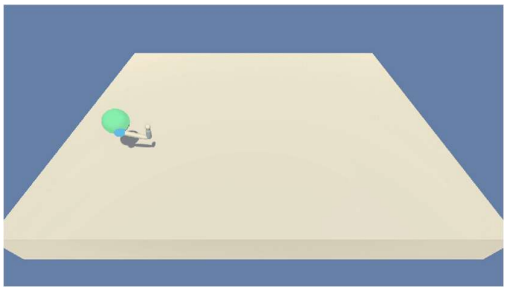
Project 2 – Continuous Control

Introduction:

The purpose of this project is to train a double-jointed robotic arm to move to a target location (Marked with a green ball). A reward of +0.1 is provided for each step that the robotic arm hand ("The agent" onwards) is in the goal location. Thus, the goal of the agent is to maintain its position at the target location for as many time steps as possible.



For this project, we will work with the Unity Reacher environment.

The observation space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the robotic arm. Each action is a vector with four numbers, corresponding to torque applicable to two joints. Every entry in the action vector should be a number between -1 and 1.

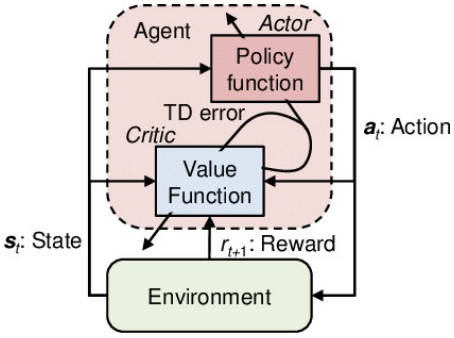
Usually to calculate a robotic arm to move to a place there are a lot of calculations to be done to, translate the desired position from the end effector to the base of the arm (Forward kinematics) translate this position to each joint (Inverse kinematics) and calculate the necessary movement of each torque to get to the desired positions (Differential Kinematics) which takes a lot of calculations (Calculate translation matrixes, calculate matrix rotations and translations, find Jacobian of matrixes, calculate the determinant of these Jacobians, …), thanks to Reinforcement Learning we avoid all these calculation, which is implicitly learnt by the RL agent without the requirement of the simulated dynamics.

Learning Algorithm:

The algorithm is an implementation of the DDPG (Deep deterministic policy gradient) algorithm with soft updates. This algorithm is an Actor-critic method, these methods main characteristics are:

- The reduction the high-variance commonly associated to policy-based agent,
- It learns faster than policy-based methods and
- have more consistent convergence than value-based agents

For this agent to be an Actor-critic, we have two learning neural networks:



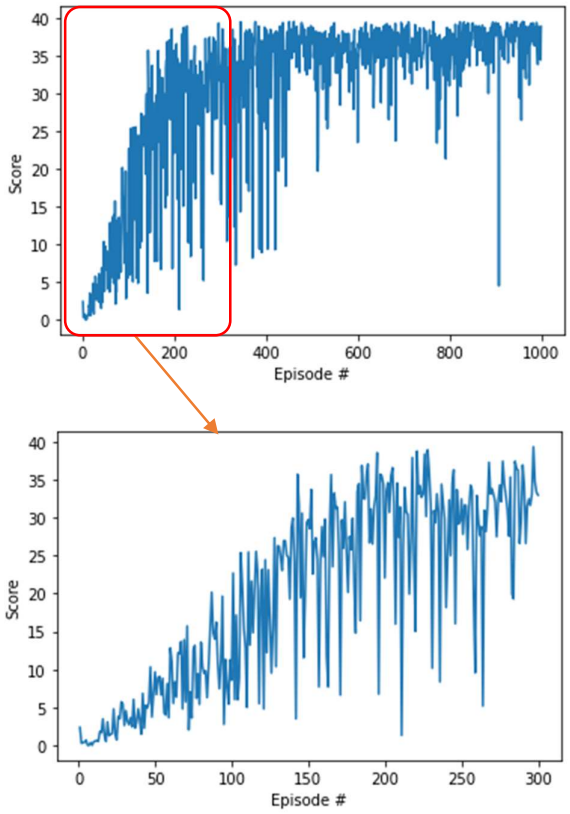
- In a standard Actor-critic method, the **Actor network** will learn the probabilities of good and bad actions. Will take in states and outputs a distribution over the possible actions. But **in the case of DDPG**, the algorithm used in this project, **the Actor will give us the best possible action based on the policy learnt, not a probability distribution.**
- The **Critic network** will learn how to estimate good actions and bad actions to help on the actor decisions. Technically it will learn to evaluate the state value function ($V\pi$), will take in states and outputs an expectation of the state value function of the policy, **this state value function will update the actor network.**

When training is complete, the trained optimal policy will be stored in the actor. The training process for this method are complicated, so a lot of improvements have been implemented to increase it learning efficiency.

On the first iteration we tried with an Actor and Critic neural networks with the following architecture:

- Actor with two HL: HL1: 512 units and HL2 of 256 units.
- Critic with two Hidden Layers: HL1 of 512 units and HL2 of 256 units too.

```
Episode 50      Average Score: 2.87     MinScore: 0.00  MaxScore: 10.32         Hour: 11:07:44
Episode 100     Average Score: 6.21     MinScore: 0.00  MaxScore: 20.16         Hour: 11:35:02
Reset values avgscore, max and minscore
Episode 150     Average Score: 14.73    MinScore: 1.00  MaxScore: 35.73         Hour: 12:09:26
Episode 200     Average Score: 23.42    MinScore: 1.00  MaxScore: 38.57         Hour: 12:38:50
Reset values avgscore, max and minscore
Episode 250     Average Score: 28.04    MinScore: 1.00  MaxScore: 38.95         Hour: 13:09:57
Episode 300     Average Score: 29.84    MinScore: 1.00  MaxScore: 39.35         Hour: 13:48:05
Reset values avgscore, max and minscore
Episode 350     Average Score: 30.15    MinScore: 1.00  MaxScore: 39.49         Hour: 14:28:06
Episode 400     Average Score: 31.27    MinScore: 1.00  MaxScore: 39.54         Hour: 14:56:34
Reset values avgscore, max and minscore
Episode 450     Average Score: 33.03    MinScore: 1.00  MaxScore: 39.42         Hour: 15:26:33
Episode 500     Average Score: 35.13    MinScore: 1.00  MaxScore: 39.57         Hour: 15:57:02
Reset values avgscore, max and minscore
Episode 550     Average Score: 35.82    MinScore: 1.00  MaxScore: 39.39         Hour: 16:28:25
Episode 600     Average Score: 34.88    MinScore: 1.00  MaxScore: 39.39         Hour: 16:58:54
Reset values avgscore, max and minscore
Episode 650     Average Score: 35.73    MinScore: 1.00  MaxScore: 39.44         Hour: 17:30:33
Episode 700     Average Score: 36.25    MinScore: 1.00  MaxScore: 39.44         Hour: 18:00:57
Reset values avgscore, max and minscore
Episode 750     Average Score: 36.23    MinScore: 1.00  MaxScore: 39.50         Hour: 18:31:52
Episode 800     Average Score: 35.51    MinScore: 1.00  MaxScore: 39.55         Hour: 19:05:10
Reset values avgscore, max and minscore
Episode 850     Average Score: 35.50    MinScore: 1.00  MaxScore: 39.51         Hour: 19:41:51
Episode 900     Average Score: 36.80    MinScore: 1.00  MaxScore: 39.51         Hour: 20:14:52
Reset values avgscore, max and minscore
Episode 950     Average Score: 37.23    MinScore: 1.00  MaxScore: 39.57         Hour: 20:48:19
Episode 1000    Average Score: 36.99    MinScore: 1.00  MaxScore: 39.57         Hour: 21:27:21
Reset values avgscore, max and minscore
```



For the second iteration we included Leaky ReLUs activation functions on the critic neural network keeping the same neural networks architecture:

- Actor with two HL: HL1: 512 units and HL2 of 256 units.
- Critic with two Hidden Layers: HL1 of 512 units and HL2 of 256 units too.

```
Episode 50      Average Score: 3.46     MinScore: 0.00  MaxScore: 16.10         Hour: 17:00:20
Episode 100     Average Score: 11.57    MinScore: 0.00  MaxScore: 39.46         Hour: 17:23:41
Reset values avgscore, max and minscore
Episode 150     Average Score: 27.08    MinScore: 1.00  MaxScore: 39.59         Hour: 17:47:41
Episode 200     Average Score: 34.24    MinScore: 1.00  MaxScore: 39.59         Hour: 18:10:54
Reset values avgscore, max and minscore
Episode 250     Average Score: 32.67    MinScore: 1.00  MaxScore: 38.78         Hour: 18:33:46
Episode 300     Average Score: 32.21    MinScore: 1.00  MaxScore: 39.10         Hour: 18:57:33
Reset values avgscore, max and minscore
Episode 350     Average Score: 33.22    MinScore: 1.00  MaxScore: 39.55         Hour: 19:21:33
Episode 400     Average Score: 34.31    MinScore: 1.00  MaxScore: 39.60         Hour: 19:45:44
Reset values avgscore, max and minscore
Episode 450     Average Score: 36.01    MinScore: 1.00  MaxScore: 39.61         Hour: 20:10:27
Episode 500     Average Score: 36.76    MinScore: 1.00  MaxScore: 39.61         Hour: 20:35:26
Reset values avgscore, max and minscore
Episode 550     Average Score: 36.92    MinScore: 1.00  MaxScore: 39.58         Hour: 21:00:52
Episode 600     Average Score: 36.96    MinScore: 1.00  MaxScore: 39.61         Hour: 21:26:32
Reset values avgscore, max and minscore
Episode 650     Average Score: 36.87    MinScore: 1.00  MaxScore: 39.61         Hour: 21:52:39
Episode 700     Average Score: 37.54    MinScore: 1.00  MaxScore: 39.65         Hour: 22:19:05
Reset values avgscore, max and minscore
Episode 750     Average Score: 38.17    MinScore: 1.00  MaxScore: 39.65         Hour: 22:45:50
Episode 800     Average Score: 37.83    MinScore: 1.00  MaxScore: 39.65         Hour: 23:12:55
Reset values avgscore, max and minscore
Episode 850     Average Score: 37.70    MinScore: 1.00  MaxScore: 39.62         Hour: 23:40:33
Episode 873     Average Score: 37.63    MinScore: 1.00  MaxScore: 39.62         Hour: 23:53:17
```

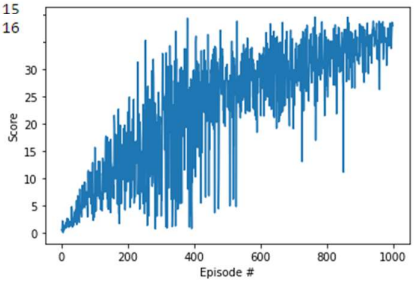
We can see that the learning speeds up, getting to an average of 30 points already in the 300 hundred iterations, while we had to wait for the 400 hundred iterations without the ReLUs functions.

On the third iteration, we created a larger network for the Critic NN, including a third Hidden Layer of 128 units.

- Actor with two HL: HL1: 512 units and HL2 of 256 units.
- Critic with three Hidden Layers: HL1 of 512 units and HL2 of 256 units and a HL3 of 128 units.

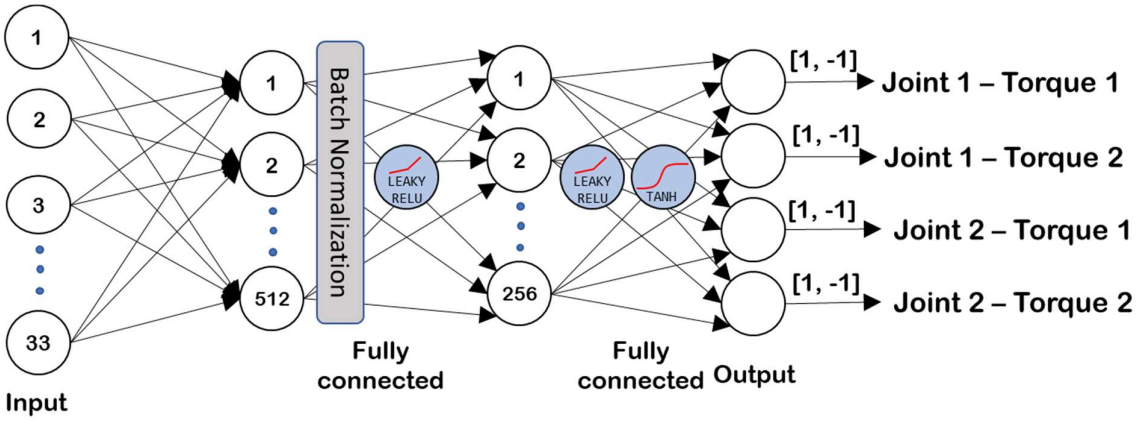
```
Episode 50      Average Score: 2.24     MinScore: 0.04  MaxScore: 6.83  Hour: 12:29:47
Episode 100     Average Score: 4.28     MinScore: 0.04  MaxScore: 12.87         Hour: 12:54:20
Reset values avgscore, max and minscore
Episode 150     Average Score: 8.13     MinScore: 1.00  MaxScore: 17.76         Hour: 13:19:16
Episode 200     Average Score: 11.04    MinScore: 1.00  MaxScore: 22.70         Hour: 13:45:29
Reset values avgscore, max and minscore
Episode 250     Average Score: 12.94    MinScore: 1.00  MaxScore: 31.33         Hour: 14:12:24
Episode 300     Average Score: 15.17    MinScore: 0.76  MaxScore: 35.34         Hour: 14:38:10
Reset values avgscore, max and minscore
Episode 350     Average Score: 17.58    MinScore: 0.88  MaxScore: 37.01         Hour: 15:05:12
Episode 400     Average Score: 19.66    MinScore: 0.80  MaxScore: 39.39         Hour: 15:32:36
Reset values avgscore, max and minscore
Episode 450     Average Score: 22.66    MinScore: 1.00  MaxScore: 36.39         Hour: 16:00:23
Episode 500     Average Score: 25.55    MinScore: 1.00  MaxScore: 37.18         Hour: 16:28:31
Reset values avgscore, max and minscore
Episode 550     Average Score: 26.16    MinScore: 1.00  MaxScore: 38.84         Hour: 16:56:58
Episode 600     Average Score: 26.84    MinScore: 1.00  MaxScore: 38.84         Hour: 17:25:58
Reset values avgscore, max and minscore
Episode 650     Average Score: 28.70    MinScore: 1.00  MaxScore: 38.37         Hour: 17:55:19
Episode 700     Average Score: 29.14    MinScore: 1.00  MaxScore: 38.62         Hour: 18:25:00
Reset values avgscore, max and minscore
Episode 750     Average Score: 29.38    MinScore: 1.00  MaxScore: 38.09         Hour: 18:57:15
Episode 800     Average Score: 30.78    MinScore: 1.00  MaxScore: 39.60         Hour: 19:27:50
Reset values avgscore, max and minscore
Episode 850     Average Score: 32.10    MinScore: 1.00  MaxScore: 38.17         Hour: 19:59:07
Episode 900     Average Score: 33.08    MinScore: 1.00  MaxScore: 39.56         Hour: 20:30:48
Reset values avgscore, max and minscore
Episode 950     Average Score: 34.28    MinScore: 1.00  MaxScore: 38.85         Hour: 21:03:15
Episode 1000    Average Score: 35.28    MinScore: 1.00  MaxScore: 38.85         Hour: 21:36:16
Reset values avgscore, max and minscore
```

We can see that the growing of our Critic neural network hurt the performance of our algorithm, so we get back to the previous Neural Network, which showed quite an acceptable performance.

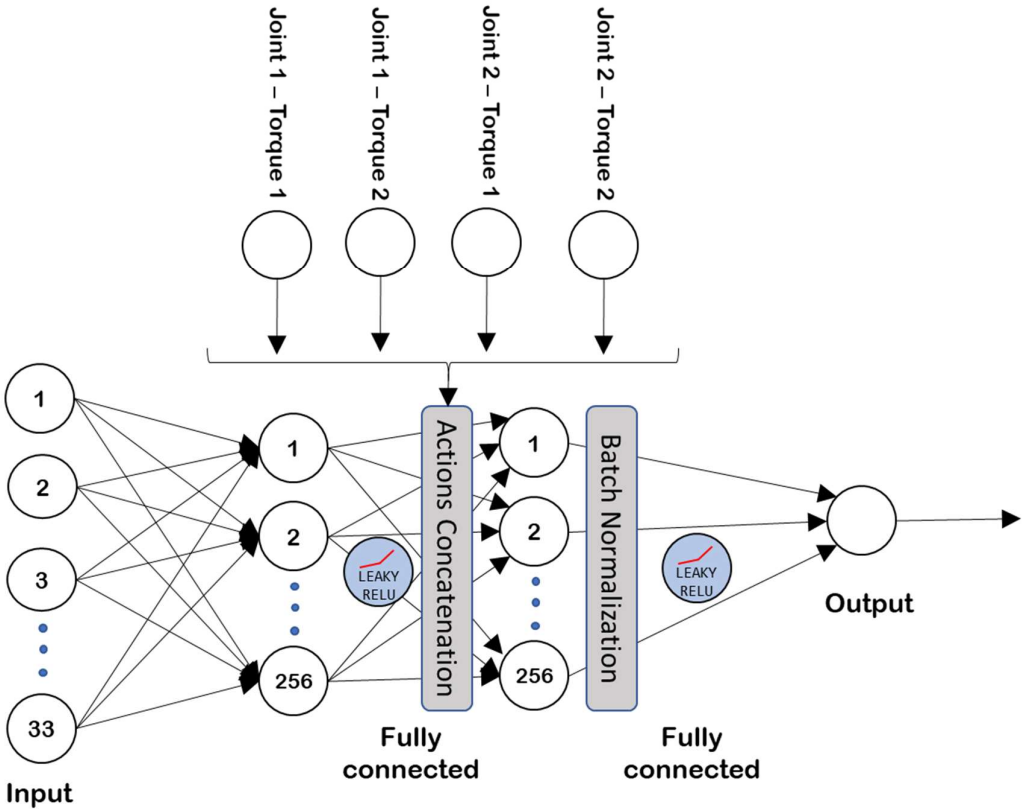


Actor Neural Network:

The actor NN used is the following one, the ending function used is a Tanh to be able to output continuous variables between 1 and -1:



Critic Neural Network:

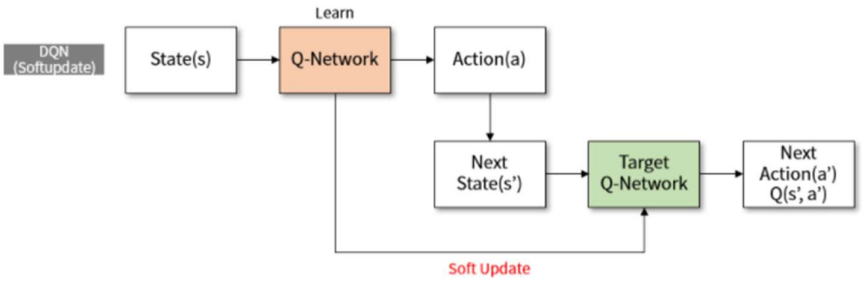


Thanks to the implementation of **experience replay**, the algorithm stores states which are rare and actions that are costly to be able to recall them, each experience (Which is formed by State, Action, Reward and Next State) is stored in a buffer as the agent is interacting with the environment, afterwards the agent randomly samples a small batch of these experiences in order to learn from them. Thanks to this:

- it learns from individual state-actions multiple times, so it recalls rare occurrences and make a better use of the experience obtained.
- Because of the random sampling, it helps breaking the correlation and prevent actions from oscillating or diverging in wrong ways.

Another critic functionality of the DDPG algorithm is a **soft update** to the target network.

The implementation of this soft update means that the algorithm has two neural networks. The called "Regular" neural network and the "Target" neural network.



In our algorithm, this soft update is done every step, so the networks are blended with a percentage of merging considering the TAU variable, so considering the TAU variable is set to 0,05%, every step a 0,05% of the regular network will be merged with the target network.
Thanks to soft update, the DDPG algorithm gets faster convergence. Soft update logic could be implemented in other off-policy algorithms that use target networks, such as DQN.

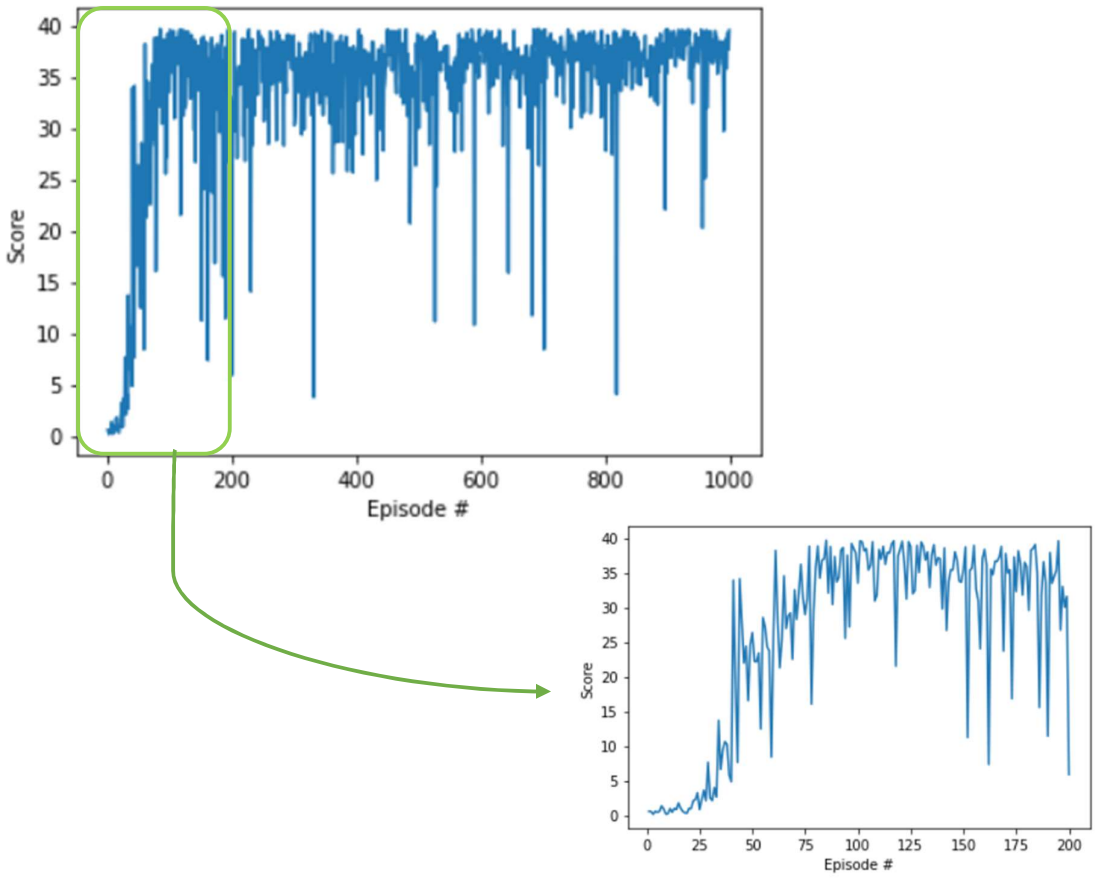
To encourage exploration in the training process, we added **noise** using the Ornstein-Uhlenbeck noise process, the noise is reduced through time, because when the agent is more experienced, the need for exploration decreases.

We used **batch normalization** in both networks for stabilizing the learning process and reduce the training time for deep neural networks, batch normalization is especially useful to limit covariate shift in environments where the input distribution changes with each step, like this.

On the critic side, we have **used leaky relu** activation functions for dealing better with the possible negative values coming from the learning (*Dying ReLU problem*) and improve training performance.

Algorithm performance:

```
Episode 50      Average Score: 6.94     MinScore: 0.13  MaxScore: 34.12         Hour: 10:57:31
Episode 100     Average Score: 18.70    MinScore: 0.13  MaxScore: 39.67         Hour: 11:21:21
Reset values avgscore, max and minscore
Episode 150     Average Score: 33.28    MinScore: 1.00  MaxScore: 39.59         Hour: 11:47:25
Episode 200     Average Score: 34.05    MinScore: 1.00  MaxScore: 39.59         Hour: 12:13:22
Reset values avgscore, max and minscore
Episode 250     Average Score: 33.37    MinScore: 1.00  MaxScore: 39.61         Hour: 12:41:49
Episode 300     Average Score: 35.48    MinScore: 1.00  MaxScore: 39.61         Hour: 13:10:26
Reset values avgscore, max and minscore
Episode 350     Average Score: 35.47    MinScore: 1.00  MaxScore: 39.53         Hour: 13:40:31
Episode 400     Average Score: 34.69    MinScore: 1.00  MaxScore: 39.53         Hour: 14:08:19
Reset values avgscore, max and minscore
Episode 450     Average Score: 35.00    MinScore: 1.00  MaxScore: 39.57         Hour: 14:33:21
Episode 500     Average Score: 35.69    MinScore: 1.00  MaxScore: 39.62         Hour: 14:59:32
Reset values avgscore, max and minscore
Episode 550     Average Score: 35.88    MinScore: 1.00  MaxScore: 39.37         Hour: 15:25:58
Episode 600     Average Score: 35.68    MinScore: 1.00  MaxScore: 39.61         Hour: 15:54:37
Reset values avgscore, max and minscore
Episode 650     Average Score: 35.89    MinScore: 1.00  MaxScore: 39.54         Hour: 16:25:17
Episode 700     Average Score: 35.70    MinScore: 1.00  MaxScore: 39.68         Hour: 16:57:06
Reset values avgscore, max and minscore
Episode 750     Average Score: 35.85    MinScore: 1.00  MaxScore: 39.59         Hour: 17:27:56
Episode 800     Average Score: 36.73    MinScore: 1.00  MaxScore: 39.59         Hour: 17:58:36
Reset values avgscore, max and minscore
Episode 850     Average Score: 36.44    MinScore: 1.00  MaxScore: 39.55         Hour: 18:29:54
Episode 900     Average Score: 36.13    MinScore: 1.00  MaxScore: 39.65         Hour: 19:02:15
Reset values avgscore, max and minscore
Episode 950     Average Score: 37.22    MinScore: 1.00  MaxScore: 39.64         Hour: 19:34:55
Episode 1000    Average Score: 37.44    MinScore: 1.00  MaxScore: 39.64         Hour: 20:08:27
```



Future improvements:

Some future improvement that could be done in this algorithm are:

1. Try making a hard-update on the target network **periodically** to make the learning process faster without losing the soft updates improvements to the learning process.
2. Decrease exploration through the learning process
3. Implementing some features described in the benchmarking DeepRL for Continuous control paper:
    - Rescale the reward by a factor of 0.1 through the learning process, this should improve stability in the learning process.
    - Change learning rates to $5x10^{-3}$, usually this value improves performance.
4. Analyze if it's possible to improve the neural network architecture by using Archai:

    https://www.microsoft.com/en-us/research/blog/archai-can-design-your-neural-network-with-state-of-the-art-neural-architecture-search-nas/