EL2805 Reinforcement Learning
Report of Laboratory 2

Franco Ruggeri
950305-8951
Division of Decision and Control Systems
KTH Royal University of Technology
Stockholm, Sweden
`fruggeri@kth.se`

Andres Felipe Cardenas Meza
010306-8516
KTH Royal University of Technology
Stockholm, Sweden
`afcm2@kth.se`

Abstract

In this laboratory, we solve the lunar lander problem from OpenAI Gym with several Reinforcement Learning algorithms. Specifically, we solve the discrete-action-space version of the lunar lander with DQN and the continuous-action-space version with DDPG and PPO. We analyze the effect of several hyperparameters and we visualize the shape of the learned value functions and policies by restricting the state to two degrees of freedom.

1 Problem 1 - DQN

In this section, we describe our solution to the lunar lander problem with discrete action space using Deep Q-Network (DQN), an off-policy deep Reinforcement Learning (RL) algorithm based on Q-learning with function approximation. Additionally, we analyze the Q-function learned by our best agent and explore the effect of some key hyperparameters for DQN on the training procedure.

1.1 Task 1b

The *replay buffer* is used for storing the *previous experiences* acquired from the interactions with the environment. The reason for its usage is that, in standard Q-learning with function approximation, successive updates are very correlated, as they use experiences along a particular trajectory. This correlation problem affects the convergence rate. The replay buffer addresses this problem by updating on randomly sampled batch of experiences from the replay buffer, instead of the last experience.

The *target network* is used to keep the targets constant in (the unbiased estimator of) the semi-gradient in Q-lerning with function approximation. The reason for its usage is that, if the target changes at each update, it might move too fast to be learned. The problem arises in general for RL algorithms combining the following three elements (*deadly triad* [1]): function approximation, bootstrapping, off-policy. Clearly, DQN has all three elements. The target network also partially justifies semi-gradient descent, which assumes constant targets.

1.2 Task 1d

We solved the lunar lander problem with DQN by using the hyperparameters reported in table 1 and the Q-network layout shown in fig. 1. We also used Combined Experience Replay (CER) as a modification of DQN to prioritize the newest experience.

Regarding the Q-network, we started by using a small Deep Neural Network (DNN) with two hidden layers of 64 neurons and ReLU activation, as shown in fig. 1. This initial architecture resulted in successful training, so we kept it without further tuning.

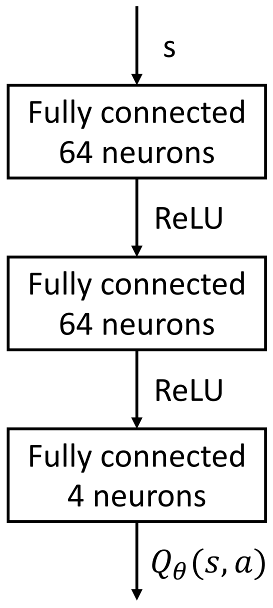
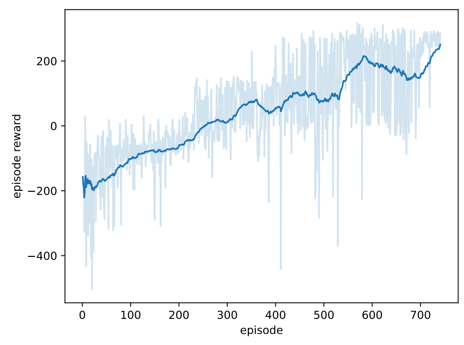

s
Fully connected 64 neurons
ReLU
Fully connected 64 neurons
ReLU
Fully connected 4 neurons
$Q_\theta(s,a)$

Figure 1: Architecture of Q-network used to solve the lunar lander with DQN.

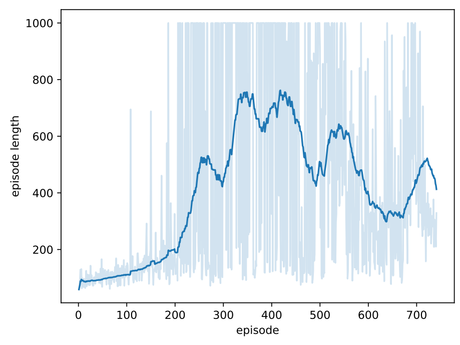
Table 1: Hyperparameters to solve the lunar lander with DQN.

Hyperparameter	Value
γ (discount factor)	0.99
L (replay buffer size)	10000
T_E (n. episodes)	1000
N (batch size)	64
C (target refresh update)	$\lfloor L/N \rfloor$
Exploration decay	Exponential
Exploration decay duration (episodes)	$90\% \cdot T_E$
ϵ_{max}	0.99
ϵ_{min}	0.05
Optimizer	Adam
Modifications	CER
Learning rate	$5 \cdot 10^{-5}$
Replay buffer min. experiences	$0.2 \cdot L$
Max. gradient norm (clipping)	1
Reward for early stop	250

As for the hyperparameters, we manually tuned only learning rate and replay buffer size L by trying a few values. For all the other hyperparameters, instead, we chose values recommended in the lab instructions. Specifically, we set the batch size to $N = 64$, which is a commonly used value and empirically performed well enough. We calculated the target refresh period as a function of replay buffer size and batch size, as $C = \lfloor L/N \rfloor$ (as suggested in the lab instructions). We trained for $T_E = 1000$ episodes but early stopped the training when the average episode reward over the last 50 episodes was above 250. We set the discount factor to $\gamma = 0.99$, which is less than 1 (as required by Q-learning) and made the agent learn to land without keeping flying. We used an exponential decay of ϵ in the ϵ-greedy policy from a maximum value $\epsilon_{max} = 0.99$ to a minimum value $\epsilon_{min} = 0.05$ reached after 90% of the episodes ($90\% \cdot T_E$), as recommended in the lab instructions.

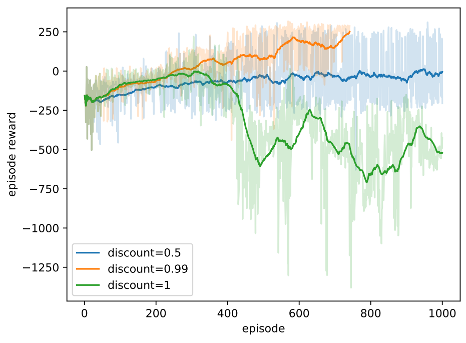


(a) Episode reward

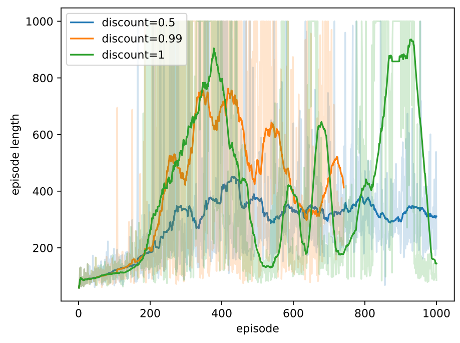


(b) Episode length

Figure 2: Learning curves of DQN solving the lunar lander.



(a) Episode reward



(b) Episode length

Figure 3: Learning curves of DQN with varying discount factor γ.

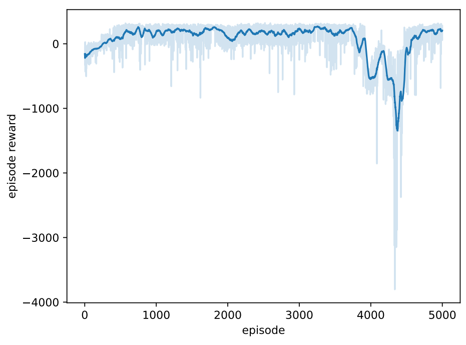
1.3 Task 1e

We trained DQN with the hyperparameters mentioned in section 1.2. The episode reward and episode length during the training process are shown in fig. 2. The agent reaches an average reward of 250 and stops before the limit of $T_E = 1000$ episodes. As can be seen in fig. 2a, the training is quite stable and reaches a high average episode reward (episode reward around 250), which means DQN solves the lunar lander.

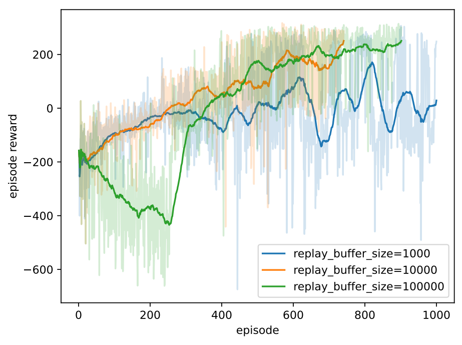
We investigated the effect of the discount factor on the training process of DQN. The results are shown in fig. 3. As expected, DQN does not converge with $\gamma = 1$, since Q-learning (and consequently DQN) requires $\gamma \in (0, 1)$ to converge. On the other hand, a too small discount factor $\gamma = 0.5$ discounts future rewards too much and results in the agent not learning to land. The discount factor $\gamma = 0.99$, instead, represents a good choice and makes DQN solve the lunar lander.

We explored also the effect of the number of training episodes on DQN. Thus, we disabled early stopping and trained for $T_E = 5000$ episodes. The result is shown in fig. 4a. We can observe that a catastrophic forgetting happens around episode 4000, and then DQN learns again. This is a good example to show how important it is to stop training when the agent has learned to solve the task.

Finally, we inspected the effect of increasing and decreasing the size of the replay buffer L, which is a key feature of DQN. The results are shown in fig. 4b. We found that $L = 10^4$ is a good choice and manages to solve the lunar lander. In our settings, DQN waits until 20% of the replay buffer has been filled up ($20\% \cdot L$. Thus, the larger the replay buffer, the longer the initial wait to start training, which explains why the performance of $L = 10^5$ is initially very low. We can observe that $L = 1000$ cannot solve the problem. The reason is that the lunar lander has many different

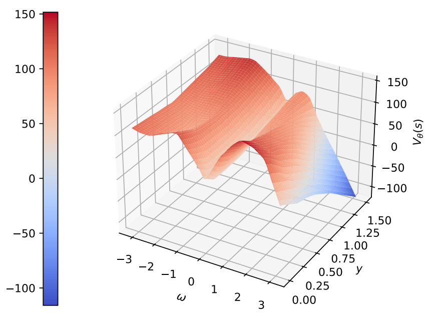


(a) Number of episodes

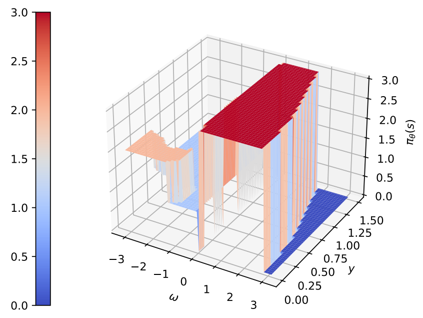


(b) Replay buffer size

Figure 4: Learning curves of DQN with high number of episodes (a) and varying replay buffer size L (b).



(a) Value function



(b) Policy

Figure 5: Value function (a) and policy (b) for varying height (y) and tilt angle (ω) of the lander, with the rest of the state set to 0.

trajectories, and a single trajectory (episode) can take up to 1000 samples, corresponding to the whole buffer. Consequently, $L = 1000$ is too small for such a problem and results in the agent training on very correlated experiences.

1.4 Task 1f

We analyzed the Q-function learned by DQN by considering the restricted state $s(y, \omega) = (0, y, 0, 0, \omega, 0, 0, 0)$. In particular, by varying the two degrees of freedom of the restricted states, we evaluated the value function ($V_\theta(s) = \max_a Q_\theta(s, a)$) and the policy ($\pi_\theta(s) = \arg\max_a Q_\theta(s, a)$) learned by DQN. The resulting 3D plots are illustrated in fig. 17.

The shape of the value function is shown in fig. 17a. Although it is hard to interpret it, we can point out some intuitive parts. The lunar lander environment provides an incremental reward $r(s, a)$ by comparing the new state s' with the current state s. Thus, the more tilted the lander is at the start, the higher total reward is available to collect by adjusting the tilt and then landing. At the extreme, if we fix the height ($y = k$) and consider the states $s(\omega) = (0, k, 0, 0, \omega, 0, 0, 0)$, the minimum value is in $\omega = 0$, where the lander starts already aligned with the landing pad. Surprisingly, we can notice that the value function is asymmetric with respect to the tilt angle. We think that this might be the result of little data collected by the ϵ-greedy behavioral policy in certain regions. Essentially, it seems that the agent does not learn to recover from an initial state which is very tilted to the right.

The shape of the policy is shown in fig. 17b. Although complicated to interpret, we can observe that the agent *fires the right engine* ($\pi_\theta(s) = 3$) with positive tilt angles ($\omega > 0$), which makes sense

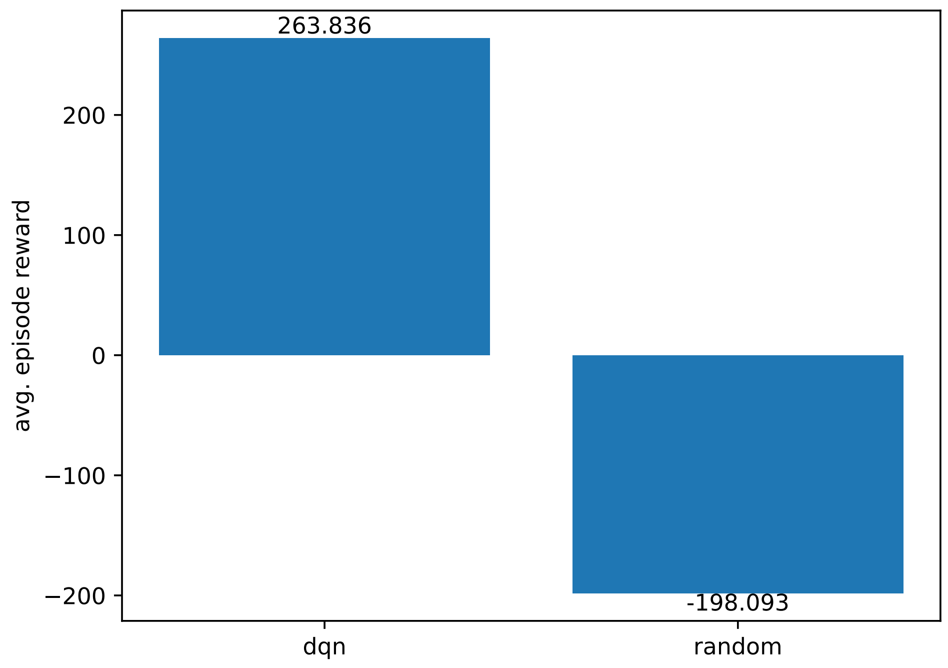


Figure 6: Performance comparison of DQN and random agent.

to adjust the tilt of the lander and make it horizontal. Additionally, when the lander is horizontally aligned ($\omega = 0$), the agent alternates between *doing nothing* ($\pi_\theta(s) = 0$) and *firing the main engine* ($\pi_\theta(s) = 2$), depending on the height (y).

1.5 Task 1g

We compared the performance of DQN against a naive agent taking random actions, which serves as a baseline. The result is shown in fig. 6. As expected, the DQN agent collects a much higher reward than the random agent, confirming once again that the DQN agent solved the lunar lander.

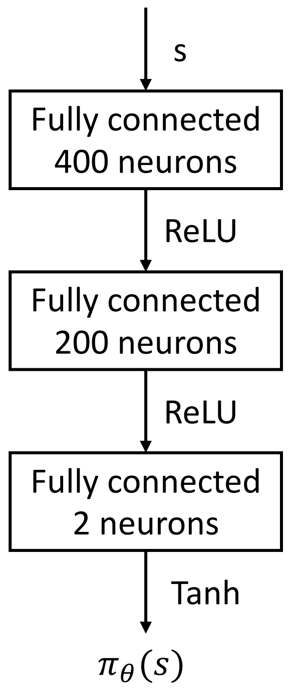
.

2 Problem 2 - DDPG

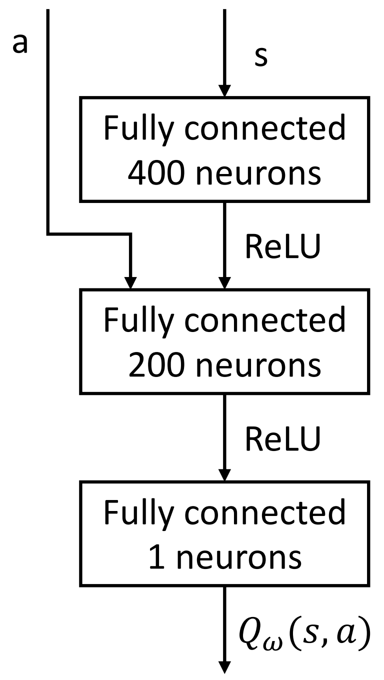
In this section, we describe our solution to the lunar lander problem with continuous action space using Deep Deteerministic Policy Gradient (DDPG), an off-policy deep RL algorithm based on actor-critic methods that uses deterministic policies. Additionally, we analyze the actor and the critic resulting from training and explore the effect of some key hyperparameters for DDPG on the training procedure.

2.1 Task 2b

In DDPG, the actor is updated using the most recent version of the critic, and not the critic's target network. The reason can be understood by recalling that, in the vanilla actor-critic algorithm, the actor is updated according to the policy gradient theorem using the critic. Target networks are introduced only to avoid the moving target problem arising with the combination of function approximation, bootstrapping, and off-policy.


s
Fully connected 400 neurons
ReLU
Fully connected 200 neurons
ReLU
Fully connected 2 neurons
Tanh
$\pi_\theta(s)$

(a) Actor


a
s
Fully connected 400 neurons
ReLU
Fully connected 200 neurons
ReLU
Fully connected 1 neurons
$Q_\omega(s,a)$

(b) Critic

Figure 7: Architecture of actor and critic used to solve the lunar lander with DDPG.

Parameter	Values
Target Update Frequency	2,4
Buffer size	30000,100000
Noise Reduction	True,False
Noise Generator	$\{\sigma_i$ = 1,σ_f = 0.1,linearly=True$\}$, $\{\sigma_i$ = 2, σ_f= 0.1, linearly = False$\}$, $\{\sigma_i$ = 0.2,σ_f = 0.1, linearly = False$\}$

Table 2: Hyper-parameters used for finding best DDPG configuration.

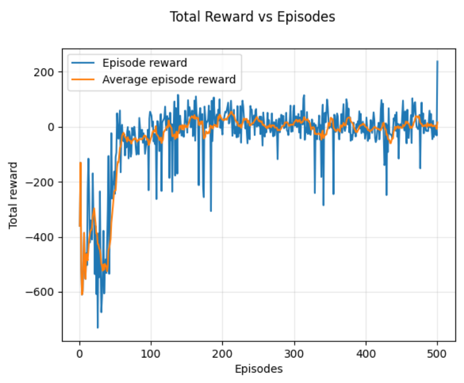
DDPG is an off-policy method because it learns an optimal policy while sampling experiences from the environment with a different behavioral policy, which has an additional noise to force exploration.

Sample complexity is an interesting performance metric for off-policy methods, since sampling from the environment is an expensive operation, especially in real-world problems which cannot be simulated.

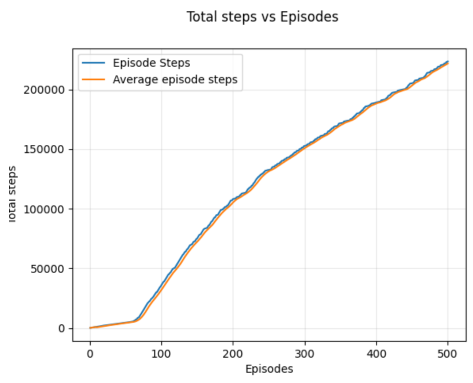
2.2 Task 2d

For solving the problem we used the following architecture for the neural networks, illustrated in fig. 7:

- *Actor*: DNN with 3 layers. It has 400 neurons in the first layer and 200 neurons in the second hidden layer, both with ReLU activiation function. The output layer uses a tanh activation function to output two-dimensional actions in $\pi_\theta(s) \in [-1,1]^2$.
- *Critic*: DNN with the same structure as the actor's. The state is input to the first layer, while the action to the second layer, after being concatenated with the output of the first layer. The output layer does not have an activation function and outputs $Q_\omega(s,a)$.



(a) Training Reward



(b) Training Steps

Figure 8: Learning curves of DDPG solving the lunar lander.

For the optimization of both the actor and the critic networks we used an Adam optimizer. However, their respective learning rates were different. The value was set to $5e-5$ for the actor and at $5e-4$ for the critic. In addition, the chosen batch size was 64. Even if not entirely correct from a theoretical point of view, we set a discount factor on 0.99 for all experiments. In addition, we set up an early stopping strategy for the network to stop training when it achieves an average reward of 200 over the previous 50 episodes.

It is worth highlighting that the learning rate for the actor ($5e-5$) is smaller than the learning rate for the critic ($5e-4$). This is in general beneficial for actor-critic methods, as the critic manages to learn the value of the actor, which changes slow enough to be tracked by the critic.

In order to have an actor that is able to explore the state-action space it is necessary to add a noise to the action vector it produces. The type of noise that we used was a *Ornstein-Uhlenbeck* process:

$$n_t = -\mu n_{t-1} + w_{t-1}$$
$$w_t \sim \mathcal{N}(0, \sigma I_m)$$

with $n_0 = 0$.

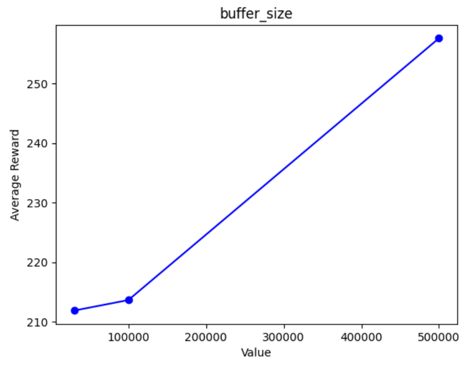
After some experimentation we noticed that with the initial parameters we were using the network was stuck and not learning to land correctly. Therefore , we decided to increase the noise for making it able to further explore the environment. Next, we had a problem as a large noise did not allow the network to remain still on the goal. As it arrived to the goal the network learnt to expect some random noise to push it so it tried to counter-back that noise even when it was absent at test time. Therefore, we proposed to use a noise which variance decreases linearly on time. We also tried a noise that after the network achieves a certain reward (so it has explored enough) reduces its variance to a lower value to allow the network to learn without such noise. We call the technique of decreasing the variance after a reward is achieved over the last N episodes *Noise Reduction*. For our experiments, we always averaged over the last $N = 30$ episodes and we decreased to a certain σ_f defined on the noise generator.

The actor and critic target networks must be updated every certain number of steps. From initial experimentation, we observed that using a low value as 2 leads to unstable training, therefore we also tried to test a value of 4. The update is done using a *soft-update*.

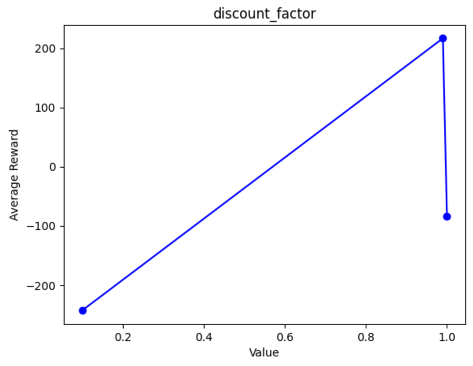
To find the best configuration of parameters we performed a grid search as shown in table 2. We found that the configuration that achieves the best result has a target update frequency of 4, a buffer size of 30000, and reduces the noise. The best noise generator is the one that starts with a variance of $\sigma_i = 2$ and does not decrease it linearly. The network trained in such way tested over 50 episodes achieves an average reward of 254.56 ± 11.56.

2.3 Task 2e

The process of training such a network is shown in fig. 8. We can see how the network during the first episodes performs very bad and flies erratically. During the successive episodes it learns how



(a) Buffer Size



(b) Discount Factor

Figure 9: Effect of the replay buffer size (a) and discount factor (b) on the performance of DDPG for the lunar lander.

to land and by the end, it learns how to land in the right spot. At the beginning of the training, the number of steps per episode is small as the spaceship crashes relatively quickly. As we can see the number of steps per episode increases when the agent learns how to fly and to land. The performance when we eliminate the noise is very satisfactory.

Using the obtained optimal hyper-parameters, we analyzed the effect of the discount factor on the performance of DDPG. fig. 9b shows the results for $\gamma = 0.1$, $\gamma = 0.99$, and $\gamma = 1$. Even though $\gamma = 1$ is theoretically correct for actor-critic methods, in practice $\gamma = 1$ does not solve the lunar lander. A discount factor $\gamma < 1$ introduces a bias but reduces the variance by acting as a regularizer (randomness from future rewards have less impact). Thus, in practice, a discount factor $\gamma = 0.99$ provides a better trade-off between bias and variance of gradient samples than $\gamma = 1$. On the other hand, a too small discount factor $\gamma = 0.1$ discounts future rewards too much and results in the agent not learning to land. The discount factor $\gamma = 0.99$, instead, represents a good choice and makes Proximal Policy Optimization (PPO) solve the lunar lander.

We explored also the size of the replay buffer on DDPG using the values $M = \{30000, 100000, 500000\}$. The results are shown in fig. 9a. In all the cases we filled the buffer to 30000 instances. It would be interesting to observe if the networks have *catastrophic forgetting* in different ways. We also tested a small buffer size of 500,which hinders the ability to learn. However, stability of training in all cases is stable. We show the learning curve for the small buffer in Figure 10.

2.4 Task 2f

We analyzed the critic and the actor of the trained DDPG by considering the restricted state $s(y, \omega) = (0, y, 0, 0, \omega, 0, 0, 0)$. In particular, by varying the two degrees of freedom of the restricted states, we evaluated the value function under the actor's policy, which can be computed from the critic ($V_\omega(s) = max_a Q_\omega(s, a)$), and the side engine control, provided by the actor ($\pi_{\theta,2}(s)$). The resulting 3D plots are illustrated in fig. 11.

The shape of the value function under the actor's policy is shown in fig. 11a. Our interpretation of it is as follows. As we said for DQN (see section 1.3, the lunar lander environment provides an incremental reward $r(s, a)$ by comparing the new state s' with the current state s. Thus, the more tilted the lander is at the start, the higher total reward is available to collect by adjusting the tilt and then landing. At the extreme, if we fix the height ($y = k$) and consider the states $s(\omega) = (0, k, 0, 0, \omega, 0, 0, 0)$, the minimum value is in $\omega = 0$, where the lander starts already aligned with the landing pad. However, unlike DQN, we have now a more symmetric value function, which is what we expect for the lunar lander problem. We can observe that after a certain threshold of tilt ($|\omega| > k$), the value function starts decreasing, which suggests the actor is not able to recover from very tilted initial states.

The shape of the side engine control, which is provided by the actor, is shown in fig. 17b and is very intuitive. Since the plot considers the lander with horizontally position aligned with the landing pad ($x = 0$), the agent needs to use the side engine only if the lander is tilted. The agent essentially tries

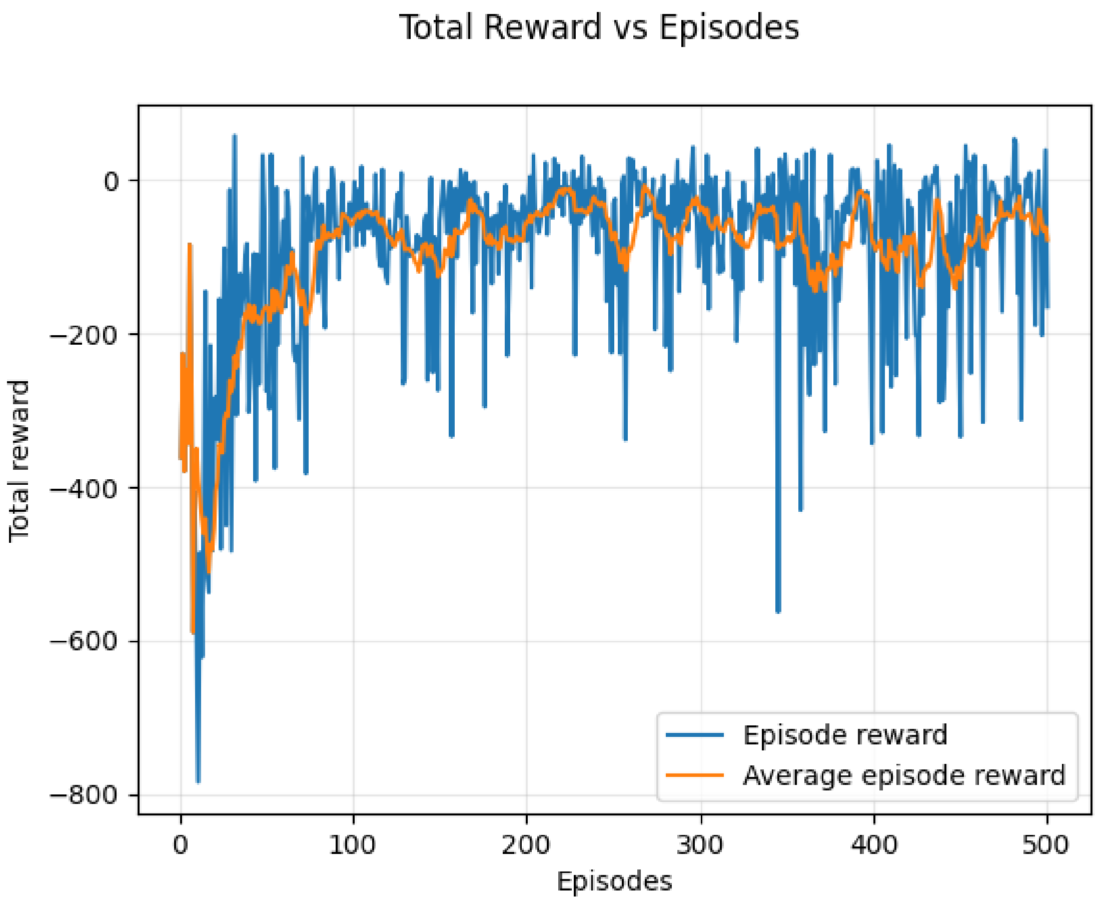
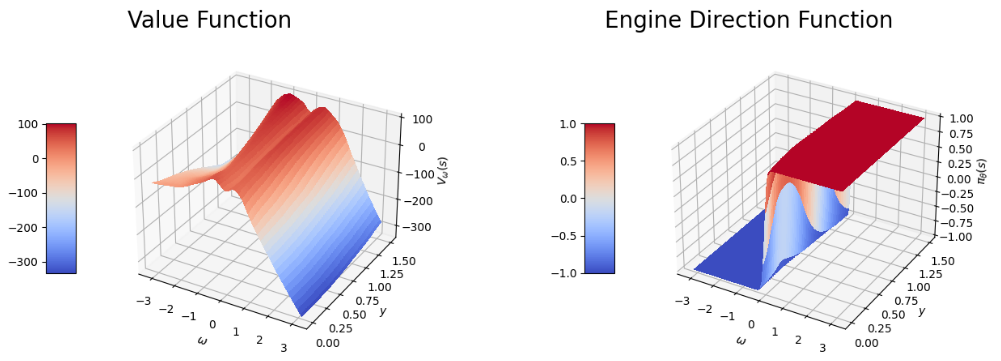


Figure 10: Learning Curve of training with small buffer of 500



(a) Value Function of the agent after training. Obtained using the critic network.

(b) Engine Direction. Obtained using the actor netowork

Figure 11: Representation of the learnt functions by the optimal agent.

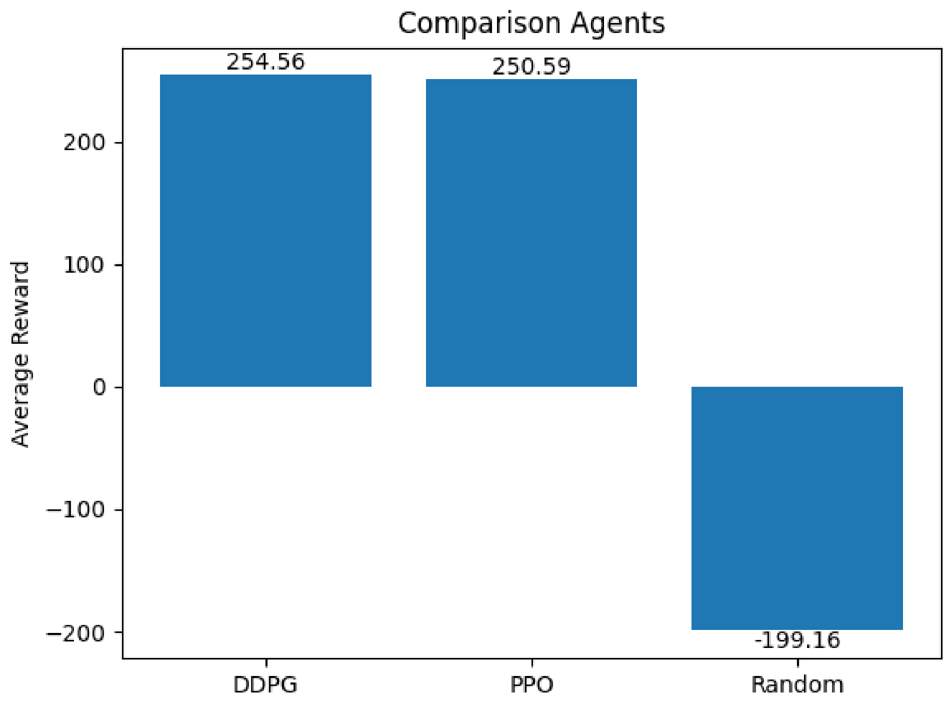


Figure 12: Performance comparison of PPO, DDPG and random agent.

to adjust the tilt of the lander to make it horizontal. In particular, the agent fires left ($\pi_{\theta,2}(s) < 0$) when the lander is tilted to the left ($\omega < 0$), and vice versa for the right.

2.5 Task 2g

We compared the performance of DDPG against a naive agent taking random actions, which serves as a baseline. The result is shown in fig. 12. As expected, the DDPG agent collects a much higher reward than the random agent, confirming once again that the algorithm learned from the sample data.

3 Problem 3 - PPO

In this section, we describe our solution to the lunar lander problem with continuous action space using PPO, an on-policy deep RL algorithm improving stability of traditional actor-critic methods by constraining the policy updates. Additionally, we analyze the actor and the critic resulting from training and explore the effect of some key hyperparameters for PPO on the training procedure.

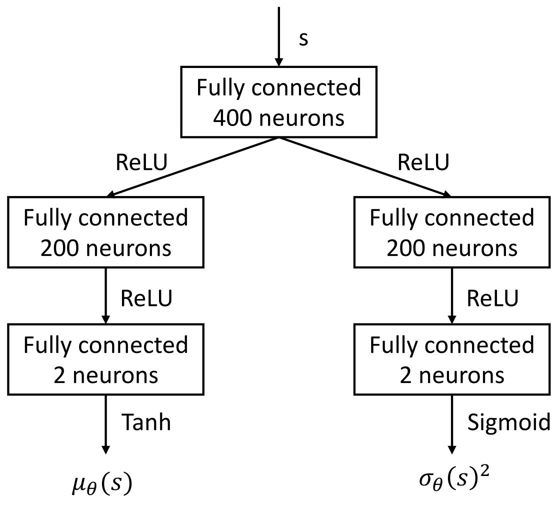
3.1 Task 3b

As mentioned in section 1.1, the target network is needed for RL algorithms combining the following three elements: function approximation, bootstrapping, off-policy. PPO does not need a target network, as it is an *on-policy* method. Furthermore, we used the Monte Carlo version of PPO, thus the critic does not even use bootstrapping.

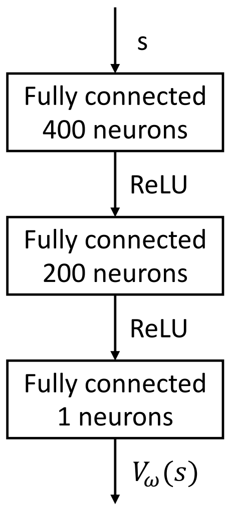
PPO is an on-policy method because it improves the same policy π_θ which it uses to sample experiences from the environment. In particular, the critic learns $V_\omega(s) \approx V^{\pi_\theta}(s)$, while the actor improves the current policy π_θ using data sampled while using π_θ (in addition to the critic). Exploration is intrinsically achieved, as the policy is randomized.

Table 3: Hyperparameters to solve the lunar lander with PPO.

Hyperparameter	Value
T_E (n. episodes)	1600
γ (discount factor)	0.99
M (n. epochs per update)	10
ϵ (policy ratio clip)	0.2
Actor learning rate	10^{-5}
Critic learning rate	10^{-3}
Optimizer (actor & critic)	Adam
Max. gradient norm (clipping)	1
Reward for early stop	250


s
Fully connected 400 neurons
ReLU
ReLU
Fully connected 200 neurons
Fully connected 200 neurons
ReLU
ReLU
Fully connected 2 neurons
Fully connected 2 neurons
Tanh
Sigmoid
$\mu_\theta(s)$
$\sigma_\theta(s)^2$

(a) Actor


s
Fully connected 400 neurons
ReLU
Fully connected 200 neurons
ReLU
Fully connected 1 neurons
$V_\omega(s)$

(b) Critic

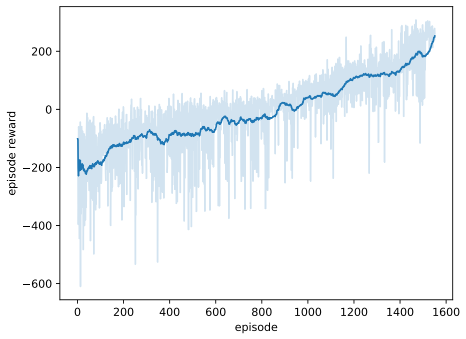
Figure 13: Architecture of actor and critic used to solve the lunar lander with PPO.

Sample complexity is an interesting performance metric for on-policy methods. On-policy methods improve the policy used to sample data from the environment. Therefore, the higher sample complexity, the more time is needed to learn a policy which solves the problem. However, a more interesting metric for on-policy methods is the regret, which measures how much reward is lost during training.

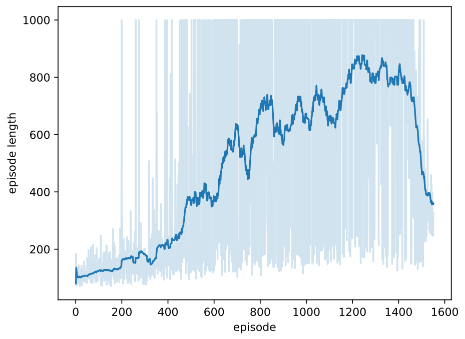
3.2 Task 3d

We solved the lunar lander with PPO by using the hyperparameters recommended in the lab instructions. Such hyperparameters are reported in table 3, while the layouts of actor and critic are shown in fig. 13. Besides the suggested hyperparameters, we chose the Adam optimizer and a maximum gradient norm 1 for gradient clipping, as they are common choices that usually perform well. Also, we trained for $T_E = 1600$ episodes but early stopped the training when the average episode reward over the last 50 episodes was above 250. We did not perform a grid search, as the recommended hyperparameters solved the lunar lander.

The randomized policy was modeled as a multi-variate Gaussian distribution with stochastically independent dimensions (diagonal covariance matrix), and the policy was parametrized by using mean and variance for each action dimension. As a consequence, as shown in fig. 13, the actor is a DNN that outputs mean and variance for each action dimension. The first layer of the actor is shared by the two successive branches and is supposed to extract features from the state that can be useful to compute both mean and variance. The successive layers, instead, specialize for either mean or variance and use the features extracted by the first shared layer. The activation function of the mean

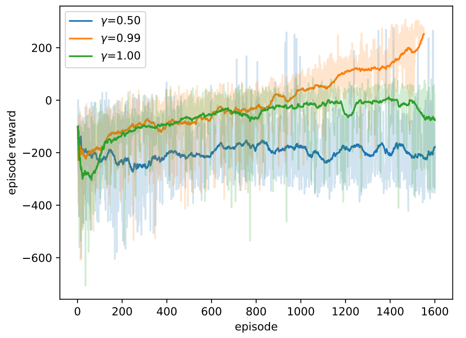


(a) Episode reward

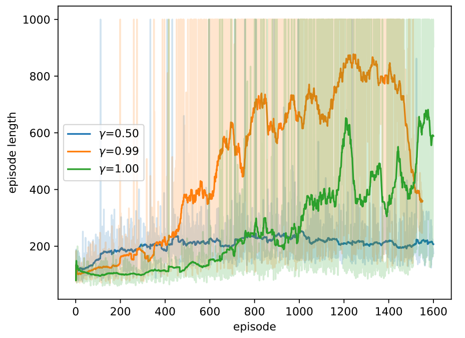


(b) Episode length

Figure 14: Learning curves of PPO solving the lunar lander.



(a) Episode reward



(b) Episode length

Figure 15: Learning curves of PPO with varying discount factor γ.

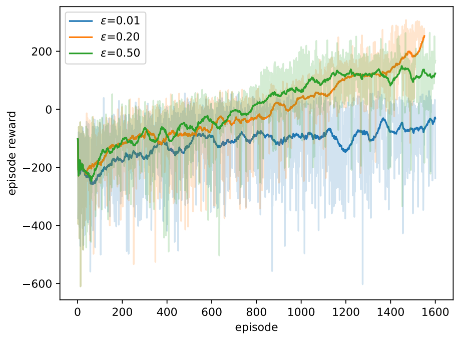
is a *tanh*, as the actions in the lunar lander problem are in $[-1, 1]$. The activation function of the variance is, instead, *sigmoid*, so as to have variance in $[0, 1]$.

In PPO, updating the actor less frequently than the critic results in a more stable training, as the estimator of the policy gradient would use a more precise estimate of the advantages. In our settings, we achieved this in a soft way by selecting the learning rates properly. In fact, the learning rate for the actor (10^{-5}) is much smaller than the learning rate for the critic (10^{-3}).

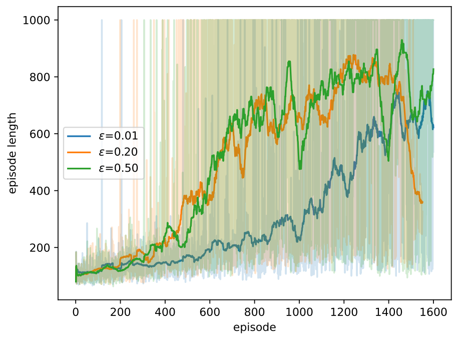
3.3 Task 3e

We trained PPO with the hyperparameters mentioned in section 3.2. The episode reward and episode length during the training process are shown in fig. 14. The agent reaches an average reward of 250 and stops before the limit of $T_E = 1600$ episodes. We can observe that, for the first 1200 episodes, the agent learns to fly and avoid crashing (episode reward below 200 and increasing episode length). After that, the agent learns to land without keeping flying (high episode reward around 200 and decreasing episode length). The learning process is quite stable, as shown by the curve of the average episode reward. We expected this result, as PPO prevents large updates of the policy by using the ratio of new policy and old policy in the surrogate objective function.

We investigated the effect of the discount factor on the training process in PPO. The results are shown in fig. 15. The same arguments made in section 2.3 for DDPG hold, and we repeat them for convenience. Even though $\gamma = 1$ is theoretically correct for actor-critic methods, in practice $\gamma = 1$ does not solve the lunar lander. A discount factor $\gamma < 1$ introduces a bias but reduces the variance by acting as a regularizer (randomness from future rewards have less impact). Thus, in practice, a discount factor $\gamma = 0.99$ provides a better trade-off between bias and variance of gradient samples

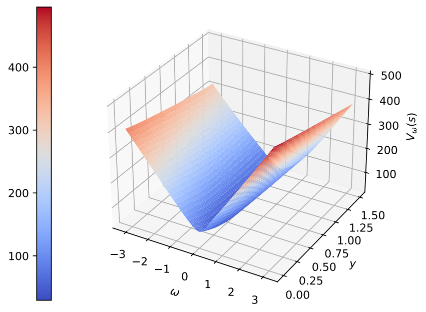


(a) Episode reward

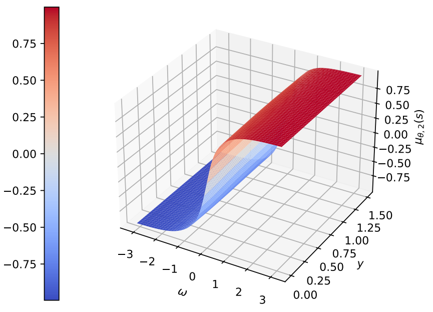


(b) Episode length

Figure 16: Learning curves of PPO with varying range for the policy ratio $[1-\epsilon, 1+\epsilon]$.



(a) Critic



(b) Actor's mean for side engine control

Figure 17: Value function from critic (a) and mean for side engine control from actor (b) for varying height (y) and tilt angle (ω) of the lander, with the rest of the state set to 0.

than $\gamma = 1$. On the other hand, a too small discount factor $\gamma = 0.5$ discounts future rewards too much and results in the agent not learning to land. The discount factor $\gamma = 0.99$, instead, represents a good choice and makes PPO solve the lunar lander.

We explored also the effect of increasing and decreasing the clipping range for the ratio between new policy and old policy, which is controlled by the parameter ϵ in PPO. The results are shown in fig. 16. It is evident that a too small clipping range ($\epsilon = 0.01$) has a negative impact, as it prevents the policy to change enough to solve the problem. On the other hand, a large clipping range ($\epsilon = 0.5$) allows larger policy updates but is less stable. Even though $\epsilon = 0.5$ initially outperforms $\epsilon = 0.2$ (roughly until episode 1200), it cannot solve the lunar lander in the long run (average episode reward less than 200). Instead, $\epsilon = 0.2$ represents the suit spot and provides a stable training process that ends up solving the lunar lander in less than 1600 episodes.

3.4 Task 3f

We analyzed the critic and the actor of the trained PPO agent by considering the restricted state $s(y, \omega) = (0, y, 0, 0, \omega, 0, 0, 0)$. In particular, by varying the two degrees of freedom of the restricted states, we evaluated the value function under the actor's policy, provided by the critic ($V_\omega(s)$), and the mean for the side engine control, provided by the actor ($\mu_{\theta,2}(s)$). The resulting 3D plots are illustrated in fig. 17.

The shape of the value function under the actor's policy, provided by the critic, is shown in fig. 17a. Our interpretation of it is as follows. As argued for DQN and DDPG (see sections 1.3 and 2.3), the lunar lander environment provides an incremental reward $r(s, a)$ by comparing the new state s' with the current state s. Thus, the more tilted the lander is at the start, the higher total reward is available to collect by adjusting the tilt and then landing. At the extreme, if we fix the height ($y = k$) and consider the states $s(\omega) = (0, k, 0, 0, \omega, 0, 0, 0)$, the minimum value is in $\omega = 0$, where the lander starts already aligned with the landing pad. We can notice that, unlike DQN and DDPG, the critic evaluates also very tilted states with high value, which suggests the actor is able to recover from such tilted initial states.

The shape of the mean for the side engine control, which is provided by the actor, shown in fig. 17b, is very intuitive and similar to the directional policy of DDPG (see section 1.3). We can make the same considerations. Since the plot considers the lander with horizontally position aligned with the landing pad ($x = 0$), the agent needs to use the side engine only if the lander is tilted ($|\omega| > 0$). We can indeed observe that the surface does not depend on the height. The agent essentially tries to adjust the tilt of the lander to make it horizontal. In particular, the agent fires left ($\mu_{\theta,2}(s) < 0$) when the lander is tilted to the left ($\omega < 0$), and vice versa for the right. Unlike DDPG, we can notice that the function is much smoother, which suggests PPO adapts the side engine throttle to the absolute value of the tilt angle.

3.5 Task 3g

We compared the performance of PPO against a naive agent taking random actions, which serves as a baseline. The result is shown in fig. 12. As expected, the PPO agent collects a much higher reward than the random agent, confirming once again that the algorithm learned from the sample data. The performance of PPO achieves a value around 250, according to the chosen early stopping threshold.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, 2018.