Report Navigation Project

In this report I describe how I solved the tennis playing project with multi agent DDPG.

Learning Algorithm

The learning algorithm was based on my DDPG implementation used for the previous project with the Reacher task. I simply extended my algorithm to work with multiple agents, which was mainly an ajustutment creating a distinc class DDPGAgent and MADDPGCollective, where DDPGAgent was containing an actor, critic, target actor and target critic network respectively and the MADDPGCollective was containing the necessary DDPG agents and was responsible for training them and also at execution time for their management.
As suggested in the project sections Benchmark implementation of Project 2, I kept a variable LEARN_AFTER_TS, which only made the agent collective learning from his experiences after LEARN_AFTER_TS timesteps and NR_UPDATES, which if the agent collective made it to the learning section in the step method, then it made the agent collective sample NR_UPDATES times an experience from which it then learned. Once I modified and cleaned up the code, I ran a training with the model which did not learn at all.
Therefore I started to experiment with the network size and the hyperparameters `LEARN_AFTER_TS` and `TIMESTEPS_PER_EPISODE.`

Hyperparameters

In the end I successfully managed to train the networks with following hyperparameters:
An episode was defined by the variable TIMESTEPS_PER_EPISODE which I set to be 500. Then I also defined an Epoch to be 100 Episodes with the variable EPISODES_PER_EPOCH set to 100. Last I chose the agent to do maximally 10 Epochs for the learning with the variable EPOCHS. Parameters taken from the Pendulum project were not changed for this project. Means they were the following:

- Buffer size = 1’000’000
- Batch size = 128
- Gamma = 0.99
- Tau = 0.001
- LR Actor = 0.001
- LR Critic = 0.001
- Weight Decay = 0

After playing a bit with NR_UPDATES and LEARN_AFTER_TS, I found myself that setting them both to 10 speeded up learning the most.
The value of GRADIENT_CLIP_THRESHOLD (critic gradient clipping) was 1, which seemed to be a common choice in the Reinforcement Learning community.

For the decay rate of the Ornstein-Uhlenbeck-noise, epsilon was set to 1 initially and then decreased after each learning step to 99% of the previous value.

Plot of Rewards

In Figure 1 I show the scores of the agent collective while training. The environment was solved after 1056 episodes of training. However, the agent was programmed to do all the desired epochs of training. The agent trained for 20 epochs, means 2000 episodes. Figure 1:

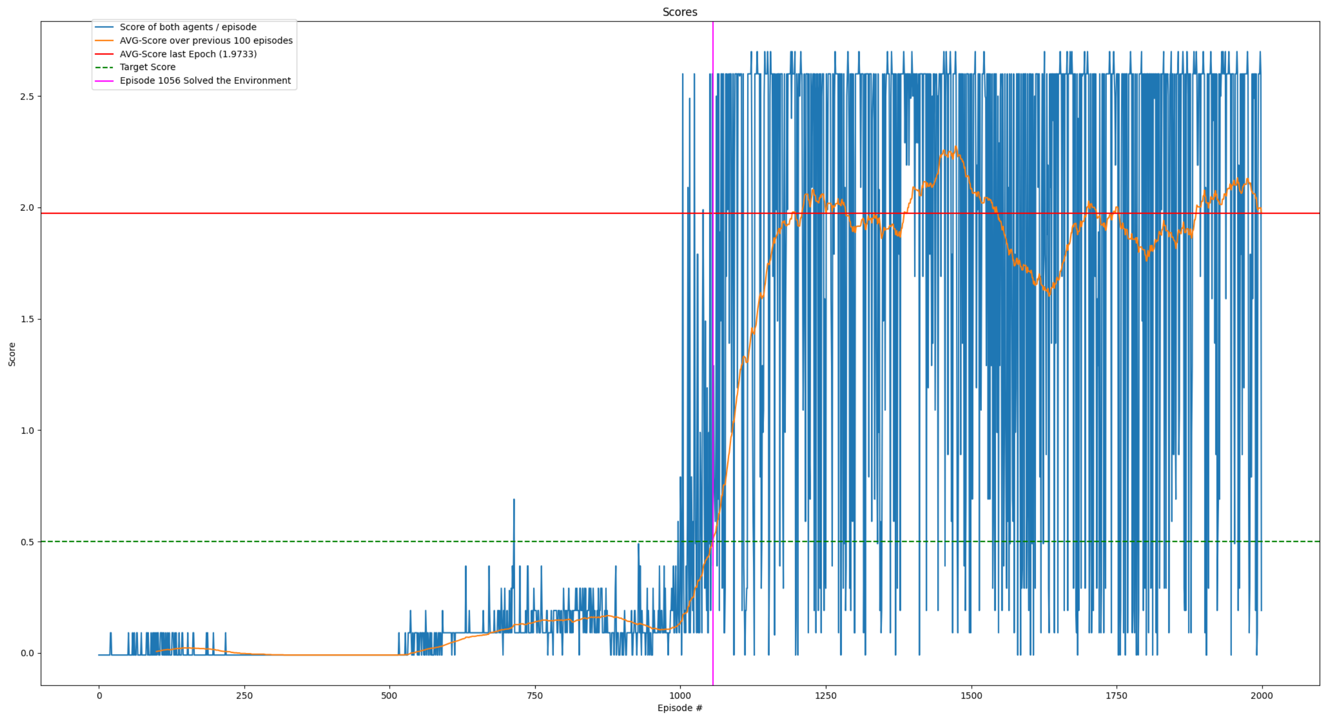

Figure 1: Scores of the training of the agent collective which solved the problem. The green dashed line shows the target score for the agents which it must be above for 100 episodes to solve the environment. The red line shows the final average score of the agent collective over 100 episodes. The magenta solid vertical line marks the episode at which the agent collective managed to solve the environment. The blue plot shows the total score of the agent collective per episode and the orange plot shows the average score over 100 episodes of the agent collective.

Ideas for Future Work

Problems of the training are that the training of the agents is very unstable. Some more hyperparameter tuning on the Ornstein-Uhlenbeck or a different exploration strategy might bring more stability to the system. Further on I suspect that a prioritized experience replay buffer might bring also better results on the training. Last but not least Iqbal et al. [1] proposed an improvement on MADDPG which utilizes attention in order to select relevant information for estimating critics.