Tutorial 8

● Graded

14 Hours, 25 Minutes Late

Student

Anirud N

Total Points

8 / 8 pts

Question 1

Task 1

2 / 2 pts

✓ **– 0 pts** Correct

Question 2

Task 2

2 / 2 pts

✓ **– 0 pts** Correct

– 2 pts incomplete.

Question 3

Task 3

2 / 2 pts

✓ **– 0 pts** Correct

– 1.5 pts incomplete

Question 4

Task 4

2 / 2 pts

✓ **– 0 pts** Correct

– 1.5 pts no code for T4

– 2 pts not attempted

– 1 pt incomplete either code (not showing the plot or output) or inference

– 1.5 pts incomplete code and inference missing

Tutorial 8 - Options

Please complete this tutorial to get an overview of options and an implementation of SMDP Q-Learning and Intra-Option Q-Learning.

References:

Recent Advances in Hierarchical Reinforcement Learning is a strong recommendation for topics in HRL that was covered in class. Watch Prof. Ravi's lectures on moodle or nptel for further understanding the core concepts. Contact the TAs for further resources if needed.

```
'''
A bunch of imports, you don't have to worry about these
'''

import numpy as np
from tqdm import tqdm
import random
import gym
#from gym.wrappers import Monitor
import glob
import io
import matplotlib.pyplot as plt
from IPython.display import HTML

'''
The environment used here is extremely similar to the openai gym ones.
At first glance it might look slightly different.
The usual commands we use for our experiments are added to this cell
to aid you
work using this environment.
'''

#Setting up the environment
from gym.envs.toy_text.cliffwalking import CliffWalkingEnv
env = CliffWalkingEnv()

env.reset()

#Current State
print(env.s)

# 4x12 grid = 48 states
print ("Number of states:", env.nS)

# Primitive Actions
action = ["up", "right", "down", "left"]
#correspond to [0,1,2,3] that's actually passed to the environment
```

```
# either go left, up, down or right
print ("Number of actions that an agent can take:", env.nA)

# Example Transitions
rnd_action = random.randint(0, 3)
print ("Action taken:", action[rnd_action])
next_state, reward, is_terminal, t_prob,dummy = env.step(rnd_action)
print ("Transition probability:", t_prob)
print ("Next state:", next_state)
print ("Reward recieved:", reward)
print ("Terminal state:", is_terminal)
print("dummy",dummy)
#env.render()
```

```
36
Number of states: 48
Number of actions that an agent can take: 4
Action taken: up
Transition probability: False
Next state: 24
Reward recieved: -1
Terminal state: False
dummy {'prob': 1.0}
```

Options

We custom define very simple options here. They might not be the logical options for this settings deliberately chosen to visualise the Q Table better.

```
# We are defining two more options here
# Option 1 ["Away"] - > Away from Cliff (ie keep going up)
# Option 2 ["Close"] - > Close to Cliff (ie keep going down)

def Away(env,state):

    optdone = False
    optact = 0

    if (int(state/12) == 0):
        optdone = True

    return [optact,optdone]

def Close(env,state):

    optdone = False
    optact = 2

    if (int(state/12) == 2):
```

Question assigned to the following page: 1

```
        optdone = True

    if (int(state/12) == 3):
        optdone = True

    return [optact,optdone]


'''
Now the new action space will contain
Primitive Actions: ["up", "right", "down", "left"]
Options: ["Away","Close"]
Total Actions :["up", "right", "down", "left", "Away", "Close"]
Corresponding to [0,1,2,3,4,5]
'''
```

```
'\nNow the new action space will contain\nPrimitive Actions: ["up",
"right", "down", "left"]\nOptions: ["Away","Close"]\nTotal Actions :
["up", "right", "down", "left", "Away", "Close"]\nCorresponding to
[0,1,2,3,4,5]\n'
```

Task 1

Complete the code cell below

```
#Q-Table: (States x Actions) === (env.ns(48) x total actions(6))
q_values_SMDP = np.zeros((48,6))

q_frequency_SMDP = np.zeros((48,6))

q_values_INTRA = np.zeros((48,6))

q_frequency_INTRA = np.zeros((48,6))
#Update_Frequency Data structure? Check TODO 4

# TODO: epsilon-greedy action selection function
def egreedy_policy(q_values,state,epsilon):
    if np.random.randint(0,1) < epsilon:
        action = np.random.randint(len(q_values[state]))
    else:
        action = np.argmax(q_values[state])
    return action
def egreedy_policy(q_values, state, epsilon):
    if np.random.uniform(0, 1) < epsilon:
        # Explore: choose a random action
        action = np.random.randint(len(q_values[state]))
    else:
        # Exploit: choose the action with the highest Q-value
```

Questions assigned to the following page: 1 and 2

```
        action = np.argmax(q_values[state])
    return action
```

Task 2

Below is an incomplete code cell with the flow of SMDP Q-Learning. Complete the cell and train the agent using SMDP Q-Learning algorithm. Keep the **final Q-table** and **Update Frequency** table handy (You'll need it in TODO 4)

```
#### SMDP Q-Learning

# Add parameters you might need here
gamma = 0.9
alpha  = 0.3
# Iterate over 1000 episodes
for _ in tqdm(range(1000)):
    state,dummy_var = env.reset()
    done = False

    # While episode is not over
    while not done:

        # Choose action
        action = egreedy_policy(q_values_SMDP, state, epsilon=0.1)

        # Checking if primitive action
        if action < 4:
            # Perform regular Q-Learning update for state-action pair
            next_state, reward, done, INFORM1,INFORM2 =
env.step(action)
            q_values_SMDP[state][action] += alpha * (reward + gamma *
np.max(q_values_SMDP[next_state]) - q_values_SMDP[state][action])
            q_frequency_SMDP[state][action] += 1


        # Checking if action chosen is an option
        reward_bar = 0
        if action == 4: # action => Away option

            optdone = False
            current_state = state
            timestep = 0
            while (optdone == False) and (done == False):

                # Think about what this function might do?
                optact,optdone = Away(env,state)
```

```
                next_state, reward, done,dymmu,dummmy3=
env.step(optact)
                timestep +=1
                # Is this formulation right? What is this term?
                reward_bar = gamma*reward_bar + reward
                state =  next_state

                # Complete SMDP Q-Learning Update
                # Remember SMDP Updates. When & What do you update?

                state = next_state
            q_values_SMDP[current_state][action] += alpha *
(reward_bar + gamma**timestep * np.max(q_values_SMDP[next_state]) -
q_values_SMDP[current_state][action])
            q_frequency_SMDP[current_state][action] += 1

        if action == 5: # action => Close option
            optdone = False

            current_state = state

            timestep = 0

            while optdone == False and done == False:

                optact, optdone = Close(env, state)

                next_state, reward, done, dummy, dymmy4 =
env.step(optact)

                timestep+=1


                reward_bar = gamma * reward_bar + reward

                state = next_state

            q_values_SMDP[current_state][action] += alpha *
(reward_bar + gamma**timestep * np.max(q_values_SMDP[next_state]) -
q_values_SMDP[current_state][action])
            q_frequency_SMDP[current_state][action] += 1

        state = next_state
```

```
  5%|▌         | 48/1000 [00:00<00:01, 479.33it/s]

100%|██████████| 1000/1000 [00:00<00:00, 2187.63it/s]
```

Question assigned to the following page: 3

Task 3

Using the same options and the SMDP code, implement Intra Option Q-Learning (In the code cell below). You *might not* always have to search through options to find the options with similar policies, think about it. Keep the **final Q-table** and **Update Frequency** table handy (You'll need it in TODO 4)

```
gamma = 0.9
alpha = 0.3



for _ in tqdm(range(1000)):
    state,_ = env.reset()
    done = False

    while not done:

        action = egreedy_policy(q_values_INTRA, state, epsilon = 0.1)


        if action ==0:
            next_state, reward, done, dummy1, dummy2 =
env.step(action)
            q_values_INTRA[state][action] += alpha * (reward + gamma *
np.max(q_values_INTRA[next_state]) - q_values_INTRA[state][action])

            q_frequency_INTRA[state][action] += 1
            optact,optdone=Away(env,state)

            if not optdone:
                q_values_INTRA[state][4] += alpha * (reward + gamma *
np.max(q_values_INTRA[next_state]) - q_values_INTRA[state][4])

                q_frequency_INTRA[state][4] += 1
            state=next_state

        if action ==2:
            next_state, reward, done, dummy1, dummy2 =
env.step(action)

            q_values_INTRA[state][action] += alpha * (reward + gamma *
np.max(q_values_INTRA[next_state]) - q_values_INTRA[state][action])

            q_frequency_INTRA[state][action] += 1

            optact,optdone=Close(env,state)
```

```
            if not optdone:
                q_values_INTRA[state][5] += alpha * (reward + gamma *
np.max(q_values_INTRA[next_state]) - q_values_INTRA[state][5])

                q_frequency_INTRA[state][5] += 1
            state=next_state

        if action ==1 or action==3:
            next_state, reward, done, dummy1, dummy2 =
env.step(action)

            q_values_INTRA[state][action] += alpha * (reward + gamma *
np.max(q_values_INTRA[next_state]) - q_values_INTRA[state][action])

            q_frequency_INTRA[state][action] += 1

            state=next_state


        if action == 4:
            optdone = False
            while not optdone:

                next_state, reward, done, dummy1, dummy2 =
env.step(optact)
                optact, optdone = Away(env,next_state)
                if optdone:
                    q_values_INTRA[state][4] += alpha * (reward +
gamma * np.max(q_values_INTRA[next_state]) - q_values_INTRA[state][4])

                    q_frequency_INTRA[state][4] += 1
                else:
                    q_values_INTRA[state][4] += alpha * (reward +
gamma * q_values_INTRA[next_state][4] - q_values_INTRA[state][4])

                    q_frequency_INTRA[state][4] += 1

                q_values_INTRA[state][optact] += alpha * (reward +
gamma * np.max(q_values_INTRA[next_state]) - q_values_INTRA[state]
[optact])

                q_frequency_INTRA[state][optact] += 1

                state = next_state

        if action == 5:
            optdone = False
            while not optdone:
                next_state, reward, done, dummy, dummy1 =
```

Questions assigned to the following page: 3 and 4

```
env.step(optact)

                optact, optdone = Close(env,next_state)

                if optdone:

                    q_values_INTRA[state][5] += alpha * (reward + 
gamma * np.max(q_values_INTRA[next_state]) - q_values_INTRA[state][5])
                    q_frequency_INTRA[state][5] += 1

                else:

                    q_values_INTRA[state][5] += alpha * (reward + 
gamma * q_values_INTRA[next_state][5] - q_values_INTRA[state][5])
                    q_frequency_INTRA[state][5] += 1


                q_values_INTRA[state][optact] += alpha * (reward + 
gamma * np.max(q_values_INTRA[next_state]) - q_values_INTRA[state]
[optact])
                q_frequency_INTRA[state][optact] += 1
                state = next_state
```

```
  0%|          | 0/1000 [00:00<?, ?it/s]

100%|██████████| 1000/1000 [00:00<00:00, 2571.29it/s]
```

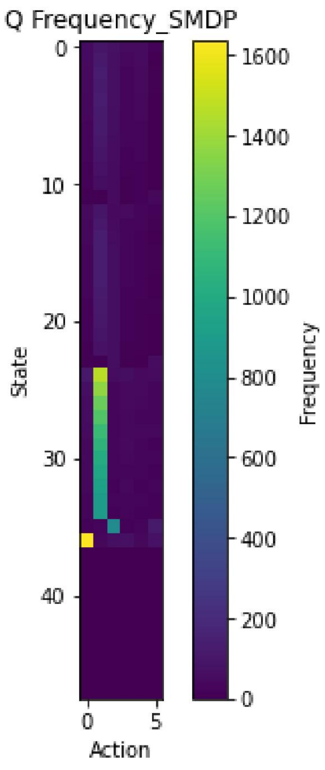
Task 4

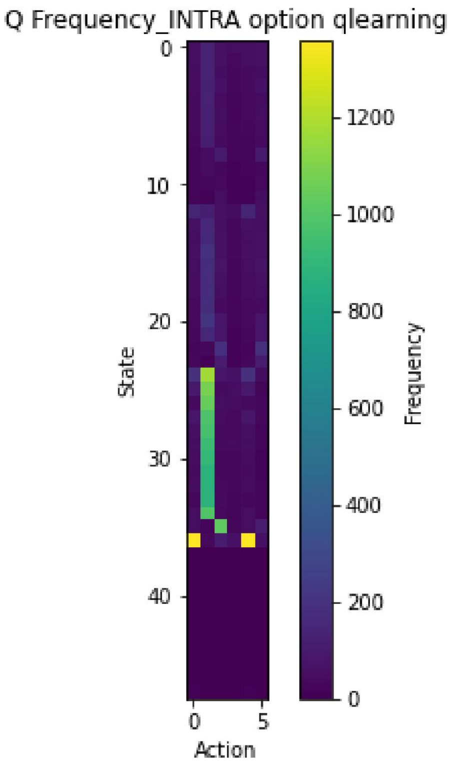
Compare the two Q-Tables and Update Frequencies and provide comments.

```
# Use this cell for Task 4 Code
fig, axs = plt.subplots(1, 2, figsize=(12, 6))


im1 = axs[0].imshow(q_frequency_SMDP)
axs[0].set_title('Q Frequency_SMDP')
axs[0].set_xlabel('Action')
axs[0].set_ylabel('State')
fig.colorbar(im1, ax=axs[0], label='Frequency')


im2 = axs[1].imshow(q_frequency_INTRA)
axs[1].set_title('Q Frequency_INTRA option qlearning')
axs[1].set_xlabel('Action')
axs[1].set_ylabel('State')
fig.colorbar(im2, ax=axs[1], label='Frequency')
```

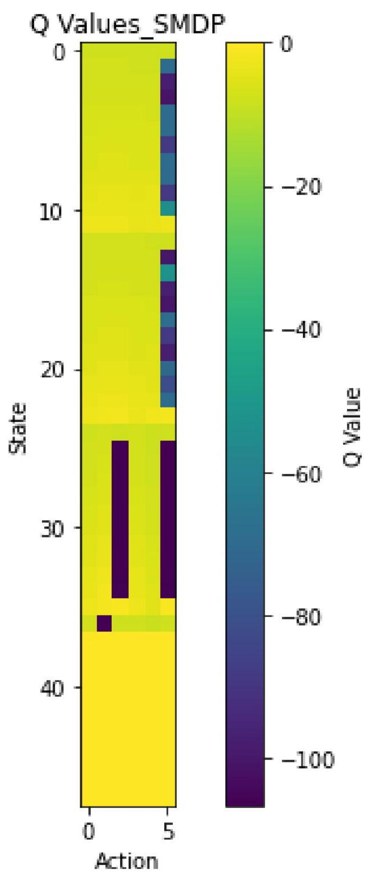
Question assigned to the following page: 4

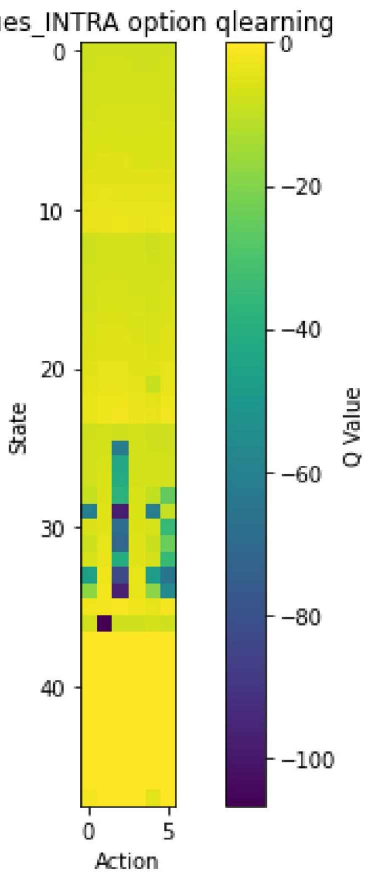
```
fig2, axs2 = plt.subplots(1, 2, figsize=(12, 6))

im3 = axs2[0].imshow(q_values_SMDP)
axs2[0].set_title('Q Values_SMDP')
axs2[0].set_xlabel('Action')
axs2[0].set_ylabel('State')
fig2.colorbar(im3, ax=axs2[0], label='Q Value')


im4 = axs2[1].imshow(q_values_INTRA)
axs2[1].set_title('Q Values_INTRA option qlearning')
axs2[1].set_xlabel('Action')
axs2[1].set_ylabel('State')
fig2.colorbar(im4, ax=axs2[1], label='Q Value')

plt.tight_layout()
plt.show()
```





Question assigned to the following page: 4





```
sum(sum(q_frequency_INTRA))

26011.0

sum(sum(q_frequency_SMDP))

22641.0
```

INFERENCE:

In Intra option q leaening ; At every step, the state-action value for the primitive action as well as the state-action value for all options that would have selected the same action are updated, regardless of the option in effect. The agent can move through several states in a single option execution in IOQL. Consequently, compared to primitive actions alone, states visited during the execution of an option will have higher visit frequency. Even though the number of episodes is the same, but in intra option case more updates are occurring for every state, allowing it to learn from more experiences.