# Project 2: Continuous control

This report includes a summary of the learning algorithm for the project titled “Continuous control”, its outcome and ideas for follow-up work.

## 1 – Environment

The environment is the Reacher from the Unity ML-Agents, see Figure 1. In this environment, a double-jointed arm can move to target locations illustrated as the green spheres. A reward of +0.1 is provided for each step that the agent's hand is in the goal location. Thus, the goal of the agent is to maintain its position at the target location as long as possible.

The observation space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm. The 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.

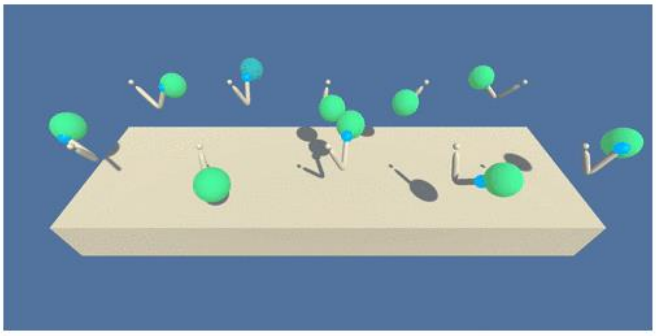

*Figure 1: The Reacher environment from the Unity ML-Agents. The goal is to maintain the position of a double-jointed arm in the target location illustrated with green spheres as long as possible.*

## 2 – Learning Algorithm

In this project, I used the DDPG agent. DDPG is an actor-critic, model-free algorithm based on the deterministic policy gradient which can be used with continuous action spaces. DDPG utilizes the strong suits of the DQN agent, and adapts them to the continuous action domain.

I used Adam for learning the neural network parameters with a learning rate of $10^{-4}$ and $10^{-3}$ for the actor and critic respectively. For Q I included no weight. I used a discount factor of $\gamma = 0.99$. For the soft target updates I used $\tau = 0.001$. The neural networks used the ReLU for all hidden layers. The final output layer of the actor was a Tanh layer, to bound the actions. The networks had 2 hidden layers with 400 and 300 units respectively. Actions were not included until the 2nd hidden layer of Q. The final layer weights and biases of both the actor and critic were initialized from a uniform distribution $[-3\text{x}10^{-3}, 3\text{x}10^{-3}]$. This was to ensure the initial outputs for the policy and value estimates were near zero. The other

layers were initialized from uniform distributions $\left[-\frac{1}{\sqrt{f}}, \frac{1}{\sqrt{f}}\right]$ where f is the fan-in of the layer. I trained with a batch size of 128 using a replay buffer size of $10^5$.

For the exploration noise process I used temporally correlated noise in order to explore well in physical environments that have momentum. I used an Ornstein-Uhlenbeck process with θ= 0.15 and σ = 0.2.

## 3 – Results

Figure 2 shows the average score per episode and the moving mean of the average score over the last 100 episodes. It can be seen that after 179 episodes, the average score of 30.28 was obtained. At that episode, the training was stopped as the requirement for the agent's success was achieved.

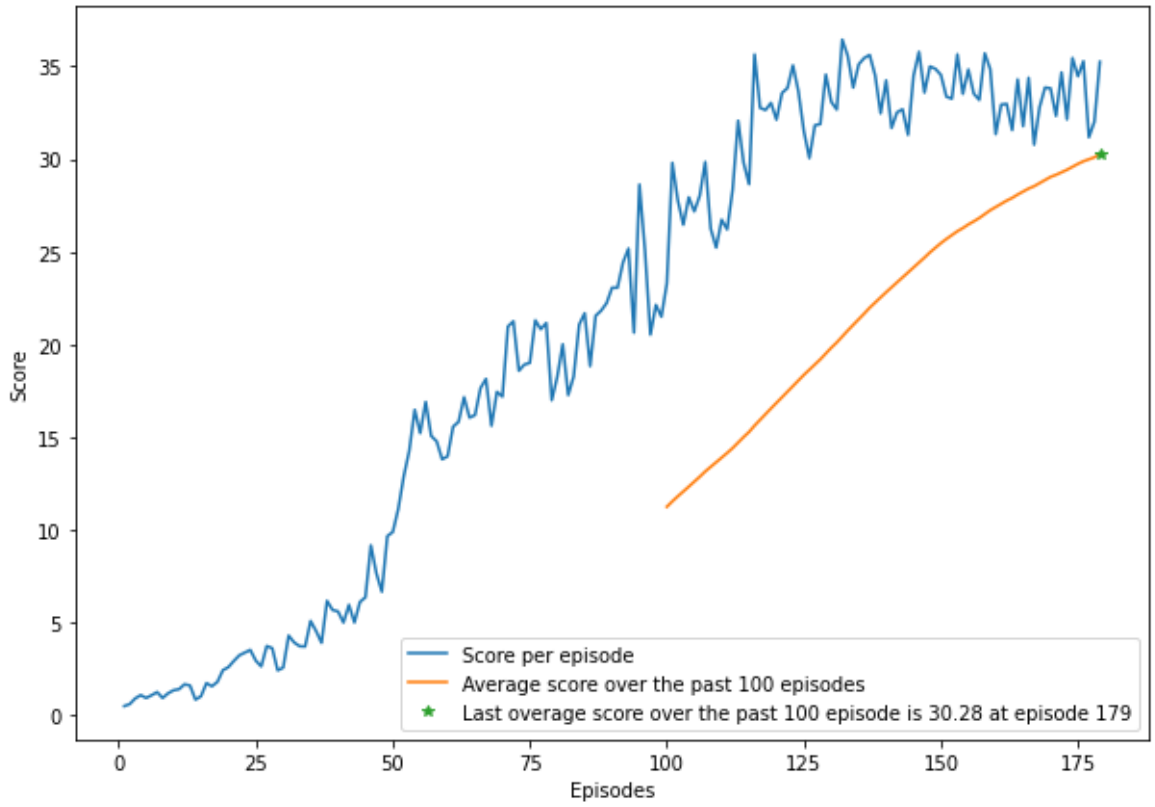


*Figure 2: Average score per episode from the 20 agents.*

## 4 – Ideas for follow-up work

The following ideas come to mind for improving the agent's performance:

- Hyperparameter tuning
- Use Prioritized experience replay
- Adding noise to the parameter of the neural networks