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
from IPython.display import clear_output
import time
```

```
Warning: Gym version v0.24.1 has a number of critical issues with
`gym.make` such that environment observation and action spaces are
incorrectly evaluated, raising incorrect errors and warning . It is
recommend to downgrading to v0.23.1 or upgrading to v0.25.1
```

```
'''
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 = env.step(rnd_action)
print ("Transition probability:", t_prob)
print ("Next state:", next_state)
print ("Reward recieved:", reward)
print ("Terminal state:", is_terminal)
env.render()
```

```
36
Number of states: 48
Number of actions that an agent can take: 4
Action taken: down
Transition probability: {'prob': 1.0}
Next state: 36
Reward recieved: -1
Terminal state: False
o  o  o  o  o  o  o  o  o  o  o  o
o  o  o  o  o  o  o  o  o  o  o  o
o  o  o  o  o  o  o  o  o  o  o  o
x  C  C  C  C  C  C  C  C  C  C  T
```

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):
```

```python
        optdone = True

    return [optact,optdone]

def Close(env,state):

    optdone = False
    optact = 2

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

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

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

# TODO: epsilon-greedy action selection function
def egreedy_policy(q_values,state,epsilon):

    if np.random.uniform(0, 1) < epsilon:
        return np.random.randint(0, 6)
    else:
        return np.argmax(q_values[state])
```

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 = 0.1


# Iterate over 1000 episodes
for i in tqdm(range(1000)):
    state = env.reset()
    done = False

    # While episode is not over
    while not done:
        # redering the environment for the last 5 episodes
        if i > 995:
            clear_output(wait=True)
            print("Episode: ", i)
            env.render()
            time.sleep(0.2)

        # 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, _ = env.step(action)
            q_values_SMDP[state, action] = q_values_SMDP[state, 
action] + alpha*(reward + gamma*np.max(q_values_SMDP[next_state]) - 
q_values_SMDP[state, action])
            update_freq_SMDP[state, action] += 1
            state = next_state

        reward_bar = 0
        tau = 0
        state_init = state

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

            optdone = False
            while (optdone == False):
```

```
                # Think about what this function might do?
                optact,optdone = Away(env,state)
                next_state, reward, done,_ = env.step(optact)

                # Is this formulation right? What is this term?
                # reward_bar = gamma*reward_bar + reward # this
formulation is incorrect
                # this is the correct formulation
                reward_bar = reward_bar + (gamma**tau)*reward
                tau += 1

                # Complete SMDP Q-Learning Update
                # Remember SMDP Updates. When & What do you update?
                # we update the q_values_SMDP[state_init, action]
after the option is complete
                state = next_state

            q_values_SMDP[state_init, action] =
q_values_SMDP[state_init, action] + alpha*(reward_bar +
(gamma**tau)*np.max(q_values_SMDP[state]) - q_values_SMDP[state_init,
action])
            update_freq_SMDP[state_init, action] += 1

        reward_bar = 0
        tau = 0
        state_init = state

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

            optdone = False
            while (optdone == False):

                optact,optdone = Close(env,state)
                next_state, reward, done,_ = env.step(optact)

                reward_bar = reward_bar + (gamma**tau)*reward
                tau += 1

                state = next_state

            q_values_SMDP[state_init, action] =
q_values_SMDP[state_init, action] + alpha*(reward_bar +
(gamma**tau)*np.max(q_values_SMDP[state]) - q_values_SMDP[state_init,
action])
            update_freq_SMDP[state_init, action] += 1
```

```
Episode:  999
o  o  o  o  o  o  o  o  o  o  o  o
o  o  o  o  o  o  o  o  o  o  o  o
```

```
o  o  o  o  o  o  o  o  o  o  o  x
o  C  C  C  C  C  C  C  C  C  C  T

100%|██████████| 1000/1000 [00:16<00:00, 60.83it/s]
```

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)

```python
#### Intra-Option Q-Learning

q_values_intra = np.zeros((48,6))
update_freq_intra = np.zeros((48,6))

# parameters
gamma = 0.9
alpha = 0.1

# for 1000 episodes
for i in tqdm(range(1000)):
    state = env.reset()
    done = False

    while not done:
        # redering the environment for the last 5 episodes
        if i > 995:
            clear_output(wait=True)
            print("Episode: ", i)
            env.render()
            time.sleep(0.2)

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

        if action < 4:
            next_state, reward, done, _ = env.step(action)
            # update the state action pair for primitive actions
            q_values_intra[state, action] = q_values_intra[state, action] + alpha*(reward + gamma*np.max(q_values_intra[next_state]) - q_values_intra[state, action])
            update_freq_intra[state, action] += 1
            # update the state option pair
            # we need to consider only the option that corresponds to the primitive action
```

```
            # Away option only for up action
            if action == 0:
                optact, optdone = Away(env, next_state)
                if optdone == False:
                    q_values_intra[state, 4] = q_values_intra[state,
4] + alpha*(reward + (gamma*q_values_intra[next_state, 4]) -
q_values_intra[state, 4])
                else:
                    q_values_intra[state, 4] = q_values_intra[state,
4] + alpha*(reward + gamma*np.max(q_values_intra[next_state]) -
q_values_intra[state, 4])
                update_freq_intra[state, 4] += 1

            # Close option only for down action
            if action == 2:
                optact, optdone = Close(env, next_state)
                if optdone == False:
                    q_values_intra[state, 5] = q_values_intra[state,
5] + alpha*(reward + (gamma*q_values_intra[next_state, 5]) -
q_values_intra[state, 5])
                else:
                    q_values_intra[state, 5] = q_values_intra[state,
5] + alpha*(reward + gamma*np.max(q_values_intra[next_state]) -
q_values_intra[state, 5])
                update_freq_intra[state, 5] += 1

            state = next_state

        if action == 4:
            optdone = False
            while optdone == False:
                optact, optdone = Away(env, state)
                next_state, reward, done, _ = env.step(optact)

                # update the state action pair for the primitive
action
                q_values_intra[state, optact] = q_values_intra[state,
optact] + alpha*(reward + gamma*np.max(q_values_intra[next_state]) -
q_values_intra[state, optact])
                update_freq_intra[state, optact] += 1

                _, optdone_next = Away(env, next_state)

                # we need not look for options that correspond to the
action performed by Away option
                # because the other option Close does not have any
common primitive actions with Away
                if optdone_next == False:
                    q_values_intra[state, 4] = q_values_intra[state,
```

```
4] + alpha*(reward + (gamma*q_values_intra[next_state, 4]) - 
q_values_intra[state, 4])
                else:
                    q_values_intra[state, 4] = q_values_intra[state, 
4] + alpha*(reward + gamma*np.max(q_values_intra[next_state]) - 
q_values_intra[state, 4])

                update_freq_intra[state, 4] += 1

                state = next_state

        if action == 5:
            optdone = False
            while optdone == False:
                optact, optdone = Close(env, state)
                next_state, reward, done, _ = env.step(optact)

                # update the state action pair for the primitive 
action
                q_values_intra[state, optact] = q_values_intra[state, 
optact] + alpha*(reward + gamma*np.max(q_values_intra[next_state]) - 
q_values_intra[state, optact])
                update_freq_intra[state, optact] += 1

                _, optdone_next = Close(env, next_state)

                # we need not look for options that correspond to the 
action performed by Close option
                # because the other option Away does not have any 
common primitive actions with Close
                if optdone_next == False:
                    q_values_intra[state, 5] = q_values_intra[state, 
5] + alpha*(reward + (gamma*q_values_intra[next_state, 5]) - 
q_values_intra[state, 5])
                else:
                    q_values_intra[state, 5] = q_values_intra[state, 
5] + alpha*(reward + gamma*np.max(q_values_intra[next_state]) - 
q_values_intra[state, 5])

                update_freq_intra[state, 5] += 1

                state = next_state
```

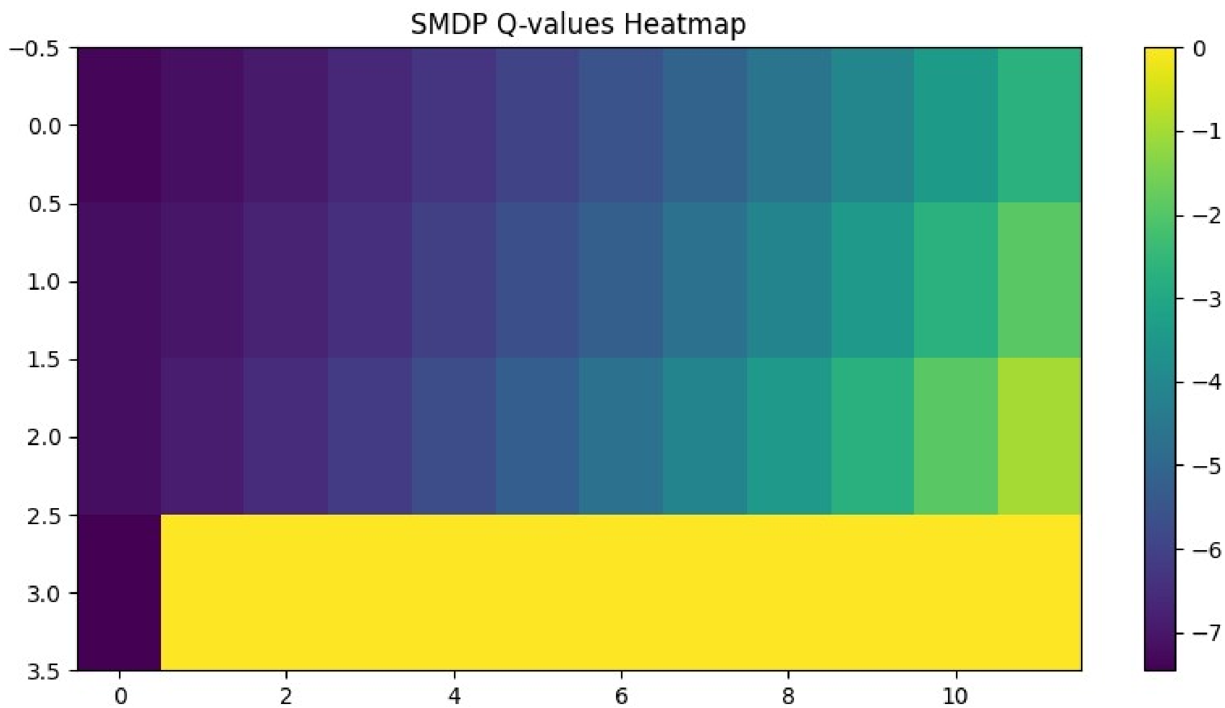
```
 100%|██████████| 1000/1000 [00:13<00:00, 39.84it/s]

Episode:  999
o  o  o  o  o  o  o  o  o  o  o  o
o  o  o  o  o  o  o  o  o  o  o  o
o  o  o  o  o  o  o  o  o  o  o  x
```

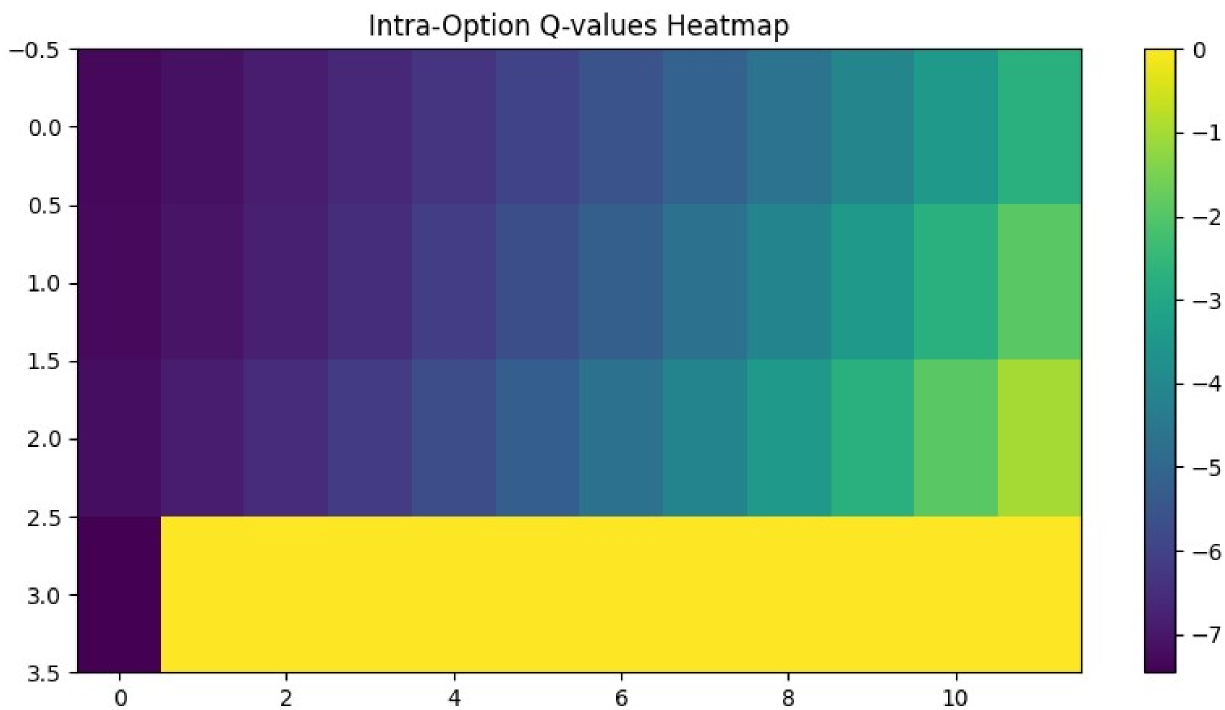
```
o  C  C  C  C  C  C  C  C  C  C  T


100%|██████████| 1000/1000 [00:13<00:00, 72.14it/s]
```

Task 4

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

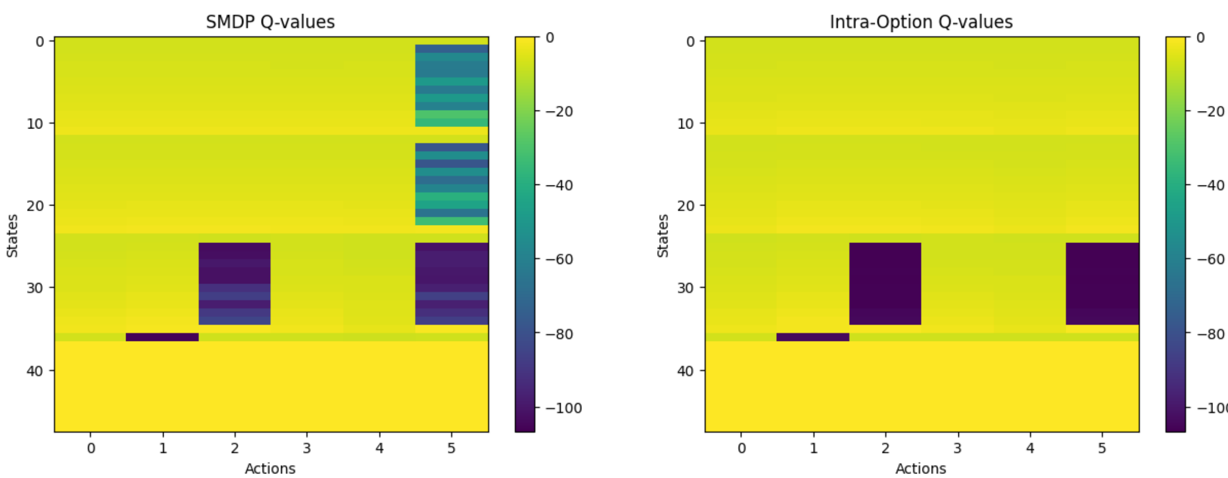
```python
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize

# plot of max q-values for SMDP and intra-option learning

q_values_SMDP_max = np.max(q_values_SMDP, axis=1).reshape(4, 12)
q_values_intra_max = np.max(q_values_intra, axis=1).reshape(4, 12)

plt.figure(figsize=(10, 5))
plt.imshow(q_values_SMDP_max, cmap='viridis', aspect='auto')
plt.title('SMDP Q-values Heatmap')
plt.colorbar()
plt.show()

plt.figure(figsize=(10, 5))
plt.imshow(q_values_intra_max, cmap='viridis', aspect='auto')
plt.title('Intra-Option Q-values Heatmap')
plt.colorbar()
plt.show()
```





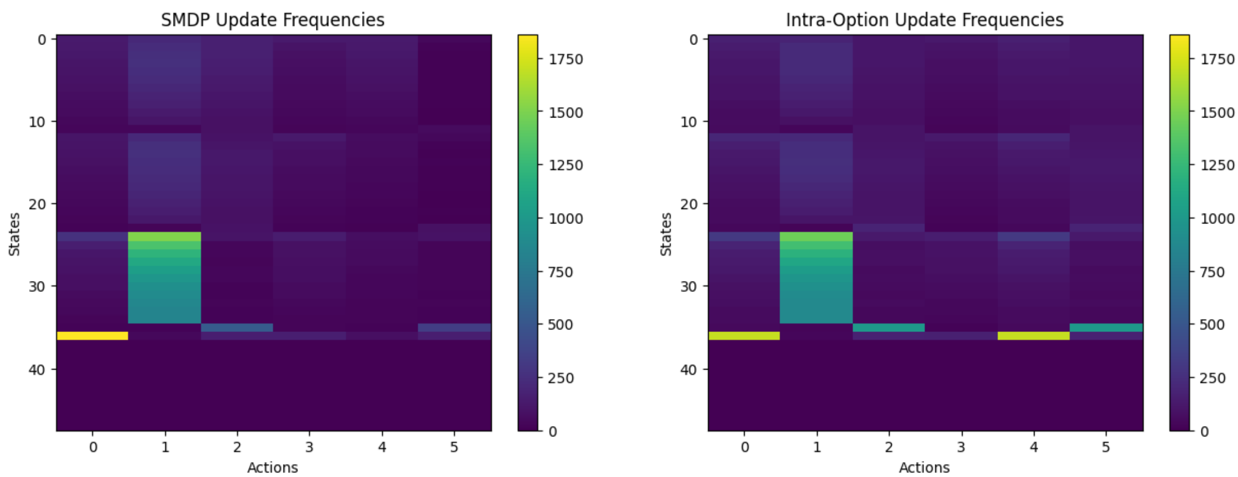

```
fig, axs = plt.subplots(1, 2, figsize=(15, 5))

# plot of the SMDP Q-values heatmap
cmap = 'viridis'
norm = Normalize(vmin=np.min([q_values_SMDP.min(),
```

```
q_values_intra.min()]),
                 vmax=np.max([q_values_SMDP.max(),
q_values_intra.max()]))
im1 = axs[0].imshow(q_values_SMDP, cmap=cmap, aspect='auto',
norm=norm)
axs[0].set_title('SMDP Q-values')
axs[0].set_xlabel('Actions')
axs[0].set_ylabel('States')
axs[0].grid(False)
fig.colorbar(im1, ax=axs[0])

# plot of the Intra-Option Q-values heatmap
im2 = axs[1].imshow(q_values_intra, cmap=cmap, aspect='auto',
norm=norm)
axs[1].set_title('Intra-Option Q-values')
axs[1].set_xlabel('Actions')
axs[1].set_ylabel('States')
axs[1].grid(False)
fig.colorbar(im2, ax=axs[1])

plt.show()
```



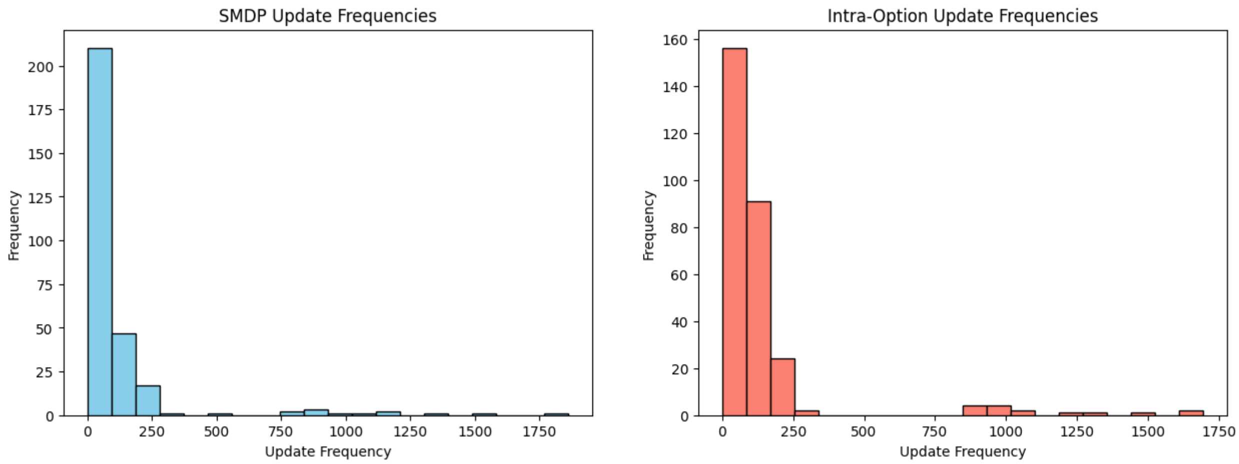

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize

fig, axs = plt.subplots(1, 2, figsize=(15, 5))

# plot of the SMDP Update Frequencies heatmap
cmap = 'viridis'
norm = Normalize(vmin=np.min([update_freq_SMDP.min(),
update_freq_intra.min()]),
                 vmax=np.max([update_freq_SMDP.max(),
update_freq_intra.max()]))
```

```
im1 = axs[0].imshow(update_freq_SMDP, cmap=cmap, aspect='auto',
norm=norm)
axs[0].set_title('SMDP Update Frequencies')
axs[0].set_xlabel('Actions')
axs[0].set_ylabel('States')
axs[0].grid(False)
fig.colorbar(im1, ax=axs[0])

# plot of the Intra-Option Update Frequencies heatmap
im2 = axs[1].imshow(update_freq_intra, cmap=cmap, aspect='auto',
norm=norm)
axs[1].set_title('Intra-Option Update Frequencies')
axs[1].set_xlabel('Actions')
axs[1].set_ylabel('States')
axs[1].grid(False)
fig.colorbar(im2, ax=axs[1])

plt.show()
```




```
fig, axs = plt.subplots(1, 2, figsize=(15, 5))

update_freq_SMDP_flat = [freq for state_freqs in update_freq_SMDP for
freq in state_freqs]
update_freq_intra_flat = [freq for state_freqs in update_freq_intra
for freq in state_freqs]

# plot of the SMDP Update Frequencies
axs[0].hist(update_freq_SMDP_flat, bins=20, color='skyblue',
edgecolor='black')
axs[0].set_title('SMDP Update Frequencies')
axs[0].set_xlabel('Update Frequency')
axs[0].set_ylabel('Frequency')

# plot of the Intra-Option Update Frequencies
```

```
axs[1].hist(update_freq_intra_flat, bins=20, color='salmon',
edgecolor='black')
axs[1].set_title('Intra-Option Update Frequencies')
axs[1].set_xlabel('Update Frequency')
axs[1].set_ylabel('Frequency')

plt.show()
```



Inferences

- From observing the max Q value heatmap for both algorithms, we observe that both of them converge to the same policy. This implies the Q values for both the algorithms converge approximately to the same values. And on observing the rendering of the environment of the final steps of training of both algorithms, we can see that the agent learns to walk on the edge of the cliff to reach the terminal state. This is in general the case with Q learning algorithms.
- From the Q value heatmap of state-action pair, we can infer that Intra-Option Q learning results in higher negative values for the actions 'down' and 'Close' for the states just above the cliff. This also the case with SMDP Q learning, but we see some negative values of Q values for actions 'down and 'Close' for some states that are not just above the cliff. This can imply more exploration was perfomed by Intra-Option Q learning algorithm.
- From observing the heatmap of Update Frequencies, we observe that Intra-Option Q learning explores more than SMDP Q learning (as expected).
- From observing the histogram of the Update Frequencies, we observe a slightly wider range for the Intra-Option Q learning algorithm, which implies a more exploration in this case when compared to SMDP Q learning. We also observe that state-action pairs are updated more frequently with Intra-Option Q learning, since we update primitive actions and options all at once for every step in contrary to SMDP Q learning.