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.

In [1]:

```
'''
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
```

In [2]:

```
'''
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 a
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: left
Transition probability: {'prob': 1.0}
Next state: 36
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.

In [3]:

```
# 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 or  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]
'''
```

Out[3]:

```
'\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

In [4]:

```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
updates_smdp = np.zeros((48,6))

# TODO: epsilon-greedy action selection function
def egreedy_policy(q_values,state,epsilon):
  rand = np.random.uniform(1,0)
  if rand < epsilon:
    return np.random.choice(range(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)

In [5]:

```python
#### SMDP Q-Learning

# Add parameters you might need here
gamma = 0.9
alpha = 0.4


# 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:
            # Perform regular Q-Learning update for state-action pair
            next_state, reward, done,_, _ = env.step(action)

            q_values_SMDP[state][action] += alpha*(reward + gamma*(q_valu
                                                     - q_values_SMDP[stat
            updates_smdp[state][action] += 1
```

```
            state = next_state

        # Checking if action chosen is an option
        reward_bar = 0
        counter = 0

        if action == 4: # action => Away option
            init_state = state
            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? No, cumul
                # r_bar = r_t+1 + Y*r_t+2 + ..... + Y^tau-1*r_t+tau
                reward_bar += reward*(gamma**counter)
                counter +=1
                # Complete SMDP Q-Learning Update
                # Remember SMDP Updates. When & What do you update?

                state = next_state

            q_values_SMDP[init_state][4] += alpha*(reward_bar + gamma*(q_
                                                  - q_values_SMDP[init
            updates_smdp[init_state][4] += 1

        if action == 5: # action => Close option
            init_state = state
            optdone = False
            while (optdone == False):

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

                # Is this formulation right? What is this term?
                reward_bar += reward*(gamma**counter)
                counter +=1

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

                state = next_state

            q_values_SMDP[init_state][5] += alpha*(reward_bar + gamma*(q_
                                                  - q_values_SMDP[init
            updates_smdp[init_state][5] += 1
```

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)

In [42]:
```
q_values = np.zeros((48,6))
updates = np.zeros((48,6))
```

In [ ]:
```
#### Intra-Option Q-Learning

# Add parameters you might need here
gamma = 0.9
alpha = 0.4


# Iterate over 1000 episodes
for i in range(1000):
    state = env.reset()
    done = False
    print(i)
    # While episode is not over
    while not done:

        # Choose action
        action = egreedy_policy(q_values, 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[state,action] += alpha*(reward + gamma*(np.max(q_val
            updates[state][action] += 1
            state = next_state

            if action == 0:
              q_values[state,4] += alpha*(reward + gamma*(np.max(q_values
              updates[state][4] += 1

            if action == 2:
              q_values[state,5] += alpha*(reward + gamma*(np.max(q_values
              updates[state][5] += 1


        # 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)

                q_values[state,optact] += alpha*(reward + gamma*(np.max(q
                updates[state,optact] += 1


                if optdone:
                  q_values[state,4] += alpha*(reward + gamma*(np.max(q_va
                  updates[state][4] += 1
                else:
                  q_values[state][4] += alpha*(reward + gamma*(q_values[n
                  updates[state][4] += 1

                if done:
```

```
                break

              state = next_state



        if action == 5: # action => Close option
            optdone = False
            while (optdone == False):

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

              q_values[state,optact] += alpha*(reward + gamma*(np.max(q
              updates[state,optact] += 1

              if optdone:
                q_values[state][5] += alpha*(reward + gamma*(np.max(q_v
                updates[state][5] += 1
              else:
                q_values[state][5] += alpha*(reward + gamma*(q_values[n
                updates[state][5] += 1

              state = next_state

              if done:
                break
```

In [ ]:

Task 4

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

In [31]:

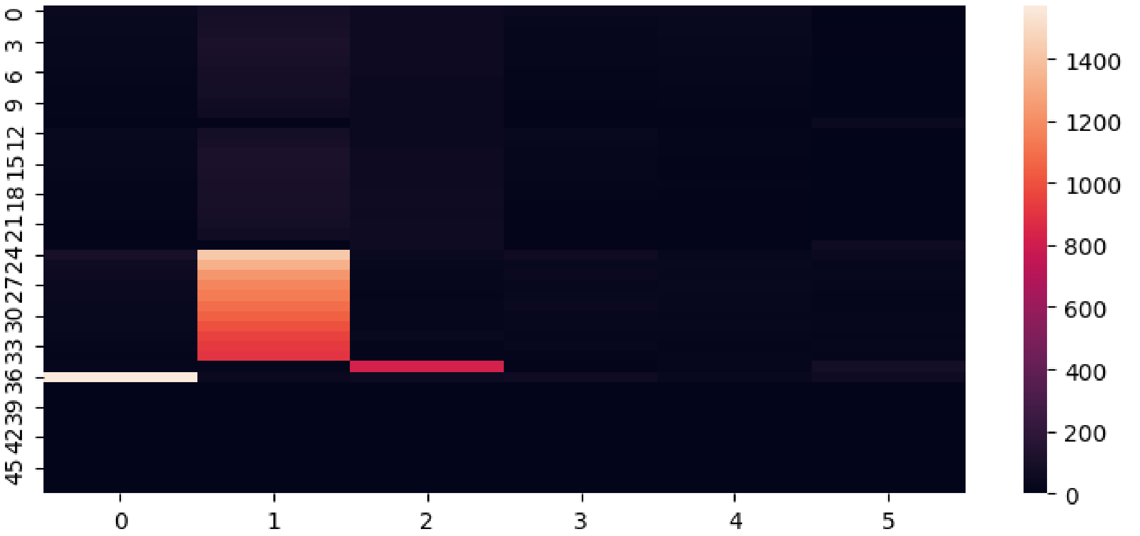
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.heatmap(updates_smdp)
```

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

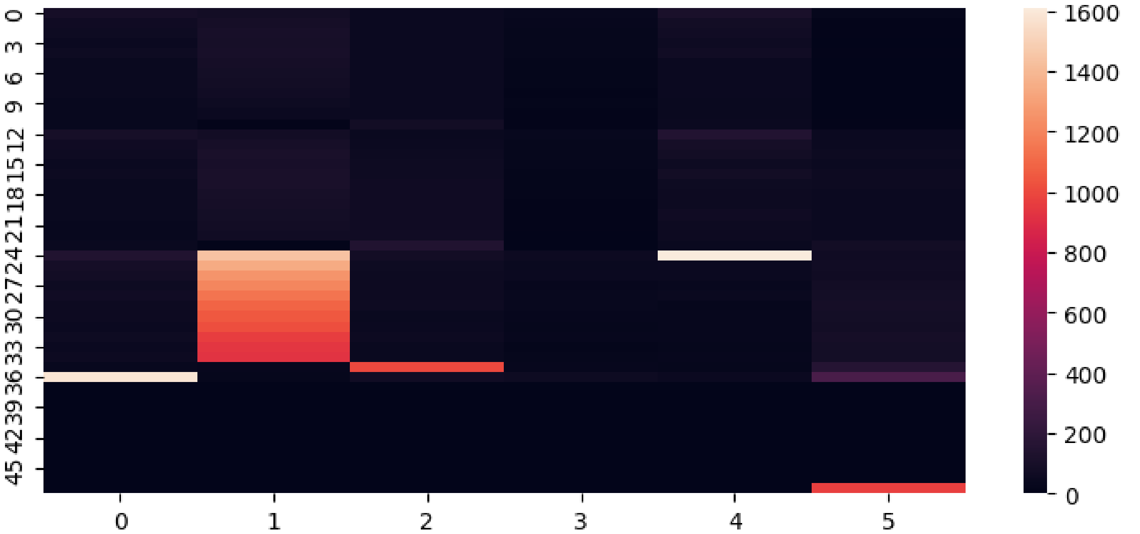
Out[31]:

```
<Axes: >
```



In [44]:
```
sns.heatmap(updates_Intra)
```

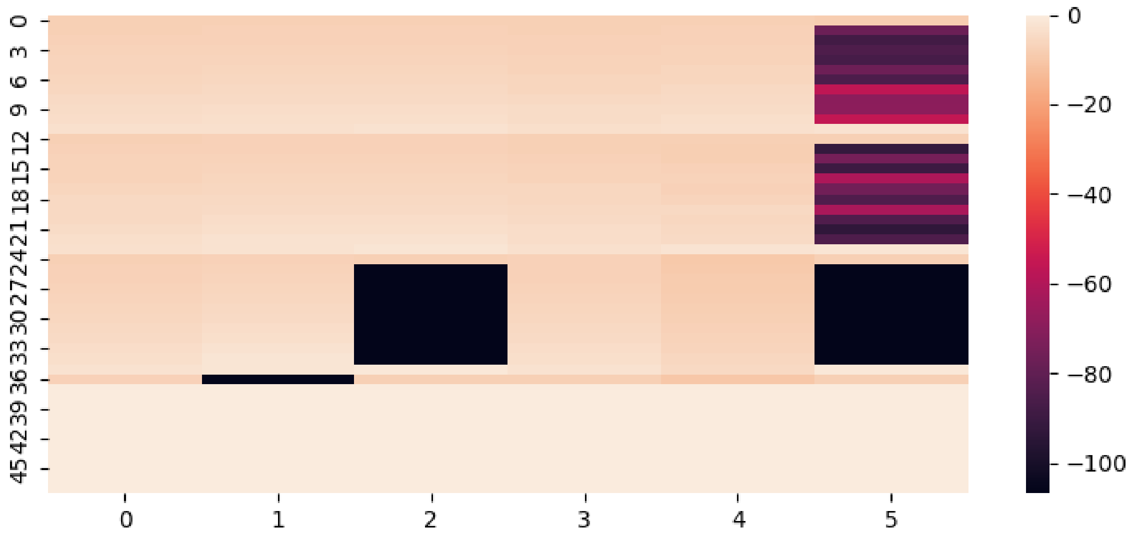
Out[44]:
```
<Axes: >
```



In [39]:
```
sns.heatmap(q_values_SMDP)
```

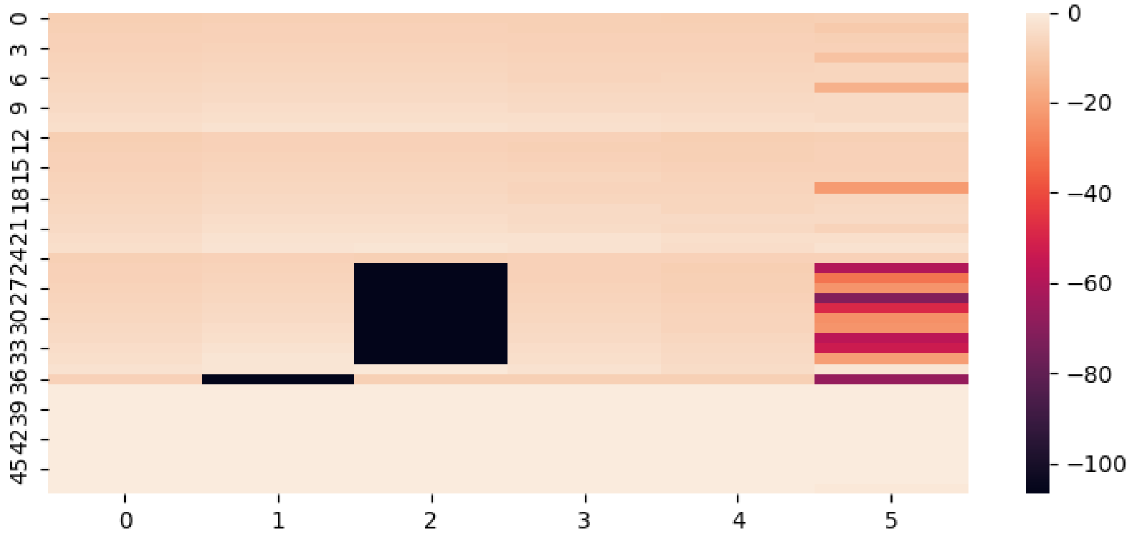
```
/usr/local/lib/python3.9/dist-packages/ipykernel/ipkernel.py:283: Depreca
tionWarning: `should_run_async` will not call `transform_cell` automatica
lly in the future. Please pass the result to `transformed_cell` argument
and any exception that happen during thetransform in `preprocessing_exc_t
uple`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

Out[39]:
```
<Axes: >
```




```
In [45]: sns.heatmap(q_values_Intra)
```

Out[45]: <Axes: >



Use this text cell for your comments - Task 4

For SMDP we update the q-values after the execution of option with a cumulative reward. We continuously take action over some time steps within the option before updating q-value for that stat-action pair.

In Intra-Option Q-Learning we update q-values for all primitive actions as well as options at each timestep. Due to this the update frequency for Intra-Option Q-Learning is higher than SMDP. But this also depends on the number of times options are choosen over primitve actions.

As for Q-Values since in SMDP they are updated with a cumulative reward after a certain number of time steps, they seem to have a negative value since we get a reward of -1 for each time step and the agent tries to maximize cumulative reward over the subtask or the option. While for Intra-Option learning Q-values seem to be much closer to zero as the agent learns to maximize reward for each time step.

In [ ]:

```
!jupyter nbconvert --to html "/content/drive/MyDrive/Colab Notebooks/CS67
```