REINFORCEMENT LEARNING AND STOCHASTIC OPTIMIZATION

A unified framework for sequential decisions

Warren B. Powell

August 22, 2021

WILEY-INTERSCIENCE

A JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Optimization Under Uncertainty: A unified framework
Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

CHAPTER 16

FORWARD ADP I: THE VALUE OF A POLICY

Chapter 14 laid the foundation for finding optimal policies for problems with discrete states, assuming that states and decisions can be enumerated, and the one-step transition matrix can be calculated. This chapter presented classical backward dynamic programming for finite horizon problems, but most of the chapter focused on infinite horizon problems, where we introduced several methods for computing the value function $V(s)$. Of these, the most important is value iteration, since this is relatively easy to compute, and it is the foundation for a number of approximation strategies.

In chapter 15, we introduced the idea of backward approximate dynamic programming (for finite horizon problems), also known as fitted value iteration for infinite horizon problems. Backward approximate dynamic programming is, surprisingly, a relatively recent invention, and while fitted value iteration is somewhat older, the attention it has received is a small fraction compared to the methods that we are going to present in this chapter and chapters 17 and 18, which are all based on the principle of *forward* approximate dynamic programming.

We suspect the reason for the relative popularity of forward approximate dynamic programming is that it captures the dynamics of an actual physical system, which moves forward in time. It has the immediate benefit of avoiding any semblance of enumerating states, which avoids "the" curse of dimensionality (which is most often associated with vector-valued states). It even avoids the need to determine how to sample the state space, as is required in backward dynamic programming.

It also avoids any need to compute the one-step transition matrix, since we are either simulating the exogenous information W_{t+1}, or we are simply observing transitions from

S_t to S_{t+1} from a physical system. When we step forward in time, it seems as if there is a natural sampling mechanism, and while this is true to a degree, we will have to pay attention to how we choose the decisions that determine (up to a point) the next state we visit. By contrast, backward ADP requires pure, random sampling, which means our choice of states is not guided at all by the physics of the problem beyond assumptions about the range of states.

Most of the work in this area still assumes discrete decisions, which enjoys a very wide set of applications. We cover vector-valued decisions, but not until chapter 18 where we limit our focus to problems with concave (convex if minimizing) contribution functions (which translates to concavity in the value function).

In this chapter, we focus primarily on the different ways of calculating $\hat{v}^n$, and then using this information to estimate a value function approximation, *for a fixed policy*. The reason we do this is to resolve the subtleties of estimating the value of a policy before we allow the policy to evolve with the iterations, which introduces a significant complication. To emphasize that we are computing values for a fixed policy, we index parameters such as the value function V^π by the policy π. After we establish the fundamentals for estimating the value of a policy, chapter 17 addresses the of process of searching for good policies.

16.1 SAMPLING THE VALUE OF A POLICY

On first glance, the problem of statistically estimating the value of a fixed policy should not be any different than estimating a function from noisy observations. In fact, this can be true, but it depends on how $\hat{v}^n$ is being calculated. In time (especially in chapter 17 when we are optimizing over policies), we will have to learn to live with the reality that $\hat{v}^n$ is almost always a biased sampled estimate of the value of being in a state.

Our normal style has been to model finite horizon problems without a discount factor. Of course, discounting is essential in infinite horizon problems, as we saw in chapter 14. Below, we are going to sometimes switch between finite and infinite horizon, so we retain a discount factor γ even for the finite horizon case.

16.1.1 Direct policy evaluation for finite horizon problems

Imagine that we have a fixed policy $X^\pi(s)$ which may take any of the forms described in chapter 11. For iteration n, if we are in state S_t^n at time t, we then choose decision $x_t^n = X^\pi(S_t^n)$, after which we sample the exogenous information W_{t+1}^n. We sometimes say that we are following sample path ω^n from which we observe $W_{t+1}^n = W_{t+1}(\omega^n)$. The exogenous information W_{t+1}^n may depend on both S_t^n and the decision x_t^n. From this, we may compute our contribution from

$$\hat{C}_t^n = C(S_t^n, x_t^n).$$

Finally, we compute our next state from our transition function

$$S_{t+1}^n = S^M(S_t^n, x_t^n, W_{t+1}^n).$$

This process continues until we reach the end of our horizon T. The basic algorithm is described in figure 16.1. In step 6, we use a batch routine to fit a statistical model. It is often more natural to use some sort of recursive procedure and imbed the updating of the value function within the iterative loop. The type of recursive procedure depends on

Step 0. Initialization:

Step 0a. Initialize $\overline{V}^0$.

Step 0b. Initialize S^1.

Step 0c. Set $n = 1$.

Step 1. Choose a sample path ω^n.

Step 2. Choose a starting state S_0^n.

Step 3. Do for $t = 0, 1, \ldots, T$:

Step 3a. $x_t^n = X^\pi(S_t^n)$.

Step 3b. $\hat{C}_t^n = C(S_t^n, x_t^n)$.

Step 3c. $W_{t+1}^n = W_{t+1}(\omega^n)$.

Step 3d. $S_{t+1}^n = S^M(S_t^n, x_t^n, W_{t+1}^n)$.

Step 4. Compute $\hat{v}_0^n = \sum_{t=0}^T \gamma^t \hat{C}_t^n$.

Step 5. Increment n. If $n \leq N$ go to Step 1.

Step 6. Use the sequence of state-value pairs $(S^i, \hat{v}^i)_{i=1}^N$ to fit a value function approximation $\overline{V}^\pi(s)$.

Figure 16.1 Basic policy evaluation procedure.

the nature of the value function approximation. Later in this chapter, we describe several recursive procedures if we are using linear regression.

Finite horizon problems are sometimes referred to as *episodic*, where an episode refers to a simulation of a policy until the end of the horizon (also known as trials). However, the term episodic can also be interpreted more broadly. For example, an emergency vehicle may repeatedly return to base where the system then restarts. Each cycle of starting from a home base and then returning to the home base can be viewed as an episode. As a result, if we are working with a finite horizon problem, we prefer to refer to these specifically as such.

Evaluating a fixed policy is mathematically equivalent to making unbiased observations of a noisy function. Fitting a functional approximation is precisely what the entire field of statistical learning has been trying to do for decades. If we are fitting a linear model, then there are some powerful recursive procedures that can be used. These are discussed below.

16.1.2 Policy evaluation for infinite horizon problems

Not surprisingly, infinite horizon problems introduce a special complication, since we cannot obtain an unbiased observation in a finite number of measurements. Below we present some methods that have been used for infinite horizon applications.

Recurrent visits

There are many problems which are infinite horizon, but where the system resets itself periodically. A simple example of this is a finite horizon problem, where hitting the end of the horizon and starting over (as would occur in a game) can be viewed as an episode. A different example is a queueing system, where perhaps we are trying to manage the admission of patients to an emergency room. From time to time the queue may become

empty, at which point the system resets and starts over. For such systems, it makes sense to estimate the value of following a policy π when starting from this base state.

Even if we do not have such a renewal system, imagine that we find ourselves in a state s. Now follow a policy π until we re-enter state s again. Let $R^n(s)$ be the reward earned, and let $\tau^n(s)$ be the number of time periods required before re-entering state s. Here, n is counting the number of times we visit state s. An observation of the average reward earned when in state s and following policy π would be given by

$$\hat{v}^n(s) = \frac{R^n(s)}{\tau^n(s)}.$$

$\hat{v}^n(s)$ would be computed when we return to state s. We might then update the *average* value of being in state s using

$$\bar{v}^n(s) = (1 - \alpha_{n-1})\bar{v}^{n-1}(s) + \alpha_{n-1}\hat{v}^n(s).$$

Note that as we make each transition from some state s' to some state s'', we are accumulating rewards in $R(s)$ for every state s that we have visited prior to reaching state s'. Each time we arrive at some state s'', we stop accumulating rewards for s'', and compute $\hat{v}^n(s'')$, and then smooth this into the current estimate of $\bar{v}(s'')$. Note that we have presented this only for the case of computing the average reward per time period.

Partial simulations

While we may not be able to simulate an infinite trajectory, we may simulate a long trajectory T, long enough to ensure that we are producing an estimate that is "good enough." When we are using discounting, we realize that eventually γ^t becomes small enough that a longer simulation does not really matter. This idea can be implemented in a relatively simple way.

Consider the algorithm in figure 16.1, and insert the calculation in step 3:

$$\bar{c}_t = \frac{t-1}{t}\bar{c}_{t-1} + \frac{1}{t}\hat{C}^n_t.$$

$\bar{c}_t$ is an average over the time periods of the contribution per time period. As we follow our policy over progressively more time periods, $\bar{c}_t$ approaches an average contribution per time period. Over an infinite horizon, we would expect to find

$$\hat{v}^n_0 = \lim_{t\to\infty} \sum_{t=0}^{\infty} \gamma^t \hat{C}^n_t = \frac{1}{1-\gamma}\bar{c}_\infty.$$

Now assume that we only progress T time periods, and let $\bar{c}_T$ be our estimate of $\bar{c}_\infty$ at this point. We would expect that

$$\begin{aligned} \hat{v}^n_0(T) &= \sum_{t=0}^{T} \gamma^t \hat{C}^n_t \\ &\approx \frac{1-\gamma^{T+1}}{1-\gamma}\bar{c}_T. \end{aligned} \tag{16.1}$$

The error between our T-period estimate $\hat{v}_0^n(T)$ and the infinite horizon estimate $\hat{v}_0^n$ is given by

$$\begin{aligned}\delta_T^n &= \frac{1}{1-\gamma}\bar{c}_\infty - \frac{1-\gamma^{T+1}}{1-\gamma}\bar{c}_T \\ &\approx \frac{1}{1-\gamma}\bar{c}_T - \frac{1-\gamma^{T+1}}{1-\gamma}\bar{c}_T \\ &= \frac{\gamma^{T+1}}{1-\gamma}\bar{c}_T.\end{aligned}$$

Thus, we just have to find T to make δ_T small enough. This strategy is imbedded in some optimal algorithms, which only require that $\delta_T^n \to 0$ as $n \to \infty$ (meaning that we have to steadily allow T to grow).

Infinite horizon projection

We can easily see from (16.1) that if we stop after T time periods, we will underestimate the infinite horizon contribution by a factor $1-\gamma^{T+1}$. Assuming that T is reasonably large (say, $\gamma^{T+1} < 0.1$), we might introduce the correction

$$\hat{v}_0^n = \frac{1}{1-\gamma^{T+1}}\hat{v}_0^n(T).$$

In essence we are taking a sample estimate of a T-period path, and projecting it out over an infinite horizon.

16.1.3 Temporal difference updates

Assume that we are in state S_t^n and we make decision x_t^n (using policy π), after which we observe the information W_{t+1} which puts us in state $S_{t+1}^n = S^M(S_t^n, x_t^n, W_{t+1}^n)$. The contribution from this transition is given by $C(S_t^n, x_t^n)$. Imagine now that we continue this until the end of our horizon T. For simplicity, we are going to drop discounting. In this case, the contribution along this path would be

$$\hat{v}_t^n = C(S_t^n, x_t^n) + C(S_{t+1}^n, x_{t+1}^n) + \ldots + C(S_T^n, x_T^n). \tag{16.2}$$

This is the contribution from following the path produced by a combination of the information from outcome ω^n (this determines $W_{t+1}^n, W_{t+2}^n, \ldots, W_T^n$) and policy π. $\hat{v}_t^n$ is an unbiased sample estimate of the value of being in state S_t and following policy π over sample path ω^n. We can use a stochastic gradient algorithm to estimate the value of being in state S_t using

$$\overline{V}_t^n(S_t^n) = \overline{V}_t^{n-1}(S_t^n) - \alpha_n\left(\overline{V}_t^{n-1}(S_t^n) - \hat{v}_t^n\right). \tag{16.3}$$

We can obtain a richer class of algorithms by breaking down our path cost in (16.2) by using

$$\begin{aligned}\hat{v}_t^n &= \sum_{\tau=t}^{T} C(S_\tau^n, x_\tau^n) \\ &\quad - \underbrace{\left\{\sum_{\tau=t}^{T}\left(\overline{V}_\tau^{n-1}(S_\tau) - \overline{V}_{\tau+1}^{n-1}(S_{\tau+1})\right)\right\}}_{=0} + (\overline{V}_t^{n-1}(S_t) - \overline{V}_{T+1}^{n-1}(S_{T+1})).\end{aligned}$$

We now use the fact that $\overline{V}_{T+1}^{n-1}(S_{T+1}) = 0$ (this is where our finite horizon model is useful). Rearranging gives

$$\hat{v}_t^n = \overline{V}_t^{n-1}(S_t) + \sum_{\tau=t}^{T} \big(C(S_\tau^n, x_\tau^n) + \overline{V}_{\tau+1}^{n-1}(S_{\tau+1}) - \overline{V}_\tau^{n-1}(S_\tau)\big).$$

Let

$$\delta_\tau = C(S_\tau^n, x_\tau^n) + \overline{V}_{\tau+1}^{n-1}(S_{\tau+1}^n) - \overline{V}_\tau^{n-1}(S_\tau^n). \tag{16.4}$$

The terms δ_τ are called *temporal differences*. If we were using a standard single-pass algorithm, then at time t, $\hat{v}_t^n = C(S_t^n, x_t^n) + \overline{V}_{t+1}^{n-1}(S_{t+1}^n)$ would be our sample observation of being in state S_t, while $\overline{V}_t^{n-1}(S_t)$ is our current estimate of the value of being in state S_t. This means that the temporal difference at time t, $\delta_t = \hat{v}_t^n - \overline{V}_t^{n-1}(S_t)$, is the difference in our estimate of the value of being in state S_t between our current estimate and the updated estimate. The temporal difference is also known as the *Bellman error*.

Using (16.4), we can write $\hat{v}_t^n$ in the more compact form

$$\hat{v}_t^n = \overline{V}_t^{n-1}(S_t) + \sum_{\tau=t}^{T} \delta_\tau. \tag{16.5}$$

Substituting (16.5) into (16.3) gives

$$\begin{aligned}
\overline{V}_t^n(S_t) &= \overline{V}_t^{n-1}(S_t) - \alpha_{n-1}\left[\overline{V}_t^{n-1}(S_t) - \left(\overline{V}_t^{n-1}(S_t) + \sum_{\tau=t}^{T}\delta_\tau\right)\right] \\
&= \overline{V}_t^{n-1}(S_t) + \alpha_{n-1}\sum_{\tau=t}^{T-1}\delta_\tau.
\end{aligned} \tag{16.6}$$

We next use this bit of algebra to build an important class of updating mechanisms for estimating value functions.

16.1.4 TD(λ)

The temporal differences δ_τ are the errors in our estimates of the value of being in state S_τ. We can think of each term in (16.6) as a correction to the estimate of the value function. It makes sense that updates farther along the path should not be given as much weight as those earlier in the path. As a result, it is common to introduce an artificial discount factor λ, producing updates of the form

$$\overline{V}_t^n(S_t) = \overline{V}_t^{n-1}(S_t) + \alpha_{n-1}\sum_{\tau=t}^{T}\lambda^{\tau-t}\delta_\tau. \tag{16.7}$$

We derived this formula without a time discount factor. We leave as an exercise to the reader to show that if we have a time discount factor γ, then the temporal-difference update becomes

$$\overline{V}_t^n(S_t) = \overline{V}_t^{n-1}(S_t) + \alpha_{n-1}\sum_{\tau=t}^{T}(\gamma\lambda)^{\tau-t}\delta_\tau. \tag{16.8}$$

Equation (16.8) shows that the discount factor γ, which is typically viewed as capturing the time value of money, and the algorithmic discount λ, which is a purely algorithmic device, have exactly the same effect. Not surprisingly, modelers in operations research have often used a discount factor γ set to a much smaller number than would be required to capture the time-value of money. Artificial discounting allows us to look into the future, but then discount the results when we feel that the results are not perfectly accurate.

Updates of the form given in equation (16.7) produce an updating procedure that is known as TD(λ) (or, temporal difference learning with discount λ). Here, λ is introduced as a form of algorithmic discounting, since it has nothing to do with the traditional use of discounting to reflect the value of money. Algorithmic discounting is a heuristic way of limiting the effect of decisions we plan on making in the future, given that our model of the future is imperfect.

The updating formula in equation (16.7) requires that we step all the way to the end of the horizon before updating our estimates of the value. There is, however, another way of implementing the updates. The temporal differences δ_τ are computed as the algorithm steps forward in time. As a result, our updating formula can be implemented recursively. Assume we are at time t' in our simulation. We would simply execute

$$\overline{V}_t^n(S_t^n) := \overline{V}_t^n(S_t) + \alpha_{n-1}\lambda^{t'-t}\delta_{t'} \quad \text{for all } t \leq t'. \tag{16.9}$$

Here, our notation ":=" means that we take the current value of $\overline{V}_t^n(S_t)$, add $\alpha_{n-1}\lambda^{t'-t}\delta_{t'}$ to it to obtain an updated value of $\overline{V}_t^n(S_t)$. When we reach time $t' = T$, our value functions would have undergone a complete update. We note that at time t', we need to update the value function for every $t \leq t'$.

16.1.5 TD(0) and approximate value iteration

An important special case of TD(λ) occurs when we use $\lambda = 0$. In this case,

$$\overline{V}_t^n(S_t^n) \quad = \quad \overline{V}_t^{n-1}(S_t^n) + \alpha_{n-1}\big(C(S_t^n, x_t^n) + \gamma\overline{V}_{t+1}^{n-1}(S^M(S_t^n, x_t^n, W_{t+1}^n)) - \overline{V}_t^{n-1}(S_t^n)\big). \tag{16.10}$$

Now consider value iteration. In chapter 14, when we did not have to deal with Monte Carlo samples and statistical noise, value iteration (for a fixed policy) looked like

$$V_t^n(s) = C(s, X^\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p^\pi(s'|s)V_{t+1}^n(s').$$

In steady state, we would write it as

$$V^n(s) = C(s, X^\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p^\pi(s'|s)V^{n-1}(s').$$

When we use approximate dynamic programming, we are following a sample path that puts us in state S_t^n, where we observe a sample realization of a contribution $\hat{C}_t^n$, after which we observe a sample realization of the next downstream state S_{t+1}^n (the decision is determined by our fixed policy). A sample observation of the value of being in state S_t^n would be computed using

$$\hat{v}_t^n = C(S_t^n, x_t^n) + \gamma\overline{V}_{t+1}^{n-1}(S_{t+1}^n).$$

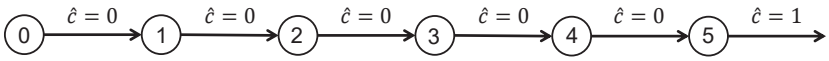

Figure 16.2 Five-state Markov chain for illustrating backward learning.

We can then use this to update our estimate of the value of being in state S_t^n using

$$\begin{aligned}\overline{V}_t^n(S_t^n) &= (1-\alpha_{n-1})\overline{V}_t^{n-1}(S_t^n) + \alpha_{n-1}\hat{v}_t^n \\ &= (1-\alpha_{n-1})\overline{V}_t^{n-1}(S_t^n) + \\ &\quad \alpha_{n-1}\big(C(S_t^n, x_t^n) + \gamma\overline{V}^{n-1}(S^M(S_t^n, x_t^n, W_{t+1}^n))\big). \qquad (16.11)\end{aligned}$$

It is not hard to see that (16.10) and (16.11) are the same. The idea is popular because it is particularly easy to implement. It is also well suited to high-dimensional decision vectors x, as we illustrate in chapter 18.

Temporal difference learning derives its name because $\overline{V}^{n-1}(S)$ is viewed as the "current" value of being in state S, while $C(S,x) + \overline{V}^{n-1}(S^M(S,x,W))$ is viewed as the updated value of being in state S. The difference $\overline{V}^{n-1}(S) - (C(S,x) + \overline{V}^{n-1}(S^M(S,x,W)))$ is the difference in these estimates across iterations (or time), hence the name. TD(0) is a form of statistical bootstrapping, because rather than simulate the full trajectory, it depends on the current estimate of the value $\overline{V}^{n-1}(S^M(S,x,W))$ of being in the downstream state $S^M(S,x,W)$.

While TD(0) can be very easy to implement, it can also produce very slow convergence. The effect is illustrated using the simple five-state Markov chain shown in figure 16.2, where the contribution of the transitions out of states 0 through 4, denoted by $\hat{c}$, are always 0, and then we receive 1 when we make the final transition out of state 5. When we apply TD(0) updating to estimate the value of each state, we produce the set of numbers shown in table 16.1. In this illustration, there are no decisions and the contribution is zero for every other time period. A stepsize of $1/n$ was used throughout.

Table 16.1 illustrates that the rate of convergence for $\overline{V}_0$ is dramatically slower than for $\overline{V}_4$. The reason is that as we smooth $\hat{v}_t$ into $\overline{V}_{t-1}$, the stepsize has a discounting effect. The problem is most pronounced when the value of being in a state at time t depends on contributions that are a number of steps into the future (imagine the challenge of training a value function to play the game of chess). For problems with long horizons, and in particular those where it takes many steps before receiving a reward, this bias can be so serious that it can appear that temporal differencing (and algorithms that use it) simply does not work. We can partially overcome the slow convergence by carefully choosing a stepsize rule. Stepsizes are discussed in depth in chapter 6. See in particular the OSAVI stepsize policy (section 6.4) which is designed specifically for estimating value functions.

16.1.6 TD learning for infinite horizon problems

We can perform updates using a general TD(λ) strategy as we did for finite horizon problems. However, there are some subtle differences. With finite horizon problems, it is common to assume that we are estimating a different function $\overline{V}_t$ for each time period t. As we step through time, we obtain information that can be used for a value function at

Iteration	$\overline{V}_0$	$\hat{v}_1$	$\overline{V}_1$	$\hat{v}_2$	$\overline{V}_2$	$\hat{v}_3$	$\overline{V}_3$	$\hat{v}_4$	$\overline{V}_4$	$\hat{v}_5$
0	0.000		0.000		0.000		0.000		0.000	1
1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	1
2	0.000	0.000	0.000	0.000	0.000	0.000	0.500	1.000	1.000	1
3	0.000	0.000	0.000	0.000	0.167	0.500	0.667	1.000	1.000	1
4	0.000	0.000	0.042	0.167	0.292	0.667	0.750	1.000	1.000	1
5	0.008	0.042	0.092	0.292	0.383	0.750	0.800	1.000	1.000	1
6	0.022	0.092	0.140	0.383	0.453	0.800	0.833	1.000	1.000	1
7	0.039	0.140	0.185	0.453	0.507	0.833	0.857	1.000	1.000	1
8	0.057	0.185	0.225	0.507	0.551	0.857	0.875	1.000	1.000	1
9	0.076	0.225	0.261	0.551	0.587	0.875	0.889	1.000	1.000	1
10	0.095	0.261	0.294	0.587	0.617	0.889	0.900	1.000	1.000	1

Table 16.1 Effect of stepsize on backward learning

a *specific* point in time. With stationary problems, each transition produces information that can be used to update the value function, which is then used in all future updates. By contrast, if we update $\overline{V}_t$ for a finite horizon problem, then this update is not used until the next forward pass through the states.

When we move to infinite horizon problems, we drop the indexing by t. Instead of stepping forward in time, we step through iterations, where at each iteration we generate a temporal difference

$$\delta^n = C(s^n, x^n) + \gamma \overline{V}^{n-1}(S^{M,x}(s^n, x^n)) - \overline{V}^{n-1}(s^n).$$

To do a proper update of the value function at each state, we would have to use an infinite series of the form

$$\overline{V}^n(s) = \overline{V}^{n-1}(s) + \alpha_n \sum_{m=0}^{\infty} (\gamma\lambda)^m \delta^{n+m}, \tag{16.12}$$

where we can use any initial starting state $s^0 = s$. Of course, we would use the same update for each state s^m that we visit, so we might write

$$\overline{V}^n(s^m) = \overline{V}^{n-1}(s^m) + \alpha_n \sum_{n=m}^{\infty} (\gamma\lambda)^{(n-m)} \delta^n. \tag{16.13}$$

Equations (16.12) and (16.13) both imply stepping forward in time (presumably a "large" number of iterations) and computing temporal differences before performing an update. A more natural way to run the algorithm is to do the updates incrementally. After we compute δ^n, we can update the value function at each of the previous states we visited. So, at iteration n, we would execute

$$\overline{V}^n(s^m) := \overline{V}^n(s^m) + \alpha_n(\gamma\lambda)^{n-m}\delta^m, \quad m = n, n-1, \ldots, 1. \tag{16.14}$$

We can now use the temporal difference δ^n to update the estimate of the value function for every state we have visited up to iteration n.

Figure 16.3 outlines the basic structure of a TD(λ) algorithm for an infinite horizon problem. Step 1 begins by computing the first post-decision state, after which step 2 makes

Step 0. Initialization:

Step 0a. Initialize $\overline{V}^0(S)$ for all S.

Step 0b. Initialize the state S^0.

Step 0c. Set $n = 1$.

Step 1. Choose ω^n.

Step 2. Solve

$$x^n = \arg\max_{x \in \mathcal{X}^n} \left(C(S^n, x) + \gamma \overline{V}^{n-1}(S^{M,x}(S^n, x))\right). \tag{16.15}$$

Step 3. Compute the temporal difference for this step:

$$\delta^n = C(S^n, x^n) + \gamma \left(\overline{V}^{n-1}(S^{M,x}(S^n, x^n)) - \overline{V}^{n-1}(S^n)\right).$$

Step 4. Update $\overline{V}$ for $m = n, n-1, \ldots, 1$:

$$\overline{V}^n(S^m) \quad = \quad \overline{V}^{n-1}(S^m) + (\gamma\lambda)^{n-m}\delta^n. \tag{16.16}$$

Step 5. Compute $S^{n+1} = S^M(S^n, x^n, W(\omega^n))$.

Step 6. Let $n = n + 1$. If $n < N$, go to step 1.

Figure 16.3 A TD(λ) algorithm for infinite horizon problems.

a single step forward. After computing the temporal-difference in step 3, we traverse previous states we have visited in Step 4 to update their value functions.

In step 3, we update all the states $(S^m)_{m=1}^n$ that we have visited up to then. Thus, at iteration n, we would have simulated the partial update

$$\overline{V}^n(S^0) \quad = \quad \overline{V}^{n-1}(S^0) + \alpha_{n-1} \sum_{m=0}^{n} (\gamma\lambda)^m \delta^m. \tag{16.17}$$

This means that at any iteration n, we have updated our values using biased sample observations (as is generally the case in value iteration). We avoided this problem for finite horizon problems by extending out to the end of the horizon. We can obtain unbiased updates for infinite horizon problems by assuming that all policies eventually put the system into an "absorbing state." For example, if we are modeling the process of holding or selling an asset, we might be able to guarantee that we eventually sell the asset.

One subtle difference between temporal difference learning for finite horizon and infinite horizon problems is that in the infinite horizon case, we may be visiting the same state two or more times on the same sample path. For the finite horizon case, the states and value functions are all indexed by the time that we visit them. Since we step forward through time, we can never visit the same state at the same point in time twice in the same sample path. By contrast, it is quite easy in a steady-state problem to revisit the same state over and over again. For example, we could trace the path of our nomadic trucker (introduced in section 2.3.4), who might go back and forth between the same pair of locations in the same sample path. As a result, we are using the value function to determine what state to visit, but at the same time we are updating the value of being in these states.

16.2 STOCHASTIC APPROXIMATION METHODS

A central idea in recursive estimation is the use of stochastic approximation methods and stochastic gradients. We have already seen this in one setting in the chapter on derivative-based stochastic optimization in section 5.3.1. We review the idea again here, but in a different context. We begin with the same stochastic optimization problem, which we originally introduced as the problem

$$\min_x \mathbb{E}F(x, W).$$

Now assume that we are choosing a scalar value v to solve the problem

$$\min_v \mathbb{E}F(v, \hat{V}), \tag{16.18}$$

where

$$F(v, \hat{V}) = \frac{1}{2}(v - \hat{V})^2,$$

and where $\hat{V}$ is a random variable with unknown mean. We would like to use a series of sample realizations $\hat{v}^n$ to guide an algorithm that generates a sequence v^n that converges to the optimal solution v^* that solves (16.18). We use the same basic strategy as we introduced in section 5.3.1 where we update v^n using

$$\begin{aligned} v^n &= v^{n-1} - \alpha_{n-1}\nabla F(v^{n-1}, \hat{v}^n) \\ &= v^{n-1} - \alpha_{n-1}(v^{n-1} - \hat{v}^n). \end{aligned} \tag{16.19}$$

Now if we make the transition that instead of updating a scalar v^n, we are updating $\overline{V}_t^n(S_t^n)$. This produces the updating equation

$$\overline{V}_t^n(S_t^n) = \overline{V}_t^{n-1}(S_t^n) - \alpha_{n-1}(\overline{V}_t^{n-1}(S_t^n) - \hat{v}^n). \tag{16.20}$$

If we use $\hat{v}^n = C(S_t^n, x_t^n) + \gamma \overline{V}^{n-1}(S_{t+1}^n)$, we quickly see that the updating equation produced using our stochastic gradient algorithm (16.20) gives us the same update that we obtained using temporal difference learning (equation (16.10)) and approximate value iteration (equation (16.11)). In equation (16.19), α_n is called a stepsize, because it controls how far we go in the direction of $\nabla F(v^{n-1}, \hat{v}^n)$, and for this reason this is the term that we adopt for α_n throughout this book. In contrast to our first use of this idea in section 5.3, where the stepsize had to serve a scaling function, in this setting the units of the variable being optimized, v^n, and the units of the gradient are the same. Indeed, we can expect that $0 < \alpha_n \leq 1$, which is a major simplification.

Now consider what happens when we replace the lookup table representation $\overline{V}(s)$ that we used above, with a linear regression $\overline{V}(s|\theta) = \theta^T\phi$. Now we want to find the best value of θ, which we can do by solving

$$\min_\theta \mathbb{E}\frac{1}{2}(\overline{V}(s|\theta) - \hat{v})^2.$$

Applying a stochastic gradient algorithm, we obtain the updating step

$$\theta^n = \theta^{n-1} - \alpha_{n-1}(\overline{V}(s|\theta^{n-1}) - \hat{v}^n)\nabla_\theta \overline{V}(s|\theta^n). \tag{16.21}$$

Since $\overline{V}(s|\theta^n) = \sum_{f\in\mathcal{F}} \theta_f^n \phi_f(s) = (\theta^n)^T\phi(s)$, the gradient with respect to θ is given by

$$\nabla_\theta \overline{V}(s|\theta^n) = \begin{pmatrix} \frac{\partial \overline{V}(s|\theta^n)}{\partial \theta_1} \\ \frac{\partial \overline{V}(s|\theta^n)}{\partial \theta_2} \\ \vdots \\ \frac{\partial \overline{V}(s|\theta^n)}{\partial \theta_F} \end{pmatrix} = \begin{pmatrix} \phi_1(s^n) \\ \phi_2(s^n) \\ \vdots \\ \phi_F(s^n) \end{pmatrix} = \phi(s^n).$$

Thus, the updating equation (16.21) is given by

$$\begin{aligned} \theta^n &= \theta^{n-1} - \alpha_{n-1}(\overline{V}(s|\theta^{n-1}) - \hat{v}^n)\phi(s^n) \\ &= \theta^{n-1} - \alpha_{n-1}(\overline{V}(s|\theta^{n-1}) - \hat{v}^n) \begin{pmatrix} \phi_1(s^n) \\ \phi_2(s^n) \\ \vdots \\ \phi_F(s^n) \end{pmatrix}. \end{aligned} \tag{16.22}$$

Using a stochastic gradient algorithm requires that we have some starting estimate θ^0 for the parameter vector, although $\theta^0 = 0$ is a common choice.

While this is a simple and elegant algorithm, we have reintroduced the problem of scaling. Just as we encountered in section 5.3, the units of θ^{n-1} and the units of $(\overline{V}(s|\theta^{n-1}) - \hat{v}^n)\phi(s^n)$ may be completely different. What we have learned about stepsizes still applies, except that we may need an initial stepsize that is quite different than 1.0 (our common starting point).

Our experimental work has suggested that the following policy works well: When you choose a stepsize formula, scale the first value of the stepsize so that the change in θ^n in the early iterations of the algorithm is approximately 20 to 50 percent (you will typically need to observe several iterations). You want to see individual elements of θ^n moving consistently in the same direction during the early iterations. If the stepsize is too large, the values can swing wildly, and the algorithm may not converge at all. If the changes are too small, the algorithm may simply stall out. It is very tempting to run the algorithm for a period of time and then conclude that it appears to have converged (presumably to a good solution). While it is important to see the individual elements moving in the same direction (consistently increasing or decreasing) in the early iterations, it is also important to see oscillatory behavior toward the end.

16.3 BELLMAN'S EQUATION USING A LINEAR MODEL*

It is possible to solve Bellman's equation for infinite horizon problems by starting with the assumption that the value function is given by a linear model $V(s) = \theta^T\phi(s)$ where $\Phi(s)$ is a column vector of basis functions for a particular state s. Of course, we are still working with a single policy, so we are using Bellman's equation only as a method for finding the best linear approximation for the infinite horizon value of a fixed policy π.

We begin with a derivation based on matrix linear algebra, which is more advanced and which does not produce expressions that can be implemented in practice. We follow this discussion with a simulation-based algorithm which can be implemented fairly easily.

16.3.1 A matrix-based derivation**

In section 16.5.2, we provided a geometric view of basis functions, drawing on the elegance and obscurity of matrix linear algebra. We are going to continue this presentation and present a version of Bellman's equation assuming linear models. However, we are not yet ready to introduce the dimension of optimizing over policies, so we are still simply trying to approximate the value of being in a state. Also, we are only considering infinite horizon models, since we have already handled the finite horizon case. This presentation can be viewed as another method for handling infinite horizon models, while using a linear architecture to approximate the value function.

First recall that Bellman's equation (for a fixed policy) is written

$$V^\pi(s) = C(s, X^\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, X^\pi(s)) V^\pi(s').$$

In vector-matrix form, we let V^π be a vector with element $V^\pi(s)$, we let c^π be a vector with element $C(s, X^\pi(s))$ and finally we let P^π be the one-step transition matrix with element $p(s'|s, X^\pi(s))$ at row s, column s'. Using this notation, Bellman's equation becomes

$$V^\pi = c^\pi + \gamma P^\pi V^\pi,$$

allowing us to solve for V^π using

$$V^\pi = (I - \gamma P^\pi)^{-1} c^\pi.$$

This works with a lookup-table representation (a value for each state). Now assume that we replace V^π with an approximation $\overline{V}^\pi = \Phi\theta$ where, Φ is a $|\mathcal{S}| \times |\mathcal{F}|$ matrix with element $\Phi_{s,f} = \phi_f(s)$. Also let

d_s^π = the steady state probability of being in state s while following policy π,

D^π = a $|\mathcal{S}| \times |\mathcal{S}|$ diagonal matrix where the state probabilities $(d_1^\pi, \ldots, d_{|\mathcal{S}|}^\pi)$ make up the diagonal.

We would like to choose θ to minimize the weighted sum of errors squared, where the error for state s is given by

$$\epsilon^n(s) = \sum_f \theta_f \phi_f(s) - \left(c^\pi(s) + \gamma \sum_{s' \in \mathcal{S}} p^\pi(s'|s, X^\pi) \sum_f \theta_f^n \phi_f(s') \right). \tag{16.23}$$

The first term on the right hand side of (16.23) is the predicted value of being in each state given θ, while the second term on the right hand side is the "predicted" value computed using the one-period contribution plus the expected value of the future which is computed using θ^n. The expected sum of errors squared is then given by

$$\min_\theta \sum_{s \in \mathcal{S}} d_s^\pi \left(\sum_f \theta_f \phi_f(s) - \left(c^\pi(s) + \gamma \sum_{s' \in \mathcal{S}} p^\pi(s'|s, X^\pi) \sum_f \theta_f^n \phi_f(s') \right) \right)^2,$$

In matrix form, this can be written

$$\min_\theta (\Phi\theta - (c^\pi + \gamma P^\pi \Phi\theta^n))^T D^\pi (\Phi\theta - (c^\pi + \gamma P^\pi \Phi\theta^n)) \tag{16.24}$$

where D^π is a $|\mathcal{S}| \times |\mathcal{S}|$ diagonal matrix with elements d_s^π which serves a scaling role (we want to focus our attention on states we visit the most). We can find the optimal value of θ (given θ^n) by taking the derivative of the function being optimized in (16.24) with respect to θ and setting it equal to zero. Let θ^{n+1} be the optimal solution, which means we can write

$$\Phi^T D^\pi \big(\Phi\theta^{n+1} - (c^\pi + \gamma P^\pi \Phi\theta^n)\big) = 0, \tag{16.25}$$

We can find a fixed point $\lim_{n\rightarrow\infty} \theta^n = \lim_{n\rightarrow\infty} \theta^{n+1} = \theta^*$, which allows us to write equation (16.25) in the form

$$A\theta^* = b, \tag{16.26}$$

where $A = \Phi^T D^\pi (I - \gamma P^\pi)\Phi$ and $b = \Phi^T D^\pi c^\pi$. This allows us, in theory at least, to solve for θ^* using

$$\theta^* = A^{-1}b, \tag{16.27}$$

which can be viewed as a scaled version of the normal equations (equation 3.40). Equation (16.27) is very similar to our calculation of the steady state value of being in each state introduced in chapter 14, given by

$$V^\pi = (I - \gamma P^\pi)^{-1} c^\pi.$$

Equation (16.27) differs only in the scaling by the probability of being in each state (D^π) and then the transformation to the feature space by Φ.

We note that equation (16.25) can also be written in the form

$$A\theta - b = \Phi^T D^\pi \left(\Phi\theta - (c^\pi + \gamma P^\pi \Phi\theta)\right). \tag{16.28}$$

The term $\Phi\theta$ can be viewed as the approximate value of each state. The term $(c^\pi + \gamma P^\pi \Phi\theta)$ can be viewed as the one-period contribution plus the expected value of the state that you transition to under policy π, again computed for each state. Let δ^π be a column vector containing the temporal difference for each state when we choose a decision according to policy π. By tradition, the temporal difference has always been written in the form $C(S_t, x) + \overline{V}(S_{t+1}) - \overline{V}(S_t)$, which can be thought of as "estimated minus predicted." If we continue to let δ^π be the traditional definition of the temporal difference, it would be written

$$\delta^\pi = -\big(\Phi\theta - (c^\pi + \gamma P^\pi \Phi\theta)\big). \tag{16.29}$$

The pre-multiplication of δ^π by D^π in (16.28) has the effect of factoring each temporal difference by the probability that we are in each state. Then pre-multiplying $D^\pi\delta^\pi$ by Φ^T has the effect of transforming this scaled temporal difference for each state into the feature space.

The goal is to find the value θ that produces $A\theta - b = 0$, which means we are trying to find the value θ that produces a scaled version of $\Phi\theta - (c^\pi + \gamma P^\pi \Phi\theta) = 0$, but transformed to the feature space.

Linear algebra offers a compact elegance, but at the same time can be hard to parse, and for this reason we encourage the reader to stop and think about the relationships. One useful exercise is to think of a set of basis functions where we have a "feature" for each state, with $\phi_f(s) = 1$ if feature f corresponds to state s. In this case, Φ is the identity

matrix. D^π, the diagonal matrix with diagonal elements d_s^π giving the probability of being in state s, can be viewed as scaling quantities for each state by the probability of being in a state. If Φ is the identity matrix, then $A = D^\pi - \gamma D^\pi P^\pi$ where $D^\pi P^\pi$ is the matrix of *joint* probabilities of being in state s *and* then transitioning to state s'. The vector b becomes the vector of the cost of being in each state (and then taking the a corresponding to policy π) times the probability of being in the state.

When we have a smaller set of basis functions, then multiplying c^π or $D^\pi(I - \gamma P^\pi)$ times Φ has the effect of scaling quantities that are indexed by the state into the feature space, which also transforms an $|\mathcal{S}|$-dimensional space into an $|\mathcal{F}|$-dimensional space.

16.3.2 A simulation-based implementation

No-one actually computes expressions such as those given in section 16.3.1. In practice, we simulate everything.

We start by simulating a trajectory of states, decisions and information,

$$(S^0, x^0, W^1, S^1, x^1, W^2, \ldots, S^n, x^n, W^{n+1}).$$

Recall that $\phi(s)$ is a column vector with an element $\phi_f(s)$ for each feature $f \in \mathcal{F}$. Using our simulation above, we also obtain a sequence of column vectors $\phi(s^i)$ and contributions $C(S^i, x^i)$. We can create a sample estimate of the $|\mathcal{F}|$ by $|\mathcal{F}|$ matrix A in the section above using

$$A^n = \frac{1}{n}\sum_{i=0}^{n-1} \phi(S^i)(\phi(S^i) - \gamma\phi(S^{i+1}))^T. \tag{16.30}$$

We can also create a sample estimate of the vector b using

$$b^n = \frac{1}{n}\sum_{i=0}^{n-1} \phi(S^i)C(S^i, x^i). \tag{16.31}$$

To gain some intuition, again stop and assume that there is a feature for every state, which means that $\phi(S^i)$ is a vector of 0's with a 1 corresponding to the element for state i, which means it is a kind of indicator variable telling us what state we are in. The term $(\phi(S^i) - \gamma\phi(S^{i+1}))$ is then a simulated version of $D^\pi(I - \gamma P^\pi)$, weighted by the probability that we are in a particular state, where we replace the probability of being in a state with a sampled realization of actually being in a particular state.

We are going to use this foundation to introduce two important algorithms for infinite horizon problems when using linear models to approximate value function approximations. These are known as *least squares temporal differences* (LSTD), and *least squares policy evaluation* (LSPE).

16.3.3 Least squares temporal differences (LSTD)

As long as A^n is invertible (which is not guaranteed), we can compute a sample estimate of θ using

$$\theta^n = (A^n)^{-1}b^n. \tag{16.32}$$

This algorithm is known in the literature as *least squares temporal differences*. As long as the number of features is not too large (as is typically the case), the inverse is not too

hard to compute. LSTD can be viewed as a batch algorithm which operates by collecting a sample of temporal differences, and then using least squares regression to find the best linear fit.

We can see the role of temporal differences more clearly by doing a little algebra. We use equations (16.30) and (16.31) to write

$$\begin{aligned} A^n\theta^n - b^n &= \frac{1}{n}\sum_{i=0}^{n-1}\left(\phi(S^i)(\phi(S^i) - \gamma\phi(S^{i+1}))^T\theta^n - \phi(S^i)C(S^i,x^i)\right) \\ &= \frac{1}{n}\sum_{i=0}^{n-1}\phi(S^i)\left(\phi(S^i)^T\theta^n - (c^\pi + \alpha\phi(S^{i+1})^T\theta^n)\right) \\ &= \frac{1}{n}\sum_{i=0}^{n-1}\phi(S^i)\delta^i(\theta^n), \end{aligned}$$

where $\delta^i(\theta^n) = \phi(S^i)^T\theta^n - (c^\pi + \alpha\phi(S^{i+1})^T\theta^n)$ is the i^{th} temporal difference given the parameter vector θ^n. Thus, we are doing a least squares regression so that the sum of the temporal differences over the simulation (which approximations the expectation) is equal to zero. We would, of course, like it if θ could be chosen so that $\delta^i(\theta) = 0$ for all i. However, when working with sample realizations the best we can expect is that the average across the observations of $\delta^i(\theta)$ tends to zero.

16.3.4 Least squares policy evaluation (LSPE)

LSTD is basically a batch algorithm, which requires collecting a sample of n observations and then using regression to fit a model. An alternative strategy uses a stochastic gradient algorithm which successively updates estimates of θ. The basic updating equation is

$$\theta^n = \theta^{n-1} - \frac{\alpha}{n}G^n\sum_{i=0}^{n-1}\phi(S^i)\delta^i(n), \tag{16.33}$$

where G^n is a scaling matrix. Although there are different strategies for computing G^n, the most natural is a simulation-based estimate of $(\Phi^T D^\pi \Phi)^{-1}$ which can be computed using

$$G^n = \left(\frac{1}{n+1}\sum_{i=0}^{n}\phi(S^i)\phi(S^i)^T\right)^{-1}.$$

To visualize G^n, return again to the assumption that there is a feature for every state. In this case, $\phi(S^i)\phi(S^i)^T$ is an $|\mathcal{S}|$ by $|\mathcal{S}|$ matrix with a 1 on the diagonal for row S^i and column S^i. As n approaches infinity, the matrix

$$\left(\frac{1}{n+1}\sum_{i=0}^{n}\phi(S^i)\phi(S^i)^T\right)$$

approaches the matrix D^π of the probability of visiting each state, stored in elements along the diagonal.

16.4 ANALYSIS OF TD(0), LSTD AND LSPE USING A SINGLE STATE*

A useful exercise to understand the behavior of recursive least squares, LSTD and LSPE is to consider what happens when they are applied to a trivial dynamic program with a single state and a single decision. Obviously, we are interested in the policy that chooses the single decision. This dynamic program is equivalent to computing the sum

$$F = \mathbb{E}\sum_{i=0}^{\infty}\gamma^i\hat{C}^i, \tag{16.34}$$

where $\hat{C}^i$ is a random variable giving the i^{th} contribution. If we let $\bar{c} = \mathbb{E}\hat{C}^i$, then clearly $F = \frac{1}{1-\gamma}\bar{c}$. But let's pretend that we do not know this, and we are using these various algorithms to compute the expectation.

16.4.1 Recursive least squares and TD(0)

Let $\hat{v}^n$ be an estimate of the value of being in state S^n. We continue to assume that the value function is approximated using

$$\overline{V}(s) = \sum_{f\in\mathcal{F}}\theta_f\phi_f(s).$$

We wish to choose θ by solving

$$\min_{\theta}\sum_{i=1}^{n}\left(\hat{v}^i - \left(\sum_{f\in\mathcal{F}}\theta_f\phi_f(S^i)\right)\right)^2.$$

Let θ^n be the optimal solution. We can determine this recursively using the techniques presented earlier in this chapter which gives us the updating equation

$$\theta^n = \theta^{n-1} - \frac{1}{1+(x^n)^TM^{n-1}x^n}M^{n-1}x^n(\overline{V}^{n-1}(S^n) - \hat{v}^n) \tag{16.35}$$

where $x^n = (\phi_1(S^n),\ldots,\phi_f(S^n),\ldots,\phi_F(S^n))$, and the matrix M^n is computed using

$$M^n = M^{n-1} - \frac{1}{1+(x^n)^TM^{n-1}x^n}\left(M^{n-1}x^n(x^n)^TM^{n-1}\right).$$

If we have only one state and one decision, we only have one basis function $\phi(s) = 1$ and one parameter $\theta^n = \overline{V}^n(s)$. Now the matrix M^n is a scalar and equation (16.35) reduces to

$$\begin{aligned} v^n &= v^{n-1} - \frac{M^{n-1}}{1+M^{n-1}}(v^{n-1} - \hat{v}^n) \\ &= \left(1 - \frac{M^{n-1}}{1+M^{n-1}}\right)v^{n-1} + \frac{M^{n-1}}{1+M^{n-1}}. \end{aligned}$$

If $M^0 = 1$, then $M^{n-1} = 1/n$, giving us

$$v^n = \frac{n-1}{n}v^{n-1} + \frac{1}{n}\hat{v}^n.$$

Imagine now that we are using TD(0) where $\hat{v}^n = \hat{C}^n + \gamma v^{n-1}$. In this case, we obtain

$$v^n = \left(1 - (1-\gamma)\frac{1}{n}\right) v^{n-1} + \frac{1}{n}\hat{C}^n. \tag{16.36}$$

Equation (16.36) can be viewed as an algorithm for finding

$$v = \sum_{n=0}^{\infty} \gamma^n \hat{C}^n,$$

where the solution is $v^* = \frac{1}{1-\gamma}\mathbb{E}\hat{C}$.

Equation (16.36) shows us that recursive least squares, when $\hat{v}^n$ is computed using temporal difference learning, has the effect of successively adding sample realizations of costs, with a "discount factor" of $1/n$. The factor $1/n$ arises directly as a result of the need to smooth out the noise in $\hat{C}^n$. For example, if $\hat{C} = c$ is a known constant, we could use standard value iteration, which would give us

$$v^n = c + \gamma v^{n-1}. \tag{16.37}$$

It is easy to see that v^n in (16.37) will rise much more quickly toward v^* than the algorithm in equation (16.36). We return to this topic in some depth in chapter 6.

16.4.2 LSPE

LSPE requires that we first generate a sequence of states S^i and contributions $\hat{C}^i$ for $i = 1, \ldots, n$. We then compute θ by solving the regression problem

$$\theta^n = \arg\min_{\theta} \sum_{i=1}^{n} \left(\sum_f \theta_f \phi_f(S^i) - \left(\hat{C}^i + \gamma \overline{V}^{n-1}(S^{i+1})\right)\right)^2.$$

For a problem with one state where $\theta^n = v^n$, this reduces to

$$v^n = \arg\min_{\theta} \sum_{i=1}^{n} \left(\theta - \left(\hat{C}^i + \gamma v^{n-1}\right)\right)^2.$$

This problem can be solved in closed form, giving us

$$v^n = \left(\frac{1}{n}\sum_{i=1}^{n} \hat{C}^i\right) + \gamma v^{n-1}.$$

16.4.3 LSTD

Finally, we showed above that the LSTD procedure finds θ by solving the system of equations

$$\sum_{i=1}^{n} \phi_f(S^i)(\phi_f(S^i) - \gamma\phi_f(S^{i+1}))^T \theta^n = \sum_{i=1}^{n} \phi_f(S^i)\hat{C}^i,$$

for each $f \in \mathcal{F}$. Again, since we have only one basis function $\phi(s) = 1$ for our single state problem, this reduces to finding the scalar $\theta^n = v^n$ using

$$v^n = \frac{1}{1-\gamma}\left(\frac{1}{n}\sum_{i=1}^{n} \hat{C}^n\right).$$

16.4.4 Discussion

This presentation illustrates three different styles for estimating an infinite horizon sum. In recursive least squares, equation (16.35) demonstrates the successive smoothing of the previous estimate v^n and the latest estimate $\hat{v}^n$. We are, at the same time, adding contributions over time while also trying to smooth out the noise.

LSPE, by contrast, separates the estimation of the mean of the single period contribution, and the process of summing contributions over time. At each iteration, we improve our estimate of $\mathbb{E}\hat{C}$, and then accumulate our latest estimate in a telescoping sum.

LSTD, finally, updates its estimate of $\mathbb{E}\hat{C}$, and then projects this over the infinite horizon by factoring the result by $1/(1-\gamma)$.

16.5 GRADIENT-BASED METHODS FOR APPROXIMATE VALUE ITERATION*

There has been a strong desire for approximation algorithms with the following features:

1) Off-policy learning.

2) Temporal-difference learning.

3) Linear models for value function approximation.

4) Complexity (in memory and computation) that is linear in the number of features.

The last requirement is primarily of interest in specialized applications which require thousands or even millions of features. Off-policy learning is desirable because it provides an important degree of control over exploration. Temporal-difference learning is useful because it is so simple, as are the use of linear models, which make it possible to provide an estimate of the entire value function with a small number of measurements.

Off-policy, temporal-difference learning was first introduced in the form of Q-learning using a lookup table representation, where it is known to converge. But we lose this property if we introduce value function approximations that are linear in the parameters. In fact, Q-learning can be shown to diverge for any positive stepsize. The reason is that there is no guarantee that our linear model is accurate, which can introduce significant instabilities in the learning process.

We begin by describing how to estimate linear value functions using approximate value iteration. Then section 16.5.2 provides a geometric view of linear models.

16.5.1 Approximate value iteration with linear models**

Q-learning and temporal difference learning can be viewed as forms of stochastic gradient algorithms, but the problem with earlier algorithms when we use linear value function approximations can be traced to the choice of objective function. For example, if we wish to find the best linear approximation $\overline{V}(s|\theta)$, a hypothetical objective function would be to minimize the expected mean squared difference between $\overline{V}(s|\theta)$ and the true value function $V(s)$. If d_s^π is the probability of being in state s, this objective would be written

$$MSE(\theta) = \frac{1}{2}\sum_s d_s^\pi (\overline{V}(s|\theta) - V(s))^2.$$

If we are using approximate value iteration, a more natural objective function is to minimize the mean squared Bellman error. We use the Bellman operator $\mathcal{M}^\pi$ (as we did in chapter 14) for policy π to represent

$$\mathcal{M}^\pi v = c^\pi + \gamma P^\pi v,$$

where v is a column vector giving the value of being in state s, and c^π is the column vector of contributions $C(s, X^\pi(s))$ if we are in state s and choose a decision x according to policy π. This allows us to define

$$\begin{aligned} MSBE(\theta) &= \frac{1}{2}\sum_s d_s^\pi \left(\overline{V}(s|\theta) - (c^\pi(s) + \gamma \sum_{s'} p^\pi(s'|s)\overline{V}(s'|\theta))\right)^2 \\ &= \|\overline{V}(\theta) - \mathcal{M}\overline{V}(\theta)\|_D^2. \end{aligned}$$

We can minimize MSBE(θ) by generating a sequence of states $(S^1, \ldots, S^i, S^{i+1}, \ldots)$ and then computing a stochastic gradient

$$\nabla_\theta \text{MSBE}(\theta) = \delta^{\pi,i}(\phi(S^i) - \gamma\phi(S^{i+1})),$$

where $\phi(S^i)$ is a column vector of basis functions evaluated at state S^i. The scalar $\delta^{\pi,i}$ is the temporal difference given by

$$\delta^{\pi,i} = \overline{V}(S^i|\theta) - (c^\pi(S^i) + \gamma\overline{V}(S^{i+1}|\theta)).$$

We note that $\delta^{\pi,i}$ depends on the policy π which affects both the single period contribution and the likelihood of transitioning to state S^{i+1}. To emphasize that we are working with a fixed policy, we carry the superscript π throughout.

For this section, we are defining the temporal difference as

$$\delta^{\pi,i} = \overline{V}(S^i|\theta) - (c^\pi(S^i) + \gamma\overline{V}(S^{i+1}|\theta))$$

because it is a natural byproduct when deriving algorithms based on stochastic gradient methods. Earlier in this chapter, we defined the temporal difference as $\delta_\tau = C(S_\tau^n, x_\tau^n) + \overline{V}_{\tau+1}^{n-1}(S_{\tau+1}^n) - \overline{V}_\tau^{n-1}(S_\tau^n)$ (see equation (16.4)), which is more natural when used to represent telescoping sums (for example, see equation (16.5)). A stochastic gradient algorithm, then, would seek to optimize θ using

$$\begin{aligned} \theta^{n+1} &= \theta^n - \alpha_n \nabla_\theta \text{MSBE}(\theta) && (16.38) \\ &= \theta^n - \alpha_n \delta^{\pi,n}(\phi(S^n) - \gamma\phi(S^{n+1})). && (16.39) \end{aligned}$$

Were we to use the more traditional definition of a temporal difference, our equation would be written

$$\theta^{n+1} = \theta^n + \alpha_n \delta^{\pi,n}(\phi(S^n) - \gamma\phi(S^{n+1})),$$

which runs counter to the classical statement of a stochastic gradient algorithm (given in equation (16.38)) for minimization problems.

A variant of this basic algorithm, called the generalized TD(0) (or, GTD(0)) algorithm, is given by

$$\theta^{n+1} = \theta^n - \alpha_n(\phi(S^n) - \gamma\phi(S^{n+1}))\phi(S^n)^T u^n, \quad (16.40)$$

where

$$u^{n+1} = u^n - \beta_n(u^n - \delta^{\pi,n}\phi(S^n)). \tag{16.41}$$

α_n and β_n are both stepsizes. u^n is a smoothed estimate of the product $\delta^{\pi,n}\phi(S^n)$.

Gradient descent methods based on temporal differences will not minimize MSBE(θ) because there does not exist a value of θ that would allow $\hat{v}(s) = c^\pi(s) + \gamma\overline{V}(s|\theta)$ to be represented as $\overline{V}(s|\theta)$. We can fix this using the mean squared projected Bellman error (MSPBE(θ)) which we compute as follows. It is more compact to do this development using matrix-vector notation. We first recall the projection operator Π given by

$$\Pi = \Phi(\Phi^T D^\pi \Phi)^{-1}\Phi^T D^\pi.$$

(See section 16.5.2 for a derivation of this operator.) If V is a vector giving the value of being in each state, ΠV is the nearest projection of V on the space generated by $\theta\phi(s)$. We are trying to find $\overline{V}(\theta)$ that will match the one-step lookahead given by $\mathcal{M}^\pi\overline{V}(\theta)$, but this produces a column vector that cannot be represented directly as $\Phi\theta$, where Φ is the $|\mathcal{S}| \times |\mathcal{F}|$ matrix of feature vectors ϕ. We accomplish this by pre-multiplying $\mathcal{M}^\pi V(\theta)$ by the projection operator Π. This allows us to form the mean squared projected Bellman error using

$$MSPBE(\theta) = \frac{1}{2}\|\overline{V}(\theta) - \Pi\mathcal{M}^\pi\overline{V}(\theta)\|_D^2 \tag{16.42}$$

$$= \frac{1}{2}\big(\overline{V}(\theta) - \Pi\mathcal{M}^\pi\overline{V}(\theta)\big)^T D\big(\overline{V}(\theta) - \Pi\mathcal{M}^\pi\overline{V}(\theta)\big). \tag{16.43}$$

We can now use this new objective function as the basis of an optimization algorithm to find θ. Recall that D^π is a $|\mathcal{S}| \times |\mathcal{S}|$ diagonal matrix with elements d_s^π, giving us the probability that we are in state s while following policy π. We use D^π as a scaling matrix to give us the probability that we are in state s. We start by noting the identities

$$\begin{aligned}
\mathbb{E}[\phi\phi^T] &= \sum_{s\in\mathcal{S}} d_s^\pi \phi_s \phi_s^T \\
&= \Phi^T D^\pi \Phi. \\
\mathbb{E}[\delta^\pi\phi] &= \sum_{s\in\mathcal{S}} d_s^\pi \phi_s \left(c^\pi(s) + \gamma \sum_{s'\in\mathcal{S}} p^\pi(s'|s)\overline{V}(s'|\theta) - \overline{V}(s|\theta)\right) \\
&= \Phi^T D^\pi(\mathcal{M}^\pi\overline{V}(\theta) - \overline{V}(\theta)).
\end{aligned}$$

The derivations here and below make extensive use of matrices, which can be difficult to parse. A useful exercise is to write out the matrices assuming that there is a feature $\phi_f(s)$ for each state s, so that $\phi_f(s) = 1$ if feature f corresponds to state s. See exercise 16.12.

We see that the role of the scaling matrix D^π is to enable us to take the expectation of the quantities $\phi\phi^T$ and $\delta^\pi\phi$. Below, we are going to simulate these quantities, where a state will occur with probability d_s^π. We also use

$$\begin{aligned}
\Pi^T D^\pi \Pi &= (\Phi(\Phi^T D^\pi\Phi)^{-1}\Phi^T D^\pi)^T D^\pi(\Phi(\Phi^T D^\pi\Phi)^{-1}\Phi^T D^\pi) \\
&= (D^\pi)^T\Phi(\Phi^T D^\pi\Phi)^{-1}\Phi^T D^\pi\Phi(\Phi^T D^\pi\Phi)^{-1}\Phi^T D^\pi \\
&= (D^\pi)^T\Phi(\Phi^T D^\pi\Phi)^{-1}\Phi^T D^\pi.
\end{aligned}$$

We have one last painful piece of linear algebra that gives us a more compact form for MSPBE(θ). Pulling the $1/2$ to the left hand side (this will later vanish when we take the

derivative), we can write

$$\begin{aligned}
2MSPBE(\theta) &= \|\overline{V}(\theta) - \Pi\mathcal{M}^{\pi}\overline{V}(\theta)\|_D^2 \\
&= \|\Pi(\overline{V}(\theta) - \mathcal{M}^{\pi}\overline{V}(\theta))\|_D^2 \\
&= (\Pi(\overline{V}(\theta) - \mathcal{M}^{\pi}\overline{V}(\theta)))^T D^{\pi}(\Pi(\overline{V}(\theta) - \mathcal{M}^{\pi}\overline{V}(\theta))) \\
&= (\overline{V}(\theta) - \mathcal{M}^{\pi}\overline{V}(\theta))^T \Pi^T D^{\pi}\Pi(\overline{V}(\theta) - \mathcal{M}^{\pi}\overline{V}(\theta)) \\
&= (\overline{V}(\theta) - \mathcal{M}^{\pi}\overline{V}(\theta))^T (D^{\pi})^T \Phi(\Phi^T(D^{\pi})\Phi)^{-1}\Phi^T D^{\pi}(\overline{V}(\theta) - \mathcal{M}^{\pi}\overline{V}(\theta)) \\
&= (\Phi^T D^{\pi}(\mathcal{M}^{\pi}\overline{V}(\theta) - \overline{V}(\theta)))^T (\Phi^T D^{\pi}\Phi)^{-1}\Phi^T D^{\pi}(\mathcal{M}\overline{V}(\theta) - \overline{V}(\theta)) \\
&= \mathbb{E}[\delta^{\pi}\phi]^T \mathbb{E}[\phi\phi^T]^{-1}\mathbb{E}[\delta^{\pi}\phi]. \qquad (16.44)
\end{aligned}$$

We next need to estimate the gradient of this error $\nabla_{\theta} MSPBE(\theta)$. Keep in mind that $\delta^{\pi} = c^{\pi} + \gamma P^{\pi}\Phi\theta - \Phi\theta$. If ϕ is the column vector with element $\phi(s)$, assume that s' occurs with probability $p^{\pi}(s'|s)$ under policy π, and let ϕ' be the corresponding column vector. Differentiating (16.44) gives

$$\begin{aligned}
\nabla_{\theta} MSPBE(\theta) &= \mathbb{E}[(\gamma\phi' - \phi)\phi^T]\mathbb{E}[\phi\phi^T]^{-1}\mathbb{E}[\delta^{\pi}\phi] \\
&= -\mathbb{E}[(\phi - \gamma\phi')\phi^T]\mathbb{E}[\phi\phi^T]^{-1}\mathbb{E}[\delta^{\pi}\phi].
\end{aligned}$$

We are going to use a standard stochastic gradient updating algorithm for minimizing the error given by $MSPBE(\theta)$, which is given by

$$\begin{aligned}
\theta^{n+1} &= \theta^n - \alpha_n \nabla_{\theta} MSPBE(\theta) \qquad (16.45) \\
&= \theta^n + \alpha_n \mathbb{E}[(\phi - \gamma\phi')\phi^T]\mathbb{E}[\phi\phi^T]^{-1}\mathbb{E}[\delta^{\pi}\phi]. \qquad (16.46)
\end{aligned}$$

We can create a linear predictor which approximates

$$w \approx \mathbb{E}[\phi\phi^T]^{-1}\mathbb{E}[\delta^{\pi}\phi].$$

where w is approximated using

$$w^{n+1} = w^n + \beta_n(\delta^{\pi,n} - (\phi^n)^T w^n)\phi^n.$$

This allows us to write the gradient

$$\begin{aligned}
\nabla_{\theta} MSPBE(\theta) &= -\mathbb{E}[(\phi - \gamma\phi')\phi^T]\mathbb{E}[\phi\phi^T]^{-1}\mathbb{E}[\delta^{\pi}\phi] \\
&\approx -\mathbb{E}[(\phi - \gamma\phi')\phi^T]w.
\end{aligned}$$

We have now created the basis for two algorithms. The first is called generalized temporal difference 2 (GTD2), given by

$$\theta^{n+1} = \theta^n + \alpha_n(\phi^n - \gamma\phi^{n+1})((\phi^n)^T w^n). \qquad (16.47)$$

Here, ϕ^n is the column vector of basis functions when we are in state S^n, while ϕ^{n+1} is the column vector of basis functions for the next state S^{n+1}. Note that if equation (16.47) is executed right to left, all calculations are linear in the number of features F.

An important feature of the algorithm, especially for applications with large number of features, is that the algorithm is linear in the number of features.

A variant, called TDC (temporal difference with gradient corrector) is derived by using a slightly modified calculation of the gradient

$$\begin{aligned}\nabla_\theta MSPBE(\theta) &= -\mathbb{E}[(\phi-\gamma\phi')\phi^T]\mathbb{E}[\phi\phi^T]^{-1}\mathbb{E}[\delta^\pi\phi]\\ &= -\left(\mathbb{E}[\phi\phi^T]-\gamma\mathbb{E}[\phi'\phi^T]\right)\mathbb{E}[\phi\phi^T]^{-1}\mathbb{E}[\delta^\pi\phi]\\ &= -\left(\mathbb{E}[\delta^\pi\phi]-\gamma\mathbb{E}[\phi'\phi^T]\mathbb{E}[\phi\phi^T]^{-1}\mathbb{E}[\delta^\pi\phi]\right)\\ &\approx -\left(\mathbb{E}[\delta^\pi\phi]-\gamma\mathbb{E}[\phi'\phi^T]w\right).\end{aligned}$$

This gives us the TDC algorithm

$$\theta^{n+1} = \theta^n+\alpha_n\left(\delta^{\pi,n}\phi^n-\gamma\phi^{n'}((\phi^n)^Tw^n)\right). \tag{16.48}$$

GTD2 and TDC are both proven to converge to the optimal value of θ for a fixed implementatino policy $X^\pi(s)$ which may be different than the behavior (sampling) policy. That is, after updating θ^n where the temporal difference $\delta^{\pi,n}$ is computed assuming we are in state S^n and follow policy π, we are allowed to follow a separate behavior policy to determine S^{n+1}. This allows us to directly control the states that we visit, rather than depending on the decisions made by the implementation policy.

16.5.2 A geometric view of linear models*

For readers comfortable with linear algebra, we can obtain an elegant perspective on the geometry of basis functions. In section 3.7.1, we found the parameter vector θ for a regression model by minimizing the expected square of the errors between our model and a set of observations. Assume now that we have a "true" value function $V(s)$ which gives the value of being in state s, and let $p(s)$ be the probability of visiting state s. We wish to find the approximate value function that best fits $V(s)$ using a given set of basis functions $(\phi_f(s))_{f\in\mathcal{F}}$. If we minimize the expected square of the errors between our approximate model and the true value function, we would want to solve

$$\min_\theta F(\theta)=\sum_{s\in\mathcal{S}}p(s)\left(V(s)-\sum_{f\in\mathcal{F}}\theta_f\phi_f(s)\right)^2, \tag{16.49}$$

where we have weighted the error for state s by the probability of actually being in state s. Our parameter vector θ is unconstrained, so we can find the optimal value by taking the derivative and setting this equal to zero. Differentiating with respect to $\theta_{f'}$ gives

$$\frac{\partial F(\theta)}{\partial\theta_{f'}} = -2\sum_{s\in\mathcal{S}}p(s)\left(V(s)-\sum_{f\in\mathcal{F}}\theta_f\phi_f(s)\right)\phi_{f'}(s).$$

Setting the derivative equal to zero and rearranging gives

$$\sum_{s\in\mathcal{S}}p(s)V(s)\phi_{f'}(s)=\sum_{s\in\mathcal{S}}p(s)\sum_{f\in\mathcal{F}}\theta_f\phi_f(s)\phi_{f'}(s). \tag{16.50}$$

At this point, it is much more elegant to revert to matrix notation. Define an $|\mathcal{S}| \times |\mathcal{S}|$ diagonal matrix D where the diagonal elements are the state probabilities $p(s)$, as follows

$$D = \begin{pmatrix} p(1) & 0 & & 0 \\ 0 & p(2) & & 0 \\ \vdots & 0 & \cdots & \vdots \\ 0 & \vdots & & p(|\mathcal{S}|) \end{pmatrix}.$$

Let V be the column vector giving the value of being in each state

$$V = \begin{pmatrix} V(1) \\ V(2) \\ \vdots \\ V(|\mathcal{S}|) \end{pmatrix}.$$

Finally, let Φ be an $|\mathcal{S}| \times |\mathcal{F}|$ matrix of the basis functions given by

$$\Phi = \begin{pmatrix} \phi_1(1) & \phi_2(1) & & \phi_{|\mathcal{F}|}(1) \\ \phi_1(2) & \phi_2(2) & & \phi_{|\mathcal{F}|}(2) \\ \vdots & \vdots & \cdots & \vdots \\ \phi_1(|\mathcal{S}|) & \phi_2(|\mathcal{S}|) & & \phi_{|\mathcal{F}|}(|\mathcal{S}|) \end{pmatrix}.$$

Recognizing that equation (16.50) is for a particular feature f', with some care it is possible to see that equation (16.50) for all features is given by the matrix equation

$$\Phi^T DV = \Phi^T D\Phi\theta. \tag{16.51}$$

It helps to keep in mind that Φ is an $|\mathcal{S}| \times |\mathcal{F}|$ matrix, D is an $|\mathcal{S}| \times |\mathcal{S}|$ diagonal matrix, V is an $|\mathcal{S}| \times 1$ column vector, and θ is an $|\mathcal{F}| \times 1$ column vector. The reader should carefully verify that (16.51) is the same as (16.50).

Now, pre-multiply both sides of (16.51) by $(\Phi^T D\Phi)^{-1}$. This gives us the optimal value of θ as

$$\theta = (\Phi^T D\Phi)^{-1}\Phi^T DV. \tag{16.52}$$

This equation is closely analogous to the normal equations of linear regression, given by equation (3.37), with the only difference being the introduction of the scaling matrix D which captures the probability that we are going to visit a state.

Now, pre-multiply both sides of (16.52) by Φ, which gives

$$\Phi\theta = \overline{V} = \Phi(\Phi^T D\Phi)^{-1}\Phi^T DV.$$

$\Phi\theta$ is, of course, our approximation of the value function, which we have denoted by $\overline{V}$. This, however, is the best possible value function given the set of functions $\phi = (\phi_f)_{f\in\mathcal{F}}$. If the vector ϕ formed a complete basis over the space formed by the value function $V(s)$ and the state space $\mathcal{S}$, then we would obtain $\Phi\theta = \overline{V} = V$. Since this is generally not the case, we can view $\overline{V}$ as the nearest point projection (where "nearest" is defined as a weighted measure using the state probabilities $p(s)$) onto the space formed by the basis functions. In fact, we can form a projection operator Π defined by

$$\Pi = \Phi(\Phi^T D\Phi)^{-1}\Phi^T D$$

so that $\overline{V} = \Pi V$ is the value function closest to V that can be produced by the set of basis functions.

This discussion brings out the geometric view of basis functions (and at the same time, the reason why we use the term "basis function"). There is an extensive literature on basis functions that has evolved in the approximation literature.

16.6 VALUE FUNCTION APPROXIMATIONS BASED ON BAYESIAN LEARNING*

A different strategy for updating value functions is one based on Bayesian learning. Assume that we start with a prior $V^0(s)$ of the value of being in state s, and we assume that we have a known covariance function $Cov(s, s')$ that captures the relationship in our belief about $V(s)$ and $V(s')$. A good example where this function would be known might be a function where s is continuous (or a discretization of a continuous surface), where we might use

$$Cov(s, s') \propto e^{-\frac{\|s-s'\|^2}{b}} \tag{16.53}$$

where b is a bandwidth. This function captures the intuitive behavior that if two states are close to each other, their covariance is higher. So, if we make an observation that raises our belief about $V(s)$, then our belief about $V(s')$ will increase also, and will increase more if s and s' are close to each other. We also assume that we have a variance function $\lambda(s)$ that captures the noise in a measurement $\hat{v}(s)$ of the function at state s.

Our Bayesian updating model is designed for applications where we have access to observations $\hat{v}^n$ of our true function $V(s)$ which we can view as coming from our prior distribution of belief. This assumption effectively precludes using updating algorithms based on approximate value iteration, Q-learning and least squares policy evaluation. We cannot eliminate the bias, but below we describe how to minimize it. We then describe Bayesian updating using lookup tables and parametric models.

16.6.1 Minimizing bias for infinite horizon problems

We would very much like to have observations $\hat{v}^n(s)$ which we can view as an unbiased observation of $V(s)$. One way to do this is to build on the methods described in section 16.1.

To illustrate, assume that we have a policy π that determines the decision x_t we take when in state S_t, generating a contribution $\hat{C}_t^n$. Assume we simulate this policy for T time periods using

$$\hat{v}^n(T) = \sum_{t=0}^{T} \gamma^t \hat{C}_t.$$

If we have a finite horizon problem and T is the end of our horizon, then we are done. If our problem has an infinite horizon, we can project the infinite horizon value of our policy by first approximating the one-period contribution using

$$\bar{c}_T^n = \frac{1}{T} \sum_{t=0}^{T} \hat{C}_t^n.$$

Now assume this estimates the average contribution per period starting at time $T+1$. Our infinite-horizon estimate would be

$$\hat{v}^n = \hat{v}_0(T) + \gamma^{T+1}\frac{1}{1-\gamma}\bar{c}_T^n.$$

Finally, we use $\hat{v}^n$ to update our value function approximation $\overline{V}^{n-1}$ to obtain $\overline{V}^n$.

We next illustrate the Bayesian updating formulas for lookup tables and parametric models.

16.6.2 Lookup tables with correlated beliefs

Previously when we have used lookup tables, if we update the value $\overline{V}^n(s)$ for some state s, we do not use this information to update the values of any other states. With our Bayesian model, we can do much more if we have access to a covariance function such as the one we illustrated in equation (16.53).

Assume that we have discrete states, and assume that we have a covariance function $Cov(s,s')$ in the form of a covariance matrix Σ where $Cov(s,s') = \Sigma(s,s')$. Let V^n be our vector of beliefs about the value $V(s)$ of being in each state (we use V^n to represent our Bayesian beliefs, so that $\overline{V}^n$ can represent our frequentist estimates). Also let Σ^n be the covariance matrix of our belief about the vector V. If $\hat{v}^n(S^n)$ is an (approximately) unbiased sample observation of $V(s)$, the Bayesian formula for updating V^n is given by

$$\overline{V}^{n+1}(s) = V^n(s) + \frac{\hat{v}^n(S^n) - V^n(s)}{\lambda(S^n) + \Sigma^n(S^n,S^n)}\Sigma^n(s,S^n).$$

This has to be computed for each s (or at least each s where $\Sigma^n(s,S^n) > 0$). We update the covariance matrix using

$$\Sigma^{n+1}(s,s') = \Sigma^n(s,s') - \frac{\Sigma^n(s,S^n)\Sigma^n(S^n,s')}{\lambda(S^n) + \Sigma^n(S^n,S^n)}.$$

16.6.3 Parametric models

For most applications, a parametric model (specifically, a linear model) is going to be much more practical. Our frequentist updating equations for our regression vector θ^n were given above as

$$\theta^n = \theta^{n-1} - \frac{1}{\gamma^n}M^{n-1}\phi^n\hat{\varepsilon}^n, \tag{16.54}$$

$$M^n = M^{n-1} - \frac{1}{\gamma^n}(M^{n-1}\phi^n(\phi^n)^T M^{n-1}), \tag{16.55}$$

$$\gamma^n = 1 + (\phi^n)^T M^{n-1}\phi^n, \tag{16.56}$$

where $\hat{\varepsilon}^n = \overline{V}(\theta^{n-1})(S^n) - \hat{v}^n$ is the difference between our current estimate $\overline{V}(\theta^{n-1})(S^n)$ of the value function at our observed state S^n and our most recent observation $\hat{v}^n$. The adaptation for a Bayesian model is quite minor. The matrix M^n represents

$$M^n = [(X^n)^T X^n]^{-1}.$$

It is possible to show that the covariance matrix Σ^θ (which is dimensioned by the number of basis functions) is given by

$$\Sigma^\theta = M^n\lambda.$$

In our Bayesian model, λ is the variance of the difference between our observation $\hat{v}^n$ and the true value function $v(S^n)$, where we assume λ is known. This variance may depend on the state that we have observed, in which case we would write it as $\lambda(s)$, but in practice, since we do not know the function $V(s)$, it is hard to believe that we would be able to specify $\lambda(s)$. We replace M^n with $\Sigma^{\theta,n}$ and rescale γ^n to create the following set of updating equations

$$\theta^n = \theta^{n-1} - \frac{1}{\gamma^n}\Sigma^{\theta,n-1}\phi^n\hat{\varepsilon}^n, \tag{16.57}$$

$$\Sigma^{\theta,n} = \Sigma^{\theta,n-1} - \frac{1}{\gamma^n}(\Sigma^{\theta,n-1}\phi^n(\phi^n)^T\Sigma^{\theta,n-1}), \tag{16.58}$$

$$\gamma^n = \lambda + (\phi^n)^T\Sigma^{\theta,n-1}\phi^n. \tag{16.59}$$

16.6.4 Creating the prior

Approximate dynamic programming has been approached from a Bayesian perspective in the research literature, but otherwise has apparently received very little attention. We suspect that while there exist many applications in stochastic search where it is valuable to use a prior distribution of belief, it is much harder to build a prior on a value function.

Lacking any specific structural knowledge of the value function, we anticipate that the easiest strategy will be to start with $V^0(s) = v^0$, which is a constant across all states. There are several strategies we might use to estimate v^0. We might sample a state S^i at random, and find the best contribution $\hat{C}^i = \max_a C(S^i, a)$. Repeat this n times and compute

$$\bar{c} = \frac{1}{n}\sum_{i=1}^{n}\hat{C}^i.$$

Finally, let $v^0 = \frac{1}{1-\gamma}\bar{c}$ if we have an infinite horizon problem. The hard part is that the variance λ has to capture the variance of the difference between v^0 and the true $V(s)$. This requires having some sense of the degree to which v^0 differs from $V(s)$. We recommend being very conservative, which is to say choose a variance λ such that $v^0+2\sqrt{\lambda}$ easily covers what $V(s)$ might be. Of course, this also requires some judgment about the likelihood of visiting different states.

16.7 LEARNING ALGORITHMS AND STEPSIZES

A useful exercise to understand the behavior of recursive least squares, LSTD and LSPE is to consider what happens when they are applied to a trivial dynamic program with a single state and a single decision. Obviously, we are interested in the policy that chooses the single decision. This dynamic program is equivalent to computing the sum

$$F = \mathbb{E}\sum_{i=0}^{\infty}\gamma^i\hat{C}^i, \tag{16.60}$$

where $\hat{C}^i$ is a random variable giving the i^{th} contribution. If we let $\bar{c} = \mathbb{E}\hat{C}^i$, then clearly $F = \frac{1}{1-\gamma}\bar{c}$. But let's pretend that we do not know this, and we are using these various algorithms to compute the expectation.

We first used the single-state problem in section 16.4, but did not focus on the implications for stepsizes. Here, we use our ability to derive analytical solutions for the optimal value function for least squares temporal differences (LSTD), least squares policy evaluation (LSPE), and recursive least squares and temporal differences. These expressions allow us to understand the types of behaviors we would like to see in a stepsize formula.

In the remainder of this section, we start by assuming that the value function is approximated using a linear model

$$\overline{V}(s) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(s).$$

However, we are going to then transition to a problem with a single state, and a single basis function $\phi(s) = 1$. We assume that $\hat{v}$ is a sampled estimate of the value of being in the single state.

16.7.1 Least squares temporal differences

In section 16.3 we showed that the LSTD method, when using a linear architecture, applied to infinite horizon problems required solving

$$\sum_{i=1}^{n} \phi_f(S^i)(\phi_f(S^i) - \gamma \phi_f(S^{i+1}))^T \theta = \sum_{i=1}^{n} \phi_f(S^i)\hat{C}^i,$$

for each $f \in \mathcal{F}$. Let θ^n be the optimal solution. Again, since we have only one basis function $\phi(s) = 1$ for our single state problem, this reduces to finding $v^n = \theta^n$

$$v^n = \frac{1}{1-\gamma}\left(\frac{1}{n}\sum_{i=1}^{n}\hat{C}^n\right). \tag{16.61}$$

Equation (16.61) shows that we are trying to estimate $\mathbb{E}\hat{C}$ using a simple average. If we let $\bar{C}^n$ be the average over n observations, we can write this recursively using

$$\bar{C}^n = \left(1 - \frac{1}{n}\right)\bar{C}^{n-1} + \frac{1}{n}\hat{C}^n.$$

For the single state (and single decision) problem, the sequence $\hat{C}^n$ comes from a stationary sequence. In this case a simple average is the best possible estimator. In a dynamic programming setting with multiple states, and where we are trying to optimize over policies, v^n would depend on the state. Also, because the policy that determines the decision we take when we are in a state is changing over the iterations, the observations $\hat{C}^n$, even when we fix a state, would be nonstationary. In this setting, simple averaging is no longer the best. Instead, it is better to use

$$\bar{C}^n = (1 - \alpha_{n-1})\bar{C}^{n-1} + \alpha_{n-1}\hat{C}^n, \tag{16.62}$$

and use one of the stepsizes described in section 6.1, 6.2 or 6.3. As a general rule, these stepsize rules do not decline as quickly as $1/n$.

16.7.2 Least squares policy evaluation

Least squares policy evaluation, which is developed using basis functions for infinite horizon applications, finds the regression vector θ by solving

$$\theta^n = \arg\min_\theta \sum_{i=1}^{n} \left(\sum_f \theta_f \phi_f(S^i) - \left(\hat{C}^i + \gamma \overline{V}^{n-1}(S^{i+1}) \right) \right)^2 .$$

When we have one state, the value of being in the single state is given by $v^n = \theta^n$ which we can write as

$$v^n = \arg\min_\theta \sum_{i=1}^{n} \left(\theta - \left(\hat{C}^i + \gamma v^{n-1} \right) \right)^2 .$$

This problem can be solved in closed form, giving us

$$v^n = \left(\frac{1}{n} \sum_{i=1}^{n} \hat{C}^i \right) + \gamma v^{n-1}.$$

Similar to LSTD, LSPE works to estimate $\mathbb{E}\hat{C}$. For a problem with a single state and decision (and therefore only one policy), the best estimate of $\mathbb{E}\hat{C}$ is a simple average. However, as we already argued with LSTD, if we have multiple states and are searching for the best policy, the observation $\hat{C}$ for a particular state will come from a nonstationary series. For such problems, we should again adopt the updating formula in (16.62) and use one of the stepsize rules described section 6.1, 6.2 or 6.3.

16.7.3 Recursive least squares

Using our linear model, we start by using the following standard least squares model to fit our approximation

$$\min_\theta \sum_{i=1}^{n} \left(\hat{v}^i - \left(\sum_{f \in \mathcal{F}} \theta_f \phi_f(S^i) \right) \right)^2 .$$

As we have already discussed in chapter 3, we can fit the parameter vector θ using least squares, which can be computed recursively using

$$\theta^n = \theta^{n-1} - \frac{1}{1 + (x^n)^T B^{n-1} x^n} B^{n-1} x^n (\overline{V}^{n-1}(S^n) - \hat{v}^n)$$

where $x^n = (\phi_1(S^n), \ldots, \phi_f(S^n), \ldots, \phi_F(S^n))$, and the matrix B^n is computed using

$$B^n = B^{n-1} - \frac{1}{1 + (x^n)^T B^{n-1} x^n} \left(B^{n-1} x^n (x^n)^T B^{n-1} \right).$$

For the special case of a single state, we use the fact that we have only one basis function $\phi(s) = 1$ and one parameter $\theta^n = \overline{V}^n(s) = v^n$. In this case, the matrix B^n is a scalar, and the updating equation for θ^n (now v^n), becomes

$$\begin{aligned} v^n &= v^{n-1} - \frac{B^{n-1}}{1 + B^{n-1}} (v^{n-1} - \hat{v}^n) \\ &= \left(1 - \frac{B^{n-1}}{1 + B^{n-1}} \right) v^{n-1} + \frac{B^{n-1}}{1 + B^{n-1}} \hat{v}^n. \end{aligned}$$

If $B^0 = 1$, $B^{n-1} = 1/n$, giving us

$$v^n = \left(1 - \frac{1}{n}\right) v^{n-1} + \frac{1}{n}\hat{v}^n. \tag{16.63}$$

Now imagine we are using approximate value iteration. In this case, $\hat{v}^n = \hat{C}^n + \gamma v^n$. Substituting this into equation (16.63) gives us

$$\begin{aligned} v^n &= \left(1 - \frac{1}{n}\right) v^{n-1} + \frac{1}{n}(\hat{C}^n + \gamma \hat{v}^n) \\ &= \left(1 - \frac{1}{n}(1-\gamma)\right) v^{n-1} + \frac{1}{n}\hat{C}^n. \end{aligned} \tag{16.64}$$

Recursive least squares has the behavior of averaging the observations of $\hat{v}$. The problem is that $\hat{v}^n = \hat{C}^n + \gamma v^n$, since $\hat{v}^n$ is also trying to be a discounted accumulation of the costs. Assume that the contribution was deterministic, where $\hat{C} = c$. If we were doing classical approximate value iteration, we would write

$$v^n = c + \gamma v^{n-1}. \tag{16.65}$$

Comparing (16.64) and (16.65), we see that the one-period contribution carries a coefficient of $1/n$ in (16.64) and a coefficient of 1 in (16.64). We can view equation (16.64) as a steepest ascent update with a stepsize of $1/n$. If we change the stepsize to 1, we obtain (16.65).

16.7.4 Bounding $1/n$ convergence for approximate value iteration

It is well known that a $1/n$ stepsize will produce a provably convergent algorithm when used with approximate value iteration. Experimentalists know that the rate of convergence can be quite slow, but people new to the field can sometimes be found using this stepsize rule. In this section, we hope to present evidence that the $1/n$ stepsize should never be used with approximate value iteration or its variants.

Figure 16.4 is a plot of v^n computed using equation (16.64) as a function of $\log_{10}(n)$ for $\gamma = 0.7, 0.8, 0.9$, and 0.95, where we have set $\hat{C} = 1$. For $\gamma = 0.90$, we need 10^{10} iterations to get $\bar{v}^n = 9$, which means we are still 10 percent from the optimal. For $\gamma = 0.95$, we are not even close to converging after 100 billion iterations.

It is possible to derive compact bounds, $\nu^L(n)$ and $\nu^U(n)$ for $\bar{v}^n$ where

$$\nu^L(n) < v^n < \nu^U(n).$$

These are given by

$$\nu^L(n) = \frac{c}{1-\gamma}\left(1 - \left(\frac{1}{1+n}\right)^{1-\gamma}\right), \tag{16.66}$$

$$\nu^U(n) = \frac{c}{1-\gamma}\left(1 - \frac{1-\gamma}{\gamma n} - \frac{1}{\gamma n^{1-\gamma}}\left(\gamma^2 + \gamma - 1\right)\right). \tag{16.67}$$

Using the formula for the lower bound (which is fairly tight when n is large enough that v^n is close to v^*), we can derive the number of iterations to achieve a particular degree of

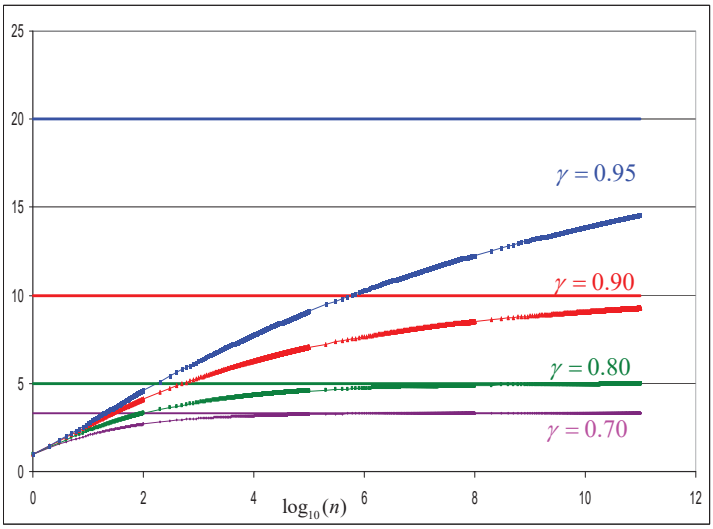

Figure 16.4 $\bar{v}^n$ plotted against $\log_{10}(n)$ when we use a $1/n$ stepsize rule for updating.

accuracy. Let $\hat{C} = 1$, which means that $v^* = 1/(1-\gamma)$. For a value of $v < 1/(1-\gamma)$, we would need at least $n(v)$ to achieve $\bar{v}^* = v$, where $n(v)$ is found (from (16.66)) to be

$$n(v) \geq [1 - (1-\gamma)v]^{-1/(1-\gamma)}. \tag{16.68}$$

If $\gamma = 0.9$, we would need $n(v) = 10^{20}$ iterations to reach a value of $v = 9.9$, which gives us a one percent error. On a 3-GHz chip, assuming we can perform one iteration per clock cycle (that is, 3×10^9 iterations per second), it would take 1,000 years to achieve this result.

16.7.5 Discussion

We can now see the challenge of choosing stepsizes for approximate value iteration, temporal-difference learning and Q-learning, compared to algorithms such as LSPE, LSTD and approximate policy iteration (the finite horizon version of LSPE). If we observe $\hat{C}$ with noise, and if the discount factor $\gamma = 0$ (which means we are not trying to accumulate contributions over time), then a stepsize of $1/n$ is ideal. We are just averaging contributions to find the average value. As the noise in $\hat{C}$ diminishes, and as γ increases, we would like a stepsize that approaches 1. In general, we have to strike a balance between accumulating contributions over time (which is more important as γ increases) and averaging the observations of contributions (for which a stepsize of $1/n$ is ideal).

By contrast, LSPE, LSTD and approximate policy iteration are all trying to estimate the average contribution per period for each state. The values $\hat{C}(s,x)$ are nonstationary because the policy that chooses the decision is changing, making the sequence $\hat{C}(s^n, x^n)$ nonstationary. But these algorithms are not trying to simultaneously accumulate contributions over time.

16.8 BIBLIOGRAPHIC NOTES

Section 16.1 - This section reviews a number of classical methods for estimating the value of a policy drawn from the reinforcement learning community. The best overall reference for this is Sutton & Barto (2018). Least-squares temporal differencing is due to Bradtke & Barto (1996).

Section 16.2 - Tsitsiklis (1994) and Jaakkola et al. (1994) were the first to make the connection between emerging algorithms in approximate dynamic programming (Q-learning, temporal difference learning) and the field of stochastic approximation theory (Robbins & Monro (1951), Blum (1954), Kushner & Yin (2003)).

Section 3.8 - L. & Soderstrom (1983) and Young (1984) provide nice treatments of recursive statistics. Precup et al. (2001) gives the first convergent algorithm for off-policy temporal-difference learning using basis functions by using an adjustment which based on the relative probabilities of choosing an action from the target and behavioral policies. Lagoudakis et al. (2002) and Bradtke & Barto (1996) present least squares methods in the context of reinforcement learning. Van Roy & Choi (2006) uses the Kalman filter to perform scaling for stochastic gradient updates, avoiding the scaling problems inherent in stochastic gradient updates such as equation (16.22). Nedic et al. (2003) describes the use of least squares equation with a linear (in the parameters) value function approximation using policy iteration and proves convergence for TD(λ) with general λ. Bertsekas et al. (2004) presents a scaled method for estimating linear value function approximations within a temporal differencing algorithm.

Section 16.3 - The development of Bellman's equation using linear models is based on Tsitsiklis & Van Roy (1997), Lagoudakis & Parr (2003) and Bertsekas (2017). Tsitsiklis & Van Roy (1997) highlights the central role of the D-norm used in this section, which also plays a central role in the design of a simulation-based version of the algorithm.

Section 16.4 - The analysis of dynamic programs with a single state is based on Ryzhov et al. (2015).

Section 16.5 - Baird (1995) provides a nice example showing that approximate value iteration may diverge when using a linear architecture, even when the linear model may fit the true value function perfectly. Tsitsiklis & Van Roy (1997) establishes the importance of using Bellman errors weighted by the probability of being in a state. de Farias & Van Roy (2000) shows that there does not necessarily exist a fixed point to the projected form of Bellman's equation $\Phi\theta = \Pi\mathcal{M}\Phi\theta$ where $\mathcal{M}$ is the max operator. This paper also shows that a fixed point does exist for a projection operator Π_D defined with respect to the norm $\|\cdot\|_D$ which weights a state s with the probability d_s of being in this state. This results is first shown for a fixed policy, and then for a class of randomized policies. GTD2 and TDC are due to Sutton et al. (2009), with material from Sutton et al. (2008).

Section 16.6 - Dearden et al. (1998*b*) introduces the idea of using Bayesian updating for Q-learning. Dearden et al. (1998*a*) then considers model-based Bayesian learning. Our presentation is based on Ryzhov & Powell (2010) which introduces the idea of correlated beliefs.

EXERCISES

Review questions

16.1 Describe in words (no mathematics) the difference between implementing TD(0) and TD(1).

16.2 Describe in words, with only necessary mathematics, the essential differences between LSTD and LSPE.

16.3 Show that updating the value of being in a state using, for example, temporal difference updates (section 16.1.3) are basically stochastic gradient updates (see section 16.2). This means that temporal difference updates are solving a particular optimization problem. What is the optimization problem?

Computational exercises

16.4 We are going to again try to use approximate dynamic programming to estimate a discounted sum of random variables:

$$F^T = \mathbb{E}\sum_{t=0}^{T}\gamma^t R_t,$$

where R_t is a random variable that is uniformly distributed between 0 and 100 (you can use this information to randomly generate outcomes, but otherwise you cannot use this information). This time we are going to use a discount factor of $\gamma = .95$. We assume that R_t is independent of prior history. We can think of this as a single state Markov decision process with no decisions.

(a) Using the fact that $\mathbb{E}R_t = 50$, give the exact value for F^{100}.

(b) Propose an approximate dynamic programming algorithm to estimate F^T. Give the value function updating equation, using a stepsize $\alpha_t = 1/t$.

(c) Perform 100 iterations of the approximate dynamic programming algorithm to produce an estimate of F^{100}. How does this compare to the true value?

(d) Compare the performance of the following stepsize rules: Kesten's rule, the stochastic gradient adaptive stepsize rule (use $\nu = .001$), $1/n^\beta$ with $\beta = .85$, the Kalman filter rule, and the optimal stepsize rule. For each one, find both the estimate of the sum and the variance of the estimate.

16.5 Figure 16.2 shows a five-state Markov chain where we transition from state 0 to 1 to 2 until transition out of state 5, earning contributions of 0 from each transition until we earn 1 when we transition from state 5, at which point we terminate. Table 16.1 shows the value of being in each state after each iteration of a TD(0) learning algorithm (otherwise known as approximate value iteration). Repeat the calculations in table 16.1 using a fixed stepsizes of

a) $\alpha = 1.0$.

b) $\alpha = 0.5$.

c) $\alpha = 0.1$.

d) $\alpha = 0.05$.

e) Compare the rates of convergence. Why wouldn't we always use $\alpha = 1.0$?

16.6 Consider a Markov decision process with a single state and single action. Assume that we do not know the expected value of the contribution $\hat{C}$, but each time it is sampled, draw a sample realization from the uniform distribution between 0 and 20. Also assume a discount factor of $\gamma = 0.90$. Let $V = \sum_{t=0}^{\infty} \gamma^t \hat{C}_t$. The exercises below can be formed in a spreadsheet. Estimate V using LSTD using 100 iterations.

16.7 Repeat exercise 16.6, estimating V with LSPE using 100 iterations.

16.8 Repeat exercise 16.6, estimating V using recursive least squares, executing the algorithm for 100 iterations.

16.9 Repeat exercise 16.6, estimating V using temporal differencing (approximate value iteration) and a stepsize of $1/n^{.7}$.

16.10 Repeat exercise 16.6, estimating V using temporal differencing (approximate value iteration) and a stepsize of $5/(5 + n - 1$.

16.11 Repeat the exercise above using a discount factor of 0.95.

Theory questions

16.12 We are going to walk through the derivation of the equations in section 16.5 assuming that there is a feature for each state, where $\phi_f(s) = 1$ if feature f corresponds to state s, and 0 otherwise. When asked for a sample of a vector or matrix, assume there are three states and three features. As above, let d_s^π be the probability of being in state s under policy π, and let D^π be the diagonal matrix consisting of the elements d_s^π.

a) What is the column vector ϕ if $s = 1$? What does $\phi\phi^T$ look like?

b) If d_s^π is the probability of being in state s under policy π, write out $\mathbb{E}[\phi\phi^T]$.

c) Write out the matrix Φ.

d) What is the projection matrix Π?

e) Write out equation (16.44) for $MSPBE(\theta)$.

16.13 Write out all the equations in section 16.5 for a problem where the state s is an integer quantity $\{0, 1, 2, \ldots, S\}$, and where

$$\overline{V}(s|\theta) = \theta_0 + \theta_1 s.$$

16.14 Write out all the equations in section 16.5 for a problem where the there is a feature $\phi_f(s)$ for each state, where $\phi_f(s) = 1$ if $f = s$.

Diary problem

The diary problem is a single problem you chose (see chapter 1 for guidelines). Answer the following for your diary problem.

16.15 Using the policy that you designed in exercise 12.13, sketch the steps for estimating the value of the policy using the following methods:

a) TD(0) - Temporal differencing with $\lambda = 0$.

b) TD(1) - Temporal differencing with $\lambda = 1$.

c) For your diary problem, discuss what appear to be the strengths and weaknesses of TD(0) and TD(1).

Bibliography

Baird, L. C. (1995), 'Residual algorithms: Reinforcement learning with function approximation', *In Proceedings of the Twelfth International Conference on Machine Learning* **pp**, 30–37.

Bertsekas, D., Borkar, V. S. & Nedic, A. (2004), Improved Temporal Difference Methods with Linear Function Approximation, *in* J. Si, A. G. Barto, W. B. Powell & D. Wunsch, eds, 'Handbook of Learning and Approximate Dynamic Programming', IEEE Press, New York, pp. 233–257.

Bertsekas, D. P. (2017), *Dynamic Programming and Optimal Control: Approximate Dynamic Programming*, 4 edn, Athena Scientific, Belmont, MA.

Blum, J. (1954), 'Multidimensional stochastic approximation methods'', *Annals of Mathematical Statistics* **25**, 737–74462.

Bradtke, S. J. & Barto, A. G. (1996), 'Linear least-squares algorithms for temporal difference learning', *Machine Learning* **22**(1), 33–57.

de Farias, D. P. & Van Roy, B. (2000), 'On the existence of fixed points for approximate value iteration and temporal-difference learning', *Journal of Optimization Theory and Applications* **105**(3), 589–608.

Dearden, R., Friedman, N. & Andre, D. (1998*a*), 'Model based Bayesian Exploration', *Learning*.

Dearden, R., Friedman, N. & Russell, S. (1998*b*), 'Bayesian Q-learning', *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE* pp. 761–768.

Jaakkola, T., Jordan, M. I. & Singh, S. P. (1994), 'On the convergence of stochastic iterative dynamic programming algorithms', *Neural Computation* **6**(6), 1185–1201.

Kushner, H. J. & Yin, G. G. (2003), *Stochastic Approximation and Recursive Algorithms and Applications*, Springer, New York.

L., l. & Soderstrom, T. (1983), *Theory and Practice of Recursive Identification*, MIT Press, Cambridge, MA.

Lagoudakis, M. & Parr, R. (2003), 'Least-squares policy iteration', *Journal of Machine Learning Research* **4**, 1107–1149.

Lagoudakis, M., Parr, R. & Littman, M. (2002), 'Least-squares methods in reinforcement learning for control', *Methods and Applications of Artificial Intelligence* pp. 752–752.

Nedic, A., Bertsekas, D. P., Science, C., Nedić, A., Nediç, A. & Nedi, A. (2003), 'Least squares policy evaluation algorithms with linear function approximation', *Discrete Event Dynamic Systems* **13**(1), 79–110.

Precup, D., Sutton, R. S. & Dasgupta, S. (2001), Off-policy temporal-difference learning with function approximation, *in* '19th International Conference on Machine Learning', pp. 417–424.

Robbins, H. & Monro, S. (1951), 'A stochastic approximation method', *The Annals of Mathematical Statistics* **22**(3), 400–407.

Ryzhov, I. O. & Powell, W. B. (2010), Approximate Dynamic Programming With Correlated Bayesian Beliefs, *in* 'Forty-Eighth Annual Allerton Conference on Communication, Control, and Computing', Monticello, IL.

Ryzhov, I. O., Frazier, P. I. & Powell, W. B. (2015), 'A new optimal stepsize for approximate dynamic programming', *IEEE Transactions on Automatic Control* **60**(3), 743–758.

Sutton, R. S. & Barto, A. G. (2018), *Reinforcement Learning: An Introduction*, 2nd edn, MIT Press, Cambridge, MA.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C. & Wiewiora, E. (2009), 'Fast gradient-descent methods for temporal-difference learning with linear function approximation', *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09* pp. 1–8.

Sutton, R. S., Szepesv, C. & Maei, H. R. (2008), A Convergent O (n) Algorithm for Off-policy Temporal-difference Learning with Linear Function Approximation, *in* 'Proceedings of the Neuro Information Processing Society', Vancouver, pp. 1–8.

Tsitsiklis, J. N. (1994), 'Asynchronous Stochastic Approximation and Q-Learning', *Machine Learning* **16**, 185–202.

Tsitsiklis, J. N. & Van Roy, B. (1997), 'An analysis of temporal-difference learning with function approximation', *IEEE Transactions on Automatic Control* **42**(5), 674–690.

Van Roy, B. & Choi, D. P. (2006), 'A Generalized Kalman Filter for Fixed Point Approximation and Efficient Temporal-Difference Learning', *Discrete Event Dynamic Systems* **16**, 207–239.

Young, P. (1984), *Recursive Estimation and Time-Series Analysis*, Springer-Verlag, Berlin, Heidelberg.