Reinforcement Learning: An Introduction - Exercises

Scott Jeen

April 23, 2021

Contents

1 Multi-armBandits

Exercise 2.1

Q

In e-greedy action selection, for the case of two actions and $e = 0.5$, what is the probability that the greedy action is selected?

A

- Greedy action selected with $p = 0.5$

Exercise 2.2

Q

Bandit example 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 e-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 A1 = 1, R1 = 1, A2 = 2, R2 = 1, A3 = 2, R3 = 2, A4 = 2, R4 = 2, A5 = 3, R5 = 0. On some of these time steps the e 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?

A

Timestep	Q_1	Q_2	Q_3	Q_4	Greedy action at timestep	Action selected
0	0	0	0	0	-	A_1
1	1	0	0	0	A_1	A_2
2	1	1	0	0	A_1/A_2	A_2
3	1	1.5	0	0	A_2	A_2
4	1	1.66	0	0	A_2	A_5
5	1	1.66	0	0	A_2	end

Table 1: Greedy actions at each timestep of a 4-armed bandit problem

- Action selection at timesteps 0, 1 and 4 are random as they are not reward maximising - see Table **??**.
- Action selection at timestep 2 could be random as A_1 is also reward maximising - see Table **??**.

Exercise 2.3

Q

In the comparison shown in Figure **??**, 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.

A

- In the limit of $t \to \infty$, both non-zero e-greedy policies will learn the optimal action abd value function $q*$. The policy with $e = 0.01$ will select the optimal action 10x more regularly than the policy with $e = 0.1$.

Exercise 2.4

Q

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?.

A

For a n-dependent α we have:

$$\begin{aligned}
Q_{n+1} &= Q_n + \alpha_n \left[R_n - Q_n\right] \\
&= \alpha_n R_n + (1 - \alpha_n) Q_n \\
&= \alpha_n R_n + (1 - \alpha_n) \left[Q_{n-1} + \alpha_{n-1} \left[R_{n-1} - Q_{n-1}\right]\right] \\
&= \alpha_n R_n + \alpha_{n-1} R_{n-1} - \alpha_n \alpha_{n-1} R_{n-1} + (1 - \alpha_n)(1 - \alpha_{n-1}) Q_{n-1} \\
&= \vdots \\
&= \sum_i^n \alpha_n R_n - R_{n-1} \prod_i^n \alpha_n + Q_1 \prod_i^n (1 - \alpha_n) && (1) \\
& && (2)
\end{aligned}$$

Exercise 2.5

Q

Design and conduct an experiment to demonstrate the difficulties that sample-average methods have for nonstationary problems. Use a modified version of the 10-armed testbed in which all the $q_*(a)$ start out equal and then take independent random walks (say by adding a normally distributed increment with mean 0 and standard deviation 0.01 to all the $q_*(a)$ on each step). Prepare plots like Figure 2.2 for an action-value method using sample averages, incrementally computed, and another action-value method using a constant step-size parameter, $\alpha = 0.1$. Use $\alpha = 0.1$ and longer runs, say of 10,000 steps.

A

This is a programming exercise, the relevent code can be found on my GitHub.

Exercise 2.6

Q

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?

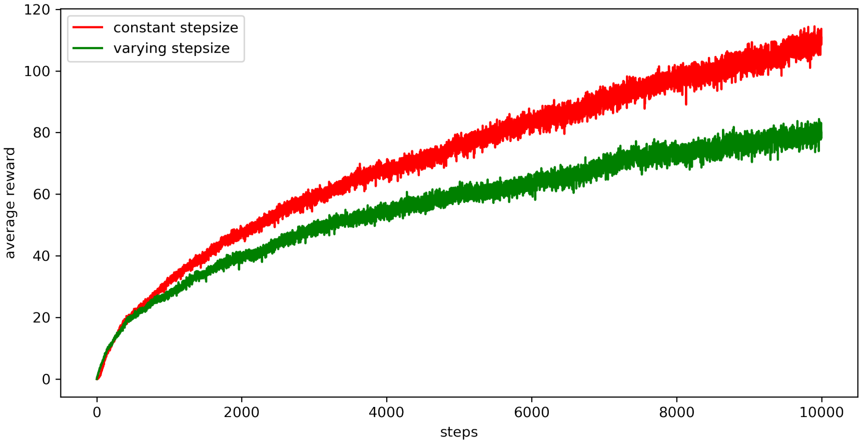


Figure 1: Stationary versus varying stepsize alpha in a 10-armed bandit problem. We see that the constant stepsize parameter (exponential decay) performs better than the varying stepsize parameter as we place more weight on the recently observed (moving) values.

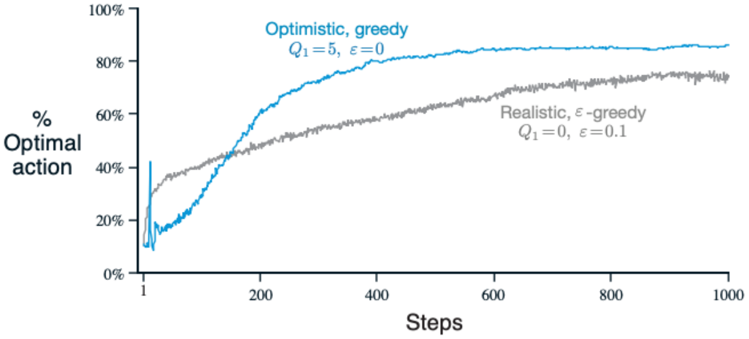


Figure 2.3: The effect of optimistic initial action-value estimates on the 10-armed testbed. Both methods used a constant step-size parameter, $\alpha = 0.1$.

Figure 2

A

The optimistic greedy policy with explore on every initial step as all value estimates are greater than their true value. It is possible, therefore, that it randomly selects the optimal action and then immediately forgets it in favour of yet-to-be-explored actions. This explains the spike at timestep ≈ 10.

Exercise 2.7

Q

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

$$\beta_n = \alpha/\bar{o_n} \tag{3}$$

to process the nth reward for a particular action, where α ¿ 0 is a conventional constant step size, and ¯on is a trace of one that starts at 0:

$$\bar{o_n} = o_{\bar{n-1}} + \alpha(1 - o_{\bar{n-1}}), forn > 0and\bar{o_0} = 0. \tag{4}$$

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

A

If we recall our answer for Exercise 2.4 for varying stepsize, we see that Q_1 is weighted by $w = \prod_{i=1}^{\infty}(1 - \alpha_i)$. When $i = 1$, $\beta_n = \alpha$, thus $w \to 0 \forall i$ and Q_1 no longer affects our estimate of Q_{n+1}.

Exercise 2.8

Q

UCB Spikes In Figure **??** 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.

A

After 10 timesteps the UCB algorithm has explored all 10 actions as, until they are selected, their upper confidence bound is infinite (as $N_t(a) = 0$) as so it guarenteed to be selected once in the first 10 actions. At this point the agent has one sample to assess the expected value of each arm and the same confidence/uncertainty in each action. With c ¡ 0 it is likely to pick the action with highest return from first sample, which will likely give it an similarly large reward, creating the spike. Now, the upper confidence bound for that action will decrease and the agent will select another, less valuable action, causing the decrease in performance at the next timestep.

Exercise 2.9

Q

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.

A

Soft-max distribution is defined as:

$$PrA_t = a \approx \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \approx \pi_t(a) \tag{5}$$

For two actions 1 and 2 this becomes:

$$
\begin{aligned}
A_t &= \frac{e^{H_t(1)}}{e^{H_t(1)+e^{H_t(2)}}} && (6)\\
&= \frac{e^{H_t(1)}}{e^{H_t(1)}}(1+\frac{e^{H_t(2)}}{e^{H_t(1)}} && (7)\\
&= \frac{1}{e^{H_t(1)}(1+e^{-x}} && (8)\\
& && (9)
\end{aligned}
$$

i.e. the sigmoid function with $x = H_t(1) - H_t(2)$.

Exercise 2.10

Q

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 10 and 20 with probability 0.5 (case A), and 90 and 80 with probability 0.5 (case B). If you are not able to tell which case you face at any step, what is the best expected reward 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 expected reward you can achieve in this task, and how should you behave to achieve it?

A

Part 1): If we do not know whether we are in task A or B we could decide pick the same action each time to maximise expected reward. Selecting either action 1 or action 2 every time would provide an expected reward of 50. Picking actions randomly would also provide an expected reward of 50 in this example. Part 2): If we know we are in task A or B we can learn the optimal action for each (A(a) = 2 and B(a) = 1). Doing so would provide us a higher expected reward than the non-contextual case of 55.

2 Finite Markov Decision Processes

Exercise 3.1

Q

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.

A

1. *Golf: Putting*
    - State: Coordinates of ball; coordinates of hole; x,y,z contour plot of green; grass length; grass type; wind speed; wind direction; ball type; putter type.
    - Actions: (X, Y) direction of aim; length of stroke
    - Rewards: -1 for each unit of distance from hole
2. *Optimizing investment portfolio*
    - State: Investment in each company; cash balance; % return over last minute/hour/day/week/month/year
    - Actions: For each investment: buy (discretized by some cash interval), sell (discretized by some cash interval), stick
    - Rewards: +1 for each £ return per day
3. *Shoe-tying robot*
    - State: Coordinates of laces; coordinates of robot arms/joints
    - Actions: Grip pressure on laces; adjust position of arms to coordinates x,y,z
    - Rewards: -1 for unit of shoe displacement from foot once tied.

Exercise 3.2

Q

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

A

The MDP framework fails when the state cannot be fully observed. For example, one may try to control the temperature of a house using a state space represented by one thermometer in one room. Our agent would control the temperature observed by that thermometer well, but would be blind to temperatures that aren't measured by said thermometer elsewhere in the house.

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

Exercise 3.3

Q

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?

A

There's a trade-off between state-action space complexity, computational expense and accuracy. If we draw the boundary at the brain we would create a state-action space contingent on the number of neurons in the brain and their interplay with physical actions like turning the steering wheel; too large to be stored or computed efficiently. Equally, if we draw the boundary at the *journey* level, then we miss the detail required to act on second-by-second state changes on the road that could lead to a crash. The fundamental limiting factor in this selection is whether the goal can be achieved safely at your chosen layer of abstraction, indeed this feels like one of the tasks of engineering more widely.

Exercise 3.4

Q

Give a table analogous to that in Example 3.3, but for $p(s0, r|s, a)$. It should have columns for s, a, s', r, and $p(s0, r|s, a)$, and a row for every 4-tuple for which $p(s0, r|s, a) > 0$.

A

As there is no probability distribution over the rewards (i.e. for each state-action pair, the agent receives some reward with $p(r) = 1$)), $p(s', r|s, a) = p(s'|s, a)$.

Exercise 3.5

Q

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).

A

Equation 3.3 from section 3.1 is:

$$\sum_{s' \in S} \sum_{r \in R} p(s', r|s, a) = 1, \forall s \in S, a \in A(s) \tag{10}$$

for episodic tasks this becomes:

$$\sum_{s' \in S^+} \sum_{r \in R} p(s', r|s, a) = 1, \forall s \in S, a \in A(s) \tag{11}$$

Exercise 3.6

Q

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?

A

In the discounted, episodic version of the pole-balancing task, the return after each episode is:

$$G_t = -\gamma^{T-t} \tag{12}$$

where T is the timestep at which the episode ends i.e. the task is failed. In the continuining case the cumulative return is:

$$G_t = -\sum_{K \in \mathcal{K}} \gamma^{K-1} \tag{13}$$

where $\mathcal{K}$ is the set of times where the task is failed. Note this reward will increase in the long run, irrespective of improved performance. Designing this as a continuous task does not therefore make sense here.

Exercise 3.7

Q

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?

A

We have designed the reward such that it receives +1 only once it has exited the maze, and are therefore not incentivising it to learn how to exit the maze faster. To do so, we would need to provide a negative reward proportional to time in the maze e.g. -1 per timestep.

Exercise 3.8

Q

Suppose $\gamma = 0.5$ and the following sequence of rewards is received R1 = -1, R2 = 2, R3 = 6, R4 = 3, and R5 = 2, with T = 5. What are $G_0, G_1, \dots, G_5$? Hint: Work backwards.

A

We know: $G_t = R_{t+1} + \gamma G_{t+1}$ Therefore, for the terminal state: $G_5 = 0$ then: $G_4 = 2 + (0.5 \times 0) = 2 G_3 = 3 + (0.5 \times 2) = 4 G_2 = 6 + (0.5 \times 4) = 8 G_1 = 2 + (0.5 \times 8) = 6 G_0 = -1 + (0.5 \times 6) = 2$ Note our expected cumulative reward G_t depends greatly on our instant reward G_{t+1} because it is not discounted.

Exercise 3.9

Q

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?

A

We know that if the reward is an infinite series of 1s, G_t is: $G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$
So for an infinite series of 7s this becomes: $G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{7}{0.1}$
Therefore:

$$\begin{aligned} G_0 &= 2 + \frac{7}{0.1} && (14) \\ &= 72 && (15) \\ & && (16) \end{aligned}$$

and

$$\begin{aligned} G_1 &= 7 + \frac{7}{0.1} && (17) \\ &= 77 && (18) \\ & && (19) \end{aligned}$$

Exercise 3.10

Q

Prove the second equality in (3.10).

A

$G_t = \sum_{k=0}^{\infty} \gamma^k$
Expanding out the geometric series we get:

$$\begin{aligned} s &= a + a\gamma + a\gamma^2 + \dots + a\gamma^n && (20) \\ s\gamma &= a\gamma + a\gamma^2 + a\gamma^3 + \dots + a\gamma^{n+1} && (21) \\ & && (22) \end{aligned}$$

Subtract the first series from the second series and we get:

$$s - s\gamma = a - a\gamma^{n+1} \tag{23}$$

$$s(1-\gamma) = a(1-\gamma^{n+1}) \tag{24}$$

$$s = \frac{a(1-\gamma^{n+1})}{1-\gamma} \tag{25}$$

$$\tag{26}$$

If $|\gamma| < 0$, in the limit as n $\rightarrow \infty$ we get:

$$s = \frac{a}{1-\gamma} \tag{27}$$

and in our special case $a = 0$ so:

$$s = G_t = \frac{1}{1-\gamma} \tag{28}$$

Exercise 3.11

Q

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

A

$\mathbb{E}_\pi[R_{t+1}|s_t] = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r]$

Exercise 3.12

Q

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

A

The state value function v_π is equal to the expected cumulative return from that state given a distribution of actions. The state-action value function q_π is the value of being in a state and taking a deterministic action. Therefore the state value function is the weighted sum of the state action value function, with the weights equal to the probabilities of selecting each action: $v_\pi = \sum_a \pi(a|s) q_\pi(s, a)$

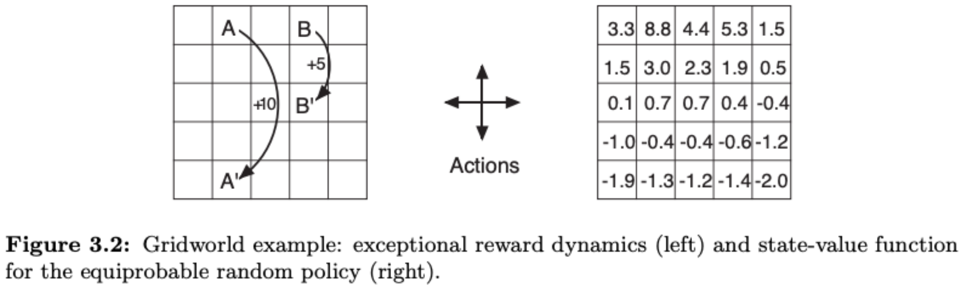
Exercise 3.13

Q

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

A

Given an action a, the state-action value function is the probability distributions over the possible next states and rewards from that action, times the one-step reward and the discounted state value function at the next timestep: $q_\pi = \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a)[r + \gamma v_\pi(s_{t+1})]$



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Figure 3.2: Gridworld example: exceptional reward dynamics (left) and state-value function for the equiprobable random policy (right).

Figure 3: Gridworld example for ex. 3.14

Exercise 3.14

Q

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 neighbouring states, valued at +2.3, +0.4, 0.4, and +0.7. (These numbers are accurate only to one decimal place.) .

A

Bellman equation for v_π is:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \tag{29}$$

for the centre square in Figure 3 we get the following:

$$\begin{align} v_\pi(s) &= 0.25\,[0.9 \times 2.3] + 0.25\,[0.9 \times 0.7] + 0.25\,[0.9 \times 0.4] + 0.25\,[0.9 \times -0.4] \tag{30}\\ &= 0.68 \approx 0.7 \tag{31}\\ \tag{32} \end{align}$$

Exercise 3.15

Q

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?

A

Part 1): The sign of the reward is of no consequence, it is indeed the interval between each reward that drives behaviour. Part 2): Equation 3.8 is as follows:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{33}$$

when we add a constant c to all rewards this becomes: $G_t = \sum_{k=0}^{\infty} \gamma^k [R_{t+k+1} + c] G_t = \frac{R_{t+k+1}}{1-\gamma} + \frac{c}{1-\gamma}$
Expected cumulative reward from every state receives a constant additive term. v_c in terms of c and γ is:

$$v_c = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k c\right] \tag{34}$$

$$= \frac{c}{1-\gamma} \tag{35}$$

$$\tag{36}$$

Exercise 3.16

Q

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.

A

In the episodic task adding a constant to all rewards does affect the agent. Since the cumulative reward now depends on the length of the episode ($G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$), timesteps that incur positive rewards act to lengthen the episode and vice versa. In the maze running example, we may have chosen to give the agent -1 reward at each timestep to ensure it completes the task quickly. If we add $c = 2$ to every reward such that the reward at each timestep is now positive, the agent is now incentivised to not find the exit, and continue collecting intermediate rewards indefinitely.

Exercise 3.17

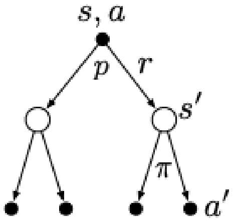


Figure 4: Backup diagram for q_π

Q

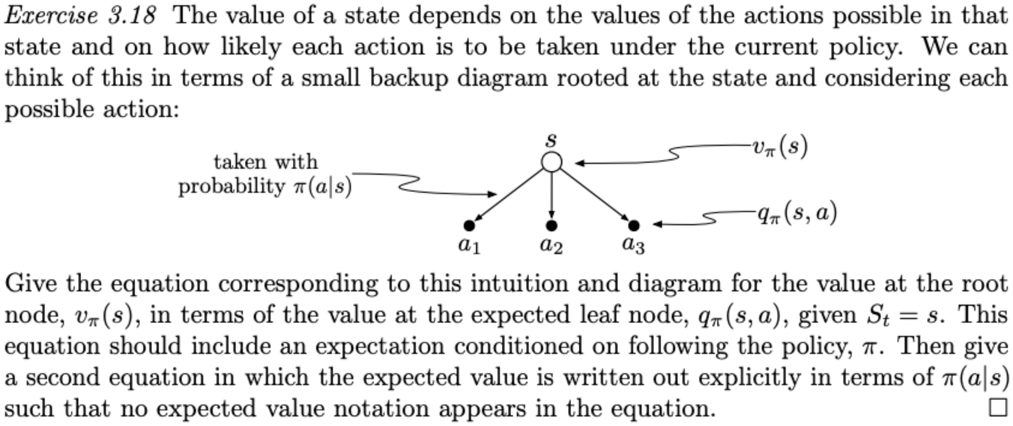
What is the Bellman equation for action values, that is, for q_π? It must give the action value $q_\pi(s,a)$ in terms of the action values, $q_\pi(s0, a0)$, of possible successors to the state–action pair (s,a). Hint: The backup diagram to the right corresponds to this equation. Show the sequence of equations analogous to (3.14), but for action values.

A

$$\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi[G_t|S_t=s, A_t=a] \qquad (37)\\
&= \mathbb{E}_\pi[R_{t+1}+\gamma G_{t+1}|S_t=s, A_t=a] \qquad (38)\\
&= \sum_{s',r} p(s',r|s,a)[r+\gamma\mathbb{E}_\pi[G_{t+1}|s',a']] \qquad (39)\\
&= \sum_{s',r} p(s',r|s,a)[r+\gamma q_\pi(s',a')] \qquad (40)\\
& \qquad (41)
\end{aligned}$$

Exercise 3.18

Q



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. □

A

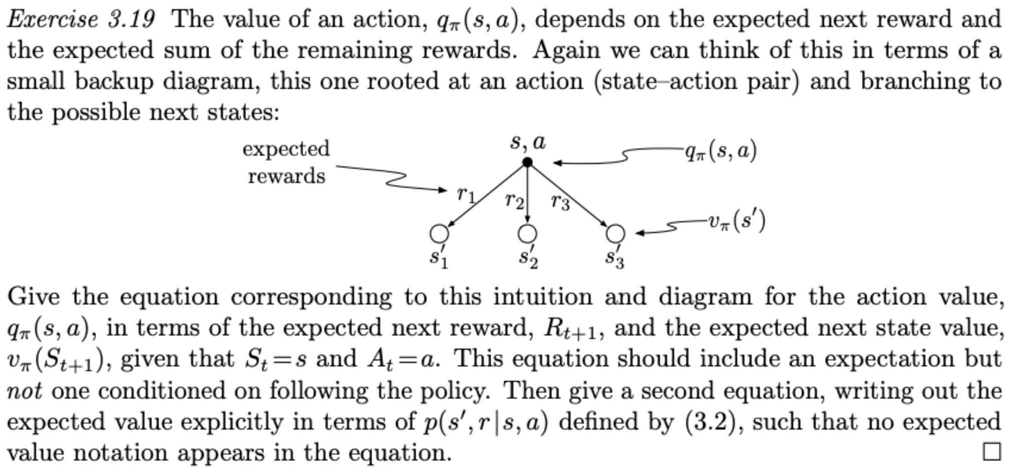
$$v_\pi(s) = \mathbb{E}_\pi\left[q_\pi(s,a)\right] v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a) \tag{42}$$

Exercise 3.19

Q

A

$$\begin{aligned}
q_\pi(s,a) &= \mathbb{E}\left[R_{t+1}+v_\pi(s')|S_t=s)\right] \qquad (43)\\
&= \sum_{s',r} p(s',r|s,a)[r+v_\pi(s')] \qquad (44)\\
& \qquad (45)
\end{aligned}$$



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:

expected
rewards
s, a
$q_\pi(s,a)$
r_1
r_2
r_3
$v_\pi(s')$
s'_1
s'_2
s'_3

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 on following the policy. Then give a second equation, writing out the expected value explicitly in terms of $p(s', r \mid s, a)$ defined by (3.2), such that no expected value notation appears in the equation. □

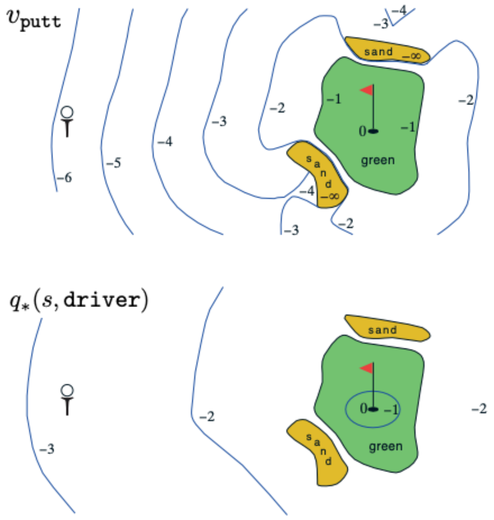


Figure 5: Value functions for golf.

Exercise 3.20

Q

Q

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

A

Qualitatively, the optimal state-value function for golf (outlined by Figure **??**) would be derived from a policy that selected the driver for the first two shots (off-the green), then selected the putter for the final shot (on the green).

Exercise 3.21

Q

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

A

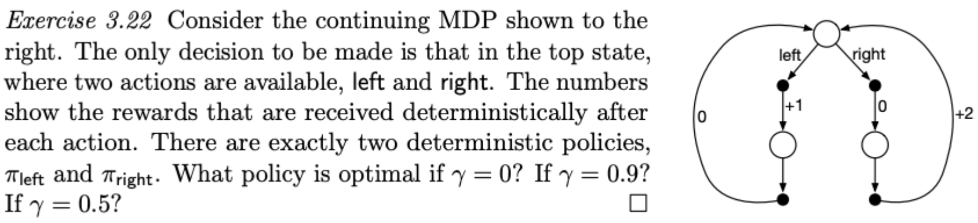
The optimal action-value function is given by the following:

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \operatorname*{argmax}_a q_*(s,a)] \tag{46}$$

Since we have only one action($putter$), the argmax collapses to $v_*(s')$, and $q_*(s,a) = v_*(s)$ - illustrated in Figure **??**.

Exercise 3.22

Q



Exercise 3.22 Consider the continuing MDP shown 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 $\gamma = 0.5$? □

A

Let's first evaluate the simplest case of $\gamma = 0$. Recall that:

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \tag{47}$$

For π_{left} we get:

$$\begin{align}
v_{\pi_{left}} &= 1 + \mathbb{E}_{\pi_{left}}[0 \times \gamma G_{t+1}] \tag{48}\\
v_{\pi_{left}} &= 1 \tag{49}\\
\tag{50}
\end{align}$$

And for π_{right} we get:

$$\begin{align}
v_{\pi_{left}} &= 0 + \mathbb{E}_{\pi_{left}}[0 \times \gamma G_{t+1}] \tag{51}\\
v_{\pi_{left}} &= 0 \tag{52}\\
\tag{53}
\end{align}$$

So the optimal policy for $\gamma = 0$ is $v_{\pi_{left}}$. If instead $\gamma = 0.9$, we get for π_{left}:

$$
\begin{aligned}
v_{\pi_{left}} &= 1 + \mathbb{E}_{\pi_{left}}\left[0.9 \times G_{t+1}\right] && (54)\\
v_{\pi_{left}} &= 1 + \mathbb{E}_{\pi_{left}}\left[0.9 \times \left[r_{t+1} + \gamma G_{t+2}\right]\right], && (55)\\
& && (56)
\end{aligned}
$$

where G_{t+2} is our expected reward back at our current state. We can negate it and just look at the first loop, as max reward over first loop will create max reward in the limit. Therefore:

$$
\begin{aligned}
v_{\pi_{left}} &= 1 + 0.9 \times 0 = 1, && (57)\\
v_{\pi_{right}} &= 0 + 0.9 \times 2 = 1.8 && (58)\\
& && (59)
\end{aligned}
$$

So the optimal policy for $\gamma = 0$ is $v_{\pi_{right}}$. If, finally, $\gamma = 0.5$, we get:

$$
\begin{aligned}
v_{\pi_{left}} &= 1 + 0.5 \times 0 = 1, && (60)\\
v_{\pi_{right}} &= 0 + 0.5 \times 2 = 1 && (61)\\
& && (62)
\end{aligned}
$$

So both polices are optimal.

Exercise 3.23

Q

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

A

$q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \operatorname{argmax}_{a \in \mathcal{A}} q_*(s',a')]$ where $\mathcal{A} = \{search, wait, recharge\}$

Exercise 3.24

Q

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.

A

We can observe from the grid that $\gamma = 22/24.4 = 0.9016$
Equation 3.8 was:

$$
G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \gamma^4 R_{t+5} + \gamma^5 R_{t+6} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (63)
$$

We can represent the optimal value function v_* as:

$$
\begin{aligned}
v_*(A) &= r + \gamma(v_*(A+1)) && (64)\\
&= 10 + 0.9016(16) && (65)\\
&= 24.426 \text{to 3 d.p.} && (66)\\
& && (67)
\end{aligned}
$$

Exercise 3.25

Q

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

A

$v_*(s) = \max_{a \in \mathcal{A}} q_{\pi_*}(s, a)$
i.e. returning to the diagram in exercise 3.18, v_* is defined as selecting the action (in that case of a possible 3) that produces the highest state-action value function

Exercise 3.26

Q

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

A

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

Exercise 3.27

Q

Give an equation for π_* in terms of q_*

A

The optimal policy is the one that acts greedily w.r.t the optimal state-action value function i.e. it picks the action that has the highest $q(s, a)$ in the following state. $\pi_*(a|s) = \max_{a \in \mathcal{A}(s)} q_*(s', a)$

Exercise 3.28

Q

Give an equation for π_* in terms of v_* and the four-argument p.
The optimal policy is the one that acts greedily w.r.t the optimal state value function conditioned on the system dynamics

A

$\pi_*(a|s) = \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s', r|s, a)[rv_*(s')]$

Exercise 3.29

Q

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

A

tbc

3 Dynamic Programming

Exercise 4.1

Q

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

A

The Bellman equation for $q(s, a)$ is:

$$q_\pi(s,a) = \sum_{s',r} p(s', r|s, a) \left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a') \right] \tag{68}$$

For the state-action pairs posed in the question, the Bellman equation becomes:

$$\begin{align} q_\pi(11, down) &= -1 \tag{69} \\ q_\pi(7, down) &= -1 + -14 \tag{70} \\ &= -15 \tag{71} \\ \tag{72} \end{align}$$

as $\gamma = 1$ because MDP is undiscounted.

Exercise 4.2

Q

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?

A

$$\begin{align} v_\pi(15) &= 0.25\left[(-1 + -20) + (-1 + -22) + (-1 + -14) + (-1 + v_\pi(15))\right] \tag{73} \\ &= 0.25\left[(-60 + v_\pi(15))\right] \tag{74} \\ &= -15 + 0.25\left[v_\pi(15)\right] \tag{75} \\ \tag{76} \end{align}$$

and we know $v_\pi(15) == v_\pi(13) == -20$ as the state transitions and value functions at next state are identical. Therefore we get:

$$\begin{align} v_\pi(15) &= -15 + 0.25\left[-20\right] \tag{77} \\ &= -20 \tag{78} \\ \tag{79} \end{align}$$

If the dynamics are changed such one can transition from state 13 into 15, the characteristic of the MDP are unchanged. Moving into state 15, which has the same value as state 13 and the same subsequent dynamics, is identical to returning back to state 13 - as was the case previously. The value function is therefore unchanged and $v_\pi(15) = -20$

Exercise 4.3

Q

What are the equations analogous to (4.3), (4.4), and (4.5), but for action-value functions instead of state-value functions?

A

$$q_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1} | S_t = s, A_t = a)\right] v_\pi(s) \tag{80}$$

$$q_\pi(s, a) = \sum_{s', r} p(s', r|s, a)\left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')\right] \tag{81}$$

$$q_{k+1}(s, a) = \sum_{s', r} p(s', r|s, a)\left[r + \gamma \sum_{a'} \pi(a'|s') q_k(s', a')\right] \tag{82}$$

$$\tag{83}$$

Exercise 4.4

Q

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 okay for pedagogy, but not for actual use. Modify the pseudocode so that convergence is guaranteed.

A

When taking the argmax over a in the policy improvement step of the algorithm, it's possible that we continue to flip backward and forward between two actions that are both optimal forever. At the moment, a tie is broken by randomly selecting between the value maximising actions. We could instead always select the first action to result from the argmax, this way we would ensure that the same optimal action is picked during iteration, switching the *policy-stable* boolean to true, and ensuring convergence.

Exercise 4.5

Q

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.

A

1. **Initialization**: $q(s, a)$ and $\pi(s)$ initialised arbitrarily as before
2. **Policy Evaluation**: Loop for each state-action pair (s, a), $s \in \mathcal{S}, a \in \mathcal{A}$:

$$q \leftarrow Q(s, a)$$

$$Q(s, a) \leftarrow \sum_{s', r} p(s', r|s, a)\left[r + \gamma \sum_{a'} \pi(a'|s') Q(s', a')\right] \tag{84}$$

$$\delta \leftarrow max(\delta, |q - Q(s, a)|)$$

3. **Policy Improvement**:

$$\begin{aligned}
&\text{policy-stable} \leftarrow True \\
&\text{Loop for each state-action pair (s,a), s} \in \mathcal{S}, a \in \mathcal{A}: \\
&old - action \leftarrow \pi(s) \\
\pi(s) \leftarrow \underset{a}{\text{argmax}} \sum_{s',r} p(s', r|s, a) &\left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a') \right] \qquad (85) \\
&\text{if old-action} \neq \pi(s), \text{then policy-stable} \leftarrow False \\
&\text{If policy-stable, then stop and return Q} \approx q_* and \pi \approx \pi_*\text{; else return to 2.}
\end{aligned}$$

Exercise 4.6

Q

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 $\epsilon/|\mathcal{A}(s)|$. 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_* on page 80.

A

During the policy improvement step (3), instead of the argmax creating a deterministic action in a state, we would update the policy such that each action $a \in \mathcal{A}$ would receive probability p(a) = $\epsilon/|\mathcal{A}(s)|$, then the max action would receive probability $1-\epsilon+\epsilon/|\mathcal{A}(s)|$. The output policy is therefore stochastic, not deterministic. During the policy evaluation step (2), instead of looping over the states only, we would loop over all states and actions, weighting the value of each state-action by the probability of the action being selecting according to our ϵ-soft policy. In step one, the policy would need to be initialised with an arbitrary distribution over the action space in each state.

Exercise 4.7

Q

Write a program for policy iteration and re-solve Jack's car rental problem with the following changes. One of Jack's employees at the first location rides a bus home each night and lives near the second location. She is happy to shuttle one car to the second location for free. Each additional car still costs \$2, as do all cars moved in the other direction. In addition, Jack has limited parking space at each location. If more than 10 cars are kept overnight at a location (after any moving of cars), then an additional cost of \$4 must be incurred to use a second parking lot (independent of how many cars are kept there). These sorts of non-linearities and arbitrary dynamics often occur in real problems and cannot easily be handled by optimization methods other than dynamic programming. To check your program, first replicate the results given for the original problem.

A

This is a programming exercise, the relevent code can be found on my GitHub.

Exercise 4.8

Q

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?

A

When the coin is bias against us it makes sense to minimise the number of flips we make as, in the limit, we cannot win. Consequently, we can win with probability 0.4 is we stake our full capital at 50. All other bets besides 50 are designed to get us back to 50 if we lose, or up to 50 if we win, from where we take our 40% chance. For example, when our capital is 55, we stake 5, knowing that we are likely to fall back to 50 where we will go for the win.

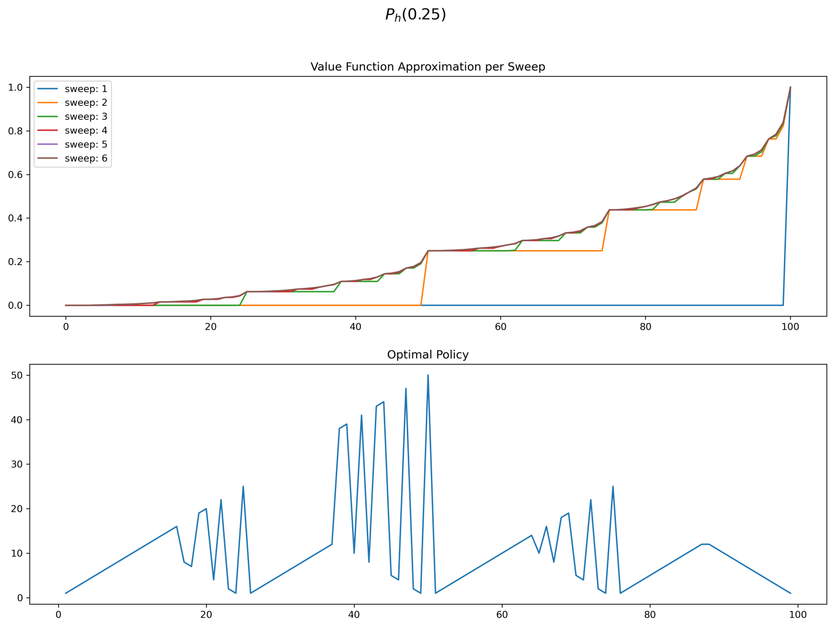
Exercise 4.9

Q

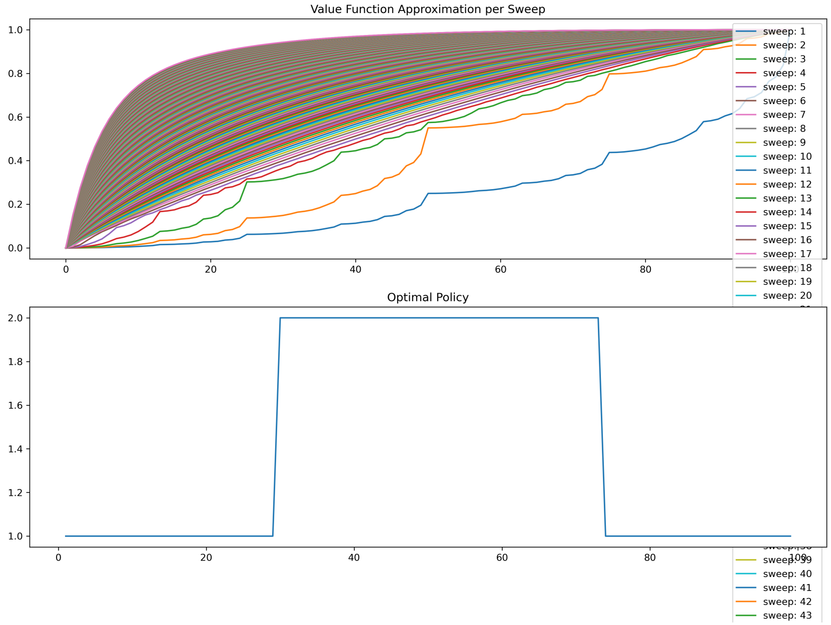
Implement value iteration for the gambler's problem and solve it for p_h = 0.25 and p_h = 0.55. In programming, you may find it convenient to introduce two dummy states corresponding to termination with capital of 0 and 100, giving them values of 0 and 1 respectively. Show your results graphically, as in Figure 4.3. Are your results stable as $\theta \rightarrow 0$?

A

This is a programming exercise, the relevent code can be found on my GitHub.



$P_h(0.55)$



Exercise 4.10

Q

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

A

$$v_{k+1}(s) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \operatorname*{argmax}_{a'} q_k(s',a')\right] \tag{86}$$

4 Monte Carlo Methods

Exercise 5.1

Q

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?

A

1. The value function jumps for the rows in the rear because the player sticks at 20 and 21, where it is unlikely the dealer beat him given her policy to twist for all hands lower than 17.
2. The value drops when the dealer holds an ace because it can be used to equal either 11 or 1, a stronger position than any of the other hands the dealer could hold.
3. When an ace is usable, the value function is higher because the player can change the value of his ace from 11 to 1 if she is about to go bust.

Exercise 5.2

Q

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?

A

The state is blackjack is monotonically increasing and memoryless (sampled with replacement), thus you can never revisit an old state in an episode once it has been first visited. Using every visit MC in this case would have no effect on the value function.

Exercise 5.3

Q

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

A

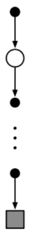


Figure 6: Monte carlo backup diagram for estimation of q_π from one episode

Exercise 5.4

Q

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.

A

Update formula from section 2.4 is:

$$Q_n = Q_n + \frac{1}{n}\left[R_n - Q_n\right] \tag{87}$$

As we reverse through the episode, we initialise value $N_{s,a}$ for each observed s and a to count the number of visits to the state. Instead of appending G to returns, we use our now initialised N, and G to update the value of Q. Doing so is more efficient as we don't need to hold a list of all returns which does not scale, we just update using the update rule outlined above.

Exercise 5.5

Q

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?

A

For first-visit estimator of the state is the return collected at the end of the episode having visited the first step, assuming we initialise $G = 0$: $G = 10$
The every-visit estimator is an average of each of the 10 returns received from the state:

$$G = \frac{1+2+3+4+5+6+7+8+9+10}{10} = 5.5 \tag{88}$$

Exercise 5.6

Q

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?

A

Equation 5.6 is as follows:

$$V(s) = \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}} \tag{89}$$

We only require that $\mathcal{T}$ tracks state action-pairs rather than just states. Equation 5.6 therefore becomes:

$$Q(s,a) = \frac{\sum_{t \in \mathcal{T}(s,a)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s,a)} \rho_{t:T(t)-1}} \tag{90}$$

Exercise 5.7

Q

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?

A

In the initial episodes, we are unlikely to see episodes from the behaviour policy that match our target policy (i.e. hit for all card sums ¡ 20 and stick thereafter), therefore our estimate for $V(s)$ will remain 0, which happens to be close to our ground truth $V_\pi(s)$. As we see some trajectories from b that match our target policy, variance in our output will be high initially, leading to a higher error, and will drop gradually as we observe further trajectories until it approaches the true value asymptotically after 10,000 episodes.

Exercise 5.8

Q

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?

A

The variance of the estimator would remain infinite, as the expected return is still 1 for every-visit to the state. The only difference between first-visit and every-visit MC in this case is that the number of terms increases α to the number of visits to the state and so would continue to run to infinity.

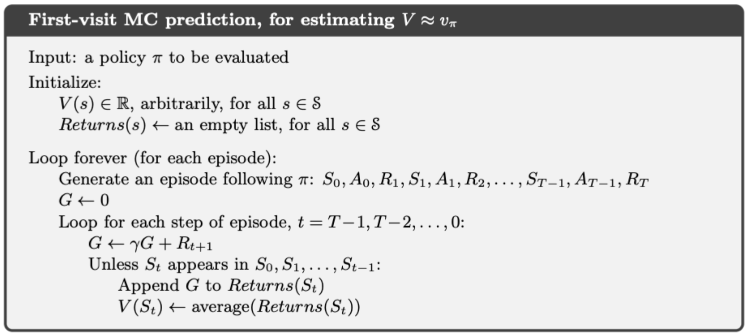
Exercise 5.9

Q

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.

A

Modifying the above to include sample averages we change the last two lines. Instead of appending G to $Returns(S_t)$ and averaging, we update $V(S_t)$ directly using: $V(S_t) = V(S_t) + \frac{1}{n}\left[G - V(S_t)\right]$
to do this we need to initialise a new variable n that counts the number of cross-episode visits to state S_t.



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

```
Input: a policy $\pi$ to be evaluated
Initialize:
    $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
    Generate an episode following $\pi$: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    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 $S_0, S_1, \ldots, S_{t-1}$:
            Append $G$ to $Returns(S_t)$
            $V(S_t) \leftarrow$ average($Returns(S_t)$)
```

Exercise 5.10

Q

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

A

We have $C_o = 0$ and $C_n = \sum_{k=1}^{n} W_k$. Therefore:

$$V_{n+1} = \frac{\sum_{k=1}^{n} W_k G_k}{C_n} \tag{91}$$

$$V_{n+1} C_n = \sum_{k=1}^{n} W_k G_k \tag{92}$$

$$V_{n+1} C_n = W_n G_n \sum_{k=1}^{n-1} W_k G_k \tag{93}$$

$$V_{n+1} C_n = W_n G_n + V_n \sum_{k=1}^{n-1} W_k \tag{94}$$

$$V_{n+1} C_n = W_n G_n + V_n C_{n-1} \tag{95}$$

$$V_{n+1} C_n = W_n G_n + V_n \left(C_n - W_n \right) \tag{96}$$

$$\vdots \tag{97}$$

$$V_{n+1} = V_n + \frac{W_n}{C_n} \left[G_n - V_n \right] \tag{98}$$

$$\tag{99}$$

Exercise 5.11

Q

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

A

Because our policy $\pi(S_t)$ is a deterministic, greedy one, we are only observing trajectories where $\pi(A_t|S_t) = 1$, hence the numerator in the equation $= 1$.

Exercise 5.12

This is a programming exercise, the relevent code can be found on my GitHub.

Exercise 5.13

Q

Show the steps to derive (5.14) from (5.12).

A

5.12 is:

$$\rho_{t:T(t)-1} R_{t+1} = R_{t+1} \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)} \tag{100}$$

$$\tag{101}$$

Exercise 5.14

Q

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.

A

...

5 Temporal-Difference Learning

Exercise 6.1

Q

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.

A

Equation 6.5 becomes: $\delta_t = R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)$ and equation 6.2 becomes: $V_{t+1}(S_t) = V_t(S_t) + \alpha \left[R_{t+1} + \gamma V_t(S_{t+1} - V_t(S_t)\right]$. We then proceed as follows:

$$\begin{aligned}
G_t - V_t(S_t) &= R_{t+1} + \gamma G_{t+1} - V_t(S_t) + \gamma V_t(S_{t+1}) - \gamma V_t(S_{t+1}) && (102)\\
&= \delta_t + \gamma(G_{t+1} - V_t(S_{t+1})) && (103)\\
&= \delta_t + \gamma(G_{t+1} - V_{t+1}(S_{t+1})) + \gamma\theta_{t+1}, \text{with } \theta_t = \alpha\left[R_{t+1} + \gamma V_t(S_{t+1} - V_t(S_t))\right] && (104)\\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2(G_{t+2} - V_{t+2}(S_{t+2})) + \gamma\theta_{t+1} + \gamma^2\theta_{t+2} && (105)\\
&= \sum_{k=t}^{T-1}\left[\gamma^{k-t}\delta_k + \gamma^{k-t+1}\theta_{k+1}\right] && (106)
\end{aligned}$$

■

Exercise 6.2

Q

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?

A

The hint above gives intuition to the answer. Let's assume we are trying to learn the optimal policy for playing a round of golf at my local course, and we assume we have already learned the value function for the course previously using monte carlo and td-learning. If 1 of the 18 holes is changed, monte carlo methods would require us to play the whole course to update our understanding of the states relating to the new hole once. With TD-learning, all we have to do is make predictions about our new hole, then we can revert to previously learned value function for the remaining holes. i.e. we bootstrap off of previous knowledge more effectively with TD-learning than monte carlo. Our convergence to the true value function using TD-learning should hence be quicker. ■

Exercise 6.3

Q

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 $V(A)$. 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?

A

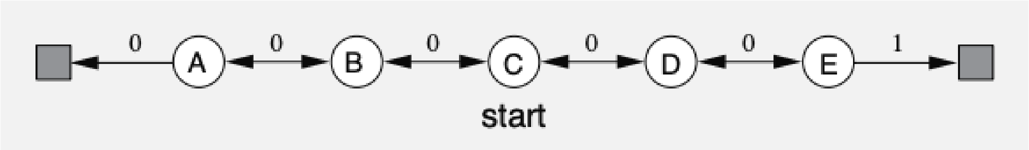


Figure 7: Random walk example

The result of the first episode suggests the episode terminated at the left block for no reward. The value of state A was updated as follows:

$$
\begin{aligned}
V(A) &= V(A) + \alpha \left[R_{t+1} + V(S_T) - V(A)\right] && (107)\\
&= 0.5 + 0.1 \left[0 + 0 - 0.5\right] && (108)\\
&= 0.45 && (109)
\end{aligned}
$$

■

Exercise 6.4

Q

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?

A

It appears from the plot the long-term accuracy of TD methods is inversely proportional to the chosen α, which makes sense given larger alphas would cause the value function to oscillate around the true value function. It is not clear from the plot whether different values of α for the MC method would have increased performance as there is no concrete difference in the algorithm performance based on α. It appears the main drawback of the monte carlo method is that it makes significantly fewer value function updates than TD methods. For 100 episodes, the MC method can make no more than 100 updates. For the TD method, the expected episode length is 6, and so it makes 600 value function updates in 100 episodes. No value of α can overcome the reduced sample rate. ■

Exercise 6.5

Q

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?

A

As discussed above, I believe this will always occur for high α as the weight given to the TD error will exaggerate small errors. The will cause the estimated value function to bounce back and forth across the true value without converging. In turn, this will cause the estimate for $V(c)$ (initialised in this example at its true value) to drift away from the correct estimate, increasing RMS error. ■

Exercise 6.6

Q

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?

A

1. **Dynamic Programming**: we know the policy π (take either action with $p(a) = 0.5$), and we know the transition function $p(s', r|s, a) = 1 \; \forall s, a$, thus the value of each state can be computed exactly.
2. **First-visit monte carlo policy prediction**: set-up the state space and run simulations with our policy π. After each episode, back-prop the returns to each state and average.

I suspect the authors used dynamic programming as it requires solving 6 trivial simultaneous equations, whereas the MC method may require thousands of episodes to converge on the true value. ■

Exercise 6.7

Q

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

A

Our TD update is:

$$V(s) = V(s) + \alpha \left[R_{t+1} + \gamma V(s') - V(s)\right] \tag{110}$$

and our importance sampling ratio is:

$$\rho_{t:t} = \frac{\pi(A|S)}{b(A|S)} \tag{111}$$

We assume an episode of experience is produced by our behaviour policy b, and the update becomes:

$$V_\pi(s) = V_\pi(s) + \alpha \left[\rho_{t:t} R_{t+1} + \rho_{t:t}\gamma V(s') - V(s)\right], \tag{112}$$

i.e. each update is weighted by the likelihood of the action being taken by the target policy compared to the behaviour policy, divided by sum of all previous importance samples. ■

Exercise 6.8

Q

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.

A

Recall, equation 6.6 was the monte carlo error written in terms of the TD error:

$$G_t - V(S_t) = \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k \tag{113}$$

We amend 6.6 as follows:

$$\begin{align}
G_t - Q(S_t, A_t) &= R_{t+1} + \gamma G_{t+1} - Q(S_t, A_t) + \gamma Q(S_{t+1}, A_{t+1}) - \gamma Q(S_{t+1}, A_{t+1}) \tag{114}\\
&= \delta_t + \gamma \left(G_{t+1} - Q(S_{t+1}, A_{t+1})\right) \tag{115}\\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2 \left(G_{t+2} - Q(S_{t+2}, A_{t+2}\right) \tag{116}\\
&\vdots \tag{117}\\
&= \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k \tag{118}
\end{align}$$

■

Exercise 6.9

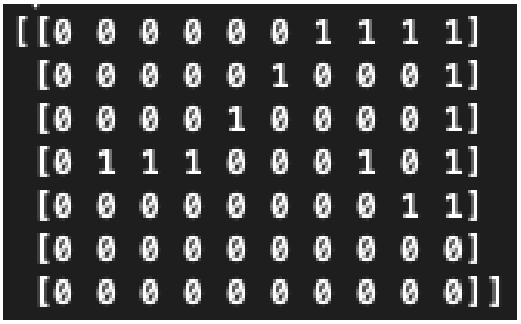
Q

Re-solve the windy gridworld assuming eight possible actions, including the diagonal moves, rather than four. How much better can you do with the extra actions? Can you do even better by including a ninth action that causes no movement at all other than that caused by the wind?

A

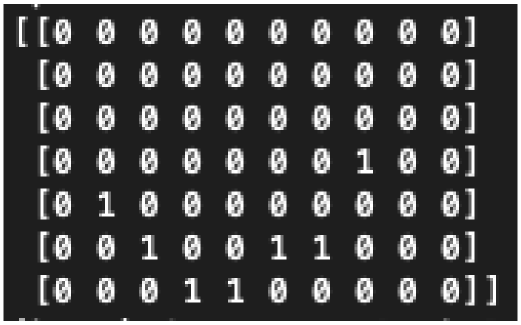
This is a programming exercise, the relevent code can be found on my GitHub.

With the extra actions, the optimal policy is now 7 moves as opposed to 15 moves with 4 actions. Adding the 9th action does not improve the optimal policy, the terminal square is 7 moves away from the start square and so adding an action that keeps the agent stationary cannot improve the policy. See Figures 8 and 9 plot the optimal policy for 4 and 8 actions respectively. Each agent learned from 10,000 episodes of self-play. ■



```
[[0 0 0 0 0 0 1 1 1 1]
 [0 0 0 0 0 1 0 0 0 1]
 [0 0 0 0 1 0 0 0 0 1]
 [0 1 1 1 0 0 0 1 0 1]
 [0 0 0 0 0 0 0 0 1 1]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]]
```

Figure 8: Optimal policy with 4 available actions



```
[[0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0]
 [0 1 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 1 1 0 0 0]
 [0 0 0 1 1 0 0 0 0 0]]
```

Figure 9: Optimal policy with 8 available actions

Exercise 6.10

Q

Re-solve the windy gridworld task with King's moves, assuming that the effect of the wind, if there is any, is stochastic, sometimes varying by 1 from the mean values given for each column. That is, a third of the time you move exactly according to these values, as in the previous exercise, but also a third of the time you move one cell above that, and another third of the time you move one cell below that. For example, if you are one cell to the right of the goal and you move left, then one-third of the time you move one cell above the goal, one-third of the time you move two cells above the goal, and one-third of the time you move to the goal.

A

Implementation is identical to above with amended transition function. Exercise not yet completed. ■

Exercise 6.11

Q

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

A

Q-learning is off-policy because the action-selection at S_{t+1} (used for the Q-update) is deterministic i.e. it chooses the greedy action with probability 1. This is in contrast with the behaviour policy used to collect the data which is ϵ-greedy. The policy being updated is therefore different from that being used to collect data, and the algorithm is off-policy. ■

Exercise 6.12

Q

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?

A

If action-selection is greedy, the algorithms become identical, but the action selections and weight updates may differ depending on the arbitrary initialisation of Q and S. For example, if each state-action pair is assigned a random value $Q(S, A) \in (0, 1]$, the greedy action selection in each case will differ. Because action selection differs, updates differ, and because neither algorithm explore, there is no guarantee they will converge on the same solution. ■

Exercise 6.13

Q

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

A

Instead of taking the maximum action over Q, we continue to follow our ϵ-greedy behaviour policy. The expected sarsa update is:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \right] \tag{119}$$

Therefore our updates become:

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q_2(S_{t+1}, a) - Q_1(S_t, A_t) \right] \tag{120}$$

$$Q_2(S_t, A_t) \leftarrow Q_2(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q_1(S_{t+1}, a) - Q_2(S_t, A_t) \right] \tag{121}$$

with our policy π being ϵ-greedy. ■

Exercise 6.14

Q

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?

A

In Jack's car rental example we each location can hold cars in the range (0,20). Up to 5 cars can be moved from one location to the other overnight to accommodate expected sales, taking one of these actions will move us deterministically to a new state. If there was uncertainty in how we transition between states, it makes sense to learn a conventional value function that probabilistically accounts for this uncertainty, but given the transition is certain, we can learn the afterstates (the states from which we will sell the cars in the morning) and proceed.

We speed convergence by reducing the number of states to be evaluated. For example, if our state in the evening is five cars at each location: $(5, 5)$ and our action is to move one car to the second location such that our morning state becomes $(4, 6)$, we can evaluate this state as the same as the state-action pair of $(4, 6)$ and choosing to move no cars. ■

6 *n*-step Bootstrapping

Exercise 7.1

Q

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.

A

The n-step error in equation 7.2 is $G_{t:t+n} - V_{t+n-1}(S_t)$ and our generalised TD error is $\delta_{t:t+n} = R_{t+n+1} + \gamma V_{t+n}(S_{t+1}) - V_{t+n}(S_t)$. As in 6.6, we can rewrite this as:

$$\begin{aligned}
G_{t:t+n} - V_{t+n-1}(S_t) &= R_{t+n+1} + \gamma G_{t+n+1} - V_{t+n}(S_t) && (122)\\
&= R_{t+n+1} + \gamma G_{t+n+1} - V_{t+n}(S_t) + \gamma V_{t+n}(S_{t+1}) - \gamma V_{t+n}(S_{t+1}) && (123)\\
&= \delta_{t:t+n} + \gamma(G_{t+n+1} + V_{t+n}(S_{t+1})) && (124)\\
&= \delta_{t:t+n} + \gamma\left[R_{t+n+2} + \gamma G_{t+n+2} - V_{t+n+1}(S_{t+2})\right] && (125)\\
&= \delta_{t:t+n} + \gamma\delta_{t+1:t+n+1} + \gamma^2(G_{t+n+2} + V_{t+n+1}(S_{t+1})) && (126)\\
&= \delta_{t:t+n} + \gamma\delta_{t+1:t+n+1} + \gamma^2\delta_{t+2:t+n+2} + \cdots + \gamma^{T-t-1}(G_{t+n+T-t-1} - V_{t+n+T-t-2}(S_T) + \gamma^{T-t}(&& (127)\\
&= \delta_{t:t+n} + \gamma\delta_{t+1:t+n+1} + \gamma^2\delta_{t+2:t+n+2} + \cdots + \gamma^{T-t-1}(G_{T+n-1} - V_{T+n-2}(S_T) + \gamma^{T-t}(0) && (128)\\
&= \sum_{k=t}^{T-1} \gamma^{k-t}\delta_{t+k:n+k} && (129)\\
& && (130)
\end{aligned}$$

■

Exercise 7.2

Q

With an n-step method, the value estimates do change from step to step, so an algorithm that used the sum of TD errors (see previous exercise) in place of the error in (7.2) would actually be a slightly different algorithm. Would it be a better algorithm or a worse one? Devise and program a small experiment to answer this question empirically.

A

This is a programming exercise, the relevent code can be found on my GitHub.

■

Exercise 7.3

Q

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?

A

- If the smaller, 5-state random walk had been used the optimal value for n would likely have been smaller. As explained in the example, setting n to 3 and observing an episode starting at C and transitioning $C \rightarrow D \rightarrow E \rightarrow Terminal$ would update C toward the reward of 1, which would represent an incorrect estimation of the true value of C. It seems that the likely optimal value in this case would be $n = 2$, and so we could make the general assumption that for smaller state-spaces, smaller values of n are more appropriate. If $n \geq 2$ then, for longer episodes, updates would no longer be bootstrapping on other state value, but instead be making updates based on their own values.
- I suspect that the change in the left value to -1 from 0 reduced the optimal value of n in this example. Setting the left value to -1 effectively locates states on the left side of the walk closer to a reward, meaning updates need not back-propagate as far to make good updates to the value of these states.

■

Exercise 7.4

Q

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

$$G_{t:t+n} = Q_{t-1}(S_t, A_t) + \sum_{k=t}^{min(t+n,T)-1} \gamma^{k-t} \left[R_{k+1} + \gamma Q_k(S_{k+1}, A_{k+1}) - Q_{k-1}(S_k, A_k)\right] \tag{131}$$

A

We have:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) \tag{132}$$

and:

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha \left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)\right] \tag{133}$$

Denote:

$$G_{t:t+n} \doteq \sum_{i=1}^{n} \gamma^{i-1} R_{t+i} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) \tag{134}$$

for $n \geq 1$ and $0 \leq t < T - n$ and with $G_{t:t+n} = G_t$ if $t + n > T$

If we set $\tau = \min(t+n, T) - 1$, we observe that:

$$\sum_{k=t}^{min(t+n,T)-1} \gamma^{k-t} \left[R_{k+1} + \gamma Q_k(S_{k+1}, A_{k+1}) - Q_{k-1}(S_k, A_k)\right] = \sum_{k=t}^{\tau} \gamma^{k-t} \left[R_{t+1} + \gamma Q_k(S_{k+1}, A_{k+1} - Q_{k-1}(S_k, A_k)\right] \tag{135}$$

$$= G_{t:t+n} - Q_{t-1}(S_t, A_t) \tag{136}$$

$$\tag{137}$$

■

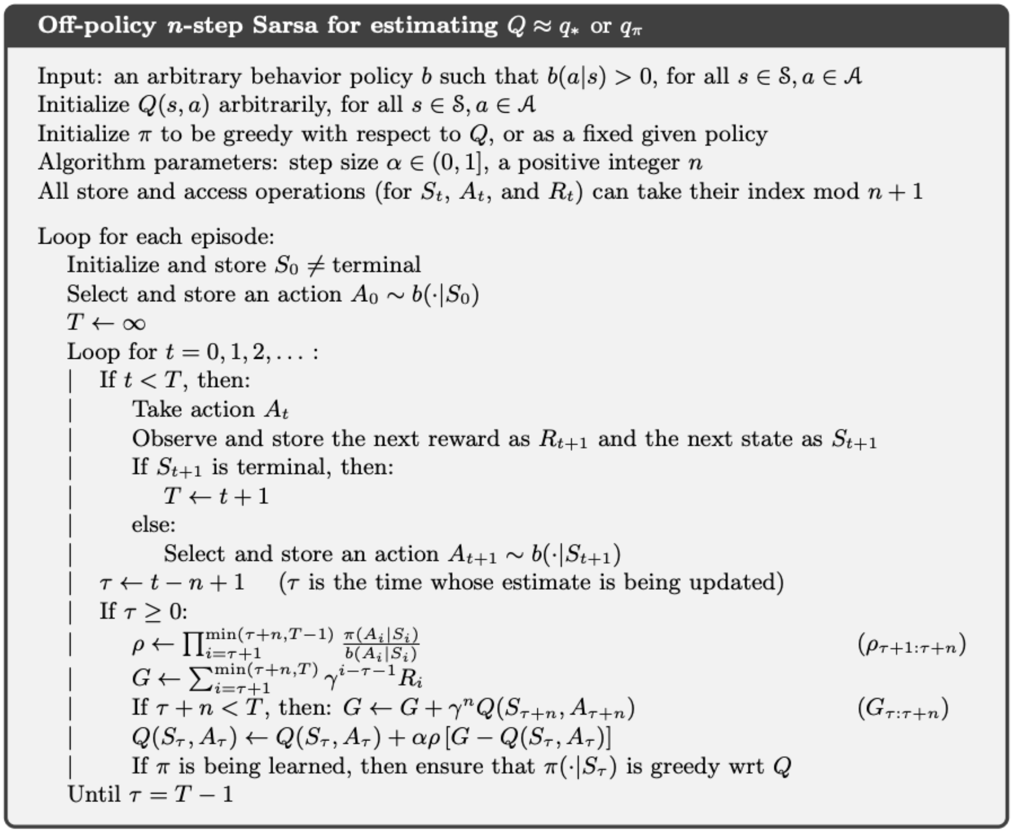
Exercise 7.5

Q

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

A

The algorithm follows much the same form as that described on page 149:



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

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

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

Figure 10: Off-policy n-step sarsa pseudocode

Of course, this time we initialize $V(s)$ arbitrarily instead of $Q(s, a)$. In the final if statement, G instead becomes:

$$G \leftarrow \rho_\tau \left(R_{\tau+1} + \sum_{i=\tau+1}^{min(\tau+n,T)} \gamma^{i-\tau-1} R_i \right) + (1-\rho_\tau) V_{\tau+n-1}(S_t) \tag{138}$$

and the state value update becomes:

$$V(S_\tau) \leftarrow V(S_\tau) + \alpha \left[G - V(S_\tau) \right] \tag{139}$$

■

Exercise 7.6

Q

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

A

In equation 7.13, we know that the expected value of $\rho_t = 1$, therefore:

$$\begin{align}
\mathbb{E}\left[(1-\rho_t)V_{h-1}(S_t)\right] &= \mathbb{E}_b\left[(1-\rho_t)V_{h-1}(S_t)\right] \tag{140} \\
&= \mathbb{E}_b\left[(1-\rho_t)V_{h-1}(S_t)\right] \tag{141} \\
&= \mathbb{E}_b\left[(1-1)V_{h-1}(S_t)\right] \tag{142} \\
&= 0 \tag{143} \\
& \tag{144}
\end{align}$$

In equation 7.14, the control variate is slightly different, we have:

$$\begin{align}
\mathbb{E}_b\left[\bar{V}_h - 1(S_{t+1} - \rho_{t+1} Q_h - 1(S_{t+1}, A_{t+1}))\right] &= \sum_a \pi(a|S_{t+1}) Q_{h-1}(S_{t+1}, a) - \sum_a b(a|S_{t+1}) \rho_{t+1} Q_{h-1}(S_{t+1}, a) \tag{145} \\
&= \sum_a \pi(a|S_{t+1}) Q_{h-1}(S_{t+1}, a) - \sum_a b(a|S_{t+1}) \frac{\pi(a|S_{t+1})}{b(a|S_{t+1})} Q_{h-1}(S_{t+1}, a) \tag{146} \\
&= 0 \tag{147} \\
& \tag{148}
\end{align}$$

■

Exercise 7.7

Q

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.

A

TBC

■

Exercise 7.8

Q

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.

A

TBC

■

Exercise 7.9

Q

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).

A

TBC

■

Exercise 7.10

Q

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

A

TBC

■

Exercise 7.11

Q

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:

A

If the action values are unchanging then we get:

$$G_{t:t+n} = R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1})Q(S_{t+1}, a) + \gamma\pi(A_{t+1}|S_{t+1})G_{t+1:t+n} \tag{149}$$

$$\tag{150}$$

We have been given:

$$\delta_t \doteq R_{t+1} + \gamma\bar{V}(S_{t+1}) - Q(S_t, A_t) \tag{151}$$

and:

$$\bar{V}_h \doteq \sum_a \pi(a|S_h)Q(S_h, a) \tag{152}$$

Then we can say:

$$\begin{aligned}
G_{t:t+n} - Q(S_t, A_t) &= R_{t+1} + \gamma\bar{V}(S_{t+1}) - \gamma\pi(A_{t+1}|S_{t+1})Q(S_{t+1}, A_{t+1}) + \gamma\pi(A_{t+1}|S_{t+1})G_{t+1:t+n} - Q(S_t, A_t) && (153)\\
&= \delta_t - \gamma\pi(A_{t+1}|S_{t+1})Q(S_{t+1}, A_{t+1}) + \gamma\pi(A_{t+1}|S_{t+1})G_{t+1:t+n} && (154)\\
&= \delta_t + \gamma\pi(A_{t+1}|S_{t+1})\left[G_{t+1:t+n} - Q(S_{t+1}, A_{t+1})\right] && (155)\\
&= \vdots && (156)\\
G_{t:t+n} &= Q(S_t, A_t) + \sum_{i=1}^{min(t+n,T)-1} \delta_i \prod_{j=t+1}^{i} \gamma\pi(A_j, S_j) && (157)\\
& && (158)
\end{aligned}$$

where the product operator has the behaviour $\prod_a^b \dot{[]} = 1$ for $a > b$. Shoutout to Bryn Hayder for this answer.

■

7 Planning and Learning with Tabular Methods

Exercise 8.1

Q

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.

A

If we used a multi-step bootstrapping method (e.g. n-step Sarsa), we could back-propogate the reward from the final timestep along the trajectory followed for some large value of n. This of course would only update the action-values of the n action-values prior to receiving the reward and leave the remain 1700 - n action values unchanged. When running a second episode using this approach, the agent would still act naively until it reached its first state-action pair with some value, which it could then bootstrap from to update previous states. Without a model of the environment (like Dyna) it cannot plan in the same way, and cannot reap the efficiency gains that Dyna enjoys.

■

Exercise 8.2

Q

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?

A

I'm not sure it is performing *better*, rather that the Dyna-Q+ algorithm is increasing the reward associated with every state at every timestep when Dyna-Q is not, so cumulative reward must be higher in absolute terms than Dyna-Q. In this sense a direct comparison on the basis of cumulative reward is perhaps unfair. Due to the increased exploration of the Dyna-Q+ algorithm, it is however likely that it found the optimal policy more quickly than Dyna-Q and so was able to exploit it quicker for higher cumulative reward

■

Exercise 8.3

Q

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?

A

Dyna-Q+ is forced to explore continually having already obtained the optimal policy, meaning that when Dyna-Q is exploiting its learned optimal policy, Dyna-Q+ is exploring sub-optimal trajectories and receiving less reward.

■

Exercise 8.4

Q

The exploration bonus described above actually changes the estimated values of states and actions. Is this necessary? Suppose the bonus $k\sqrt{\tau}$ was used not in updates, but solely in action selection. That is, suppose the action selected was always that for which $Q(S_t, a) + k\sqrt{\tau(S_t, a)}$ was maximal. Carry out a gridworld experiment that tests and illustrates the strengths and weaknesses of this alternate approach.

A

This is a programming exercise, the relevent code can be found on my GitHub.
Summary of the models:

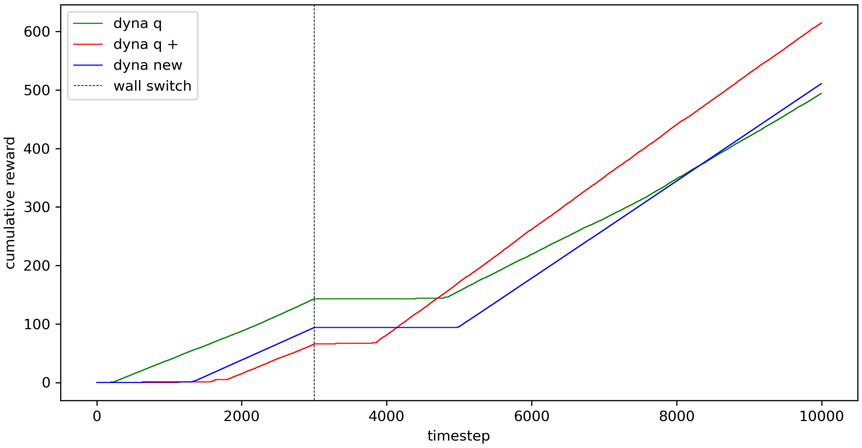


Figure 11: Cumulative reward for three model-based agents on the blocked maze task after 10000 timesteps

Dyna Q uses a conventional e-greedy policy, and plans with a model that uses only previously received states and rewards.

Dyna Q+ uses a conventional e-greedy policy, and plans with a model that gives additional value to the action proportionate to the time passed since they were last selected. Doing so adds arbitrary value to the value function.

Dyna New uses a novel policy that adds value to each action before selection, in the same way as Dyna Q+, then selects the value maximising action. This combines exploration and exploitation unlike the above two that keep these regimes separate.

Figure 11 illustrates the different learning rates of the three agents. Dyna Q learns fastest initially as it can use its model to learn the optimal policy quickly, but struggles to adapt to the maze block changes after timestep 10000 as its model is inflexible to change. Dyna Q+ has a higher degree of exploration than Dyna Q (because it explores both in the real environment and the simulated environment while planning) and so does not find the optimal policy as quickly as Dyna Q. It does, however, adapt to the wall switch speedily, and exploits its new knowledge to find the optimal policy. Finally, Dyna New, finds the optimal policy before the wall switch, then struggles to capture it post-switch. This is because Dyna New cannot perform exploration in planning, it only explores when taking actions in the real environment. This means that it will take many timesteps before a string of actions taking it to the other side of the grid build enough value for it to be explored.

Dyna Q Plus performs best in the long run by combining exploration in planning, and exploitation in the real environment. It does however seem these results are subject to initialisation and hyper-parameters. In some runs of the experiment Dyna New never found the optimal policy before the wall switch, its performance seems to be closely linked to the initialisation of τ

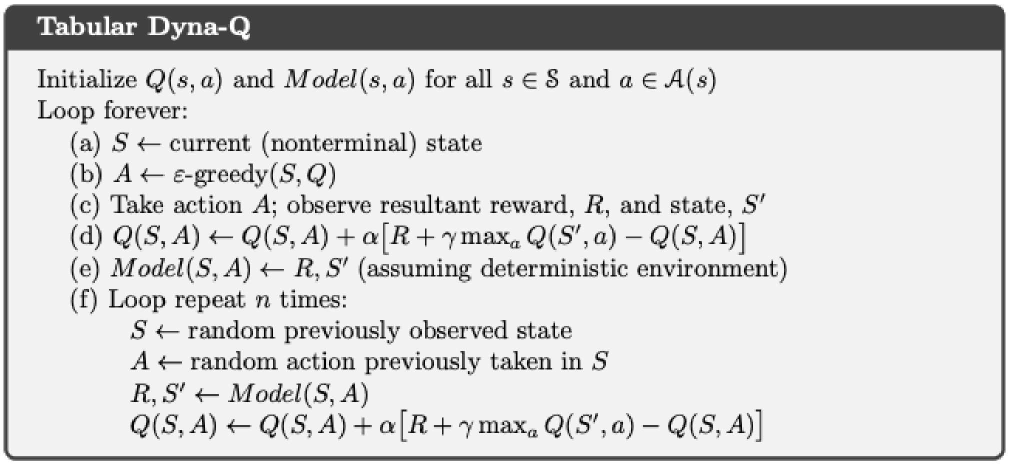
■

Exercise 8.5

Q

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?

A



```
Tabular Dyna-Q
Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
    (a) $S \leftarrow$ current (nonterminal) state
    (b) $A \leftarrow \varepsilon$-greedy$(S, Q)$
    (c) Take action $A$; observe resultant reward, $R$, and state, $S'$
    (d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    (e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
    (f) Loop repeat $n$ times:
            $S \leftarrow$ random previously observed state
            $A \leftarrow$ random action previously taken in $S$
            $R, S' \leftarrow Model(S,A)$
            $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
```

Figure 12: Tabular Dyna-Q algorithm

- In part (e) of the algorithm, instead of updating R and S' directly, we would store samples of R and S' from which we can compute distributions and thus expectations.
- This would perform poorly because it would be bias toward earlier observations made in the unchanged environment.
- We could likely rectify this by weighting more recent observations, or discounting past observations in the distribution. Equally we could quantify the agent's confidence in its model e.g. if it hasn't selected a state-action pair in a long time, its confidence in its model should be low and vice versa. This could manifest as a relationship with τ as discussed with Dyna-Q+.

■

Exercise 8.6

Q

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.

A

A skewed distribution would strengthen the case for sample-based updates. We are more likely to sample from weighted parts of the distribution and so our estimate of the expected value will approach the true value quicker than would be the case with a uniform distribution i.e. our initial samples will provide a good estimate of the true value. The expectation will require the same computational effort regardless of the shape of the distribution.

■

Exercise 8.7

Q

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?

A

In the uniform distribution case with $b = 1$ the start-state is updated every $|\mathcal{T}|$ updates, at which point it's value will be moved toward it's new value. Whereas in the on-policy case, the start state is visited with probability 0.1 (given this is the probability of termination) and so it is updated far more regularly. The time taken to wait for the updates in the uniform case (when other states not close to the start state are being updated) is when the scallops appear in the curves.

■