Reinforcement Learning Lab02

Chieh-Ju Wu, Bix Eriksson

December 21, 2020

Contents

1 Introduction

The goal is to manoeuvre the lunar lander so that it lands between the two flags. The landing pad is always at coordinates $(0, 0)$. To consider the problem solved, your policy should achieve +200 points. Landing outside the landing pad is possible. Fuel is infinite, so an agent can learn to fly and then land on its first attempt. (In reality, there is a limit of 1000 steps in each episode, in order to limit the simulation time.)

1.1 State space

The state s is an 8-dimensional variable represented as $(s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8)$
where
s_1, s_2 = position in x and y axis
s_3, s_4 = velocity in x and y axis
s_5 = angle of the lunar lander
s_6 = angular velocity
s_7 = left contact points (boolean values, indicating if the space ship has touched the ground)
s_8 = right contact points (boolean values, indicating if the space ship has touched the ground)

1.2 Discrete action space

4 discrete actions are available:
$a_0 = 0$: Do nothing
$a_1 = 1$: Fire left orientation engine
$a_2 = 2$: Fire main engine
$a_3 = 3$: Fire right orientation engine

1.3 Reward

Reward for moving from the top of the screen to the landing pad and zero speed is about 100 to 140 points. If the lander moves away from the landing pad, it loses its reward. The episode finishes if the lander crashes or comes to rest, receiving an additional -100 or $+100$ points. Each leg with ground contact is $+10$ points. Firing the main engine is -0.35 points in each frame. Firing the side engine is -0.03 points in each frame.

2 Task b: Experience replay buffer and fixed target network in DQN

2.1 Experience Replay Buffer

2.1.1 What is Experience Replay Buffer

Experience replay maintains a buffer B of previous experiences (s, a, r, s'). Then the agent samples mini-batches of fixed size k from B uniformly at random, and update θ accordingly.

2.1.2 Why do we need it

Since successive updates are strongly correlated, this will affect the convergence rate of the policy. So by sampling uniformly random from the stored experiences, we can decorrelate the data or diminishing a particular trajectory.

2.1.3 Pseudo Code

Sample k experiences from the buffer B
For $i = 1, ..., k$, experiences (s_i, a_i, r_i, s_i')

$$\theta \leftarrow \theta + \alpha(r_i + \lambda \max_b Q_\theta(s_i', b) - Q_\theta(s_i, a_i))\nabla_\theta Q_\theta(s_i, a_i) \tag{2.1}$$

2.2 Fixed Target Network

2.2.1 What is Fixed Target Network

The target network is a separate network that uses a set of weight copies from the original Q network to calculate the target state-action pair for the next step $Q(s', a')$. The reason for using a separate network for this is to make sure that the target is independent of the weight-set θ, since updating the weights otherwise would affect the the target as well. If both the target and the Q-function were to be updated each step with the same weights, they would have a tendency to move in the same direction which would lead to convergence problems when we're trying to minimize the loss function.

2.2.2 Pseudo Code

Fix the target for C successive steps. For every step:

$$\theta \leftarrow \theta + \alpha(r_i + \lambda \max_b Q_\phi(s'_i, b) - Q_\theta(s_i, a_i))\nabla_\theta Q_\theta(s_i, a_i) \tag{2.2}$$

Every C steps, update the target $\phi \leftarrow \theta$

3 Task d: Choice of Network Layout, Optimizer and Parameters

3.1 Neural Network Layout

The neural network that has been implemented in a linear layout structure. This means that it uses a linear combination of a weight set $\theta = \theta_1, \theta_2, ..., \theta_n$ to approximate the Q-function.

In particular, the network chosen is a Dueling DQN which is used to approximate both the value function $V(s)$ and the advantage function $A(s, a)$. The Q-function can then be approximated according to equation 3.1 where m is the number of actions.

$$Q(s, a) = V(s) + A(s, a) - \frac{1}{m} \sum_{a'} A(s, a') \tag{3.1}$$

The reason for choosing a dueling DQN is because the model had trouble learning with the original network. A dueling DQN has a more faster and stable learning outcome which provided us with better results. The network configuration consists of 1 input layer, 2 hidden layers and 1 output layer. The input layer consists of 8 nodes which correspond to the dimension of each state input $s_i = [x_i, y_i, \dot{x}_i, \dot{y}_i, \alpha_i, \dot{\alpha}_i, boolean_{left}, boolean_{right}]$. The two hidden layers each use 128 nodes and the output layer consists of 4+1 nodes corresponding to the action space $A = [0, 1, 2, 3]$ and the value function.

3.2 Optimizer

The optimizer is used to optimize the weight-set θ in order to approximate $Q(s, a)$ to $Q^*(s, a)$. The optimizer used here is an Adam optimizer which performs a form of stochastic gradient descent(SGD) on the loss function, L. The learning rate $\alpha = 0.0005$ provided a good convergence result.

3.3 Parameters

Table 1 is the hyperparameters we chose for our neural network. For our learning rate, we take the mean of the suggested value ($10^{-3} - 10^{-4}$. For memory size, we chose a much bigger size of memory buffer with some experiments we done in the next section. For number of episodes, we chose 500, but whenever the average score of 100 is reached, we stopped training. For the frequency of updating the Q target network, we used 100, since taking the suggestion from lab

2 instruction $C \approx L/N$ does not return good results for our model. For ϵ we chose the max to be 1 and min to be 0.01, then linearly decreasing it along the training.

Parameter	Value	Description
γ	0.99	Discount factor
α	0.0005	Learning rate
L	100000	Memory size
T_E	500	Number of episodes
C	100	C steps for updating Q target network
N	64	Batch size
ϵ	1.0 - 0.01	linearly decremented

Table 1: Neural Network parameters

3.4 Choosing Memory Size

We did a little experiment with different memory sizes as follow. The goal here is to find what size of the memory size results to the fastest learning rate, in other words, taking the least episodes to reach an average reward of 100. In table 2, we can see that the model with memory size (L = 100000) is the fastest one to reach the goal. Hence, we chose memory size (L = 100000).

Different Memory size for achieving 100 avg reward with episodes taken						
γ	α	Memory size (L)	Batch size (N)	C steps	Episodes taken	avg. reward
0.99	0.0005	5000	64	50	125	100
0.99	0.0005	15000	64	50	116	100
0.99	0.0005	30000	64	50	129	100
0.99	0.0005	100000	64	50	103	100

Table 2: Different Memory size for achieving 100 avg reward with episodes taken

4 Task e: Plots and Analysis

4.1 Total episodic reward and Total number of steps taken per episode during training

4.1.1 Random Agent

Random agent takes random actions in all states. For the initial ~ 40 episodes the random agent shows a high degree of spread in total rewards according to figure 1. As the number of episodes increases the average random rewards converges to some value around $\sim (-150)$ and never shows an improvement from that point onward.

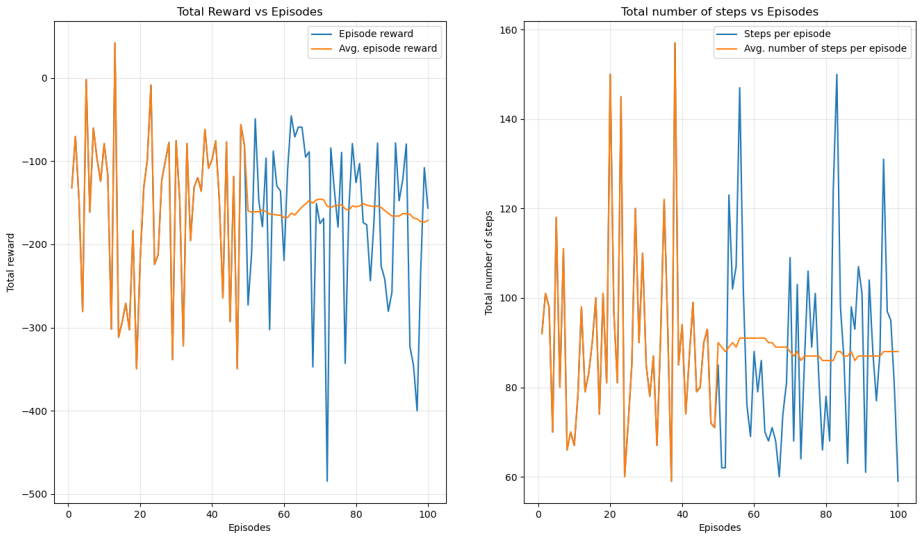


Figure 1: Random Agent

4.1.2 My Agent

As the plot shows, after roughly 20 episodes, the agent has found a policy which results in an average reward over 100. However, due to epsilon-greedy policy, the occasional random actions contributes to high oscillations in the early stage of the training. These occasional random actions may also results in an increase in steps since these more steps and actions are required to keep the space ship back on track. About 50 episodes later, when epsilon gradually decreases to its minimal value 0.01, the oscillation of average episodic rewards and number of steps become

steadier.

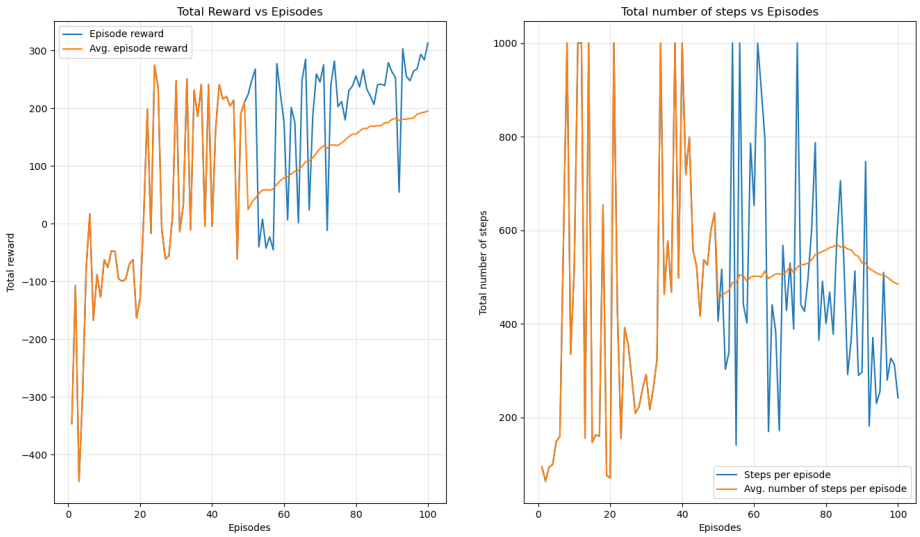


Figure 2: My Agent

4.2 Altering discount factor γ

The higher the discount factor γ, the more far-sighted the agent is. This leads to longer time training time per each iteration, however, it this also makes the evaluation of value function closer to the true value function, which results in a higher average of episodic reward in less episodes of training.

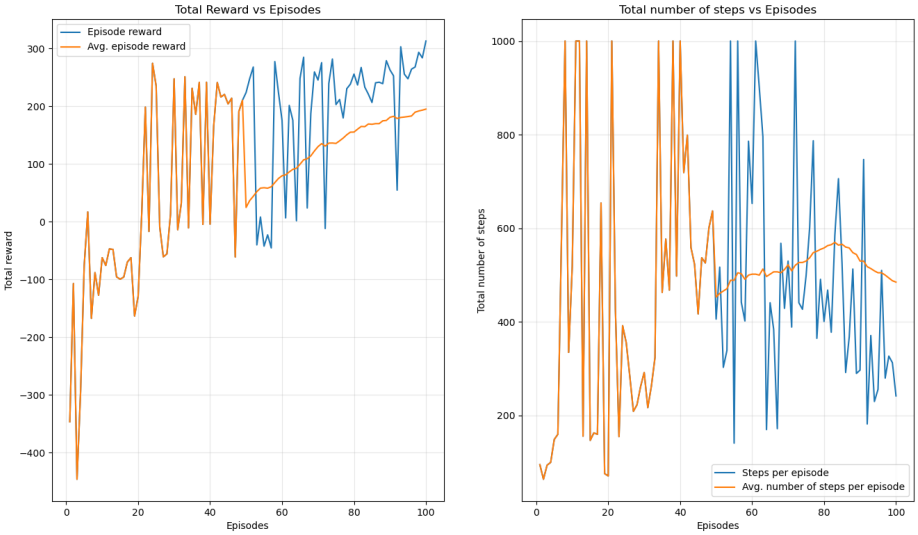


Figure 3: $\gamma_0 = 0.99$

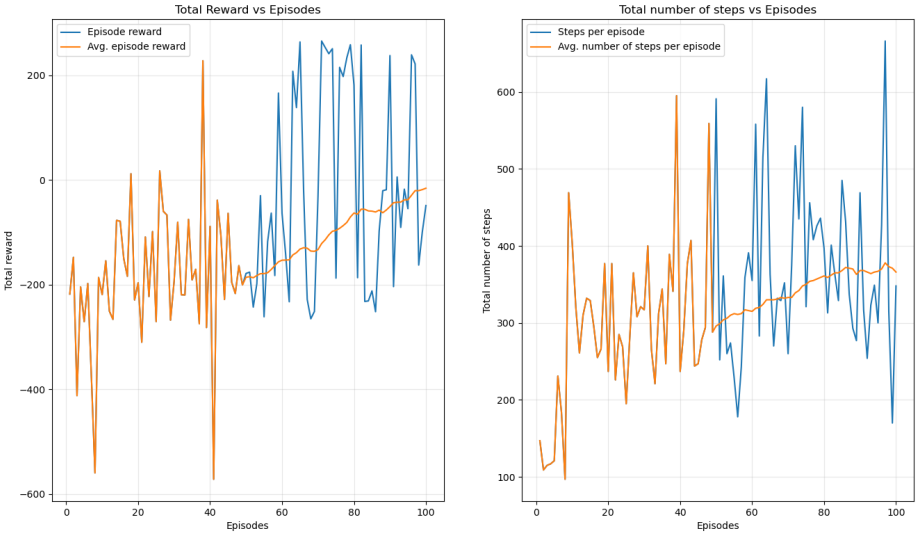


Figure 4: $\gamma_0 = 0.1$

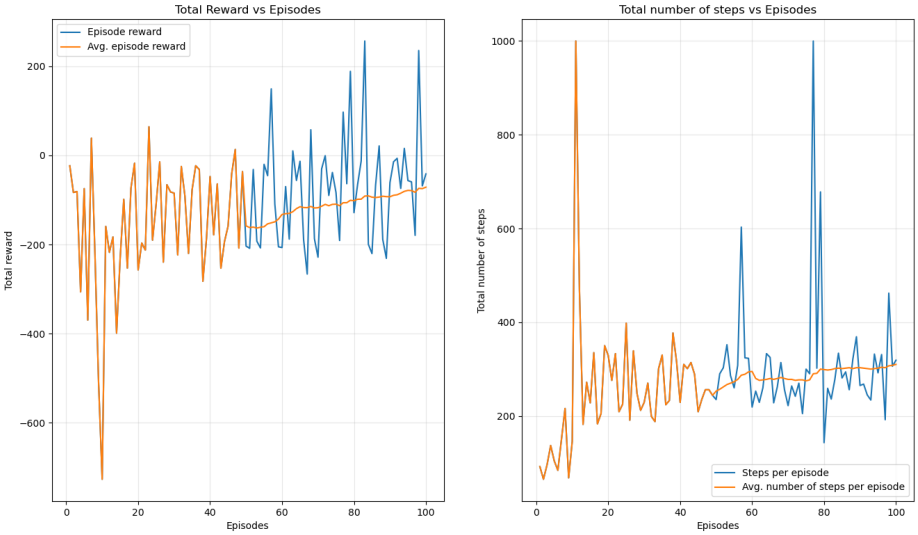


Figure 5: $\gamma_0 = 0.3$

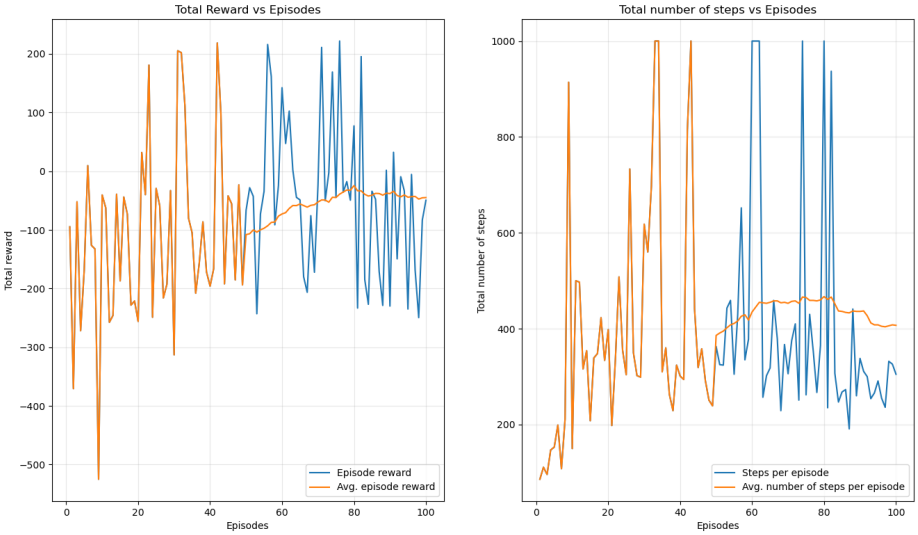


Figure 6: $\gamma_0 = 0.5$

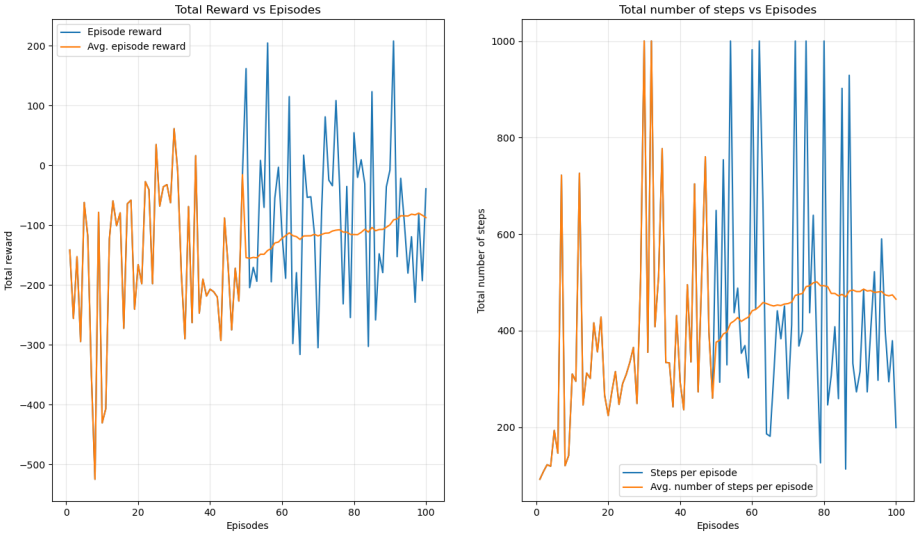


Figure 7: $\gamma_0 = 0.7$

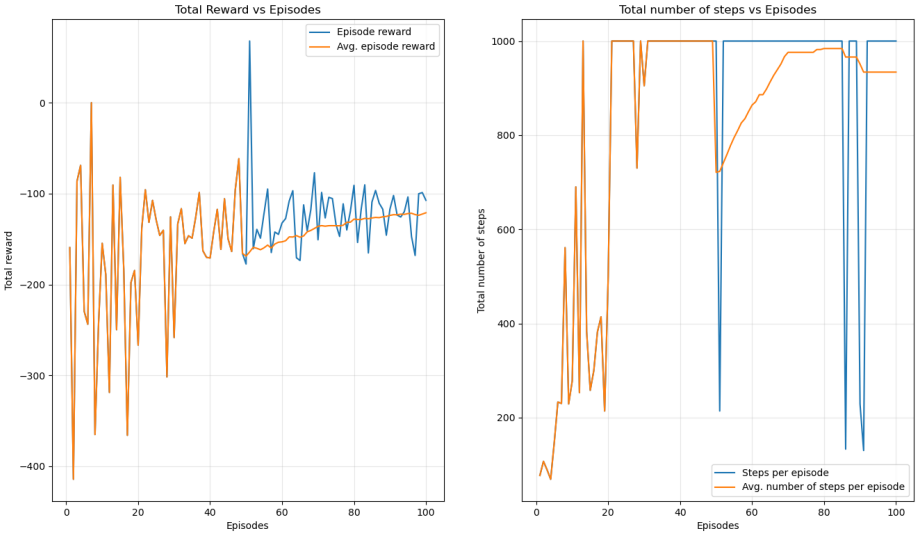


Figure 8: $\gamma_0 = 0.9$

4.3 Altering memory size L

For us, changing the memory size of the buffer does not really make a big difference regarding total episodic reward and total number of steps taken per episode during training.

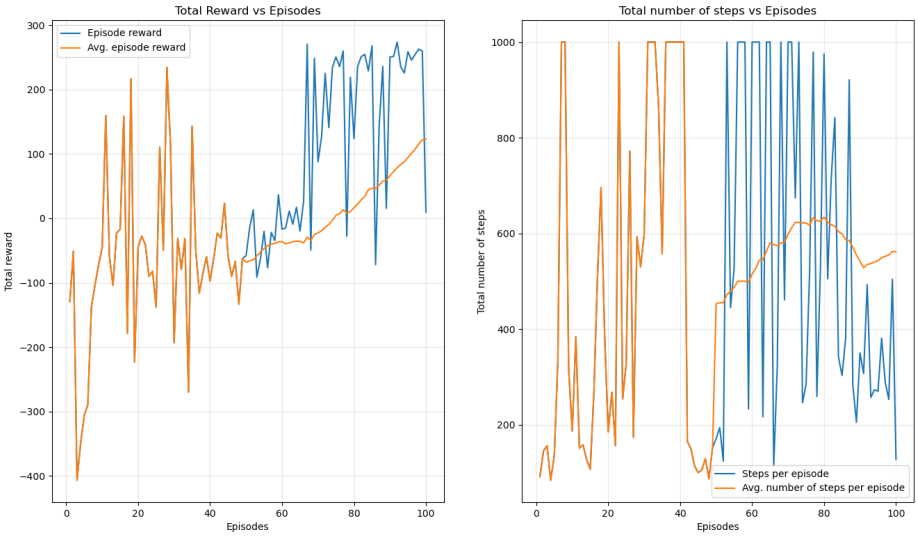


Figure 9: L = 5000

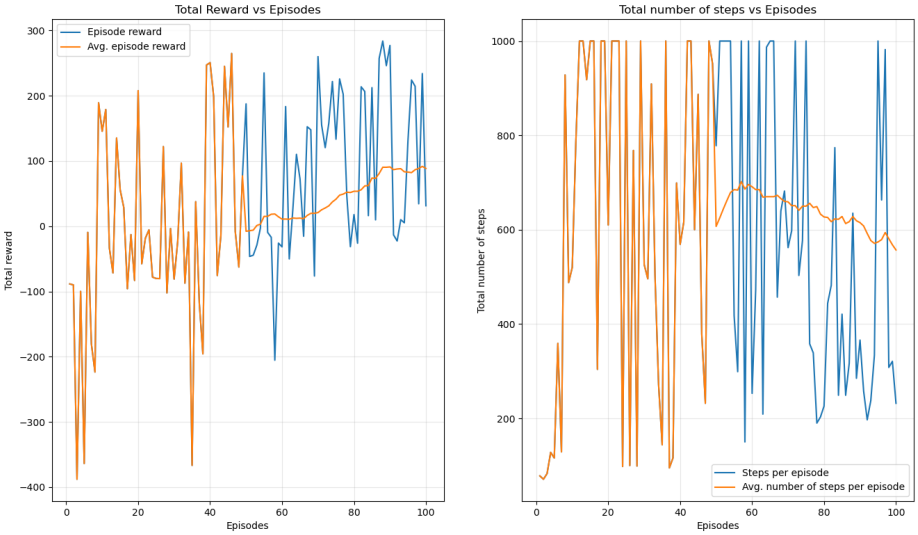


Figure 10: L = 10000

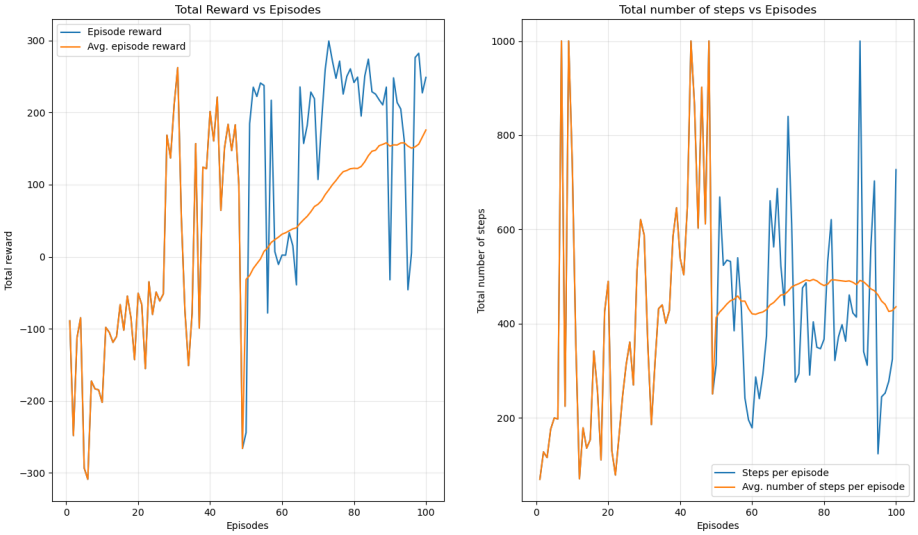


Figure 11: L = 15000

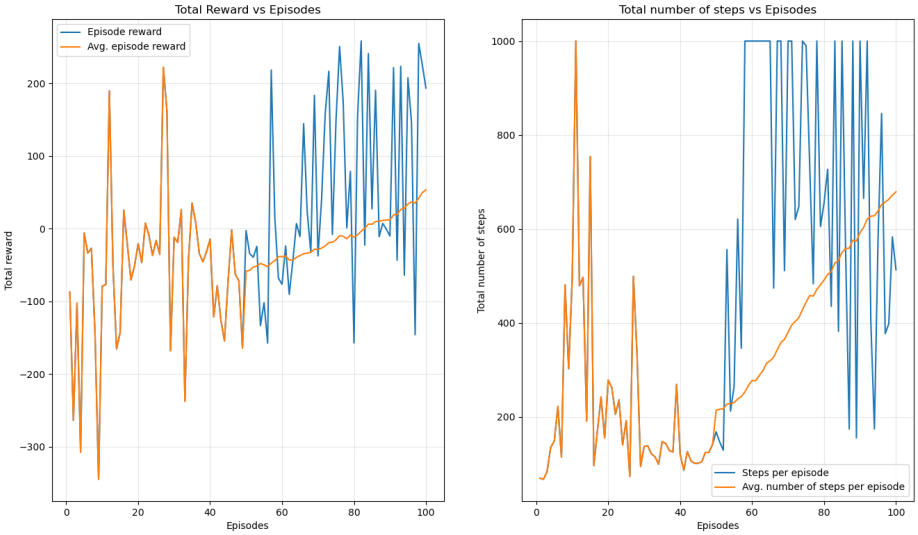


Figure 12: L = 30000

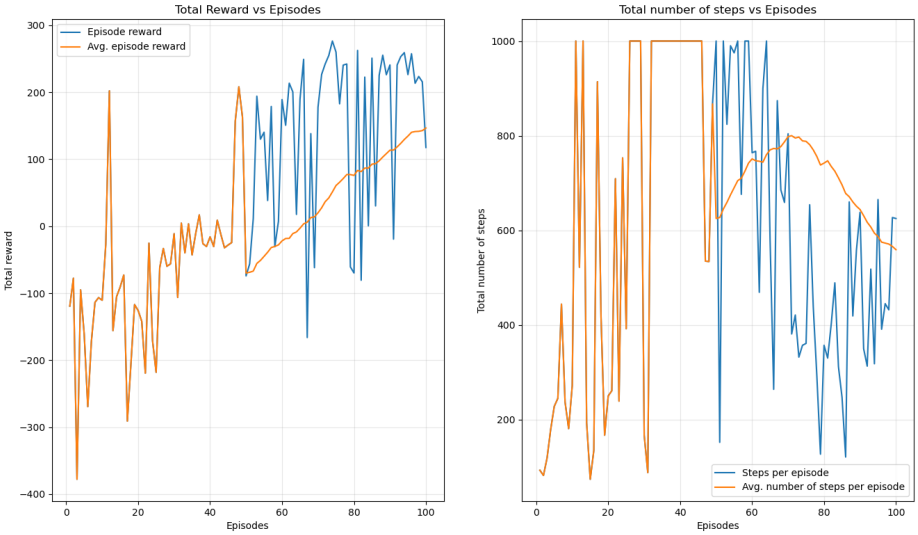


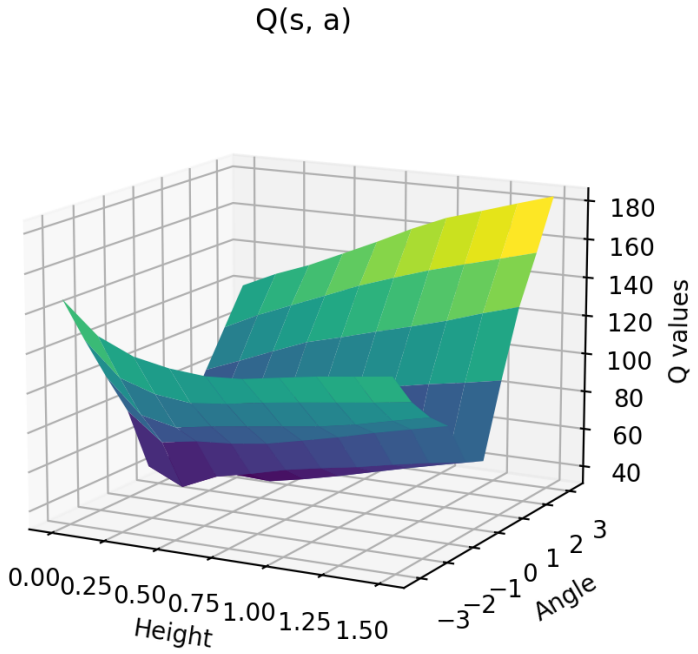
Figure 13: L = 100000

5 Plots and Analysis for Q-network

Consider the following restriction of the state $s(y, w) = (0, y, 0, 0, w, 0, 0, 0)$ where y and w are the height and angle of the lunar lander. Height refers to the distance between the moon and the lunar lander.

5.1 Plot $\max_a Q_\theta(s(y, w), a)$ for varying $y \in [0, 1.5]$ and $w \in [-\pi, \pi]$

In the plot, we can see that the Q values are higher on the edge and lower in the middle. It is because when the space ship is high and have high degrees of angle, any action that would make itself back on track will return higher value. On the other hand, when the space ship is zero degree and close to the ground, any action will easily make itself disoriented, thus, returns lower Q values.



5.2 Plot $argmax_a\ Q(s(y, w), a)$ for varying $y \in [0, 1.5]$ and $w \in [-\pi, \pi]$

From the plot, we can see that when the angle of the lunar lander is positive, the agent will use the right orientation engine (action 3) to make it back to zero degrees and when the angle of the lunar lander is negative, the agent will use the left orientation engine (action 1) to make itself straight. When the angle is zero while almost reaching the ground, the best policy is to do nothing (action 0).