EQ2305 Lab2 report

Siyi Qian
qsiyi@kth.se
200012298709

Chenting Zhang
chzha@kth.se
200205146202

December 22, 2023

1 Problem 1

1.1 b

Replay buffer is used to store a collection of past experiences, which are (state, action, reward, next state, done). The reason of using replay buffer is to break the correlations between sequential experiences which would lead to inefficient learning and high variance. Thus, randomly sampling from the replay buffer can solve this problem.

Target network is used for fixing the "target" in the Q-learning network. Since DQN algorithm is a semi-gradient method and it only considers the gradient of the $Q_\theta(S_t, a_t)$ with respect to θ. Thus, it is more stable to make the target staionary. And the way to reach that is to have its own network which is the copy of the Q-network.

1.2 d

Network Layout We have two hidden layers and each layer contains 64 neurons. The input size is the dimension of the state which is 8. The output size is 4, which is the number of all possible actions. Figure 1 illustrated the summary of our Q network. The total trainable parameters are 4996. The hidden layer is a fully connected linear layer each with an activation ReLU function following.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                   [-1, 64]             576
              ReLU-2                   [-1, 64]               0
            Linear-3                   [-1, 64]           4,160
              ReLU-4                   [-1, 64]               0
            Linear-5                    [-1, 4]             260
================================================================
Total params: 4,996
Trainable params: 4,996
Non-trainable params: 0
```

Figure 1: Network Summary

Optimizer For the optimizer, we chose the Adam optimizer with a learning rate of $5 \cdot 10^{-4}$. We chose the Adam optimizer because it combines the benefits of other optimizers like AdaGrad and RMSProp and is efficient. The gradient clipping used here is of norm 1.

Discount Factor The discount factor we use is 0.99.

Buffer Size The buffer size we use is 30000. A larger buffer size means we have more samples to train the Q-network, while it will slow down the training. However, since we can adjust the episode number, we finally chose 30000 as our buffer size.

Number of Episodes is decided to be 500 because the average episode reward increases far less around episode 500 and it achieves 200 points.

Update Step To make the model stable, the update step C is usually computed from $C \approx L/N$, where L is the buffer size and N is the size of the random batch sampled from the buffer.

Batch Size The batch size should be of the order from 4 to 128. Generally, a small batch size can provide a higher training speed and a better model. However, we found that a batch size of 128 works well so we decided to use this value.

Epsilon We use the ϵ-greedy policy when we choose the action. Since Q-learning is an off-policy algorithm, we want it to go through every state-action pair infinitely often. However, we also want it to choose the action that gives the best reward. Thus we use the ϵ-greedy policy with a decaying ϵ. There are two ways to decrease the ϵ which are linear decay and exponential decay. During our experiment, we found that linear decay performs better. The parameters chosen are $\epsilon_{min} = 0.05$, $\epsilon_{max} = 0.99$, and $Z = 0.93N$.

1.3 e

(1)

The following figure shows the reward (left) and the step (right).

From the reward plot, we can first see that the reward is increasing steadily. The reward is minus and increasing at a slow speed for the first 300 episodes. After episode 300, the reward increases to positive values and it increases faster. Then at around episode 400, the increasing speed again gets slower.

From the step plot, we can see that the steps taken are increasing before episode 350, which means the agent can survive more time before the lander crashes and it is trying to land in the correct place. After episode 350, the steps taken decrease significantly, which indicates that the agent is learning how to land and the lander is not going to crash now.

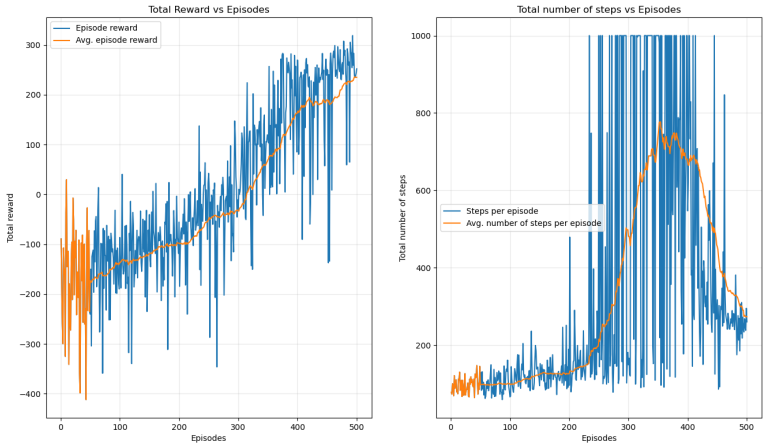

Figure 2: The total episodic reward and the total number of steps taken per episode during training.

(2)

The discount factor measures how important the reward will be gained if the next state is entered is. When the discount factor chosen is 1, it means the agent only considers the future reward. We get the following figure 3.

From the reward plot, we can see that the average reward seldom goes higher than zero and it is getting lower and lower. From the step plot, we can see that the total steps also increase and then decrease, but the steps per episode reach the step maximum limitation very often, which means the lander is neither crashing nor landing.

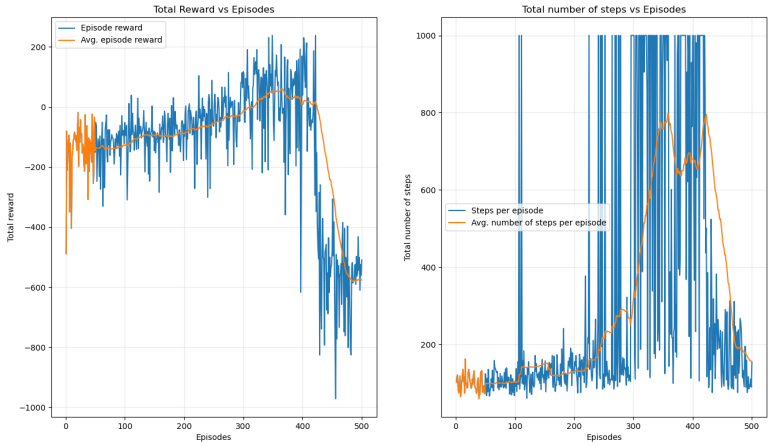

Figure 3: The total episodic reward and the total number of steps taken per episode during training when $\gamma = 1$.

When the discount factor is very low, as we choose 0.03 here, the agent only considers the reward gained from the next state. We get the following figure 4.

From the reward plot, we can see that the average reward is steadily between 0 and -200. The step plot performs similarly to the one with the discount factor equal to 1. We can find many episodes reaching around 1000 steps, meaning the lander is neither crashing nor landing.

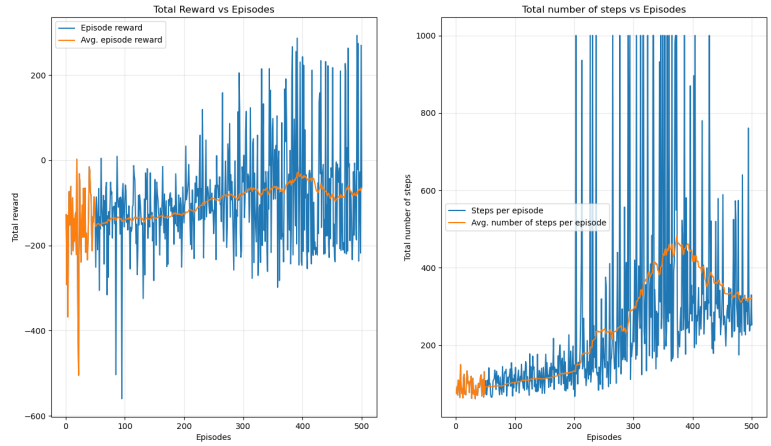

Figure 4: The total episodic reward and the total number of steps taken per episode during training when $\gamma = 0.03$.

(3)

Without changing other parameters, we changed the episode numbers and got the following four figures with 100, 400, 700, and 1000 episodes respectively. When the model has 100 episodes, it is obvious that training is not completed. The average reward just reaches zero and the steps keep increasing. When there are 400 episodes, the model is close to our final model though the reward doesn't reach 200 points. When the number of episodes goes to 700 and 1000, we can see that the total reward and the average number of steps are going up and down repeatedly. One explanation of this is that when there are too many episodes, the model may forget what it learned before and later learn it again.

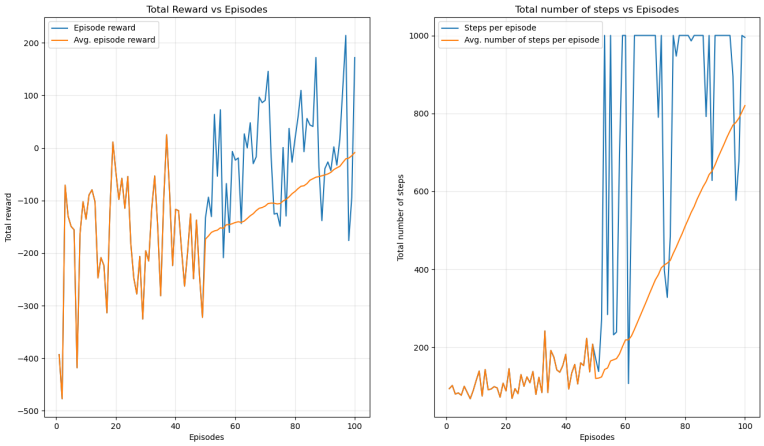

Figure 5: The total episodic reward and the total number of steps taken per episode during training with 100 episodes.

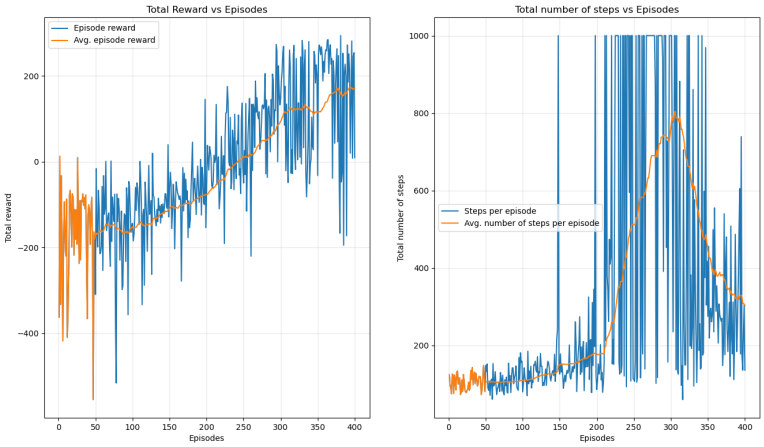

Figure 6: The total episodic reward and the total number of steps taken per episode during training with 400 episodes.

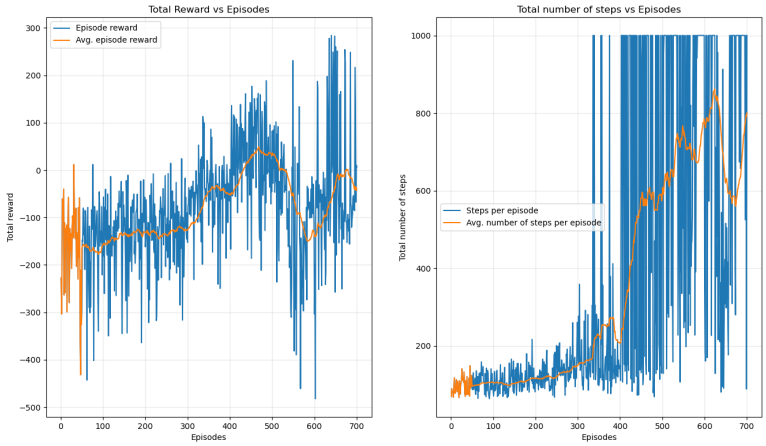

Figure 7: The total episodic reward and the total number of steps taken per episode during training with 700 episodes.

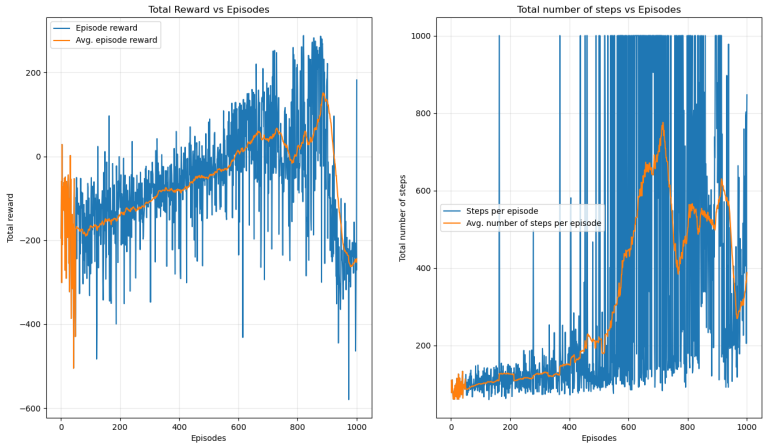

Figure 8: The total episodic reward and the total number of steps taken per episode during training with 1000 episodes.

The following figures show our model with different buffer sizes. When the buffer size is 5000, we can see from the reward plot that the agent is learning at a low pace. Sometimes it seems that it forgets something and then it learns again, for example, around episode 350 the average reward goes down and then goes up. The same situation happens again when the buffer size is 20000. We can see that the average reward is going up and down and so does the average number of steps. So we can conclude that when the buffer size is not large enough, the agent can easily forget about what was learned before and learn it again.

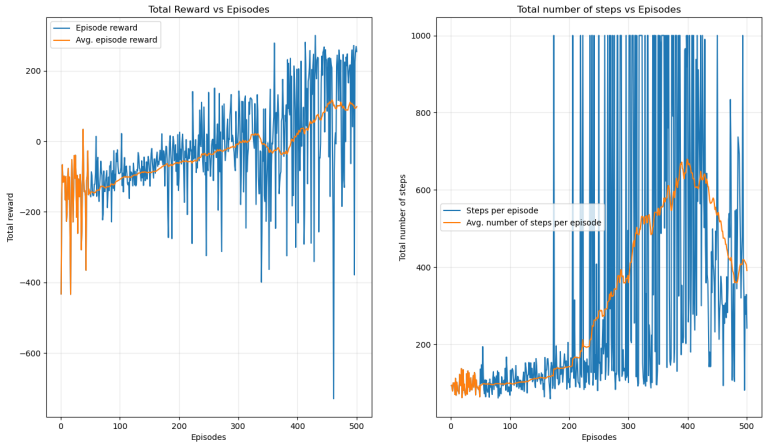

Figure 9: The total episodic reward and the total number of steps taken per episode during training with buffer size 5000.

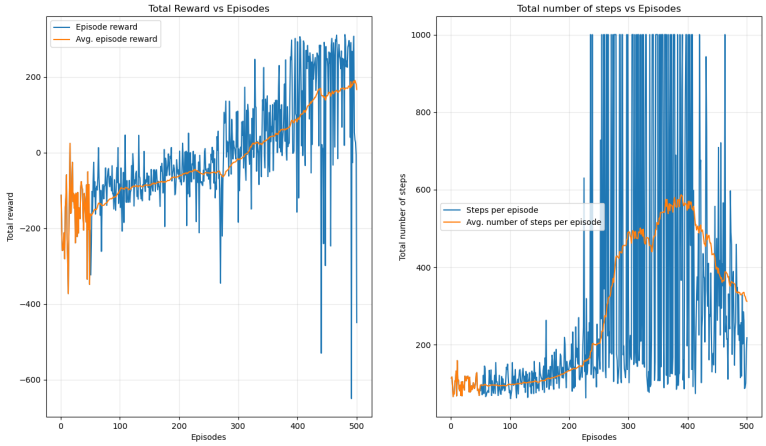

Figure 10: The total episodic reward and the total number of steps taken per episode during training with buffer size 10000.

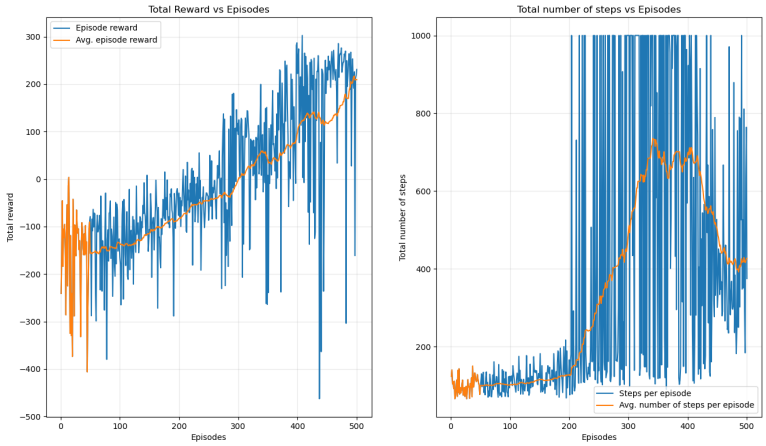

Figure 11: The total episodic reward and the total number of steps taken per episode during training with buffer size 20000.

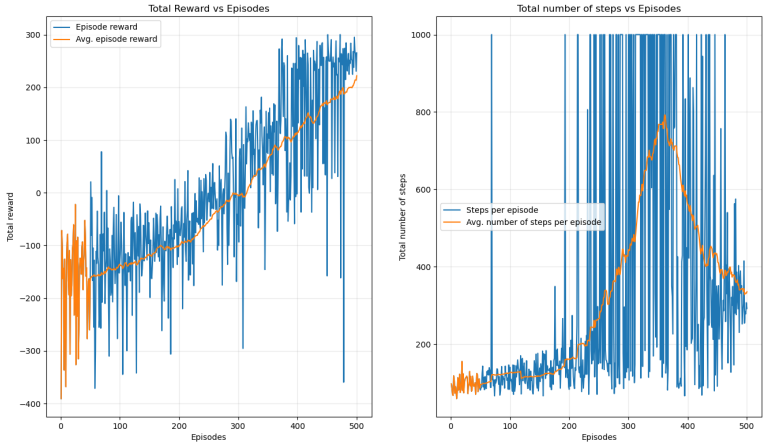

Figure 12: The total episodic reward and the total number of steps taken per episode during training with buffer size 30000.

1.4 f

In this question, we modified our state vector by only considering the $2th$ and the $5th$ dimensions, namely only computing the Q-values and gradient descent with respect to the height of the lander y and the angle of the lander w. From the "Max Q-values" graph on the left, we can observe a smooth surface, indicating a gradient of Q-values changing with the lander's height and angle. Higher values (in yellow) are associated with specific ranges of y and ω, suggesting that the model is more confident in the lander's success in those states. Lower values (in blue) would then indicate less optimal states. As seen from this figure, the Q-values change smoothly which is a good indication that the learning process was stable and that the model has developed a general understanding of the task.

From the "actions" figure on the right, we can see that the actions in the action space are discrete and are only from set $0, 1, 2, 3$ indicating $nothing, powerleft, powermain, powerright$ respectively. This action chose 3D graph is justified for the following reasons. The first reason for this is when the angle w is positive large, indicating that the lander has a tendency to lean to the left, the best action is to power right. Vice versa, when The first reason for this is when the angle w is negative large, indicating that the lander has a tendency to lean to the right, the best action is to power left which is justified in this actions figure. The second reason for this is when the lander is still high from

the goal, and the direction is not too skewed, the action selection is mostly doing nothing which is justified.

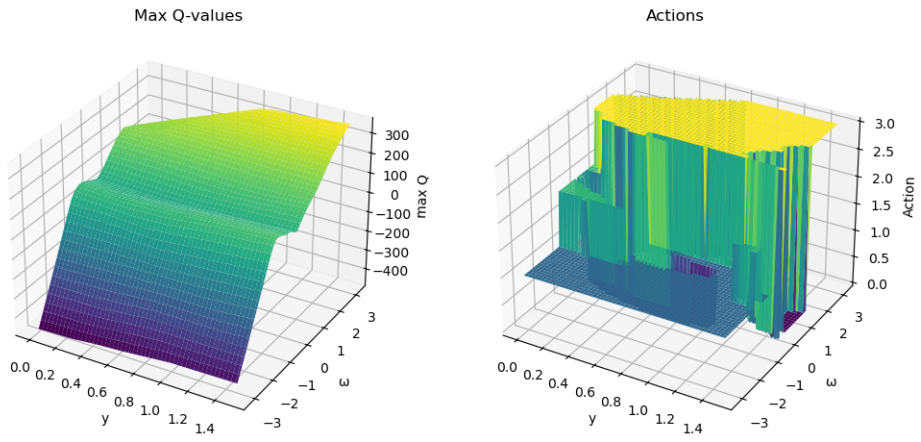

Figure 13: Q values and $argmax_a(Q)$ in 3D plot

1.5 g

From figure 14, we can observe that after the first 50 episodes, the average episode reward of DQN is much better than choosing the action from the action space randomly.

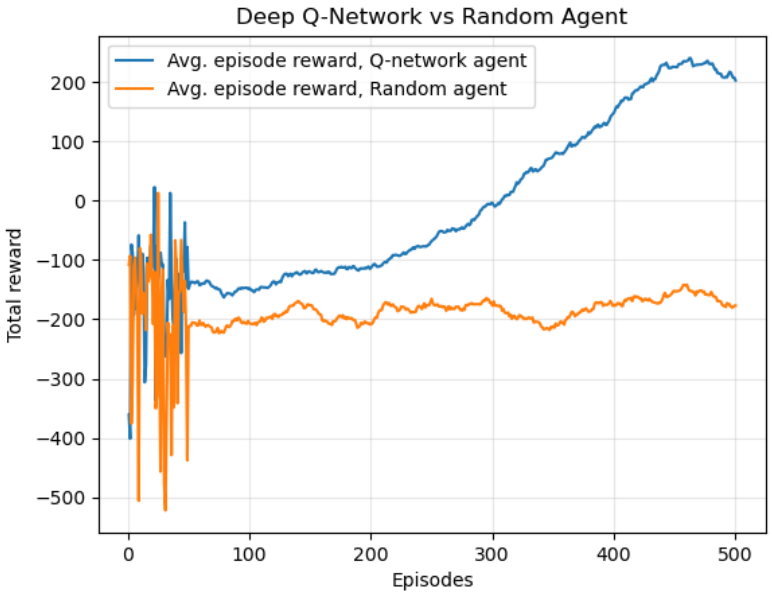

Figure 14: Deep Q-Network vs Random Agent