EECE 5698 - ST: Reinforcement Learning

Project 1

Spring 2024 Shuhao Liu

Part A

Epsilon-greedy	Average of action value $Q(a^1)$ of 100 runs	**True action value** $Q^*(a^1)$	Average of action value $Q(a^2)$ of 100 runs	**True action value** $Q(a^2)$
ε = 0 (greedy)	1.248	5	-1.540	7
ε = 0.1	2.193	5	3.520	7
ε = 0.2	2.932	5	3.320	7
ε = 0.5 (random)	4.332	5	4.793	7

Table 1Average final Q-values for $\alpha = 1$

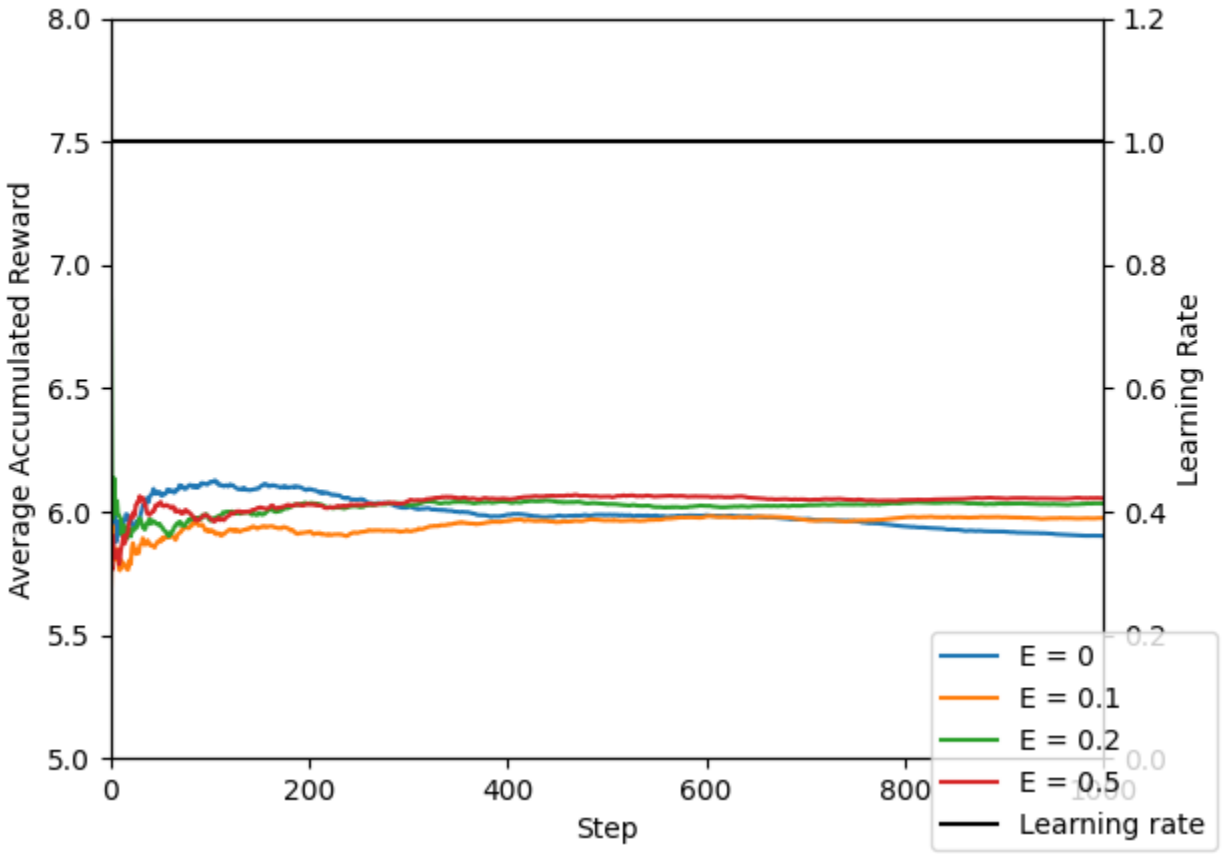

Figure 1 Average Accumulated Reward for $\alpha = 1$

Epsilon-greedy	Average of action value $Q(a^1)$ of 100 runs	**True action value** $Q^*(a^1)$	Average of action value $Q(a^2)$ of 100 runs	**True action value** $Q(a^2)$
ε = 0 (greedy)	3.114	5	3.875	7
ε = 0.1	3.478	5	4.905	7
ε = 0.2	3.821	5	5.698	7
ε = 0.5 (random)	4.623	5	6.407	7

Table 2 Average final Q-values for $\alpha = 0.9^k$

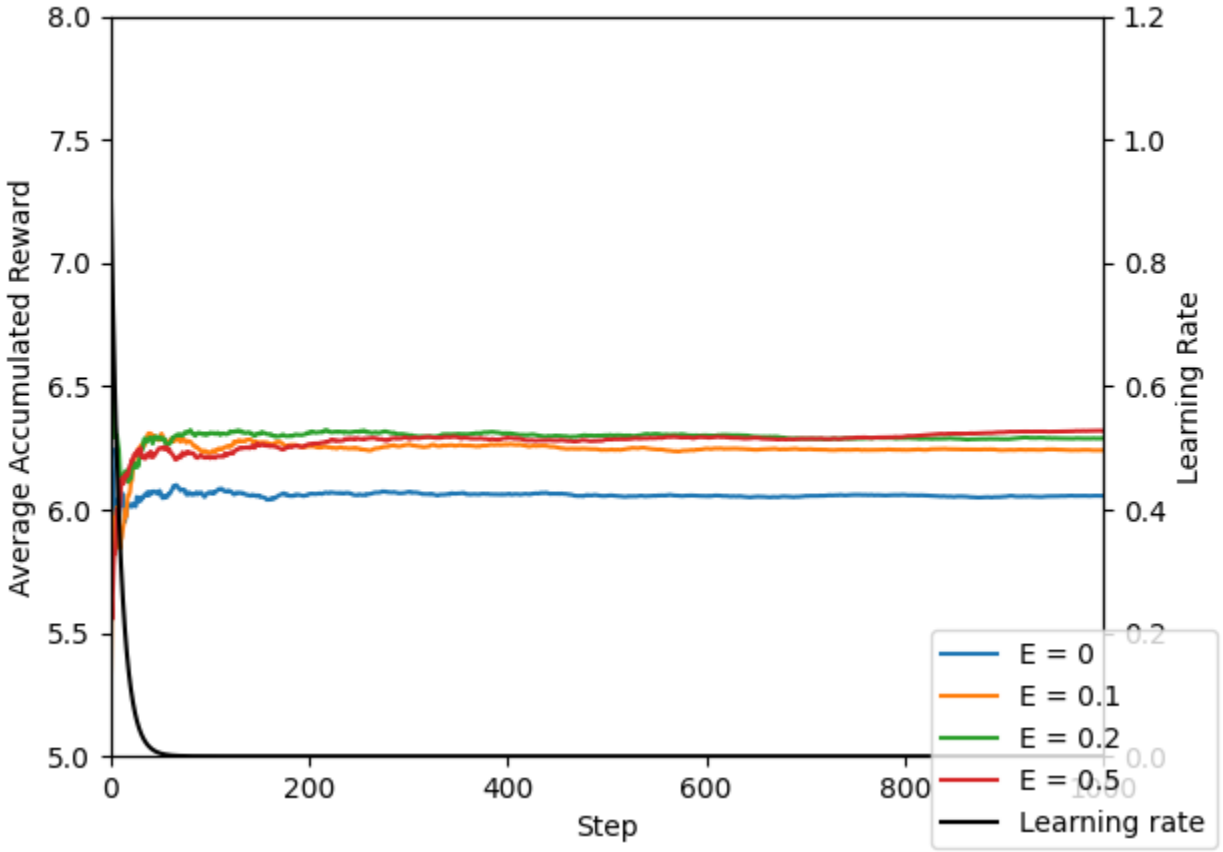

Figure 2 Average Accumulated Reward for $\alpha = 0.9^k$

Epsilon-greedy	Average of action value $Q(a^1)$ of 100 runs	**True action value** $Q^*(a^1)$	Average of action value $Q(a^2)$ of 100 runs	**True action value** $Q(a^2)$
ε = 0 (greedy)	3.334	5	5.962	7
ε = 0.1	4.477	5	6.840	7
ε = 0.2	4.722	5	6.715	7
ε = 0.5 (random)	4.854	5	6.859	7

Table 3 Average final Q-values for $\alpha = \frac{1}{1+\ln(1+k)}$

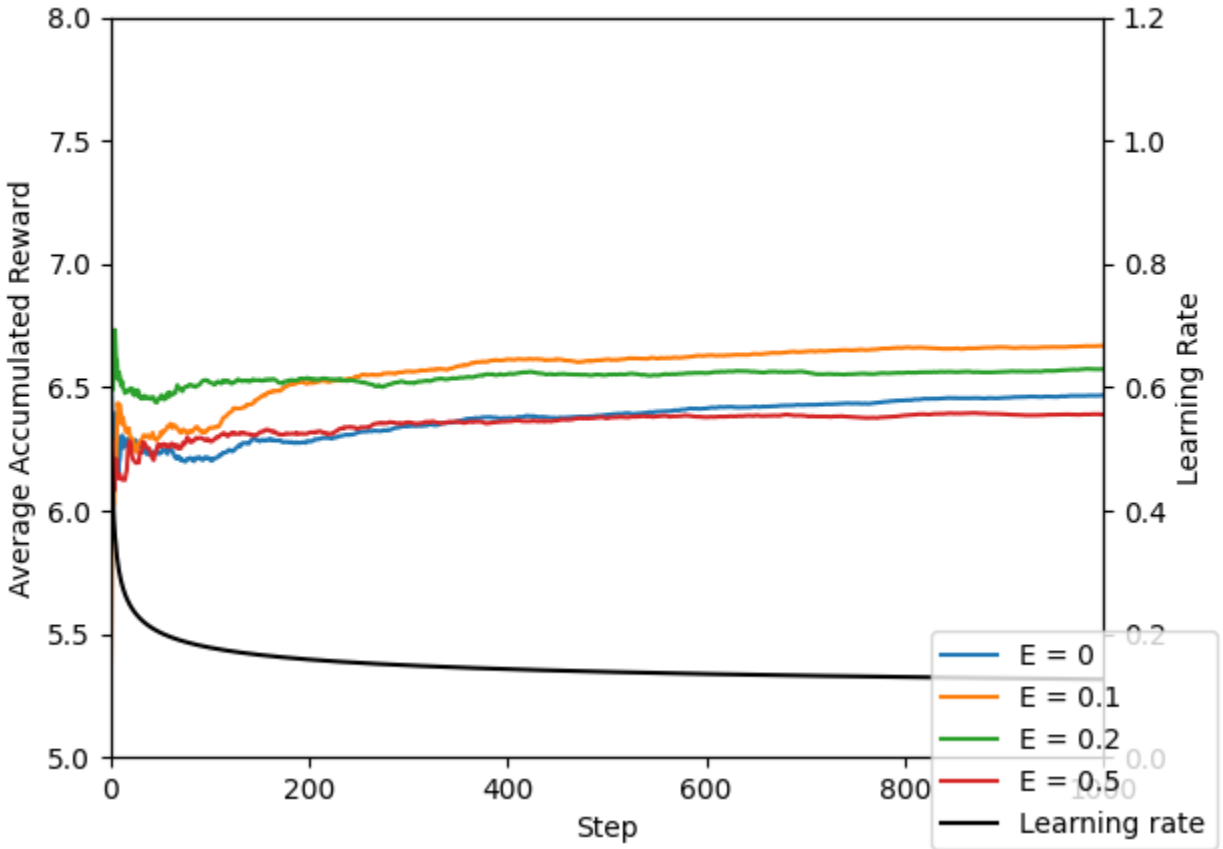

Figure 3 Average Accumulated Reward for $\alpha = \frac{1}{1+\ln(1+k)}$

Epsilon-greedy	Average of action value $Q(a^1)$ of 100 runs	**True action value** $Q^*(a^1)$	Average of action value $Q(a^2)$ of 100 runs	**True action value** $Q(a^2)$
ε = 0 (greedy)	4.619	5	5.473	7
ε = 0.1	4.515	5	5.350	7
ε = 0.2	4.487	5	6.241	7
ε = 0.5 (random)	4.812	5	6.895	7

Table 4 Average final Q-values for $\alpha = \frac{1}{k}$

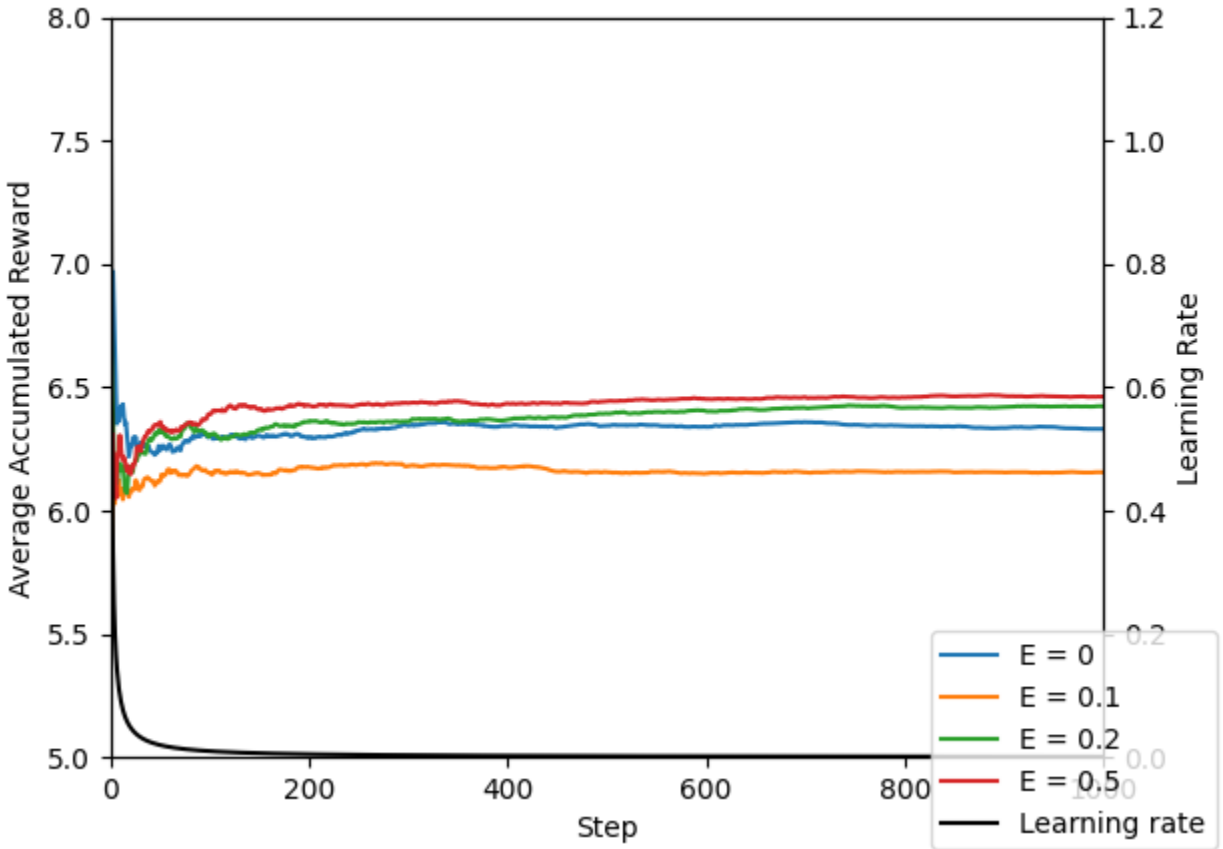

Figure 4 Average Accumulated Reward for $\alpha = \frac{1}{k}$

In this part, we assume the reward distributions are unknown, and the agent only see a realization of reward after selecting an action. We chose different learning rate $\alpha = 1, \alpha = 0.9^k$, $\alpha = \frac{1}{1+\ln(1+k)}$ and $\alpha = \frac{1}{k}$

And use the E-greedy policy of $\varepsilon = 0, 0.1, 0.2, 0.5$ to keep track of accumulated rewards by recording the average accumulated rewards.

When learning rate is always 1, the update of the Q-value for the corresponding action will always be new reward. In this case, Q value will be updated faster than other chosen learning rate. While for the other 3 learning rates, all of them will be close to 0 as the steps growing. The average accumulated rewards will be increasingly stable and this trend can be observed in all the four charts.

Comparing all the plots of the four tested learning rates, the one that seems to better follow this reasoning is

$\alpha = \frac{1}{1+\ln(1+k)}$. This makes sense, since it is the learning rate that achieves the highest average accumulated reward.

In this case, the learning rate changes from 1 to 0.126, compared with $\alpha = 0.9^k$ and $\alpha = \frac{1}{k}$, it has a higher learning rate as the steps growing from 0 to 1000. The maximum accumulated reward was achieved with the third learning rate. If we left the algorithm run for a longer time, it would achieve an even higher accumulated reward.

As for the Epsilon-Greedy policy, $\varepsilon = 0$ (greedy) always shows one of the lowest average action values. For all learning rate cases except $\alpha = \frac{1}{1+\ln(1+k)}$, the $\varepsilon = 0.5$ is the one that achieves the highest accumulated reward. We can see that those final values are closer to the true Q-values for the $\alpha = \frac{1}{1+\ln(1+k)}$ and $\alpha = \frac{1}{k}$ learning rate values tested. Considering the average accumulated reward for $\alpha = \frac{1}{1+\ln(1+k)}$, we can see that the greedy $\varepsilon = 0$ and $\varepsilon = 0.5$ curves are the ones that achieve the lower average accumulated rewards while the greedy $\varepsilon = 0.1$ shows the best, followed by the greedy $\varepsilon = 0.2$.

Part B

Epsilon-greedy	Average of action value $Q(a^1)$ of 100 runs	**True action value** $Q^*(a^1)$	Average of action value $Q(a^2)$ of 100 runs	**True action value** $Q(a^2)$
Q= [0 0]	4.577	5	6.856	7
Q= [5 7]	4.600	5	6.811	7
Q= [20 20]	4.729	5	6.762	7

Table 5 Average final Q-values for $\alpha = 0.1$ and $\varepsilon = 0.1$ for different optimistic initial Q-values

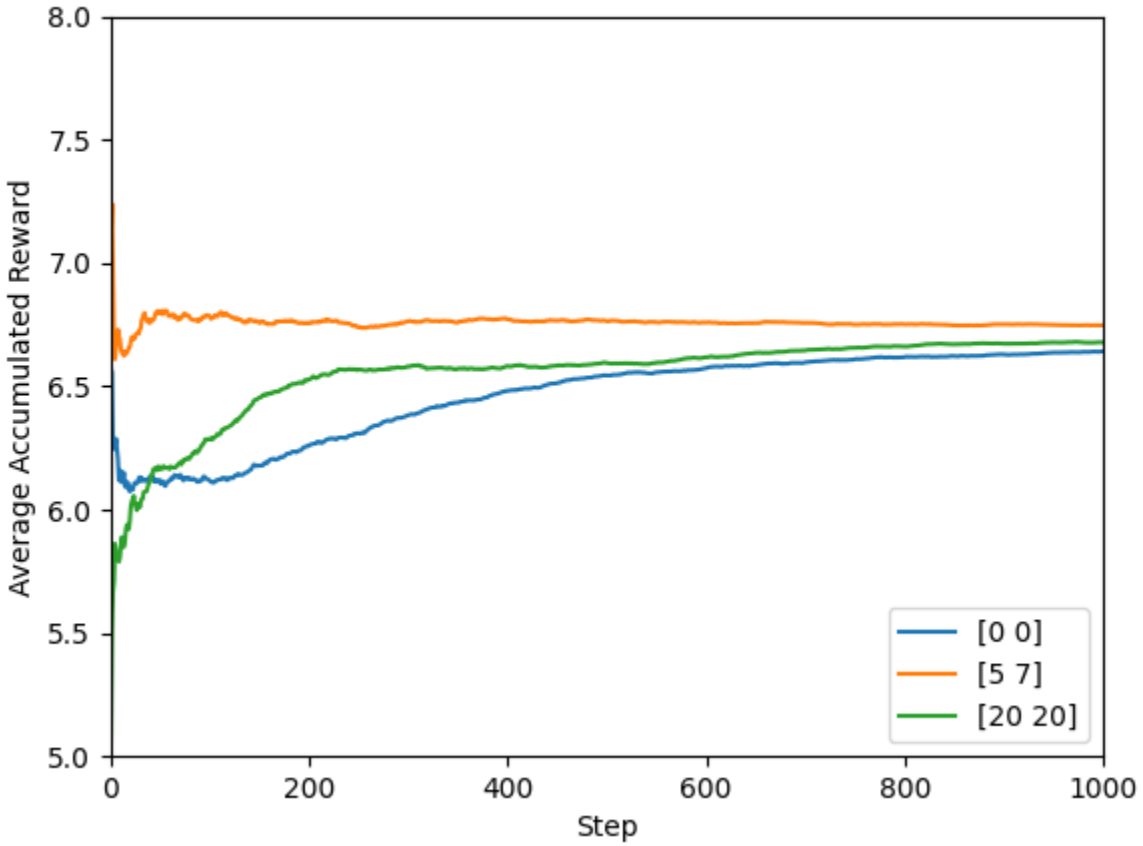

Figure 5 Average Accumulated Reward for 3 optimistic initial Q values

For a fixed a = 0.1 and e = 0.1, about 400 steps later, all the average accumulated reward come close to the true action values [5 7] as supposed.

By analyzing the figures, we can see that the best case is when the initial Q-values are equal to the true Q-values, which intuitively makes sense. This case has as a good estimation of the action values from the very beginning, and therefore doesn't require much exploration.

Part C

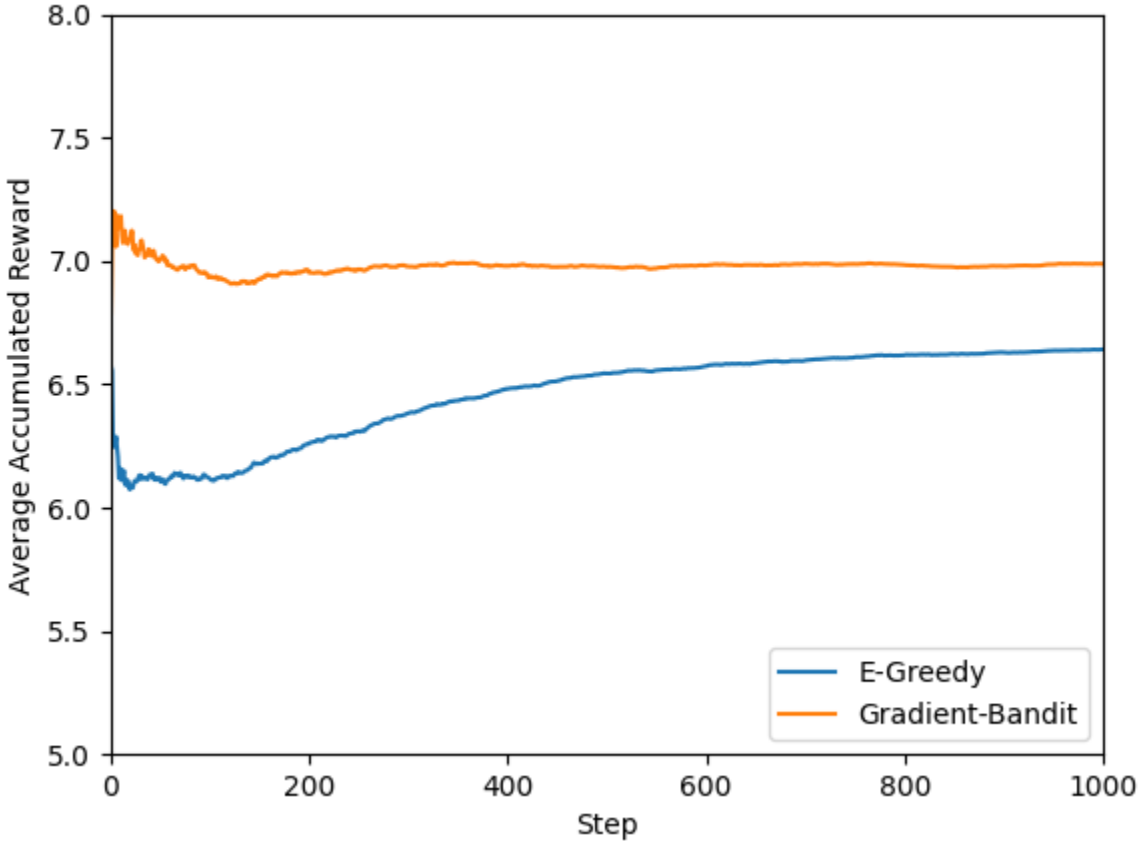

Figure 6 Comparison of Average Accumulated Reward between ε-Greedy and Gradient-Bandit policies

We can clearly see that the Gradient-Bandit case achieves higher rewards, both initially and in the long term. The Gradient-Bandit case achieved around 7 in the beginning and stay around that. This shows that Gradient-Bandit case captures the higher true action value for action 2, and has a trend for more likely to pick that option and this preference make it has a better performance among these two policies.

Code

```
"""
EECE5698 Spring 2024
Shuhao Liu
Project 1
"""

import numpy as np
from numpy import random
import matplotlib.pyplot as plt

# Initial settings
STEPS = 1000
RUNS = 100
MU1 = 5
VAR1 = 10
MU2 = 10
VAR2 = 15
MU3 = 4
VAR3 = 10
EPS = [0, 0.1, 0.2, 0.5]
Q_B = [[0, 0], [5, 7], [20, 20]]
EPS_B = 0.1
A_B = 0.1
A_C = 0.1
MIN_Y = 5
MAX_Y = 8

np.random.seed(1)


def reward_f(action):
    if action == 1:
        reward = random.normal(loc=MU1, scale=np.sqrt(VAR1))
    else:    # action 2
        if random.uniform() > 0.5:
            reward = random.normal(loc=MU2, scale=np.sqrt(VAR2))
        else:
            reward = random.normal(loc=MU3, scale=np.sqrt(VAR3))
    return reward


def e_greedy(epsilon, q1, q2):
    if (random.uniform() < epsilon) | (q1 == q2): # random
        action = round(random.uniform() + 1)
    else:    # greedy
        if q1 > q2:
            action = 1
        else:
            action = 2
```

```
    reward = reward_f(action)
#    print("Epsilon: {} - Action: {} - Reward: {}".format(epsilon, action, reward))
    return (action, reward)


def gradient(pi1):
    if (random.uniform() < pi1):
        action = 1
    else:
        action = 2
    reward = reward_f(action)
    return (action, reward)


print("----- PART A -----")


# Initialize storage matrices
R = np.zeros([RUNS, STEPS])    # rewards
AR = np.zeros([4, 4, STEPS])    # average accumulated rewards
Q1 = np.zeros(RUNS)    # Q1 for each run
Q2 = np.zeros(RUNS)    # Q2 for each run
Q1a = np.zeros([4, 4])    # average final Q1
Q2a = np.zeros([4, 4])    # average final Q2
A = np.zeros([4, STEPS])    # learning rates (alpha)
for a in range(4):
    for k in range(STEPS):
        if      a == 0: A[a][k] = 1
        elif a == 1: A[a][k] = 0.9**(k + 1)
        elif a == 2: A[a][k] = 1/(1 + np.log(1 + (k + 1)))
        elif a == 3: A[a][k] = 1/(k + 1)

for a in range(4):
    for e in range(4):
        for i in range(RUNS):
            for k in range(STEPS):
                act, r = e_greedy(EPS[e], Q1[i], Q2[i])
                if act == 1: Q1[i] = Q1[i] + A[a][k]*(r - Q1[i])
                else:                Q2[i] = Q2[i] + A[a][k]*(r - Q2[i])
                R[i][k] = r    # store each step's reward
            # convert them to accumulated rewards
            for k in reversed(range(STEPS)):
                R[i][k] = np.mean(R[i][:k+1])
        Q1a[a][e] = np.mean(Q1)
        Q2a[a][e] = np.mean(Q2)
        # average the accumulated rewards over all runs
        for k in range(STEPS):
            acc = 0
            for i in range(RUNS):
                acc = acc + R[i][k]
```

```
            AR[a][e][k] = acc/RUNS

for a in range (4):
    fig, ax1 = plt.subplots()
    for e in range(4):
        print("Alpha: {} - Epsilon: {}\t- Ave final Q1: {:.4}\t- Ave final Q2: {:.4}".format(a, EPS[e], Q1a[a][e], Q2a[a][e]))
        ax1.plot(AR[a][e][:])
    ax2 = ax1.twinx()    # instantiate a second axes that shares the same x-axis
    ax2.plot(A[a][:], 'k')
    fig.legend(["E = {}".format(EPS[0]), "E = {}".format(EPS[1]), "E = {}".format(EPS[2]), "E = {}".format(EPS[3]), "Learning rate"],
loc = "lower right")
    ax1.axis([0, STEPS, MIN_Y, MAX_Y])
    ax2.axis([0, STEPS, 0, 1.2])
    ax1.set_ylabel('Average Accumulated Reward')
    ax2.set_ylabel('Learning Rate')
    ax1.set_xlabel('Step')
    plt.show(block=False)


print("----- PART B -----")


R     = np.zeros([RUNS, STEPS])    # rewards
AR    = np.zeros([3, STEPS])    # average accumulated rewards
Q1a = np.zeros(3)    # average final Q1
Q2a = np.zeros(3)    # average final Q2

for q in range(3):
    Q1 = np.ones(RUNS)*Q_B[q][0]    # Q1 for each run
    Q2 = np.ones(RUNS)*Q_B[q][1]    # Q2 for each run
    for i in range(RUNS):
        for k in range(STEPS):
            act, r = e_greedy(EPS_B, Q1[i], Q2[i])
            if act == 1: Q1[i] = Q1[i] + A_B*(r - Q1[i])
            else:              Q2[i] = Q2[i] + A_B*(r - Q2[i])
            R[i][k] = r    # store each step's reward
        # convert them to accumulated rewards
        for k in reversed(range(STEPS)):
            R[i][k] = np.mean(R[i][:k+1])
    Q1a[q] = np.mean(Q1)
    Q2a[q] = np.mean(Q2)
    # average the accumulated rewards over all runs
    for k in range(STEPS):
        acc = 0
        for i in range(RUNS):
            acc = acc + R[i][k]
        AR[q][k] = acc/RUNS

plt.figure()
for q in range(3):
```

```
    print("Initial Q: {}\t- Ave final Q1: {:.4}\t- Ave final Q2: {:.4}".format(Q_B[q], Q1a[q], Q2a[q]))
    plt.plot(AR[q][:])
plt.legend(["[0 0]", "[5 7]", "[20 20]"], loc="lower right")
plt.axis([0, STEPS, MIN_Y, MAX_Y])
plt.ylabel('Average Accumulated Reward')
plt.xlabel('Step')
plt.show(block=False)


print("----- PART C -----")


H1 = 0
H2 = 0
Ra  = np.zeros([RUNS, STEPS])  # accumulated rewards
ARc = np.zeros([STEPS])  # average accumulated rewards

for i in range(RUNS):
    R  = np.array([])  # rewards
    for k in range(STEPS):
        pi1 = np.exp(H1)/(np.exp(H1) + np.exp(H2))
    #   pi2 = np.exp(H2)/(np.exp(H1) + np.exp(H2))
        pi2 = 1 - pi1  # faster, same result
        act, r = gradient(pi1)
        R = np.append(R, r)  # store each step's reward
        Ra[i][k] = np.mean(R)  # accumulated rewards up to current step
        if act == 1:
            H1 = H1 + A_C*(r - Ra[i][k])*(1 - pi1)
            H2 = H2 - A_C*(r - Ra[i][k])*pi2
        else:
            H2 = H2 + A_C*(r - Ra[i][k])*(1 - pi2)
            H1 = H1 - A_C*(r - Ra[i][k])*pi1

# average the accumulated rewards over all runs
for k in range(STEPS):
    acc = 0
    for i in range(RUNS):
        acc = acc + Ra[i][k]
    ARc[k] = acc/RUNS

plt.figure()
plt.plot(AR[0][:])  # [0, 0] from part B
plt.plot(ARc[:])
plt.legend(["E-Greedy", "Gradient-Bandit"], loc="lower right")
plt.axis([0, STEPS, MIN_Y, MAX_Y])
plt.ylabel('Average Accumulated Reward')
plt.xlabel('Step')
plt.show(block=False)

print("All results printed successfully")
```