Computer Lab 2 - Deep Reinforcement Learning

Guanyu Lin 19980514-5035 guanyul@kth.se

Tawsiful Islam 20001110-2035 tawsiful@kth.se

December 22, 2023

1 Problem 1 (Mandatory): Deep Q-Networks (DQN)

(b) Why do we use a replay buffer and target network in DQN?

The Replay Buffer is employed to enhance sample efficiency and stability. By storing and randomly sampling past experiences (state-action-reward-nextstate tuples), the replay buffer provides a diverse set of training data. This helps break the temporal correlation between sequential samples, promoting stability during learning. Additionally, reusing past experiences multiple times contributes to more effective learning.

Target Network is used to stabilize the learning process. It provides a stable target for Q-value estimation during training, mitigating issues related to moving targets. To further enhance stability, a soft update mechanism is applied, gradually updating the target network parameters. This approach ensures smoother transitions and increased stability in the learning process.

(d) Network Architecture, Optimization, and Hyperparameters

(d).1 Neural Network Architecture

The neural network architecture used in our reinforcement learning implementation is defined by the `QNetwork` class. It consists of three fully connected layers (`fc1`, `fc2`, and `fc3`). The input layer (`fc1`) has `state_size` nodes, and both hidden layers (`fc2` and `fc3`) have 64 nodes. The output layer (`fc3`) has 4 nodes, representing the Q-values for each possible action. The activation function used between layers is the rectified linear unit (ReLU), applied through `F.relu` in the `forward` method

(d).2 Choice of Optimizer

The choice of optimizer in our implementation is the Adam optimizer, selected based on the recommendation of lab document. In the context of our DQNAgent class, the optimizer

Listing 1: Adam Optimizer Initialization

```
self.optimizer = torch.optim.Adam(self.q_network.parameters())
```

Here, `self.q_network.parameters()` represents the parameters of the Q-network (`QNetwork`) that the optimizer will update during the training process.

(d).3 Parameter Settings

The hyperparameters chosen for our reinforcement learning agent are as follows:

- **Memory Size:** 1×10^5 (memory_size)
- **Batch Size (N):** 64 (batch_size)
- **Discount Factor (γ):** 0.99 (gamma)
- **Soft Update Parameter (τ):** 1×10^{-3} (tau) – Note: Setting $\tau = 0$ corresponds to a hard update.
- **Learning Rate (LR):** 1×10^{-4} (LR)
- **Update Interval (C):** 4 (update_intervall) – How often to update the Q network.
- **Exploration Factor (ϵ):** 1 (eps) – Initial exploration factor for epsilon-greedy action selection.
- **Epsilon Decay Rate:** 0.999 (eps_decay_rate) – The rate at which ϵ is decayed over time.
- **Minimum Epsilon (ϵ_{min}):** 0.05 (eps_min) – The minimum value to which ϵ is decayed.

Our choice of hyperparameters is based on recommendations from the lab documentation and extensive experimentation. These parameters represent the most reasonable configuration achieved through multiple trial-and-error iterations.

(d).4 Modification

As an enhancement to the standard Deep Q-Network (DQN) algorithm, we introduced a soft update mechanism for updating the target network. The soft update is performed by iteratively updating each parameter of the target network based on a weighted average of the corresponding parameter from the Q-network and the target network.

The soft update equation is expressed as:

$$\text{New Target Parameter} = \tau \cdot \text{Q-Network Parameter} + (1 - \tau) \cdot \text{Old Target Parameter}$$

Here, τ is a hyperparameter that controls the blending ratio. A smaller τ results in a more gradual update, while setting $\tau = 1$ corresponds to a hard update.

This modification aids in stabilizing the learning process by introducing a controlled transition between the Q-network and the target network, preventing abrupt changes that might affect the training stability.

(e) Plots and analysis

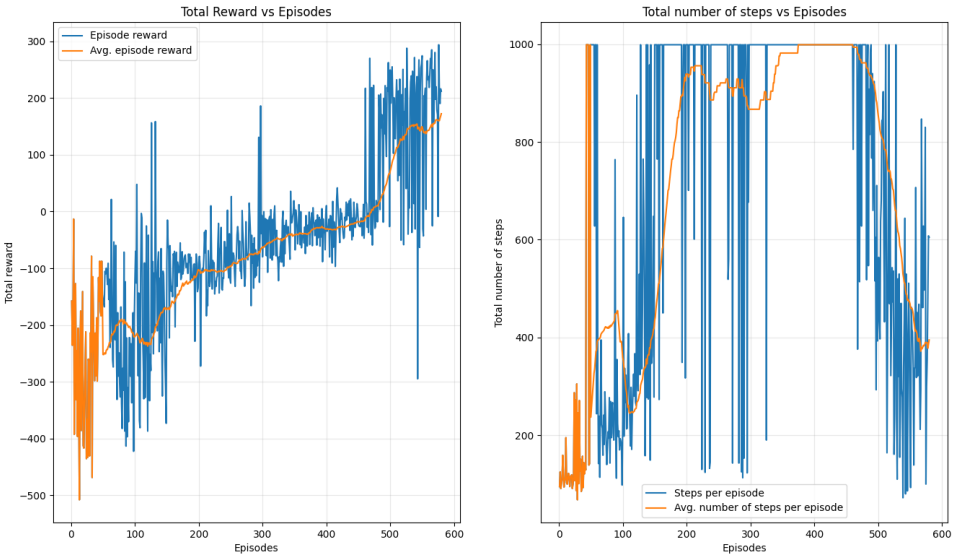


Figure 1: The total episodic reward and the total number of steps taken per episode during training with the DQN agent.

We can observe that, as the training progresses, the episodic reward gradually increases. The number of steps required for each episode is initially small. I speculate that this is because, at the beginning, the algorithm doesn't know how to control the lunar lander, causing it to crash to the ground instantly and result in termination.

After approximately 200 episodes, the lunar lander learns how to avoid free-fall crashes, but it still doesn't know how to land safely. Therefore, each episode requires the maximum number of steps, which is 1000 steps. Towards the end, the lunar lander becomes proficient at landing safely and rapidly, leading to a gradual reduction in the required number of steps.

(e).1 Changed gamma value

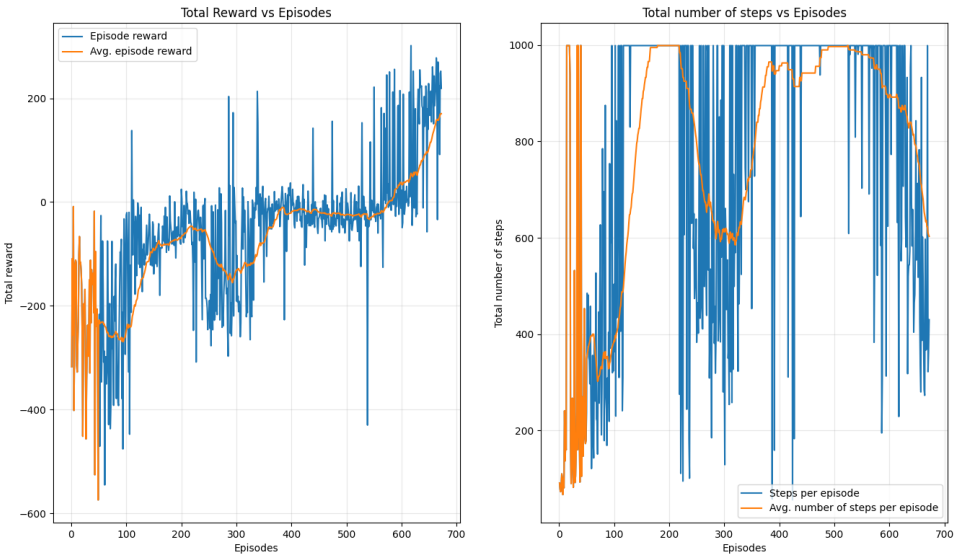


Figure 2: The total episodic reward and the total number of steps taken per episode during training for $\gamma = 1$ with the DQN agent.

We can observe that the main difference between $\gamma = 0.99$ and $\gamma = 1$ is that the episodes needed increased from 600 to about 700 episodes. Since the agent considers future rewards without discounting, the algorithm strives to maximize the long-term cumulative reward.

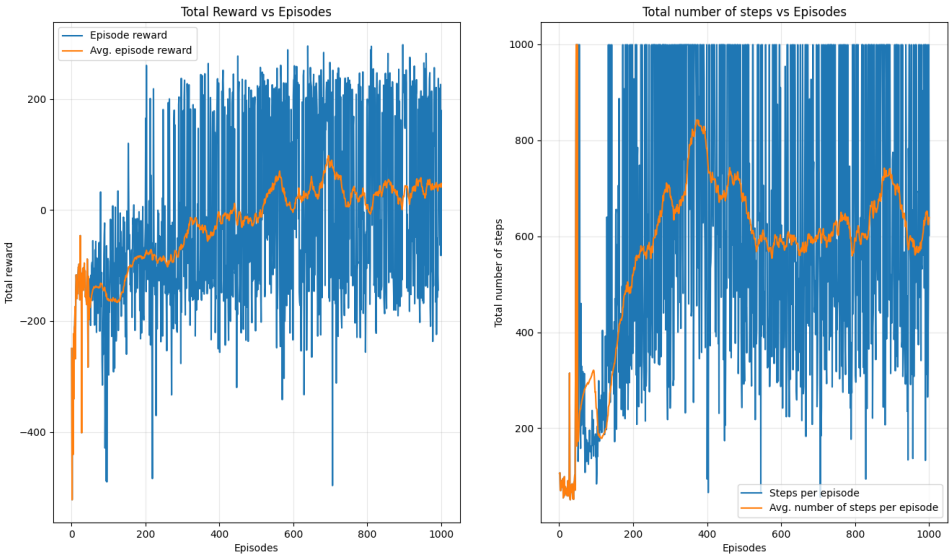


Figure 3: The total episodic reward and the total number of steps taken per episode during training for $\gamma = 0.8$ with the DQN agent.

Adjusting (γ) from 0.99 to 0.8 had significant repercussions. With a lower γ, the algorithm placed less emphasis on future rewards, compromising its ability to consider long-term consequences and potentially leading to suboptimal policies that prioritize short-term gains over sustained benefits.

The shift to a lower γ resulted in a decreased average reward per episode, signalling a deviation from optimal policies. Moreover, the steps per episode failed to reach the step limit, indicating premature episode termination. This behaviour suggests challenges in effective policy learning or exploration, emphasizing the delicate balance required in choosing hyperparameters for reinforcement learning tasks.

(f) Plots of the Q-network

(f).1 The value of the optimal policy

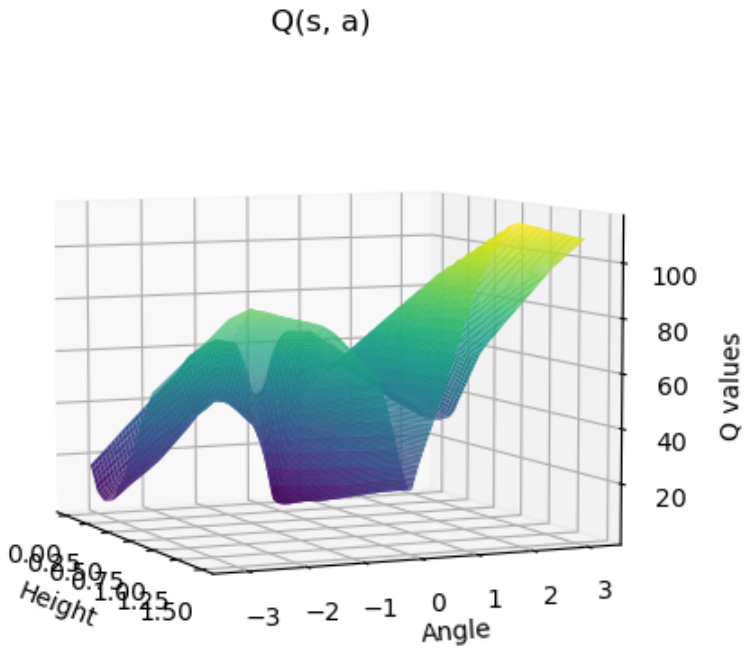
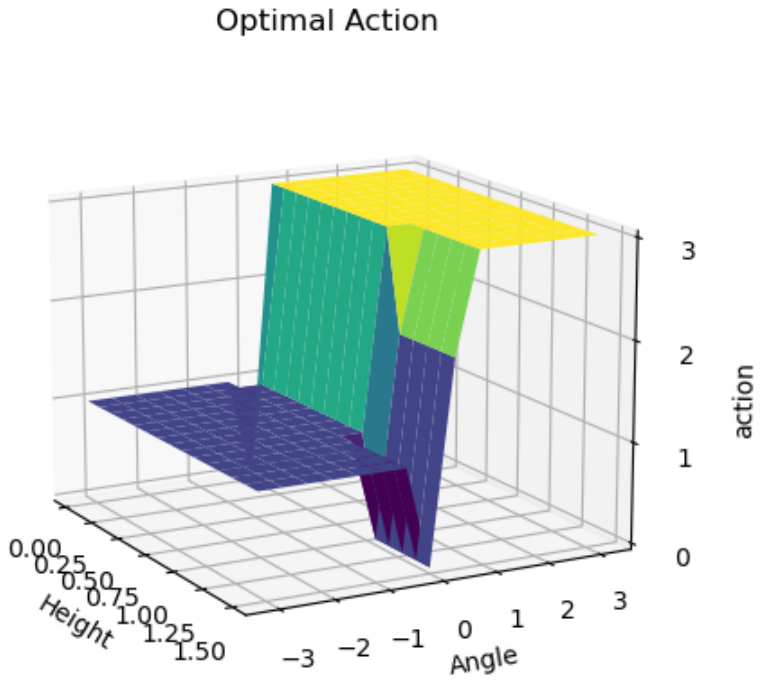


Figure 4: 3D plot of The value of the optimal policy with the DQN agent.

We can observe from the plot that Q values are high when both the height and the angle are huge, which is reasonable because the right action here gives good rewards. When the height decreases, we can see that the Q value is decreasing, because the lunar lander is almost on the ground, so the future reward will be small. However, one could argue that it should be symmetrical on both sides to equally value a larger deviation in the rotation. The results that we see could depend on how the data are generated for the buffer or our hyperparameter. Additionally, we did not achieve 200 points in rewards, so the problem wasn't fully solved according to the documentation even if it passed the tests in this lab assignment.

(f).2 The action of the optimal policy



The policy is optimal in our opinion, when the angle is positive it takes action 3 and when the angle is negative it takes action 1 to make the lunar lander flat again by doing the opposite action.

(g) Compare with random agent

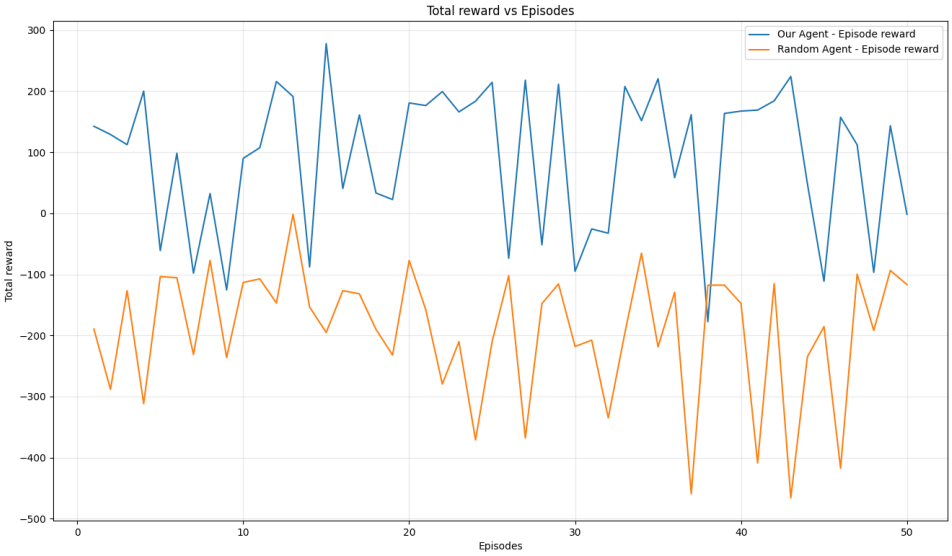


Figure 5: Comparing with random agent

By comparing with the random polity, we can say that our policy has much higher rewards per episode than the random policy. By calculation, our policy achieves an average total reward of 119.7 $\pm$ 27.8 with confidence 95%, and the random policy achieves an average total reward of -161.5 $\pm$ 30.6 with confidence 95%

2 Problem 2: Deep Deterministic Policy Gradient (DDPG)

(b) Deep networks and off-policy regarding DDPG

(b).1 Why don't we use the critic's target network when updating the actor network? Would it make a difference?

DDPG is an improvement on Q-learning and is adapted for continuous action spaces. Q-learning has the issue that it requires several iterations until the uncertainty in the estimated Q-values is small enough. To avoid this, you have two networks where the actor is updated less frequently using the TD-error from the critic network. The critic network is updated more frequently to avoid the over-estimation that happens in Q-learning. Using the critic's target to update the actor would have a different issue where the target network is updated with soft updates and slower than the main network. The result one would see is that the rewards don't reach the desired value.

(b).2 Is DDPG off-policy? Is sample complexity an issue for off-policy methods (compared to on-policy ones)?

The DDPG is an off-policy as the agent (actor network) is not updating its policy based on the data it generates. In practice, the data used to update the policy is from the random sampling from the buffer which contains states and actions from different time steps rather than the latest. Additionally, noise is added to the action that was taken from the generated data, thus it's not on-policy. On-policy methods have higher sample complexity as they require more exploration (more data samples) to obtain a good policy when one data sample is used once. With a buffer and off-policy method, the actions of the same data sample can be used multiple times, improving the policy faster than an on-policy method.

(d) Network Architecture, Optimization, and Hyperparameters

(d).1 Neural Network Architecture

The neural network architecture follows the instructions that were given. The actor network takes the states as input which goes through two fully-connected hidden layers that contain 400 respective 200 neurons with ReLU as the activation function. The output is the action with two dimensions with outputs between -1 and 1. The output has the tanh as the activation function to keep the output between -1 and 1. The critic network takes in the state and action as input. The hidden layers are similar to the actor network with the same number of neurons and activation functions. The output is one dimensional with one neuron without an activation function as we estimate the Q-value. This has been designed according to the given instructions.

(d).2 Choice of Optimizer

The suggested optimiser was the Adam optimizer which we used with the learning rate 10^{-4} and 10^{-5} for critic and actor network respectively.

(d).3 Parameter Settings

The hyperparameters chosen for our reinforcement learning agent are as follows according to the recommended parameters:

- **Memory Size:** 3×10^4 (buffer_size)
- **Batch Size (N):** 64 (batch_size)
- **Number of episodes:** 300 (N_episodes)
- **Discount Factor (γ):** 0.99 (gamma)
- **Soft Update Parameter (τ):** 1×10^{-3} (tau)

- **Learning Rate for critic:** 1×10^{-4} (criticLrate)
- **Learning Rate for actor:** 1×10^{-5} (actorLrate)
- **Update Interval (d):** 2 (d) – How often to update the Q network.
- **Noise mean (μ):** 0.15 (mu)
- **Noise deviation (σ):** 0.2 (sigma)

(d).4 Learning rate between actor and critic networks

To avoid bad estimations on the Q-value, it needs to be calculated several times and you want to converge quickly to the correct value. For that reason it's reasonable to have a higher learning rate for the critic network compared to the actor network. The actor network will be less updated and to avoid overshoots in the estimation and not catch up to the target, a smaller learning rate can avoid this problem.

(e) Plots and analysis

(e).1 Training process for main agent

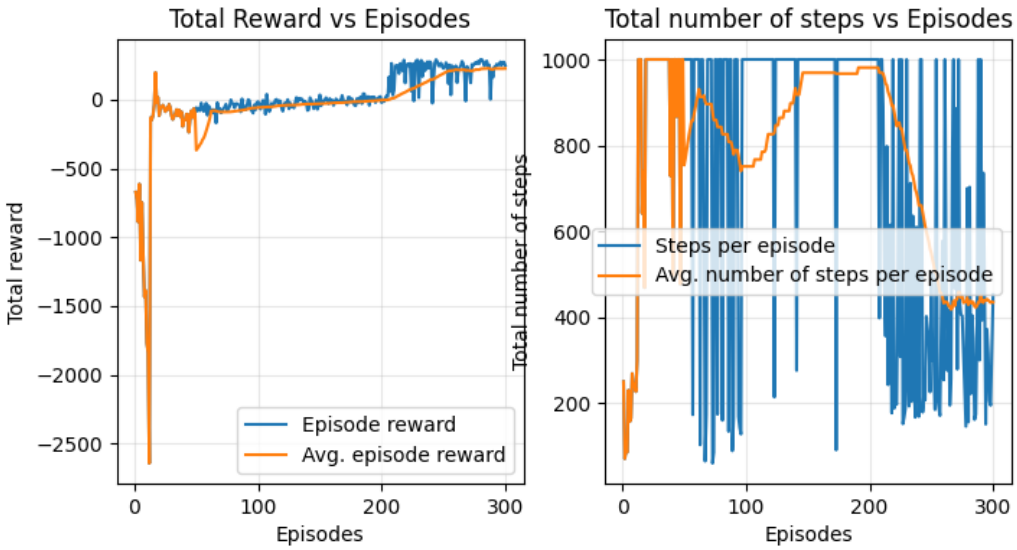


Figure 6: The total episodic reward and the total number of steps taken per episode during training for the DDPG agent.

During the training process for the DDPG agent we see that the rewards are very small in the beginning when it's reliant on the random actions by the random agent from the buffer

and takes few steps to avoid taking wrong actions. Finally, when the rewards improved, the agent focused on taking more steps to perfect the landing and could at and reduce the number of steps until it had perfected the control. The reductions in step can be seen in figure 6 after episode 200 where the rewards are at the highest and the number of steps decreases drastically.

(e).2 Changed gamma values

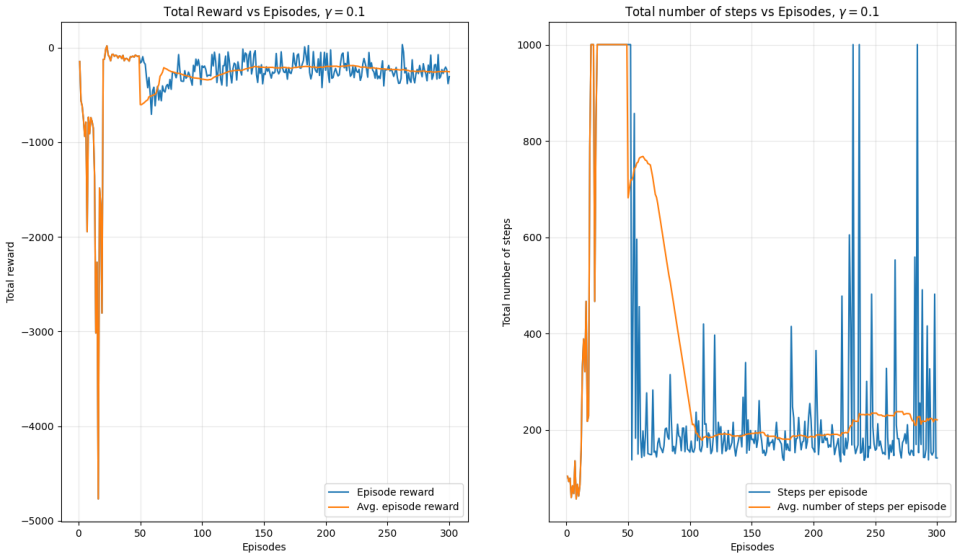


Figure 7: The total episodic reward and the total number of steps taken per episode during training for the DDPG agent with $\gamma = 0.1$.

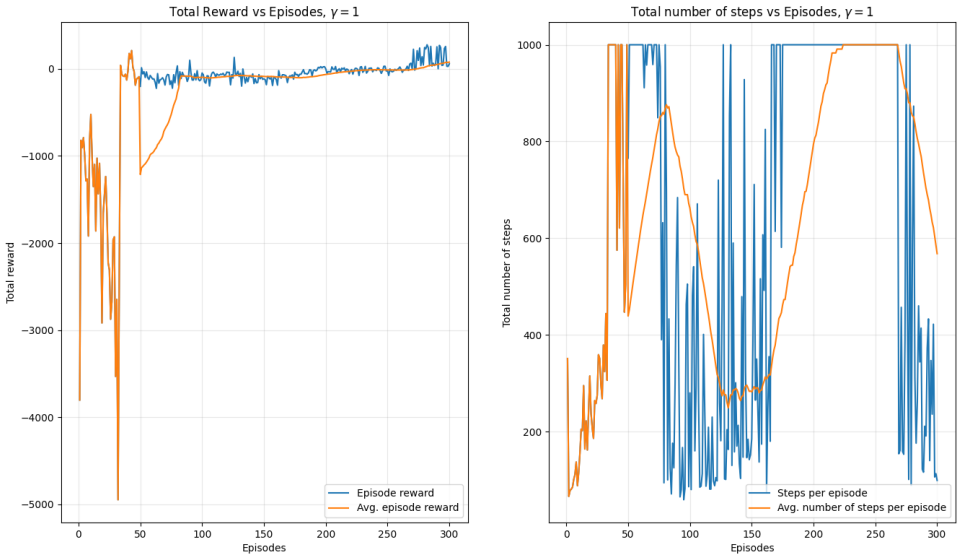


Figure 8: The total episodic reward and the total number of steps taken per episode during training for the DDPG agent with $\gamma = 1$.

Two values of γ as discount factor were used, 0.1 and 1. When comparing the training process for $\gamma = 0.99$, we see that for $\gamma = 1$ the the rewards are more stable at the end, but the steps seem to decrease later. This suggests that it takes longer time for the model to converge to its best possible performance. Having a lower discount factor converges faster as it's seen in figure 7 that the number of steps goes to 200 already at episode 100. However, it's clearly seen that the rewards are not positive compared to the rewards seen in figure 6 and 8. Having a lower discount factor means that future rewards are less important and prioritising to land the Luna lander correctly is not properly for in the training. The difference in the training process with and without a discount factor is minimal and works well in this case when we have a terminal state, but a discount factor seems to help to achieve some better performance from the model with higher rewards.

(e).3 Changed memory size

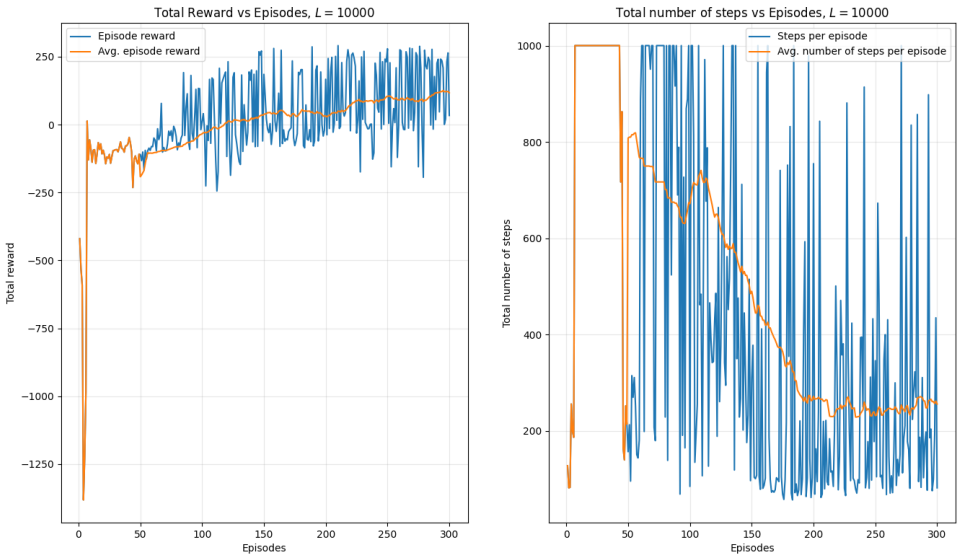


Figure 9: The total episodic reward and the total number of steps taken per episode during training for the DDPG agent with different buffer size $L = 10000$.

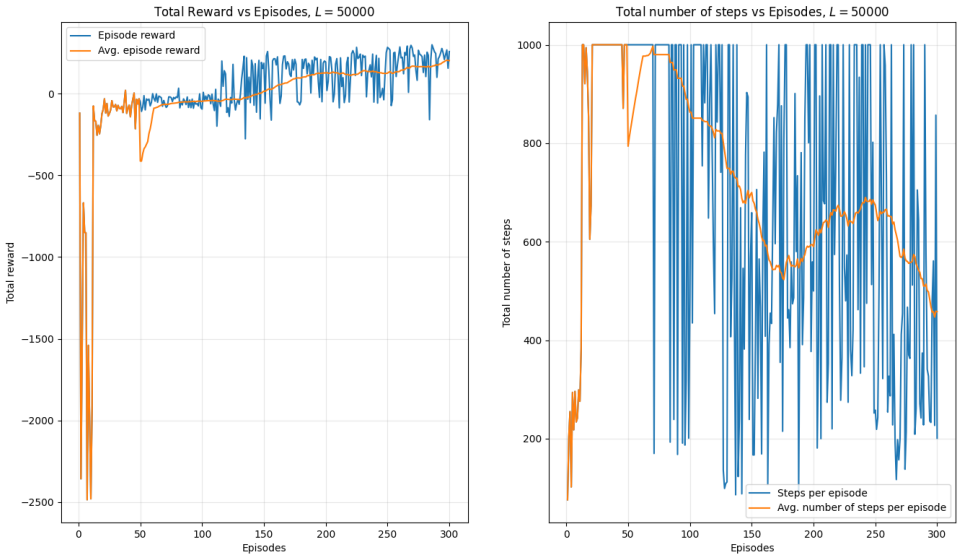


Figure 10: The total episodic reward and the total number of steps taken per episode during training for the DDPG agent with different buffer size $L = 50000$.

When changing the buffer or memory size we see that the training is more stable when having a larger buffer size compared to a smaller buffer size. This is because the data samples are less likely to correlate which helps to improve the model's robustness. The downside of having a larger buffer size is that the training takes longer as we can see that the number of steps required to land the Luna lander decreases and converges faster when having $L = 10000$ as seen in figure 9 compared to figure 10. In figure 10 the number of steps goes down to 600 at around episode 150 but then goes up before going down again to 450 steps. Note also that the number of steps is higher than 220 steps that were achieved with $L = 10000$.

(f) Plots of the network's Q-values and policy

(f).1 The plot for Q-values

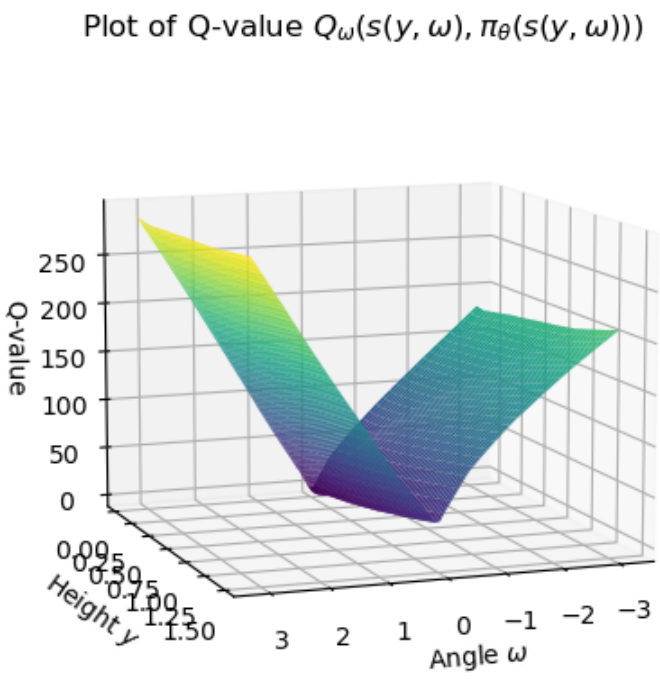


Figure 11: The highest Q-values that are achieved depending on height y and angle ω in state $s = (0, y, 0, 0, \omega, 0, 0, 0)$ for the DDPG agent.

When plotting the Q-values for the states $s = (0, y, 0, 0, \omega, 0, 0, 0)$ with $y \in [0, 1.5]$ and $\omega \in [-\pi, \pi]$ we the plot seen in figure 11. We see that the Q-values are almost symmetric around $\omega = 0$ and are high at the end and lower when the lander is more centered. This suggests that correction, when the angle is deviated, is more rewarded than when the lander is stabilised. This is also reasonable in the sense that we can control which engine to fire but not the power from the engine. If you fire when you are almost stabilised, then you will overshoot and become destabilised again. The figure also suggests that the height is not a significant factor but there is a higher reward to correct small deviation when it's high up rather than when the lander is closer to the ground.

(f).2 The plot for best action policy

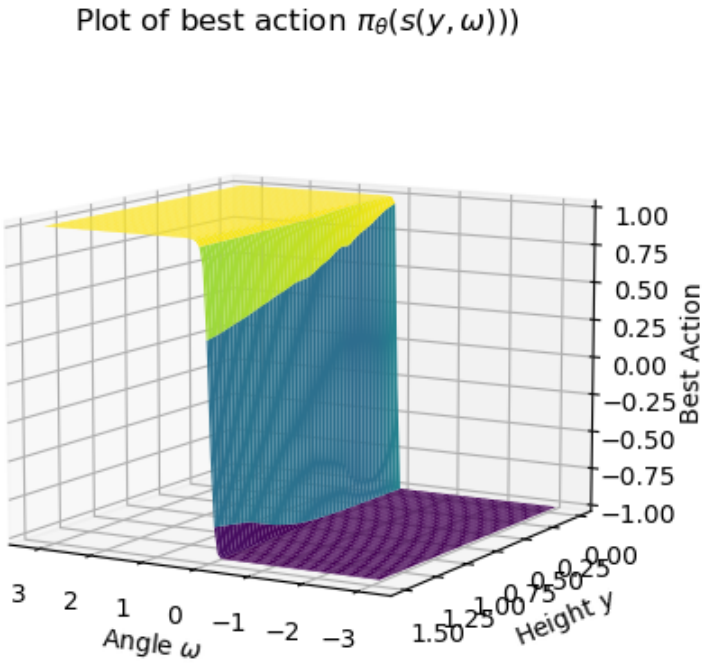


Figure 12: The best policy action that is chosen depending on height y and angle ω in state $s = (0, y, 0, 0, \omega, 0, 0, 0)$ for the DDPG agent.

When plotting the action policy for the model, we see that the actions are either full counterclockwise or clockwise rotation depending on its rotation. The actions are also independent of the height mainly because we plot the lateral engines. It can be discussed why it fires full power even at smaller angles. It could fully depend on that this policy is what the agent found to be the best solution.

(g) Comparison between random agent and DDPG agent

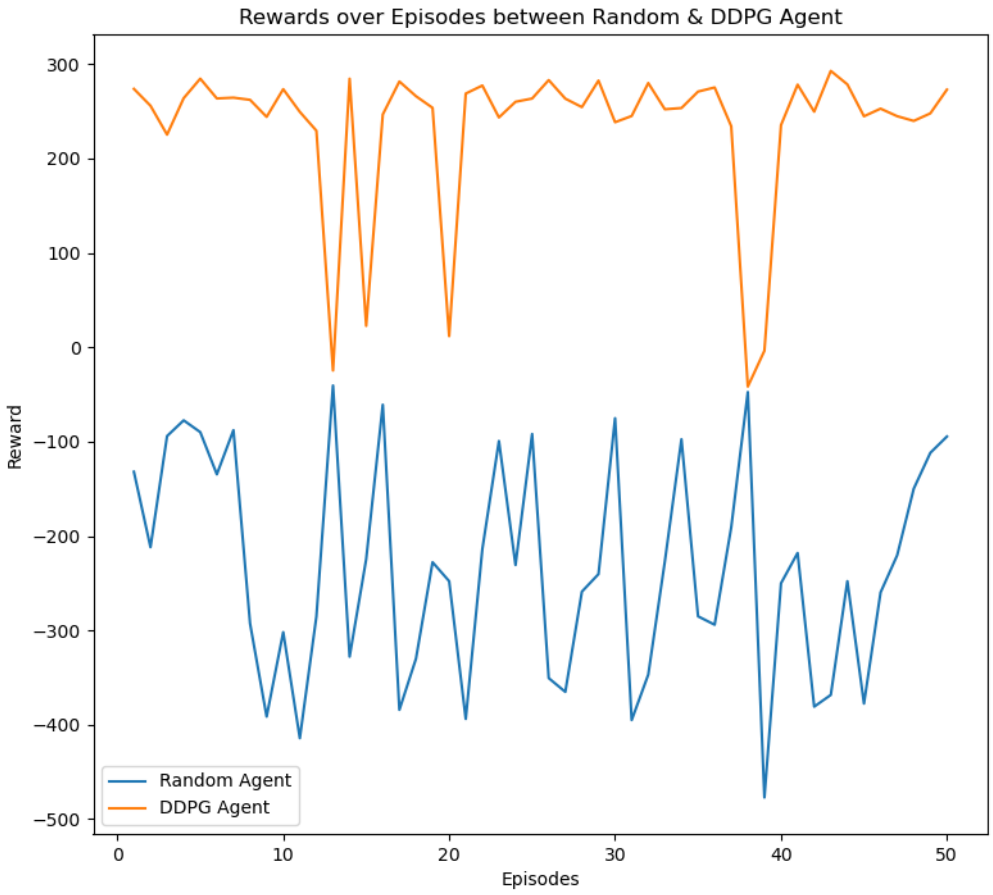


Figure 13: The total episodic reward during training for the DDPG agent in comparison to the random agent.

Figure 13 shows the difference in episodic rewards for the DDPG agent compared to the agent that makes random actions. The average reward that is achieved with the DDPG agent is 225.2 ± 26.0 with confidence 95% and the random agent has -161 ± 30.6 with 95% confidence.

3 Problem 3: Proximal Policy Optimization

(b) Target network and on-policy regarding PPO

(b).1 Why don't we use target networks in PPO?

PPO (Proximal Policy Optimization) doesn't use target networks, unlike DQN, as it operates in a policy optimization framework. Target networks, which are used in DQN, stabilize training by providing delayed and slowly changing targets for Q-value predictions. In PPO, stability is achieved through a clipped surrogate objective and trust region optimization, without the need for target networks.

(b).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, updating its policy based on current data. On-policy methods, including PPO, may face sample complexity issues, requiring more interactions with the environment for learning. This is because they rely on collecting new samples for each policy update, potentially making them less sample-efficient compared to off-policy methods.

(d) Network Architecture, Optimization, and Hyperparameters

(d).1 Neural Network Architecture

(d).2 Choice of Optimizer

The suggested optimiser was the Adam optimizer which we used with the learning rate 5×10^{-4} and 10^{-5} for critic and actor network respectively.

(d).3 Parameter Settings

The hyperparameters chosen for our reinforcement learning agent are as follows according to the recommended parameters:

- **Memory Size:** 3×10^4 (buffer_size)
- **Number of episodes:** 1600 (N_episodes)
- **Discount Factor (γ):** 0.99 (gamma)
- **Epochs (M):** 10 (M_epochs)
- **Learning Rate for critic:** 5×10^{-4} (criticLrate)
- **Learning Rate for actor:** 1×10^{-5} (actorLrate)
- **Clipping (ϵ):** 0.2 (epsilon)

(e) Plots and analysis

(e).1 Training process for main agent

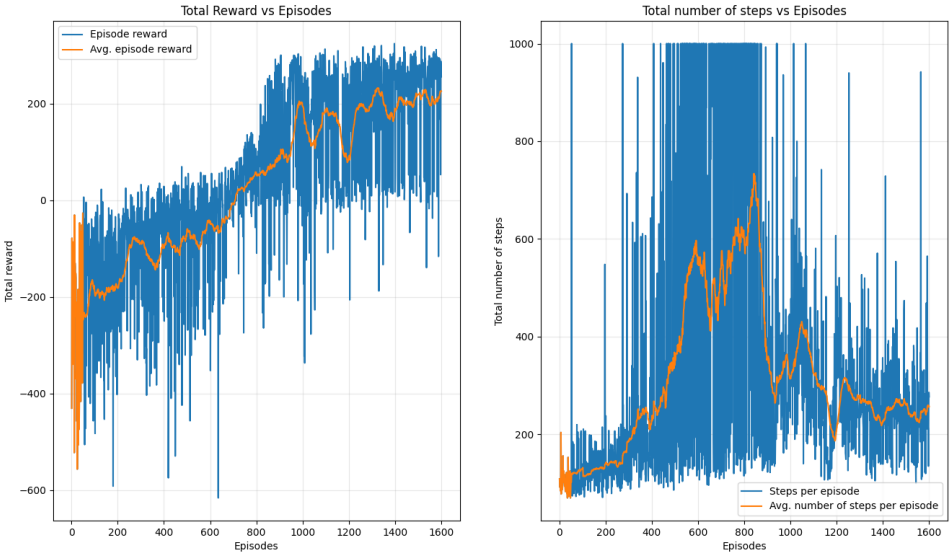


Figure 14: The total episodic reward and the total number of steps taken per episode during training for the PPO agent.

As we can see in the figure 14, the episode reward is very small at the beginning and increases as the training progresses. The total number of steps is very small at the beginning as well, because the agent is terminated very soon since it does not know how to control. Then after around 400 episodes, the agent found a way to avoid crashing so every episode took the maximum steps to learn how to control. Finally, the agent found a way to land after around 1000 episodes, so in the majority of episodes, the agent can land in a small total of steps. Compared to DDPG the rewards vary a lot which shows the training is more unstable with the PPO algorithm. However, looking at the step, the algorithm is faster as it uses few steps in the beginning and the end. This result in the training could also depend on that the data generation is done with the PPO agent instead of random actions.

(e).2 Changed gamma values

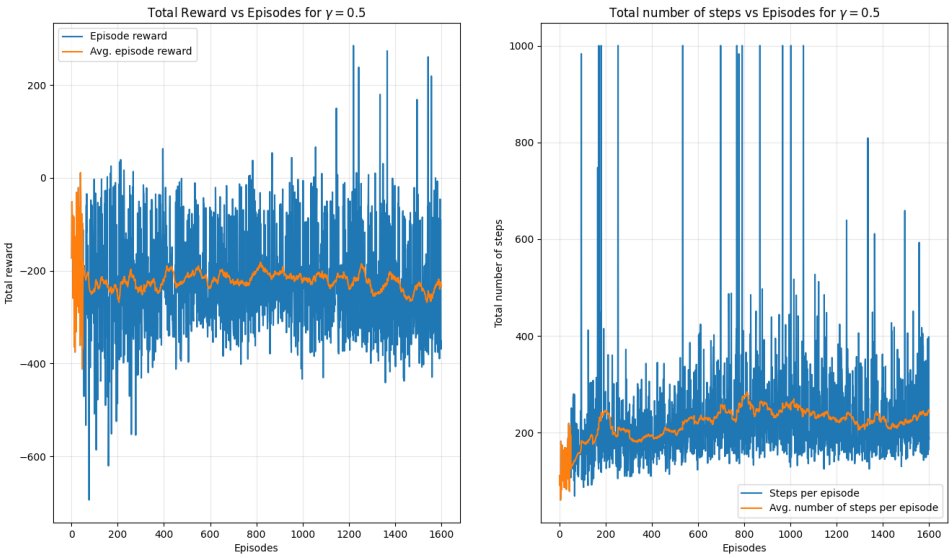


Figure 15: The total episodic reward and the total number of steps taken per episode during training for the PPO agent with $\gamma = 0.5$.

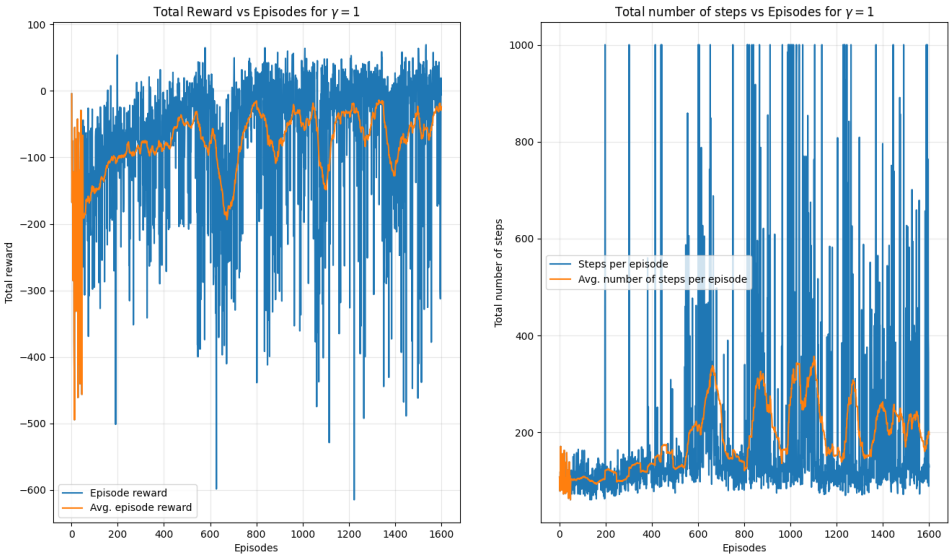


Figure 16: The total episodic reward and the total number of steps taken per episode during training for the PPO agent with $\gamma = 1$.

What we see in the figures 15 and 16 is similar to what was concluded with DDPG. Low gamma makes it difficult to achieve the desired rewards and converges fast. However, with no discount factor, achieving the desired average rewards was difficult, but it was closer to the positive values. One part of the reason is the few numbers of steps it took which gave minimum time and samples to learn how to control. The conclusion that can be drawn is that a high discount factor $\gamma < 1$ is needed for good performance.

(e).3 Changed epsilon values

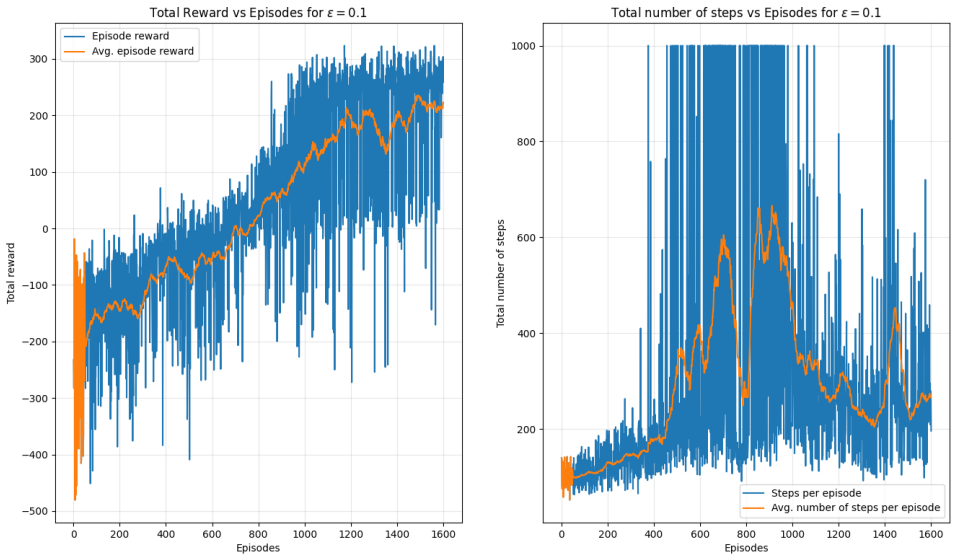


Figure 17: The total episodic reward and the total number of steps taken per episode during training for the PPO agent with $\epsilon = 0.1$.

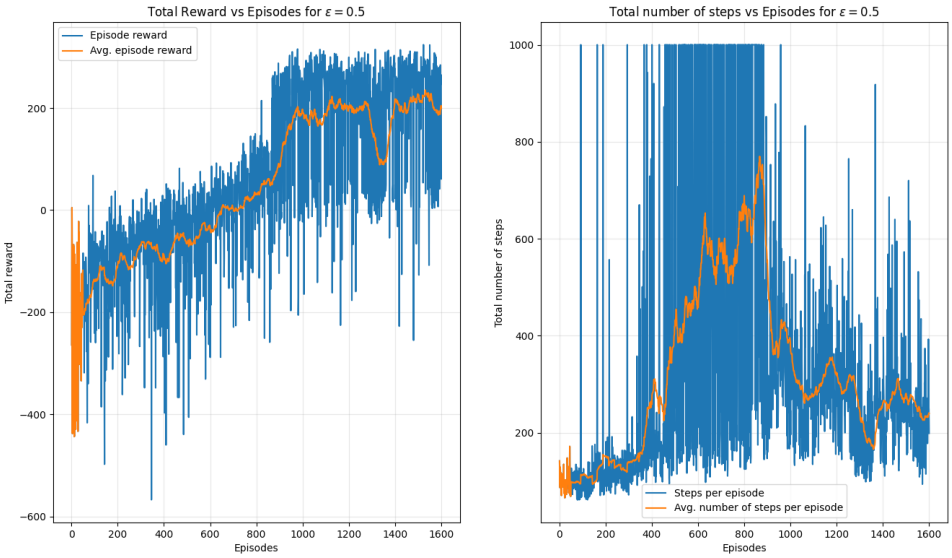


Figure 18: The total episodic reward and the total number of steps taken per episode during training for the PPO agent with $\epsilon = 0.5$.

When comparing figure 17 against 18, more oscillations are seen when having higher ϵ. The training is more stable with lower ϵ but it's also seen in the plot with the steps that the number of steps reduces quicker for $\epsilon = 0.5$. The learning is faster with higher ϵ where there are higher chances to explore more, but a lower value makes slower and safer updates.

(f) Plots of the network's V-values and policy

(f).1 The plot for V-values

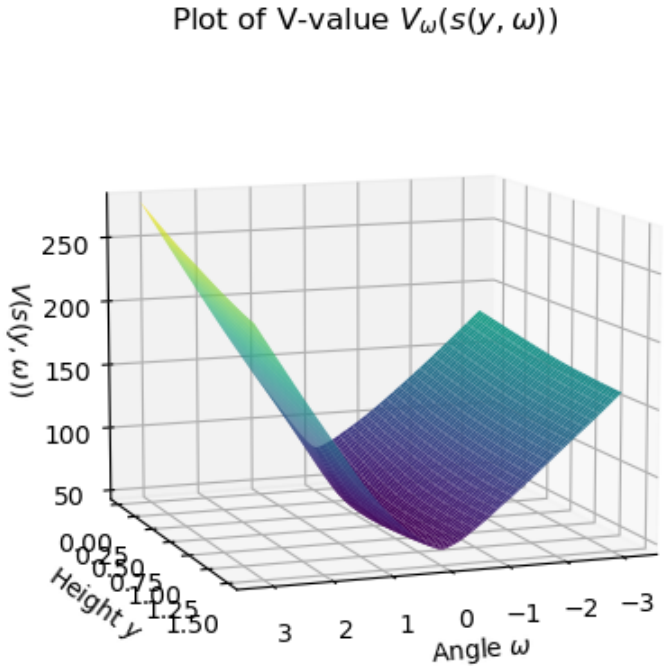


Figure 19: The highest V-values that are achieved depending on height y and angle ω in state $s = (0, y, 0, 0, \omega, 0, 0, 0)$ for the PPO agent.

We observed a similar result as in the Problem 2, that the V-values plot is symmetric, which means that it has a higher reward on the edge when the angle is far away from zero to encourage stabilisation. Compared to figure 11 for the DDPG, figure 19 shows that the rewards are less dependent on the height as it seems to mostly keep the same values at $\omega = 0$. Depending on the goal, this could be something one could reflect where controls at the last minute at small y might not be desired.

(f).2 The plot for best action policy

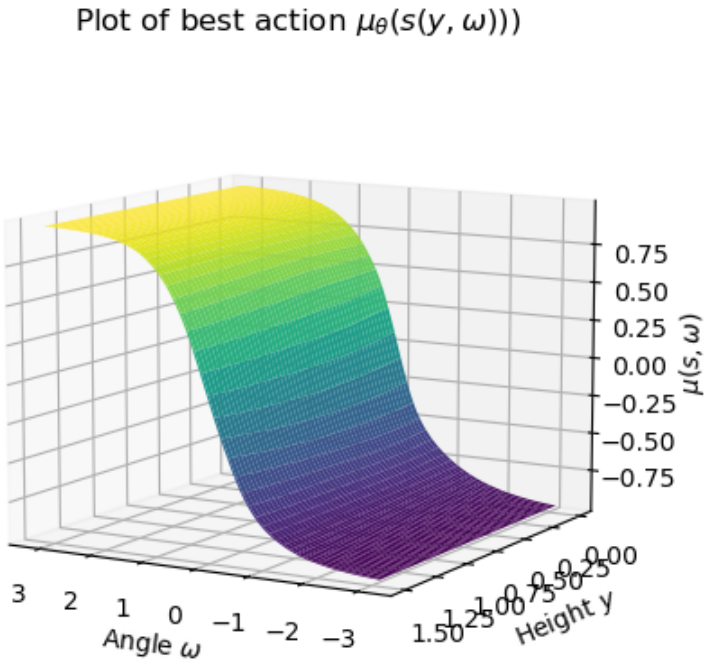


Figure 20: The best policy action that is chosen depending on height y and angle ω in state $s = (0, y, 0, 0, \omega, 0, 0, 0)$ for the PPO agent.

Once again, we see a similar result as in problem 2, we see that the actions depend on its rotational direction. The larger its rotation is, the larger the action. The actions are also independent of the height. Here we have smoother action, which could be desired as you might not want to make large control outputs μ_θ to stabilise the Luna lander when you are closer to $\omega = 0$. This is a reasonable result, and also that it's the mean of the multidimensional π_θ where you have the direction of the engine also accounted for which is continous.

(g) Comparison between random agent and PPO agent

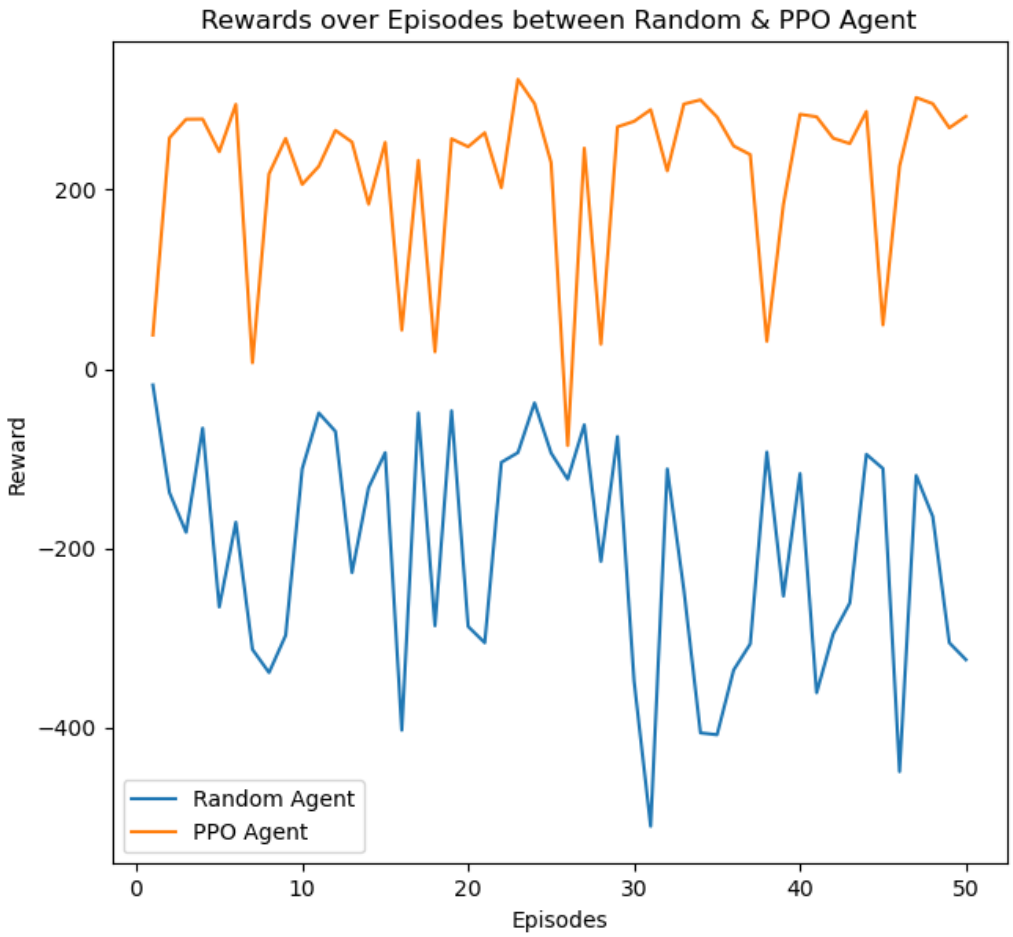


Figure 21: The total episodic reward during training for the PPO agent in comparison to the random agent.

Figure 21 shows the difference in episodic rewards for the PPO agent compared to the agent that makes random actions. The average reward that is achieved with the DDPG agent is 239.9 ± 25.2 with confidence 95% and the random agent has -161 $\pm$ 30.6 with 95% confidence.