1 Chapter 1

2 Chapter 2

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

Write $G := \{\text{greedy action is selected}\}$. We have

$$\begin{aligned}\mathbb{P}[\,G\,] =& \mathbb{P}[\,G\,, random] + \mathbb{P}[\,G\,, optimal]\\ &= \varepsilon\frac{1}{2} + (1-\varepsilon)\\ &= 0.75.\end{aligned}$$

Exercise 2.2. *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 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 $A1 = 1, R1 = -1, A2 = 2, R2 = 1, A3 = 2, R3 = -2, A4 = 2, R4 = 2, A5 = 3, R5 = 0$. On some of these time steps the 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?*

We denote the sample average action value after n-steps as

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i 1_{A_i=a}}{\sum_{i=1}^{t-1} 1_{A_i=a}}. \tag{1}$$

Timestep t	$Q_{t+1}(1)$	$Q_{t+1}(2)$	$Q_{t+1}(3)$	$Q_{t+1}(4)$	Greedy action	Action selected
t=0	0	0	0	0	-	$A_1 = 1$
t=1	$\frac{R_1}{1_{A_1=1}} = -1$	0	0	0	$2, 3$ or 4	$A_2 = 2$
t=2	-1	1	0	0	2	$A_3 = 2$
t=3	-1	$\frac{R_2+R_3}{2} = -0.5$	0	0	3 or 4	$A_4 = 2$
t=4	-1	$\frac{R_2+R_3+R_4}{3} = \frac{1}{3}$	0	0	3	$A_5 = 3$
t=5	-1	$1/3$	0	0	3	end

1. action $A_1 = 1$ can be either exploration or exploitation because the optimal value is zero for all actions.
2. action $A_2 = 2$ could be either exploration or exploitation because action=2 is optimal too.
3. action $A_3 = 2$ could be either exploration or exploitation because action=2 is optimal too.
4. action $A_4 = 2$ is exploration because the action 2 is not optimal.
5. action $A_5 = 4$ could be either exploration or exploitation because action=3 is the only optimal choice.

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

cumulative reward We have for optimal action a_*

$$\mathbb{E}\left[R_i | A_i = a_*\right] = q_*(a_*). \tag{2}$$

The probability of choosing the optimal action among n-options is

$$\mathbb{P}\left[A_i = a_*\right] = (1-\varepsilon) + \frac{\varepsilon}{n}, \tag{3}$$

and $\mathbb{P}\left[exploration\right] = 1 - \mathbb{P}\left[A_i = a_*\right]$. The true value $q_*(a)$ each of the ten actions $a = 1, ..., 10$ was selected according to a normal distribution with mean zero and unit variance and so

$$\mathbb{E}\left[\mathbb{E}\left[R_i | exploration\right]\right] = \mathbb{E}\left[\sum_{action\ a} \mathbb{E}\left[R_i | A_i = a\right]\right] = \sum_{action\ a} \mathbb{E}\left[q_*(a)\right] = \sum_{action\ a} 0 = 0. \tag{4}$$

So the cumulative reward is from the law of total expectation

$$\begin{aligned}\mathbb{E}\left[R_i\right] =& \mathbb{E}\left[\mathbb{E}\left[R_i | A_i = a_*\right]\right] \mathbb{P}\left[A_i = a_*\right] + \mathbb{E}\left[\mathbb{E}\left[R_i | exploration\right]\right] \mathbb{P}\left[exploration\right] \\ =& \mathbb{E}\left[q_*(a_*)\right]\left((1-\varepsilon) + \frac{\varepsilon}{n}\right) + 0\left(\varepsilon - \frac{\varepsilon}{n}\right) \\ =& \mathbb{E}\left[q_*(a_*)\right]\left((1-\varepsilon) + \frac{\varepsilon}{n}\right).\end{aligned}$$

We have

$$\mathbb{E}_{0.01}[R_i] = \mathbb{E}\left[q_*(a_*)\right] 0.991 > \mathbb{E}\left[q_*(a_*)\right] 0.91 = \mathbb{E}_{0.1}[R_i]. \tag{5}$$

probability of selecting the best action Write $G := \{\text{greedy action is selected}\}$. We have for $\varepsilon = 0.01$

$$\begin{aligned}\mathbb{P}_{0.01}(\,G\,) =& \mathbb{P}(\,G\,, random) + \mathbb{P}(\,G\,, optimal) \\ =& \varepsilon\frac{1}{k} + (1-\varepsilon) \\ =& 0.01\frac{1}{10} + 0.99 = 0.991\end{aligned}$$

and similarly for $\varepsilon = 0.1$

$$\begin{aligned}Prob_{0.1}(\,G\,) =& Prob(\,G\,, random) + Prob(\,G\,, optimal) \\ =& \varepsilon\frac{1}{k} + (1-\varepsilon) \\ =& 0.1\frac{1}{10} + 0.9 = 0.91\end{aligned}$$

So we see that the first probability is slightly higher than the second one.

Exercise 2.4. *If the step-size parameters, a_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?*

We start with

$$Q_{n+1} = Q_n + \alpha_n(R_n - Q_n) = \alpha_n R_n + (1-\alpha_n)Q_n \tag{6}$$

and then we do the first two steps

$$\begin{aligned}Q_{n+1} =& \alpha_n R_n + (1-\alpha_n)Q_n \\ =& \alpha_n R_n + (1-\alpha_n)\alpha_{n-1}R_{n-1} + (1-\alpha_n)(1-\alpha_{n-1})Q_{n-1} \\ =& \alpha_n R_n + (1-\alpha_n)\alpha_{n-1}R_{n-1} + (1-\alpha_n)(1-\alpha_{n-1})\alpha_{n-2}R_{n-2} + (1-\alpha_n)(1-\alpha_{n-1})(1-\alpha_{n-2})Q_{n-2}.\end{aligned} \tag{7}$$

So if we keep going, we get

$$Q_{n+1} = \alpha_n R_n + \sum_{k=1}^{n-1}(\alpha_k \prod_{i=k+1}^{n} (1-\alpha_i))R_k + (\prod_{i=1}^{n}(1-\alpha_i))Q_1. \tag{8}$$

Exercise 2.5. *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 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 by $a = 1/n$, and another n action-value method using a constant step-size parameter, $a = 0.1$. Use $\varepsilon = 0.1$ and, if necessary, runs longer than 1000 steps.*

```
import numpy as np
import matplotlib.pyplot as plt

def RandomWalk(x):
dim=np.size(x)
walk_set=[-1,1]
for i in range(dim):
x[i]=x[i]+0.01*np.random.choice(walk_set)
return x


def exploit_explore(epsilon, Q):
# This function returns an action chosen by epsilon-greedy algorithm
#applied to the current action value vector Q
i=np.argmax(Q)
dim=np.size(Q)
action_space=range(0,dim,1)
sample=np.random.uniform(0,1)
if sample<=1-epsilon:
return i
else:
np.delete(action_space,i)
return np.random.choice(action_space)


#Algorithm for varying step size 1/k

def multiarm_varyingstep(epsilon, max_iter, tasks, arms):
'''
epsilon = the probability te agent doesn't selection a greedy action
max_iter = the number of steps taken by the value function in the random walk
tasks = the number of random bandits we will average across
arms = the number of actions a bandit can take at each step

This function returns two vectors (constR and variabR) that quanitfy the average reward
achieved across 500 bandits based on reward estimated via constant/variable stepsizes
at each timestep
'''

rows, cols = tasks, arms

# create true value of arms with random walk
q = np.array( [([0]*arms) for i in range(rows)] )
variabQ = np.array( [([0]*cols) for i in range(rows)] )
variabN = np.array( [([0]*cols) for i in range(rows)] )
variabR = np.zeros(max_iter)

# iteration loop (within each iteration we have a random number of bandits (tasks) that we will average over)
for i in range(max_iter):

# task loop (the random bandits we cycle through per each iteration)
for j in range(tasks):
# random walk of each arm
```

```
task_q = q[j, :]
task_q = RandomWalk(task_q)
q[j,:] = task_q


# get one arm from set of tasks
task_variabQ = variabQ[j,:]
task_variabN = variabN[j,:]

# find arm to pick
action_index_v = exploit_explore(epsilon,task_variabQ)

# get reward from true values of arms
reward_variab = q[j,action_index_v]

# add reward to total rewards for this iteration
variabR[i] = variabR[i] + reward_variab

# update number of times arm has been selected
task_variabN[action_index_v] = task_variabN[action_index_v] + 1
variabN[j,:] = task_variabN

# set stepsize to 1 for first iteration and varying step size 1/k otherwise.
if i == 0:
beta = 1
else:
beta = (1/task_variabN[action_index_v])

# update estimated value of action based on observation
task_variabQ[action_index_v] = task_variabQ[action_index_v] +\
                        beta*(reward_variab-task_variabQ[action_index_v])
variabQ[j,:] = task_variabQ

# get average reward across all bandits for each iteration

variabR [i] = variabR[i] / tasks

return variabR


#Algorithm for constant step size
def multiarm_constantstep(epsilon=0.1, max_iter=10000, tasks=500, arms=10, alpha=0.1):
'''
epsilon = the probability te agent doesn't selection a greedy action
max_iter = the number of steps taken by the value function in the random walk
tasks = the number of random bandits we will average across
arms = the number of actions a bandit can take at each step
alpha = the constant stepsize parameter used to calcuate new reward estimates

This function returns two vectors (constR and variabR) that quanitfy the average reward
achieved across 500 bandits based on reward estimated via constant/variable stepsizes
at each timestep
'''

rows, cols = tasks, arms

# create true value of arms with random walk
q = np.array( [([0]*arms) for i in range(rows)] )
constQ = np.array( [([0]*cols) for i in range(rows)] )
constN = np.array( [([0]*cols) for i in range(rows)] )
constR = np.zeros(max_iter)


# iteration loop (within each iteration we have a random number of bandits (tasks) that we will average over)
for i in range(max_iter):

# task loop (the random bandits we cycle through per each iteration)
for j in range(tasks):
# random walk of each arm
task_q = q[j, :]
task_q = RandomWalk(task_q)
q[j,:] = task_q
```

```
# CONSTANT STEPSIZE

# get one random arm from set of tasks
task_constQ = constQ[j,:]
task_constN = constN[j,:]

# find the arm to pick
action_index_c = exploit_explore(epsilon,task_constQ)

# get reward
reward_const = q[j,action_index_c]

# add reward to total rewards for this iteration (to be averaged later)
constR[i] = constR[i] + reward_const

# update estimated value of arm based on observation
task_constQ[action_index_c] = task_constQ[action_index_c] + alpha*(reward_const-task_constQ[action_index_c])
constQ[j,:] = task_constQ

# update number of times arm has been selected
task_constN[action_index_c] = task_constN[action_index_c] + 1
constN[j:] = task_constN

# get average reward across all bandits for each iteration
constR[i] = constR[i] / tasks

return constR


# calculate and plot
R_c_step=multiarm_constantstep()
R_v_step = multiarm_varyingstep()

fig = plt.figure(figsize=(10,5))
fig.add_subplot(111)
plt.xlabel('steps')
plt.ylabel('average reward')
plt.plot(R_c_step, 'r', label='constant stepsize')
plt.plot(R_v_step, 'g', label='varying stepsize')
plt.legend(loc='upper left')
plt.savefig('ex2.5.png', dpi=300)
plt.show()
```

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

(a) Initially in the optimistic-method all the rewards are at $Q_1(a) = +5$. However, since the mean is zero for $q_*(a) \sim N(0,1)$, this is an overshoot guess.

1. Therefore, once we explore any of the actions eg. $A_1 = 1$, the reward will likely be much less eg. $Q_1(1) = 0.1 \ll 5$.
2. So $a = 1$ will **not** be a greedy action, and with at least probability $1-\epsilon$, it will be ignored.
3. We will quickly go through all the actions, some of which will be the optimal ones and some of which will very low-reward. So we will oscillate between high and low rewards in the first few steps as we remove and update from the initial optimistic bias of +5.
4. Once the we update to more *realistic* rewards, we are back to the original 10-armed bandit.

(b) If we have some idea of which are the optimal actions, we distribute the initial optimistic rewards accordingly and also closer to the actual mean zero.

Any other ideas?

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

$$\beta_n = \frac{\alpha}{\bar{o}_n}, \tag{9}$$

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

$$\bar{o}_n = \bar{o}_{n-1} + \alpha(1 - \bar{o}_{n-1}), \text{ for } n > 0 \text{ and } \bar{o}_0 = 0. \tag{10}$$

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

First we make some observations. Initially we see that $\bar{o}_1 = \alpha$ and so $\beta_1 = 1$. And eventually by solving the recursion for the trace

$$\begin{aligned}
\bar{o}_n &= \alpha + (1-\alpha)\bar{o}_{n-1} \\
&= \alpha + (1-\alpha)(\alpha + (1-\alpha)\bar{o}_{n-2}) \\
&= \alpha + (1-\alpha)\alpha + (1-\alpha)^2\bar{o}_{n-2} \\
&\vdots \\
&= \alpha\sum_{k=0}^{n}(1-\alpha)^k + (1-\alpha)^n\bar{o}_0 \\
&= \alpha\frac{1-(1-\alpha)^{n+1}}{1-(1-\alpha)} + (1-\alpha)^n \cdot 0 \\
&= 1 - (1-\alpha)^{n+1},
\end{aligned} \tag{11}$$

we see that $\bar{o}_n \uparrow 1$ and so $\beta_n \downarrow \alpha$. Returning to (8) we have

$$Q_{n+1} = \beta_n R_n + \sum_{k=1}^{n-1}(\beta_k \prod_{i=k+1}^{n}(1-\beta_i))R_k + (\prod_{i=1}^{n}(1-\beta_i))Q_1. \tag{12}$$

We will compute each of those terms using the recursion (10). We have

$$\begin{aligned}
\prod_{i=k+1}^{n}(1-\beta_i) &= \prod_{i=k+1}^{n}\frac{1}{\bar{o}_i}(\bar{o}_i - \alpha) \\
&=(1-\alpha)^{n-k-1}\prod_{i=k+1}^{n}\frac{\bar{o}_{i-1}}{\bar{o}_i} \\
&=(1-\alpha)^{n-k-1}\frac{\bar{o}_k}{\bar{o}_n},
\end{aligned} \tag{13}$$

and

$$\beta_k\prod_{i=k+1}^{n}(1-\beta_i) = \frac{\alpha(1-\alpha)^{n-k-1}}{\bar{o}_n}. \tag{14}$$

So

$$\begin{aligned}
(12) &= \frac{\alpha}{\bar{o}_n}R_n + \sum_{k=1}^{n-1}\frac{\alpha(1-\alpha)^{n-k-1}}{\bar{o}_n}R_k + (1-\alpha)^{n-1}\frac{\bar{o}_0}{\bar{o}_n}Q_1 \\
&= \frac{\alpha}{\bar{o}_n}R_n + \sum_{k=1}^{n-1}\frac{\alpha(1-\alpha)^{n-k-1}}{\bar{o}_n}R_k + 0.
\end{aligned} \tag{15}$$

Since $\bar{o}_n \longrightarrow 1$, this is approximately

$$(15) \approx \alpha R_n + \sum_{k=1}^{n-1} \alpha(1-\alpha)^{n-k-1} R_k. \tag{16}$$

So we get exponential recency and no dependence on Q_1 i.e. no initial bias.

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

For the first ten steps we are forced to explore since $\frac{1}{N_1(a)} = +\infty$. So since there is a spike at $t = 11$, that means that this was an exploitation step that went for near or at the optimal action a_*. However, once we did the optimal action a_*, the $N_t(a_*)$ went up and so the other actions became competitive again.

In the long-run we have $N_t(a_*) \approx ct$

Exercise 2.9.

$\square$

Exercise 2.10.

$\square$

Exercise 2.11.

$\square$