Chapter 10

Simulation in Tabular Models

You've got to win a little, lose a little, yes, and always have the blues a little[1].

From the song *The Glory of Love*, by Billy Hill, 1899-1940.

This is the first of two chapters on reinforcement learning, focusing on how trial-and-error learning is implemented in simulated and real-world environments. As the song lyric suggests, learning is an ongoing interplay of winning and losing. Successes reinforce behavior while failures diminish it. This dynamic lies at the heart of reinforcement learning and underpins the temporal difference and Q-learning methods introduced in this chapter.

This chapter describes simulation-based methods for evaluating value and state-action value functions, a crucial step in identifying *effective* policies. The focus is on models that can be represented in *tabular* form, that is, when the sets of states and actions are sufficiently small so that value functions, state-action value functions and stationary policies can be represented in look-up tables indexed by either states or state-action pairs. Chapter 11 builds on this foundation by extending methods to complex environments that instead rely on approximations to represent value functions, state-action value functions, and policies.

A brief comment on terminology:

1. The term *methods* replaces algorithms[2] because the proposed approaches do not

[1] Hill (1936).

[2] Note that in the text, they are still indexed as algorithms.

have stopping criteria that guarantee precise error bounds. The number of iterations or replicates is often chosen arbitrarily.

2. The expressions *good* or *effective* distinguish policies identified by the methods herein from optimal or ϵ-optimal policies generated by the algorithms in Chapters 5, 6 and 7.

3. The term *simulation* refers broadly to any mechanism, be it computer-based or a physical system operating in the real world, that as shown in Figure 10.1, generates successor states and rewards from current states and actions based on an underlying known or unknown MDP.

Figure 10.1: Symbolic representation of the data generation process underlying the methods in this chapter. The simulator, be it computer-based or a real-time process, generates the subsequent state-reward pair (s', r) from a current state-action pair (s, a).

This chapter will consider three broad classes of methods:

1. Monte Carlo

2. Temporal differencing

3. Q-learning

Each has numerous variants that will be explored below. Policy gradient methods, which are compatible with simulation, will be discussed in the next chapter.

Model classification

The methods herein are frequently classified as either *model-based* or *model-free*. The expression "model-free" is somewhat a misnomer; it refers to settings where system dynamics are governed by a Markov decision process, but where the Markov decision process is not used to estimate value functions, state-action value functions or policies. Instead the system learns by interacting directly with the environment; this feedback guides policy choice. Model-free approaches may be used when the underlying model is very complex and challenging to formulate, when the state space and/or reward function is unknown, or when complex dynamics makes it easier to use a simulator.

More explicitly:

Model-based: Analysis is based on a Markov decision process model of the system. This means that S, $\{A_s|s \in S\}$, $r(s,a,j)$ and $p(j|s,a)$ are known and can be used to generate data by sampling s' from $p(\cdot|s,a)$ and setting $r' = r(s,a,s')$. Moreover, they can be used directly in computation. Examples include inventory management, queuing control, and revenue management models.

Model-free: There is an underlying model but it is not used for analysis because it may be complex or not known. *At a minimum the analyst knows the actions available at each decision epoch.* The set of states may or may not be known. State transitions and rewards are generated by a simulator or a real-world process as depicted in Figure 10.1. Methods must be based on value functions, state-action value functions or policies only. Applications include gaming, autonomous vehicle control, and robotics. This approach is often applied to episodic models in which there is a task to complete. Moreover, there may not be an explicit reward structure; the analyst must design one to achieve the objectives as efficiently as possible.

The distinction between model-based and model-free reinforcement learning is conceptually useful but often blurred in practice. Model-based methods rely on explicit knowledge or estimation of the underlying MDP — namely, the transition probabilities and reward function — and use this model to evaluate or optimize policies. Model-free methods, in contrast, learn value functions or policies directly from experience, without requiring knowledge of the MDP's structure.

For example, in applications such as advance appointment scheduling (Section 3.7) a Markov decision process model can be explicitly formulated, but it may be preferable to adopt a model-free approach[3].

This chapter describes and illustrates policy evaluation and optimization methods, focusing on episodic and infinite horizon discounted models, with a brief discussion of average reward models. Its intent is not to be comprehensive and theoretically rigorous, but to provide a high-level framework for delving into Chapter 11 and the voluminous and rapidly advancing reinforcement learning literature.

The key concept underlying the methods in this chapter and the next is that:

> Simulation replaces computation of expectations in Bellman equations.

10.1 Preliminaries

This section discusses data generation and underlying trade-offs when choosing methods.

[3]The idea here is that instead of deriving a (complicated) analytical model for the MDP, it might be easier to simulate events, apply a policy, and observe the outcome.

10.1.1 Data generation

The two primary sources of data are computer simulation and real-world interaction. They can be used for policy evaluation or optimization. Each creates data sequences by transforming states and actions to successor states and rewards as discussed above:

> **Simulation:** A computer-based simulator generates successor states and rewards using either underlying probabilistic mechanisms or MDP model components.
>
> **Real-world interaction:** A system, process or experiment generates successor states and rewards.

Repeated application of this mechanism generates *trajectories*. A few examples may help clarify this distinction:

1. In a queuing service rate control model, sampling the number of arrivals and service completions in a period transforms a state and action to a successor state and reward. This is a classic example of an application of simulation. It is preferable to simulating directly from the MDP model, which would require first computing a convolution of the service and arrival distributions. Clearly, this latter approach would not be scalable to larger models.

2. A physical navigational robot interacts with its environment by choosing an action in a given state and (possibly) moving to a new location and generating a reward. This could be costly if some actions could damage the robot. Alternatively a simulation of robot motion could avoid such costs. However if the robot is complex, policies generated through simulation might not perform identically in real life.

3. In Chess, White's choice of a move results in a new board configuration prior to White's next move. Randomness arises through the implementation of Black's strategy. This can be implemented in actual play or more likely through a simulator. Self-play is often used to develop computer-based Chess players.

4. In the pre-board screening application described in the Section 8.6.5, one approach is to develop an MDP model and derive and then simulate system behavior using the derived transition probabilities. Clearly, this is an onerous task. Instead, one could develop a simulation of the system where arrival and service probability distributions combine with operational policies to update the system state and determine one-period costs. Real-world interaction would be impractical since it would impact passenger wait times.

5. In managing patient flow in a health care system, historical data on lengths of stay and transfer decisions could provide appropriate data for analyses.

Our computational experiments suggest that distinguishing computer-based simulations from real-world interaction was important to bring to the forefront for the following reason. In a simulation environment, the analyst may restart the system at any state at any time, while in real-world interaction, states usually follow system dynamics. As a case in point, consider the pre-board screening application referred to above. Clearly the system state cannot be altered in real time. It is determined by passenger arrivals and inspection completions. Computer simulation allows testing policies in a wide range of configurations that might not have been observed or that occur infrequently.

Data generation mechanisms

Starting at $s^1 \in S$, a simulator or process will generate a sequence

$$(s^1, a^1, r^1, s^2, a^2, r^2, \ldots, s^N, r^N),$$

where N may be random or fixed. In the model-based environment, $r^n = r(s^n, a^n, s^{n+1})$, so it can be computed from a sequence of state-action pairs. In the model-free environment, the reward will be observed directly.

Data in reinforcement learning applications may be generated in a variety of ways, depending on the setting, objectives, and available resources. The following four categories capture common approaches, each with different implications for learning, evaluation, and exploration. Their significance will become clearer in the examples that follow.

Trajectory-based: Specify a stationary randomized or deterministic policy π. Starting from an initial state $s^1 \in S$, generate a trajectory $(s^1, a^1, r^1, s^2, a^2, r^2, \ldots, s^N, r^N)$ from a simulator or a real-world process. The initial state may be fixed or chosen randomly from a pre-specified distribution.

State-based: Specify a stationary randomized or deterministic decision rule $d(s)$. Sample $s \in S$ from a specified distribution. Apply $d(s)$ to generate[4] a successor state s' and a reward r. In state-based data generation, s is not the result of a transition from the previous state; it is sampled from a pre-specified probability distribution. This is sometimes referred to as *random restart* data generation.

Action-based: Given $s \in S$, generate $a \in A_s$ by some mechanism that does not correspond[5] to a pre-specified policy or decision rule. Then generate s' and r.

[4]When d is randomized, this action is generated from $w_d(a|s)$. When d is deterministic, $d(s)$ denotes an action.

[5]Logically, any way of generating actions from states may be viewed as a policy but it need not be explicitly stated as such or used in the system.

State-action pair-based: Sample $s \in S$ and $a \in A_s$ from specified distributions, and then generate r and s'. State-action based generation combines exploration in *both* the state and action spaces.

Comments regarding the use, benefits, and limitations of these data generation approaches follow:

1. Data may be generated either *sequentially* or by *sampling states or state-action pairs after each iteration*. Sequential data may be realized in real-time as the output of a real or simulated process. In contrast, in most applications, simulation must be used to generate random restart data because random restarts would not be able to be physically realized.

2. Trajectory-based and state-based approaches provide the basis for policy evaluation.

3. Trajectory-based methods generate data that behaves like the stationary distribution of a Markov chain generated by a policy.

4. State-based methods, in which states are generated either randomly or deterministically, enable investigation of system behavior in states that are infrequently (or never) visited under trajectory-based data generation.

5. Action-based methods are particularly useful in optimization because they facilitate exploration in the action space (and state space) enabling identification of potentially beneficial actions.

6. State-action pair-based methods combine the features of state-based and action-based methods. They combine exploration in the action space with the ability to generate states that are infrequently observed under policies generated by optimization methods.

Common random numbers

Simulation-based comparisons of policies or methods can be enhanced using *common random numbers*. The idea is to use the same random number stream to evaluate multiple policies or algorithmic parameter choices. The benefit is that by eliminating one source of variability, namely the random number sequence seed, the variability of comparisons is reduced[6].

[6]Suppose X and Y are two random variables. Then

$$\mathrm{Var}(X - Y) = \mathrm{Var}(X) + \mathrm{Var}(Y) - 2\mathrm{Cov}(X, Y).$$

When X and Y are positively correlated, that is when $\mathrm{Cov}(X, Y) > 0$, the variance of the difference of X and Y is reduced. Common random numbers seek to achieve this by removing one source of variability, the random number sequence seed.

In the context of this chapter and the next, common random numbers potentially reduce the variance of the *differences* of estimates of value function and state-action value functions across policies or methods. In this chapter, they will be primarily used for comparing the performance of algorithms under different parameter choices or implementation methods.

Examples of applications of common random numbers include using the same stream of arrivals and service times in queuing control models or the same demand stream in inventory models such as the newsvendor model.

10.1.2 Trade-offs

This section describes some terminology and trade-offs that may be encountered when using the methods described in subsequent sections.

1. **Offline vs. online:** *Online* methods update value functions or state-action value functions after each transition. *Offline* (*batch*) methods use a complete realization of a trajectory for computation. In practice one may combine these two approaches by first using an offline implementation to obtain good starting values for subsequent application of an online method in real-world interaction.

2. **Exploration vs. exploitation:** In light of one of the most famous quotes commonly mis-attributed[7] to Albert Einstein (1879-1955):

   > *Insanity is doing the same thing over and over again and expecting different results,*

   it is advantageous to try new things occasionally. Thus, to find good policies, a method needs to deviate from greedy action selection so as to investigate the impact of other actions on system performance. Greedy action selection is referred to as *exploitation* and non-greedy action selection as *exploration*. Exploration methods include ϵ-greedy and softmax sampling, which are described below. Exploitation may be attractive when the decision maker wishes to accrue rewards during the learning phase or improve estimation of value functions and state-action value functions.

3. **On-policy vs. off-policy:** A method is said to be *on-policy* if it evaluates the data-generating policy, often referred to as the *learning* or *behavioral* policy. A method is said to be *off-policy* if it evaluates a policy that differs from the behavioral policy. This distinction is relevant when designing algorithms and when the reward received during learning is of concern to the investigator. Off-policy methods should be considered when reusing previously collected data or when safety or cost constraints limit exploration under the behavioral policy.

[7] The source of the quote appears to be the book *Sudden Death* by Rita Mae Brown, Bantam, 1983, p. 68, as noted in *The Ultimate Quotable Einstein*, by Alice Calaprice, Princeton University Press, 2011, p. 474.

4. **Variance vs. bias:** In statistical estimation one often seeks a minimum-variance unbiased estimator. When evaluating a method, the analyst may be willing to accept some bias in exchange for reduced variance. The *root mean squared error* for distinguished states or averaged over states captures this trade-off. (See Appendix 10.11.1 for a detailed discussion of this issue.)

5. **Synchronous vs. asynchronous:** A *synchronous* method simultaneously updates value functions in **all** states or state-action value functions in **all** state-action pairs in a similar way to value and policy iteration. An *asynchronous* method updates these quantities one state or state-action pair at a time in a similar way to Gauss-Seidel iteration. With the exception of TD(γ), all the methods in this chapter are asynchronous. The reason for this distinction is that in asynchronous methods, the precision of estimates varies over states or state-action pairs.

Table 10.1 distinguishes the methods discussed in this chapter with respect to their input data, use, and updating approach.

Method	Data	Use	Updating
Monte Carlo	Trajectory-based	Policy Evaluation	Offline
Temporal differencing	Trajectory-based or state-based	Policy Evaluation	Online
Q-learning	Trajectory-based or state-action pair-based	Optimization	Online

Table 10.1: Methods, compatible data generation mechanisms, and purpose.

10.2 Methods overview: The learning newsvendor

This section provides an overview of the methods introduced in this chapter by demonstrating how they can be used to determine the optimal order quantity in the classic newsvendor model, which was introduced in Section 3.1.2. The newsvendor model occupies a central role in the operations research literature due to its analytical tractability and real-world relevance. It applies to a wide-range of products and services with uncertain demand and a limited selling season including bread and dairy products, daily (printed) newspapers, fashion items and time-sensitive services such as hotel rooms and airplane seats.

The reason for focusing on the newsvendor problem is that it provides a transparent yet rich framework, namely a single-state, one-period model, in which to highlight core concepts without the complicating features of multi-period dynamics and multiple states. Thus, it provides an ideal setting to introduce Monte Carlo methods, temporal differencing, and Q-learning.

As noted in Section 3.1.2, the newsvendor's objective is to choose an order quantity $a^* \in A_0$ that maximizes the expected revenue

$$E[r(a, Z)] = \sum_{z=0}^{M} r(a, z) f(z), \tag{10.1}$$

where A_0 denotes the set of possible order quantities and M denotes a finite upper bound on the demand[8]. Recall that the random variable Z denotes the demand, $f(z) := P[Z = z]$, and $r(a, z)$, the revenue if a units of the item are ordered and the demand is z, was shown to be equal to

$$r(a, z) := G \min(z, a) - L(a - z)^+. \tag{10.2}$$

In this expression G denotes the profit (selling price minus cost) obtained by selling a unit and L the loss (cost minus salvage value) associated with disposing of an unsold item.

Moreover, it was shown that if the demand distribution is known and continuous, the optimal order quantity is available in closed form as

$$a^* = F^{-1}\left(\frac{G}{G + L}\right), \tag{10.3}$$

where $F(z) := P[Z \leq z]$.

10.2.1 Analysis by simulation

This may be regarded as either an episodic model with all episodes of length one, or equivalently, as a one-period model as formulated in Chapter 3. The following rather formal notation will help relate it to later material in this chapter and as well to general Markov decision process notation. Let[9] $q(a), a \in A_0$, denote the expected reward when choosing order quantity (action) a in state 0, that is,

$$q(a) := E[r(a, Z)].$$

Since in this simple model decision rules correspond to actions, the expected total reward of using decision rule $d'(0) = a'$ is given by $v^{d'}(0) = q(a')$. Note the equivalence between the value function and state-action value function here because in general:

> A value function equals a state-action value function evaluated at a specific decision rule.

Offline (batch) and online approaches for obtaining estimates $\hat{q}(a)$ of $q(a)$ are described below.

[8] In contrast to Chapter 3, the demand is truncated to facilitate demand simulation and analyses.

[9] In the standard state-action value function notation, this quantity would be written as $q(0, a)$. The expression $q(a)$ is used for simplicity, since there is only one state.

Monte Carlo estimation

Monte Carlo estimation is an offline (batch) approach. A policy corresponds to a single fixed order quantity $\bar{a}$. To evaluate it using a Monte Carlo approach, take N samples $\{z^1, \ldots, z^N\}$ from $f(\cdot)$, set

$$r^n := r(\bar{a}, z^n) \tag{10.4}$$

and

$$\hat{q}(\bar{a}) = \frac{1}{N} \sum_{n=1}^{N} r^n. \tag{10.5}$$

To optimize using Monte Carlo, repeat the above simulation for each $a \in A_0$ and set

$$a^* \in \arg\max_{a \in A_0} \hat{q}(a). \tag{10.6}$$

Note that

1. the same demand sequence could be used for evaluating all actions,
2. the ordering of r^n does not affect the estimate, and
3. estimates of the standard deviation and other distributional properties of $q(a)$ can be easily obtained.

Temporal differencing

Temporal differencing provides an online approach for policy value function estimation[10]. Instead of waiting until all N values of r^n have been realized, estimates of $q(\bar{a})$ are updated as data arrives. It works as follows. To evaluate the policy $\bar{a}$, set $q^1(\bar{a})$ to an arbitrary value[11] and after realizing z^n, compute r^n using (10.4) and update the estimate of $q(a)$ according to

$$q^{n+1}(\bar{a}) = q^n(\bar{a}) + \tau_n\big(r^n - q^n(\bar{a})\big), \tag{10.7}$$

where $0 < \tau_n \leq 1$ for $n = 1, 2, \ldots$ is a sequence of positive constants converging to 0. The expression (10.7) can be thought of as follows. The quantity $q^n(\bar{a})$ is the "prediction" of $q(\bar{a})$ before observing r^n. After observing r^n a "new prediction" is obtained by adding a *fraction* of the difference between the observation and the previous prediction to the previous prediction. Referring to τ_n as the *learning rate*, the above recursion can be expressed as

> New prediction = old prediction + learning rate × (observation − old prediction).

[10] This is referred to as a *prediction problem* in the reinforcement learning literature.
[11] 0 is a good choice.

Choosing a learning rate less than 1 avoids over-correction and smooths out noise. Requiring that it approaches 0 as n increases stabilizes the estimate. Moreover, choosing the learning rate to satisfy some technical conditions described in this chapter and Appendix 10.11.2 provides theoretical guarantees of convergence.

Some comments about temporal differencing follow:

1. This approach is referred to as TD(0) for reasons to be discussed below.
2. Choosing $\tau_n = 1/n$ is equivalent to online estimation of the sample mean[12].
3. Choosing the learning rate in practice requires experimentation.
4. The order of the r^n does impact the value of the estimate for non-constant learning rate sequences.
5. Distributional properties of $q^n(\bar{a})$ can be obtained by choosing multiple permutations or bootstrap[13] samples from $(r^1, \ldots, r^n)$.

Q-learning

Instead of computing an estimate of $q(a)$ for each action and then maximizing, one could instead incorporate the maximization directly in the recursion. This is the idea underlying the *Q-learning* recursion:

$$q^{n+1}(a) = q^n(a) + \tau_n \left(\max_{a \in A_0} r(a, z^n) - q^n(a) \right). \tag{10.8}$$

When A_0 consists of a single action, Q-learning is equivalent to temporal differencing.

Because of noise (sampling variability) alternatives to greedy action choice may be preferable. Recursions based on this approach can be expressed as

$$q^{n+1}(a) = q^n(a) + \tau_n \big(r(a', z^n) - q^n(a) \big), \tag{10.9}$$

where a' is chosen randomly in some way[14].

One approach is to use a version of ϵ-*greedy* action selection that samples a' according to:

$$a' = \begin{cases} a^* \in \arg\max_{a \in A_0} r(a, z^n) & \text{with probability } 1 - \epsilon_n \\ a \in A_0 \setminus \{a^*\} & \text{with probability } \epsilon_n / (|A_0| - 1) \end{cases} \tag{10.10}$$

[12] See Example 10.14 in the chapter appendix

[13] Bootstrapping here refers to the statistical procedure that resamples from a data set with replacement to obtain properties of estimators.

[14] In a one-period model such as this, this algorithm is identical to an algorithm referred to as SARSA below.

for some sequence of $\epsilon_n, n = 1, 2 \ldots$. Note that decreasing ϵ_n leads to less exploration later on in the iterative process. Moreover, setting $\epsilon_n = 1$ for all n corresponds to randomly sampling actions and $\epsilon_n = 0$ for all n to using greedy action selection.

An alternative is to use *softmax* sampling (action selection) in which action a' is chosen with probability

$$p(a') := \frac{e^{\eta_n q(a')}}{\sum_{a \in A_0} e^{\eta_n q(a)}}, \tag{10.11}$$

where the sequence $\eta_n, n = 1, 2, \ldots$, affects the relative weighting of actions. Large values of η_n put more weight on actions with the largest values of $q(a')$, while small values of η_n make the weight on actions more uniform. Note that when all values of $q(a)$ are equal, softmax sampling corresponds to random sampling.

The ϵ-greedy method selects actions other than the greedy action with equal probability, while the softmax approach samples among actions with greater rewards. When there a few actions in each state, either method would be appropriate. However, when there are many actions, the softmax sampling may be more appropriate because it concentrates exploration on fewer actions, namely those with greater q-values. This distinction is significant in the calculations below.

Q-learning: An "Earning while learning" perspective

Instead of simply describing a way to find good actions, recursions (10.8) and (10.9) may be viewed from the perspective of an agent that learns by trial-and-error and accumulates reward beginning as follows.

1. Choose $q^1(a)$ and set $R^0 = 0$.
2. Choose a^1 greedily, ϵ-greedily or by softmax sampling.
3. Implement a^1 and observe r^1.
4. Set $R^1 = R^0 + r^1$.
5. Update $q^2(a^1)$ using (10.8) or (10.9).

The fourth step highlights the subtle distinction with the Q-learning approach described previously. Here, the perspective is that rewards earned are tracked while learning takes place.

By repeating this procedure[15] for N steps, the newsvendor obtains total reward R^N, and uses $q^N(a)$ to choose order quantities. Variants of this approach[16] can be compared on the basis of a running average of the cumulative reward and on how accurately they determine the known optimal order quantity.

[15] We choose not to state this formally here.

[16] Variants may correspond to different choices of tuning parameters and update methods.

10.2.2 A numerical study

This section illustrates and discusses computation using the above approaches.

Model

Demand is generated from either a discrete (rounded) and truncated normal distribution with mean 50 and standard deviation 15 or a gamma distribution[17] with shape parameter 20 and scale parameter 4. The item cost is 20, the salvage value is 5, and price varies among 21, 30, and 100. The corresponding G values are 1, 10, and 80, and $L = 15$. The corresponding ratios of $G/(G+L)$ are $0.0625, 0.400$, and 0.842, illustrating several distinct quantiles of the demand distribution. Set $A_0 = \{0, 1, \ldots, 120\}$.

Monte Carlo estimation

Monte Carlo methods are compared using sample sizes of $N = 100$ and $N = 5{,}000$. A replicate consists of estimating $q(a)$ for each $a \in A_0$ for each price. The expected reward is estimated using (10.5) and the optimal order quantity is chosen according to (10.6). Using a common random number seed, the same realized demand was used in each replicate. There was no need to resample; estimates could be based on stored demand samples.

Table 10.2 gives the optimal order quantity (for a typical replicate) using the closed form representation (10.3) and that based on the arg max of the Monte Carlo estimates of $q(a)$. Of course, choosing $N = 5{,}000$ produces more accurate estimates of the optimal order quantity than using $N = 100$ but the difference was surprisingly small. An exception was when demand was normal and price was 21, when the optimal order quantity corresponded to $G/(G + L) = 0.0625$. This is because larger sample sizes are needed to estimate values in the tail of a distribution.

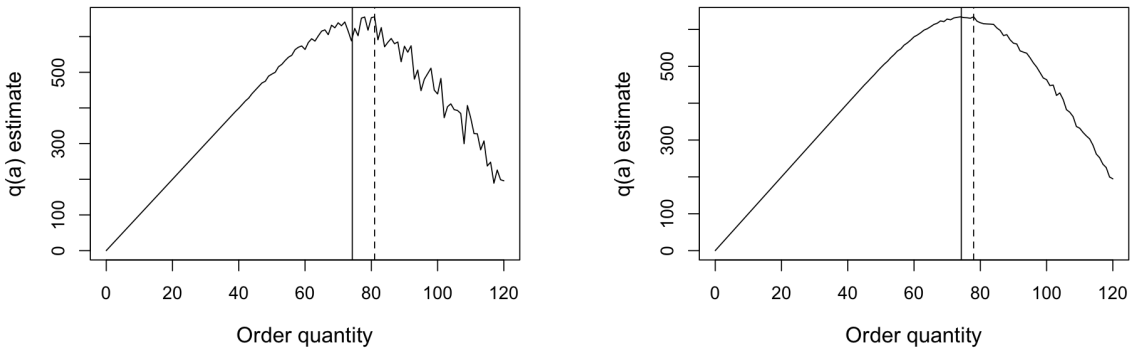
Distribution	Price	Optimal	Monte Carlo ($N = 100$)	Monte Carlo ($N = 5{,}000$)
Normal	21	27	36	26
	30	46	42	44
	100	65	69	64
Gamma	21	55	57	55
	30	74	81	78
	100	98	109	104

Table 10.2: Optimal order quantity and that based on Monte Carlo estimates of $q(a)$ in a typical replicate of the newsvendor model using $N = 100$ and $N = 5{,}000$.

Figure 10.2 shows $\hat{q}(a)$ based on a single replicate with gamma distributed demand, price equal to 30 and two values of N. Observe that the estimates with $N = 100$ are

[17] For this choice of parameters, the mean of the gamma distribution is 80 and its standard deviation is 17.89.

considerably noisier than those with $N = 5{,}000$. However, surprisingly, the estimated optimal order quantities vary little. Note that for both values of N the curves are quite similar for small values of a. Moreover, Figure 10.2b shows that $\hat{q}(a)$ is quite flat near the true optimum, suggesting that there may be considerable variability in choosing the action using Monte Carlo estimates.



(a) $N = 100$ (b) $N = 5{,}000$

Figure 10.2: Estimates of $q(a)$ based on a single replicate with gamma distributed demand and price equal to 30 for two sample sizes. The solid vertical line represents the theoretical optimal order quantity and the dashed line that obtained using (10.6).

Temporal differencing

This section illustrates the use of temporal differencing by applying it to estimate $q(80)$ for $G = 80$, $L = 15$ and gamma distributed demand. It compares learning rates $\tau_n = 0.05, n^{-1}, 0.5n^{-0.75}$, and $10/(40 + n)$. Figure 10.3 shows the evolution of the iterates of (10.7) over a typical replicate with $N = 1{,}000$ iterations starting from 0 and the same demand sequence for each learning rate. Observe that the sample mean, which corresponds to $\tau_n = n^{-1}$, best approximates the true value, with the estimate using $\tau_n = 0.5n^{-0.75}$ quite close. Note also that the estimate corresponding to $\tau_n = 0.05$ oscillates. More formally, estimates can be compared using the RMSE as described in Appendix 10.11.1. The respective RMSE values corresponding to the above learning rates are 467.21, 163.87, 214.29 and 419.08, consistent with Figure 10.3. Such calculations over multiple configurations and with many replicates provide the basis for parameter studies in this chapter and the next.

Q-learning

Applying Q-learning using (10.9) was more challenging than the methods above. The ϵ-greedy action selection method was unreliable. Even when ϵ_n was slightly less than one,

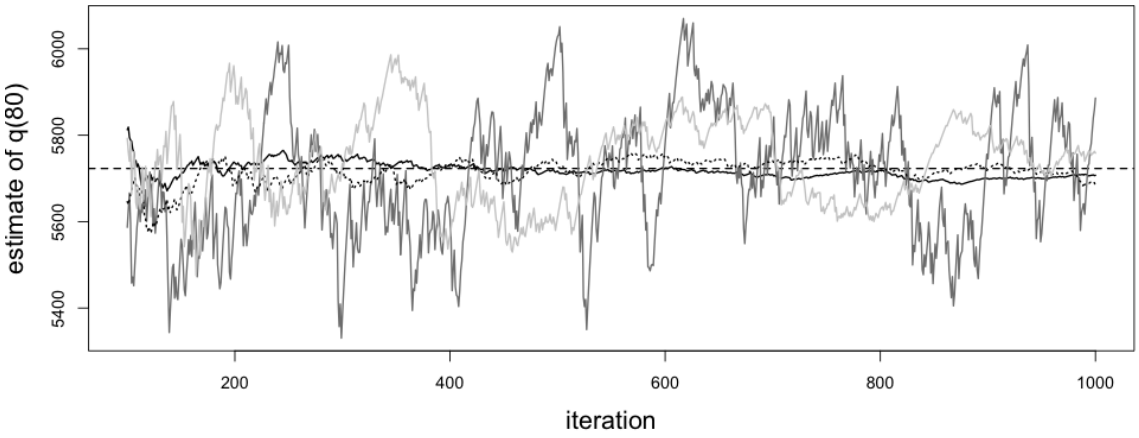


Figure 10.3: Comparison, based on a single replicate, for estimates of $q(80)$ for the four choices of τ_n described in the text. The horizontal dashed line represents the exact value $q(80) = 5{,}723$, and the other four lines represent different choices of τ_n. The dark grey line corresponds to $\tau_n = 0.05$, the solid line to $\tau_n = n^{-1}$, the dotted line to $\tau_n = 0.5/(n^{0.75})$ and the light grey line to $\tau_n = 10/(n + 40)$.

a single high-value action was sampled too frequently; all other actions were sampled equally and infrequently. Thus the recursion "got stuck" at a potentially non-optimal action.

Softmax action selection was better suited to this problem since there are 121 actions and $q(a)$ was quite flat near the optimum and dropped off quickly as shown in Figure 10.2b. Consequently, softmax sampled high-value actions more frequently, thus providing more accurate estimates of $q(a)$ close to the optimum.

After preliminary experimentation, three learning rates, $\tau_n = 5/(20 + n)$, $\tau_n = 0.5n^{-0.5}$ and $\tau_n = 0.05/(1 + (10^{-5})n^2)$, were chosen for further analyses. The third choice for τ_n is justified later in this chapter. Note that estimates based on $\tau_n = n^{-1}$ and $\tau_n = 0.5n^{-0.75}$ converged too quickly, producing very inaccurate estimates of $q(a)$ and the optimal order quantity.

This study used the scenarios in the model description above and generated 40 replicates for each using common random number seeds for each replicate. For each scenario, the recursions began with $q(a) = 0$, set $N = 50{,}000$ and the softmax parameter $\eta_n = 1/6{,}000$ for all n. The number of times an action was previously chosen (as opposed to the iteration number) was used as the index of the learning rate.

Table 10.3 summarizes results in terms of the mean and standard deviation of the optimal order quantify over 40 replicates for each configuration of distribution, price and learning rate. The learning rate $\tau_n = 0.5n^{-0.5}$ best estimated the optimal order quantity on average. However, the estimates were more variable when the price equaled 100. Note that when $N = 10{,}000$, $\tau_n = 0.5n^{-0.5}$ produced the most accurate estimates

but their variability was greater.

Distribution	Price	Optimal	$\tau_n = 5/(20+n)$	$\tau_n = 0.05/\left(1+(10^{-5})n^2\right)$	$\tau_n = 0.5n^{-0.5}$
Normal	21	27	29.79 (2.89)	30.72 (3.97)	28.59 (2.73)
	30	46	48.64 (3.09)	48.41 (4.00)	46.62 (2.92)
	100	65	67.69 (5.91)	68.74 (6.70)	67.18 (8.01)
Gamma	21	55	57.62 (3.05)	57.15 (3.66)	55.18 (2.60)
	30	74	76.41 (3.91)	76.64 (4.22)	74.64 (4.54)
	120	98	99.87 (6.32)	104.13 (6.71)	99.15 (8.65)

Table 10.3: Optimal order quantity in the newsvendor model and mean (standard deviation) of estimates based on 40 replicates of Q-learning with three different learning rates.

Q-learning: An "Earning while learning" perspective

The above analysis is from the perspective of solving an optimization problem; that is, Q-learning is an online method to optimize the order quantity. Alternatively, from a simultaneous earning and learning perspective, the newsvendor might seek to maximize revenue while learning what quantity to order.

To illustrate this approach, consider an instance with normal demand, price equal 30, $N = 100{,}000$ iterates, the three learning rates used above, and softmax action selection with $\eta_n = 600n^{0.2}$. The index for the learning rate was the number of times an action was previously used and the index for softmax sampling was the iteration number. The effect of this specification for η_n is to allow random action choice at the beginning and then make it more likely that sampling is done from actions with high values of the estimate of $q(a)$, which is shown in Figure 10.4. Observe that in this replicate, the estimate of $q(a)$ is smoothest around the optimal order quantity where most of the actions are chosen.

Figure 10.5 shows the running average revenue in a typical replicate[18]. Observe that all three learning rates result in learning as indicated by increasing average revenue per iterate. Using the learning rate $\tau_n = 5/(20+n)$ resulted in the greatest revenue early on. Eventually, revenue streams from all three learning rates become approximately the same but that using $\tau_n = 5/(20+n)$ continued to have the greatest average revenue. Note that over 100,000 iterates even a small increase in average revenue contributes to a large increase in total revenue.

10.2.3 Conclusions

The purpose of this section was to provide an introduction to the simulation methods in this chapter. By choosing a practical and easy to understand **one-state** example, several implementation issues that arise in multi-state models were swept under the rug.

[18] In 40 random sequences, the average revenues always exhibited this pattern.

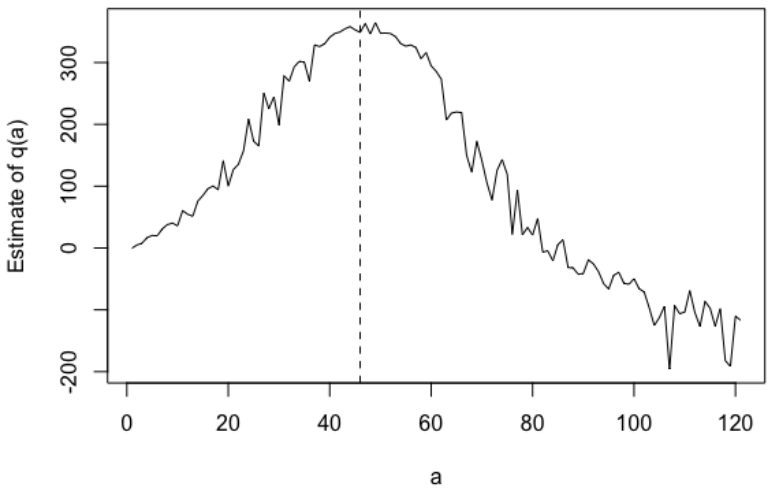


Figure 10.4: Estimate of $q(a)$ based on 100,000 iterates of Q-learning. The vertical dashed line corresponds to a^* found using (10.3).

However, this example was sufficiently rich to provide insight into issues arising when using these techniques. The remainder of this chapter generalizes these approaches to multi-state models.

Some noteworthy observations include:

1. Monte Carlo methods are easy to understand, however learning does not occur until all data has been collected. As the above calculations show, Monte Carlo methods involve costly experimentation for actions that may be far from optimal. However, because $q(a)$ is relatively flat near the maximum, large sample sizes near this value are useful. Note that Monte Carlo could be integrated with a search procedure to reduce experimentation.

2. Temporal differencing provides an online approach for policy value function estimation. This means that estimates are available after each observation. However, the quality of the approximation depends significantly on the learning rate. Note that $\tau_n = n^{-1}$ corresponds to an online version of Monte Carlo.

3. Q-learning worked best with softmax sampling and on-policy updates using (10.9). Again, outcomes were sensitive to the learning rates. After preliminary experimentation three promising learning rates were used in a more detailed study. Optimal order quantity estimates were quite similar with the exponential learning rate working slightly better than the others in determining an order quantity. From the perspective of learning, the ratio learning rate generated the greatest revenue. Differences in revenue over learning rates was greatest initially.

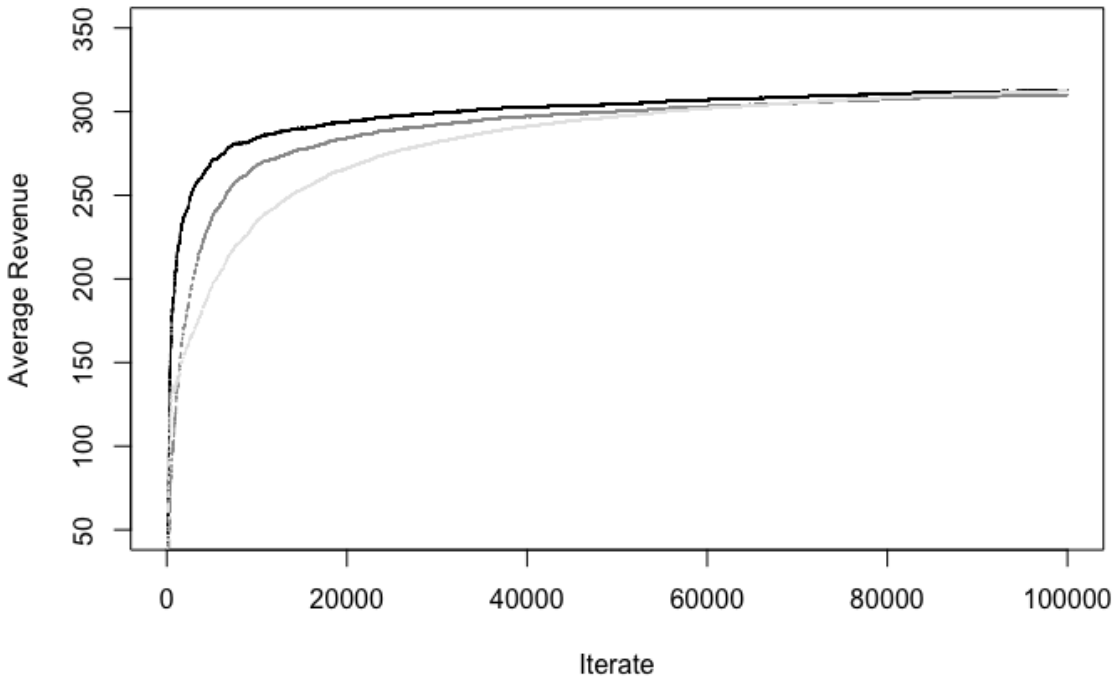


Figure 10.5: Cumulative average revenue vs. iterate using Q-learning. Data is from a single replicate of newsvendor model using softmax sampling and three learning rates. The black line corresponds to $\tau_n = 5/(20 + n)$, the dark grey line (between the other two) to $\tau_n = 0.05/(1 + (10^{-5})n^2)$ and the light grey to $\tau_n = 0.5n^{-0.5}$.

10.3 Policy evaluation: Episodic models

In an episodic model, the system accrues rewards until it reaches a reward-free absorbing state at some (possibly) random stopping time in the future. Usually this will be when a task is completed or a game ends. This section focuses on models in which stopping times are determined by the evolution of the system. For example, in a robot navigation model, an episode terminates when the robot achieves its goal[19].

As noted in Section 2.3.2, a discounted model may be viewed as an episodic model in which the episode terminates at a geometrically distributed stopping time independent of the evolution of the system. Policy evaluation for discounted models will be discussed in Section 10.4 below.

Finite horizon models may also be regarded to be episodic models in which the stopping time is fixed and after which the system transitions to a zero-reward absorbing

[19]The episode could end when the robot fails, but computation in various applications suggests that it is more expedient to terminate an episode only when the goal is achieved.

state. This chapter will not discuss finite horizon models[20] but many of the techniques here can be generalized to apply to them.

In episodic models, the expected total reward of stationary policy $\pi = d^\infty$ is given by

$$v^\pi(s) = E^\pi \left[\sum_{n=1}^{N-1} r(X_n, Y_n, X_{n+1}) + g(X_N) \middle| X_1 = s \right], \tag{10.12}$$

where the random variable N represents the random time at which the process terminates, X_n denotes the state at decision epoch n, Y_n denotes the action chosen at decision epoch n, X_{n+1} denotes the state at decision epoch $n+1$ that results from choosing action Y_n in state X_n, and $g(X_N)$ denotes the reward when terminating in state X_N[21]. The random variable $N = \min\{n \geq 1 \mid X_n \in \Delta\}$, sometimes referred to as the *effective horizon*, denotes the time the system enters a set of zero-reward absorbing states, Δ.

Chapter 6 established the optimality of deterministic policies[22] for such models. However, allowing for randomized policies broadens the applicability of the methods herein. Chapter 6 shows that in several classes of models[23], $v^\pi(s)$ is well-defined for most policies, including any policy that could be optimal.

10.3.1 Monte Carlo methods

The most obvious approach for estimating $v^\pi(\bar{s})$, for a fixed state $\bar{s}$, is to sum up the rewards for each episode and average over episodes. This leads to the following *Monte Carlo* algorithms, which differ according to which quantities are retained from each episode. Each assumes that an episode terminates when the system reaches a state in the set Δ.

Starting-state Monte Carlo

This simple Monte Carlo algorithm is stated for illustrative purposes. It produces an estimate for a single state $\bar{s}$ only. First-visit and every-visit variants are preferable and slightly more involved to state but easy to implement.

Algorithm 10.1. Starting-state Monte Carlo policy evaluation for an episodic model

[20] Chapter 4 showed that in finite horizon models, optimal policies are usually non-stationary.

[21] One could alternatively set $g(\cdot) = 0$ and include the termination reward in the definition of $r(X_{N-1}, Y_{N-1}, X_N)$. This is how the model was written in Chapter 6 but the representation with a specified terminal reward is more natural here.

[22] When T is stochastic, optimal policies will also be stationary; when T is fixed and constant, non-stationary policies are often optimal.

[23] Transient models, stochastic shortest path models, positive models and negative models.

1. **Initialize:**
   (a) Specify $d \in D^{\mathrm{MR}}$ and the number of episodes K.
   (b) Specify $\bar{s} \in S \setminus \Delta$ and create an empty list VALUES.
   (c) $k \leftarrow 1$.
2. **Iterate:** While $k \leq K$:
   (a) $s \leftarrow \bar{s}$ and $v \leftarrow 0$.
   (b) **Generate episode:** While $s \notin \Delta$:
      i. Sample a from $w_d(\cdot|s)$.
      ii. Simulate (s', r) or sample s' from $p(\cdot|s, a)$ and set $r \leftarrow r(s, a, s')$.
      iii. Update the value:
      $$v \leftarrow r + v. \tag{10.13}$$
      iv. $s \leftarrow s'$.
   (c) **Terminate episode ($s \in \Delta$):**
      i. Append $v + g(s)$ to VALUES.
      ii. $k \leftarrow k + 1$.
3. **Terminate:** Return
$$\hat{v}^{d^\infty}(\bar{s}) = \mathrm{mean}(\textsc{Values}).$$

Some comments about the algorithm follow:

1. The above algorithm estimates $v^{d^\infty}(\bar{s})$ by setting it equal to the average of observed total rewards for all episodes that start in state $\bar{s}$. In most cases, one would seek estimates for all $s \in S \setminus \Delta$.
2. If $d \in D^{\mathrm{MD}}$, step 2(b)i is replaced by "Set $a = d(s)$".
3. The algorithm statement includes a model-free variant in which state transitions and rewards are simulated or observed, and a model-based variant in which states are sampled and rewards computed using the triple (s, a, s').
4. Since the total rewards from different episodes are independent and identically distributed, $\hat{v}^{d^\infty}(\bar{s})$ converges almost surely to its mean, $v^{d^\infty}(\bar{s})$, by the *strong law of large numbers*, and is asymptotically normal by the *central limit theorem*.
5. By storing estimates of $v(\bar{s})$ in the list VALUES instead of updating the mean sequentially, one can investigate distributional properties of the estimates such as the standard deviation and percentiles. Moreover, one can compute standard

errors (i.e., standard deviation divided by the square root of the number of observations) and confidence intervals for the estimates, and use the width to guide the choice of K.

6. The algorithm as stated above *updates values only for the starting state* $\bar{s}$ of each episode. Obviously data would be used more efficiently if it also estimated $v(s)$ for *all* states visited during an episode. If a state repeats, the estimate for that state may be based on the total return accumulated from the first visit to that state (or from every visit to that state). If such an approach is used, $v(s)$ should be updated for *all* states previously visited in step 2(b). The first-visit algorithm is stated below.

7. Algorithm 10.1 is an offline algorithm since it requires all data to estimate $v^{d^\infty}(s)$ in step 3.

First-visit Monte Carlo

The following algorithm uses data more efficiently by storing cumulative rewards for each distinct state visited during an episode.

Algorithm 10.2. First-visit Monte Carlo policy evaluation for an episodic model

1. **Initialize:**
   (a) Specify $d \in D^{\text{MR}}$ and the number of episodes K.
   (b) For all $s \in S \setminus \Delta$ create an empty list VALUES(s).
   (c) $k \leftarrow 1$.
2. **Iterate:** While $k \leq K$:
   (a) Create empty list VISITED.
   (b) $v(s) \leftarrow 0$ for all $s \in S \setminus \Delta$.
   (c) Specify $s \in S \setminus \Delta$.
   (d) **Generate episode:** While $s \notin \Delta$:
      i. If $s \notin$ VISITED, append s to VISITED.
      ii. Sample a from $w_d(\cdot|s)$.
      iii. Simulate (s', r) or sample s' from $p(\cdot|s, a)$ and set $r \leftarrow r(s, a, s')$.
      iv. For all $s'' \in$ VISITED
      $$v(s'') \leftarrow r + v(s''). \tag{10.14}$$
      v. $s \leftarrow s'$.

(e) **Terminate episode ($s \in \Delta$):**

i. For all $s'' \in$ VISITED, append $v(s'') + g(s)$ to VALUES(s'').

ii. $k \leftarrow k + 1$.

3. **Terminate:** For all $s \in S \setminus \Delta$, return

$$\hat{v}^{d^\infty}(s) = \text{mean}(\text{VALUES}(s)).$$

Some comments about this algorithm follow:

1. The list VISITED keeps track of the *distinct* states visited during an episode and 2(d)iv updates the value for each.

2. An alternative to this algorithm would be an every-visit version, which in each replicate starts a new total every time a state is visited. That is, if the sequence of states was $(s_1, s_2, s_5, s_2, \ldots)$, such an algorithm would generate two estimates for s_2, one beginning from the first visit and one beginning form the second visit. It is left to the reader to formally state and apply such an algorithm.

An example

This section illustrates the starting-state and first-visit Monte Carlo algorithms by evaluating a policy for the Gridworld model of Section 3.2. The calculations assume an episode ends when *either* the robot reaches the office (cell 1) or falls down the stairs (cell 7). Thus $S = \{1, \ldots, 15\}$ and $\Delta = \{1, 7\}$. The cell configuration diagram is repeated in Figure 10.6 below.

Consider the Gridworld model represented in Figure 10.6 with $p = 0.8$, a cost of 200 for falling down the stairs, a reward of 50 for successfully delivering the coffee and a per unit move cost of 1. This example analyzes the stationary policy d^∞ in which the robot seeks to move to the lowest numbered neighboring cell except when in state 10 when it instead tries to move toward cell 11. It moves to the intended cell with probability p and to each other adjacent cell with equal probability. The formulation assumes that the robot cannot move diagonally. For example, from cell 14, the robot moves to cell 11 with probability p and to cells 13 or 15 with probability $(1 - p)/2$.

The example compares Algorithm 10.1 and Algorithm 10.2, using $K = 2{,}000$ replicates for each starting state. Figures 10.7a and 10.7b show the estimates of the values starting in each state using the two algorithms.

Observe that estimates based on first-visit Monte Carlo had narrower simultaneous 95% confidence intervals[24] than those based on starting-state Monte Carlo. For example, the standard errors when starting in cell 13 were 2.34 for starting-state and 2.07

[24]Such confidence intervals are constructed so that in 95% of samples all intervals contain the true mean. A Bonferroni correction (0.025 divided by the number of estimates) is used to obtain the appropriate t-value.

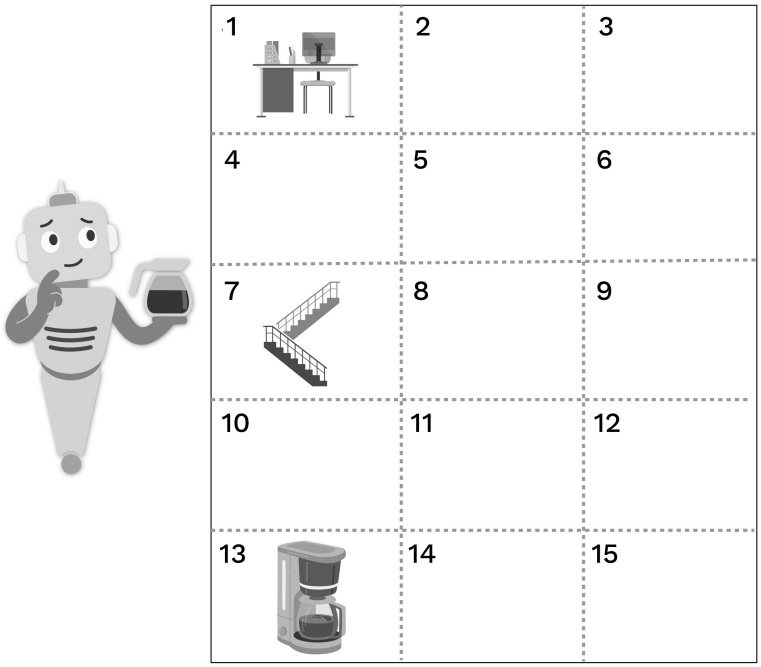


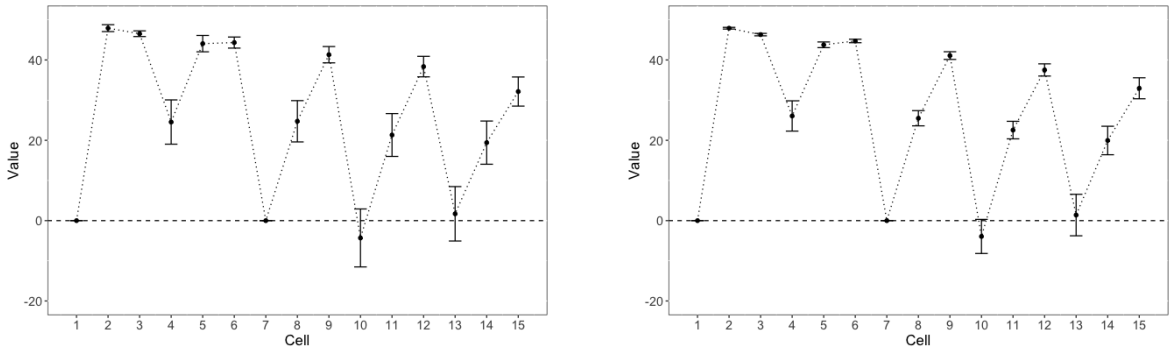
Figure 10.6: Schematic layout for Gridworld navigation example.

for first-visit. When starting in cell 2, the respective standard errors were 0.30 and 0.08. Thus, as expected, the first-visit Monte Carlo algorithm provides more accurate estimates with the same number of replicates especially for the most frequently visited states.

Figure 10.8 provides a histogram of values for cell 13 when using first-visit Monte Carlo. Observe that the distribution is extremely bi-modal corresponding to the possibilities of falling down the stairs or successfully delivering the coffee. Consequently, the mean does not represent this distribution well and is perhaps a poor choice of a value of a policy. Note that the 5th percentile is -204, the 25th percentile is 38, the median is 42 and the 75th percentile and 95th percentiles equal 44. Thus, one might wish to consider the use of different summaries of the distribution (such as the median or 75th percentile) when analyzing such models.

10.3.2 Temporal differencing

Temporal differencing represents one of the most significant innovations in reinforcement learning. It refers to methods that update policy value function estimates based



(a) Starting-state Monte Carlo

(b) First-visit Monte Carlo

Figure 10.7: Value function estimates for the specified policy based on 2,000 replicates for each starting state and $p = 0.8$. Error bars in Figures 10.7a and 10.7b represent simultaneous (Bonferroni) 95% confidence intervals.

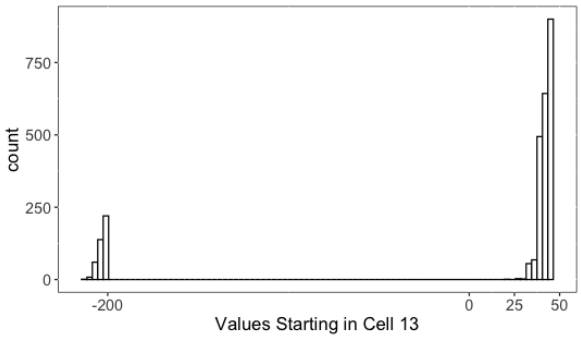


Figure 10.8: Histogram of estimates of $v(13)$ using Algorithm 10.2. Note the mean of these estimates is 0.64.

on a fraction of the difference between a current estimate and a one-step prediction equal to an observed reward and the value in a successor state. The difference is called the *temporal difference*. The term, temporal differencing, has its roots in behavioral neuroscience referring to situations in which actions are reinforced when predictions exceed expectations and degraded when they do not.

To motivate this approach, consider the operator

$$L_d\mathbf{v} := \mathbf{r}_d + \lambda \mathbf{P}_d\mathbf{v}$$

on the set of real-valued vectors on S when using a fixed decision rule $d \in D^{\text{MR}}$. When $0 \leq \lambda < 1$, it plays a fundamental role in discounted models (Chapter 5), and when $\lambda = 1$, it is fundamental in transient models and applies to proper policies in stochastic shortest path models (Chapter 6). In these models, value iteration, expressed as

$$\mathbf{v}' = L_d\mathbf{v} \tag{10.15}$$

converges to the unique fixed point of L_d, which was shown to equal the value function of d^∞. Rewriting (10.15) as

$$\mathbf{v}' = \mathbf{v} + (L_d\mathbf{v} - \mathbf{v}) \tag{10.16}$$

provides a conceptually different approach for thinking of value iteration, namely:

> new estimate = old estimate + correction.

A related numerical iterative scheme, referred to as *successive over-relaxation*, modifies value iteration by introducing a scaling factor τ satisfying $0 \leq \tau \leq 1$ that down weights the correction to obtain:

$$\mathbf{v}' = \mathbf{v} + \tau(L_d\mathbf{v} - \mathbf{v}). \tag{10.17}$$

It is not hard to show that the expression on the right hand side of (10.17) is a contraction mapping and that it also converges to the unique fixed point of L_d.

Suppose instead of computing $L_d\mathbf{v}$ directly, one instead observes a random sample $\mathbf{u}$ from a distribution with expectation[25] $L_d\mathbf{v}$, then the obvious generalization of (10.17) becomes

$$\mathbf{v}' = \mathbf{v} + \tau\boldsymbol{\delta} \tag{10.18}$$

where

$$\boldsymbol{\delta} = \mathbf{u} - \mathbf{v}.$$

Temporal differencing applies such an approach on a state-by-state basis by choosing $s \in S$, using the decision rule $d(s)$ to obtain action $a \in A_s$, observing r' and s', and setting:

$$\delta = r' + \lambda v(s') - v(s)$$

and

$$v'(s) = v(s) + \tau\delta.$$

In this expression, δ is the temporal difference. This approach uses a concept that is often referred to as *bootstrapping*, namely it uses current value estimates to update other value estimates.

Stochastic approximation using whole trajectories

As a different motivation for temporal differencing, consider updates obtained using stochastic approximation based on *complete* episodes. Appendix 10.11.2 of this chapter introduces the following recursion for estimating the mean μ of a random variable:

[25] This means that $E[\mathbf{u}] = L_d\mathbf{v}$, for example when $\mathbf{u} = L_d\mathbf{v} + \boldsymbol{\epsilon}$ with $E[\boldsymbol{\epsilon}] = \mathbf{0}$.

$$\mu^{n+1} = \mu^n + \tau_n\big(y^n - \mu^n\big). \tag{10.19}$$

The appendix shows that this recursion is the result of applying stochastic gradient descent to obtain least squares estimates of a mean. The parameters $\tau_n, n = 1, 2, \ldots$ are referred to as *learning rates* or *step-sizes*.

Convergence of μ^n to μ occurs with probability 1 if the following conditions[26] are satisfied.

Definition 10.1. The sequence $\tau_n, n = 1, 2, \ldots$, satisfies the *Robbins-Monro step-size conditions* if

$$\tau_n \geq 0, \quad \sum_{n=1}^{\infty} \tau_n = \infty \quad \text{and} \quad \sum_{n=1}^{\infty} \tau_n^2 < \infty. \tag{10.20}$$

Let $(h^1, h^2, \ldots)$ denote independent samples from a distribution $H(\cdot)$ with expectation equal to the right hand side of (10.12). Then, applying (10.19) provides recursive estimates of $v^{d^\infty}(s)$ given by

$$v^{n+1}(s) = v^n(s) + \tau_n\big(h^n - v^n(s)\big). \tag{10.21}$$

One way to produce these samples is by generating complete episodes and setting

$$h^n = \sum_{l=1}^{N^n} r_l^n$$

where r_l^n is the l-th reward in the n-th episode and N^n is the length of the n-th episode. The astute reader will recognize that this is equivalent to generating a sequence of starting-state Monte Carlo estimates and updating the value function estimate using (10.19) after each episode completes[27].

The downsides of this approach are that:

1. It delays updating the estimate until the completion of an episode.
2. It wastes considerable within-episode information.
3. A separate estimate must be obtained for each $s \in S$.

[26] These are frequently referred to simply as the *Robbins-Monro conditions* or the *usual conditions*.

[27] Sutton and Barto (2018) refers to such an approach as Monte Carlo-α and h^n as the *target* of the update.

An online alternative

To address these concerns, instead of basing updates on an entire reward sequence, temporal differencing bases updates on the (one-step) evaluation equation

$$v(s) = E^d\left[r(X, Y, X') + v(X') \,\middle|\, X = s\right] = E^d\left[r(s, Y, X') + v(X')\right] \tag{10.22}$$

derived in (6.23). In this expression, the expectation is with respect to the distribution of the action a chosen by decision rule d in state s (when it is randomized) and the resulting transition probabilities $p(\cdot|s, a)$.

An implementable recursion is based on replacing the sampled trajectories h^n in (10.21) by the results of a single simulation step. In this step, a is deterministically chosen as $a = d(s)$ or sampled from the randomized distribution $w_d(\cdot|s)$. The successor state s' and reward r are jointly simulated or s' is sampled from $p(\cdot|s, a)$ and the reward is computed by $r = r(s, a, s')$. This enables direct calculation of $r + v^n(s')$. Thus, the following recursion

$$v^{n+1}(s) = v^n(s) + \tau_n\big(\, r + v^n(s') - v^n(s)\,\big) \tag{10.23}$$

provides an online update of an estimate of $v(s)$ without waiting for an episode to complete.

Often this recursion appears as

$$v^{n+1}(s) = v^n(s) + \tau_n \delta^n \tag{10.24}$$

where

$$\delta^n := r + v^n(s') - v^n(s).$$

As noted above, the quantity δ^n is referred to as a temporal difference.

When applying this recursion, the usual approach is to start in some state, generate a transition using the policy, update value estimates using (10.23) and terminate when reaching a reward-free absorbing state. However, to obtain accurate estimates in each state, one would need to repeat this over multiple episodes by restarting this process in a state chosen either deterministically or randomly.

The following subtleties arise when using this recursion:

1. **Approximation:** This recursion uses *bootstrapping*[28]. That is, it uses an approximation of the value function $v^n(\cdot)$ at each step instead of a known quantity.

[28] This use of the expression bootstrapping here is consistent with the reinforcement learning literature. In the statistical literature, bootstrapping refers to methods based on resampling from a given set of data.

2. **Multiplicity of successor states:** The recursion requires estimates of $v^n(s')$ for all states reachable from s in one step under d.

3. **Asynchronicity:** The quality (variance) of the approximations will vary across states. Values of states frequently visited under d will be estimated accurately while those infrequently visited will be less accurate.

As a consequence of these remarks, a rigorous theoretical analysis of this algorithm requires methods above and beyond those used to establish convergence of stochastic approximation. In spite of this, many proofs of the following result are available[29]. This issue is explored in Chapter 11 in a more transparent setting where value functions are parameterized by a vector $\boldsymbol{\beta}$ and a distinction is made between gradients and semi-gradients.

Theorem 10.1. Let $\mathbf{v}^{d^\infty}$ denote the unique solution to (10.22). Then for every $\mathbf{v}^0$, if every state is visited infinitely often and τ_n satisfies (10.20), $\mathbf{v}^n$ converges to $\mathbf{v}^{d^\infty}$ with probability 1.

Note that the expression "with probability 1" can be interpreted as "on almost every sample path", where sample paths consist of sequences of states, actions and rewards determined by the decision rule, the transition probabilities, and the distribution used to start a new episode. To ensure that every state is visited infinitely often, after reaching a terminal state, the system must be restarted using a distribution that ensures all states are reached with positive probability.

Example 10.1. The example depicted in Figure 10.9 illustrates the above challenges of applying (10.23). In this example of an episodic model, there is a single stationary policy that uses action a_1 in s_1, a_2 in s_2 and a_3 in s_3. Actions choose arcs with the indicated probabilities and generate the indicated rewards. Note s_3 is a zero-reward absorbing state. When s_3 is reached, an episode ends.

Suppose the system starts in s_1, then in 999 out of 1,000 trajectories will immediately jump to s_2, remain there for on average 500 transitions and then jump to either s_1 or s_3 with equal probability. Therefore the value of s_2 will be accurately estimated but the value of s_1 need not be.

State-based sampling provides an attractive alternative to requiring a large number of episodes to accurately estimate $v(s_1)$. Of course, in a "real-world" system such restarts may not be possible.

[29]The proofs of this result are beyond the level of this book; references appear in the Bibliographic Remarks section.

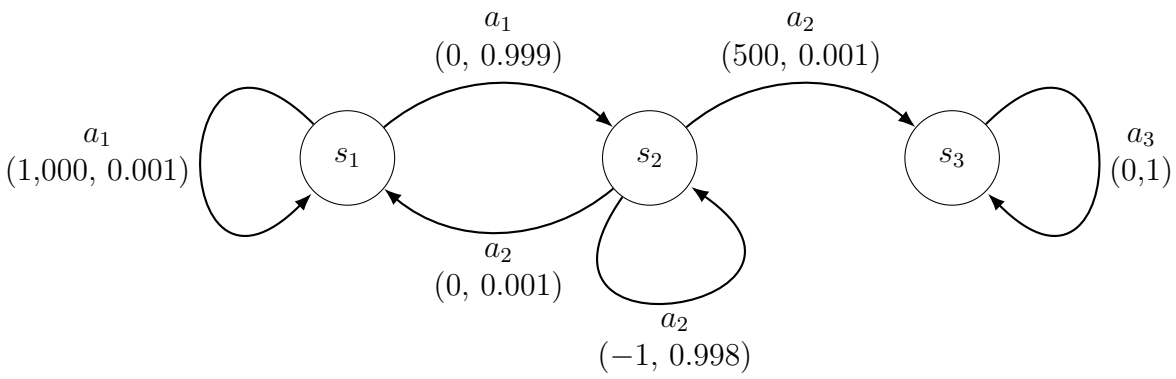


Figure 10.9: Graphical representation of model in Example 10.1. Numbers in parentheses represent $(r(s,a,s'), p(j|s,a))$.

Temporal differencing in algorithmic form

This section describes an algorithm to implement (10.23) in an episodic model. The algorithm evaluates a randomized stationary policy d^∞, and combines both model-free and model-based implementations.

Algorithm 10.3. Temporal differencing for policy evaluation for an episodic model

1. **Initialize:**
   (a) Specify $d \in D^{\mathrm{MR}}$.
   (b) Specify the number of episodes K.
   (c) Specify the learning rate sequence $\tau_n, n = 1, 2, \ldots$.
   (d) $v(s) \leftarrow 0$ for all $s \in S$.
   (e) $\mathrm{count}(s) \leftarrow 0$ for all $s \in S$.
   (f) $k \leftarrow 1$.
2. **Iterate:** While $k \leq K$:
   (a) Specify $s \in S \setminus \Delta$.
   (b) **Generate episode:** While $s \notin \Delta$:
      i. $\mathrm{count}(s) \leftarrow \mathrm{count}(s) + 1$.
      ii. Sample a from $w_d(\cdot|s)$.
      iii. Simulate (s', r) or sample s' from $p(\cdot|s,a)$ and set $r \leftarrow r(s,a,s')$.

iv. Evaluate temporal difference

$$\delta \leftarrow r + v(s') - v(s). \tag{10.25}$$

v. Set

$$v(s) \leftarrow v(s) + \tau_{\text{count}(s)}\delta. \tag{10.26}$$

vi. $s \leftarrow s'$.

(c) $k \leftarrow k + 1$.

3. **Terminate:** Return $v(s)$ for all $s \in S$.

Some comments about Algorithm 10.3 follow:

1. Temporal differencing is an online algorithm with asynchronous updates.
2. It provides estimates of the policy value function for every state visited during at least one episode when episodes are replicated.
3. In contrast to Monte Carlo estimation in which value function updates occur at the end of an episode, this algorithm updates the value function estimate within each episode after each transition. At the end of an episode, a new initial state is generated in step 2(a).
4. The initial state may be generated by any, possibly random, mechanism that is appropriate for a problem. For example, it is often advantageous to start far from absorbing states so that many states may be visited (and the values corresponding to those states updated) during each episode. However, a random start over all states may provide accurate estimates for states visited infrequently under decision rule d. This will be particularly relevant when seeking an optimal policy.
5. Monte Carlo methods might be used to obtain good starting values for $v(s)$ before applying temporal differencing.
6. The algorithm uses the number of visits to a state as the index for the learning rate parameter τ_n to account for asynchronicity in updates.
7. Choosing the learning rate τ_n is somewhat of an art. A systematic approach (sometimes referred to as a *parameter study*) that combines RMSE with graphical summaries for guidance is described in the chapter appendix 10.11.1 and illustrated in examples below. The next section expands on this issue.

8. After applying (10.26) in state s, it is possible to update values in states previously visited during an episode. This is the idea underlying TD(γ) algorithms, which are described below.

9. In step 2(b)iii, the next state s' (and the reward r) may be directly obtained using a simulator in a model-free environment or generated from transition probabilities and rewards in a model-based environment. Note that when d is deterministic, step 2(b)ii becomes "Set $a = d(s)$".

Learning rate choice

The following observation[30] is consistent with our experience choosing the learning rates (step-sizes): *"Step-size rules are important. As you experiment with ADP[31], it is possible to find problems where provably convergent algorithms simply do not work, and the only reason is a poor choice of step-sizes."*

Ideally, learning rates should produce fast learning early and slow learning later. Theorem 10.1 provides conditions (10.20) that do this and as well guarantee convergence of temporal differencing. However, ensuring that learning rates satisfy these conditions is not sufficient to obtain accurate estimates in practice.

Learning rate sequences that satisfy the Robbins-Monro conditions include:

Polynomial:

$$\tau_n = \alpha n^{-\beta}$$

where $0.5 < \beta \leq 1$. Note that $1/n$ is a special case corresponding to $\alpha = \beta = 1$.

Ratio:

$$\tau_n = \frac{\alpha}{\beta + n}$$

for $\alpha > 0$ and $\beta \geq 0$. Note that this sequence starts at $\alpha/(\beta + 1)$. Moreover when $\alpha = 1$ and $\beta = 0$ this reduces to $1/n$.

Search-then-converge (STC)[32]:

$$\tau_n = \alpha \left(\frac{1 + \frac{\beta}{\alpha}\frac{n}{\gamma}}{1 + \frac{\beta}{\alpha}\frac{n}{\gamma} + \frac{n^2}{\gamma}} \right)$$

When n is much smaller than γ, τ_n is nearly constant while for large n it behaves as β/n. Notice that choosing $\beta = 0$ makes τ_n decay more quickly.

Log:

$$\tau_n = \log(n + 1)/n$$

A multiplicative constant can be added to increase flexibility.

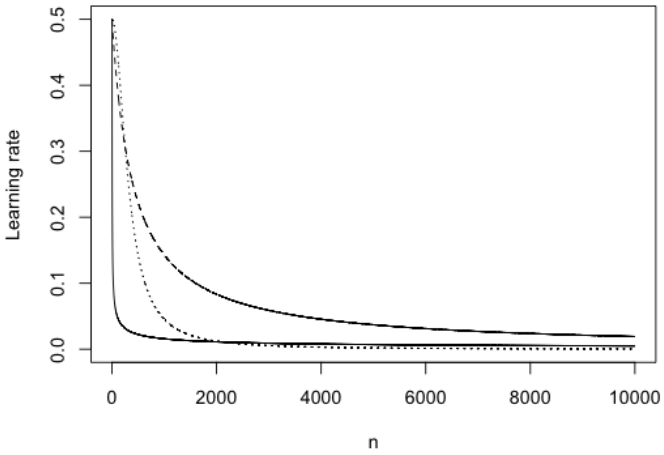


Figure 10.10: Graphical representation of three learning rate sequences. The solid line represents the polynomial sequence with $\alpha = \beta = 0.5$, the dashed line indicates the ratio sequence with $\alpha = 200$ and $\beta = 400$ and the dotted line corresponds to the STC sequence with $\alpha = 0.5$, $\beta = 0.0001$ and $\gamma = 100{,}000$.

Figure 10.10 shows how the learning rate decays with n for three specific τ_n. Observe that for the indicated parameter values, the polynomial sequence decays very quickly, the ratio sequence decays more slowly and the STC sequence lies somewhere in between.

Note that the quantity n in the above specifications may represent either:

1. the iteration number, or
2. the number of times a state[33] has been visited.

The iteration number benefits from simplicity. However, in algorithms using trajectory-based sampling, in which some states are visited infrequently, using the number of times a state has been visited seems more reasonable. It ensures that infrequently visited states are updated at a larger learning rate than if the iteration number had been used. Of course this involves extra storage but when a tabular model specification is appropriate, this should not be problematic. When states, or state-action pairs, are sampled randomly, either specification can be used.

[30] Powell, p. 184.
[31] In this context, ADP refers to simulation-based methods.
[32] This approach was described in Darken et al. [1992].
[33] Or state-action pair when estimating a state-action value function.

Temporal differencing in the Gridworld model

This section applies Algorithm 10.3 to the Gridworld model analyzed above using Monte Carlo methods in Section 10.3.1. It estimates the value function for the policy specified in that example. It compares four learning rates over 40 replicates of 10,000 episodes each with common random number seeds for each replicate. Each episode starts in cell 13 (the coffee room) and terminates when the robot reaches either cell 1 or 7.

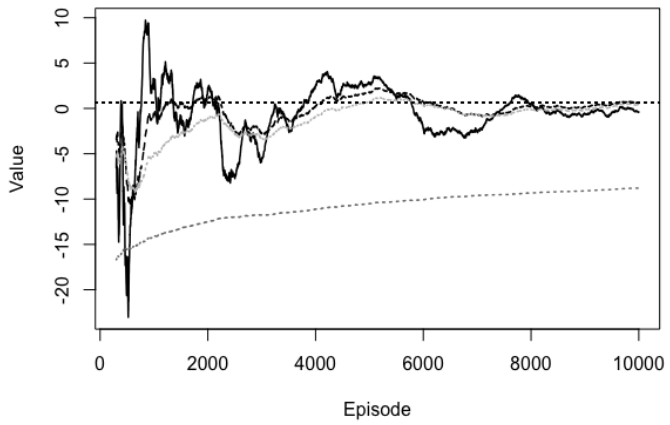


Figure 10.11: Estimates of $v^{d^\infty}(13)$ in the Gridworld model using Algorithm 10.3 based on a single replicate of 10,000 episodes. The figure shows the sequence of iterates for $\tau_n = 20/(50+n)$ (solid black), $\tau_n = \log(n+1)/n$ (dashed black), $\tau_n = 0.8n^{-0.75}$ (light grey) and $\tau_n = 0.8n^{-0.95}$ (dashed grey). The horizontal dotted line corresponds to the theoretical (true) value of this quantity 0.638.

Figure 10.11 displays a sequence of temporal difference estimates of $v^{d^\infty}(13)$, the expected total reward achieved by the robot starting in the coffee room. It shows iterates for the four choices of τ_n described in the figure caption. These were chosen based on some preliminary parameter studies.

Observe that $\tau_n = \log(n+1)/n$ and $\tau_n = 0.8n^{-0.75}$ produced the best estimates in this replicate. They converged quickly to the true value and oscillations damped out. Estimates using $\tau_n = 20/(50+n)$ oscillated around the true value but were more variable than the other two choices. That for $\tau_n = 0.8n^{-0.95}$ stabilized quickly and had low variance but was extremely inaccurate.

Figure 10.12 illustrates the variability in RMSE across 40 replicates where the

RMSE is computed over non-terminal states using

$$\text{RMSE} = \left(\frac{1}{13} \sum_{s \in S \setminus \Delta} \big(\hat{v}(s) - v(s) \big)^2 \right)^{\frac{1}{2}}. \tag{10.27}$$

In this expression, $\hat{v}(s)$ is based on the final episode for that state, while the true value is determined exactly using the methods from Chapter 6. This measure accounts for estimation error in all non-terminal states.

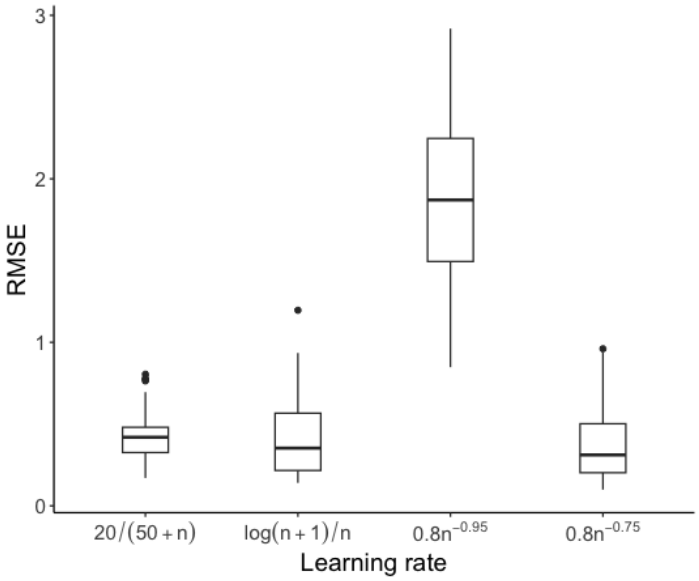


Figure 10.12: Boxplots of RMSE showing effect of learning rate when applying temporal differencing in Gridworld model.

It shows that the ratio estimate was least variable but slightly less accurate than those based on the logarithmic learning rate and the polynomial learning rate $0.8n^{-0.75}$. Clearly $\tau_n = 0.8n^{-0.95}$ was the worst choice. Note that RMSE weights all states equally. Alternative weightings are possible.

10.3.3 TD(γ)

The temporal differencing method above updates values one state at a time. Consequently, it ignores the effect of the current temporal difference on previously visited state values. This section describes an approach that overcomes this. It may be viewed as either an *offline or online method*.

As motivation, suppose in the Gridworld model (Figure 10.6) that the robot starts in cell 4 and visits cells 5, 6, 3, and 2 before terminating in cell 1. Recall that each step costs 1 and termination in cell 1 generates a reward of 50.

Consider updates starting in cell 6. After a transition to cell 3, the temporal difference update would be

$$v(6) \leftarrow v(6) + \tau(-1 + v(3) - v(6)).$$

But the value function could just as well be updated using a two-step update based on transitions from 6 to 3 and then 3 to 2 according to

$$v(6) \leftarrow v(6) + \tau(-1 - 1 + v(2) - v(6)),$$

or even a three-step update

$$v(6) \leftarrow v(6) + \tau(-1 - 1 - 1 + 50 + v(1) - v(6)).$$

Instead of selecting just one, it might be expedient to combine these three updates in some way. TD(γ)[34] provides a structured way to do so.

Two-step evaluation equations

The following formalizes the above argument in a model-based environment. The temporal difference algorithm above updates an estimate of the value of stationary policy d^∞ in state s according to

$$v(s) \leftarrow v(s) + \tau\big(r(s, a, s') + v(s') - v(s)\big),$$

where a is sampled from $w_d(\cdot|s)$ and the subsequent state s' is sampled from $p(\cdot|s, a)$. As noted above, this may be viewed as applying gradient descent to minimize the squared difference between the two sides of

$$v(s) = E^{d^\infty}\big[r(X_n, Y_n, X_{n+1}) + v(X_{n+1}) \,\big|\, X_n = s\big]. \tag{10.28}$$

Applying a similar recursion for $v(X_{n+1})$ and substituting it into (10.28) leads to the two-step evaluation equation beginning at decision epoch n:

$$v(s) = E^{d^\infty}\big[r(X_n, Y_n, X_{n+1}) + r(X_{n+1}, Y_{n+1}, X_{n+2}) + v(X_{n+2})\big|X_n = s\big]. \tag{10.29}$$

This suggests an alternative two-step update for $v(s)$ given by

$$\begin{aligned} v(s) &\leftarrow v(s) + \tau\big(r(s, a, s') + r(s', a', s'') + v(s'') - v(s)\big) \\ &= v(s) + \tau\big(r(s, a, s') + v(s') - v(s) + r(s', a', s'') + v(s'') - v(s')\big) \\ &= v(s) + \tau\big(\delta + \delta'\big), \end{aligned} \tag{10.30}$$

[34] Note that most of the literature refers to this approach as TD(λ) but since this book uses λ to denote the discount factor, it instead uses γ to represent the algorithmic parameter.

where a' denotes the realization of the action at decision epoch $n+1$ and s'' is a realization of the state at decision epoch $n+2$,

$$\delta = r(s,a,s') + v(s') - v(s) \quad \text{and} \quad \delta' = r(s',a',s'') + v(s'') - v(s').$$

This implies that the update of $v(s)$ is delayed until two decision epochs later.

However, this update can be viewed as two one-step updates as follows. After the first transition, update

$$v(s) \leftarrow v(s) + \tau_1 \delta$$

and leave the values the same for all other states. Then, in the next iteration, update

$$v(s) \leftarrow v(s) + \tau_2 \delta' \text{ and } v(s') \leftarrow v(s') + \tau_2 \delta'$$

and leave values for other states unchanged. Note that in contrast to the combined two-step update in (10.30), different learning rates can be used in each iteration.

Online TD(γ) modifies the second update for state s by reducing its impact by a factor of γ, $0 \leq \gamma \leq 1$, so that the second update becomes

$$v(s) \leftarrow v(s) + \tau_2 \gamma \delta' \quad \text{and} \quad v(s') \leftarrow v(s') + \tau_2 \delta'.$$

Multi-step and weighted evaluation equations

Repeating the above argument leads to the m-step evaluation equation (beginning at decision epoch n):

$$v(s) = E^{d^\infty}\left[\sum_{k=0}^{m-1} r(X_{n+k}, Y_{n+k}, X_{n+k+1}) + v(X_{n+m}) \middle| X_n = s\right]. \tag{10.31}$$

Define

$$R_n^m := \sum_{k=0}^{m-1} r(X_{n+k}, Y_{n+k}, X_{n+k+1}) + v(X_{n+m}) \tag{10.32}$$

for $m \geq 1$. The random variable R_n^m represents the total reward over decision epochs $n, n+1, \ldots, n+m-1$ plus a terminal reward $v(X_{n+m})$. These cumulative rewards may be viewed as the result of *rolling out* a policy over multiple periods starting at decision epoch n. Observe that (10.31) can be rewritten compactly as

$$v(s) = E\left[R_n^m | X_n = s\right].$$

Instead of arbitrarily combining these rollouts or using a pre-specified number of them, an option is to use the following exponential weighted average:

$$v(s) = E\left[\sum_{m=1}^{\infty} (1-\gamma)\gamma^{m-1} R_n^m \,\middle|\, X_n = s\right]. \tag{10.33}$$

From (10.32), R_n^m is itself a summation, so interchanging the order of summation as carried out in Appendix 10.11.4 leads to the following equivalent representation for $v(s)$:

$$v(s) = v(s) + \sum_{m=0}^{\infty} \gamma^m D_{n+m}, \tag{10.34}$$

where

$$D_{n+m} := E\big[r(X_{n+m}, Y_{n+m}, X_{n+m+1}) + v(X_{n+m+1}) - v(X_{n+m})|X_n = s\big].$$

Offline TD(γ)

Equation (10.34) provides the basis for deriving an offline representation for TD(γ). As shown in Appendix 10.11.4, the offline TD(γ) representation becomes

$$v(s^n) \leftarrow v(s^n) + \tau \sum_{m=n}^{\infty} \gamma^{m-n} \delta_m, \tag{10.35}$$

where

$$\delta_k := r(s^k, a^k, s^{k+1}) + v(s^{k+1}) - v(s^k).$$

Under the assumption of a transient model or a proper policy in a stochastic shortest path model, the rewards based on a trajectory $(s^1, a^1, s^2, \ldots)$ equal zero after some finite number of terms. Therefore, even though (10.35) has an infinite number of terms, they will equal zero after some trajectory-specific index N so that the above sum can be evaluated.

This can be applied offline in a similar way to a sequential implementation of first-visit or every-visit Monte Carlo over many trajectories. To do so, start with initial values of $v(s)$ for all $s \in S \setminus \Delta$. After each trajectory is completed, update the values for all states visited during that trajectory using (10.35) with τ decreasing over trajectories.

Observe that when $\gamma = 0$, (10.35) becomes

$$v(s^n) \leftarrow v(s^n) + \tau \delta_n,$$

which is exactly the temporal differencing update encountered earlier. Thus:

Temporal differencing is equivalent to TD(0).

When $\gamma = 1$, (10.35) becomes

$$v(s^n) \leftarrow v(s^n) + \tau \sum_{m=n}^{\infty} \delta_m = (1-\tau)v(s^n) + \tau \sum_{m=n}^{\infty} r(s^m, a^m, s^{m+1}).$$

Since the quantity in the sum is a Monte Carlo estimate of the total reward starting in state s^n, TD(1) is equivalent to a stochastic approximation method that updates the value function estimate for each state using the Monte Carlo update as in (10.21). Thus:

A variant of Monte Carlo estimation is equivalent to TD(1).

From a theoretical perspective[35], offline (batch) TD(γ) converges with probability one to $v^{d^\infty}(s)$ for all $s \in S \setminus \Delta$ provided there are no inaccessible states under the initial distribution, $v^{d^\infty}(s)$ is finite for all $s \in S \setminus \Delta$, and the Robbins-Monro step-size conditions hold.

Online TD(γ)

To obtain an online version of TD(γ) requires a change in perspective. In each episode, it uses the most recently observed temporal difference to update values in *all previously* visited states in that episode. This is in contrast to the temporal differencing method described above which updates values for the most recently visited state. The weightings are chosen to be consistent with the geometric weightings in offline TD(γ).

Suppose that, under a specified stationary policy, the states visited during an episode of length N are denoted by $(s^1, s^2, \ldots, s^N)$. Then after observing s^n and computing δ_n, online TD(γ) updates the policy value function in states $(s^1, s^2, \ldots, s^n)$ according to:

$$\begin{aligned}
v(s^n) &\leftarrow v(s^n) + \tau_n \delta_n \\
v(s^{n-1}) &\leftarrow v(s^{n-1}) + \tau_n \gamma \delta_n \\
&\vdots \\
v(s^{n-k}) &\leftarrow v(s^{n-k}) + \tau_n \gamma^k \delta_n \\
&\vdots \\
v(s^1) &\leftarrow v(s^1) + \tau_n \gamma^{n-1} \delta_n.
\end{aligned} \tag{10.36}$$

[35] See Dayan and Sejnowski [1994] which establishes this result for models with linear value function approximations, tabular models being a special case.

Observe that the update in state s^n is the same as in the temporal differencing algorithm and that the temporal difference δ_n is used to update all previously visited states.

It is important to note that when the current state s^n has been visited more than once in a trajectory, the above scheme updates its value multiple times. An alternative approach would be to only update the value *once* using the temporal differencing update $v(s^n) \leftarrow v(s^n) + \tau_n \delta_n$.

Eligibility traces

The *eligibility trace* is an innovation that assigns the appropriate weighting of the current temporal difference when updating values for previously visited states. Two variants have been proposed, the *accumulating eligibility trace* and the *replacement eligibility trace*, differing with respect to how values in repeat observations are updated. Often they will be referred to as simply an accumulating trace or a replacement trace.

More formally, an eligibility trace is a sequence of non-negative real-valued functions defined on a sequence of states $(s^1, s^2, \ldots)$ in $S \setminus \Delta$. The accumulating trace, which corresponds to updating values corresponding to all previous visits to the current state, is defined recursively by $e_1(s) = I_{\{s^1\}}(s)$ and

$$e_n(s) := \gamma e_{n-1}(s) + I_{\{s^n\}}(s). \tag{10.37}$$

Observe that for $n = 1$,

$$e_1(s) = \begin{cases} 1 & \text{for} \quad s = s^1 \\ 0 & \text{for} \quad s \neq s^1 \end{cases}$$

and $n > 1$,

$$e_n(s) = \begin{cases} \gamma e_{n-1}(s) + 1 & \text{for} \quad s = s^n \\ \gamma e_{n-1}(s) & \text{for} \quad s \neq s^n. \end{cases} \tag{10.38}$$

Using the eligibility trace as defined above, the sequence of updates (10.36) for $s \in S \setminus \Delta$ can be written in condensed form as:

$$v(s) \leftarrow v(s) + \tau_n \delta_n e_n(s). \tag{10.39}$$

The replacement trace modifies (10.38) for $n > 1$ to be

$$e_n(s) = \begin{cases} 1 & \text{for} \quad s = s^n \\ \gamma e_{n-1}(s) & \text{for} \quad s \neq s^n. \end{cases} \tag{10.40}$$

The effect of this modification becomes clear by using it in (10.39). It is easy to see that the consequence of using the replacement trace is that the update of $v(s^n)$ at iteration n is $v(s^n) \leftarrow v(s^n) + \tau_n \delta_n$, whereas when using the accumulating trace the expression $\tau_n \delta_n$ would be multiplied by a polynomial in γ.

An example

The following example[36] compares online and offline TD(γ) and also illustrates the effect of trace types on online updates. Consider a two-state model with a single policy. The state space consists of a state s_0 and a zero-reward absorbing state Δ. There is a single policy that generates a reward of r in s_0 at each decision epoch and the system remains in s_0 with probability p or jumps to Δ with probability $1-p$.

Since there is a single non-absorbing state, all temporal differences prior to absorption in Δ simplify to

$$\delta_k = r + v(s_0) - v(s_0) = r.$$

It is easy to see that $v(s_0) = r/(1-p)$, but that is not the point of this example. Suppose the system starts in s_0 and is absorbed in Δ after two transitions so that the trajectory is (s_0, s_0, Δ). The following calculations compare online and offline TD(γ).

Initialize calculations with $v(s_0) = c$. Then the offline update corresponding to this trajectory is

$$v^{\text{offline}}(s_0) \leftarrow c + \tau(1+\gamma)r.$$

The online updates using the accumulating trace and assuming $\tau_1 = \tau_2 = \tau$ are $e_1(s_0) = 1$ and

$$v_1^{\text{online}}(s_0) \leftarrow c + \tau r$$

corresponding to the first transition and since $e_2(s_0) = 1 + \gamma$,

$$v_2^{\text{online}}(s_0) \leftarrow c + \tau r + \tau(1+\gamma)r = c + 2\tau r + \tau\gamma r$$

corresponding to the second transition.

Using the replacement trace, $e_1(s_0) = e_2(s_0) = 1$ so that the first update is the same as using the accumulation trace, but the second update becomes

$$v_2^{\text{online'}}(s_0) \leftarrow c + 2\tau r.$$

Observe that:

1. The initial condition $v(s_0) = c$ persists in all updates. Therefore, it must be set to $c = 0$ to ensure convergence to the appropriate quantity.

2. The online and offline updates differ by a factor that is linear in τ.

3. The two online updates differ by a factor of $\tau\gamma$.

4. In trajectories of length n, the online update with replacement trace will be $c + n\tau r$. Since the expected total reward is $E[N]r$, the online estimate has exactly the same form as the quantity being estimated.

Consequently, if τ is small, estimates based on longer trajectories will be close.

[36] This is motivated by Example 5.7 in Bertsekas and Tsitsiklis [1996]. Their example has two non-absorbing states so it also shows the effect of the learning rate on transitions.

An online TD(γ) algorithm

The following algorithm implements online TD(γ). It incorporates the eligibility trace in the algorithmic flow. The precise update depends on whether an accumulating or replacement trace is used.

Algorithm 10.4. TD(γ) for policy evaluation for an episodic model

1. **Initialize:**
   (a) Specify $d \in D^{\mathrm{MR}}$.
   (b) Set $v(s) \leftarrow 0$, $e(s) \leftarrow 0$ and $\mathrm{count}(s) \leftarrow 0$ for all $s \in S$.
   (c) Specify $0 \leq \gamma \leq 1$ and learning rate sequence $\tau_n, n = 1, 2, \ldots$.
   (d) Specify the number of episodes K and set $k \leftarrow 1$.
2. **Iterate:** While $k \leq K$:
   (a) Specify $s \in S \setminus \Delta$.
   (b) **Generate episode:** While $s \notin \Delta$:
      i. $e(s) \leftarrow e(s) + 1$ (accumulating trace) or $e(s) \leftarrow 1$ (replacement trace).
      ii. $\mathrm{count}(s) \leftarrow \mathrm{count}(s) + 1$.
      iii. Sample a from $w_d(\cdot|s)$.
      iv. Simulate (s', r) or sample s' from $p(\cdot|s, a)$ and set $r \leftarrow r(s, a, s')$.
      v. Evaluate temporal difference
      $$\delta \leftarrow r + v(s') - v(s). \tag{10.41}$$
      vi. For all $s \in S$,
      $$v(s) \leftarrow v(s) + \tau_{\mathrm{count}(s)} \delta e(s) \tag{10.42}$$
      and
      $$e(s) \leftarrow \gamma e(s). \tag{10.43}$$
      vii. $s \leftarrow s'$.
   (c) $k \leftarrow k + 1$.
3. **Terminate:** Return $v(s)$ for all $s \in S$.

Some observations regarding this algorithm follow:

1. It is an online algorithm. It differs from Algorithm 10.3 in that at each state it updates values for **all** previously visited states. The magnitude of the update

decreases exponentially at rate γ.

2. A variant of the algorithm would reset eligibility traces to 0 at the end of each episode.

3. Small values of γ result in updating values for only a few previously visited states while values near 1 update this value far into the past.

4. The algorithm updates the eligibility trace online in steps 2(b)i and 2(b)vi. These ensure that the eligibility trace values agree with (10.38) and (10.40).

5. Convergence results apply to the algorithm implemented over infinitely many episodes. If every state is visited infinitely often over repeated episodes and the Robbins-Monro conditions are met, the online TD(γ) estimates converge[37] with probability one to the true policy value function.

An example

Before applying this algorithm in an example, examine how the eligibility trace updating works. Suppose in the Gridworld example the robot follows the trajectory $4 \to 5 \to 2 \to 5 \to 4 \to 1$. The above algorithm updates the eligibility trace at step 2(b)i prior to applying (10.42) and again in step 2(b)vi through (10.43).

Non-zero values of $e(s)$ (when using the accumulating trace) prior to applying (10.43) are given below:

1. $e(4) = 1$.

2. $e(5) = 1$, $e(4) = \gamma$.

3. $e(2) = 1$, $e(4) = \gamma^2$, $e(5) = \gamma$.

4. $e(5) = 1 + \gamma$, $e(2) = \gamma$, $e(4) = \gamma^3$.

5. $e(4) = 1 + \gamma^3$, $e(5) = \gamma(1 + \gamma)$, $e(2) = \gamma^2$.

Note that in step 5, $e(5)$ combines two terms to account for the two visits to cell 5.

If instead, the replacement trace had been used, the last two steps would be different. In step 4, $e(5) = 1$, and in step 5, $e(4) = 1$ and $e(5) = \gamma$. Note that if γ were near 1, the accumulating trace can exceed 1 in value.

The following example illustrates the application of TD(γ) to evaluate a policy in the Gridworld model.

[37] See Bertsekas and Tsitsiklis [1996] and Jaakkola et al. [1994] for details.

Example 10.2. Policy evaluation using TD(γ) in an episodic model

This example applies TD(γ) to the Gridworld model. Each episode starts in cell 13 and terminates when the robot reaches either cell 1 or cell 7. TD(γ) is used to again evaluate the policy that aims to move the robot to the lowest number adjacent cell, except in cell 10 when it attempts to move it to cell 11. It explores the effect of γ, trace type and learning rate on the accuracy of estimates.

Forty replicates of $K = 5{,}000$ episodes were compared for each combination of $\gamma \in \{0, 0.2, 0.4, 0.6, 0.8, 0.95\}$, accumulating and replacement eligibility trace and $\tau_n = 0.8n^{-0.75}$, $\tau_n = 20/(50+n)$, $\tau_n = \log(n+1)/n$ with the same seed in each replicate for all combinations. Results were assessed on the basis of the RMSE of the final estimates as defined in (10.27).

When $p = 0.8$, the most accurate estimates were obtained using $\tau_n = 0.8n^{-0.75}$, $0 \leq \gamma \leq 0.6$ with the trace type having little effect when γ was small (Figure 10.13). Observe also that for large values of γ, using the replacement trace resulted in more accurate estimates.

Moreover with this learning rate, the TD(γ) estimates were similar for $\gamma = 0, 0.2, 0.4$ with minimal differences between their accuracy. Consequently, there was minimal benefit using TD(γ) instead of temporal differencing (TD(0)).

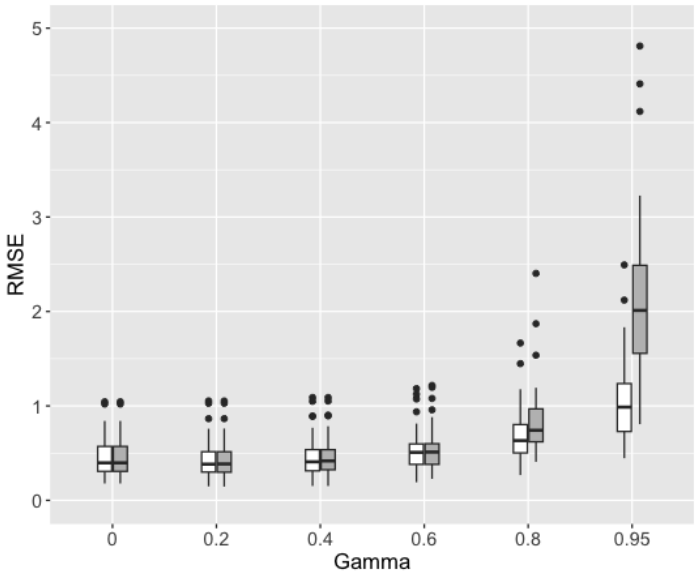


Figure 10.13: Boxplot of RMSE for Gridworld model showing the effect of γ and trace type (white represents replacement trace and grey represents accumulating trace) for learning rate $\tau_n = 0.8n^{-0.75}$.

10.4 Policy evaluation: Infinite horizon discounted models

This section discusses Monte Carlo and TD(γ) approaches for estimating the value of a policy in a discounted model.

10.4.1 Monte Carlo methods

Monte Carlo methods in an infinite horizon discounted model are based on either truncating the reward sequence or viewing the discount rate as the parameter of a geometrically distributed stopping time. Only starting-state Monte Carlo is applicable because the truncation index or geometric stopping time are based on the first decision epoch. For example, suppose an episode is truncated (or stopped) after 50 decisions, the system starts in state s^1 and visits states $s^2, s^3, \ldots, s^{50}$. Then an episode starting in state s^{25} would be based on accumulating only 25 rewards, while that starting in state s^1 would be based on accumulating 50 rewards. Therefore, an estimate based on s^{25} would not estimate the value function to the same precision as one based on s^1. This differs from the discussion above of episodic models where stopping occurs when a state in Δ is reached.

More formally, a policy value function in a discounted model may be written as

$$v_\lambda^{d^\infty}(s) = E^{d^\infty}\left[\sum_{n=1}^{\infty} \lambda^{n-1} r(X_n, Y_n, X_{n+1}) \,\middle|\, X_1 = s\right].$$

Because of the infinite sum, Monte Carlo methods cannot be applied directly to evaluate $v_\lambda^{d^\infty}(s)$. Instead, they can be based on approximating $v_\lambda^{d^\infty}(s)$ by either a *truncated sum of discounted rewards*

$$v_\lambda^{d^\infty}(s) \approx E^{d^\infty}\left[\sum_{n=1}^{M} \lambda^{n-1} r(X_n, Y_n, X_{n+1}) \,\middle|\, X_1 = s\right] \tag{10.44}$$

for some finite and sufficiently large M or the *undiscounted random sum* representation

$$v_\lambda^{d^\infty}(s) = E_T^{d^\infty}\left[\sum_{n=1}^{T} r(X_n, Y_n, X_{n+1}) \,\middle|\, X_1 = s\right], \tag{10.45}$$

where T is an independent geometric random variable with parameter 1–λ. In the latter case, the expectation is with respect to the probability distribution of $(X_1, Y_1, X_2, \ldots)$ generated by the specified policy and T. In (10.45), the discount factor appears implicitly in the expectation of T. Note that equation 10.45 is exact while 10.44 is an approximation.

Truncated reward sequences

Given a realization of states and actions, $(s^1, a^1, s^2, \ldots)$, at issue is when to truncate $\sum_{n=1}^{\infty} \lambda^{n-1} r(s^n, a^n, s^{n+1})$. This can be done *a priori* by choosing M so that

$$\lambda^{M-1} \max_{s \in S, a \in A_s, s' \in S} |r(s, a, s')| \leq \epsilon \tag{10.46}$$

for some pre-specified tolerance ϵ. Alternatively, M can be chosen arbitrarily or by stopping when the change in values of successive estimates is small.

Algorithm 10.5. Monte Carlo policy evaluation for a discounted model using truncation

1. **Initialize:**
   (a) Specify $d \in D^{\mathrm{MR}}$.
   (b) Specify the number of replicates K and set $k \leftarrow 1$.
   (c) Specify the truncation level M.
   (d) Specify $\bar{s} \in S$.
   (e) Create an empty list VALUES.
2. **Replicate:** While $k \leq K$:
   (a) $s \leftarrow \bar{s}$, $m \leftarrow 1$ and $u \leftarrow 0$.
   (b) **Generate a replicate:** While $m \leq M$:
      i. Sample a from $w_d(\cdot|s)$.
      ii. Simulate (s', r) or sample s' from $p(\cdot|s, a)$ and set $r \leftarrow r(s, a, s')$.
      iii. $u \leftarrow u + \lambda^{m-1} r$.
      iv. $s \leftarrow s'$.
      v. $m \leftarrow m + 1$.
   (c) **Update:**
      i. Append u to VALUES.
      ii. $k \leftarrow k + 1$.
3. **Terminate:** Return

$$\hat{v}_{\lambda}^{d^\infty}(\bar{s}) = \mathrm{mean}(\textsc{Values}).$$

Some comments about the algorithm follow:

1. This algorithm uses the expression *replicate* to refer to a single realization of a

truncated reward series in contrast to the use of the expression *episode* in episodic Monte Carlo.

2. As stated, the algorithm updates *online* within each replicate to obtain an estimate u to append to the list of values generated so far. Alternatively, this algorithm can be implemented *offline* by generating a sequence $(r^1, \ldots, r^M)$ and setting $u = \sum_{m=1}^{M} \lambda^{m-1} r^m$.

3. As stated, the algorithm generates a list VALUES. The advantage of doing so is that it can be used to investigate distributional properties of estimators.

4. After evaluating u based on a complete replicate, the estimate of $v_\lambda^{d^\infty}(s)$ can be updated using the stochastic approximation recursion:

$$v(\bar{s}) \leftarrow v(\bar{s}) + \tau_n\big(u - v(\bar{s})\big). \tag{10.47}$$

Of course, choosing $\tau_n = n^{-1}$ gives an estimate of the mean.

Geometric stopping times

Using representation (10.45) for $v_\lambda^{d^\infty}(s)$ is equivalent to terminating each episode at the first "success" in a sequence of Bernoulli trials with success probability $1-\lambda$.

To implement this approach modify Algorithm 10.5 as follows:

1. At the start of each replicate, generate a random stopping time from a geometric distribution[38].

2. Set $\lambda = 1$ in 2(b)iii so that the update is $u \leftarrow u + r$.

An example

The following example compares variants of Monte Carlo estimation using the two-state model of Section 2.5. Of course it would not be prudent to use simulation in such a simple model but it provides simple illustrations of the issues arising when estimating policy value functions for a discounted model using simulation.

Example 10.3. Comparison of truncation and random geometric stopping time estimates

This example analyzes the two-state model using Monte Carlo estimation with truncation and geometric stopping times. It evaluates the deterministic stationary policy d^∞ with $d(s_1) = a_{1,1}$ and $d(s_2) = a_{2,2}$. Setting $\lambda = 0.9$ and solving

[38] Note that some programming languages (such as R) represent the geometric distribution as the number of failures up to the first success. If this is the case, it is necessary to add 1 to the simulated value to be consistent with (10.45).

$(\mathbf{I} - \lambda \mathbf{P}_d)\mathbf{v} = \mathbf{r}_d$ establishes that $v_\lambda^{d^\infty}(s_1) = 30.147$ and $v_\lambda^{d^\infty}(s_2) = 27.941$.

The example compares the estimate of $v_\lambda^{d^\infty}(s)$ based on fixed truncation at $M = 30, 50, 70$ and random geometric truncation where the estimates are updated between replicates using (10.47) with learning rates $\tau_n = \log(1 + n)/n$, $20/(50 + n)$, n^{-1}, $0.8n^{-0.75}$. As noted earlier, $\tau_n = n^{-1}$ provides a running estimate of the mean. For each combination of method and learning rate, 100 instances of 1,000 replicates each were generated with common random number seeds for each instance. The quality of the estimates was assessed on the basis of the RMSE over the two states for the 100 instances.

Figure 10.14 provides boxplots of the RMSEs by method and learning rate. Observe that:

1. For each method, the estimate based on the mean ($\tau_n = n^{-1}$) had the smallest median RMSE and was the least variable. This is to be expected since the mean over instances is the minimum variance unbiased estimator of the mean.

2. Estimates using the ratio learning rate were the least accurate and most variable.

3. The quality of truncation estimates improved with increasing M with the average accuracy of those for $M = 50$ and $M = 70$ similar. The mean RMSE for $M = 50$ and $M = 70$ differed by 0.005 and the standard deviation when $M = 50$ was 0.44 and that for $M = 70$ was 0.48.

4. For all learning rates, the estimates based on geometric stopping times were most variable. This could be partially explained by noting that the mean number of episodes evaluated using geometric stopping times is $(1-\lambda)^{-1} = 10$.

The above observations indicate that using the Monte Carlo estimate with $M = 50$ or $M = 70$ gave the most precise estimates. Nothing was to be gained by using the hybrid TD-Monte Carlo approach based on (10.47). The reason for the imprecision of the geometric estimates was the presence of an additional source of variability resulting from simulating the reward sequence length.

10.4.2 Online TD(γ) in an infinite horizon discounted model

Because reward sequences for discounted models are infinitely long, offline TD(γ) faces the same challenges that impacted Monte Carlo estimation. Consequently, TD(γ) is typically implemented *online* using either trajectory-based or state-based data. While both data types are appropriate in simulations, only trajectory-based data is viable for real-time applications.

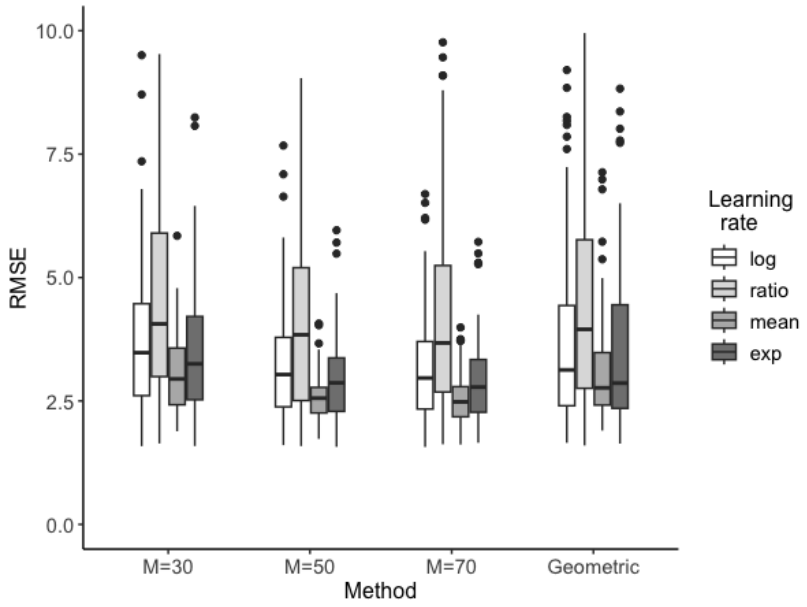


Figure 10.14: Boxplots of RMSEs for Example 10.3 by method and learning rate.

The TD(γ) algorithm below assumes that data is trajectory-based and the accumulating trace is used. A modification for a state-based approach is described below.

Algorithm 10.6. TD(γ) for policy evaluation for a discounted model

1. **Initialize:**
   (a) Specify $d \in D^{\mathrm{MR}}$.
   (b) Set $v(s) \leftarrow 0$, $e(s) \leftarrow 0$ and $\mathrm{count}(s) \leftarrow 0$ for all $s \in S$.
   (c) Specify $0 \leq \gamma \leq 1$ and learning rate sequence $\tau_n, n = 1, 2, \ldots$.
   (d) Specify the number of iterations N, set $n \leftarrow 1$ and specify $s \in S$.
2. **Iterate:** While $n \leq N$:
   (a) $e(s) \leftarrow e(s) + 1$.
   (b) $\mathrm{count}(s) \leftarrow \mathrm{count}(s) + 1$.
   (c) Sample a from $w_d(\cdot|s)$.
   (d) Simulate (s', r) or sample s' from $p(\cdot|s, a)$ and set $r \leftarrow r(s, a, s')$.
   (e) Evaluate temporal difference

$$\delta \leftarrow r(s, d(s), s') + \lambda v(s') - v(s). \tag{10.48}$$

(f) For all $s \in S$,

$$v(s) \leftarrow v(s) + \tau_{\text{count}(s)} \delta e(s) \tag{10.49}$$

and

$$e(s) \leftarrow \lambda\gamma e(s). \tag{10.50}$$

(g) $s \leftarrow s'$ and $n \leftarrow n + 1$.

3. **Terminate:** Return $v(s)$ for all $s \in S$.

Comments on online TD(γ) for discounted models follow:

1. In contrast to the undiscounted episodic model, the discount factor λ appears explicitly in (10.48) and (10.50).

2. The above algorithm is stated for an accumulating eligibility trace. Because in (10.50) the eligibility trace is dampened by the factor $\lambda\gamma$, the impact on previous terms is less than in the episodic model. Thus, distinguishing between the accumulating trace and the replacement trace should have less effect on estimates than in an episodic model. To use a replacement trace, replace step 2(a) by $e(s) \leftarrow 1$. The example below investigates the impact of trace type on estimate quality.

3. The Markov chain generated in step 2(d) may visit some states infrequently leading to high variability in policy value function estimates. Instead, when data is simulated, a state-based approach may be used. If that is the case, the first part of step 2(g) would be replaced by "Sample s from a discrete distribution on S." The distribution could be uniform or put more weight on critical states. This approach is sometimes referred to as *random restart*.

4. The algorithm may be sensitive to initial values. Choosing a good initial value such as

$$v_0(s) = (1 - \lambda)^{-1} r_d(s) \tag{10.51}$$

where $r_d(s) = E[r(X_0, d(X_0), X_1)|X_0 = s]$ for each $s \in S$ may enhance convergence. This value of $v_0(s)$ corresponds to starting the system in state s and receiving an identical reward of $r_d(s)$ at each decision epoch over the infinite horizon.

5. Online TD(γ) converges if all states are visited infinitely often and the usual conditions on τ_n apply.

6. Temporal differencing or TD(0) is a special case corresponding to $\gamma = 0$. In this case, the discount rate enters only through the temporal difference (10.48) since (10.50) sets $e(s) = 0$ for all $s \in S$.

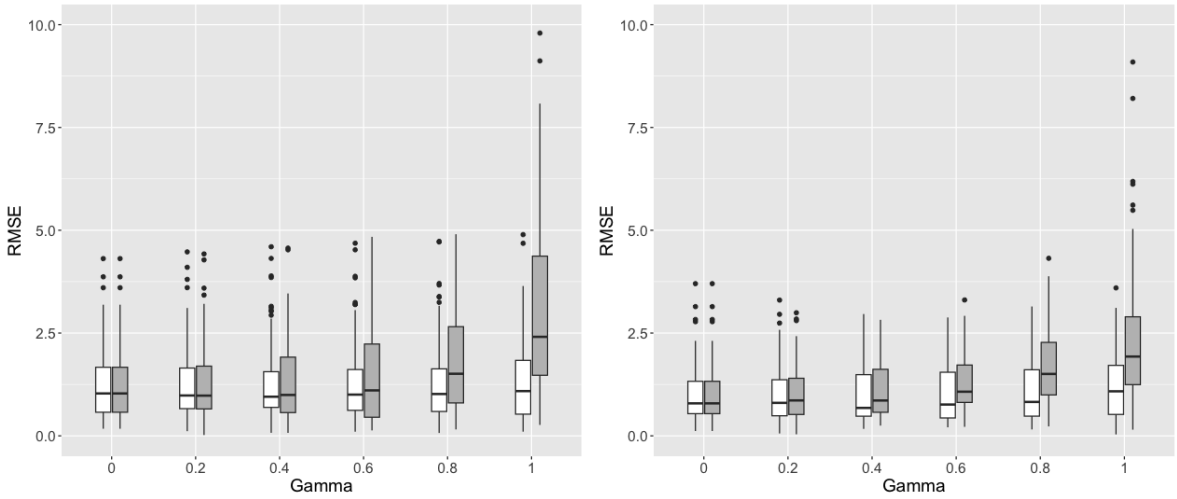
An example

The following example applies online TD(γ) to the two-state model in Section 2.5. It estimates the expected infinite horizon discounted reward, $v_\lambda^{d^\infty}(s)$, for the stationary deterministic policy d^∞ with $d(s_1) = a_{1,2}$ and $d(s_2) = a_{2,2}$. Setting $\lambda = 0.9$ and solving $(\mathbf{I} - \lambda \mathbf{P}_d)\mathbf{v} = \mathbf{r}_d$ establishes that $v_\lambda^{d^\infty}(s_1) = 30.147$ and $v_\lambda^{d^\infty}(s_2) = 27.941$. Estimation methods were compared using the RMSE of the final observation averaged over both states.

Experiments consisting of 40 replicates of length $N = 5{,}000$ with common random number seeds. They compared the following four factors:

1. Trace: accumulating vs. replacement;
2. Learning rate[39]: $\tau_n = 20/(50 + n)$ vs. $\tau_n = \log(n + 1)/n$;
3. TD(γ): $\gamma = 0, 0.2, 0.4, 0.6, 0.8$, and 1;
4. Simulation method: Trajectory-based vs. state-based (random restart using a uniform distribution).

Output was analyzed graphically and using analysis of variance[40] (ANOVA). Results include:



(a) Trajectory-based (b) State-based

Figure 10.15: Graphical summary of an experiment applying online TD(γ) to a discounted version of the two-state example. It shows variability in RMSE by γ and trace type (accumulating - grey; replacement - white) over 40 replicates of length 5,000.

[39] These values were chosen based on preliminary experimentation.

[40] *Analysis of variance* is a statistical methodology for investigating the individual and simultaneous effect of discrete factors on the mean value of a response variable.

1. The most accurate estimate of $v_\lambda^{d^\infty}(s)$ over all combinations of factors was the combination of logarithmic learning rate, replacement trace, random restart and $\gamma = 0.4$. For this configuration, the average RMSE was 0.96.

2. ANOVA established that the effect of learning rate on RMSE was statistically insignificant but that simulation method, trace type and γ were highly significant.

3. ANOVA also showed that the interaction between γ and trace type was significant, meaning that the effect of trace type differed depending on γ as shown in Figure 10.15. As expected, γ affected results for the accumulating trace and not the replacement trace. This observation was to be expected because in this two-state example, using the replacement trace, especially in the case of trajectory-based data, frequently reset $e(s)$ to 1 so that the effect of γ was minimal. In contrast, when using the accumulating trace, γ impacted the values of $e(s)$ and hence the estimates of $v_\lambda^{d^\infty}(s)$.

4. For trajectory-based data, Figure 10.15a shows that $\gamma = 0$ resulted in the minimal median and least variable RMSE (1.25) over all configurations. Of course, trace type does not impact TD(0) estimates.

5. For state-based data, the results were more complicated. Figure 10.15b shows that the (median) RMSE was increasing in γ for the accumulating trace. For the replacement trace, the median RMSE was minimal at $\gamma = 0.4$ but the differences with respect to the values associated with other γ values were small.

6. Estimates for trajectory-based data were more variable than those for the random restart data. This is to be expected since, when following the Markov chain corresponding to long trajectory state generation, state s_1 is visited twice as frequently as state s_2.

The above results suggest that in this small example, the replacement trace is preferable to the accumulating trace, and that TD(0) provides the best estimates in trajectory-based data. If state-based simulation is used, there is a small benefit to choosing $\gamma > 0$. It is left to the reader to compare this approach to results using Monte Carlo estimation.

Of course, the same conclusions need not hold in general. However, the data generation and analysis of this example suggest an approach to follow in other applications.

10.5 Policy evaluation: Infinite horizon average reward models

This section provides a brief overview of issues arising when applying simulation in the context of infinite horizon average reward models. As a prelude to policy optimization, computing only the average reward (or gain) of a policy does not suffice; an estimate of the bias is also required to apply the Bellman equation.

Most research in simulation-based dynamic programming has focused on infinite horizon episodic and discounted models. This is because the average reward criterion is less sensitive to short-term actions; it depends more on what happens in the distant future. *Therefore, while online learning is fundamental in episodic and discounted models, it is less relevant when using the average reward criterion.* This means that to evaluate policies and find optimal policies for average reward models:

1. Large data sets are required for accurate estimation of the average reward and bias.

2. Offline analyses are practical in average reward settings but online methods exist and still can be used.

This section will describe Monte Carlo and TD(0) approaches for estimating the gain and bias of a stationary policy. Generalization to TD(γ) is left to the reader.

Technical preliminaries

The focus of this section will be on policies with constant gain, therefore, it implicitly assumes a recurrent or unichain model[41]. Recall from Chapter 7 that for a stationary policy d^∞, its average reward (or gain) is defined by

$$g^{d^\infty}(s) = \lim_{N\to\infty} \frac{1}{N} E^{d^\infty}\left[\sum_{n=1}^{N} r(X_n, Y_n, X_{n+1}) \middle| X_1 = s\right] \tag{10.52}$$

and its bias by

$$h^{d^\infty}(s) = E^{d^\infty}\left[\sum_{n=1}^{\infty} \left(r(X_n, Y_n, X_{n+1}) - g^{d^\infty}(X_n)\right) \middle| X_1 = s\right]. \tag{10.53}$$

Moreover, Chapter 7 establishes that in unichain models, the gain and bias of a stationary policy d^∞ satisfy the system of equations

$$h(s) = \sum_{j\in S} p(j|s, d(s)) \left(r(s, d(s), j) + h(j)\right) - g, \quad \forall s \in S. \tag{10.54}$$

[41] Or a regular or unichain policy in a multi-chain model.

Since there are $|S|$ equations in $|S|+1$ unknowns, an extra condition is required to guarantee a unique solution. Solving (10.54) subject to $h(s_0)=0$ for some s_0 provides the *relative values*, denoted $h^{d^\infty}_{\text{rel}}(s)$, which suffice for optimization using the Bellman equation.

When setting $h(s_0)=0$, the equation corresponding to s_0 becomes

$$g=\sum_{j\in S}p(j|s_0,d(s_0))\big(r(s_0,d(s_0),j)+h(j)\big). \tag{10.55}$$

This equation will prove useful when motivating TD(0) below.

10.5.1 Monte Carlo methods

Monte Carlo methods are based on the representations for the gain and bias in (10.52) and (10.53). For the same reasons as in the discounted case, only starting-state Monte Carlo provides estimates with the same degree of accuracy for all states.

Algorithm 10.7. Monte Carlo policy evaluation for an average reward model

1. **Initialize:**
   (a) Specify $d\in D^{\text{MR}}$.
   (b) Specify the number of replicates K_g and K_h, and set $k_g\leftarrow 1$ and $k_h\leftarrow 1$.
   (c) Specify the truncation levels M_g and M_h, and set $m_g\leftarrow 1$ and $m_h\leftarrow 1$.
   (d) Specify $\bar{s}\in S$.
   (e) Create empty lists VALUES_g and VALUES_h.
2. **Estimate the gain:** While $k_g\leq K_g$:
   (a) $m_g\leftarrow 1$ and $u\leftarrow 0$.
   (b) Sample $s\in S$ from a random distribution.
   (c) **Generate a replicate:** While $m_g\leq M_g$:
      i. Sample a from $w_d(\cdot|s)$.
      ii. Simulate (s',r) or sample s' from $p(\cdot|s,a)$ and set $r\leftarrow r(s,a,s')$.
      iii. $u\leftarrow u+r$.
      iv. $s\leftarrow s'$.
      v. $m_g\leftarrow m_g+1$.
   (d) **Update:**

i. Append u/M_g to $\textsc{Values}_g$.
ii. $k_g \leftarrow k_g + 1$.

3. Set $g = \text{mean}(\textsc{Values}_g)$.
4. **Estimate the bias**: While $k_h \leq K_h$:
   (a) $m_h \leftarrow 1$ and $u \leftarrow 0$.
   (b) $s \leftarrow \bar{s}$.
   (c) **Generate a replicate:** While $m_h \leq M_h$:
      i. Sample a from $w_d(\cdot|s)$.
      ii. Simulate (s', r) or sample s' from $p(\cdot|s, a)$ and set $r \leftarrow r(s, a, s')$.
      iii. $u \leftarrow u + (r - g)$.
      iv. $s \leftarrow s'$.
      v. $m_h \leftarrow m_h + 1$.
   (d) **Update:**
      i. Append u to $\textsc{Values}_h$.
      ii. $k_h \leftarrow k_h + 1$.
5. Set $h(\bar{s}) = \text{mean}(\textsc{Values}_h)$.
6. **Terminate:** Return g and $h(\bar{s})$.

Some comments follow:

1. This rather lengthy algorithmic statement provides estimates of g and $h(\bar{s})$ using truncation with possibly different truncation levels and number of replicates for each quantity estimated.
2. Replicates to estimate g can start from the same state or be randomly chosen.
3. It returns estimates of the bias for a single state. To obtain estimates of the bias for all states requires repeating step 4 for all $\bar{s} \in S$.
4. Note that an estimate of the gain is required before estimating the bias.
5. Of course, Monte Carlo is an offline algorithm. It applies to both model-free and model-based settings.

Example 10.4. Average reward Monte Carlo policy evaluation
This example evaluates the gain and bias in the two-state model for the station-

ary policy d^∞ with $d(s_1) = a_{1,2}$ and $d(s_2) = a_{2,2}$. Solving the evaluation equations (10.54) with $h(s_1) = 0$ gives $g^{d^\infty} = 2.857$ and $h^{d^\infty}_{\text{rel}}(s_2) = -2.143$. Alternatively, solving (10.54) subject to $\mathbf{P}^*_d\mathbf{h} = \mathbf{0}$ or equivalently using the matrix representation in Theorem 7.3, establishes that $h^{d^\infty}(s_1) = 1.531$ and $h^{d^\infty}(s_2) = -0.612$.

The gain, bias and relative values were estimated using Algorithm 10.7. Preliminary calculations showed that estimates, especially of the relative value, were extremely variable. Consequently a large number of replicates was required. The following values were used to obtain the estimates in Figure 10.4: $K_g = 850$, $M_g = 850$, $K_h = 10{,}000$ and $M_h = 500$. Results described below were based on 50 instances using the above configuration with different random number seeds for each instance.

Estimates of the gain had mean 2.850 and standard deviation of 0.038 over all instances. Estimates of the bias $h^{d^\infty}(s)$ were extremely variable and unreliable, however estimates of relative values based on $h^{d^\infty}(s_2) - h^{d^\infty}(s_1)$ were more accurate with mean -2.130 and standard deviation 0.204 over all instances. Figure 10.4 displays the variability of the estimates of the gain and relative value across instances.

Because the estimate of g^{d^∞} in step 3 of the algorithm impacts the estimate of the bias in step 4(c)iii, one might hypothesize that the estimates of the gain and relative values were correlated. Surprisingly, this was not the case.

One can conclude that Monte Carlo methods require very large samples to estimate relative values. So, they may not be appropriate for algorithmic optimization.

10.5.2 Temporal differencing (TD(0))

This section provides a temporal differencing method to evaluate the average reward or gain, g^{d^∞}, and relative values $h^{d^\infty}_{\text{rel}}(s)$ of a stationary policy d^∞. For simplicity, they are expressed as g and $h(s)$.

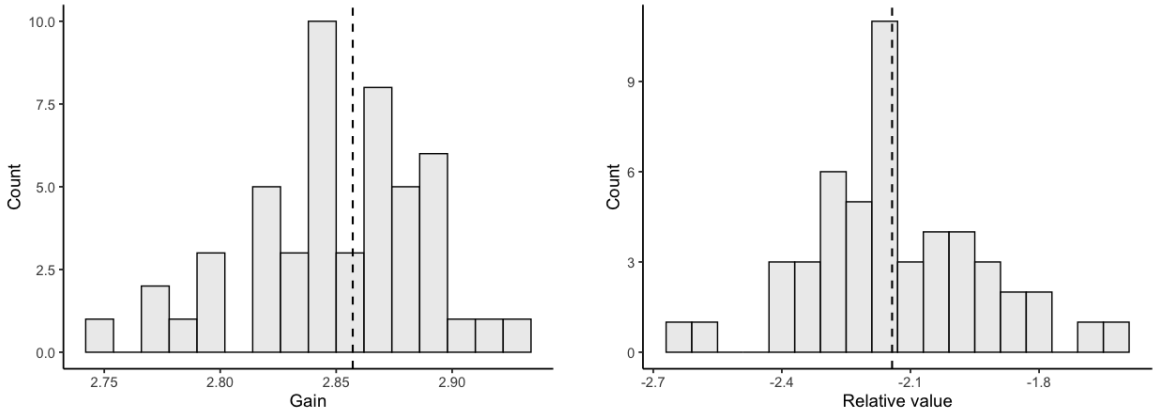
To motivate temporal differencing, express (10.54) in expectation form as

$$h(s) = E^{d^\infty}\left[r(X, Y, X') + h(X') - g \,\middle|\, X = s\right] \tag{10.56}$$

and (10.55) as

$$g = E^{d^\infty}\left[r(X, Y, X') + h(X') \,\middle|\, X = s_0\right]. \tag{10.57}$$

Applying stochastic approximation to these two expressions and replacing g and $h(s)$ by the most recent estimates leads to the following algorithm. Note that it applies to trajectory-based and state-based data.



(a) Estimates of g^{d^∞}.

(b) Estimates of $h_{\text{rel}}^{d^\infty}(s_2)$.

Figure 10.16: Histograms showing variability of Monte Carlo estimates of gain and relative value over 50 instances in Example 10.4. Vertical dashed lines give the true value of the quantities being estimated. Note that the axis scale differs for the two estimates.

Algorithm 10.8. Temporal differencing for policy evaluation for an average reward model

1. **Initialize:**
   (a) Specify $d \in D^{\text{MR}}$ and a distinguished state s_0.
   (b) Specify sequences τ_n and β_n, $n = 1, 2, \ldots$.
   (c) Set $g \leftarrow 0$, $h(s) \leftarrow 0$ and $\text{count}(s) \leftarrow 0$ for all $s \in S$.
   (d) Specify $s \in S$.
   (e) Specify the number of iterations N and set $n \leftarrow 1$.
2. **Iterate:** While $n \leq N$:
   (a) Sample a from $w_d(\cdot|s)$.
   (b) Simulate (s', r) or sample s' from $p(\cdot|s, a)$ and set $r \leftarrow r(s, a, s')$.
   (c) $\text{count}(s) \leftarrow \text{count}(s) + 1$.
   (d) If $s \neq s_0$,

$$h(s) \leftarrow h(s) + \tau_{\text{count}(s)} \left(r(s, d(s), s') - g + h(s') - h(s) \right) \tag{10.58}$$

   and if $s = s_0$,

$$g \leftarrow g + \beta_{\text{count}(s_0)} \left(r(s_0, d(s_0), s') + h(s') - g \right). \tag{10.59}$$

(e) $n \leftarrow n + 1$.

(f) $s \leftarrow s'$ (trajectory-based) or sample s from a specified distribution on S (state-based).

3. **Terminate:** Return g, and $h(s)$ for all $s \in S$.

Some comments follow:

1. The above algorithm uses the relative evaluation equation with $h(s_0) = 0$ in step 2(d). It is based on (10.57). Consequently, the algorithm generates relative values.

2. A variant of the algorithm[42] replaces equation (10.59) with

$$g \leftarrow g + \beta_n(r(s, d(s), s') - g) \tag{10.60}$$

at **every** iteration. Thus, the index for β_n is the iteration number. Note that $\beta_n = n^{-1}$ is equivalent to recursive estimation of the mean of g based on the observed rewards. This is equivalent to online Monte Carlo estimation of the average reward.

3. The algorithm allows for different learning rates in (10.58) and (10.59).

4. Note that in both variants of the algorithm, updates are asynchronous; when using the algorithm with (10.59), g is only updated whenever s_0 is visited and $h(s)$ is only updated when visiting state $s \neq s_0$. When using the algorithm with (10.60), g is updated at every iteration.

5. A TD(γ) variant of this algorithm using update equation (10.60) converges[43] with probability one under the assumption that the Markov chain corresponding to d^∞ is irreducible and aperiodic, and the learning rates satisfy the Robbins-Monro conditions and are linearly related.

The following example investigates the two variants of the algorithm with different learning rates using the frequently analyzed two-state model.

Example 10.5. Temporal differencing in an average reward model. This example applies the above algorithm and its variant to analyze the policy d^∞ where $d(s_1) = a_{1,2}$ and $d(s_2) = a_{2,2}$. The gain and relative values are given in Example 10.4.

It compares the quality of estimates based on Algorithm 10.8 and its variant

[42] This variant has been proposed by Tsitsiklis and Roy (1999).

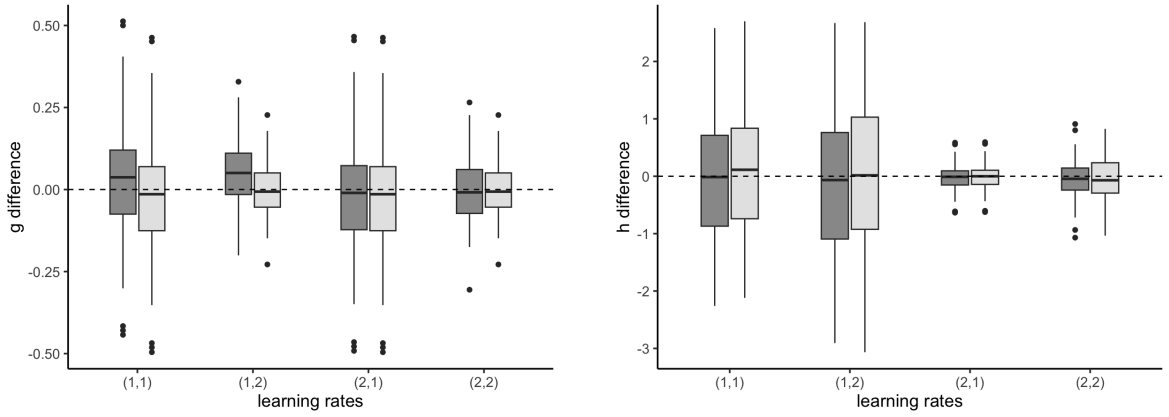
[43] Tsitsiklis and Roy (1999) established this result as well for models with function approximation.

that replaces (10.59) with (10.60) using 50 replicates of $N = 25{,}000$ iterates. After some preliminary calculations, two values for each learning rate were chosen for further investigation: $\tau_n^1 = n^{-1}\log(n+1)$, $\tau_n^2 = 150/(300+n)$ and $\beta_n^1 = n^{-1}\log(n+1)$, $\beta_n^2 = n^{-1}$. Thus, for each random number seed, eight instances were compared: each instance corresponded to a different pair of learning rate and update equation for g. Only the trajectory-based version of the algorithm was investigated.

Estimates are compared on the basis of the difference between the final estimates g and $h(s_2)$ and the true values g^{d^∞} and $h_{\text{rel}}^{d^\infty}(s_2)$. Results are displayed as boxplots in Figure 10.17. The figure shows that when estimating the gain, estimates based on (10.60) were more accurate for all learning rate choices. Clearly, using $\beta_n^2 = n^{-1}$ provided the least variable estimates of the gain.

However, the results for estimating the relative value differed. Using the learning rate τ_n^2 provided less variable estimates than using τ_n^1. Moreover estimates based on the two specifications for the gain differed slightly when using τ_n^2. Note also that the combination of learning rates (τ_n^2, β_n^1) produced the most accurate estimates of the bias.

Overall the estimates of the gain were more accurate than those of the relative value. This is important because the relative value update (10.58) uses the estimate of the gain explicitly. However, accurate estimates of the relative value are required when seeking optimal policies so that precision of the relative values should guide algorithmic and learning rate choice.



(a) $g - g^{d^\infty}$ (b) $h(s_2) - h_{\text{rel}}^{d^\infty}(s_2)$

Figure 10.17: Boxplots comparing accuracy of estimates of the gain and relative values for the eight instances in Example 10.5. Learning rates τ_n^i and β_n^j are represented as (i, j). Dark fill corresponds to g update equation (10.59) and light fill to (10.60). Note that the vertical scale differs in the two plots.

10.6 Policy optimization

This section describes simulation-based methods for finding effective policies. The methods described here either optimize step-by-step, in a similar fashion to value iteration, or combine one of Monte Carlo or temporal differencing with some form of policy improvement. Most are based on state-action value functions, as opposed to value functions. One such approach, Q-learning, represents one of the most significant advances in the development of reinforcement learning methods.

10.6.1 Two perspectives

As noted earlier, simulation-based optimization can be viewed from two perspectives:

1. That of a decision maker or agent interacting with its environment in order to *learn* a good policy through experimentation. Often, the agent may also seek "earning while learning", that is to maximize accumulated rewards during the learning phase.

2. That of an external controller (or analyst) using simulation or real-world experimentation as a tool to learn good policies to be used in subsequent applications.

Of course these are not mutually exclusive; policies identified during learning may be subsequently implemented in the future[44].

10.6.2 Model selection

To assess the quality of estimates of the value of a fixed policy, standard *goodness-of fit* metrics (Section 10.11.1) can be used. When a model is available, these estimates can be compared to the exact policy value function obtained by solving appropriate evaluation equations. Alternatively, comparisons can be based on results of real-time implementations or simulation output.

Since the optimization methods in this chapter are based on state-action value functions, similar RMSE and graphical approaches can be used to assess how well a method estimates the optimal state-action value function. However these should *not* be the primary measure of the quality of a method. Such approaches are limited by two considerations:

[44] In the reinforcement learning literature this is referred to as *planning*.

1. The objective of optimization is to identify high-quality policies. State-action value functions are a construct to do so.

2. The better the method, the more quickly it will identify effective greedy policies. As a result, many state-action pairs will be visited infrequently resulting in inaccurate estimates of state-action value functions for some state-action pairs.

The following *policy-based* considerations should guide method selection:

1. **Policy agreement:** In experiments with known optimal policies, the agreement between these and ones identified by a method can be assessed graphically or through measures of policy closeness.

2. **Long-run policy quality:** The long-run performance of the policies can be compared on the basis of the specified optimality criterion. In a model-based environment this can be based on comparing the exact value of the identified policy computed by solving policy evaluation equations to the optimal value function obtained through algorithms such as policy iteration. In a model-free environment, these can be assessed against simulations using good heuristics.

3. **Policy variability:** Methods that produce similar policies across multiple replications are preferable.

4. **Accumulated reward:** In learning environments, the agent may seek to maximize the accumulated reward during the learning phase of an experiment. Methods that achieve higher cumulative rewards during training are preferable.

5. **Data efficiency:** Methods that require less data to obtain high quality policies are preferable.

6. **Robustness:** Methods that identify high quality policies across different applications and parameter settings are preferable.

Definitive statements require appropriate metrics and carefully designed experiments spanning diverse environments[45].

10.6.3 Preliminaries

Given an estimate of the value function $v(\cdot)$, iterative optimization algorithms in Chapters 5-7 are based on evaluating expressions of the form

$$\max_{a \in A_s} \left(\sum_{j \in S} p(j|s,a)(r(s,a,j) + \lambda v(j)) \right),$$

[45] Patterson et al. 2024 provides an in-depth discussion of issues to consider when evaluating a reinforcement learning method.

where $0 < \lambda \leq 1$ and choosing actions *greedily*, that is, those in

$$\arg\max_{a \in A_s} \left(\sum_{j \in S} p(j|s,a)(r(s,a,j) + \lambda v(j)) \right).$$

Actions chosen in this way are referred to as *greedy*[46] actions. Instead of focusing on value functions, the methods in this section focus on state-action value functions

$$q(s,a) := \sum_{j \in S} p(j|s,a)(r(s,a,j) + \lambda v(j))$$

with $\lambda = 1$ in episodic and average reward[47] models and $0 \leq \lambda < 1$ in discounted models. In particular, they seek to find the optimal state-action value function

$$q^*(s,a) := \sum_{j \in S} p(j|s,a)(r(s,a,j) + \lambda v^*(j)), \tag{10.61}$$

where $v^*(\cdot)$ is the optimal value function for the corresponding MDP. If this quantity were available, optimal policies can be found by selecting actions greedily using $q^*(s,a)$.

The elegance of these methods is that they regard $q(s,a)$ as the primitive. Implementation does not require knowledge of the underlying model. All that is needed are:

1. methods to generate actions in states,
2. methods to generate rewards and transitions to new states, and
3. methods to recursively update state-action value functions.

Recall that $q^*(s,a)$ solves the Bellman equation

$$q(s,a) = \sum_{j \in S} p(j|s,a)\big(r(s,a,j) + \lambda \max_{a' \in A_j} q(j,a')\big) \tag{10.62}$$

with $\lambda = 1$ in episodic models and $0 \leq \lambda < 1$ in discounted models. The solution is unique in transient and stochastic shortest path episodic models and discounted models (Theorems 6.6 and 5.7).

[46] They are also referred to as **v**-greedy actions below.

[47] In average reward models, the bias or relative value functions replace $v(s)$.

10.6.4 Selecting actions: Exploration vs. exploitation

To find optimal policies using simulation or real-time experimentation, **all** state-action pairs must be evaluated sufficiently often to obtain accurate estimates of state-action value functions $q(s,a)$. Doing this is referred to as *exploration* to distinguish it from *exploitation*, which only implements greedy actions.

When seeking a good policy, it is desirable to:

1. begin by exploring broadly and frequently,
2. then narrow exploration by searching among actions with large $q(s,a)$ values, and
3. finally, implement greedy actions frequently.

Two approaches to achieve these objectives are ϵ-greedy sampling and softmax sampling. Both use randomized decision rules to sample actions.

ϵ-greedy sampling

This approach selects[48] an action a' in state $s \in S$ according to

$$a' = \begin{cases} a^* \in \arg\max_{a \in A_s} q(s,a) & \text{with probability } 1-\epsilon \\ a \in A_s \setminus \{a^*\} & \text{with probability } \epsilon/(|A_s|-1). \end{cases} \tag{10.63}$$

Alternatively,

$$a' = \begin{cases} a^* \in \arg\max_{a \in A_s} q(s,a) & \text{with probability } 1-\epsilon+\epsilon/|A_s| \\ a \in A_s \setminus \{a^*\} & \text{with probability } \epsilon/|A_s|. \end{cases}$$

When A_s contains only a few elements, such as in the examples below, the former specification enables more exploration. To see this, suppose A_s contains three elements and $\epsilon = 0.3$. The first specification chooses the greedy action with probability 0.7 and each other action with probability 0.15, while the second specification chooses the greedy action with probability 0.8 and each other action with probability 0.1. In models with large action sets, such a distinction is irrelevant.

From another perspective, ϵ-greedy action selection corresponds to implementing the *randomized* decision rule $d(s)$ defined by

$$w_d(a|s) := \begin{cases} 1-\epsilon & a \in \arg\max_{a' \in A_s} q(s,a') \\ \epsilon/(|A_s|-1) & a \neq \arg\max_{a' \in A_s} q(s,a'). \end{cases} \tag{10.64}$$

[48] This specification needs some modification if the arg max is not unique. For example if there are two elements attaining the arg max select each with probability $(1-\epsilon)/2$ when using (10.63).

Softmax (Boltzmann) sampling

Another approach to exploration is to choose action a in state s with probability determined by the *softmax* function:

$$\frac{e^{\eta q(s,a)}}{\sum_{a' \in A_s} e^{\eta q(s,a')}}. \tag{10.65}$$

The quantity η is a parameter that affects action choice probabilities and also helps avoid numerical instability[49]. Sometimes $1/\eta$ is referred to as the *temperature parameter*. When η is small, the differences in $q(s,a)$ have less effect on action choice probabilities than when η is large. For small η, softmax sampling selects actions with nearly equal probabilities. Conversely, when η is large, the softmax function makes it more likely to select actions with large $q(s,a)$ values.

As above, softmax sampling may be viewed as implementing a *randomized* decision rule $d(s)$ defined by

$$w_d(a|s) := \frac{e^{\eta q(s,a)}}{\sum_{a' \in A_s} e^{\eta q(s,a')}}. \tag{10.66}$$

Note that viewing softmax sampling from the perspective of a randomized decision rule is fundamental to the policy gradient and actor-critic methods in Chapter 11.

Discussion

Both methods depend on parameters that trade-off exploration and exploitation. Clearly these parameters should be varied through the learning (or computational) process to impose early exploration and late exploitation. This means that ϵ should decrease and η increase with respect to either the iteration number or the count of the number of visits to a state. Therefore algorithms are usually specified in terms of sequences of ϵ_n or η_n, $n = 1, 2, \ldots$.

The main difference between these methods is that for each $s \in S$, softmax sampling assigns probabilities to actions based on the relative values of $q(s,a)$, ensuring that all actions have a non-zero probability of being selected but favoring those with the greatest values of $q(s,a)$. In contrast, ϵ-greedy sampling selects greedy actions with probability $1-\epsilon$ and does not distinguish between non-greedy actions; all have a small and equal probability of being selected regardless of their $q(s,a)$ values.

The use of either of these methods with appropriate parameter choices ensures that all state-action pairs are visited infinitely often, which is usually sufficient for theoretical convergence.

[49] We have frequently encountered divergence of parameter estimates in our calculations.

10.7 Q-learning and SARSA

How would one go about using state-action value functions to learn good policies? An obvious iterative approach would be to start with a guess for $q(s,a)$, select a state s, use softmax or ϵ-greedy sampling to select an action a, implement it and observe a reward r and a transition to a subsequent state s'.

Q-learning and SARSA follow this general approach, but differ in how they use this information to update $q(s,a)$. Each may use exploration to generate an action a' in state s', but Q-learning updates $q(s,a)$ according to

$$q(s,a) \leftarrow q(s,a) + \tau \left(r + \lambda \max_{a' \in A_{s'}} q(s',a') - q(s,a) \right) \tag{10.67}$$

while SARSA updates $q(s,a)$ according to

$$q(s,a) \leftarrow q(s,a) + \tau \big(r + \lambda q(s',a') - q(s,a) \big), \tag{10.68}$$

where a' denotes an action selected by exploration in the subsequent state. The acronym SARSA stands for "state-action-reward-state-action", corresponding to the sequence of objects (s,a,r,s',a') involved in the update in (10.68) [50].

Another way of thinking of this is through the concept of a *learning* or *behavioral* policy.

Definition 10.2. A *learning policy* or *behavioral policy* is the policy used to generate rewards in a simulation or real-world implementation.

Suppose both Q-learning and SARSA use the same behavioral policy to generate action a'. SARSA corresponds to an on-policy update of $q(s,a)$ in agreement with the behavioral policy, while the off-policy update using Q-learning may differ[51] as shown in Figure 10.18.

More formally, Q-learning updates estimates of state-action value functions using a temporal difference relationship of the form

$$q(s,a) \leftarrow q(s,a) + \tau \left(r(s,a,s') + \lambda \max_{a' \in A_s} q(s',a') - q(s,a) \right) \tag{10.69}$$

where $0 \leq \tau \leq 1$. This recursion is based on using a variant[52] of gradient descent to

[50] Note that when the reward depends on the subsequent state, the unappealing acronym SASRA would be more appropriate.

[51] They will agree when $a' \in \arg\max_{a \in A_{s'}} q(s',a)$.

[52] This variant, based on a construct referred to as a *semi-gradient*, is discussed in Section 11.2.2 in the next chapter.

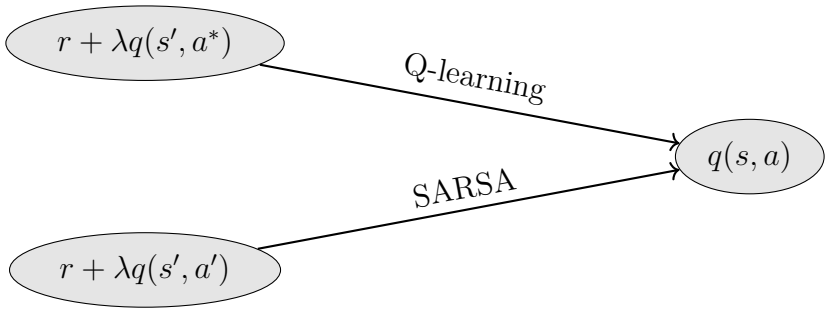


Figure 10.18: Comparison of quantity used to update $q(s, a)$ using SARSA and Q-learning. These updates differ when the action chosen by the greedy policy a^* differs from the action chosen by the learning policy a'.

minimize the expected squared error between the two sides of the expression

$$q(s,a) = E\left[r(X,Y,X') + \lambda \max_{a' \in A_{X'}} q(X',a') \,\middle|\, X = s, Y = a\right], \tag{10.70}$$

which is the Bellman equation (10.62) written in expectation form. Note that the expectation is with respect to the transition probability $p(j|s,a)$ that specifies the distribution of X'. Note that in episodic and average reward models, $\lambda = 1$. In the latter case, (10.69) would also contain a $-g$ term to correspond to the appropriate recursion for the gain and bias.

SARSA

SARSA updates estimates of state-action value functions according to:

$$q(s,a) \leftarrow q(s,a) + \tau\big(r(s,a,s') + \lambda q(s',a') - q(s,a)\big), \tag{10.71}$$

where a' in state s' is chosen based on the randomized policy that was used to choose a in state s. The justification for SARSA is more subtle than Q-learning because after generating s' from $p(\cdot|s,a)$, it requires a distribution for choosing a' at the subsequent decision epoch.

Given a randomized decision rule d, SARSA is based on replacing (10.70) by

$$q(s,a) = E^d\left[r(X,Y,X') + \lambda q(X',Y') \,\middle|\, X = s, Y = a\right], \tag{10.72}$$

where X' has distribution $p(j|s,a)$ and Y' has conditional distribution $w_d(\cdot|X')$ corresponding to the exploration method.

Thus, in essence, SARSA is applying a variant of gradient descent to estimate the state-action value function corresponding to the behavioral policy d^∞. This would

be appropriate if the objective was to estimate $q^{d^\infty}(s,a)$, but the objective is to find the **optimal** state-action value function. Therefore, the decision d must evolve over time so that it approaches the decision rule corresponding to an optimal policy. This is typically achieved using ϵ-greedy or softmax action selection with parameters that ensure that eventually only greedy actions are selected. This requirement is described in more detail following the description of SARSA in algorithmic form.

10.7.1 Episodic models

Q-learning and SARSA algorithms for episodic models are presented, discussed and compared below. Recall that in an episodic model, an episode terminates when a state in Δ is reached.

Q-learning

The following online algorithm can be implemented in a model-free or model-based environment.

Algorithm 10.9. Q-learning for an episodic model

1. **Initialize:**
   (a) Set $q(s,a) \leftarrow 0$ and $\text{count}(s,a) \leftarrow 0$ for each $a \in A_s$ and $s \in S$.
   (b) Specify the learning rate $\tau_n, n = 1, 2, \ldots$.
   (c) Specify $\bar{s} \in S$ and $\bar{a} \in A_{\bar{s}}$.
   (d) Specify the number of episodes K.
   (e) Specify a sequence $\epsilon_n, n = 1, 2, \ldots$ (or η_n for softmax sampling).
   (f) $k \leftarrow 1$.
2. **Iterate:** While $k \leq K$:
   (a) $s \leftarrow \bar{s}$ and $a \leftarrow \bar{a}$.
   (b) **Generate episode:** While $s \notin \Delta$:
      i. $\text{count}(s,a) \leftarrow \text{count}(s,a) + 1$.
      ii. Simulate (s', r) or sample s' from $p(\cdot|s,a)$ and set $r \leftarrow r(s,a,s')$.
      iii. **Update** $q(s,a)$**:**

$$q(s,a) \leftarrow q(s,a) + \tau_{\text{count}(s,a)} \left(r + \max_{a' \in A_{s'}} q(s',a') - q(s,a) \right). \tag{10.73}$$

iv. Choose $a \in A_{s'}$ using ϵ-greedy (or softmax) sampling.

v. $s \leftarrow s'$.

(c) $k \leftarrow k + 1$.

3. **Terminate:** Return $q(s, a)$ for all $a \in A_s$ and $s \in S$.

Some comments follow:

1. Note that in (10.73), r depends only on the *current* action a while $\max_{a' \in A_{s'}} q(s', a')$ accounts for action choice at the *next* decision epoch. The consequence of this is that updates of $q(s, a)$ use the value associated with a greedy action although the action generated by the learning policy in step 2(b)iv to generate rewards and transition probabilities at the next decision epoch need not be the greedy action. Because the learning policy may differ from the policy used to compute the temporal difference in (10.73), Q-learning is said to be an *off-policy algorithm*.
2. The index for ϵ or η is best set to $\sum_{a \in A_s} \text{count}(s, a)$, corresponding to the number of times the state was previously visited.
3. The nature of the application determines whether it is most prudent to start each episode in the same state (as the algorithm is currently written) or in a random state.
4. Using a similar approach to TD(γ), past state-action values can be updated at each iteration.
5. Under the assumption that each state-action pair is visited infinitely often and the learning rate satisfies the Robbins-Monro conditions, the state-action value function estimates converge with probability one to the optimal state-action value function and hence can be used to identify an optimal policy in the limit.

SARSA

SARSA provides the following *on-policy* variant to Q-learning.

Algorithm 10.10. SARSA for an episodic model

1. **Initialize:** Same as in Algorithm 10.9.
2. **Iterate:** While $k \leq K$:
   (a) $s \leftarrow \bar{s}$ and $a \leftarrow \bar{a}$.
   (b) **Generate episode:** While $s \notin \Delta$:
      i. $\text{count}(s, a) \leftarrow \text{count}(s, a) + 1$.

ii. Simulate (s', r) or sample s' from $p(\cdot|s, a)$ and set $r \leftarrow r(s, a, s')$.

iii. Choose $a' \in A_{s'}$ using ϵ-greedy (or softmax) sampling.

iv. **Update** $q(s, a)$:

$$q(s, a) \leftarrow q(s, a) + \tau_{\text{count}(s,a)} \left(r + q(s', a') - q(s, a)\right). \quad (10.74)$$

v. $s \leftarrow s'$, $a \leftarrow a'$.

(c) $k \leftarrow k + 1$.

3. **Terminate:** Return $q(s, a)$ for all $a \in A_s$ and $s \in S$.

Note that both SARSA and Q-learning use ϵ-greedy or softmax action selection but SARSA uses the action to *both* update the state-action value function and implement at the subsequent iteration. On the other hand, Q-learning uses ϵ-greedy or softmax action selection *only* to select the next action. Thus, the updates of $q(s, a)$ using SARSA correspond to the "implemented" action or the behavioral policy, while the updates using Q-learning correspond to the greedy action, which may differ from the implemented action (see Figure 10.18).

Without further conditions, *SARSA may converge to a sub-optimal policy.* For example, when the exploration rate does not decrease to zero, the state-action value functions may correspond to a non-optimal randomized policy. In order for SARSA to converge to the optimal state-action value function and identify an optimal policy requires, the learning policy must

1. sample each state-action pair infinitely often, and
2. select greedy actions in the limit with probability one.

Such learning policies are referred to as *greedy in the limit with infinite exploration*[53] . Thus, if the learning policy is greedy in the limit with infinite exploration, SARSA will converge to the optimal state-action value function under the Robbins-Monro step-size conditions.

The on-policy versus off-policy dichotomy

Why should one care about this distinction? Analysis of a **deterministic** version of the Gridworld example in Figure 10.6 provides some insight.

Example 10.6. Distinguishing on-policy and off-policy updates
Suppose in a deterministic model the coffee-delivering robot occupies cell 11

[53] Singh et al. [2000] introduces this concept and provides technical conditions that ensure convergence of SARSA.

(see Figure 10.6), plans to move upward (denoted action 1) to cell 8 and the learning policy based on ϵ-greedy or softmax sampling in cell 8 selects the action that moves it to the left (denoted action 2). As a result, the robot falls down the stairs and incurs a penalty of 200. If this happens, the SARSA and Q-learning updates of $q(11, 1)$ differ.

The SARSA update is

$$q(11, 1) \leftarrow q(11, 1) + \tau\big(-1 + q(8, 2) - q(11, 1)\big)$$

while the Q-learning update is

$$q(11, 1) \leftarrow q(11, 1) + \tau\left(-1 + \max_{a' \in A_8} q(8, a') - q(11, 1)\right).$$

Choosing $q(s, a)$ equal to the optimal value (in the deterministic problem) sets $q(11, 1) = 46$, $q(8, 2) = -201$ and $q(8, 1) = 47$. Thus, the SARSA update becomes

$$q(11, 1) \leftarrow 46 + \tau(-1 - 201 - 46) = 46 - 248\tau$$

while the Q-learning update becomes

$$q(11, 1) \leftarrow 46 + \tau(-1 + 47 - 46) = 46.$$

Thus in future episodes, under SARSA, it would be less likely for the robot to choose the action "move up' in cell 11 and instead move to the right to cell 12, while under Q-learning the robot would move to cell 8 risking the likelihood of falling down the stairs.

This example shows that because of the presence of action sampling, the SARSA guided robot will take more conservative actions than the Q-learning guided robot. However, Q-learning preserves the optimal state-action value function while SARSA does not.

What this example shows is that when using exploration with either ϵ-greedy or softmax sampling:

Use Q-learning if your objective is to find an optimal policy for future use, while SARSA might be preferable when reward acquisition during the learning phase is of concern.

Computational examples

This section compares Q-learning and SARSA using two versions of the Gridworld model.

Example 10.7. Model-free deterministic Gridworld

In this version of the Gridworld model, the robot can select any of the actions "up", "right", and "left" in any cell. Action choice results in a deterministic transition in the intended direction when possible. Otherwise the robot remains in the current cell. For example, if the robot is in cell 13 and tries to move left, it will remain in cell 13. Each time the robot chooses an action it incurs a cost of 1, even if it cannot move.

This example applies Q-learning and SARSA with $q(s, a)$ initialized at 0 and with initial state designated as cell 13. In contrast to earlier Gridworld examples, an episode ends when the robot reaches cell 1. Thus, an episode may consist of instances when the robot falls down the stairs (cell 7) and incurs a penalty. In such instances, the robot returns to cell 13, refills the coffee cup and then attempts to deliver the coffee.

Exploration introduces randomness. The results below apply softmax sampling with $\eta_n = 0.04$ and updates $q(s, a)$ using a learning rate of $\tau_n = 0.8n^{-0.75}$ where n denotes the number of visits to pair (s, a). These parameter choices were based on considerable experimentation. Both Q-learning and SARSA used $K = 500$ episodes and used the same random number seed for each episode.

Using Q-learning, the estimated state-action value function obtained from a typical replicate of 500 episodes appears in Table 10.4. The entries labeled "n/a" correspond to the termination cells 1 and 7, where no action choice is available. Starting from cell 13, the greedy policy's path, represented by the bold entries in the table, is $13 \rightarrow 10 \rightarrow 11 \rightarrow 8 \rightarrow 5 \rightarrow 2 \rightarrow 1$. Note that $q(13, \text{"up"}) = 42.89$, while the optimal deterministic policy total reward starting in 13 and following the shortest path generates a reward of 44. The discrepancy is a result of exploration and the small number of episodes. When K is increased to 5,000, $q(13, \text{"up"}) = 44$ and there also were several optimal policies.

SARSA with softmax sampling and $K = 500$ identified the same optimal policy but the $q(s, a)$ values were considerably smaller than those obtained using Q-learning. Moreover "optimal" action choice differed in cell 15. In cell 15 SARSA chose the more conservative action "up" and Q-learning chose "left".

Figure 10.19 shows the sequence of running-average rewards acquired through the 500 episodes using Q-learning and SARSA. Observe that in this instance the running-average reward during learning using SARSA exceeded that of Q-learning. The mean (standard deviation) of the total reward per episode for Q-learning was 31.36 (14.94) and for SARSA was 34.58 (16.92). This pattern persisted across all experiments.

As a result of the above computational results, the robot can abandon exploration, encode the optimal policy and obtain the maximum reward.

The results in Example 10.7 are quite remarkable. *With no intervention, and no knowledge of the world, the robot identified the optimal policy and accurately estimated*

state	up	right	left
1	n/a	n/a	n/a
2	48.00	46.99	**49.00**
3	47.00	47.00	48.00
4	49.00	46.87	47.96
5	**47.99**	45.81	47.97
6	46.98	45.86	46.89
7	n/a	n/a	n/a
8	**46.94**	44.41	-160.8
9	45.81	42.25	42.86
10	-160.8	**44.52**	43.14
11	**45.78**	42.25	42.86
12	44.22	41.49	44.20
13	**42.89**	42.33	40.52
14	44.17	37.08	39.95
15	39.40	27.90	41.41

Table 10.4: Table of estimates of $q^*(s,a)$ using Algorithm 10.9 in a single experiment consisting of 500 episodes using Q-learning.

the optimal state-action value function.

The following example considers the Gridworld model with noisy transitions.

Example 10.8. Comparison of Q-learning and SARSA in Gridworld

Using the Gridworld model with $p = 0.8$, this example investigates the effect of learning rate and algorithm (Q-learning versus SARSA) on:

1. The extent of agreement between the greedy policy based on the estimated state-action value function and the optimal policy,
2. the difference between the value function of the greedy policy and that of the optimal policy, and
3. the RMSE of the estimated state-action value function.

Since the model is available it can be solved to find the optimal policy and the optimal value function. If it were not, Monte Carlo policy evaluation could be used.

Preliminary calculations investigating exploration method, learning rate and number of episodes suggested the following choices:

1. **Exploration:** ϵ-greedy sampling with

$$\epsilon_n = \frac{200}{500+n}, \tag{10.75}$$

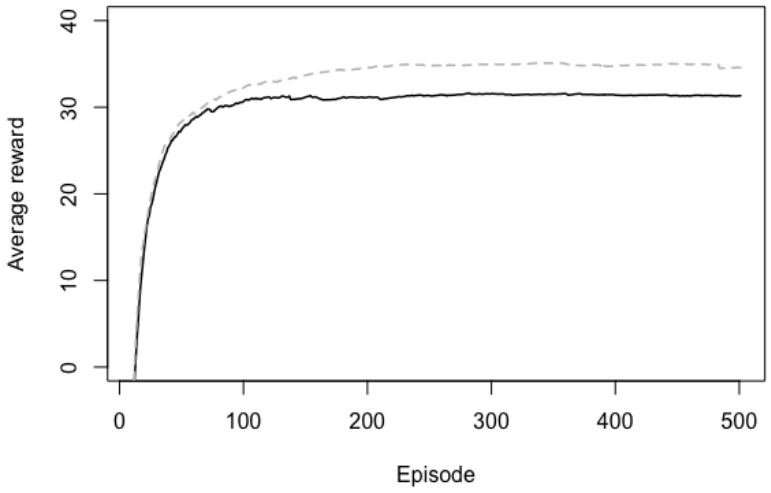


Figure 10.19: Running-average reward using SARSA (dashed line) and Q-learning (solid line) based on a single experiment with 500 episodes.

where n denotes the episode number, produced the most accurate estimates of $q(s, a)$.

2. **Learning rates:**

   Exp1: $\tau_n = 0.4n^{-0.5}$

   Exp2: $\tau_n = 0.8^{-0.75}$

   STC: $\tau_n = 0.1/(1 + 10^{-5}n^2)$

   Log: $\tau_n = n^{-1}\log(n + 1)$

   where the learning rate index n represented the number of times a state-action pair was previously visited.

3. **Replicates:** 40 starting in cell 13 using common random number seeds for each combination of factors in each replicate.

4. **Number of episodes:** 5,000

Table 10.5 and Figure 10.20 summarize the results. Observe that:

1. **Policy optimality:** The greedy policy derived from the final estimate of $q(s, a)$ agreed with the optimal policy in at most 21 of the 40 replications (SARSA with learning rate Exp2). Averaged over the four learning rates,

the policy identified by SARSA was more likely to be optimal than that identified by Q-learning.

2. **Features of non-optimal policies:** When greedy policies were non-optimal, they differed from the optimal policy in at most 2 states (cells 4 and cell 8) for all but one instance[a]. These states were visited infrequently under the optimal policy, which steered the robot towards the right wall before reaching the stairs. This could be remedied by additional sampling of infrequently sampled state-action pairs.

3. **Impact of non-optimal actions:** Table 10.6 shows the effect of incorrect action choice on policy value functions found using the exact methods in Chapter 6. Observe that having one or two non-optimal actions had little effect on the estimate for replications starting in cell 13.

4. **State-action value function estimation:** For all learning rates, Q-learning estimates of $q(s, a)$ had smaller median and mean RMSEs[b] than those obtained with SARSA. The effect of learning rate on the accuracy of the estimate of the optimal state-action value function was not consistent for the two methods.

5. **Value vs. policy** The learning rates that most accurately estimated the optimal state-action value function differed from those that were more likely to identify the optimal policy.

The take away from this example is that, in most cases, greedy policies obtained from estimated state-action value functions identified policies with values that differed little from those of the optimal policy. On the other hand, accuracy of the estimates of state-action value functions varied considerably. Increased sampling of infrequently visited state-action pairs may improve the quality of estimates and perhaps policies.

[a] In this instance, the greedy policy differed from the optimal policy in cells 4, 8 and 14. Cells 4 and 8 were sampled infrequently and in cell 14 there was only a small difference in estimated state-action values.

[b] The RMSE compared the estimated state-action value function to the optimal state-action value function obtained using exact methods.

10.7.2 Infinite horizon discounted models

This section describes and examines the use of Q-learning and SARSA in infinite horizon discounted models assuming trajectory-based data.

Non-optimal actions	Q-learning				SARSA			
	Exp1	Exp2	STC	Log	Exp1	Exp2	STC	Log
0	14	7	14	20	20	21	17	16
1	20	26	20	17	16	15	18	21
2	6	6	6	3	4	4	5	3
3	0	1	0	0	0	0	0	0

Table 10.5: Comparison of update method and learning rate on the basis of the number of non-optimal actions determined by the greedy policy.

Non-optimal action(s)	$v^*(13) - v(13)$	Median difference
4	0.12	0.12
8	0.35	0.30
4,8	0.44	0.41
4,8,14	2.22	0.54

Table 10.6: Effect of non-optimal actions on the deviation of the optimal value function from that of policies with indicated non-optimal actions. The last column indicates the median difference over all states. As a point of reference $v^*(13) = 29.22$ and the median of $v^*(s)$ over all s was 35.29.

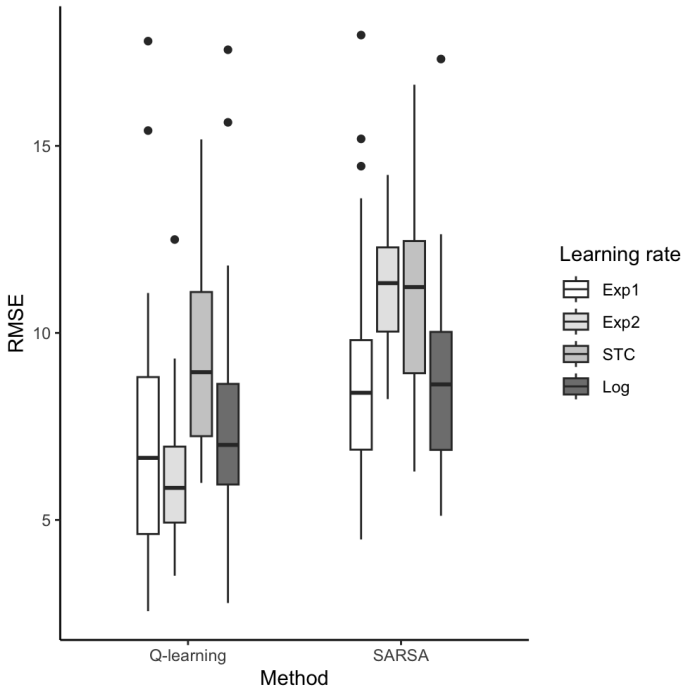


Figure 10.20: Boxplots of RMSE by learning rate and method for Example 10.8.

Algorithm 10.11. Q-learning for a discounted model

1. **Initialize:**
   (a) Set $q(s,a) \leftarrow 0$ and $\text{count}(s,a) \leftarrow 0$ for each $a \in A_s$ and $s \in S$.
   (b) Specify the learning rate $\tau_n, n = 1, 2, \ldots$.
   (c) Specify the number of iterations N.
   (d) Specify a sequence $\epsilon_n, n = 1, 2, \ldots$ (or η_n for softmax sampling).
   (e) Specify $s \in S$ and $a \in A_s$.
   (f) $n \leftarrow 1$.
2. **Iterate:** While $n \leq N$:
   (a) $\text{count}(s,a) \leftarrow \text{count}(s,a) + 1$.
   (b) Simulate (s', r) or sample s' from $p(\cdot|s,a)$ and set $r \leftarrow r(s,a,s')$.
   (c) **Update** $q(s,a)$**:**

$$q(s,a) \leftarrow q(s,a) + \tau_{\text{count}(s,a)} \left(r + \lambda \max_{a' \in A_s} q(s',a') - q(s,a) \right). \quad (10.76)$$

   (d) Choose $a \in A_{s'}$ using ϵ-greedy (or softmax) sampling.
   (e) $s \leftarrow s'$.
   (f) $n \leftarrow n + 1$.
3. **Terminate:** Return $q(s,a)$ for all $a \in A_s$ and $s \in S$.

Some comments follow:

1. The above algorithm is trajectory-based. It can be modified to be state-action pair-based by randomly sampling states and actions prior to generating s' and r in step 2(b) and removing step 2(e). Such a modification can be implemented in a simulator and may apply in some real systems.
2. As noted in the episodic case, SARSA replaces steps 2(c)-2(e) with:
   (c) Sample $a' \in A_{s'}$
   (d) Update $q(s,a)$ by

$$q(s,a) \leftarrow q(s,a) + \tau_{\text{count}(s,a)} \left(r + \lambda q(s',a') - q(s,a) \right). \quad (10.77)$$

   (e) $s \leftarrow s'$, $a \leftarrow a'$.

The difference is that it uses action a' to *both* update $q(s,a)$ and to generate a state and reward at the next iteration.

3. The algorithm returns a state-action value function that can be used to identify a greedy decision rule $d^*(s)$ and policy $(d^*)^\infty$. The value of this policy can be approximated by $q(s,d^*(s))$, evaluated exactly by solving the policy evaluation equations if a model is available, or by simulation or real-time implementation otherwise.

4. If every state-action pair is visited infinitely often and the learning rates satisfy the Robbins-Monro conditions, then the iterates of Q-learning converge with probability one to the optimal state-action value function.

Example of Q-learning and SARSA in a discounted model

The following example illustrates the challenges of selecting parameters for Q-learning and SARSA by investigating the impact of exploration and learning rate on policy choice and state-action value function estimation.

It applies Algorithm 10.11 and its SARSA variant to find estimates of $q^*(s,a)$ in a discounted ($\lambda = 0.9$) version of the two-state model from Section 2.5. Although this example has only four stationary policies, it provides insights that can be applied in more complex settings such as the example in Section 10.9.

Results below are based on setting $N = 30{,}000$ and using trajectory-based data. They compare learning rates $\tau_n = 0.5n^{-0.6}$ (Exp), $\tau_n = 150/(300+n)$ (Ratio), $\tau_n = 0.1(1+n^2/10^5)^{-1}$ (STC) and $\tau_n = \log(n+1)/n$ (Log), and exploration approaches ϵ-greedy with step function decay $\epsilon_n = 0.01 + 0.34I_{[1,\,5{,}000]}(n) + 0.09I_{[0,\,10{,}000]}(n)$ (Step), ϵ-greedy with ratio decay $\epsilon_n = 150/(500+n)$ (Ratio) and with $\eta_n = 0.01n^{0.5}$ (Softmax).

For the learning rate, n equals the number of times a state-action pair was selected. For action choice, n was set to number of times a state was visited. This was done to account for the asynchronous evaluation of state-action pairs. Comparison to setting n equal to the iteration number is left as an exercise.

The simulation study compared all 24 combinations of learning rate, exploration method and recursion (SARSA vs. Q-learning) using 40 replicates with common random number seed for each. Results are summarized in terms of:

1. the fraction of instances in which the greedy policy and the optimal policy were identical, and

2. the RMSE of the estimated matrix of state-action values compared to the exact optimal state-action value function represented by the matrix $\mathbf{Q}^*$.

Solving for $v_\lambda^{d^\infty}(s)$ and applying (10.61) yields

$$\mathbf{Q}^* = \begin{bmatrix} 29.74 & \mathbf{30.15} \\ 20.15 & \mathbf{27.94} \end{bmatrix},$$

where the entries in bold represent the maximum of the row. Thus the optimal policy $(d^*)^\infty$, corresponds to $d^*(s_1) = a_{1,2}$ and $d^*(s_2) = a_{2,2}$, with value $\mathbf{v}^*_\lambda = (30.15, 27.94)$. Recall that the next best policy with $d(s_1) = a_{1,1}$ and $d(s_2) = a_{2,2}$ had value $(27.18, 25.62)$.

		Q-learning			SARSA		
Exploration		Step	Ratio	Softmax	Step	Ratio	Softmax
Learning Rate	Exp	67	74	56	79	18	85
	Ratio	62	74	67	28	51	**97**
	STC	**92**	90	62	0	46	46
	Log	67	90	72	79	38	92

Table 10.7: Percentage of replicates in which the greedy policy agreed with the optimal policy.

Table 10.7 shows the percentage of replicates in which the optimal policy was identified by greedy action choice using the estimated state-action value function. Observe that:

1. There was considerable variability in accuracy.
2. **Main effects:** Averaged over all other factors, Q-learning chose the optimal policy more frequently than SARSA (73% vs. 55%), softmax exploration chose the optimal policy most frequently (72%) among exploration methods, and the Log learning rate chose the optimal policy most frequently (73%) among learning rates.
3. **Q-learning:** STC with step function and ratio exploration chose the optimal policy most frequently. Softmax exploration was the least accurate.
4. **SARSA:** Softmax exploration was most accurate, especially in the case of Ratio and Log learning rates.

Figure 10.21 provides boxplots of the RMSE as it varied across 40 replicates for each combination of learning rate, exploration method and recursion.

They show that:

1. Q-learning estimates were considerably more accurate than SARSA estimates.
2. Q-learning estimates were less sensitive to exploration method than SARSA estimates. This is to be expected since, as noted above, SARSA evaluates the learning policy implied by the exploration method.
3. For Q-learning, the Log learning rate produced the most accurate estimates. In this case, estimates based on exploration with the step function or ratio estimate produced the most accurate and least variable estimates, respectively.

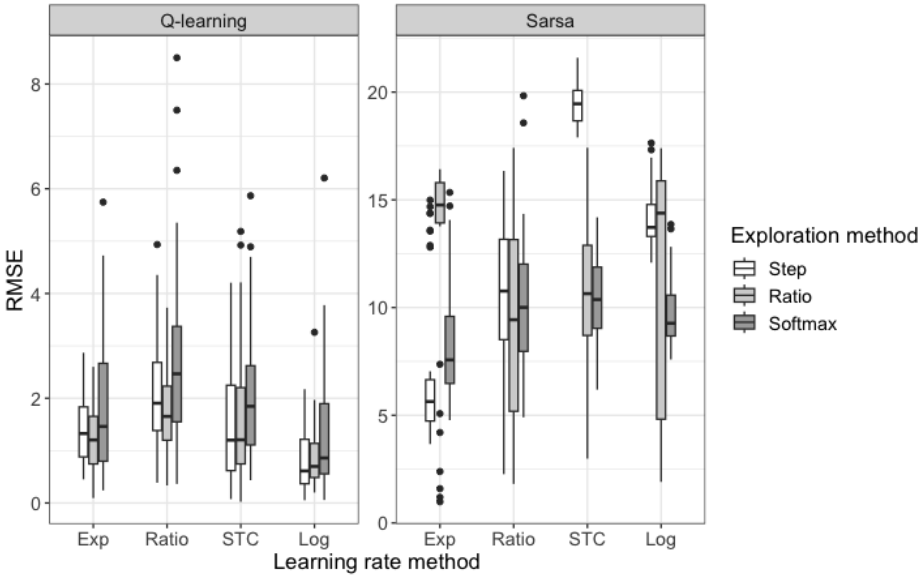


Figure 10.21: Boxplot of RMSE based on 40 simulated replicates. Note that the vertical scales on the left and right plots differ.

4. For SARSA, the exponential decay learning rate with step function exploration produced the most accurate estimates. The estimates using a ratio learning rate were highly variable.

These results show that the most accurate estimates in terms of RMSE were not always in agreement with the methods that chose the best policy. Since SARSA methods were less accurate in estimating the q-function, softmax may be able to account for this by using the relative values of the q-function as its basis for action choice.

The following state-action value matrices, generated using a specific random number seed, offer deeper insight into the effects of parameters and method on estimates. The matrix labels use subscripts to denote the method (Q-learning or SARSA), learning rate, and exploration technique. The bold entry within each row denotes the greedy action in that state.

$$\mathbf{Q}_{Q,STC,Step} = \begin{bmatrix} 30.36 & \mathbf{30.51} \\ 21.00 & \mathbf{28.51} \end{bmatrix} \quad \mathbf{Q}_{Q,Log,Rat} = \begin{bmatrix} 29.92 & \mathbf{30.64} \\ 21.00 & \mathbf{28.42} \end{bmatrix}$$

$$\mathbf{Q}_{Q,Exp,Smax} = \begin{bmatrix} \mathbf{30.93} & 30.79 \\ 20.75 & \mathbf{29.86} \end{bmatrix} \quad \mathbf{Q}_{S,Rat,Smax} = \begin{bmatrix} 15.00 & \mathbf{30.56} \\ 13.58 & \mathbf{29.17} \end{bmatrix}$$

$$\mathbf{Q}_{S,Exp,Step} = \begin{bmatrix} 20.25 & \mathbf{26.90} \\ 16.68 & \mathbf{26.20} \end{bmatrix} \quad \mathbf{Q}_{S,Log,Rat} = \begin{bmatrix} \mathbf{16.15} & 11.90 \\ 7.45 & \mathbf{13.95} \end{bmatrix}.$$

Observe that for Q-learning, the instances $Q, STC, Step$ and Q, Log, Rat accurately estimate the optimal state-action value functions and identify the optimal policy, while the instance $Q, Exp, Smax$ quite accurately estimated $\mathbf{Q}^*$ but the greedy action was

not optimal. In the case of SARSA, no instance provided an accurate estimate of $\mathbf{Q}^*$. The instances for $S, Rat, Smax$ and $S, Exp, Step$ identified the optimal policy while S, Log, Rat did not. Note also that when the greedy policy was not optimal, the discrepancy occurred in state s_1 where the two values in $\mathbf{Q}^*$ are quite close.

The above analysis provides a framework for use in other applications but cautions against generalizing specific recommendations regarding selection of learning rates and exploration methods. As in the episodic example, results are sensitive to learning rates and exploration method.

10.7.3 Infinite horizon average reward models

This section describes and illustrates two Q-learning approaches for finding effective policies for infinite horizon models with the average reward criterion. Reviewing Chapter 7 provides relevant background.

Following (7.63), the optimal state-action value function $q^*(s,a)$ for a recurrent or unichain average reward model is given by:

$$q^*(s,a) = \sum_{j \in S} p(j|s,a)\big(r(s,a,j) - g^* + h^*(j)\big) \tag{10.78}$$

and the optimal state-action value function and the optimal gain is a solution of the Bellman equation:

$$q(s,a) = \sum_{j \in S} p(j|s,a)\left(r(s,a,j) - g + \max_{a \in A_j} q(j,a)\right). \tag{10.79}$$

Written as an expected value, (10.79) is equivalent to:

$$q(s,a) = E^{d^*}\left[r(X,Y,X') - g + \max_{a' \in A_{X'}} q(X',a') \,\middle|\, X = s, Y = a\right].$$

Applying gradient descent[54] to minimize the squared difference between $q(s,a)$ and its realized values suggests the following model-based recursion for estimating $q(s,a)$ using simulation or process data:

$$q(s,a) \leftarrow q(s,a) + \tau\left(r(s,a,s') - g + \max_{a' \in A_{s'}} q(s',a') - q(s,a)\right) \tag{10.80}$$

where s' is sampled from $p(\cdot|s,a)$. The Q-learning algorithm below implements this recursion by combining it with a recursion that updates the estimate of g at *every* iteration. Note that the model-free version replaces $r(s,a,s')$ by an observed reward r.

[54] With the caveat that the semi-gradient replaces the gradient.

Q-learning

The following Q-learning algorithm generalizes the variant of Algorithm 10.8 that uses (10.60) to update the gain.

Algorithm 10.12. Q-learning for an average reward model

1. **Initialize:**
   (a) Set $q(s,a) \leftarrow 0$, $g \leftarrow 0$ and $\text{count}(s,a) \leftarrow 0$ each $a \in A_s$ and $s \in S$.
   (b) Specify learning rates $\tau_n, n = 1, 2, \ldots$ and $\beta_n, n = 1, 2, \ldots$.
   (c) Specify the number of iterations N.
   (d) Specify a sequence $\epsilon_n, n = 1, 2, \ldots$ (or η_n for softmax sampling).
   (e) Specify $s \in S$ and $a \in A_s$.
   (f) $n \leftarrow 1$.
2. **Iterate:** While $n \leq N$:
   (a) $\text{count}(s,a) \leftarrow \text{count}(s,a) + 1$.
   (b) Simulate (s', r) or sample s' from $p(\cdot|s,a)$ and set $r \leftarrow r(s,a,s')$.
   (c) **Update** $q(s,a)$**:**

$$q(s,a) \leftarrow q(s,a) + \tau_{\text{count}(s,a)} \left(r - g + \max_{a' \in A_{s'}} q(s',a') - q(s,a) \right). \tag{10.81}$$

   (d) **Update** g**:**

$$g \leftarrow g + \beta_n (r - g). \tag{10.82}$$

   (e) Generate $a \in A_{s'}$ using ϵ-greedy (or softmax) sampling.
   (f) $s \leftarrow s'$.
   (g) $n \leftarrow n + 1$.
3. **Terminate:** Return $q(s,a)$ for all $a \in A_s$ and $s \in S$.

Some comments follow:

1. Note that since g is updated at every iteration, its learning rate β_n in (10.82) is indexed by the iteration index n. On the other hand, the learning rate for $q(s,a)$ in (10.81) may be indexed by n or the number of times that state-action pair (s,a) has been previously evaluated.

2. Perhaps better estimates could be obtained by updating g using (10.82) before updating $q(s,a)$ using (10.81).

3. Observe that there are four hyperparameters to specify: two learning rates, the exploration rate and the number of iterations. Experimentation with these parameters in the context of policy evaluation through temporal differencing (Algorithm 10.8) might guide choice.

4. As noted earlier in this chapter, the above implementation is off-policy, that is the action chosen for execution at the next iteration need not correspond to the value $\max_{a' \in A_s} q(s',a')$ used in the update (10.81). A SARSA (on-policy update) replaces this expression by

$$q(s,a) \leftarrow q(s,a) + \tau_{\text{count}(s,a)} \left(r - g + q(s',a') - q(s,a) \right).$$

where a' denotes an ϵ-greedy action choice in state s'. Note that in numerical experiments, SARSA did not perform well.

5. Algorithm 10.12 converges[55] under the conditions that every state-action pair is visited infinitely often, the Markov chain of every stationary policy is regular and aperiodic, and the Robbins-Monro conditions on the learning rates hold.

6. If the algorithm is implemented using simulation, state-action based sampling at the start of step 2 might enhance empirical performance through more accurate estimates of $q(s,a)$ for infrequently visited states.

Relative Q-learning

Numerical studies suggest that the iterates in the above algorithm frequently diverge. The following alternative approach[56] generalizes relative value iteration (Algorithm 7.2). It considers updates of the form

$$q(s,a) \leftarrow q(s,a) + \tau \left(r - f(q) + \max_{a' \in A_{s'}} q(s',a') - q(s,a) \right), \tag{10.83}$$

where possible choices for $f(q)$ include:

1. $f(q) = q(\bar{s},\bar{a})$ for some specified pair $(\bar{s},\bar{a})$,

2. $f(q) = \max_{a \in A_{\bar{s}}} q(\bar{s},a)$ for some specified state $\bar{s}$, and

3. $f(q)$ equal to the average of $q(s,a)$ over all state-action pairs.

[55] See Tsitsiklis and Roy (1999).

[56] This algorithm and variants was proposed and analyzed by Abounadi et al. (2001).

Many implementations of relative value iteration use the specification $f(q) = q(\bar{s}, \bar{a})$, which was used to analyze (numerically) relative value iteration in Chapter 7. However, its performance is sensitive to the designation of $(\bar{s}, \bar{a})$. If visited infrequently, estimates will be inaccurate resulting in inaccurate estimates of $q(s, a)$. If possible, frequent restarts in $(\bar{s}, \bar{a})$ may result in improved performance.

Algorithm 10.13. Relative Q-learning for an average reward model

1. **Initialize:**
   (a) Specify a real-valued function $f(q)$.
   (b) Set $q(s, a) \leftarrow 0$ and $\text{count}(s, a) \leftarrow 0$ each $a \in A_s$ and $s \in S$.
   (c) Specify the learning rate $\tau_n, n = 1, 2, \ldots$
   (d) Specify the number of iterations N.
   (e) Specify a sequence $\epsilon_n, n = 1, 2, \ldots$ (or η_n for softmax sampling).
   (f) Specify $s \in S$ and $a \in A_s$.
   (g) $n \leftarrow 1$.
2. **Iterate:** While $n \leq N$:
   (a) $\text{count}(s, a) \leftarrow \text{count}(s, a) + 1$.
   (b) Simulate (s', r) or sample s' from $p(\cdot|s, a)$ and set $r \leftarrow r(s, a, s')$.
   (c) **Update** $q(s, a)$**:**

$$q(s, a) \leftarrow q(s, a) + \tau_{\text{count}(s,a)} \left(r - f(q) + \max_{a' \in A_{s'}} q(s', a') - q(s, a) \right). \quad (10.84)$$

   (d) Generate $a \in A_{s'}$ using ϵ-greedy (or softmax) sampling.
   (e) $n \leftarrow n + 1$ and $s \leftarrow s'$.
3. **Terminate:** Return $q(s, a)$ for all $a \in A_s$ and $s \in S$.

Observe that Relative Q-learning and Q-learning differ in how they estimate g. Relative Q-learning computes the estimate of g, $f(q)$, in step 2(c) using the previously evaluated $q(s, a)$, while Q-learning directly updates g at every iteration using (10.82).

From a theoretical perspective, in a unchain model:

1. The iterates of $f(q)$ converge to g^*,
2. The iterates of $q(s, a)$ converge to $q^*(s, a)$ normalized so that $f(q^*) = g$,
3. The greedy policy based on $q^*(s, a)$ is optimal.

These results hold under the Robbins-Monro step-size conditions and any of the above specifications for $f(q)$.

The two-state example

Applying Q-learning and Relative Q-learning to the two-state model of Section 2.5 highlights some inherent challenges when using these methods. Calculations in Section 7.7.2 establish that the stationary policy $(d^*)^\infty$ that chooses $d^*(s_1) = a_{1,2}$ and $d^*(s_2) = a_{2,2}$ is optimal, that $g^* = 2.86$, and the relative value obtained setting $h^*(s_1) = 0$ is $h^*_{\text{rel}}(s_2) = h^*(s_2) - h^*(s_1) = -2.14$.

An immediate issue is how to compare methods and model specifications. Comparing resulting policies on the basis of their gain and agreement with the optimal policy is straightforward. Comparisons based on the state-action value functions is problematic because $q^*(s,a)$ is unique up to a constant that is independent of (s,a). Instead results can be compared on the basis of relative values[57], under a specification such as $h^*(s_1) = 0$.

To determine an easy to use representation for relative values, recall that $q^*(s,a)$ is defined in Chapter 7 by

$$q^*(s,a) = E^{(d^*)^\infty}\left[\sum_{n=1}^{\infty}\left(r(X_n,Y_n) - g^*\right)\middle| X_1 = s, Y_1 = a\right].$$

Therefore $h^*(s) = q^*(s, d^*(s))$ for all $s \in S$ where $(d^*)^\infty$ is an optimal stationary policy. Since

$$d^*(s) \in \arg\max_{a \in A_s} q^*(s,a),$$

$h^*(s) = \max_{a\in A_s} q^*(s,a)$. To ensure $h^*(s_1) = 0$, subtract $h^*(s_1)$ from $h^*(s)$ for all $s \in S$ so that

$$h^*_{\text{rel}}(s) := h^*(s) - h^*(s_1)$$

for all $s \in S$.

Putting this altogether, if in this example $q(s,a)$ is obtained by applying Q-learning or Relative Q-learning, an estimate of $h^*_{\text{rel}}(s_2)$ is given by

$$h_{\text{rel}}(s_2) = q(s_2, d^*(s_2)) - q(s_1, d^*(s_1)) = \max_{a\in A_{s_2}} q(s,a) - \max_{a\in A_{s_1}} q(s_1,a) \tag{10.85}$$

Therefore, $h_{\text{rel}}(s_2)$ can be compared to $h^*(s_2) = -2.14$. The beauty of using this representation for $h_{\text{rel}}(s)$ is that it extends easily to larger applications and avoids evaluating the greedy policy explicitly. It also could be the basis for a formal termination criterion for Q-learning or Relative Q-learning.

Estimates using Q-learning and Relative Q-learning set $\tau_n = n^{-1}\log(n+1)$, $\beta_n = n^{-1}$ (for Q-learning), and used ϵ_n-greedy exploration with $\epsilon_n = 150/(300+n)$. The

[57] This quantity is more stable than $q^*(s,a)$ since some state-action pairs may be visited infrequently and not accurately estimated.

index for τ_n was the number of visits to each state-action pair and the index for ϵ_n was the number of previous visits to that state[58]. Relative Q-learning used seven choices for $f(q)$: $f(q) = q(\bar{s}, \bar{a})$ where $(\bar{s}, \bar{a})$ varied over all (four) state-action pairs (denoted RQL1-RQL4), $f(q) = \max_{a \in A_s} q(\bar{s}, a)$ for some $\bar{s}$ (denoted RQL5-RQL6) and $f(q) = \text{mean}\, q(\cdot, \cdot)$ (denoted RQL7). Comparisons were based on 40 independent replicates. Each replicate evaluated Q-learning and Relative Q-learning across all 7 choices of $f(q)$, with $N = 20{,}000$, 35,000, and 50,000. Common random number seeds were used for each of 24 instances evaluated within a replicate.

Table 10.8 gives the number of replications in which there were the designated number of non-optimal actions, classified by algorithm version and number of iterations. Observe that:

1. Q-learning most accurately identified the optimal policy. The impact of run length was minimal.

2. Relative value results varied with the form of $f(q)$. Accuracy for all improved as the number of iterations increased.

3. The differences between RQL1-RQL4 show that the method is sensitive to choice of $(\bar{s}, \bar{a})$.

4. RQL5, which used $f(q) = \max_{a \in A_{s_1}} q(s_1, a)$, more accurately chose the optimal policy than RQL6, which used $\bar{s} = s_2$.

Iterations	Non-optimal actions	QL	RQL1	RQL2	RQL3	RQL4	RQL5	RQL6	RQL7
20,000	0	38	33	23	30	27	34	28	32
	1	0	5	10	7	4	3	7	6
	2	2	2	7	3	9	3	5	2
35,000	0	39	34	28	34	26	36	29	34
	1	0	4	10	5	6	2	8	5
	2	1	2	2	1	8	2	3	1
50,000	0	39	34	28	37	26	37	31	36
	1	0	6	10	2	8	1	6	3
	2	1	0	2	1	6	2	3	1

Table 10.8: Distribution of the number of non-optimal actions in 40 replicates broken down by method and run-length for average reward version of the two-state model.

Figure 10.22 shows the accuracy and variability of estimates of g^* and $h^*(s_2)$, classified by run length and optimization method. Observe that:

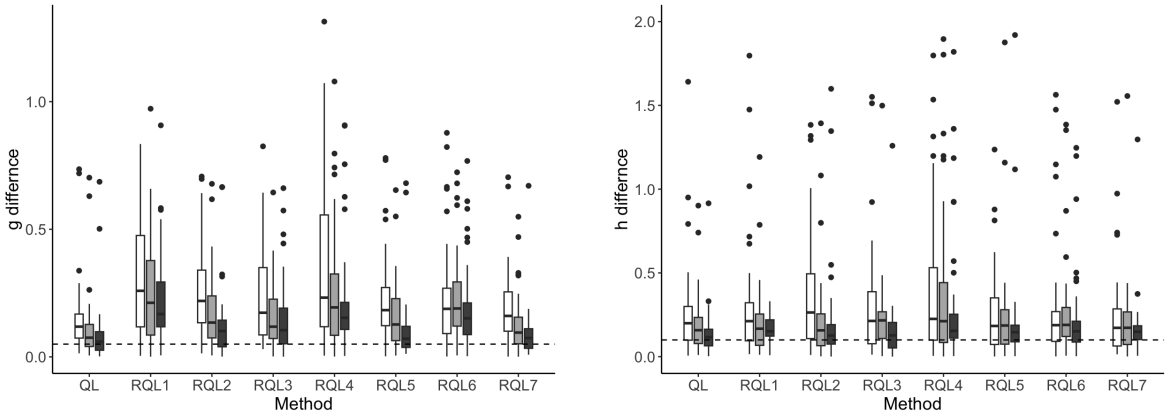
1. Q-learning produced the most accurate and least variable estimates of g^*. Its variability decreased as N increased.

[58] For state s, $n = \sum_{a \in A_s} \text{count}(s, a)$.

2. RQL7 and RQL5 provided the best estimates of g^* among the Relative Q-learning methods. These methods used the most iterations to estimate $f(q)$. Overall, Relative Q-learning methods were more sensitive to the number of iterations than Q-learning.

3. Q-learning estimates of $h^*(s_2)$ were as at least as accurate as those of any other methods. Precision increased and variability decreased with the number of iterations.

4. The accuracy and variability of Relative Q-learning estimators were similar across all choices of $f(q)$ and most were comparable to that of Q-learning. Accuracy increased and variability decreased with increasing number of iterations.



(a) $|g - g^*|$ (b) $|h - \text{rel}(s_2) - h^*_{\text{rel}}(s_2)|$

Figure 10.22: Difference between estimates and true values classified by estimation and number of iterations (white fill - 25,000; gray fill - 35,000; dark gray fill - 50,000). Dashed lines are included to facilitate comparisons and not meant as targets for the estimators. Note that the scale of the two figures differ.

As a consequence of these observations, Q-learning, as implemented in Algorithm 10.12, most frequently and efficiently[59] identified the optimal policy. Moreover it provided the most accurate and least variable estimates of g^* and $h^*(s_2)$.

One reason for the accuracy of estimating the gain was that estimates were updated at *every* iteration (using a running mean). Accuracy and variability of Relative Q-learning methods were sensitive to choice of $f(q)$ and were surpassed by Q-learning.

Of course, these conclusions apply only to this example and the specified learning rate and action sampling parameters. To establish broader validity requires similar comparisons in other environments and a review of relevant recommendations in the literature.

[59]The accuracy of the policies identified with $N = 35{,}000$ and $N = 50{,}000$ were the same.

10.8 Policy iteration type algorithms*

There is little general agreement on the structure of policy iteration algorithms in simulation-based environments. The methods proposed below provide plausible implementations.

This section focuses on discounted models; development of simulation-based policy iteration algorithms for episodic and average reward models is left to the reader. Q-learning and SARSA may be viewed as adaptations of value iteration algorithms from Section 5.6 to a simulation environment. Each iterate involves some form of maximization with Q-learning using the "max" to update $q(s,a)$ and SARSA using ϵ-greedy or softmax sampling to select actions at each update.

As was seen in Chapter 5, policy iteration and modified policy iteration involved less frequent maximizations and faster convergence, so it may be fruitful to investigate simulation-based versions of policy iteration (Section 5.7) and modified policy iteration (Section 5.8). In general, the structure of such algorithms can be represented by Figure 10.23. In earlier chapters, such an algorithm was implemented using the known Markov decision process model to compute $q^{d^\infty}(s,a)$ exactly using (5.64). This section explores the use of simulation methods to estimate this intermediate quantity. As a consequence of the variability introduced through simulation, the stopping rule "stop when $d' = d$" may require modification.

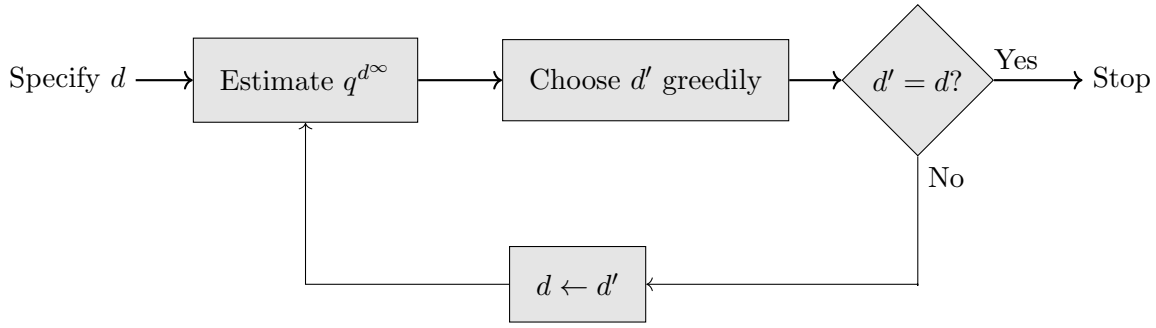


Figure 10.23: Structure of a policy iteration algorithm.

Simulation-based policy iteration methods can implement the estimation step in different ways:

1. Estimate $v_\lambda^{d^\infty}(s)$ and compute $q_\lambda^{d^\infty}(s,a)$ using the model.
2. Sample to estimate $r(s,a,j)$ and $p(j|s,a)$, then use these quantities to estimate $v_\lambda^{d^\infty}(s)$ and $q_\lambda^{d^\infty}(s,a)$.
3. Estimate $v_\lambda^{d^\infty}(s)$ and then simulate to estimate $q_\lambda^{d^\infty}(s,a)$.
4. Estimate $q_\lambda^{d^\infty}(s,a)$ directly.

A discussion of the first and last methods follow. Analysis of the other two approaches is left to the reader.

10.8.1 Hybrid policy improvement

First, consider a value function-based algorithm in which the evaluation step uses simulation to estimate the policy value function and then implements "exact" improvement in the same way as deterministic policy iteration in Chapter 5. While this may be impractical to implement when either S or A_s are large, and contrasts with the philosophy of this chapter, it will provide a useful baseline for algorithms in which improvement also involves simulation.

Algorithm 10.14. Hybrid policy improvement for a discounted model

1. **Initialize:** Specify $d \in D^{\mathrm{MD}}$, stopping tolerance $\nu \in \mathbb{Z}_+$, and set $\Gamma = S$.
2. **Iterate:** While $|\Gamma| > \nu$:
   (a) **Evaluation:** Evaluate d using TD(0), TD(γ) or Monte Carlo to obtain $\hat{v}(s)$ for $s \in S$.
   (b) **Construct $\hat{q}(s,a)$:** For all $s \in S$ and $a \in A_s$, set
   $$\hat{q}(s,a) \leftarrow p(j|s,a)(r(s,a,j) + \lambda \hat{v}(j)). \tag{10.86}$$
   (c) **Exact improvement:** For all $s \in S$,
   $$d'(s) \in \arg\max_{a \in A_s} \hat{q}(s,a), \tag{10.87}$$
   setting $d'(s) = d(s)$ if possible.
   (d) $\Gamma \leftarrow \{s \in S \,|\, d'(s) \neq d(s)\}$.
   (e) $d \leftarrow d'$.
3. **Terminate:** Return d.

Note that the termination criterion is based on the number of states in which actions change in successive iterations of step 2. While the condition $\nu = 0$ works for exact methods, empirical results suggest that it is too stringent for practical applications.

This algorithm does not apply to model-free settings in which $p(j|s,a)$ and $r(s,a,j)$ are not known. However, it can be adapted to a model-free setting using a two-stage approach in which $\hat{p}(j|s,a)$ and $\hat{r}(s,a,j)$ are learned from data. To facilitate this, the algorithm should start with a decision rule that randomizes action choice in all states. After obtaining these estimates, Algorithm 10.14 can be applied using

$$\hat{q}(s,a) := \hat{p}(j|s,a)(\hat{r}(s,a,j) + \lambda \hat{v}(j)). \tag{10.88}$$

Furthermore these estimates can be improved while evaluating subsequent decision rules.

Example 10.9. Hybrid policy improvement in the two-state model.

This example applies Algorithm 10.14 to the two-state model from Section 2.5 with $\lambda = 0.9$ and stopping tolerance $\nu = 0$. Results from the example in Section 10.7.2 guide parameter choice. It uses TD(0) for evaluation, sets $N = 5{,}000$ and compares two learning rates: $\tau_n = \log(n+1)/n$ and $\tau_n = 20/(50+n)$, where n represents the number of visits to the state. It also compares implementing TD(0) using state-based uniform random sampling with trajectory-based sampling. The algorithm is initialized with the sub-optimal policy $d = (a_{1,1}, a_{2,1})$.

The state-action value functions corresponding to this policy are

$$\begin{aligned}
\hat{q}(s_1, a_{1,1}) &= 3 + 0.9\big(0.8\hat{v}(s_1) + 0.2\hat{v}(s_2)\big) \\
\hat{q}(s_1, a_{1,2}) &= 5 + 0.9\hat{v}(s_2) \\
\hat{q}(s_2, a_{2,1}) &= -5 + 0.9\hat{v}(s_2) \\
\hat{q}(s_1, a_{2,2}) &= 2 + 0.9\big(0.4\hat{v}(s_1) + 0.6\hat{v}(s_2)\big)
\end{aligned}$$

where $\hat{v}(s)$ is an estimate of $v_\lambda^{d^\infty}(s)$ for the policy being evaluated.

Experiments consisted of 50 replicates using common random number seeds to compare learning rates and methods for generating the next state. Comparisons were based on the number of iterations to converge, the policy identified and the RMSE between the estimated $\hat{q}(s,a)$ and the known value of $q_\lambda^*(s,a)$.

In *all* cases the algorithm terminated with the optimal policy in a few iterations. For $N = 5{,}000$, the termination condition was achieved in most cases in five or fewer iterations. When N was increased to 20,000, trajectory-based sampling ensured convergence within three iterations for both learning rate choices. However, with state-based sampling, a few replications required up to five iterations.

Table 10.9 shows that TD(0) implemented with the log learning rate and trajectory-based state sampling most accurately estimated $q_\lambda^*(s,a)$ for both values of N. Furthermore, increasing N from 5,000 to 20,000 significantly improved the quality of estimates and resulted in faster algorithmic convergence.

It is left to the reader to apply variants of policy iteration that:

1. Estimate $r(s,a,j)$ and $p(j|s,a)$ assuming a model-free environment and then using them to estimate $q_\lambda^{d^\infty}(s,a)$, and
2. Estimate $\sum_{j\in S} p(j|s,a)\big(r(s,a,j) + \lambda\hat{v}(j)\big)$ using sampling after estimating $\hat{v}(\cdot)$.

Learning rate	State sampling	RMSE	
		$N = 5{,}000$	$N = 20{,}000$
$\log(n+1)/n$	State-based	1.88 (1.37)	0.92 (0.73)
	Trajectory-based	1.59 (1.43)	0.66 (0.55)
$20/(50+n)$	State-based	2.36 (1.54)	1.00 (0.75)
	Trajectory-based	2.22 (1.31)	0.77 (0.79)

Table 10.9: Mean (standard deviation) of RMSE errors of q-function estimates using hybrid policy iteration classified by learning rates and state generation method based on 50 replicates.

10.8.2 Hybrid policy iteration bounds

While theoretically justified in exact computation, the stopping rule in Algorithm 10.14 lacks a theoretical basis. However, one would expect that if values were accurately estimated, the condition to stop when $d' = d$ would eventually hold. Value function bounds, previously discussed in Chapter 5, provide a more rigorous approach for stopping, as well as bounding the difference between the value of a policy and the optimal value function.

Theorem 5.12 shows that for any function $v(s)$ on S,

$$v(s) + (1-\lambda)^{-1} \min_{s' \in S} \{L\mathbf{v}(s') - v(s')\} \leq v_\lambda^{d^\infty}(s) \leq v_\lambda^*(s) \leq v(s) + (1-\lambda)^{-1} \max_{s' \in S} \{L\mathbf{v}(s') - v(s')\}$$

where

$$L\mathbf{v}(s) := \max_{a \in A_s} \left\{ \sum_{j \in S} p(j|s,a) \left(r(s,a,j) + \lambda v(j) \right) \right\} = \max_{a \in A_s} q(s,a)$$

and

$$d(s) \in \arg\max_{a \in A_s} q(s,a).$$

Applying the above expressions to the iterates of hybrid policy iteration results in the following readily applicable bounds[60].

[60] We do not believe these have previously been used with simulated data.

Theorem 10.2. Suppose $\hat{v}(s)$, $\hat{q}(s,a)$ and d' are determined in step 2 of Algorithm 10.14. Then

$$\hat{v}(s) + (1-\lambda)^{-1} \min_{s' \in S} \left\{ \max_{a' \in A_{s'}} \hat{q}(s',a') - \hat{v}(s') \right\} \leq v_\lambda^{(d')^\infty}(s) \tag{10.89}$$
$$\leq v_\lambda^*(s) \leq \hat{v}(s) + (1-\lambda)^{-1} \max_{s' \in S} \left\{ \max_{a' \in A_{s'}} \hat{q}(s',a') - \hat{v}(s') \right\}$$

It is important to emphasize that for (10.89) to be valid $\hat{q}(s,a)$ must be defined by (10.86). While one might hypothesize that this inequality remains valid when $\hat{q}(s,a)$ is estimated directly (as in Q-learning), our numerical experiments indicated that this supposition is false. Obtaining efficient stopping rules for algorithms based on state-action value functions remains an open question. One approach in a model-based environment would be to intermittently set $\hat{v}(s) = \max_{a \in A_s} \hat{q}(s,a)$ and subsequently use (10.86) to obtain $\hat{q}(s,a)$ that is compatible with the bounds.

Since all of the quantities in (10.89) are available after step 2(b) of hybrid policy iteration, the above bounds provide the basis for the following stopping criterion for Algorithm 10.14.

Corollary 10.1. Let

$$\delta := (1-\lambda)^{-1} \left(\max_{s' \in S} \left\{ \max_{a' \in A_{s'}} \hat{q}(s',a') - \hat{v}(s') \right\} - \min_{s' \in S} \left\{ \max_{a' \in A_{s'}} \hat{q}(s',a') - \hat{v}(s') \right\} \right). \tag{10.90}$$

Then if $\delta < \epsilon$,

$$v_\lambda^*(s) - \hat{v}(s) < \epsilon \quad \text{for all } s \in S.$$

Moreover (10.89) provides upper and lower bounds on $v_\lambda^*(s)$.

Example 10.10. This example illustrates the application of the stopping criterion in Corollary 10.1 and the bounds on the optimal value function in Theorem 10.2 when hybrid policy iteration is applied to the two-state model. Guided by results in Example 10.9, it generates 50 replicates of hybrid policy iteration with the policy value function estimated by TD(0) with trajectory-based state sampling, a log learning rate and $N = 20{,}000$.

The stopping criterion was implemented with $\epsilon = 2$ meaning that the algorithm terminates when $\delta < 2$. This tolerance was chosen since it was known for this model with $\lambda = 0.9$ that the difference[a] between the value of the optimal policy and that of its closest stationary policy exceeded 2. Smaller values of δ were problematic because of variability in estimates of $\hat{v}(s)$.

Table 10.10 shows the distribution of iterations required to satisfy $\delta < 2$ across replicates. Observe that the algorithm terminated in four or fewer iterations in 86% of the replicates.

Upper and lower bounds along with the value of δ for a typical replicate appear in Table 10.11. In this replicate, hybrid policy iteration would terminate after four iterations. Note that δ is not monotone as a result of variability in TD(0) estimates of $\hat{v}(s)$.

The empirical validity of the bound is supported by the observation that $\mathbf{v}^*_\lambda = (30.15, 27.94)$ lies within *all* computed upper and lower bounds. Moreover, the value of the second-best stationary policy, $(27.19, 25.63)$, lies outside of the bounds for all iterations except the first.

[a] Of course this tolerance would not be known a priori in practice.

Iterations	2	3	4	5	6	7
Count	10	15	18	2	4	1

Table 10.10: Distribution of the number of iterations for hybrid policy iteration to satisfy $\delta < 2$ in 50 replicates.

Iteration	Lower bound	Upper bound	δ
1	$\begin{bmatrix} -59.42.62 \\ -109.09 \end{bmatrix}$	$\begin{bmatrix} 248.51 \\ 198.83 \end{bmatrix}$	307.92
2	$\begin{bmatrix} 27.71 \\ 26.03 \end{bmatrix}$	$\begin{bmatrix} 34.91 \\ 33.23 \end{bmatrix}$	7.20
3	$\begin{bmatrix} 28.54 \\ 26.69 \end{bmatrix}$	$\begin{bmatrix} 33.29 \\ 31.43 \end{bmatrix}$	4.74
4	$\begin{bmatrix} 30.10 \\ 27.90 \end{bmatrix}$	$\begin{bmatrix} 30.25 \\ 28.05 \end{bmatrix}$	0.15
5	$\begin{bmatrix} 29.11 \\ 26.79 \end{bmatrix}$	$\begin{bmatrix} 30.67 \\ 28.36 \end{bmatrix}$	1.57

Table 10.11: Upper and lower bounds for $v^*_\lambda(s)$, and the bound, δ, obtained from a typical replicate in Example 10.10.

10.8.3 Online policy improvement

The following policy iteration algorithm is suitable for both model-free and model-based implementations and is designed for online implementation. Its beauty is that it estimates the optimal state-action function directly without first estimating a policy value function, making a model unnecessary. Simulation-based evaluation of $q(s,a)$ is analogous to modified policy iteration.

The improvement step of a policy improvement algorithm requires an estimate of

$$q_\lambda^{d^\infty}(s,a) := \sum_{j\in S} p(j|s,a)\big(r(s,a,j) + \lambda v_\lambda^{d^\infty}(j)\big). \tag{10.91}$$

Therefore, it is not sufficient to just evaluate d through simulation since that will only provide an estimate of $q_\lambda^{d^\infty}(s,d(s)) = v_\lambda^{d^\infty}(s)$.

To overcome the absence of estimates of $q(s,a)$ for $a \neq d(s)$, the proposed algorithm estimates $q(s,a)$ for all states and actions by replacing a deterministic policy d^∞ by a randomized policy that is "close" to d^∞. To do this, the algorithm evaluates the *ϵ-tweaked policy* d_ϵ that chooses actions in state s according to

$$w_{d_\epsilon}(a|s) = \begin{cases} 1-\epsilon & a = d(s) \\ \epsilon/(|S|-1) & a \neq d(s). \end{cases} \tag{10.92}$$

It is important to emphasize that this is different than an ϵ-greedy policy because the policy it is based on need not be optimal. This policy can be implemented in a simulation by sampling action $d(s)$ with probability $1-\epsilon$ and choosing a specific other action with probability $\epsilon/(|S|-1)$. It may be prudent to decrease ϵ as calculations ensue.

Algorithm 10.15. Online simulated policy improvement for a discounted model

1. **Initialize:**
   (a) Specify $d \in D^{\mathrm{MD}}$, stopping tolerance $\nu \in \mathbb{Z}_+$, and set $\Gamma = S$.
   (b) Specify the number of evaluation iterations N.
   (c) Specify a sequence $\epsilon_n, n = 1,2,\ldots$ for ϵ-tweaked action sampling and learning rate sequence $\tau_n, n = 1,2,\ldots$.
2. **Iterate:** While $|\Gamma| > \nu$:
   (a) Specify $s \in S$ and set $n \leftarrow 1$.
   (b) $q(s,a) \leftarrow 0$ and $\mathrm{count}(s,a) \leftarrow 0$ for all $a \in A_s$ and $s \in S$.
   (c) **Evaluate** $q^{d^\infty}(s,a)$**:** While $n \leq N$:
      i. $i \leftarrow \sum_{a\in A_s} \mathrm{count}(s,a)$ and $\epsilon \leftarrow \epsilon_i$.

ii. Sample a from $w_{d_\epsilon}(\cdot|s)$.

iii. Simulate (s', r) or sample s' from $p(\cdot|s, a)$ and set $r \leftarrow r(s, a, s')$.

iv. **Update** $q(s, a)$**:**

$$q(s, a) \leftarrow q(s, a) + \tau_{\text{count}(s,a)}(r + \lambda q(s', d(s')) - q(s, a)). \quad (10.93)$$

v. $\text{count}(s, a) \leftarrow \text{count}(s, a) + 1$ and $n \leftarrow n + 1$.

vi. $s \leftarrow s'$.

(d) **Greedy improvement:** For all $s \in S$,

$$d'(s) \in \arg\max_{a \in A_s} q(s, a), \quad (10.94)$$

setting $d'(s) = d(s)$ if possible.

(e) $\Gamma = \{s \in S \,|\, d'(s) \neq d(s)\}$.

(f) $d \leftarrow d'$.

3. **Terminate:** Return d.

Some comments about applying this algorithm follow:

1. The algorithm might be initiated with a randomized policy that chooses actions in each state with equal probability to obtain a good preliminary estimate of $q(s, a)$, and ϵ could be set to 1 for the first pass through the evaluation step.

2. Choosing $\epsilon_1 = 1 - (1/|S|)$ starts the evaluation with a decision rule that randomizes over all actions. In general, ϵ_n should decrease with n but not too quickly. The algorithm uses the number of times a state has been visited (as opposed to the iteration number) to ensure that there is sufficient action sampling in each state.

3. Step 2(a) can be implemented with a fixed or random s.

4. The algorithm stops when d' and d differ in at most a small number, ν, of states. Setting $\nu = 0$ recovers the stopping rule "stop when a decision rule repeats". However, this rule may be too stringent when q-functions are estimated directly. A simpler option is to specify the number of times to execute step 2.

5. There are two options for the update of $q(s, a)$ in (10.93). As stated, it is an off-policy algorithm that corresponds to Q-learning where the maximum is over the single policy d. Alternatively an on-policy update, corresponding to SARSA, requires sampling the action a' in state s' according to the ϵ-tweaked decision rule d_ϵ and replacing $d(s')$ in (10.93) by a'. This would require adding $a \leftarrow a'$ at the end of the Evaluate loop.

One would suspect that the off-policy variant would be preferable since the objective is to obtain an estimate of $q_\lambda^{d^\infty}(s,a)$ and not $q_\lambda^{d_\epsilon^\infty}(s,a)$.

6. Step 2(c)vi is specified for trajectory-based sampling. State-based sampling replaces $s \leftarrow s'$ by "Sample $s \in S$". Alternatively, state-action pair sampling would also replace Step 2(c)ii by "sample $a \in A_s$".

7. The algorithm is stated with TD(0) updates. Alternatively, TD(γ) updates could be used.

Example 10.11. Online simulated policy iteration in the two-state model

This example applies Algorithm 10.15 to the two-state model with $\tau_n = \log(n+1)/n$, $\epsilon_n = 1{,}500/(3{,}000+n)$, $N = 50{,}000$ and $\nu = 0$ corresponding to stopping when two successive policies are identical. Forty replicates with common random number seeds were used to investigate the combined effect of using:

1. on-policy of off-policy updates in (10.93), and

2. trajectory-based or state-based sampling in step 2(c)vi.

In all cases, the algorithm terminated with an optimal policy. The on-policy variant terminated in three iterations regardless of how the next state was generated while the off-policy variant converged in three iterations in 73 out of 80 cases and in five iterations in 7 out of 80 cases.

Table 10.12 shows the accuracy of the q-function estimates based on the RMSE between the estimated and true values. As expected, the off-policy methods, which are related to Q-learning, were considerably more accurate than the on-policy methods, which are related to SARSA.

The off-policy trajectory-based version produced the most accurate estimates suggesting that an online implementation produces good results. But, the algorithm cycled with less judicious choices of τ_n and ϵ_n so care is needed when specifying parameters.

Update	State sampling	RMSE (mean ± SD)
On-Policy	State-based	17.48 ± 7.70
	Trajectory-based	20.69 ± 5.35
Off-Policy	State-based	0.88 ± 1.21
	Trajectory-based	0.31 ± 0.35

Table 10.12: Summary of the accuracy of q-function estimates using variants of online simulated policy iteration for the two-state model based on 40 replicates with common random number seeds.

10.9 Queuing service rate control revisited

The infinite horizon version of the queuing service rate control model introduced in Section 3.4.1 provides a good platform to investigate some challenges that arise when implementing the above algorithms in a simple but more realistic model. Particular features that make this problem attractive for analysis include:

1. the ordered and interpretable state space,
2. the known structure of optimal value functions, and
3. the proven monotonicity of the optimal policy.

This section focuses on the discounted version but a short discussion of the average reward version is also included. The objective of this analysis is to find an optimal policy for future use so it is not necessary to use an online implementation or evaluate performance while learning.

Even though implementation involves only 51 states with three actions in each, the problem is sufficiently complex to raise a range of implementation issues that must be considered when dealing with modern large-scale applications. Experiments with a wide range of options are reported and provide a framework that may be applied broadly.

The analysis truncates the length of the queue at 50 and uses the parameters: arrival rate $b = 0.2$, service rates $a_1 = 0.2$, $a_2 = 0.4$ and $a_3 = 0.6$, holding cost $f(s) = s^2$, service rate cost $m(a_k) = 5k^3$ for $k = 1, 2, 3$, and discount rate $\lambda = 0.9$. Because the objective is to minimize the expected discounted cost, it is formulated as a *minimization* problem with positive costs.

Using value iteration and policy iteration (Chapter 5) when $\lambda = 0.9$, the optimal policy $(d^*)^\infty$ was found to be

$$d^*(s) = \begin{cases} a_1 = 0.2 & s \in \{0, \ldots, 10\} \\ a_2 = 0.4 & s \in \{11, \ldots, 29\} \\ a_3 = 0.6 & s \in \{30, \ldots, 50\} \end{cases} \tag{10.95}$$

and the optimal value function was monotone increasing and had monotone increasing differences (was convex) on $\{0, 1, \ldots, 48\}$[61]. Under the average reward criterion, the optimal policy uses actions $a_1 = 0.2$ in states $\{0, 1, 2\}$, $a_2 = 0.4$ in states $\{3, \ldots, 8\}$ and $a_3 = 0.6$ in states $\{9, \ldots, 50\}$.

10.9.1 Monte Carlo policy evaluation

This section explores the use of truncation and geometric stopping time Monte Carlo estimates of the discounted value function for the policy d^∞ with $d(s) = a_2$ for all

[61] As a result of truncation, the values for $s \in \{49, 50\}$ deviated from this pattern.

$s \in S$. Recall that estimates of $v_\lambda^{d^\infty}(s)$ in each state are obtained by averaging over replicates in that state.

Some observations and comments follow:

1. Only starting-state Monte Carlo applies. This is because the truncation horizon or geometric stopping time are defined in terms of the length of the horizon starting from the first transition. Hence, first-visit estimates, which would start later, would correspond to shorter horizons.

2. In this model, the estimated standard deviation $\hat{\sigma}(s)$ of $\hat{v}(s)$ (Figure 10.24a) varied over the states. Thus, sample sizes (number of Monte Carlo replicates) needed to obtain the same precision for all states must vary with the state. Note that the "artificial" boundary at $s = 50$ resulted in the non-monotone behavior near the upper boundary.

3. By the Central Limit Theorem, the Monte Carlo estimate for state s is asymptotically normal with standard deviation approximately equal to $\hat{\sigma}(s)/n^{-0.5}$. This relationship can be used to choose the sample size to obtain a pre-specified level of confidence.

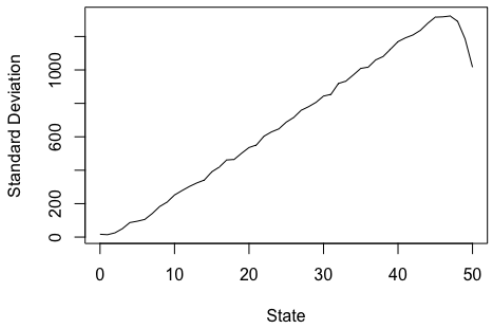
4. In light of the previous comment, the truncation estimator used $K = \max\{100, 4s^2\}$ replicates in state s resulting in a total of 172,080 replicates.

5. Differences between the estimated and true value function are shown in Figure 10.24b for the truncation estimate. Note the RMSE of these estimates (over states) was 12.3. These estimates were more precise than TD(0) estimates described below (Figure 10.26a). However, the total number of replicates was considerably larger.

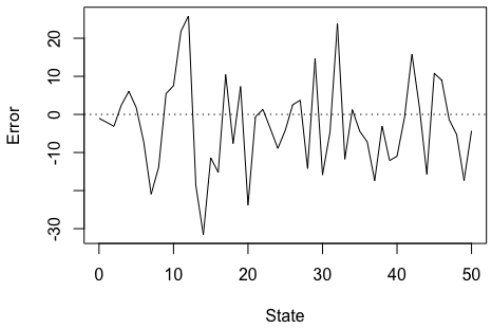
6. Geometric stopping time estimates were considerably more variable than the truncation estimates (standard deviation in state 50 was 200 times greater than that of the truncation estimator) and much less accurate overall (RMSE = 103.7). Note that the expected number of terms in the geometric sums was 11 as opposed to 125 used in truncation.

10.9.2 TD(γ) policy evaluation

Evaluation using TD(γ) presents some challenges. Under the policy that uses action a_1 in every state, the system is a symmetric reflecting random walk on $\{0, \ldots, 50\}$ so that in steady state all states are visited with equal probability. Therefore applying TD(γ) for evaluation of this policy with trajectory-based sampling is straightforward since all states will be visited approximately the same number of times in a simulation. However, evaluating other reasonable policies becomes problematic because the system



(a) Estimated standard deviation of Monte Carlo policy value function estimates.



(b) Difference between estimated and true policy value function. The number of replicates in state s was set at $\max\{100, 4s^2\}$.

Figure 10.24: Some results for Monte Carlo estimation based on reward sequence truncation at $M = 125$.

primarily visits the low-numbered states[62]. Hence, estimates of value functions or state-action value functions using trajectory-based sampling will be highly variable for the infrequently visited, high-occupancy states. Therefore, it seems plausible that state-based sampling after each transition would improve the accuracy of TD(γ) methods.

The following small study investigated the effect of learning rate, parameter of TD(γ), trace type and policy on the accuracy of policy value function estimates. The study consisted of 40 replicates of 100,000 iterations for each of five learning rates (exponential, ratio $(150/(300+n)$ and $15/(30+n))$, STC, and Log), four values of γ, two trace types (replacement and accumulation) and two stationary policies (one that used action a_2 in every state and another that repeated the sequence (a_1, a_2, a_3) beginning in state 0 and continuing through state 50). The discount rate was set equal to $\lambda = 0.9$, states were generated randomly at each iteration and common random number seeds were used in each replicate. The index of the learning rate was the number of previous visits to a state.

Since the true policy value function could be found using exact methods for policy evaluation, replicates were summarized in terms of the RMSE of the policy value function estimates over all states. Figure 10.25 displays the main results graphically. Observe that:

1. Trace type and policy had insignificant effect on results and were not included in the figure.

[62] It is left as an exercise to compute the stationary distribution under policies that use action a_2 or a_3 in all states.

2. The Log and ratio $(150/(300+n))$ learning rate generated very high RMSEs and were not included in the figure.

3. The ratio learning rate $\tau_n = 15/(30+n)$ generated estimates with the smallest RMSEs. For this learning rate, $\gamma = 0$, corresponding to TD(0), gave the smallest RMSE (41.2). For the STC learning rate, $\gamma = 0.2$ gave the smallest RMSE.

4. Combined TD(0) and ratio estimates were most accurate in low-numbered states on an absolute level (Figure 10.26).

Figure 10.26 shows that the TD(0) policy value function estimates deviated from the true policy value function by at most 112. On a relative basis this error is at most 2.3%.

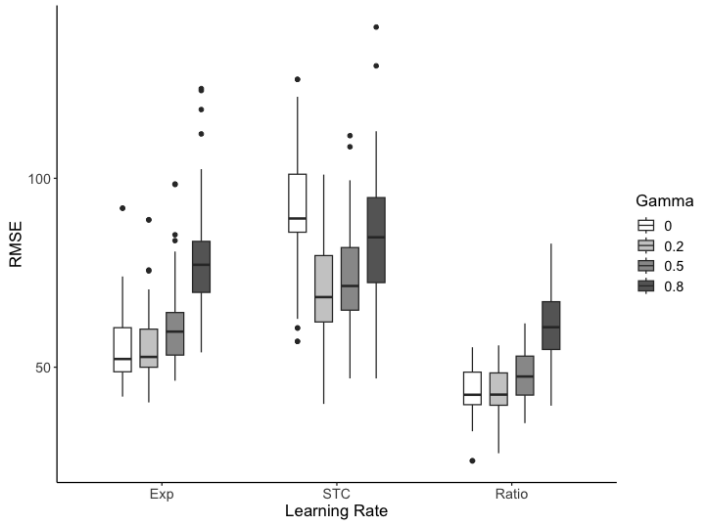
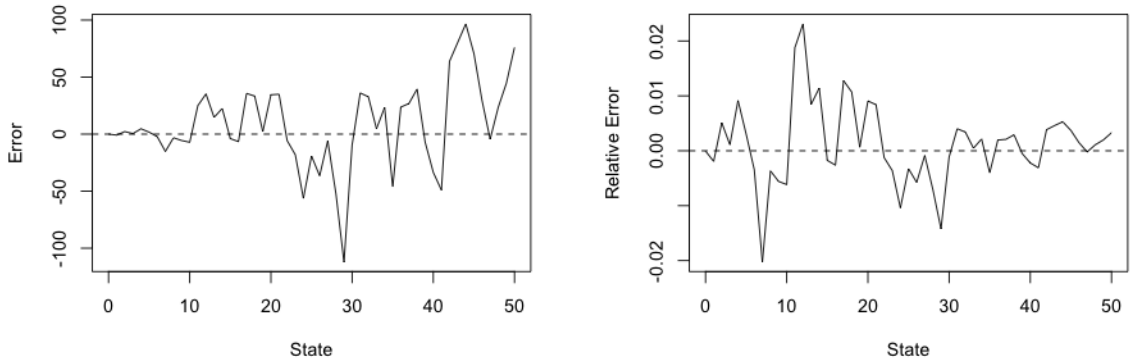


Figure 10.25: Graphical summary of the dependence of RMSE on learning rate and γ when using TD(γ) for policy value function approximation for d^∞ with $d(s) = a_2$ for all $s \in S$. Each boxplot represents a summary over 40 replicates. Note the effect of trace and policy were insignificant and are not shown.

Comparison of TD(γ) and Monte Carlo estimates

Table 10.13 compares two Monte Carlo estimates with the best TD(γ) estimate on the basis of RMSE and the number of transitions evaluated to obtain a policy value function estimate for all states. Note that the RMSE for TD(0) was the average over 40 replicates of length 100,000 but in practice one would only generate one such estimate. On the other hand the large number of replicates required to obtain Monte Carlo estimates was due to the need to obtain multiple replicates for each starting state. Thus it appears that the TD(0) method is the computationally most efficient (at the expense of parameter tuning).



(a) Difference between estimated and true policy value function.

(b) Relative difference between estimated and true policy value function.

Figure 10.26: Error and relative error using TD(0) with learning rate $\tau_n = 15/(30+n)$ for estimating the policy value function of d^∞ where $d(s) = a_2$ for all $s \in S$ based on single randomly selected replicate.

Method	RMSE	Total number of transitions to estimate the policy value function
Monte Carlo - Truncation	12.3	182,384 × 125
Monte Carlo - Geometric	103.7	182,384 × 11
TD(0) with ratio learning	41.2	100,000

Table 10.13: Summary of value function calculations. Note 182,384 represents the total number of sequences generated when obtaining estimates.

10.9.3 Q-learning

This section explores the use of the Q-learning algorithm (Algorithm 10.11) and its SARSA variant. Since the optimal value function and policy can be computed using methods in Chapter 5, the quality of the estimates can be assessed by comparing the policy value function of the greedy policy[63] to the optimal value function on the basis of RMSE over all states. In addition, the greedy policy can be compared to the optimal policy on the basis of the percentage of states on which they agree.

State-based versus trajectory-based sampling

The first step in the analysis was a comparison of trajectory-based sampling and uniform state-based sampling in individual replicates. Results for a typical replicate are displayed in Figure 10.27. Figures 10.27a and 10.27c show the greedy policy (points)

[63] Found by solving $(\mathbf{I} - \lambda \mathbf{P}_d)\mathbf{v} = \mathbf{r}_d$ where d^∞ denotes the greedy policy.

and the optimal policy (dashed line). Figures 10.27b and 10.27d show the difference in values (on different scales) of the greedy policy and the optimal policy.

Observe that the greedy policy derived using state-based sampling closely matched the optimal policy and achieved a value very close to it. On the other hand, the greedy policy derived using trajectory-based sampling had a similar structure and value to the optimal policy only in states $0 - 17$.

The reason for such poor performance in high-occupancy states using trajectory-based sampling is that in 450,000 replicates, states $18 - 50$ were visited infrequently: for example, states $30 - 50$ were visited at most three times. Hence, in these states, the $q(s, a)$ estimates were extremely inaccurate. Theoretically, although the underlying Markov chain is regular, the limiting probabilities for high-occupancy states are extremely small making them rarely observed under trajectory-based sampling.

In practice, this limitation may not impact overall system performance since high-occupancy state are seldom encountered. However, this issue can be addressed by:

1. Comparing values using RMSE weighted by the limiting probabilities of the optimal policy[64].

2. Using state-based (or state-action based sampling) to ensure all states or state-action pairs are sampled sufficiently.

This issue is not limited to this queuing model. Similar limitations arise when using Q-learning with trajectory-based sampling in other process-type applications. Thus, where the objective is to find an optimal policy applicable in **all** states, state-based or state-action based sampling is preferred.

State-based sampling

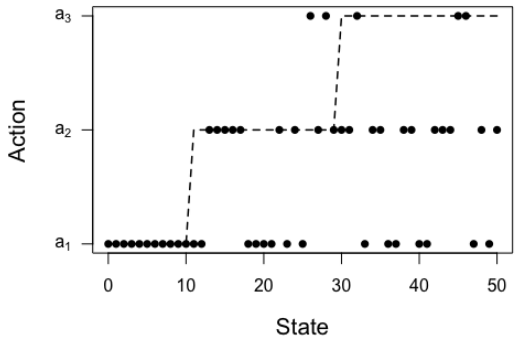
The following results were based on an experiment that used 40 replicates of length 450,000 with random state-based sampling at each iteration. It compared five learning rate specifications (used for policy evaluation above), four choices (0.1, $150/(300+n)$, $15/(30+n)$, $0.7n^{-0.5}$) for ϵ_n and two estimation methods (Q-learning and SARSA).

Q-learning and SARSA produced equivalent results. The most accurate learning rate τ_n and ϵ_n-greedy specifications among those considered were

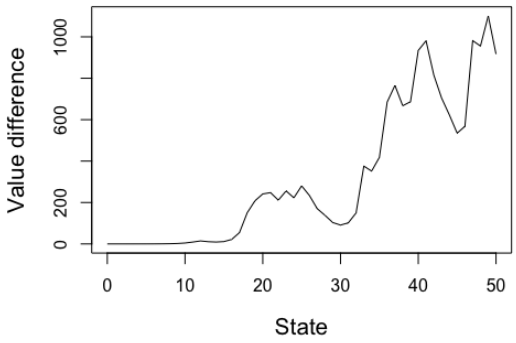
$$\tau_n = \frac{0.1}{1 + n^2/10^5} \quad \text{and} \quad \epsilon_n = \frac{150}{300 + n}. \tag{10.96}$$

For the learning rate, n represented the number of visits to a state-action pair and for ϵ_n, n represented the number of visits to a state. This combination generated the

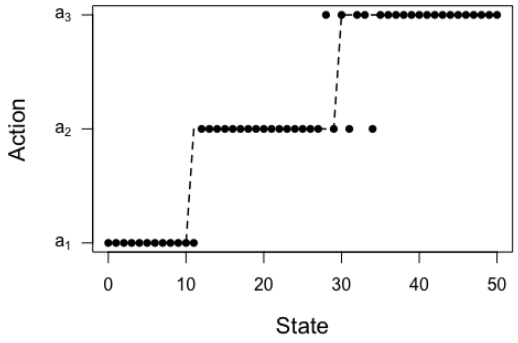
[64] Of course these probabilities would not be known in practice. Instead, one could assign weights based on observed empirical frequencies or the importance of states. To use a sports analogy, training should focus on skills for situations most likely to be encountered in competition and limited for unlikely scenarios.



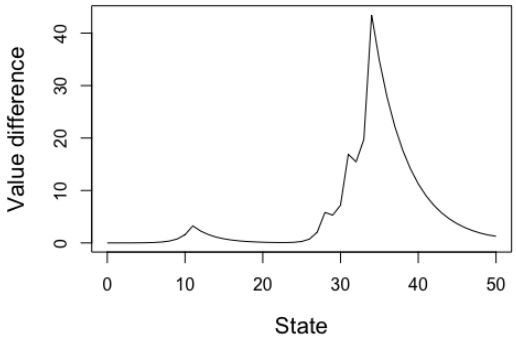
(a) Policy - Trajectory-based



(b) Value - Trajectory-based



(c) Policy - State-based



(d) Value - State-based

Figure 10.27: Comparison of policies and values using Q-learning with trajectory-based and state-based sampling. Note the difference in the scales in Figures 10.27b and 10.27d.

most accurate estimates of the state-action value function and the highest percentage of states in which the greedy policy was optimal over 40 replicates.

Results in Figures 10.27c and 10.27d were typical for this specification. Using Q-learning, the state-action value function estimates were extremely accurate. The mean RMSE of the state-action value function (across replicates) was 19.8 with standard deviation of 11.1, which is small relative to the range of the $q(s, a)$ values spanning 70 to 22,600. Greedy policies deviated from the optimal policy in only a few states. On average, 90.6% of actions generated by the greedy policy agreed with the optimal policy, with a standard deviation of 3.7%.

10.9.4 Policy iteration type algorithms

This section describes the use of hybrid policy iteration (Algorithm 10.14) and online policy iteration (Algorithm 10.15). Recall hybrid policy iteration estimates a policy value function using simulation and then uses the model to compute the q-function, while online policy iteration estimates q-functions directly. Implementation choices[65] include:

Policy evaluation method: TD(0), TD(γ), or Monte Carlo with truncated discounted rewards or geometric stopping times;

Algorithmic parameters: learning rates, ϵ-greedy exploration settings, and number of iterations;

Sampling method: state-based or trajectory-based updates;

Improvement step: exact vs. estimated q-functions;

Policy value function estimates: smoothed vs. unsmoothed value functions;

q-function estimates: smoothed vs. unsmoothed state-action value functions;

Smoothing technique: Parametric model used for smoothing;

Stopping criterion: Extent of agreement of policies at successive iterations.

Configurations were compared on the basis of whether the algorithms terminated or cycled, how similar the identified policy was to the optimal policy, and how well the estimated value function approximated the optimal value function.

To obtain convergence in online policy iteration, it was necessary to smooth $q(s, a)$ as a function of s prior to greedy improvement. Moreover, smoothing could potentially enhance convergence of hybrid policy iteration. A convenient choice for this purpose was a *quadratic B-spline*[66]. Chapter 11 provides a systematic analysis for combining simulation and function approximation.

Hybrid policy iteration: Implementation

Algorithm 10.14 combines policy value function estimation with greedy improvement. Given a policy value function estimate $\hat{v}(s)$, it uses the model to evaluate (10.86) according to

$$q(s, a) = \sum_{j \in S} p(j|s, a)(r(s, a, j) + \hat{v}(j)) \tag{10.97}$$

[65] Not all apply to each method.

[66] B-splines are linear combinations of polynomials that flexibly approximate nonlinear functions. In R, they are specified with the function `bs()` and fit with function `lm()`. To avoid pre-specifying knot locations, our implementations here used the specification `degree=2` and `df=2`.

$$= \begin{cases} 5a^3 + (1-q)\hat{v}(0) + q\hat{v}(1) & s = 0 \\ s^2 + 5a^3 + a\hat{v}(s-1) + (1-a-q)\hat{v}(s) + q\hat{v}(s+1) & s \in \{1, 2, \ldots, 49\} \\ 50^2 + 5a^3 + a\hat{v}(49) + (1-a)\hat{v}(50) & s = 50. \end{cases}$$

Note that $r(s, a, j)$ is independent of j.

Policy evaluation used TD(0) and starting-state Monte Carlo with truncation and geometric stopping times, and both smoothed and unsmoothed q-functions. For each of these six combinations, 40 replicates with common seeds across each configuration within a replicate were evaluated.

Details of the implementation follow:

1. TD(0) parameters were specified as $\tau_n = 15/(30 + n)$ and $N = 100{,}000$. States were resampled after each transition.

2. Each replicate of Monte Carlo evaluated all starting states. Both truncated and geometric stopping time versions used $\max(100, 3s^2)$ iterations in state s. Truncation was set at $M = 100$[67].

3. The stopping criterion was "stop when $|\Gamma| \leq 2$" where Γ denotes[68] the set of states on which the policy at two successive iterates differ. More restrictive conditions led to cycling.

Hybrid policy iteration: Results

Results of the comparisons are summarized in Table 10.14. Without smoothing, hybrid policy iteration with TD(0) and Monte Carlo with truncation achieved the stopping criteria in approximately six improvement steps, with identical ranges of 2 to 18 steps. The algorithm did not converge when Monte Carlo estimates were based on geometric stopping times.

Average RMSE[69] for both approaches were comparable but values of greedy policies based on Monte Carlo replicates were less variable. Hybrid policy iteration with TD(0) identified the optimal policy in 4 of 40 cases while the Monte Carlo variant with truncation identified the optimal policy in 5 of 40 cases. However, when q-function smoothing was applied, both methods reduced the number of improvement steps to two and almost always identified the optimal policy.

Since both TD(0) and Monte Carlo with truncation accurately estimated the policy value function, hybrid policy iteration using these estimates achieved the stopping criterion and produced accurate estimates without smoothing. Given the greater computational effort to implement Monte Carlo with truncation, TD(0) is the preferred evaluation choice if hybrid policy iteration is to be used without smoothing. When

[67] These values were modified from Section 10.9.1 to speed up execution. Nonetheless, Monte Carlo with truncation was extremely time consuming (several hours on a Macbook Pro M3).

[68] See step 2(d) in the algorithm.

[69] Between the greedy policy and the optimal policy.

Evaluation Method	Smoothing q-function	Average Iterations	Range Iterations	Average RMSE	Range RMSE
TD(0)	No	5.58	(2, 18)	15.48	(0, 71.07)
	Yes	2	2	0	0
Monte Carlo (truncation)	No	5.85	(2, 18)	17.0	(0, 39.8)
	Yes	2	2	0	0
Monte Carlo (geometric stopping)	No	Did not converge	-	-	-
	Yes	2	2	0.31	(0, 1.52)

Table 10.14: Comparison of results of hybrid policy improvement methods based on 40 replications with common random number seeds.

smoothing is applied, all evaluation methods, including Monte Carlo with geometric stopping, perform similarly.

Online policy iteration: Implementation

Algorithm 10.15 was applied using $N = 450{,}000$, $\nu = 2$, learning rates $\tau_n = 0.1/(1 + n^2/10^5)$ and exploration parameters $\epsilon_n = 1{,}500/(3{,}000 + n)$. State transitions were modeled using random state-based sampling and actions were sampled using a ϵ_n-tweaked decision rule d_{ϵ_n}.

Both off-policy and on-policy updates for the state-action value function were examined. The on-policy update used

$$q(s,a) = q(s,a) + \tau_n\big(r(s,a,a') + \lambda q(s,a') - q(s,a)\big), \tag{10.98}$$

where a' is chosen using the ϵ_n-tweaked decision rule d_{ϵ_n}. The off-policy update uses (10.93).

Additionally, the effect of q-function smoothing was investigated. A total of 40 replicates were evaluated using common random numbers in each.

Online policy iteration: Results

Table 10.15 summarizes the results. Results for unsmoothed q-functions are omitted because decision rules generated using both on-policy and off-policy updates failed to achieve the proposed stopping criterion, even after 25 evaluation-improvement cycles. In fact, between 20 and 30 actions changed at each cycle.

Results in the above table suggests that using d for updating the q-function provides more reliable performance. This makes sense because $q(s,a)$ reflects the value of using action a in state s for one period and then following the policy d^∞, not the exploratory policy based on d_{ϵ_n}.

Thus, when applying online policy iteration, smoothing appears necessary to obtain convergence. Even in small problems, this suggests the potential value of function approximation.

Decision rule used in (10.93)	Smoothing q-function	Average Iterations	Range Iterations	Average RMSE	Range RMSE
d	Yes	3.95	(2, 10)	15.02	(0, 68.3)
d_{ϵ_n}	Yes	4.33	(2, 10)	12.81	(0, 58.4)

Table 10.15: Summary of results based on 40 replicates of Algorithm 10.15. In two replicates, the variant using d_{ϵ_n} in the update terminated after two iterations with sub-optimal policies (RMSE = 1523). These two outliers are excluded from the reported RMSE values for d_{ϵ_n}.

10.9.5 Computational summary: Discounted model

Analysis of the discounted queuing control model suggests:

1. Convergence and accuracy of algorithms was highly sensitive to learning rate and, when appropriate, to the exploration parameter ϵ_n. The specification of n (state-action pair visits for learning rate or state visits for exploration rate) impacted convergence.

2. TD(0) provided reliable estimates of the policy value function. TD(γ) estimates were less accurate.

3. Starting-state Monte Carlo provided accurate policy value function estimates but required more replicates than TD(0) to obtain comparable precision. Methods based on truncated discounted rewards were more accurate than those using geometric stopping times.

4. Optimization methods required state-based sampling to obtain near optimal policies. Trajectory-based methods sampled some parts of the state space too infrequently.

5. Using state-based sampling, Q-learning produced policies that differed from the optimal policies in a few states. Although smoothing the q-function was not necessary to obtain convergence, it likely would have generated monotone policies.

6. Policy iteration algorithms behaved best with a stopping criterion of the form "Stop when the decision rule at successive iterates differs in at most two states." Tighter stopping rules often led to cycling.

7. Hybrid policy improvement with state-based sampling terminated without q-function smoothing. However, q-function smoothing resulted in faster convergence to the optimal policy.

8. Online policy iteration required q-function smoothing for convergence. The off-policy variant as stated in Algorithm 10.15 converged more reliably.

These findings suggest that when a model is available, hybrid policy iteration with TD(0) or truncated Monte Carlo policy value function estimation were most reliable and precise. However, the Monte Carlo estimates required considerably more computation. In the absence of a model, Q-learning was easiest to apply and provided good results without smoothing the q-function. Simulated policy iteration required q-function smoothing to ensure convergence. In contrast to the two-state example above, methods were most reliable when states were randomly sampled at each transition as opposed to trajectory-based sampling.

Smoothing the q-function enhanced convergence and produced monotone policies. Smoothing involved two-steps at each iteration, estimating the q-function for all state-action pairs and then approximating it by a smooth function prior to greedy action selection. This was possible in a small model, the next chapter investigates incorporating function approximation directly in the optimization algorithm enabling generalization to larger models.

10.9.6 Average reward queuing control

This section investigates the application of Algorithms 10.12 and 10.13 to an average reward version of the queuing service rate control model. Preliminary analyses suggest that the only effective approach was Relative Q-learning with state-based sampling.

Q-learning (Algorithm 10.12) consistently identified suboptimal policies that deviated significantly from the optimal policy under any state sampling regime. The key issue was the inaccuracy of estimates of g, which resulted in inaccurate estimates of state-action value functions. Relative Q-learning with trajectory-based sampling also failed to identify good policies, primarily because of infrequent visits to high-occupancy states.

Implementation and results

A discussion of the application of Relative Q-learning with state-based sampling follows. The implementation specified $f(q) = q(0, a_1)$ meaning this quantity was used to estimate the gain in (10.83). Computations compared configurations with learning rates $100/(200+n)$ and $0.2/(1+10^{-6}n^2)$ paired with either ϵ-greedy exploration with $\epsilon_n = 100/(200+n)$ or softmax exploration[70] with $\eta = 1/2{,}000$.

Each configuration was evaluated over 40 replicates with common random number seeds, simulating $N = 300{,}000$ total iterates (roughly 6,000 per state) per replicate. Methods were assessed on the accuracy of estimates of the optimal average reward g^* and the number of states in which the greedy policy differed from the optimal policy.

Table 10.16 summarizes the results. Clearly ϵ-greedy exploration was preferable to softmax exploration, the latter being highly sensitive to parameter choice. There was

[70] Results were sensitive to specification of η; choosing it too small resulted in limited search across actions.

a small benefit to using the STC learning rate as opposed to the ratio learning rate. As the table shows, its estimate of the optimal gain was more accurate.

Learning Rate	Exploration	Non-optimal Actions	Average Reward Difference
Ratio	ϵ-greedy	2.18 (1.31)	2.41 (1.70)
	Softmax	11.40 (3.33)	4.71 (2.73)
STC	ϵ-greedy	2.18 (1.20)	0.95 (0.84)
	Softmax	4.85 (2.19)	2.96 (1.49)

Table 10.16: Accuracy of Relative Q-learning estimates as a function of learning rate and exploration method. Entries are means (standard deviations) over 40 replicates.

Figure 10.28 compares the greedy policy to the optimal policy for a typical selected replicate. Observe that the resulting policy closely approximates the optimal policy, differing from it in only three states. In this replicate, the estimate of $g^* = 19.42$ was 16.58.

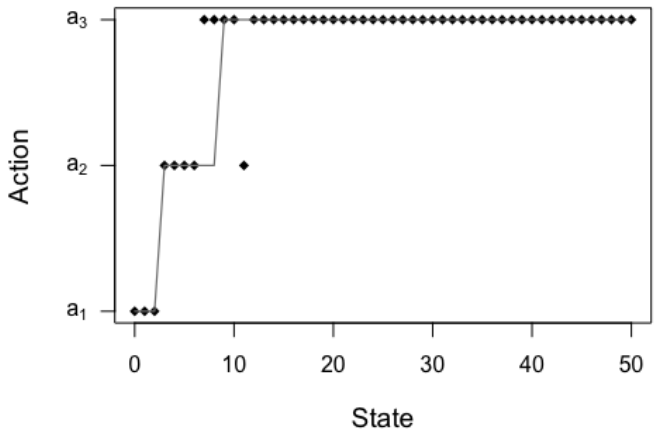


Figure 10.28: Plot of greedy decision rule (diamonds) and average optimal decision rule (line) for a randomly selected replicate using STC learning rate and ϵ-greedy exploration.

It was encouraging to find that Relative Q-learning, without any additional smoothing, identified policies that are close to optimal. The implementation used state-based sampling; actions were chosen by exploration. It is possible that state-action based sampling would result in even more accurate results.

10.10 Conclusion

The foundational methods in this chapter provide a "toolbox" of potential approaches (Monte Carlo, temporal differencing, TD(γ) for policy evaluation; Q-learning, SARSA and policy iteration type algorithms for optimization) to apply in simulated or process-based environments suitable for tabular representations. However, applying them requires considerable trial-and error tuning of learning rates, exploration parameters, sampling methods and run lengths to produce useful results. Moreover, appropriate specifications for a given application are not obvious *a priori*.

10.11 Technical appendix

10.11.1 Choosing good estimators

Policy evaluation methods in this chapter are concerned with estimates of an unknown quantity based on a sequence of observations or replicates of an experiment. Estimates may be computed offline using the whole data set, for example a sample mean, or online, updated sequentially using stochastic approximation algorithms described in Appendix 10.11.2. Standard statistical concepts such as bias, variance and mean squared error are useful for assessing the quality of an estimator.

Many examples in this chapter require choosing among estimates such as those represented by the different lines in Figure 10.29. They may be generated by different algorithms or from a single algorithm under different parameter specifications. In the figure, the light gray estimate is the least variable but fails to converge to the true value given by the horizontal black line, in other words, it is biased. The dark gray and black estimates both oscillate around the true value but the black is less variable and therefore preferred. This illustrates that choosing an estimator involves trading-off bias and variability as described below.

First, some caveats are in order.

1. The estimates in Figure 10.29 are based on a single replicate of a simulation using a common random number seed. Results may differ with different random sequences.

2. In practice, a single estimation method and a pre-specified number of iterates will be specified.

3. Results are sensitive to run lengths. If sampling stopped after approximately 4,200 steps, one might conclude that the estimate corresponding to the dark grey line gave the best approximation of the unknown quantity.

Visualizing results across different run lengths, as in this figure, facilitates selection of estimators that produce good estimates over a range of run lengths.

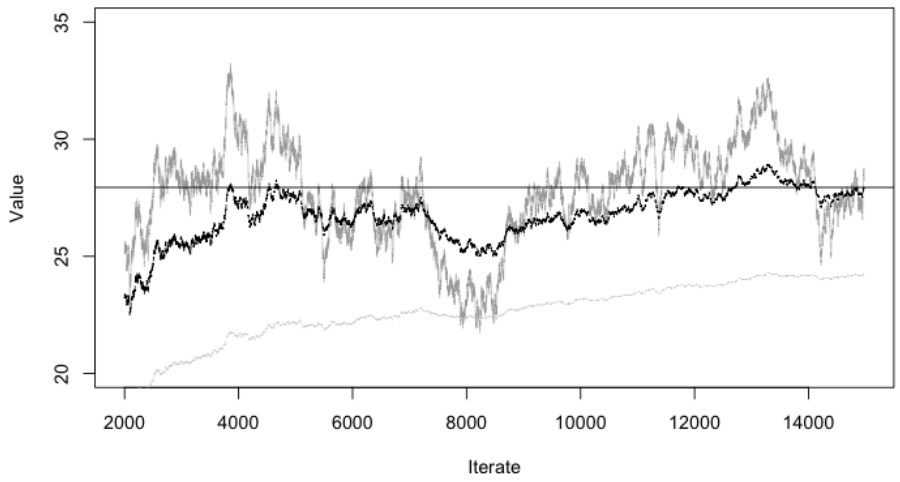


Figure 10.29: Three sequences of estimates of a quantity based on a single replicate of an online algorithm with run length of 15,000. The horizontal line represents the true value which is unknown in practice but known in the illustrative experiments used in many of the examples in Chapters 10 and 11.

RMSE for point estimation

Consider an experiment with K replicates that generates estimates $y^k, k = 1, 2, \ldots, K$ of a fixed scalar C. For example, y^k may be the estimate of a policy value function in a specified state obtained from a single Monte Carlo replicate.

The quality of the estimate is often assessed using the *mean squared error (MSE)* defined by

$$\text{MSE} := \frac{1}{K}\sum_{k=1}^{K}(y^k - C)^2.$$

Since the MSE is expressed in squared units, the *root mean squared error (RMSE)*, defined by

$$\text{RMSE} = \sqrt{\text{MSE}},$$

is preferred because it is on the same scale as the estimates and hence provides a more interpretable measure of accuracy.

The MSE is related to two other statistical measures, the *sample bias* given by

$$\text{Bias} := |\bar{y} - C|$$

and the *sample variance* given by

$$\text{Variance} := \frac{1}{K-1}\sum_{k=1}^{K}(y^k - \bar{y})^2,$$

where $\bar{y}$ denotes the sample mean. Note that the variance and MSE measure deviations from different quantities, the sample mean and true value, respectively. When K replaces $K-1$ in the variance, this quantity is referred to as the *population variance*.

These quantities are related through the expression:

$$\text{MSE} = \left(\frac{K-1}{K}\right)\text{Variance} + \text{Bias}^2. \tag{10.99}$$

To see this, note that

$$\begin{aligned}\sum_{k=1}^{K}(y^k - C)^2 &= \sum_{k=1}^{K}(y^k - \bar{y} + \bar{y} - C)^2 \\ &= \sum_{k=1}^{K}(y^k - \bar{y})^2 + 2\sum_{k=1}^{K}(y^k - \bar{y})(\bar{y} - C) + \sum_{k=1}^{K}(\bar{Y} - C)^2 \\ &= \sum_{k=1}^{K}(y^k - \bar{Y})^2 + K(\bar{Y} - C)^2,\end{aligned}$$

since the cross product term sums to zero, the result follows by dividing both sides by K.

Hence the mean squared error contains both a variance and a bias component. If the population variance replaces the sample variance in (10.99) the fraction disappears so that this relationship becomes:

$$\text{MSE} = \text{Population Variance} + \text{Bias}^2. \tag{10.100}$$

Many conclusions in examples will be based on the RMSE, so the breakdown in either (10.99) or (10.100) is intended primarily for interpretability of this concept. It is referred to as the *bias-variance tradeoff*.

Table 10.17 provides summary measures[71] for the estimators represented in Figure 10.29. Observe that the estimator represented by the light grey line has small

[71] This example does not fall exactly within the above framework, since in it, y^k represents the k-th value in a single replicate of length 15,000 for a single estimator. To interpret results, assume that statistical measures are computed after obtaining all 15,000 values so $\bar{y}$ is the overall mean value for that estimator.

variance and high bias, the estimator represented in dark grey has high variance and low bias, and that in black has both small variance and small bias. Moreover, the RMSE of the estimator in black is the smallest. On this basis one would prefer the estimator corresponding to the black line, consistent with the conclusions based on inspecting the figure directly.

Estimator	RMSE	MSE	Population Variance	Bias^2
Black	1.59	2.54	1.17	1.37
Dark-grey	2.09	4.37	4.33	0.04
Light-grey	5.47	29.93	1.58	28.35

Table 10.17: Statistical summaries of estimators in Figure 10.29.

RMSE for value function estimation

The RMSE is frequently used in a different context in this chapter, namely for assessing the accuracy of different approaches for estimating policy or optimal value functions. Suppose $\hat{v}(s)$ is an estimator of a known function $v(s)$ defined over a finite state space S. The RMSE of the estimate is defined by:

$$\text{RMSE}(\hat{\mathbf{v}}) := \sqrt{\frac{1}{|S|}\sum_{s\in S}(\hat{v}(s) - v(s))^2}. \tag{10.101}$$

Studies of simulation-based algorithms, such as in Section 10.9, often generate K replicates $\hat{v}^1(s), \ldots, \hat{v}^K(s)$ by varying the random number seed. In such settings, algorithmic performance is often assessed using summary statistics such as the mean, standard deviation and range of the sequence[72] of RMSE values $\text{RMSE}(\hat{\mathbf{v}}^1), \ldots, \text{RMSE}(\hat{\mathbf{v}}^K)$ displayed in tables or graphically using boxplots.

Of course when the primary objective is to find effective policies, as discussed in Section 10.6.2, precise estimation of value or state-action value functions may be of secondary importance. Algorithmic comparisons should be based on how close policies and their theoretical or simulated policy value functions are to optimal.

Weighted root mean squared error

Often it makes sense to replace RMSE by a *weighted RMSE*, especially when:

1. evaluating value functions which differ by orders of magnitude over S, or
2. in trajectory-based simulations where subsets of states are visited infrequently so that values in these states are not accurately estimated.

[72] Recall $\mathbf{v}$ represents a vector of values over states.

Moreover, a related concept, the weighted supremum norm is sometimes used to derive theoretical properties of estimators, especially when S is not finite.

The weighted RMSE of an estimator $\hat{\mathbf{v}}$ of a vector $\mathbf{v}$ is defined by

$$\text{wRMSE}(\hat{\mathbf{v}}) := \sqrt{\sum_{s \in S} w(s)\big(\hat{v}(s) - v(s)\big)^2}, \tag{10.102}$$

where $w(s) \geq 0$ for $s \in S$. Note setting $w(s) = 1/|S|$ yields the unweighted RMSE.

Suppose $w(s)$ is chosen to be the *stationary distribution* of the Markov chain corresponding to a policy being evaluated or that identified by an algorithm when optimizing. Such a measure better represents the accuracy (or potential loss of optimality) that will be observed in practice when implementing the policy. Of course, this distribution will not be known in optimization problems. However, it can be calculated in experimental settings in which the optimal policy is known.

Alternative choices for weights include the reciprocal of the variance of $\hat{v}(s)$ over multiple replicates of an experiment or an exponential or polynomial growth function.

10.11.2 Stochastic approximation

A new type of approximation process called "stochastic approximation" will play an important role in multistage decision processes[73].

Richard P. Bellman, Mathematician and father of dynamic programming, 1920-1984.

This prescient remark by Bellman underscores our decision to include this lengthy appendix on stochastic approximation. As anticipated by Bellman, stochastic approximation underlies the temporal differencing, Q-learning and policy gradient methods described in this chapter and in Chapter 11.

Stochastic approximation originates with the seminal paper Robbins and Monro [1951][74], which developed an online algorithm for finding the root of a function based on noisy observations of the function value at selected points. It has become especially important for analyzing simulation data because it avoids storing large amounts of data.

The Robbins-Monro procedure

Stochastic approximation is an *online* method for finding an x^* that solves

$$E[Y|X = x] = 0.$$

[73] Bellman [1963].

[74] As an aside, Herbert Robbins was the PhD advisor of Cy Derman, who was the PhD advisor of Arthur F. Veinott Jr., who was the PhD advisor of author MLP.

It uses samples $y^n, n = 1, 2, \ldots$ from a distribution with conditional mean $f(x) := E[Y|X = x]$ as the basis of an estimator. For example, the sequence y^n may be generated by $y^n = f(x^n) + \epsilon_n$ where ϵ_n denotes a random error term with mean 0 and the sequence $x^n, n = 1, 2, \ldots$ is chosen in some way.

A Monte Carlo approach would generate multiple samples at different values of x and use them to approximate $f(x)$ and estimate where it equals 0. Robbins and Monro [1951] proposed something different. Their procedure generates a sequence $x^n, n = 1, 2, \ldots$ that represents both estimates of x^* and points at which to apply a recursion of the form:

The Robbins-Monro recursion:

$$x^{n+1} = x^n - \tau_n y^n, \tag{10.103}$$

where y^n denotes an observation from a distribution with conditional mean $E[Y|X = x^n]$ and τ_n, referred to as the *step-size* or *learning rate*, represents a sequence of non-negative constants converging to 0.

The intuition underlying (10.103) (see Figure 10.30) is that when y^n is greater than 0, x^{n+1} will be *corrected* downward from x^n and conversely when y^n is less than 0, x^{n+1} will be corrected upward from x^n. Moreover, when y^n is close to zero the correction will be smaller as will be the case for large n when τ_n is smaller.

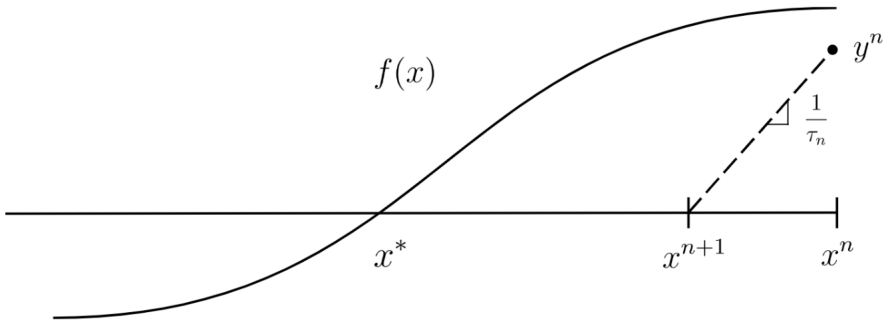


Figure 10.30: Geometric representation of Robbins-Monro recursion for solving $f(x) = 0$. Since $y^n/(x^n - x^{n+1}) = 1/\tau^n$, $x^{n+1} = x^n - \tau_n y^n$.

This recursion may be viewed as a stochastic generalization of Newton's method. When $f(x^n)$ is observable (and differentiable), Newton's method finds its zero by iter-

ating

$$x^{n+1} = x^n - (f'(x^n))^{-1} f(x^n). \tag{10.104}$$

Equation (10.103) replaces the reciprocal of the derivative by the constant τ_n and $f(x^n)$ by an observation sampled from a distribution with mean $f(x^n)$.

The following theorem (stated without proof) describes the limiting behavior of the sequence of iterates generated by (10.103).

Theorem 10.3. Suppose $f(x) = E[Y|X = x]$,

1. $f(x)$ is monotone,
2. $E[(Y - f(X))^2|X = x] < M < \infty$ for some constant M, and
3. $|f(x)| \leq d|x| + c$ for finite positive constants c and d,
4. the sequence $\tau_n, n = 1, 2, \ldots$ satisfies

$$\tau_n \geq 0, \quad \sum_{n=1}^{\infty} \tau_n = \infty \quad \text{and} \quad \sum_{n=1}^{\infty} \tau_n^2 < \infty. \tag{10.105}$$

Then for any x^1, the sequence

$$x^{n+1} = x^n - \tau_n y^n$$

converges almost surely to x^* where $f(x^*) = 0$.

Some comments about this theorem follow:

1. In most of the applications of this result in this book, x^n will represent an estimate of a parameter (weight) or vector of parameters and be denoted by $\boldsymbol{\beta}^n$.
2. The theorem provides a stronger version of the result in Robbins and Monro [1951] under slightly different conditions on $f(x)$. It is due to Blum [1954]. It establishes almost sure convergence[75] as opposed to L^2 convergence. The key conditions on the step-size in (10.105) remain the same.
3. Proofs of this theorem and its variants are beyond the scope of this book. They use subtle arguments and advanced probabilistic concepts. See the Bibliographic Remarks section for references.
4. Much of the literature refers to (10.105) as "the usual conditions on τ_n".

[75] Also called convergence with probability one.

5. The conditions on τ_n in Theorem 10.3 are satisfied when $\tau_n = k/n^b$ for $b \in (.5, 1]$, $\tau_n = c/(d+n)$ where c and d are positive constants and $\tau_n = \log(n+1)/n$. Note the choice $\tau_n = 1/n$ satisfies these conditions.

6. Note that in addition to specifying the correction term to the current estimate of the solution of $f(x) = 0$, the Robbins-Monro recursion specifies at which value of x^n to generate a new observation.

7. This theorem applies to a univariate method; a vector version is also valid. It will be discussed in the context of gradient methods below.

8. Equation (10.103) can be rewritten as

$$x^{n+1} = (1 - \tau_n)x^n + \tau_n(x^n - y^n)$$

This representation writes x^{n+1} as a weighted average of the previous estimate x^n and its difference with the new observation y^n. This is referred to as *exponential smoothing* in the forecasting literature.

Example 10.12 below shows that in agreement with theory, when $\sum_{n=1}^{\infty} \tau_n < \infty$, the step-sizes converge to 0 too quickly leading to convergence to an incorrect value. Moreover, when $\sum_{n=1}^{\infty} \tau_n^2 = \infty$, variability is not damped out so that iterates oscillate rapidly. When the conditions on the sums of τ_n and τ_n^2 are satisfied, the iterates converge.

Example 10.12. Impact of τ_n on stochastic approximation estimates.
This example shows how the conditions in (10.105) impact convergence of the stochastic approximation iterates. Consider the problem of finding the fifth root of 2. Put into the stochastic approximation framework, it can be expressed as solving

$$f(x) := x^{1/5} - 2 = 0$$

based on realizations of $f(x) + \epsilon$ where ϵ is normally distributed with mean 0 and standard deviation 1.

The example compares estimates based on four choices for τ_n corresponding to cases when (10.105) hold and are violated in two different ways. Each estimate is based on 50,000 iterates, common random number sequences for the four choices of τ_n and $x_0 = 20$. Of course $f(32) = 0$ so that $x^* = 32$.

Clearly conditions 1 and 3 of Theorem 10.3 hold and since $\text{var}(\epsilon) = 1$, condition 2 holds. Choosing $\tau_n = 0.9n^{-0.5}$ and $\tau_n = 100/(400+n)$ correspond to the cases in which both summation conditions in (10.105) are satisfied, choosing $\tau_n = n^{-1.0000001}$ corresponds to $\sum_{n=1}^{\infty} \tau_n < \infty$ and $\sum_{n=1}^{\infty} \tau_n^2 < \infty$ and choosing $\tau_n = n^{-0.25}$ corresponds to $\sum_{n=1}^{\infty} \tau_n = \infty$ and $\sum_{n=1}^{\infty} \tau_n^2 = \infty$. Note that $\sum_{n=1}^{\infty} \tau_n < \infty$ and $\sum_{n=1}^{\infty} \tau_n^2 = \infty$ is impossible since for $\tau_n > 0$ small, $\tau_n^2 < \tau_n$. (As an exercise,

verify that these sequences satisfy the indicated conditions.)

Figure 10.31 displays the sequence x^n corresponding to each of the four step-size sequences. The figure legend identifies the sequences. The figure shows that:

1. sequences corresponding to $\tau_n = 0.9n^{-0.5}$ and $\tau_n = 100/(400+n)$ converge to x^* with that based on the ratio more stable,
2. the sequence corresponding to $\tau_n = n^{-1.0000001}$, converges but to the wrong value, and
3. the sequence corresponding to $\tau_n = n^{-0.25}$ oscillates around the target value $x^* = 32$.

Consequently, different violations of the assumptions on τ_n have different impacts on the sequence of iterates x^n. Moreover, the ratio step-size parameter choice provides the most stable estimates.

The following example illustrates the use of stochastic approximation in a non-standard application.

Example 10.13. Using stochastic approximation to find a percentile

Since the p-th, $0 \leq p \leq 100$, percentile of the distribution of a random variable Y can be represented as the solution x^* of $E[I_{\{Y \leq x\}}] = p/100$, (10.103) can be used to find this percentile. To do so, specify x, sample y from its distribution and compute $I_{\{y \leq x\}}$.

For example, consider the problem of finding the 75th percentile of a standard Cauchy distribution[a]. In this case, (10.103) becomes

$$x^{n+1} = x^n - \tau_n \left(I_{\{y^n \leq x^n\}} - 0.75 \right).$$

Note that the quantity $I_{\{y^n \leq x^n\}} - 0.75$ only takes on the values 0.25 and -0.75.

Stochastic approximation is applied to estimate the 10th, 20th, 50th, 75th and 85th percentile of the Cauchy distribution with $\tau_n = 20/(200+n)$, $\tau_n = n^{-0.75}$ and $\tau_n = n^{-1}$. A total of 40 replicates of 10,000 iterates were generated using a common random number seed for each replicate and an initial value $x^1 = 0$.

Estimates were compared on the basis of within-replicate RMSE values (after deleting first 200 estimates) and visual inspection of plots. Table 10.18 summarizes the results. Observe that estimates using $\tau_n = n^{-1}$ (the sample average) produced the least accurate estimates for all percentiles so clearly it is advantageous to explore using stochastic approximation with learning rates other than $\tau_n = n^{-1}$. The reason that the running mean performed so poorly is that τ_n

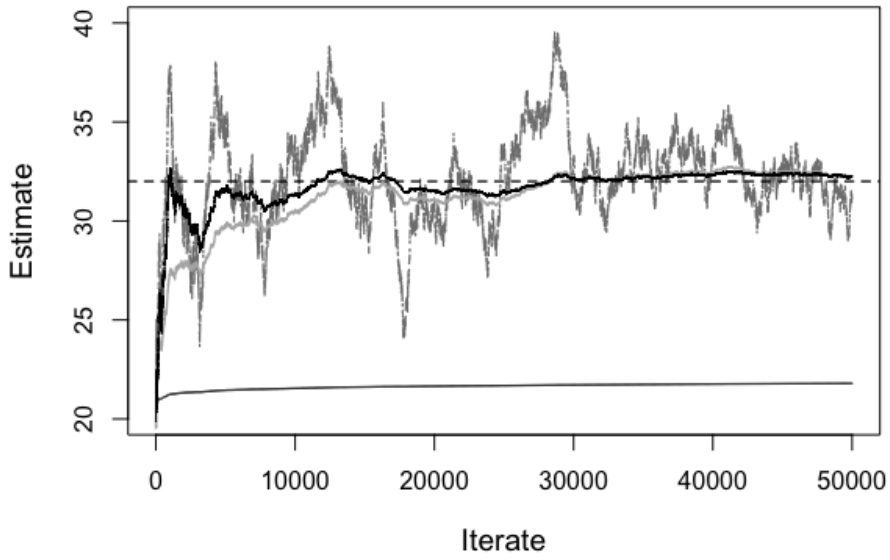


Figure 10.31: Graphical representation of sequence of estimates of x^* in Example 10.12. The horizontal dashed line gives the solution $x^* = 32$, the dark grey line, the iterates when $\tau_n = n^{-0.25}$, the line near the bottom of the figure corresponds to $\tau_n = n^{-1.0000001}$ and the grey and black lines that stabilize around 32 correspond to $\tau_n = 0.9n^{-0.5}$ and $\tau_n = 100/(400 + n)$, respectively.

decreased too quickly to allow learning.

The quality of estimates declined the further the percentile was from the mean 0. When accounting for both the mean and standard deviation, the ratio learning rate produced the most accurate estimates, especially in the tails of the distribution. Figure 10.32 shows a typical replicate comparing the three learning rates, substantiating the conclusions above.

[a]Recall that the Cauchy distribution has infinite variance so large values occur frequently. A standard Cauchy distribution has location parameter 0 and scale parameter 1.

10.11.3 Stochastic approximation and optimization

Stochastic approximation plays a key role in optimization where the objective is to minimize an expected loss function. It serves as a foundational concept for the online learning algorithms in this chapter and the next as follows.

Consider the problem of finding the value that minimizes a real-valued function of

Percentile	$\tau_n = \frac{20}{200+n}$	$\tau_n = n^{-0.75}$	$\tau_n = \frac{1}{n}$
10	3.43 (5.06)	5.02 (6.60)	17.40 (28.91)
20	0.64 (0.36)	0.57 (0.68)	5.58 (7.81)
50	0.29 (0.08)	0.13 (0.07)	0.72 (0.77)
75	0.45 (0.23)	0.38 (0.39)	5.43 (9.30)
85	1.08 (0.97)	3.82 (8.34)	12.42 (22.16)

Table 10.18: Mean (standard deviation) of within-replicate RMSE over 40 replicates of stochastic approximation estimates of percentiles of a Cauchy distribution using three learning rates.

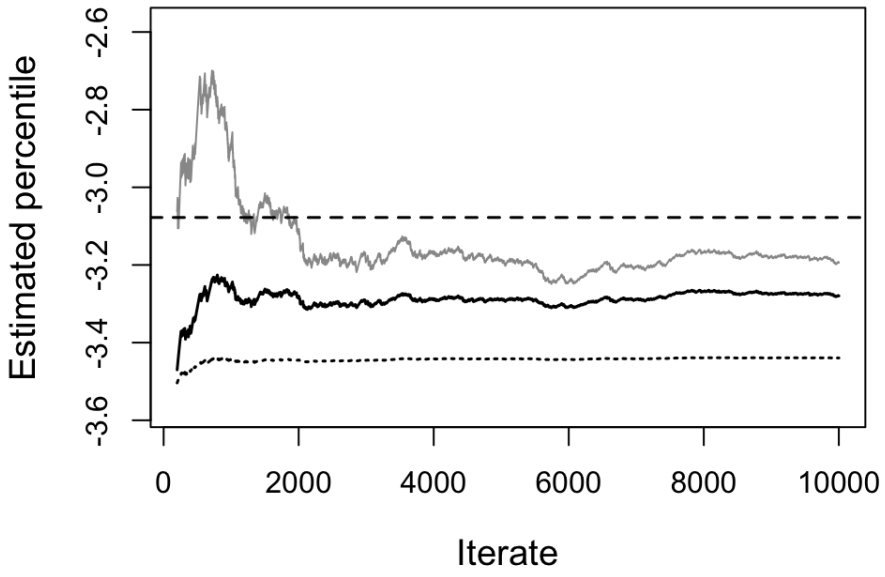


Figure 10.32: Plot of a single replicate of estimates for three learning rates of the 10th percentile of a Cauchy distribution in Example 10.13. The grey line corresponds to $\tau_n = 20/(200+n)$, the black line to $\tau_n = n^{-0.75}$ and the dotted line to $\tau_n = n^{-1}$. The horizontal line represents the true value of the 10th percentile.

a vector of variables $\boldsymbol{\beta}$, $G(\boldsymbol{\beta})$. That is

$$\boldsymbol{\beta}^* \in \arg\min_{\boldsymbol{\beta}} G(\boldsymbol{\beta}). \tag{10.106}$$

Gradient descent methods achieve this by solving the equation

$$\nabla_{\boldsymbol{\beta}} G(\boldsymbol{\beta}) = 0, \tag{10.107}$$

where $\nabla_{\boldsymbol{\beta}} G(\boldsymbol{\beta})$ denotes the gradient of $G(\boldsymbol{\beta})$. Implicit is the assumption that $G(\boldsymbol{\beta})$ is differentiable with respect to each component of $\boldsymbol{\beta}$. Under mild conditions, this minimum is unique but its possible that there may be multiple minima.

Gradient descent is an iterative procedure that chooses $\boldsymbol{\beta}$, observes $G(\boldsymbol{\beta})$, evaluates its gradient at $\boldsymbol{\beta}$ and updates $\boldsymbol{\beta}$ according to the recursion

$$\boldsymbol{\beta}' \leftarrow \boldsymbol{\beta} - \tau \nabla_{\boldsymbol{\beta}} G(\boldsymbol{\beta}), \tag{10.108}$$

where τ is the learning rate or step-size. This method should be well known to readers of this book.

Instead of observing $G(\boldsymbol{\beta})$ and computing its gradient, many of the methods in Chapters 10 and 11 are based on estimating the gradient using **sampled** data. That is, instead of computing the gradient directly, observations are generated from a distribution with **expectation** equal to $\nabla_{\boldsymbol{\beta}} G(\boldsymbol{\beta})$. Then stochastic approximation is applied using the sample gradient $\widehat{\nabla_{\boldsymbol{\beta}} G(\boldsymbol{\beta})}$ to update $\boldsymbol{\beta}$ according to

$$\boldsymbol{\beta}' \leftarrow \boldsymbol{\beta} - \tau \widehat{\nabla_{\boldsymbol{\beta}} G(\boldsymbol{\beta})} \tag{10.109}$$

This recursion is often referred to as *stochastic gradient descent (SGD)* but other definitions of SGD appear in the literature. This book will refer to recursion (10.109) as SGD. Note that when tracking iteration number (as in Appendix of Chapter 9) this expression can be written as

$$\boldsymbol{\beta}^{n+1} = \boldsymbol{\beta}^n - \tau_n \widehat{\nabla_{\boldsymbol{\beta}} G(\boldsymbol{\beta}^n)}.$$

Doing so allows specifying the explicit dependence of τ on n.

The following section illustrates this approach in a very important application.

Least squares regression

Least squares regression assumes a model for data of the form

$$y_j = f(\mathbf{x}_j; \boldsymbol{\beta}) + \epsilon_j \tag{10.110}$$

for $j = 1, \ldots, J$, where $\mathbf{x}_j$ denotes the j-th value of a $(I+1)$-dimensional vector of features[76], $\boldsymbol{\beta}$ denotes a $(I+1)$-dimensional vector of parameters, y_j denotes the j-th observation of the dependent variable and ϵ_j is an unobserved random disturbance

[76] Features are referred to as covariates or independent variables in statistical models.

with mean 0 and constant variance. The function $f(\mathbf{x}_j;\boldsymbol{\beta})$ denotes a **known** real-valued function that can be either linear or nonlinear in the parameters.

Least squares regression seeks parameter values $\boldsymbol{\beta}^*$ that minimize the *expected squared error loss function*

$$G(\boldsymbol{\beta}) := E[g(\boldsymbol{\beta})] = E[(Y - f(\mathbf{x};\boldsymbol{\beta}))^2], \tag{10.111}$$

where the expectation is with respect to the joint distribution of column vector $\mathbf{x}$ and scalar Y and the random variable

$$g(\boldsymbol{\beta}) := (Y - f(\mathbf{x};\boldsymbol{\beta}))^2.$$

This representation assumes that the random vector $\mathbf{x}$ is sampled from some distribution and Y is sampled from the conditional distribution of Y given $\mathbf{x}$ which has mean $f(\mathbf{x};\boldsymbol{\beta})$.

A standard approach to obtaining parameter estimates is to solve the equation

$$\nabla_{\boldsymbol{\beta}} G(\boldsymbol{\beta}) = \mathbf{0}.$$

However, doing this might be complicated because of the expectation inside the gradient. Instead, under mild regularity conditions,

$$\nabla_{\boldsymbol{\beta}} G(\boldsymbol{\beta}) = E[\nabla_{\boldsymbol{\beta}} g(\boldsymbol{\beta})] \tag{10.112}$$

so that

$$\nabla_{\boldsymbol{\beta}} G(\boldsymbol{\beta}) = E[-2(Y - f(\mathbf{x};\boldsymbol{\beta}))\nabla_{\boldsymbol{\beta}} f(\mathbf{x};\boldsymbol{\beta})], \tag{10.113}$$

which in the linear case, when $f(\mathbf{x};\boldsymbol{\beta}) = \mathbf{x}^\mathsf{T}\boldsymbol{\beta}$, simplifies to

$$E[\nabla_{\boldsymbol{\beta}} g(\boldsymbol{\beta})] = E[-2(Y - \mathbf{x}^\mathsf{T}\boldsymbol{\beta})\mathbf{x}]. \tag{10.114}$$

Note that in this expression the expected gradient is a column vector with the number of components equal to the dimension of $\boldsymbol{\beta}$.

The reason for this formality is to set things up to apply stochastic approximation. Instead of solving $\nabla_{\boldsymbol{\beta}} G(\boldsymbol{\beta}) = \mathbf{0}$ directly to minimize the expected loss in (10.111), samples are obtained from a *distribution* with mean $E[\nabla_{\boldsymbol{\beta}} g(\boldsymbol{\beta})]$ and the Robbins-Monro recursion is applied as follows.

Let $(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^N, y^N)$ denote a sequence of samples from the joint distribution of $\mathbf{x}$ and Y[77]. Then this sequence is used to estimate the gradient according to:

$$[\nabla_{\boldsymbol{\beta}} g(\boldsymbol{\beta})]^n := -2(y^n - f(\mathbf{x}^n;\boldsymbol{\beta}))\nabla_{\boldsymbol{\beta}} f(\mathbf{x}^n;\boldsymbol{\beta}) \tag{10.115}$$

where the superscript refers to the n-th sample of the gradient. When $f(\mathbf{x};\boldsymbol{\beta})$ is a linear function of the parameters this expression becomes

$$[\nabla_{\boldsymbol{\beta}} g(\boldsymbol{\beta})]^n = -2\big(y^n - (\mathbf{x}^n)^\mathsf{T}\boldsymbol{\beta}\big)\mathbf{x}^n. \tag{10.116}$$

[77] In the context of regression, subscripts indexed by j indicate data points, whereas superscripts indexed by n indicate a sequence of iterates.

In the linear case, the Robbins-Monro recursion becomes

$$\boldsymbol{\beta}' \leftarrow \boldsymbol{\beta} + \tau_n\big(y^n - (\mathbf{x}^n)^\mathsf{T}\boldsymbol{\beta}\big)\mathbf{x}^n, \tag{10.117}$$

where the multiplier 2 in the gradient expression has been absorbed into the learning rate and the negative sign in the gradient leads to a plus sign before τ_n.

Figure 10.33 summarizes the approach used to apply stochastic approximation when optimizing an expected value.

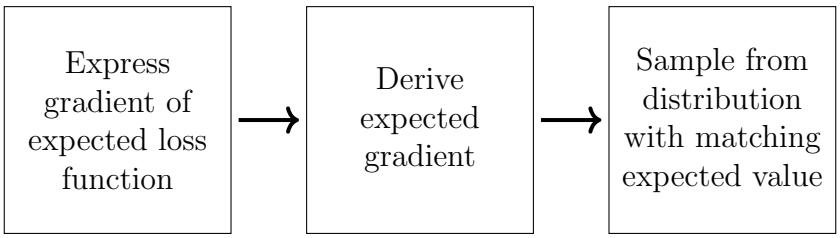


Figure 10.33: General approach for applying Robbins-Monro stochastic approximation to minimize expected loss.

A key theoretical result is that if the hypotheses of Theorem 10.3 hold for the gradient of the loss function, the sequence of iterates generated by stochastic gradient descent converges (in expectation) to a zero of the expected gradient.

Examples

Two applications of this approach follow.

Example 10.14. Using stochastic approximation to find a mean.
This example applies (10.117) to obtain an online recursion for estimating the mean μ of a random variable Y. In the additive error case, this can be viewed as a least squares regression problem in which $\boldsymbol{\beta}$ is a scalar equal to μ and $\mathbf{x}$ is a scalar equal to 1. This means that observations are generated according to

$$y^n = \mu + \epsilon^n$$

and the Robbins-Munro recursion becomes

$$\mu^{n+1} = \mu^n + \tau_n(y^n - \mu^n). \tag{10.118}$$

This recursion is the basis for the standard online formula for the sample mean

$\bar{y}^n$

$$\bar{y}^n = \frac{1}{n}\big((n-1)\bar{y}^{n-1} + y^n\big). \tag{10.119}$$

To see this, set $\tau_n = 1/n$ for $n = 1, 2, \ldots$, choose $\mu^1 = 0$ and apply (10.118) to obtain $\mu^2 = y^1$,

$$\mu^3 = \mu^2 + \frac{1}{2}(y^2 - \mu^2) = \frac{1}{2}(y^1 + y^2) = \bar{y}^2$$

and in general $\mu^{n+1} = \sum_{i=1}^n y^i/n = \bar{y}^n$. Then using (10.118)

$$\bar{y}^n = \mu^{n+1} = \mu^n + \frac{1}{n}(y^n - \mu^n) = \frac{n-1}{n}\mu^n - \frac{1}{n}y^n = \frac{1}{n}\big((n-1)\bar{y}^{n-1} + y^n\big),$$

which is the desired result. Observe that it requires storing only the current estimate of the mean and the number of observations. This approach has been referred to often in this chapter.

To illustrate this approach numerically, (10.118) is applied to estimate the mean of an exponential distribution. For illustrative purposes, learning rates $\tau_n = 20/(200+n)$, $n^{-0.75}$, and n^{-1} are compared. Simulations use a common random number seed and 5,000 iterates starting with $y^1 = 1$.

Figure (10.34) shows that the estimate using $\tau_n = n^{-1}$ (corresponding to the sample mean) is least variable and best estimates the true mean. Moreover its RMSE was 1.62 compared to 34.14 for the ratio learning rate and 10.10 when the learning rate was $n^{-0.75}$. Note that this conclusion appears to be valid even when observations are auto-correlated.

The following example illustrates the vector version of (10.117).

Example 10.15. Online regression

This example shows how to obtain estimates of regression parameters using (10.117). Consider a quadratic regression model of the form

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon.$$

To write it in vector form, define

$$\boldsymbol{\beta} = \begin{bmatrix}\beta_0\\ \beta_1\\ \beta_2\end{bmatrix} \text{ and } \mathbf{x} = \begin{bmatrix}1\\ x\\ x^2\end{bmatrix}$$

so that (10.117) becomes

$$\begin{bmatrix}\beta_0'\\ \beta_1'\\ \beta_2'\end{bmatrix} \leftarrow \begin{bmatrix}\beta_0\\ \beta_1\\ \beta_2\end{bmatrix} + \tau_n(y^n - \boldsymbol{\beta}^{\mathsf{T}}\mathbf{x}^n)\begin{bmatrix}1\\ x^n\\ (x^n)^2\end{bmatrix}. \tag{10.120}$$

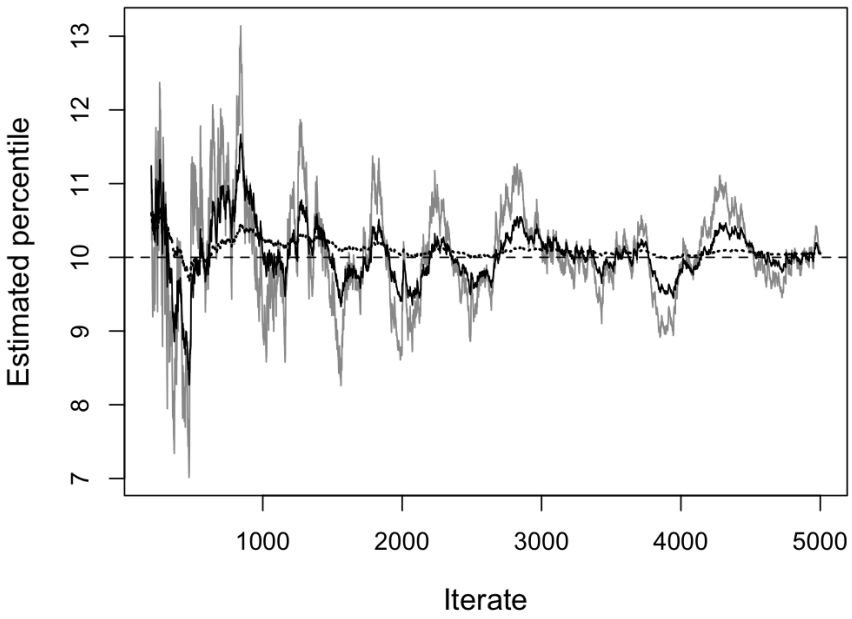


Figure 10.34: Results from a single replicate showing stochastic approximation estimates using (10.118) based on observations generated by an exponential distribution with mean 10. The gray line corresponds to $\tau_n = 20/(200+n)$, the solid black line to $\tau_n = n^{-0.75}$ and the dashed black line to $\tau_n = n^{-1}$.

Observe that each component of the vector $\mathbf{x}^n$ is multiplied by the same scalar quantity $\tau_n(y^n - \boldsymbol{\beta}^\mathsf{T}\mathbf{x}^n)$. Note also that is common practice to scale $\mathbf{x}^n$ (and sometimes y^n) to have mean zero and standard deviation one. This can be done by keeping online estimates of the mean and standard deviations of $\mathbf{x}^n$ and y^n.

As a numerical example suppose

$$y = -2 + 4x + 2x^2 + \epsilon$$

with ϵ sampled from a uniform distribution on $[-2, 2]$. Applying (10.120) with $\tau_n = 10/(80+n)$, each component of $\boldsymbol{\beta}$ initialized to zero and x randomly sampled from a normal distribution with mean 0 and standard deviation 1, the estimates from a typical replicate appear in Figure 10.35. They show convergence to true values. Note that when x was more variable, the estimates often diverged.

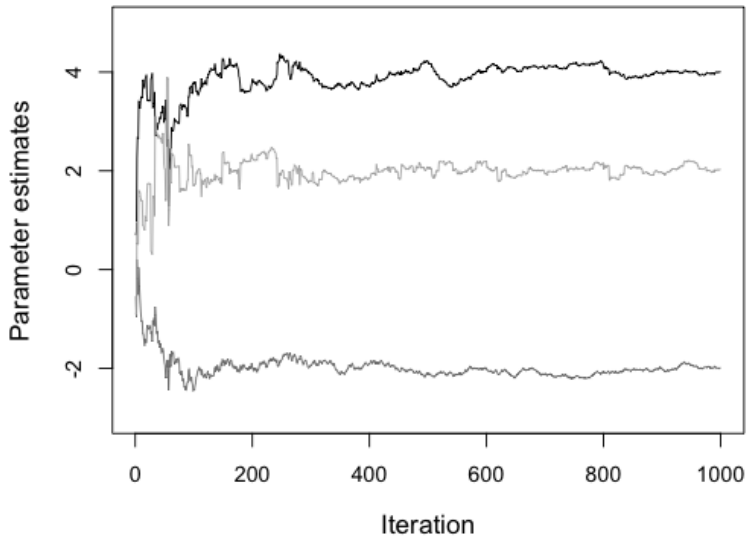


Figure 10.35: Parameter estimates from a single replicate in Example 10.15. The dark grey line corresponds to $\beta_0 = -2$, the black line to $\beta_1 = 4$ and the light grey line to $\beta_2 = 2$.

10.11.4 Offline TD(γ): Technical details

This technical section derives a representation for TD(γ) that is fundamental to the use of eligibility traces. The motivation is that value function estimation may be enhanced by using a weighted average of rollouts of different lengths.

The following lemma is fundamental to deriving the forward looking expression (10.34). Is proof is straightforward and involves interchanging the order of summation.

Lemma 10.1. Let $0 \leq \gamma \leq 1$ and r^k and v^k for $k = 0, 1, \ldots$ denote scalars for which the summations in the following expressions are convergent. Then

$$(1-\gamma)\sum_{m=1}^{\infty}\gamma^{m-1}\left(\sum_{k=0}^{m-1}r^k + v^m\right) = v^0 + \sum_{m=0}^{\infty}\gamma^m\left(r^m + v^{m+1} - v^m\right). \quad (10.121)$$

Proof. Reverse the order of summation in the first term on the left-hand side of (10.121) to obtain

$$(1-\gamma)\sum_{m=1}^{\infty}\left(\gamma^{m-1}\sum_{k=0}^{m-1}r^k\right) = (1-\gamma)\sum_{m=0}^{\infty}r^m\sum_{k=m}^{\infty}\gamma^k = \sum_{m=0}^{\infty}\gamma^m r^m,$$

where the last equality follows by using the standard formula for the sum of a geometric series.

Next consider the second term. Simple algebra shows

$$(1-\gamma)\sum_{m=1}^{\infty}\gamma^{m-1}v^m = \sum_{m=1}^{\infty}(\gamma^{m-1}-\gamma^m)v^m = v^1 + \sum_{m=1}^{\infty}\gamma^m(v^{m+1}-v^m)$$
$$= v^0 + \sum_{m=0}^{\infty}\gamma^m(v^{m+1}-v^m),$$

where the last equality comes from adding and subtracting v^0. Combining these expressions and rearranging terms yields (10.121). □

To apply this result to expression (10.33) expand it as

$$v(s^n) = (1-\gamma)E^{d^\infty}\left[\sum_{m=1}^{\infty}\gamma^{m-1}\left(\sum_{k=0}^{m-1}r(X_{n+k},Y_{n+k},X_{n+k+1}) + v(X_{n+m})\right)\middle| X_n = s^n\right],$$

set

$$r^k = E^{d^\infty}\left[r(X_{n+k},Y_{n+k},X_{n+k+1})\middle| X_n = s^n\right]$$

and

$$v^m = E^{d^\infty}\left[v(X_{n+m})\middle| X_n = s^n\right].$$

Applying the above lemma shows that:

$$v(s^n) = v(s^n) + E^{d^\infty}\left[\sum_{m=0}^{\infty}\gamma^m\big(r(X_{n+m},Y_{n+m},X_{n+m+1}) \right. \tag{10.122}$$
$$\left. + v(X_{n+m+1}) - v(X_{n+m})\big)\middle| X_n = s^n\right]. \tag{10.123}$$

Offline TD(γ) applies stochastic approximation to (10.122) as follows. Given an observed trajectory $(s^n, a^n, s^{n+1}, \ldots)$, stochastic approximation results in the recursion

$$v(s^n) \leftarrow v(s^n) + \tau\sum_{m=0}^{\infty}\gamma^m\Big(r(s^{n+m},a^{n+m},s^{n+m+1}) + v(s^{n+m+1}) - v(s^{n+m})\Big)$$
$$= v(s^n) + \tau\sum_{m=0}^{\infty}\gamma^m\delta_{n+m}$$

where

$$\delta_k := r(s^k,a^k,s^{k+1}) + v(s^{k+1}) - v(s^k).$$

Then changing variables for the summation index yields the closed form representation for an offline implementation:

$$v(s^n) \leftarrow v(s^n) + \tau \sum_{m=n}^{\infty} \gamma^{m-n} \delta_m \tag{10.124}$$

in which τ denotes a learning rate.

This representation provides the basis for the theoretical analysis of TD(γ) in Bertsekas and Tsitsiklis [1996], generalization to function approximation and provides motivation for an online implementation. To make the argument above rigorous in an episodic model requires that the decision rule be transient[78], that is, it generates a process that terminates in a finite time with probability one.

A similar representation is also valid in discounted models with discount factor λ in which case (10.124) becomes

$$v(s^n) \leftarrow v(s^n) + \tau \sum_{m=n}^{\infty} (\lambda\gamma)^{m-n} \delta_m. \tag{10.125}$$

Derivation of this expression is left as an exercise.

[78] Or proper in a stochastic shortest path problem.

Bibliographic Remarks

This chapter aimed to provide a user-friendly and practical introduction to simulation-based methods applicable to both model-free and model-based environments, with a focus on tabular models. It is not intended to be comprehensive or fully up to date, as this area of research and application is rapidly advancing.

Gosavi [2015] provides an accessible and informative overview of simulation methods for Markov decision processes, well-suited to readers of this book. His insights and comments significantly improved both the results and exposition in this chapter, particularly regarding the use of importance ratios, stochastic trace correction (STC), log-based learning rates, and choices for ϵ_n in ϵ-greedy exploration. In addition, our discussions with him regarding Relative Q-learning in average reward models enabled us to solve that version of the queuing control problem.

Noteworthy is the book Sutton and Barto [2018], which presents a reinforcement learning perspective on this material in a clear and engaging style. It served as an important source for this chapter and the next. For more rigorous and in-depth approaches grounded in Markov decision process and control theory, see Bertsekas and Tsitsiklis [1996], Bertsekas [2012], and Meyn [2022], which were our primary references for the underlying theory. In addition, there are a wealth of teaching notes and lectures available online to support further learning.

The computational results in this chapter highlight the importance of distinguishing among data generation approaches. Xiao et al. [2022] addresses this issue from a complexity-theoretic perspective, which motivates the discussion in Section 10.1.1.

Temporal-difference (TD) learning, which originated in the seminal work of Sutton (see references in Sutton and Barto [2018]), is based on applying stochastic approximation to policy evaluation, which is referred to as *prediction* in that context. Our discussion of learning rates draws on Powell [2007] and other sources.

The analysis of TD(γ) follows Sutton and Barto [2018], Tsitsiklis and Roy [1997], and especially Bertsekas and Tsitsiklis [1996], pp. 195–197. TD(γ) originated in foundational papers by Sutton cited in these works.

Solving Markov decision processes in a model-free online environment traces back to Watkins' Ph.D. thesis and the seminal paper Watkins and Dayan [1992] on Q-learning. That work includes a convergence proof for Q-learning under a discounted model when values are stored in a lookup table and all state-action pairs are sampled infinitely often. Singh et al. [2000] provides an in-depth analysis of single-step SARSA.

Average-reward temporal-difference learning and Q-learning are based on Tsitsiklis and Roy [1999], Mahadevan [1996], and pp. 548–551 in Bertsekas [2012]. Our treatment of relative value iteration follows Abounadi et al. [2001].

We believe the material on simulated policy iteration is original. It was partially inspired by Buşoniu et al. [2010], which describes a more general approach that incorporates function approximation. To our knowledge, the introduction of bounds for hybrid policy iteration is new.

The description of stochastic approximation in the appendix draws on the seminal paper by Robbins and Monro [1951] and the follow-up by Blum [1954]. More modern treatments appear in Bertsekas [2012], Powell [2007], and Meyn [2022]. The example using stochastic approximation to estimate percentiles is due to Ribes et al. [2019]. Sugiyama [1994] provides an excellent overview of stochastic approximation with several detailed examples. That work also drew our attention to Bellman's prescient remark quoted at the start of Section 10.11.2.

The next chapter extends these methods to models with function approximation and contains many additional references.

Exercises

1. The exercise compares data generation mechanisms in Section 10.1.1 in the context of the queuing service rate control model analyzed in Section 10.9.

   (a) Compare and contrast trajectory-based sampling and state-based sampling by developing code snippets for evaluating a fixed randomized stationary policy.

   (b) Compare and contrast all four sampling methods by developing code snippets for finding an optimal policy using Q-learning.

2. Repeat the Q-learning analysis of the newsvendor model for other prices, demand distributions, learning rates and choices of η_n.

3. Extend Algorithm 10.2 to the every-visit variant and apply it to evaluate the policy in the Gridworld model in Section 10.3.1. Compare its computational effort to starting-state and first-visit Monte Carlo.

4. Evaluate the policy in Example 10.1 using Monte Carlo methods and TD(0). Note the challenges encountered in obtaining accurate estimates.

5. Show that the operator defined by the right hand side of (10.17) is a contraction mapping and that the recursion converges to the same fixed point for every choice of τ.

6. Extend the study of Q-learning and SARSA in Gridworld (Example 10.8) by considering other exploration methods, varying the number of episodes in each replicate and model parameters, especially p.

7. Repeat the analysis in the example in Section 10.7.2 in which the index of the learning rate and exploration method is the number of iterations as opposed to the count of state-action pair or state, visits respectively. How do these changes impact results?

8. This example illustrates the difference between SARSA and Q-learning. Consider the snippet of a deterministic Markov decision process depicted in Figure 10.36. In state s there are two actions, a and b and in state u there are two actions $\bar{a}$

and $\bar{b}$. Choosing action a in state s yields a reward of 1 and a transition to state u. All other rewards are immaterial.

Suppose that $q(s,a) = 0,\ q(u,\bar{a}) = 5,\ q(u,\bar{b}) = 3$ and exploration resulted in choosing action $\bar{b}$ in state u. Show that the Q-learning and SARSA updates of $q(s,a)$ differ.

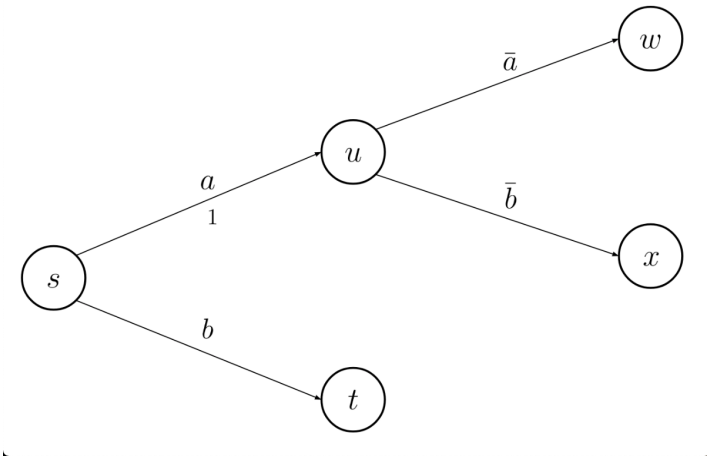


Figure 10.36: Graphical representation of model in Exercise 8.

9. Consider the deterministic version of Gridworld as in Example 10.7 except assume that the robot can move "up", "down", "right" and "left". Apply Q-learning and SARSA to find good policies using softmax and ϵ-greedy exploration. Compare these methods on the basis of the mean and standard deviation of the total reward per replicate.

10. Develop simulation-based policy iteration algorithms for an episodic model and apply them to the Gridworld model. Summarize your conclusions verbally and graphically.

11. Repeat the analysis in Example 10.8 with a small fraction of episodes starting in cell 4 in which the state-action frequencies were inaccurately estimated when beginning all episodes in cell 13. How do your results differ from those in the example? Also explore the use of softmax exploration instead of ϵ-greedy exploration.

12. **Finite Horizon Q-learning:** Develop a Q-learning algorithm for a **finite horizon** model. Apply it to solve the revenue management problem analyzed in Section 4.3.2.

13. **Cliff walking**[79]**.** In this problem on a 4×12 grid, graphically represented in Figure 10.37, a robot must traverse the grid by moving from the origin "O" in

[79] This colorful example was proposed by Sutton and Barto [2018]. It has a similar structure to our Gridworld example of a coffee delivering robot.

cell (1,1) in the lower left hand corner of the grid to the destination "D" in cell (1,12) in the lower right hand corner of the grid without falling off the cliff in cells (1,2) to (1,11). The robot incurs a cost of 1 unit for each step it takes and a penalty of 100 if it falls off the cliff. If it falls off the cliff it returns to the origin and starts again.

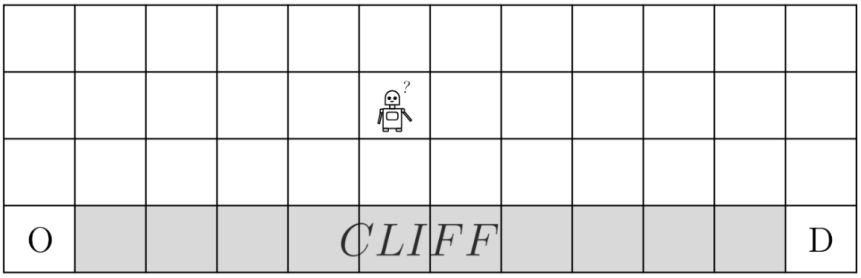


Figure 10.37: The cliff walking problem

The robot moves deterministically and can choose to move "up", "down", "left" or "right" in every cell. If it tries to move in a direction which is not possible, it incurs a cost of one unit and remains in that cell.

The goal is to learn an optimal path through this grid. Assume the robot does not know the layout and must learn it by ϵ-greedy exploration.

(a) Formulate this problem as an episodic model in which an episode starts with the robot at the origin and concludes when it reaches its destination.

(b) Find (and state) an optimal policy using Q-learning and SARSA with a learning rate $\tau = 0.1$, ϵ-greedy exploration parameter $\epsilon = 0.1$ and a discount rate of 1.

(c) Accumulate the total reward per episode for Q-learning and SARSA for the first 500 episodes of each. Compare them graphically.

(d) Investigate the impact of τ and ϵ on algorithmic performance.

14. Show that using TD(γ) for policy evaluation is equivalent to temporal differencing when $\gamma = 0$ and Monte Carlo estimation when $\gamma = 1$. Note in the latter case some care has to go into choosing the trace, specifying τ_n, and distinguishing the various forms of Monte Carlo estimation.

15. Analyze the model in Example 10.9 using the following two variants of hybrid policy iteration:

(a) Estimate $r(s,a,j)$ and $p(j|s,a)$ assuming a model-free environment and then use them to estimate $q^{d^\infty}(s,a)$ as in the example.

(b) Estimate $\sum_{j\in S} p(j|s,a)\big(r(s,a,j) + \lambda\hat{v}(j)\big)$ using Monte Carlo simulation after estimating $\hat{v}(\cdot)$.

Apply the bounds in (10.89) and stopping criterion implicit in (10.90) and summarize your observations.

16. Show either theoretically or using simulation that in the queuing control model, the steady state probabilities when the service rate and arrival rates are identical (for example, each equal to 0.2) is a symmetric random walk on the state space and that the limiting probabilities are equal for each state. Find the limiting probabilities when the service rate exceeds the arrival rate for example when action $a_2 = 0.4$ or 0.6.

17. In the context of the queuing control model in Section 10.9, investigate the convergence of a variant of Algorithm 10.14 that smooths the policy value function prior to exact derivation using (10.86). Compare the effects of smoothing both the policy value function and the q-function to smoothing only the policy value function.

18. **A large queuing service rate control model.** Use Q-learning and the policy iteration algorithms of Section 10.8 to analyze the larger version of the discounted queuing service rate control model previously solved with (deterministic) policy iteration in Section 5.11.4. In it the queue capacity is 5,000, $A_s = \{0.2, 0.3, \ldots, 0.7\}$, $b = 0.2$, $m(a_k) = 2k^3$, $f(s) = s^2$ and the discount rate equals 0.4. A low discount rate was chosen to obtain an interesting optimal policy.

    Use Relative Q-learning to solve an average reward version of this problem instance.

19. **Inventory control with hidden Markov demand.** Consider the following inventory model. At the end of each day, the inventory manager can order j units from a supplier that arrive the next morning at a cost of c per item plus a fixed ordering cost of k per order. When the demand d on any day exceeds the number of units on hand, the excess demand is unfilled and lost. If demand is less than or equal the number of units received, the system receives a revenue r for each unit sold. Unsold inventory is carried over to the next day at a cost of h per unit.

    Suppose that daily demand follows a binomial distribution with parameters D and p and moreover p takes on the values p_L and p_H, which vary from day to day according to a Markov chain with transition probability matrix

$$P = \begin{bmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \end{bmatrix},$$

where row 1 corresponds to p_L. Suppose that $c = 10, K = 5, r = 20, h = 3, p_H = 0.5, p_L = 0.2$ and $D = 20$.

(a) Develop a simulation model of this system and use it to evaluate the policy that orders 5 units every day.

(b) Find a good policy for the infinite horizon discounted model with $\lambda = 0.95$.

20. Consider the model in Exercise 1 of Chapter 2. In performing the following calculations investigate the effect of learning rate sequence $\tau_n, n = 1, 2, , \ldots$ and sample size on the accuracy of policy evaluation. Compare results on the basis of RMSE and visual inspection of the policy value functions.

    In addition, when optimizing, investigate the impact of exploration method. Summarize results in terms of the similarity of the policies to the optimum, the true values of the policies, and appropriate RMSE measures.

    (a) Use temporal differencing, TD(γ), and Monte Carlo with truncated discounted rewards and geometric sampling to estimate the infinite horizon discounted reward with $\lambda = 0.9$ for the following policies:

        i. The deterministic stationary policy that uses $d(s_i) = a_{i,1}$ in state s_i for $i = 1, 2, 3$.

        ii. The randomized stationary policy that uses action $a_{i,1}$ with probability $e^{-0.5i}$ and action $a_{i,2}$ with probability $1 - e^{-0.5i}$ in state s_i for $i = 1, 2, 3$.

    (b) Evaluate the average reward and bias of these policies using Monte Carlo methods and temporal differencing.

    (c) Develop and apply a TD(γ) algorithm for an average reward model.

    (d) Use Q-learning, SARSA, hybrid policy iteration and online policy iteration to find an optimal policy for a discounted version of the model. Consider both $\lambda = 0.9$ and $\lambda = 0.999$. Comment on the effect of λ on the execution of the algorithm.

    (e) Use Q-learning and Relative Q-learning to find an optimal policy for the average reward version of this model.

21. Consider the admission control queuing model model of Section 3.4.2 with reward $R = 20$, holding cost $f(j) = 1.05j$, service probability $w = 0.45$, and arrival probability $b = 0.5$. Truncate the state space at $M = 100$.

    (a) Write down recursions for the state-action value function assuming:

        i. The holding cost is incurred before an arrival or service completion.

        ii. The holding cost is incurred after an arrival or service completion.

    (b) Find an optimal policy for a discounted version of this model with $\lambda = 0.95$ and investigate its sensitivity to the problem data.

(c) Use Q-learning, hybrid policy iteration and online policy iteration to find a good policy for this model using simulation.

(d) Investigate the impact of truncation on the optimal policy and its gain by varying the truncation level M.

(e) Use Q-learning and Relative Q-learning to find good policies for an average reward variant of this model.

22. Develop simulation-based policy iteration algorithms for an average reward model and use it to solve the queuing service rate control model. Compare results to those obtained using Relative Q-learning in this chapter.

23. Develop a Q-learning algorithm for a finite horizon model and apply it to the revenue management problem in Section 3.3. Note that this requires defining state-action value functions that vary with decision epoch.

    Based on these observations, what can you conclude about the challenges of using trajectory-based sampling for value function estimation and optimization?

24. Use the recursion (10.103) to find a root of the function $f(x) = 1/(1+e^{-0.5x})-0.5$ based on observing $f(x) + \epsilon$ where ϵ is normally distributed with mean 0 and standard deviation 0.1. Investigate the variability and accuracy of estimates as you vary the learning rate τ_n and the number of observations.

25. Derive (10.125) by modifying the argument at the start of Section 10.11.4 to include a discount factor.

26. **Queuing with a budget constraint.** Formulate and analyze the queuing service rate control model in which an episode ends when a finite budget has been expended and the objective is to maximize *throughput*, that is the number of customers served. Assuming the same parameter values as in Section 10.9, investigate the impact of the budget constraint level on your results.