University of Trento
Department of Industrial Engineering

UNIVERSITY OF TRENTO

Optimization Based Robot Control

Assignment 03

Deep Q-Network

Professor:
Prof. Andrea Del Prete

Students:

Lorenzo Colturato	lorenzo.colturato@studenti.unitn.it	233301	from DII
Luca Zardini	luca.zardini@studenti.unitn.it	229366	from DISI

The code is available at:
`https://github.com/lucaZardini/optimization-based-robot-control/tree/main/03_assignment`

Academic Year 2022-2023

Introduction & Task Formalization

The goal of the assignment is to implement the Deep Q-Learning (or Deep Q-Network) algorithm to find the optimal control policy to obtain the swing up with subsequent upward balancing of a single pendulum and a double pendulum. The latter has only one actuated joint, i.e. the first one. For this task, the two pendulums have continuous states and discrete actions. Since the number of states is continuous, Q-Learning cannot be used because the associated Q-table needs discrete states and actions. As the name suggests, the Deep Q-Learning algorithm is able to approximate the Q-table (or Q-function) and, in particular, it returns the best action (based on what has been learned) given an input state. The Q-function (also called action-value function) denotes the value (Q-value) of taking an action in a state following a policy π. It can be seen as the cost we get starting from a state x, applying an action u and following a policy π, i.e.:

$$\begin{aligned} Q^{\pi}(x,u) &= J_t(x_t = x, u_t = u, u_{k>t} \sim \pi) \\ &= l(x,u) + \gamma \cdot Q^{\pi}(f(x,u), \pi(f(x,u))) \\ &= l(x,u) + \gamma \cdot V^{\pi}(f(x,u)) \end{aligned}$$

where $V^{\pi}(x)$ is the value-function over policy π and γ is the discount factor used to scale down the rewards more and more after each step so that the total sum remains bounded.

The basic operation of Deep Q-Learning is shown in Figure 1.

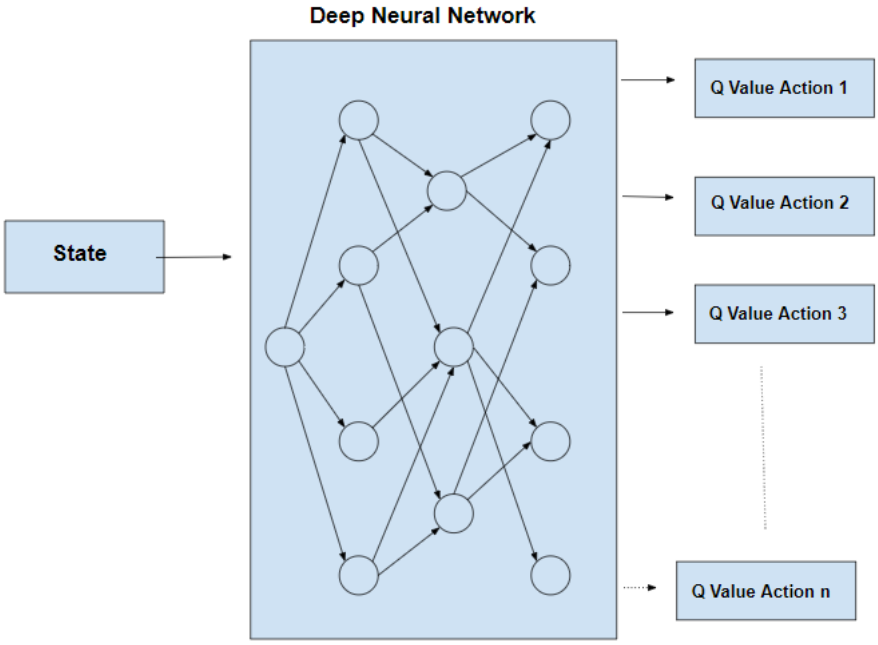


Figure 1: Basic operation of Deep Q-Learning

As can be seen from Figure 1 there is a deep neural network that is used to approximate the Q-function. In particular, the initial state is fed as an input to the neural network and returns the Q-value of all possible

actions as an output. This allows the algorithm to handle environments with a large number of states and actions, as well as learning from high-dimensional inputs, as opposed to Q-learning that is only practical for very small environments.

This algorithm belongs to the category of reinforcement learning. More in detail, it is necessary to define an agent and an environment: in our case the agent is the network, while the environment are the two pendulums, which live in the *ROS (Robot Operating System)* workspace. The two environments have been provided during the laboratory sessions. One of the main characteristics of Deep Q-Learning is that, unlike other typical Deep Learning processes, the target is not stationary. For this reason there are two neural networks: one is used to learn the parameters of the network (called critic), while the other, which has the same architecture as the first but frozen parameters, is used to calculate the target (called target). After a fixed number of iterations in the critic network, the parameters are copied to the target network.

Optimal Control Problem Formulation

Deep Q-Network algorithms use neural networks to represent the Q-function (i.e. $Q(x, u, w)$ with w being the vector of weights) and they rely on ε-greedy policies with ε decreasing over time. They also use experience replay, a technique where the agent stores a subset of its experiences (state, action, reward, next state) in a memory buffer and samples (every few steps) a mini-batch of transitions from this buffer to update the Q-function. This helps to decorrelate the data and make the learning process more stable, allowing the agent to learn from past experiences.

The implemented DQN algorithm is shown below with the name of the parameters we used in our code:

Algorithm 1 Deep Q-Learning for single/double pendulum environment

Initialize replay memory *experience_replay* to capacity *buffer_size*
Initialize action-value function Q with random weights
for episode = 1, *episodes* **do**
  Initialize state x_t
  for t = 1, *max_iterations* **do**
    With probability ε select a random action u_t
    otherwise select $u_t = \min_u Q^*(x_t, u, w)$
    Execute action u_t and observe cost l_t and state x_{t+1}
    Store transition (x_t, u_t, l_t, x_{t+1}) in *experience_replay*
    Set $x_{t+1} = x_t$
    Sample random minibatch of transitions (x_t, u_t, l_t, x_{t+1}) from *experience_replay*
    Set $y_j = l_j + \gamma \cdot \min_{u_{t+1}} Q(x_{t+1}, u_{t+1}, w^-)$
    Perform a gradient descent step on $(y_j - Q(x_t, u_j, w))^2$
  end for
end for

Generally speaking Q aims to maximize the expected future reward for taking an action at a state. In our case, since we have an optimal control problem, we aim to minimize the cost that allows to hit the target/goal. To do that we apply the *Bellman optimality equation* to Q and we minimize for w:

$$\min_w \mathbb{E}\left[l + \gamma \cdot \min_{u_{t+1}} \mathrm{Q}(x_{t+1},\, u_{t+1},\, w^-) - Q(x, u, w)\right]$$

where l is the cost, $Q|_{w^-}$ is the Q-function relative to the target network and $Q|_w$ is the Q-function relative to the critic model. This formulation is the core of the Deep Q-Learning weights update. In addition, after a specified number of steps, the critic weights are assigned to the target ones.

Formulation of the cost function

In general an optimal control problem is formulated as follows:

$$\begin{aligned} \min_{\bar{x}} \quad & \text{Objective/Cost function} \\ \text{s.t.} \quad & \text{Constraint equations/inequalities} \end{aligned}$$

where $\bar{x}$ represents the minimization variables of the specific problem.

The cost function that we have decided to use for the single pendulum is the following:

$$\begin{aligned} \min_{q,v,u} \quad & q^2 + 0.1 \cdot v^2 + 0.001 \cdot u^2 \\ \text{s.t.} \quad & -u_{max} \leq u \leq u_{max} \\ & -v_{max} \leq v \leq v_{max} \\ & -\frac{\pi}{2} \leq q \leq \frac{\pi}{2} \end{aligned}$$

where:

- u is the control/action
- q is the joint position in *rad*
- v is the joint velocity in *rad/s*

As can be noted the term with the highest weight within the cost is on the position of the joint: the farther the pendulum is from the goal (i.e. the higher the joint angle), the higher the cost. Since the cost must be minimized, the pendulum must learn to swing upwards. There are also some weights on the velocity of the joint and the control action that allows the positioning of the pendulum, but their value is lower w.r.t. the weight on the joint angle.

The cost function chosen for the double pendulum is defined as:

$$\begin{aligned} \min_{q,v,u} \quad & 10 \cdot (q_1 + q_2)^2 + q_1^2 + 5 \cdot q_2^2 \\ \text{s.t.} \quad & -u_{max} \leq u \leq u_{max} \\ & -v_{max} \leq \vec{v} \leq v_{max} \\ & -\frac{\pi}{2} \leq \vec{q} \leq \frac{\pi}{2} \end{aligned}$$

where:

- u is the control/action at the actuated joint
- $\vec{q}$ is the joints position vector in *rad*
- $\vec{v}$ is the joints velocity vector in *rad/s*

For the double pendulum we decided not to introduce terms relating to the joints velocity and the action applied to the actuated joint in the cost function. The only terms present concern the joint angles. In particular, we decided to penalize more the relative position of the non-actuated joint w.r.t. the position of the actuated joint (term $(q_1 + q_2)^2$). This translates into saying that the second link must always point upwards as intended by the end goal. Then, the other two terms concern the position of the two joints independently of each other. In the desired final position the two links should both point upwards, a configuration which corresponds to having joint angles equal to zero ($\{q_1 = 0, q_2 = 0\}$). From this point of view we have given more weight to the position of the second joint (term q_2^2). In fact, we have seen through a *trial&error* process that, by giving greater priority (and therefore greater weight) to the achievement of the objective by the first joint, the second link was unable to move up as wanted.

Table 1 shows the chosen values for the velocity and control limits for both single pendulum and double pendulum.

	Single pendulum	**Double pendulum**
$\mathbf{u_{max}}$ **(Nm)**	2	3
$\mathbf{v_{max}}$ **(rad/s)**	8	12

Table 1: Control/velocity limit values

For the double pendulum, being under-actuated, we decided to increase the limits on the velocity of the joints and the control/torque provided at the actuated joint to further help the pendulum achieve its goal.

Discretization of the actions

Single pendulum

The discretization of the actions is done within a range that goes from -2 Nm to +2 Nm (limit values shown in Table 1). The discretization depends on the chosen number of Q-Value actions, i.e. on the number of neurons in the last layer of the neural network. In particular, since we have decided to build a deep neural network that returns 11 output Q-Value actions, the control is discretized as follows:

$$\begin{aligned}\text{Discretized control} &= \left[(0,1,2,...,11-1) - \frac{(11-1)}{2}\right] \cdot \frac{2 \cdot \overbrace{u_{max}}^{=2}}{11} \\ &= \left(-5 \cdot \frac{4}{11}, -4 \cdot \frac{4}{11}, -3 \cdot \frac{4}{11}, ..., 5 \cdot \frac{4}{11}\right) \\ &= (-1.82, ..., 1.82)\end{aligned}$$

As can be seen the actions are limited between the range [-2, 2] and, in particular, the maximum (in module) value assumed is about 1.82 Nm.

Double pendulum

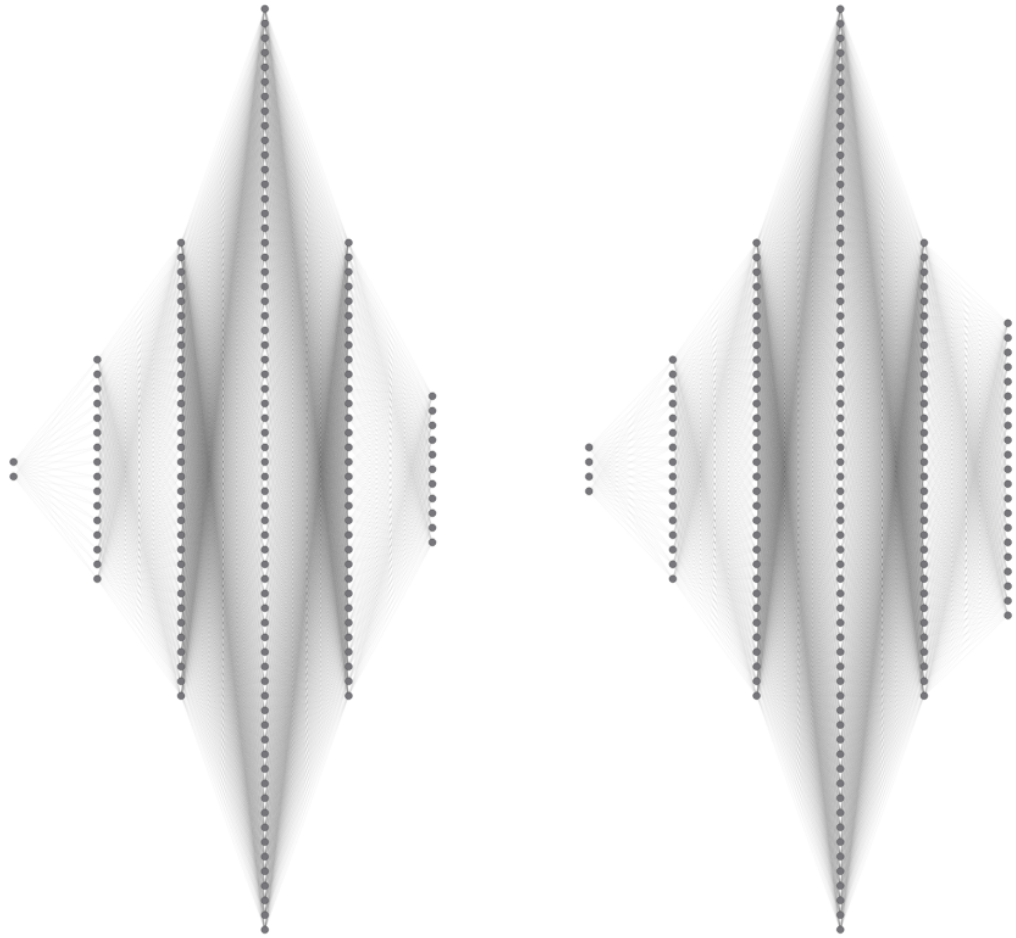
The discretization of the controls for the double pendulum was done similarly to what has already been explained for the single pendulum. The only differences are in the chosen maximum torque limit/value (u_{max}) and the number of Q-Value actions chosen for the utilized deep neural network. Since we use different networks for the two environments and since the network used to train the double pendulum has 21 nodes in the last layer, the discretization of the actions is done as follows:

$$\begin{aligned}\text{Discretized control} &= \left[(0,1,2,...,21-1) - \frac{(21-1)}{2}\right] \cdot \frac{2 \cdot \overbrace{u_{max}}^{=3}}{21} \\ &= \left(-10 \cdot \frac{6}{21}, -9 \cdot \frac{6}{21}, -8 \cdot \frac{6}{21}, ..., 10 \cdot \frac{6}{21}\right) \\ &= (-2.86, ..., 2.86)\end{aligned}$$

The discretization allows to choose between 21 actions that goes from -2.86 Nm to 2.86 Nm. This allows to have better control than what was done for the single pendulum, which represented a simpler system than the double pendulum.

Structure of the Neural Network

The two implemented neural networks consist both of six layers. The input dimension depends on the environment the network is dealing with. In particular, if the environment is the single pendulum, the state consists of two values, i.e. the angle and the velocity of the joint. On the other hand, if the environment is the double pendulum, the state consists of four values, i.e. the angles and the velocities of the two joints. Figures 2a and 2b show, respectively, the network for the single and double pendulum.



(a) Architecture used for the single pendulum **(b)** Architecture used for the double pendulum

Figure 2: Deep neural networks used

From Figures 2a and 2b is possible to notice that each layer is composed of a specific number of neurons, which are stated in Table 2.

		Number of neurons
Layer	1	2 (for single pendulum) 4 (for double pendulum)
	2	16
	3	32
	4	64
	5	32
	6	11 (for single pendulum) 21 (for double pendulum)

Table 2: Specification of the number of neurons per layer

As already explained earlier, for the single pendulum we have decided to set the number of possible actions (i.e., the number of neurons on the last layer) to 11. For the double pendulum case we decided instead to provide the final layer of 21 neurons, so as to increase the number of actions the network can choose from to satisfy the goal.

As for the activation function we decided to use for all the layers the *rectified linear activation function (ReLU)*, which is one of the the most famous and commonly implemented for feed-forward neural network applications. ReLU is a piecewise linear function that outputs the input (of a neuron) directly if it is positive, otherwise, it will output zero. This behaviour is showed in Figure 3, where the values assumed by the input are shown on the abscissa axis while the output values are on the ordinate axis.

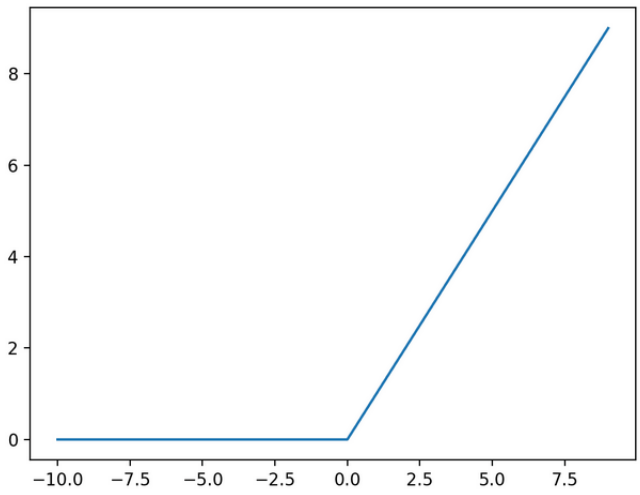


Figure 3: ReLU activation function

We also decided to use the *Adaptive Momentum (Adam) optimizer*, the most common optimizer used for all neural network tasks, which is a stochastic gradient descent method that, unlike the classical stochastic gradient descent procedure, is based on adaptive estimation of first-order and second-order moments in order to adapt the learning rate for each weight of the neural network.

Results for the Single Pendulum

Table 3 collects the hyper-parameters used to find the optimal control policy to achieve the goal of swinging the single pendulum up.

Hyper-parameter | ***name given in the code***	**Value**
Learning rate | *learning_rate*	0.001
Discount factor γ | *discount*	0.99
Experience replay | *experience_replay*	10000
Batch size | *batch_size*	32
Initial ε | *epsilon_start*	1
ε decay | *epsilon_decay*	0.9995
ε min | *epsilon_min*	0.002
Iterations per episode | *max_iterations*	500
Number of episodes | *episodes*	300
Update critic parameters | *update_critic_params*	10
Update target parameters | *update_target_params*	1000

Table 3: Hyper-parameters of the algorithm for the single pendulum

Some of these values are standard, such as the learning rate, the initial ε and the ε decay, which is necessary to progressively reduce the possibility to choose random actions. In particular, after two episodes ε reaches values around 0.6 and after 24 episodes it reaches the value of ε min, i.e. 0.002. Indeed, in order not to completely cancel the randomness of the choices, it is important that ε does not go to zero and therefore we have provided a minimum limit value to ε which, once reached, means that there is a 99.8% probability that an action from the model will be chosen and a 0.2% probability that the chosen action is random. We have added a relatively high discount factor since we thought that the next actions should not be penalized too much. As for the other parameters, the size of the experience replay buffer has been set to 10000 iterations, i.e. 20 complete episodes, whereas the target parameters are updated every 1000 steps/iterations, that are two complete episodes. Moreover, instead of updating the critic weights every step, we decided to do it after each 10 steps. The reason behind this choice relies of the fact that we can collect more and various new data before updating the weights. Finally, we have chosen to give 300 iterations per episode because we have seen that above this value there are no significant improvements and this number is sufficient for the pendulum to reach its goal.

Considering the number of episodes chosen, the training phase took approximately 19.38 minutes. Another 41.05 minutes were needed to evaluate the models (i.e. to select the model with the best weights), for a total time of 60.43 minutes, just over an hour.

Selection of the best model

In the training phase, each episode provides some weights. These weights are used to simulate the behavior of the pendulum starting from known initial configurations (position and velocity) that we consider challenging, which are expressed in Table 4.

Joint angle (rad)	**Joint velocity (rad/s)**
π	0
$\pi/2$	0
$-\pi/2$	0
$-\pi/4$	0.5
$\pi/4$	-0.5
$3\pi/4$	1
$-3\pi/4$	-1

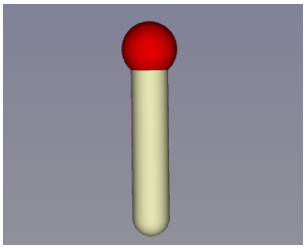
Table 4: Initial simulation configurations for the selection of the best model

The weights provided by each episode are used to simulate the pendulum starting from these initial values of position and velocity of the joint. Then the costs deriving from the seven simulations carried out are added together to obtain a single cost and subsequently, episode after episode, this cost is compared with the cost relating to the episode which provides the best model up to that moment. If the cost of the new episode is lower than the cost of the best episode up to that point, the new episode becomes the best (that is, it becomes the benchmark for subsequent episodes), otherwise the old best episode remains that way. Also, the best episode weights are automatically saved in a file which is overwritten each time a new episode becomes the best by the weights that it provides. The 41.05 minutes of time necessary for the evaluation of the models is due precisely to the simulations that are carried out episode by episode with the new weights of the network. Since there are 300 episodes in total, the seven simulations run take about 8 seconds per episode, about double what is required of the network to being trained and calculate the new weights to be used in the simulations. Doing this, with the hyper-parameters of Table 3, we have found that the best model (that provides the overall minimal cost) was found in training at episode 256.

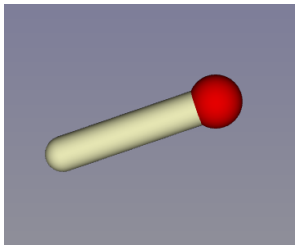
Plots related to the optimal model

Once we obtained the weights of the best model from the training phase, we used these weights to simulate the pendulum starting from three different initial configurations in order to verify that, regardless of the starting configuration (position and velocity), the weights found were such to allow the pendulum to always reach the goal.

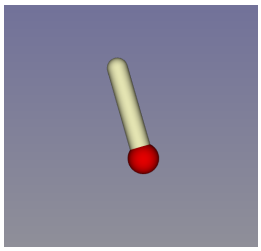
Figure 4 shows all the plots for the three configurations. In particular, Figure 4a shows the robot positioned on the negative vertical. We wanted to see if, starting from rest with an angle equal to π (therefore pendulum pointing downwards), the pendulum was able to go to the positive vertical thus reaching the set objective. The related position and velocity plots are Figures 4d and 4g. Instead, Figures 4e and 4h are related to the configuration of Figure 4b, which is $\{q = -1.89rad, v = -3.15rad/s\}$. Finally, the plots associated to the configuration of Figure 4c, which is $\{q = -0.27rad, v = 3.60rad/s\}$, are 4f and 4i, respectively position and velocity. The plots about the control actions to swing the pendulum up are also reported, respectively 4j, 4k and 4l for the three initial configurations.



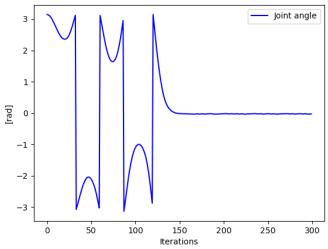
(a) Initial configuration



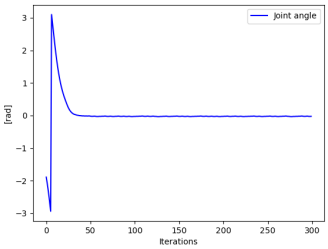
(b) Initial configuration



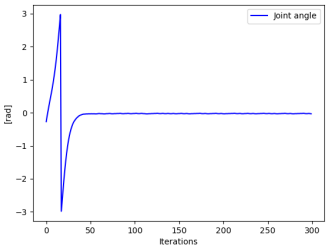
(c) Initial configuration



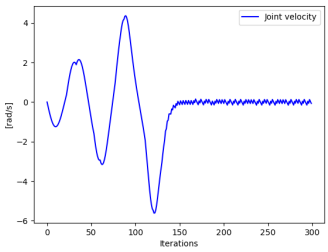
(d) Joint position



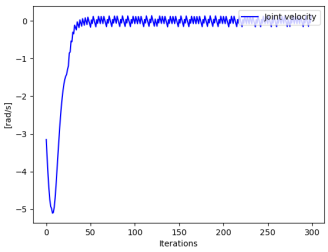
(e) Joint position



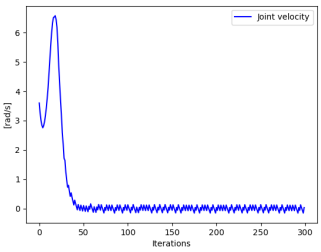
(f) Joint position



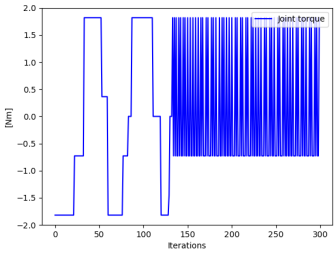
(g) Joint velocity



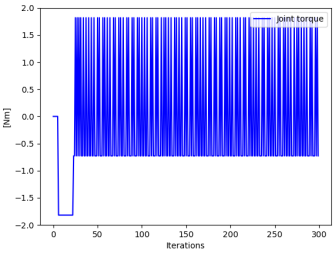
(h) Joint velocity



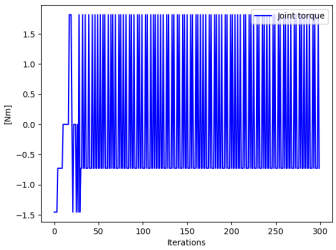
(i) Joint velocity



(j) Joint torque



(k) Joint torque



(l) Joint torque

Figure 4: Simulation results considering three different starting episode: (a) $\{q = \pi, v = 0\}$, (b) $\{q = -1.89rad, v = -3.15rad/s\}$, (c) $\{q = -0.27rad, v = 3.60rad/s\}$

The most challenging initial configuration is perhaps the one where the pendulum starts from the bottom with zero initial velocity. From the plots we can see that, limiting the torque to a maximum absolute value of 2 Nm, it takes approximately 150 iterations to stabilize the pendulum in the upper vertical position. After the pendulum is up, in order to keep it in the neighborhood of the wanted position. After the 150 iterations of upward stabilization, to keep it in the neighborhood of the desired position, impulsive actions are applied with torque ranging from maximum positive values (about 1.82 Nm) to smaller negative values in modulus, about -0.7 Nm. This allows to keep the pendulum in position by swinging it left and right very quickly so as not to unbalance it and not make it rotate downwards. In the second configuration the pendulum has a starting velocity that is pushing it down. The network waits until the pendulum reaches joint angle equal to π (the torque applied is equal to zero), then it pushes with the maximum allowed torque to reach the balance. When the pendulum reaches the equilibrium, it remains up as explained above. The last configuration starts with the pendulum near to the desired position and with a velocity that pushes it to the wanted direction with very high intensity. The control first tries to reduce the intensity of the starting velocity by applying a

high torque in the opposite direction but, after some iterations, the pendulum goes downward. Then, taking advantage of the high velocity, actions are applied to rotate the pendulum clockwise to reach the goal. Also here, when the pendulum reaches the desired configuration it remains up as already said.

Figure 5 shows the average cost-to-go, which is the total discounted cost obtained by the algorithm during the entire training phase. It is possible to notice that its trend is not predictable and there are lots of fluctuations. This is mainly due to the fact that each episode has a random starting configuration: in some cases the pendulum is already close to the desired configuration, other times it takes a lot of time to reach the goal, and other times it is not able to reach it, achieving so a very high cost-to-go.

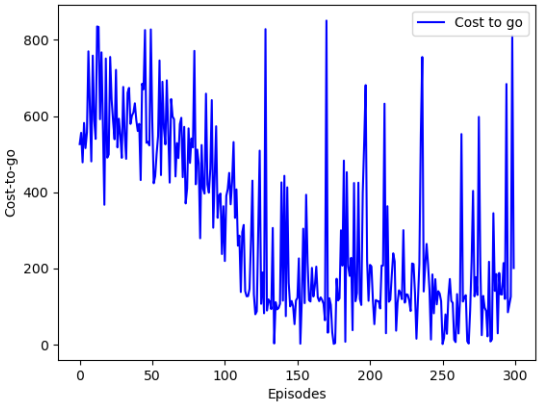


Figure 5: Discounted average cost-to-go

Finally, the training loss is shown in Figure 6. In the first iterations, the loss is very high, since the two networks start from random weights and perform two different tasks. After some iterations this loss decreases, meaning that the critical network has learned about the weights, which are then copied to the target one. Due to the instability of reinforcement learning, the loss decrease is not linear, and in some upgrades, the loss is still quite high.

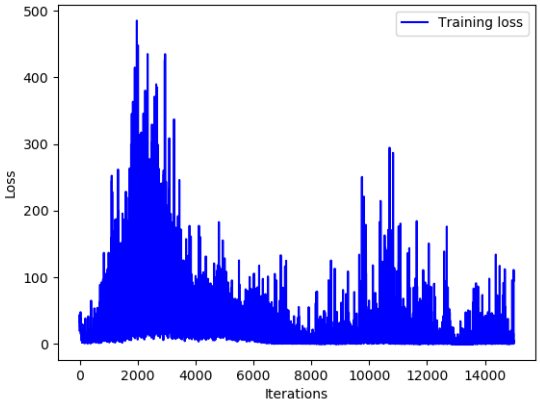
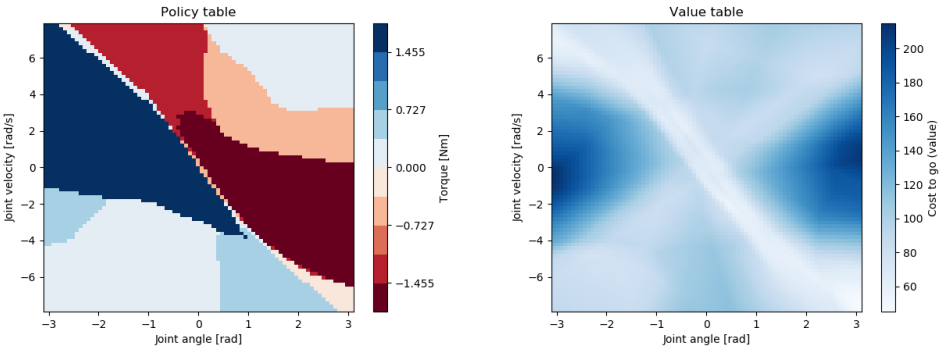


Figure 6: Training loss

Figure 7 shows the policy and value tables for the single pendulum swing up problem. To generate the tables, the continuous states have been discretized.



(a) Policy table for Deep Q learning

(b) Value table for Deep Q-Learning

Figure 7: The policy and value functions obtained solving the swing up single pendulum problem with Deep Q learning

The value table (Figure 7b) shows the minimum costs associated with a specific configuration. In particular, it is possible to see that the middle of the value table has an overall low cost, whereas the higher values are when the joint angle is far from the goal. The policy table (Figure 7a) shows instead the best action to perform in a specific configuration. For example, in the wanted configuration ($\{q = 0, v = 0\}$), the best action consists of standing in that position. In this figure it is possible to notice that, when the pendulum is near to the goal for both velocity and angle, the model use a very high torque to reach the goal. This behaviour is confirmed also in Figure 4j, 4k and 4l. Here, the pendulum alternates a relatively high positive torque and then applies a small negative torque to counteract the effect.

Results for the Double Pendulum

Table 5 collects the hyper-parameters used to train the model of the double pendulum.

Hyper-parameter | *name given in the code*	**Value**
Learning rate | *learning_rate*	0.001
Discount factor γ | *discount*	0.99
Experience replay | *experience_replay*	10000
Batch size | *batch_size*	128
Initial ε | *epsilon_start*	1
ε decay | *epsilon_decay*	0.9995
ε min | *epsilon_min*	0.002
Iterations per episode | *max_iterations*	500
Number of episodes | *episodes*	2000
Update critic parameters | *update_critic_params*	10
Update target parameters | *update_target_params*	1000

Table 5: Hyper-parameters of the algorithm for the double pendulum

A lot of hyper-parameters are the same as those of the single pendulum. The main difference is that we have increased the batch size to 128 to better support training as the target to be achieved is more complex than that of single pendulum. In addition, we have also increased the number of episodes to 2000 to allow the model to train better to return good weights to perform the required task.

Selection of the best model

The selection of the best model was conducted exactly as for the single pendulum, i.e. by considering specific initial configurations of the joints' position and velocity; the only difference is that now there are two joints. In this case, for reasons related to the higher simulation time of the models compared to the single pendulum case, and the fact that the double pendulum requires more episodes to be trained, we have decided to take five different initial configurations which are reported in Table 6.

First joint angle	**Second joint angle**	**First joint velocity**	**Second joint velocity**
π	0	0	0
$\pi/2$	$\pi/2$	0	0
$-\pi/2$	$-\pi/2$	0	0
$\pi/2$	$-\pi/2$	0.5	0.5
$\pi/2$	$-\pi/2$	-0.5	-0.5

Table 6: Initial simulation configurations for the selection of the best model

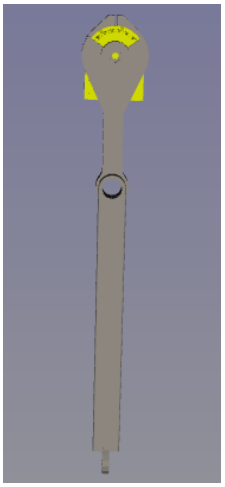
The same operations of models evaluation already explained for the single pendulum are performed here

to find which model among the various trained ones has the best weights.

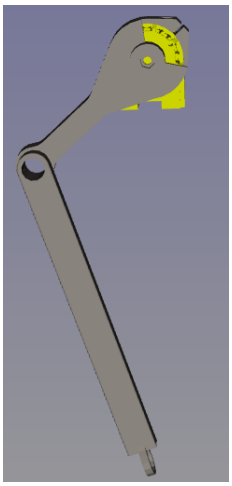
The training required approximately 266.7 minutes (~ 4.4 hours) of time, or 8 seconds per episode, about double that was required by the single pendulum, while the evaluation of the models to find each episode the new best one has required approximately 8.33 hours (15 s per episode). The total time amounts more or less to 12.73 hours for 2000 episodes, much more than required by the single pendulum which, proportionally, took about 6.71 hours, which is almost the half.

Plots related to the optimal model

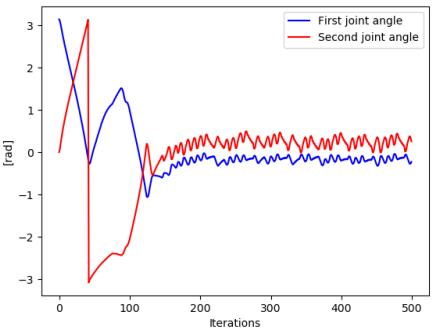
Once the weights of the best model were obtained, we used them to simulate the behaviour of the pendulum starting from two different initial configurations: one where the system at rest starts by pointing downwards (configuration of Figure 8a - $\{q_1 = \pi, q_2 = 0, v_1 = 0, v_2 = 0\}$), the other (randomly chosen) where the system starts with some initial velocity on the joints and with the two links not aligned (configuration of Figure 8b - $\{q_1 = -2.34rad, q_2 = 1.12rad, v_1 = -3.2rad/s, v_2 = 2.63rad/s\}$). This is done to verify that, regardless of the starting configuration, the weights found were such to allow the pendulum to always reach the goal. The position and velocity plots related to the first configuration (pendulum on the negative vertical) are Figures 8c and 8e. Instead, Figures 8d and 8f are related to the second configuration (the random one). The plots about the control actions to swing the pendulum up are also reported, respectively 8g, 8h for the two configurations.



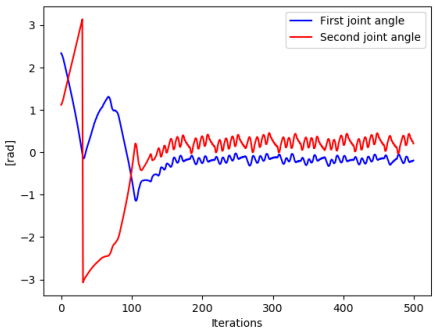
(a) Initial configuration



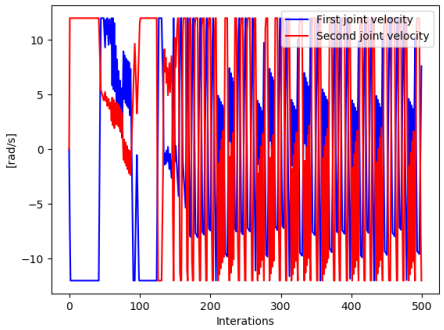
(b) Initial configuration



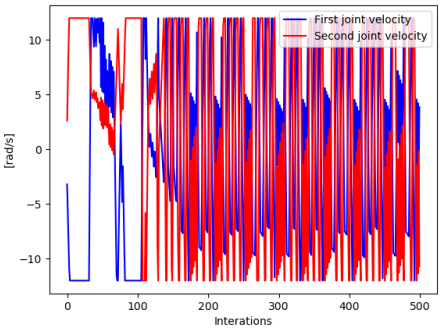
(c) Joints position



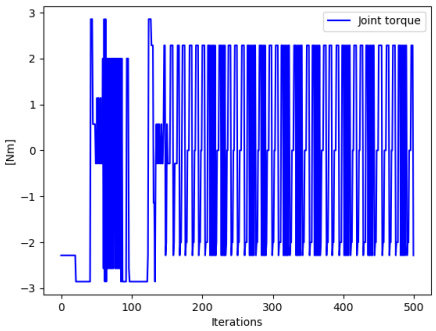
(d) Joints position



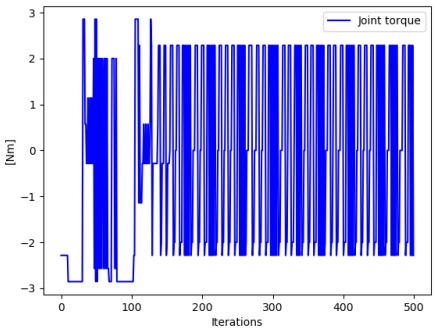
(e) Joints velocity



(f) Joints velocity



(g) Actuated joint torque



(h) Actuated joint torque

Figure 8: Simulation results considering two different starting episode: (a) $\{q_1 = \pi, q_2 = 0, v_1 = 0, v_2 = 0\}$, (b) $\{q_1 = -2.34rad, q_2 = 1.12rad, v_1 = -3.2rad/s, v_2 = 2.63rad/s\}$ The second joint angle is relative to the first one and not absolute.

In the first initial configuration the pendulum starts at the bottom with zero velocity on the two joints. To try to achieve the goal, it begins to apply a negative torque on the actuated joint, thus causing it to rotate clockwise. Once the first joint reaches the top position it receives an instantaneous push on the other direction causing a decrease in its angle, while the second joint starts rotating counter-clockwise. Then, the first joint receives a push to start rotating clockwise. This push allows the first joint to align with the second one, whose previous rotation is stopped. At this point, after about 100 iterations, the first joint is able to support the second and begins to receive pushes that balance its movement. So, the first joint continues to rotate clockwise and counter-clockwise very quickly to maintain balance and support the second joint at the top. To do so, the actuated joint continues to alter relatively high positive and negative impulses. In the second configuration the pendulum starts pointing downwards with the second joint not aligned with the first. The first joint has a negative initial velocity which pushes it upwards, while the second joint, subject to a positive velocity, is also pushed upwards. The first joint therefore receives a negative thrust which increases its speed (in modulus). When the pendulum reaches the top, its behavior is very similar to what was explained for the first initial configuration.

Figure 9 shows the cost-to-go, i.e. the total discounted cost achieved by the algorithm during the training phase. As for the single pendulum, the trend is not predictable, and this is mainly due to the random beginning of each episode. However, it is possible to notice a gradually decreasing cost. This means that, regardless of starting position, the trained model, as episodes increase, is able to achieve a lower cost-to-go.

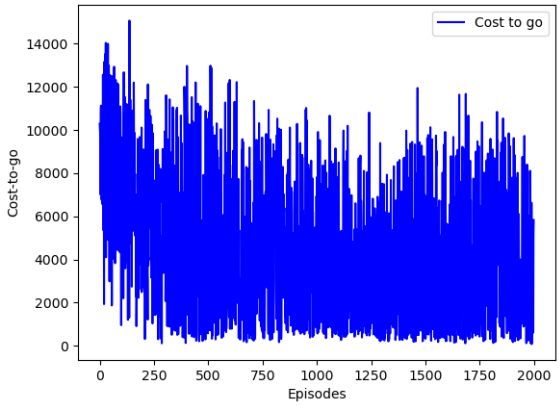


Figure 9: Discounted cost-to-go

Finally, Figure 10 shows the achieved training loss. Here, the loss is quite low in the first iterations, but it starts increasing gradually with high peaks in correspondence of the target weights update. Since the two networks make different computations, the loss then starts again to decrease. Like the single pendulum case, due to the fact that the target is not stationary and due to the instability of reinforcement learning, the loss does not decrease linearly.

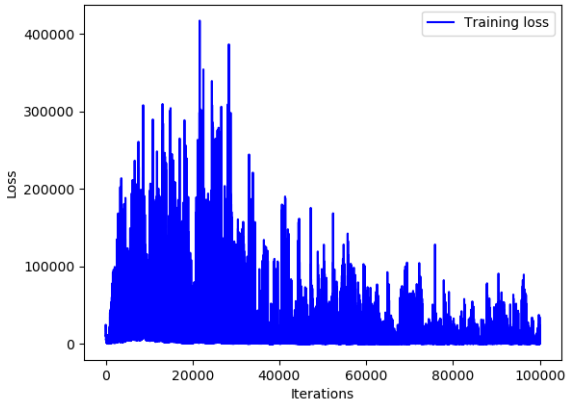


Figure 10: Training loss

Conclusions

Final considerations

To find the best models for the single pendulum and for the double pendulum, we based ourselves on a *trial&error* approach, gradually modifying the values of the hyper-parameters (mainly the number of episodes and the number of iterations as well as the batch size and the experience replay), but above all the cost and associated weights. In doing so we have conducted some experiments to choose the best models for each environment. Regarding the single pendulum environment, we started with a higher torque limit equal to 5 Nm. We then tried to alter the torque limits, reducing them, to see if the pendulum was still able to reach the desired configuration with less torque available. We have seen that, with a lower maximum torque, the pendulum still reaches the goal but needs more time since, having less actuation power, it must learn to make greater use of the effect of gravity to create the momentum necessary to go upwards. We believe that this model is more interesting since it is able to interact with external forces and to learn how to deal with them. Regarding the double pendulum environment, we have set the maximum torque to 3 Nm. In fact, we didn't want to stray too far from the torque values made available to the single pendulum. However, with this limited value of torque available and maintaining the speed of 8 rad/s as in the single pendulum, we realized that the double pendulum was not able to go up. We have therefore increased the maximum speed at the joints to 12 rad/s. Moreover, to have more sensitivity, we have increased the number of actions. Indeed, since the task is more complex, more actions are needed to balance better the double pendulum; we have seen that lower number of actions brings to worse control and performance (there is low sensitivity), but at the same time a higher number of actions increases the complexity of the network and so the task (more neurons mean more weights to train).

Potential improvements

A possible problem is related to the choice of the *update_ critic_ params* and the *update_ target_ params* hyper-parameters. In particular, the critic weights are updated every 10 iterations and they are copied into the target network every 1000 iterations. Therefore, the target network is updated every 100 updated of the critic one. By increasing this update ratio, the loss might decrease thus improving the performance. However, due to the low computational resources available, higher ratio values could not be tested. A possible improvement related to the single pendulum implementation is related to the implemented neural network. The architecture used for the single and the double pendulum is the same, except for the input and output layers. Since the single pendulum task is relatively simple, a smaller neural network should be enough to find an optimal control that allows the pendulum to reach the desired goal; in this way, it would be possible to reduce the number of parameters to be learnt and the training time. Regarding the double pendulum, the implemented cost can be for sure improved. In particular, this cost does not take into account the joints velocity. We have tried to add a velocity cost but the performance were not so good and satisfactory and the double pendulum was not able to reach the goal. Instead, the configuration we have found is able to reach an upstanding position in most cases, so we have decided to keep the model trained with the described cost. We also believe that with an adequate cost it is possible to train the system to achieve the goal with

a lower maximum speed on the joints, which is now fixed at 12 rad/s. In fact, we have noticed that with such a maximum velocity value the pendulum does not learn adequately and its movements do not seem very natural and practical for a real pendulum. Another aspect that we wanted to consider, but which we were unable to do due to time constraints, concerns the actuation of the joints. In particular, we wanted to see if implementing the second joint (and no longer the first) the pendulum still managed, with a suitably chosen cost, to reach the final goal of positioning upwards with both links aligned.