Deep Reinforcement Learning - Project 3

Peter Toke Heden Ahlgren[1]
[1]Mjølner Power ApS

Keywords: Continuous Control, DDPG, Tennis

19th March, 2021

1 Introduction

In the following report we solve the Unity Environment "Tennis" with reinforcement learning agents. The environment consists of 2 agents playing tennis against each other. For every timestep where an agent gets the ball over the net, a reward of 0.1 is given to that agent - otherwise it receives -0.01. For the current project an average score of the best agent per game across 100 episodes of 0.5 is considered a success. The agent gets 24 different inputs as its state and has 2 continuous action variables.

2 Reinforcement Learning Agent

As agent we use a deep reinforcement learner, *i.e.*, a reinforcement learner agent where the Q-table is replaced by a neural network. For this structure to successfully learn, a few tricks has to be implemented: instead of only one network replacing the Q-table we use a *local* and *target* network, such that it is not the same network that for every step is updated and take an action - we also utilize a replay buffer, giving the agent memory, such that also old experiences can still be used. The structure with both *local* and *target* networks are used for both an *actor* and a *critic*, working together in order to solve the environment. The particular implementation chosen for this project is a *Deep Deterministic Policy Gradient* (DDPG) in which the actor learns a policy and the critic learns a Q-value function. DDPG can be thought of as deep Q-learning for continuous actions space. As such, DDPG is an off-policy algorithm. While the actor seeks to find the best policy for the given observation space, the critic tries to estimate the Q-value of the actions taken by the actors, given the current observations.

This project has been the most challenging to solve. As a first attempt a MADDPG agent, as described in the OpenAI multi agent DDPG, MADDPG[*] was used. Here agents share a common critic observing the entire state space, thus being able to judge whether the chosen actions are good depending on all information available to both agents. However, training this setup was *very* unstable, some configurations only worked for a few random number generator seeds, others only for a very narrow range of other parameter settings. Event though some of these setups resulted in unusual high scores, it seems to be a result of overfitting rather than anything else.

However, going back to a single DDPG agent, only observing its own states, and letting that play on both sides of the net, made the trick and resulted in stable learning . Hence, the current solution is more or less the same DDPG agent as used in Project 2, for the case of only 1 Reacher agent. Since this is the last project, things like only soft updating every n'th step or using a Prioritized Replay Buffer has also been implemented.

3 Parameters

For the basic structure, we choose a network having 2 hidden layers for both the actor and the critic with 512 and 256 nodes in the hidden layers. Adding a bit of dropout (0.2) in the critic stabilized learning a little. Activation functions are ReLU between all layers. The actor(s) ends with a $\tanh$ activation function, mapping the output to the $[-1, 1]$ interval. The critic ends in a fully connected layer.

For all examples we use the same values of the following variables:

- Replay buffer size: $1\text{e}5$
- Mini-batch size: 512
- Discount factor, γ: 0.99
- Parameter for soft updating of target, τ: $2\text{e}{-1}$
- Learning rate, actor: $1\text{e}{-4}$

[*]Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments, by Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, Igor Mordatch, https://arxiv.org/abs/1706.02275

- Learning rate, critic: $3e{-}4$
- Learning (updating networks) every n'th step: 10
- Run N update step for every step where learn happens according to the parameter above: 10
- α for Prioritized Replay: 0.6
- β for Prioritized Replay: 0.4

4 Results

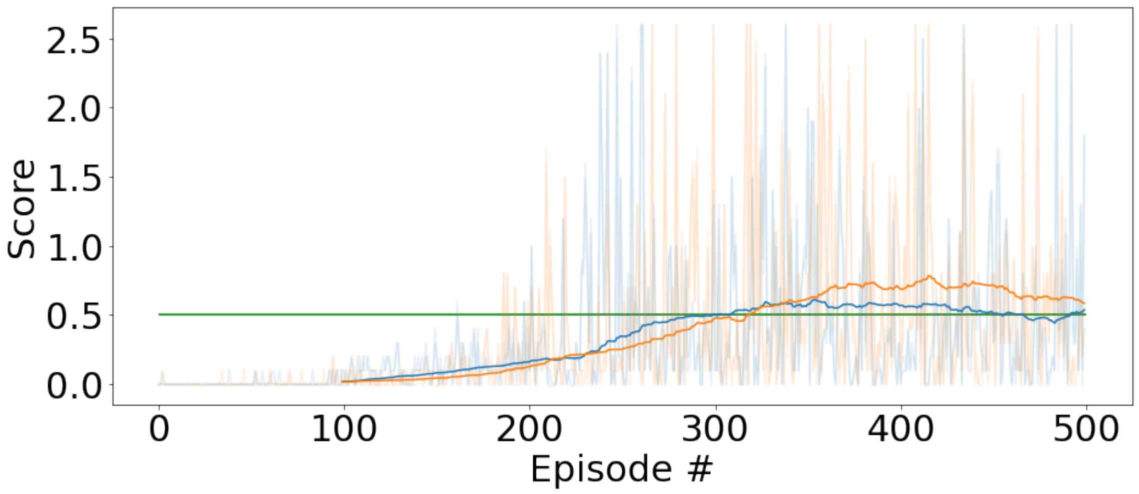

Fig. 1. The reward dynamics for the examined environment. In blue, the learning dynamics of the agent using a Prioritized Replay Buffer is seen, in orange the agent using an Uniform Replay Buffer. The semi-transparent lines are the raw scores, while the solid lines are the running averages of 100 episodes. The success threshold of 0.5 is depicted in green.

The resulting rewards for the environments are depicted on figure 1.

Training the agent was more difficult than for the previous project environments. One quickly finds parameter settings fitting a certain random number seed, but changing the seed could ruin everything. It was only when going for the simple approach with one DDPG-agent playing on both sides of the net, that an actual stable learning process started. For some settings with more advanced agents, rolling mean scores could end up way above 1.5, but these findings were not stable across random seeds. Hence they cannot be considered a true solution of the environment.

On the way to finding this more stable configuration, a Prioritized Replay Buffer was implemented. While making it a lot easier to find values for remaining parameters that leads to a solution, it costs on the upside, in general pushing the best rolling average scores down. However, they tend to increase earlier in the training process than with the regular, Uniform Replay Buffer.

Reaching a solution of the environment for each the 2 versions having different replay buffers, happens almost simultaneously. Hence we consider the environment solved for both agents at **step 400**.

General findings for both agents are that the learning rate for the critic should be higher than the actor, larger mini-batch stabilizes learning (up until some point) and it seems important, to stabilize learning further by only updating networks around every 10th time step. Adding reward bootstrap did not seem to make any noticeable difference, at least in the current implementation.

5 Future Work

First, it would be interesting to make the agent using the Prioritized Replay Buffer perform better than what is the case right now. Both parameters α and β have not been tuned yet. It would also be interesting to solve the environment using PPO, or other agent types. Finally it would be nice to understand why the initial trials with agents being aware of not only own but also the opponents state, was so difficult to get working.