Exercise 3

Gengcong Yan - 1009903
ELEC-E8125 - Reinforcement Learning

October 17, 2021

1 Task 1.1

```python
def get_action(state, q_values, epsilon, greedy=False):
    # TODO: Implement epsilon-greedy

    if random.uniform(0,1)<epsilon:
        # random action
        return env.action_space.sample() # 0/1
    else:
        # Choose current best action
        return np.argmax(q_values[get_cell_index(state)])

def update_q_value(old_state, action, new_state, reward, done,
   q_array):
    # TODO: Implement Q-value update
    old_cell_index = get_cell_index(old_state)
    new_cell_index = get_cell_index(new_state)

    q_values=[q_array[new_cell_index+ (a,)] for a in range(env.
       action_space.n)]
    q_max=max(q_values)

    q_array[old_cell_index+(action,)]+=alpha*(reward + gamma *
       q_max - q_array[old_cell_index+(action,)])
```

The result are shown in Fig.1 and Fig.2.

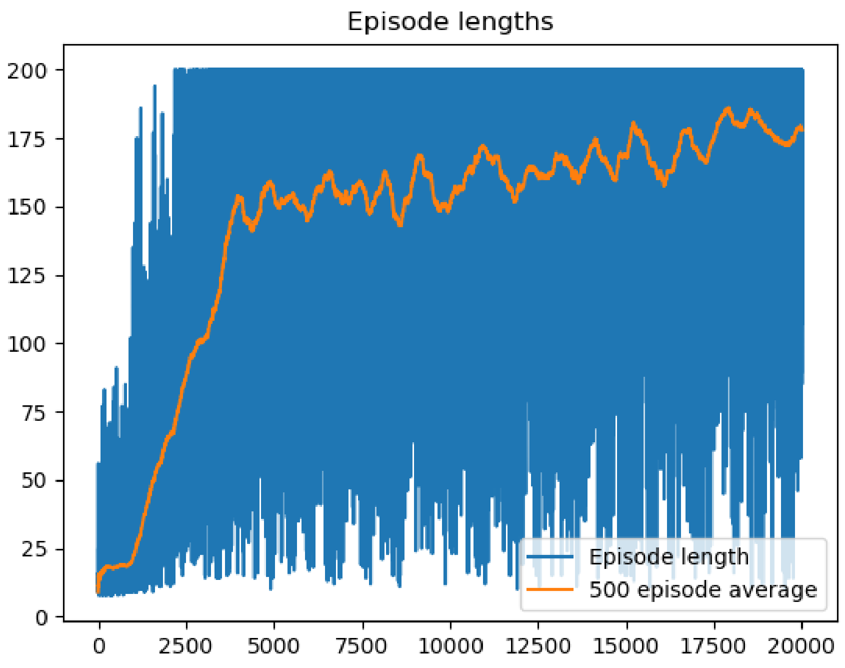


Figure 1: Episode length with $\epsilon = 0.2$

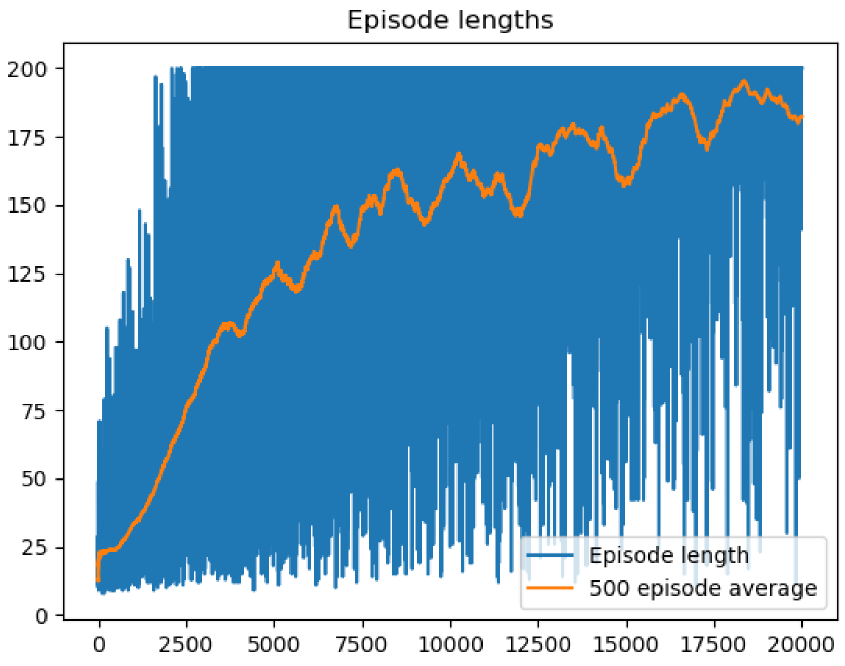


Figure 2: Episode length with GLIE

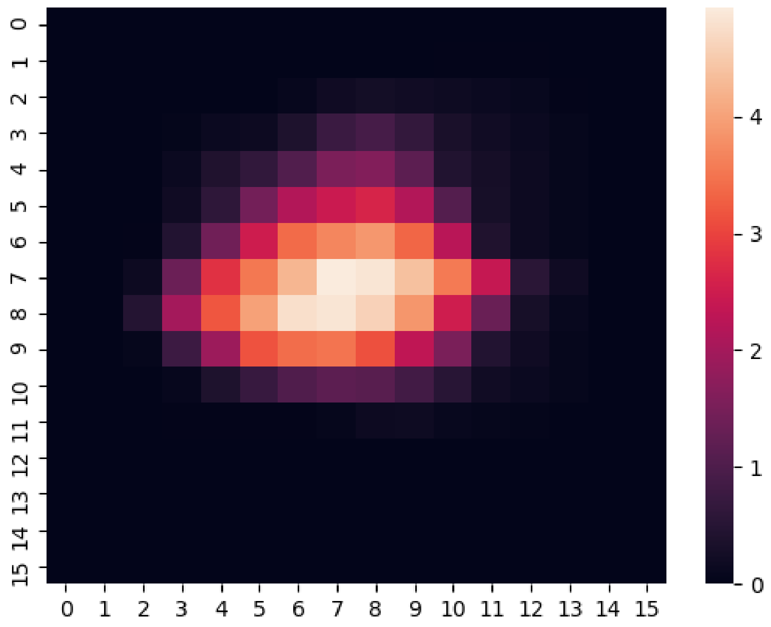


Figure 3: Optimal value function of each state

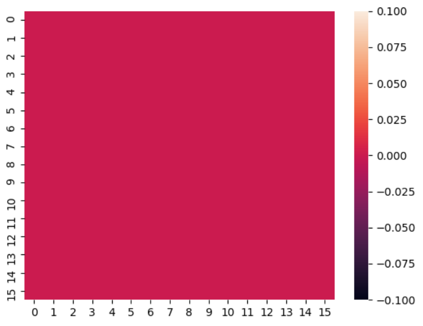
2 Task 1.2

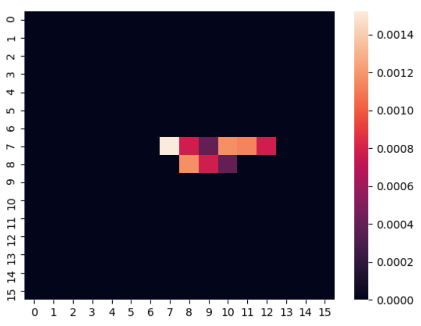
```
# Save the Q-value array
np.save("q_values.npy", q_grid)  # TODO: SUBMIT THIS Q_VALUES.
    NPY ARRAY

# Calculate the value function
values = np.amax(q_grid,axis=4) # TODO: COMPUTE THE VALUE
    FUNCTION FROM THE Q-GRID
np.save("value_func.npy", values)  # TODO: SUBMIT THIS
    VALUE_FUNC.NPY ARRAY

# Plot the heatmap
# TODO: Plot the heatmap here using Seaborn or Matplotlib
sb.heatmap(np.mean(values, axis=(1, 3)))
plt.show()
```

The result is shown in Fig.3.

3 Question 1

(a) Before training. All values are 0 because our initial setting makes them all 0.


(a) Before the training


(b) After a single episode

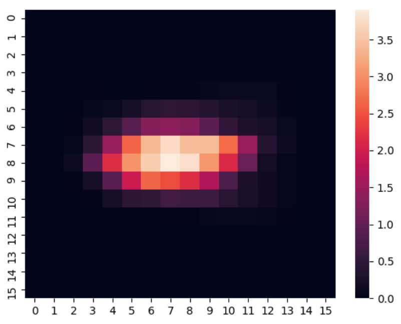

(c) After 10000 episodes

Figure 4: Value function heatmaps

(b) After 1 episode. Just a tiny amount of states updated Q values, the rest of grid are still 0.

(c) After 10000 episodes. The heatmap now looks similar to the heatmap in the end. States currently are non-zero will mostly keep to the end. These areas are frequently visited and modified, which mainly lies around the center of grid. So I think the optimal policy of the model is trying to keep velocity and position close to 0, resulting in longer timesteps.

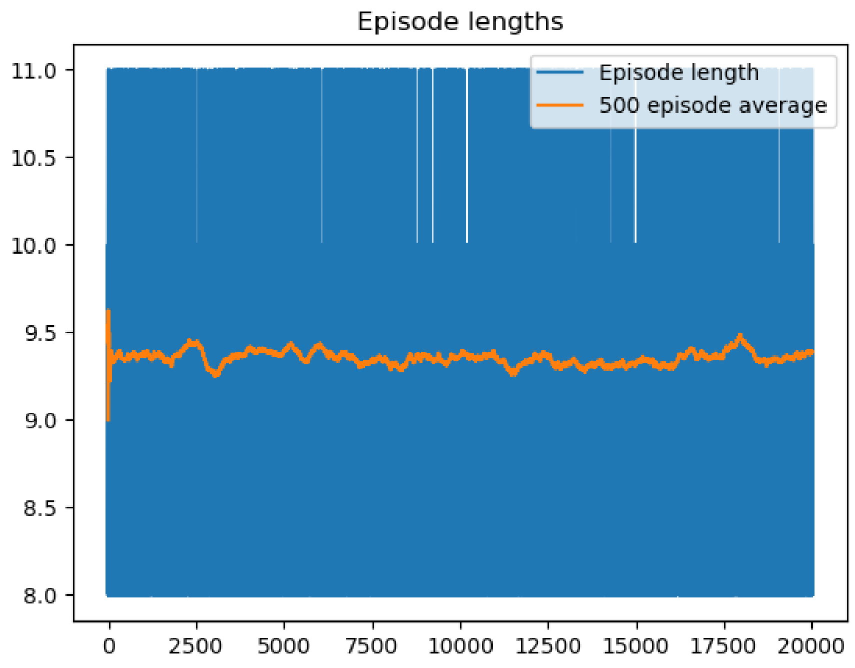
4 Task 1.3

The result plots are in Fig.5.

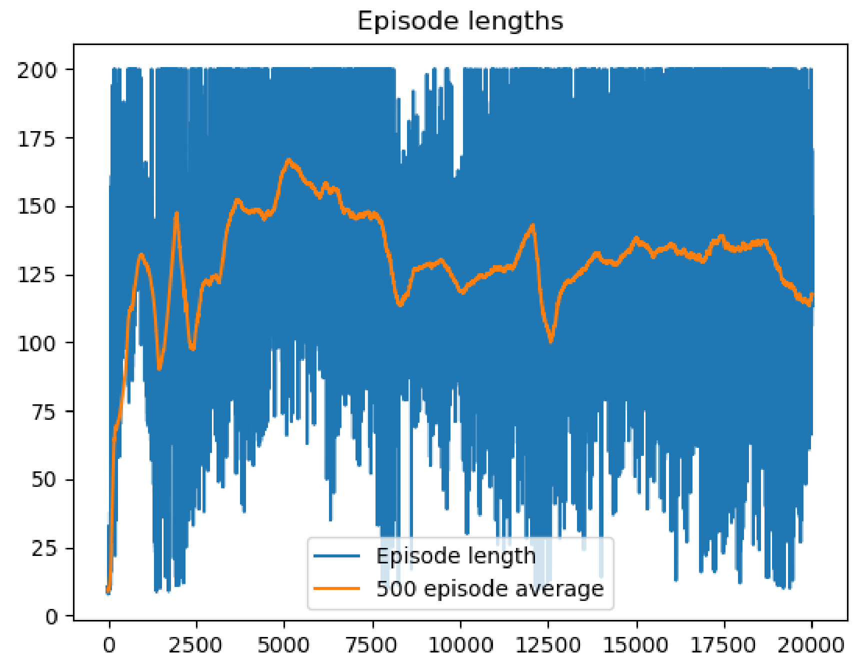
5 Question 2

5.1 Question2.1

After comparing the initial estimates of the Q function to 0 and 50, we can see the model with initial $Q = 50$ performs better. In the beginning, I didn't consider the situation where



(a) Initial Q=0



(b) Initial Q=50

Figure 5: Episode lengths with different Q

done==True. So $R + \gamma \max Q(S_{T+1}, a) - Q(S_T, A)$ always equals to 0 $(1 + 0.98 \times 50 - 50)$. I got the same plot with the plot of Q=0, even when Q=50. after I read the discussion in Slack channel, I added this scenario that I hadn't considered.

5.2 Question2.2

The reason why the model with Q=50 performs better is that, starting from the high values, the model can explore every state-action pairs even if the updated Q values get smaller than the original Q values. The policy will look at each state at least one time. although we set $\epsilon == 0$, the model can still explore possible new states. The exploration ability based on initialization of Q values helps model quickly find the better state-action pairs,you can see in Fig.5b episode length increased rapidly in the beginning process. At the same time, this ability also causes instability of learning, the model will keep exploring after it has learned many good state-action pairs.

6 Task 2

```python
# Didn't list all the code

# For LunarLander, use the following values:
#          [  x       y   xdot ydot theta  thetadot cl   cr
# s_min = [ -1.2   -0.3   -2.4   -2   -6.28   -8        0    0 ]
# s_max = [  1.2    1.2    2.4    2    6.28    8        1    1 ]

l_x_min, l_x_max = -1.2, 1.2
l_y_min, l_y_max = -0.3, 1.2
l_xdot_min, l_xdot_max = -2.4, 2.4
l_ydot_min, l_ydot_max = -2.0, 2.0
l_th_min, l_th_max = -6.28,6.28
l_thdot_min, l_thdot_max = -8, 8
l_cl_max, l_cl_min=0,1
l_cr_max, l_cr_min=0,1

l_x_grid = np.linspace(l_x_min, l_x_max, discr)
l_y_grid = np.linspace(l_y_min, l_y_max, discr)
l_xdot_grid = np.linspace(l_xdot_min, l_xdot_max, discr)
l_ydot_grid = np.linspace(l_ydot_min, l_ydot_max, discr)
l_th_grid = np.linspace(l_th_min, l_th_max, discr)
l_thdot_grid = np.linspace(l_thdot_min, l_thdot_max, discr)
l_cl_grid = np.linspace(l_cl_max, l_cl_min, 2) # binary choices
l_cr_grid = np.linspace(l_cr_max, l_cr_min, 2)

q_grid = np.zeros((discr, discr, discr, discr,discr, discr, 2 ,
    2 ,num_of_actions)) + initial_q

def get_cell_index_lunar(state):
    x = find_nearest(l_x_grid, state[0])
    y = find_nearest(l_y_grid, state[1])
    xdot=find_nearest(l_xdot_grid,state[2])
    ydot=find_nearest(l_ydot_grid,state[3])
    th = find_nearest(l_th_grid, state[4])
    thdot = find_nearest(l_thdot_grid, state[5])
    cl=find_nearest(l_cr_grid,state[6])
    cr=find_nearest(l_cr_grid,state[7])

    return x, y, xdot, ydot, th, thdot , cl , cr
```

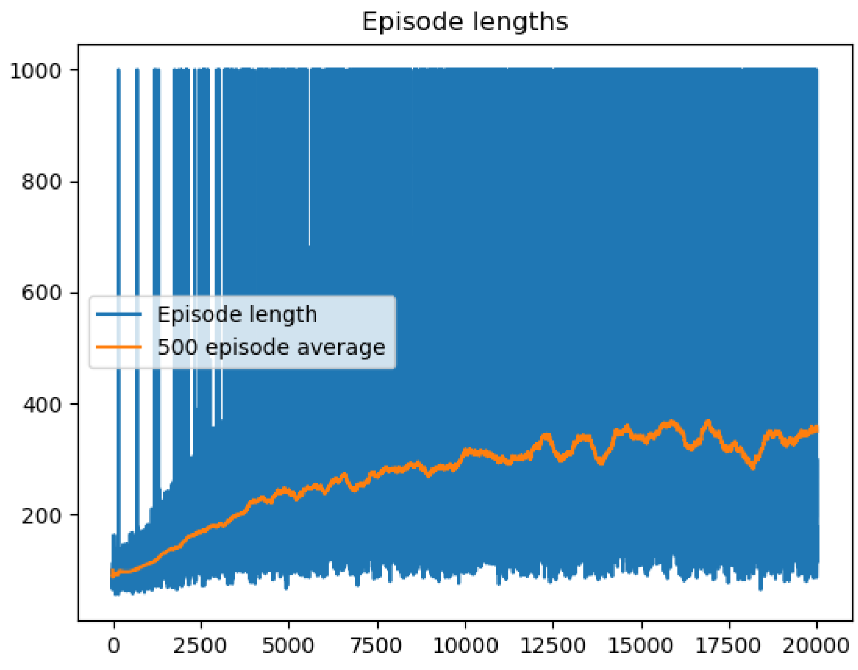


Figure 6: Episode length and average

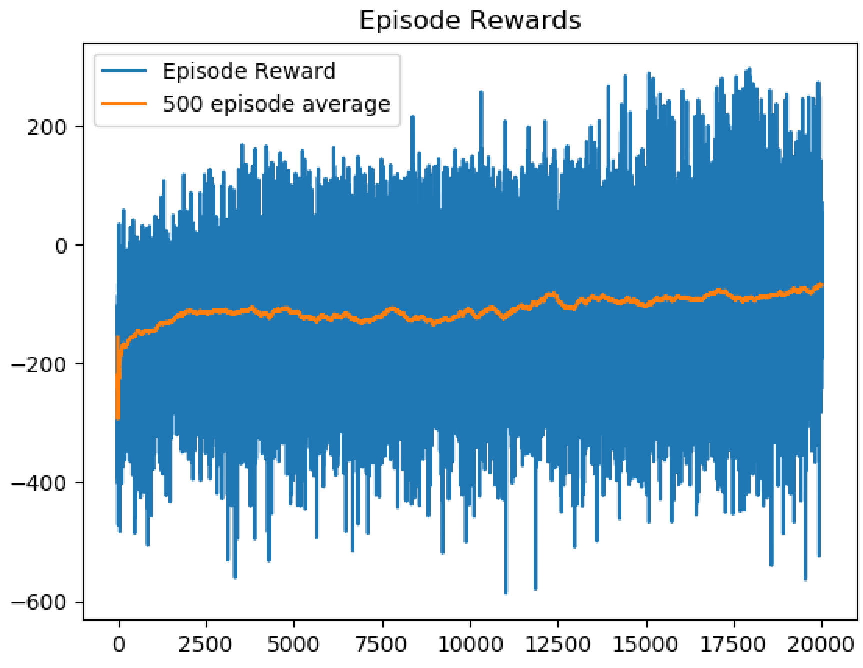


Figure 7: Reward values and average

7 Question 3

From the plot of episode length and reward in Fig.6 and Fig.7, we can see the lander failed to learn any useful behavior in training. The episode lengths and reward only increased a little during the training of 20000 episodes. Q-learning is a table function, which is valid for simple finite state tasks, if it is a continuous space, or a complex discrete space, it is not possible to get the Q value of a state by looking up the table, we should use a neural network instead. The state space is a 8 dim vector and action space is a 4 dim vector. Applying Q-learning in this task are too complicated to train a better result.

8 Feedback

1. 8h