Deep Reinforcement Learning Nanodegree

Project 3 Collaboration and Competition “Tennis”

Ben Hosken - December 2018

Overview

This document provides details of the process, learnings and results from the Collaboration and Competition project completed as part of the Deep RL Nanodegree.

In this environment, two agents control rackets to bounce a ball over a net. If an agent hits the ball over the net, it receives a reward of +0.1. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. Thus, the goal of each agent is to keep the ball in play.

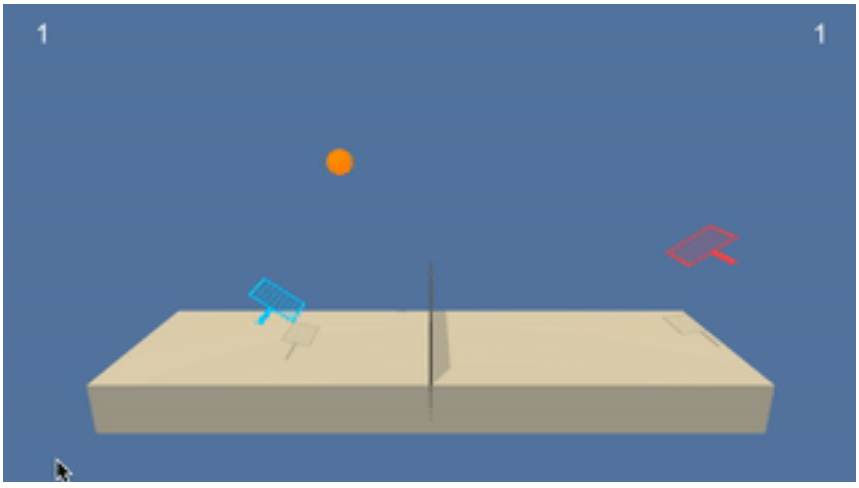
The observation space consists of 8 variables corresponding to the position and velocity of the ball and racket. Each agent receives its own, local observation. Two continuous actions are available, corresponding to movement toward (or away from) the net, and jumping.

The task is episodic, and in order to solve the environment, your agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents)

In this project, we used a single agent which was shared between the two player agents.

Ultimately, the problem was solved using the DDPG algorithm with a two hidden layers (512, 256) with reward clipping, batch normalisation and minor adjustments to the agent parameters.

The agent was trained on a MacBook Pro and solved the environment within 15 minutes while travelling on the train home :)



Approach

I used the same Deep Deterministic Policy Gradients algorithm from the prior project for this project. DDPG is an actor-critic, model free algorithm that works well for continuous action spaces. The actor component focuses on learning the policy trajectories, while the critic learns an estimated value function.

There are two neural networks within the DDPG algorithm. The actor estimates the policy and the critic estimates the value. Experience replay buffering is used to avoid over correlation in the learning. Soft updates are made to the target networks and replay sampling can be done each episode or after a number of episodes.

Model architecture and hyper-parameters

Ultimately, we utilised a two hidden layer model with 512 units in the first and 256 units in the second and used the same model in the actor and final layers of the critic. Batch normalisation was applied in both the actor and critic after a single layer and ReLu activations were used.

Soft updates were used with weight value increased from the prior project. Additionally each learning step applied updated 4 times and updates were made every 10 steps. No noise was applied.

The following hyper parameters were used in the final run.

```
BUFFER_SIZE = int(1e6)  # replay buffer size
BATCH_SIZE = 128        # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-1              # for soft update of target parameters
LR_ACTOR = 1e-4         # learning rate of the actor
LR_CRITIC = 3e-4        # learning rate of the critic
WEIGHT_DECAY = 0.0000   # L2 weight decay
NUM_UPDATES = 4         # update the agent multiple times sample
```

Challenges and learnings

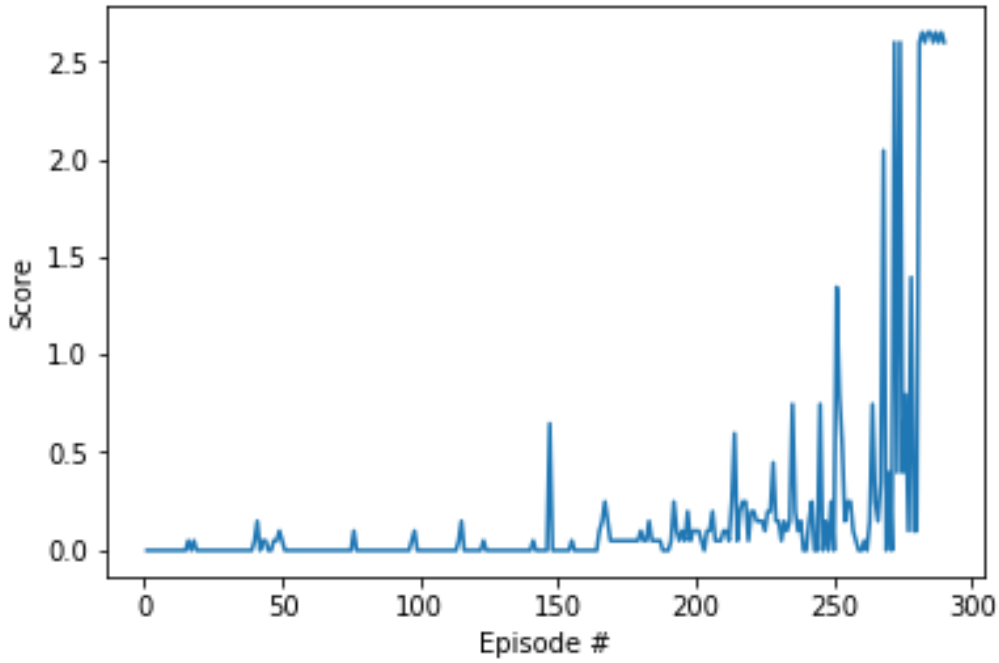
Overall, this project was fairly straightforward when compared with the prior project. The environment was solved fairly quickly and only minimal exploration/changes to the hyper parameters were required to achieve success.

It was fairly clear, and quick, to identify when there was no learning taking place as the average score remained very low for a long time. In all successful runs, with differing parameters, there was a slow start then and exponential improvement. This pattern meant we could easily identify non learning solutions.

In this instance, larger soft updates dramatically improved performance and the slightly different model architecture size also helped.

Results

Using the structure and parameters detail above, the environment was solved in 290 episodes. Below is a chart showing the results from the final run. The results were consistent across a range of runs with the environment being solved in around 300 episodes.



As can be seen, there was an initial period of slow improvement followed by an acceleration in learning towards the end.

Future Ideas and improvements

Further enhancements would be to use two agents which work independently and then share the weights every so often.

Surprisingly, the environment took very little time to train. As such, there is little suggested improvements for the processing.