Exercise 3

Amr Alkhashab - 905833
ELEC-E8125 - Reinforcement Learning

October 5, 2020

1 Task 1

Two functions were created for this task. First the Qlearning (The Cartpole code is presented in qleaning.py):

```
1  def update_q_value(old_state, action, new_state, reward, done,
       q_array):
2      # TODO: Implement Q-value update
3      if done == False: #for non terninating next state
4          old_cell_index = get_cell_index(old_state)
5          new_cell_index = get_cell_index(new_state)
6          q_old = q_array[old_cell_index[0], old_cell_index[1],
               old_cell_index[2], old_cell_index[3], action]
7          q_max = np.max(q_array[new_cell_index[0],
               new_cell_index[1], new_cell_index[2], new_cell_index
               [3], :])
8          q_old = q_old + alpha * (reward + (gamma * q_max) -
               q_old)
9          q_array[old_cell_index[0], old_cell_index[1],
               old_cell_index[2], old_cell_index[3], action] =
               q_old
10     else: #for terninating next state we set next state state-
           action value to zero.
11         old_cell_index = get_cell_index(old_state)
12         q_old = q_array[old_cell_index[0], old_cell_index[1],
               old_cell_index[2], old_cell_index[3], action]
13         q_old = q_old + alpha * (reward - q_old)
14         q_array[old_cell_index[0], old_cell_index[1],
               old_cell_index[2], old_cell_index[3], action] =
               q_old
```

The second function is for action selection based on epsilon:

```
def get_action(state, q_values, greedy=False):
    #TODO: Implement epsilon-greedy
    cell_index = get_cell_index(state)
    greedyaction = -1
    min = float('-inf')

    #Value a is determined by round((target_eps * 20000)/(1 -
       target_eps)), while K is increminted inside the episode
       loop.
    #epsilon = a / (a + K)
    #epsilon = 0

    # Find greedy action
    for i in range(num_of_actions):
        if q_grid[cell_index[0], cell_index[1], cell_index[2],
           cell_index[3], i] > min:
            min = q_grid[cell_index[0], cell_index[1],
               cell_index[2], cell_index[3], i]
            greedyaction = i

    #select between all action with probability epsilon
    if np.random.random() < epsilon:
        action = np.random.randint(0, num_of_actions)
    else:
        action = greedyaction

    return int(action)
```

The result of constant 0.2:

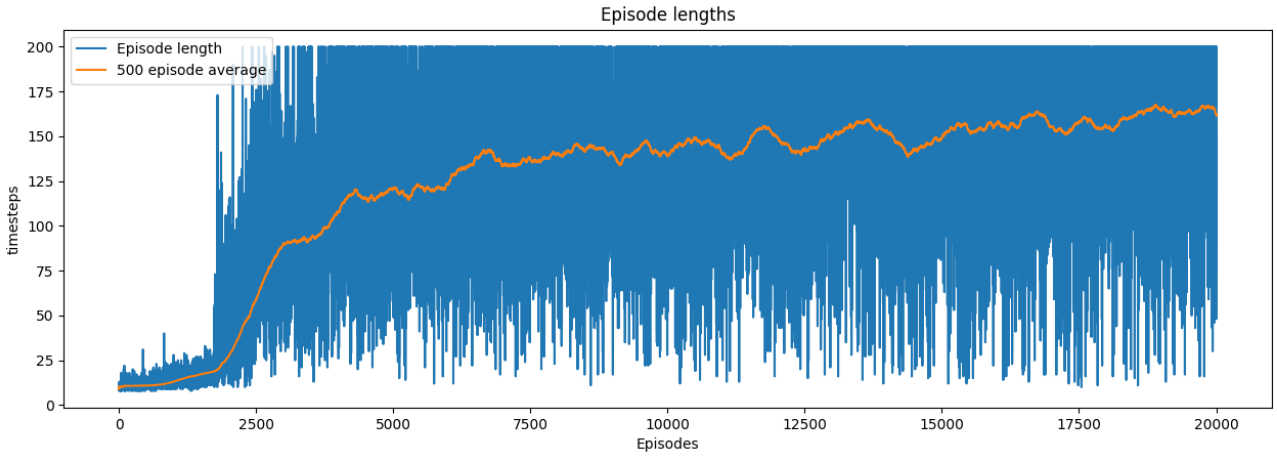

Figure 1: Training Cartpole with constant Epsilon 0.2

The result of constant GLIE:

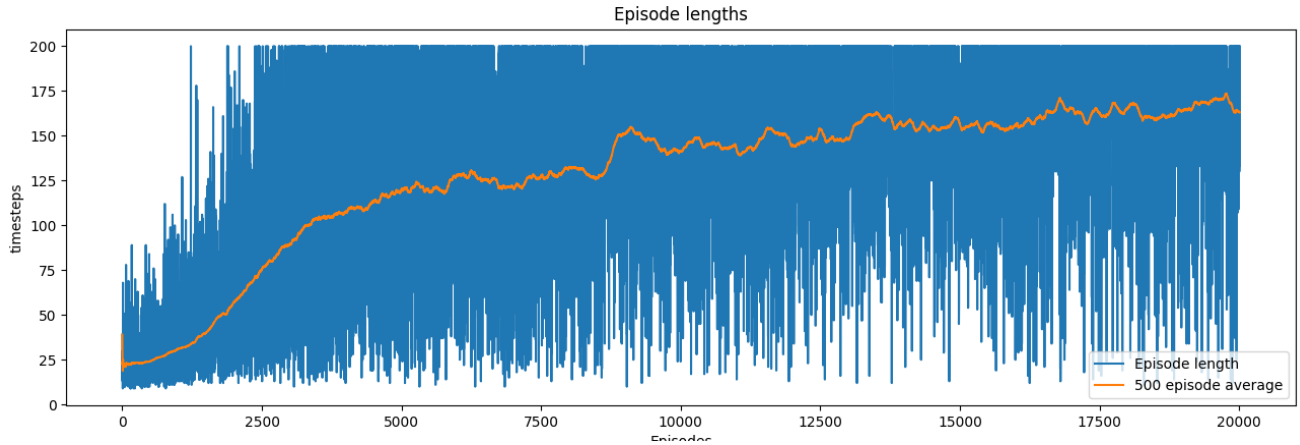

Figure 2: Training Cartpole with GLIE

2 Task 2

The Optimal value function can be easily determine by taking the maximum of state-actions value for each state.

```
values = np.max(q_grid, axis=4)
```

For the heatmap a function is created to draw the x and theta, by taking avearge over velocity and angular velocity:

```
#hearmap drawing
def heatmap(q_grid):
    value= np.zeros(q_grid.shape[:-1])  # TODO: COMPUTE THE
       VALUE FUNCTION FROM THE Q-GRID
    #value_function
    value = np.max(q_grid, axis=4)
    values_array = np.zeros(q_grid.shape[:-3])
#values_array is 2d array with mean values over velocity and
   angualr velocity for each position and theta
    values_array = np.mean(np.mean(value, axis = 3), axis = 1)
    ax = sns.heatmap(values_array)
    plt.xlabel("X")
    plt.ylabel("Theta")
    plt.show()
```

The heatmap after the training looks like this:

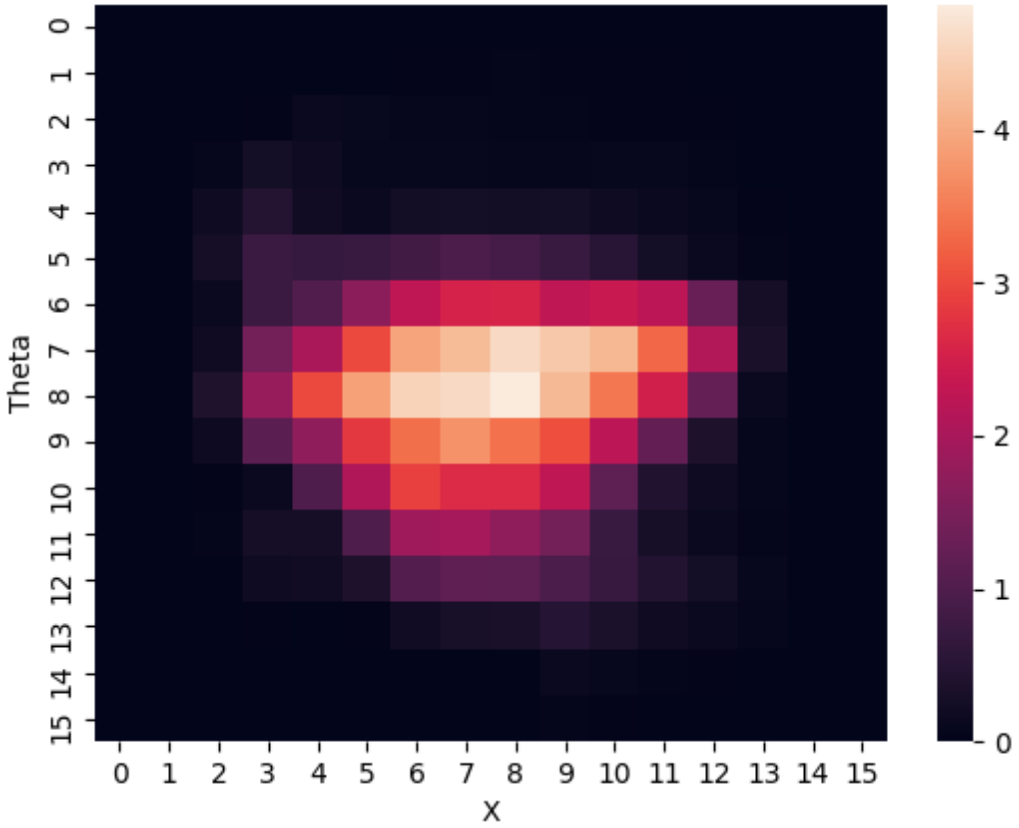

Figure 3: Heatmap after training

3 Question 1

The three plot will be summited with the files, since attaching to report is not required.

3.1 (a)

Before training, the heatmap will all be zeros,since no training has taken place and the values will be the intial intialized values (zeros).

3.2 (b)

After single episode, the area of change is still very small from observation it is between X of grid (7 and 12) and theta grid of around (7 to 9), it is still far away from Figure. 3. The value around the center will change very slightly (very small change) toward the optimal value, due to small step size of 0.1 and the small covearage area is because the agent still didnot explore most of the other states yet and thus the avearge is nearly zero.

3.3 (c)

Halfway through, It will look more like Figure. 3, the agent was able to explore more states and due to multiple episode the values converged more and more toward the true value, as they have updated more than once, and more states has been visited.

4 Task 3

(a)- Running the training for intial 0 and episolon equals 0: (b)- Running the training for

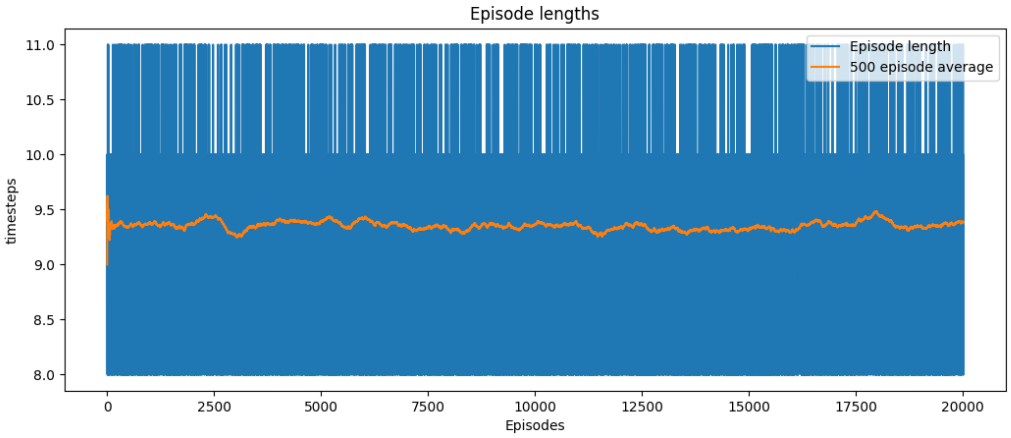

Figure 4: Training for Episolon = 0 and intial Q = 0

intial 50 and episolon equals 0:

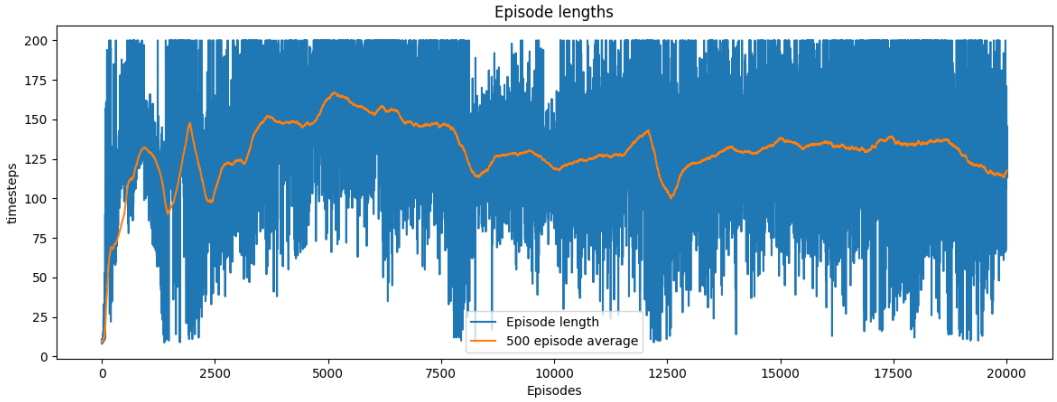

Figure 5: Training for Episolon = 0 and intial Q = 50

5 Question 2

5.1 Question 2.1

The model perform better in the second case, where the intial estimate is set to 50.

5.2 Question 2.2

For the first case, since all values are zero which is lower than the optimal value for each state, no exploration will take place and the only greedy action will be selected. The method of the second case is called Optimistic Initial Values. Since the true optimal state-value function is less than the intial value, every time an action is selected the estimate decreases, Thus, all state-action value pair are selected atleast once and tried several time before it converges. Thus, The agent does fair amount of exploration even if greedy actions are selected most of the time. As long the intial values are larger than the optimal values for each state, there will be optimistic exploration, the larger the value, the more the exploration.

6 Task 4

The code is done in new python file (qlearning2). The grid size of the first continous parameter are selected as 16 for each, similar to previous Task1. The results:

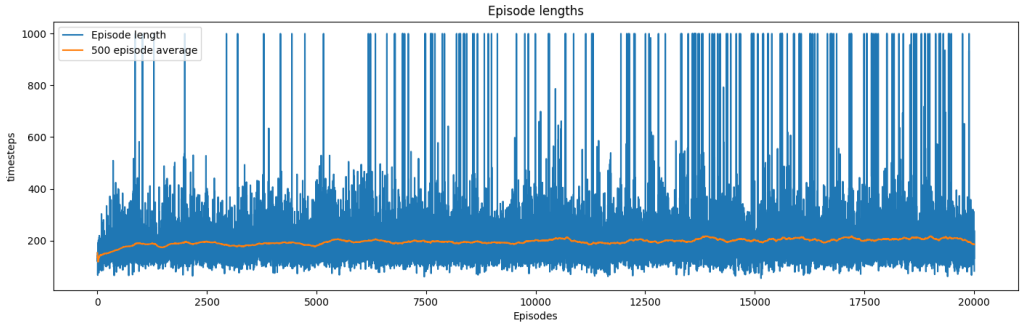

Figure 6: Lunar lander training

7 Question 3.1

The lander doesnot learn much. This is due to two reasons:

1. The range of the observation is much larger than the previous Tasks, and it still is divided across 16 grid, which mean states representation is by far less accurate.
2. The action-state values of Task1 was $16*16*16*16*2 = 131072$, for lunar lander, the action-state values are of size $16*16*16*16*16*16*2*2*4 = 268435456$, which is 2048 more values than previous Task to explore and converge. Thus, it is impossible to train for the same episodes.