Miniproject - DDPG

Antoine Maier, Aude Maier

Introduction

In this project, we will explore the Deep Deterministic Policy Gradient (DDPG) algorithm, which is designed to handle continuous action spaces in Reinforcement Learning. We will use the Pendulum-v1 environment implemented in OpenAI Gym to implement the DDPG algorithm from scratch to solve the classical control problem of stabilising an inverted pendulum. Throughout the development, we will incrementally build the components of DDPG and explore the impact of target networks and Ornstein-Uhlenbeck noise on the learning process.

3 Heuristic Policy

In this work, the action space is renormalised to $[-1, 1]$ where -1 corresponds to the maximum torque in the clockwise direction and 1 to the maximum torque in the counterclockwise direction. This convention is used throughout the rest of this report.

First we will compare two policies: a random policy that selects actions uniformly at random from the action space, and a heuristic policy designed to accelerate the pendulum when on the lower half of the domain and deccelerate it otherwise.

The average return over 100 episodes and its standard deviation for the random policy is -1230 ± 270, which sets a baseline for the performance of the policies.

On Fig. 1, we see the average return over 100 episodes for the heuristic policy depending on the magnitude of the fixed torque. The performance increases linearly from -1190 ± 330 for a null fixed torque to -590 ± 200 for a fixed torque of 0.5, and then increases with a smaller slope up to -430 ± 110 for a fixed torque of 1. The exact performance varies from one run to another, but when the performance increases the standard deviation decreases. We observe that the heuristic policy performs better than the random policy for all nonzero values of the fixed torque.

The performance of these two policies can be compared to the next policies of this homework in Fig. 7.

4 Q-function of the heuristic policy

We now implement a critic, i.e. a Q-network, whose goal is to estimate the Q-function of the heuristic policy with torque intensity 1 (i.e. the value that achieved the highest cumulative reward in the previous section). We use a neural network with two hidden layers of 32 neurons each, with ReLU activation functions. It is updated using the semi-gradient of the mean squared error between the predicted and target Q-values.

In Fig. 2 we see the training curve with batches of size 128 and 1000 epochs. We observe a rapid increase of the Q-loss during the first episodes. This is due to the fact that the replay buffer is almost empty at that stage and therefore the network is trained on the same (or very similar) samples over and over. We will encounter this phenomenon for every network training in this work. After this initial peak, the loss decreases rapidely to a value of approximately 70 after 100 episodes. The training then slows down, with the loss reaching a value close to 50 after 1000 episodes.

We then plot and compare the Q-values given by the Q-network before and after training, for several values of torque and velocity, by means of heatmaps. The result is shown in the top two rows of Fig. 3, where a positive velocity corresponds to the pendulum moving in the trigonometric direction and a positive torque corresponds to a torque applied in the trigonometric direction.

The Q-values before training are almost null constants compared to after the training. In fact, if plotted on a different color scale, we would see that the Q-values before training are a mix of randomness and continuity of order $O(10^{-2})$, which is easily explained by the random initialization of the weights and the intrinsic continuity of the network, and it is certainly not due to the fact that the policy achieves a high return and that the Q-network is able to predict it.

The Q-values after training are 4 orders of magnitude larger (in absolute value). The heatmap corresponding to velocity=0 and torque=0 shows an accurate representation of the reward distribution, i.e. the closer you are to $\alpha = 0°$, the more reward you get and the easier it will be to be close to $\alpha = 0°$ in the future. A notable fact is that this representation appeared without any explicit training of the action corresponding to torque=0 but is an interpolation of the knowledge acquired for torque=± 1.

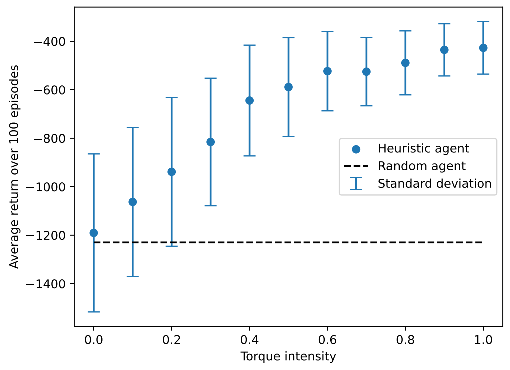

FIGURE 1 : Average return of the heuristic agent and the random agent over 100 episodes. The heuristic agent is trained with 11 different torque intensities between 0 and 1.

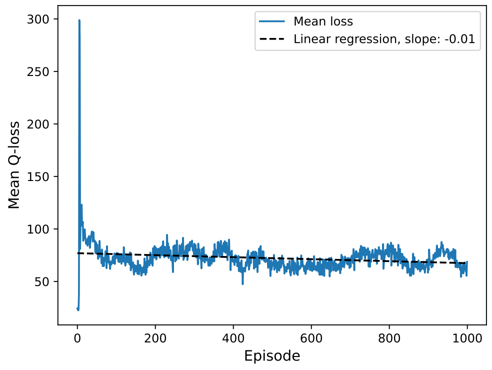

FIGURE 2 : Mean Q-loss per episode during training of the Q-network for the heuristic agent. The slope of the linear regression line is -0.01.

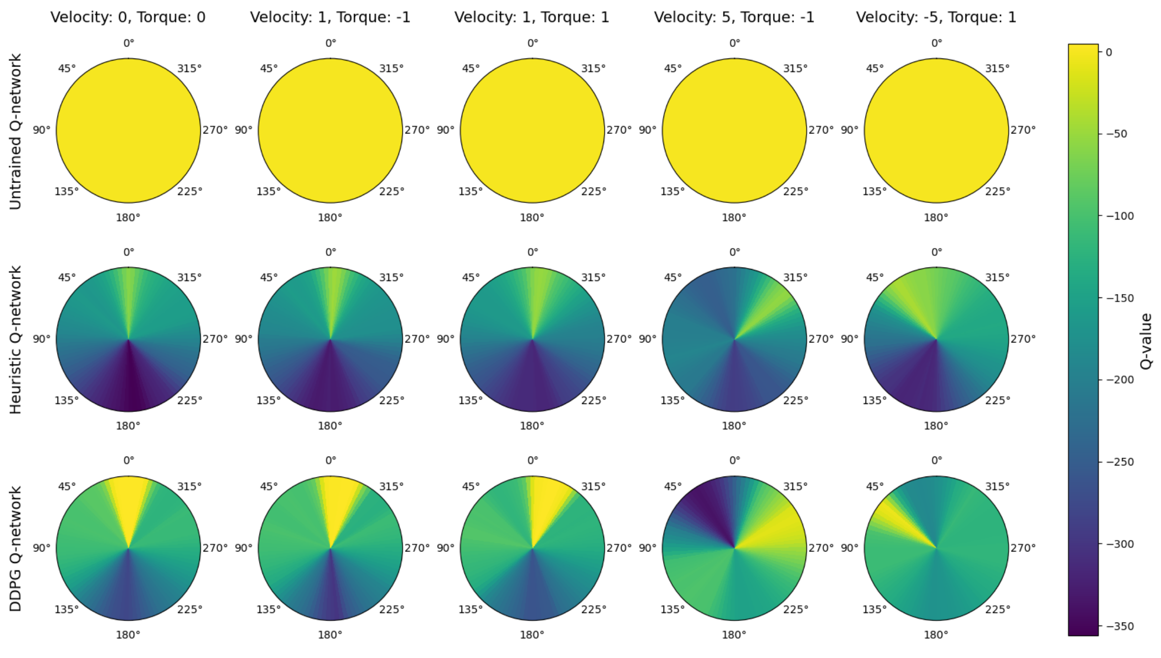

Figure 3: Q-value as a function of the position of the pendulum, for a given velocity and torque. The two first rows plot the Q-values computed by the heuristic Q-network before and after training, respectively. The third row shows the Q-values computed by the Q-network of the DDPG agent. For each row, five different velocity-torque pairs are considered.

The heatmap of velocity=1, torque=-1 is similar to the previous one, but with a slight rotation in the clockwise direction. This is due to the fact that with this (small) velocity, the heuristic policy will decelerate the pendulum to try to balance it in the inverted position, thus leading to a high reward for a small negative angle. We note that the lower left and right quadrants are situations that the Q-network has never experienced, since the heuristic policy is perfectly deterministic. With velocity=1 but torque=1, these are the two upper quadrants that are unexperienced situations for the Q-network. However, the heatmap is very similar to the previous one, hence it looks as if the Q-network has inferred the Q-values for the unexperienced situations from the experienced ones. Unfortunately, we can doubt that in such a situation of positive velocity and accelerated pendulum the Q-values would be as high as claimed by the Q-network.

Then, the heatmap with velocity=5 and torque=-1 resembles the one with velocity=1 and torque=-1, but with the high-reward circular sector rotated in the clockwise direction. This is due to the fact that with a greater velocity, the heuristic policy will be able to balance the pendulum if it decelerates it earlier. This time, we chose to change sign of both the velocity and the torque. We expect the heatmap to be the mirrored image of the previous one, which holds up to a certain extent. We can imagine that the differences between the two heatmaps are due to the fact that the Q-network has not been perfectly trained.

5 Minimal implementation of DDPG

In this section, we replace the heuristic agent with an agent equipped with a policy network that compute the next action to take based on the current state. We use a neural network with two hidden layers of 32 neurons each, with ReLU activation functions, and a tanh activation on the output layer, ensuring that the returned action is between -1 and 1. The policy network is updated to maximize the mean Q-value. To enable better exploration of the state space when generating new transitions, a small Gaussian noise is added to the action returned by the policy network.

In Fig. 4 we can see the Q-network and policy network training curves with batches of size 128 over 1000 episodes, as well as return per episode. We observe that the trainings of the two networks are correlated. Initially, both networks overfit and then rapidly decrease their loss in $\sim$ 100 episodes. Afterward, the curve plateaus for around 100 episodes, and finally, both networks learn slowly but steadily until the loss reaches $\sim$ 0 at episode 500. We observe that the networks indeed learned well because the return per episode increases steadily until episode 500, after which it plateaus. This indicates that stopping the training at episode 500 would have been a good choice.

The trained policy achieves an average return and standard deviation over 100 episodes of -180 ± 100, which is significantly better than the heuristic policy. The comparison can be seen in Fig. 7.

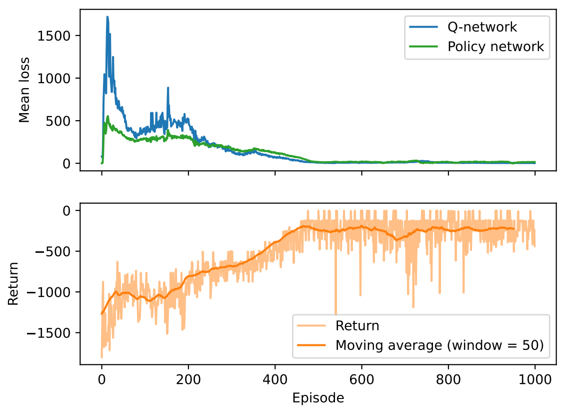

Figure 4: Training curves for the Q-network and the policy network of the minimal DDPG agent. The return is also plotted with a moving average smoothing.

We also want to compare the Q-values given by the Q-network trained on the heuristic policy and the agent trained with a policy network. The results are shown in the two bottom rows of Fig. 3. Firstly, we observe that with the policy network, the high-reward sector is much larger than with the heuristic policy. This is due to the fact that the policy network is more versatile and can react to a wider range of situations. In particular, for a null or small velocity, the value of the torque has only a small influence on the location of the high-reward sector because if the torque is not optimal with respect to the velocity, the policy network will compensate for this in the next step, i.e almost immediately. This is not the case with the heuristic policy, which is deterministic.

With velocity=5, we would again expect the two heatmaps to be the mirror versions of each other. Even though the high-reward sectors are indeed mirrored, with negative torque, it is larger and there exists a region with low-reward in the upper-left region, which is nonexistent in the other heatmap. This might be due to the fact that a velocity of magnitude 5 in the upper quadrants was a rare situation during the training, and thus the Q-network has not been able to learn the Q-values well for these situations. This is an example of missgeneralization[1].

[1]Shah, Rohin, et al. *Goal Misgeneralization: Why Correct Specifications Aren't Enough For Correct Goals.* 2 Nov. 2022, https://arxiv.org/pdf/2210.01790.pdf.

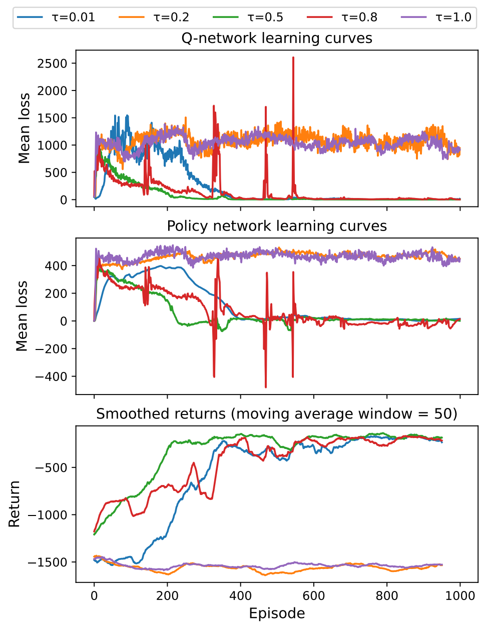

FIGURE 5 : Training curves for the Q-network and the policy network, and return per episode, for the DDPG agent equipped with target networks, for different values of update parameter τ.

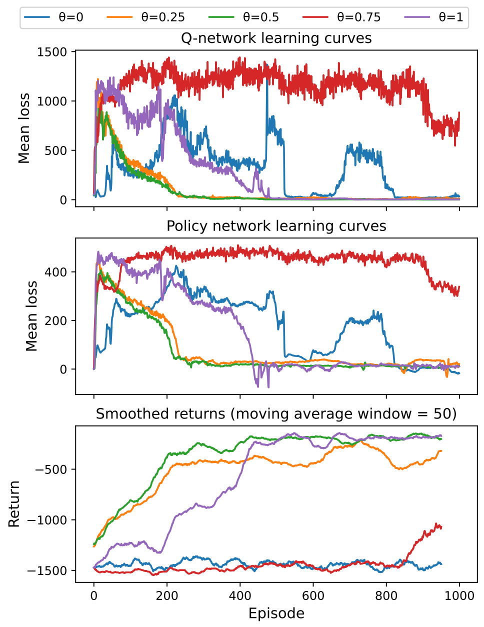

FIGURE 6 : Training curves for the Q-network and the policy network, and return per episode, for the DDPG agent equipped with target networks (update parameter $\tau = 0.5$), and Ornstein-Uhlenbeck noise, for different values of correlation parameter θ.

6 Target networks

When experimenting with the first implementation of DDPG above, we noticed that depending on the choice of the seed, the training was sometimes unstable. This motivates the use of target networks, which are copies of the actor and critic networks that are "soft updated". This means that at each step, they are updated with a weighted average between the weights of the target network and the weights of the original network. In other words, each weight w_{target} of the actor/critic target network are updated using this equation:

$$w_{\text{target}}^{(t+1)} = \tau \cdot w_{\text{original}}^{(t)} + (1 - \tau) \cdot w_{\text{target}}^{(t)}$$

with $\tau \in (0, 1)$. The target networks are used instead of the true networks in the computation of the target Q-value.

We implemented this modification, and we see in Fig. 5 the training curves of the Q-network and policy network, as well as the return per episode, using batch size 128 over 1000 episodes, for 5 values of τ.

We see that the stability of the training varies with the value of τ. For $\tau = 0.2$ and $\tau = 1$ (which corresponds to the original DDPG algorithm), the training does not converge, with the Q-network loss oscillating around 1000. For $\tau = 0.01$, the initial increase of the loss, due to overfitting, is slower because, by design, a small update parameter slows down the update of the networks. Then, it shows a similar behaviour as the $\tau = 0.2$ case up to episode 200 until the losses start decreasing to approach zero. Hence, despite the observed convergence, it looks likely that with another seed, the losses would have continued oscillating. For $\tau = 0.8$, the training curves show several abrupt peaks during the first 600 episodes. The most stable training is obtained for $\tau = 0.5$, with the losses decreasing smoothly and the return increasing faster than for the other values of τ. Therefore, it looks like a sweet spot between larger values that do not allow a sufficient effect of the target networks and smaller values that update the networks too slowly, making them unable to converge.

The resulting average return and standard deviation over 100 test episodes for each of the five trained policies are: -170 ± 90 ($\tau = 0.01$), -1510 ± 110 ($\tau = 0.2$), -140 ± 80 ($\tau = 0.5$), -150 ± 80 ($\tau = 0.8$), and -1480 ± 60 ($\tau = 1$). In Fig. 7, we compare the performance of the policy trained with $\tau = 0.5$ with the other policies of this work.

7 Ornstein-Uhlenbeck noise

Adding noise to the action helps exploration of the state space, which is necessary to learn a good policy. However, adding uncorrelated noise to subsequent actions is not very effective in practice, as it leads to trajectories close to the deterministic one. To improve the exploration of the algorithm, we replace the Gaussian noise with a (simplified) Ornstein-Uhlenbeck noise defined by the iterative equation:

$$\text{noise}^{(n+1)} = (1-\theta) \cdot \text{noise}^{(n)} + \mathcal{N}(0, \sigma^2)$$

with $\theta \in (0, 1)$.

Fig. 6 shows the training curves and return per episode during training of a DDPG agent with target networks and Ornstein-Uhlenbeck noise, using batch size 128 over 1000 episodes. The training was done for 5 different values of θ.

We observe that with $\theta = 0$, the training looks chaotic. This can be explained by a probabilistic phenomenon. The noise added at the n-th step can be written as

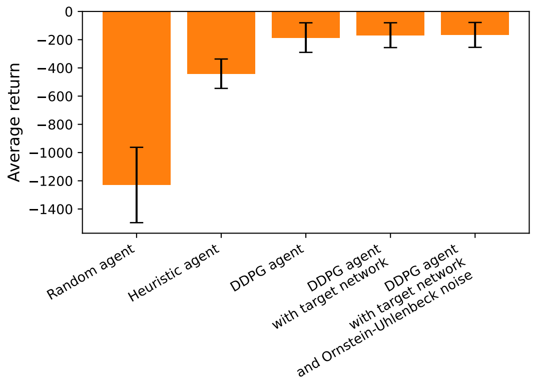

Figure 7: Comparison of average return and standard deviation over 100 episodes for the random policy, the heuristic policy with best torque intensity (=1), DDPG agent, DDPG agent with target network with best update parameter ($\tau = 0.5$), and DDPG agent with target network ($\tau = 0.5$) and Ornstein-Uhlenbeck noise with best correlation parameter ($\theta = 0.5$).

$$X_n \sim \mathcal{N}(0, \sigma^2) + \sum_{i=1}^{n-1} (1-\theta)^{2i} \mathcal{N}(0, \sigma^2) \overset{d}{=} \mathcal{N}\left(0, \sigma^2 \frac{1-(1-\theta)^{2n}}{1-(1-\theta)^2}\right)$$

for $\theta > 0$ and $\mathcal{N}(0, n\sigma^2)$ for $\theta = 0$. Hence, in the case $\theta = 0$, the variance grows linearly with n and, already at the 25th step of an episode, the probability for the noise to be greater than 1 in absolute value, causing the action to be likely clipped to the boundaries of the action space (± 1), is ~ 0.5, and this probability will continue to grow during the episode. The noise thus completely dwarfs the action chosen by the policy. As a result, the vast majority of actions taken during training have a value ± 1, chosen randomly. Under this condition, it is no surprise that the agent is unable to learn and achieve good performances.

With a non-null θ, however, while still growing with n, the variance is bounded due to the exponential decay of the cumulative noise history, and the distribution tends toward $X_{n\to\infty} \sim \mathcal{N}\left(0, \frac{\sigma^2}{1-(1-\theta)^2}\right)$. In the case $\theta = 0.25$, even in the limit $n \to \infty$, the probability of $|X_\infty| > 1$ is ~ 0.02.

On the other hand, when using large θ values (0.75 and 1), the noise is (almost) uncorrelated, and we recover the situation of the previous section. However, even though we used the best update parameter found in section 6 ($\tau = 0.5$), we observe that the training can still fail to converge.

The fastest learnings are achieved by small but non-zero values of θ (0.25 and 0.5). This shows the crucial role played by the value of the correlation parameter θ in effectively introducing correlation in noise while ensuring that the policy's decisions remain significant.

The resulting average return and standard deviation over 100 test episodes for each of the five trained policies are: -1110 ± 370 ($\theta = 0$), -170 ± 100 ($\theta = 0.25$), -150 ± 80 ($\theta = 0.5$), -970 ± 470 ($\theta = 0.75$), and -150 ± 90 ($\theta = 1$). In Fig. 7, we compare the performance of the policy trained with $\theta = 0.5$ with the other policies of this work.

Conclusion

In this project, we implemented the DDPG algorithm from scratch to solve the classical control problem of stabilising an inverted pendulum. First, we trained a Q-network on a heuristic policy, and then we trained an actor-critic network using the DDPG algorithm. We then studied the importance of using target networks to stabilise the learning process and found a sweet spot for the update parameter of the target network, $\tau = 0.5$. We also experimented with using Ornstein-Uhlenbeck noise to explore the action space and explained mathematically why a null correlation parameter prevents the agent from learning. Finally, we compared the performance of all the policies used in this project in Fig. 7. We observe that the DDPG algorithm is better than the heuristic policy, and although adding target networks and Ornstein-Uhlenbeck noise does not lead to better performance, they help stabilise and accelerate the learning process, respectively.