# Cooperation-Competition

## Introduction

The objective of this project is to train a group of agents capable of controlling tennis rackets with the objective of maintaining a ball in flight.

### The Environment

We will work using a Unity machine learning environment called *Tennis.* In this environment, we control 2 agents trying to bounce a ball over a net. If an agent successfully hits the ball, it receives a reward ok +0.1, but if it lets the ball hit ground or it hits the ball out of bound, it receives a penalty of -0.01.

This environment will be considered solved when we reach an average score of +0.5 over 100 consecutive episodes (taking into account in each episode the maximum score between the 2 agents).

## Algorithms

The environment uses continuous domains for both states and actions. Therefore, we opted to implement Deep Deterministic Policy Gradient, an actor-critic algorithm that could potentially work well within the parameters of the problem. We used a single agent as a starting point to avoid synchronization issues.

Another interesting approach would have been to test how a multi-agent DDPG performed on this task.

## Method

I first tried to use an adaptation of the abstraction I used for the *continuous control* project, but it did not appear to learn even after 10,000 episodes. This could be due to a suboptimal hyper-parameter configuration or some bug in the code.

Nonetheless, after trying to fix the problem for some time, I decided to take a step back and work on a variation of the code reviewed in the previous lessons. Using this abstraction, the code seemed to perform better and we managed to reach a successful result.

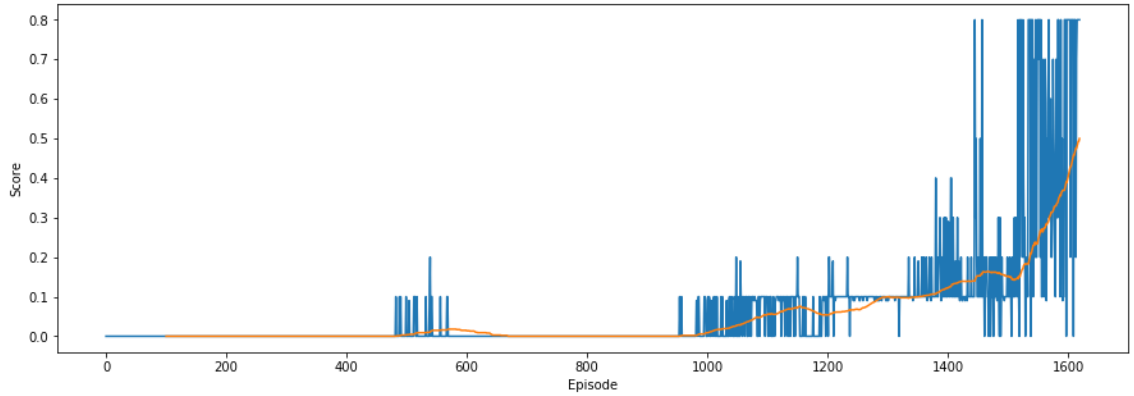
To reach that result, we performed an iterative training over a span of episodes long enough, using early stopping to cease the training process and save the resulting models.

For the actor model, we are using a fully connected neural net with 2 hidden layers (128 nodes and 64 nodes), trained using an Adam optimizer with a learning rate of 0.001 and clipping the output using a hyperbolic tangent.

The critic model consists of a fully connected neural net with 2 hidden layers (128 nodes and 64 nodes), trained using an Adam optimizer with a learning rate of 0.001 and clipping the gradients to 0.5.

To add variability to the training process, we are generating noise over the action vector using the Ornstein-Uhlenbeck process, but the scale of this noise decreases as the training progresses.

## Results




| Episode | Average Score |
|---|---|
| **500** | 0.00 |
| **1000** | 0.01 |
| **1500** | 0.15 |
| **1620** | 0.51 |

The algorithm reaches the success criteria in the interval [1521-1620]

## Conclusions

This environment presents a challenge on how we should add and maintain a variability over the actions. Without any prior knowledge, the most probable outcome is that the ball falls directly to the ground and, without positive examples, the algorithm appears to take a lot of episodes to start learning.

## Next Steps

First of all, we could try to optimize some hyperparameters to check if we can make the learning process more efficient and we could let the algorithm learn unbound to check how learning evolves passed the success criteria.

It would also be interesting to experiment with multi-agent focused algorithms (like MADDPG) to check if it could improve performance.

Another option would be to use different networks for both agents and experiment if there is any advantage on sharing information and how would we the optimum way to perform this synchronization.