Deep RL Nanodegree – Collaboration & Competition

Udacity Nanodegree Program

Prabowo Setiawan

May 24, 2020

Introduction

This report aims to examine and discuss the collaboration and competition project, both the implementation and the results. The environment consists of 2 agents playing table tennis. A reward of +0.1 is given if the agent is successfully hit the ball over the net. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01 instead.

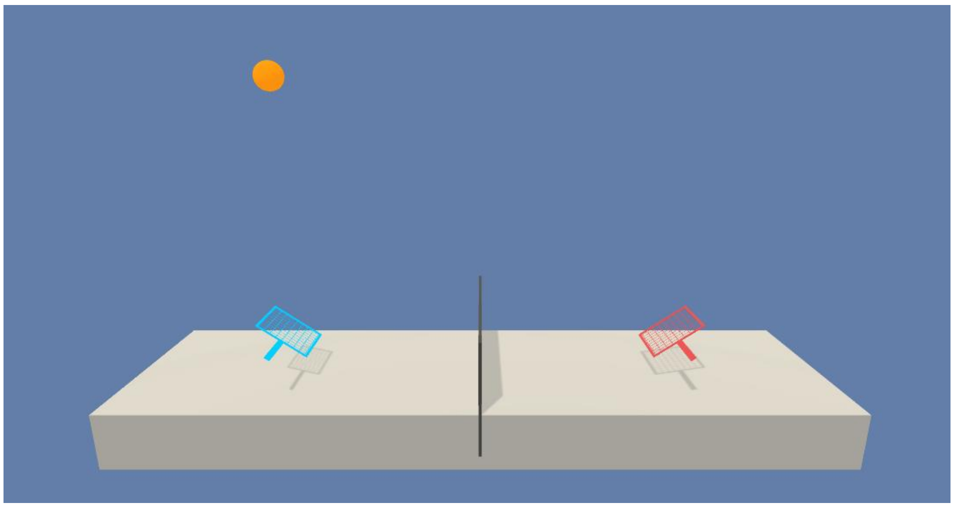

Figure 1: The Unity environment

In each episode, the agents are expected to obtain as many rewards or scores as possible. There are several ways that the agent can achieve this in the environment. The following are important components of the tennis environment:

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

Based on the information above, the two agents need to adjust their actions to attain the highest rewards by playing well against each other. To put it simply, the ideal goal of this environment is to have the two agents having the longest rally as possible. The agents are considered to solve the environment once they attain an average score of at least 0.5 over the past 100 consecutive episodes given that the maximum score between the two agents is the one to be computed and stored.

Project Approach

In order to solve the environment, a policy-based method known as Deep Deterministic Policy Gradient (DDPG) is implemented. DDPG also uses several other components which are not commonly implemented in the DQN. One common aspect, however, is that the DDPG still uses replay buffer in stabilizing the learning process of the agent. It is noted, however, that the project involves two agents instead of a single agent. To address this issue, a method of Multi Agent DDPG is implemented. Multi Agent DDPG or MADDPG is essentially DDPG with more than one agent involved within the environment.

DDPG is a model-free approach in which the algorithm learns competitive policies in solving an environment (Lillicrap et al, 2016). In this algorithm, Actor and Critic network is defined using the following architecture:

- Input: size of state space
- Layer 1: 256 layers with ReLU activation function
- Layer 2: 128 layers with ReLU activation function
    - Critic's layer 2 input is a concatenation of the first layer output and action vectors

- Output: size of action vectors
    - Actor's output layer uses tanh activation function while Critic's does not

This configuration is chosen based on trial and error, although it did provide a well performed model. The next parameter to be determined is the noise generation. In order to encourage the algorithm to explore the environment, a normal distribution random noise is introduced. This noise has its assigned weight, minimum value, as well as decay rate. The following combination of values are found to be quite optimal in training the agents:

- noise_weight = 1
- noise_min = 0.1
- noise_decay = 0.997

The noise is added onto the actions determined by the DDPG single agent class, thus encouraging the agents to explore the environment initially, then learn to exploit their options eventually. This is essentially quite similar to the implementation of epsilon-greedy choosing actions.

After implementing these aspects, the additional change made to the DDPG algorithm is basically how to put the entire algorithm in a separate class that relies on the DDPG single agent class. This class is saved under ma_ddpg.py and is the main class of MADDPG algorithm. The code's structure is written by keeping the MADDPG pseudocode in mind (Lowe et al, 2017).

Results

Based on the agent trained using the chosen hyperparameters and random seed number, the MADDPG's performance can be observed below.

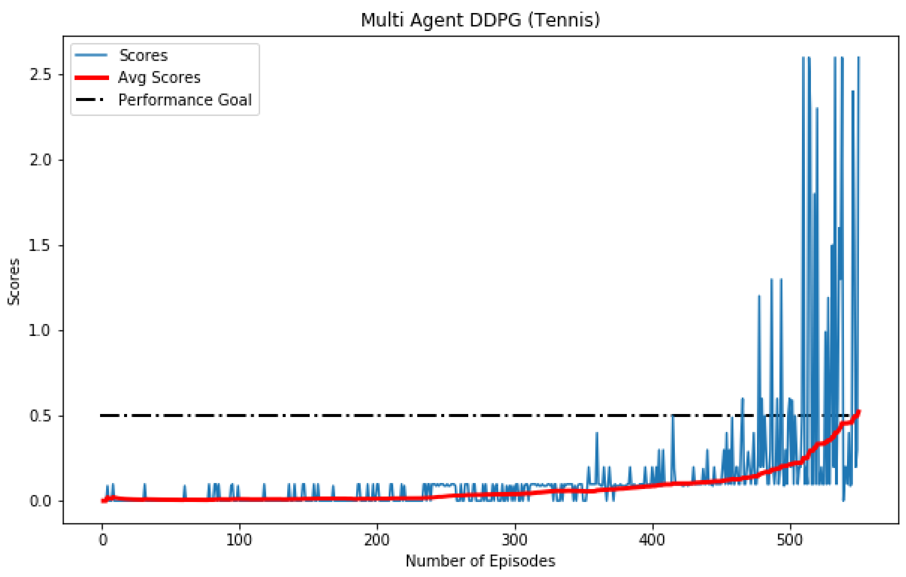

Figure 2: MADDPG Scores and Average Scores

It can be seen from Figure 2 that MADDPG is able to solve the environment after 550 episodes, ending up with a final average of 0.52. The plot shows how the score values are basically non-changing for the first 350 episodes. There are spikes indicating how the scores of 0.09 to 0.10 occur at the beginning of the episodes. Yet, the rest seem to be zero, or in other words, the agents are not able to hit the ball even once. This is an interesting observation which will be discussed further in the next section. Keep in mind that this result is attained even after further exploration encouragement from the normal distribution random noise.

Discussion

The plots shown above suggest that the MADDPG is successful in completing the environment. Unlike the previous projects in which the agent has shown consistent climb in scores, this method does seem to be oscillating a lot at the beginning of the training phase.

Rather than concluding that the agents are unable to learn consistently, however, it is more likely that the agents are lacking in terms of positive feedback in episode 0 to 300. Most of the scores obtained during this period are mostly 0s. This caused problem for the agents to learn in a timely manner. However, once the number of scores above 0 starts to populate the plot, the agents show a significant increase in scores starting from around episode 350 which eventually shoots up to 2.5 or even more after 500 episodes. In the process of completing the project, numerous combinations of hyperparameters were implemented for DDPG. Among 10 or so combinations, these hyperparameters were found to be the most optimal:

1. noise_weight = 1.0
2. noise_min = 0.1
3. noise_decay = 0.997
4. gamma = 0.99
5. tau = 0.01
6. learning_rate
    a. 0.0001 for Actor
    b. 0.0003 for Critic
7. neural network
    a. hidden_layer_1 = 256, with ReLU
    b. hidden_layer_2 = 128, with ReLU
    c. output_layer
        i. tanh activation function for Actor
        ii. no activation function for Critic
8. targetUpdateNet = 1
9. num_update = 1
10. noise_update = 1
11. seed = 0

It is crucial to note that this method can be significantly improved by finding a methodical solution to sampling the possible actions at the beginning of the training phase more effectively. One limiting factor in exploring all of these hyperparameters is definitely the time it takes to train the agent.

Therefore, the model can be improved further by either tweaking the MADDPG's hyperparameters or the architecture of the neural network itself, e.g. learning rate, number of hidden layers, number of neurons per hidden layer, activation function (LeakyReLU instead of ReLU), etc.

Future Improvements

Just like other environments, the Tennis environment has its own unique challenges. The main challenge in solving this problem is to establish an effective way of initial exploration of the environment. It seems that the algorithm needs a relatively large number of episodes due to the abundant of 0 scores at the beginning of the training process. Therefore, a method to better sample this random noise is strongly preferred.

Another method to speed up the learning process is by implementing Prioritized Experience Replay (PER). This allows the algorithm to sample many important and significant experiences rather than just randomly sampling the experience replay memory. In addition, other methods such as Proximal Policy Optimization (PPO) can also be explored in solving this environment.

References

Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. 2016. Continuous control with deep reinforcement learning. In: *ICLR 2016*. [Online]. [Accessed 22 May 2020]. Available from: https://arxiv.org/pdf/1509.02971.pdf

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. [Online]. [Accessed 23 May 2020]. Available from: https://arxiv.org/pdf/1706.02275.pdf