Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control. James C. Spall
Copyright © 2003 John Wiley & Sons, Inc.,
ISBN: 0-471-33052-3

CHAPTER 11

REINFORCEMENT LEARNING VIA TEMPORAL DIFFERENCES

The fields of artificial intelligence, computer science, and control have been a source of numerous methods in stochastic search and optimization. The evolutionary algorithms of Chapters 9 and 10 are one such class of methods. We continue in this vein here by describing an approach in the general area of "learning." At some level, learning is devoted to the broadest of issues in mathematical modeling and understanding from data: How can one best learn about a system from data? Chapter 13 also deals with some aspects of this question by addressing the tradeoff in bias and variance for a model, the choice of model form, and the quantification of the information available for estimation through the (Fisher) information matrix.

This chapter takes the model form as a given and describes methods in reinforcement (trial and error) learning. The focus in this chapter is one important instance of reinforcement learning—the temporal difference method. This algorithm is useful for prediction problems arising in, say, machine learning, artificial intelligence, forecasting, and optimal control. As the name would suggest, this method is based on information available from certain differences over time (with these differences related to model predictions or actual system performance). The reader should be warned that the results here represent only a sliver of the available results in general learning.

Section 11.1 gives some general background on reinforcement learning. Section 11.2 connects reinforcement learning to the idea of temporal differences and Section 11.3 presents the basic temporal difference algorithm for parameter estimation. Section 11.4 discusses extensions of the temporal difference algorithm to batch and online implementations. Section 11.5 presents some examples of the temporal difference algorithm and Section 11.6 makes some connections to the root-finding stochastic approximation framework of Chapters 4 and 5. Section 11.7 offers some concluding remarks.

11.1 INTRODUCTION

The term *learning* has been used in a number of fields to refer to computational approaches to understanding input–output relationships in a system of interest. Learning takes place when input–output data are collected on the system and

processed in a meaningful way with an appropriate algorithm. This, of course, does not differ fundamentally from the aim of other algorithms studied in this book. Nevertheless, the term *learning* has often been associated with algorithms popularized in the fields of computer science and artificial intelligence. In fact, the term *machine learning* is often used as a synonym for learning, motivating the computer science connection even more strongly. Certainly, statisticians have been attacking learning problems for as long as the field of statistics has existed, although the terminology and focus differ somewhat from those in computer science and related areas. For a variety of historical and technical reasons, areas such as data mining, system identification, supervised and unsupervised training (as for a neural network), and pattern recognition—all of which heavily involve statistical notions—have not been the traditional province of statisticians.

Reinforcement learning is a fundamental set of methods within the broader class of learning approaches. Reinforcement learning is a formal means of doing trial-and-error learning. Of course, most members of the animal kingdom (including humans) implement trial and error without the formalized structure of a mathematical learning algorithm. In developing a formal algorithmic implementation of this principle, one aims for an approach satisfying the following:

> Reinforcement learning is based on the common-sense idea that if an action is followed by a satisfactory state of affairs, or by an improvement in the state of affairs (as determined in some clearly defined way), then the tendency to produce that action is strengthened (i.e., reinforced) (Sutton et al., 1992).

Conversely, an undesirable—or worsening—result leads to a weakened tendency to follow the action.

Within the field of machine learning, the reinforcement learning principle is used in general artificial intelligence applications through tools such as computational agents, reinforcement signals, and learning agents. We do not need the full generality of these tools here as our goals are focused on the more specific (familiar) problem of minimizing a loss function $L(\boldsymbol{\theta})$. In particular, we will not consider rule-based or symbolic estimation and prediction; rather, we focus on the familiar problem of *numerical* estimation. Some specific reinforcement learning methods include adaptive critic methods (Barto, 1992), dynamic programming (Bertsekas, 1995a), Q-learning (Watkins and Dayan, 1992), and temporal difference learning. The latter of these is considered in the remainder of this chapter.

The reader interested in more general principles of learning may wish to consult Russell and Norvig (1995, Chaps. 18–21), Jang et al. (1997), Mitchell (1997), Cherkassky and Mulier (1998), Michalski et al. (1998), and Sutton and Barto (1998).

11.2 DELAYED REINFORCEMENT AND FORMULATION FOR TEMPORAL DIFFERENCE LEARNING

One of the major settings for machine learning is the delayed reinforcement problem. The prototype form for this problem involves a sequence of actions, with no feedback until after the last action as to whether the sequence was good or not.

As an illustration of this problem, suppose that a sequence of estimates of one quantity is formed over time, and only after the full sequence is formed is the true value of the quantity being estimated revealed. Hence, for all but the last estimate in the sequence, there is a delay in determining if a particular estimate is "good" in the sense that at least one intervening estimate must be formed. In particular, suppose that a scalar random outcome Z occurs at some time in the future, and that prior to observing Z, there exists a sequence of $n + 1$ predictions $\hat{z}_0, \hat{z}_1, \ldots, \hat{z}_n$, where $\hat{z}_0$ occurs before $\hat{z}_1$, $\hat{z}_1$ occurs before $\hat{z}_2$, and so on, and $\hat{z}_n$ is the final prediction before the event yielding Z. As we will see below, it is also convenient to define a "null" prediction $\hat{z}_{n+1} = Z$. Figure 11.1 depicts the process.

This delayed reinforcement problem is pervasive. Sutton (1988) discusses weather forecasting as one example. Suppose that for each week, one is interested in issuing a forecast on each day of the week for the weather on the following Saturday. The input information for each day's forecast would be current and recent past weather and other meteorological indicators. Conventional training algorithms (e.g., "supervised learning" as in the backpropagation algorithm of Section 5.2) perform the parameter estimation based on comparisons of predictions and actual outcomes. That is, in developing the function for predicting Saturday's weather on Wednesday, one would compare a sequence of Wednesday predictions and Saturday outcomes. The temporal difference (TD) learning procedure, on the other hand, builds up its parameter estimates by looking at successive predictions. For example, if Wednesday's prediction is for a 50 percent chance of rain on Saturday and Thursday's prediction is for a 75 percent chance, then subsequent days similar to Wednesday will have a prediction greater than 50 percent. The intuitive rationale for this adjustment is that predictions closer to the actual event will be more reliable.

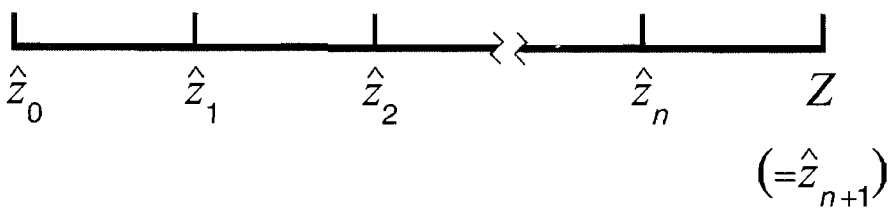

Figure 11.1. Relationship between the predictions $\hat{z}_0, \hat{z}_1, \ldots, \hat{z}_n$ and event Z in the delayed reinforcement problem. Time is depicted as moving left to right.

Many other examples of delayed reinforcement exist. For instance, when humans see or hear something, they receive a stream of input, constantly updating their perception of the object(s) being seen or heard. Humans typically do not wait until they have a final resolution of the nature of the object to update their perception of the object. Rather, a person takes the stream of input together with the previous predictions to continuously (or periodically) recalibrate the prediction of the nature of the object. Similarly, a business selling a new product does not wait until the end of the fiscal year to determine if the product is a success; the assessment of the product is made continuously from the product's introduction based on current input and past predictions. The expected outcome of a chess game or golf match is similarly updated throughout the competition.

The aim here is to construct a function for producing the predictions $\hat{z}_0$, $\hat{z}_1$, ..., $\hat{z}_n$. As with any such function, there are adjustable parameters ($\boldsymbol{\theta}$) to be estimated. Following the general regression framework of Section 5.1, let $\hat{z}_\tau = h_\tau(\boldsymbol{\theta}, \boldsymbol{x}_\tau)$ where $h_\tau(\cdot)$ is a (possibly nonlinear) regression function and $\boldsymbol{x}_\tau$ is a vector of input observations available at time τ to be used in forming the prediction. The subscript τ is being used to emphasize time (rather than the more generic k in previous chapters). Section 5.2 focused on the common case where the regression function is a time-invariant neural network (i.e., $h(\cdot) = h_\tau(\cdot)$). We will, however, consider the time-*varying* form here (see, e.g., Examples 11.1 and 11.2 in Section 11.5).

There is a countless variety of general forms for the regression functions. Chapter 3 considered linear regression and Section 5.2 considered a notable nonlinear case—neural networks—where $\boldsymbol{\theta}$ represents the connection weights. Suppose that the functional forms of the prediction functions have been fixed (e.g., feedforward NNs with a specified number of hidden layers and nodes per layer). Hence, we must cope with the problem of the delayed response in estimating the unknown parameters $\boldsymbol{\theta}$. Note the contrast with conventional supervised learning, such as backpropagation as seen in Section 5.2. In supervised learning, the algorithm compares a predicted output $\hat{z}_\tau$ directly with an actual output z_τ, using the difference $z_\tau - \hat{z}_\tau$ in the training process of updating the parameters in the prediction function. For each $\hat{z}_\tau$, there is no significant delay in the sense that there are no subsequent predictions of z_τ that arrive before forming the comparison of $\hat{z}_\tau$ and z_τ for use in training $\boldsymbol{\theta}$. This contrasts with the delayed reinforcement setting for TD learning.

The essential characteristic of the TD algorithm is that, rather than use the difference between predictions ($\hat{z}_\tau$) and the actual outcome (Z), it uses the temporally successive collection of predictions $\hat{z}_{\tau+1} - \hat{z}_\tau$. Samuel (1959) used the idea in the problem of automated checkers playing; Sutton (1988) provided a more general setting and coined the term *temporal difference learning*. The TD method applies to multistep prediction problems, and is a prominent example of a delayed reinforcement algorithm. TD learning is closely connected to another

form of reinforcement learning called *Q-learning*, which itself is a type of stochastic dynamic programming. The connections between TD learning, *Q*-learning, and dynamic programming are explored, for example, in Jaakkola et al. (1994) and Sutton and Barto (1998, Chaps. 4–7).

Further, there are strong connections of these learning methods to problems in control. In control, one attempts to find an optimal policy based on predictions of future system behavior. Sutton et al. (1992), Jaakkola et al. (1994), and Sutton and Barto (1998, Chaps. 4–7) are among the many references exploring the connection of reinforcement learning methods to control. In particular, TD has a strong connection to control-oriented methods such as dynamic programming where one is trying to pick a sequence of optimal policies to cause a delayed reinforcement system to behave acceptably until some future termination event occurs (an excellent overview of dynamic programming is Bertsekas, 1995a). This chapter, however, does not dwell on the control aspects of TD. The control-oriented implementations rely extensively on the methods for the straight prediction problems considered here.

Why should one use TD? First, of course, TD is especially appropriate in the delayed reinforcement problem, a setting not explicitly considered in the other approaches of this book. Further, as shown in the sections to follow, TD methods are suitable for exploiting prior information about the behavior of a dynamic system. The prior information is embedded in past predictions (and the knowledge used to create the past predictions). TD uses this prior information in forming current and future predictions.

In contrast, traditional training methods for prediction modeling (e.g., the supervised learning of backpropagation—see Section 5.2) use only the input variables and the outcomes (Z), ignoring the predictions that contain useful prior information about the process. Further, the supervised learning methods must wait until an outcome is observed before an update is possible to $\boldsymbol{\theta}$, which can delay learning considerably. In contrast, some versions of TD (e.g., the online version in Section 11.4) allow for updates at every time step. Finally, although there is apparently no theoretical comparison of efficiency of TD to other learning methods (even in the linear problems where TD is known to converge), numerical evidence suggests its superiority in a range of prediction problems (e.g., Sutton and Barto, 1998, pp. 138–139).

As with any other method, of course, TD has its drawbacks as well. Among the relative shortcomings is the weaker convergence theory for TD in comparison to the stochastic gradient methods for supervised learning (Chapter 5). While root-finding (Robbins–Monro) SA can be used to characterize the convergence of both TD and the stochastic gradient methods, the conditions for TD are more stringent than the conditions for the stochastic gradient methods. Most important, as discussed in Section 11.6, the current conditions for convergence of TD are directed toward models that are *linear* in $\boldsymbol{\theta}$; no such restriction exists for stochastic gradient methods such as backpropagation for neural networks, which are fully justified for nonlinear models. (Despite the lack of formal theory, TD methods are sometimes *used*, quite successfully, in

nonlinear systems.) There are also published counterexamples to TD convergence in linear and nonlinear models (e.g., Bertsekas, 1995b; Tsitsiklis and Van Roy, 1997). TD is not likely to perform as well as supervised learning when a given set of inputs (the $\boldsymbol{x}_\tau$) will repeatedly produce the *same* outcome Z (i.e., there is no noise in the process). In this relatively rare noise-free setting, there is no better source of information about the process than the observed Z.

11.3 BASIC TEMPORAL DIFFERENCE ALGORITHM

The fundamental problem is to train a predictor of future outcomes from one or more "trial" sequences of inputs and outputs $\{\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_n; Z\}$. This training process estimates the parameters $\boldsymbol{\theta}$ appearing in the prediction functions $h_\tau(\cdot)$, $\tau = 0, 1, \ldots, n$. The final $\boldsymbol{\theta}$ estimate will be used with the predictors $h_\tau(\cdot)$ as they apply in future problems having their own inputs ($\boldsymbol{x}_\tau$) and outputs (Z). There are several ways in which TD learning for updating $\boldsymbol{\theta}$ can be implemented. Let us first assume that $\boldsymbol{\theta}$ is updated only once after one full sequence of measurements $\{\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_n; Z\}$. We call this the *basic TD algorithm* (this form is the focus of Sutton, 1988). Alternative batch and online methods for TD are discussed in the next section. The basic TD method is a special case of batch TD, with the number of batches being one.

As mentioned above, all predictions during the training in basic TD are based on a fixed $\boldsymbol{\theta}$, taken as an initial estimate $\hat{\boldsymbol{\theta}}_0$. So, $\hat{z}_\tau = h_\tau(\hat{\boldsymbol{\theta}}_0, \boldsymbol{x}_\tau)$ for all $\tau \leq n$ here. After passing through the sequence $\{\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_n; Z\}$, basic TD produces a new value of $\boldsymbol{\theta}$, say $\hat{\boldsymbol{\theta}}_{\text{new}}$. The new value of $\boldsymbol{\theta}$ is related to the initial value of $\boldsymbol{\theta}$ by the following equation:

$$\hat{\boldsymbol{\theta}}_{\text{new}} = \hat{\boldsymbol{\theta}}_0 + \sum_{\tau=0}^{n} (\delta\boldsymbol{\theta})_\tau , \tag{11.1}$$

where $(\delta\boldsymbol{\theta})_\tau$ is an increment in $\boldsymbol{\theta}$ computed based on the difference in predictions $\hat{z}_{\tau+1} - \hat{z}_\tau$ as discussed below. So the aim is to calculate the sequence of increments, $(\delta\boldsymbol{\theta})_\tau$, by which (11.1) produces a new parameter value.

The stochastic gradient algorithm in Section 5.1 forms the starting point for deriving the increments $(\delta\boldsymbol{\theta})_\tau$. Recall that the stochastic gradient algorithm was applied to stochastic nonlinear regression problems. Let us consider the standard mean-squared-error criterion, $\frac{1}{2}E\{[Z - h_\tau(\boldsymbol{\theta}, \boldsymbol{x}_\tau)]^2\}$. From the basic stochastic gradient algorithm in Section 5.1, an updated value of $\boldsymbol{\theta}$ is produced by adding the quantity below to the original value $\hat{\boldsymbol{\theta}}_0$:

$$a(Z - \hat{z}_\tau)\left[\frac{\partial h_\tau(\boldsymbol{\theta}, \boldsymbol{x}_\tau)}{\partial \boldsymbol{\theta}}\right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_0} , \tag{11.2}$$

where a is the familiar nonnegative gain coefficient, usually taken as a constant across τ in applications here. This contrasts with the indexed form of gain coefficient considered in most of Chapters 4–7. We will, however, discuss an indexed form in Section 11.6 in making some connections to stochastic approximation. Expression (11.2) arises in the traditional supervised learning setting where $\boldsymbol{\theta}$ is updated based on the outcome of a process. Note that (11.2) cannot be implemented prior to observing the final outcome because it requires that Z be available. This is where TD comes to the fore. The TD procedure *can* be implemented in transitioning between time points prior to the final outcome.

Let us now derive the form for the TD increments $(\delta\boldsymbol{\theta})_\tau$. Following the framework of (11.1), an update of $\boldsymbol{\theta}$ is available by summing the stochastic gradient-based increments in (11.2) over $0 \le \tau \le n$, yielding

$$\hat{\boldsymbol{\theta}}_{\text{new}} = \hat{\boldsymbol{\theta}}_0 + a\sum_{\tau=0}^{n}\left[(Z-\hat{z}_\tau)\frac{\partial h_\tau(\boldsymbol{\theta}, \boldsymbol{x}_\tau)}{\partial\boldsymbol{\theta}}\right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_0}. \tag{11.3}$$

It is possible to rewrite $Z - \hat{z}_\tau$ in (11.3) as a telescoping sum involving the full set of predictions beginning at time τ:

$$Z - \hat{z}_\tau = \sum_{i=\tau}^{n}(\hat{z}_{i+1} - \hat{z}_i) \tag{11.4}$$

(recall that $\hat{z}_{n+1} = Z$). Substituting (11.4) into (11.3) and rearranging the double sums (Exercise 11.1) yields

$$\hat{\boldsymbol{\theta}}_{\text{new}} = \hat{\boldsymbol{\theta}}_0 + a\sum_{\tau=0}^{n}(\hat{z}_{\tau+1} - \hat{z}_\tau)\sum_{i=0}^{\tau}\left[\frac{\partial h_i(\boldsymbol{\theta}, \boldsymbol{x}_i)}{\partial\boldsymbol{\theta}}\right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_0}. \tag{11.5}$$

From (11.5), one can peel off the incremental change $(\delta\boldsymbol{\theta})_\tau$ to be used in (11.1). In particular,

$$(\delta\boldsymbol{\theta})_\tau = a(\hat{z}_{\tau+1} - \hat{z}_\tau)\sum_{i=0}^{\tau}\left[\frac{\partial h_i(\boldsymbol{\theta}, \boldsymbol{x}_i)}{\partial\boldsymbol{\theta}}\right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_0} \tag{11.6}$$

for $\tau = 0, 1, \ldots, n$. Eqn. (11.6) represents a special case of the TD learning algorithm (the TD(1) algorithm according to the notation below). Unlike (11.2), which requires Z at all values of τ, the incremental update in (11.6) depends only on the difference of predictions (at the parameter value $\hat{\boldsymbol{\theta}}_0$) and the sum of the gradients $\partial h_i/\partial\boldsymbol{\theta}$ at the various $\boldsymbol{x}_i$ inputs. Hence, (11.6) can be computed without Z until the final increment at $\tau = n$.

There is an obvious generalization of expression (11.6) for estimating the increment $(\delta\boldsymbol{\theta})_\tau$ to allow for more weight to be assigned to more recent system measurements. This generalization is called the TD(λ) family of learning algorithms. In particular, we might expect that past information (past $\boldsymbol{x}_i$) should be given less weight than more recent information in computing a current increment to $\boldsymbol{\theta}$. Suppose that we damp past information according to geometric decay, with $0 \leq \lambda \leq 1$ being the decay factor. Then, the sum on the right-hand side of (11.6) can be replaced with a weighted sum according to

$$(\delta\boldsymbol{\theta})_\tau = a(\hat{z}_{\tau+1} - \hat{z}_\tau)\sum_{i=0}^{\tau}\lambda^{\tau-i}\left[\frac{\partial h_i(\boldsymbol{\theta}, \boldsymbol{x}_i)}{\partial\boldsymbol{\theta}}\right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_0}. \tag{11.7}$$

Eqn. (11.7) is called the *TD(λ) learning rule*. Together with (11.1), this rule can be used to generate an updated $\boldsymbol{\theta}$ value, $\hat{\boldsymbol{\theta}}_{\text{new}}$. For computational purposes, it may be convenient to accumulate the sum on the right-hand side of (11.7). In particular, for $\tau \geq 0$, let

$$\boldsymbol{s}_{\tau+1} = \lambda\boldsymbol{s}_\tau + \left[\frac{\partial h_{\tau+1}(\boldsymbol{\theta}, \boldsymbol{x}_{\tau+1})}{\partial\boldsymbol{\theta}}\right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_0},$$

where $\boldsymbol{s}_0 = \left[\partial h_0(\boldsymbol{\theta}, \boldsymbol{x}_0)/\partial\boldsymbol{\theta}\right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_0}$. Then, (11.7) can be written as

$$(\delta\boldsymbol{\theta})_\tau = a(\hat{z}_{\tau+1} - \hat{z}_\tau)\boldsymbol{s}_\tau. \tag{11.8}$$

The extreme versions, TD(0) and TD(1), are important special cases, with TD(0) being defined using the convention $0^0 = 1$. In the TD(0) algorithm, only the most recent measurement is used:

$$\text{TD(0): } (\delta\boldsymbol{\theta})_\tau = a(\hat{z}_{\tau+1} - \hat{z}_\tau)\left[\frac{\partial h_\tau(\boldsymbol{\theta}, \boldsymbol{x}_\tau)}{\partial\boldsymbol{\theta}}\right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_0}.$$

In the TD(1) algorithm, all the past prediction gradients are weighted equally:

$$\text{TD(1): See eqn. (11.6).}$$

The TD(0) algorithm, where only the most recent prediction affects the update to $\boldsymbol{\theta}$, corresponds to a conventional *dynamic programming* formulation for stagewise optimization. Dynamic programming is important in stochastic control and learning theory, with entire books devoted to the subject (e.g., Bertsekas, 1995a). The TD(1) algorithm, at the other extreme, assigns equal weight to all previous gradient evaluations. This implies the equivalence of TD(1) to supervised learning in the following sense.

From (11.3)—which is equivalent to the TD(1) substitution of $(\delta\theta)_\tau$ appearing in (11.6) into (11.1)—the update from $\hat{\theta}_0$ to $\hat{\theta}_{new}$ is the same as the sum of stochastic gradient increments in (11.2). Further, the sum of the stochastic gradient increments is the solution corresponding to *one step* of a supervised learning algorithm where the loss function is the sum of mean-squared errors:

$$L(\theta) = \frac{1}{2}\sum_{\tau=0}^{n} E\left\{[Z - h_\tau(\theta, x_\tau)]^2\right\}. \tag{11.9}$$

(This supervised learning approach is directly comparable to the solution associated with the batch stochastic gradient form in Subsection 5.1.3 with the exception that there is a *common* output Z here—the delayed response—as opposed to the separate responses for each input in conventional stochastic gradient implementations, as in Chapter 5.) In particular, the TD(1) algorithm ((11.6) into (11.1)) and one step of the supervised learning algorithm applied to $L(\theta)$ above yield the same update to θ (from $\hat{\theta}_0$ to $\hat{\theta}_{new}$). On the other hand, the individual TD(1) increments $(\delta\theta)_\tau$ from (11.6) are *not* the same as the stochastic gradient increments in (11.2) (why?)[1], although the *sums* of the increments from $\tau = 0$ to $\tau = n$ ((11.1) for TD(1); (11.3) for stochastic gradient) are by construction identical. The fundamental difference in implementation is that the supervised learning formulation requires the final outcome (Z) for every increment, while TD(1) needs Z only at the final increment. The latter is useful in real-time applications, as it allows the θ increments to be computed in concert with the inputs x_τ and associated predictions $\hat{z}_\tau$ without having to wait until the final outcome is available.

Note that in the special case where $\hat{z}_\tau$ is a linear predictor with time-invariant regression function (i.e., $\hat{z}_\tau = h(\theta, x_\tau) = \theta^T x_\tau$), TD(1) is equivalent to the least-mean-squares (LMS) representation of the stochastic gradient algorithm of Section 5.1 in the following sense: The sum of updates $(\delta\theta)_\tau$ under TD(1) corresponds to the stochastic gradient as applied to the loss function (11.9). In particular, each of the increments according to the stochastic gradient form (11.2) is

$$a(Z - \theta^T x_\tau)x_\tau,$$

which is the same as the increment in Subsection 5.1.4 for LMS after making the obvious notational changes and accounting for the "–" sign preceding the increment in Subsection 5.1.4 versus the "+" sign here. One difference from conventional LMS is that each input here (x_τ) is associated with the *same* output Z (the delayed response of delayed reinforcement learning). In contrast, in

[1]The nonequality of the increments is apparent by noting that only the final TD increment requires Z, while *all* of the stochastic gradient increments require Z.

conventional LMS, each input produces a unique output. From the equivalence of the sum in (11.3) to the sum represented by the substitution of $(\delta\boldsymbol{\theta})_\tau$ into (11.1), there is equality of the sums of the increments

$$\underbrace{a\sum_{\tau=0}^{n}(Z-\boldsymbol{\theta}^T\boldsymbol{x}_\tau)\boldsymbol{x}_\tau}_{\text{LMS}} = \underbrace{\sum_{\tau=0}^{n}(\delta\boldsymbol{\theta})_\tau = a\sum_{\tau=0}^{n}(\hat{z}_{\tau+1}-\boldsymbol{\theta}^T\boldsymbol{x}_\tau)\sum_{i=0}^{\tau}\boldsymbol{x}_i}_{\text{TD(1)}}, \qquad (11.10)$$

where $\hat{z}_{\tau+1} = \boldsymbol{\theta}^T\boldsymbol{x}_{\tau+1}$, except that $\hat{z}_{n+1} = Z$. So, TD(1) is the same as LMS in the linear case when applied to a delayed reinforcement system.

What is the best value of λ to use in practice? As with the selection of coefficients for other algorithms we have seen, there is no unique "best" value. Let us summarize why. By making a connection to maximum likelihood estimation, Sutton and Barto (1998, Sect. 6.3) establish the superiority of $\lambda = 0$ for the batch class of TD implementations, which includes the basic TD form above together with the batch generalization in the next section. On the other hand, Bertsekas (1995b) considers a class of nonlinear problems for which $\lambda = 1$ is optimal, with behavior getting progressively worse as λ decreases to zero. Numerical studies on one example in Sutton (1988) reveal that the best performance is found with $\lambda \approx 0.3$. These results are consistent with the types of behavior we have seen with other algorithms, where the choice of optimal algorithm coefficients is typically problem-specific. In practice, experimentation is likely to be needed to pick the best value of λ (and step-size coefficient a) for individual problems.

11.4 BATCH AND ONLINE IMPLEMENTATIONS OF TD LEARNING

The discussion above focused on *one* update of $\boldsymbol{\theta}$ (from $\hat{\boldsymbol{\theta}}_0$ to $\hat{\boldsymbol{\theta}}_{\text{new}}$). In practice, it is sometimes possible to repeat the process several times with new values of the input variables. For example, in the weather forecasting problem above, we may be able to compare the weekday forecasts and Saturday outcomes for several successive weeks, with each week having its own input and outcome conditions. In this case, each realization of the process represents the update from one $\boldsymbol{\theta}$ value to another, which occurs after each Saturday weather outcome. So $\hat{\boldsymbol{\theta}}_{\text{new}}$ at the conclusion of one repetition becomes $\hat{\boldsymbol{\theta}}_0$ at the next.

The combination of (11.1) and (11.7) in this iterative manner is sometimes referred to as the *batch TD* method. This label springs from the batch of realizations used in forming the *total* update of $\boldsymbol{\theta}$. A sequence of predictions is made at a fixed value of $\boldsymbol{\theta}$ during each realization, leading to a *partial* update of $\boldsymbol{\theta}$ at the conclusion of the realization. For each realization the predictions are made for the inputs at $\tau = 0, 1, \ldots, n_k$, where n_k denotes the time of the last prediction during the kth realization (so n_k corresponds to n in the basic TD

method above). One iteration of batch TD corresponds to one complete pass through the sequence of predictions, resulting in the partial update of $\boldsymbol{\theta}$. For indexing purposes, τ is assumed to restart at 0 for each realization.

In place of the one update in basic TD from the prior value $\hat{\boldsymbol{\theta}}_0$ to the updated value $\hat{\boldsymbol{\theta}}_{\text{new}}$, the batch form is based on updates from $\hat{\boldsymbol{\theta}}_{k-1}$ to $\hat{\boldsymbol{\theta}}_k$ as we move from the $(k-1)$st process realization to the kth realization. In the update, $\hat{\boldsymbol{\theta}}_{k-1}$ and $\hat{\boldsymbol{\theta}}_k$ are, respectively, analogous to $\hat{\boldsymbol{\theta}}_0$ and $\hat{\boldsymbol{\theta}}_{\text{new}}$. For the batch generalization of the predictions used in the basic TD algorithm of (11.1) with (11.7), we define $\hat{z}_\tau^{(k)} = h_\tau(\hat{\boldsymbol{\theta}}_{k-1}, \boldsymbol{x}_\tau^{(k)})$, where $\boldsymbol{x}_\tau^{(k)}$ represents the input at time $\tau = 0, 1, \ldots, n_k$ during the kth batch. This notation is an extension of the notation for basic TD. Analogous to $\hat{z}_{n+1}$ in basic TD, $\hat{z}_{n_k+1}^{(k)}$ represents the outcome $Z^{(k)}$ for the kth realization (so the arguments in the function $h_{n_k+1}(\hat{\boldsymbol{\theta}}_{k-1}, \boldsymbol{x}_{n_k+1}^{(k)})$ are superfluous). Figure 11.2 depicts the batch TD process as an extension of the updating process in basic TD, showing a total of N realizations of the process.

An alternative to batch TD—sometimes called *online TD* or *intra-sequence updating*—is for $\boldsymbol{\theta}$ to be updated after each input. In this algorithm, an updated value of $\boldsymbol{\theta}$ is used for each prediction. In particular, the increment in $\boldsymbol{\theta}$ generated at each measurement is used immediately afterward to update $\boldsymbol{\theta}$. This contrasts with the batch method where the increments are accumulated over all of the measurements at $\tau = 0, 1, \ldots, n_k$, with $\boldsymbol{\theta}$ being updated only after the entire sequence of measurements in each realization has been processed. In the weather problem, for example, the online method updates $\boldsymbol{\theta}$ after each day's forecast and after the Saturday outcome for as many weeks as data are collected (recall, as in Figure 11.1, that the Saturday outcome is also considered a forecast).

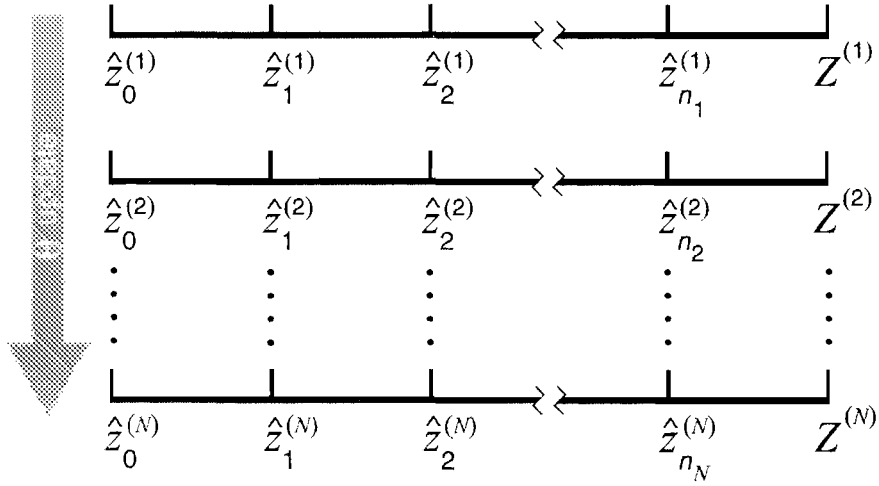

Figure 11.2. Relationship between the predictions, outcomes, and $\boldsymbol{\theta}$ updating in a batch TD process with N realizations.

Unlike the batch method, the online process is not restarted with each new week. Rather, the days of the weeks under evaluation are strung together with $\boldsymbol{\theta}$ being updated continuously. Intuitively, the online method seems more natural, as it promptly uses the information available to (one hopes) improve the subsequent predictions. In fact, as noted in Sutton (1988), "all previously studied TD methods have operated in this more fully incremental way." A further advantage of online learning is that $\boldsymbol{\theta}$ can be updated without observing an actual outcome; rather, an update occurs after each successive prediction. A downside of the online approach is the more difficult theory. Now, each temporal difference in the predictions depends on changes in the inputs *and* the value of $\boldsymbol{\theta}$.

Aside from the more frequent updating of $\boldsymbol{\theta}$, the overall structure of online TD is closely related to the basic TD form of (11.1) and (11.7). In particular, online TD is based on modifying (11.7) to reflect the current parameter value at every iteration as follows:

$$\hat{\boldsymbol{\theta}}_{\tau+1} = \hat{\boldsymbol{\theta}}_{\tau} + a_{\tau}\left[h_{\tau+1}(\hat{\boldsymbol{\theta}}_{\tau}, \boldsymbol{x}_{\tau+1}) - h_{\tau}(\hat{\boldsymbol{\theta}}_{\tau}, \boldsymbol{x}_{\tau})\right]\sum_{i=0}^{\tau}\lambda^{\tau-i}\left[\frac{\partial h_i(\boldsymbol{\theta}, \boldsymbol{x}_i)}{\partial \boldsymbol{\theta}}\right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_{\tau}}, \quad (11.11)$$

where a_{τ} is a potentially time-varying gain (step size) coefficient. This contrasts with the use of $\hat{z}_{\tau+1} = h_{\tau+1}(\hat{\boldsymbol{\theta}}_0, \boldsymbol{x}_{\tau+1})$, $\hat{z}_{\tau} = h_{\tau}(\hat{\boldsymbol{\theta}}_0, \boldsymbol{x}_{\tau})$, and $[\partial h_i(\boldsymbol{\theta}, \boldsymbol{x}_i)/\partial \boldsymbol{\theta}]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_0}$ for all τ and i in the generation of the increments to $\boldsymbol{\theta}$ in the basic algorithm of (11.7) (equivalently (11.8)) together with (11.1) (Exercise 11.3 discusses another version of online TD). The difference $\hat{\boldsymbol{\theta}}_{\tau+1} - \hat{\boldsymbol{\theta}}_{\tau}$ in (11.11) (i.e., the term to the right of the "+" sign) is analogous to the increment $(\delta\boldsymbol{\theta})_{\tau}$ in (11.7) and (11.8). Unlike the basic form in (11.8) with its recursion on $\boldsymbol{s}_{\tau}$, there is no nice recursion to replace the sum over i appearing on the right-hand side of (11.11) because all summands depend on the most recent parameter estimate. (An exception to this is when the regression functions are linear in $\boldsymbol{\theta}$ since the gradients then do not depend on $\boldsymbol{\theta}$; see, e.g., Tsitlikis and Van Roy, 1997, for such a recursion.)

11.5 SOME EXAMPLES

It may seem counterintuitive that TD methods can work better than supervised learning. After all, supervised learning is based on comparing the predictions with the real outcome, and what could be better than using the real outcome in training? The examples below are meant to provide an intuitive understanding of how the use of previous predictions for learning in TD can actually be better than the use of the real outcome. The essence of these examples is that the predictions used in TD learning better reflect historical information than the usually limited information available via the outcome Z. In this sense, TD provides a ready way to incorporate prior information, whereas supervised learning in its standard form does not.

On the other hand, if Z is formed from an average of many independent experiments—or is otherwise very reliable in indicating typical performance for

the inputs ($\boldsymbol{x}_\tau$) of interest—TD learning may offer no advantages over supervised learning. In fact, TD learning may be distinctly inferior in such a setting because it does not give full weight to the Z values. The algorithm considered in the first two examples (Examples 11.1 and 11.2) is basic TD (eqns. (11.1) and (11.7)), which is equivalent to one iteration through the batch method of TD in Figure 11.2. The third illustration (Example 11.3) uses batch TD. One can illustrate the power of TD methods using more sophisticated delayed reinforcement examples—as in the backgammon problem of Tesauro (1995)—but the basic message on the value of using successive predictions in the training process remains the same.

Example 11.1—Conceptual example in traffic modeling. Consider a major artery—Easy Street—in a traffic network. We are interested in establishing a model for predicting whether traffic conditions befit the artery's moniker. In particular, the model can be used to predict the likelihood of benign traffic conditions consistent with an easy traverse during the evening rush period given traffic patterns earlier in the day at test locations in the traffic network. The model produces predictions based on traffic patterns up to two hours before the beginning of the rush period. Let Z be a binary random variable, where $Z = 1$ denotes severe congestion by some specific criterion and $Z = 0$ denotes the desired low-congestion state. Suppose that there are two prediction functions,

$$\hat{z}_0 = h_0(\boldsymbol{\theta}, \boldsymbol{x}_0),$$

$$\hat{z}_1 = h_1(\boldsymbol{\theta}, \boldsymbol{x}_1),$$

where $\hat{z}_\tau$, $\tau = 0$ or 1, represents the probability of severe congestion given the input $\boldsymbol{x}_\tau$ at time τ, with $\tau = 0$ representing a prediction two hours before evening rush period and $\tau = 1$ representing a prediction one hour before. The input represents some indicator of the level of traffic in the network at one or two hours before the rush period. In all instances below, $\hat{z}_\tau$ depends on $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}_0$.

Consider the problem of updating $\boldsymbol{\theta}$ from one day's worth of data (i.e., one iteration of the batch method for TD; the online version of TD in (11.11) is considered in Exercise 11.7). We compare the TD method with a standard supervised learning method applied to the instantaneous error loss functions $\frac{1}{2}E\{[Z - h_\tau(\boldsymbol{\theta}, \boldsymbol{x}_\tau)]^2\}$. Note that the supervised learning here is the "standard" recursive stochastic gradient formulation of updating each regression (prediction) function with the experimental outcome. This is not the same as the summation-based loss function in (11.9); hence the supervised learning here is not the same as TD(1). In carrying out the supervised learning, one creates pairs based on the two inputs $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$. In particular, the τth increment to $\boldsymbol{\theta}$ has the form

$$(\delta\boldsymbol{\theta})_\tau^{\mathrm{SL}} \equiv a(Z - \hat{z}_\tau)\left[\frac{\partial h_\tau(\boldsymbol{\theta}, \boldsymbol{x}_\tau)}{\partial \boldsymbol{\theta}}\right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_0} \tag{11.12}$$

for $\tau = 0$ or 1, which is the same as the stochastic gradient update of (11.2). Hence, the supervised learning estimate for $\boldsymbol{\theta}$ is $\hat{\boldsymbol{\theta}}_{\text{new}}^{\text{SL}} \equiv \hat{\boldsymbol{\theta}}_0 + (\delta\boldsymbol{\theta})_0^{\text{SL}} + (\delta\boldsymbol{\theta})_1^{\text{SL}}$.

In contrast, a TD method forms pairs based on the differences of predictions. For example, with TD(0),

$$(\delta\boldsymbol{\theta})_0 = a(\hat{z}_1 - \hat{z}_0)\left[\frac{\partial h_0(\boldsymbol{\theta}, \boldsymbol{x}_0)}{\partial \boldsymbol{\theta}}\right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_0} \tag{11.13}$$

and

$$(\delta\boldsymbol{\theta})_1 = a(Z - \hat{z}_1)\left[\frac{\partial h_1(\boldsymbol{\theta}, \boldsymbol{x}_1)}{\partial \boldsymbol{\theta}}\right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_0}. \tag{11.14}$$

Note that Z in (11.14) is equivalent to the "prediction" $\hat{z}_2$. From (11.1), the new estimate for $\boldsymbol{\theta}$ is $\hat{\boldsymbol{\theta}}_{\text{new}} = \hat{\boldsymbol{\theta}}_0 + (\delta\boldsymbol{\theta})_0 + (\delta\boldsymbol{\theta})_1$. Conditioned on historical data providing a prior value of $\boldsymbol{\theta}$, $(\delta\boldsymbol{\theta})_0$ is purely deterministic, while $(\delta\boldsymbol{\theta})_1$ is random because it depends directly on the outcome Z.

To see how the TD(0) formulation can provide a better estimate, suppose without loss of generality that the components $\partial h_\tau(\boldsymbol{\theta}, \boldsymbol{x}_\tau)/\partial\boldsymbol{\theta}$ are positive (so an increase in the value of the elements of $\boldsymbol{\theta}$ cause an increase in the predicted probability of congestion on Easy Street).[2] Based on historical data, it is known that if the traffic network has a general set of bad conditions one hour prior to the evening rush period (reflected in $\boldsymbol{x}_1$), then there is an 80 percent chance that there will be severe congestion on Easy Street during the evening rush period. Let us call this state BAD, in contrast to two other possible states, GOOD and MODERATE.

Suppose a development company wishes to alter the entrance and exit patterns at a shopping mall in the traffic network, and there is concern about the effect of this change on the congestion on Easy Street. Before making the alterations permanent, the local government demands that a test be performed to determine if the changes in the entrance/exit patterns are likely to increase the likelihood of severe congestion on Easy Street. Because of the risks of serious disruption to the network traffic patterns during the testing phase, the developer is restricted to collecting data for only one day. Suppose during this test, that state BAD occurs, but that there is *not* severe traffic congestion on Easy Street during the evening rush period. Hence, $Z = 0$, while from the historical data, $\hat{z}_1 = 0.80$. Note that $(\delta\boldsymbol{\theta})_1^{\text{SL}} = (\delta\boldsymbol{\theta})_1$, but as a consequence of the difference between Z and $\hat{z}_1$, $(\delta\boldsymbol{\theta})_0^{\text{SL}} \neq (\delta\boldsymbol{\theta})_0$. In particular, the elements of $(\delta\boldsymbol{\theta})_0^{\text{SL}}$ are less than the corresponding elements of $(\delta\boldsymbol{\theta})_0$, indicating that the elements of $\hat{\boldsymbol{\theta}}_{\text{new}}^{\text{SL}}$ are lower than the elements of $\hat{\boldsymbol{\theta}}_{\text{new}}$.

---

[2] If some elements of $\boldsymbol{\theta}$ have a decreasing effect on $h_\tau(\cdot)$, the results to follow are unaffected because the TD learning process automatically alters the estimate of those elements to compensate for the opposite sign in the gradient elements.

Because the supervised learning parameter estimates are artificially lowered by the anomalous outcome of nonsevere congestion following the BAD state for $\boldsymbol{x}_1$, supervised learning produces predictions for severe congestion that are too low (recalling the monotonic relationship between the elements of θ and h_τ). This assumes, of course, that the shopping center alterations introduce no other effects that may mitigate the BAD state. The TD(0) method, on the other hand, more accurately reflects the historical connection between the BAD state and severe congestion, producing parameter values (and predictions) that are greater than the supervised learning method.

What if the experimental outcome involved the BAD state for $\boldsymbol{x}_1$ (as above), but severe congestion on Easy Street? In this case the TD(0) method still produces a better estimate. The supervised learning method associates the BAD state fully with the outcome of severe congestion, while the TD(0) method more accurately reflects that, historically, 20 percent of the time, severe congestion does *not* follow the BAD state. Hence, in this case the TD(0) parameter estimates (and predictions) are lower than those from supervised learning. ❑

Example 11.2—Numerical results for example in traffic modeling. Continuing with Example 11.1, suppose that the prediction function for the probability of severe congestion on Easy Street is

$$\hat{z}_\tau = h_\tau(\theta, x_\tau) = \frac{1}{1+10\exp[-\theta(\tau+1)x_\tau]}, \tag{11.15}$$

where the domain for the scalar θ satisfies $\Theta = [0, \infty)$. The scalar input x_τ can take on one of three values: 0, 1, or 2, representing network traffic conditions that are GOOD, MODERATE, or BAD, respectively (representing low congestion to high congestion). From Example 11.1, historical data indicate that $h_1(\theta, 2) = 0.80$. From (11.15), this uniquely determines a value for θ, taken as the prior estimate $\hat{\theta}_0 = 0.922$. Note that for a given θ and $x_0 = x_1$, the prediction $\hat{z}_0$ is less than $\hat{z}_1$ as a reflection of: (i) the normal state of no congestion on Easy Street (hence the eponymous name!) and (ii) the reduced ability to use current conditions to predict congestion two hours in advance versus one hour in advance (e.g., with a MODERATE congestion input, $\hat{z}_0 = h_0(\hat{\theta}_0, 1) = 0.201$ versus $\hat{z}_1 = h_1(\hat{\theta}_0, 1) = 0.387$). For use in the learning algorithms, note that

$$\frac{\partial h_\tau(\theta, x_\tau)}{\partial \theta} = \frac{10(\tau+1)x_\tau \exp[-\theta(\tau+1)x_\tau]}{\{1+10\exp[-\theta(\tau+1)x_\tau]\}^2}.$$

From Example 11.1, $x_1 = 2$ and $Z = 0$; also suppose that traffic conditions are MODERATE two hours before rush period (i.e., $x_0 = 1$). Letting $a = 1$, expression (11.12) for the increments in θ implies that the two supervised training increments are

$$(\delta\theta)_0^{SL} = (0.0 - 0.201)\frac{3.9764}{(1+3.9764)^2} = -0.032,$$

$$(\delta\theta)_1^{SL} = (0.0 - 0.800)\frac{1.0001}{(1+0.2500)^2} = -0.512,$$

whereas from (11.13) and (11.14), the TD(0) increments are

$$(\delta\theta)_0 = (0.800 - 0.201)\frac{3.9764}{(1+3.9764)^2} = 0.096$$

$$(\delta\theta)_1 = (\delta\theta)_1^{SL} = -0.512.$$

Using these increments, the new values of θ from supervised learning and TD(0) are

$$\hat{\theta}_{new}^{SL} \equiv \hat{\theta}_0 + (\delta\theta)_0^{SL} + (\delta\theta)_1^{SL} = 0.922 - 0.032 - 0.512 = 0.378,$$

$$\hat{\theta}_{new} = \hat{\theta}_0 + (\delta\theta)_0 + (\delta\theta)_1 = 0.922 + 0.096 - 0.512 = 0.506.$$

The TD(0) method produces a more accurate parameter estimate in the sense that it better preserves the prior information associating the BAD state one hour before the rush period with congestion during the rush period. For $\tau = 1$ (i.e., one hour ahead), Table 11.1 compares the predictions based on $\hat{\theta}_{new}^{SL}$ and $\hat{\theta}_{new}$ with those from historical data.

Relative to supervised learning, the TD(0) estimates are closer to the historical estimates for all input state values, reflecting some level of congestion

Table 11.1. Values of the prediction $\hat{z}_1$ for the three levels of input (x_1) under the parameter values $\hat{\theta}_0$, $\hat{\theta}_{new}^{SL}$, and $\hat{\theta}_{new}$.

Input state x_1	Historical estimate (from $\hat{\theta}_0$)	Supervised learning (from $\hat{\theta}_{new}^{SL}$)	Basic TD(0) (from $\hat{\theta}_{new}$)
0 (GOOD)	0.09	0.09	0.09
1 (MODERATE)	0.39	0.18	0.22
2 (BAD)	0.80	0.31	0.43

in the network one hour prior to the rush period ($x_1 = 1$ or 2). As a consequence of the experimental outcome (Z), however, the supervised learning and TD(0) predictions lie below the historical predictions for these nonzero input states. ❑

Example 11.3—Random-walk model. Let us summarize an interesting example from Sutton and Barto (1998, pp. 139–141). Consider a process with seven possible states as shown in Figure 11.3. The random walk is initialized at state S_3 and proceeds until it reaches one of the termination states T_{left} or T_{right}. A possible walk might be S_3–S_4–S_3–S_4–S_5–T_{right}. The aim is to estimate the probability of the walk ending at T_{right} given that the walk is in any one of the five nonterminal states. For example, if the process is in state S_2, we are interested in the probability of the process ultimately reaching T_{right}. The outcome is $Z = 1$ if termination occurs at T_{right}; $Z = 0$ if termination occurs at T_{left}. Hence, $P(Z = 1 | S_i) = E(Z|S_i)$. Consider the simple linear prediction function $\hat{z}_\tau = h_\tau(\boldsymbol{\theta}, \boldsymbol{x}_\tau) = \boldsymbol{\theta}^T \boldsymbol{x}_\tau$, where $\boldsymbol{x}_\tau$ is a vector of five elements, four of which are zero and one of which is unity as an indicator of the state in which the walk resides. For example, if the walk is in state S_2 at time τ, then $\boldsymbol{x}_\tau = [0, 1, 0, 0, 0]^T$. Let $\boldsymbol{\theta} \in \Theta = [0, 1]^5$ be a five-dimensional vector of the probabilities of interest for the five nonterminal states.

Batch TD (Section 11.4) is applied to estimating these five probabilities based on an initial condition $\hat{\boldsymbol{\theta}}_0 = [0.5, 0.5, 0.5, 0.5, 0.5]^T$. From Sutton and Barto (1998, Fig. 6.6), Table 11.2 shows the estimated probabilities after batch sizes of $N = 1$, 10, and 100 using a constant step size $a = 0.1$. The true probabilities are found in Sutton (1988) as $\boldsymbol{\theta}^* = [1/6, 1/3, 1/2, 2/3, 5/6]^T$ (see Exercise 11.8). That only one parameter (the first element of $\boldsymbol{\theta}$, corresponding to state S_1) changes with $N = 1$ is a direct consequence of having only one outcome Z ($= Z^{(0)}$) in the training process (see Exercise 11.9). As expected, the estimates improve with increasing N. The estimate at $N = 100$, however, is about as close to truth as the algorithm can ever get. This is a consequence of the constant (nondecaying) gain causing the $\boldsymbol{\theta}$ estimates to fluctuate indefinitely with each new realization in the batch process (see Exercises 11.10 and 11.11). ❑

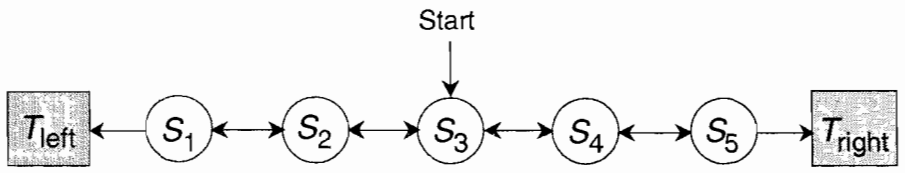

Figure 11.3. Schematic of process for generating random walks. All walks begin in the center state S_3. For each of states S_1 to S_5, the walk has a 50 percent chance of moving either left or right. The walk stops if the process reaches either of the termination states T_{left} or T_{right}.

Table 11.2. True probabilities and estimated probabilities of reaching T_{right} for the five states based on N realizations in batch TD(0).

State	True probability	$N = 1$	$N = 10$	$N = 100$
S_1	0.17	0.45	0.35	0.13
S_2	0.33	0.50	0.50	0.30
S_3	0.50	0.50	0.52	0.48
S_4	0.67	0.50	0.56	0.71
S_5	0.83	0.50	0.71	0.80

11.6 CONNECTIONS TO STOCHASTIC APPROXIMATION

The TD formulation has a ready connection to root finding and the stochastic gradient form of SA that was discussed in Chapter 5. This connection has been exploited to make rigorous statements about the convergence of TD methods. Using methods other than SA, there were prior attempts at showing formal convergence (e.g., Sutton, 1988), but these were largely unsatisfactory as a result of the weak mode of convergence shown[3] and the conditions being so restrictive that few practical settings would qualify. Once the connection to SA was identified, more satisfying convergence results were possible in terms of the strength of convergence (a.s.), the rigor of the analysis, and the general applicability of the results. The TD conditions are directly related to the type of conditions seen in root-finding (Robbins–Monro) SA (Section 4.3). This section will only outline the connections of TD to the SA formulation. The interested reader is directed to the indicated references for the formal convergence conditions and proofs.

Let us first consider the batch form of TD represented in Figure 11.2. We comment on the online version (the form in (11.11)) below. Dayan and Sejnowski (1994), Jaakkola et al. (1994), and Pineda (1997) use SA to show formal (a.s.) convergence of batch TD. Replacing the fixed a with the typical gain coefficient indexed by k, the batch TD(λ) algorithm can then be written in SA form as

$$\hat{\boldsymbol{\theta}}_k = \hat{\boldsymbol{\theta}}_{k-1} - a_{k-1}\boldsymbol{Y}_{k-1}(\hat{\boldsymbol{\theta}}_{k-1}) \tag{11.16}$$

[3]For example, Sutton (1988) shows only that the mean of the prediction $h_\tau(\cdot)$ using TD(0) training converges. This convergence *of* mean is a very weak mode of convergence (see Corollary 1 to Theorem C.1 in Appendix C), far weaker than convergence *in* mean (Theorem C.1 with q = 1). For example, an i.i.d. sequence of nondegenerate random variables will never converge in any meaningful way, but the mean of the sequence is trivially convergent from the very first element in the sequence.

(relative to the usual SA form in this book, we are showing a form lagged by one in k to avoid messy super/subscript notation below). From (11.1) and (11.7),

$$\boldsymbol{Y}_{k-1}(\hat{\boldsymbol{\theta}}_{k-1}) = -\sum_{\tau=0}^{n_k} \left(\hat{z}_{\tau+1}^{(k)} - \hat{z}_{\tau}^{(k)} \right) \sum_{i=0}^{\tau} \lambda^{\tau-i} \left[\frac{\partial h_i(\boldsymbol{\theta}, \boldsymbol{x}_i^{(k)})}{\partial \boldsymbol{\theta}} \right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_{k-1}} . \quad (11.17)$$

Note that at any $\boldsymbol{\theta}$, $\boldsymbol{Y}_k(\boldsymbol{\theta})$ is an unbiased measurement of the *time-varying* function (e.g., Subsection 4.5.4) for which we wish to find a zero:

$$\boldsymbol{g}_{k-1}(\boldsymbol{\theta}) \equiv -E \left\{ \sum_{\tau=0}^{n_k} \left[h_{\tau+1}(\boldsymbol{\theta}, \boldsymbol{x}_{\tau+1}^{(k)}) - h_{\tau}(\boldsymbol{\theta}, \boldsymbol{x}_{\tau}^{(k)}) \right] \sum_{i=0}^{\tau} \lambda^{\tau-i} \frac{\partial h_i(\boldsymbol{\theta}, \boldsymbol{x}_i^{(k)})}{\partial \boldsymbol{\theta}} \right\}, \quad (11.18)$$

where the expectation is computed with respect to the random outcome (the $Z^{(k)} = \hat{z}_{n_k+1}^{(k)} = h_{n_k+1}(\cdot)$) values at the end of the batches), the potential randomness in the inputs values $\boldsymbol{x}_0^{(k)}$, $\boldsymbol{x}_1^{(k)}, \ldots, \boldsymbol{x}_{n_k}^{(k)}$, and the potential randomness in the realization length n_k (e.g., random realization lengths as in Example 11.3).

The arguments of Section 5.1 provide the justification for $\boldsymbol{Y}_k(\boldsymbol{\theta})$ being an unbiased measurement of $\boldsymbol{g}_k(\boldsymbol{\theta})$. Recall that these arguments rely on the probability distribution of the fundamental randomness in $\boldsymbol{Y}_k(\boldsymbol{\theta})$ being independent of $\boldsymbol{\theta}$. That is, for the application here, the distribution of the $Z^{(k)}$, $\boldsymbol{x}_{\tau}^{(k)}$, and n_k should not depend on $\boldsymbol{\theta}$ (which is entirely reasonable since the *physical* $Z^{(k)}$, $\boldsymbol{x}_{\tau}^{(k)}$, and n_k may be generated without regard to the *model* parameters $\boldsymbol{\theta}$). Then, from Section 5.1, an unbiased measurement of $\boldsymbol{g}_k(\boldsymbol{\theta})$ is the argument inside the $\{\cdot\}$ braces in (11.18), which is the same as $\boldsymbol{Y}_k(\hat{\boldsymbol{\theta}}_k)$ in (11.17). (These ideas provide the basis for some of the counterexamples to TD learning mentioned at the end of Section 11.2. In control applications, the control actions $\boldsymbol{x}_t^{(k)}$ depend on $\boldsymbol{\theta}$ through feedback. Hence, the probability distribution of the fundamental randomness entering $\boldsymbol{Y}_k(\boldsymbol{\theta})$ depends on $\boldsymbol{\theta}$.)

Relative to batch TD, online TD has a more direct connection to the traditional SA formulation since $\boldsymbol{\theta}$ is naturally updated as every new input value is collected. The formal convergence proofs, however, require a modification to the basic online form in (11.11). In particular, there is a need for frequent examples of "truth" to ensure convergence to a *meaningful* $\boldsymbol{\theta}$ (convergence to *some* value of $\boldsymbol{\theta}$ may be possible without the true outcomes, but this $\boldsymbol{\theta}$ may not be a "good" value relative to the true system). Note that in (11.11), it is possible for the algorithm to run arbitrarily long with only one actual outcome; this is an insufficient instance of truth for meaningful convergence. The available online convergence proofs (e.g., Jaakkola et al., 1994; Tsitsiklis and Van Roy, 1997, 1999) overcome this difficulty by posing the problem in the framework of dynamic programming, with the goal of predicting some type of average or cumulative cost for a process based on a sequence of observed instantaneous

costs. The observed costs at each iteration represent measurements of the actual physical system. The cost inputs provide the necessary information to ensure meaningful convergence.

The online version of TD once again uses the basic SA recursion (11.16), but rather than input (11.17), the input is

$$\boldsymbol{Y}_k(\hat{\boldsymbol{\theta}}_k) = -\left[h_{k+1}(\hat{\boldsymbol{\theta}}_k, \boldsymbol{x}_{k+1}) - h_k(\hat{\boldsymbol{\theta}}_k, \boldsymbol{x}_k) + \text{cost}_k\right] \sum_{i=0}^{k} \lambda^{k-i} \left[\frac{\partial h_i(\boldsymbol{\theta}, \boldsymbol{x}_i)}{\partial \boldsymbol{\theta}}\right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_k}, \tag{11.19}$$

where the temporal difference of predictions includes the instantaneous cost measurement, cost_k. (Jang et al., 1997, pp. 268–270, includes a brief tutorial on the use of TD methods in such cost estimation.) As with the batch form, $\boldsymbol{Y}_k(\boldsymbol{\theta})$ represents a measurement of some time-varying function $\boldsymbol{g}_k(\boldsymbol{\theta})$.

While the form in (11.16) with input of either (11.17) or (11.19) fits well within the framework of root-finding SA with a time-varying nonlinear functions, existing convergence results are only for the case where the $h_\tau(\boldsymbol{\theta}, \boldsymbol{x})$ are linear functions in $\boldsymbol{\theta}$. The simplest such case is the pure linear form $\boldsymbol{\theta}^T\boldsymbol{x}$ considered, for example, in Dayan and Sejnowski (1994). The more general curvilinear case (linear in $\boldsymbol{\theta}$, nonlinear in $\boldsymbol{x}$ with, as in Section 3.1, $\dim(\boldsymbol{\theta})$ not necessarily equal to $\dim(\boldsymbol{x})$) is considered in Tsitsiklis and Van Roy (1997, 1999). Significant difficulties exist for proving convergence in more general functions nonlinear in $\boldsymbol{\theta}$ (e.g., neural networks with weights $\boldsymbol{\theta}$). One of the obvious difficulties to showing convergence in general nonlinear problems is that the function $\boldsymbol{g}_k(\cdot)$ is time varying. To illustrate the difficulties in convergence for general nonlinear functions, Tsitsiklis and Van Roy (1997, Sect. 10) provides a counterexample to convergence using a fairly simple nonlinear form for the $h_\tau(\boldsymbol{\theta}, \boldsymbol{x})$.

11.7 CONCLUDING REMARKS

We have looked at aspects of the delayed reinforcement problem. A fundamental challenge is that there is no system output that indicates "truth" at each time step of the learning process. The temporal difference algorithm is one powerful method for addressing this challenge by processing the differences of model-based *predictions* of the process. The temporal differences—although not based on the final outcome of the process until the last time step—contain valuable information for improving the prediction process. In the language of machine learning and artificial intelligence, the learning system is to predict the final state of the system (or final reward) without the benefit of a teacher signal indicating the correct value at each time step.

The TD algorithm is able to develop improved final predictions based on the prior information contained in the current model with its associated intermediate predictions. The basic and enhanced TD algorithms are built from connections to the stochastic gradient algorithm (Chapter 5). There are additional connections to stochastic approximation in the batch and online versions of TD learning; these connections are useful in establishing formal convergence.

A number of papers in the literature pertain to the connection of TD learning to Markov chains and dynamic programming, including the use in stochastic control problems (e.g., Jaakkola et al., 1994; Tsitsiklis and Van Roy, 1997). This chapter did not explore these connections. The focus here was pure prediction problems. In control, prediction is used to determine the states to which the system is expected to go with specific control inputs. The control values can be adjusted over time to produce predictions that are close to desired values. The combined control and delayed reinforcement learning problem, of course, introduces complications in theory and implementation not present in the pure prediction problems. Sutton et al (1992) and Tesauro (1992), among others, consider control in TD or more general reinforcement learning.

EXERCISES

11.1 Fill in the missing steps in the double-sum formula (11.5) used in obtaining the basic TD algorithm.

11.2 Show by direct calculation the validity of the equality of LMS and TD(1) in (11.10).

11.3 Relative to (11.11), an alternative online form of TD learning for constant gains $a = a_\tau$ is

$$\hat{\boldsymbol{\theta}}_{\tau+1} = \hat{\boldsymbol{\theta}}_\tau + a\left[h_{\tau+1}(\hat{\boldsymbol{\theta}}_\tau, \boldsymbol{x}_{\tau+1}) - h_\tau(\hat{\boldsymbol{\theta}}_{\tau-1}, \boldsymbol{x}_\tau)\right] \sum_{i=0}^{\tau} \lambda^{\tau-i} \left[\frac{\partial h_i(\boldsymbol{\theta}, \boldsymbol{x}_i)}{\partial \boldsymbol{\theta}} \right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_{i-1}},$$

where $\hat{\boldsymbol{\theta}}_{-1} = \hat{\boldsymbol{\theta}}_0$. This form is natural in the sense that each regression function is associated with its most recent parameter estimate.

(a) Derive a recursion analogous to the $\boldsymbol{s}_\tau$ recursion used in (11.8) so that the sum over i does not have to be explicitly computed at each iteration.

(b) Provide at least two reasons why this online form of TD is likely to be inferior to the form in (11.11).

11.4 In the manner of Example 11.1, design a *conceptual* example illustrating the potential power of the TD method to produce more reliable estimates than the supervised learning method. The example should be substantively different from the traffic problem in Example 11.1.

11.5 Provide a conceptual example illustrating how it is possible for the TD method to tend to provide *poorer* estimates than supervised learning.

11.6 Consider the setting of Example 11.2 with TD(0.5) replacing TD(0) while continuing with the basic algorithm of (11.1) and (11.7):

(a) Produce a table (analogous to Table 11.1) that includes the historical, supervised learning, and TD(0.5) predictions two hours (versus one hour in Table 11.1) in advance.

(b) Update Table 11.1 (based on the possible values of x_1) to reflect the change to TD(0.5). Comment on the relative performance of TD(0.5) and TD(0).

11.7 Compare the predictions for basic TD(0) in Table 11.1 with corresponding predictions from online TD(0) (Section 11.4) using the information in Example 11.2.

11.8 Construct the 7×7 transition matrix (see Appendix E) describing the probability of going from one state to another in Example 11.3. Use this transition matrix to obtain the true probabilities shown in Table 11.2.

11.9 As shown in Table 11.2, only the first element of $\boldsymbol{\theta}$ changes when $N = 1$ for the random walk example.

(a) Illustrate how this result occurred.

(b) Discuss the other possible estimate for $\boldsymbol{\theta}$ that could have occurred.

(c) How do the results in parts (a) and (b) differ fundamentally from those that would be obtained if TD(λ), $\lambda > 0$, were used instead of TD(0)?

11.10 Consider Example 11.3. With $a = 0.10$ and $\hat{\boldsymbol{\theta}}_0 = [0.5, 0.5, 0.5, 0.5, 0.5]^T$, as in the example, calculate terminal $\boldsymbol{\theta}$ estimates (a single realization) using $N = 1000$ and $N = 2000$ batches with TD(0). Verify that these estimates based on a constant gain improve little from the $N = 100$ values in Table 11.2.

11.11 Consider the setup in Exercise 11.10 with the exception of using *decaying* gains a_k of the standard harmonic-sequence form $a_k = a/(k+1)$ for some $a > 0$ (Section 4.3). The gain sequence should decay from realization to realization (not within the realization), so N realizations in the batch updating requires $a_0, a_1, \ldots, a_{N-1}$. As an example of the convergence results for batch TD (Section 11.6), show numerical convergence as k ranges from $k = 0$ to $k = N - 1 = 1999$.