EL2805: REINFORCEMENT LEARNING LAB2

Author
Adel Abdelsamed 2000321-T455
Alexander Weers 19990711-T455

Contents

1 Problem1: Deep Q-Networks (DQN)

1.1 Task b: Theoretical Questions

Experience replays are used as a mean to decorrelate the training data of the Q-function. In off-policy learning, successive updates resulting from following the behaviour policy are strongly correlated, which affects the convergence rate. By maintaining an experience replay buffer with fixed size L, a batch of N training samples can be sampled, such that temporal correlation is insignificant.

When the Q-function is being estimated using SGD, the following parameters update is done

$$\theta = \theta + (r_t + \gamma Q_\theta(s_{t+1}, a_t) - Q_\theta(s_t, a_t)) \cdot \nabla_\theta Q_\theta(s_t, a_t). \tag{1}$$

Since both the target and the estimated Q-function depend on θ, this update will render the target as well as the estimated Q-function non-stationary. A Target Network is introduced as a copy of the main network. It is used to calculate the TD target for every transition in the replay buffer. The use of a Target Network helps to stabilize the training process. This is because, in DQN, the TD error (or the loss) is calculated as the difference between the TD target (Q-Target) and the current Q-value (estimation of Q) as seen in the equation above. If the same network is used to calculate both these values, it can lead to a moving target problem, where the network is constantly trying to catch up with the changing values. By using a separate Target Network to calculate the TD target, this issue is mitigated, leading to more stable training. Hence, the target network is fixed for C successive steps and periodically update the parameters of the target network ϕ every C steps.

$$\theta = \theta + (r_t + \gamma Q_\phi(s_{t+1}, a_t) - Q_\theta(s_t, a_t)) \cdot \nabla_\theta Q_\theta(s_t, a_t). \tag{2}$$

The experience replay buffer's size L, the batch size of the training samples N as well as the frequency of the target update C are further treated as additional hyperparameters.

1.2 Task c and d: Training Base Dueling DQN Agent

Structure of the neural network for dueling DQN

The selected feed-forward neural network consists of two layers, an input layer with eight inputs (one for each state dimension) and a hidden layer consisting of 100 neurons. Then two separate streams are connected to the 100 neurons in the hidden layer; the advantage function stream and the value function stream. The advantage function stream A(s,a) outputs four values (one for each action), while the value function stream V(s) outputs a scalar value. The output of both streams are then combined using the following equation

$$Q(s, a) = V(s) + A(s, a) - \sum_{i=1}^{N_{act}} \frac{1}{N_{act}} A(s, i). \tag{3}$$

The addition of the mean over all the actions of the advantage function is utilized to renders the mapping identifiable. All layers except the output layers of the two streams utilized ReLU-functions as activation functions. Only one hidden layer was chosen at first, to reduce the complexity of the network, which yielded satisfying results. The number of neurons was

varied from 32 to 128 and best performance was found using 100 neurons. Finally, the network was modified to the dueling-DQN architecture, which proved to enhance training stability.

Training Parameters

The learning rate α was set to 1e-4 and the number of episodes T_E to 400 after iterative fine-tuning. The small learning rate aided in the stability of the training process and reduced oscillations. Furthermore, a buffer size L was initially set to 10000 and the batch size N was varied within the range 30 - 128. Best performance was found with a batch size of 96 samples. The frequency of the target network update was set to approximately L/N as recommended in the instructions. Hence the target network was updated every 104 steps.

The exponentially decaying exploration rate in combination with the dueling-DQN architecture was found to have the best results, despite taking longer to train. The decay rate remained between $\epsilon_{\max} = 0.99$ and $\epsilon_{\min} = 0.05$ to ensure a thorough exploration during early training stages and a goal-directed behavior during later stages. The epsilon decay horizon Z was chosen to be 90% of the maximum 300 episodes.

Problem-dependent Parameters

Lastly, the discount factor γ was also fine-tuned iteratively. The discount factor was varied from 0.8 to 0.99 in 5 steps. Best performances w.r.t training stability, average rewards and average steps taken were observed for $\gamma = 0.99$. In general a smaller γ violated the 1,000 steps per episode limit. Increasing it, however, lead to a more goal-focused behavior. We hypothesize that a lower value for γ weights the immediate negative reward of firing the engines too high relative to the expected discounted positive reward of a successful landing.

A summary of the chosen hyperparameters is presented in Table 1.

Table 1: Hyperparameters for the dueling DQN network trained in Task c.

Discount factor	γ	0.99
Buffer size	L	10.000
Number of episodes	T_E	400
Target network update frequency	C	104
Training batch size	N	96
Learning rate	α	$1 \cdot 10^{-4}$
Minimum epsilon	$\epsilon_{\min}$	0.05
Maximum epsilon	$\epsilon_{\max}$	0.99
Epsilon decay horizon	Z	360
Exploration rate	ϵ_k	$\max(\epsilon_{\min}, \epsilon_{\max} \left(\frac{\epsilon_{\min}}{\epsilon_{\max}}\right)^{(k-1)/(Z-1)}$

1.3 Task e: Hyperparameter Analysis

The total episodic rewards and the steps taken for the base network is shown in Figure 1. Initially the rewards are random as the agent is taking random actions. The agent then tries to avoid crashing by not landing at all. This is also reflected from the number of steps, which exceed 1000 steps. Around 200 episodes the start of a decreasing trend can be seen in the number of total steps per episode. This starting point coincides with the rewards increasing suddenly. Over the remaining episodes the total episodic rewards continue to increase, while the number of steps decrease as the agent learns to land much earlier.

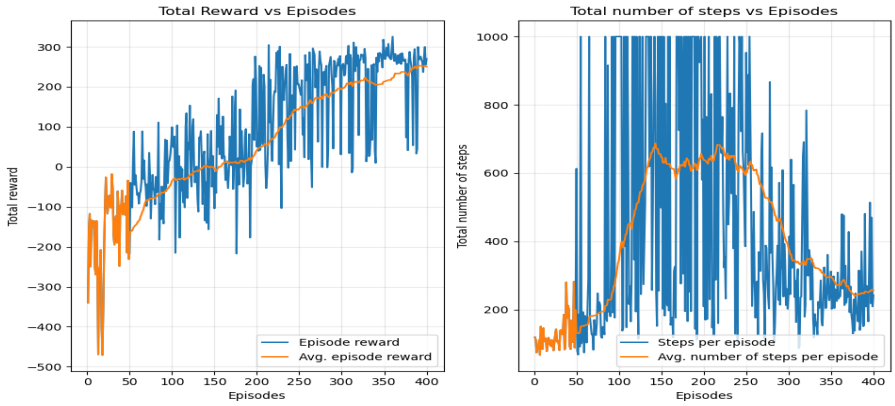

Figure 1: Training Process of the base Q-network for the Lunar Lander environment using the parameters in 1. In orange the running average over the last 50 episodes is shown.

Next, the effect of the discount factor γ is analyzed. The effect of the varying discount factors are shown in Figure 2. All other hyperparameters are unchanged. The choice of the discount factor is crucial for the training stability. Setting γ to 1, leads to the agent not learning at all compared to the base model. This choice of γ does not discount any future rewards and hence prioritizes the rewards that are very far in the future the same as immediate rewards. This clearly has a negative effect on the training process. In contrast, setting $\gamma = 0.4$ prioritizes short term rewards more, making the agent more short-sighted. This affects the learning process drastically and appears to be slowing it down, possibly also not learning at all.

Next, the influence of the episodes number T_E is investigated. All other parameters remain unchanged w.r.t the base model. The increased number of episodes proves to show the stability of the training process under the chosen hyperparameters. Figure 3 shows the episodic rewards and the number of steps taken in an episode for two distinct number of episodes. A rapid increase in the episodic reward can be seen until episode 400, which coincides with the plot for $T_E = 400$. After 400 episodes, the total episodic rewards plateaus at approximately 200. The additional 400 episodes was found to have a minimal positive effect on both the total reward and the number of steps in an episode. Hence, training the neural network can be reduced to 400 episodes.

Lastly, we investigate the effect of the memory size. Figure 4 shows the episodic rewards and the number of steps taken in an episode for three distinct number of episodes. Increasing the experience replay buffer size allows the the training process to sample from more possible trajectories, however, the data might not be as current. Increasing the buffer size to 20000 exhibits a very similar training as to the baseline model. However, increasing the buffer size to 30000 slows down the training process in a particular sense. The network requires more episodes until a positive trend is detected. This also conforms with the intuition, since an increased memory under the same training batch size will need more episodes to train on fresh data.

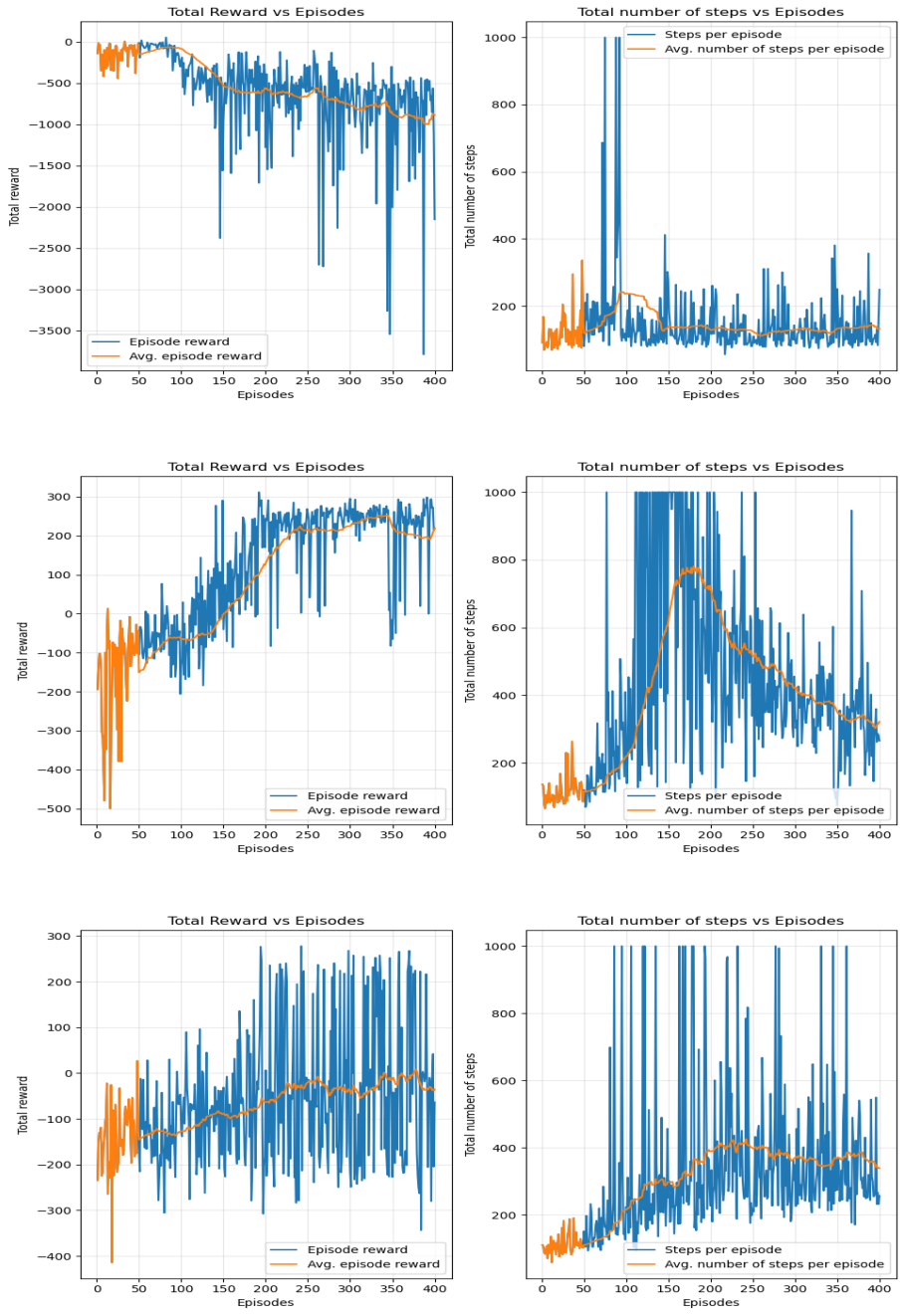

Figure 2: Effect of varying discount factors on the training process. Training process is shown (up) for $\gamma_1 = 1$, (middle) for γ_0 and (bottom) for $\gamma_2 = 0.4$. In orange the running average over the last 50 episodes is shown.

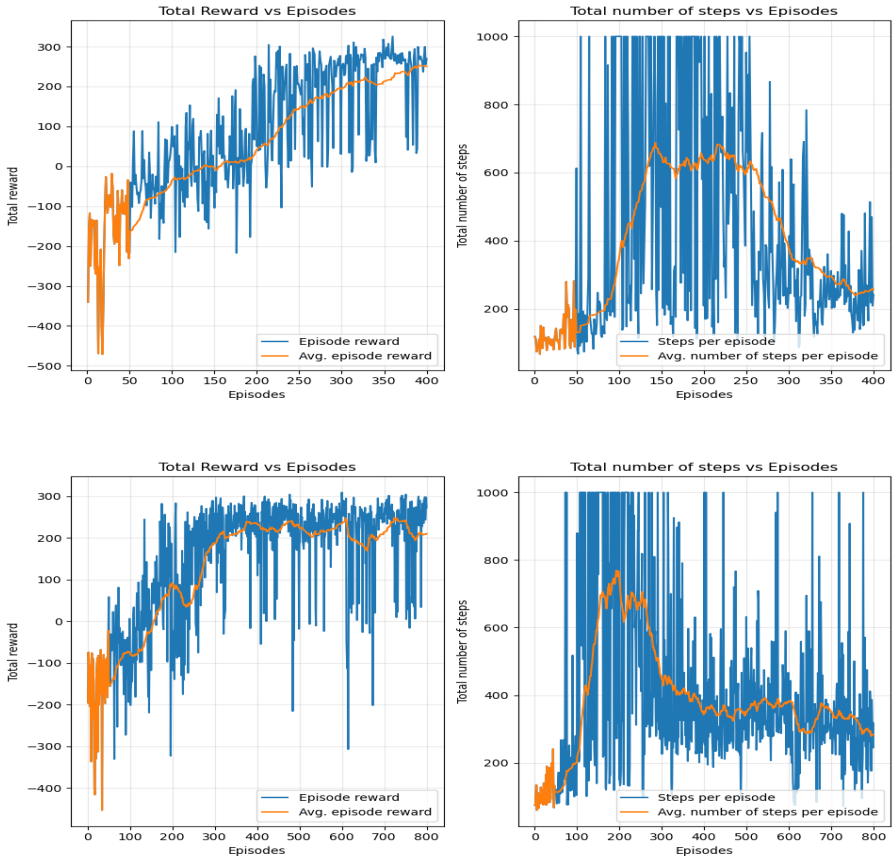

Figure 3: Effect of varying the number of episodes T_E on the training process. Training process is shown for $T_E = 400$ (up) and for $T_E = 800$ (bottom). In orange the running average over the last 50 episodes is shown.

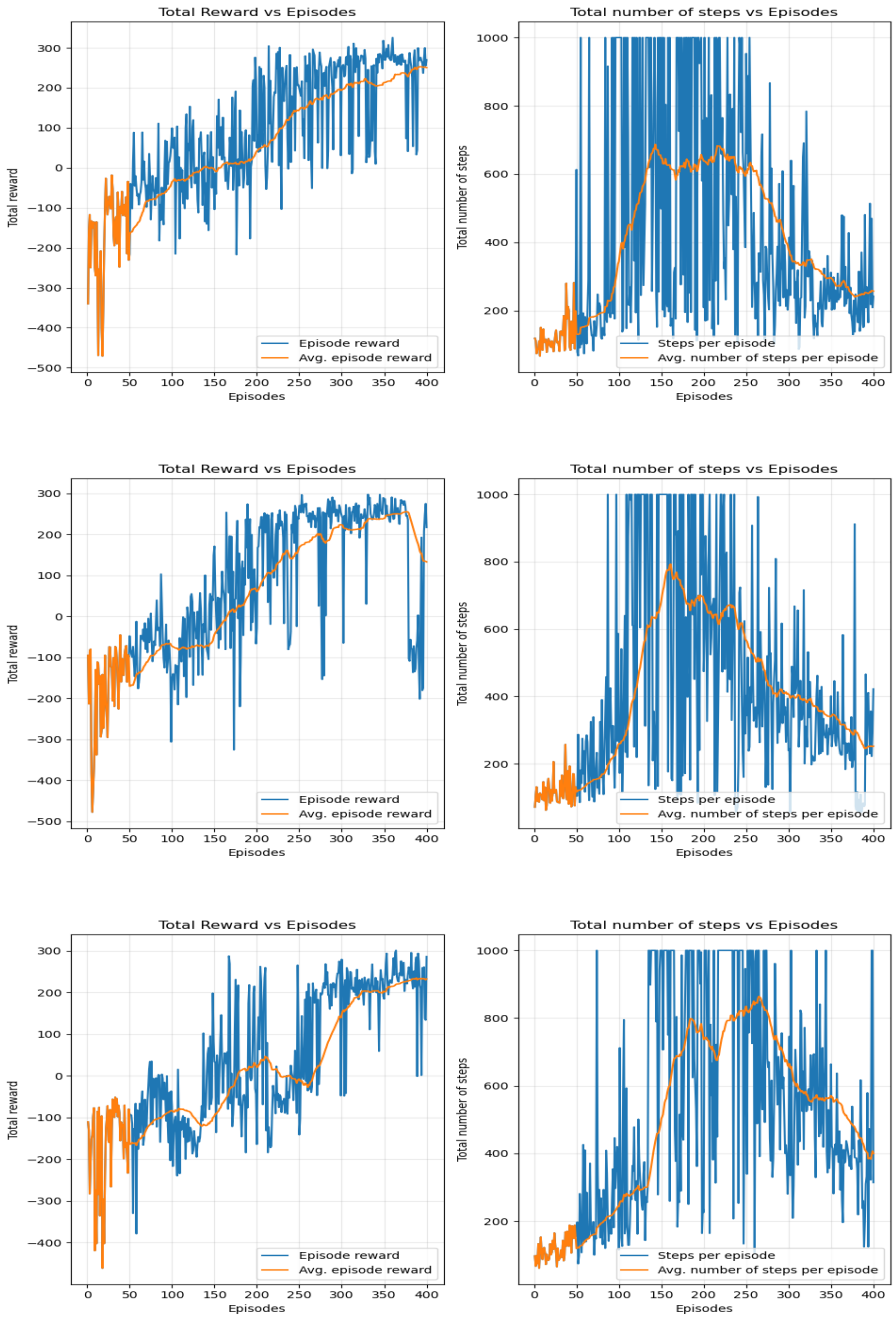

Figure 4: Effect of varying the memory size L on the training process. Training process is shown for $L = 10000$ (up), for $L = 20000$ (middle) and for $L = 30000$ (bottom). In orange the running average over the last 50 episodes is shown.

1.4 Task f: Q-Function Analysis

In the following the trained Q-network is analyzed. The state vector is restricted to $s = (0, y, 0, 0, \omega, 0, 0, 0)$, where $y \in [0, 1.5]$ and $\omega \in [-\pi, \pi]$. y denotes the height of the lander and ω denotes the angle of the lander. Figure 5 displays the Q-function values for the optimal actions a', $\max_{a'} Q(s, a')$. Given a constant y, intuitively the Q-function should be symmetrical w.r.t the axis where the lander angle is zero. However, this is not reflected in the learned Q-function. For positive angle values, the Q-function has high values, while for negative angles, the Q-function decreases. For zero lander angle, a decrease in the Q-function as y decreases should be expected, since or y = 0 and $\omega = 0$, the expected reward should be zero.

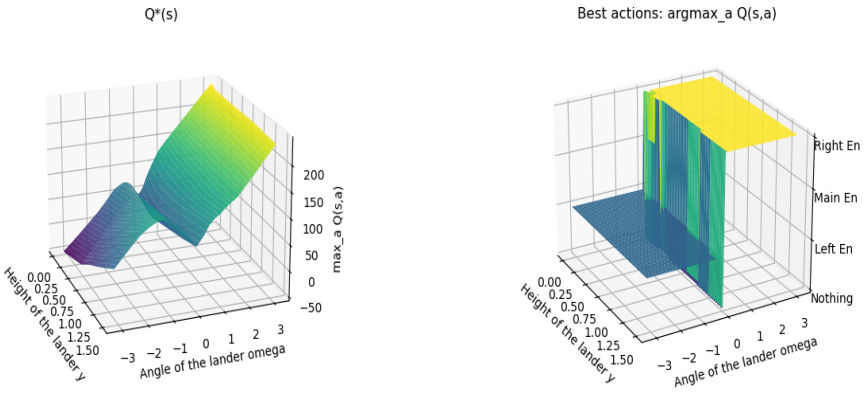

Figure 5: Q-function values corresponding to the optimal action a' (left), $Q^*(s) = \max_{a'} Q(s, a')$, and the corresponding optimal actions $a* = argmax_{a'} Q(s, a')$ (right). The state vector $s = (0, y, 0, 0, \omega, 0, 0, 0)$, varies within $y \in [0, 1.5]$ and $\omega \in [-\pi, \pi]$.

Different training parameters recovered the symmetrical form of the Q-function, but still no decrease in Q is found for decreasing y.

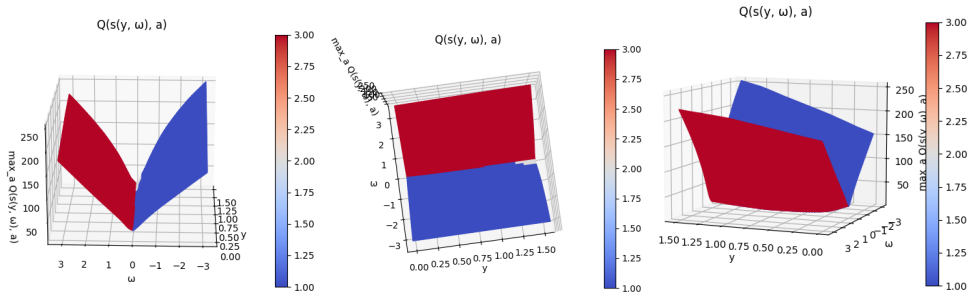

Figure 6: Q-function values corresponding to the optimal action a' (left), $Q^*(s) = \max_{a'} Q(s, a')$, and the corresponding optimal actions $a* = argmax_{a'} Q(s, a')$ under different training parameters.

Figure 5 illustrates the optimal actions $a^* = argmax_{a'} Q(s, a')$ according to the trained Q-network, where s is under the same restriction as above. Although the learned Q-function seems unintuitive, the optimal policy is accurate. For negative lander angles, the left engine

is fired to balance out the spaceship. For positive lander angles the right engine is fired. For a lander angle of zero, the actions vary from doing nothing to firing the main engine.

1.5 Task g: Dueling DQN vs. Random Agent

The DQN-agent is compared to a random agent in Figure 7. The trained DQN-agent achieves an average episodic reward of approximately 200, while the random agent achieves an average of -200.

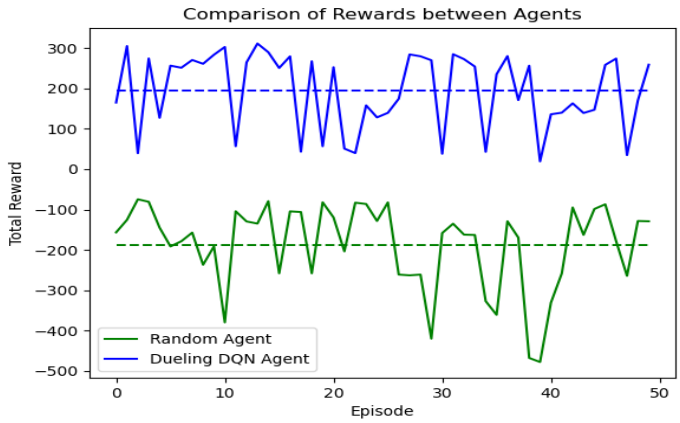

Figure 7: Comparison between a random agent and the DQN-agent over 50 episodes. Dashed lines indicate the mean over the 50 episodes.

1.6 Task h: Validation of Solution

Our trained Dueling DDP agent clearly and consistently achieves an average total reward of more than 75, as required for this task (see Figure 8).

```
Episode 49: 100%| | 50/50 [00:09<00:00,  5.32it/s]
Policy achieves an average total reward of 197.9 +/- 24.8 with confidence 95%.
Your policy passed the test!
```

Figure 8: Output of `check_solution.py`.

2 Deep Deterministic Policy Gradient (DDPG)

2.1 Task b: Theoretical Questions

DDPG is an actor-critic algorithm. The policy is parameterized directly using a parameter θ. The policy update is based on the policy gradient theorem for deterministic policies, where the objective function in the case of off-policy methods is $J_b(\theta) = \mathrm{E}_{s\sim\xi_b}[V^{\pi_\theta}(s)]$ with ξ_b being the stationary distribution under the behaviour policy.

Policy Gradient Theorem (Off-policy) The gradient of the objective function $J(\theta)$ w.r.t θ is

$$\nabla J_b(\theta) = \mathrm{E}_{s\sim\xi_b}\left[\nabla_\theta \pi_\theta(s) \cdot \nabla_a Q^{\pi_\theta}(s,a)|_{a=\pi_\theta(s)}\right] \tag{4}$$

The Q-function appears in the Policy Gradient Theorem. Therefore, Temporal Learning is used to evaluate Q^{π_θ}. Hence, DDPG is ideal for RL problems with continuous state and action spaces.

The critic's target network hence only serves to stabilize the evaluation of the current parameterized policy π_θ. One way to obtain the actor network from the evaluated Q^{π_θ} is to find the action $a^* = arg\max_{a'} Q^{\pi_\theta}(s,a')$. This requires solving a global optimization problem for all states, which is not trivial since $Q^{\pi_\theta}(s,a)$ is a function of two continuous variables and arbitrarily complex.

Clearly, DDPG is an off-policy reinforcement learning algorithm. Since an experience replay buffer is used to break temporal correlations, the experience we sample is not necessarily generated by the on-policy actor and critic. The utilization of an experience replay improves data efficiency, since past experiences are stored in the buffer and can be resampled to train the agent.

2.2 Task c and d: Training Base DDPG Agent

Structure of the actor's and the critic's network First, the actor's network is described. The actor's network consisted of three layers and takes as input the state vector. The input layer contains 400 neurons with ReLU activation, the hidden layer has 200 neurons with ReLU activation and the output is the dimensionality of the action space. Lastly, the output passes on to a hyperbolic tangent activation function to restrict the action to [-1, 1].

The critic's network is similar, with two differences: The network takes the concatenated vector of the state and action as input and there is no output activation function. The network outputs a scalar value.

The choice of the network architecture is based on the recommendations on the task. However, the critic's network was changed to the form described above, which yielded better results, at the expense of slowing down the training.

Training Parameters The actor's learning rate was set to $5e{-}5$ and the critic's learning rate to $5e-4$. Furthermore, the buffer size L was fixed to 30000 with a training batch size N of 64 samples. The actor update is delayed and performed together with the update of the target networks every $d = 2$ episodes. Additionally, the parameters of the target networks

are updated softly using the soft update parameter $\tau = 0.001$. The agent was trained for $T_E = 300$ episodes. All parameters shown here are based on the recommendations section in the task. The training was stopped as soon as an average reward of 150 episodes was observed over 50 episodes.

Problem-dependent Parameters The discount factor was set to 0.99. Furthermore, an Ornstein-Uhlenbeck process is used to add noise to the action taken by the actor's network to encourage exploration. The noise filter is updated based on the following recursive relation

$$n_t = -\mu n_{t-1} + w_{t-1}, \tag{5}$$

where $n_0 = 0$ and $w_i \sim \mathcal{N}(o, \sigma^2 I_m)$. The decay parameter μ was chosen to be 0.15 and the variance of the white noise $\sigma^2 = 0.04$, which again are based on suggestions in the task.

All hyperparameters of the DDPG agent are summarized in Table 2.

Table 2: Hyperparameters for the DDPG agent trained in Task c.

Discount factor	γ	0.99
Buffer size	L	30.000
Number of episodes	T_E	300
Policy and target networks update frequency	d	2
Training batch size	N	64
Actor learning rate	α_{actor}	$5 \cdot 10^{-5}$
Critic learning rate	α_{critic}	$5 \cdot 10^{-4}$
Soft update parameter	τ	$1 \cdot 10^{-3}$
Noise Decay Rate	μ	0.15
White noise variance	σ	0.2

In general, a high learning rate for a network would result in instability during training and might lead to oscillations or divergence. The critic's role is to estimate the value of the Q-function under the current parameterized policy. In contrast, the actor's rule is to adjust the policy parameters based on the policy gradient theorem, which depends on the critic's network.

Hence, it is better to have a larger learning rate for the critic's network to be more responsive to the current policy π_θ, while the smaller learning rate of the actor makes it more robust to the noisy updates in the critic's network in the presence of high variance.

2.3 Task e: Hyperparameter Analysis

The total episodic rewards and the steps taken for the base model of the DDPG agent is shown in Figure 9. A similar trend to the discrete case can be observed. After an initial phase, where the buffer is filled with random experiences, the average reward Stabilizes and start increasing. The training was stopped as soon as the average reward exceeded 150 over the last 50 episodes.

Similar to Problem 1, the effect of varying discount factors is analyzed. Figure 10 shows the total episodic reward and the total number of steps per episodes against the number of episodes for three different discount factors. It is clear that for $\gamma_1 = 1$ and $\gamma_2 = 0.4$ the agent does not learn successfully. For $\gamma_1 = 1$, the DDPG algorithm should converge in theory, since the gradient is sampled from the correct distribution. In reality, the average

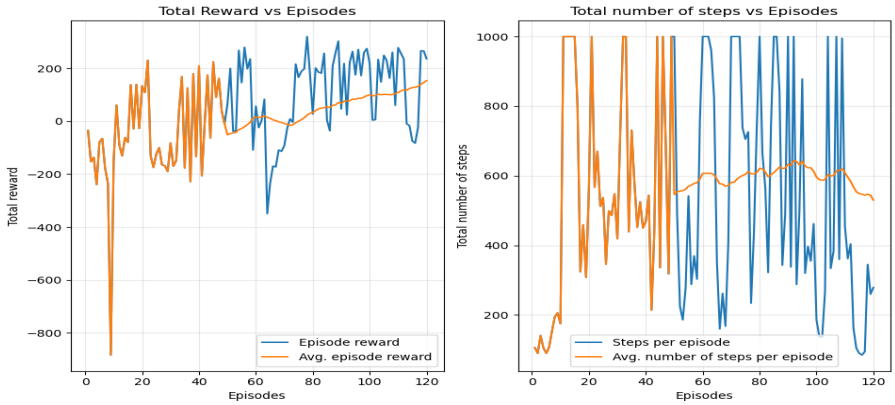

Figure 9: Training Process of the DDPG agent for the continuous Lunar Lander environment using the parameters in 2. In orange the running average over the last 50 episodes is shown. The training was interrupted as soon as the average reward was above 150 over the last 50 episodes.

total rewards shows a decreasing trend and the agent's performance actually degrades over the episodes. For $\gamma_2 = 0.4$, future rewards are discounted heavily, s.t. the agent is short-sighted. In this case, the agent seems to not improve at all with the average total reward being nearly constant over the episodes. Although in theory the use of a discount factor results in the gradient being sampled from the wrong distribution, since the gradient is still sampled from the stationary distribution, the base model still converges with a discount factor of $\gamma_0 = 0.99$.

In the following, the effect of varying the memory size is investigated. Figure 11 shows the total episodic reward and the total number of steps per episodes against the number of episodes for three different memory size L. For $L_1 = 10000$, the training process is not stable and the average total episodic rewards is oscillating over the episodes. For $L_2 = 40000$, the training process is very similar to the base model and the training process is stopped before the 300 episodes as the average total episodic reward exceeds 150.

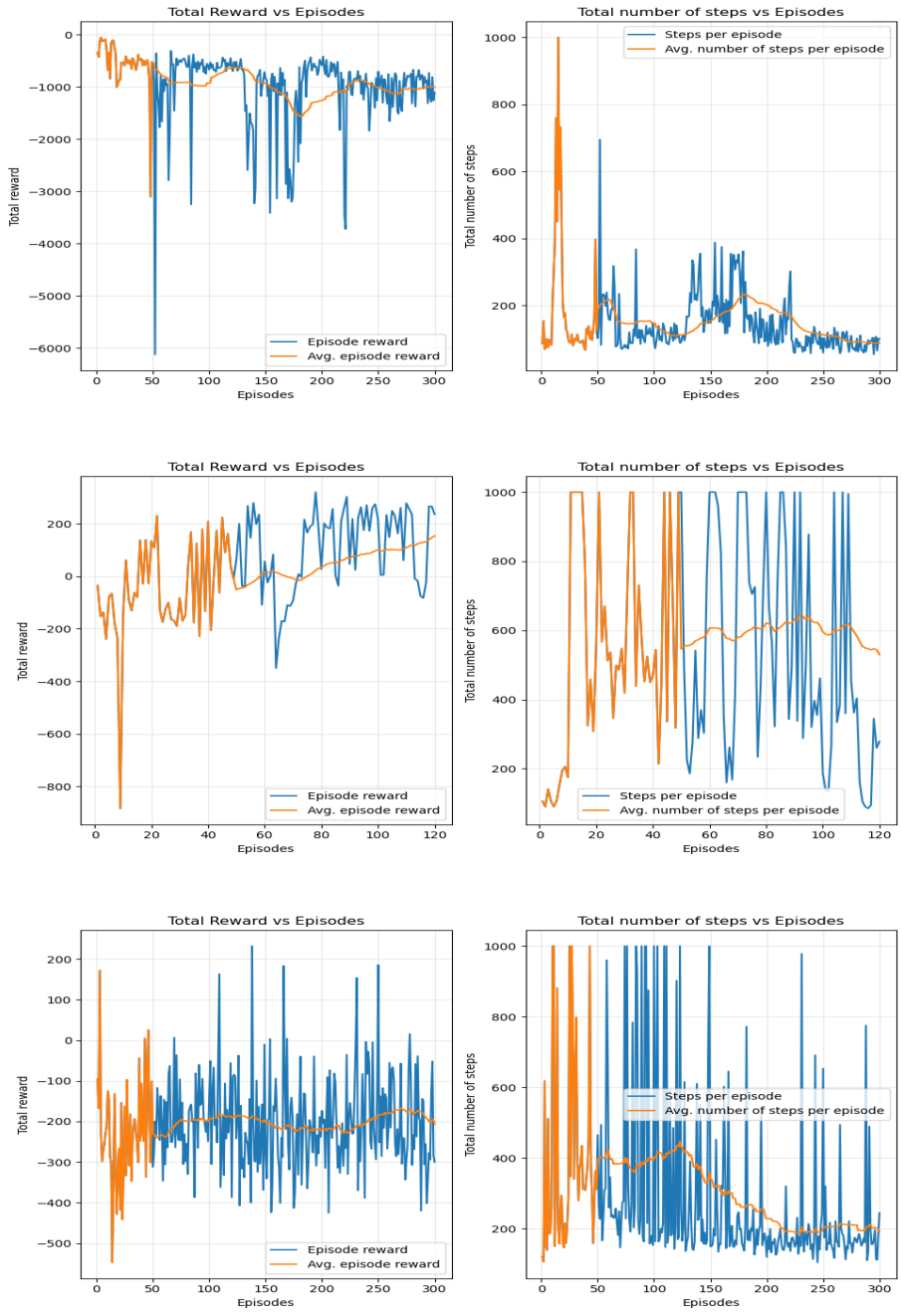

Figure 10: Effect of varying discount factors on the training process. Training process is shown (up) for $\gamma_1 = 1$, (middle) for γ_0 and (bottom) for $\gamma_2 = 0.4$. In orange the running average over the last 50 episodes is shown.

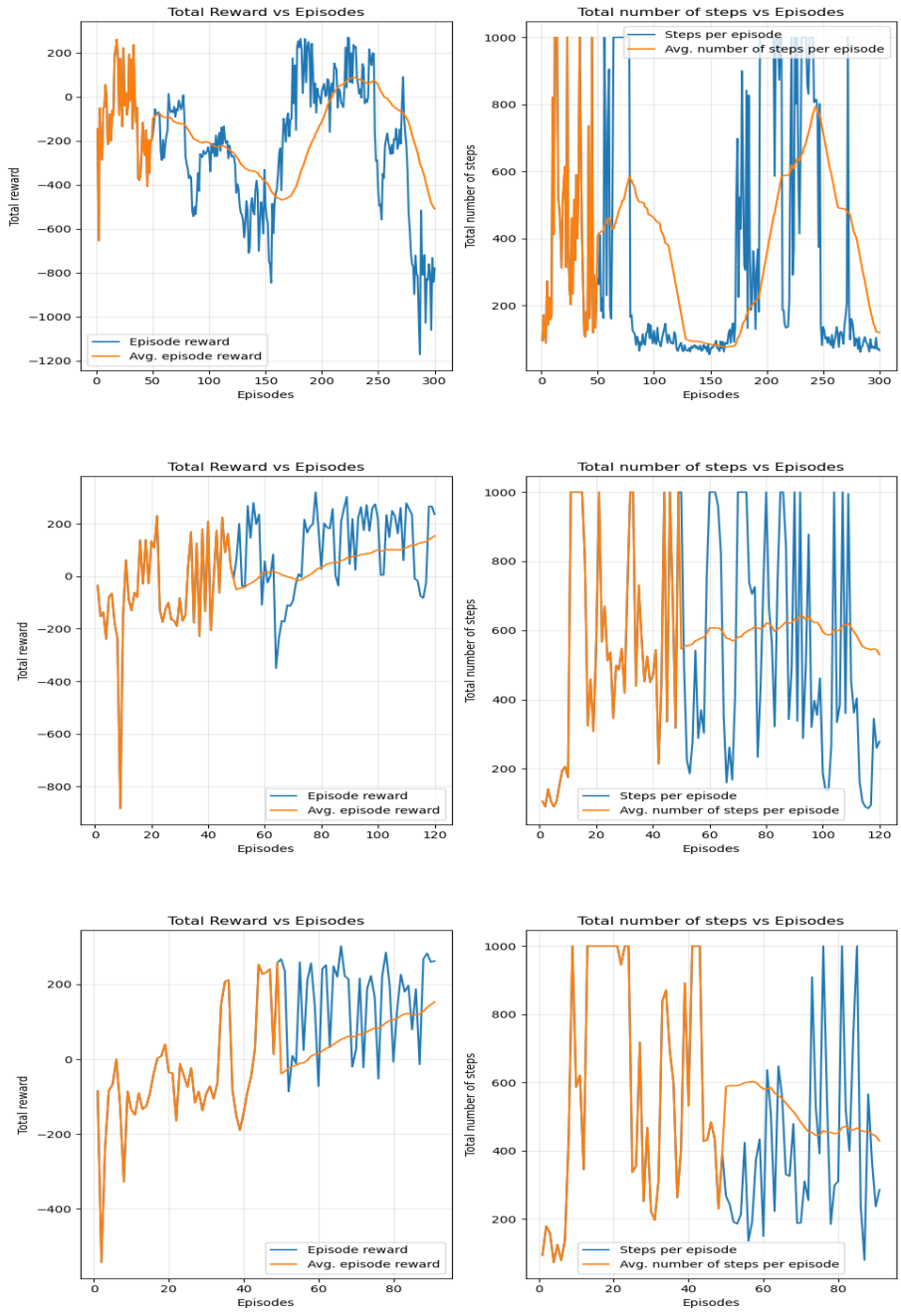

Figure 11: Effect of varying the memory size L on the training process. Training process is shown for $L_1 = 10000$ (up), for $L_0 = 30000$ (middle) and for $L_2 = 40000$ (bottom). In orange, the running average over the last 50 episodes is shown.

2.4 Task f: Q-Function Analysis

In the following, the trained Q-network and the policy are analyzed. The state vector is restricted to $s = (0, y, 0, 0, \omega, 0, 0, 0)$, where $y \in [0, 1.5]$ and $\omega \in [-\pi, \pi]$. y denotes the height of the lander and ω denotes the angle of the lander. Figure 12 displays the Q-function values $Q_w(s, \pi_\theta(s))$ at the state s and the action $a = \pi_\theta(s)$. Given a constant y, intuitively the Q-function should be symmetrical w.r.t the axis where the lander angle is zero. This is more evident than in the case of the discrete actions space. For positive and negative angle values, the Q-function has high values. Still the Q-function is biased towards the states with negative lander angle. There is a local minimum for zero lander angle, which makes sense, since firing the main engine incurs a higher negative reward.

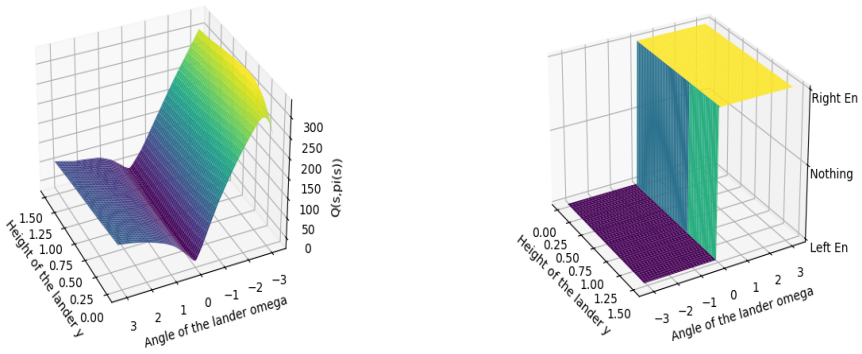

Figure 12: Q-function values corresponding to the trained critic's network (left), $Q_w(s) = Q_w(s, \pi_\theta(s))$, and the direction of the firing engine actions from the actor's network $[\pi(s)]_2$ (right). The state vector $s = (0, y, 0, 0, \omega, 0, 0, 0)$, varies within the ranges $y \in [0, 1.5]$ and $\omega \in [-\pi, \pi]$.

Figure 12 illustrates the learned policy by plotting the direction of the firing engine according to the trained actor network, where s is under the same restriction as above. The policy makes sense. For negative lander angles, the left engine is fired to balance out the spaceship. For positive lander angles the right engine is fired. For a lander angle of zero, the actions vary from doing nothing to firing the main engine.

2.5 Task g: DDPG vs. Random Agent

The DDPG-agent is compared to a random agent in Figure 13. The trained DDPG-agent achieves an average episodic reward of approximately 190, while the random agent achieves an average of -210.

2.6 Task h: Validation of Solution

Our trained DDPG agent clearly and consistently achieves an average total reward of more than 125, as required for this task (see Figure 14).

Figure 13: Comparison between a random agent and the trained DDPG-agent over 50 episodes. Dashed lines indicate the mean over the 50 episodes.

```
Checking solution...
Episode 49: 100%|                                                  | 50/50 [00:40<00:00,  1.23it/s]
Policy achieves an average total reward of 184.0 +/- 18.1 with confidence 95%.
Your policy passed the test!
```

Figure 14: Output of `check_solution.py`.

3 Proximal Policy Optimization (PPO)

3.1 Task b: Theoretical Questions

PPO doesn't use target networks, because they are not required. While in other learning based RL algorithms there is a need for target networks to provide a stable target to optimize toward, in PPO the policy update rule is different. This is mainly because in PPO, e.g. in contrast to Q-learning, there are already two different networks to determine and evaluate actions. Secondly, PPO clips the gradients, such that the updated policy has to stay close to the previous policy.

PPO is an on-policy method, since it learns from and updates the current policy directly, rather than using something like a behavior policy. While it uses a buffer like e.g. DDPG, this buffer is episodic and all samples are generated with the current policy.

The same reasons explain why PPO has the issue of sample complexity. PPO requires continuously new samples and does not reuse previously generated sequences for future updates. It therefore requires more data, which might be computationally expensive to generate.

3.2 Task c and d: Training Base PPO Agent

Following *Algorithm 3* in the lab description an agent using the PPO algorithm was implemented.

We used the parameters suggested in the exercise and did not have to modify those (see Table 3). The only modification deployed was to standardize the returns, which resulted in a great improvement of learning a good policy. Without it, we barely reached the 125 average reward threshold within 1.600 episodes and even then failed the validation with

`check_solution` due to a high confidence interval. With standardized returns however, we reached the threshold consistently and passed the validation with a significant margin.
In the implemented version of PPO we update the actor and critic networks simultaneously, both M times after each episode. The optimizer for the critic networks uses a significantly higher learning rate (factor 100) to ensure a good value estimation and less noisy updates in the actor network output. An unbalanced update strategy where the critic is updated more frequently than the actor could lead to the same stability or even slightly improve it. But since the training process is already rather stable we do not think it is necessary to limit the updates of the actor network and hypothesize that that might even to a slower training and bigger data requirements (worse sample efficiency).

Table 3: Hyperparameter for the PPO model.

Discount factor	γ	0.99
Buffer size	L	one episode
Number of episodes	T_E	$\overline{r} > 135$ (max 1600)
Learning rate actor	α_{actor}	$3 \cdot 10^{-5}$
Learning rate critic	α_{critic}	$3 \cdot 10^{-3}$
Update steps per episode	M	10
Gradient clipping value	ϵ	0.2

3.3 Task e: Hyperparameter Analysis

To analyze the effect of the discount factor γ and gradient clipping value ϵ we plot the total reward and total number of steps per episode for different values and compare them to our manually tuned values. In Figure 15, we show the training process of the selected hyperparameters ($\gamma_0 = 0.99$ and $\epsilon_0 = 0.2$).

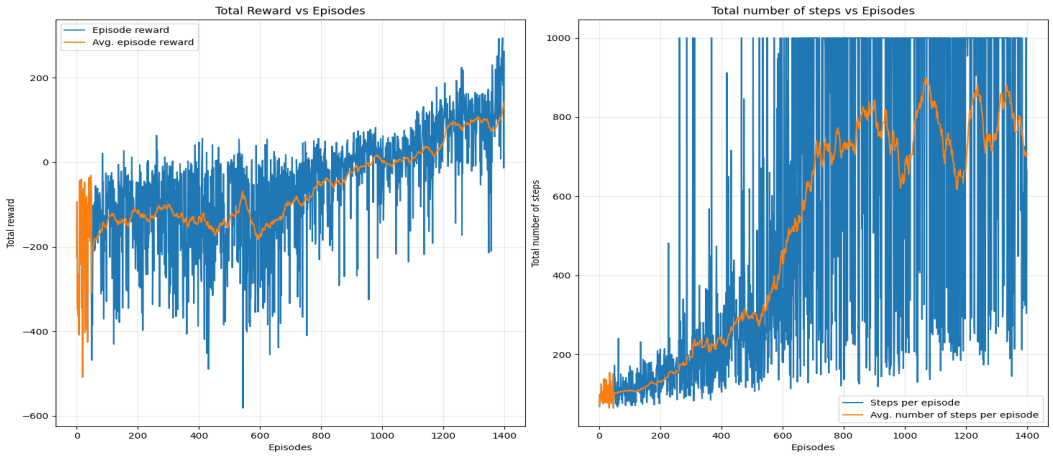

Figure 15: Training Process of the PPO agent for the continuous Lunar Lander environment using the parameters in Table 3. In orange, the running average over the last 50 episodes is shown. The training was interrupted as soon as the average reward was above 130 over the last 50 episodes.

It is visible that the agent starts from a negative total reward per episode with high variance and generally short episodes. In those the lander probably crashes rather fast,

resulting in high negative reward and short episodes. The agent is quickly able to generate longer episodes, due to a better understanding of the engines and avoidance of crashing. Interestingly the reward does increases rather slowly compared to the episode length and has it steepest ascent when the episode length gets shorter again. This also seems logical, as this is likely the point where the agent successfully lands the lander at least sometimes. In Figure 16, we show the training process for a lower and a higher value for the discount factor γ. Both have a strong effect of the evolution of total reward and number of steps per epoch. For the smaller discount factor $\gamma_2 = 0.5$ the agent seems not capable of learning anything within the 1,600 episodes, as neither the reward nor the number of steps change. That small of a discount factor seems to limit the horizon of the agent, such that it does not really plan for future rewards/penalties. While a higher discount factor $\gamma_1 = 1$ also does not lead to a increase in reward over 1,600 episodes, towards the end the episodes start to get longer. This might result in a delayed learning capability compared to our chosen value $\gamma_0 = 0.99$. Both values clearly perform worse compared to our chosen value.

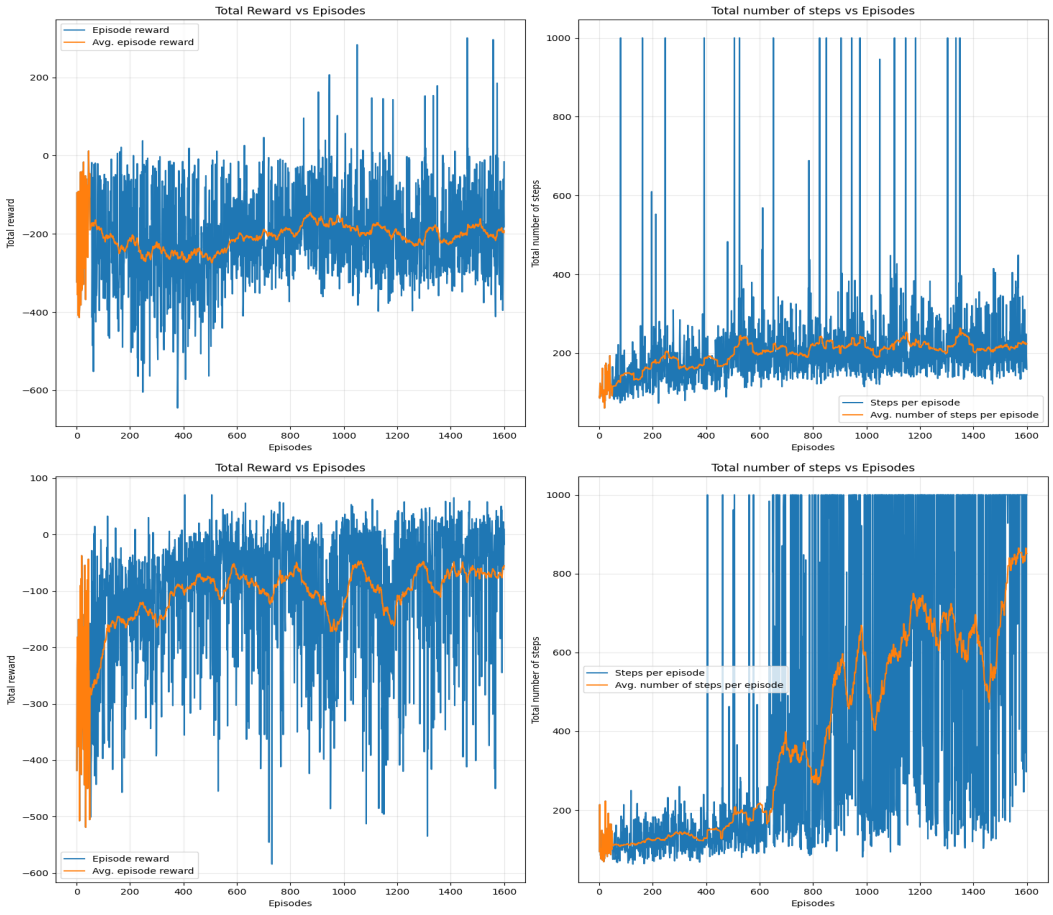

Figure 16: Total episodic reward and total number of steps taken during each episode in the training process of the PPO agent for different discount factors $\gamma_2 = 0.5$ (top) and $\gamma_1 = 1$ (bottom).

We performed the same kind of analysis with respect to the value of the gradient clipping value ϵ (see Figure 17). Even though we try change the value by a significant factor,

the resulting training processes are similar to our initial training run. Neither the small gradients nor the significantly larger ones hurt the performance. The only difference is that the threshold of an averaged reward of 135 is reached faster for the higher gradient clipping value $\epsilon_1 = 0.5$, which seems logical. Those gradients seem to be stable enough to ensure a continuous training and just result in larger steps. An even higher value would likely lead to an unstable training process, as this would defeat the reason for the clipping.

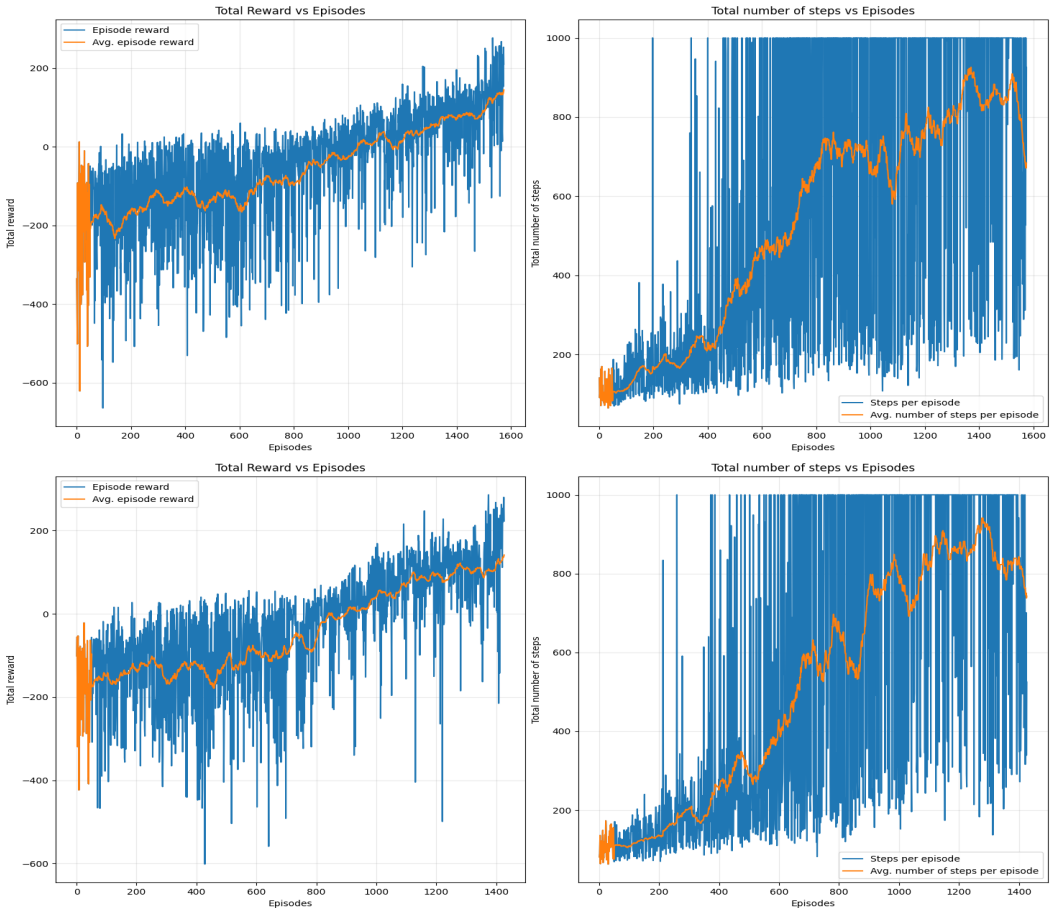

Figure 17: Total episodic reward and total number of steps taken during each episode in the training process of the PPO agent for different gradient clipping values $\epsilon_2 = 0.05$ (top) and $\epsilon_1 = 0.5$ (bottom).

3.4 Task f: Analysis of Value Function

For further analysis of the Value function we show two 3d-plots in Figure 18 and Figure 19. In Figure 18, we show the value for different states by varying the height y and angle ω of the lander. There is some symmetry visible for values of higher absolute angles, which makes intuitively sense: The agent should not value a rotation to one side other than to the opposite side.

In Figure 19, we show the value of the second component of the action vector for different states. This value determines the activation of the left and right engines: Values between -1 and -0.5 fire the left engine with 100% to 50%, and values between 0.5 and 1 correspond

to an activation of the right engine of 50% to 100%. As expected uses the agent always the engine on that side, that is required to return to a neutral position, and doesn't fire for values very close to a neutral position. The height however does not play a role in the determination of the activation of side engines.

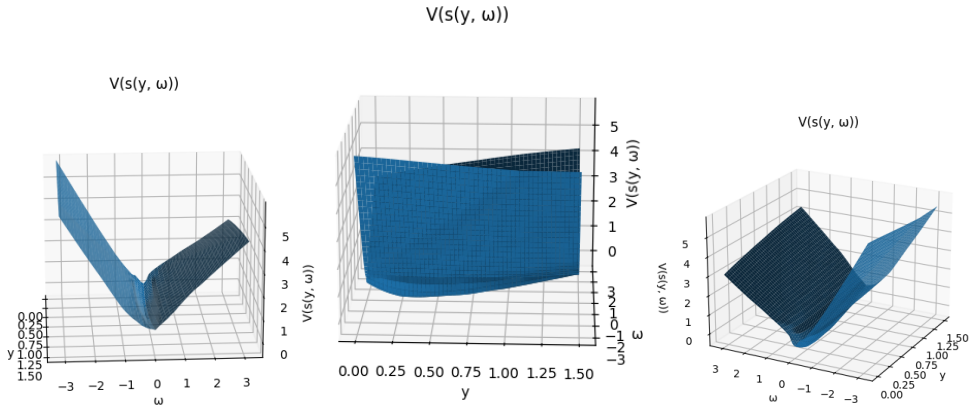

Figure 18: Three views of the plot of the Value-function with different values for the height y and angle ω of the lander.

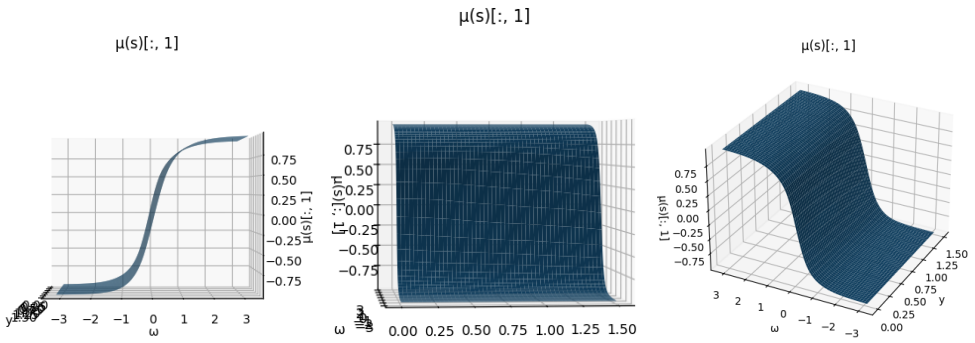

Figure 19: Three views of the plot of the second component of the action for states with different values for the height y and angle ω of the lander.

3.5 Task g: PPO vs. Random Agent

In Figure 20, we show the performance of the PPO agent in contrast to the random agent. The PPO agent clearly outperforms the random agent by consistently achieving a higher total reward per episode. It is also shown, that the episodes of the random agent are rather short compared to the PPO agent, probably the result of uncontrolled behavior resulting in crashes.

3.6 Task h: Validation of Solution

Our trained PPO agent clearly and consistently achieves an average total reward of more than 125, as required for this task (see Figure 21).

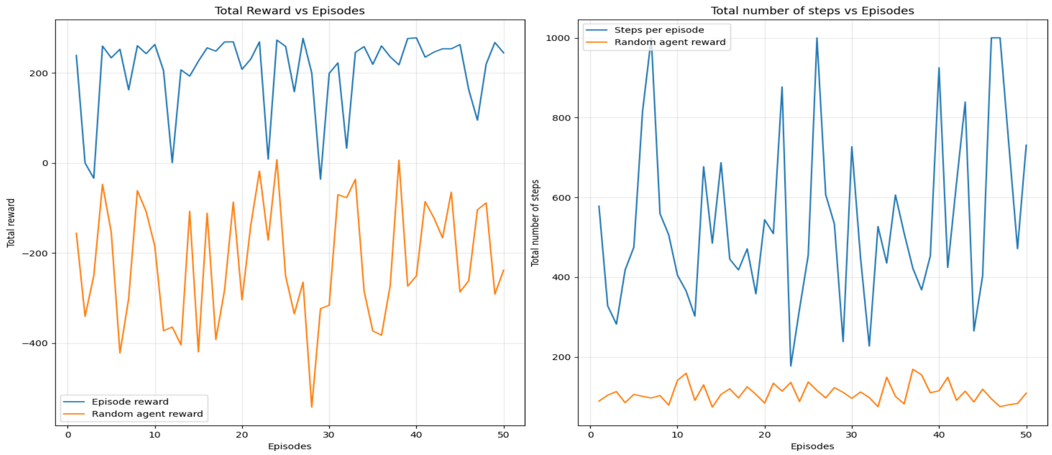

Figure 20: Total episodic reward and total number of steps taken during each episode for a random agent (orange) and a PPO agent (blue).

```
Network model: ActorModel(
  (actor_shared): Sequential(
    (0): Linear(in_features=8, out_features=400, bias=True)
    (1): ReLU()
  )
  (actor_mean): Sequential(
    (0): Linear(in_features=400, out_features=200, bias=True)
    (1): ReLU()
    (2): Linear(in_features=200, out_features=2, bias=True)
    (3): Tanh()
  )
  (actor_var): Sequential(
    (0): Linear(in_features=400, out_features=200, bias=True)
    (1): ReLU()
    (2): Linear(in_features=200, out_features=2, bias=True)
    (3): Sigmoid()
  )
)
Checking solution...
Episode 0:   0%|                                                                                    | 0/50 [00:00<?, ?it/s]
/Users/aweers/git/el2805/lab2/problem3/PPO_check_solution.py:66: UserWarning: Creating a tensor from a list of numpy.ndarrays is extremely slow. Please consider converting the list to a si
ngle numpy.ndarray with numpy.array() before converting to a tensor. (Triggered internally at /Users/runner/work/pytorch/pytorch/pytorch/torch/csrc/utils/tensor_new.cpp:264.)
  mu, var = model(torch.tensor([state]))
Episode 49: 100%|████████████████████████████████████████████████████████████████████████| 50/50 [00:02<00:00, 19.36it/s]
Policy achieves an average total reward of 214.8 +/- 22.0 with confidence 95%.
Your policy passed the test!
```

Figure 21: Output of `check_solution.py`.