#Tutorial 5 - Options Intro

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.

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

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

```python
'''
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: up
Transition probability: False
Next state: 24
Reward recieved: -1
Terminal state: False
```

```
/usr/local/lib/python3.9/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

### *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
```

```python
    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]
'''
```

{"type":"string"}

## 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))
q_values_IQL = np.zeros((48,6))

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

# TODO: epsilon-greedy action selection function
def egreedy_policy(q_values,state,epsilon):
    if(np.random.uniform() <epsilon):
        action = np.random.choice(np.arange(5))
    else:
        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 = 0.4

reward_sum = 0
steps = 0

# Iterate over 1000 episodes
for _ in range(1000):
    state = 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:
            next_state, reward, done, _, _ = env.step(action)
            reward_sum += reward
            steps += 1

            q_values_SMDP[state, action] += alpha*(reward +
gamma*max(q_values_SMDP[next_state, :]) - q_values_SMDP[state,
action])
            updates_SMDP[state, action] += 1

            state = next_state

        # Checking if action chosen is an option
        reward_bar = 0
        tau = 0
        start_state = state

        if action == 4: # action => Away option

            optdone = False
            while (optdone == False and not done):

                # Think about what this function might do?
                # So, optact is 0 and represents up. The Away function
instructs us given a state on what to do when executing the Option and
when the Option should end.
                optact, optdone = Away(env, state)
```

```
                next_state, reward, done, _, _ = env.step(optact)
                reward_sum += reward
                steps += 1

                # Is this formulation right? What is this term?
                # the discounted sum of all the steps taken in an
option is reward_bar
                reward_bar = gamma*reward_bar + reward

                # Complete SMDP Q-Learning Update
                # Remember SMDP Updates. When & What do you update?
                # Updating the value of option only in the initiated
state
                tau += 1
                state = next_state

            q_values_SMDP[start_state, action] += alpha*(reward_bar +
gamma**(tau)*max(q_values_SMDP[next_state, :]) -
q_values_SMDP[start_state, action])
            updates_SMDP[start_state, action] += 1

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

            optdone = False

            while (optdone == False and not done):

                optact, optdone = Close(env, state)
                next_state, reward, done, _, _ = env.step(optact)
                reward_sum += reward
                steps += 1

                reward_bar = gamma*reward_bar + reward

                tau += 1
                state = next_state

            q_values_SMDP[start_state, action] += alpha*(reward_bar +
gamma**(tau)*max(q_values_SMDP[next_state, :]) -
q_values_SMDP[start_state, action])
            updates_SMDP[start_state, action] += 1

updates_SMDP=updates_SMDP/steps
print('Average Rewards = ', reward_sum/1000)
```

```
Average Rewards =  -79.97
```

## 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)

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

gamma = 0.9
alpha = 0.4
away_action = 4
close_action = 5
reward_sum = 0
steps = 0

for _ in range(1000):
    state = env.reset()
    done = False
    optdone = True


    while not done:

        if optdone == True:
          ep_action = egreedy_policy(q_values_IQL, state, epsilon=0.1)

        if ep_action < 4:
          action = ep_action
        elif ep_action == away_action:
          action, optdone = Away(env, state)
        elif ep_action == close_action:
          action, optdone = Close(env, state)

        next_state, reward, done, _, _ = env.step(action)
        steps += 1
        reward_sum += reward

        ##Increment the primal action Q-values
        q_values_IQL[state, action] += alpha*(reward +
gamma*max(q_values_IQL[next_state, :]) - q_values_IQL[state, action])
        updates_IQL[state, action] += 1

        action_away, away_terminal = Away(env, state)
        action_close, close_terminal = Close(env, state)

        ##Increment the option Q-values
        #In the non terminal case
        if action_away == action and away_terminal == True:
```

```
            q_values_IQL[state, away_action] += alpha*(reward +
gamma*max(q_values_IQL[next_state, :]) - q_values_IQL[state,
away_action])
            updates_IQL[state, away_action] += 1

        elif action_close == action and close_terminal == True:
            q_values_IQL[state, close_action] += alpha*(reward +
gamma*max(q_values_IQL[next_state, :]) - q_values_IQL[state,
close_action])
            updates_IQL[state, close_action] += 1

        #In the terminal case
        elif action_away == action and away_terminal == False:
            q_values_IQL[state, away_action] += alpha*(reward +
gamma*q_values_IQL[next_state, away_action] - q_values_IQL[state,
away_action])
            updates_IQL[state, away_action] += 1

        elif action_close == action and close_terminal == False:
            q_values_IQL[state, close_action] += alpha*(reward +
gamma*q_values_IQL[next_state, close_action] - q_values_IQL[state,
close_action])
            updates_IQL[state, close_action] += 1

        state = next_state

updates_IQL = updates_IQL/steps
print('Average Rewards = ', reward_sum/1000)
```

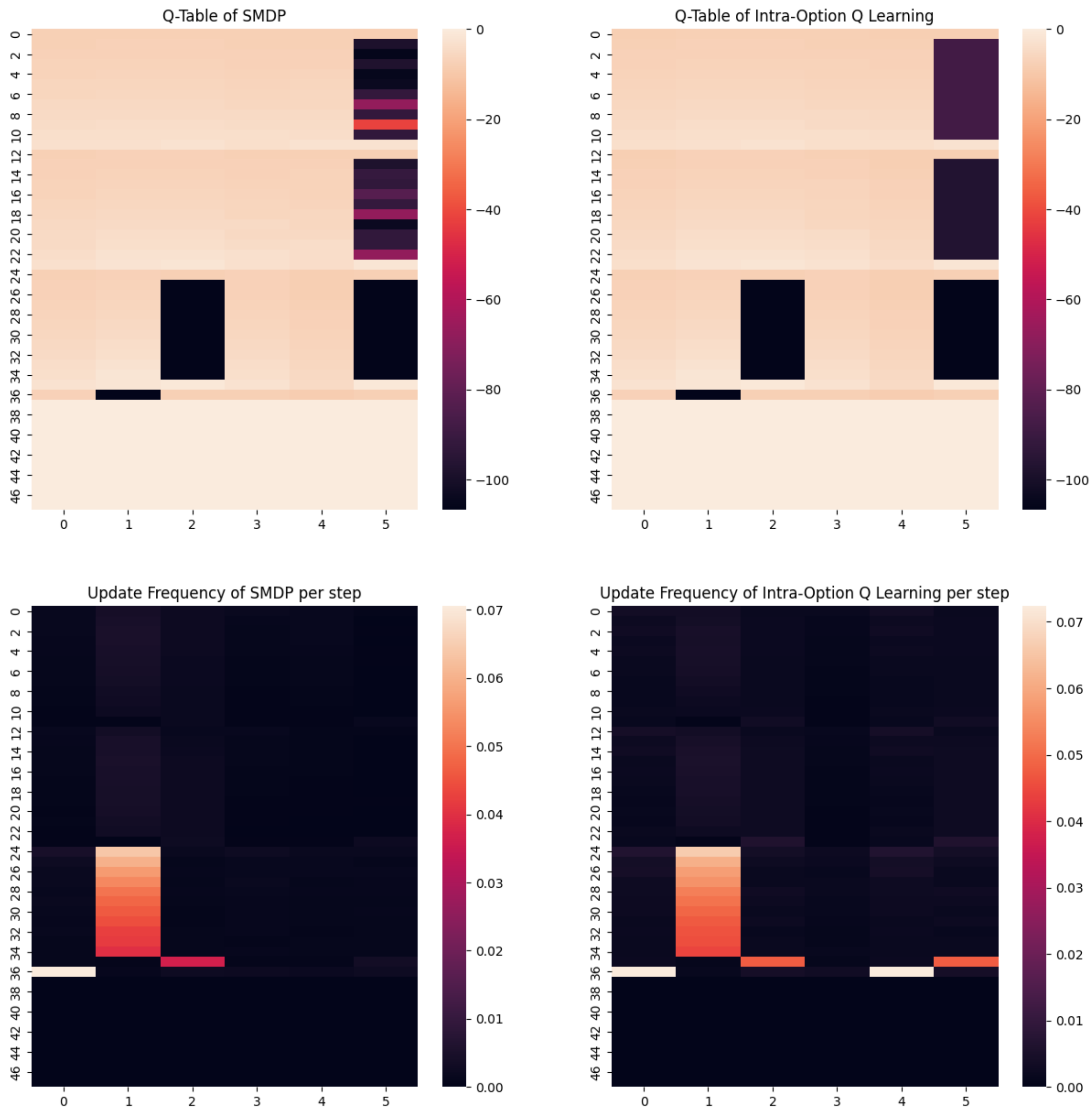
```
Average Rewards =  -68.071
```

## Task 4

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

```
# Use this cell for Task 4 Code

plt.figure(figsize = (15,15))

plt.subplot(2,2,1)
plt.title('Q-Table of SMDP')
sns.heatmap(q_values_SMDP);

plt.subplot(2,2,2)
plt.title('Q-Table of Intra-Option Q Learning')
sns.heatmap(q_values_IQL);

plt.subplot(2,2,3)
plt.title('Update Frequency of SMDP per step')
```

```
sns.heatmap(updates_SMDP);

plt.subplot(2,2,4)
plt.title('Update Frequency of Intra-Option Q Learning per step')
sns.heatmap(updates_IQL);
```




- It is evident that low q-values in the Q-table indicate suboptimal policies, such as turning right at the outset, taking a downward step in the first row, or moving closer in any column other than the first and last ones, which result in extremely low Q-values. Nevertheless, in contrast to the first two actions listed above, the "close" action is not as well-defined in SMDP as it is in IQL, and it should be almost entirely avoided like the other two.
- We measure the update frequency by dividing the number of updates by the total number of steps taken. A higher frequency, as seen in IQL, indicates multiple

learning steps. The first two actions mentioned earlier are updated more frequently than the "close" action. Hence, we did not completely eliminate the "close" action.

- Our increased options for changes enable us to make nearly three times as many modifications in IQL compared to SMDP, resulting in roughly twice as many updates in IQL.