name: Zhengming Zhu, Xianao Lu
ID: 19990130-2035 20021201-3338

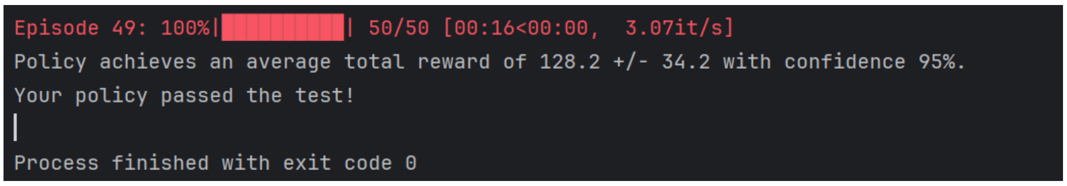
1 Problem1

1.1 Why do we use a replay buffer and target network in DQN?

The replay buffer is for storing the past experience we collects before. With the replay buffer, we can sample batches from it randomly rather than using the consecutive trajectory in standard Q-learning method. The target network can keep the target unchanged for a while. If the target changes too fast, it might make the agent hard to converge to the optimal policy.

1.2 Solution Checking

As Figure 1 shows, our policy successfully passed the test.



```
Episode 49: 100%|██████████| 50/50 [00:16<00:00,  3.07it/s]
Policy achieves an average total reward of 128.2 +/- 34.2 with confidence 95%.
Your policy passed the test!

Process finished with exit code 0
```

Figur 1: check the solution

1.3 Explain the layout of the network that you used; the choice of the optimizer; the parameters that you used $(\gamma, L, T_E, C, N, \varepsilon)$ and which modification did you implement (if any). Motivate why you made those choices.

We designed the network under the lab instruction. The first two layers have 64 neurons with ReLU activation, and the output layer has no activation.
The parameters are chosen as below.

Parameters:

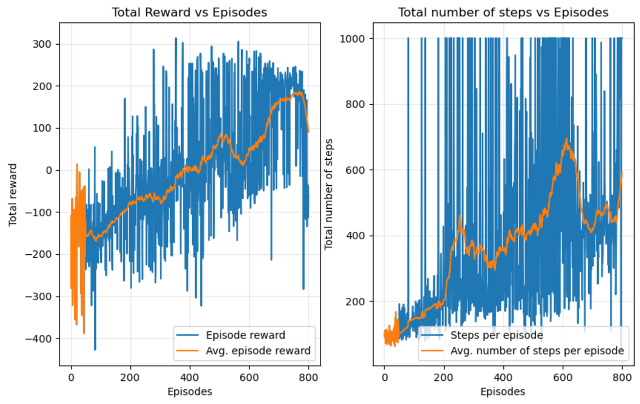
- N_episodes = 750
- $\gamma = 0.98$
- n_ep_running_average = 50
- learning rate = 5e-4
- $\varepsilon_{min} = 0.05$
- $\varepsilon_{max} = 0.99$
- batch_size = 64

- L = 10000
- C = L / batch_size

1.4 Once you have solved the problem, do the following analysis:

1.4.1 Plot the total episodic reward and the total number of steps taken per episode during training. What can you say regarding the training process?

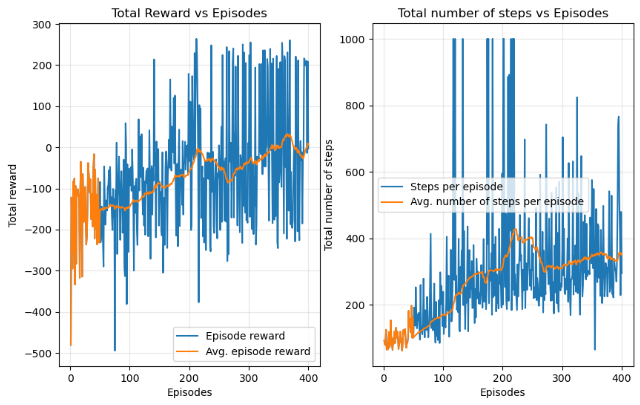
Figure 2 shows that the agent is exploring in the first few episodes, therefore, agent got very low total reward and small steps at first, and the rewards vary a lot in this period. After about 150 episodes, the result suggests that agent have learning better policy to land, the total reward increases. And after around 600 episodes, the agent comes to converge around a optimal policy.
The total number of steps of every episode increases at first because agent have not learned a good policy and explore more space. Then, the total number of steps becomes decrease when the total rewards is positive because the agent has known a good policy to land.



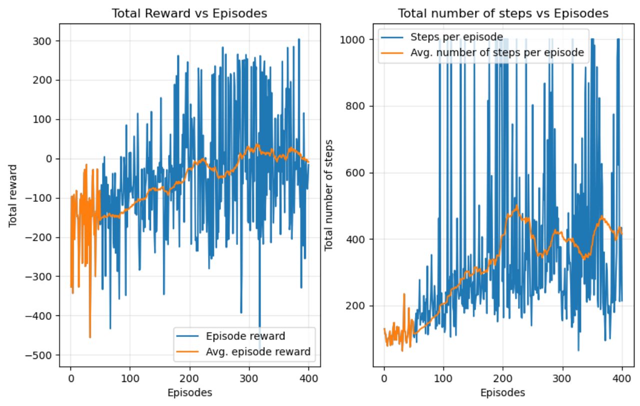
Figur 2: Reward and steps across time with default parameters

1.4.2 Let γ_0 be the discount factor you chose that solves the problem. Now choose a discount factor $\gamma_1 = 1$, and a discount factor $\gamma_2 \ll \gamma_0$. Redo the plots for γ_1 and γ_2 (don't change the other parameters): what can you say regarding the choice of the discount factor? How does it impact the training process?

As we can see from the Figure 3, we set the $\gamma = 0.8$, the too low discount factor result in a bad policy, because the low discount factor means that the agent attach less importance to future rewards.
On the other hand, when $\gamma = 1$, it means that the agent won't learn from the past experience. So under such scenario, the policy also can't converge properly.



Figur 3: $\gamma = 0.8$

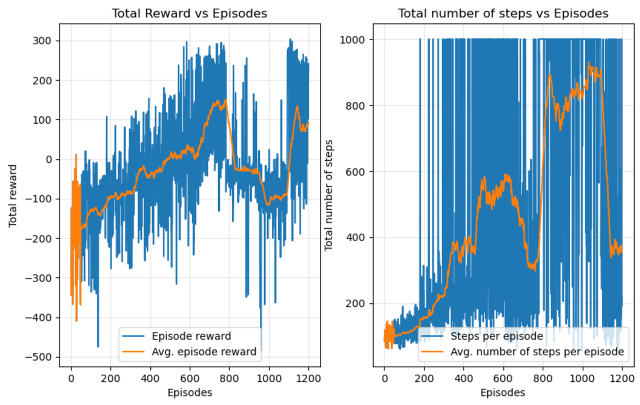


Figur 4: $\gamma = 1$

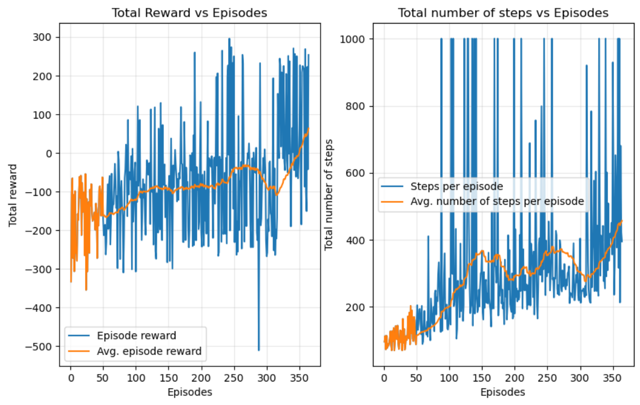
1.4.3 For your initial choice of the discount factor γ_0 investigate the effect of decreasing (or increasing) the number of episodes. Also investigate the effect of reducing (or increasing) the memory size. Document your findings with representative plots.

As we can see in Figure 5, if we increase the episode number, the average reward will drop down around episode 800. It occurred since the training process is too long, the agent started to forgetthe past experience. And in the Figure 6, we can see that the policy haven't converge yet.

In Figure 7, increasing the memory size seems can improve the training effects.



Figur 5: Episodes = 1200

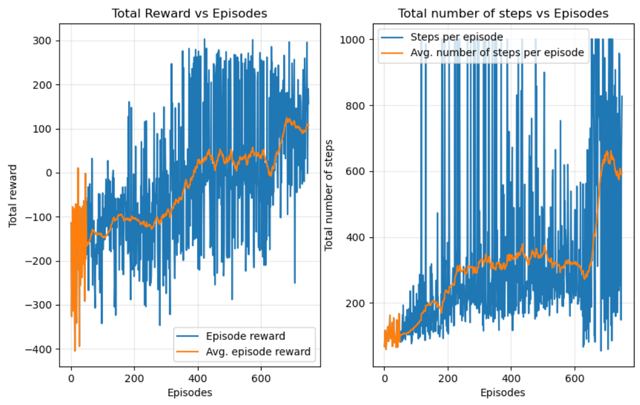


Figur 6: Episodes = 400

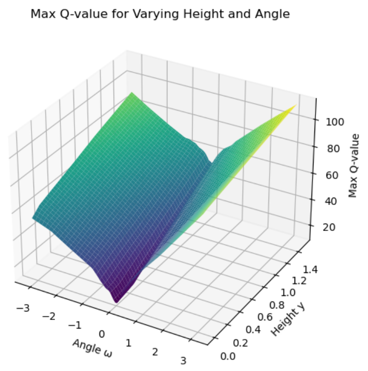
1.5 For the Q-network Q_θ that solves the problem, generate the following two plots:

1.5.1 Consider the following restriction of the state $s(y, \omega) = (0, y, 0, 0, \omega, 0, 0, 0)$, where y is the height of the lander and ω is the angle of the lander. Plot $Q_\omega(s(y, \omega), a)$ for varying $y \in [0, 1.5]$ and $\omega \in [-\pi, \pi]$. You should obtain a 3D plot. Does the value of the optimal policy you found make sense? Explain it.

As Figure 8 shows, the variation of ω affects the Q-value significantly. When $\omega = 0$, the Q-value achieves the lowest point, which makes sense since the spacecraft should land to the landing point vertically to get higher rewards.



Figur 7: memory_size=20000



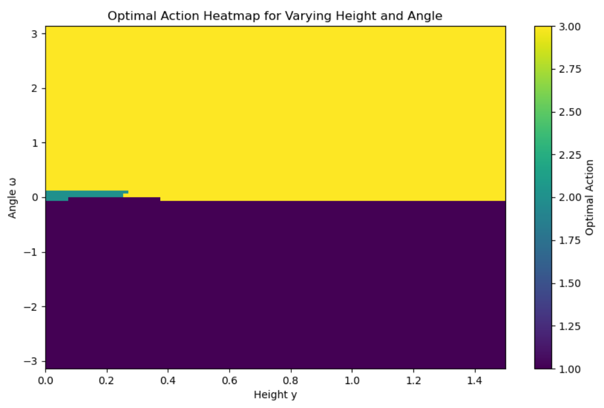
Figur 8: Q-value for varying y and ω

1.5.2 Let s be the same as the previous question, and plot$argmax_a Q(s(y,\omega),a)$ for varying y and ω (as in the previous question). You should obtain a 3D plot. Does the behaviour of the optimal policy make sense? Explain it.

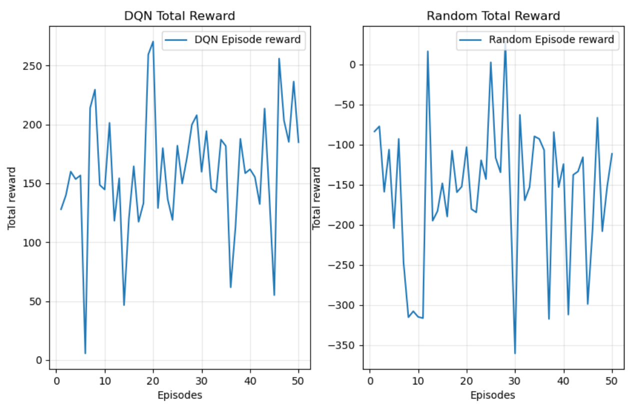
As Figure 9 shows, the policy seems to make sense because when the ω is away from 0, or to say, the spacecraft is more tilted, then the side engine should provide a bigger power to make the spacecraft back to the vertical state. When the spacecraft is vertical($\omega = 0$), the side engine do not need to provide much power.

1.6 Compare the Q-network you found with the random agent in (a). Show the total episodic reward over 50 episodes of both agents.

As the Figure 10 shows, the DQN agent can achieve higher total episodic rewards.



Figur 9: $argmax_a Q(s(y,\omega),a)$ for varying y and ω



Figur 10: total rewards under DQN agent and random agent

2 Problem2

2.1 Some questions

2.1.1 Why don't we use the critic's target network when updating the actor network? Would it make a difference?

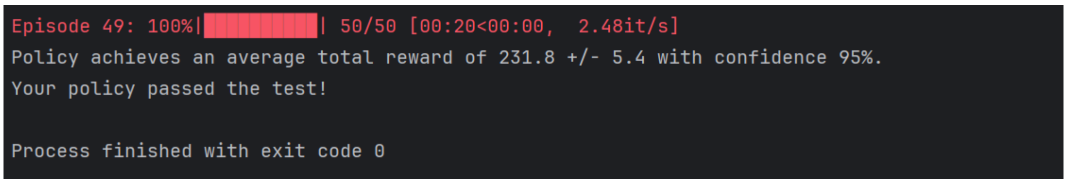
The target network is slowly updated from the main network in the training process, which helps stabilize training. Also, the objective of updating the actor network is to maximize the expected return, this is achieved by the real time evaluation from critic network, not the slow updated target critic network. If using the critic's target network when updating the actor network, it could slow down the learning process, because of the outdated evaluation.

2.1.2 Is DDPG off-policy? Is sample complexity an issue for off-policy methods (compared to on-policy ones)?

Yes, it is off-policy, DDPG samples experience that was generated by a different policy from what it is optimizing, because it reuses an experience replay buffer. Off-policy methods have better sample efficiency compared to on-policy method, for the reason that they can learning from past experience for many times, whereas on-policy methods rely on the experience generated by current policy, which is very helpful for training when there is limited information.

2.2 Implement the modified version of DDPG (shown in Algorithm 2) and solve the problem

By running check_solution.py, we demonstrate our policy passed the test, and got an average total reward of 231.8 +/- 5.4 with confidence 95%.



```
Episode 49: 100%|██████████| 50/50 [00:20<00:00,  2.48it/s]
Policy achieves an average total reward of 231.8 +/- 5.4 with confidence 95%.
Your policy passed the test!

Process finished with exit code 0
```

Figur 11: check the solution

2.3 Answer the following

2.3.1 Explain the layout of the network that you used; the choice of the optimizer; the parameters that you used. Motivate why you made those choices.

As recommended in the instruction, the parameters and structure of the model as follows:

The layout of the network:

	input	output	activation function
layer 1	dim_{state}	400 neurons	Relu
layer 2	400 neurons	200 neurons	Relu
output layer	200 neurons	dim_{action}	Tanh

Tabell 1: Actor MLP network

	input	output	activation function
layer 1	dim_{state}	400 neurons	Relu
layer 2	400 neurons + dim_{action}	200 neurons	Relu
output layer	200 neurons	1	No activation function

Tabell 2: Critic MLP network

Both of the two network, we use Adam optimizer. Besides, for the actor network, we add the tanh activation to the output to constraint the action to be between [-1, 1], and we concatenate the output of the input layer with the action. In that way the critic network learning the state information first and then add the action information.
Parameters:

- N_episodes = 300
- $\gamma = 0.98$
- n_ep_running_average = 50
- actor_lr = 5e-5
- critic_lr = 5e-4
- ε = 1e-3
- batch_size = 64
- L = 30000
- $\tau = 0.0015$
- d = 2
- $\mu = 0.15$
- $\sigma = 0.2$

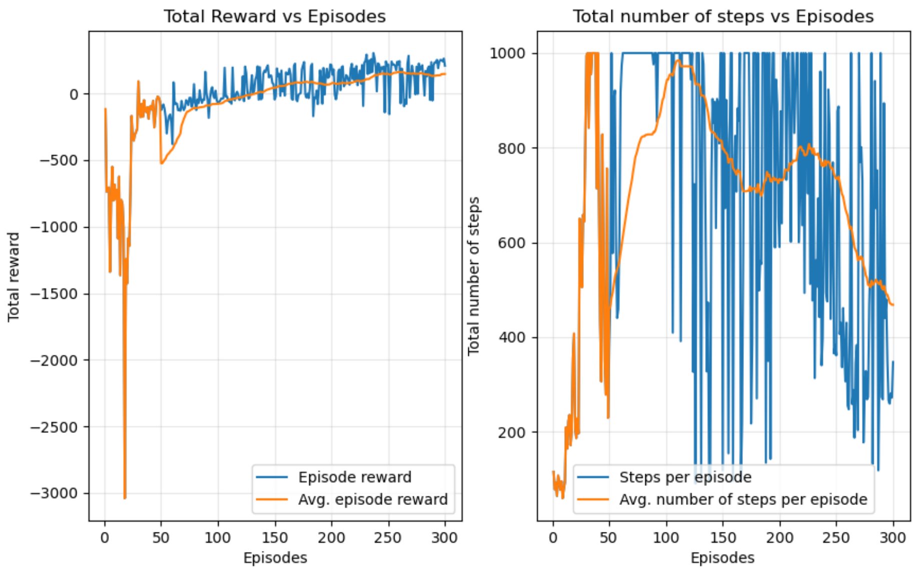
2.3.2 In general, do you think it is better to have a larger learning rate for the critic or the actor? Explain why.

In the DDPG, the actor network learns the policy and the critic network evaluate the value function. A larger learning rate for critic network might be beneficial to quickly adapt to the changing in value landscape. On the other hand, a slower learning rate for actor is often preferred to ensure the steady and conservative improvement.

2.4 Once you have solved the problem, do the following analysis.

2.4.1 Plot the total episodic reward and the total number of steps taken per episode during training. What can you say regarding the training process?

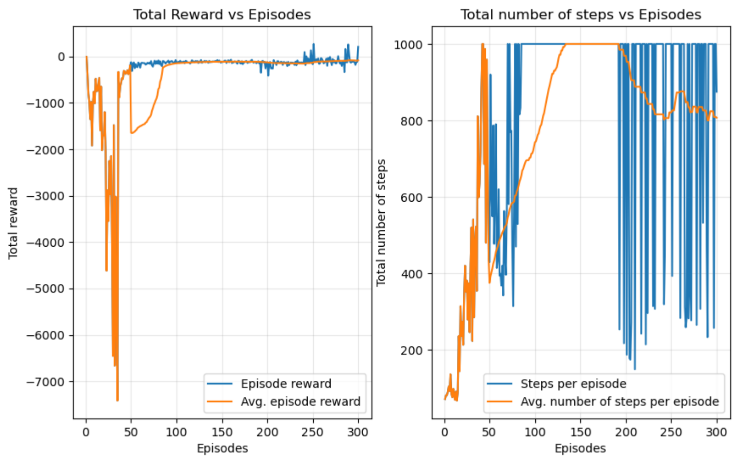
The figure shows that the agent is exploring in the first few episodes, therefore, agent got very low total reward and small steps at first. After about 150 episodes, the result suggests that agent have learning better policy to land, the total reward becomes positive and increases. The total number of steps of every episode increases at first because agent have not learned a good policy and explore more space. Then, the total number of steps becomes decrease when the total rewards is positive because agent has known a good policy to land.



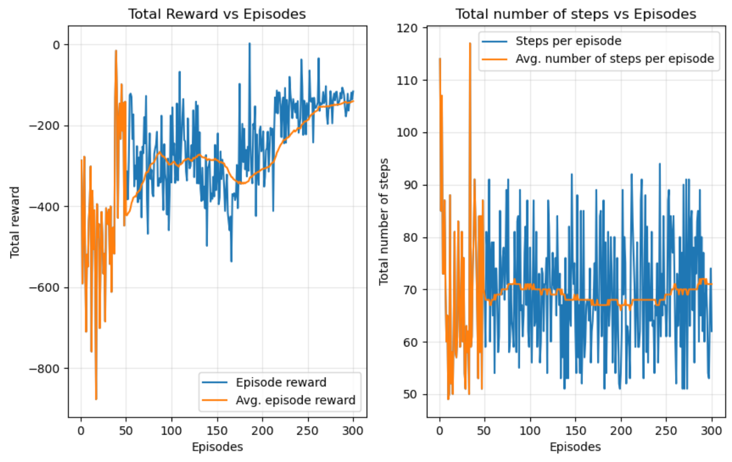
Figur 12: Reward and steps across time with default parameters

2.4.2 Let γ_0 be the discount factor you chose that solves the problem. Now choose a discount factor $\gamma_1 = 1$, and a discount factor $\gamma_2 << \gamma_0$. Redo the plots for γ_1 and γ_2 (don't change the other parameters): what can you say regarding the choice of the discount factor? How does it impact the training process?

As we can see from the figure, we set the $\gamma = 0.8$, the too low discount factor result in a bad policy, because the low discount factor means that the agent attach less importance to future rewards.
On the other hand, the discount factor can be 1 for the MDP question with terminal state, so, the total reward is still increasing at the episode 300. It may slow down the optimizing speed according to the figure.



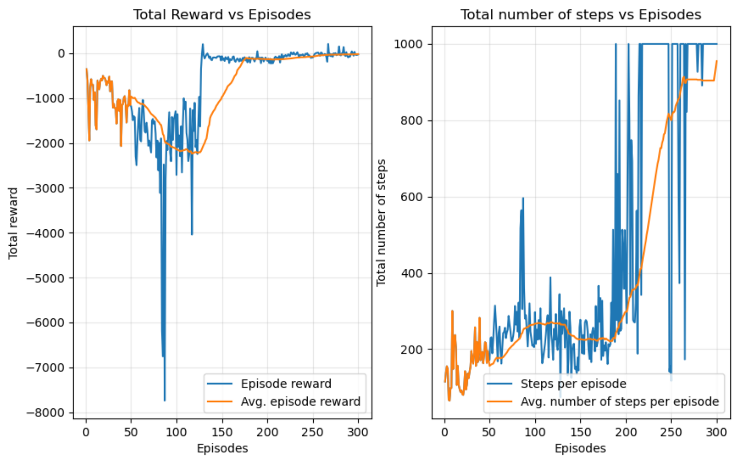
Figur 13: $\gamma = 0.8$



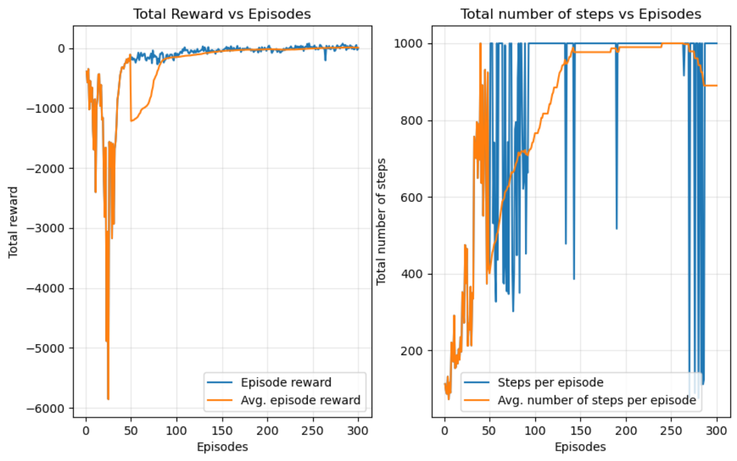
Figur 14: $\gamma = 1$

2.4.3 For your initial choice of the discount factor γ_0 investigate the effect of increasing/decreasing the memory size (show results for 2 different values of L). Does training become more/less stable? Document your findings with relevant plots.

In the experiment, we have set the memory size equals 20000 and 40000 separately. As the result shows, using a small experience replay buffer cause the training process wave significantly. One possible reason could be that small experience replay buffer only provides recently experience that is not stable.
On the other hand, big experience replay buffer stores more information from the past, which helps stable training process. However, it might slow down the learning process because agent reuse very old experience for a long time.



Figur 15: memory_size=20000

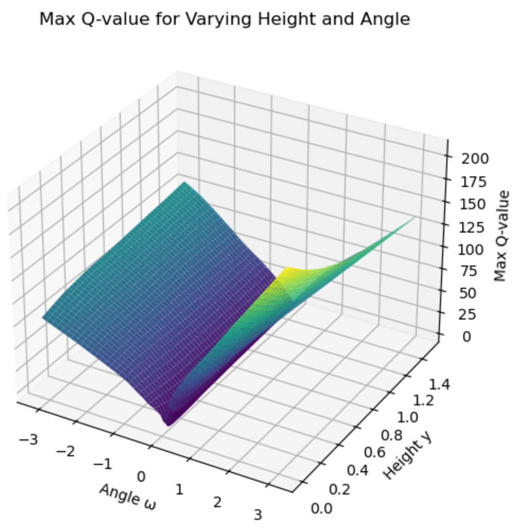


Figur 16: memory_size=40000

2.5 For the networks π_θ, Q_ω that solve the problem, generate the following two plots:

2.5.1 Consider the following restriction of the state $s(y,\omega) = (0, y, 0, 0, \omega, 0, 0, 0)$, where y is the height of the lander and ω is the angle of the lander. Plot $Q_\omega(s(y,\omega), \pi_\theta(s(y,\omega)))$ for varying $y \in [0, 1.5]$ and $\omega \in [-\pi, \pi]$. You should obtain a 3D plot. Does the value of the optimal policy you found make sense? Explain it.

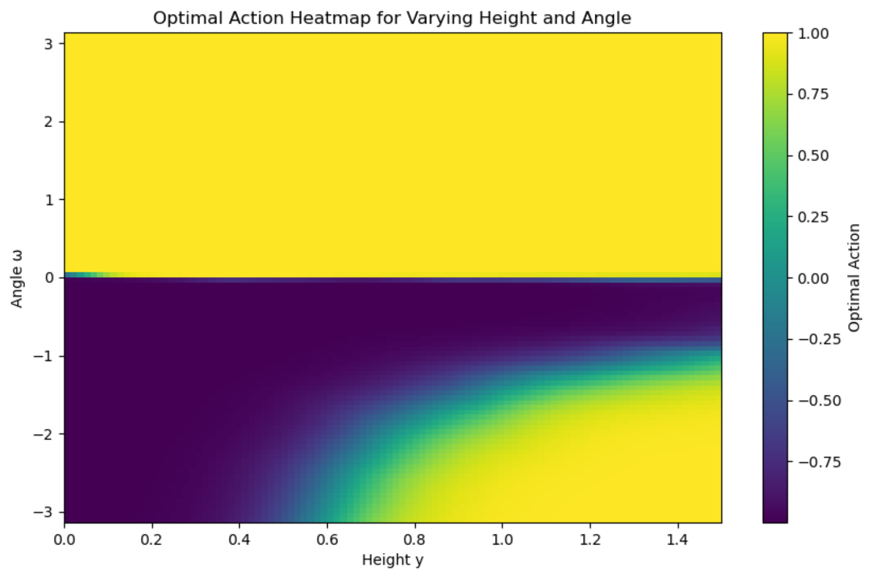
We follows the restriction of the state in the aspect of height and angle, and plot the Q value evaluated by critic network. As we can see, the Q value decreases when the angle is from the edge to the middle value, and it is symmetrical. For another aspect, the change of the height does not effect the Q value much.



Figur 17: $Q_\omega(s(y,\omega), \pi_\theta(s(y,\omega)))$ for varying $y \in [0, 1.5]$ and $\omega \in [-\pi, \pi]$

2.5.2 Let s be the same as the previous question. Remember that the action is a bi-dimensional vector, where the second element denotes the engine direction. Plot the engine direction $\pi_\theta(s(y,\omega))_2$ for varying y and ω (as in the previous question). You should obtain a 3D plot. Does the behaviour of the optimal policy make sense? Explain it.

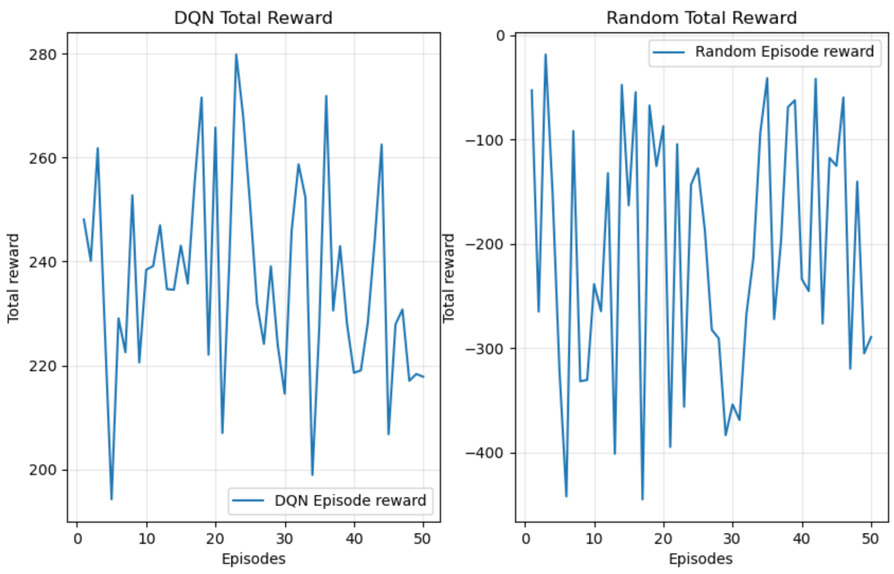
The following figure show the engine direction chosen by the policy depending on the state, which is predicted by the actor network. Also, the optimal action generally relies on the angle not the height, the engine action of direction is aiming to keep agent's angle to be 0. Therefore, it also seems like symmetrical with respect to angle.



Figur 18: $\pi_\theta(s(y,\omega))_2$ for varying y and ω

2.6 Compare the policy you found with the random agent in (a). Show the total episodic reward over 50 episodes of both agents.

The agent following the policy trained by DDPG method has higher total reward than the random agent. The data shows that we have learned a good policy that can obtain average 200 points.



Figur 19: Compare the policy you found with the random agent

3 Problem3

3.1 Some questions

3.1.1 Why don't we use the target networks in PPO?

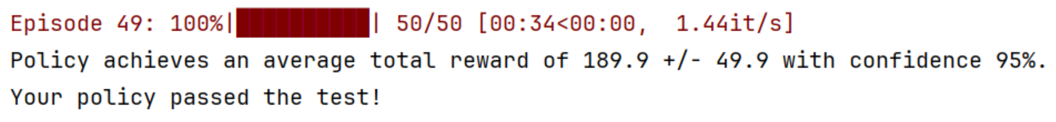
We don't need the target networks in PPO because PPO algorithm can generate the actions independently rather than according to the Q-value function. Such that, the frequency we update V-value function will not affect the convergence.

3.1.2 Is PPO an on-policy method? If yes, explain why. Furthermore, is sample complexity an issue for on-policy methods?

PPO is an on-policy method. Because we update the policy according to the data we collect from the same policy.
In general, on-policy method needs higher sample complexity than off-policy method. Because on-policy methods use the data generated from the current policy for training, so that it may limit the exploration of the environment from some aspects.

3.2 Implement the PPO algorithm (shown in Algorithm 3) and solve the problem

By running check_solution.py, we demonstrate our policy passed the test, and got an average total reward of 189.9 +/- 49.9 with confidence 95%.



```
Episode 49: 100%|██████████| 50/50 [00:34<00:00,  1.44it/s]
Policy achieves an average total reward of 189.9 +/- 49.9 with confidence 95%.
Your policy passed the test!
```

Figur 20: check the solution

3.3 Answer the following

3.3.1 Explain the layout of the network that you used; the choice of the optimizer; the parameters that you used. Motivate why you made those choices.

We design the layout of the network and choose the optimizer and parameters according to the lab instruction.
For the actor network, the first layer has 400 neurons and ReLU activation shared by two heads. And the two heads each have a hidden layer with 200 neurons using ReLU activation. The output layers use **Tanh** activation for μ , and **Sigmoid** activation for σ^2. As for the critic network, we use 400 neurons with ReLU activation for the first layer, and 200 neurons with ReLU activation for the second layer.
We choose the parameters as below. Specially, we use a smaller learning rate for actor

network updating for better convergence.
Parameters:

- N_episodes = 1600
- $\gamma = 0.99$
- actor_lr = 1e-5
- critic_lr = 1e-3
- $\varepsilon = 0.2$
- buffer_size = 20000
- epoch_number = 10

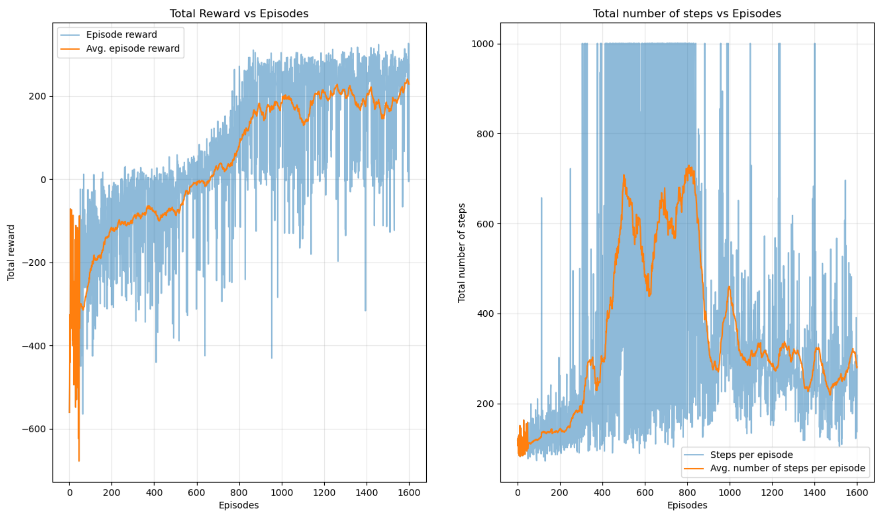
3.3.2 Do you think that updating the actor less frequently (compared to the critic) would result in a better training process (i.e., more stable)?

Yes. Updating the actor frequently maybe will prevent the critic from learning thoroughly for the environment. And this is also why we choose a smaller learning rate for actor.

3.4 Once you have solved the problem, do the following analysis:

3.4.1 Plot the total episodic reward and the total number of steps taken per episode during training. What can you say regarding the training process?

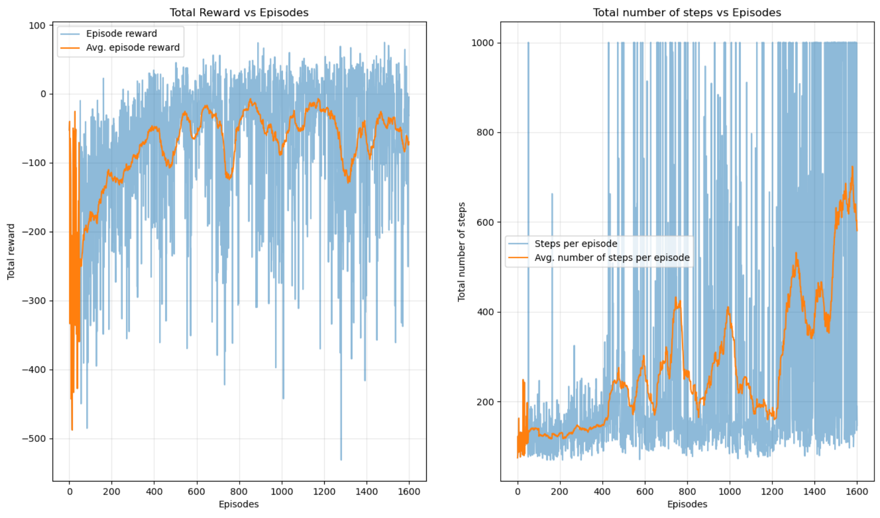
The figure shows that the agent is exploring in the first few episodes, therefore, agent got very low total reward and small steps at first, and the rewards vary a lot in this period. After about 150 episodes, the result suggests that agent have learning better policy to land, the total reward increases. And after around 900 episodes, the agent comes to converge around a optimal policy.
The total number of steps of every episode increases at first because agent have not learned a good policy and explore more space. Then, the total number of steps becomes decrease when the total rewards is positive because agent has known a good policy to land.



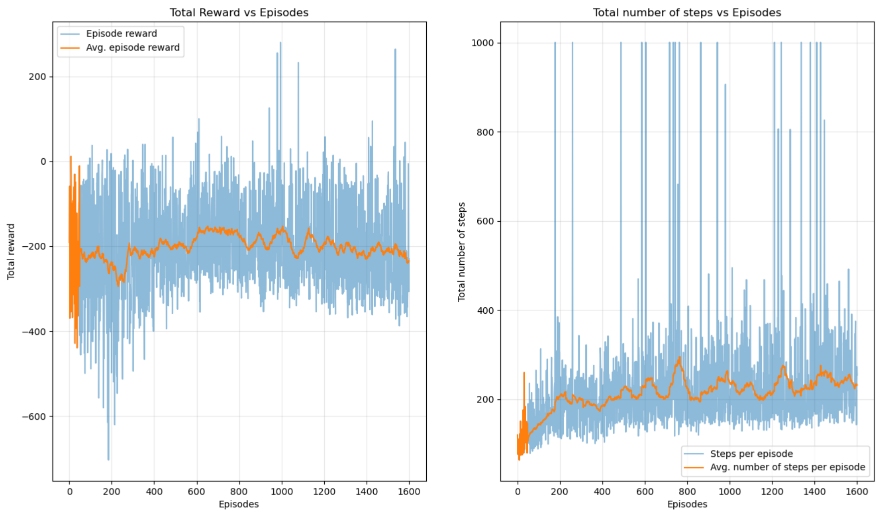
Figur 21: Reward and steps across time with default parameters

3.4.2 Let γ_0 be the discount factor you chose that solves the problem. Now choose a discount factor $\gamma_1 = 1$, and a discount factor $\gamma_2 \ll \gamma_0$. Redo the plots for γ_1 and γ_2 (don't change the other parameters): what can you say regarding the choice of the discount factor? How does it impact the training process?

As we can see from the figure, we set the $\gamma = 0.8$, the too low discount factor result in a bad policy, because the low discount factor means that the agent attach less importance to future rewards.
On the other hand, when $\gamma = 1$, it means that the agent won't learn from the past experience. So under such scenario, the policy also can't converge properly.



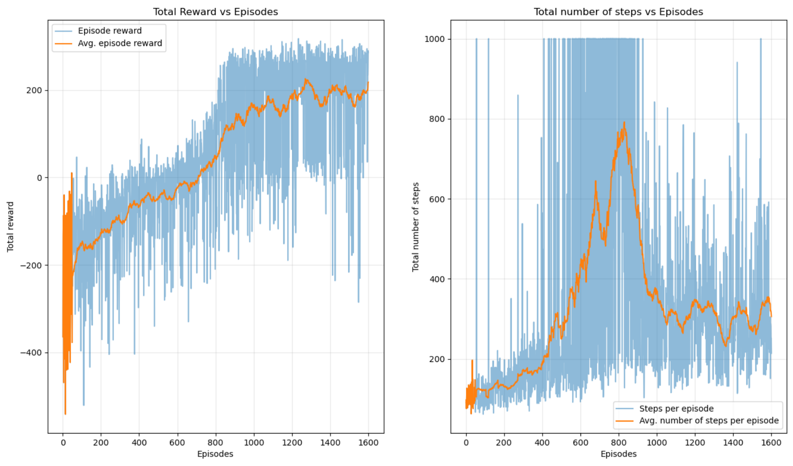
Figur 22: $\gamma_1 = 1$



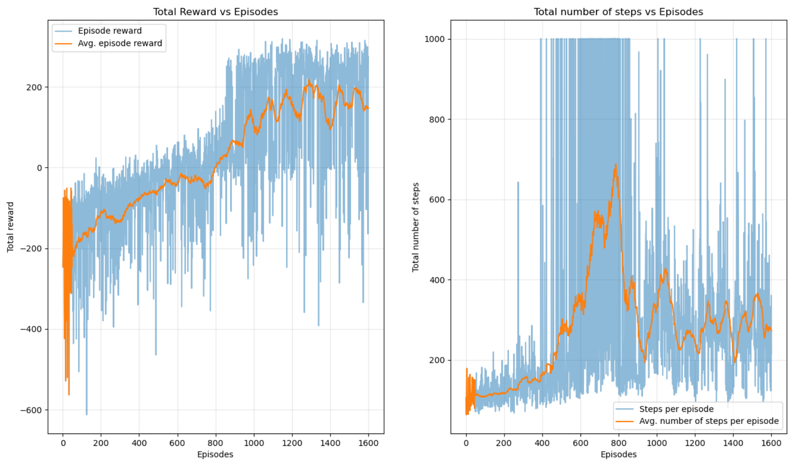
Figur 23: $\gamma_2 = 0.1$

3.4.3 For your initial choice of the discount factor γ_0 investigate the effect of increasing/decreasing ε (show results for 2 different values of ε). Does training become more/less stable? Document your findings with relevant plots.

As the figures shows below, we can tell that a smaller ε will help the training process be more stable.



Figur 24: $\varepsilon = 0.05$

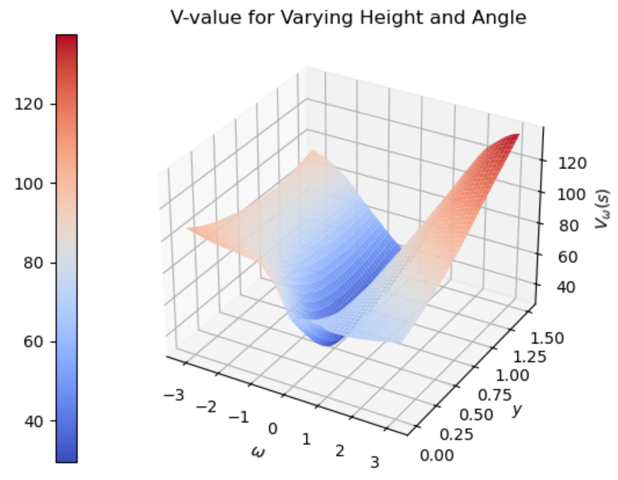


Figur 25: $\varepsilon = 0.5$

3.5 For the networks π_θ, V_ω that solve the problem, create the following two plots:

3.5.1 Consider the following restriction of the state $s(y,\omega) = (0, y, 0, 0, \omega, 0, 0, 0)$, where y is the height of the lander and ω is the angle of the lander. Plot $V_\omega(s(y,\omega))$ for varying $y \in [0, 1.5]$ and $\omega \in [-\pi, \pi]$. You should obtain a 3D plot. Does the value of the optimal policy you found make sense? Explain it.

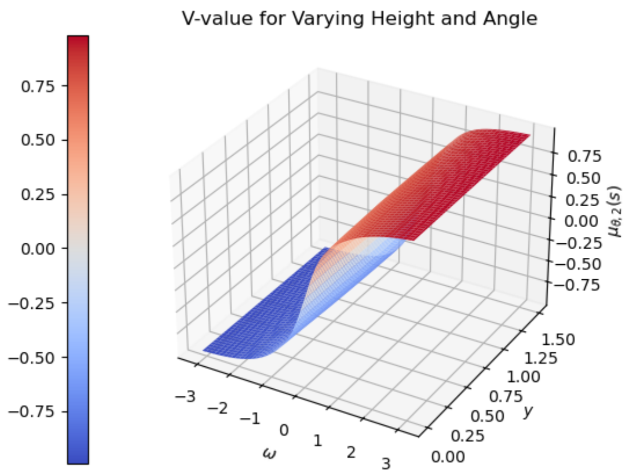
The variation of ω affects the V-value significantly. When $\omega = 0$, the V-value achieves the lowest point, which makes sense since the spacecraft should land to the landing point vertically to get higher rewards.



Figur 26: V-value function for varying y and ω

3.5.2 Let s be the same as the previous question. Remember that the action is a bi-dimensional vector, where the second element denotes the engine direction. Let the mean value of $\pi_\theta(s)$ be $\mu_\theta(s)$: plot the engine direction $\mu_\theta(s(y,\omega))_2$ for varying y and ω (as in the previous question). You should obtain a 3D plot. Does the behaviour of the optimal policy make sense? Explain it.

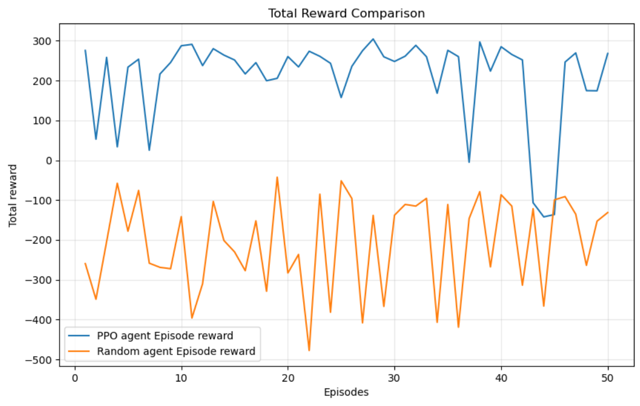
As Figure 27 shows, the policy seems to make sense because when the ω is away from 0, or to say, the spacecraft is more tilted, then the side engine should provide a bigger power to make the spacecraft back to the vertical state.When the spacecraft is vertical($\omega = 0$), the side engine do not need to provide much power.



Figur 27: Action decided for varying y and ω

3.6 Compare the policy you found with the random agent in (a). Show the total episodic reward over 50 episodes of both agents.

As Figure 28 shows, the PPO agent we trained can achieve a much higher reward than the random agent.



Figur 28: Total rewards under PPO agent and random agent