Project 3: Collaboration and Competition – Report

Introduction

The goal of the project is to train an agent to solve the Tennis environment. 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, the agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents). Specifically,

- After each episode, we add up the rewards that each agent received (without discounting), to get a score for each agent. This yields 2 (potentially different) scores. We then take the maximum of these 2 scores.

- This yields a single score for each episode.

The environment is considered solved, when the average (over 100 episodes) of those scores is at least +0.5.

Model

I used an actor and a critic model. Both are Deep Neural Networks (DNN) with 2 hidden layers. Both of them are activated with the relu function, in order to introduce non-linearity into the model. Further, both DNNs have an input layer with 24 neurons, matching the state space, which consists of 3 consecutive observations with a size of 8 as explained in the introduction. The hidden layers of of the actor model have 256 and 128 neurons. The output of the actor model has 2 neurons, matching the actions space. The hidden layers of the critic model have a baseline of 256 and 128 neurons. However, the input of the second hidden layer gets compounded with the 2 output neurons of the actor network resulting in 130 neurons for the second hidden layer. The output layer of the critic network has 1 neuron representing the target value.

Algorithm

To train the agent, I used the DDPG algorithm, which implements a deterministic policy for a continuous action space. DDPG can be seen as an actor-critical method because it uses the same structure as actor-critical methods, but the underlying mechanism is similar to the DQN algorithm. Therefore, it can also be seen as an extension of DQN to continuous action space. I uses my implementation of the second project as a starting point, which was based on the ddpg-pendulum example from the udacity github:

https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-pendulum.

To make the code work, I changed the way data is exchanged with the environment, since there are differences between openai-gym environments and unity environments. I also extended the code to work with multiple agents. This was done by adding the experience of both agents to the replay buffer and extending the OUNoise class to generate process noise for multiple agents. To solve the second project, I introduced some changes to the example code from udacity as explained in the following:

1) I changed the process noise from a uniform distribution to a normal distribution. The reason was that the uniform distribution samples between [0,1] and introduces a bias because negative numbers are not considered. I also implemented a noise reduction. So the influence of the process noise decreases over time.

2) I reduced the number of neurons in the critic network to avoid overfitting. I also added a batch normalization layer after the input layer of the critic to make training faster and more stable. The reason why this is effective is still under discussion. The authors of the original paper claim that batch normalization can mitigate the problem of internal covariance shift, but recent papers claim different reasons. Nevertheless, in my case it had a big impact on the learning speed. I didn't change the rest of the implementation, as it already gave sufficient results.

With these changes the second project could be solved with sufficient results, but for the third project they weren't quite enough. After a lot of experimenting, I found out that the training success is very sensitive to the hyperparameters, especially the learning rate. By trying different hyperparameters, I found a combination that reliably solved the environment. However, how fast the agent learns is quite variable. The best result I had was solving the environment after 160 episodes, which I couldn't reproduce at all, even without changing anything in the code. It seems that the learning success depends strongly on the randomness of the training process. I was able to make it a bit more stable by doubling the batch size and introducing a new hyperparameter that controls how many timesteps are collected before learning.

The following hyperparameters were used for training:

- Replay buffer size: 1e5
- Minibatch size:256
- Discount factor = 0.99
- Soft update of target parameters: 1e-2
- Learning rate of the actor:1e-4
- Learning rate of the critic: 1e-4
- L2 weight decay: 0
- Time steps before learning: 10
- Number of learning runs: 20

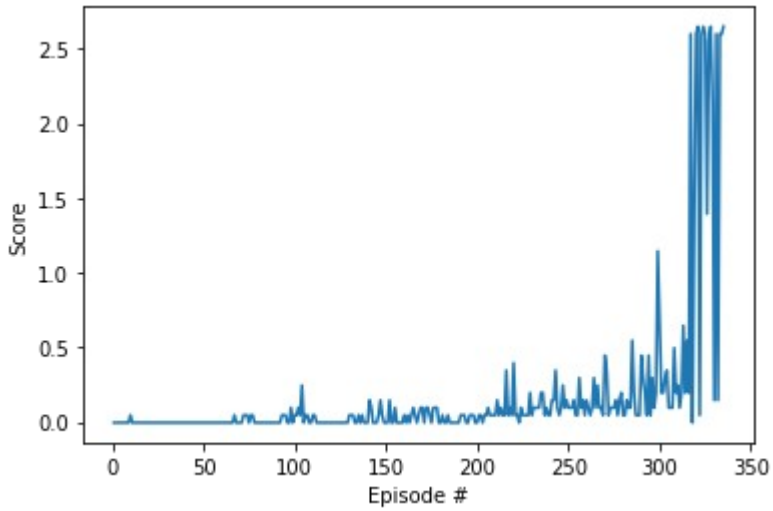
Results

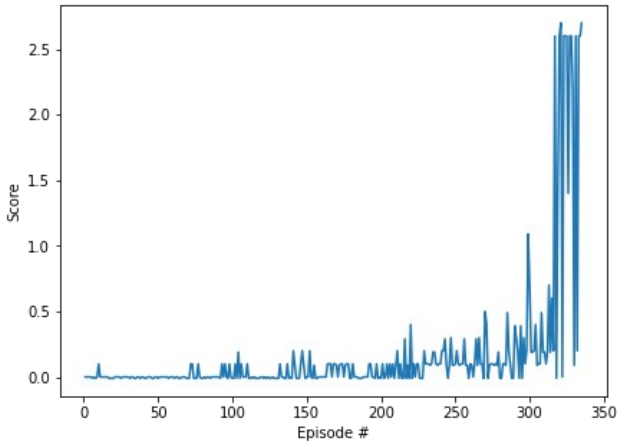
With the described network and training algorithm, an average score above 0.5 over 100 consecutive episodes was reached after 336 episodes as shown in the output below.

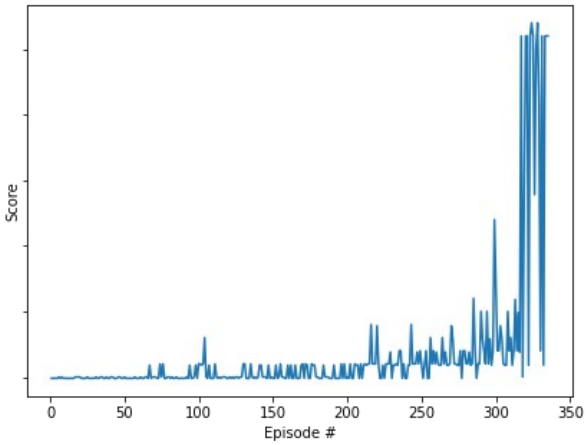
```
Episode 10      Average Score: 0.00     Score min: -0.01        Score max: 0.10 Score avg: 0.05
Episode 20      Average Score: -0.00    Score min: -0.01        Score max: 0.00 Score avg: -0.00
Episode 30      Average Score: -0.00    Score min: -0.01        Score max: 0.00 Score avg: -0.00
Episode 40      Average Score: -0.00    Score min: -0.01        Score max: 0.00 Score avg: -0.00
Episode 50      Average Score: -0.00    Score min: -0.01        Score max: 0.00 Score avg: -0.00
Episode 60      Average Score: -0.00    Score min: -0.01        Score max: 0.00 Score avg: -0.00
Episode 70      Average Score: -0.00    Score min: -0.01        Score max: 0.00 Score avg: -0.00
Episode 80      Average Score: -0.00    Score min: -0.01        Score max: 0.00 Score avg: -0.00
Episode 90      Average Score: -0.00    Score min: -0.01        Score max: 0.00 Score avg: -0.00
Episode 100     Average Score: 0.00     Score min: -0.01        Score max: 0.10 Score avg: 0.05
Episode 110     Average Score: 0.01     Score min: -0.01        Score max: 0.10 Score avg: 0.05
Episode 120     Average Score: 0.01     Score min: -0.01        Score max: 0.00 Score avg: -0.00
Episode 130     Average Score: 0.01     Score min: -0.01        Score max: 0.10 Score avg: 0.05
Episode 140     Average Score: 0.01     Score min: -0.01        Score max: 0.00 Score avg: -0.00
Episode 150     Average Score: 0.02     Score min: -0.01        Score max: 0.00 Score avg: -0.00
Episode 160     Average Score: 0.02     Score min: 0.00 Score max: 0.09 Score avg: 0.05
Episode 170     Average Score: 0.02     Score min: 0.09 Score max: 0.10 Score avg: 0.10
Episode 180     Average Score: 0.03     Score min: -0.01        Score max: 0.00 Score avg: -0.00
Episode 190     Average Score: 0.03     Score min: -0.01        Score max: 0.00 Score avg: -0.00
Episode 200     Average Score: 0.03     Score min: -0.01        Score max: 0.00 Score avg: -0.00
Episode 210     Average Score: 0.03     Score min: -0.01        Score max: 0.10 Score avg: 0.05
Episode 220     Average Score: 0.04     Score min: 0.39 Score max: 0.40 Score avg: 0.40
Episode 230     Average Score: 0.04     Score min: -0.01        Score max: 0.10 Score avg: 0.05
Episode 240     Average Score: 0.05     Score min: -0.01        Score max: 0.10 Score avg: 0.05
Episode 250     Average Score: 0.06     Score min: 0.09 Score max: 0.10 Score avg: 0.10
Episode 260     Average Score: 0.07     Score min: 0.10 Score max: 0.19 Score avg: 0.15
Episode 270     Average Score: 0.08     Score min: 0.39 Score max: 0.50 Score avg: 0.45
Episode 280     Average Score: 0.09     Score min: -0.01        Score max: 0.10 Score avg: 0.05
Episode 290     Average Score: 0.11     Score min: 0.39 Score max: 0.50 Score avg: 0.45
Episode 300     Average Score: 0.14     Score min: 0.69 Score max: 0.70 Score avg: 0.70
Episode 310     Average Score: 0.16     Score min: 0.19 Score max: 0.30 Score avg: 0.25
Episode 320     Average Score: 0.23     Score min: 2.60 Score max: 2.60 Score avg: 2.60
Episode 330     Average Score: 0.42     Score min: 0.09 Score max: 0.20 Score avg: 0.15

Environment solved in 335 episodes!     Average Score: 0.52
```

A plot of the progression of the average score over both agents can be found below, followed by the score of each agent separately.







Future Work

For the future, I would like to make the training process more stable. I think that implementing a prioritized replay buffer would be the best chance for improvement. Even though the prioritized replay buffer is mostly used to speed up the learning process, I would imagine that by reducing the randomness of the learning process, the process would become more stable.