model presented in the Appendix. Specifically, let the probability that a neuron fires within a small period of time be, $p_i = \varphi_i/\Theta_i$. Furthermore, let the synaptic weight for output symbol, i, on transition, j satisfy

$$w_{ij} = P^{\Delta}_{(q_j,a_j)}(o_i) \sum_{k=1}^{|\Delta|} w_{kj}.$$

Since all of the output neurons are mutually inhibiting, the first one to fire inhibits the rest and becomes the only one to fire. Also, because all output neurons have the same input, the probability of one firing first will be proportional to its synaptic weight w_{ij}. Consequently, the probability that output symbol i is selected is approximately $w_{ij}/\sum_{k=1}^{|\Delta|} w_{kj} = P^{\Delta}_{(q_j,a_j)}(o_i)$.

If the confidence function is taken to be

$$C(q,a) = \sum_{k=1}^{|\Delta|} w_{(q,a),k}$$

then a constant change in weight, ζ, leads to a change in first firing probability given by:

$$p_i' = \frac{w_{ij}+\zeta}{\sum_{k=1}^{|\Delta|} w_{(q,a),k}+\zeta} = \frac{p_i + \zeta\frac{1}{\sum_{k=1}^{|\Delta|} w_{(q,a),k}}}{1+\zeta\frac{1}{\sum_{k=1}^{|\Delta|} w_{(q,a),k}}} = \frac{p_i+\zeta(1/C(q,a))}{1+\zeta(1/C(q,a))}$$

which is the same learning form as in the cybernetic automaton model. Learning based on constant incremental changes in weights can be found in the neuron model in the Appendix. Instead of being mediated by simultaneous active inputs, however, the weight changes here must be triggered by a global mediator which is activated by changes in the output symbols. There must also be a mechanism for each synapse to "remember" that it has recently activated the output. Though not

serving exactly the same function, enzymes have been found that are activated by synaptic activity and which affect further neurochemical activity (Aoki & Siekevitz, 1988). Similarly, evidence has been found that N-methyl-D-aspartate (NMDA) may act as a mediator which affects the synaptic weight (Kalil, 1989).

Likewise, the form of the confidence function given here is correctly decreased as a result of changes in expectation. Let the weights associated with a transition be decreased by the factor of $(1 - \beta|\Delta e|)$. Thus the confidence function will be reduced by a factor of $(1 - \beta|\Delta e|)$ as specified in the model.

6.3 Summary

This chapter views the problem of implementing the cybernetic automaton model much like an engineering problem where the neurons are the components out of which the design is built. In keeping with that analogy, the design presented here is at the block diagram level. No attempt has been made to present a neuron-by-neuron detailed design. Several engineering details have been left out. For instance, what happens in the output selection module if more than one output neuron simultaneously fires first? Similarly, the detailed mechanism by which a change in output symbol is detected is not addressed. Issues such as these are left as design details which could be addressed with relative ease. The design presented here is intended to be much like a "feasibility proof" demonstrating that the model can indeed be implemented using neuron-like elements.

Since neuron-like elements have been used to implement a model which has been shown to mimic some of natural learning, the question of biological faithfulness naturally arises. To what degree does the implementation presented in this chapter represent how real brains work? Unfortunately, there is not a clear answer to this. Since the neurons used here are largely based on those discussed in the Appendix and since those in the Appendix have shown some success in modeling a natural brain, it seem that the neurons here might be a reasonable abstraction of real ones. It is also interesting to note that use has been made of neurons of various types: ones that do not learn and ones that show differing types of learning. This is consistent with what is known about natural brains, especially those of higher organisms. Namely, that natural brains have a variety of neuron types.

The question of complexity is less favorable to biological faithfulness. Since use is made of individual neurons which represent each transition and since this is the largest single group of neurons, the number of neurons used is on the order of the number of transitions. Similarly, since each state (except q_0) has exactly one entering transition, the number of transitions is order the number of states. Finally, because the automaton is formed as a tree, the number of states is $\mathcal{O}(|\Sigma|^d)$ where d is the depth bound on the tree. If the input alphabet has 100 symbols and the depth bound is 10, then the number of neurons needed is on the order of 10^{20}. However, most estimates place the number of neurons in the human brain at 10^{10} or 10^{11}. Consequently, to the extent that natural brains implement the cybernetic automaton model, they must use a more efficient design than that which is presented here.

Chapter 7

Conclusions

The computational model presented here satisfies many of the goals for artificial intelligence expressed in Chapter 1. In particular, it exhibits probabilistic and adaptive behavior. It mimics some significant aspects of natural learning, and there exists an implementation of the model using model neurons that are reasonable abstractions of biological ones.

This chapter summarizes the experimental and theoretical results that support the claim that this model satisfies the goals. It also presents some open questions raised by the work and presents directions that may be taken in further work.

7.1 Contributions of the Model

The results summarized in this section demonstrate that the cybernetic automaton model does satisfy the goals set in Chapter 1. In brief, cybernetic automata do mimic much of natural learning, cybernetic automaton learning is provably complete in some sense, and there exists a neural implementation of the model. These characteristics satisfy the requirements set forth in detail in Chapter 2.

7.1.1 Summary of Theoretical Results

The primary theoretical result of cybernetic automata theory is the cybernetic automata learning theorem from Chapter 3. This theorem states that cybernetic automata learning is, in a sense, complete. It is complete in that for any POA given as a target, there exists a string which will cause an initial cybernetic automaton to learn a structure and set of probability distributions equivalent to the target POA.

Chapter 6 presents some other theoretical considerations related to how closely the cybernetic automaton is related to natural intelligence. These considerations lead to an implementation of the model using model neurons. While the model neurons are somewhat biologically plausible, the overall implementation is far less plausible. In particular, the model neurons are based on those which are used in the Appendix to successfully model some of the learning that has been observed in the

hermissenda mollusc. However, the number of such neurons required by the model implementation given in Chapter 6 is too large to be plausible as a model of natural brains.

7.1.2 Summary of Experimental Results

The conditioning simulation results demonstrate that the cybernetic automaton model exhibits the following set of conditioning properties:

1. First-Order Delayed Conditioning
2. Second-Order Delayed Conditioning
3. Third and Higher-Order Conditioning
4. First-Order Simultaneous Conditioning
5. Second-Order Simultaneous Conditioning
6. Extinction
7. Silent Extinction
8. Extinction in Second-Order Conditioning
9. Latent Inhibition
10. Partial Reinforcement
11. Compound Conditioning
12. Blocking
13. Sensory Preconditioning

Simulations of reinforced learning also demonstrate that cybernetic automata exhibit reward-based learning as in the Skinner box and other reinforcement learning as in the balance problem. Furthermore, considerations into the nature of the learning algorithm indicate that the model accounts for stimulus generalization and the effects of stimulus strength to some extent.

While all of the simulations illustrate qualitative exhibition of these properties, no attempt has been made to quantitatively emulate any particular species. It is reasonable, however, to expect that this may be possible for at least some species through careful tuning of the model's parameters.

Cybernetic automata do not exhibit any of the following known learning phenomena:

1. Backward Conditioning
2. Temporal Conditioning
3. Spontaneous Recovery

It is perhaps not fair to list the lack of backward conditioning as a criticism of the model since it is only occasionally found in natural learning. Both temporal conditioning and spontaneous recovery require a more advanced representation and use of time than exists in the current model. Spontaneous recovery is particularly difficult to address because any spontaneous changes in probability distributions or structure would adversely affect the learning theorem.

7.2 Open Questions Raised by the Model

The most obvious open issues raised by the model are those related to the phenomena which are not exhibited by it. For instance, what mechanisms should be in place for backward conditioning to work and in what circumstances? Or are there circumstances in which backward conditioning will occur in cybernetic automata? If the proper set of expectation links were in place, then it is conceivable that backward conditioning would occur. Similarly, how might temporal conditioning be accounted for? If ϵ-transitions were allowed to produce output and if the time required to trigger them were learnable, then temporal conditioning could take place. As mentioned above, the issue of spontaneous recovery is particularly sticky, and similar care must be taken in suggesting any additional mechanisms to insure that they do not adversely affect the existing characteristics.

It is clear that cybernetic automata do not explain all of intelligence or learning. However, Minsky (1986) has suggested that the brain might be a collection of interacting "modules," each of which performs some primitive activities. The investigation of whether or not some of these modules might be cybernetic automata could produce some interesting results. If Minsky's suggestion could indeed be turned into a fruitful model and if some or all of the modules could be structured as cybernetic automata, then the goal of shedding light on natural intelligence would certainly be fulfilled. Whether or not such a project were successful, the presentation of this one model for some part of intelligence is significant.

Traditionally, the development of theoretical models in science raises specific questions which can be answered experimentally. This model is no different. The proposed existence of the confidence and expectation functions describes not only the behavior considered thus far but other behaviors too. For instance, if conditioning is carried out much longer than was done in the extinction experiments, extinction does not occur. This is because during the conditioning, the confidence function was increased to the point that during extinction trials it did not decrease enough for there to be significant change in probabilities before it began to increase again. This effect raises the question of whether or not natural systems exhibit the same behavior.

7.3 Conclusion

Two fundamental goals of artificial intelligence are to mimic the behavior of naturally intelligent systems and to provide plausible models of natural intelligence. The cybernetic automaton model of computation does this. It exhibits much of the behavior which is classified as classical conditioning and instrumental learning. The suggested neural implementation opens the possibility that the neural structures implementing functions like those in cybernetic automata might be found.

The existence of the cybernetic automaton learning theorem is a significant element of cybernetic

automata theory as a computational model. It provides an element of generality, much as does the Church-Turing thesis for Turing machines.

Cybernetic automata do not represent the last word in machine learning or artificial intelligence. In many ways the theory represents the first word in a new direction. It models the low-level learning found in natural intelligences with a novel automaton model which can be implemented using biologically plausible neurons. Such a direction is indeed new and the success of this initial endeavor suggests that it is a fruitful approach to continued research in artificial intelligence.

Appendix A

A Model of Learning in Hermissenda Mollusc

Introduction

Much of the recent work in neural networks has shown some promise of contributing to our knowledge of how biological brains work. The resulting models contribute to artificial intelligence by simplifying the solution to problems in some domains. One approach to developing such models of neural computation is to base them on current knowledge of biological neural systems. That is the approach taken here. The model presented in this paper is based on current understanding of the functional characteristics of natural neural systems. Then the model is used to accurately simulate some aspects of the learning behavior of the hermissenda mollusc.

Alkon and his colleagues have done extensive studies of the mollusc hermissenda crassicornis. Two of their results are of particular interest here. First, they have mapped many of the neural pathways responsible for the natural phototaxis response. This will be discussed in a later section. Second, they have observed conditioning resulting from paired light presentation and rotation (Crow & Alkon, 1978; Farley & Alkon, 1980; Alkon, 1986; Farley, 1986). Successive presentations of paired light and rotation result in a suppression of the animal's phototaxis response. Presentations of light alone or rotation alone yield no learning. In contrast, if learning took place whenever light (or rotation) was present, then the learning could be accounted for by a non-associative process. Thus one can conclude that associative learning takes place rather than simple habituation or sensitization.

This paper will consider a neural-based model of computation which successfully simulates the molluscan responses described here. No attempt is made to describe or to accurately simulate the molecular mechanisms responsible for the functional characteristics observed. Instead, the model presented is meant to functionally simulate the behavior observed in the biological preparation. This is true for both behavior in the absence of learning and for behavior during and after training.

Functional Model of a Neuron

The neuron model used in these experiments represents a moderate level of abstraction of biological neural characteristics. A neuron in this model consists of a dendritic tree, a cell body and an axon. The dendritic tree forms a set of inputs to the neuron. These inputs are fed by the outputs of other neurons which are modulated by synaptic weights. The cell body performs temporal and spatial integration of the inputs and produces output signals which exit the neuron via the axon.

The output signal, which is sent along the axon, is a train of impulses each of which is generated whenever an internal charge state exceeds the neuron's firing threshold. This is expressed by

$$o_i(t + \delta_i) = \delta(t - t_0), \text{ for } \varphi_i(t_0) \geq \Theta_i$$

where $\varphi_i(t)$ is the ith neuron's charge state at time t, Θ_i is the neuron's threshold, δ_i is the neuron's delay time, and $\delta(t)$ is defined by

$$\delta(t) = 0, \text{ for } t \neq 0,$$

and

$$\int_{-\infty}^{+\infty} \delta(t)dt = 1.$$

At time t_0, when the charge state exceeds the threshold, the charge state $\varphi_i(t_0)$ is reduced to 0.

The charge state mentioned above is the result of a lossy time and space integration over the neuron's inputs. This integration is expressed by the equation

$$\varphi_i(t') = \int_{t_0}^{t'} \sum_j (\omega_{ij}(t)o_j(t)) + \mu_i - \rho_i\varphi_i(t)dt$$

where $\varphi_i(t)$ is the internal state of the cell body (and may represent some type of electrical or chemical charge), ω_{ij} is the weight applied to the connection from unit j to unit i, μ_i is a charge pump which can generate spontaneous activity in resting cells, and ρ_i is the charge loss from unit i. The charge pump parameter μ_i allows a normally idle neuron to fire spontaneously. This model allows incoming impulses which are not simultaneous but which are close in time to be integrated together. The sensitivity to time proximity is controlled by the loss parameter ρ_i.

In sensory neurons, one input is used for the external stimulus. Here the stimulus presents a continuing level of input modulated by its respective input weight. The stimulus input level is generally a monotonic function of the strength of the physical stimulus itself.

Learning Model of a Neuron

The model of learning used in this work is modified Hebbian learning. In most statements of Hebbian learning, the synaptic weight is modified according to the correlation of pre- and post-synaptic activity. No regard is given to the correlation among separate inputs.

Here learning entails the modification of synaptic weight based on the correlation of post-synaptic activity with correlated activity of multiple inputs. In particular, when the post-synaptic neuron fires an output pulse, all active inputs increase the strength of other active excitatory synapses. It

is important to note that active inhibitory inputs as well as excitatory inputs increase the synaptic strength of other active excitatory inputs.

The activity level of an input is determined by a low-pass filtering of the input. Specifically, the activity level, A_{ij}, of the output of neuron j impinging on neuron i is given by

$$A_{ij}(t) = \alpha_{ij} A_{ij}(t-1) + (1-\alpha_{ij}) o_j \omega_{ij}$$

where α_{ij} is the time constant of the filter. The input is considered to be active when $|A_{ij}| > 0.1|\omega_{ij}|$. The weight modification is then given by

$$\Delta\omega_{ij} = \sum_{\substack{k \\ k \neq j}} \begin{cases} sgn(\omega_{ij})\eta|A_{ik}|, & o_i \neq 0,\ |A_{ik}| > 0.1|\omega_{ik}|; \\ 0, & \text{otherwise,} \end{cases}$$

where k runs over all neurons impinging upon neuron i and η is the learning rate. The synaptic modification is applied to all inputs at each time step. While input weights may be either positive or negative, they are constrained to less than or equal to unity magnitude.

Neural Organization of the Hermissenda Mollusc

As mentioned above, Alkon and his associates have done extensive work in mapping the neural pathways of the hermissenda (Alkon & Fuortes, 1972; Alkon, 1973, 1975; Akaike & Alkon, 1980; Tabata & Alkon, 1982). The simulations described here use a simplified neural circuitry. Whereas the actual mollusc has five neurons in each eye, two called 'A' cells and three called 'B' cells, the simulations here use one of each. Likewise, all the cells of the molluscan optic ganglion are simulated by one 'E' cell and one 'C' cell.

Figure A.1 shows the neural circuitry for one half of the simulated molluscan nervous system. The other half is identical to the one shown with the 'B' cells feeding the contralateral 'C' cells. As mentioned above, the 'A' and 'B' cells are photoreceptors. The cells labeled 'Ceph,' 'In,' and 'Caud' are part of the mollusc statocyst, which senses external rotational forces applied to the animal. The 'Ceph' cell responds to rotational forces directed toward the animal's head. Conversely, the 'Caud' cell responds to forces directed away from the animal's head. In the actual mollusc, the 'In' cells are situated at a 45° angle on either side of the 'Caud' cells. Finally, the 'M' cell drives the ipsilateral muscle fibers.

The initial (prior to learning) weights on each connection are given in Table A.1. Other neural parameters of each cell (Θ_i, δ_i, μ_i, and ρ_i) are given in Table A.2. The δ_is are given in units of time steps and the values for μ_i are added at each time step.

Few indications of synaptic weights and other parameters have been given in Alkon's work. An attempt has been made, however, to be faithful to those which are given. Others were determined empirically. First the parameters were set so that the simulated animal would possess the requisite phototaxis response. Then they were adjusted so that the learning would operate correctly without destroying the original response. For all neurons, the α_{ij} parameter was set to 0.9, and the η parameter was set to 0.00005.

The light stimulus applied to all four photoreceptor neurons was continuous and unity magnitude. To prevent both halves of the simulated net from responding identically and synchronously, and thus

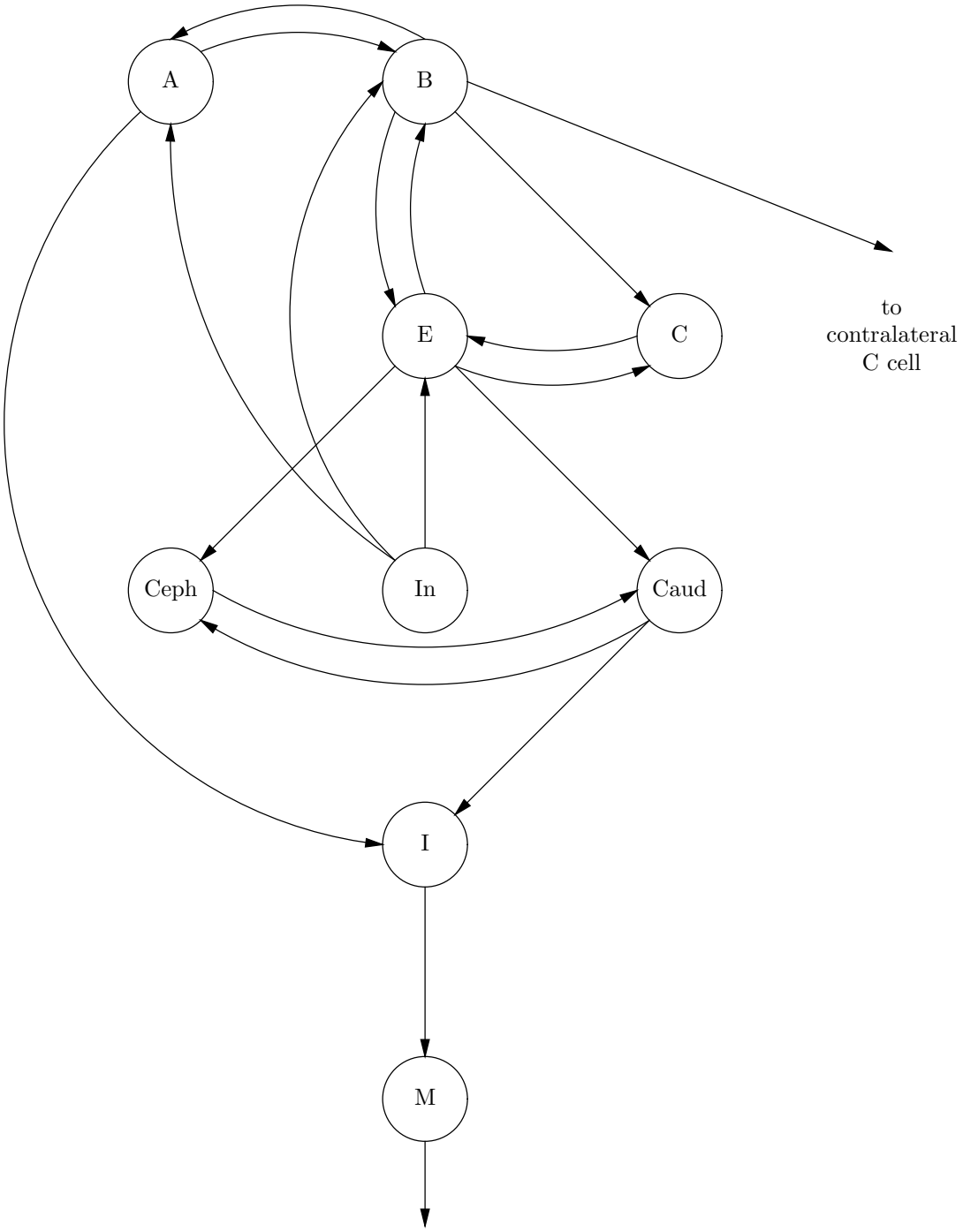

Figure A.1: Half of the Simulated Molluscan Brain

Table A.1: Connection Strengths (ω_{ij})

$i \backslash j$	A	B	C	E	Ceph	In	Caud	I	M
A	—	−0.9	—	—	—	−0.5	—	—	—
B	−0.3	—	—	0.1	—	−0.3	—	—	—
C*	—	−0.9	—	0.2	—	—	—	—	—
E	—	−0.9	−0.9	—	—	−0.7	—	—	—
Ceph	—	—	—	0.1	—	—	−0.7	—	—
In	—	—	—	—	—	—	—	—	—
Caud	—	—	—	−0.7	−0.7	—	—	—	—
I	0.7	—	—	—	—	—	0.3	—	—
M	—	—	—	—	—	—	—	0.7	—

* In addition to the input from the ipsilateral 'B' cell, each 'C' cell has an input from the contralateral 'B' cell with weight −0.9.

Table A.2: Neuron Parameters

	ρ_i	Θ_i	δ_i	μ_i
A	5.0	1.0	50	0.0
B	20.0	1.0	10	0.0
C	5.0	1.0	10	0.0
E	5.0	1.0	10	0.15
Ceph	20.0	1.0	10	0.0
In	20.0	1.0	10	0.0
Caud	20.0	1.0	10	0.0
I	5.0	1.0	10	0.0
M	5.0	1.0	10	0.0

to make the simulation more realistic, a small amount of noise was added to the light stimulus. This noise was distributed between $-1/10$ and $1/10$. Rotational stimulus was also continuous and was equal to the magnitude of the rotational velocity. Simulations were conducted with the "animals'" heads oriented toward the center of the rotation. Therefore, the rotational stimulus was only applied to the 'In' and 'Caud' neurons. For the simulations described here, the 'In' neurons respond to the same stimulus as the 'Caud' neurons. As described in a previous section, sensory stimuli are modulated with input weights, as are synaptic inputs. For all the sensory inputs here, the initial weights were set to 0.1.

Results

All simulations reported here used a training regime of five cycles of stimulus and no stimulus. Each cycle consisted of five seconds of no stimulus followed by ten seconds of stimulus, followed by another five seconds of no stimulus. Thus each cycle lasted 20 seconds with consecutive ten-second stimulus periods being separated by a ten-second no stimulus period. Rotational stimuli in appropriate simulations were set to π radians per second (approximately 33rpm). No directionality was simulated for the eyes, as the animals were held positioned toward the light in the original experiments.

After the training set for each particular simulation, a three-second result test was made. This test consisted of one-half second of no stimulus followed by two seconds of light only stimulus followed by another one-half second of no stimulus. No learning took place during the test.

The simulations were performed with one millisecond time steps. All outputs were computed and then the learning algorithm was run. The learning algorithm was run for all cells during each time step.

Figure A.2 shows the behavior of the net during the test for training with both light and rotation stimuli. Time runs from left to right in the diagram. Each spike on a line represents a single neural impulse for one neuron. Likewise, Figure A.3 shows the test results for training with light stimulus only, and Figure A.4 shows the results for training with rotational stimulus only.

Discussion and Conclusions

It can be seen from the diagrams in Figures A.2 through A.4, that the network learns to suppress the phototaxis response for paired presentations of light and rotation. It does not learn for presentations of either light or rotation alone. The traces for the 'B' neuron in these diagrams indicate that learning takes place by an increase in the light response of that cell. Because the 'B' cell inhibits the 'A' cell, its increased firing response to light reduces 'A's activity below the threshold of firing 'I'. For training sets with light only or rotation only, 'B's response to light is not as strongly enhanced, and consequently 'A' continues to fire triggering 'I' which in turn triggers 'M.' This response to learning matches that observed by Alkon and his associates (Alkon, 1986; Farley, 1986).

There are clearly some other issues raised by this work. One issue for these simulations is how sensitive the learning is to the initial weights and parameters. In other words, how far can the initial weights and parameters be from those used in this work and still lead to correct learning behavior?

Table A.3: Sign of $\Delta\omega_{ij}$

Post-Synaptic Neuron Active/Inactive		Affected Input Exciter	Affected Input Inhibitor
Effector	Exciter	+/U	U/U
Input	Inhibitor	+/U	U/U

Similarly, when no existing system is available to guide the choice of weights and parameters, how should those values be chosen?

Some issues are more fundamentally tied to the learning model itself. For example, the model thus far does not address under what conditions an excitatory input should be weakened. Likewise, it does not address what should be done for inhibitory inputs to learn. These issues can be illustrated in Table A.3 which shows how one input affects the strength of another. The '+'s in the chart indicate positive $\Delta\omega_{ij}$s and the 'U's are unknown entries which were taken to be 0 for this work.

Another issue involves non-associative learning such as habituation and sensitization. This issue is related to the effect on learning of extra light-only or rotation-only stimuli during the training schedule. These extra stimuli are known to reduce or to extinguish the degree of learning achieved with paired stimulus training.

Despite the array of new questions raised by this work, the results presented here indicate that this model and its learning paradigm has promise for increased understanding of neural function. The model's features represent a functional abstraction of some of the known principles of neurobiology. The learning algorithm is based on Hebb's (1949) original statement, but rather than being the simplest interpretation, it is directly designed to exhibit associative learning. As a model for simulating some aspects of the neural computation and associative learning found in nature, it is successful. Furthermore, it shows the promise that it may be further refined and that it can be extended to become a more complete model of neural computation.

Figure A.2: Results of Learning with Paired Light and Rotation

Figure A.3: Results of Learning with Light Alone

Figure A.4: Results of Learning with Rotation Alone

BIBLIOGRAPHY

Akaike, T. & Alkon, D. L. (1980). Sensory Convergence on Central Visual Neurons in *Hermissenda*, *The Journal of Neurophysiology, 44,* 501–513.

Alkon, D. L. (1973). Intersensory Interactions in *Hermissenda*, *The Journal of General Physiology, 62,* 185–202.

Alkon, D. L. (1975). Responses of Hair Cells to Statocyst Rotation, *The Journal of General Physiology, 66,* 507–530.

Alkon, D. L. (1986). Changes of Membrane Currents and Calcium-Dependent Phosphorylation during Associative Learning. In D. L. Alkon & C. D. Woody (Eds.) *Neural Mechanisms of Conditioning* (pp. 3–18). New York: Plenum Press.

Alkon, D. L. & Fuortes, M. G. F. (1972). Responses of Photoreceptors in *Hermissenda*, *The Journal of General Physiology, 60,* 631–649.

Aoki, C. & Siekevitz, P. (1988). Plasticity in Brain Development. *Scientific American, 259*(6) 56–64.

Ash, T. (1989). *Dynamic Node Creation in Backpropagation Networks.* (ICS Report No. 8901) San Diego: University of California, Institute for Cognitive Science.

Babbage, H. P. (1984). *Babbage's Calculating Engines: A Collection of Papers.* In *The Charles Babbage Reprint Series for the History of Computing* (Vol. 2). Cambridge: MIT Press. (Original work published 1889)

Charniak, E. & McDermott, D. (1987). *Introduction to Artificial Intelligence.* Reading, MA: Addison-Wesley.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike Adaptive Elements that can Solve Difficult Learning Control Problems. *IEEE Transactions on Systems, Man and Cybernetics, 13,* 834–846.

Cohen, D. I. A. (1986). *Introduction to Computer Theory.* New York: John Wiley & Sons.

Cohen, P. R. & Feigenbaum, E. A. (Eds.). (1982). *The Handbook of Artificial Intelligence* (Vol. 3). Reading MA: Addison-Wesley.

Crow, T. J. & Alkon, D. L. (1978). Retention of an Associative Behavior Change in *Hermissenda. Science, 201*, 1239–1241.

Farley, J. (1986). Cellular Mechanisms of Causal Detection in a Mollusk. In D. L. Alkon & C. D. Woody (Eds.) *Neural Mechanisms of Conditioning* (pp. 19–54). New York: Plenum Press.

Farley, J. & Alkon, D. L. (1980). Neural Organization Predicts Stimulus Specificity for a Retained Associative Behavioral Change. *Science 210,* 1373–1374.

Fukushima, K. (1975). Cognitron: A Self-Organizing Multilayered Neural Network. *Biological Cybernetics, 20,* 121–136.

Fukushima, K. (1984). A Hierarchical Neural Network for Associative Memory. *Biological Cybernetics, 50,* 105–113.

Fukushima, K. (1986). A Neural Network Model for Selective Attention in Visual Pattern Recognition. *Biological Cybernetics, 55,* 5–15.

Fukushima, K. (1987). Neural Network Model for Selective Attention in Visual Pattern Recognition and Associative Recall. *Applied Optics, 26,* 4985–4992.

Fukushima, K. & Miyake, S. (1982). Neocognitron: A New Algorithm for Pattern Recognition Tolerant of Deformations and Shifts in Position. *Pattern Recognition, 15,* 455–469.

Gardner, M. (1962). Mathematical Games. *Scientific American 206*(3), 138–144.

Gelperin, A., Hopfield, J. J., & Tank, D. W. (1985). The Logic of Limax Learning. In A. I. Selverston (Ed.) *Model Neural Networks and Behavior* (pp. 237–261). New York: Plenum Press.

Glushkov, V. M. (1966). *Introduction to Cybernetics.* (Scripta Technica, Inc., Trans. G. M. Kranc, Ed.) New York: Academic Press. (Original work published 1964)

Hall, J. F. (1976). *Classical Conditioning and Instrumental Learning: A Contemporary Approach.* Philadelphia: J. B. Lippincott.

Hebb, D. O. (1949). *The Organization of Behavior,* New York: Wiley.

Hopcroft, J. E. & Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation.* Reading, MA: Addison-Wesley.

Kalil, R. E. (1989). Synapse Formation in the Developing Brain. *Scientific American, 261*(6), 76–85.

Lee. Y. C., Qian, S., Jones, R. D., Barnes, C. W., Flake, G. W., O'Rourke, M. K., Lee, K., Chen, H. H., Sun, G. Z., Zhang, Y. Q., Chen, D., & Giles, C. L. (1990). Adaptive Stochastic Cellular Automata: Theory. *Physica D, 45,* 159–180.

Lewis, H. R. & Papadimitriou, C. H. (1981). *Elements of the Theory of Computation.* Englewood Cliffs, NJ: Pretice-Hall.

McCulloch, W. S. & Pitts, W. (1943). A Logical Calculus of the Ideas Imminent in Nervous Activity. *Bulletin of Mathematical Biophysics, 5,* 115–133.

Minsky, M. L. (1967). Computation: Finite and Infinite Machines. Englewood Cliffs, NJ: Pretice-Hall.

Minsky, M. L. (1986). *The Society of Mind.* New York: Simon & Schuster.

Minsky, M. L. & Papert, S. A. (1969). *Perceptrons.* Cambridge, MA: MIT Press.

Misiak, H. & Sexton, V. S. (1966). *History of Psychology.* New York: Grune & Stratton.

Nilsson, N. J. (1965). *Learning Machines.* New York: McGraw-Hill.

Paz, A. (1966). Some Aspects of Probabilistic Automata. *Information and Control, 9,* 26–60.

Qian, S., Lee, Y. C., Jones, R. D., Barnes, C. W., Flake, G. W., O'Rourke, M. K., Lee, K., Chen, H. H., Sun, G. Z., Zhang, Y. Q., Chen, D., & Giles, C. L. (1990). Adaptive Stochastic Cellular Automata: Applications. *Physica D, 45,* 181–188.

Rabin, M. O. (1963). Probabilistic Automata. *Information and Control, 6,* 230–245.

Rosenblatt, F. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Washington: Spartan Books.

Rumelhart, D. E., Hinton, G. E., & McClelland, J. L. (1986). A General Framework for Parallel Distributed Processing. In D. E. Rumelhart & J. L. McClelland (Eds.) *Parallel Distributed Processing: Vol. 1. Foundations* (pp. 45–76). Cambridge: MIT Press.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning Internal Representations by Error Propagation. In D. E. Rumelhart & J. L. McClelland (Eds.) *Parallel Distributed Processing: Vol. 1. Foundations* (pp. 318–362). Cambridge: MIT Press.

Rumelhart, D. E. & McClelland, J. L. (Eds.). (1986). *Parallel Distributed Processing: Vol. 1: Foundations.* Cambridge: MIT Press.

Sahley, C. L., Rudy, J. W., & Gelperin, A. (1984). Associative Learning in a Mollusk: A Comparative Analysis. In D. L. Alkon & J. Farley (Eds.) *Primary Neural Substrates of Learning and Behavioral Change* (pp. 243–258). Cambridge University Press.

Samuel, A. L. (1959). Some Studies of Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development, 3,* 211–229.

Tabata, M. & Alkon, D. L. (1982). Positive Synaptic Feedback in Visual System of Nudibranch Mollusk *Hermissenda crassicornis, The Journal of Neurophysiology, 48,* 174–191.

Thompson, R. F. (1972). Sensory Preconditioning. In R. F. Thompson & J. F. Voss (Eds.) *Topics in Learning and Performance* (pp. 105–129). New York: Academic Press.

Turing, A. M. (1970). Intelligent Machinery. In B. Meltzer & D. Michie (Eds.) *Machine Intelligence 5* (pp. 3–23). New York: American Elsevier Publishing Company.

von Neumann, J. (1958). *The Computer and the Brain.* New Haven: Yale University Press.