Answers to Exercises
Reinforcement Learning: Chapter 6

Exercise 6.1 If V changes during the episode, then (6.6) only holds approximately; what would the difference be between the two sides? Let V_t denote the array of state values used at time t in the TD error (6.5) and in the TD update (6.2). Redo the derivation above to determine the additional amount that must be added to the sum of TD errors in order to equal the Monte Carlo error.

Answer:
The TD error, now in terms of V_t, is $\delta_t \doteq R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)$.
Then

$$\begin{aligned}
G_t - V_t(S_t) &= R_{t+1} + \gamma G_{t+1} - V_t(S_t) + \gamma V_t(S_{t+1}) - \gamma V_t(S_{t+1}) && \text{(from (3.9))} \\
&= \delta_t + \gamma\big(G_{t+1} - V_t(S_{t+1})\big) \\
&= \delta_t + \gamma\big(G_{t+1} - V_{t+1}(S_{t+1})\big) + \gamma\big(V_{t+1}(S_{t+1}) - V_t(S_{t+1})\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\big(G_{t+2} - V(S_{t+2})\big) + \gamma^2\big(V_{t+2}(S_{t+2}) - V_{t+1}(S_{t+2})\big) + \gamma\big(V_{t+1}(S_{t+1}) - V_t(S_{t+1})\big) \\
&= \delta_t + \gamma\delta_{t+1} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}\big(G_T - V(S_T)\big) + \sum_{k=t}^{T-1}\gamma^{k-t+1}\big(V_{k+1}(S_{k+1}) - V_k(S_{k+1})\big) \\
&= \delta_t + \gamma\delta_{t+1} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}\big(0 - 0\big) + \sum_{k=t}^{T-1}\gamma^{k-t+1}\big(V_{k+1}(S_{k+1}) - V_k(S_{k+1})\big) \\
&= \sum_{k=t}^{T-1}\gamma^{k-t}\delta_k \quad + \quad \sum_{k=t}^{T-1}\gamma^{k-t+1}\big(V_{k+1}(S_{k+1}) - V_k(S_{k+1})\big).
\end{aligned}$$

□

Exercise 6.2 This is an exercise to help develop your intuition about why TD methods are often more efficient than Monte Carlo methods. Consider the driving home example and how it is addressed by TD and Monte Carlo methods. Can you imagine a scenario in which a TD update would be better on average than a Monte Carlo update? Give an example scenario—a description of past experience and a current state—in which you would expect the TD update to be better. Here's a hint: Suppose you have lots of experience driving home from work. Then you move to a new building and a new parking lot (but you still enter the highway at the same place). Now you are starting to learn predictions for the new building. Can you see why TD updates are likely to be much better, at least initially, in this case? Might the same sort of thing happen in the original scenario?

Answer: In learning predictions from the new building with a TD method you will be able to take advantage of your existing, very accurate estimates of travel times after entering the highway. Monte Carlo samples will be noisy because of variations in how long it takes you to get to the highway *and* because of variations in traveling from the highway entrance to your home, whereas TD methods will be noisy almost entirely due to the former. Variations in the second portion of the journey will not contribute noise to the TD estimate.

The same sort of thing might well happen in the original problem as well, particularly if you can reach the same state from different initial paths. When you come across the same state from a new direction, a TD method enables you to take advantage of what you have previously learned about that state along the other path. □

Exercise 6.3 From the results shown in the left graph of the random walk example it appears that the first episode results in a change in only $V(\mathsf{A})$. What does this tell you about what happened on the first episode? Why was only the estimate for this one state changed? By exactly how much was it changed?

Answer: That only $V(\mathsf{A})$ was changed after the first episode tells us that the first episode terminated on the left, from A. Only this value was changed because all other state transitions had a TD error of zero (all predictions were the same and all rewards were zero on these other transitions). The TD error for the transition from A into the terminal state involved a change in prediction from 0.5 to 0 (the terminal state has value 0) and a reward of zero. The total change was

$$\alpha\big[R + V(\texttt{terminal}) - V(\mathsf{A})\big] = 0.1[0 + 0 - 0.5] = -.05. \qquad \square$$

Exercise 6.4 The specific results shown in the right graph of the random walk example are dependent on the value of the step-size parameter, α. Do you think the conclusions about which algorithm is better would be affected if a wider range of α values were used? Is there a different, fixed value of α at which either algorithm would have performed significantly better than shown? Why or why not?

Answer: It is very unlikely that either algorithm could have done significantly better with another value of α. Higher α values tend to result in faster learning but greater later variability and thus lower long-term performance. Thus the $\alpha = 0.15$ error curve for TD comes down fastest, but ends higher than the other TD curves, and similarly for the $\alpha = 0.04$ Monte Carlo error curve. Better performance is obtained at α values less than these, as we see in the figure. On the other hand, it is not likely that one could do better than shown in the figure by choosing α even lower, at least not over the time span shown in the figure. Note that the $\alpha = 0.05$ TD curve and the $\alpha = 0.01$ Monte Carlo curve are already very slow to improve relative to the others of their kind. If α was reduced even further, the overall performance over this time span would presumably be even further diminished.

In other words, error performance is in general a U-shaped function of α, and the range of values tried here appears to bracket the region of likely best value. $\square$

***Exercise 6.5** In the right graph of the random walk example, the RMS error of the TD method seems to go down and then up again, particularly at high α's. What could have caused this? Do you think this always occurs, or might it be a function of how the approximate value function was initialized?

Answer: This effect is in fact real and totally dependent on the initialization of the approximate value function. If the estimate are all started inappropriately, e.g., all at 5.0, then this down and up effect is not seen. Remember that, for TD methods, new estimates are built upon existing estimates. The initial estimate of 0.5 is not that far off from the true values of the states. To see the reason for the effect intuitively, consider the extreme case in which the initial estimates were set to be exactly correct (as they were for state C). The initial error would then be zero. But the estimates would still change in response to random fluctuations—as epsiodes end on one side or the other—and thus the error would have to increase from zero. Essentially the same effect seems to be happening with the 0.5 initialization. $\square$

Exercise 6.6 In Example 6.2 we stated that the true values for the random walk example are $\frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}$, and $\frac{5}{6}$, for states A through E. Describe at least two different ways that these could have been computed. Which would you guess we actually used? Why?

Answer: Here are some ways of computing these values:

- We could have used the DP method of iterative policy evalution. This is the method where we use knowledge of the environment dynamics (a model) to look ahead one step and backup the values, i.e., full DP backups for v_π.
- We could have analytically solved for the solution to the Bellman equations for v_π. This would involve solving a system of five linear equations, i.e., inverting a small matrix.
- We could have used a Monte Carlo method for policy evaluation. This would involve generating lots of sample walks from each state and taking the fraction that ended on each side.
- We could have guessed at the values and then checked that the Bellman equation for v_π held. We know that the solution to the Bellman equations is unique, so any solution consistent with the Bellman equation must be correct.

Actually, I think we used the guessing method, but either that or the DP method is most suitable. The Monte Carlo method is less satisfactory in this case because it converges only approximately and after many episodes, whereas the DP method converges very quickly on a small problem like this. The big advantage of the guessing method is that it confirms an exact answer. Even if we used one of the other methods, we likely looked at the final numerical answer, noted it closeness to the sixth fractions, and checked those fractions against the Bellman equations to see if they were in fact the exact answers. □

***Exercise 6.7** Design an off-policy version of the TD(0) update that can be used with arbitrary target policy π and covering behavior policy b, using at each step t the importance sampling ratio $\rho_{t:t}$ (5.3).

Answer:

$$V(S_t) \leftarrow V(S_t) + \alpha\rho_{t:t}\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big] \qquad \square$$

Exercise 6.8 Show that an action-value version of (6.6) holds for the action-value form of the TD error $\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$, again assuming that the values don't change from step to step.

Answer:

$$\begin{aligned} G_t - Q(S_t, A_t) &= R_{t+1} + \gamma G_{t+1} - Q(S_t, A_t) + \gamma Q(S_{t+1}, A_{t+1}) - \gamma Q(S_{t+1}, A_{t+1}) \\ &= \delta_t + \gamma\big(G_{t+1} - Q(S_{t+1}, A_{t+1})\big) \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2\big(G_{t+2} - Q(S_{t+2}, A_{t+2})\big) \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}\big(G_T - Q(S_T, A_T)\big) \end{aligned} \tag{3.9}$$

(where the value of any action in the terminal state is interpreted as zero)

$$\begin{aligned} &= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}\big(0 - 0\big) \\ &= \sum_{k=t}^{T-1} \gamma^{k-t}\delta_k \end{aligned}$$

□

Exercise 6.11 Why is Q-learning considered an *off-policy* control method?

Answer: Q-learning learns about the greedy policy, which gradually becomes the optimal policy, independent of what policy is actually being followed by the agent (as long as it tries all state-action pairs). Thus, it is learning about one policy while following another, which by definition makes it an off-policy method. □

Exercise 6.12 Suppose action selection is greedy. Is Q-learning then exactly the same algorithm as Sarsa? Will they make exactly the same action selections and weight updates?

Answer: The two algorithms will actually be subtly different. In particular, they will not always make the same action selections. To see this, consider a transition $t \to t+1$ in which the state does not change and the learning update changes which action is the greedy action. On time step $t+1$, Q-learning will take the newly greedy action, but Sarsa will take the formerly greedy action. This is because Sarsa must commit to A_{t+1} at t, before the update, as this action is used in the update at time t. □

***Exercise 6.13** What are the update equations for Double Expected Sarsa with an ε-greedy target policy?

Answer: Let Q_1 and Q_2 be the two action-value functions and let π_1 and π_2 be their ε-greedy policies respectively. Then the updates would be:

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \sum_a \pi_2(a|S_{t+1})Q_1(S_{t+1}, a) - Q_1(S_t, A_t)\Big]$$

and

$$Q_2(S_t, A_t) \leftarrow Q_2(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \sum_a \pi_1(a|S_{t+1})Q_2(S_{t+1}, a) - Q_2(S_t, A_t)\Big],$$

one of which would be done on each step as usual. □

Exercise 6.14 Describe how the task of Jack's Car Rental (Example 4.2) could be reformulated in terms of afterstates. Why, in terms of this specific task, would such a reformulation be likely to speed convergence?

Answer: The afterstates would be the configurations *after* the cars had been moved at the end of the day, but before the next day's pickup and dropoff events. The results of the moving actions at the end of the day are completely known; it is only the pickups and dropoffs that are uncertain. Thus, different numbers of moved cars from different states could result in the same afterstate. For example, one car at each location, after moving, could arise from the same configuration, moving no cars, or from a configuration with both at one location, moving one car to the other location. In the original formulation, two different action values have to be learned for these two situations, each using only part of the data. In the afterstate formulation, learning about the one afterstate can use the data from both cases, and thus should learn faster. In addition, much less memory is required because only an afterstate-value function need be stored, not a full action-value function. □