Control in a continuous action space with DDPG

Quinten Dewulf, Zimi Liu

June 4, 2023

1 Introduction

For this mini-project, we'll try to tackle the well-known inverted pendulum problem using the DDPG algorithm. The goal of the algorithm is to produce a policy that given any state of the pendulum returns a suitable torque to be applied to the pendulum in order to get it balanced upside down. DDPG belongs to the class of actor-critic reinforcement learning algorithms as it possesses a neural network for estimating Q-values (critic) and an additional network that acts as the policy (actor). The critic network is trained using the TD error calculated with predicted Q-values and the observed reward. DDPG is an off-policy algorithm because this error eventually has to become zero for every transition (s_j, a_j, r_j, s_{j+1}) regardless of which policy was used to predict a_j, so we sample from a replay buffer with stored transitions during training. Because the policy network is configured to output one or more real numbers its output can be interpreted as a continuous action, making DDPG a well-suited algorithm for reinforcement learning problems in continuous space.

2 Heuristic Policy

The first policy that we examined was the random policy. This policy gave wildly varying results, for the average cumulative reward of 10 episodes we noted -1356.861. However, different runs of 10 episodes still gave quite different average cumulative rewards (but -1356.861 certainly has the right order of magnitude).

Hereafter we implemented the heuristic policy as described in the assignment. To determine the best possible value of the amplitude of the fixed torque we ran a small 1D-grid search. It appears that a higher torque amplitude works better for this policy, but the effect lessens when 1 is approached. We settled on the value 0.95 for our fixed torque amplitude and noted that the average cumulative reward of 10 episodes was -442.855. This means that our simple heuristic policy scores approximately three times better than the random policy.

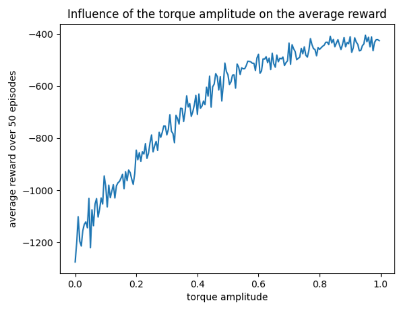

Figure 1: Influence of torque amplitude

3 Q function of the heuristic policy

For this part, we'll use an agent provided with the previously mentioned heuristic policy, a neural network for the Q function and a replay buffer to store/sample past transitions. We attempt to train the Q network to output the correct Q value of the heuristic policy for a given state and action. In order to do this we must minimise the MSE of the TD errors:

$$\frac{1}{N}\sum_{n=1}^{N}(r_j + \gamma Q_\theta(s_{j+1}, \pi_{heuristic}(s_{j+1})) - Q_\theta(s_j, a_j))^2 \quad \text{for all } N \text{ transitions } (s_j, a_j, r_j, s_{j+1}) \text{ in the replay buffer}$$

This minimisation will be done using stochastic gradient descent, sampling batches of 128 transitions from the buffer at a time.
In order to accommodate the part of the assignment that specifies that target $r_T + \gamma Q_\theta(s_{T+1}, \pi_{heuristic}(s_{T+1})) = r_T$ when $T =$ `max_it` we add one extra term to the loss function of the Q network and it becomes:

$$\frac{1}{N}\sum_{n=1}^{N}(r_j + \gamma Q_\theta(s_{j+1}, \pi_{heuristic}(s_{j+1})) * \mathbb{1}_{j\neq\texttt{max\_it}} - Q_\theta(s_j, a_j))^2$$

In order to link this addition with the theory of Markov decision processes one could argue that the goal of this inverted pendulum problem is to find a policy to optimize the horizon-`max_it` values since every episode always truncates at `max_it` steps. If we view this as such a finite-horizon problem then it indeed follows that $Q_\theta^{(1)}(s_{\texttt{max\_it}}, a_{\texttt{max\_it}}) = r_{\texttt{max\_it}}$. However, for the rest of the transitions, we assume to be working with an infinite-horizon problem since there is no mention of a time variable t.

3.1 Training of the Q network

We ran the training algorithm for the Q network for 1000 episodes and the specified hyperparameters. We first encounter a tremendous increase in the network loss before observing a slow and erratic decrease. A possible hypothesis for this behaviour is that initially the network only produces Q-values with very small absolute values (due to the network being randomly initialised with small weights) so for the first steps the TD error is approximately equal to the reward squared (and the rewards have quite small absolute values). After this initial phase with relatively low loss the network weights get updated based on the drawn batches of transitions. This means that the Q network will output Q-values with higher absolute values but because we're using stochastic gradient updates the network is not yet able to generalise to unseen transitions and every new batch will have a higher loss. With further training the network starts to be able to generalise to unseen transitions and the last phase of the training thus consists of a slow decrease of the loss value.

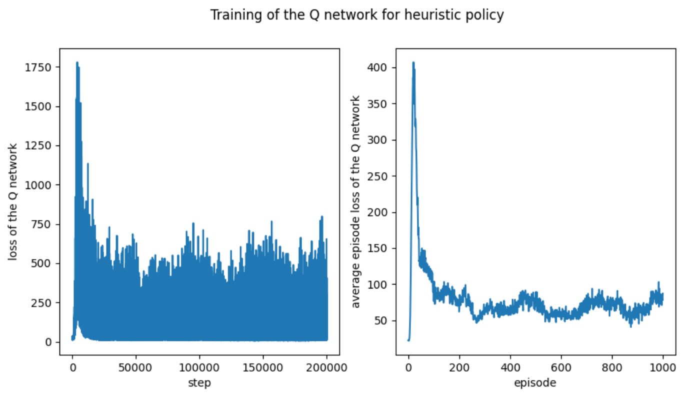

Figure 2: Evolution of the loss of the Q network through training

3.2 Polar plots of the Q function of the heuristic policy

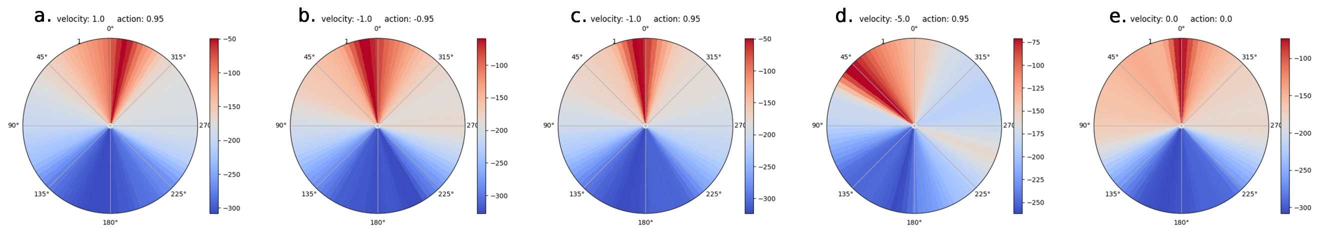

Figure 3: Polar plots of the Q function of the heuristic policy for 5 different pairs of angular velocity and torque values

Because the Q network has only been trained with transitions that have actions $\in \{-0.95, 0, +0.95\}$ we've limited ourselves to this exact range for the actions when making visualisations of the function. Plot a. and plot b. visualise the case where velocity and chosen action have the same sign, because the heuristic policy only lets this happen when the pendulum is in the lower half of the unit circle we should only pay attention to this part of those plots. After all, the network has never been trained with transitions that contain the pendulum in the upper half and velocity and torque sharing the same sign, so we can't expect it to output meaningful values for these situations. A similar remark is to be given about the lower half of plots c. and d.
Plots a. and b. show that when the pendulum is in the lower half of the unit circle and has a relatively low speed,

its best position is to be as horizontal as possible. This way it can swing through the whole lower part, picking up enough speed to make it far into the upper part or shortly swing in the upper part, quickly switch direction and then pick up speed by swinging through the lower part. They also show that the Q function has successfully grasped the inherent symmetry of the problem. The only difference between a. and b. is the direction of the pendulum and this is reflected in the fact that these plots are almost perfectly mirrored. Plots c. and d. both represent the situation where the pendulum is swinging clockwise, but d. has a much greater speed than c. We see that the Q function successfully learned that the optimal state for situation d. is further away from the target state (highest position) than the optimal state for situation c. This is because the pendulum in situation d. will need more time to slow down under the heuristic policy. Lastly the Q network has successfully identified that in the situation where the pendulum has no velocity or torque (e.) the best position to maximise long-term reward is the target position.

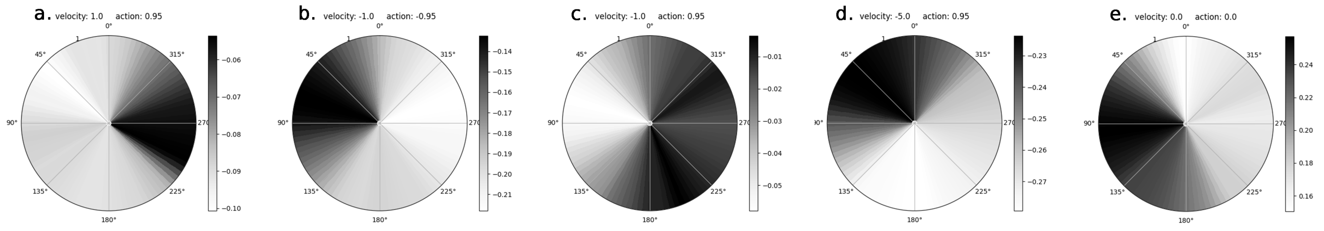

Figure 4: Polar plots of the untrained Q network for 5 different pairs of angular velocity and torque values

Visualisations of the untrained Q network show that it consistently outputs values with small absolute values that have no connection to the problem at hand.

4 Minimal implementation of DDPG

We now introduce the policy network π_ψ which we will train simultaneously with the Q network Q_θ using the minimal implementation of DDPG. The loss function of the policy network is the following:

$$-\frac{1}{N}\sum_{n=1}^{N} Q_\theta(s_j, \pi_\psi(s_j)) \quad \text{for all } N \text{ transitions } (s_j, a_j, r_j, s_{j+1}) \text{ in the replay buffer}$$

The minimisation of this loss will be done using stochastic gradient descent, sampling batches of 128 transitions from the replay buffer at a time.

4.1 Training of the minimal DDPG agent

We trained the minimal implementation of DDPG for 1000 episodes using the provided hyperparameters. Both the policy and Q network start with a low loss value, only for it to shoot up and then slowly and erratically decrease, with the policy network being more tempered. We can use the hypothesis for the behaviour of the loss of the Q network for the heuristic policy to explain the behaviour of the Q network again since both graphs look quite similar. This hypothesis also seems applicable to the policy network. Because the network is randomly initialised it will produce random actions with low absolute values in the first phase, leading to an overall low loss value. The next phase consists of the network producing actions with larger absolute values but not being able to generalise, leading to high loss values, lastly, the policy network slowly learns to generalise and the loss converges. The better the Q network becomes at predicting Q values of the policy network and the better the policy network becomes at being the optimal policy, the better the cumulative reward per episode gets, eventually becoming almost 0. After training we executed 100 test episodes with the trained policy and got an average cumulative reward of -155.720.

4.2 Polar plots of the Q function

Overall these plots are quite similar to the ones for the heuristic policy. One can again infer a swinging behaviour from plots a. and b. and observe that even the trained policy needs more time to slow down a pendulum with speed -5 than one with speed -1 from plots c. and d. The biggest visual improvement is that the optimal zones for the pendulum to be in (the red zones) consistently are broader than they were for the heuristic policy. This means

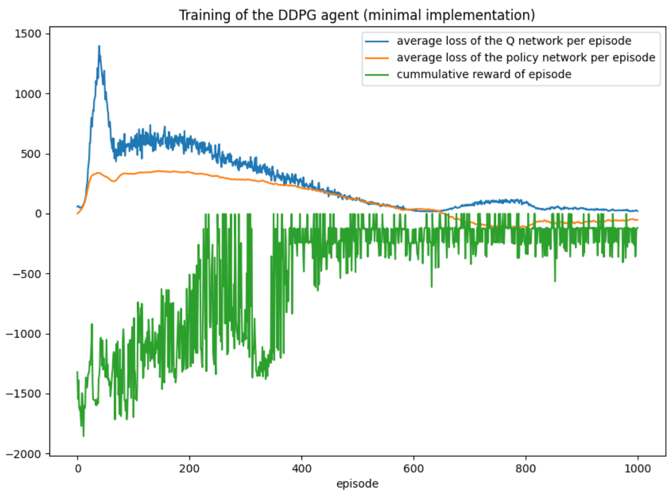

Figure 5: Training process of the minimal DDPG agent

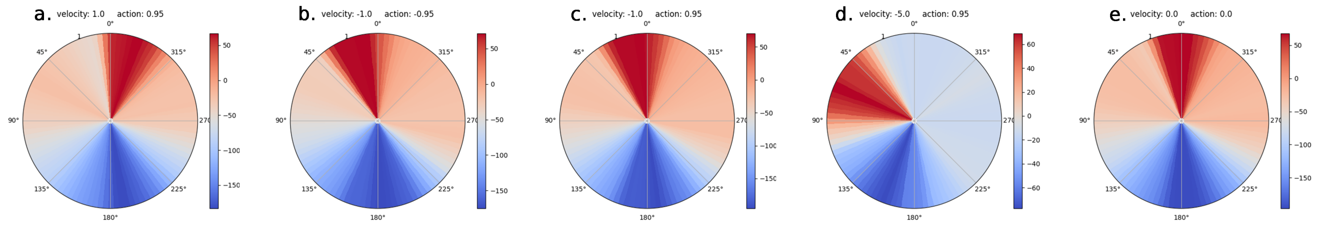

Figure 6: Polar plots of the Q network of the DDPG agent for 5 different pairs of angular velocity and torque values

that the trained policy is more flexible and can ensure a maximal reward for a larger amount of starting positions given a certain velocity and torque. Some other differences are that the transitions are less gradual and that the Q network has (incorrectly) learned that there can be positive Q values. That last change might not be so bad since it is only the relative value of Q values that matters for the policy optimisation.

5 Target networks

In the DDPG (Deep Deterministic Policy Gradient) algorithm, the target networks are introduced to stabilize the training of the policy and Q networks. To achieve this, we create a second set of networks, called the target actor and target critic, which are initialized with the same weights as the original actor and critic networks. The target networks' weights are updated using "soft updates" controlled by the hyper-parameter τ, taking a weighted average of the current actor and critic weights and the old target weights, which helps maintain smooth, stable training. The formula for the soft update is as follows:

$$\theta_{target} \leftarrow \tau\theta_{local} + (1-\tau)\theta_{target}$$

In the implementation, we added target actor and critic networks in the minimal DDPG agent and defined a method named `update_target_params()` to perform the soft updates. The training method was modified to use the target actor and critic to compute the critic loss and update the target parameters with the soft update rule. We then experimented with the DDPG agent having target networks for 5 different values of τ between 0.01 and 1.0 and executed 100 test episodes without adding noise to the actions.

5.1 Training of DDPG Agent with Target Networks

In the training process, we analyzed the impact of the soft update hyperparameter τ on the performance of the DDPG agent. The choice of τ is crucial in balancing the trade-off between the stability and speed of learning. A smaller value of τ results in slower updates, leading to increased stability but potentially making the learning process slower. On the other hand, a larger value of τ may cause faster convergence but can lead to instability in learning.

We examined the effects of varying τ for the values 0.01, 0.05, 0.1, 0.5, and 1. The following qualitative observations were made for each value:

- $\tau = 0.01$ and $\tau = 0.05$: The learning process was relatively stable due to the small τ, but the convergence was slow, taking a considerable number of episodes to achieve good performance. This is because updates to the target networks were slow, causing the learning improvements to propagate slowly through the networks.
- $\tau = 0.1$: The agent showed a balance between stability and speed of learning. Convergence occurred at an acceptable pace, and the learning process was relatively stable, making this τ a good candidate for further tuning and exploration.
- $\tau = 0.5$: At this value, we witnessed faster convergence, but at the cost of reduced stability in the learning process. Oscillations and occasional instability were observed in the learning curves, indicating that $\tau = 0.5$ might not be ideal for achieving the best balance between stability and speed.
- $\tau = 1$: With $\tau = 1$, the target networks were completely replaced by the current networks after each update, eliminating the soft update mechanism.

Based on these observations, it appears that a τ value of 0.1 provides a good balance between stability and speed of learning for the given problem. Under this case, we analyzed the learning curves and observed that the Q network loss has a more stable and consistent decrease when using target networks compared to the minimal implementation, as shown in the figure below.

The learning curve of the actor, as in the orange line, is very similar to the minimal implementation. However, the figure also shows that the actor's learning curve is smoother and more stable when using target networks compared, especially in the ending phase of the training.

The learning curve of the critic (blue line) is also less erratic compared to the minimal implementation. The figure shows that the critic loss has a more stable and consistent decrease when using target networks compared to the minimal implementation. In the second half of the training process, the fluctuation of the critic loss is relatively small, indicating that the critic has converged to a stable policy. An interesting finding is that at the beginning, the loss takes longer to decrease compared to the minimal implementation, showing that the target networks and soft update slow down the learning process at the beginning.

The figures illustrate that the critic loss has a more stable and consistent decrease when using target networks compared to the minimal implementation, with its maximum value significantly lower than the minimal implementation.

The cumulative reward throughout the training presents (green line) also follows a similar trend to the minimal implementation. However, at the ending phase, the cumulative reward has a smaller variance, which is about half of the original implementation.

5.2 Testing Results with target networks

In the testing phase, the tuned DDPG agent with target networks achieved an average cumulative reward of -136.54. This demonstrates the efficiency and stability introduced by the target networks, resulting in better overall control of the pendulum system.

6 Ornstein-Uhlenbeck noise

In DDPG algorithms, exploration noise is often added to actions to encourage the agent to explore the environment more effectively. While Gaussian noise is commonly used, it might not be ideal in certain cases as it promotes trajectories in the neighborhood of the deterministic ones, potentially limiting the agent's ability to visit unexplored states. The Ornstein-Uhlenbeck (OU) process is a stochastic process that generates temporally correlated noise, which can be used as an alternative to Gaussian noise.

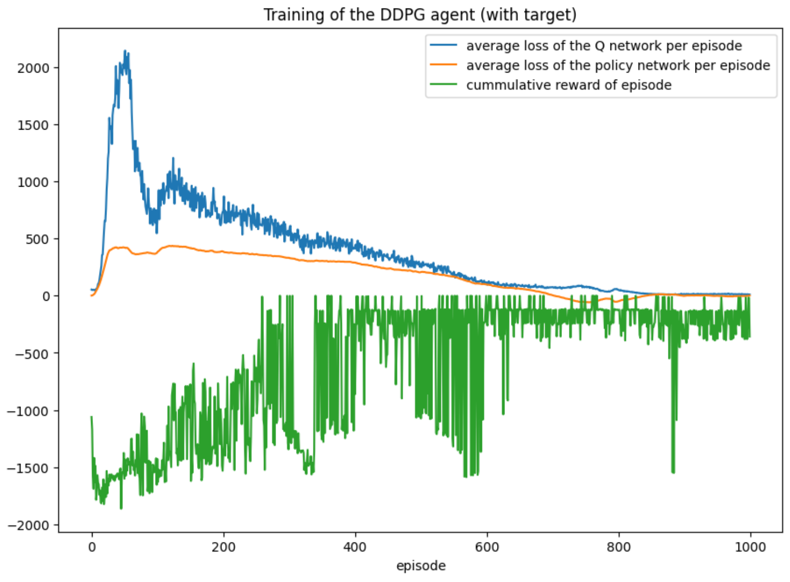

Figure 7: Training process of the DDPG agent with target network

In this section, we implement a simplified version of Ornstein-Uhlenbeck (OU) noise as an alternative for Gaussian noise, studying its impact on the training.

6.1 Implementation and Training of the OU Noise

We created the `OUActionNoise` class, which includes two methods: `get_noisy_action()` and `evolve_state()`. The OU noise stores the action noise selected in the previous step, and evolves the noise through a correlation mechanism, producing correlated action perturbations. This allows the noise to be more cohesive and better guides the agent's exploration.

For our experiment, we selected $\tau = 0.1$ as previously used with the DDPG agent, and ran the training for 5 different values of the θ hyper-parameter between 0 and 1 (using a $\sigma = 0.3$).

Based on network settings, we analyzed the impact of different values of the θ hyper-parameter in the Ornstein-Uhlenbeck noise on the training. We explored five different θ values: 0.01, 0.05, 0.1, 0.5, and 1. The following qualitative observations were made for each value:

- $\theta = 0.01$ and $\theta = 0.05$: These low values of θ led to noise that was barely correlated from one action step to another. As a result, the exploration resembled a weak version of Gaussian noise, limiting the agent's ability to explore unvisited states. The learning process appeared relatively stable; however, it required more episodes to achieve good performance due to less efficient exploration.
- $\theta = 0.1$: At this value, the agent managed to strike a balance between stability and exploration capability. This θ value can be considered a good candidate for further tuning and investigation.
- $\theta = 0.5$: With this value, the OU noise was substantially correlated, which allowed the agent to explore more distant regions of the state space. However, the increased exploration capability came at the cost of reduced stability in the learning process.
- $\theta = 1$: With $\theta = 1$, the OU noise was completely uncorrelated, which is equivalent to Gaussian noise.

To summarize, the choice of the θ parameter for the Ornstein-Uhlenbeck noise has a substantial impact on the balance between exploration efficiency and learning stability, in the network setting and problem space given above, a θ value at 0.1 appears to provide a good trade-off between these factors.

From the chosen hyper-parameters and the analysis of the learning curves and cumulative rewards as shown in Figure 8, we can draw the following conclusions:

The learning curve of the policy network (orange line) show that the DDPG agent with target networks and OU noise has the most stable and consistent decrease in the loss function compared to the minimal implementation and the DDPG agent with target networks and Gaussian noise. It also achieves the lowest final loss value. This indicates that the OU noise is more effective in guiding the agent's exploration, resulting in a more stable learning process and better overall control of the pendulum system.

The learning curve of the Q network in this setup has a stableness and consistency between the minimal implementation and the DDPG agent with target networks and Gaussian noise. In the ending episodes, the critic loss rises while the actor loss decreases, resulting in a reversed trend of the learning curves. It suggest that the Q network has learned a reasonably good approximation of Q-values, and the policy is improving its efforts towards maximizing those values.

The cumulative reward throughout the training presents (green line) also has a stableness and consistency between the minimal implementation and the DDPG agent with target networks and Gaussian noise.

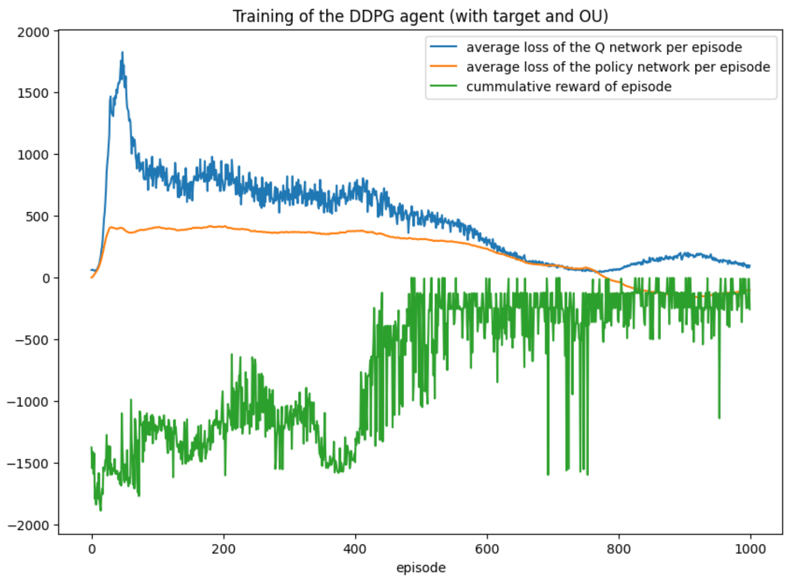

Figure 8: Training process of the DDPG agent with target network and OU noise

6.2 Testing Results with target networks and OU noise

In the testing phase, the tuned DDPG agent with target networks achieved an average cumulative reward of -154.03. It demonstrates a performance between the minimal implementation and the DDPG agent with target networks and Gaussian noise. This indicates that the OU noise is more effective in guiding the agent's exploration, but it might also requires more episodes to achieve good performance due to less efficient exploration.

7 Conclusion

In our project, we have demonstrated the applicability and effectiveness of the Deep Deterministic Policy Gradient (DDPG) algorithm in solving the classical control problem of stabilizing an inverted pendulum. Through an incremental building approach, we thoroughly examined each component of the DDPG algorithm, illuminating their individual contributions to successful learning. Our proposed modifications, which included the application of target networks and the incorporation of Ornstein-Uhlenbeck noise, have proven to be genuinely beneficial in practice.