Third Assignment ORC
DQN Network

Bertelli Davide
mat. 223718
davide.bertelli-1@studenti.unitn.it

Hueller Jhonny
mat. 221254
jhonny.hueller@studenti.unitn.it

1 Abstract

The third assignment revolved around the task to implement a deep Q-Learning network able to proficiently tackle the task to balance a pendulum. The environment in which the network works is a continuous one and it had at its disposal only the discrete set of controls available for the pendulum and the cost returned by the environment when an action is performed. The solution proposed shows good results both in terms of cost and final position.

2 Introduction

Working with automated agents in a continuous environment is a particulary difficult task due to the amount of events that may happen. Specifically when an agent is exploited in an environment in which it is requested to make any action that changes it the complexity rises. To address this increasig complexity it has been made necessary to employ systems that were able to adapt to the environment independently by how many stimula they may receive. For this reason the deep neural networks have been introduced being highly non-linear and able to "understand" a continuous environment performing actions based on few inputs. In our case the environment is represented by a pendulum which, starting from any position, has to reach the uppermost position and keeps it resulting in a balanced system. Even though the environment is continuous the controls available for the pendulum are made discrete through a discretization step and so they vary in resulution.

3 Network

The network in use exploits a deep fully connected architecture in which the input is the state of the pendulum, its position in radiants and its angular velocity, and the output is the predicted Q value for each possible control of the joints. The architecture has been inspired by the one proposed in [1]. Specifically the network is composed of 4 dense layers having respectively 16, 32, 64 and 64 activation cells. The output of the network has dimension equal to the number of discrete controls at the power the number of joints. So, if we are working with a double pendulum with $n\_joints = 2$ and a control resolution equal to $n\_controls = 10$ the resulting output of the network will be $n\_controls^{n\_joints} = 10^2 = 100$. In Figure 3 is shown a graphical representation of the network in use.

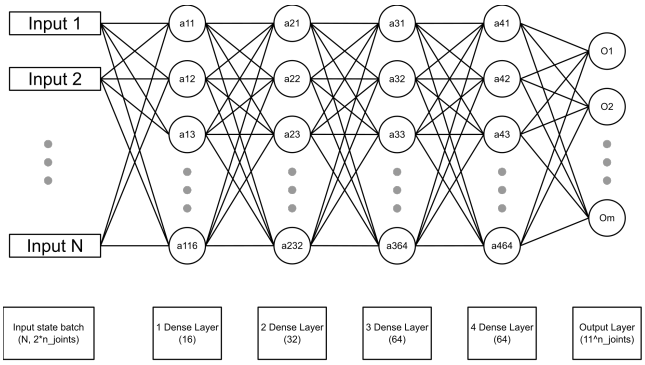

Figure 1: Network architecture in case of n_joints composing the pendulum and a control discretization over 11 steps.

4 Policy

The policy used to chose the action to undertake is a greedy one. Specifically, the Q Network output is made up of all the possible combinations of the controls and it will be passed to the policy in order to chose the next action. The policy is allowed to explore depending by a set probability, otherwise it will act greedy by selecting the lowest Q Value among the one given by the deep neural network.

5 Experience Replay

In order to counteract the shift of the target to which the network have to converge, a buffer has been introduced allowing the training to run smoothly over already known solutions without having the network diverge. The buffer has been implemented as a class having a fixed size and allowing for random sampling. When empty, the buffer is filled by querying a network acting as the critique of the system and the policy which will by used by it. Having those elements the input and outputs of the critique are saved inside the buffer until the desired size is reached.

6 System setup

The system provided as solution is composed by several objects:

- **Buffer**: It is the object responsible to store the examples used by the actor network to train. Only the most recent examples are retained over the training

to avoid that the actor network will learn a too old configuration of the system.

- **Policy**: It is the object responsible to chose the best control with which to feed the robot and allowing the movement among states.
- **Actor Network**: It is the network which has to be trained ov the environment and which output represents the Q Value obtained through the experience replay and exploration tasks.
- **Critique Network**: It is the target network which knows the environment and which is used to judge how well the actor network behaves. Its weights are fixed and it is not subjected to training.

6.1 Actor Network training

The training of the network is carried on using the Mean Squared Error of the Q Values provided by the network as the loss function which has to be minimized and employing Adam as the optimizer. To run the training the user is required to provide two mandatory values: the number of episodes and the length of the episodes. These two parameters define, respectively, the number of the random starting positions from which the robot will start the training and the length of the search executed for each episode. At each episode, and for all the length of the episodes, the buffer is randomly sampled obtaining batches of tuples containing states, controls, next states and costs. Then the provided batch is feeded to the Q network working as actor and its output is feeded to the policy which will return the optimal control for the next state, relying its judgment only on the Q Values provided, or it will try to explore the environment selecting a random control depending by the probability set. Once the policy had given its output the new control is applied to the robot and the next state and cost are retrieved from the environment. Having those informations, the states and controls, the next states and Q Values, the actor network is feeded using the states and controls providing in output its Q values. At this point the Q Values of the batch and those of the actor network are used to compute the MSE representing our loss, lastly a gradient update is performed. Every 15 episodes the best network is saved by checking the average cost among them.

7 Actor Network testing

The testing of the actor network was carried out first, varying the starting position of the robot randomly and secondly making it start from the lowest position to ensure the goodness of the results. For each configuration three episodes were run and the sum of costs gathered during their total length is used as the evaluation measure. The testing phase works by gathering the robot state, querying the policy for the next action, apply it and take the cost from the environment for all the length of the episode.

8 Experiments

The system has been tested over different configurations of the robot, specifically with 1, 2 and more joints, different control resolution and different number and length of episodes. The evaluation metrics used are the loss of the training, the training time and the total cost in the testing phase.

9 Results

9.1 1 joint

Following are shown the results in case the robot has 1 joint.

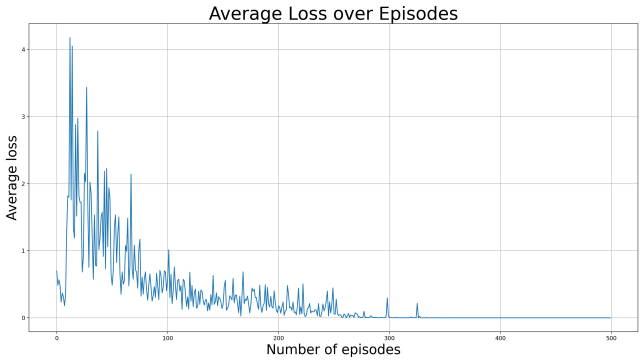

Figure 2: Training loss having 1 joint.

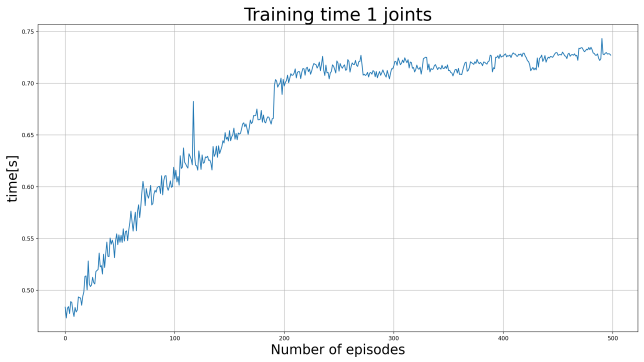

Figure 3: Training time having 1 joints.

Following are the results given by the testing phase.

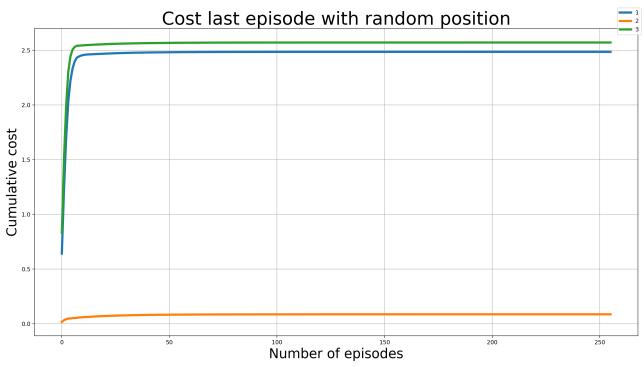

Figure 4: Total cost over the test episodes for the robot starting from random positions.

Figure 5: Total cost over the test episodes for the robot starting from the down position.

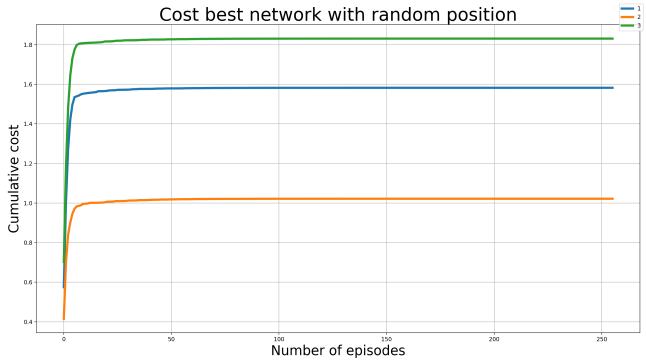

Figure 6: Total cost over the test episodes for the robot starting from random positions using the best performing network.

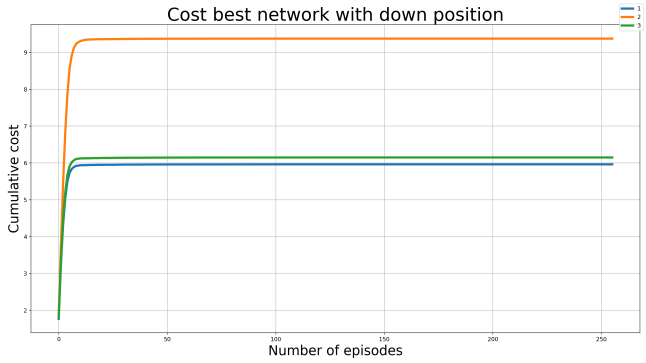

Figure 7: Total cost over the test episodes for the robot starting from down positions using the best performing network.

9.2 2 joints

Following are shown the results in case the robot has 2 joints.

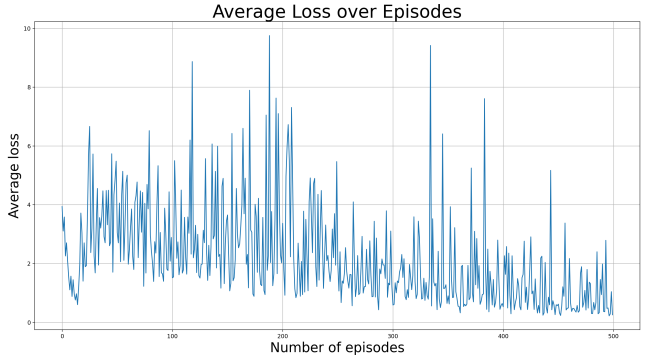

Figure 8: Training loss having 2 joints.

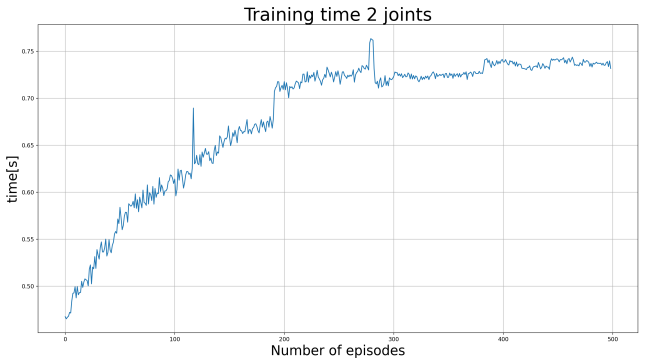

Figure 9: Training time having 2 joints.

Following are the results given by the testing phase.

Figure 10: Total cost over the test episodes for the robot starting from random positions.

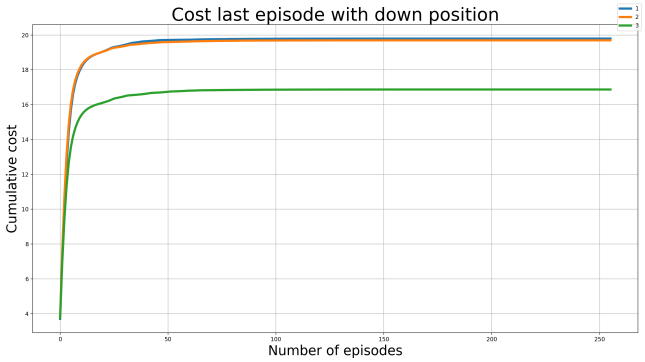

Figure 11: Total cost over the test episodes for the robot starting from the down position.

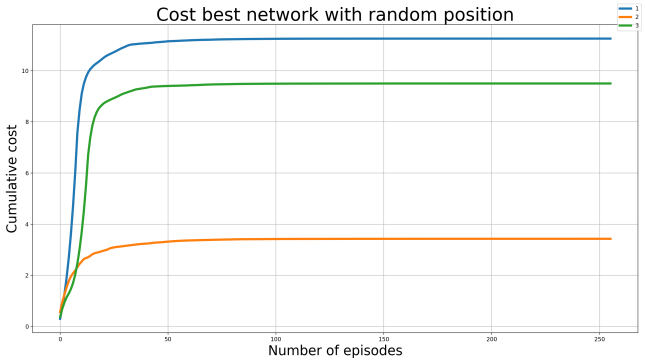

Figure 12: Total cost over the test episodes for the robot starting from random positions using the best performing network.

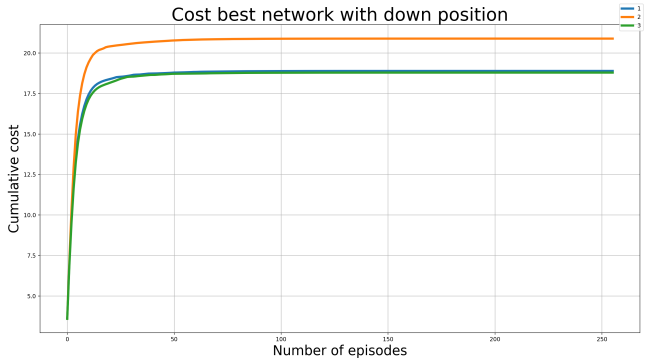

Figure 13: Total cost over the test episodes for the robot starting from down positions using the best performing network.

9.3 N joints

9.4 General remarks

As shown in figures 9.1, 9.2 we can observe how the loss during training shows a progressive reduction with the increasing number of episodes. The spikes in the graph has to be attributed to the exploration task that is conducted by the policy.

In the testing phase it is possible to observe that the cumulative cost saturates meaning that the robot is performing actions leading to the best solution possible which is returning a cost almost 0.

Regarding the training time plots, it is possible to observe how independently by the number of joints the time required for the computations saturates without exceeding the 0.8 seconds per episode.

10 Conclusions

References

[1] Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015). https://doi.org/10.1038/nature14236