Chapter 13

Learning and Sequential Decision Making

Andrew G. Barto, Richard S. Sutton, and Christopher J. C. H. Watkins

1 Introduction

In addition to being studied by life scientists, learning is studied by engineers and computer scientists interested in developing useful devices and programs. The study of synthetic learning has produced an extensive collection of methods and mathematical theories pertaining to such tasks as pattern classification, prediction, and the adaptive control of dynamical systems. Within these theories learning is usually formulated as a search conducted in an abstractly defined space, and a large collection of mathematical concepts can be brought to bear on the problems of understanding and designing procedures, or algorithms, for enabling a device or program to improve its performance over time. What is the nature of the correspondence, if there is any, between the behavior of an animal in a classical conditioning experiment and the mathematical theories and computational procedures developed for synthetic learning? Is this behavior trivial from a computational perspective, or is it complex and subtle? The answers to these questions that we attempt to justify in this chapter are that the behavior observed in classical conditioning experiments is far from computationally trivial; its strongest ties are to mathematical theories and computational procedures that are exceedingly useful in practice and surprisingly complex. By relating the behavior of an animal undergoing classical conditioning to perspectives developed for understanding synthetic learning systems, we hope to provide a framework that may lead to increased understanding of animal behavior and to novel computational procedures for practical tasks.

Our analysis is based on the temporal-difference (TD) model of classical conditioning described by Sutton and Barto in chapter 12 of the present volume and in their 1987 paper. In this chapter we view the TD model as a computational method that can be useful in solving engineering problems. Sutton (1988) has shown how computational methods making use of "temporal differences," including the TD model of conditioning, are useful for adaptive prediction, and other publications illustrate how TD methods can be used as components of synthetic learning systems (Anderson 1987; Barto et al. 1983; Sutton 1984). Here, we restrict our attention to a slightly simplified version of the TD model, which we call the *TD procedure*. We show how the TD procedure is related to theoretical principles which serve both to explain the operation of TD methods and to connect them to existing theories of prediction, control, and learning. Some of the observations we make are further elaborated in Sutton 1988 and in Watkins 1989, and some of the connections to existing theory were previously described in Werbos 1977 and in Werbos 1987.

We show how a TD method can be understood as the synthesis of concepts from two existing theoretical frameworks: the theory of *stochastic dynamic programming*, which addresses sequential decision tasks in which both short-term and long-term consequences of decisions must be considered, and the theory of *parameter estimation*, which provides the appropriate context for studying learning rules in the form of equations for updating associative strengths in behavioral models, or connection weights in connectionist networks. Although a clear explanation of how the relevant theoretical ideas fit together requires a certain amount of mathematical notation, it is not our purpose to present a mathematically rigorous account of these ideas, and there are many issues that we do not pursue very deeply or do not discuss at all. Our goals are to explain the main ideas clearly and to provide some access to the large body of relevant theoretical literature.[1]

Drawing precise parallels between behavioral models and computational procedures facilitates the exchange of ideas between researchers studying natural learning and those studying synthetic learning. Sutton and Barto (1981) pointed out that the Rescorla-Wagner (1972) model of classical conditioning is identical (with some minor caveats) to the equation presented by Widrow and Hoff (1960) as a procedure for approximating solutions of systems of linear equations. As a behavioral model, this equation provides a remarkably simple account of a range of stimulus context effects in classical conditioning; as a computational procedure, it has proved exceedingly useful in technological applications, where it is called the LMS (least mean squares) algorithm. This and closely related algorithms are widely used in the fields of signal processing and pattern classification, and are playing a major role in the emerging field of connectionist modeling (Anderson and Rosenfeld 1988; Hinton and Anderson 1981; McCelland and Rumelhart 1986; Rumelhart and McClelland 1986). The connection between the experimental and computational literatures due to the parallel between the Rescorla-Wagner model and the LMS algorithm has allowed a fruitful exchange between researchers having widely differing ranges of expertise. In this chapter we hope to expand this basis for exchange by extending the parallels pointed out in Sutton and Barto 1981. Within the framework adopted here, the Rescorla-Wagner model—and hence also the LMS algorithm—appears as a specialization of the TD procedure.

The relationship between the TD procedure and dynamic programming outlined in this chapter also has the potential for fostering communication between animal learning theorists and behavioral ecologists. Dynamic programming has been used extensively in behavioral ecology for the analysis of animal behavior (Krebs et al. 1978; Houston et al. 1988; Mangel and Clark 1988). In these studies, dynamic programming is used to determine decision strategies meeting certain definitions of optimality to which animal behavior is compared. Behavioral ecologists do not suggest that the animals themselves perform dynamic programming—indeed, most of the forms of behavior studied are regarded as innate, and dynamic programming would appear to provide a poor model of learning. In a sense that will be made clear below, dynamic programming methods work backward, from the end of a decision task to its beginning, calculating information pertinent to decision making at each stage on the basis of information previously calculated from that stage to the task's end. As a result of this back-to-front processing, it is difficult to see how dynamic programming can be related to learning processes that operate in real time as a system interacts with its environment. However, we show how the TD procedure can accomplish much the same result as a dynamic programming method by repeated forward passes through a decision task (that is, through repeated trials) instead of by explicit back-to-front computation. This relationship of the TD procedure to dynamic programming suggests a range of research questions involving links between animal behavior in carefully controlled learning experiments and the less restricted forms of behavior studied by behavioral ecologists.

Sections 2 and 5 present the basic ideas of stochastic dynamic programming. Section 2 describes, in both informal and mathematical terms, a general class of tasks, known as *stochastic sequential decision tasks*, to which the methods of stochastic dynamic programming apply; section 5 describes some of these methods. This material is tutorial in nature and is based on the formulation of Ross (1983). Section 6 is a tutorial on parameter estimation based on the view taken in the field of adaptive control as described by Goodwin and Sin (1984). Some of this material also appears in Barto (in press). Section 7 shows how the TD procedure emerges as a synthesis of the ideas from dynamic programming and parameter estimation covered in the aforementioned sections. Here we also show how the TD procedure can be used in conjunction with another procedure and applied to stochastic sequential decision tasks to produce an analogue of instrumental learning. The combination of these two procedures corresponds to the use of the "adaptive critic element" and the "associative search element" in the pole balancer of Barto et al. (1983). The framework of stochastic sequential-decision theory helps explain the interaction of these two procedures and suggests other learning methods for this and related tasks. We conclude with a discussion of what the theoretical basis of the TD procedure suggests about animal learning and of some directions that can be taken in extending this approach.

2 *Sequential Decision Tasks*

Animals face many situations in which they have to make sequences of actions to bring about circumstances favorable for their survival. We are interested in tasks in which the consequences of an action can emerge at a multitude of times after the action is taken, and we shall be concerned with strategies for selecting actions on the basis of both their short-term and their long-term consequences. Tasks of this kind can be formulated in terms of a dynamical system whose behavior unfolds over time under the influence of a decision maker's actions.

Modeling the behavior of such a system is greatly simplified by the concept of *state*. The state of a system at a particular time is a description of the condition of the system at that time that it is sufficient to determine all aspects of the future behavior of the system when combined with knowledge of the system's future input. Whatever happened to the system in the past that is relevant to its future behavior is summed up in its current state—future behavior does not depend on how the system arrived at its current state, a property sometimes called "path independence" of the system description. The concept of state is also useful in describing systems which operate according to probabilistic rules. In this case, the system state and future input determine the probabilities of all aspects of the future behavior of the system independently of how the state was reached. This is the *Markov property* of a stochastic dynamical system.

Consider a decision-making agent, which we simply call an *agent*, facing the following task: The agent interacts with a system in such a way that at the beginning of each of a series of discrete time periods it observes the system and is able to determine the system state at that time. On the basis of observed state, the agent performs an action, thereby causing the system to deliver to the agent a "payoff," which we think of as a number whose value depends on the system state, on the agent's action, and possibly on random disturbances. The system then makes a transition to a new state determined by its current state, by the agent's action, and possibly by random disturbances. Upon observing the new state, the agent performs another action and receives another payoff, and the system changes state again. This cycle of state observation, action, payoff, and state change repeats for a sequence of time periods. The agent's task, described mathematically in section 4, is to select the actions that maximize the total, or cumulative, amount of payoff it receives over time. This is more difficult than merely trying to maximize each individual payoff. Some actions may be useful in producing a high immediate payoff, but these same actions may cause the system to enter states from which *later* high payoffs are unlikely or impossible. Hence, performing these actions would result in a smaller total amount of payoff than might be possible otherwise. Conversely, some actions that may produce low payoff in the short term are necessary to set the stage for greater payoff in the future. The agent's decision-making method must somehow account for both the short-term and the long-term consequences of actions.

The total amount of payoff received by the agent over many time periods depends on the number of time periods over which this total is determined, on the sequences of actions and states that occur over these time periods, and on the outcomes of whatever random factors influence the payoffs and the state transitions. The number of time periods over which the total amount of payoff is determined is called the *horizon* of the decision task. If the horizon is finite, the total amount of payoff is simply the sum of the individual payoffs received at each time period until the task's horizon is reached. If the horizon is infinite, however, this sum may not be finite—a difficulty that is remedied by introducing a *discount factor* that allows payoffs to be weighted according to when they occur. In this case, what we mean by the total amount of payoff over an infinite number of time periods is a weighted sum of the infinite number of individual payoffs received, where the weights decrease with increasing temporal remoteness of the payoffs (we define this precisely in section 4). If the discount factor is chosen appropriately, then this weighted sum will always have a finite value despite its dependence on an infinite number of payoffs. In chapter 12, Sutton and Barto refer to this as *imminence weighting*. In this chapter, we restrict our attention to infinite-horizon tasks where a discount factor is used in determining the relevant measure of the total amount of payoff.

Describing a decision task in terms of system states permits one to make a relatively simple statement of how action and state sequences determine the total amount of payoff an agent receives. Suppose the agent uses a rule to select action depending on system state. This rule, called the agent's *decision policy*, or simply its *policy*, associates an action with each system state. The agent's action upon observing a state is the action associated with that state by the policy. If no random factors are involved in the task, then the sequences of actions and states depend only on the agent's policy and on the system at the beginning of the task (i.e., the task's initial state). Consequently, the total amount of payoff received until the task's horizon is reached (where the total amount is determined by discounting if the task has an infinite horizon) also depends only on the agent's policy and on the task's initial state. By a policy's *return* for a given system state we mean the total amount of payoff the agent receives until the task's horizon is reached, assuming that the task's initial state is the given state and the agent uses the given policy. For infinite-horizon tasks where a discount factor is used, a policy's return for a state is the weighted sum (where the weights depend on the discount factor) of the payoffs the agent would receive over an infinite number of time periods for the given initial state if the agent were to use the given policy to select an infinite sequence of actions.[2] Thus, when no random factors are involved in a sequential decision task, the payoff for a system state depends on a single action of the agent, but the return for a state depends on the consequences of the agent's decisions as specified by its policy for the duration of the task.

When a decision task involves random factors, a policy's return for each system state is a random variable. In this case, one can define the *expected return* for each policy and system state. For the infinite-horizon case with discounting, which is our concern here, the expected return for a policy and a system state is the mathematical expectation, or mean, of the random variable giving the return for that policy and state. The expected return depends on the distribution functions of all the random factors influencing the task and can be thought of as the average of an infinite number of instances of the decision task, where the agent uses the same policy and the system starts in the same initial state in each instance. As formulated here, the objective of a sequential decision task is defined in terms of expected return: The objective is to find a policy that maximizes the expected return for all possible initial system states. Such a policy is called an *optimal policy*.

Although we discuss tasks requiring maximizing expected return, this class includes as special cases tasks in which the objective is to obtain any payoff at all. For example, suppose that for all but one system state the payoffs received by the agent are zero no matter what action the agent selects. Also suppose that the task ends when a nonzero payoff is obtained. One can think of the state from which nonzero payoff is available as the *goal state*. In this case, a policy's return for each initial state is zero unless its use by the agent brings about the goal state. Hence, in this case, selecting actions to maximize return is the same as selecting actions that cause the system to enter the goal state. If a discount factor is used, it turns out that the return is maximized by selecting actions that bring about the goal state in the fewest time periods. Tasks such as this, in which the objective is to reach a designated goal, are included in the theory we describe, and the example used throughout this chapter is an instance of this type of task.

There are numerous examples of sequential decision tasks, many of which have great practical significance. The task of finding the least costly route from one place to another is perhaps the most generic example. Choice points along a route correspond to the states of the system, actions determine what place is reached next, and the magnitude of the payoff received in response to an action is inversely related to the cost of the path traveled (so that by maximizing the total amount of payoff, one minimizes the total cost of the path). More complex tasks involving resource allocation, investment, gambling, and foraging for food are also examples of sequential decision tasks. Most of the planning and problem-solving tasks studied by artificial intelligence researchers are sequential decision tasks. Other examples are studied by control engineers, such as the problem of placing a spacecraft into a desired orbit using the least amount of fuel. In some sequential decision tasks, the distinction between the agent and the system underlying the decision task may not be as clear-cut as our discussion would lead one to believe. For example, in a foraging model in behavioral ecology, the state of the system may be the forager's energy reserves (Mangel and Clark 1988), a quantity apparently describing an aspect of the agent itself instead an external system. It can be misleading to identify an agent with an entire organism.

We use a simple route-finding task to illustrate the concepts and methods described in this chapter. This is merely an example; these concepts and methods are applicable to tasks that are much more complex.

Example

Figure 1 shows a grid representing a region of space. Each intersection of the grid lines is a "location," and the region contains a C-shaped barrier and a goal location. For the 8-by-12 grid shown, there are 96 locations. Two locations are adjacent if they are connected by a grid line that does not pass through any other locations. A path is a set of line segments tracing a route through the region, where each segment connects two adjacent locations in the region. The length of a path is the number of distinct line segments it contains. The task we consider is to find, for each location, a path to the goal that begins at the given location, does not cross the barrier, and has the smallest possible length. Each such shortest path is an optimal path.

We can formulate this task as a sequential decision task by considering an agent that can move from its current location to an adjacent location in each time period. The spatial environment defines the system underlying the decision task. For each time period, the state of the system is the current location of the agent, and the state at the next time period—the new location of the agent—is determined by the the current location of the

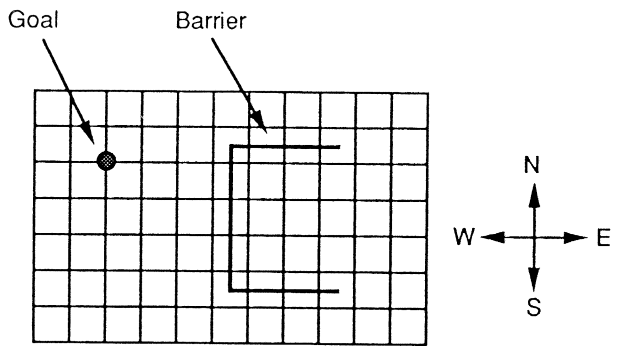

Figure 1
Plan view of a spatial environment for the route-finding example. The intersections of the lines are possible locations for the agent.

agent and by the action chosen. We let the agent choose any one of the four actions North (N), South (S), East (E), and West (W) at each time period. The effect of an action depends on the current system state, i.e., the current location of the agent. For most locations, the action causes the agent to move to the location adjacent to its current location in the direction indicated by the action. However, for locations from which a move is blocked by a barrier or would take the agent out of bounds, the effect of any "disallowed" action is to leave the agent's location unchanged. If the agent is located at the goal, it stays there no matter what action it performs. Thus, the set of actions available to the agent is always the same, but actions can have differing consequences depending on the agent's locations.[3]

A policy, in this example, is a rule that assigns an action to each location. One could think of a policy as one of the many possible patterns of placing at each location a signpost (indicating N, S, E, or W) which the agent is compelled to follow. The objective of the task is to form a policy (i.e., to place a pattern of 96 signposts) that directs the agent from each location to the goal in the fewest possible time periods—that is, that directs the agent along an optimal path. To formulate this as the problem of finding a policy that maximizes expected return, we effectively punish the agent every time period in which it is not at the goal. The agent always receives a payoff of -1 unless the agent is located at the goal, in which case it receives a payoff of 0 for any action. Therefore, the sum of payoffs over a path from a starting location to the goal, i.e, the return produced over the path, is the negative of the number of time periods taken to reach the goal (assuming no discounting). Selecting actions to maximize return therefore minimizes the number of time periods taken to reach the goal. An optimal policy directs the agent along an optimal path from each location.

In what follows, we discuss several versions of this task that differ in terms of the amount of knowledge we assume for the agent. Although all these tasks are relatively easy instances of the tasks encompassed by the theory, some of the sources of additional complexity can be appreciated clearly by means of the route-finding example. For example, the payoff received for a move need not always be -1 but can instead reflect the distance of a path, or the degree of difficulty encountered in traversing it. Additionally, the payoffs and the state transitions need not be deterministic. Actions may only influence the probabilities that specific places are reached. Finally, additional complexity, which we do not address at all in what follows, occurs if the agent does not have access to complete state information—a situation that would appear in the route-finding example when the agent is unable to distinguish all possible locations.

Before discussing solution methods for sequential decision tasks, we make several observations about the theoretical framework implied by this class of tasks. All these observations are reminders that it can be misleading to take the abstractions involved in this framework too literally.

- *Discrete time.* What do the discrete time periods of a decision task mean in terms of real time? In the kinds of discrete-time models to which we restrict our attention, these time periods are called *time steps* and are merely *computational instants* that may correspond to instants of real time separated by some interval, or to separate collections of events (such as trials). In modeling conditioning behavior, for example, a time step may represent some small interval of real time, as in the TD model presented by Sutton and Barto in this volume. Alternatively, each time step might be an entire conditioning trial, as in a trial-level model such as that of Rescorla and Wagner (1972). In the discussion to follow, a time step merely refers to a time period of an abstract sequential decision task.
- *Receiving payoff.* The framework described above is sometimes interpreted to mean that payoffs are delivered to the agent as if there were a single sensory pathway dedicated to this function. If one identifies the agent with an entire animal (which can be misleading, as we emphasized above), this view suggests that an animal has a single sensory input dedicated to the function of receiving all primary reinforcement, which is obviously not the case. It is better to think of the payoff at each time period as a concise way of summarizing the affective significance of the immediate consequences of performing an action. Because the immediate payoff and the system's next state are both functions of the current action and the system's current state, it is a special case to regard each payoff simply as a function of the system's current state. One can think of this function as being computed by the agent itself and not by the system underlying the decision task.

• *Complete state information.* The assumption is made that at each time step a complete description of the state of the system underlying the task is available to the agent. Clearly, this is a strong assumption. Although the consequences of weakening this assumption are theoretically significant and relevant to the study of natural learning, they are complex and beyond the scope of the present chapter. However, it is important to keep in mind the following two observations. First, one need not think of the current system state as something that must be read directly from the agent's sense organs. More generally, the state of the system can be provided through the aid of complex world models and memories of past sensations and behavior. The theory requires the availability of state information, but it is not sensitive to how this information is obtained. Second, knowing the current state of the system underlying a decision task is not the same as knowing beforehand how the system will behave in response to actions; that is, it is not the same as having an accurate model of the decision task.

• *Optimality.* The objective of a decision task considered here is to find an optimal decision policy—a policy that maximizes the expectation of the discounted sum of future payoffs. Some theories of animal behavior invoking optimality principles are "molar" optimality theories which propose that behavior can be understood in terms of optimization, but they do not specify mechanistic computational procedures by which the conditions of optimality can be achieved. "Molecular" optimization theories, on the other hand, specify procedures that operate from moment to moment but which only approximate optimal solutions in most tasks (see, for example, Rachlin et al. 1981 and Staddon 1980). Although here it is not our goal to provide a theory of animal behavior, we can describe the kind of theory with which the perspective taken in this chapter is consistent. The framework of sequential decision theory adopted here provides a molar view of behavior involving a specific optimality criterion, and the temporal-difference and policy-adjustment procedures we describe provide molecular accounts of how optimal behavior might be approximated. Because these procedures usually only approximate optimal policies, they do not imply that optimal behavior will be achieved. The performance of these procedures in specific decision tasks depends on the specification of many details, such as the manner of representing system states. Choices made in specifying these details strongly influence the degree of optimality achievable.

3 *Solving Sequential Decision Tasks*

Because so many problems of practical interest can be formulated as sequential decision tasks, there is an extensive literature devoted to the study of solution methods for this type of task, the large majority of which require the agent to have a complete model of the decision task. Even if one has a complete model of the decision task, which means knowing all the state-transition and payoff probabilities of the system underlying the task, determining the best policy can require can extremely large amount of computation because it effectively requires a search through the set of all possible state sequences generated by all possible sequences of actions. Except for very specialized tasks in which analytical methods can be used instead of search, the required amount of computation increases so rapidly with increases in a task's size (as determined by its horizon, its number of states, and its number of actions) that it is not feasible to perform this search for large tasks. Dynamic programming, a term introduced by R. E. Bellman (1957), consists of particular methods for organizing the search under the assumption that a complete model of the decision task is available. Although these methods are much more efficient than explicit exhaustive search of all possible state sequences, the amount of computation still grows so rapidly with the size of the task that large tasks remain intractable. Search methods specialized to take advantage of specific kinds of "heuristic" knowledge can be applied to some types of larger tasks, but these methods also require a complete model of the decision tasks and can still require prohibitive amounts of computation.[4]

Methods for estimating optimal policies in the absence of a complete model of the decision task are known as *adaptive* or *learning* methods. Because the most difficult aspect of applying dynamic programming is often the accurate modeling of the decision task, adaptive methods have great practical importance. In addition, if an adaptive method can improve a decision policy sufficiently rapidly, the amount of computation required may be less than would be required by an explicit solution via dynamic programming. How can an optimal policy be constructed when a complete model of the decision task is not available? Instead of being able to generate a solution by manipulating a task model, it is necessary to learn about the system underlying the task while interacting with it. Two general approaches are possible. The one that has been much more widely studied is the *model-based* approach, which requires constructing a model of the decision task in the form of estimates of state-transition and payoff probabilities. These probabilities can be estimated by keeping track of the frequencies with which the various state transitions and payoffs occur while interacting with the system underlying the decision task. Assuming that these estimates constitute an accurate model of the decision task, one can then apply a computational technique for finding an optimal policy, such as a dynamic programming technique, which requires an accurate model of the decision task.[5]

In this chapter our concern is with other approaches to learning how to solve sequential decision tasks, which we call *direct* approaches. Instead of learning a model of the decision task (that is, instead of estimating state-transition and payoff probabilities), a direct method adjusts the policy as a result of its observed consequences. Actions cannot be evaluated unless they are actually performed. The agent has to try out a variety of decisions, observe their consequences, and adjust its policy in order to improve performance. We call this process *reinforcement learning* after Mendel and McLaren (1970), who describe its relevance to adaptive control.[6] To facilitate the direct learning of a policy, it is possible to adaptively improve the criteria for evaluating actions so that the long-term consequences of actions become reflected in evaluations that are available immediately after an action is performed. The TD procedure is a method for accomplishing this.

In terms of theories of animal learning, a policy is a stimulus-response (S-R) rule, and the payoffs delivered to the agent at each stage of the decision task correspond to primary reinforcement. The TD procedure provides a gradient of secondary reinforcement by anticipating events that provide primary reinforcement. According to this view, secondary reinforcement is defined in terms of estimates of the expected *sum of future primary reinforcement* (possibly discounted). In subsection 7.2 we describe a direct method for adjusting policies on the basis of primary and secondary reinforcement, but it is not our aim to propose this method as a model of animal learning in instrumental tasks. Such learning probably involves more than can be accounted for by this kind of S-R model (see, for example, Dickinson 1980 and Rescorla 1987). Combining the TD procedure with model-based methods may provide a more satisfactory account of animal behavior in instrumental conditioning tasks. Watkins (1989) discusses some of the issues that arise in combining direct and model-based learning methods.

4 Mathematical Framework

4.1 Systems and Policies

We assume that the system underlying the decision task is a discrete-time dynamical system with a finite set of states, X. At any time step $t = 0, 1, 2, \ldots,$ the system can be in a state $x \in X$. After observing the system state at time step t, the agent selects (and performs) an action from a finite set, A, of possible actions. Suppose that at time step t, the agent observes state x and selects action a. Then, independent of its history, the system makes a transition from state x to state y with a probability that depends on the action a. We denote this probability $P_{xy}(a)$. When this state transition occurs, the agent receives a payoff, denoted r, which is determined randomly in a manner depending on x and a. This sequence of events repeats for an indefinite number of time steps. Because we shall be concerned only with the *expectation* of the total amount of payoff accumulated over time, it is not necessary to specify the details of the probabilistic process by which a payoff depends on states and actions. It suffices to specify only how the expected value of a payoff depends on actions and states. We let $R(x, a)$ denote the expected value of a payoff received as a consequence of performing action a when observing state x. We assume that the payoff whose expected value depends on the system's state and on the agent's action at time step t is received by the agent at the next time step: $t + 1$. Although this formalism can be modified to let the set of actions available to the agent depend on the system's state, for simplicity we have chosen to let the set of actions, A, be the same for each state.

The objective of the decision task is to find a policy for selecting actions that is optimal in some well-defined sense. In general, the action specified by the agent's policy at a time step can depend on the entire history of the system. Here we restrict attention to *stationary* policies, which specify actions entirely on the basis of the current state of the system. A stationary policy can be represented by a mapping, denoted π, that assigns an action to each state. The action specified by policy π when the system is in state x is denoted $\pi(x)$. For the route-finding example, a policy specifies what direction to move from any location. In subsection 7.2, we consider policies that specify actions probabilistically based on the current state of the system.

Letting x_t denote the system state at time step t, if the agent is using policy π then the action it takes at step t is $a_t = \pi(x_t)$. The system state changes according to

$$\begin{aligned}&\text{Prob}\{x_{t+1} = y \mid x_0, a_0, x_1, a_1, \ldots, x_t = x, a_t = a\}\\&\quad = \text{Prob}\{x_{t+1} = y \mid x_t = x, a_t = a\} = P_{xy}(a).\end{aligned}$$

Letting r_{t+1} denote the payoff received by the agent at time step $t + 1$, we have

$$E[r_{t+1} \mid x_t, a_t] = R(x_t, a_t), \tag{1}$$

where E indicates expected value. Figure 2 illustrates this formulation by showing when the relevant quantities are available to the agent, how they change over time, and on what information they depend. Because we assume the policy π is stationary, the sequence of states forms a Markov chain with transition probabilities $P_{xy} = P_{xy}(\pi(x))$. For this reason, decision tasks of this kind are called *Markov decision tasks*.

For the route-finding example, the state set, X, consists of the 96 different locations, one of which is the goal; the action set, A, is $\{N, S, E, W\}$; $R(x, a) = -1$ for all states x and actions a, except for the goal state, g, for which $R(g, a) = 0$ for all actions a. Because the system in this example is

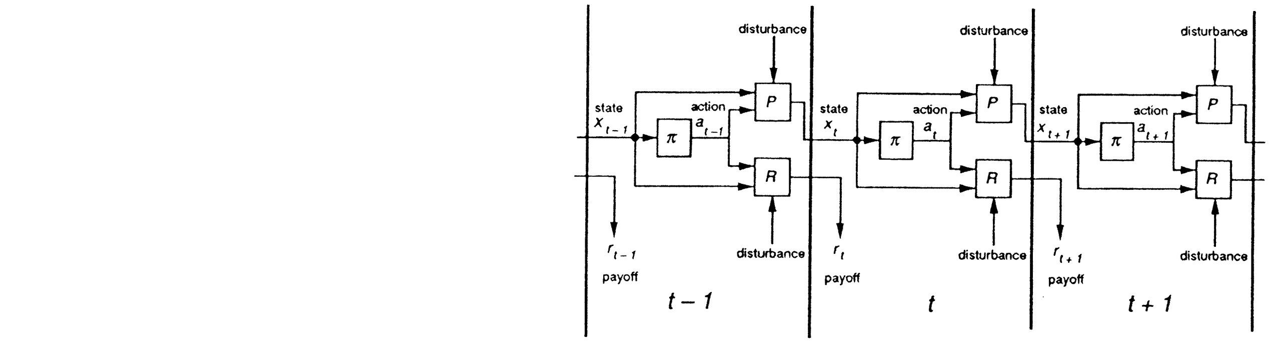

Figure 2
How the state, the action, and the payoffs change over time in a sequential decision task. The computation of the action, the state, and the payoff are shown for three successive time steps. The squares labeled π, P, and R respectively represent the decision policy, the state-transition computation, and the payoff computation. The stochastic aspects of the system are illustrated as disturbances influencing the latter two computations. In "space-time" diagrams of this type, several of which appear below, the quantities between two bold vertical lines are labeled with the same time subscript (exceptions to this which occur below are explained in the text). The repetition of a rectangle representing a system component indicates the same component viewed at different times.

deterministic, $R(x, a)$ is the actual payoff received by the agent for performing action a in location x, and the transition probabilities equal either 1 or 0: $P_{xy}(a) = 1$ if action a moves the agent from location x to location y, and is zero otherwise.

4.2 *Return and Evaluation Function*

It is now possible to define a policy's return, which for our purposes will be the infinite-horizon discounted return. Suppose an agent begins interacting with a system at time step $t = 0$ and is able to continue for an infinite number of time steps. Using discount factor γ, the measure of the total amount of payoff that the agent will receive over this infinite time period is

$$r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots + \gamma^n r_{n+1} + \cdots. \tag{2}$$

When $0 < \gamma < 1$, the powers of γ weighting the payoffs form a decreasing sequence so that later payoffs are weighted less than earlier ones, with payoffs in the far distant future contributing negligibly to the return. If $\gamma = 0$, the sum in expression 2 is simply the immediate payoff, r_1, due to the first action, a_0; if $\gamma = 1$, it is the sum of all the payoffs received and it generally is not finite.[7] The discount factor adjusts the degree to which the long-term consequences of actions must be accounted for in a decision task and influences the rate at which learning occurs. Although we do not discuss the problem of determining what discount factor is appropriate for a particular application, it is clear that a value close to 1 is appropriate when there is a high degree of certainty about how the system will evolve over time. One would want to sacrifice short-term return only for long-term return that is highly certain. We also do not discuss procedures for dynamically adjusting γ with experience, even though this would be useful in many applications.

In addition to depending on the system underlying the decision task, the initial system state, and the decision policy used by the agent, the sum given by expression 2 depends on unknown or random factors influencing the state transitions and the payoffs. The expected value of this sum over all possible decision tasks starting with initial state x when the agent uses policy π is

$$E_\pi\left[\sum_{t=0}^{\infty} \gamma^k r_{t+1} \,|\, x_0 = x\right], \tag{3}$$

where E_π indicates that the expected value is taken under the assumption that policy π is used to select actions. We can think of this quantity as the value of a function, denoted V^π, assigning an expected return to each system state x:

$$V^\pi(x) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \,|\, x_0 = x\right]. \tag{4}$$

This function is the *evaluation function* for policy π. For each state x, $V^\pi(x)$ is the expected return over the infinite number of time steps beginning at $t = 0$ under the conditions that the system begins in state x and the agent uses policy π, where it is understood that the discount factor, γ, has some specified value. We call $V^\pi(x)$ the *evaluation* of state x.

Because a system state has the property that the future behavior of the system does not depend on how or when the system arrives at a state, the evaluation of a state is in fact the expected return over an infinite number of time steps beginning *whenever* the system enters that state under the condition that the agent continues to use policy π thereafter. The evaluation of a state is a *prediction* of the return that will accrue throughout the future whenever this state is encountered. If one can determine the evaluation of a state merely from observing the system when it is in that state, then this prediction is effectively available *immediately* when the system enters that state, even though the prediction contains information about the entire future.

For the route-finding example, the evaluation of location x for policy π depends on the expected number of time steps the agent takes to reach the goal from location x using policy π. If π always brings the agent to the goal in, say, n time steps from a location x, then

$$V^{\pi}(x) = -1 - \gamma - \gamma^2 - \cdots - \gamma^{n-1}.$$

The series has n terms because the payoff is 0 for all time steps after the goal is reached. When $\gamma = 1$, this series sums to $-n$, and as γ decreases, the sum becomes less negative, approaching -1 as γ approaches 0. The evaluation of the goal, $V^{\pi}(g)$, is 0 for any policy. In this task, because a location's evaluation is directly related (depending on γ) to the negative of the number of time steps to the goal, the larger a location's evaluation for policy π, the fewer time steps are required to move from it to the goal using π. If π does not generate a path from x to the goal in any finite number of steps, which can happen if π produces a looped path, then

$$V^{\pi}(x) = -\sum_{t=0}^{\infty} \gamma^t,$$

which converges to

$$-1/(1-\gamma) \leq -1$$

if $0 \leq \gamma < 1$.

4.3 *Optimality*

Let π and π' be any two policies. Policy π' is an improvement over policy π if the expected return for π' is no smaller than that for π for any state, and is larger for at least one state. More precisely, π' is an improvement over π if $V^{\pi'}(x) \geq V^{\pi}(x)$ for all states x, with strict inequality holding for at least one state. A policy is an *optimal policy*, which we denote π^*, if no policy is an improvement over it. Because the optimality of policies depends on the discount factor, technically we should refer to *γ-optimal policies*. As γ changes, different policies become optimal because a policy best for short-term versions of a task will generally not be best for long-term versions. However, for simplicity, we omit reference to γ if no confusion is likely to result, and we assume that whenever policies are compared they are compared according to expected returns defined for the same γ. In the route-finding example, recalling that a location's evaluation is directly related to the negative of the number of time steps to the goal, an optimal policy takes the agent from any location to the goal in the fewest possible time steps.

There can be many optimal policies for a given decision task. For example, in the route-finding illustration, there are several different ways to move from most locations to the goal in the minimum number of time steps. However, the evaluation functions for any two optimal policies must be identical. In the route-finding example, this corresponds to the simple observation that all the minimum-time paths from any location to the goal must be traversable in the same number of time steps. We can therefore define the unique *optimal evaluation function* (which we denote V^*) for a given decision task as the evaluation function for any optimal policy. For any optimal policy π^*,

$$V^{\pi^*}(x) = V^*(x)$$

for all states x. An agent using an optimal policy will maximize the expected return from any system state. The object of a sequential decision task is to find one of the optimal policies.

5 *Stochastic Dynamic Programming*

The definition of the evaluation function for a given policy given by equation 4 does not immediately suggest how the values of this function can be computed. For policy π, the evaluation of a state depends on *all possible* state sequences that can occur when policy π is used, and on the probabilities that these sequences occur. Even if one knows all the transition probabilities and the expected values of all payoffs, it is not immediately obvious how one can compute the evaluations of states. Similarly, it is not obvious how to determine the optimal evaluation function or an optimal policy. Stochastic dynamic programming provides methods for computing the evaluations of states, or for approximating their evaluations as closely as desired, as well as for approximating the optimal evaluation function and an optimal policy, under the assumption that one has an accurate model of the decision task. Having an accurate model of the decision task means knowing the payoff expectations, $R(x, a)$, and state-transition probabilities, $P_{xy}(a)$, for all states x and y and all actions a.

We first describe a dynamic programming method for finding the evaluations of states for a given policy, π, under the assumption that we have an accurate model of the decision task.[8] Then we describe dynamic programming methods for finding the optimal evaluation function and an optimal policy, again assuming an accurate model of the decision task. Consequently, in this section we discuss the *computation* of these various functions given that we have all the relevant knowledge about the decision task. These computational methods provide the background for understanding how these various functions can be *learned* without assuming this kind of knowledge using the methods presented in subsections 7.1 and 7.2.

5.1 *Computing State Evaluations for a Given Policy*

The evaluation of state x for policy π, $V^\pi(x)$, gives the expected return if policy π is used throughout an infinite-horizon decision task beginning in state x. One method for determining a state's evaluation is to approximate it as the evaluation for a finite-horizon version of the task with a horizon chosen large enough to produce an approximation of sufficient accuracy. As the horizon of this finite-horizon task increases, the approximation approaches the state's true evaluation. This is a simple method of successive approximations, although it can be computationally costly if there is a large number of states.

If the agent interacts with the system by using policy π for some finite number of time steps, n, then we can consider the expected return for this finite-horizon task. This n-step expected return for state x, denoted $V_n^\pi(x)$, is the expected value of the sum of the first n terms of expression 2. We call this the *n-step evaluation* of state x. The successive approximations method begins with $n = 1$. It is easy to find the one-step evaluation of state x, $V_1^\pi(x)$. This is the expected value of the payoff the agent will receive by using policy π for one step starting in state x. According to the framework described in section 4, we know that the action the agent will make is determined by applying policy π to the state x. This action is $a = \pi(x)$. We also know from equation 1 that the expected value of the payoff as a result of performing action a from state x is $R(x, a)$. Then we have

$$V_1^\pi(x) = R(x, a) = R(x, \pi(x)).$$

Applying this equation to all system states completely determines the one-step evaluation function, V_1^π, for policy π.

Suppose this has been accomplished so that V_1^π is known, and now suppose that the agent faces the two-step task starting in state x. After the agent performs action $a = \pi(x)$, it receives payoff with expected value $R(x, a)$, and the next system state is y with probability $P_{xy}(a)$. If we call the actual next state y, then the agent faces the one-step task from state y, and it already knows that the expected return for using policy π for one step beginning in state y is the one-step evaluation of y, $V_1^\pi(y)$. Thus, if the next system state is y, the two-step evaluation of x (i.e., the two-step expected return for state x) will be the expected value of the immediate payoff, $R(x, a)$, plus the one-step evaluation, discounted by γ, of y, which is $\gamma V_1^\pi(y)$. However, in general, the transition to a next state, y, occurs with probability $P_{xy}(a)$. Thus, instead of using the one-step evaluation of the actual next state, one must use the expectation over all possible next states of the one-step evaluations of these states, which is

$$\sum_y P_{xy}(a) V_1^\pi(y).$$

We therefore have

$$V_2^\pi(x) = R(x, a) + \gamma \sum_{y \in X} P_{xy}(a) V_1^\pi(y) \tag{5}$$

for all states x, where $a = \pi(x)$. In a similar manner we can determine V_3^π from V_2^π, V_4^π from V_3^π, and so on. The general rule for determining V_n^π in terms of V_{n-1}^π is a basic equation of stochastic dynamic programming:

$$V_n^\pi(x) = R(x, \pi(x)) + \gamma \sum_{y \in X} P_{xy}(\pi(x)) V_{n-1}^\pi(y) \tag{6}$$

for all system states x.

This iterative process for computing the evaluations of system states makes it clear why dynamic programming methods are said to proceed from a tasks's end to its beginning. Computing V_1^π can be viewed as considering the decision task when one step remains before the horizon is reached; computing V_2^π can be viewed as considering the decision task when two steps remain before the horizon is reached; and so on. For an infinite-horizon task, which is our major concern here, it turns out that the desired evaluation function is the unique limiting function generated by continued successive application of equation 6 (provided $\gamma < 1$) and, further, this evaluation function is the unique function, V^π, satisfying the following equation for each state x:

$$V^\pi(x) = R(x, \pi(x)) + \gamma \sum_{y \in X} P_{xy}(\pi(x)) V^\pi(y). \tag{7}$$

In fact, the function V^π satisfying this equation is the unique limiting function generated by successive application of equation 6 beginning with an *arbitrary* real-valued function of system states, and not just with V_1^π. Because of the discount factor, the influence of the initial function on V_n^π decreases with repeated applications of equation 6. However, choosing the initial function to be a good approximation of the desired evaluation function can considerably reduce the number of iterations required to achieve a given degree of accuracy, as can reducing γ.[9]

The successive application of equation 6 can be illustrated with the route-finding example. First, we know from the description of the task in section 2 that $V_1^\pi(x) = -1$ for all locations x except the goal, where it is 0. Now apply equation 6 with $n = 2$ to determine $V_2^\pi(x)$ for each location x. To do this , note that, because the task is deterministic, for any location x, $P_{xy}(\pi(x)) = 1$ only for the one location y (say $y = x'$) that is always reached from x by performing action $\pi(x)$. Therefore,

$$\sum_y P_{xy}(\pi(x)) V_1^\pi(y) = V_1^\pi(x').$$

So we need only consider in turn each location, x, and its immediate sequel, x', determined by the action $\pi(x)$. Applying equation 6 with $n = 2$, if neither x nor x' is the goal, g, we have

$$V_2^\pi(x) = -1 + \gamma \times (-1) = -1 - \gamma;$$

if $x \neq g$ but $x' = g$, then

$$V_2^\pi(x) = -1 + \gamma \times 0 = -1;$$

and if $x = g$ then it must be that $x' = g$ and

$$V_2^\pi(x) = 0 + \gamma \times 0 = 0.$$

We can now apply equation 6 with $n = 3$ to each location to determine V_3^π. Again consider in turn each location, x, and its immediate sequel, x', determined by the action $\pi(x)$. Applying equation 6 with $n = 3$, if neither x nor x' is the goal, we have

$$V_3^\pi(x) = -1 + \gamma \times (-1 - \gamma) = -1 - \gamma - \gamma^2;$$

if $x \neq g$ but $x' = g$, then

$$V_3^\pi(x) = -1 + \gamma \times 0 = -1;$$

and if $x = g$ then it must be that $x' = g$ and

$$V_3^\pi(x) = 0 + \gamma \times 0 = 0.$$

Continuing the application of equation 6 eventually produces the location (or state) evaluations given in subsection 4.2.

5.2 *Computing an Optimal Policy*

We now discuss two methods for finding a policy that maximizes expected return under the assumption that a complete and accurate model of the decision task is available. Consequently, we assume that $R(x, a)$ and $P_{xy}(a)$ are known for all actions a and for all states x and y. Recall that an optimal policy is a policy that cannot be improved upon in terms of expected return—that is, in terms of the expected total amount of payoff (suitably discounted and summed) it produces over the course of the decision task. The optimal evaluation function is the evaluation function corresponding to an optimal policy. Although there can be many optimal policies, there is only one optimal evaluation function.

If we know the optimal evaluation function, V^*, it is relatively easy to form an optimal policy. The optimal action when the system is in state x is the action, a, that maximizes

$$R(x, a) + \gamma \sum_{y \in X} P_{xy}(a) V^*(y). \tag{8}$$

This is the sum of the expected value of the immediate payoff, $R(x, a)$, and the expected value, taken over the states that can be reached in one step, of the infinite-horizon evaluations of possible next states, discounted by γ, under the assumption that an optimal policy is used throughout the future. Selecting an action that maximizes expression 8 effectively answers the question "What action is best to perform *now*, on the assumption that an optimal policy will be followed in selecting all the future actions?" If we know V^*, it is relatively easy to select an action that maximizes the quantity given by expression 8. We need only examine the consequences of applying action sequences of length 1, because the consequences of all the longer action sequences are summarized in the known optimal evaluation function, V^*. Because we are assuming a complete model of the decision task and knowledge of V^*, all the quantities involved in expression 8 are known. Hence, we can find an optimal action for state x by calculating expression 8 for x and each action, a, and selecting the action that produces the largest result. Doing this for all system states x produces an optimal policy. For a discount factor close to 0, the optimal action for each state is most strongly influenced by the expected value of the immediate payoff, $R(x, a)$; for a discount factor equal to 1, on the other hand, the estimated total amount of payoff that will be accumulated throughout the future, given by

$$\sum_{y \in X} P_{xy}(a) V^*(y),$$

is given a weight nearly equal to that of immediate payoff. Therefore the discount factor can be regarded as determining how the agent trades off an estimate of future return with an estimate—which is usually more accurate—of immediate payoff.

According to the above analysis, for an infinite-horizon task, finding an optimal policy can be reduced to the task of finding the optimal evaluation function. The optimal evaluation function can be approximated by a method of successive approximations similar to the one described above for approximating the evaluation function for a given policy (equation 6). Here, however, it is necessary to determine the *maximum* expected return at each step. Let V_n^* denote the optimal evaluation function for the first n steps of the infinite-horizon decision task; that is, $V_n^*(x)$ is the maximum expected return if the decision task is terminated n steps after starting in state x. For $n = 1$, the maximum expected return is just the maximum of the expected values for the immediate payoff:

$$V_1^*(x) = \max_{a \in A} R(x, a). \tag{9}$$

Suppose, then, that we know V_1^*, and the agent now faces the two-step task starting in state x. If the agent performs an action a (whatever it may be), it will receive a payoff with expected value $R(x, a)$, and the next state of the system will be y with probability $P_{xy}(a)$. If we call the actual next state y, then the agent faces the one-step task from state y, and we already know that the maximum expected return for this one-step task is $V_1^*(y)$. Thus, if the next system state is y, the maximum expected return for two steps starting in state x is obtained by performing the action that maximizes the sum of the expected value of the immediate payoff, $R(x, a)$, and the one-step optimal evaluation, discounted by the factor γ, of y, which is $\gamma V_1^*(y)$. However, because the transition to the next state, y, occurs with probability $P_{xy}(a)$, instead of using the one-step optimal evaluation of the actual next state, one must use the expectation over all possible next states of the one-step optimal evaluations of these states, which is

$$\sum_y P_{xy}(a) V_1^*(y).$$

We therefore have

$$V_2^*(x) = \max_{a \in A} \left[R(x, a) + \gamma \sum_{y \in X} P_{xy}(a) V_1^*(y) \right] \tag{10}$$

for all states x. In a similar manner we can determine V_3^* from V_2^*, V_4^* from V_3^*, and so on. The rule for determining a V_n^* in terms of V_{n-1}^* is

$$V_n^*(x) = \max_{a \in A} \left[R(x, a) + \gamma \sum_{y \in X} P_{xy}(a) V_{n-1}^*(y) \right] \tag{11}$$

for all states x (cf. equation 6).

The limiting values obtained by the repeated application of equation 11 are the values of the desired optimal evaluation function, V^*. Moreover, this function is the unique function that satisfies the following equation, which is another basic equation of stochastic dynamic programming (cf. equation 7):

$$V^*(x) = \max_{a \in A} \left[R(x, a) + \gamma \sum_{y \in X} P_{xy}(a) V^*(y) \right] \tag{12}$$

for all states x. The repeated application of equation 11 is a method of successive approximations, called *value iteration*, for computing the optimal evaluation function. Here too, it is not necessary to begin the process with V_1^*; one can begin with an *arbitrary* real-valued function of the system state.[10] Choosing this function to be a good approximation of the optimal evaluation function can reduce the number of iterations required to achieve a given degree of accuracy.

In order to find an optimal policy, then, one first approximates the optimal evaluation function, V^*, by value iteration (repeated applications of equation 11) and then determines an optimal policy, π^*, by defining it to select an action that maximizes expression 8 for each state x. If more than one action maximizes equation 8, it does not matter which one is selected. If V^* is computed with sufficient accuracy, this process produces a policy (there may be others) that maximizes the expected return given by equation 4.

To make value iteration more concrete, consider how it applies to the route-finding task. Assuming that we have no better approximation for the optimal evaluation function, we begin by determining the one-step optimal evaluation function using equation 9. For the one-step task, no matter what action is chosen from any non-goal location, the one-step return is -1. Hence, $V_1^*(x) = -1$ for all locations $x \neq g$, and clearly $V_1^*(g) = 0$. Applying equation 11 with $n = 2$ to find the optimal two-step evaluation function can be visualized as follows. For each location x, determine the result of applying each of the four actions in terms of the sum of the immediate payoff and the discounted evaluation, γV_1^*, of the location that results from the action. The maximum of these four sums is $V_2^*(x)$. In this case, $V_2^*(x) = -1 - \gamma$ for all locations x except the goal, g, and those locations immediately adjacent to g: g retains an evaluation of 0, and locations adjacent to g retain the evaluation -1. In a similar manner, one can see that $V_3^*(x)$ is 0 for $x = g$; it is -1 for locations immediately adjacent to g, it is $-1 - \gamma$ for locations two steps away from g, and it is $-1 - \gamma - \gamma^2$ for all other locations. As this computation is repeated, the evaluations of locations stabilize from the goal outward until the evaluations of all locations are determined. In the route-finding task this happens in a finite number of applications of equation 11, because all optimal paths must reach the goal, after which payoff no longer accumulates. An optimal policy always moves the agent to the adjacent location that has the largest evaluation according to the optimal evaluation function. If two or more adjacent locations have this maximum evaluation, it does not matter which one is chosen.

We now describe a different method of successive approximations for constructing an optimal policy that is more closely related to the learning method we describe in section 7 than is value iteration. This method is called the *policy-improvement method*, or *policy iteration* (Howard 1960; Ross 1983), because it generates a sequence of policies, each of which is an improvement over its predecessor.

Although one needs to know the optimal evaluation function in order to define an optimal policy, the policy-improvement method is based on the observation that all one need know in order to *improve* a given policy is the evaluation function for that given policy. Suppose that the current policy is π and that we have constructed the evaluation function, V^π, for this policy. Further, suppose that π' is some proposed policy, and that we wish to see if π' is an improvement over π. Recall that a policy π' is an improvement over π if $V^{\pi'}(x) \geq V^\pi(x)$ for all system states x, with strict inequality holding for at least one state, where $V^{\pi'}$ and V^π are the evaluation functions, respectively, for policies π' and π. Thus, one way to determine if π' is an improvement over π would be to compute $V^{\pi'}(x)$ and to compare it with $V^\pi(x)$ for every state x. However, there is a much simpler way to do this that does not require constructing $V^{\pi'}$. Consider the expected return, starting in state x, that would result from using policy π' for one step and then using policy π thereafter. The expected return for this composite policy is

$$R(x, \pi'(x)) + \gamma \sum_{y \in X} P_{xy}(\pi(x)) V^{\pi}(y). \tag{13}$$

If this quantity is greater than or equal to $V^{\pi}(x)$ for all states x, and strictly greater for at least one state, then it can be shown that π' is an improvement over π.

Instead of using expression 13 just to check if a proposed policy is an improvement over the current policy, one can use it to define a new policy that improves over the current one. Define a new policy to be the one that for each state, x, specifies the action, $\pi'(x)$, that maximizes expression 13. Thus, the new policy is an optimal policy with respect to the evaluation function of the current policy, V^{π}. It can be shown that either this new policy is an improvement over the current policy or else both the current policy and the new policy are optimal (Ross 1983). Because we are assuming that we know the evaluation function for the current policy, determining the new policy by maximizing expression 13 requires looking only one step ahead from every state.

The policy-improvement method is based on the foregoing ideas and works as follows. First select an *arbitrary* initial policy; call it π_0. Then compute the evaluation function, V^{π_0}, for π_0 by solving the system of equations defined by equation 7 by the repeated application of equation 6 or some other means. Given V^{π_0}, determine a policy which is optimal with respect to it by defining that policy to select for each state, x, an action, a, that maximizes

$$R(x, a) + \gamma \sum_{y \in X} P_{xy}(a) V^{\pi_0}(y).$$

Call the resulting policy π_1. Either π_1 is an improvement over π_0 or else they are both optimal. If they are not optimal, the next step is to compute the evaluation function, V^{π_1}, for policy π_1, and then define a policy that is optimal with respect to it. Call this policy π_2. It will be an improvement over π_1, or else both π_1 and π_2 are optimal. The method continues in this way; policies are formed, evaluation functions corresponding to them are computed, and then new, improved policies are formed. If the state set, X, of the system underlying the decision task is finite, this method will produce an optimal policy after some finite number of iterations.

This completes our exposition of the basic concepts and computational methods of stochastic dynamic programming, all of which are well known to control and decision theorists. In addition to requiring a complete and accurate model of the decision task in the form of knowledge of all the state-transition and payoff probabilities, the procedures described above for approximating the evaluation function for a given policy, the optimal evaluation function, and an optimal policy can require considerable amounts of computation. The concepts and methods of stochastic dynamic programming permit clear statements of some of the most important things we would like an agent to learn from its experiences interacting with a dynamical system, but they do not provide more than hints about how to devise learning methods that can operate in real time and that do not require a complete model of the decision task. For this we have to turn to another body of theory, also well known within mathematical and engineering disciplines, that concerns techniques for estimating parameter values. We shall see in section 7 that the temporal-difference procedure is a method for estimating the parameters specifying the evaluation function for a given policy in the absence of a model of the decision task. We shall also describe how the TD procedure can be incorporated into real-time procedures for improving decision policies.

6 Parameter Estimation

The TD procedure is a method for estimating an evaluation function by means of *parameter estimation*. Methods for parameter estimation are central to the fields of pattern classification (Duda and Hart 1973; Sklansky and Wassel 1981), adaptive signal processing (Widrow and Stearns 1985), and adaptive control (Goodwin and Sin 1984), as well as to the field of connectionist modeling (Anderson and Rosenfeld 1988; Hinton and Anderson 1981; McClelland and Rumelhart 1986; Rumelhart and McClelland 1986). For example, one way to construct a useful pattern classifier is to assume that the classification rule, or decision rule, is a member of a specific class of rules, where each rule in the class is specified by selecting values for a set of parameters. The problem of constructing a useful decision rule is therefore reduced to the problem of finding parameter values that specify a useful decision rule. The latter problem is usually formulated as one of estimating the parameter values that are optimal according to some predefined measure of decision-rule performance.

In signal processing and control, one is often seeking mathematical formulas capable of producing numerical values that match values measured from some physical system. One may wish, for example, to have a formula giving a patient's blood pressure as a function of the amount of a drug administered (Widrow and Stearns 1985). Such a formula constitutes a *model* of a functional relationship implemented by the physical system, and it can be used for a variety of prediction and control purposes. We call this functional relationship the *modeled function*.[11] Given an encoding of some input information (e.g., the amount of a drug administered), the model output should match the output of the modeled function for that same input (e.g., the patient's resulting blood pressure).

Among the most important elements in formulating a parameter-estimation task is the selection of a method for representing the physical aspects of the problem as signals that act as input and output of the decision rule or model. What aspects of objects, inputs, states, etc. should be measured to provide data for decision making or modeling? If basic physical variables (e.g., blood pressure) are obvious candidates, should they be represented simply as real-valued measurements, or should they be coded in some other way? Similarly, how is the output of the decision rule or model to be interpreted? There is little useful theory to guide these aspects of formulating a parameter-estimation task.

Another aspect of formulating parameter-estimation tasks for which there is little theory is the selection of the parameterized class of decision rules or models, a choice that includes deciding how many parameters to use and how their values define the rule or model. Generally, one must make a compromise between the complexity of a rule or a model and its adequacy for the application. In this chapter, we restrict our attention to one of the simplest classes of models and use a very simple representation for the route-finding example, but we indicate how more complex choices can be accommodated.

Other important elements in formulating a parameter-estimation task are the choice of a performance measure and the type of estimation algorithm most appropriate.[12] Estimating parameters depends on comparing the performance of the different decision rules or models specified by different parameter values. The best measure of performance might evaluate the overall, final performance of the system that makes use of the decision rule or model. However, in practice one has to devise simpler criteria that are closely correlated with overall performance and yet can be measured easily and quickly. It is common to measure performance by the average of the squared error between the output of the decision rule or model and some target values.

Parameter-estimation methods are classified as either *off-line* or *on-line* methods. Off-line methods operate when all the information relevant to measuring the performance of a decision rule or model is available at one time. This information is collected before the analysis that determines suitable parameter values is performed. On-line methods, on the other hand, process information as it becomes available over time. Because on-line methods of parameter estimation incrementally adjust parameter values while a decision rule or model is in use, they are often regarded as learning methods, and the information they use to adjust parameter values is called *training information*. For applying on-line methods to classification tasks, training information takes the form of a collection of training patterns together with their correct classifications, which are assumed to be supplied by a "teacher." For modeling tasks, training information consists of examples of how the modeled function behaves for a variety of inputs. In this case, the role of the teacher is played by the modeled function.

We shall be concerned exclusively with on-line methods of parameter estimation in this chapter. Nearly all the learning methods for connectionist networks are on-line parameter-estimation methods in which the form of the decision rule or model is determined by the structure of the network (which includes the types of units, how many of them there are, and how they are interconnected). The parameter values correspond to the synaptic weights. Similarly, many models of animal learning, such as the Rescorla-Wagner model and the TD model discussed by Sutton and Barto in this volume, can be regarded as on-line parameter-estimation methods in which the parameter values correspond to the associative strengths of the components of stimuli.

Viewing a learning process as on-line parameter estimation does not imply a *tabula rasa* view of that process. Prior knowledge is incorporated in the task representation, in the class of decision rules or models, in the initial parameter values, and in the constraints enforced among parameter values during learning. Using increased levels of prior knowledge in selecting these aspects of a task can both accelerate the learning process and improve the utility of the solutions achieved. The roles played by prior knowledge in parameter estimation may correspond to the roles played in animal learning by genetically specified mechanisms and behavior.

6.1 *Feature Vectors, Decision Rules, and Models*

The framework in which parameter-estimation techniques are applied is essentially the same for classification tasks and tasks requiring the modeling of functional relationships. For a classification task, let x denote any one of the objects we wish to classify. For a function-modeling task, let x denote any of the collections of data that can be provided as input to the modeled function. In what follows we will call all such collections of data *patterns*, even though this is not the usual term in many function-modeling tasks. Suppose that a set of measurements can be made of pattern x to produce a description in the form of a list of attribute or feature values. If there are n measurements, let ϕ_i $(i = 1, \ldots, n)$ denote the "measuring instruments." If we assume that the measurements are real numbers, as is usual, each ϕ_i is a function from the set of possible patterns to the real numbers.[13] With $\phi_i(x)$ denoting the value of the ith measurement of x,

$$\phi(x) = (\phi_1(x), \ldots, \phi_m(x))^{\mathrm{T}}$$

is the *feature-vector* description of x. (We assume here that vectors are column vectors, so the superscript T indicates that the row form shown is the transpose of the default column form.) We can think of $\phi(x)$ as the parameter-estimation system's internal representation of pattern x.

For a classification task, suppose that each decision rule among the rules to be considered is defined by a vector of $n + 1$ parameters,

$$v = (v_1, \ldots, v_{n+1})^{\mathrm{T}},$$

where each v_i ($i = 1, \ldots, n + 1$) is a real number. For example, one might assume that the decision is made by weighting each feature value $\phi_i(x)$ by v_i and comparing the weighted sum to a threshold given by $-v_{n+1}$ (the minus sign is present for a technical reason to be made clear below). The object is in class 1 if the weighted sum exceeds 0; it is in class 2 if the weighted sum is less than 0. Each decision rule of this type has the form

$$x \text{ is } \begin{cases} \text{class 1} & \text{if } v_1\phi_1(x) + \cdots + v_n\phi_n(x) > -v_{n+1} \\ \text{class 2} & \text{if } v_1\phi_1(x) + \cdots + v_n\phi_n(x) < -v_{n+1} \end{cases} \tag{14}$$

for some parameter vector v. This is the linear threshold rule often used in connectionist networks, where it is implemented by an abstraction of a neuron. The parameter values (synaptic weights) specifying the best approximation to the desired decision rule are initially unknown and are estimated through the use of a parameter-estimation method.

In a modeling task, one of the simplest classes of models consists of linear models defined by weighted sums of n feature values, plus the parameter v_{n+1}:

$$v_1\phi_1(x) + \cdots + v_n\phi_n(x) + v_{n+1}. \tag{15}$$

This sum is the output of the model specified by the parameter vector v for input x. Although the development of efficient parameter-estimation methods for nonlinear models is important for the utility of the methods discussed in this chapter, the issues we address here are very different, and for simplicity we assume the use of linear models. It should be clear, however, that the learning procedures we discuss can be extended to nonlinear cases (Anderson 1986, 1987).[14]

Two conventions are usually followed when decision rules or models in the form of expressions 14 and 15 are used. First, to the list of n functions ϕ_i, one adds an $(n + 1)$st, which is always 1; that is, for any pattern x, $\phi_{n+1}(x) = 1$. This makes it possible to write the decision rule given by expression 14 as follows:

$$x \text{ is } \begin{cases} \text{class 1} & \text{if } v^{\mathrm{T}}\phi(x) > 0 \\ \text{class 2} & \text{if } v^{\mathrm{T}}\phi(x) < 0, \end{cases} \tag{16}$$

where $v^{\mathrm{T}}\phi(x) = v_1\phi_1(x) + \cdots + v_{n+1}\phi_{n+1}(x)$ is the inner product, or scalar product, of the vectors v and $\phi(x)$. Similarly, the parameterized linear model given by expression 15 can be written as

$$v^{\mathrm{T}}\phi(x). \tag{17}$$

A second useful convention is to refer to the *time* at which pattern x appears for classification, or as input to the modeled function, instead of referring directly to x. Let ϕ_t and v_t respectively denote the feature vector and the parameter vector at time t. The set of values that the time variable, t, can take depends on whether one considers continuous or discrete-time parameter estimation. We restrict attention to discrete-time methods, so t is a non-negative integer.

6.2 *Parameter-Estimation Methods*

The methods for parameter estimation that we consider are error-correction methods because they adjust parameters on the basis of errors between actual and desired outputs of the decision rule or model. Error-correction methods operate by receiving sequences of training pairs, each pair consisting of a feature vector and the output desired of the decision rule or model when it is presented with that feature vector as input. The collection of all training pairs that are available constitutes the training data for the parameter-estimation task. This paradigm is known as "supervised learning" or "learning with a teacher." In the classification task a "teacher" provides the correct object classifications, whereas in the modeling task the modeled function acts as a teacher. In what follows, we discuss parameter estimation in the context of modeling a functional relationship instead of forming a decision rule, because the TD procedure can be best understood in this context. Parameter estimation applied to forming a decision rule becomes relevant in subsection 7.2, where we discuss the estimation of optimal policies.

An on-line estimation method operates as follows (see figure 3). At time step t, there is a current estimate, v_t, for the unknown parameter vector that specifies the best model within the class of models under consideration. There is also available a feature vector, ϕ_t, describing the current input, x_t, to the modeled function (so that ϕ_t is a simpler way to write $\phi(x_t)$). The feature vector is used as input to the model specified by the parameter estimate v_t, and the resulting output of this model is an estimate for the output of the modeled function given input x_t. For the linear model given by expression 17, this estimate is $v_t^{\mathrm{T}}\phi_t$. This estimate is then compared with the actual output of the modeled function, which we assume becomes available at the next time step, $t + 1$. This actual output, denoted y_{t+1}, can be thought of as the "target" value for the estimated output $v_t^{\mathrm{T}}\phi_t$, and the estimation error is the scalar

$$\varepsilon_{t+1} = y_{t+1} - v_t^{\mathrm{T}}\phi_t. \tag{18}$$

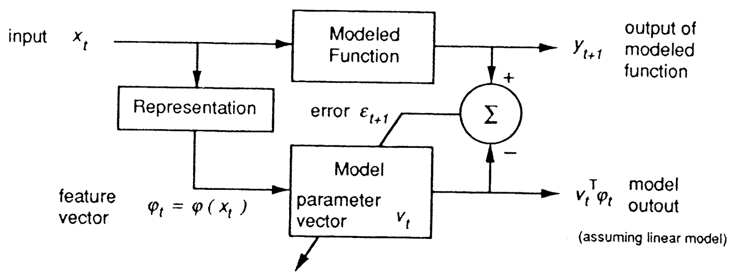

Figure 3
On-line parameter estimation for modeling a functional relationship. The error, ε_{t+1}, resulting from comparing the output of the modeled function, y_{t+1}, with the model's estimate of this output, $v_t^T \phi_t$, is used to update the parameter vector, v_t. The input to the model is the feature vector, $\phi_t = \phi(x_t)$, determined from the input to the modeled function, x_t, by the box labeled "Representation." The output of the modeled function is assumed to be a linear function of the feature vectors. The circle labeled "$\sum$" in this and following figures is a summation unit, with the signs of the summed quantities labeling each input. Error updates parameter vector.

In the most widely used methods, a new parameter vector, v_{t+1}, is determined from the current vector, v_t, as follows:

$$v_{t+1} = v_t + M_t \varepsilon_{t+1} \phi_t, \qquad (19)$$

where M_t is sometimes called the algorithm "gain" at step t. A number of on-line methods for parameter estimation follow the form of equation 19 but differ in what M_t is and in how it is computed. In some methods, M_t is an $(n + 1) \times (n + 1)$ matrix computed in an iterative manner on the basis of the training data (Goodwin and Sin 1984). Here we restrict our attention to the simplest case, in which M_t is replaced by a positive constant, β,[15] which, together with the magnitude of the error, determines the magnitude of changes in parameter values from one step to the next. In this case we can write the following parameter-up date rule:

$$v_{t+1} = v_t + \beta \varepsilon_{t+1} \phi_t. \qquad (20)$$

Figure 4 is an elaboration of figure 3 showing how the various quantities involved in on-line parameter estimation change over time and showing on what quantities their successive values depend. It shows these quantities for two successive time steps. The box labeled "Update Parameters" implements a parameter-update rule such as the one given by equation 20. The inputs to this box correspond to the variables on the right-hand side of equation 20.

Because we express parameter-update rules in the vector notation of equation 20 throughout this chapter, it is important to understand how an equation written in this form specifies how the value of each parameter

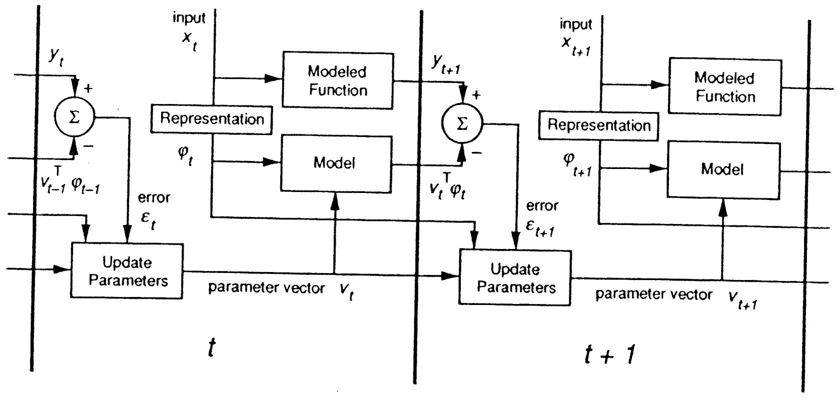

Figure 4
How the various quantities involved in on-line parameter estimation change over time. Values for two successive time steps are shown, and a linear model is assumed. The box (shown once for each time step) labeled "Update Parameters" implements a parameter-update rule such as the LMS rule.

making up the parameter vector is updated. In equation 20, β and ε_{t+1} are single numbers (scalars) and v_t and ϕ_t are vectors having the same number of components. Hence, this equation indicates that at time step t one adds $\beta\varepsilon_{t+1}$ times the feature vector ϕ_t to the parameter vector v_t to form the next parameter vector, v_{t+1}. This means that the value of the ith parameter at step $t + 1$ is its old value plus $\beta\varepsilon_{t+1}$ times the value of the ith feature at step t.

If the parameterized linear model given by expression 17 is assumed, then equation 20 can be expanded via equation 18 to yield

$$v_{t+1} = v_t + \beta(y_{t+1} - v_t^{\mathrm{T}}\phi_t)\phi_t. \tag{21}$$

This estimation method, presented by Widrow and Hoff (1960) and also known as the LMS rule, is essentially the same as the Rescorla-Wagner model of classical conditioning (Rescorla and Wagner 1972; Sutton and Barto 1981).[16]

Some on-line estimation procedures, such as the LMS rule, can be understood as gradient-descent procedures, which are methods for finding a local minimum of a surface by moving down the slope, or gradient, of the surface.[17] According to this view, there is a surface, defined over the space of parameter vectors, whose height at each parameter vector gives a measure of the error over all patterns of the model specified by that parameter vector. The task is to search for a parameter vector at which this surface is minimal. Updating estimates according to equation 19 tends to form new estimates that are steps down on this error surface. The gain matrix, M_t, which in the LMS rule (equation 21) is equal to the constant β, adjusts the direction and size of the step on the error surface. For the LMS rule, the error measure is the mean value of the squared error of the model over all patterns (hence its name).[18]

We briefly discuss how the LMS rule is derived in terms of gradient descent because the derivation requires a step similar to one we use in explaining the TD procedure. The objective is to adjust the parameter values of the linear model in order to minimize a measure of model error over all patterns. Letting $\varepsilon_{t+1} = y_{t+1} - v_t^{\mathrm{T}}\phi_t$, as above, one wants to minimize $E[\varepsilon_t^2]$, where E is the expectation over all patterns and where the error is squared to produce a quantity that is always positive. This quantity—the mean square error—depends on the parameter values. At each step of the estimation process, one would like to adjust the parameter values in such a way that the mean square error is reduced. Because the mathematical expression defining this error measure is known, it is possible to precompute the gradient of the error measure as a function of the parameter values. The result of this computation specifies that one has to add to the current parameter vector, v_t, a vector proportional to

$$E[\varepsilon_{t+1}\phi_t], \tag{22}$$

where, again E is the expectation over all possible patterns.

Unfortunately, in an on-line procedure one cannot determine at each time step the quantity given by expression 22, because it depends on how well the current parameter values work for all possible patterns. Therefore, the crucial step in deriving the LMS rule is to assume that $\varepsilon_{t+1}\phi_t$ for just the *current* feature vector, ϕ_t, is a good estimate of the quantity given by expression 22. This assumption yields the parameter-update equation of the LMS rule given above (equation 21). Because the expected value of $\varepsilon_{t+1}\phi_t$ over all patterns is the gradient of the error measure that is to be minimized (expression 22), $\varepsilon_{t+1}\phi_t$ is an unbiased estimate of this gradient. For the class of parameterized linear models defined by expression 17, the LMS rule can be shown to minimize the mean square error over all patterns under appropriate conditions, with certain caveats that need not concern us here (Widrow and Stearns 1985).

The theory of the LMS rule, and of the parameter-estimation methods, is very well-developed but is beyond the scope to the present chapter. Theoretical treatments of parameter estimation can be found in the following references: Duda and Hart 1973; Goodwin and Sin 1984; Ljung and Söderstrom 1983; Sklansky and Wassel 1981; Widrow and Stearns 1985. This theory specifies conditions under which a parameter-estimation method converges to a final estimate and what criterion of best fit the final estimate satisfies. To obtain some of these theoretical results, one must assume that the manner of presenting training data to the algorithm satisfies certain properties. For modeling tasks, for example, the output of the modeled function has to be observed for a sequence of inputs that are sufficiently varied to reveal the function's form. Theoretical results concerning parameter estimation for classification tasks similarly require a training regime that repeats the training pairs sufficiently often or uses as training data a sufficiently large sample of the set of possible patterns.

7 *Learning and Sequential Decision Tasks*

We are now in a position to combine the concepts from stochastic dynamic programming described in section 5 with those just described concerning parameter estimation. We address tasks that require *learning* evaluation functions and optimal decision policies in the absence of the knowledge necessary to apply the dynamic programming techniques presented in section 5. Although these dynamic programming techniques are therefore not applicable, the principles they involve can be applied to the design of learning methods. In subsection 7.1 we consider the problem of learning the evaluation function for a given decision policy. The TD procedure (equation 29) emerges from this analysis as an on-line method for this task. Although a method for learning an evaluation function for a given policy does not itself improve a policy in terms of expected return, the evaluation-function estimates that result from applying such a method have great utility for improving policies. An evaluation function provides an immediately accessible prediction of the long-term return a policy will achieve. This information can be used in adjusting policies so that decisions are not dominated by the short-term consequences of behavior. In subsection 7.2 we consider learning the optimal evaluation function and an optimal decision policy by a method that takes advantage of evaluation-function estimates produced by the TD procedure.

For all these learning tasks, we must assume that the agent has the opportunity to interact with the system underlying the task in a manner sufficient to produce good parameter estimates. How this requirement can be met generally depends on details of the system, and the issues that arise are beyond the scope of the present chapter. Here it suffices to assume that the system underlying the decision task is set to a randomly chosen state, either periodically or when certain conditions are met. For example, in the route-finding task we place the agent at a randomly chosen location and let it interact with the system (i.e., move around) while updating its parameter values for a given number of time steps or until it reaches the goal. This constitutes a trial. We run a sequence of trials until some performance criterion is satisfied or a time limit is reached.

7.1 *Learning an Evaluation Function*

Here we consider the problem of learning an evaluation function for a given policy in a sequential decision task. The learning system is given the opportunity to interact for a sequence of trials with the system underlying the decision task. A fixed policy, π, is used throughout the sequence of trials. We wish to form the evaluation function, V^π, for policy π without having prior knowledge of the quantities $R(x, a)$ and $P_{xy}(a)$, for system states x and y and action a. Recall that V^π assigns to each state, x, its evaluation, which is the expected infinite-horizon discounted return if policy π is used to select actions after the system enters state x (equation 4).

We formulate the problem of learning an evaluation function as a parameter-estimation task. A set of features for representing system states must be selected. Let $\phi(x)$ denote the feature vector representing state x. A parameterized class of models for the evaluation function is then selected, and the task is to adjust the parameter values in order to best approximate the true evaluation function according to a measure of how closely the model's evaluations match the true evaluations of states.[19] Here, for simplicity, we select the class of linear models given by expression 15 or 17 and consider models that assign to each state, x, the evaluation estimate

$$v_1\phi_1(x) + \cdots + v_n\phi_n(x) + v_{n+1} = v^{\mathrm{T}}\phi(x) \tag{23}$$

for parameter vector $v = (v_1, \ldots, v_{n+1})^{\mathrm{T}}$. Expression 23 is a linear model of the evaluation function V^π.

Because we are seeking an on-line estimation method, we refer to evaluation estimates, parameter vectors, and feature vectors by the times at which they occur. We let V_t denote the estimate at time step t for the evaluation function V^π. Letting v_t denote the parameter vector at time step t, the estimate at time step t for $V^\pi(x)$ is

$$V_t(x) = v_t^{\mathrm{T}}\phi(x). \tag{24}$$

If we were to follow the general form for parameter estimation described in subsection 6.2, given the current system state x_t, an error, ε_{t+1}, would be determined by comparing $V_t(x_t)$, the estimated evaluation of x_t, with $V^\pi(x_t)$, the actual evaluation of x_t for policy π:

$$\varepsilon_{t+1} = V^\pi(x_t) - V_t(x_t). \tag{25}$$

Unfortunately, because we do not know the true evaluation function V^π, we do not know $V^\pi(x_t)$. What quantity can be used as a target to define error useful for parameter estimation?

The true evaluation of state x_t is the expected infinite-horizon discounted return if policy π is used to select actions after the system enters state x_t. Therefore, although it is obviously not practical, we might consider *waiting forever* and using as a target the actual infinite-horizon discounted return that would accrue while policy π is used from time step t onward. Because the expected value of this actual return would be the true evaluation of x_t, this actual return would be an unbiased estimate of the true evaluation and could serve as a useful target. More practically, we could wait n time steps and use what Watkins (1989) calls the *n-step truncated return* as a target:

$$r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{n-1} r_{t+n}. \tag{26}$$

Depending on γ, one can choose an n large enough to make the n-step truncated return suitably approximate the actual return. As γ decreases toward zero, truncated returns more closely approximate actual returns.

However, it is possible to define a better target than the n-step truncated return by using the current parameter vector, v_t, to determine a "correction" to the n-step truncated return. That is, one can use as a target what Watkins (1989) calls the *corrected n-step truncated return*:

$$r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(x_{t+n}), \tag{27}$$

where $V_t(x_{t+n})$ is the estimated evaluation of state x_{t+n} using the parameter vector v_t. Because $V_t(x_{t+n})$ is an estimate of the expected return that would accrue while policy π is used from time step $t + n$ onward, $\gamma^{n-1} V_t(x_{t+n})$ is an estimate of the sum of the terms of the actual return (expression 26) that are not present in the n-step truncated return (expression 27). To see this, note that $\gamma^{n-1} V_t(x_{t+n})$ is an estimate of the expected value of

$$\gamma^n[r_{t+n+1} + \gamma r_{t+n+2} + \gamma^2 r_{t+n+3} + \cdots].$$

Multiplying through by γ^n, this equals

$$\gamma^n r_{t+n+1} + \gamma^{n+1} r_{t+n+2} + \cdots,$$

which is the part of the actual return not included in the n-step truncated return. Therefore, to the extent that the correction term $\gamma^{n-1} V_t(x_{t+n})$ is a good estimate for this truncated part of the actual return, the corrected n-step truncated return will be a good estimate of the expected infinite-horizon discounted return.

But why would one expect this correction term, which is based on the estimated evaluation function V_t, to be a good estimate for the truncated part of the actual return when it is V_t itself that we are attempting to estimate? The key observation (Watkins 1989) is that if the parameterized class of models chosen for modeling the evaluation function is adequate, then the corrected n-step truncated return starting in state x_t is always a better estimate of the true evaluation $V^\pi(x_t)$ than is $V_t(x_t)$. Intuitively, this is true because the corrected n-step truncated return is based on more data than is $V_t(x_t)$—namely, the payoffs r_{t+k}, where $k = 1, \ldots, n$. Hence, even if V_t is a poor estimate for the evaluation function, the estimated evaluation

$V_t(x_t)$ can be *improved* by adjusting the parameter vector to make $V_t(x_t)$ more closely match the corrected n-step truncated return for state x_t.

On the basis of the argument outlined above, one can approximate the error given by equation 25 by using the corrected n-step truncated return as a target. In particular, one can use the corrected one-step truncated return as a target to define what we call the TD error:

$$\tilde{\varepsilon}_{t+1} = [r_{t+1} + \gamma V_t(x_{t+1})] - V_t(x_t). \tag{28}$$

Substituting this error into the equation for an on-line estimation method (equation 20) yields the TD procedure:

$$v_{t+1} = v_t + \beta[r_{t+1} + \gamma V_t(x_{t+1}) - V_t(x_t)]\phi_t. \tag{29}$$

This is identical to a special case of the TD model of classical conditioning (Sutton and Barto, this volume) obtained by setting the parameters α_i of that model to 1 and the parameter δ to 0. If δ were present within the framework developed here, setting it to a nonzero value would have the effect of replacing ϕ_t in equation 29 with ϕ_t plus a weighted sum of the feature vectors representing states through which the system passed in time steps preceding step t. Using such a stimulus trace in the TD procedure can accelerate the learning process (Sutton 1988), and this kind of stimulus trace can be viewed within the framework of dynamic programming (Watkins 1989).

Additional insight into the TD procedure can be gained by relating the TD error (equation 28) to one of the basic equations of stochastic dynamic programming. According to the discussion of stochastic dynamic programming in section 5, we know that the evaluation function V^π must satisfy equation 7 for each state x. In particular, we know that

$$V\pi(x_t) = R(x_t, a_t) + \gamma \sum_{y \in X} P_{xt,y}(a_t) V^\pi(y),$$

and we can define an error on the basis of how much the current estimate for the evaluation function depends from this condition for state x_t:

$$\left[R(x_t, a_t) + \gamma \sum_{y \in X} P_{x_t,y}(a_t) V_t(y)\right] - V_t(x_t). \tag{30}$$

If it were possible to adjust the parameter vector v_t to reduce this error, the new evaluation function estimate would tend to be closer to the actual evaluation function. Updating the parameter vector this way would be similar to applying a step of the successive-approximation method specified by equation 6, but only to state x_t. The quantity

$$R(x_t, a_t) + \gamma \sum_{y \in X} P_{x_t,y}(a_t) V_t(y)$$

is likely to be a better estimate for $V^\pi(x_t)$ than is $V_t(x_t)$, because it takes into account the expected consequences of performing action a_t when the system is in state x_t, whereas $V_t(x_t)$ does not.

However, it is not possible to determine the error given by expression 30 in an on-line estimation method, because it depends on the unknown payoff expectations, $R(x_t, a_t)$, and the unknown transition probabilities, $P_{x_t,y}(a_t)$, for a_t, x_t, and all state y. The TD error (equation 28) can be regarded as the result replacing these unknown quantities with approximations that are available during on-line learning. First, the payoff actually received at step $t + 1$, which is r_{t+1}, is substituted for the expected value of this payoff, $R(x_t, a_t)$. Because $E[r_{t+1} | x_t, a_t] = R(x_t, a_t)$ (equation 1), r_{t+1} is an unbiased estimate of $R(x_t, a_t)$. Second, the current evaluation estimate of the state *actually reached* in one step is substituted for the expectation of the evaluation estimates over the states *reachable* in one step. That is, $V_t(x_{t+1})$ is used in place of $\sum_y P_{x_t,y}(a_t) V_t(y)$ in equation 30. $V_t(x_{t+1}) = v_t^{\mathrm{T}} \phi_{t+1}$ is the evaluation estimate of the *next* system state using the *current* parameter values. These substitutions yield the TD error given by equation 28, which involves only quantities available at time step $t + 1$.

This substitution of quantities available at each time step for unknown expected values is similar to a step taken in deriving the LMS rule as a gradient-descent procedure, as discussed in subsection 6.2. The LMS rule is a consequence of assuming that the error in responding to a single training pattern can be used to approximate the model's expected error over all patterns. Similarly, the TD procedure is a consequence of assuming that the payoff actually received at a time step is a good estimate for the expected value of the payoff at that step, and that the evaluation estimate of the next state is a good estimate of the expected value of the evaluations of the possible next states.

Despite this similarity to the LMS rule, the situation is considerably more complex for the TD procedure, and it is not as easy to interpret the TD procedure in terms of gradient descent. For example, if the objective were to minimize the expected value of the square of the error given by expression 30 over all states by gradient descent, one would arrive at a parameter-update rule that would differ slightly from the TD procedure and would not perform as well in practice.[20] Another source of complexity is that the training examples experienced by the TD procedure can be influenced by the agent's actions so as to bias the estimate of $\sum_y P_{x_t,y}(a_t) V_t(y)$. Sutton (1988) has been able to prove that, under certain conditions, the TD procedure given by equation 29 converges to form the desired evaluation function with repeated trials, each consisting of a finite number of time steps. Among the conditions required for convergence is that the parameter values are not updated during a trial. The parameter adjustments determined by the TD procedure are accumulated during each trial, and are used to update the parameter values only at the end of that trial. In practice, however, it is often adequate to update parameter values at each time step.

It is beyond the scope of this chapter to discuss the more theoretical aspects of the TD procedure. The theory of the TD procedure is neither as simple nor as fully developed as the theory of the LMS rule. However, it is noteworthy that a special case of the TD procedure is thoroughly understood because it *is* the LMS rule. When γ is set equal to 0 in equation 29, the result is

$$v_{t+1} = v_t + \beta[r_{t+1} - V_t(x_t)]\phi_t, \tag{31}$$

which is exactly the LMS rule give by equation 21. We can think of the LMS rule used in this way as a one-step-ahead adaptive predictor, which is a standard application of the LMS rule (Widrow and Stearns 1985). Applied to a sequential decision task, this rule adjusts the parameter values so that $V_t(x_t)$, the evaluation estimate of the state at time step t, predicts the payoff to be received at the next time step: r_{t+1}. Setting γ equal to 0 in equation 29 therefore renders the TD procedure capable of estimating only the most "short-sighted" evaluation function.

Let us summarize the application of the TD procedure given by equation 29. Figure 5 shows two successive time steps of the application of the TD procedure. This figure is a combination of the diagram of the system underlying the decision task shown in figure 2 (appearing at the top of figure 5, although here the "disturbances" shown in figure 2 are omitted to simplify the figure) with the diagram of on-line parameter estimation shown in figure 4, where the latter has been modified according to the special features of the TD procedure.

An agent employing the TD procedure interacts with a dynamical system that can be in any one of a number of states at each time step. The agent uses some fixed policy, π, for selecting its actions, which influence the behavior of the system. At present, it does not matter what this policy is; it could be a good policy or a bad one in terms of its expected return for system states. The agent is attempting to estimate the function that associates to each state an estimate for the expected return of policy π.

Although the expected return depends on the current state of the system, the agent does not necessarily receive enough information to accurately determine the state. At time step t, the agent has access to the feature vector $\phi_t = \phi(x_t)$, determined from state x_t by the box labeled "Representation" in figure 5. One can think of this feature vector as a pattern of sensory stimulation produced by the current state of the system, or as a representation provided by some mechanism (not discussed here) that estimates the system's state. Although states with different evaluations may give rise to the same feature vector, and states with similar evaluations

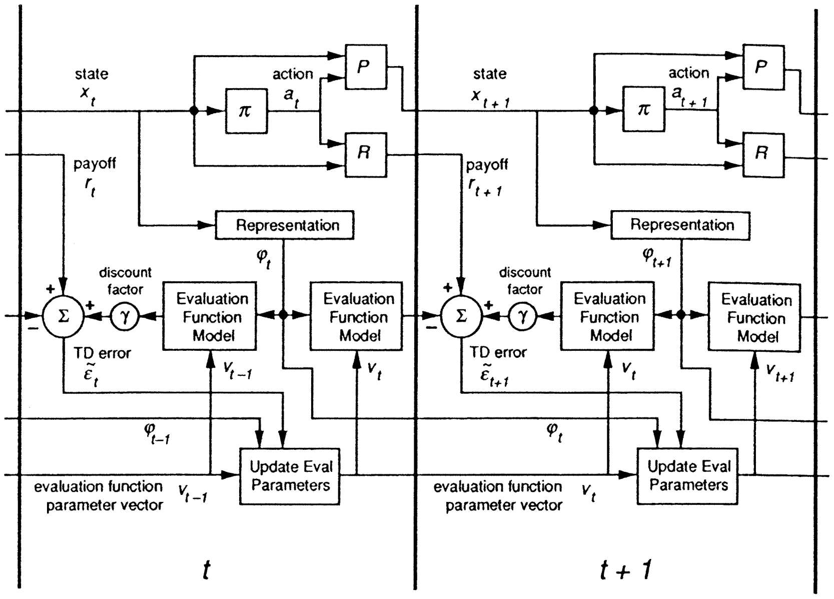

Figure 5
Two successive time steps of application of the TD procedure. This figure is a combination of the diagram of the system underlying the decision task shown in figure 2 (here appearing at the top of the figure) with the diagram of on-line parameter estimation shown in figure 4, where the latter is modified according to the special features of the TD procedure. Notice that if $\gamma = 0$, the left box labeled "Evaluation Function Model" in each time period becomes disconnected, and the lower part of the figure takes the same form as figure 4. In this figure and in figure 6, crossing lines do not represent connections unless there is a dot on the intersection.

may give rise to different feature vectors, it is obviously better if the representation assigns similar feature vectors to states with similar evaluations. Better estimates and faster learning are possible if the representation is suited to the task. Ideally, one would like features that are highly correlated with the evaluations of the states. For example, a wind-blown odor may be strongly correlated with high return if the policy is to travel upwind. In any case, assume that the representation is given as well as the parameterized class of models for the evaluation function. In this chapter we consider only linear models for the evaluation function; that is, we assume that the evaluation function is a weighted sum of the feature values. The agent's task is to find values of these weights to produce a good estimate of the actual evaluation function.

Upon receiving the feature vector ϕ_t, the agent determines its estimate for the expected return using its current parameter vector, v_t. This is the evaluation estimate of state x_t given by $V_t(x_t) = v_t^T \phi_t$. It is computed by the *rightmost* box labeled "Evaluation Function Model" within the time $= t$ part of figure 5. (At this point, the leftmost box labeled "Evaluation Function Model" within the t part of the figure has already been used to provide information for updating v_{t-1} to form the current parameter vector v_t, a process we explain below in terms of time step $t + 1$). Policy π is used to select an action on the basis of the current system state, x_t. After the action is performed at step t, the system makes a transition to a new state, x_{t+1}, and the agent receives another feature vector, ϕ_{t+1}, and a payoff, r_{t+1}.

Refer now to the $t + 1$ part of figure 5. Using parameter vector v_t (which has not been updated yet), the agent computes an estimate for the return expected from time step $t + 1$ onward. This is

$$V_t(x_{t+1}) = v_t^T \phi_{t+1},$$

which is computed by the *leftmost* box labeled "Evaluation Function Model" within the $t + 1$ part of figure 5. Now, if the linear model of the evaluation function is adequate and the parameter vector is correct, then on the average it should be true that the evaluation estimate of the state at time t,

$$V_t(x_t) = v_t^T \phi_t,$$

equals r_{t+1} plus γ times the evaluation estimate of the state at time $t + 1$; that is, $V_t(x_t)$ should, on the average, equal $r_{t+1} + \gamma V_t(x_{t+1})$. The discrepancy between these estimates is the error, $\tilde{\varepsilon}_{t+1}$ (determined by the summation unit on the left-hand side of the $t + 1$ part of figure 5), which is used to update the parameter vector v_t. This update process is shown in figure 5 as the box in the $t + 1$ area labeled "Update Eval Parameters," which implements equation 29 to produce the parameter vector v_{t+1}. Using this parameter vector, state x_{t+1} is *reevaluated* by the rightmost box labeled "Evaluation Function Model" in figure 5, and the result is available at time step $t + 2$ for a repeat of the steps described above.[21]

Figure 5 makes a subtlety of the TD procedure clear: At each time step, two estimates of the evaluation of the system state are computed—the first using the parameter vector before it is updated and the second using the parameter vector after it has been updated. This is why there are two boxes labeled "Evaluation Function Model" for each time step (although, of course, one can think of the computation for each time step as using the same box twice, with a different parameter vector each time).[22]

The Route-Finding Example To illustrate the application of the TD procedure to the route-finding example, we use one of the simplest state representations possible. We assume that the 96 states (spatial locations) are numbered 1 to 96, and we represent each state, x, by a feature vector of 96 components:

$$\phi(x) = (\phi_1(x), \ldots, \phi_{96}(x))^T,$$

where

$$\phi_i(x) = \begin{cases} 1 & \text{if } x = i \\ 0 & \text{otherwise.} \end{cases} \tag{32}$$

The feature vector $\phi(x)$ therefore is a list of zeros with a single one in the position corresponding to state x. With this set of features, the linear model of V^π given by equation 23 takes the simple form

$$v_1\phi_1(x) + \cdots + v_{96}\phi_{96}(x) + v_{97} = v_x + v_{97}$$

for parameter vector $(v_1, \ldots, v_x, \ldots, v_{97})^T$, where v_x is the parameter, or weight, associated with state x. For simplicity, we let $v_{97} = 0$,[23] and we let v_x denote the value at time step t of the parameter associated with state x. Thus, the estimate at time step t for the evaluation of state x is simply

$$V_t(x) = v_x. \tag{33}$$

Whereas $V_t(x)$ denotes the estimate at time step t of the evaluation of state x, it is not necessarily the case that x is the system state at time step t.

Suppose the agent uses a fixed policy to decide how to move around the space. Assume that it moves from location x at time step t to location y at step $t + 1$. If x is not the goal, then a payoff of -1 is received. Applying the TD procedure in the form it takes given the simple way we represent locations, only the parameter v_x, which is the current estimate of the evaluation of location x, is updated after the agent makes this move. This evaluation estimate, v_x, is adjusted by adding to it $\beta[-1 + \gamma v_y - v_x]$. To simplify things even more, let us assume that $\gamma = 1$ and $\beta = 1$. Then the new evaluation estimate of x, i.e., the new value of the parameter v_x, is

$$v_x + [-1 + v_y - v_x] = -1 + v_y.$$

Given the foregoing manner of representing states and evaluation estimates of states, adjusting parameter v_x directly adjusts the evaluation estimate of state, or location, x. A feature vector can be thought of as indexing into a lookup table that lists the evaluation estimates of all the locations.[24] Because $\phi(x)$ contains a single 1 in the position corresponding to state x, and appears multiplicatively on the right-hand side of equation 29, at each time step, the TD procedure updates only the evaluation estimate of the agent's current location. We use feature vectors defined by equation 32 because they make the application of the TD procedure easy to understand, and they allow us to represent an arbitrary evaluation function using a linear model. However, this is a very special case of the range of possibilities encompassed within the theoretical framework presented here, and many interesting phenomena involving the transfer of evaluation information among the states do not appear if one uses the feature vectors defined by equation 32.[25]

Normally, one would begin the process of estimating the evaluation function for a given decision policy by setting the parameters to provide as good an approximation to the true evaluation function as is possible. For simplicity, however, suppose we begin by setting all the parameters to 0 and then run a series of trials, beginning each trial by placing the agent in a randomly chosen location and letting it take some reasonably large number of steps. As the agent moves in the first trial, it will leave a trail of -1s in its wake as the parameters that correspond to each location visited are updated. If it reaches the goal, the parameter determining the goal's evaluation, v_g, will remain 0. On the second trial, the same thing will happen except when the agent's path joins and moves along a path taken in a previous trial (once the agent's path intersects a path previously traversed, it will always follow that path if the decision policy is deterministic). The parameter for each location visited one step before visiting a location already assigned a -1 will be set to -2. If the goal is reached, the location visited one step earlier will remain at -1. On the third trial, some locations will be assigned -1, some will be assigned -2 (as in trial 2), and some will be assigned -3 (if the next place visited had been assigned -2 in the second trial).

Notice what happens if, in this third trial, the agent approaches the goal on a path traveled in both preceding trials. It will reach a location (call it x) assigned an evaluation estimate of -2, then move to a location assigned an evaluation estimate of -1 (call it y), and then reach the goal, g, which still has an evaluation estimate equal to 0. Moving from x to y generates a payoff of -1, and the TD rule will update v_x to produce the new evaluation estimate

$$-1 + v_y = -1 - 1 = -2,$$

which is the same as its previous value. Similarly, v_y will not change when g is reached, because its current evaluation estimate already equals the payoff received upon moving to the goal, -1, plus the goal's evaluation, 0. As various paths are followed on subsequent trials, evaluation estimates stabilize in a similar fashion, from the goal outward, to minus the number of steps required to reach the goal using the given decision policy. If the policy prevents the goal from being reached from some location, then the evaluation estimate of that location can become more negative with each trial, being decremented by the TD rule each time the location is visited. With an unbounded number of trials, evaluation estimates for such locations grow negative without bound, which is correct (since the number of steps to the goal using the policy is indeed infinite) but which may be considered an undesirable form of instability. The use of a discount factor with a value less than 1 eliminates the possibility that evaluation estimates can grow without bound.

7.2 Learning an Optimal Decision Policy

Subsection 7.1 addressed the problem of learning the evaluation function for a fixed policy in the absence of a model of the decision task. The TD procedure is a parameter-estimation method for estimating such evaluation functions. Here we consider one way of using the TD procedure to facilitate the approximation of an optimal policy, again in the absence of a model of the decision task. Although the resulting method is not guaranteed to converge to an optimal policy in all tasks, it is relatively simple and is closely related to traditional views of animal learning. Throughout the present subsection we will assume that we do not know the quantities $R(x, a)$ and $P_{xy}(a)$, for system states x and y and actions a, which are required for the application of dynamic programming methods.

The approach described here for learning an optimal policy is related to the policy-improvement method described in subsection 5.2. In the policy-improvement method, one computes the evaluation function for some initial policy and then defines a new policy that specifies the actions that maximize expected return as measured by that evaluation function. One then computes the evaluation function for this new policy, defines another policy so as to maximize the expected return as measured by this second evaluation function, computes another evaluation function, and so on. The policies defined in this sequence keep improving until an optimal policy is obtained. The method for forming an optimal policy described below can be thought of as combining, on a step-by-step basis, the process of forming an evaluation function with the process of forming a new policy. Whereas the policy-improvement method computes a *complete* evaluation function for a given policy (that is, it computes the evaluation of *all* states for that policy, and then uses these evaluations to define an improved policy), the method described below performs one step of an on-line procedure for estimating an evaluation function for a given policy and at the same time performs one step of an on-line procedure for improving that policy. Consequently, both the current evaluation-function estimate and the current policy change at each time step while the agent interacts with the system underlying the decision task. Because each policy adjustment is based on an estimate of the evaluation function for the current policy, the policy will not necessarily improve at each step, as it does in the policy-improvement method. However, because the evaluation estimates improve over time, the policy also tends to improve.

Two on-line procedures are therefore required for the approach to learning an optimal policy considered here. The procedure for estimating the evaluation function is the TD procedure, and the procedure for improving the decision policy is an example of a reinforcement learning procedure (see section 3). Because our goal in this chapter is to describe the computational aspects of the TD procedure, we do not provide a detailed exposition of reinforcement learning. Instead, we describe the general characteristics of these methods and discuss a specific example as it applies to the route-finding task. The processes involved are diagrammed in figure 6, to which we will refer throughout this subsection. This figure is an expansion of figure 5. Whereas in figure 5 the fixed policy is represented by the boxes labeled π, in figure 6 the policy is adjusted by the procedure diagrammed in the top part of the figure. The bottom part of figure 6 shows the operation of the TD procedure and is identical to figure 5.

In subsection 7.1 the TD procedure was developed as an on-line parameter-estimation method for estimating an evaluation function. Similarly, the procedure for improving policies that we describe is an on-line parameter-estimation method. Referring to the concept of a parameterized model described in section 6, each policy available to the agent is assumed to be given by some parameterized model, called the policy model. Selecting specific values for the parameters, which we call the policy parameters, determines a specific policy. Policy parameters are adjusted in an attempt to improve the policy they specify (accomplished by the boxes labeled "Update Policy Parameters" in figure 6). At the same time that the policy parameters are adjusted, the TD procedure adjusts different parameters specifying the evaluation-function model.

At time step t, the state of the system underlying the decision task is represented by the feature vector $\phi_t = \phi(x_t)$, which acts as input to the model of the evaluation function. In figure 6, this representation is produced by the box labeled "Representation A" in the t portion of the figure. Another feature vector representating the state at step t acts as input to the model of the policy. This feature vector is denoted ϕ_t' and is shown in figure 6 as the output of the box labeled "Representation B" in the t portion of the figure. Although it may be not always be necessary to use different feature vectors for the evaluation-function model and the policy models, in general these two models do not place equal demands on state representations. However, two different parameter vectors are necessary: the parameter vector v, which specifies the evaluation function as described in subsection 7.1, and the vector w, which specifies the policy. For simplicity, we consider only linear models for evaluation functions and decision policies.

As the agent interacts with the system, it senses the feature vectors of successive states and receives a payoff at each time step. The TD procedure (equation 29) is applied exactly as described in subsection 7.1 to update the parameter vector v. Here, however, the policy changes over time as the

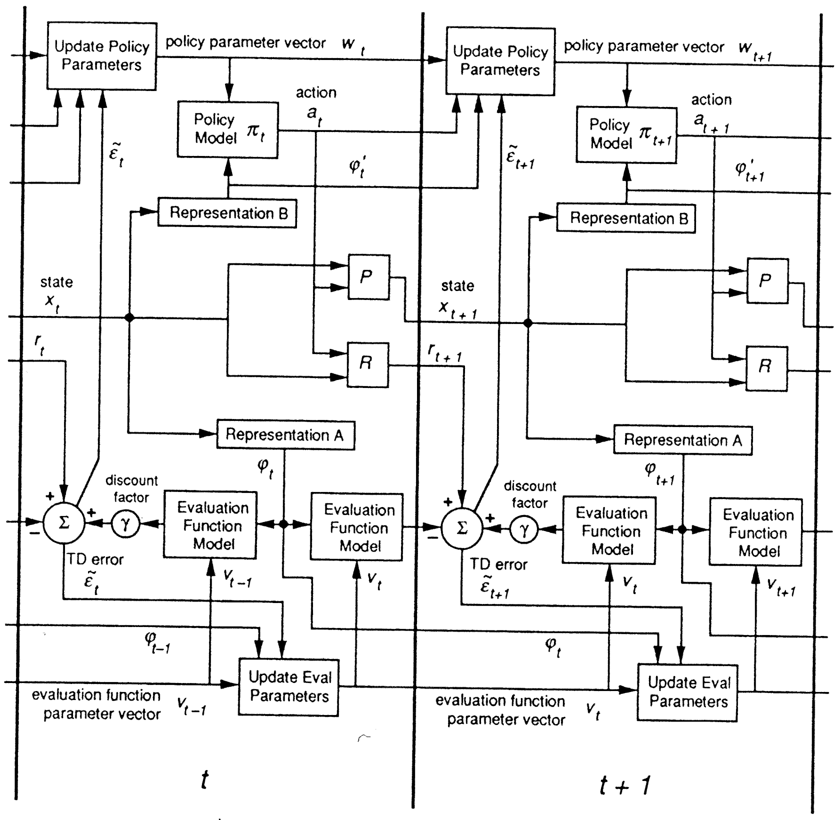

Figure 6
An elaboration of figure 5 showing the operation of an on-line method for adjusting a decision policy. The policy is adjusted through the process diagrammed in the upper part of the figure. The lower part of the figure shows the operation of the TD procedure and is identical to figure 5.

parameter vector w is adjusted. Let π_t denote the policy used to select the action at time step t. This policy is specified by w_t, the value of the parameter vector w at time step t for the policy model. Although at each time step t the TD procedure is approximating the evaluation function for policy π_t, over many time steps it is approximating the policy to which π_t is tending as t increases. Because under ideal circumstances this limiting policy should approximate an optimal policy, we regard the TD procedure used in this way as approximating the optimal evaluation function.

How does w_t specify the policy π_t, and how is w_t updated? A policy is a mapping, or function, from system states to actions that can be parameterized in many different ways. If a policy can assign to each system state one of only two possible actions, a policy might be represented as a linear decision rule having the same form used in pattern classification (equation 16). If more than two actions are possible, as in the route-finding example, a more complicated parameterization is required. Several possibilities for parameterizing multi-category decision rules are suggested in the pattern-classification literature (Duda and Hart 1973; Sklansky and Wassel 1981). One method, which we adapt for the route-finding example is to use a separate parameterized function for each possible action. Each of these functions assigns a number to each state. An action is then selected by comparing the numbers produced by these functions for the current state and selecting the action whose number is largest. This is a kind of competition among the actions that can be implemented by connectionist networks with lateral inhibition (Feldman and Ballard 1982).

An additional consideration in parameterizing policies is that some policy-adjustment methods require stochastic policies instead of deterministic policies. In this case, one has to parameterize functions that assign action probabilities, instead of specific actions, to system states. We shall describe an example of a parameterized class of multi-action stochastic policies in the context of the route-finding example. In the meantime, however, it is not misleading to think in terms of the simpler case of deterministic, two-action policies, the simplest example being policies defined by the linear threshold rule given by expression 16.

If it were possible to know what action is desired for each state, it would be possible to adjust the policy parameters by applying a parameter-estimation method, such as the LMS rule given by equation 21, using the supervised learning paradigm. If we knew the expected values, $R(x_t, a)$, of all the payoffs and transition probabilities, $P_{x_t,y}(a)$, for the current system state, x_t, and all possible next states y under all the actions a, then we could use the current evaluation function, V_t, to select the desired action. It would be the action, a, that maximizes

$$R(x_t, a) + \gamma \sum_{y \in X} P_{x_t,y}(a) V_t(y). \tag{34}$$

A policy selecting this action for state x_t, and otherwise following policy π_t, either improves policy π_t or leaves its performance unchanged, under the assumption that the estimate of the current evaluation function, V_t, is an accurate estimate of the true evaluation function for policy π_t. Unfortunately, lacking knowledge of the expected payoff and transition probabilities that appear in expression 34, and also lacking estimates for these quantities, we cannot directly specify a desired action. Therefore, we cannot update the current policy using a rule for supervised learning, such as the LMS rule, which requires a known desired response for each input vector. It is necessary to employ a method that is capable of learning to perform the desired actions (here, the actions that maximize expression 34) when these actions are not known by a "teacher." In order to do this, we use a reinforcement learning method instead of a supervised learning method.

The agent might overcome the unavailability of a teacher capable of providing desired responses by forming estimates of the quantities in expression 34 while it interacts with the system underlying the decision task. This means that it would effectively have to construct a model of the decision task. However, reinforcement learning is possible without such a model: Think of the agent as beginning an instrumental conditioning trial at each time step. At time step t, it emits an action, a_t, based on a policy that shows some form of variation. At time step $t + 1$, it receives payoff r_{t+1} and estimates the evaluation of the state reached at $t + 1$ by applying the evaluation-function estimate specified by the parameter vector v_t (see equation 24). This evaluation estimate, $V_t(x_{t+1}) = v_t^T \phi_{t+1}$, which is an estimate of expected return, is added to the immediate payoff r_{t+1}, after discounting by γ, to produce a measure of the utility of emitting a_t, i.e., a measure of how useful a_t is in causing high return. This utility measure is

$$r_{t+1} + \gamma V_t(x_{t+1}). \tag{35}$$

This measure, which is identical to the sum of the first two terms of the TD error $\tilde{\varepsilon}_{t+1}$ (equation 28), can be used to alter the strength of the associative link between the feature vector ϕ_t' and a_t in order to change the probability that a_t will be performed whenever state x_t, or a state represented by a feature vector similar to ϕ_t', is encountered in the future. The expected value of the measure given by expression 35 is largest for the action that, if performed at time step t, would be the best action for improving policy π_t under the assumption that V_t is an accurate estimate of the evaluation function for the current policy. We need a learning rule that increases the probability of performing action a_t when the utility measure given by expression 35 is large, and that decreases this probability when it is small.

But what values of this utility measure should be considered "large" or "small"? How can a measure of the utility of an action, such as that given by expression 35, be translated into a quantity that can be used effectively as a "reinforcement factor" to adjust action probabilities? If one had access to the expected value of the utility measure over all actions performed when the system is in a given state, it would make sense to treat measures larger than this expected value as indicating performance improvements and to treat measures smaller than this expected value as indicating performance decrements.[26] According to this approach, a reinforcement factor would be obtained by subtracting the expected value of the utility measure, or an estimate of this expected value, from the actual utility measure obtained for a given action. Using this reinforcement factor, a learning rule could favor actions leading to better-than-expected performance while selecting against actions leading to worse-than-expected performance.

Conveniently, within the framework described here, for each system state we already have an estimate for the expected value of the utility measure given by expression 35 over actions performed in that state: We can use the evaluation estimate of a state given at time step t by the evaluation function estimate V_t (equation 24). To see why the evaluation estimate of a state can serve this purpose, consider what happens to the TD error

$$\tilde{\varepsilon}_{t+1} = r_{t+1} + \gamma V_t(x_{t+1}) - V_t(x_t)$$

(equation 28) as the TD procedure adjusts the parameter vector, v_t, defining V_t: The expected value (over system states) of $\tilde{\varepsilon}_{t+1}$ should tend to 0, and we would expect $V_t(x_t)$ to approach the expected value of $r_{t+1} + \gamma V_t(x_{t+1})$, which is the utility measure of performing action a_t given by expression 35. Consequently, a reinforcement factor can be formed by subtracting this evaluation estimate, $V_t(x_t)$, from $r_{t+1} + \gamma V_t(x_{t+1})$. The result it just the TD error itself (equation 28), repeated here for emphasis:

$$\tilde{\varepsilon}_{t+1} = [r_{t+1} + \gamma V_t(x_{t+1})] - V_t(x_t). \qquad (36)$$

The TD error, $\tilde{\varepsilon}_{t+1}$, therefore provides information about how good action a_t is in comparison with how good previous actions taken in response to ϕ_t' (see figure 6) have been "on the average." If $\tilde{\varepsilon}_{t+1}$ is positive, then a_t is better than average and should be made to occur more frequently in response to ϕ_t' in the future; if $\tilde{\varepsilon}_{t+1}$ is negative, then a_t is worse than average and should be made to occur less frequently in response to ϕ_t'. It is relatively easy to use a quantity having these properties as a reinforcement factor for adjusting the policy parameters. This is accomplished by the component labeled "Update Policy Parameters" in figure 6, which receives the TD error as one of its inputs.

Several types of rules have been studied for adjusting a policy on the basis of a reinforcement factor, such as $\tilde{\varepsilon}_{t+1}$, instead of direct specifications of desired actions required for supervised learning. In order for these reinforcement learning rules to work effectively, different actions must be performed in response to each system state so that the relative merits of performing different actions can be assessed. One way to maintain the necessary variability in activity is to let the agent "explore" by using and improving stochastic policies. It is beyond the scope of this chapter to describe stochastic reinforcement learning in detail, but we can give an example of this approach in the context of the route-finding example.[27]

The Route-Finding Example A deterministic policy for this task assigns to each location one of the four actions N, S, E, W. Although we would like to end up with a deterministic policy selecting a shortest path to the goal from each location, the learning process we describe modifies a stochastic policy. A stochastic policy selects actions probabilistically, so that over time many actions are tried from each location. This exploratory behavior is refined as learning continues, favoring the actions appearing to yield the highest return. In other words, the stochastic policy is adjusted, by modifying the parameters specifying it, so that for each location the probability of one action (which should be the optimal action) approaches 1, while the probabilities of the other actions approach 0.

Of the many different ways to define stochastic policies for the route-finding example, we used the following. The set of policy parameters, i.e., the parameters defining a stochastic policy, consists of a separate parameter for each location, x, and each action, $a \in \{N, S, E, W\}$. We let $w(x, a)$ denote the value of the parameter for location x and action a. This value can be an arbitrary real number. The probability that any one of the actions is emitted when the agent visits location x is determined by the parameters corresponding to location x: $w(x, N)$, $w(x, S)$, $w(x, E)$, and $w(x, W)$. Actions are selected by means of the following probabilistic competitive process. At each time step t, four positive random numbers are drawn from an exponential distribution so that smaller values are more probable than larger values. If the agent is positioned at location x_t, a different one of these random numbers is added to each of $w(x_t, N)$, $w(x_t, S)$, $w(x_t, E)$, and $w(x_t, W)$, and the action for which this sum is the largest is performed by the agent, which moves to a new location and receives payoff r_{t+1}. In other words, the action, a_t, performed at time step t is the action a such that $w(x, a) + \eta_a$ is largest, where η_a is the random number drawn for a. The parameter values $w(x, a)$ are not altered in this selection process. Selecting actions using this competitive process has the advantage that no special precautions have to be taken to ensure that valid action probabilities are maintained throughout learning.

Now, the parameter value $w(x_t, a_t)$, corresponding to the action actually selected, a_t, is updated so that the new value is

$$w(x_t, a_t) + \rho\tilde{\varepsilon}_{t+1}, \tag{37}$$

where x_t is the agent's location at time step t, ρ is a positive constant, and $\tilde{\varepsilon}_{t+1}$ is the reinforcement factor (equation 36) corresponding to performing action a_t when the system state is x_t. Thus, if $\tilde{\varepsilon}_{t+1}$ is positive, the move that was taken from location x_t will be chosen with a higher probability when location x_t is visited again. If $\tilde{\varepsilon}_{t+1}$ is negative, a_t will be less likely to be chosen from location x_t. The parameters corresponding to the actions not selected are not changed, although the probabilities of these actions being selected are changed as a result of the change in the values $w(x_t, a_t)$ and the competitive action-selection process described above. As the parameter values, $w(x, a)$, for the various places visited and actions selected change according to expression 37, the decision policy changes, becoming more deterministic as specific actions become selected with higher probabilities. At the same that the parameter values $w(x, a)$ are updated, the TD procedure (equation 29) is applied to adjust the values of the parameters specifying the evaluation function.

Because the reinforcement factor, $\tilde{\varepsilon}_{t+1}$, used to adjust the policy parameters according to expression 37 is the same as the error used by the TD procedure (equation 29), $w(x_t, a_t)$ is updated in exactly the same way that the TD procedure updates the parameter v_{x_t} specifying the evaluation estimate of state x_t. Indeed, there is only one difference between the TD procedure and the rule used to update the policy parameters. Whereas the TD procedure updates the evaluation-function parameters in a manner that is insensitive to the action chosen at time step t, the rule given by expression 37 updates only those policy parameters corresponding to the action chosen at step t. More generally, if the decision policy is represented in a manner more complicated than the one illustrated here—for example, if actions are represented as patterns of activity over a set of connectionist units—then the updating of policy parameters is modulated by the representation of the chosen action so as to appropriately influence the action probabilities. In figure 6, notice that the component "Update Policy Parameters" receives actions as input, whereas the component "Update Eval Parameters" does not. Aside from this dependence of the policy-update procedure on the action performed, and possibly different values for the constants β in equation 29 and ρ in equation 37, the parameter-update procedures for evaluation estimation and policy improvement are exactly the same.

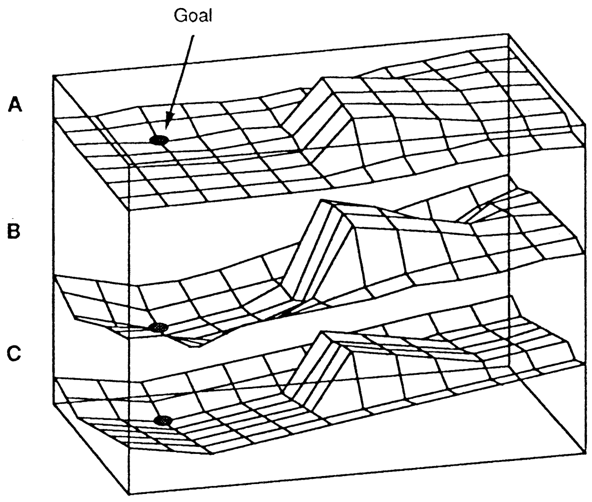

Figure 7
Estimates of the optimal evaluation function for the route-finding examples. Surface A: after 50 trials. Surface B: after 500 trials. Surface C: after 5000 trials. The surface height at each location is the estimate of the minimum number of steps to the goal from that location (and hence is the graph of the negative of the corresponding estimate of the optimal evaluation function). Because surface C corresponds to the optimal evaluation, an optimal policy is any policy selecting the actions leading most steeply down this surface, with the exception, of course, of the actions blocked by the barrier (see figure 1 for the position of the barrier).

Figure 7 shows the estimates for the optimal evaluation function for the route-finding example after various numbers of trials in which both the TD procedure and the policy-update procedure just described were applied at each time step. A trial begins when the agent is placed at a randomly chosen location and ends when the agent reaches the goal. Each evaluation-function estimate is shown as a surface whose height at each location is the estimate of the minimum number of steps to the goal from that location. Since evaluation functions in this task give the *negative* of the number of steps to the goal, each surface shown in figure 7 is actually a graph of the negation of an estimate of the optimal evaluation function. The goal is marked on each surface, and the reader should refer to figure 1 for the position of the barrier. If a surface corresponds to the optimal evaluation function, as can be shown for surface C in figure 7, then an optimal policy is any policy selecting the actions leading most-steeply down the surface (with the exception, of course, of the actions blocked by the barrier). By the definition of the optimal evaluation function, the surface corresponding to the optimal evaluation function can have no local minimum that is not at the goal.

Surface A is the result after 50 trials. Surfaces B and C show the results after 500 and 5000 trials, respectively. Examination of surface C reveals that it is a close approximation of the (negated) optimal evaluation function for this task. Additionally, the agent's policy improves over trials until it becomes an optimal policy with a slight degree of nondeterminism. The average over 100 trials of the number of steps required to reach the goal before learning was 2690 steps per trial. After 50 learning trials the average over 100 non-learning trials was 41; after 500 trials it was 9.14; after 5000 trials it was 6, which is near the minimum possible for this task. From these data one can see that, although many trials were required to optimize performance, rapid initial improvement occurred. This improvement would not have occurred if the agent had spent these trials estimating only state-transition and payoff probabilities.

Although the route-finding example illustrates many of the properties of the learning methods we have described, it is too simple to illustrate others. For example, the applicability of the methods to stochastic decision tasks is not illustrated. In the example, an action performed in a given location always causes the same payoff and the same state transition. This determinism in the state-transition and payoff processes makes the application of the TD procedure and policy learning procedure easier to understand, but it does not test the ability of this learning method to estimate an optimal evaluation function and improve a decision policy in the presence of uncertainty. Sutton (1988) and Watkins (1989) present examples of the TD procedure applied to stochastic tasks. Another simplification present in the route-finding example is due to the particular representation used. The estimates of the optimal evaluation function and the optimal policy are represented in the form of a lookup table, with a separate set of parameters for each system state. As was mentioned in subsection 7.1, this representation prevents generalization: the agent must visit a location in order to update the estimates associated with that location.

The method described in this section for improving a policy consists of two concurrent adaptive processes: evaluation estimation and policy adjustment. The agent continually estimates an evaluation function for its current policy, and also uses its current evaluation function to improve the policy. To improve its policy, the agent must experiment with different actions; in the course of its visits to each state, the agent must repeatedly try out all the actions possible at that state to establish which action is best. In the learning method described here, the agent's current policy is stochastic; thus, the agent will try a variety of actions at each state while following its policy. As time goes on, the stochastic policy is adjusted so that in each state the probabilities that less-than-optimal actions are chosen should, under ideal circumstances, decrease toward zero.

Methods such as the one described in this section involve a tradeoff between acquiring information to aid decision making in the future and using the information already acquired in the attempt to select optimal actions. Always performing actions that appear optimal on the basis of current estimates (of evaluation functions, etc.) can prevent the experimentation required to improve these estimates. Although adjusting a stochastic policy is one way to decrease the severity of this problem, it is not easy to design methods that work well in all decision tasks. The amount of experimentation that an agent using a stochastic policy can do is limited by the degree of randomness of the policy. In the later stages of learning, the policy becomes almost deterministic, and the rate of experimentation (and, hence, the rate of policy improvement) becomes very low. At intermediate stages of learning, the policy may become less than optimal and almost deterministic for some states. Because experimentation for these states is conducted at such a low rate, changes in the policy for these states may be very slow. Although the method of adjusting a stochastic policy while estimating an evaluation function described in this section does perform effectively in a variety of tasks, especially if learning involves a series of trials with random starting states, we know of no one who has worked out conditions under which it always converges to an optimal policy.

Watkins (1989) has suggested a general approach to minimizing the difficulties caused by the tradeoff between estimation and control. The idea is to separate exploration from evaluation estimation by letting the agent deviate from the policy for which it is estimating an evaluation function—its *estimation policy*—as long as it switches off its evaluation-estimation procedure when such a deviation is made. The agent may do anything it likes, as long as it does not modify its estimate of the evaluation function on the basis of information acquired when it deviates from its estimation policy. The agent is only correct to use its experience to estimate the evaluation function for the estimation policy when it is actually following that policy; however, it can adjust its estimation policy all the time, whether it is following it or not. This enables an agent to experiment while maintaining the integrity of its current estimation policy and its current evaluation-function estimate. The agent might choose experiments with more randomness than is specified by its estimation policy; it might have specific innate curiosities that lead it to try certain actions in certain circumstances; or it might imitate the actions of others. Beneficial outcomes of any of these experiments can be incorporated into the estimation policy.

8 Conclusion

We have shown how the temporal-difference procedure emerges as a combination of techniques from stochastic dynamic programming and parameter estimation. Embedding the TD procedure within these theoretical frameworks connects it with very rich traditions of theoretical research and helps explain what kinds of computational tasks it can help solve. Placing the TD procedure in this computational perspective suggests how this procedure might be applied to practical engineering problems as well as how it might help explain aspects of animal behavior.

We think the best way to relate the sequential-decision framework and the TD procedure to a real-time view of classical conditioning is as follows. Think of the process of updating an evaluation-function estimate as occurring continuously throughout each trial in response to continuously varying stimulus patterns and reinforcement values. The stimulus patterns provide information (possibly incomplete) about how the state of some underlying system is changing continuously over time, where the dynamics of this system are determined by the experimental procedure. The application of the TD procedure at discrete time intervals is a discrete-time simulation of a process having no explicitly demarcated steps. According to this view, the learning mechanisms that play the major role in classical conditioning construct a gradient of secondary reinforcement by facilitating connections from stimuli, and perhaps from internally stored representations, that are predictive cues for events that provide primary reinforcement. The values r_{t+1} and $\gamma V_t(x_{t+1}) - V_t(x_t)$ respectively correspond to the primary and the secondary reinforcement received at time $t + 1$, with the total reinforcement being the sum of these given by the TD error, $\tilde{\varepsilon}_{t+1}$. The evaluation $V_t(x_t)$ is an estimate of the expected discounted sum of future primary reinforcement available after time t. On this interpretation, a secondary reinforcer is any stimulus that causes a change in the prediction of cumulative primary reinforcement, a view developed more completely by Sutton and Barto in chapter 12 of this volume.

The task of improving a decision policy within the framework developed here is similar to the task faced by an animal in instrumental conditioning. The amount of payoff the agent receives depends on its actions, and feature vectors representing the states of the system underlying the decision task act as discriminative stimuli. The method of updating a decision policy that we outlined in the context of the route-finding example is a rather straightforward implementation of an S-R view of instrumental conditioning based on the Law of Effect (Thorndike 1911): Actions are selected according to their consequences in producing payoff by altering the strengths of connections between those actions and stimuli present when they were emitted. In the method we described, the consequences of actions are assessed by means of the sum of the immediate primary and secondary reinforcement. Although this role of secondary reinforcement in instrumental learning is also a standard feature of traditional views of animal learning, here we define secondary reinforcement as the state-to-state change in an estimate of an evaluation function, i.e., $\gamma V_t(x_{t+1}) - V_t(x_t)$, where the estimate is updated through time by the TD procedure.

Although learning methods similar to the one we described in subsection 7.2 for improving policies have been studied by means of computer simulation (Anderson 1987; Barto 1985; Barto and Anandan 1985; Sutton 1984), we do not claim that the method we described always produces an optimal policy or that it is a valid model of animal behavior in instrumental conditioning experiments. However, we think it is significant that the procedures constituting this method—one of which updates an evaluation-function estimate (the TD procedure) and other of which updates the decision policy—are almost identical. The same parameter-update rule is used in these procedures, but whereas it is applied uniformly to all the parameters specifying the evaluation function, it is applied to the parameters specifying the decision policy in a fashion that is modulated by a representation of the action that was emitted by the agent.

Whatever method the agent uses to adjust its decision policy, a means for approximating the evaluation function corresponding to a policy, such as the TD procedure, has obvious utility. An evaluation function provides an immediately accessible prediction of the long-term return a policy will achieve throughout the future. This information can be used in adjusting policies so that decisions are not dominated by the short-term consequences of behavior. If the evaluation function for a given policy is available, it is possible to select actions on the basis of their future consequences without waiting for the future to unfold. Because it is unlikely that exact evaluation functions are ever available to a behaving animal, evaluation-function estimates must serve instead. The TD procedure is a simple on-line method for forming these estimates.

We distinguished the TD procedure from model-based methods which use a model of the decision task in the form of estimates for the state-transition and payoff probabilities of the system underlying the decision task. The TD procedure is a direct method for estimating an evaluation function. It does not rely on a model of the decision task. The lack of such a model rules out many alternative strategies for basing decisions on estimates of the long-term consequences of behavior. Without a model of the decision task, it is not possible for the agent to implement conventional dynamic programming methods, such as value or policy iteration, or to perform explicit look-ahead planning of the type widely studied in artificial intelligence. Lacking a model, the agent cannot evaluate actions or action sequences without actually performing them. Model-based methods are more advanced than direct methods and can provide capabilities impossible to obtain with direct methods.

However, the additional capabilities of model-based methods should not diminish the appeal of direct methods as computational procedures and as models of animal learning. Direct methods are not only much simpler than model-based methods, they can be used in natural ways as components of model-based methods. For example, it is as easy to apply the TD procedure to state sequences contemplated with the aid of a task model as to state sequences that are actually experienced. Additionally, direct methods applied to actual experience cannot be misled by modeling inaccuracies in the same way that model-based methods can be. The desired consequences of executing a carefully crafted plan based on an erroneous model of reality can bear little relation to what actually happens. Although direct methods are themselves subject to many sources of inaccuracy, they do not risk the compounding of errors due to applying iterative methods to inaccurate task models. By using, as it were, the real task as its own model, direct methods maintain contact with the actual consequences of behavior.

Although we presented the TD procedure as a synthesis of ideas from stochastic dynamic programming and parameter estimation, the route-finding example used throughout this chapter involves only the simplest elements of these theories. Being a task with very simple, deterministic state-transition and payoff structures, it does not illustrate the generality of the dynamic programming concepts. Similarly, our applications of learning methods to this task do not adequately illustrate the wide range of possibilities for representing evaluation-function estimates and policies. Lookup tables are easy to understand and can be useful in solving some tasks, but the learning procedures presented here apply also when evaluation-function estimates and policies are represented by other means, such as connectionist networks parameterized by connection weights. Indeed, some of the most interesting open questions concerning the learning procedures described in this chapter concern their use with various kinds of state representations and parameterized classes of functions. For example, the kind of generalization produced by a representation can either aid the learning process or hinder it, and methods can be studied for adaptively re-representing functions during learning as proposed by Michie and Chambers (1968) and Riordon (1969). Good features for representing policies will tend to be highly correlated with optimal actions, and good features for representing evaluation functions will tend to provide descriptions of states that are highly correlated with the optimal return available from those states.

In addition to providing a framework in which to understand the operation of the TD procedure, the formalism of sequential decision making provides a framework in which to study a variety of other problems involving learning. Tasks requiring the control of incompletely known dynamical systems are ubiquitous in engineering applications. They are clearly related to problems that animals routinely solve by means of learning, and they offer a rich set of challenges to designers of learning algorithms. It is our hope that the connections described in this chapter encourage research which brings together ideas from animal learning theory, behavioral ecology, and adaptive-control engineering. As we have at attempted to show for the TD procedure, computational methods inspired by biological learning are not necessarily simple special cases of existing methods.

Acknowledgments

The authors acknowledge their indebtedness to C. W. Anderson and A. H. Klopf, who contributed greatly to the development of the methods presented here. We also thank S. Bradtke, J. E. Desmond, J. Franklin, J. C. Houk, A. I. Houston, and E. J. Kehoe for their helpful comments on earlier drafts of this chapter, M. Rosenfield for figure reconstruction, and J. W. Moore for his helpful criticism. A. G. Barto gratefully acknowledges the support of the Air Force Office of Scientific Research, Bolling AFB, through grants AFOSR-87-0030 and AFOSR-89-0526; the National Science Foundation, through grant INT-8815252; and the King's College Research Centre in Cambridge, England, where much of this chapter was written.

Notes

1. Because of its connection to theories and computational methods that are in widespread use in many disciplines, it is not possible to describe all the literature relevant to the TD procedure. However, we are not aware that methods are currently in use which combine parameter estimation and dynamic programming in the way they are combined in the TD procedure, although there has been much research on related problems. (Ross 1983 and Dreyfus and Law 1977 provide good expositions of dynamic programming, and notes 5 and 6 provide references to some of the related engineering research on the adaptive control of Markov processes.) To the best of our knowledge, the earliest example of a TD procedure is a technique used by Samuel (1959, 1967) in a checkers-playing program. Samuel's program used the difference between the evaluation of a board configuration and the evaluation of a likely future board configuration to modify the equation used to evaluate moves. The evaluation equation was adjusted so that its value when applied to the current board configuration came to reflect the utility of configurations that were likely to arise later in the game. Using this method, it was possible to "assign credit" to moves that were instrumental in setting the stage for later moves that directly captured the opponent's pieces. Minsky (1954, 1961) discussed the credit-assignment problem and methods similar to Samuel's in terms of connectionist networks and animal learning. Mendel and McLaren (1970) discussed similar methods in the context of control problems, and the learning method of Witten (1977), presented in the context of Markov decision problems, is closely related to the method we describe here. Werbos (1977) independently suggested a class of methods closely related to the TD procedure and was the first, to the best of our knowledge, to explicitly relate them to dynamic programming. Werbos calls these "heuristic dynamic

programming" methods. A similar connection was made recently by Watkins (1989), who uses the term "incremental dynamic programming."

Sutton (1984) developed the "adaptive heuristic critic" algorithm, which is closely related to Samuel's method but is extended, improved, and abstracted away from the domain of game playing. This work began with the interest of Sutton and Barto in classical conditioning and with the exploration of Klopf's (1972, 1982) idea of "generalized reinforcement," which emphasized the importance of sequentiality in a neuronal model of learning. The adaptive heuristic critic algorithm was used (although in slightly different form) in the reinforcement-learning pole balancer of Barto, Sutton, and Anderson (1983), where it was incorporated into a neuron-like unit called the "adaptive critic element." This system, which was inspired by the "BOXES" system of Michie and Chambers (1968), was further studied by Selfridge, Sutton, and Barto (1985) and by Anderson (1986, 1987). Since then, Sutton (1988) has extended the theory and has proved a number of theoretical results. His results suggest the TD procedures can have advantages over other methods for adaptive prediction.

A number of other researchers have independently developed and experimented with methods that use TD principles or closely related ones. The "bucket brigade" algorithm of Holland (1986) is closely related to the TD procedure as discussed by Sutton (1988) and by Liepins, Hilliard, and Palmer (in press). Booker's (1982) learning system employs a TD procedure, as does Hampson's (1983) proposed learning system, which is very similar to the one we discuss here. Other related procedures have been proposed as models of classical conditioning in publications cited in chapter 12 of the present volume.

2. Other formulations of sequential decision tasks result from different definitions of a policy's return. One formulation commonly studied defines a policy's return as the average amount of payoff per time step over a task's duration. Ross (1983) also discusses this formulation.
3. An action in a sequential decision task is not the same as a component of the agent's observable behavior. Observable behavior is a joint consequence of the agent's action and the state of the system underlying the decision task.
4. A large component of artificial intelligence research concerns search strategies of this type, called "heuristic search" strategies, although their objective is usually not to maximize a measure of cumulative payoff. See Pearl 1984.
5. Most of the methods for the adaptive control of Markov processes described in the engineering literature are model-based—see, for examples, Borkar and Varaiya 1979; El-Fattah 1981; Kumar and Lin 1982; Lyubchik and Poznyak 1974; Mandl 1974; Riordon 1969; Sato et al. 1982. Most of these methods apply to the case in which the return is the average payoff per time step and the underlying system is an ergodic Markov chain for each possible policy. They differ in how the policy is adjusted over time on the basis of the current estimates for the transition and payoff probabilities.
6. For examples of various direct methods for learning how to solve sequential-decision tasks, see Lyubchik and Poznyak 1974; Lakshmivarahan 1981; Wheeler and Narendra 1986; Witten 1977. Most of these methods rely on results about the collective behavior of stochastic learning automata and ergodic Markov chains (see also Narendra and Thathachar 1989).
7. One situation in which this sum is finite when $\gamma = 1$ is when the structure of the task ensures that all payoffs are 0 after a certain point in the decision task. However, restricting γ to be greater than 0 and strictly less than 1 ensures a finite return under all circumstances (if the payoffs themselves are finite).
8. In this case, it is not necessary to know the transition probabilities and the payoff expectations for state-action pairs that can never occur with the given policy.

9. See Ross 1983 for a detailed discussion and proof of these results. Equation 7 defines a set of simultaneous linear equations, one equation for each system state, which can be solved by matrix inversion. The successive application of equation 6 is an iterative procedure for solving this system of equations. In practice, one continues to apply equation 6 for increasing values of n until the difference between successive approximations becomes smaller than a predefined amount, or until some time limit is reached. Of course, any resulting function can also be checked via equation 7 to see if it is the desired evaluation function. Although this method of successive approximations generally only approximates the actual evaluation function, in what follows we loosely refer to the result of this computation as the actual evaluation function.
10. See Ross 1983 for a detailed discussion and proof of these results. If there is a finite number of states and actions, as we are assuming throughout this chapter, this method converges after a finite number of iterations.
11. In adaptive control, where the modeling process is called *system identification*, the functional relationship being modeled is a dynamical system which is usually more complicated than a function from input to output (owing to the influence of internal states—see Goodwin and Sin 1984). For our purposes, however, it suffices to discuss only the modeling of input-output functions.
12. See Goodwin and Sin 1984. Their discussion of these issues forms the basis for our remarks.
13. For some applications, it is common to let some, or all, measurements produce only two values to indicate, for example, whether a logical predicate applied to the object or state is true or false. This is a special case of the framework adopted here.
14. Even in the case of a linear model, the model output can be a nonlinear function of input patterns, because the ϕ functions can be nonlinear functions of the patterns. But because these functions are not parameterized, they do not directly enter into the derivation of a parameter-estimation method.
15. This is the special case in which each entry in M_t is equal β for each t.
16. If one replaces the linear model of equation 15 with the linear threshold decision rule given by expression 16, one obtains the perceptron learning rule of Rosenblatt (1961), which is identical to equation 21 except that the weighted sum $v_t^T\phi_t$ is replaced by the threshold of the weighted sum as defined by expression 16. These correspondences are explained in more detail in Barto 1985, Duda and Hart 1973, and Sutton and Barto 1981.
17. A function, f, from an n-dimensional space to the real numbers can be viewed as a surface. If x is a point in the n-dimensional space, then the gradient of f at x is the vector

$$\left(\frac{\partial f}{\partial x_1}(x), \ldots, \frac{\partial f}{\partial x_n}(x)\right)^T,$$

where

$$\frac{\partial f}{\partial x_i}(x)$$

is the partial derivative of f with respect the ith dimension evaluated at the point x. This vector points in the direction of the steepest increase of the surface at the point x.
18. The error back-propagation method (Rumelhart et al. 1986) is derived by computing this error gradient for a particular class of models in the form of layered connectionist networks.
19. The model considered here is a model of the evaluation function, not of the decision task; that is, it does not model the state-transition and payoff probabilities of the system underlying the decision task.

20. The resulting parameter update rule would be

$$v_{t+1} = v_t + \beta[r_{t+1} + \gamma V_t(x_{t+1}) - V_t(x_t)]\,(\phi_t - \gamma\phi_{t+1}),$$

which is obtained by differentiating expression 30 with respect to v_t. See Werbos (in preparation).

21. When $\gamma = 0$ (the case of the TD procedure equivalent to a one-step-ahead LMS predictor), the leftmost "Evaluation Function Model" box for each time step becomes disconnected, and the flow of computation is identical to that shown in figure 4 for the task of modeling an unknown function whose true evaluations become available to the estimation process in one time step.

22. The procedure used by Barto et al. (1983) for the "Adaptive Critic Element" in their pole balancer differs from the TD procedure described here in that it computes a single evaluation for each feature vector at each time step. In terms of figure 5, the leftmost "Evaluation Function Model" for each time step is missing and the result of the rightmost computation is fed leftward into the summation unit. If changes in parameter values remain small, this is a reasonable approximation to the TD procedure described here, but it can become unstable if the parameter values change rapidly (Sutton 1984, 1988). The Adaptive Critic Element also uses stimulus traces as in the TD model of conditioning described by Sutton and Barto in chapter 12 of this volume.

23. Setting $v_{97} = 0$ in the linear model does not limit the model's ability to represent arbitrary evaluation functions of the 96 states, but it does prevent a simple form of generalization in which v_{97} might be adjusted to equal, for example, the average state evaluation.

24. A lookup-table representation of a function is simply a list of its values. To evaluate the function for a particular item in its domain, one simply accesses the entry in the place in the table corresponding to that item. Here, $\phi(x)$ can be thought of as accessing the place where v_x is stored.

25. Much of the study of parameter estimation, especially in the context of connectionist networks, is concerned with the problem of finding representations and classes of models that facilitate useful forms of transfer or extrapolation, often called generalization. See, of example, Anderson and Rosenfeld 1988; Hinton and Anderson 1981; McClelland and Rumelhart 1986; Rumelhart and McClelland 1986. Most of these methods for representation and modeling obviously can be used to advantage with the TD procedure in the route-finding example, but the lookup-table representation allows us to address in a simple form the issues most relevant to the focus of this chapter. Some of the issues that arise when more complicated representations are used in sequential decision tasks are discussed in Sutton 1984, and examples involving other representations are provided by Anderson's (1987) use of layered networks in the pole-balancing problem and Watkin's (1989) use of the representation method proposed by Albus (1979) in a model of the cerebellum.

26. Sutton (1984) calls this the "reinforcement comparison" approach to reinforcement learning.

27. Some of the rules for adjusting stochastic policies have been inspired by the theory of stochastic learning automata developed by cyberneticians and engineers (Narendra and Thathachar 1989; Tsetlin 1973). A precursor of this theory is the statistical learning theory developed by psychologists (Bush and Mosteller 1955; Estes 1950). Barto and Anandan (1985) and Barto (1985, 1989) discuss a learning rule of this kind called the Associative Reward/Penalty (A_{R-P}) rule. Sutton (1984), Anderson (1986, 1987), and Gullapalli (1988) describe the results of computer simulations of related methods. Williams (1986, 1987) provides theoretical analysis of a more general class of stochastic learning rules. There are other approaches to reinforcement learning that we do not

address at all in this chapter. These include the estimation of evaluations of state-action pairs instead of just states (Watkins 1989), the computation of evaluation gradients using the model of the evaluation function (Munro 1987); Werbos 1987, 1988; Robinson and Fallside 1987; Williams 1988, and the use of systematic, instead of stochastic, variation in activity.

References

Albus, J. S. (1979) Mechanisms of planning and problem solving in the brain. *Mathematical Biosciences* 45: 247–293.

Anderson, C. W. (1986) Learning and Problem Solving with Multilayer Connectionist Systems. Ph.D. thesis, University of Massachusetts, Amherst.

Anderson, C. W. (1987) Strategy Learning with Multilayer Connectionist Representations. Technical report TR87-509.3, GTE Laboratories, Inc., Waltham, Mass. This is a corrected version of the report published in *Proceedings of the Fourth International Workshop on Machine Learning* (Morgan Kaufmann).

Anderson, J. A., and Rosenfeld, E., editors (1988) *Neurocomputing: Foundations of Research*. MIT Press.

Barto, A. G. Connectionist learning for control: An overview. In Miller, T., Sutton, R. S., and Werbos, P. J., editors, *Neural Networks for Control* (MIT Press, forthcoming).

Barto, A. G. (1985) Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology* 4: 229–256.

Barto, A. G. (1989) From chemotaxis to cooperativity: Abstract exercises in neuronal learning strategies. In Durbin, R., Maill, R., and Mitchison, G., editors. *The Computing Neuron* (Addison-Wesley).

Barto, A. G., and Anandan, P. (1985) Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics* 15: 360–375.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983) Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics* 13: 835–846. Reprinted in J. A. Anderson and E. Rosenfeld, eds., *Neurocomputing: Foundations of Research* (MIT Press, 1988).

Bellman, R. E. (1957) *Dynamic Programming*. Princeton University Press.

Booker, L. B. (1982) Intelligent Behavior as an Adaptation to the Task Environment. Ph.D. thesis, University of Michigan, Ann Arbor.

Borkar, V., and Varaiya, P. (1979) Adaptive control of Markov chains I: Finite parameter set. *IEEE Transactions on Automatic Control* 24: 953–957.

Bush, R. R., and Mosteller, F. (1955) *Stochastic Models for Learning*. Wiley.

Dickinson, A. (1980) *Contemporary Animal Learning Theory*. Cambridge University Press.

Dreyfus, S. E., and Law, A. M. (1977) *The Art and Theory of Dynamic Programming*. Academic Press.

Duda, R. O., and Hart, P. E. (1973) *Pattern Classification and Scene Analysis*. Wiley.

El-Fattah, Y. (1981) Recursive algorithms for adaptive control of finite Markov chains. *IEEE Transactions on Systems, Man, and Cybernetics* 11: 135–144.

Estes, W. K. (1950) Toward a statistical theory of learning. *Psychological Review* 57: 94–107.

Feldman, J. A., and Ballard, D. H. (1982) Connectionist models and their properties. *Cognitive Science* 6: 205–254.

Goodwin, G. C., and Sin, K. S. (1984) *Adaptive Filtering Prediction and Control*. Prentice-Hall.

Gullapalli, V. (1988) A stochastic algorithm for learning real-valued functions via reinforcement feedback. Technical report 88-91, University of Massachusetts, Amherst.

Hampson, S. E. (1983) A Neural Model of Adaptive Behavior. Ph.D. thesis, University of California, Irvine.

Hinton, G. E., and Anderson, J. A., editors (1981) *Parallel Models of Associative Memory.* Erlbaum.

Holland, J. H. (1986) Escaping brittleness: The possibility of general-purpose learning algorithms applied to rule-based systems. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2 (Morgan Kaufmann).

Houston, A., Clark, C., McNamara, J., and Mangel, M. (1988) Dynamic model in behavioral and evolutionary ecology. *Nature* 332: 29–34.

Howard, R. (1960) *Dynamic Programming and Markov Processes.* MIT Press.

Klopf, A. H. (1972) Brain function and adaptive systems—A heterostatic theory. Technical report AFCRL-72-0164, Air Force Cambridge Research Laboratories, Bedford, Mass. A summary appears in *Proceedings of the International Conference on Systems, Man, and Cybernetics*, 1974 (IEEE Systems, Man, and Cybernetics Society, Dallas).

Klopf, A. H. (1982) *The Hedonistic Neuron: A Theory of Memory, Learning, and Intelligence.* Hemisphere.

Krebs, J. R., Kacelnik, A., and Taylor, P. (1978) Test of optimal sampling by foraging great tits. *Nature* 275: 2M–31.

Kumar, P. R., and Lin, W. (1982) Optimal adaptive controllers for unknown Markov chains. *IEEE Transactions on Automatic Control* 25: 765–774.

Lakshmivarahan, S. (1981) *Learning Algorithms and Applications.* Springer-Verlag.

Liepins, G. E., Hilliard, M. R., and Palmer, M. Credit assignment and discovery in classifier systems. *International Journal of Intelligent Systems* (in press).

Ljung, L., and Söderstom, T. (1983) *Theory and Practice of Recursive Identification.* MIT Press.

Lyubchik, L. M., and Poznyak, A. S. (1974) Learning automata in stochastic plant control problems. *Automatic and Remote Control (USSR)* 35: 777–789.

Mandl, P. (1974) Estimation and control in Markov chains. *Advances in Applied Probability* 6: 40–60.

Mangel, M., and Clark, C. W. (1988) *Dynamic Modeling in Behavioral Ecology.* Princeton University Press.

McClelland, J. L., et al. (1986) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2: *Applications.* MIT Press.

Mendel, J. M., and McLaren, R. W. (1970) Reinforcement learning control and pattern recognition systems. In Mendel, J. M., and Fu, K. S., editors, *Adaptive Learning and Pattern Recognition Systems: Theory and Applications* (Academic Press).

Michie, D., and Chambers, R. A. (1968) BOXES: An experiment in adaptive control. In Dale, E., and Michie, D., editors, *Machine Intelligence 2* (Oliver and Boyd).

Minsky, M. L. (1954) Theory of Neural-Analog Reinforcement Systems and its Application to the Brain-Model Problem. Ph.D. thesis, Princeton University.

Minsky, M. L. (1961) Steps toward artificial intelligence. *Proceedings of the Institute of Radio Engineers* 49: 8–30. Reprinted in E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought* (McGraw-Hill).

Munro, P. (1987) A dual back-propagation scheme for scalar reward learning. In *Proccedings of the Ninth Annual Conference of the Cognitive Science Society* (Erlbaum).

Narendra, K., and Thathachar, M. A. L. (1989) *Learning Automata: An Introduction.* Prentice-Hall.

Pearl, J. (1984) *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley.

Rachlin, H., Battalio, R., Kagel, J., and Green, L. (1981) Maximization theory in behavioral psychology. *Behavioral and Brain Sciences* 4: 371–417.

Rescorla, R. A. (1987) A Pavlovian analysis of goal-directed behavior. *American Psychologist* 42: 119–129.

Rescorla, R. A., and Wagner, A. R. (1972) A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. In Black, A. H., and Prokasy, W. F., editors, *Classical Conditioning II* (Appleton-Century-Crofts).

Riordon, J. S. (1969) An adaptive automaton controller for discrete-time Markov processes. *Automatica* 5: 721–730.

Robinson, A. J., and Fallside, F. (1987) The Utility Driven Dynamic Error Propagation Network. Technical report CUED/F-INFENG/TR.1, Cambridge University Engineering Department.

Rosenblatt, F. (1961) *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books.

Ross, S. (1983) *Introduction to Stochastic Dynamic Programming*. Academic Press.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986) Learning internal representations by error propagation. In Rumelhart, D. E., et al. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1: *Foundations* (MIT Press).

Rumelhart, D. E., et al. (1986) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1: *Foundations*. MIT Press.

Samuel, A. L. (1959) Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, pp. 210–229. Reprinted in E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought* (McGraw-Hill, 1963).

Samuel, A. L. (1967) Some studies in machine learning using the game of checkers. II—Recent progress. *IBM Journal on Research and Development*, pp. 601–617.

Sato, M., Abe, K., and Takeda, H. (1982) Learning control of finite Markov chains with unknown transition probabilities. *IEEE Transactions on Automatic Control* 27: 502–505.

Selfridge, O., Sutton, R. S., and Barto, A. G. (1985) Training and tracking in robotics. In Joshi, A., editor, *Proceedings of the Ninth International Joint Conference of Artificial Intelligence* (Morgan Kaufmann).

Sklansky, J., and Wassel, G. N. (1981) *Pattern Classifiers and Trainable Machines*. Springer-Verlag.

Staddon, J. E. R. (1980) Optimality analyses of operant behavior and their relation to optimal foraging. In Staddon, J. E. R., editor, *Limits to Action: The Allocation of Individual Behavior* (Academic Press).

Sutton, R. S. (1984) *Temporal Credit Assignment in Reinforcement Learning*. Ph.D. thesis, University of Massachusetts, Amherst.

Sutton, R. S. (1988) Learning to predict by the methods of temporal differences. *Machine Learning* 3: 9–44.

Sutton, R. S., and Barto, A. G. (1981) Toward a moden theory of adaptive networks: Expectation and prediction. *Psychological Review* 88: 135–171.

Sutton, R. S., and Barto, A. G. (1987) A temporal-difference model of classical conditioning. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society* (Erlbaum).

Thorndike, E. L. (1911) *Animal Intelligence*. Darien, Conn.: Hafner.

Tsetlin, M. L. (1973) *Automaton Theory and Modeling of Biological Systems*. Academic Press.

Watkins, C. J. C. H. (1989) Learning from Delayed Rewards. Ph.D. thesis, Cambridge University.

Werbos, P. J. (1977) Advanced forecasting methods for global crisis warning and models of intelligence. *General Systems Yearbook* 22: 25–38.

Werbos, P. J. (1987) Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *IEEE Transactions on Systems, Man, and Cybernetics* 17: 7–20.

Werbos, P. J. (1988) Generalization of back propagation with applications to a recurrent gas market model. *Neural Networks* 1: 339–356.

Werbos, P. J. (1990) Consistency of HDP applied to simple reinforcement learning problem. *Neural Networks* 3: 179–189.

Wheeler, R. M., and Narendra, K. S. (1986) Decentralized learning in finite Markov chains. *IEEE Transactions on Automatic Control* 31: 519–526.

Widrow, B., and Hoff, M. E. (1960) Adaptive switching circuits. In *1960 WESCON Convention Record*. Reprinted in J. A. Anderson and E. Rosenfeld, eds., *Neurocomputing: Foundations of Research* (MIT Press, 1988).

Widrow, B., and Stearns, S. D. (1985) *Adaptive Signal Processing*. Prentice-Hall.

Williams, R. J. (1986) Reinforcement learning in connectionist networks: A mathematical analysis. Technical report 8605, Institute for Cognitive Science, University of California of San Diego.

Williams, R. J. (1987) Reinforcement-learning connectionist system. Technical report 87-3, College of Computer Science, Northeastern University.

Williams, R. J. (1988) On the use of backpropagation in associative reinforcement learning. In *Proceedings of the IEEE International Conference on Neural Networks*, San Diego.

Witten, I. H. (1977) An adaptive optimal controller for discrete-time Markov environments. *Information and Control* 34: 286–295.