Answers to Exercises

for

Reinforcement Learning: An Introduction

2nd Edition

Richard S. Sutton and Andrew G. Barto

Answers to Exercises
Reinforcement Learning: Chapter 1

Exercise 1.1: *Self-Play* Suppose, instead of playing against a random opponent, the reinforcement learning algorithm described above played against itself, with both sides learning. What do you think would happen in this case? Would it learn a different policy for selecting moves?

Answer: Yes, it would learn a different move-selection policy. Contrary to what one might at first suppose, this new way of playing will not involve the same few games being played over and over. Random exploratory moves will still cause all positions to be encountered, at least occasionally. Just as playing a static imperfect opponent results in the reinforcement learning agent learning to play optimally (maximum probability of winning) against that opponent (except for exploratory moves), so playing itself will result in it learning to play optimally against itself (except for exploratory moves).

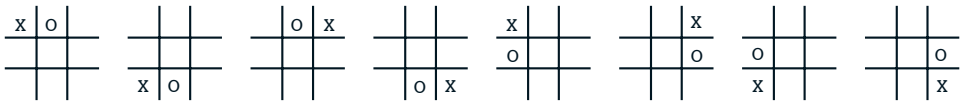
What is less clear is exactly what optimal play against itself would mean. There is a tendency to think that the self-play player would learn to play "better," closer to a universal "optimal" way of playing. But these terms don't have any meaning except with respect to a particular opponent. The self-play player may learn to play better against many opponents, but I think we could easily construct an opponent against whom it would actually fare worse than the original player.

What *can* we say about how the self-play player will play? Clearly all positions in which it can force a win will be played correctly (except for exploratory moves). Because it does make occasional exploratory moves, it will favor lines of play in which it is unlikely to not win even if it explores. And because occasional random moves are made by the opponent, the reinforcement learning agent will prefer positions in which many such moves are losing moves. What about positions in which one side cannot win, but could either lose or draw? Because that player doesn't care about the difference between these two outcomes, it will not try to avoid losses. The other player will prefer these positions over those in which its opponent cannot lose.

One person answering this question pointed out to me that technically it is not possible for the reinforcement learning agent as described to play against itself. This is because the described reinforcement learning agent only learned to evaluate positions after second-player moves. The player who moves first needs to learn the values of positions after first-player moves. Because these two sets of positions are disjoint, and the reinforcement learning agent as described learns separately about each individual position, learning about the two kinds of positions will be totally separate. Thus the reinforcement learning agent cannot play against itself, but only against a copy of itself adjusted to learn after first player moves.

Exercise 1.2: *Symmetries* Many tic-tac-toe positions appear different but are really the same because of symmetries. How might we amend the learning process described above to take advantage of this? In what ways would this change improve the learning process? Now think again. Suppose the opponent did not take advantage of symmetries. In that case, should we? Is it true, then, that symmetrically equivalent positions should necessarily have the same value?

Answer: There are three axes of symmetry in Tic-Tac-Toe: up-down, right-left, and along the diagonal. By reflecting all positions into a standard form, the set of positions whose value needs to be learned can be reduced by a factor of about eight. For example, only one of the following positions need be represented and learned about—the learning will generalize automatically to the other seven:



This would improve the reinforcement learning agent by reducing its memory requirements and reducing the amount of time (number of games) needed to learn. However, if the opponent is imperfect and does not play symmetrically, then this may be counterproductive. For example, if the opponent played correctly in the first position above but incorrectly in the fourth, then our agent ("O") should prefer playing to the fourth position. That is, these symmetrically equivalent positions should not really have the same value. This would not be possible if they were represented as the same because of symmetries.

What is the right solution? One good way to proceed would be to use two tables, one smaller table that collapsed symmetric positions onto the same entry, and one larger table which did not. The approximate value of a position would be the average of the two table entries, one in each table, that applied to the position. Both table entries that applied to the position would also be updated. The result would be that the system would generalize somewhat between symmetrically equivalent positions, but if needed it would also be able to learn distinct values for them. This approach would not provide any savings in memory or computation time per move (in fact it requires more), but it should speed learning in terms of number of games played, without sacrificing asymptotic performance. This is a first step towards using generalizion and function approximation rather than table lookup. We will consider many such possibilities later in the book. □

Exercise 1.3: *Greedy Play* . Suppose the reinforcement learning player was *greedy*, that is, it always played the move that brought it to the position that it rated the best. Would it learn to play better, or worse, than a non-greedy player? What problems might occur?

Answer: A greedy reinforcement learning player might actually end up playing worse. The problem is that it might get permanently stuck playing less than optimal moves. Suppose there is a position where one move will win 60% of the time but a better move will win 90% of the time. Initially both moves are valued at 0.5, just as all positions are. Suppose the first time it encountered the position by chance it played the 60% move and won. The value of that move will be bumped up, say to 0.51. As a result, that move will become the greedy choice, and next time it will be selected again. It's value will go to 0.6, while the other move, the 90% winning move, will stay valued at 0.5 and never be tried. □

Exercise 1.4: *Learning from Exploration* Suppose learning updates occurred after *all* moves, including exploratory moves. If the step-size parameter is appropriately reduced over time (but not the tendency to explore), then the state values would converge to a set of probabilities. What are the two sets of probabilities computed when we do, and when we do not, learn from exploratory moves? Assuming that we do continue to make exploratory moves, which set of probabilities might be better to learn? Which would result in more wins?

Answer: When we don't learn from exploratory moves we learn the probabilities of winning from each position if we played optimally from then on. But of course we don't play exactly optimally from then on—we occasionally make exploratory moves. When we learn from exploratory moves we learn the probability of winning from each position taking into account the fact of these explorations. The best moves given given explorations may be different from the best ones without exploration. Because we do continue to explore, the second approach will actually end up winning more games than the first. □

Exercise 1.5: *Other Improvements* Can you think of other ways to improve the reinforcement learning player? Can you think of any better way to solve the tic-tac-toe problem as posed?

Answer: Of course many other improvements are possible. Some of these are:

- Using generalizing function approximation to speed learning.
- Using search when choosing moves to look ahead farther than one step.
- Incorporating a priori expert knowledge. A natural way to do this is in the initial value function.
- When a new position is encountered, adjustments could be made not just to the immediately preceding position, but to earlier positions as well. This might speed learning.
- We might learn a model of the opponent (how frequently he makes each move in each positions) and use it off-line to improve the value function.
- We might learn to play differently against different opponents.
- Against a stationary opponent, we might reduce the frequency of exploration over time.

□

Answers to Exercises
Reinforcement Learning: Chapter 2

Exercise 2.1 In ε-greedy action selection, for the case of two actions and $\varepsilon = 0.5$, what is the probability that the greedy action is selected?

Answer: 0.75. There is a 0.5 chance of selecting the greedy action directly, plus a 0.25 chance of selecting it as one of the two actions when a random action is selected. □

Exercise 2.2: *Bandit example* This exercise appeared in the first printing slightly differently than intended. However, it is still a valid exercise, and below we give the answer first for the excercise as intended and then as it appeared in the first printing.

Exercise 2.2: *Bandit example as intended* Consider a k-armed bandit problem with $k = 4$ actions, denoted 1, 2, 3, and 4. Consider applying to this problem a bandit algorithm using ε-greedy action selection, sample-average action-value estimates, and initial estimates of $Q_1(a) = 0$, for all a. Suppose the initial sequence of actions and rewards is $A_1 = 1$, $R_1 = -1$, $A_2 = 2$, $R_2 = 1$, $A_3 = 2$, $R_3 = -2$, $A_4 = 2$, $R_4 = 2$, $A_5 = 3$, $R_5 = 0$. On some of these time steps the ε case may have occurred, causing an action to be selected at random. On which time steps did this definitely occur? On which time steps could this possibly have occurred?

Answer: The ε case definitely came up on steps 4 and 5, and could have come up on any of the steps.

To see this clearly, make a table with the estimates, the set of greedy of actions, and the data at each step:

t	Q_1	Q_2	Q_3	Q_4	$\{A_t^*\}$	A_t	ε-case?	R_t
1	0	0	0	0	$\{1,2,3,4\}$	1	maybe	-1
2	$\mathbf{-1}$	0	0	0	$\{2,3,4\}$	2	maybe	1
3	-1	**1**	0	0	$\{2\}$	2	maybe	-2
4	-1	$\mathbf{-0.5}$	0	0	$\{3,4\}$	2	yes	2
5	-1	**0.3333**	0	0	$\{2\}$	3	yes	0

The estimate that changed on each step is bolded. If the action taken is not in the greedy set, as on time steps 4 and 5, then the ε case must have come up. On steps 1–3, the greedy action was taken, but still it is possible that the ε case come up and the greedy action was taken by chance. Thus, the answer to the second question is *all the time steps.* □

Exercise 2.2: *Bandit example as first printed* Consider a k-armed bandit problem with $k = 4$ actions, denoted 1, 2, 3, and 4. Consider applying to this problem a bandit algorithm using ε-greedy action selection, sample-average action-value estimates, and initial estimates of $Q_1(a) = 0$, for all a. Suppose the initial sequence of actions and rewards is $A_1 = 1$, $R_1 = 1$, $A_2 = 2$, $R_2 = 1$, $A_3 = 2$, $R_3 = 2$, $A_4 = 2$, $R_4 = 2$, $A_5 = 3$, $R_5 = 0$. On some of these time steps the ε case may have occurred, causing an action to be selected at random. On which time steps did this definitely occur? On which time steps could this possibly have occurred?

Answer: The ε case definitely came up on steps 2 and 5, and could have come up on any of the steps.

To see this clearly, make a table with the estimates, the set of greedy of actions, and the data at each step:

t	Q_1	Q_2	Q_3	Q_4	$\{A_t^*\}$	A_t	ε-case?	R_t
1	0	0	0	0	$\{1,2,3,4\}$	1	maybe	1
2	**1**	0	0	0	$\{1\}$	2	yes	1
3	1	**1**	0	0	$\{1,2\}$	2	maybe	2
4	1	**1.5**	0	0	$\{2\}$	2	maybe	2
5	1	**1.6666**	0	0	$\{2\}$	3	yes	0

The estimate that changed on each step is bolded. If the action taken is not in the greedy set, as on time steps 2 and 5, then the ε case must have come up. On step 4, the sole greedy action was taken, but it is possible that the ε case come up and the greedy action was taken by chance. Thus, the answer to the second question is *all the time steps*. □

Exercise 2.3 In the comparison shown in Figure 2.2, which method will perform best in the long run in terms of cumulative reward and probability of selecting the best action? How much better will it be? Express your answer quantitatively.

Answer: The cumulative performance measures are essentially the areas under the curves in Figure 2.2. In the near term, the $\varepsilon = 0.1$ method has a larger area, as shown in the figure. But in the long run, the $\varepsilon = 0.01$ method will reach and sustain a higher level. For example, it will eventually learn to select the best action 99.1% of the time, whereas the $\varepsilon = 0.1$ method will never select the best action more than 91% of the time, for a difference of 8.1%. Thus, in the long run the area under the $\varepsilon = 0.01$ curve will be greater. The greedy method will remain at essentially the same low level shown in the figure for a very long time.

In the long run, the $\varepsilon = 0.01$ method will be greater than the $\varepsilon = 0.1$ method by 8.1% of the difference between the value of the best action and the value of an average action. □

Exercise 2.4 If the step-size parameters, α_n, are not constant, then the estimate Q_n is a weighted average of previously received rewards with a weighting different from that given by (2.6). What is the weighting on each prior reward for the general case, analogous to (2.6), in terms of the sequence of step-size parameters?

Answer: As in the book, let us focus on the sequence of rewards, action values, and step-sizes corresponding to one particular action. Let us denote these by R_n, Q_n, and α_n. This greatly simplifies the notation. Then we can do a derivation similar to the one in (2.6):

$$\begin{aligned}
Q_{n+1} &= Q_n + \alpha_n\Big[R_n - Q_n\Big] \\
&= \alpha_n R_n + (1-\alpha_n)Q_n \\
&= \alpha_n R_n + (1-\alpha_n)\left[\alpha_{n-1}R_{n-1} + (1-\alpha_{n-1})Q_{n-1}\right] \\
&= \alpha_n R_n + (1-\alpha_n)\alpha_{n-1}R_{n-1} + (1-\alpha_n)(1-\alpha_{n-1})Q_{n-1} \\
&= \alpha_n R_n + (1-\alpha_n)\alpha_n R_{n-1} + (1-\alpha_n)(1-\alpha_{n-1})\alpha_{n-2}R_{n-2} + \\
&\qquad\qquad \cdots + (1-\alpha_n)(1-\alpha_{n-1})\cdots(1-\alpha_2)\alpha_1 R_1 \\
&\qquad + \ (1-\alpha_n)(1-\alpha_{n-1})\cdots(1-\alpha_1)Q_1 \\
&= \prod_{i=1}^{n}(1-\alpha_i)Q_1 + \sum_{n=1}^{n}\alpha_n R_n \prod_{i=n+1}^{n}(1-\alpha_i).
\end{aligned}$$

In other words, the weighting on R_n in Q_n is $\alpha_n \prod_{i=n+1}^{n}(1-\alpha_i)$. □

Exercise 2.6: *Mysterious Spikes* The results shown in Figure 2.3 should be quite reliable because they are averages over 2000 individual, randomly chosen 10-armed bandit tasks. Why, then, are there oscillations and spikes in the early part of the curve for the optimistic method? In other words, what might make this method perform particularly better or worse, on average, on particular early steps?

Answer: Because the initial values are optimistic, whatever actions are selected on the very first plays will disappoint, and will not be reselected until all the other actions have been tried at least once. Thus, the first 10 plays will just be a sweep through the 10 actions in some random order. The % optimal action on these 10 plays will thus be at the 10% chance level. The first opportunity to make a better than chance action selection will be on the 11th play. This is the first spike in the graph. This is where the action that did best in the previous 10 plays is selected again. This action naturally has a greater than chance probability of being the optimal action. However, even if it is optimal, it still disappoints because its action value estimate is still optimistic. Remember, these action-value estimates are computed with a constant step-size parameter and shift only gradually from their initial values to the level of the observed rewards. Thus, although the action on the 11th play was the one that did best on the first 10 plays, on the 11th play its value will be pushed *down*, closer to its real value rather its optimistic value. On the 12th play the action that did second best in the first 10 plays will be selected, and on the 13th play the action that did third best will be selected. And so on. By trial 21 we can expect a second spike, but it will be less well defined because the values are starting to become more accurate and less optimistic. □

Exercise 2.7: *Unbiased Constant-Step-Size Trick* In most of this chapter we have used sample averages to estimate action values because sample averages do not produce the initial bias that constant step sizes do (see the analysis leading to (2.6)). However, sample averages are not a completely satisfactory solution because they may perform poorly on nonstationary problems. Is it possible to avoid the bias of constant step sizes while retaining their advantages on nonstationary problems? One way is to use a step size of

$$\beta_n \doteq \alpha/\bar{o}_n, \tag{2.1}$$

to process the nth reward for a particular action, where $\alpha > 0$ is a conventional constant step size, and $\bar{o}_n$ is a trace of one that starts at 0:

$$\bar{o}_n \doteq \bar{o}_{n-1} + \alpha(1 - \bar{o}_{n-1}), \quad \text{for } n \geq 0, \quad \text{with } \bar{o}_0 \doteq 0. \tag{2.2}$$

Carry out an analysis like that in (2.6) to show that Q_n is an exponential recency-weighted average *without initial bias.*

Answer: TBD □

Exercise 2.8: *UCB Spikes* In Figure 2.4 the UCB algorithm shows a distinct spike in performance on the 11th step. Why is this? Note that for your answer to be fully satisfactory it must explain both why the reward increases on the 11th step and why it decreases on the subsequent steps. Hint: if $c = 1$, then the spike is less prominent.

Answer: There are two things to explain here: why the jump up at step 11, and why the smaller jump down on step 12. Together these give us the spike. The jump up at step 11 is due to that being the first step on which the action selected depends on the received rewards—the first ten actions being random without replacement. The action taken on step 11 will be that which got the greatest reward on its single first pick. From the graph it is clear that this is not necessarily the best action (the average reward eventually exceeds this height) but it is far better than a random action. This is why performance jumps up at step 11.

At step 12, the second part of the UCB equation (2.10) kicks in (because now the actions vary in the number of times they have been tried). The one action that did well in the first 10 steps, and was repeated at step 11 will now be at a disadvantage because it has been selected twice while the others only once. If c is large, then this effect dominates and the action that performed best in the first 10 steps is ruled out on step 12. In this case the action selected on step 12 is the action that performed second best in the first 10 steps. On average, this action is less good than the one that performed best in the first 10 steps, so there is a smaller jump down on step 12, giving us the complete spike.

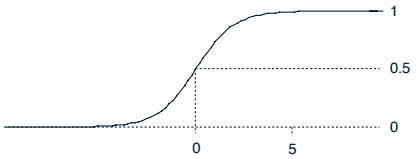
If is smaller (e.g., $c = 1$), then the jump up at step 11 is just the same. However, the jump down at step 12 is smaller because the advantage that comes from having been selected few times is smaller. It is now possible that the action that performed best in steps 1-10, and which was selected again in step 11, will be repeated in step 12. Thus, there is a smaller jump down, although long-term performance may be worse because of the reduced exploration. □

Exercise 2.9 Show that in the case of two actions, the soft-max distribution is the same as that given by the logistic, or sigmoid, function often used in statistics and artificial neural networks.

Answer: The logistic function is

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

where, x is the total evidence (e.g., a weighted sum) in favor of a positive decision, and $\sigma(x)$ is the probability of a positive decision (or, e.g., the output of an artificial neuron). The logistic function looks like this:



In the case of two actions, with two estimated action values, call then $Q(a)$ and $Q(b)$, the soft-max probability of selecting a is

$$\frac{e^{Q(a)}}{e^{Q(a)} + e^{Q(b)}} = \frac{1}{1 + \frac{e^{Q(b)}}{e^{Q(a)}}} = \frac{1}{1 + e^{(Q(b) - Q(a))}} = \sigma(Q(a) - Q(b)).$$

Thus, the Gibbs distribution is making a stochastic decision equivalent in the case of two actions to the logistic in the difference of the action values. Only the difference in the values values matters. □

Exercise 2.10 Suppose you face a 2-armed bandit task whose true action values change randomly from time step to time step. Specifically, suppose that, for any time step, the true values of actions 1 and 2 are respectively 0.1 and 0.2 with probability 0.5 (case A), and 0.9 and 0.8 with probability 0.5 (case B). If you are not able to tell which case you face at any step, what is the best expectation of success you can achieve and how should you behave to achieve it? Now suppose that on each step you are told whether you are facing case A or case B (although you still don't know the true action values). This is an associative search task. What is the best expectation of success you can achieve in this task, and how should you behave to achieve it?

Answer: If you can't tell which bandit problem you are facing, then action 1 results in expected reward of $0.5 \cdot 0.1 + 0.5 \cdot 0.9 = 0.5$ and action 2 results in expected reward of $0.5 \cdot 0.2 + 0.5 \cdot 0.8 = 0.5$. Thus, you would receive an expected reward of 0.5 no matter how you select actions.

If you are told which of the two cases you are facing on each play, then you can learn the right action for each case: action 2 in case A, and action 1 in case B. If you behaved in this way, then you would obtain an expected reward of $0.5 \cdot 0.2 + 0.5 \cdot 0.9 = 0.55$. □

Answers to Exercises
Reinforcement Learning: Chapter 3

The first three exercises are meant simply to be thought provoking and do not have specific answers.

Exercise 3.1 Devise three example tasks of your own that fit into the MDP framework, identifying for each its states, actions, and rewards. Make the three examples as *different* from each other as possible. The framework is abstract and flexible and can be applied in many different ways. Stretch its limits in some way in at least one of your examples.

Answer: This exercise is marked primarily on the basis of your identifying the states, actions and rewards for each example. □

Exercise 3.2 Is the MDP framework adequate to usefully represent *all* goal-directed learning tasks? Can you think of any clear exceptions?

Answer: Any thoughtful answer will do here. I suppose there are exceptions, but the framework seems relevant to almost all the interesting cases. □

Exercise 3.3 Consider the problem of driving. You could define the actions in terms of the accelerator, steering wheel, and brake, that is, where your body meets the machine. Or you could define them farther out—say, where the rubber meets the road, considering your actions to be tire torques. Or you could define them farther in—say, where your brain meets your body, the actions being muscle twitches to control your limbs. Or you could go to a really high level and say that your actions are your choices of *where* to drive. What is the right level, the right place to draw the line between agent and environment? On what basis is one location of the line to be preferred over another? Is there any fundamental reason for preferring one location over another, or is it a free choice?

Answer: I would say that this is largely a free choice, dependent on what level one wishes to address the problem, what one wants to treat as completely controllable. □

Exercise 3.4 Give a table analogous to that in Example 3.3, but for $p(s', r \mid s, a)$. It should have columns for s, a, s', r, and $p(s', r \mid s, a)$, and a row for every 4-tuple for which $p(s', r \mid s, a) > 0$.

Answer:

s	a	s'	r	$p(s', r \mid s, a)$
high	search	high	$r_{\texttt{search}}$	α
high	search	low	$r_{\texttt{search}}$	$1-\alpha$
low	search	high	-3	$1-\beta$
low	search	low	$r_{\texttt{search}}$	β
high	wait	high	$r_{\texttt{wait}}$	1
high	wait	low	$r_{\texttt{wait}}$	0
low	wait	high	$r_{\texttt{wait}}$	0
low	wait	low	$r_{\texttt{wait}}$	1
low	recharge	high	0	1
low	recharge	low	0	0

□

Exercise 3.5 The equations in Section 3.1 are for the continuing case and need to be modified (very slightly) to apply to episodic tasks. Show that you know the modifications needed by giving the modified version of (3.3).

Answer: The sum over all states $\mathcal{S}$ must be modified to a sum over all states $\mathcal{S}^+$. The same is true for (3.5). □

Exercise 3.6 Suppose you treated pole-balancing as an episodic task but also used discounting, with all rewards zero except for -1 upon failure. What then would the return be at each time? How does this return differ from that in the discounted, continuing formulation of this task?

Answer: In this case the return at each time would be $-\gamma^{k-1}$ where k is the number of steps until failure. This differs from the continuing formulation of the task in that what happens on the following episode is ignored. For example, suppose the current policy always avoided failure for exactly 10 steps from the start state. Then under the episodic formulation the return from the start state would be $-\gamma^{10}$ whereas under the continuing formulation it would be $-\gamma^{10} - \gamma^{20} - \gamma^{30} - \cdots$. □

Exercise 3.7 Imagine that you are designing a robot to run a maze. You decide to give it a reward of $+1$ for escaping from the maze and a reward of zero at all other times. The task seems to break down naturally into episodes—the successive runs through the maze—so you decide to treat it as an episodic task, where the goal is to maximize expected total reward (3.7). After running the learning agent for a while, you find that it is showing no improvement in escaping from the maze. What is going wrong? Have you effectively communicated to the agent what you want it to achieve?

Answer: Notice that the returns in this task are always $+1$ no matter how long it takes the agent to exit the maze. Thus there is no incentive in the problem formulation to find short paths: all are equally good. If you want the agent to find short paths, then this formulation does not capture this, and in that sense you have not communicated your desire to the agent. □

Exercise 3.8 Suppose $\gamma = 0.5$ and the following sequence of rewards is received $R_1 = -1$, $R_2 = 2$, $R_3 = 6$, $R_4 = 3$, and $R_5 = 2$, with $T = 5$. What are G_0, G_1, ..., G_5? Hint: Work backwards.

Answer: The trick to making this easy is to use (3.9) and to compute the returns backwards, starting with G_5, then G_4, and so on. The reward sequence is -1, 2, 6, 3, 2, so the returns are

$G_5 = 2$
$G_4 = 3 + 0.5 \cdot G_5 = 4$
$G_3 = 6 + 0.5 \cdot G_4 = 8$
$G_2 = 2 + 0.5 \cdot G_3 = 6$
$G_1 = -1 + 0.5 \cdot G_2 = 2$ □

Exercise 3.9 Suppose $\gamma = 0.9$ and the reward sequence is $R_1 = 2$ followed by an infinite sequence of 7s. What are G_1 and G_0?

Answer: Using (3.10), we get $G_1 = 70$. Using (3.9) we get $G_0 = 2 + 0.9 \cdot G_1 = 65$. □

Exercise 3.10 Prove the second equality in (3.10).

Answer: Because all rewards are 1, all the returns are equal: $G_t = G_{t+1} \doteq G$. Starting from (3.9), then, we have:

$$\begin{aligned} G_t &= R_{t+1} + \gamma G_{t+1} \qquad (3.9) \\ G &= 1 + \gamma G \\ G - \gamma G &= 1 \\ G(1-\gamma) &= 1 \\ G &= \frac{1}{1-\gamma}. \end{aligned}$$

QED. □

Exercise 3.11 If the current state is S_t, and actions are selected according to stochastic policy π, then what is the expectation of R_{t+1} in terms of π and the four-argument function p (3.2)?

Answer:

$$\begin{aligned} \mathbb{E}[R_{t+1} \mid S_t, A_t \sim \pi] &= \sum_a \pi(a|S_t) \mathbb{E}[R_{t+1} \mid S_t, A_t = a] \\ &= \sum_a \pi(a|S_t) \sum_{s',r} p(s', r \,|\, s, a) r \end{aligned}$$

□

Exercise 3.12 Give an equation for v_π in terms of q_π and π.

Answer:

$$v_\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s) q_\pi(s, a)$$

□

Exercise 3.13 Give an equation for q_π in terms of v_π and the four-argument p.

Answer:

$$q_\pi(s, a) = \sum_{s \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \,|\, s, a) \left[r + \gamma v_\pi(s') \right]$$

□

Exercise 3.14 The Bellman equation (3.14) must hold for each state for the value function v_π shown in Figure 3.2 (right) of Example 3.5. Show numerically that this equation holds for the center state, valued at +0.7, with respect to its four neighboring states, valued at +2.3, +0.4, −0.4, and +0.7. (These numbers are accurate only to one decimal place.)

Answer:

$$\begin{aligned} v_\pi(s) &= \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \big[r + \gamma v_\pi(s') \big] \qquad (3.14) \\ &= \sum_a 0.25 \cdot \Big[0 + 0.9 \cdot (2.3 + 0.4 - 0.4 + 0.7) \Big] \\ &= 0.25 \cdot [0.9 \cdot 3.0] = 0.675 \approx 0.7 \end{aligned}$$

□

Exercise 3.15 In the gridworld example, rewards are positive for goals, negative for running into the edge of the world, and zero the rest of the time. Are the signs of these rewards important, or only the intervals between them? Prove, using (3.8), that adding a constant c to all the rewards adds a constant, v_c, to the values of all states, and thus does not affect the relative values of any states under any policies. What is v_c in terms of c and γ?

Answer: The values are the expected values of the returns, and the returns are all changed by a constant:

$$G_t = \sum_{k=0}^{\infty} \gamma^k (R_{t+k+1} + c) = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} + \sum_{k=0}^{\infty} \gamma^k c$$

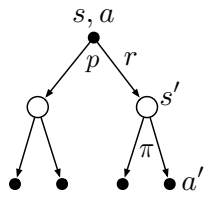
$$v_c = \sum_{k=0}^{\infty} \gamma^k c = c \sum_{k=0}^{\infty} \gamma^k = \frac{c}{1-\gamma}$$

□

Exercise 3.16 Now consider adding a constant c to all the rewards in an episodic task, such as maze running. Would this have any effect, or would it leave the task unchanged as in the continuing task above? Why or why not? Give an example.

Answer: The absolute values of the rewards do matter in the total reward case because they strongly affect the relative values of trajectories of different lengths. For example, consider a maze task where the object is to reach the terminal state in the smallest number of steps. We can arrange for this by setting the reward on each step to be -1. However, if we add the constant $+2$ to all the rewards, then the system obtains a $+1$ reward for every step; instead of finding the shortest way out it will find the longest way to stay in! □

Exercise 3.17 What is the Bellman equation for action values, that is, for q_π? It must give the action value $q_\pi(s,a)$ in terms of the action values, $q_\pi(s',a')$, of possible successors to the state–action pair (s,a). Hint: the backup diagram to the right corresponds to this equation. Show the sequence of equations analogous to (3.14), but for action values.



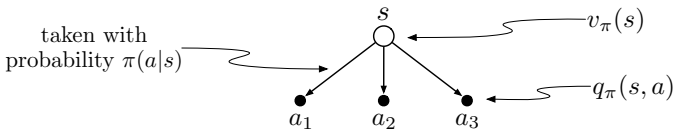
q_π backup diagram

Answer:

$$\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] && \text{(by (3.9))} \\
&= \sum_{s'} \sum_{r} p(s', r \mid s, a) \Big[r + \gamma \mathbb{E}_\pi\big[G_{t+1} \mid S_{t+1} = s'\big] \Big] \\
&= \sum_{s', r} p(s', r \mid s, a) \Big[r + \gamma \sum_{a'} \pi(a' \mid s') \mathbb{E}_\pi\big[G_{t+1} \mid S_{t+1} = s', A_{t+1} = a'\big] \Big] \\
&= \sum_{s', r} p(s', r \mid s, a) \Big[r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s', a') \Big]
\end{aligned}$$

□

Exercise 3.18 The value of a state depends on the values of the actions possible in that state and on how likely each action is to be taken under the current policy. We can think of this in terms of a small backup diagram rooted at the state and considering each possible action:



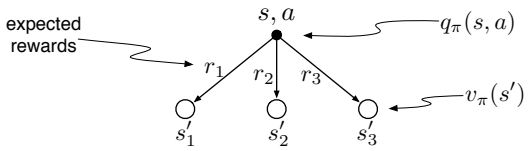
Give the equation corresponding to this intuition and diagram for the value at the root node, $v_\pi(s)$, in terms of the value at the expected leaf node, $q_\pi(s,a)$, given $S_t = s$. This equation should include an expectation conditioned on following the policy, π. Then give a second equation in which the expected value is written out explicitly in terms of $\pi(a|s)$ such that no expected value notation appears in the equation.

Answer:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[q_\pi(S_t, A_t) \mid S_t = s] \\ &= \sum_a \pi(a|s) q_\pi(s,a) \end{aligned}$$

□

Exercise 3.19 The value of an action, $q_\pi(s,a)$, depends on the expected next reward and the expected sum of the remaining rewards. Again we can think of this in terms of a small backup diagram, this one rooted at an action (state–action pair) and branching to the possible next states:



Give the equation corresponding to this intuition and diagram for the action value, $q_\pi(s,a)$, in terms of the expected next reward, R_{t+1}, and the expected next state value, $v_\pi(S_{t+1})$, given that $S_t = s$ and $A_t = a$. This equation should include an expectation but *not* one conditioned conditioned on following the policy. Then give a second equation, writing out the expected value explicitly in terms of $p(s', r \,|\, s, a)$ defined by (3.2), such that no expected value notation appears in the equation.

Answer:

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r \,|\, s, a) \big[r + \gamma v_\pi(s') \big] \end{aligned}$$

□

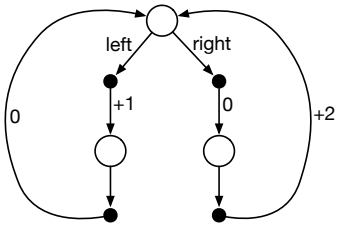
Exercise 3.20 Draw or describe the optimal state-value function for the golf example.

Answer: The optimal state-value function gives the value of each state given optimal behavior. It is -1 anywhere on the green, then -2 up to about midway down the fairway, then -3. In other words, it is exactly the function formed by taking the max of the two functions shown in Figure 3.3. □

Exercise 3.21 Draw or describe the contours of the optimal action-value function for putting, $q_*(s, \texttt{putter})$, for the golf example.

Answer: This function gives the value of each state if we putt once, then afterwards behave optimally. It is -1 anywhere on the green, -2 anywhere within reach of a putt from the green, then -3 up to anywhere from which we can putt to within the -2 contour of $q_*(s, \texttt{driver})$ shown in Figure 3.3. Similarly, it is -4 anywhere from which we can putt to within the -3 contour of $q_*(s, \texttt{driver})$ shown in Figure 3.3. The sand traps are both -3 because the initial putt does nothing and then we drive out of the trap and putt to the hole. □

Exercise 3.22 Consider the continuing MDP shown on to the right. The only decision to be made is that in the top state, where two actions are available, left and right. The numbers show the rewards that are received deterministically after each action. There are exactly two deterministic policies, π_{left} and π_{right}. What policy is optimal if $\gamma = 0$? If $\gamma = 0.9$? If $\gamma = 0.5$?



Answer: If $\gamma = 0$, then delayed rewards do not count at all. The left action has value 1 from the top state, and the right action has value 0; π_{left} is optimal.

If $\gamma = 0.9$, then delayed rewards are given substantial weight. Under π_{left}, the sequence of rewards from the top state is 1, 0, 1, 0, 1, 0, ..., and the corresponding return and value is $1 + \gamma^2 + \gamma^4 + \cdots = 1/(1 - \gamma^2) \approx 5.26$. Under π_{right}, the sequence of rewards from the top state is 0, 2, 0, 2, 0, 2, ..., and the corresponding return and value is $2\gamma + 2\gamma^3 + 2\gamma^5 + \cdots = 2\gamma(1 + \gamma^2 + \gamma^4 + \cdots) = 2\gamma/(1 - \gamma^2)$, which is clearly better as $2\gamma > 1$. Only π_{right} is optimal.

If $\gamma = 0.5$, we have a borderline case. The returns are exactly as computed above, but now we have $2\gamma = 1$, and thus the two actions have the same value. Both policies are optimal. □

Exercise 3.23 Give the Bellman equation for q_* for the recycling robot.

Answer: In general, the Bellman equation for q_* is

$$q_*(s,a) = \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma \max_{a'} q_*(s', a')\big]$$

For the recycling robot, there are 5 possible state-action pairs, so the Bellman equation for q_* is actually 5 equations. Abbreviate the states `high` and `low`, and the actions `search`, `wait`, and `recharge` respectively by `h`, `l`, `s`, `w`, and `r`.

$$\begin{aligned}
q_*(\texttt{h},\texttt{s}) &= \alpha[r_{\texttt{search}} + \gamma \max\{q_*(\texttt{h},\texttt{s}), q_*(\texttt{h},\texttt{w})\}] \\
&\quad + (1-\alpha)[r_{\texttt{search}} + \gamma \max\{q_*(\texttt{l},\texttt{s}), q_*(\texttt{l},\texttt{w}), q_*(\texttt{l},\texttt{r})\}] \\
q_*(\texttt{l},\texttt{s}) &= (1-\beta)[-3 + \gamma \max\{q_*(\texttt{h},\texttt{s}), q_*(\texttt{h},\texttt{w})\}] \\
&\quad + \beta[r_{\texttt{search}} + \gamma \max\{q_*(\texttt{l},\texttt{s}), q_*(\texttt{l},\texttt{w}), q_*(\texttt{l},\texttt{r})\}] \\
q_*(\texttt{h},\texttt{w}) &= r_{\texttt{wait}} + \gamma \max\{q_*(\texttt{h},\texttt{s}), q_*(\texttt{h},\texttt{w})\} \\
q_*(\texttt{l},\texttt{w}) &= r_{\texttt{wait}} + \gamma \max\{q_*(\texttt{l},\texttt{s}), q_*(\texttt{l},\texttt{w}), q_*(\texttt{l},\texttt{r})\} \\
q_*(\texttt{l},\texttt{r}) &= \gamma \max\{q_*(\texttt{h},\texttt{s}), q_*(\texttt{h},\texttt{w})\}
\end{aligned}$$

□

Exercise 3.24 Figure 3.5 gives the optimal value of the best state of the gridworld as 24.4, to one decimal place. Use your knowledge of the optimal policy and (3.8) to express this value symbolically, and then to compute it to three decimal places.

Answer: Following the optimal policy from the best state the agent receives a first reward of +10 and then four rewards of 0, then the sequence repeats. The optimal value of the state then must be

$$\begin{aligned}
v_*(s) &= \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; S_t = s\right] \\
&= 10 + \gamma 0 + \gamma^2 0 + \gamma^3 0 + \gamma^4 0 + \gamma^5 10 + \gamma^6 0 + \gamma^7 + \gamma^8 0 + \gamma^9 0 + \gamma^{10} 10 + \cdots \\
&= \gamma^0 10 + \gamma^5 10 + \gamma^{10} 10 + \cdots \\
&= 10 \sum_{k=0}^{\infty} (\gamma^5)^k \\
&= \frac{10}{1-\gamma^5}
\end{aligned}$$

Substituting $\gamma = 0.9$, we obtain $v_*(s) = 24.419$. □

Exercise 3.25 Give an equation for v_* in terms of q_*.

Answer:

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_*(s,a) \quad \text{for all } s \in \mathcal{S} \qquad \square$$

Exercise 3.26 Give an equation for q_* in terms of v_* and the four-argument p.

Answer:

$$q_*(s,a) = \sum_{s' \in \mathcal{S}^+} \sum_{r \in \mathcal{R}} p(s', r \,|\, s, a)\big(r + \gamma v_*(s')\big) \quad \text{for all } s \in \mathcal{S}, a \in \mathcal{A}(s) \qquad \square$$

Exercise 3.27 Give an equation for π_* in terms of q_*.

Answer:

$$\pi_*(s) = \operatorname*{arg\,max}_{a \in \mathcal{A}(s)} q_*(s,a) \quad \text{for all } s \in \mathcal{S} \qquad \square$$

Exercise 3.28 Give an equation for π_* in terms of v_* and the four-argument p.

Answer:

$$\pi_*(s) = \operatorname*{arg\,max}_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}^+} \sum_{r \in \mathcal{R}} p(s', r \,|\, s, a)\big(r + \gamma v_*(s')\big) \quad \text{for all } s \in \mathcal{S} \qquad \square$$

Exercise 3.29 Rewrite the four Bellman equations for the four value functions (v_π, v_*, q_π, and q_*) in terms of the three argument function p (3.4) and the two-argument function r (3.5).

Answer:

$$v_\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}^+} p(s' \,|\, s, a) v_\pi(s') \right) \quad \text{for all } s \in \mathcal{S}$$

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}^+} p(s' \,|\, s, a) v_*(s') \right) \quad \text{for all } s \in \mathcal{S}$$

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}^+} p(s' \,|\, s, a) \sum_{a' \in \mathcal{A}(s')} \pi(a'|s') q_\pi(s', a') \quad \text{for all } s \in \mathcal{S}, a \in \mathcal{A}(s)$$

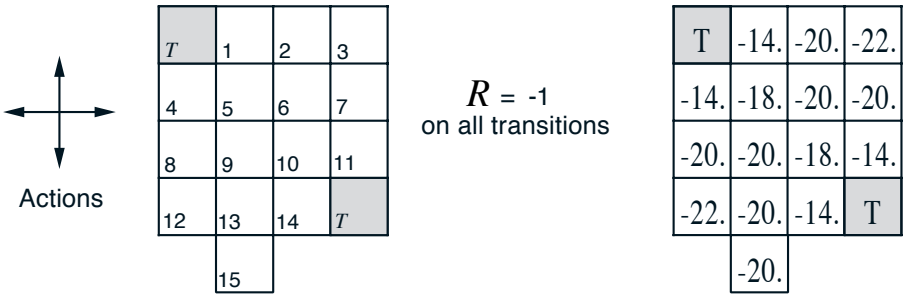
$$q_*(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}^+} p(s' \,|\, s, a) \max_{a' \in \mathcal{A}(s')} q_*(s', a') \quad \text{for all } s \in \mathcal{S}, a \in \mathcal{A}(s) \qquad \square$$

Answers to Exercises
Reinforcement Learning: Chapter 4

Exercise 4.1 In Example 4.1, if π is the equiprobable random policy, what is $q_\pi(11, \texttt{down})$? What is $q_\pi(7, \texttt{down})$?

Answer: $q_\pi(11, \texttt{down}) = -1$. $q_\pi(7, \texttt{down}) = -15$. □

Exercise 4.2 In Example 4.1, suppose a new state 15 is added to the gridworld just below state 13, and its actions, `left`, `up`, `right`, and `down`, take the agent to states 12, 13, 14, and 15, respectively. Assume that the transitions *from* the original states are unchanged. What, then, is $v_\pi(15)$ for the equiprobable random policy? Now suppose the dynamics of state 13 are also changed, such that action `down` from state 13 takes the agent to the new state 15. What is $v_\pi(15)$ for the equiprobable random policy in this case?



Answer: In the case where none of the other states have their outgoing transitions changed, then the new state's value under the random policy is

$$\begin{aligned} v_\pi(15) &= \mathbb{E}_\pi[R_{t+1} + v_\pi(S_{t+1}) \mid S_t = s] \\ &= -1 + \frac{1}{4}v_\pi(12) + \frac{1}{4}v_\pi(13) + \frac{1}{4}v_\pi(14) + \frac{1}{4}v_\pi(15) \end{aligned}$$

Plugging in the asymptotic values for $v_\infty = v_\pi$ for states 12, 13, and 14 from Figure 4.1 (and above, right) and solving for $v_\pi(15)$ yields

$$v_\pi(15) = -1 - \frac{1}{4}22 - \frac{1}{4}20 - \frac{1}{4}14 - \frac{1}{4}v_\pi(15)$$

$$v_\pi(15)\left(1 - \frac{1}{4}\right) = -15$$

$$v_\pi(15) = -20$$

If the dynamics of state 13 also change, then it turns out that the answer is the same! This can be most easily seen by hypothesizing that $v_\pi(15) = -20$ and then checking that all states still satisfy the Bellman equation for v_π. □

Exercise 4.3 What are the equations analogous to (4.3), (4.4), and (4.5) for the action-value function q_π and its successive approximation by a sequence of functions $q_0, q_1, q_2, \ldots$?

Answer:

$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a] \tag{4.3}$$

$$= \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a') \right] \tag{4.4}$$

$$q_{k+1}(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_k(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

$$= \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \sum_{a'} \pi(a'|s') q_k(s', a') \right] \tag{4.5}$$

□

Exercise 4.4 The policy iteration algorithm on page 80 has a subtle bug in that it may never terminate if the policy continually switches between two or more policies that are equally good. This is ok for pedagogy, but not for actual use. Modify the pseudocode so that convergence is guaranteed.

Answer: TBD. □

Exercise 4.5 How would policy iteration be defined for action values? Give a complete algorithm for computing q_*, analogous to that on page 80 for computing v_*. Please pay special attention to this exercise, because the ideas involved will be used throughout the rest of the book.

Answer: Just as for state values, we would have an alternation of policy improvement and policy evalution steps, only this time in q rather than in v:

$$\pi_0 \xrightarrow{\text{PE}} q_{\pi_0} \xrightarrow{\text{PI}} \pi_1 \xrightarrow{\text{PE}} q_{\pi_1} \xrightarrow{\text{PI}} \pi_2 \xrightarrow{\text{PE}} \cdots \xrightarrow{\text{PI}} \pi_* \xrightarrow{\text{PE}} q_*$$

Each policy evaluation step, $\pi_i \xrightarrow{\text{PE}} q_{\pi_i}$, would involve multiple iterations of equation (4.5) above, until convergence, or some other way of computing q_{π_i}. Each policy improvement step, $q_{\pi_i} \xrightarrow{\text{PI}} \pi_{i+1}$, would be a greedification with respect to q_{π_i}, i.e.:

$$\pi_{i+1}(s) = \arg\max_a q_{\pi_i}(s, a).$$

A boxed algorithm for policy iteration to q_* is:

Policy iteration for action values (using iterative policy evaluation)

1. Initialization
   $Q(s, a) \in \mathbb{R}$ arbitrarily for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$
   $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$ and $a \in \mathcal{A}$:
   $q \leftarrow Q(s, a)$
   $Q(s, a) \leftarrow \sum_{s',r} p(s', r \,|\, s, a)\Big[r + \gamma Q(s', \pi(s'))\Big]$
   $\Delta \leftarrow \max(\Delta, |q - Q(s, a)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $a \leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a Q(s, a)$
   If $a \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return Q and π; else go to 2

In the "arg max" step, it is important that ties be broken in a consistent order. □

Exercise 4.6 Suppose you are restricted to considering only policies that are *ε-soft*, meaning that the probability of selecting each action in each state, s, is at least $\varepsilon/|\mathcal{A}(s)|$. Describe qualitatively the changes that would be required in each of the steps 3, 2, and 1, in that order, of the policy iteration algorithm for v_* (page 80).

Answer: Step 3, the policy improvement step, would have to be changed such that the new policy is not the deterministic greedy policy, but the closest ϵ-soft policy. That is, all non-greedy actions would be given the minimal probability, $\frac{\epsilon}{|\mathcal{A}(s)|}$, and all the rest of the probability would go to the greedy action. The check for termination would also need to be changed. Somehow we would have to check for a change in the action with the bulk of the probability.

Step 2, policy evalution, would need to be generalized to accomodate stochastic policies. A new equation analogous to (4.5) would be needed.

Step 1, initialization, would need be changed only to permit the initial policy to be stochastic. □

Exercise 4.8 Why does the optimal policy for the gambler's problem have such a curious form? In particular, for capital of 50 it bets it all on one flip, but for capital of 51 it does not. Why is this a good policy?

Answer: In this problem, with $p = 0.4$, the coin is biased against the gambler. Because of this, the gambler want to minimize his number of flips. If he makes many small bets he is likely to lose. Thus, with a stake of 50 he can bet it all and have a 0.4 probability of winning. On the other hand, with stake of 51 he can do slightly better. If he bets 1, then even if he loses he still has 50 and thus a 0.4 chance of winning. And if he wins he ends up with 52. With 52 he can bet 2 and maybe end up with 54 etc. In these cases there is a chance he can get up to 75 without ever risking it all on one bet, yet he can always fall back (if he loses) on one big bet. And if he gets to 75 he can safely bet 25, possibly winning in one, while still being able to fall back to 50. It is this sort of logic which causes such big changes in the policy with small changes in stake, particularly at multiples of the negative powers of two. □

Exercise 4.10 What is the analog of the value iteration update (4.10) for action values, $q_{k+1}(s, a)$?

Answer: Value iteration in action values is defined by

$$\begin{aligned} q_{k+1}(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_k(S_{t+1}, a') \;\middle|\; S_t = s, A_t = a\right] \\ &= \sum_{s',r} p(s', r \,|\, s, a)\left[r + \gamma \max_{a'} q_k(s', a')\right], \end{aligned}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$. For arbitrary q_0, the sequence $\{q_k\}$ converges to q_*. □

Answers to Exercises
Reinforcement Learning: Chapter 5

Exercise 5.1 Consider the diagrams on the right in Figure 5.1. Why does the estimated value function jump up for the last two rows in the rear? Why does it drop off for the whole last row on the left? Why are the frontmost values higher in the upper diagrams than in the lower?

Answer: The value function jumps up for the last two rows in the rear because these rows correspond to sums of 20 and 21, for which the policy is to stick, whereas for sums less than 20 the policy is to hit. Sticking is a much better action in this region of state space, so the values under the policy are much better for these states.

Value drops off for the leftmost row because these are the states for which the dealer is showing an ace. This is the worst case for the player because the dealer has a usable ace, giving him extra flexibility—in effect, two chances to get close to 21.

The values are generally higher in the upper diagrams because here the player has a usable ace and thus the extra flexibility. □

Exercise 5.2 Suppose every-visit MC was used instead of first-visit MC on the blackjack task. Would you expect the results to be very different? Why or why not?

Answer: The results would be exactly the same, because in this problem the same state is never visited more than once in a single episode. □

Exercise 5.3 What is the backup diagram for Monte Carlo estimation of q_π?

Answer:

□

Exercise 5.4 The pseudocode for Monte Carlo ES is inefficient because, for each state–action pair, it maintains a list of all returns and repeatedly calculates their mean. It would be more efficient to use techniques similar to those explained in Section 2.4 to maintain just the mean and a count (for each state–action pair) and update them incrementally. Describe how the pseudocode would be altered to achieve this.

Answer: The initialization section would be modified by replacing the line for the *Returns* with one that initialized a count $N(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$. Then, the second and third to last lines are replaced by:

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} \left[G - Q(S_t, A_t)\right]. \qquad \square$$

Exercise 5.5 Consider an MDP with a single nonterminal state and a single action that transitions back to the nonterminal state with probability p and transitions to the terminal state with probability $1-p$. Let the reward be $+1$ on all transitions, and let $\gamma = 1$. Suppose you observe one episode that lasts 10 steps, with a return of 10. What are the first-visit and every-visit estimators of the value of the nonterminal state?

Answer: First-visit estimator: $V(s) = 10$
Every-visit estimator:

$$V(s) = \frac{10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1}{10} = 5.5 \qquad \square$$

Exercise 5.6 What is the equation analogous to (5.6) for *action* values $Q(s,a)$ instead of state values $V(s)$, again given returns generated using b?

Answer: There are two changes needed. First, we need a notation $\mathcal{T}(s,a)$ for the set of time steps in which the state–action *pair* occurred (or first occurred in the episode for a first-visit method). Second, we need to correct for one less importance-sampling ratio, because, in an action-value estimate, we don't have to correct for the first action. Thus:

$$Q(s,a) \doteq \frac{\sum_{t \in \mathcal{T}(s,a)} \rho_{t+1:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s,a)} \rho_{t+1:T(t)-1}}. \qquad \square$$

Exercise 5.7 In learning curves such as those shown in Figure 5.3 error generally decreases with training, as indeed happened for the ordinary importance-sampling method. But for the weighted importance-sampling method error first increased and then decreased. Why do you think this happened?

Answer: This shape of the learning curve for the weighted importance sampling estimator is quite robust. Recall that the weighted importance sampling estimator is initially biased. I believe this bias is toward zero, where in fact the true answer lies, and that this results in low initial mean square error. $\square$

Exercise 5.8 The results with Example 5.5 and shown in Figure 5.4 used a first-visit MC method. Suppose that instead an every-visit MC method was used on the same problem. Would the variance of the estimator still be infinite? Why or why not?

Answer: TBD. $\square$

Exercise 5.9 Modify the algorithm for first-visit MC policy evaluation (Section 5.1) to use the incremental implementation for sample averages described in Section 2.4.

Answer:

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$N(s) \leftarrow 0$, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

$N(S_t) \leftarrow N(S_t) + 1$

$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}\left[G - V(S_t)\right]$

□

Exercise 5.10 Derive the weighted-average update rule (5.8) from (5.7). Follow the pattern of the derivation of the unweighted rule (2.3).

Answer: Here is one way to do it:

$$
\begin{aligned}
V_{n+1} &= \frac{\sum_{k=1}^{n} W_k G_k}{\sum_{k=1}^{n} W_k} && \text{(from (5.7))} \\
&= \frac{W_n G_n + \sum_{k=1}^{n-1} W_k G_k}{W_n + \sum_{k=1}^{n-1} W_k} \\
&= \frac{W_n G_n + \left(\sum_{k=1}^{n-1} W_k\right) V_n + W_n V_n - W_n V_n}{W_n + \sum_{k=1}^{n-1} W_k} \\
&= V_n + \frac{W_n G_n - W_n V_n}{\sum_{k=1}^{n} W_k} \\
&= V_n + \frac{W_n}{\sum_{k=1}^{n} W_k}[G_n - V_n] \\
&= V_n + \frac{W_n}{C_n}[G_n - V_n].
\end{aligned}
$$

□

Exercise 5.11 In the boxed algorithm for off-policy MC control, you may have been expecting the W update to have involved the importance-sampling ratio $\frac{\pi(A_t|S_t)}{b(A_t|S_t)}$, but instead it involves $\frac{1}{b(A_t|S_t)}$. Why is this nevertheless correct?

Answer: In this algorithm, π is a deterministic policy, so, for the action actually taken, its probability of being taken is always 1. □

***Exercise 5.13** Show the steps to derive (5.14) from (5.12).

Answer: TBD. □

***Exercise 5.14** Modify the algorithm for off-policy Monte Carlo control (page 111) to use the idea of the truncated weighted-average estimator (5.10). Note that you will first need to convert this equation to action values.

Answer: TBD. □

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:

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

(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. □

Answers to Exercises
Reinforcement Learning: Chapter 7

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 TD errors (again if the value estimates don't change) generalizing the earlier result.

Answer: We adopt the convention that $G_{t:t} \doteq V(S_t)$. The recursive relationship among successive returns,

$$G_{t:t+n} = R_{t+1} + \gamma G_{t+1:t+n},$$

then holds even for $n = 1$. If $t + n > T$, then all of our n-step returns (Gs) are in fact full returns, and the earlier result applies. Thus, here we can assume $t + n \leq T$. Then we have

$$\begin{aligned}
G_{t:t+n} - V(S_t) &= R_{t+1} + \gamma G_{t+1:t+n} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) \qquad \text{(from (3.9))} \\
&= \delta_t + \gamma\big(G_{t+1:t+n} - V(S_{t+1})\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\big(G_{t+2:t+n} - V(S_{t+2})\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{n-1}\delta_{t+n-1} + \gamma^n\big(G_{t+n:t+n} - V(S_{t+n})\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{n-1}\delta_{t+n-1} + \gamma^n\big(V(S_{t+n}) - V(S_{t+n})\big) \\
&= \sum_{k=t}^{t+n-1} \gamma^{k-t}\delta_k,
\end{aligned}$$

Q.E.D. □

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?

Answer: A larger random walk task was used to better show off the advantages of longer backups. Remember that the point of eligibility traces is to backup farther than a single step per experience. This has its largest effect when the episodes last many steps. Thus, a smaller task would have presumably benefited less from long backups, and best performance would have been shifted to a smaller value of n. The change in the left-side outcome from 0 to -1 should have absolutely no effect, assuming the initial values were also shifted from 0.5 to 0, as they were. The only effect would have been an overall increase in the level of error due to the greater span from -1 to 1. □

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

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

Answer: First, assume $t + n < T$. Then (7.6) can be written

$$\begin{aligned}
G_{t:t+n} &= Q_{t-1}(S_t, A_t) + \sum_{k=t}^{t+n-1} \gamma^{k-t} \left[R_{k+1} + \gamma Q_k(S_{k+1}, A_{k+1}) - Q_{k-1}(S_k, A_k) \right] \\
&= Q_{t-1}(S_t, A_t) + \gamma^0 \left[R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) - Q_{t-1}(S_t, A_t) \right] \\
&\quad + \gamma^1 \left[R_{t+2} + \gamma Q_{t+1}(S_{t+2}, A_{t+2}) - Q_t(S_{t+1}, A_{t+1}) \right] \\
&\quad + \gamma^2 \left[R_{t+3} + \gamma Q_{t+2}(S_{t+3}, A_{t+3}) - Q_{t+1}(S_{t+2}, A_{t+2}) \right] \\
&\quad \vdots \\
&\quad + \gamma^{n-2} \left[R_{t+n-1} + \gamma Q_{t+n-2}(S_{t+n-1}, A_{t+n-1}) - Q_{t+n-3}(S_{t+n-2}, A_{t+n-2}) \right] \\
&\quad + \gamma^{n-1} \left[R_{t+n} + \gamma Q_{t+n-1}(S_{t+n}, A_{t+n}) - Q_{t+n-2}(S_{t+n-1}, A_{t+n-1}) \right].
\end{aligned}$$

Now we notice that all the Q terms cancel except for the first one on the last line, and that what remains is exactly Sarsa's n-step return (7.4). This proves (7.6) for the case in which $t + n < T$. If $t + n \geq T$, then a similar expansion yields

$$\begin{aligned}
G_{t:t+n} &= Q_{t-1}(S_t, A_t) + \sum_{k=t}^{T-1} \gamma^{k-t} \left[R_{k+1} + \gamma Q_k(S_{k+1}, A_{k+1}) - Q_{k-1}(S_k, A_k) \right] \\
&= Q_{t-1}(S_t, A_t) + \gamma^0 \left[R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) - Q_{t-1}(S_t, A_t) \right] \\
&\quad + \gamma^1 \left[R_{t+2} + \gamma Q_{t+1}(S_{t+2}, A_{t+2}) - Q_t(S_{t+1}, A_{t+1}) \right] \\
&\quad + \gamma^2 \left[R_{t+3} + \gamma Q_{t+2}(S_{t+3}, A_{t+3}) - Q_{t+1}(S_{t+2}, A_{t+2}) \right] \\
&\quad \vdots \\
&\quad + \gamma^{T-t-2} \left[R_{t+n-1} + \gamma Q_{T-2}(S_{T-1}, A_{T-1}) - Q_{T-3}(S_{T-2}, A_{T-2}) \right] \\
&\quad + \gamma^{T-t-1} \left[R_{t+n} + \gamma Q_{T-1}(S_T, A_T) - Q_{T-2}(S_{T-1}, A_{T-1}) \right].
\end{aligned}$$

Again, all the Q terms cancel except the next to the last one, which in this case is an action value at the terminal state, which we take always to be zero, leaving only the reward terms of the conventional return G_t. Q.E.D. □

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

Answer: TBD. □

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

Answer: TBD. □

***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.

Answer: TBD. □

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.

Answer: TBD. □

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).

Answer: TBD. □

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).

Answer:

$$\begin{aligned}
G_{t:t+n} &\doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q(S_{t+1}, a) \;+\; \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+n} \\
&= Q(S_t, A_t) + \delta_t - \gamma \pi(A_{t+1}|S_{t+1}) Q(S_{t+1}, A_{t+1}) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+n} \\
&= Q(S_t, A_t) + \delta_t + \gamma \pi(A_{t+1}|S_{t+1}) \big(G_{t+1:t+n} - Q(S_{t+1}, A_{t+1})\big) \\
&= Q(S_t, A_t) + \delta_t + \gamma \pi(A_{t+1}|S_{t+1}) \delta_{t+1} + \gamma^2 \pi(A_{t+1}|S_{t+1}) \pi(A_{t+2}|S_{t+2}) \delta_{t+2} + \cdots \\
&= Q(S_t, A_t) + \sum_{k=t}^{\min(t+n-1,T-1)} \delta_k \prod_{i=t+1}^{k} \gamma \pi(A_i|S_i).
\end{aligned}$$

□

Reinforcement Learning: Chapter 8
Answers to Exercises

Exercise 8.1 The nonplanning method looks particularly poor in Figure 8.3 because it is a one-step method; a method using multi-step bootstrapping would do better. Do you think one of the multi-step bootstrapping methods from Chapter 7 could do as well as the Dyna method? Explain why or why not.

Answer: A multi-step bootstrapping method is unlikely to do as well as a method based on learning a model and then planning, like a Dyna method. Multi-step methods can pass credit back farther along a trajectory, but still do not make full use of every experienced transition. For example, a model-based method can use the knowledge gained from a transition occurring many, many time steps before reaching the goal. This information is added to the model and then can be used later. Multi-step methods, on the other hand, must wait until the actual TD errors occur and at that time can only pass back credit to recently taken actions. Multi-step methods are also more prone to learning too much from a few early trials and then not exploring enough later. □

Exercise 8.2 Why did the Dyna agent with exploration bonus, Dyna-Q+, perform better in the first phase as well as in the second phase of the blocking and shortcut experiments?

Answer: Dyna-Q+ performed better early on because the exploration bonus caused the method to explore more effectively. Dyna-Q+ actively sought out the areas that it had not yet experienced and thus found the goal more quickly and performed better earlier. □

Exercise 8.3 Careful inspection of Figure 8.5 reveals that the difference between Dyna-Q+ and Dyna-Q narrowed slightly over the first part of the experiment. What is the reason for this?

Answer: Dyna-Q+ is at a slight disadvantage in that its exploration bonus causes it to constantly explore. This worked to its advantage earlier in that it helped it to discover the change in the environment faster. But now that the environment is stable, the constant exploring actually causes Dyna-Q+ to do somewhat worse. Such is the cost of persistent curiousity. □

Exercise 8.5 How might the tabular Dyna-Q algorithm shown on page 164 be modified to handle stochastic environments? How might this modification perform poorly on changing environments such as considered in this section? How could the algorithm be modified to handle stochastic environments *and* changing environments?

Answer: The model could be modified to handle stochastic environments by keeping track, for each state and action, how many times each possible subsequent state occurred, along with the average reward over all subsequent states. One count and one average reward would be updated in place of step (e). Then, in place of the third substep of (f), R, S' would be generated as the stored average reward for S, A and a sample next state generated with probabilities proportional to the counts for S, A. The rest of the algorithm could be unchanged.

If the environment changed, this algorithm could be very slow to adapt, as it would essentially be averaging together the environment's old and new behavior, as if it was just stochastic variation. If the environment consistently behaved one way on a thousand steps, then consistently another way for the next thousand, then even after the second thousand

this method would think there was a 0.5 chance of the enviroment behaving the old way on the next step.

To handle both stochastic *and* changing environments the model learning process would require some sort of fading, forgetful model. We would introduce a new parameter to indicate how fast the agent should adapt to changes. The system would maintain estimates for each S, A of the relative probability of going to each next state (and of the reward). On experiencing a transition $S, A \to S', R$, all the previous transition probabilities for S, A would be reduced and the transition probability for S' would be incremented. The estimate of the expected reward would be updated similarly. Obviously, this is just a sketch of the idea of a proper algorithm. The unbiased constant-step-size trick from Exercise 2.7 would be useful here. □

Exercise 8.6 The analysis above assumed that all of the b possible next states were equally likely to occur. Suppose instead that the distribution was highly skewed, that some of the b states were much more likely to occur than most. Would this strengthen or weaken the case for sample updates over expected updates? Support your answer.

Answer: I would think a highly skewed distribution would actually strengthen the case for sample backups. For example, suppose the distribution was so skewed that it was deterministic. Then one sample would get the same result as a full backup that adds up all outcomes (weighted all except one with a zero probability). Or suppose 99% of the probability went to one outcome, with thousands of other outcomes sharing the remaining 1%. Then sample backups would rapidly sample this predominant outcome and quickly and correctly assess the result. Full backups would still have to spend equal time with all the outcomes, even though many of them are of very low probability. This should strengthen the computational advantage of sampling. □

Exercise 8.7 Some of the graphs in Figure 8.8 seem to be scalloped in their early portions, particularly the upper graph for $b = 1$ and the uniform distribution. Why do you think this is? What aspects of the data shown support your hypothesis?

Answer: In the uniform case, particularly early in planning, most backups have little effect on the greedy policy from the start state because most are for state-action pairs that are far from the start state, that are very unlikely to be reached before termination. The backups that will have the greatest effect are those near the start state, particularly those of the state–actions pairs in which the start state is the state. This suggests that the upticks in the improvement of the greedy policy are due to these states coming up in the cycle through states. This is possible because the start state is a special, distinguished state; perhaps it is the first state considered in the cycle. This hypothesis is strengthened by the observation that the upticks seem to occur every 2000 backups in the upper graph for $b = 1$. In this case there are 1000 states and 2 actions, thus 2000 state-action pairs to be backed up in one cycle.

But I guess it is still not clear why there should seem to be real curves here, not just upticks every 2000 backups. □

Reinforcement Learning: Chapter 9
Answers to Exercises

Exercise 9.1 Show that tabular methods such as presented in Part I of this book are a special case of linear function approximation. What would the feature vectors be?

Answer: The feature vectors would have as many components as there are states, $d = |\mathcal{S}|$, and all components would be zero except for that corresponding to the current state, which would be 1. The gradient methods presented in this chapter would then be exactly equivalent to the corresponding tabular methods in Part I. □

Exercise 9.2 Why does (9.17) define $(n+1)^k$ distinct features for dimension k?

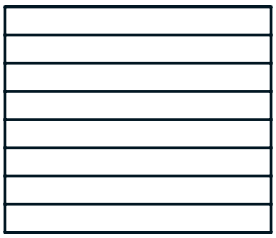
Answer: Defining a feature i requires choosing k numbers, $c^{i,j}$, each having $n+1$ possible values. By the counting principle of combinatorics, this can be done in $(n+1)^k$ different ways. □

Exercise 9.3 What n and $c_{i,j}$ produce the feature vectors $\mathbf{x}(s) = (1, s_1, s_2, s_1 s_2, s_1^2, s_2^2, s_1 s_2^2, s_1^2 s_2, s_1^2 s_2^2)^\top$?

Answer: The order $n = 2$, so there are 9 different features. Feature x_1 is $1 = s_1^0 s_2^0$, so $c^{1,1} = 0$ and $c^{1,2} = 0$. Feature x_2 is $s_1 = s_1^1 s_2^0$, so $c^{2,1} = 1$ and $c^{2,2} = 0$. Feature x_3 is $s_2 = s_1^0 s_2^1$, so $c^{3,1} = 0$ and $c^{3,2} = 1$. Feature $x_4 = s_1 s_2 = s_1^1 s_2^1$, so $c^{4,1} = 1$ and $c^{4,2} = 1$. Feature $x_5 = s_1^2 = s_1^2 s_2^0$, so $c^{5,1} = 2$ and $c^{5,2} = 0$. And so on, with feature $x_9 = s_1^2 s_2^2$, so $c^{9,1} = 2$ and $c^{9,2} = 2$. □

Exercise 9.4 Suppose we believe that one of two state dimensions is more likely to have an effect on the value function than is the other, that generalization should be primarily across this dimension rather than along it. What kind of tilings could be used to take advantage of this prior knowledge?

Answer: There is no reason why the tilings have to be uniform in all dimensions. In a two dimensional case, tilings of the following sort would encourage generalization across the horizontal dimension and discourage it across the vertical dimension:



One could include some tilings like this *and* some tilings with vertical stripes *and* some with 2D grids. The number of tilings of each type will determine the tendency of the system to generalize in each way. □

Exercise 9.5 Suppose you are using tile coding to transform a seven-dimensional continuous state space into binary feature vectors to estimate a state value function $\hat{v}(s,\mathbf{w}) \approx v_\pi(s)$. You believe that the dimensions do not interact strongly, so you decide to use eight tilings of each dimension separately (stripe tilings), for $7 \times 8 = 56$ tilings. In addition, in case there are some pairwise interactions between the dimensions, you also take all $\binom{7}{2} = 21$ pairs of dimensions and tile each pair conjunctively with rectangular tiles. You make two tilings for each pair of dimensions, making a grand total of $21 \times 2 + 56 = 98$ tilings. Given these feature vectors, you suspect that you still have to average out some noise, so you decide that you want learning to be gradual, taking about 10 presentations with the same feature vector before learning nears its asymptote. What step-size parameter α should you use? Why?

Answer: According to (9.19), you should use $\alpha = \frac{1}{10 \cdot 98} \approx 0.001$. The time scale τ here is 10, and the norm of all feature vectors is $\mathbf{x}^\top \mathbf{x} = 98$. □

Reinforcement Learning: Chapter 10
Answers to Exercises

Exercise 10.1 We have not explicitly considered or given pseudocode for any Monte Carlo methods in this chapter. What would they be like? Why is it reasonable not to give pseudocode for them? How would they perform on the Mountain Car task?

Answer: Monte Carlo methods as we have treated them in this book learn only from complete episodes. Arguably, this alone makes them inappropriate for control problems, and that is why we don't consider Monte Carlo methods in this chapter. Because they don't learn during the episode, the episode could be arbitrarily bad, or last arbitrarily long, before any learning is done. The agent could burn its hand on the fire many times, yet would not learn to change its behavior until an episode ends. On the Mountain Car task, the agent's policy might as some point cause the next episode never to end, in which case a Monte Carlo method would never learn again. □

Exercise 10.2 Give pseudocode for semi-gradient one-step *Expected* Sarsa for control.

Answer:

Semi-gradient One-step Expected Sarsa for Control

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Input: a potentially weight-dependent policy parameterization $\pi : \mathcal{A} \times \mathcal{S} \times \mathbb{R}^d \to [0, 1]$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  $S \leftarrow$ initial state of episode
  Loop while S is not terminal (for each step of episode):
    Select action A as a function of S and $\mathbf{w}$ (e.g., ε-greedy)
    Take action A, observe R, S'
    If S' is terminal, then:
      $V_{\text{next}} \leftarrow 0$
    else:
      $V_{\text{next}} \leftarrow \sum_a \pi(a|S', \mathbf{w}) \hat{q}(S', a, \mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha \big[R + \gamma V_{\text{next}} - \hat{q}(S, A, \mathbf{w}) \big] \nabla \hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$

□

Exercise 10.3 Why do the results shown in Figure 10.4 have higher standard errors at large n than at small n?

Answer: The larger n is, the more random rewards are involved in the n-step return. There are special cases where longer n-step returns have less variance than shorter ones, but usually they have more variance because the variances of the individual rewards roughly add up or, as in the case of the Mountain Car task, because the returns are just larger. Recall that in the Mountain Car task most n-step returns are exactly $-n$. □

Exercise 10.4 Give pseudocode for a differential version of semi-gradient Q-learning.

Answer:

Differential semi-gradient Q-learning for estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step sizes $\alpha, \beta > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Initialize average reward estimate $\bar{R} \in \mathbb{R}$ arbitrarily (e.g., $\bar{R} = 0$)

Initialize state S
Loop while S is not terminal (for each step of episode):
  Select action A as a function of S and $\mathbf{w}$ (e.g., ε-greedy)
  Take action A, observe R, S'
  $\delta \leftarrow R - \bar{R} + \max_a \hat{q}(S', a, \mathbf{w}) - \hat{q}(S, A, \mathbf{w})$
  If $\hat{q}(S, A, \mathbf{w}) = \max_a \hat{q}(S, a, \mathbf{w})$, then: $\bar{R} \leftarrow \bar{R} + \beta\delta$
  $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\nabla\hat{q}(S, A, \mathbf{w})$
  $S \leftarrow S'$

This is a variation of the R-learning algorithm due to Anton Schwartz (1993). There are several possible variations, and it would be interesting to compare them experimentally. □

Exercise 10.5 What equations are needed (beyond 10.10) to specify the differential version of TD(0)?

Answer: We need the update rules for $\mathbf{w}$ and $\bar{R}$:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\delta_t\nabla\hat{v}(S_t,\mathbf{w}_t),$$

after (10.12), and

$$\bar{R}_{t+1} \doteq \bar{R}_t + \beta\delta_t,$$

after the boxed algorithm for differential semi-gradient Sarsa on page 251. □

It is best to do Exercise 10.7 prior to the following exercise.

Exercise 10.6 Consider a Markov reward process consisting of a ring of three states A, B, and C, with state transitions going deterministically around the ring. A reward of $+1$ is received upon arrival in A and otherwise the reward is 0. What are the differential values of the three states using (10.13)?

Answer: From (10.6) we have that $r(\pi) = \frac{1}{3}$.

$$\begin{aligned}
v_\pi(\mathsf{C}) &= \lim_{\gamma\to 1}\lim_{h\to\infty}\sum_{t=0}^{h}\gamma^t\Big(\mathbb{E}_\pi[R_{t+1}|S_0{=}\mathsf{A}] - \frac{1}{3}\Big) \\
&= \lim_{\gamma\to 1}\Big(\frac{2}{3} - \gamma\frac{1}{3} - \gamma^2\frac{1}{3} + \gamma^3\frac{2}{3} - \gamma^4\frac{1}{3} - \gamma^5\frac{1}{3} + \cdots\Big) \\
&= \lim_{\gamma\to 1}\frac{1}{3}\Big(\frac{2}{1-\gamma^3} - \gamma\frac{1}{1-\gamma^3} - \gamma^2\frac{1}{1-\gamma^3}\Big) \\
&= \lim_{\gamma\to 1}\frac{1}{3}\cdot\frac{2-\gamma-\gamma^2}{1-\gamma^3} \\
&= \lim_{\gamma\to 1}\frac{1}{3}\cdot\frac{-1-2\gamma}{-3\gamma^2} && \text{(by L'Hôpital's rule)} \\
&= \frac{1}{3}
\end{aligned}$$

$$\begin{aligned}
v_\pi(\mathsf{B}) &= \sum_a \pi(a|s)\sum_{r,s'} p(s', r\,|\,s, a)\Big[r - r(\pi) + v_\pi(\mathsf{C})\Big] && \text{(from the Bellman equation)} \\
&= \Big[0 - \frac{1}{3} + \frac{1}{3}\Big] \\
&= 0
\end{aligned}$$

$$\begin{aligned}
v_\pi(\mathsf{A}) &= \sum_a \pi(a|s)\sum_{r,s'} p(s', r\,|\,s, a)\Big[r - r(\pi) + v_\pi(\mathsf{B})\Big] && \text{(from the Bellman equation)} \\
&= \Big[0 - \frac{1}{3} + 0\Big] \\
&= -\frac{1}{3}
\end{aligned}$$

□

Exercise 10.7 Suppose there is an MDP that under any policy produces the deterministic sequence of rewards $+1, 0, +1, 0, +1, 0, \ldots$ going on forever. Technically, this is not allowed because it violates ergodicity; there is no stationary limiting distribution μ_π and the limit (10.7) does not exist. Nevertheless, the average reward (10.6) is well defined; What is it? Now consider two states in this MDP. From A, the reward sequence is exactly as described above, starting with a $+1$, whereas, from B, the reward sequence starts with a 0 and then continues with $+1, 0, +1, 0, \ldots$. The differential return (10.9) is not well defined for this case as the limit does not exist. To repair this, one could alternately define the value of a state as

$$v_\pi(s) \doteq \lim_{\gamma\to 1} \lim_{h\to\infty} \sum_{t=0}^{h} \gamma^t \Big(\mathbb{E}_\pi[R_{t+1} | S_0 = s] - r(\pi) \Big). \tag{10.3}$$

Under this definition, what are the values of states A and B?

Answer: The average reward, according to (10.6) is $r(\pi) = \frac{1}{2}$.

$$\begin{aligned}
v_\pi(\mathsf{A}) &= \lim_{\gamma\to 1} \lim_{h\to\infty} \sum_{t=0}^{h} \gamma^t \Big(\mathbb{E}_\pi[R_{t+1} | S_0 = \mathsf{A}] - \frac{1}{2} \Big) \\
&= \lim_{\gamma\to 1} \Big(\frac{1}{2} - \gamma \frac{1}{2} + \gamma^2 \frac{1}{2} - \gamma^3 \frac{1}{2} + \cdots \Big) \\
&= \lim_{\gamma\to 1} \frac{1}{2} \Big(\frac{1}{1-\gamma^2} - \gamma \frac{1}{1-\gamma^2} \Big) \\
&= \lim_{\gamma\to 1} \frac{1}{2} \cdot \frac{1-\gamma}{1-\gamma^2} \\
&= \lim_{\gamma\to 1} \frac{1}{2} \cdot \frac{-1}{-2\gamma} && \text{(by L'Hôpital's rule)} \\
&= \frac{1}{4}
\end{aligned}$$

$$\begin{aligned}
v_\pi(\mathsf{B}) &= \sum_a \pi(a|s) \sum_{r,s'} p(s', r \,|\, s, a) \Big[r - r(\pi) + v_\pi(\mathsf{B}) \Big] && \text{(from the Bellman equation)} \\
&= \Big[0 - \frac{1}{2} + \frac{1}{4} \Big] \\
&= -\frac{1}{4}
\end{aligned}$$

□

Exercise 10.8 The pseudocode in the box on page 251 updates $\bar{R}_t$ using δ_t as an error rather than simply $R_{t+1} - \bar{R}_t$. Both errors work, but using δ_t is better. To see why, consider the ring MRP of three states from Exercise 10.7. The estimate of the average reward should tend towards its true value of $\frac{1}{3}$. Suppose it was already there and was held stuck there. What would the sequence of $R_{t+1} - \bar{R}_t$ errors be? What would the sequence of δ_t errors be (using (10.10))? Which error sequence would produce a more stable estimate of the average reward if the estimate were allowed to change in response to the errors? Why?

Answer: On $\mathsf{C} \to \mathsf{A}$ transitions, $R_{t+1} - \bar{R}_t = 1 - \frac{1}{3} = \frac{2}{3}$, and on the other two kinds of transitions it is $-\frac{1}{3}$. Thus the sequence would be these three errors repeating one after the other ($\frac{2}{3}$, $-\frac{1}{3}$, $-\frac{1}{3}$, $\frac{2}{3}$, $-\frac{1}{3}$, $-\frac{1}{3}$, ...), averaging out to zero but still causing fluxuations in $\bar{R}$ of size depending on β.

Now consider the sequence of δ_t for the same situation, assuming the values have already converged to the values computed in the previous exercise. For $\mathsf{C} \to \mathsf{A}$ transitions, we have

$$\begin{aligned}
\delta_t &= R_{t+1} - \bar{R}_t + \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t) \\
&= 1 - \frac{1}{3} + v_\pi(\mathsf{A}) - v_\pi(\mathsf{C}) \\
&= 1 - \frac{1}{3} + (-\frac{1}{3}) - \frac{1}{3} \\
&= 0.
\end{aligned}$$

For $\mathsf{B} \to \mathsf{C}$ transitions, we have

$$\begin{aligned}
\delta_t &= R_{t+1} - \bar{R}_t + \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t) \\
&= 0 - \frac{1}{3} + v_\pi(\mathsf{C}) - v_\pi(\mathsf{B}) \\
&= 0 - \frac{1}{3} + \frac{1}{3} - 0 \\
&= 0.
\end{aligned}$$

Finally, for $\mathsf{A} \to \mathsf{B}$ transitions, we have

$$\begin{aligned}
\delta_t &= R_{t+1} - \bar{R}_t + \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t) \\
&= 0 - \frac{1}{3} + v_\pi(\mathsf{B}) - v_\pi(\mathsf{A}) \\
&= 0 - \frac{1}{3} + 0 - (-\frac{1}{3}) \\
&= 0.
\end{aligned}$$

Thus, all the errors would be zero. If all the approximate state values were started at their correct values, and then allowed to change, they would not change at all because all the errors would be zero. There would be no fluxuations at all for any choice of the step-size parameters α and β. For these reasons, we can expect the estimate of the average reward to be much more stable using the δ_t error. □

Exercise 10.9 In the differential semi-gradient n-step Sarsa algorithm, the step-size parameter on the average reward, β, needs to be quite small so that $\bar{R}$ becomes a good long-term estimate of the average reward. Unfortunately, $\bar{R}$ will then be biased by its initial value for many steps, which may make learning inefficient. Alternatively, one could use a sample average of the observed rewards for $\bar{R}$. That would initially adapt rapidly but in the long run would also adapt slowly. As the policy slowly changed, $\bar{R}$ would also change; the potential for such long-term nonstationarity makes sample-average methods ill-suited. In fact, the step-size parameter on the average reward is a perfect place to use the unbiased constant-step-size trick from Exercise 2.7. Describe the specific changes needed to the boxed algorithm for differential semi-gradient n-step Sarsa to use this trick.

Answer: TBD. □

Reinforcement Learning: Chapter 11
Answers to Exercises

Exercise 11.1 Convert the equation of n-step off-policy TD (7.9) to semi-gradient form. Give accompanying definitions of the return for both the episodic and continuing cases.

Answer: :

$$\mathbf{w}_{t+n} \doteq \mathbf{w}_{t+n-1} + \alpha\rho_t\rho_{t+1}\cdots\rho_{t+n-1}\left[G_{t:t+n} - \hat{v}(S_t,\mathbf{w}_{t+n-1})\right]\nabla\hat{v}(S_t,\mathbf{w}_{t+n-1}),$$

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1}R_{t+n} + \gamma^n\hat{v}(S_{t+n},\mathbf{w}_{t+n-1}), \qquad \text{(episodic)}$$

$$G_{t:t+n} \doteq R_{t+1} - \bar{R}_t + \cdots + R_{t+n} - \bar{R}_{t+n-1} + \hat{v}(S_{t+n},\mathbf{w}_{t+n-1}) \qquad \text{(continuing)}$$

□

***Exercise 11.2** Convert the equations of n-step $Q(\sigma)$ (7.11 and 7.17) to semi-gradient form. Give definitions that cover both the episodic and continuing cases.

Answer: :

$$\mathbf{w}_{t+n} \doteq \mathbf{w}_{t+n-1} + \alpha\rho_{t+1:t+n-1}\left[G_{t:t+n} - \hat{q}(S_t,A_t,\mathbf{w}_{t+n-1})\right]\nabla\hat{q}(S_t,A_t,\mathbf{w}_{t+n-1}),$$

where

$$\rho_{t:h} \doteq \prod_{k=t}^{\min(h,T-1)}\left(\sigma_k\frac{\pi(A_k|S_k)}{b(A_k|S_k)} + 1 - \sigma_k\right),$$

and

$$G_{t:t+n} \doteq \hat{q}(S_t,A_t,\mathbf{w}_{t-1}) + \sum_{k=t}^{\min(t+n-1,T-1)}\delta_k\prod_{i=t+1}^{k}\gamma\big[(1-\sigma_i)\pi(A_i|S_i) + \sigma_i\big],$$

with γ taken as 1 in the continuing case, and

$$\delta_t \doteq R_{t+1} + \gamma\left[\sigma_{t+1}\hat{q}(S_{t+1},A_{t+1},\mathbf{w}_t) + (1-\sigma_{t+1})\sum_a\pi(a|S_{t+1})\hat{q}(S_{t+1},a,\mathbf{w}_t)\right] - \hat{q}(S_t,A_t,\mathbf{w}_t),$$

(episodic)

$$\delta_t \doteq R_{t+1} - \bar{R}_t + \sigma_{t+1}\hat{q}(S_{t+1},A_{t+1},\mathbf{w}_t) + (1-\sigma_{t+1})\sum_a\pi(a|S_{t+1})\hat{q}(S_{t+1},a,\mathbf{w}_t) - \hat{q}(S_t,A_t,\mathbf{w}_t).$$

(continuing)

□

***Exercise 11.4** Prove (11.24). Hint: Write the $\overline{\text{RE}}$ as an expectation over possible states s of the expectation of the squared error given that $S_t = s$. Then add and subtract the true value of state s from the error (before squaring), grouping the subtracted true value with the return and the added true value with the estimated value. Then, if you expand the square, the most complex term will end up being zero, leaving you with (11.24).

Answer:

$$
\begin{aligned}
\overline{\text{RE}}(\mathbf{w}) &= \mathbb{E}\Big[\big(G_t - \hat{v}(S_t,\mathbf{w})\big)^2\Big] \\
&= \mathop{\mathbb{E}}_{s\sim\mu}\mathbb{E}\Big[\big(G_t - \hat{v}(s,\mathbf{w})\big)^2 \;\Big|\; S_t = s\Big] \\
&= \mathop{\mathbb{E}}_{s\sim\mu}\mathbb{E}\Big[\Big(\big(G_t - v_\pi(s)\big) + \big(v_\pi(s) - \hat{v}(s,\mathbf{w})\big)\Big)^2 \;\Big|\; S_t = s\Big] \\
&= \mathop{\mathbb{E}}_{s\sim\mu}\mathbb{E}\Big[\big(G_t - v_\pi(s)\big)^2 + \big(v_\pi(s) - \hat{v}(s,\mathbf{w})\big)^2 \;\Big|\; S_t = s\Big] \\
&\quad + \mathop{\mathbb{E}}_{s\sim\mu}\mathbb{E}\big[2\big(G_t - v_\pi(s)\big)\big(v_\pi(s) - \hat{v}(s,\mathbf{w})\big) \;\big|\; S_t = s\big] \\
&= \mathbb{E}\Big[\big(G_t - v_\pi(S_t)\big)^2\Big] + \mathbb{E}\Big[\big(v_\pi(S_t) - \hat{v}(S_t,\mathbf{w})\big)^2\Big] \\
&\quad + \mathop{\mathbb{E}}_{s\sim\mu}\Big[2\big(v_\pi(s) - v_\pi(s)\big)\big(v_\pi(s) - \hat{v}(s,\mathbf{w})\big)\Big] \\
&= \overline{\text{VE}}(\mathbf{w}) + \mathbb{E}\Big[\big(G_t - v_\pi(S_t)\big)^2\Big]
\end{aligned}
$$

□

Reinforcement Learning: Chapter 12
Answers to Exercises

Exercise 12.1 Just as the return can be written recursively in terms of the first reward and itself one-step later (3.9), so can the λ-return. Derive the analogous recursive relationship from (12.2) and (12.1).

Answer:

$$\begin{aligned}
G_t^\lambda &= (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+n} && (12.2)\\
&= (1-\lambda)\left[G_{t:t+1} + \lambda G_{t:t+2} + \lambda^2 G_{t:t+3} + \ldots\right]\\
&= (1-\lambda)\Bigg[1\Big(R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w}_t)\Big)\\
&\qquad + \lambda\Big(R_{t+1} + \gamma R_{t+2} + \gamma^2\hat{v}(S_{t+2},\mathbf{w}_{t+1})\Big) && (\text{using } (12.1))\\
&\qquad + \lambda^2\Big(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3\hat{v}(S_{t+3},\mathbf{w}_{t+1})\Big)\\
&\qquad\qquad \vdots \qquad\qquad \Bigg]\\
&= R_{t+1} + \gamma\lambda G_{t+1}^\lambda + \gamma(1-\lambda)\hat{v}(S_{t+1},\mathbf{w}_t)
\end{aligned}$$

□

Exercise 12.2 The parameter λ characterizes how fast the exponential weighting in Figure 12.2 falls off, and thus how far into the future the λ-return algorithm looks in determining its update. But a rate factor such as λ is sometimes an awkward way of characterizing the speed of the decay. For some purposes it is better to specify a time constant, or half-life. What is the equation relating λ and the half-life, τ_λ, the time by which the weighting sequence will have fallen to half of its initial value?

Answer: Basically we have to solve the equation $\lambda^n = \frac{1}{2}$ for n; that n is the desired value for τ_λ.

$$\begin{aligned}
\lambda^n &= \frac{1}{2}\\
(e^{\ln\lambda})^n &= \frac{1}{2}\\
e^{n\ln\lambda} &= \frac{1}{2}\\
n\ln\lambda &= \ln\frac{1}{2}\\
n &= \frac{\ln\frac{1}{2}}{\ln\lambda} = \tau_\lambda
\end{aligned}$$

□

Exercise 12.3 Some insight into how TD(λ) can closely approximate the off-line λ-return algorithm can be gained by seeing that the latter's error term (in brackets in (12.4)) can be written as the sum of TD errors (12.6) for a single fixed $\mathbf{w}$. Show this, following the pattern of (6.6), and using the recursive relationship for the λ-return you obtained in Exercise 12.1.

Answer:

$$\begin{aligned}
G_t^\lambda - \hat{v}(S_t,\mathbf{w}) &= R_{t+1} + \gamma\lambda G_{t+1}^\lambda + \gamma(1-\lambda)\hat{v}(S_{t+1},\mathbf{w}) - \hat{v}(S_t,\mathbf{w}) \\
&= R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w}) - \hat{v}(S_t,\mathbf{w}) + \gamma\lambda G_{t+1}^\lambda - \gamma\lambda\hat{v}(S_{t+1},\mathbf{w}) \\
&= \delta_t + \gamma\lambda\left(G_{t+1}^\lambda - \hat{v}(S_{t+1},\mathbf{w})\right) \\
&= \delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\left(G_{t+2}^\lambda - \hat{v}(S_{t+2},\mathbf{w})\right) \\
&= \delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \cdots + (\gamma\lambda)^{T-t-1}\delta_{T-1} + (\gamma\lambda)^{T-t}\left(G_T^\lambda - \hat{v}(S_T,\mathbf{w})\right) \\
&= \delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \cdots + (\gamma\lambda)^{T-t-1}\delta_{T-1} + (\gamma\lambda)^{T-t}(0-0) \\
&= \sum_{k=t}^{T-1}(\gamma\lambda)^{k-t}\delta_k
\end{aligned}$$

□

Exercise 12.4 Use your result from the preceding exercise to show that, if the weight updates over an episode were computed on each step but not actually used to change the weights ($\mathbf{w}$ remained fixed), then the sum of TD(λ)'s weight updates would be the same as the sum of the off-line λ-return algorithm's updates.

Answer: TBD. □

Exercise 12.5 Several times in this book (often in exercises) we have established that returns can be written as sums of TD errors if the value function is held constant. Why is (12.10) another instance of this? Prove (12.10).

Answer: TBD. □

Exercise 12.6 Modify the pseudocode for Sarsa(λ) to use dutch traces (12.11) without the other features of a true online algorithm. Assume linear function approximation and binary features.

Answer: TBD. □

Exercise 12.7 Generalize the three recursive equations above to their truncated versions, defining $G_{t:h}^{\lambda s}$ and $G_{t:h}^{\lambda a}$.

Answer: TBD. □

Exercise 12.8 Prove that (12.24) becomes exact if the value function does not change. To save writing, consider the case of $t = 0$, and use the notation $V_k \doteq \hat{v}(S_k,\mathbf{w})$.

Answer: For convenience of notation, take t to be 0. Then

$$\begin{aligned}
G_0^{\lambda s} - V_0 &= \rho_0\Big(R_1 + \gamma_1\big((1-\lambda_1)V_1 + \lambda_1 G_1^{\lambda s}\big)\Big) + (1-\rho_0)V_0 - V_0 \\
&= \rho_0\Big(R_1 + \gamma_1 V_1 - V_0 + \gamma_1\big(-\lambda_1 V_1 + \lambda_1 G_1^{\lambda s}\big)\Big) \\
&= \rho_0\Big(\delta_0^s + \gamma_1\lambda_1\big(G_1^{\lambda s} - V_1\big)\Big) \\
&= \rho_0\Big(\delta_0^s + \gamma_1\lambda_1\rho_1\Big(\delta_1^s + \gamma_2\lambda_2\big(G_2^{\lambda s} - V_2\big)\Big)\Big) \\
&= \rho_0\Big(\delta_0^s + \gamma_1\lambda_1\rho_1\delta_1^s + \gamma_1\lambda_1\rho_1\gamma_2\lambda_2\big(G_2^{\lambda s} - V_2\big)\Big) \\
&= \rho_0\Big(\delta_0^s + \gamma_1\lambda_1\rho_1\delta_1^s + \gamma_1\lambda_1\rho_1\gamma_2\lambda_2\rho_2\Big(\delta_2^s + \gamma_3\lambda_3\big(G_3^{\lambda s} - V_3\big)\Big)\Big) \\
&= \rho_0\Big(\delta_0^s + \gamma_1\lambda_1\rho_1\delta_1^s + \gamma_1\lambda_1\rho_1\gamma_2\lambda_2\rho_2\delta_2^s + \gamma_1\lambda_1\rho_1\gamma_2\lambda_2\rho_2\gamma_3\lambda_3\big(G_3^{\lambda s} - V_3\big)\Big) \\
&= \rho_0\Big(\delta_0^s + \gamma_1\lambda_1\rho_1\delta_1^s + \gamma_1\lambda_1\rho_1\gamma_2\lambda_2\rho_2\delta_2^s + \gamma_1\lambda_1\rho_1\gamma_2\lambda_2\rho_2\gamma_3\lambda_3\rho_3\delta_3^s + \cdots\Big) \\
&= \rho_0\sum_{k=0}^{\infty}\delta_k^s\prod_{i=1}^{k}\gamma_i\lambda_i\rho_i,
\end{aligned}$$

or, translating back to general t,

$$G_t^{\lambda s} = V_t + \rho_t\sum_{k=t}^{\infty}\delta_k^s\prod_{i=t+1}^{k}\gamma_i\lambda_i\rho_i. \qquad \text{(QED)}$$

□

Exercise 12.9 The truncated version of the general off-policy return is denoted $G_{t:h}^{\lambda s}$. Guess the correct equation, based on (12.24).

Answer: Just like (12.24) except that the upper limit on the sum is not ∞ but h. □

Exercise 12.10 Prove that (12.27) becomes exact if the value function does not change. To save writing, consider the case of $t = 0$, and use the notation $Q_k = \hat{q}(S_k, A_k, \mathbf{w})$. Hint: Start by writing out δ_0^a and $G_0^{\lambda a}$, then $G_0^{\lambda a} - Q_0$.

Answer:

$$\delta_0^a = R_1 + \gamma_1 Q_1 - Q_0$$

$$G_0^{\lambda a} = R_1 + \gamma_1\Big(Q_1 + \lambda_1\rho_1\Big[G_1^{\lambda a} - Q_1\Big]\Big)$$

$$\begin{aligned}
G_0^{\lambda a} - Q_0 &= \delta_0^a + \gamma_1\lambda_1\rho_1\big(G_1^{\lambda a} - Q_1\big) && \text{(combining the above two equations)}\\
&= \delta_0^a + \gamma_1\lambda_1\rho_1\Big(\delta_1^a + \gamma_2\lambda_2\rho_2\big(G_2^{\lambda a} - Q_2\big)\Big) && \text{(recursing)}\\
&= \delta_0^a + \gamma_1\lambda_1\rho_1\delta_1^a + \gamma_1\lambda_1\rho_1\gamma_2\lambda_2\rho_2\big(G_2^{\lambda a} - Q_2\big)\\
&= \delta_0^a + \gamma_1\lambda_1\rho_1\delta_1^a + \gamma_1\lambda_1\rho_1\gamma_2\lambda_2\rho_2\Big(\delta_2^a + \gamma_3\lambda_3\rho_3\big(G_3^{\lambda a} - Q_3\big)\Big)\\
&= \delta_0^a + \gamma_1\lambda_1\rho_1\delta_1^a + \gamma_1\lambda_1\rho_1\gamma_2\lambda_2\rho_2\delta_2^a + \gamma_1\lambda_1\rho_1\gamma_2\lambda_2\rho_2\gamma_3\lambda_3\rho_3\big(G_3^{\lambda a} - Q_3\big)\\
&= \delta_0^a + \gamma_1\lambda_1\rho_1\delta_1^a + \gamma_1\lambda_1\rho_1\gamma_2\lambda_2\rho_2\delta_2^a + \gamma_1\lambda_1\rho_1\gamma_2\lambda_2\rho_2\gamma_3\lambda_3\rho_3\delta_3^a + \cdots\\
&= \sum_{k=0}^{\infty}\delta_k^a\prod_{i=1}^{k}\gamma_i\lambda_i\rho_i
\end{aligned}$$

or, translating back to general t,

$$G_t^{\lambda a} = Q_t + \sum_{k=t}^{\infty}\delta_k^a\prod_{i=t+1}^{k}\gamma_i\lambda_i\rho_i. \qquad \text{(QED)}$$

□

Exercise 12.11 The truncated version of the general off-policy return is denoted $G_{t:h}^{\lambda a}$. Guess the correct equation for it, based on (12.27).

Answer: Just like (12.27) except that the upper limit on the sum is not ∞ but h. □

Exercise 12.12 Show in detail the steps outlined above for deriving (12.29) from (12.27). Start with the update (12.15), substitute $G_t^{\lambda a}$ from (12.26) for G_t^λ, then follow similar steps as led to (12.25).

Answer:

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha\left(G_t^{\lambda a} - \hat{q}(S_t, A_t, \mathbf{w}_t)\right)\nabla\hat{q}(S_t, A_t, \mathbf{w}_t)\\ &\approx \mathbf{w}_t + \alpha\left(\sum_{k=t}^{\infty}\delta_k^a\prod_{i=t+1}^{k}\gamma_i\lambda_i\rho_i\right)\nabla\hat{q}(S_t, A_t, \mathbf{w}_t)\end{aligned}$$

The increments can be summed over time steps to produce a form that looks like it can be turned into a backward view:

$$\begin{aligned}\sum_{t=1}^{\infty}(\mathbf{w}_{t+1} - \mathbf{w}_t) &\approx \sum_{t=1}^{\infty}\sum_{k=t}^{\infty}\alpha\delta_k^a\nabla\hat{q}(S_t, A_t, \mathbf{w}_t)\prod_{i=t+1}^{k}\gamma_i\lambda_i\rho_i\\ &= \sum_{k=1}^{\infty}\sum_{t=1}^{k}\alpha\nabla\hat{q}(S_t, A_t, \mathbf{w}_t)\delta_k^a\prod_{i=t+1}^{k}\gamma_i\lambda_i\rho_i\\ &\qquad\qquad\text{(using the summation rule: } \textstyle\sum_{t=x}^{y}\sum_{k=t}^{y} = \sum_{k=x}^{y}\sum_{t=x}^{k}\text{)}\\ &= \sum_{k=1}^{\infty}\alpha\delta_k^a\sum_{t=1}^{k}\nabla\hat{q}(S_t, A_t, \mathbf{w}_t)\prod_{i=t+1}^{k}\gamma_i\lambda_i\rho_i,\end{aligned}$$

which is of the standard backward-view form if we can incrementally compute the eligibility trace:

$$\begin{aligned}\mathbf{z}_k &= \sum_{t=1}^{k}\nabla\hat{q}(S_t, A_t, \mathbf{w}_t)\prod_{i=t+1}^{k}\gamma_i\lambda_i\rho_i\\ &= \sum_{t=1}^{k-1}\nabla\hat{q}(S_t, A_t, \mathbf{w}_t)\prod_{i=t+1}^{k}\gamma_i\lambda_i\rho_i \quad + \quad \nabla\hat{q}(S_k, A_k, \mathbf{w}_k)\\ &= \gamma_k\lambda_k\rho_k\underbrace{\sum_{t=1}^{k-1}\nabla\hat{q}(S_t, A_t, \mathbf{w}_t)\prod_{i=t+1}^{k-1}\gamma_i\lambda_i\rho_i}_{\mathbf{z}_{k-1}} \quad + \quad \nabla\hat{q}(S_k, A_k, \mathbf{w}_k)\\ &= \gamma_k\lambda_k\rho_k\mathbf{z}_{k-1} + \nabla\hat{q}(S_k, A_k, \mathbf{w}_k),\end{aligned}$$

which, changing the index from k to t, is the general accumulating trace update for action values:

$$\mathbf{z}_t \doteq \gamma_t\lambda_t\rho_t\mathbf{z}_{t-1} + \nabla\hat{q}(S_t, A_t, \mathbf{w}_t). \qquad \text{(QED)}$$

□

Exercise 12.13 What are the dutch-trace and replacing-trace versions of off-policy eligibility traces for state-value and action-value methods?

Answer: TBD. □

***Exercise 12.14** How might Double Expected Sarsa be extended to eligibility traces?

Answer: TBD. □

Reinforcement Learning: Chapter 13
Answers to Exercises

Exercise 13.1 Use your knowledge of the gridworld and its dynamics to determine an *exact* symbolic expression for the optimal probability of selecting the right action in Example 13.1.

Answer: Because all states appear the same, policies in this example are equivalent to probabilities p of taking the right action in all states. Our plan is to determine an equation for $J(\boldsymbol{\theta}) = v_\pi(\mathsf{S})$ as a function of p, take its derivative with respect to p, set the derivative to zero to find the optimum, and then solve for that value of p. We get an equation for $v_\pi(\mathsf{S})$ by solving the Bellman equation for v_π, for each of the three non-terminal states, which we denote as S (Start), M (Middle), and R (right):

$$\begin{aligned}
v_\pi(\mathsf{S}) &= p\,(-1 + v_\pi(\mathsf{M})) + (1-p)\,(-1 + v_\pi(\mathsf{S})) \\
v_\pi(\mathsf{M}) &= p\,(-1 + v_\pi(\mathsf{S})) + (1-p)\,(-1 + v_\pi(\mathsf{R})) \\
v_\pi(\mathsf{R}) &= p\,(-1 + 0) + (1-p)\,(-1 + v_\pi(\mathsf{M}))\,.
\end{aligned}$$

Solving this system of equations for $v_\pi(\mathsf{S})$ gives:

$$v_\pi(\mathsf{S}) = \frac{4-2p}{p^2-p}$$

Taking the derivative with respect to p yields:

$$\frac{dv_\pi(s_0)}{dp} = -2\frac{p^2-4p+2}{p^2(p-1)^2}$$

Setting the derivative equal to zero and solving gives:

$$\begin{aligned}
-2\frac{p^2-4p+2}{p^2(p-1)^2} &= 0 \\
p^2 - 4p + 2 &= 0 \\
p &= 2 \pm \sqrt{2}.
\end{aligned}$$

But because p is a probability it must be less than or equal to 1. Thus

$$\begin{aligned}
p &= 2 - \sqrt{2} \\
&\approx 0.58578.
\end{aligned}$$

□

***Exercise 13.2** Generalize the box on page 199, the policy gradient theorem (13.5), the proof of the policy gradient theorem (page 325), and the steps leading to the REINFORCE update equation (13.8), so that (13.8) ends up with a factor of γ^t and thus aligns with the general algorithm given in the pseudocode.

Answer: TBD. □

Exercise 13.3 In Section 13.1 we considered policy parameterizations using the soft-max in action preferences (13.2) with linear action preferences (13.3). For this parameterization, prove that the eligibility vector is

$$\nabla \ln \pi(a|s,\boldsymbol{\theta}) = \mathbf{x}(s,a) - \sum_b \pi(b|s,\boldsymbol{\theta})\mathbf{x}(s,b), \tag{13.9}$$

using the definitions and elementary calculus.

Answer: TBD. □

Exercise 13.4 Show that for the gaussian policy parameterization (13.19) the eligibility vector has the following two parts:

$$\nabla \ln \pi(a|s,\boldsymbol{\theta}_\mu) = \frac{\nabla \pi(a|s,\boldsymbol{\theta}_\mu)}{\pi(a|s,\boldsymbol{\theta})} = \frac{1}{\sigma(s,\boldsymbol{\theta})^2}\big(a - \mu(s,\boldsymbol{\theta})\big)\mathbf{x}_\mu(s), \text{ and}$$

$$\nabla \ln \pi(a|s,\boldsymbol{\theta}_\sigma) = \frac{\nabla \pi(a|s,\boldsymbol{\theta}_\sigma)}{\pi(a|s,\boldsymbol{\theta})} = \left(\frac{\big(a - \mu(s,\boldsymbol{\theta})\big)^2}{\sigma(s,\boldsymbol{\theta})^2} - 1\right)\mathbf{x}_\sigma(s).$$

Answer: TBD. □

In the following exercise, the Hint is slightly changed from the printed book.

Exercise 13.5 A *Bernoulli-logistic unit* is a stochastic neuron-like unit used in some ANNs (Section 9.6). Its input at time t is a feature vector $\mathbf{x}(S_t)$; its output, A_t, is a random variable having two values, 0 and 1, with $\Pr\{A_t = 1\} = P_t$ and $\Pr\{A_t = 0\} = 1 - P_t$ (the Bernoulli distribution). Let $h(s, 0, \boldsymbol{\theta})$ and $h(s, 1, \boldsymbol{\theta})$ be the preferences in state s for the unit's two actions given policy parameter $\boldsymbol{\theta}$. Assume that the difference between the action preferences is given by a weighted sum of the unit's input vector, that is, assume that $h(s, 1, \boldsymbol{\theta}) - h(s, 0, \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{x}(s)$, where $\boldsymbol{\theta}$ is the unit's weight vector.

(a) Show that if the exponential soft-max distribution (13.2) is used to convert action preferences to policies, then $P_t = \pi(1|S_t, \boldsymbol{\theta}_t) = 1/(1 + \exp(-\boldsymbol{\theta}_t^\top \mathbf{x}(S_t)))$ (the logistic function).

(b) What is the Monte-Carlo REINFORCE update of $\boldsymbol{\theta}_t$ to $\boldsymbol{\theta}_{t+1}$ upon receipt of return G_t?

(c) Express the eligibility $\nabla \ln \pi(a|s,\boldsymbol{\theta})$ for a Bernoulli-logistic unit, in terms of a, $\mathbf{x}(s)$, and $\pi(a|s,\boldsymbol{\theta})$ by calculating the gradient.

Hint for part (c): Define $P = \pi(1|s,\boldsymbol{\theta})$ and compute the derivative of the logarithm, for each action, using the chain rule on P. Combine the two results into one expression that depends on a and P, and then use the chain rule again, this time on $\boldsymbol{\theta}^\top \mathbf{x}(s)$, noting that the derivative of the logistic function $f(x) = 1/(1 + e^{-x})$ is $f(x)(1 - f(x))$.

Answer:

(a) Starting from the exponential soft-max distribution (13.2)

$$\begin{aligned} P_t = \pi(1|S_t,\boldsymbol{\theta}_t) &= \frac{e^{h(S_t,1,\boldsymbol{\theta}_t)}}{e^{h(S_t,1,\boldsymbol{\theta}_t)} + e^{h(S_t,0,\boldsymbol{\theta}_t)}} \\ &= \frac{1}{1 + \frac{e^{h(S_t,0,\boldsymbol{\theta}_t)}}{e^{h(S_t,1,\boldsymbol{\theta}_t)}}} \\ &= \frac{1}{1 + e^{(h(S_t,0,\boldsymbol{\theta}_t) - h(S_t,1,\boldsymbol{\theta}_t))}} = \frac{1}{1 + e^{-\boldsymbol{\theta}_t^\top \mathbf{x}(S_t)}} \end{aligned}$$

(b) From the box on page 328:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\gamma^t G_t \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}_t)$$

(c) Letting $P \doteq \pi(1|s, \boldsymbol{\theta}) = 1/(1 + e^{-\boldsymbol{\theta}^\top \mathbf{x}(s)})$ and taking all gradients with respect to $\boldsymbol{\theta}$, we have

$$\nabla \ln \pi(1|s, \boldsymbol{\theta}) = \nabla \ln P = \frac{d(\ln P)}{dP} \nabla P = \frac{1}{P} \nabla P, \text{ and}$$
$$\nabla \ln \pi(0|s, \boldsymbol{\theta}) = \nabla \ln(1 - P) = \frac{d(\ln(1-P))}{dP} \nabla P = \frac{-1}{1-P} \nabla P.$$

Combining these expressions as a function of a, where $a = 1$ or 0, yields:

$$\nabla \ln \pi(a|s, \boldsymbol{\theta}) = \frac{a - P}{P(1-P)} \nabla P. \qquad (*)$$

(Note that P is never equal to 0 or 1 because it is computed via the logistic function.) From the hint about the derivative of the logistic function $f(x) = 1/(1 + e^{-x})$, we have via the chain rule

$$\begin{aligned}
\nabla P = \nabla \pi(1|s, \boldsymbol{\theta}) &= \nabla(1/(1 + e^{(-\boldsymbol{\theta}^\top \mathbf{x}(s))}) = \nabla f(\boldsymbol{\theta}^\top \mathbf{x}(s)) \\
&= \frac{df}{d(\boldsymbol{\theta}^\top \mathbf{x}(s))} \nabla(\boldsymbol{\theta}^\top \mathbf{x}(s)) \qquad \text{(chain rule)} \\
&= f(\boldsymbol{\theta}^\top \mathbf{x}(s))(1 - f(\boldsymbol{\theta}^\top \mathbf{x}(s)) \nabla(\boldsymbol{\theta}^\top \mathbf{x}(s)) \\
&= P(1-P) \nabla(\boldsymbol{\theta}^\top \mathbf{x}(s)) \\
&= P(1-P)\mathbf{x}(s).
\end{aligned}$$

Substituting this into expression * yields

$$\nabla \ln \pi(a|s, \boldsymbol{\theta}) = \frac{a - P}{P(1-P)} P(1-P)\mathbf{x}(s) = (a - P)\mathbf{x}(s) = (a - \pi(1|s, \boldsymbol{\theta}))\mathbf{x}(s).$$

For the state and action occurring at time t, we have

$$\nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}) = \big(A_t - \pi(1|S_t, \boldsymbol{\theta})\big)\mathbf{x}(S_t).$$

□

Reinforcement Learning: Chapter 17
Answers to Exercises

Exercise 17.1 This section has presented options for the discounted case, but discounting is arguably inappropriate for control when using function approximation (Section 10.4). What is the natural Bellman equation for a hierarchical policy, analogous to (17.4), but for the average reward setting (Section 10.3)? What are the two parts of the option model, analogous to (17.2) and (17.3), for the average reward setting?

Answer: TBD. □