Exercises Reinforcement Learning: An Introduction

Contents

Code for exercises can be found at github.com/brynhayder/reinforcement_learning_an_introduction

Note that equation numbers in questions will refer to the original text.

1 Introduction

1.1 Exercise 1.1: Self-Play

Q

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?

A

- Would learn a different policy than playing a fixed opponent since the opponent would also be changing in this case.
- May not be able to learn an optimal strategy as the opponent keeps changing also.
- Could get stuck in loops.
- Policy could remain static since on average they would draw each iteration.

1.2 Exercise 1.2: Symmetries

Q

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?

A

- We could label the states as unique up to symmetries so that our search space is smaller, this way we will get a better estimate of optimal play.
- If we are playing an opponent who does not take symmetries into account when they are playing then we should not label the states as the same since the opponent is part of the environment and the environment is not the same in those states.

1.3 Exercise 1.3: Greedy Play

Q

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. Might it learn to play better, or worse, than a nongreedy player? What problems might occur

A

- The greedy player will not explore, so will in general perform worse than the non-greedy player
- If the greedy player had a perfect estimate of the value of states then this would be fine.

1.4 Exercise 1.4: Learning from Exploration

Q

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?

A

I think that an estimate for the probability of the state producing a win should be based on the optimal moves from that state.

- The one in which we only record the optimal moves is the probability of our optimal agent winning. If we include exploration then this is the probability of the training agent winning.
- Better to learn the probability of winning with no exploration since this is how the agent will perform in real time play.
- Updating from optimal moves only will increase probability of winning.

1.5 Exercise 1.5: Other Improvements

Q

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?

A

I'm not too sure here...

- We could rank the draws as better than the losses.
- We might like to try running multiple iterations of games before updating our weights as this might give a better estimate.

2 Multi-armed Bandits

2.1 Exercise 2.1

Q

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?

A

0.5.

2.2 Exercise 2.2: Bandit example

Q

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?

A

A_2 and A_5 were definitely exploratory. Any of the others *could* have been exploratory.

2.3 Exercise 2.3

Q

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.

A

The $\varepsilon = 0.01$ will perform better because in both cases as $t \to \infty$ we have $Q_t \to q_*$. The total reward and probability of choosing the optimal action will therefore be 10 times larger in this case than for $\varepsilon = 0.1$.

2.4 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

Let $\alpha_0 = 1$, then

$$Q_{n+1} = \left(\prod_{i=1}^{n}(1-\alpha_i)\right) Q_1 + \sum_{i=1}^{n} \alpha_i R_i \prod_{k=i+1}^{n} (1-\alpha_k). \tag{1}$$

Where $\prod_{i=x}^{y} f(i) \doteq 1$ if $x > y$.

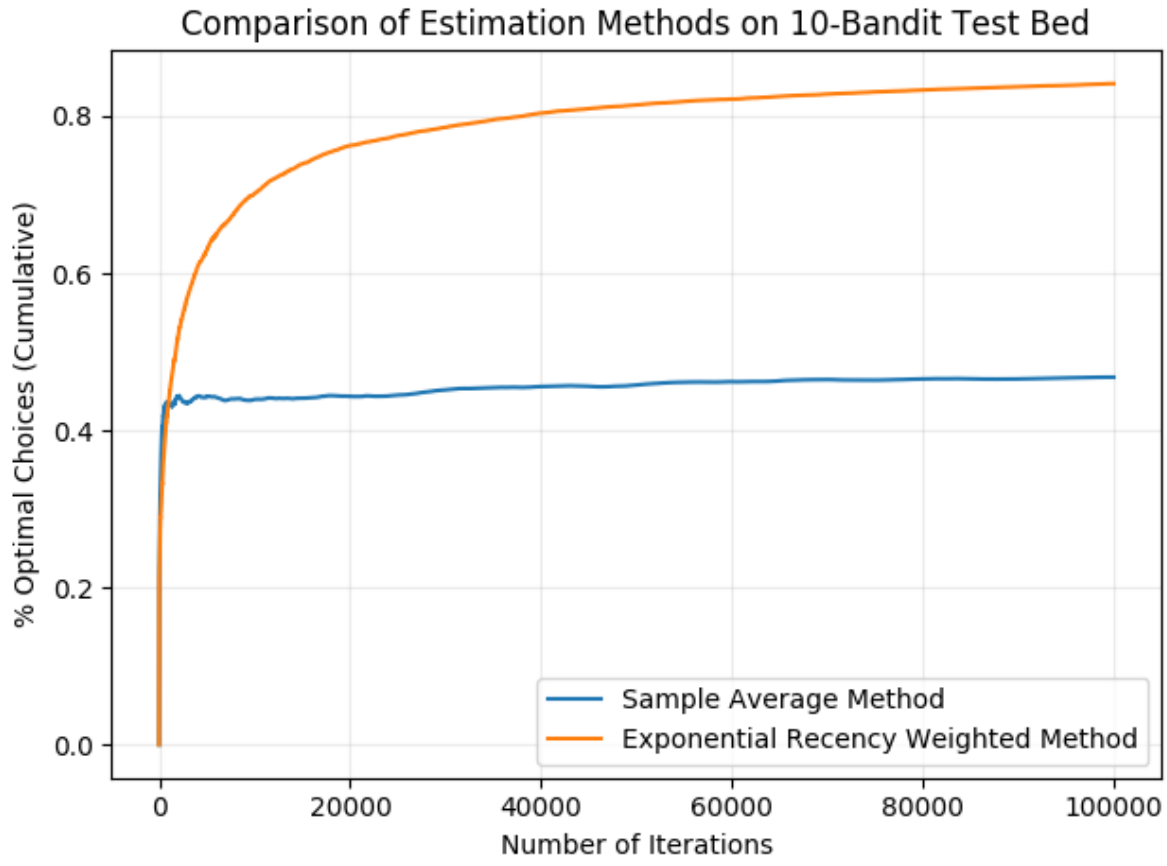
2.5 Exercise 2.5 (programming)

Q

Design and conduct an experiment to demonstrate the difficulties that sample-average methods have for non-stationary 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 zero 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 $\varepsilon = 0.1$ and longer runs, say of 10,000 steps.

A

This is a programming exercise. For the relevant code please see the repo.



2.6 Exercise 2.6: Mysterious Values

Q

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?

A

At some point after step 10, the agent will find the optimal value. It will then choose this value greedily. The small step-size parameter (small relative to the initialisation value of 5) means that the estimate of the optimal value will converge slowly towards its true value.

It is likely that this true value is less than 5. This means that, due to the small step size, one of the sub-optimal actions will still have a value close to 5. Thus, at some point, the agent begins to act sub-optimally again.

2.7 Exercise 2.7: Unbiased Constant Step Trick

Q

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 in (2.6)). However, sample averages are not a completely satisfactory solution because they may perform poorly on non-stationary problems. Is it possible to avoid the bias of constant step sizes while retaining their advantages on non-stationary problems? One way is to use a step size of

$$\beta_t \doteq \alpha/\bar{o}_t, \tag{2}$$

where $\alpha > 0$ is a conventional constant step size and $\bar{o}_t$ is a trace of one that starts at 0:

$$\bar{o}_{t+1} = \bar{o}_t + \alpha(1 - \bar{o}_t) \tag{3}$$

for $t \geq 1$ and with $\bar{o}_1 \doteq \alpha$.

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

A

Consider the answer to Exercise 2.4. There is no dependence of Q_k on Q_1 for $k > 1$ since $\beta_1 = 1$. Now it remains to show that the weights in the remaining sum decrease as we look further into the past. That is

$$w_i = \beta_i \prod_{k=i+1}^{n} (1 - \beta_k) \tag{4}$$

increases with i for fixed n. For this, observe that

$$\frac{w_{i+1}}{w_i} = \frac{\beta_{i+1}}{\beta_i(1 - \beta_{i+1})} = \frac{1}{1 - \alpha} > 1 \tag{5}$$

where we have assumed $\alpha < 1$. If $\alpha = 1$ then $\beta_t = 1 \; \forall \, t$.

2.8 Exercise 2.8: UCB Spikes

Q

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.

A

In the first 10 steps the agent cycles through all of the actions because when $N_t(a) = 0$ then a is considered maximal. On the 11th step the agent will most often then choose greedily. The agent will continue to choose greedily until $\ln(t)$ overtakes $N_t(a)$ for one of the other actions, in which case the agent begins to explore again hence reducing rewards.

Note that, in the long run, $N_t = O(t)$ and $\ln(t)/t \to 1$. So this agent is 'asymptotically greedy'.

2.9 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

Let the two actions be denoted by 0 and 1. Now

$$\mathbb{P}(A_t = 1) = \frac{e^{H_t(1)}}{e^{H_t(1)} + e^{H_t(0)}} = \frac{1}{1 + e^{-x}}, \tag{6}$$

where $x = H_t(1) - H_t(0)$ is the relative preference of 1 over 0.

2.10 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 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?

A

I assume the rewards are stationary.

One should choose the action with the highest expected reward. In the first case, both action 1 and 2 have expected value of 0.5, so it doesn't matter which you pick.

In the second case one should run a normal bandit method separately on each colour. The expected reward from identifying the optimal actions in each case is 0.55.

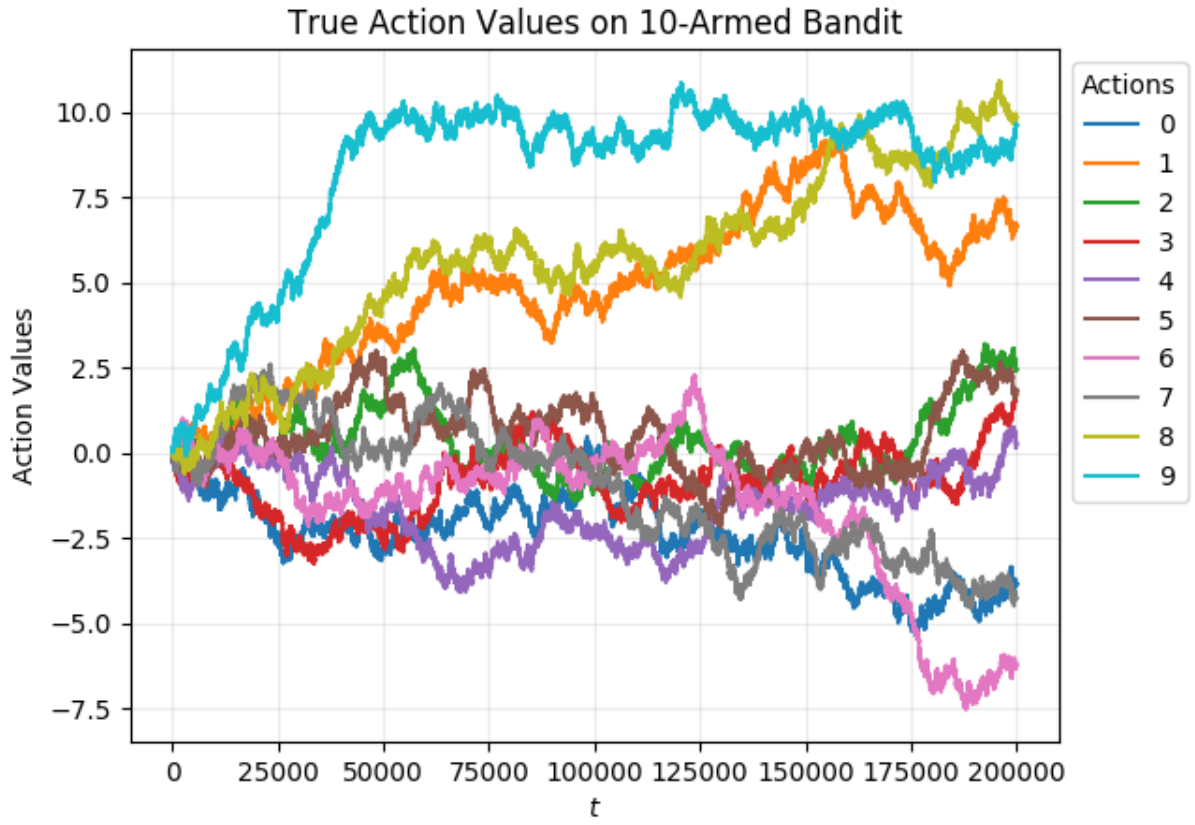
2.11 Exercise 2.11 (programming)

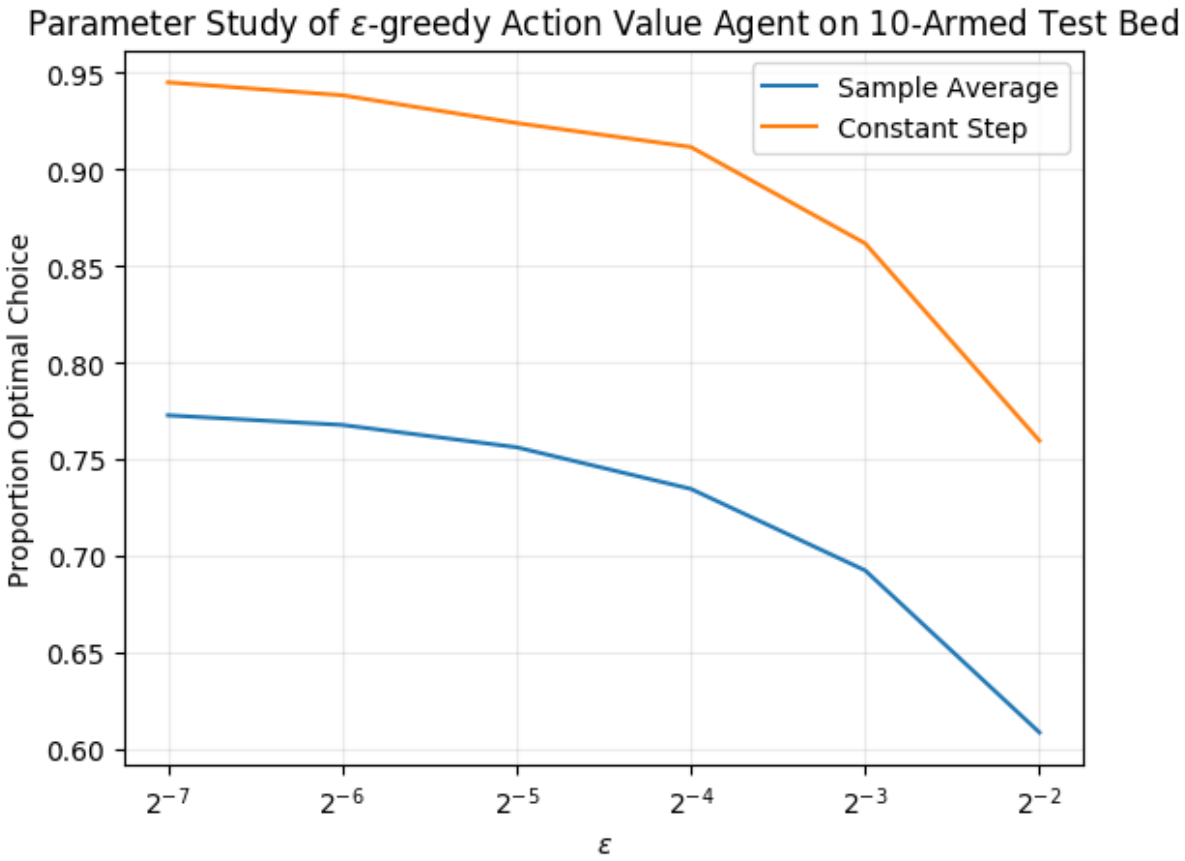
Q

Make a figure analogous to Figure 2.6 for the non-stationary case outlined in Exercise 2.5. Include the constant-step-size ε-greedy algorithm with $\alpha = 0.1$. Use runs of 200,000 steps and, as a performance measure for each algorithm and parameter setting, use the average reward over the last 100,000 steps.

A

This is a programming exercise. For the relevant code please see the repo.




Parameter Study of ε-greedy Action Value Agent on 10-Armed Test Bed
Sample Average
Constant Step
Proportion Optimal Choice
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
2^{-7}
2^{-6}
2^{-5}
2^{-4}
2^{-3}
2^{-2}
ε

3 Finite Markov Decision Processes

3.1 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. Simple example is a robot that hoovers a room. The state can be how much dust there is on the ground and where the robot is (including it's orientation). Actions can be to move and hoover. The reward can be the amount by which it reduces dust in the room on that action. This is Markov because all that is important from the previous state to the future is where the dust is left (maybe also how much).

2. Outlandish example is a football coach. Actions are playing strategies. Rewards are goals. State is current score, team fitness, etc..

3. financial trader. State is their current holdings on an asset. Reward is money from a trade. Actions are buy/sell. Maybe not markov because the environment may change predictably based on information from multiple steps ago.

3.2 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 main thing about the MDP is that Markov property. There are tasks where this does not hold. For instance, in Poker, the previous states will determine what is in the deck and what is not. This does not obey Markov property.

3.3 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

- The natural distinction depends on the task. If the task is to go from one location to another, the actions might be in terms of directing the car and altering its speed.
- This also depends on what we consider to be the decision making part here. If we consider the decisions to be made by the brain of a human driving, then their physical body will form part of the environment – if they break their leg then it will effect their goal.
- Actions have to be things that the agent can actually control. Take the example of an autonomous vehicle. One might consider that the agent has complete control over the car's break, accelerator and steering wheel. These operations would form the actions for the agent.

3.4 Exercise 3.4

Q

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

A

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

Table 1: Transition table

3.5 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

$$\sum_{s' \in \mathcal{S}^+} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1 \tag{7}$$

3.6 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

Note that the pole will fall eventually with probability 1

$$G_t = -\gamma^{T-t}. \tag{8}$$

Whereas in the continuing case the value is

$$G_t = -\sum_{k \in \mathcal{K}} \gamma^{k-t}, \tag{9}$$

where $\mathcal{K}$ is the set of times after t at which the pole falls over.

3.7 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

If the agent keeps going randomly, it will reach the end of the maze with probability 1, so the value of G under most strategies is 1. What you actually want is for the thing to leave the maze as quickly as possible.

Note also that there are some instances in which the agent might just get stuck in a loop. You would have to impose another rule to put the agent into a terminal state here.

3.8 Exercise 3.8

Q

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.

A

$G_0 = 2$, $G_1 = 3$, $G_2 = 2$, $G_3 = \frac{1}{2}$, $G_4 = \frac{1}{8}$, $G_5 = 0$.

3.9 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

$$G_1 = 7\frac{\gamma}{1-\gamma} \tag{10}$$

$$G_0 = 2 + 7\frac{\gamma}{1-\gamma} \tag{11}$$

3.10 Exercise 3.10

Q

Prove (3.10).

A

Take $r \in \mathbb{C}$ with $|r| < 1$, then

$$\begin{aligned} S_N &\doteq \sum_{i=0}^{N} r^i \\ rS_N - S_N &= r^{N+1} - 1 \\ S_N &= \frac{1 - r^{N+1}}{1 - r} \\ S &\doteq \lim_{N \to \infty} S_N = \frac{1}{1 - r} \end{aligned}$$

3.11 Exercise 3.11

Q

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)?

A

$$\mathbb{E}_\pi[R_{t+1}|S_t = s] = \sum_a \pi(a|s) \sum_{s',r} rp(s', r|s, a) \tag{12}$$

3.12 Exercise 3.12

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

$$\begin{aligned} v_\pi(\texttt{center}) &\doteq \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \\ &= \frac{1}{4} \times 0.9 \times \sum_{s'} v_\pi(s') \\ &= \frac{1}{4} \times 0.9 \times 3.0 \\ &= 0.675 \end{aligned}$$

3.13 Exercise 3.13

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, $q_\pi(s', a')$, 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.

A

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

3.14 Exercise 3.14

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

We choose actions by relative (additive) values. Add c to all states and

$$G_t \mapsto G_t + \frac{c}{1-\gamma} \tag{13}$$

so relative values unchanged.

$v_c = \frac{c}{1-\gamma}$

3.15 Exercise 3.15

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

Let terminal time be T. In this case we have

$$G_t \mapsto G_t + c\frac{1-\gamma^T}{1-\gamma}, \tag{14}$$

so if the agent can procrastinate termination, then all else being equal it will increase v_π.

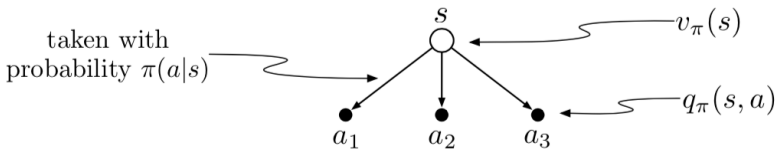
Suppose that we have an episodic task with one state S and two actions A_0, A_1. A_0 takes agent to terminal state with reward 1, while A_1 takes agent back to S with reward 0. In this case the agent should terminate to maximise reward.

If we add 1 to each reward then the return for doing A_1 forever is $\frac{1}{1-\gamma}$. Which can be bigger than 2 if we choose a discount factor smaller than $\frac{1}{2}$.

3.16 Exercise 3.16

Q

Exercise 3.16 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. □

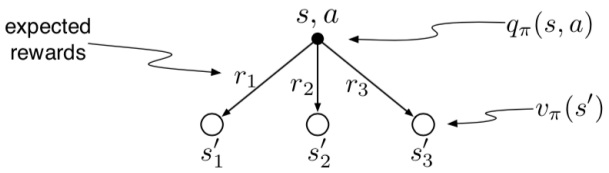
A

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

3.17 Exercise 3.17

Q

Exercise 3.17 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 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. □

A

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

3.18 Exercise 3.18

Q

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

A

(Using the pictures.) Optimal state value gives values according to `driver` when off the green, then according to `putter` on the green.

Optimal policy is to use `driver` when off green and `putter` when on green.

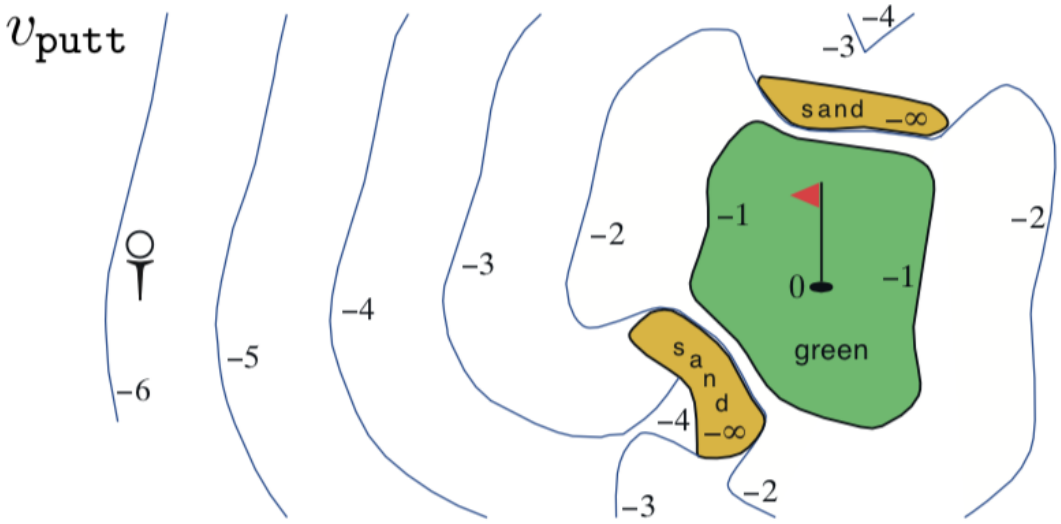
3.19 Exercise 3.19

Q

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

A

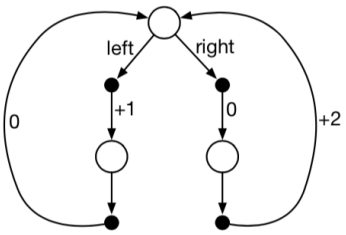
Should be the same as the value for the policy that always uses the `putter`.



3.20 Exercise 3.20

Q

Exercise 3.20 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 $\gamma = 0.5$? □



A

When $\gamma = 0$, $v_{\pi_{\texttt{left}}}$ is optimal. When $\gamma = 0.5$, they are both optimal. When $\gamma = 0.9$, $v_{\pi_{\texttt{right}}}$ is optimal.

3.21 Exercise 3.21

Q

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

A

This is just writing out the equation below, filling in the values given in the robot example.

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q_*(s',a')] \tag{15}$$

3.22 Exercise 3.22

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

All actions tak the agent to A' with reward 10. We can see that $\gamma = 16.0/17.8 = 0.9$. This means that

$$v = 10 + 16 \times 0.9 = 24.4. \tag{16}$$

This is using the following framework

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]. \tag{17}$$

3.23 Exercise 3.23

Q

Give an equation for v_* in terms of q_*

A

$$v_*(s) = \sum_a \pi^*(a|s) q_*(s,a) \tag{18}$$

3.24 Exercise 3.24

Q

Give an equation for q_* in terms of v_* and the world's dynamics $p(s', r|s, a)$.

A

$$\begin{aligned} q_*(s,a) &= \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] && (19) \\ &= \sum_{s',r} (r + \gamma v_*(s')) p(s', r|s, a) && (20) \end{aligned}$$

3.25 Exercise 3.25

Q

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

A

This is just any policy that acts greedily w.r.t. the optimal action-value function.

$$\pi_*(a|s) = \frac{\mathbb{1}\{a = \text{argmax}_{a'} \, q_*(a', s)\}}{\sum_a \mathbb{1}\{a = \text{argmax}_{a'} \, q_*(a', s)\}} \tag{21}$$

3.26 Exercise 3.26

Q

Give an equation for π_* in terms of v_* and the world's dynamics $p(s', r|s, a)$.

A

This is just the answer to 3.25 with the answer to 3.24 substituted in for q_*.

4 Dynamic Programming

4.1 Exercise 4.1

Q

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

A

$q_\pi(11, \texttt{down}) = -1$ since goes to terminal state. $q_\pi(7, \texttt{down}) = -15$.

4.2 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

$v_\pi(15) = -20$ if dynamics unchanged. If dynamics changed then apparently the state value is the same, but you would need to verify Bellman equations for all states for this.

4.3 Exercise 4.3

Q

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

A

$$q_{k+1}(s,a) = \sum_{s',r} p(s',r|s,a) \left[r + \gamma \sum_{a'} \pi(a'|s) q_k(s',a') \right] \tag{22}$$

4.4 Exercise 4.4

Q

The policy iteration algorithm on the previous page 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.

A

One problem is that the argmax_a has ties broken arbitrarily, this means that the same value function can give rise to different policies.

The way to solve this is to change the algorithm to take the whole set of maximal actions on each step and see if this set is stable and see if the policy is stable with respect to choosing actions from this set.

4.5 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 q_*. Please pay special attention to this exercise, because the ideas involved will be used throughout the rest of the book.

A

We know that

$$v_\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s) q_\pi(s, a) \tag{23}$$

so we know that

$$q_\pi(s, \pi'(s)) \geq \sum_{a \in \mathcal{A}(s)} \pi(a|s) q_\pi(s, a) \tag{24}$$

if π' is greedy with respect to π. So we know the algorithm still works for action values.

All there is now is to substitute the update for the action-value update and make the policy greedy with respect to the last iteration's action-values. Also need to make sure that the argmax_a is done consistently.

4.6 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 $\varepsilon/|\mathcal{A}(s)|$. 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_π (page 80).

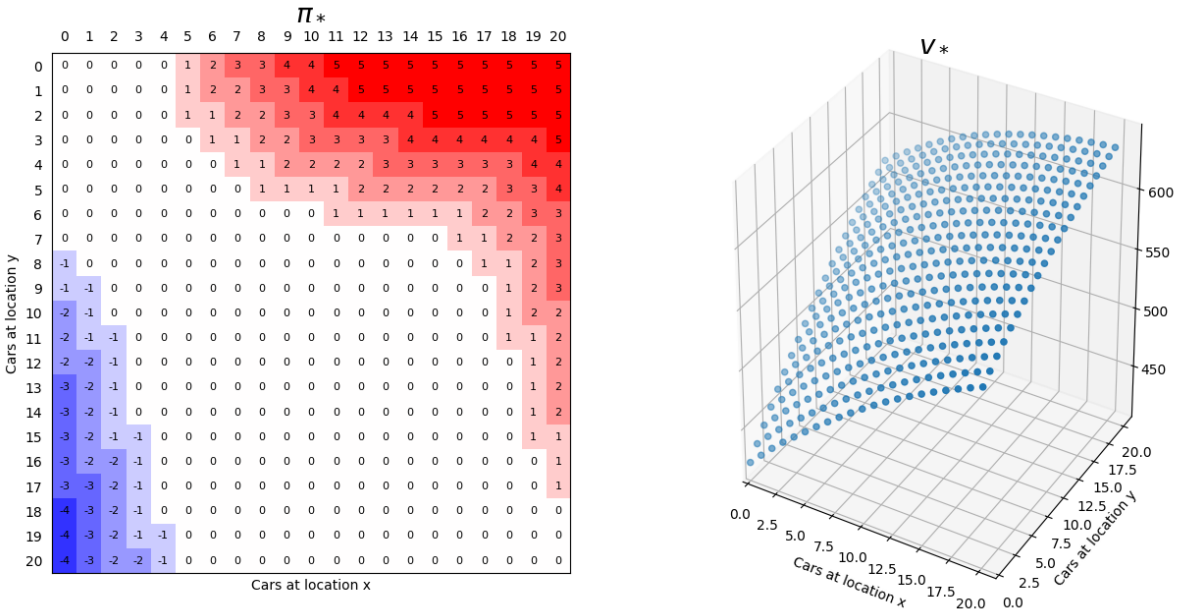
A

1. No change (but need policy to be able to be stochastic of course)
2. Need to re-write the Bellman update $v(s) \longleftarrow \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v(s')\right]$
3. Construct a greedy policy that puts weight on the greedy actions but is ε-soft. Be careful with the consistency of the argmax.

4.7 Exercise 4.7 (programming): Jack's Car Rental

Example 4.2: Jack's Car Rental Jack manages two locations for a nationwide car rental company. Each day, some number of customers arrive at each location to rent cars. If Jack has a car available, he rents it out and is credited \$10 by the national company. If he is out of cars at that location, then the business is lost. Cars become available for renting the day after they are returned. To help ensure that cars are available where they are needed, Jack can move them between the two locations overnight, at a cost of \$2 per car moved. We assume that the number of cars requested and returned at each location are Poisson random variables, meaning that the probability that the number is n is $\frac{\lambda^n}{n!}e^{-\lambda}$, where λ is the expected number. Suppose λ is 3 and 4 for rental requests at the first and second locations and 3 and 2 for returns. To simplify the problem slightly, we assume that there can be no more than 20 cars at each location (any additional cars are returned to the nationwide company, and thus disappear from the problem) and a maximum of five cars can be moved from one location to the other in one night. We take the discount rate to be $\gamma = 0.9$ and formulate this as a continuing finite MDP, where the time steps are days, the state is the number of cars at each location at the end of the day, and the actions are the net numbers of cars moved between the two locations overnight. Figure 4.2 shows the sequence of policies found by policy iteration starting from the policy that never moves any cars. ■

First we reproduce the original results.



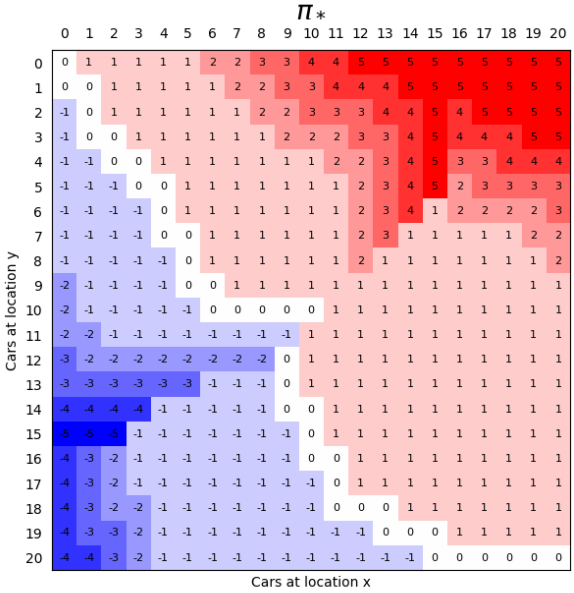
Q

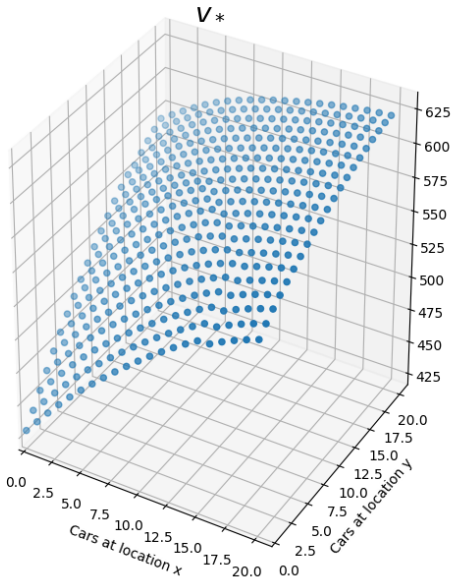
Write a program for policy iteration and re-solve Jack's car rental problem with the following changes. One of Jack's employees at the first location rides a bus home each night and lives near the second location. She is happy to shuttle one car to the second location for free. Each additional car still 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 nonlinearities and arbitrary dynamics often occur in real problems and cannot easily be handled by optimisation methods other than dynamic programming. To check your program, first replicate the results given for the original problem. If your computer is too slow

for the full problem, cut all the numbers of cars in half.

A

This is a programming exercise. For the relevant code please see the repo.





4.8 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

Since the coin is biased against us, we want to minimize the number of flips that we take. At 50 we can win with probability 0.4. At 51 if we bet small then we can get up to 52, but if we lose then we are still only back to 50 and we can again with with probability 0.4. (There is a whole paper on this problem called how to gamble if you must.)

4.9 Exercise 4.9 (programming): Gambler's Problem

Q

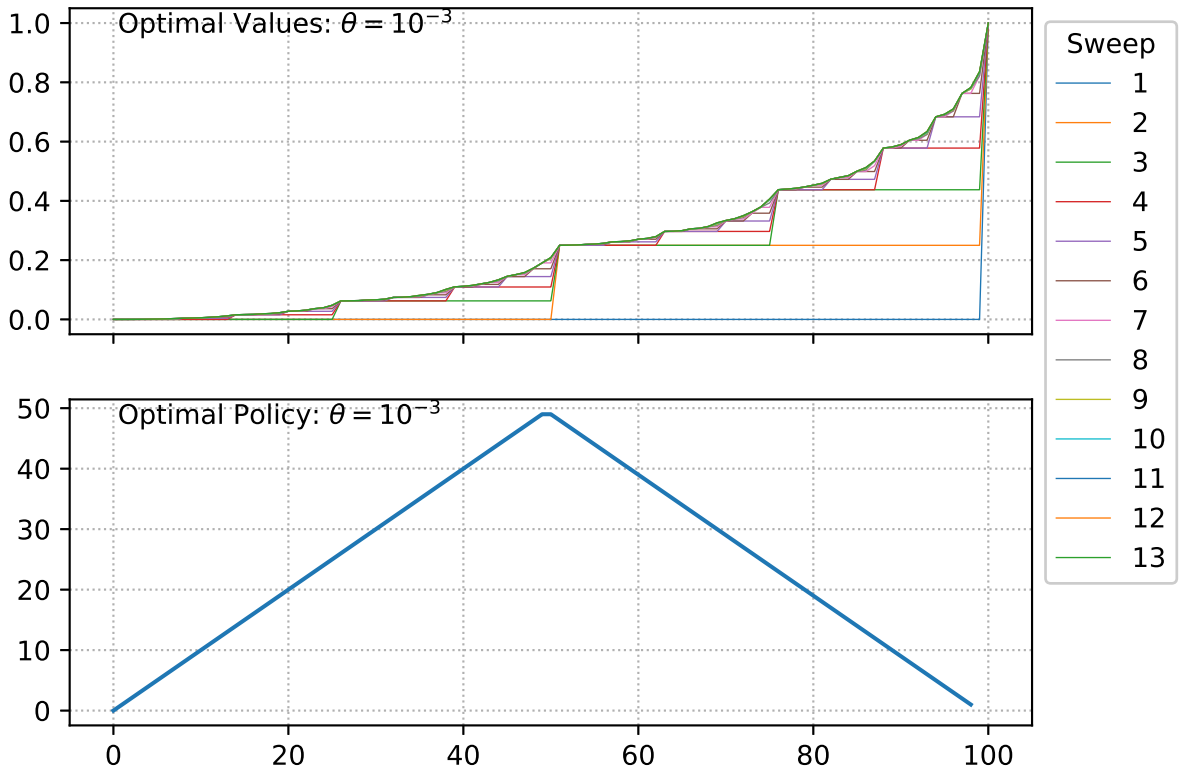
Implement value iteration for the gambler's problem and solve it for $p_h = 0.25$ and $p_h = 0.55$. In programming, you may find it convenient to introduce two dummy states corresponding to termination with capital of 0 and 100, giving them values of 0 and 1 respectively. Show your results graphically,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

A

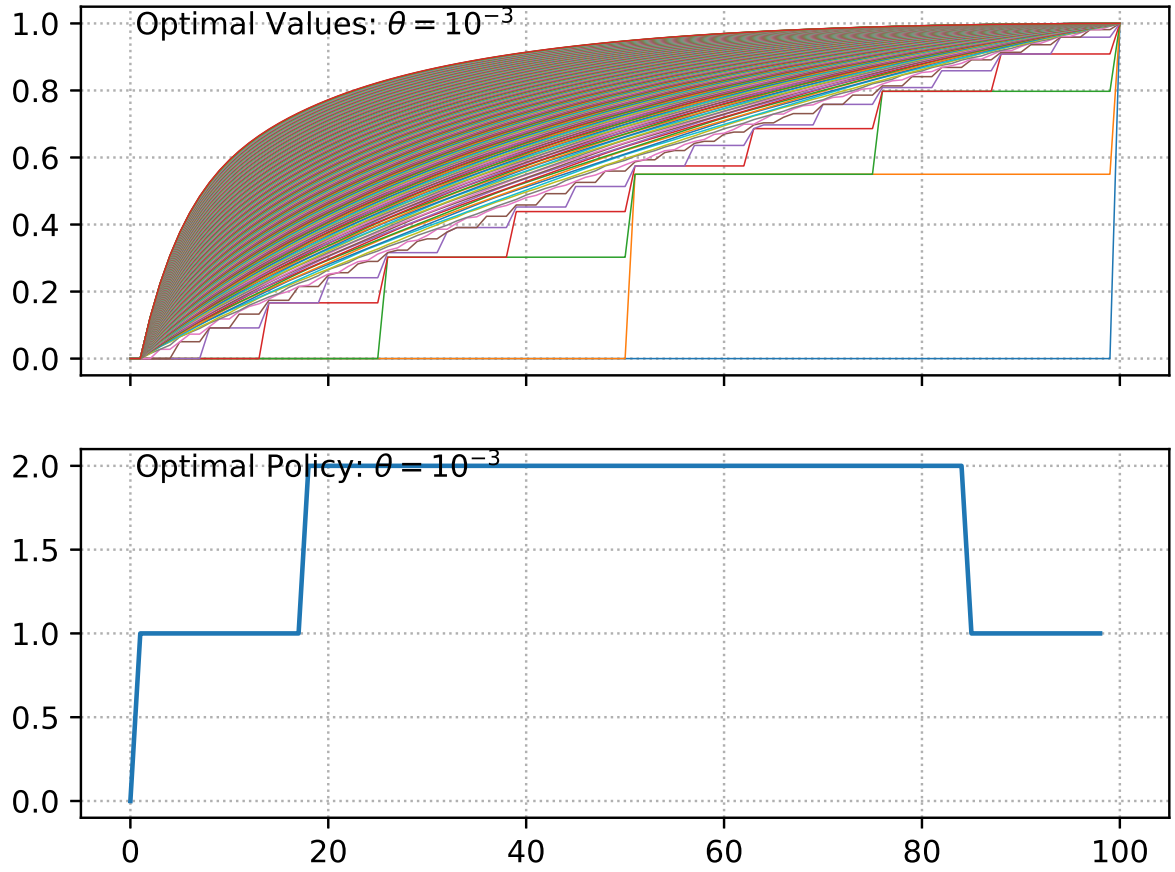
This is a programming exercise. For the relevant code please see the repo.

The process was stable as $\theta \to 0$ for $\mathbb{P}(\texttt{win}) < 0.5$.

$\mathbb{P}(\texttt{win}) = 0.25$



$$\mathbb{P}(\texttt{win}) = 0.55$$



4.10 Exercise 4.10

Q

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

A

$$q_{k+1} = \max_{a'} \sum_{s',r} p(s', r|s, a) \left[r + \gamma q_k(s', a') \right] \tag{25}$$

5 Monte-Carlo Methods

5.1 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

- Policy is to hit unless $S \geq 20$. So you run a rik of going bust if you have 12-19, but you most likely win when you stick on 20 or 21
- Drops off because dealer has a usable ace
- Frontmost higher because you're less likely to go bust, but you still might get to 20 or 21 (π always hits here).

5.2 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

Results would be the same because this game is memoryless (cards are drawn with replacement).

5.3 Exercise 5.3

Q

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

A

The same as the one shown in the book for state valus, only we have state-action pairs instead of states.

5.4 Exercise 5.4

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

We condition on taking action a in state s.

$$q_\pi(s, a) = \mathbb{E}_\pi[\rho_{t+1:T-1} G_t | S_t = s, A_t = s]$$

with returns generated from b. We estimate this quantity by

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

where $\mathcal{T}(s,a)$ now contains timestamps of visits to state-action pairs.

5.5 Exercise 5.5

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

When there are fewer episodes the importance sampling ratios will be zero with higher probability since the behaviour policy will stick on values smaller than 20 (since it is random). Zero happens to be close to $v_\pi(s)$.

This effect lessens as we get more diversity in the episode trajectories.

Then after this the error reduces because the variance in the estimator reduces.

5.6 Exercise 5.6

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

Yes, all terms in the sum are ≥ 0 and there woud just be more of them.

5.7 Exercise 5.7

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

Algo is the same apart from

- Initialise $V(s) = 0 \quad \forall s \in S$
- Don't need *Returns(s)* lists.
- Remove the last two lines and put in

$$V(S_t) \leftarrow V(S_t) + \frac{1}{T-t}[G_t - V(S_t)]$$

5.8 Exercise 5.8

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

Have $C_0 = 0$, $C_n = \sum_{k=1}^{n} W_k$ and

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

Therefore,

$$\begin{aligned} C_n V_{n+1} &= \sum_{k+1}^{n-1} W_k G_k + W_k G_k && (26) \\ &= C_{n-1} V_n + W_n G_n && (27) \\ &= (C_n - W_n) V_n + W_n G_n. && (28) \end{aligned}$$

Finally

$$V_{n+1} = V_n + \frac{W_n}{C_n}[G_n - V_n].$$

5.9 Exercise 5.9

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 $\pi(A_t|S_t)$, but instead it involves $1/b(A_t|S_t)$. Why is this nevertheless correct?

A

π is greedy, so

$$\pi(a|s) = \mathbb{1}\{a = \operatorname*{argmax}_{a'} Q(s, a')\}.$$

5.10 Exercise 5.10 (programming): Racetrack

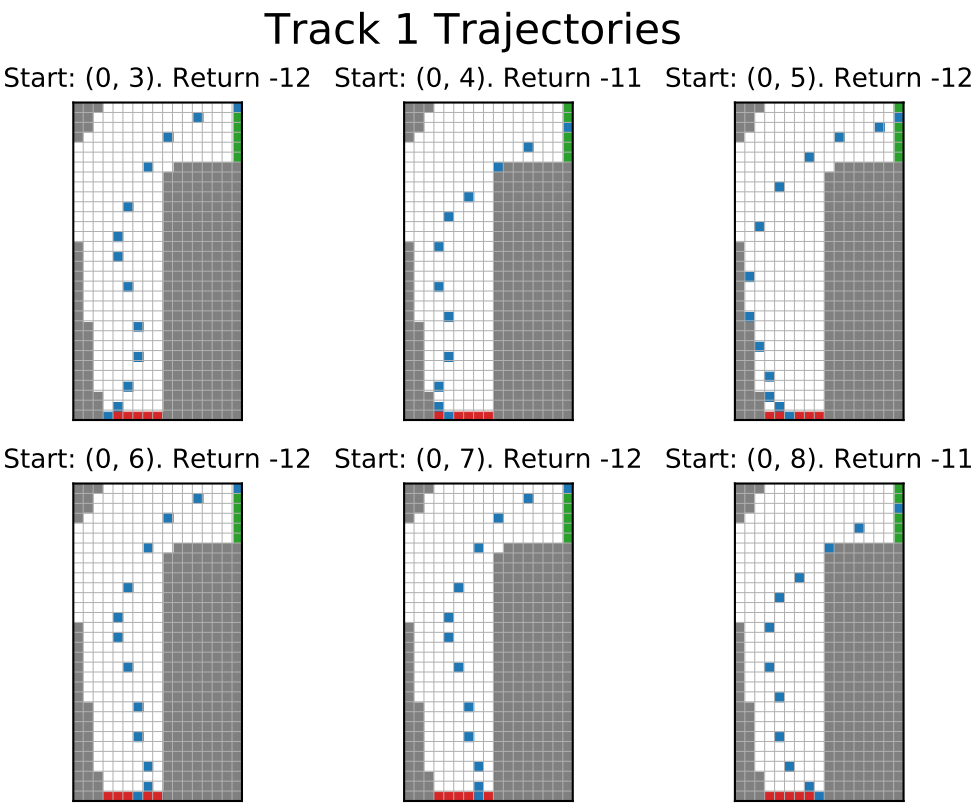
Q

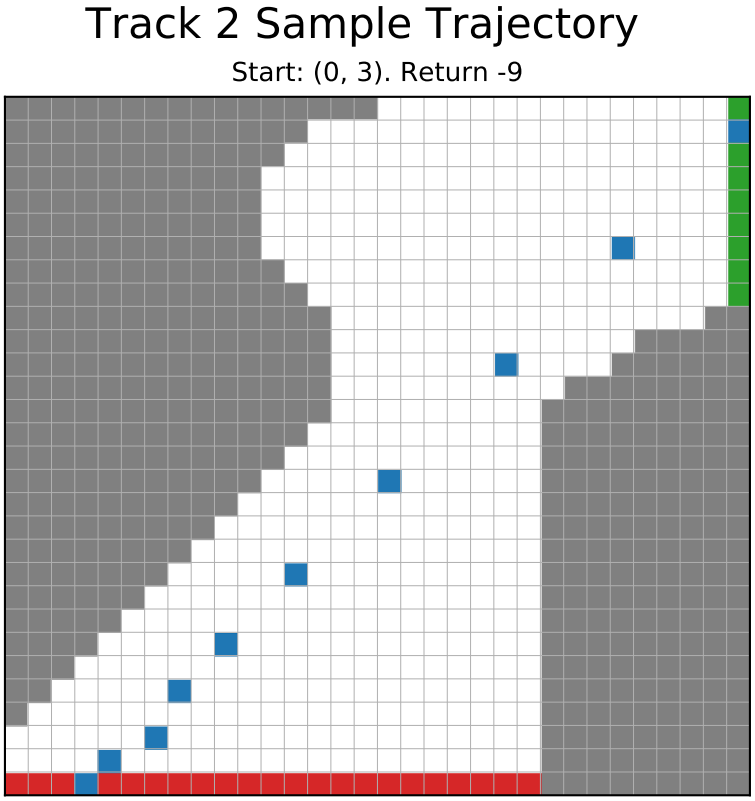
Consider driving a race car around a turn like those shown in Figure 5.5. You want to go as fast as possible, but not so fast as to run off the track. In our simplified racetrack, the car is at one of a discrete set of grid positions, the cells in the diagram. The velocity is also discrete, a number of grid cells moved horizontally and vertically per time step. The actions are increments to the velocity components. Each may be changed by $+1$, -1, or 0 in each step, for a total of nine (3×3) actions. Both velocity components are restricted to be nonnegative and less than 5, and they cannot both be zero except at the starting line. Each episode begins in one of the randomly selected start states with both velocity components zero and ends when the car crosses the finish line. The rewards are -1 for each step until the car crosses the finish line. If the car hits the track boundary, it is moved back to a random position on the starting line, both velocity components are reduced to zero, and the episode continues. Before updating the car's location at each time step, check to see if the projected path of the car intersects the track boundary. If it intersects the finish line, the episode ends; if it intersects anywhere else, the car is considered to have hit the track boundary and is sent

back to the starting line. To make the task more challenging, with probability 0.1 at each time step the velocity increments are both zero, independently of the intended increments. Apply a Monte Carlo control method to this task to compute the optimal policy from each starting state. Exhibit several trajectories following the optimal policy (but turn the noise off for these trajectories).

A

This is a programming exercise. For the relevant code please see the repo.





5.11 *Exercise 5.11

Q

Modify the algorithm for off-policy Monte Carlo control (page 110) 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

...

6 Temporal-Difference Learning

6.1 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

Write

$$\delta_t \doteq R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t),$$

then

$$\begin{aligned}
G_t - V_T(S_t) &= R_{t+1} + \gamma G_{t+1} - V_t(S_t) && (29)\\
&= R_{t+1} + \gamma(G_{t+1} - V_T(S_{t+1})) + \gamma(V_T(S_{t+1}) - V_t(S_{t+1})) - (V_t(S_k) - V_k(S_k)) && (30)\\
&= R_{t+1} + \gamma(G_{t+1} - V_T(S_{t+1})) + \gamma\varepsilon_T^t - \varepsilon_t^t && (31)\\
&= \sum_{k=t}^{T-1} \gamma^{k-t}\delta_k + \sum_{k=t}^{T-1} \gamma^{k-t}[\gamma\varepsilon_{k+1}^t - \varepsilon_k^t], && (32)
\end{aligned}$$

where

$$\varepsilon_k^t \doteq V_T(S_k) - V_t(S_k).$$

6.2 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 task?

A

TD updates incorporate prior information. Suppose we had a good value estimate for a trajectory $\tau = S_1, S_2, \ldots, S_T$, then if we try to estimate the trajectory $\tau = S_0, S_1, S_2, \ldots, S_T$ using MC then we need to see multiple episodes of this to get a good estimate of $V(S_0)$, not leveraging the info we already have on τ. TD would use info on τ to back up the value of S_0 and hence converge much quicker. The key differences here are bootstrapping and online learning.

6.3 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

All states apart from the terminal states were initialised to the same value (the terminal states must be initialised to 0) and the reward for non-terminal transitions is 0, so the TD(0) updates do nothing on the first pass to states that cannot lead directly to termination.

In the first run, the agent terminated on the left.

$$V_1(A) = V_0(A) + \alpha[0 + \gamma \times 0 + V_0(A)] \tag{33}$$
$$= (1 - \alpha)V_0(A) \tag{34}$$
$$= 0.9 \times 0.5 \tag{35}$$
$$= \frac{9}{20}. \tag{36}$$

The value of the estimate for A reduced by $\alpha V_0(A) = 0.05$.

6.4 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

- General arguments given earlier about the benefits of TD are independent of α
- Increases in α make the curve more
- Decreases in α make the curve more smooth but make it converge slower.
- We see enough of a range here to decide between the two methods

6.5 *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 αs. 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

The state C happens to have een initialised to its true value. As training starts, updates occur on outer states (making them more accurate individually) which makes the error across all states reduce. This happens until the residual inaccuracies in the outer states propagate to C. The higher values of α make this effect more pronounced, because the value estimate for C changes more readily in these cases.

6.6 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

Could have used DP, but probably did the following calculation.

First note that

$$V(s) = \mathbb{E}[\mathbb{1}\{\text{terminate on right from } S\}] = \mathbb{P}(\text{terminate on right from } S).$$

Also recognise that symmetry now implies $V(C) = 0.5$. Now

$$\begin{aligned} V(E) &= \frac{1}{2} \times 1 + \frac{1}{2} \times V(D) \\ &= \frac{1}{2} + \frac{1}{4}[V(C) + V(E)], \end{aligned}$$

so $V(E) = \frac{5}{6}$. We then get $V(D) = \frac{4}{6}$ and we can calculate the other states in the same way.

6.7 *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 behavior policy b, using at each step t the importance sampling ratio $\rho_{t:t}$(5.3).

A

Let G_t be returns from an episode generated by b. Then

$$\begin{aligned} v_\pi(s) &= \mathbb{E}[\rho_{t:T-1} G_t | S_t = s] \\ &= \mathbb{E}[\rho_{t:T-1} R_{t+1} + \gamma \rho_{t:T-1} G_{t+1} | S_t = s] \\ &= \rho_{t:t} \mathbb{E}[R_{t+1} | S_t = s] + \gamma \rho_{t:t} \mathbb{E}[\rho_{t+1:T-1} G_{t+1} | S_t = s] \\ &= \rho_{t:t} \left(\mathbb{E}[R_{t+1} | S_t = s] + \mathbb{E}[\rho_{t+1:T-1} G_t | S_t = s] \right) \\ &= \rho_{t:t} \left(r(s, A_t) + v_\pi(S_{t+1}) \right). \end{aligned}$$

So the update for off-policy TD(0) (by sampling approximation) is

$$V(S_t) \leftarrow V(S_t) + \alpha \left[\rho_{t:t} R_{t+1} + \rho_{t:t} \gamma V(S_{t+1}) - V(S_t) \right]. \tag{37}$$

6.8 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

Write $\delta_t \doteq R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$. Then the Monte-Carlo error is

$$\begin{aligned} G_t - Q(S_t, A_t) &= R_{t+1} + \gamma G_{t+1} - Q(S_t, A_t) \\ &= \delta_t - \gamma[Q(S_{t+1}, A_{t+1}) + G_{t+1}] \\ &= \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k \end{aligned}$$

6.9 Exercise 6.9 (programming): Windy Grid World with King's Moves

Q

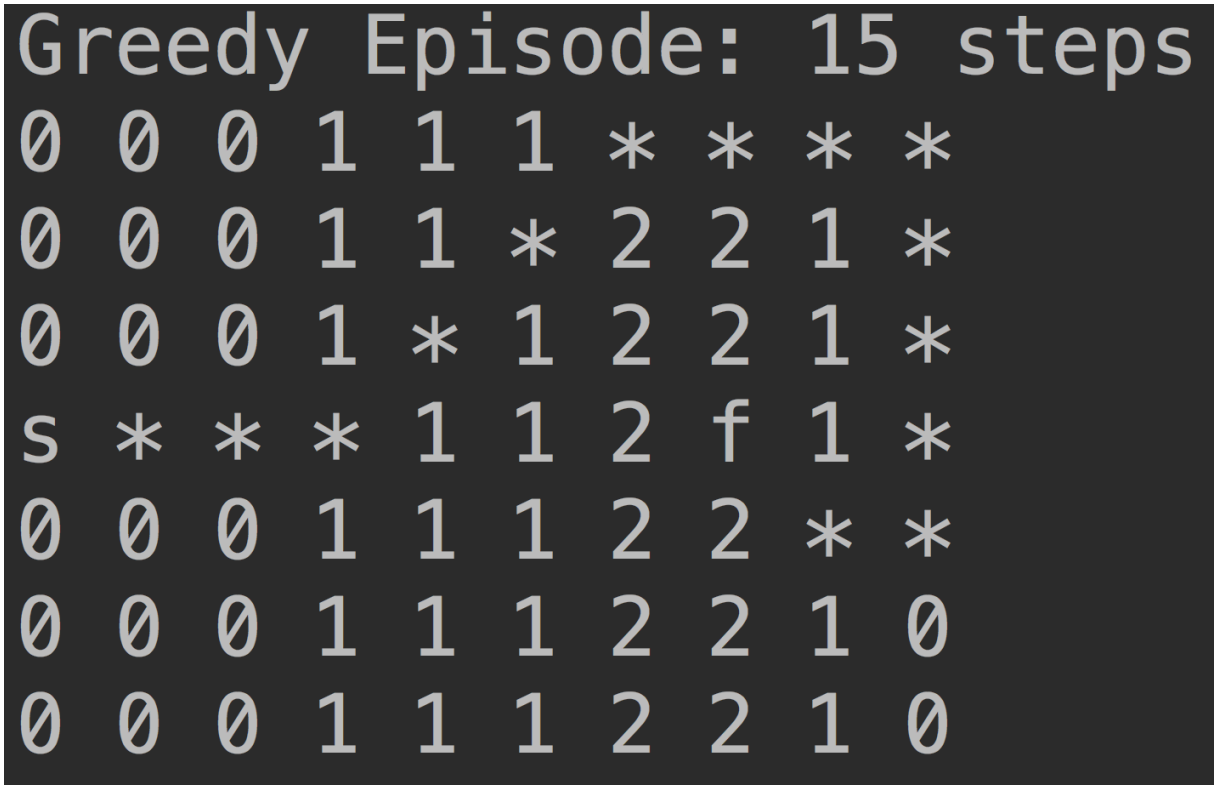
Re-solve the windy gridworld assuming eight possible actions, including the diagonal moves, rather than the usual four. How much better can you do with the extra actions? Can you do even better by including a ninth action that causes no movement at all other than that caused by the wind?

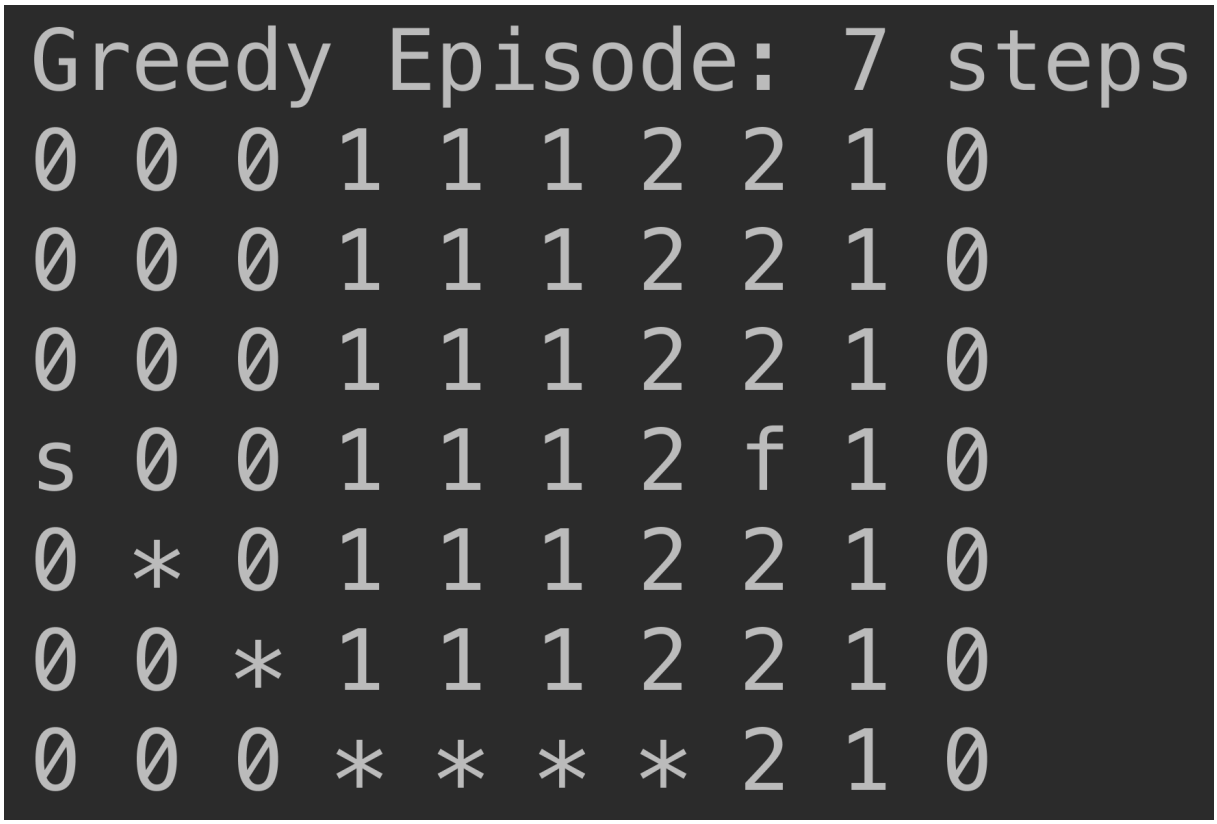
A

This is a programming exercise. For the relevant code please see the repo.

Optimal trajectory is now 7 steps, rather than 15. Including the do-nothing action is not helpful in this example because the wind blows vertically and the goal position is not vertically separated from the start position. It could be useful in other wind environments though.

Below are the optimal trajectories for the book example (no diagonal moves) and for the exercise (king's moves). The numbers represent the wind strength in that position.





6.10 Exercise 6.10 (programming): Stochastic Wind

Q

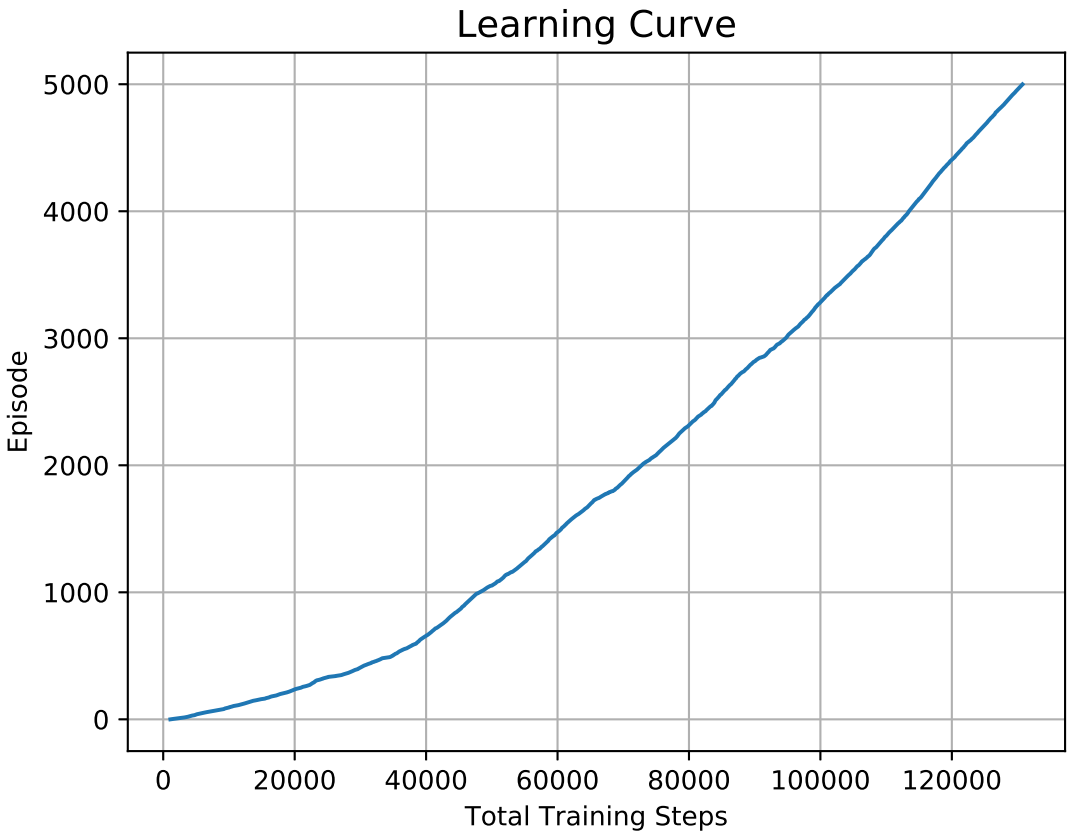
Re-solve the windy gridworld task with King's moves, assuming that the effect of the wind, if there is any, is stochastic, sometimes varying by 1 from the mean values given for each column. That is, a third of the time you move exactly according to these values, as in the previous exercise, but also 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

This is a programming exercise. For the relevant code please see the repo.

Greedy trajectory and learning curve shown below. Note that although the gradient of the learning curve becomes constant (so the algorithm converges), the greedy episode shown suffers from stochasticity in the wind.

```
Greedy Episode: 26 steps
0 0 0 1 1 1 2 2 1 0
0 0 0 1 1 1 2 2 1 0
0 0 0 1 1 1 2 2 * *
s 0 0 1 1 1 2 f * *
0 * * 1 1 1 2 * * *
0 0 0 1 1 1 * * 1 0
0 0 0 * * * 2 2 * 0
```



6.11 Exercise 6.11

Q

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

A

The returns are sampled as if the agent followed the greedy policy with respect to Q.

6.12 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

Yes (?)

6.13 Exercise 6.13

Q

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

A

Expected SARSA has the update

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}) | S_{t+1}] - Q(S_t, A_t) \right]$$

The update for S_t, A_t is

$$R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t).$$

Double expected SARSA would be keeping two Q arrays and updating one of them each timestep, chosen with equal probability.

For an ε-greedy policy we would increment $Q_1(S_t, A_t)$ by

$$\alpha \left[R_{t+1} + \gamma \left(\frac{\varepsilon}{|\mathcal{A}(a)|} \sum_a Q_2(S_{t+1}, a) + (1 - \varepsilon) \max_a \{ Q_2(S_{t+1}, a) \} \right) - Q_1(S_t, A_t) \right] \tag{38}$$

and the same with 1 and 2 reversed.

6.14 Exercise 6.14

Q

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

A

One might have coded this up initially with the states as the number of cars in each garage each evening. The agent then takes some action (moves some cars) and we transition stochastically to some state.

An alternative would be to introduce the number of cars in the morning (after the agent has moved cars) as an afterstate. This is because the agent is able to deterministically change the environment from evening to next morning (before rentals or returns).

In this case we would speed convergence by reducing the number of action-values to be calculated. For instance, we can now evaluate $(10, 0)$ moving one car and $(9, 1)$ moving no cars as the same afterstate $(9, 1)$.