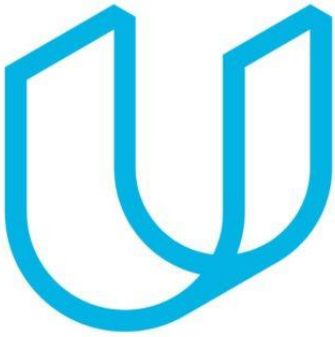

UDACITY

DEEP REINFORCEMENT LEARNING

MULTI-AGENT REINFORCEMENT LEARNING

PROJECT: *COLLABORATION AND COMPETITION*

AUTHOR: ALBERTO GARCÍA GARCÍA

Introduction

The problem to be solved in this project consists in training two agents bouncing a ball over a net using rackets. The agents need to keep the ball in play, obtaining a positive reward of +0.1 for every time they pass the ball over the net and a negative reward of -0.01 when the ball hits the ground or goes out of bounds. The state space consists of 8 variables including information about the velocity and the position of both, the ball and the racket. Notice that each agent receives its own local observation. Furthermore, there are 2 continuous actions, corresponding to the horizontal movement of the racket (toward or away from the net) and jumping. An image of such environment is presented in Fig. 1. The problem is considered to be solved when the agent's average score of the last 100 episodes is equal or greater than **0.5 points**.

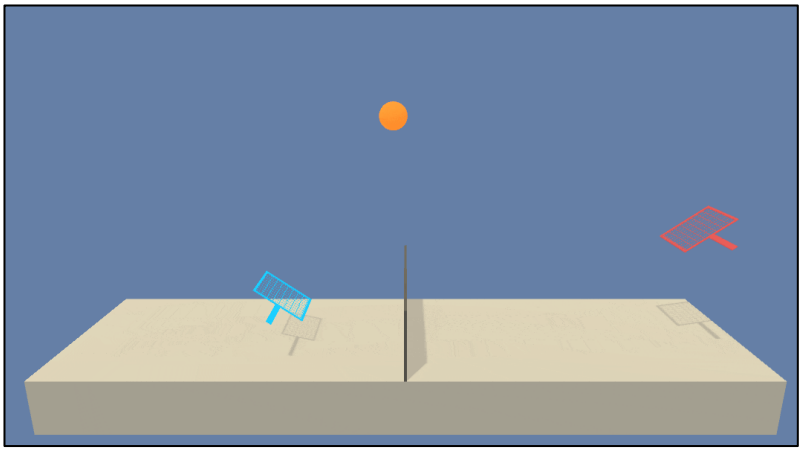

Fig. 1. Environment

This document includes the details of the algorithm developed to train the agents, the conclusions obtained after the completion of the project and some future ideas to improve the current implementation.

Implementation

Since the problem environment presents a continuous action space with more than one agent, the Multi-Agent Deep Deterministic Policy Gradient (**MADDPG**) algorithm has been chosen to train the agents. MADDPG is a variation of the DDPG algorithm, that allows training several agents simultaneously. DDPG is an off-policy reinforcement algorithm that combines elements of both policy-based and value-based methods, using an actor neural network to learn a deterministic policy and a critic neural network to approximate the value of the action (similar to the deep Q-network from Deep Q-Learning). In MADDPG, each agent has its own actor and critic networks, but share the experiences retrieved in a common replay memory. This allows a centralized training while a decentralized execution.

Given that both, the policy and the value of the action change as the models learn from passed experiences (moving targets problem), this algorithm also includes target networks for both the actor and the critic of each agent in order to smooth the learning and increase stability. Furthermore, some noise is added to the actions during training to encourage some level of exploration. Thanks to these features, MADDPG has been proved to be successful in tasks with multiple agents and continuous actions spaces.

In the solution provided, the actor and critic networks of each agent have been implemented using the PyTorch framework and share the following properties:

- The **actor networks** receive a state as input and return an action as output. They include an input layer of 24 neurons (the state space size stacked three times), four hidden layers of 128, 64, 32 and 16 neurons respectively, and a final output layer of 2 neurons (the action space size). All the hidden layers include batch normalization and ReLU as activation function, while the output layer uses the hyperbolic tangent as activation function in order to keep the action values within the valid limits [-1, 1]. Adam has been the chosen optimizer, with a learning rate of 0.001.
- The **critic network** receives an action-state tuple as input and return the estimated Q-value. It presents a slightly more complex structure, with one input layer of 52 neurons (this includes the states and the actions of both agents concatenated), four hidden layers of 512, 64, 32 and 16 neurons respectively and a final output layer of one neuron (to provide the Q-value mapped to the state-action tuple). Once again, the hidden layers include batch normalization and ReLU as activation function, while the output layer does not use any activation function at all. In this network, Adam with a learning rate of 0.001 has also been chosen as optimizer.

These networks properties, along with an exhaustive fine-tuning of the rest of hyperparameters (the reward discount, the batch size, the τ parameter for soft updating the target networks, etc.) have been the key to solve the task. Adapting the critic network for multi-agent training (changing the input with respect to the standard DDPG) and choosing the proper tensor dimensions have also been other critical points for the solution.

Conclusion

The implementation described in the previous section is able to solve the problem in **3395 episodes** (averaging 0.5 points from episode 3395 to 3495), as shown in Fig. 2. Such performance demonstrates the effectiveness of the chosen algorithm.

Engaging with this project has been a great opportunity to explore the MADDPG algorithm and its outstanding capabilities in tasks with multiple agents and continuous action spaces. Nevertheless, it is important to remark that the agents training has not been a trivial task and a careful fine-tuning of their hyperparameters has been required.

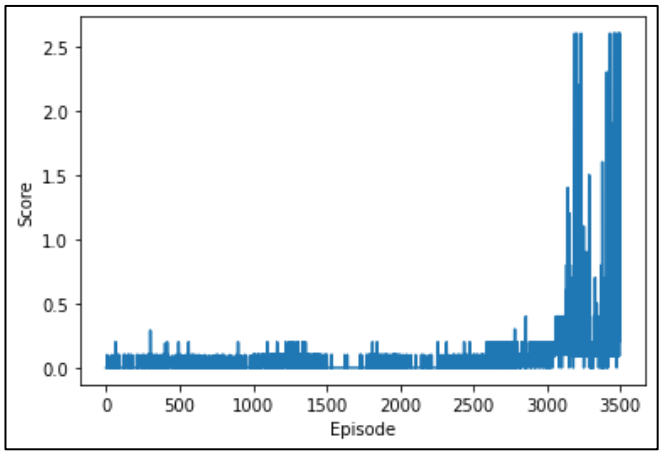

Fig. 2. Agent's score history

Future Work

Even though the agents have successfully learnt to solve the problem, there are some ideas that could have been explored to improve even more their current performance. The most straight forward extension that could be tried to increase the convergence of the algorithm would be adding prioritized experience replay, which would allow the agents to focus on experiences with higher impact. It could also be interesting to try other actor-critic multi-agent methods or even policy-based methods (like Multi-Agent Proximal Policy Optimization or Multi-Agent Asynchronous Advantage Actor-Critic) and compare their results with the MADDPG algorithm.