Continuous Control

Introduction

The objective of this project is to train an agent capable to control a double-jointed arm to remain inside a moving target zone. The success criteria is defined by reaching an average score above 30 on a 100 episode interval.

The Environment

We will work using a Unity machine learning environment called *Reacher*. This environment displays a controlled number of double-jointed arms and a moving sphere for each one that acts as a target zone.

This environment provides information on 33 variables for each arm to define the state vector and expects a 4D vector of float values clipped between -1 and 1 as an action, corresponding to the torque applied both joints. A reward of +0.1 is provided for each step that the TCP (*tool center point*) is touching the target location.

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.

Other possible solutions would have been to try to implement discretization techniques or lean toward parallelized training techniques like A3C.

Method

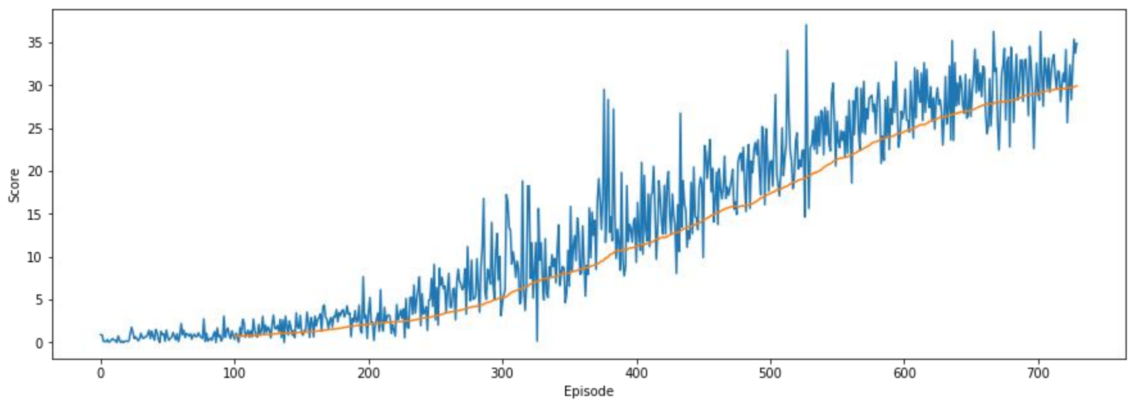
We created a custom abstraction of the DDPG algorithm aiming to create a system that allowed both training with a single agent, but that it also allowed for a smooth transition toward a parallelized training.

We run the model with an initial set of parameters, and it yielded results that were good enough for this project and its time constraints.

In this case we used fully connected networks with 2 hidden layers (400 nodes and 200 nodes) for both the actor and the critic with an Adam optimizer using a learning rate of 0.001 and no weight decay.

We also used a buffer to store previous experiences with a capacity for 10^6 items. This buffer was configured to provide a batch of 1024 experiences by randomly sampling all its items using an equiprobable distribution.

Results



Episode	Average Score
100	0.72
200	2.12
300	5.27
400	11.24
500	17.36
600	24.57
700	29.07
730	30.05

The algorithm reaches the success criteria in the interval [631-730]

Conclusions

DDPG shows an acceptable behavior in this environment and reaches the success criteria in a reasonable period of time.

Next Steps

First of all, it would be interesting to experiment tuning some hyperparameters to try to improve the speed of success and gain more insights on how they can affect training in this environment.

Another approach that I have been trying out is implementing A3C (or another multi-agent training system) to check if we can gain performance by parallelizing some training tasks. Regretfully, due to time constraints, I could not have a viable algorithm at this point.