Chapter 7

n-step Bootstrapping

In this chapter we unify the Monte Carlo (MC) methods and the one-step temporal-difference (TD) methods presented in the previous two chapters. Neither MC methods nor one-step TD methods are always the best. In this chapter we present *n-step TD methods* that generalize both methods so that one can shift from one to the other smoothly as needed to meet the demands of a particular task. n-step methods span a spectrum with MC methods at one end and one-step TD methods at the other. The best methods are often intermediate between the two extremes.

Another way of looking at the benefits of n-step methods is that they free you from the tyranny of the time step. With one-step TD methods the same time step determines how often the action can be changed and the time interval over which bootstrapping is done. In many applications one wants to be able to update the action very fast to take into account anything that has changed, but bootstrapping works best if it is over a length of time in which a significant and recognizable state change has occurred. With one-step TD methods, these time intervals are the same, and so a compromise must be made. n-step methods enable bootstrapping to occur over multiple steps, freeing us from the tyranny of the single time step.

The idea of n-step methods is usually used as an introduction to the algorithmic idea of *eligibility traces* (Chapter 12), which enable bootstrapping over multiple time intervals simultaneously. Here we instead consider the n-step bootstrapping idea on its own, postponing the treatment of eligibility-trace mechanisms until later. This allows us to separate the issues better, dealing with as many of them as possible in the simpler n-step setting.

As usual, we first consider the prediction problem and then the control problem. That is, we first consider how n-step methods can help in predicting returns as a function of state for a fixed policy (i.e., in estimating v_π). Then we extend the ideas to action values and control methods.

7.1 *n*-step TD Prediction

What is the space of methods lying between Monte Carlo and TD methods? Consider estimating v_π from sample episodes generated using π. Monte Carlo methods perform an update for each state based on the entire sequence of observed rewards from that state until the end of the episode. The update of one-step TD methods, on the other hand, is based on just the one next reward, bootstrapping from the value of the state one step later as a proxy for the remaining rewards. One kind of intermediate method, then, would perform an update based on an intermediate number of rewards: more than one, but less than all of them until termination. For example, a two-step update would be based on the first two rewards and the estimated value of the state two steps later. Similarly, we could have three-step updates, four-step updates, and so on. Figure 7.1 shows the backup diagrams of the spectrum of *n-step updates* for v_π, with the one-step TD update on the left and the up-until-termination Monte Carlo update on the right.

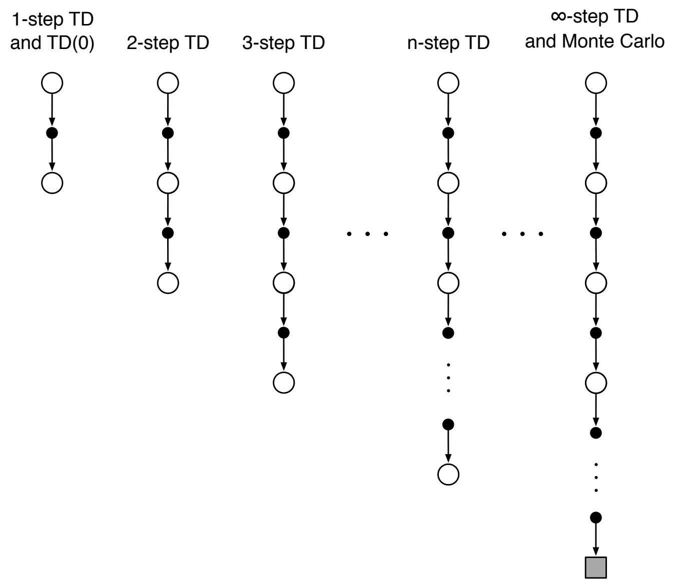


Figure 7.1: The backup diagrams of n-step methods. These methods form a spectrum ranging from one-step TD methods to Monte Carlo methods.

The methods that use n-step updates are still TD methods because they still change an earlier estimate based on how it differs from a later estimate. Now the later estimate is not one step later, but n steps later. Methods in which the temporal difference extends over n steps are called *n-step TD methods*. The TD methods introduced in the previous chapter all used one-step updates, which is why we called them one-step TD methods.

More formally, consider the update of the estimated value of state S_t as a result of the state–reward sequence, $S_t, R_{t+1}, S_{t+1}, R_{t+2}, \dots, R_T, S_T$ (omitting the actions). We know that in Monte Carlo updates the estimate of $v_\pi(S_t)$ is updated in the direction of the complete return:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T,$$

where T is the last time step of the episode. Let us call this quantity the *target* of the update. Whereas in Monte Carlo updates the target is the return, in one-step updates the target is the first reward plus the discounted estimated value of the next state, which we call the *one-step return*:

$$G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1}),$$

where $V_t : \mathcal{S} \rightarrow \mathbb{R}$ here is the estimate at time t of v_π. The subscripts on $G_{t:t+1}$ indicate that it is a truncated return for time t using rewards up until time $t+1$, with the discounted estimate $\gamma V_t(S_{t+1})$ taking the place of the other terms $\gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$ of the full return, as discussed in the previous chapter. Our point now is that this idea makes just as much sense after two steps as it does after one. The target for a two-step update is the *two-step return*:

$$G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2}),$$

where now $\gamma^2 V_{t+1}(S_{t+2})$ corrects for the absence of the terms $\gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots + \gamma^{T-t-1} R_T$. Similarly, the target for an arbitrary n-step update is the *n-step return*:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}), \tag{7.1}$$

for all n, t such that $n \geq 1$ and $0 \leq t < T - n$. All n-step returns can be considered approximations to the full return, truncated after n steps and then corrected for the remaining missing terms by $V_{t+n-1}(S_{t+n})$. If $t + n \geq T$ (if the n-step return extends to or beyond termination), then all the missing terms are taken as zero, and the n-step return defined to be equal to the ordinary full return ($G_{t:t+n} \doteq G_t$ if $t + n \geq T$).

Note that n-step returns for $n > 1$ involve future rewards and states that are not available at the time of transition from t to $t + 1$. No real algorithm can use the n-step return until after it has seen R_{t+n} and computed V_{t+n-1}. The first time these are available is $t + n$. The natural state-value learning algorithm for using n-step returns is thus

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \big[G_{t:t+n} - V_{t+n-1}(S_t)\big], \qquad 0 \leq t < T, \tag{7.2}$$

while the values of all other states remain unchanged: $V_{t+n}(s) = V_{t+n-1}(s)$, for all $s \neq S_t$. We call this algorithm *n-step TD*. Note that no changes at all are made during the first $n - 1$ steps of each episode. To make up for that, an equal number of additional updates are made at the end of the episode, after termination and before starting the next episode. Complete pseudocode is given in the box on the next page.

Exercise 7.1 In Chapter 6 we noted that the Monte Carlo error can be written as the sum of TD errors (6.6) if the value estimates don't change from step to step. Show that the n-step error used in (7.2) can also be written as a sum of TD errors (again if the value estimates don't change) generalizing the earlier result. □

Exercise 7.2 (programming) With an n-step method, the value estimates *do* change from step to step, so an algorithm that used the sum of TD errors (see previous exercise) in

n-step TD for estimating $V \approx v_\pi$

```
Input: a policy $\pi$
Algorithm parameters: step size $\alpha \in (0,1]$, a positive integer $n$
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for $S_t$ and $R_t$) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$ :
  |  If $t < T$, then:
  |      Take an action according to $\pi(\cdot|S_t)$
  |      Observe and store the next reward as $R_{t+1}$ and the next state as $S_{t+1}$
  |      If $S_{t+1}$ is terminal, then $T \leftarrow t+1$
  |  $\tau \leftarrow t - n + 1$   ($\tau$ is the time whose state's estimate is being updated)
  |  If $\tau \geq 0$:
  |      $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  |      If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$          $(G_{\tau:\tau+n})$
  |      $V(S_\tau) \leftarrow V(S_\tau) + \alpha\,[G - V(S_\tau)]$
  Until $\tau = T - 1$
```

place of the error in (7.2) would actually be a slightly different algorithm. Would it be a better algorithm or a worse one? Devise and program a small experiment to answer this question empirically. □

The n-step return uses the value function V_{t+n-1} to correct for the missing rewards beyond R_{t+n}. An important property of n-step returns is that their expectation is guaranteed to be a better estimate of v_π than V_{t+n-1} is, in a worst-state sense. That is, the worst error of the expected n-step return is guaranteed to be less than or equal to γ^n times the worst error under V_{t+n-1}:

$$\max_s \Big|\mathbb{E}_\pi[G_{t:t+n}|S_t = s] - v_\pi(s)\Big| \;\leq\; \gamma^n \max_s \Big|V_{t+n-1}(s) - v_\pi(s)\Big|, \tag{7.3}$$

for all $n \geq 1$. This is called the *error reduction property* of n-step returns. Because of the error reduction property, one can show formally that all n-step TD methods converge to the correct predictions under appropriate technical conditions. The n-step TD methods thus form a family of sound methods, with one-step TD methods and Monte Carlo methods as extreme members.

Example 7.1: n-step TD Methods on the Random Walk Consider using n-step TD methods on the 5-state random walk task described in Example 6.2 (page 125). Suppose the first episode progressed directly from the center state, C, to the right, through D and E, and then terminated on the right with a return of 1. Recall that the estimated values of all the states started at an intermediate value, $V(s) = 0.5$. As a result of this experience, a one-step method would change only the estimate for the last state,

$V(\mathsf{E})$, which would be incremented toward 1, the observed return. A two-step method, on the other hand, would increment the values of the two states preceding termination: $V(\mathsf{D})$ and $V(\mathsf{E})$ both would be incremented toward 1. A three-step method, or any n-step method for $n > 2$, would increment the values of all three of the visited states toward 1, all by the same amount.

Which value of n is better? Figure 7.2 shows the results of a simple empirical test for a larger random walk process, with 19 states instead of 5 (and with a -1 outcome on the left, all values initialized to 0), which we use as a running example in this chapter. Results are shown for n-step TD methods with a range of values for n and α. The performance measure for each parameter setting, shown on the vertical axis, is the square-root of the average squared error between the predictions at the end of the episode for the 19 states and their true values, then averaged over the first 10 episodes and 100 repetitions of the whole experiment (the same sets of walks were used for all parameter settings). Note that methods with an intermediate value of n worked best. This illustrates how the generalization of TD and Monte Carlo methods to n-step methods can potentially perform better than either of the two extreme methods.

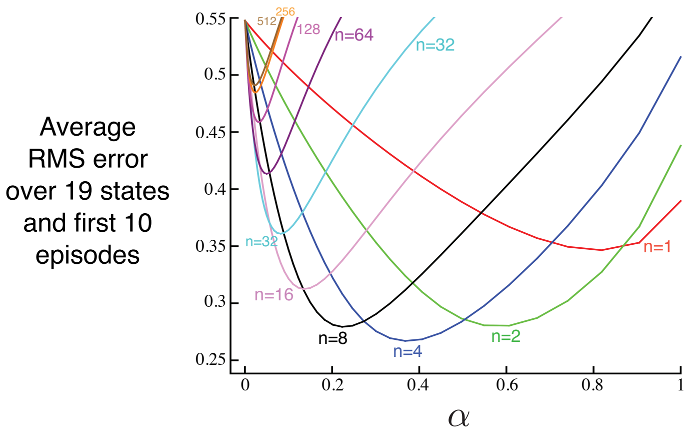


Figure 7.2: Performance of n-step TD methods as a function of α, for various values of n, on a 19-state random walk task (Example 7.1). ■

Exercise 7.3 Why do you think a larger random walk task (19 states instead of 5) was used in the examples of this chapter? Would a smaller walk have shifted the advantage to a different value of n? How about the change in left-side outcome from 0 to -1 made in the larger walk? Do you think that made any difference in the best value of n? □

7.2 *n*-step Sarsa

How can n-step methods be used not just for prediction, but for control? In this section we show how n-step methods can be combined with Sarsa in a straightforward way to produce an on-policy TD control method. The n-step version of Sarsa we call n-step Sarsa, and the original version presented in the previous chapter we henceforth call *one-step Sarsa*, or *Sarsa(0)*.

The main idea is to simply switch states for actions (state–action pairs) and then use an ε-greedy policy. The backup diagrams for n-step Sarsa (shown in Figure 7.3), like those of n-step TD (Figure 7.1), are strings of alternating states and actions, except that the Sarsa ones all start and end with an action rather a state. We redefine n-step returns (update targets) in terms of estimated action values:

$$G_{t:t+n} \doteq R_{t+1}+\gamma R_{t+2}+\cdots+\gamma^{n-1}R_{t+n}+\gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}), \quad n \geq 1, 0 \leq t < T-n, \tag{7.4}$$

with $G_{t:t+n} \doteq G_t$ if $t+n \geq T$. The natural algorithm is then

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha\left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)\right], \qquad 0 \leq t < T, \tag{7.5}$$

while the values of all other states remain unchanged: $Q_{t+n}(s,a) = Q_{t+n-1}(s,a)$, for all s, a such that $s \neq S_t$ or $a \neq A_t$. This is the algorithm we call *n-step Sarsa*. Pseudocode is shown in the box on the next page, and an example of why it can speed up learning compared to one-step methods is given in Figure 7.4.

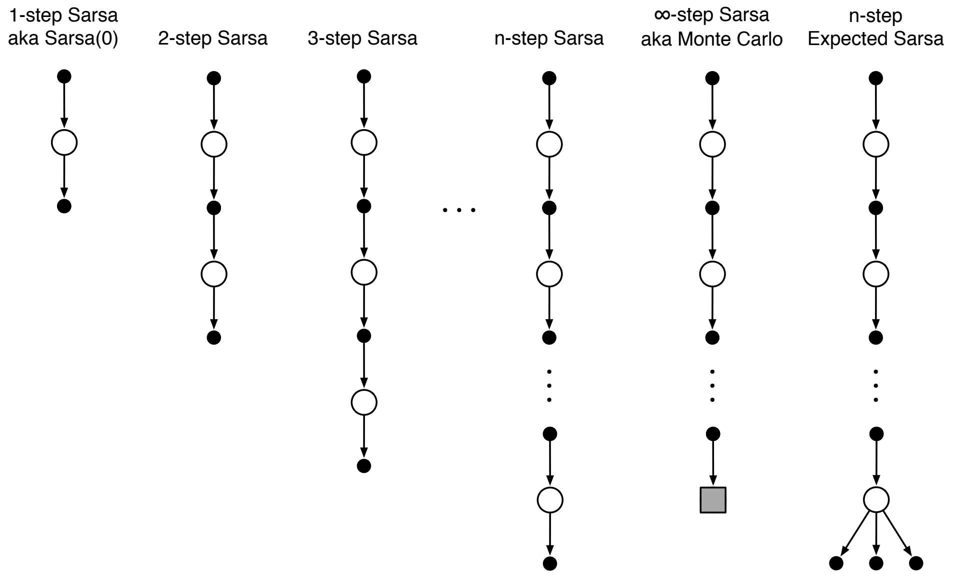


Figure 7.3: The backup diagrams for the spectrum of n-step methods for state–action values. They range from the one-step update of Sarsa(0) to the up-until-termination update of the Monte Carlo method. In between are the n-step updates, based on n steps of real rewards and the estimated value of the nth next state–action pair, all appropriately discounted. On the far right is the backup diagram for n-step Expected Sarsa.

n-step Sarsa for estimating $Q \approx q_*$ or q_π

```
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $\pi$ to be $\varepsilon$-greedy with respect to $Q$, or to a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer $n$
All store and access operations (for $S_t$, $A_t$, and $R_t$) can take their index mod $n+1$

Loop for each episode:
    Initialize and store $S_0 \neq$ terminal
    Select and store an action $A_0 \sim \pi(\cdot|S_0)$
    $T \leftarrow \infty$
    Loop for $t = 0, 1, 2, \dots$ :
    |   If $t < T$, then:
    |       Take action $A_t$
    |       Observe and store the next reward as $R_{t+1}$ and the next state as $S_{t+1}$
    |       If $S_{t+1}$ is terminal, then:
    |           $T \leftarrow t+1$
    |       else:
    |           Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
    |   $\tau \leftarrow t - n + 1$    ($\tau$ is the time whose estimate is being updated)
    |   If $\tau \geq 0$:
    |       $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
    |       If $\tau + n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$        $(G_{\tau:\tau+n})$
    |       $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha [G - Q(S_\tau, A_\tau)]$
    |       If $\pi$ is being learned, then ensure that $\pi(\cdot|S_\tau)$ is $\varepsilon$-greedy wrt $Q$
    Until $\tau = T - 1$
```

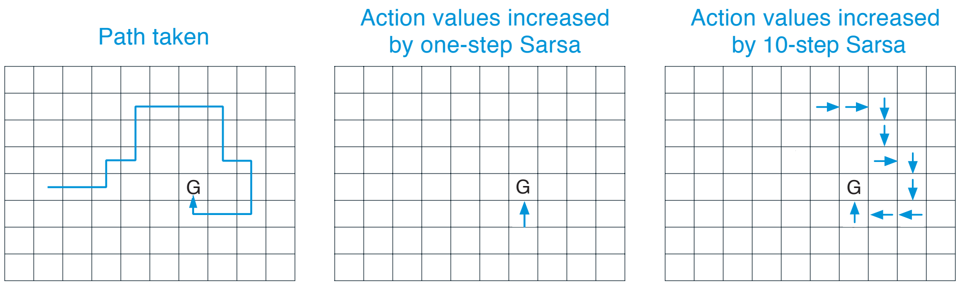

Path taken
Action values increased by one-step Sarsa
Action values increased by 10-step Sarsa
G
G
G

Figure 7.4: Gridworld example of the speedup of policy learning due to the use of n-step methods. The first panel shows the path taken by an agent in a single episode, ending at a location of high reward, marked by the G. In this example the values were all initially 0, and all rewards were zero except for a positive reward at G. The arrows in the other two panels show which action values were strengthened as a result of this path by one-step and n-step Sarsa methods. The one-step method strengthens only the last action of the sequence of actions that led to the high reward, whereas the n-step method strengthens the last n actions of the sequence, so that much more is learned from the one episode.

Exercise 7.4 Prove that the n-step return of Sarsa (7.4) can be written exactly in terms of a novel TD error, as

$$G_{t:t+n} = Q_{t-1}(S_t, A_t) + \sum_{k=t}^{\min(t+n,T)-1} \gamma^{k-t} \left[R_{k+1} + \gamma Q_k(S_{k+1}, A_{k+1}) - Q_{k-1}(S_k, A_k)\right]. \tag{7.6}$$

□

What about Expected Sarsa? The backup diagram for the n-step version of Expected Sarsa is shown on the far right in Figure 7.3. It consists of a linear string of sample actions and states, just as in n-step Sarsa, except that its last element is a branch over all action possibilities weighted, as always, by their probability under π. This algorithm can be described by the same equation as n-step Sarsa (above) except with the n-step return redefined as

$$G_{t:t+n} \doteq R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \bar{V}_{t+n-1}(S_{t+n}), \qquad t+n < T, \tag{7.7}$$

(with $G_{t:t+n} \doteq G_t$ for $t+n \geq T$) where $\bar{V}_t(s)$ is the *expected approximate value* of state s, using the estimated action values at time t, under the target policy:

$$\bar{V}_t(s) \doteq \sum_a \pi(a|s) Q_t(s,a), \qquad \text{for all } s \in \mathcal{S}. \tag{7.8}$$

Expected approximate values are used in developing many of the action-value methods in the rest of this book. If s is terminal, then its expected approximate value is defined to be 0.

7.3 n-step Off-policy Learning

Recall that off-policy learning is learning the value function for one policy, π, while following another policy, b. Often, π is the greedy policy for the current action-value-function estimate, and b is a more exploratory policy, perhaps ε-greedy. In order to use the data from b we must take into account the difference between the two policies, using their relative probability of taking the actions that were taken (see Section 5.5). In n-step methods, returns are constructed over n steps, so we are interested in the relative probability of just those n actions. For example, to make a simple off-policy version of n-step TD, the update for time t (actually made at time $t+n$) can simply be weighted by $\rho_{t:t+n-1}$:

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \rho_{t:t+n-1} \left[G_{t:t+n} - V_{t+n-1}(S_t)\right], \quad 0 \leq t < T, \tag{7.9}$$

where $\rho_{t:t+n-1}$, called the *importance sampling ratio*, is the relative probability under the two policies of taking the n actions from A_t to A_{t+n-1} (cf. Eq. 5.3):

$$\rho_{t:h} \doteq \prod_{k=t}^{\min(h,T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}. \tag{7.10}$$

For example, if any one of the actions would never be taken by π (i.e., $\pi(A_k|S_k) = 0$) then the n-step return should be given zero weight and be totally ignored. On the other hand, if by chance an action is taken that π would take with much greater probability than b does, then this will increase the weight that would otherwise be given to the return. This makes sense because that action is characteristic of π (and therefore we want to learn about it) but is selected only rarely by b and thus rarely appears in the data. To make up for this we have to over-weight it when it does occur. Note that if the two policies are actually the same (the on-policy case) then the importance sampling ratio is always 1. Thus our new update (7.9) generalizes and can completely replace our earlier n-step TD update. Similarly, our previous n-step Sarsa update can be completely replaced by a simple off-policy form:

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \rho_{t+1:t+n} \left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)\right], \tag{7.11}$$

for $0 \leq t < T$. Note that the importance sampling ratio here starts and ends one step later than for n-step TD (7.9). This is because here we are updating a state–action pair. We do not have to care how likely we were to select the action; now that we have selected it we want to learn fully from what happens, with importance sampling only for subsequent actions. Pseudocode for the full algorithm is shown in the box below.

Off-policy n-step Sarsa for estimating $Q \approx q_*$ or q_π

```
Input: an arbitrary behavior policy $b$ such that $b(a|s) > 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $\pi$ to be greedy with respect to $Q$, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, a positive integer $n$
All store and access operations (for $S_t$, $A_t$, and $R_t$) can take their index mod $n+1$

Loop for each episode:
   Initialize and store $S_0 \neq$ terminal
   Select and store an action $A_0 \sim b(\cdot|S_0)$
   $T \leftarrow \infty$
   Loop for $t = 0, 1, 2, \ldots$ :
   |  If $t < T$, then:
   |      Take action $A_t$
   |      Observe and store the next reward as $R_{t+1}$ and the next state as $S_{t+1}$
   |      If $S_{t+1}$ is terminal, then:
   |          $T \leftarrow t+1$
   |      else:
   |          Select and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$
   |  $\tau \leftarrow t - n + 1$    ($\tau$ is the time whose estimate is being updated)
   |  If $\tau \geq 0$:
   |      $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n,T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$                    $(\rho_{\tau+1:\tau+n})$
   |      $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
   |      If $\tau + n < T$, then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$          $(G_{\tau:\tau+n})$
   |      $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho\,[G - Q(S_\tau, A_\tau)]$
   |      If $\pi$ is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt $Q$
   Until $\tau = T - 1$
```

The off-policy version of n-step Expected Sarsa would use the same update as above for n-step Sarsa except that the importance sampling ratio would have one less factor in it. That is, the above equation would use $\rho_{t+1:t+n-1}$ instead of $\rho_{t+1:t+n}$, and of course it would use the Expected Sarsa version of the n-step return (7.7). This is because in Expected Sarsa all possible actions are taken into account in the last state; the one actually taken has no effect and does not have to be corrected for.

7.4 *Per-decision Methods with Control Variates

The multi-step off-policy methods presented in the previous section are simple and conceptually clear, but are probably not the most efficient. A more sophisticated approach would use per-decision importance sampling ideas such as were introduced in Section 5.9. To understand this approach, first note that the ordinary n-step return (7.1), like all returns, can be written recursively. For the n steps ending at horizon h, the n-step return can be written

$$G_{t:h} = R_{t+1} + \gamma G_{t+1:h}, \qquad t < h < T, \tag{7.12}$$

where $G_{h:h} \doteq V_{h-1}(S_h)$. (Recall that this return is used at time h, previously denoted $t+n$.) Now consider the effect of following a behavior policy b that is not the same as the target policy π. All of the resulting experience, including the first reward R_{t+1} and the next state S_{t+1}, must be weighted by the importance sampling ratio for time t, $\rho_t = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$. One might be tempted to simply weight the righthand side of the above equation, but one can do better. Suppose the action at time t would never be selected by π, so that ρ_t is zero. Then a simple weighting would result in the n-step return being zero, which could result in high variance when it was used as a target. Instead, in this more sophisticated approach, one uses an alternate, *off-policy* definition of the n-step return ending at horizon h, as

$$G_{t:h} \doteq \rho_t \left(R_{t+1} + \gamma G_{t+1:h}\right) + (1-\rho_t)V_{h-1}(S_t), \qquad t < h < T, \tag{7.13}$$

where again $G_{h:h} \doteq V_{h-1}(S_h)$. In this approach, if ρ_t is zero, then instead of the target being zero and causing the estimate to shrink, the target is the same as the estimate and causes no change. The importance sampling ratio being zero means we should ignore the sample, so leaving the estimate unchanged seems appropriate. The second, additional term in (7.13) is called a *control variate* (for obscure reasons). Notice that the control variate does not change the expected update; the importance sampling ratio has expected value one (Section 5.9) and is uncorrelated with the estimate, so the expected value of the control variate is zero. Also note that the off-policy definition (7.13) is a strict generalization of the earlier on-policy definition of the n-step return (7.1), as the two are identical in the on-policy case, in which ρ_t is always 1.

For a conventional n-step method, the learning rule to use in conjunction with (7.13) is the n-step TD update (7.2), which has no explicit importance sampling ratios other than those embedded in the return.

Exercise 7.5 Write the pseudocode for the off-policy state-value prediction algorithm described above. □

For action values, the off-policy definition of the n-step return is a little different because the first action does not play a role in the importance sampling. That first action is the one being learned; it does not matter if it was unlikely or even impossible under the target policy—it has been taken and now full unit weight must be given to the reward and state that follows it. Importance sampling will apply only to the actions that follow it.

First note that for action values the n-step *on-policy* return ending at horizon h, expectation form (7.7), can be written recursively just as in (7.12), except that for action values the recursion ends with $G_{h:h} \doteq \bar{V}_{h-1}(S_h)$ as in (7.8). An off-policy form with control variates is

$$\begin{aligned} G_{t:h} &\doteq R_{t+1} + \gamma\Big(\rho_{t+1} G_{t+1:h} + \bar{V}_{h-1}(S_{t+1}) - \rho_{t+1} Q_{h-1}(S_{t+1}, A_{t+1})\Big), \\ &= R_{t+1} + \gamma\rho_{t+1}\Big(G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1})\Big) + \gamma\bar{V}_{h-1}(S_{t+1}), \quad t < h \le T. \end{aligned} \tag{7.14}$$

If $h < T$, then the recursion ends with $G_{h:h} \doteq Q_{h-1}(S_h, A_h)$, whereas, if $h \ge T$, the recursion ends with and $G_{T-1:h} \doteq R_T$. The resultant prediction algorithm (after combining with (7.5)) is analogous to Expected Sarsa.

Exercise 7.6 Prove that the control variate in the above equations does not change the expected value of the return. □

**Exercise 7.7* Write the pseudocode for the off-policy action-value prediction algorithm described immediately above. Pay particular attention to the termination conditions for the recursion upon hitting the horizon or the end of episode. □

Exercise 7.8 Show that the general (off-policy) version of the n-step return (7.13) can still be written exactly and compactly as the sum of state-based TD errors (6.5) if the approximate state value function does not change. □

Exercise 7.9 Repeat the above exercise for the action version of the off-policy n-step return (7.14) and the Expected Sarsa TD error (the quantity in brackets in Equation 6.9). □

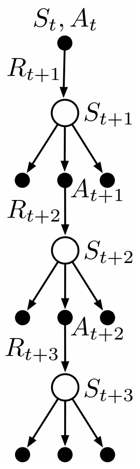
Exercise 7.10 (programming) Devise a small off-policy prediction problem and use it to show that the off-policy learning algorithm using (7.13) and (7.2) is more data efficient than the simpler algorithm using (7.1) and (7.9). □

The importance sampling that we have used in this section, the previous section, and in Chapter 5, enables sound off-policy learning, but also results in high variance updates, forcing the use of a small step-size parameter and thereby causing learning to be slow. It is probably inevitable that off-policy training is slower than on-policy training—after all, the data is less relevant to what is being learned. However, it is probably also true that these methods can be improved on. The control variates are one way of reducing the variance. Another is to rapidly adapt the step sizes to the observed variance, as in the Autostep method (Mahmood, Sutton, Degris and Pilarski, 2012). Yet another promising approach is the invariant updates of Karampatziakis and Langford (2010) as extended to TD by Tian (in preparation). The usage technique of Mahmood (2017; Mahmood and Sutton, 2015) may also be part of the solution. In the next section we consider an off-policy learning method that does not use importance sampling.

7.5 Off-policy Learning Without Importance Sampling: The n-step Tree Backup Algorithm

Is off-policy learning possible without importance sampling? Q-learning and Expected Sarsa from Chapter 6 do this for the one-step case, but is there a corresponding multi-step algorithm? In this section we present just such an n-step method, called the *tree-backup algorithm*.

The idea of the algorithm is suggested by the 3-step tree-backup backup diagram shown to the right. Down the central spine and labeled in the diagram are three sample states and rewards, and two sample actions. These are the random variables representing the events occurring after the initial state–action pair S_t, A_t. Hanging off to the sides of each state are the actions that were *not* selected. (For the last state, all the actions are considered to have not (yet) been selected.) Because we have no sample data for the unselected actions, we bootstrap and use the estimates of their values in forming the target for the update. This slightly extends the idea of a backup diagram. So far we have always updated the estimated value of the node at the top of the diagram toward a target combining the rewards along the way (appropriately discounted) and the estimated values of the nodes at the bottom. In the tree-backup update, the target includes all these things *plus* the estimated values of the dangling action nodes hanging off the sides, at all levels. This is why it is called a *tree-backup* update; it is an update from the entire tree of estimated action values.



the 3-step tree-backup update

More precisely, the update is from the estimated action values of the *leaf nodes* of the tree. The action nodes in the interior, corresponding to the actual actions taken, do not participate. Each leaf node contributes to the target with a weight proportional to its probability of occurring under the target policy π. Thus each first-level action a contributes with a weight of $\pi(a|S_{t+1})$, except that the action actually taken, A_{t+1}, does not contribute at all. Its probability, $\pi(A_{t+1}|S_{t+1})$, is used to weight all the second-level action values. Thus, each non-selected second-level action a' contributes with weight $\pi(A_{t+1}|S_{t+1})\pi(a'|S_{t+2})$. Each third-level action contributes with weight $\pi(A_{t+1}|S_{t+1})\pi(A_{t+2}|S_{t+2})\pi(a''|S_{t+3})$, and so on. It is as if each arrow to an action node in the diagram is weighted by the action's probability of being selected under the target policy and, if there is a tree below the action, then that weight applies to all the leaf nodes in the tree.

We can think of the 3-step tree-backup update as consisting of 6 half-steps, alternating between sample half-steps from an action to a subsequent state, and expected half-steps considering from that state all possible actions with their probabilities of occurring under the policy.

Now let us develop the detailed equations for the n-step tree-backup algorithm. The one-step return (target) is the same as that of Expected Sarsa,

$$G_{t:t+1} \doteq R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q_t(S_{t+1}, a), \tag{7.15}$$

for $t < T-1$, and the two-step tree-backup return is

$$\begin{aligned} G_{t:t+2} \doteq R_{t+1} &+ \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a) \\ &+ \gamma \pi(A_{t+1}|S_{t+1}) \Big(R_{t+2} + \gamma \sum_a \pi(a|S_{t+2}) Q_{t+1}(S_{t+2}, a) \Big) \\ = R_{t+1} &+ \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+2}, \end{aligned}$$

for $t < T-2$. The latter form suggests the general recursive definition of the tree-backup n-step return:

$$G_{t:t+n} \doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+n-1}(S_{t+1}, a) \; + \; \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+n}, \tag{7.16}$$

for $t < T-1, n \geq 2$, with the $n = 1$ case handled by (7.15) except for $G_{T-1:t+n} \doteq R_T$. This target is then used with the usual action-value update rule from n-step Sarsa:

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t) \right],$$

for $0 \leq t < T$, while the values of all other state–action pairs remain unchanged: $Q_{t+n}(s, a) = Q_{t+n-1}(s, a)$, for all s, a such that $s \neq S_t$ or $a \neq A_t$. Pseudocode for this algorithm is shown in the box on the next page.

Exercise 7.11 Show that if the approximate action values are unchanging, then the tree-backup return (7.16) can be written as a sum of expectation-based TD errors:

$$G_{t:t+n} = Q(S_t, A_t) + \sum_{k=t}^{\min(t+n-1, T-1)} \delta_k \prod_{i=t+1}^{k} \gamma \pi(A_i|S_i),$$

where $\delta_t \doteq R_{t+1} + \gamma \bar{V}_t(S_{t+1}) - Q(S_t, A_t)$ and $\bar{V}_t$ is given by (7.8). □

n-step Tree Backup for estimating $Q \approx q_*$ or q_π

```
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $\pi$ to be greedy with respect to $Q$, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, a positive integer $n$
All store and access operations can take their index mod $n+1$

Loop for each episode:
    Initialize and store $S_0 \neq$ terminal
    Choose an action $A_0$ arbitrarily as a function of $S_0$; Store $A_0$
    $T \leftarrow \infty$
    Loop for $t = 0, 1, 2, \ldots$ :
    |   If $t < T$:
    |       Take action $A_t$; observe and store the next reward and state as $R_{t+1}, S_{t+1}$
    |       If $S_{t+1}$ is terminal:
    |           $T \leftarrow t + 1$
    |       else:
    |           Choose an action $A_{t+1}$ arbitrarily as a function of $S_{t+1}$; Store $A_{t+1}$
    |   $\tau \leftarrow t + 1 - n$    ($\tau$ is the time whose estimate is being updated)
    |   If $\tau \geq 0$:
    |       If $t + 1 \geq T$:
    |           $G \leftarrow R_T$
    |       else
    |           $G \leftarrow R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a)$
    |       Loop for $k = \min(t, T-1)$ down through $\tau + 1$:
    |           $G \leftarrow R_k + \gamma \sum_{a \neq A_k} \pi(a|S_k) Q(S_k, a) + \gamma \pi(A_k|S_k) G$
    |       $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha [G - Q(S_\tau, A_\tau)]$
    |       If $\pi$ is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt $Q$
    Until $\tau = T - 1$
```

7.6 *A Unifying Algorithm: n-step $Q(\sigma)$

So far in this chapter we have considered three different kinds of action-value algorithms, corresponding to the first three backup diagrams shown in Figure 7.5. n-step Sarsa has all sample transitions, the tree-backup algorithm has all state-to-action transitions fully branched without sampling, and n-step Expected Sarsa has all sample transitions except for the last state-to-action one, which is fully branched with an expected value. To what extent can these algorithms be unified?

One idea for unification is suggested by the fourth backup diagram in Figure 7.5. This is the idea that one might decide on a step-by-step basis whether one wanted to take the action as a sample, as in Sarsa, or consider the expectation over all actions instead, as in the tree-backup update. Then, if one chose always to sample, one would obtain Sarsa, whereas if one chose never to sample, one would get the tree-backup algorithm. Expected Sarsa would be the case where one chose to sample for all steps except for the last one.

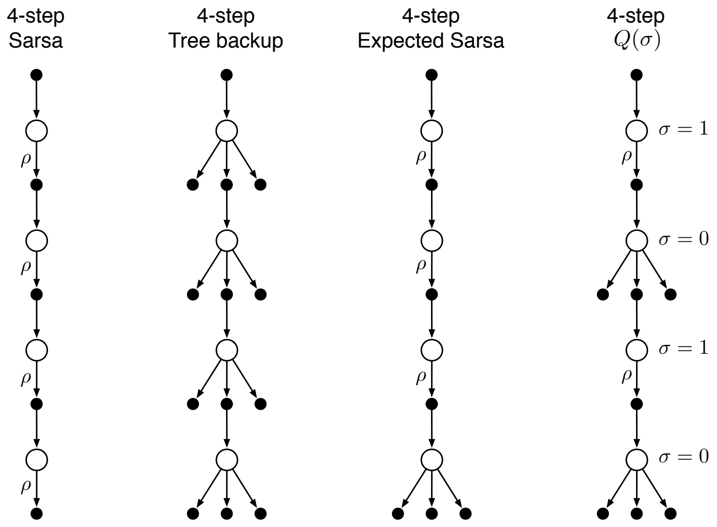


Figure 7.5: The backup diagrams of the three kinds of n-step action-value updates considered so far in this chapter (4-step case) plus the backup diagram of a fourth kind of update that unifies them all. The label 'ρ' indicates half transitions on which importance sampling is required in the off-policy case. The fourth kind of update unifies all the others by choosing on a state-by-state basis whether to sample ($\sigma_t = 1$) or not ($\sigma_t = 0$).

And of course there would be many other possibilities, as suggested by the last diagram in the figure. To increase the possibilities even further we can consider a continuous variation between sampling and expectation. Let $\sigma_t \in [0, 1]$ denote the degree of sampling on step t, with $\sigma = 1$ denoting full sampling and $\sigma = 0$ denoting a pure expectation with no sampling. The random variable σ_t might be set as a function of the state, action, or state–action pair at time t. We call this proposed new algorithm n-step $Q(\sigma)$.

Now let us develop the equations of n-step $Q(\sigma)$. First we write the tree-backup n-step return (7.16) in terms of the horizon $h = t + n$ and then in terms of the expected approximate value $\bar{V}$ (7.8):

$$\begin{aligned}
G_{t:h} &= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{h-1}(S_{t+1}, a) \;+\; \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:h} \\
&= R_{t+1} + \gamma \bar{V}_{h-1}(S_{t+1}) - \gamma \pi(A_{t+1}|S_{t+1}) Q_{h-1}(S_{t+1}, A_{t+1}) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:h} \\
&= R_{t+1} + \gamma \pi(A_{t+1}|S_{t+1}) \Big(G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1}) \Big) + \gamma \bar{V}_{h-1}(S_{t+1}),
\end{aligned}$$

after which it is exactly like the n-step return for Sarsa with control variates (7.14) except with the action probability $\pi(A_{t+1}|S_{t+1})$ substituted for the importance-sampling ratio ρ_{t+1}. For $Q(\sigma)$, we slide linearly between these two cases:

$$\begin{aligned}
G_{t:h} \doteq R_{t+1} &+ \gamma \Big(\sigma_{t+1} \rho_{t+1} + (1 - \sigma_{t+1}) \pi(A_{t+1}|S_{t+1}) \Big) \Big(G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1}) \Big) \\
&+ \gamma \bar{V}_{h-1}(S_{t+1}),
\end{aligned} \tag{7.17}$$

for $t < h \le T$. The recursion ends with $G_{h:h} \doteq Q_{h-1}(S_h, A_h)$ if $h < T$, or with $G_{T-1:T} \doteq R_T$ if $h = T$. Then we use the earlier update for n-step Sarsa without importance-sampling ratios (7.5) instead of (7.11), because now the ratios are incorporated in the n-step return. A complete algorithm is given in the box.

Off-policy n-step $Q(\sigma)$ for estimating $Q \approx q_*$ or q_π

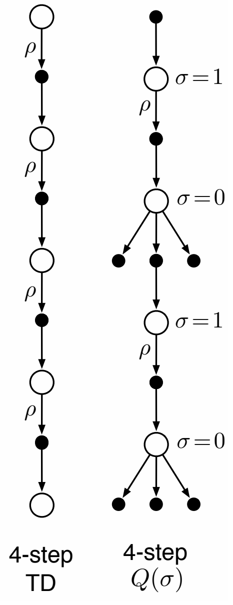
```
Input: an arbitrary behavior policy $b$ such that $b(a|s) > 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $\pi$ to be greedy with respect to $Q$, or else it is a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, a positive integer $n$
All store and access operations can take their index mod $n+1$

Loop for each episode:
   Initialize and store $S_0 \neq$ terminal
   Choose and store an action $A_0 \sim b(\cdot|S_0)$
   $T \leftarrow \infty$
   Loop for $t = 0, 1, 2, \ldots$ :
   |  If $t < T$:
   |     Take action $A_t$; observe and store the next reward and state as $R_{t+1}, S_{t+1}$
   |     If $S_{t+1}$ is terminal:
   |        $T \leftarrow t+1$
   |     else:
   |        Choose and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$
   |        Select and store $\sigma_{t+1}$
   |        Store $\frac{\pi(A_{t+1}|S_{t+1})}{b(A_{t+1}|S_{t+1})}$ as $\rho_{t+1}$
   |  $\tau \leftarrow t - n + 1$   ($\tau$ is the time whose estimate is being updated)
   |  If $\tau \ge 0$:
   |     If $t + 1 < T$:
   |        $G \leftarrow Q(S_{t+1}, A_{t+1})$
   |     Loop for $k = \min(t+1, T)$ down through $\tau + 1$:
   |        if $k = T$:
   |           $G \leftarrow R_T$
   |        else:
   |           $\bar{V} \leftarrow \sum_a \pi(a|S_k)Q(S_k, a)$
   |           $G \leftarrow R_k + \gamma\big(\sigma_k\rho_k + (1-\sigma_k)\pi(A_k|S_k)\big)\big(G - Q(S_k, A_k)\big) + \gamma\bar{V}$
   |     $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\,[G - Q(S_\tau, A_\tau)]$
   |     If $\pi$ is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt $Q$
   Until $\tau = T - 1$
```

7.7 Summary

In this chapter we have developed a range of temporal-difference learning methods that lie in between the one-step TD methods of the previous chapter and the Monte Carlo methods of the chapter before. Methods that involve an intermediate amount of bootstrapping are important because they will typically perform better than either extreme.

Our focus in this chapter has been on n-step methods, which look ahead to the next n rewards, states, and actions. The two 4-step backup diagrams to the right together summarize most of the methods introduced. The state-value update shown is for n-step TD with importance sampling, and the action-value update is for n-step $Q(\sigma)$, which generalizes Expected Sarsa and Q-learning. All n-step methods involve a delay of n time steps before updating, as only then are all the required future events known. A further drawback is that they involve more computation per time step than previous methods. Compared to one-step methods, n-step methods also require more memory to record the states, actions, rewards, and sometimes other variables over the last n time steps. Eventually, in Chapter 12, we will see how multi-step TD methods can be implemented with minimal memory and computational complexity using eligibility traces, but there will always be some additional computation beyond one-step methods. Such costs can be well worth paying to escape the tyranny of the single time step.



Although n-step methods are more complex than those using eligibility traces, they have the great benefit of being conceptually clear. We have sought to take advantage of this by developing two approaches to off-policy learning in the n-step case. One, based on importance sampling is conceptually simple but can be of high variance. If the target and behavior policies are very different it probably needs some new algorithmic ideas before it can be efficient and practical. The other, based on tree-backup updates, is the natural extension of Q-learning to the multi-step case with stochastic target policies. It involves no importance sampling but, again if the target and behavior policies are substantially different, the bootstrapping may span only a few steps even if n is large.

Bibliographical and Historical Remarks

The notion of n-step returns is due to Watkins (1989), who also first discussed their error reduction property. n-step algorithms were explored in the first edition of this book, in which they were treated as of conceptual interest, but not feasible in practice. The work of Cichosz (1995) and particularly van Seijen (2016) showed that they are actually completely practical algorithms. Given this, and their conceptual clarity and simplicity, we have chosen to highlight them here in the second edition. In particular, we now postpone all discussion of the backward view and of eligibility traces until Chapter 12.

7.1–2 The results in the random walk examples were made for this text based on work of Sutton (1988) and Singh and Sutton (1996). The use of backup diagrams to describe these and other algorithms in this chapter is new.

7.3–5 The developments in these sections are based on the work of Precup, Sutton, and Singh (2000), Precup, Sutton, and Dasgupta (2001), and Sutton, Mahmood, Precup, and van Hasselt (2014).

The tree-backup algorithm is due to Precup, Sutton, and Singh (2000), but the presentation of it here is new.

7.6 The $Q(\sigma)$ algorithm is new to this text, but closely related algorithms have been explored further by De Asis, Hernandez-Garcia, Holland, and Sutton (2017).