Project 3: Collaboration and Competition

In this report I briefly summarize the learnings and final modeling decisions taken as part of the Collaboration and Competition project. After several attempts with different hyperparameters and models, I was able to find a setup that solves the environment with around 3200 steps. I imple- mented several optimizations but the main criterion to solve the environment in my experience was a long enough training time (sufficient number of episodes).

Basic approach

After working with DDPG in Project 2 - Continuous Control I decided to stick with it and use Multi Agent Deep Deterministic Policy Gradients (MADDPG [1]) in this project.

In this model every agent itself is modeled as a Deep Deterministic Policy Gradient (DDPG[2]) agent where, however, some information is shared between the agents. In particular, each of the agents in this model has its own actor and critic model. The actors each receive as input the individual state (observations) of the agent and output a (two-dimensional) action. The critic model of each actor, however, receives the states and actions of all actors concatenated. This should facilitate the information sharing between the agents. As in the Multi Agend RL lab in the nanodegree and as pointed out by other participants, I chose to input a concatenation of all states and all actions into the critic models already into the first hidden layer (contrary to the last project where the actions were concatenated to the output of the first hidden layer).

Throughout training the agents all use a common experience replay buffer (a set of stored pre- vious 1-step experiences) and draw independent samples.

Network Architectures

Throughout my experimentation phase I did not change the basic network architectures much in terms of number of layers and units: it was always 2 fully connected hidden layers with ReLu activations for both the actors and critics. I tried several combinations for the two hidden layers and used 256/256 in my final setup.

Since it made a big difference for me in the last project I decided to keep one batch normalization layer in the architecture for all actor and all critic models. I tried adding another batch normalization layer per model but it did not make a big difference. Another element I kept from the last project was to set the weights of local and target actor, respectively, local and target critic to the same values at initialization time (see the self.hard copy weights function in the DDPG class.) I also experimented with a few different random seeds for the agent but did not see any major differences in convergence behavior.

Hyperparameters

In terms of hyperparameters I did a fair bit of experimenting. Here are the final choices and some observations. For the learning rates I chose 10^{-4} for the actors and 10^{-3} for the critics. Larger learning rates did not seem to work or give significant speed ups. The optimizer was Adam in all cases.

[1]"Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments" by Lowe et al. [2]"Continuous Control with Deep Reinforcement Learning" by Lillicrap et al.

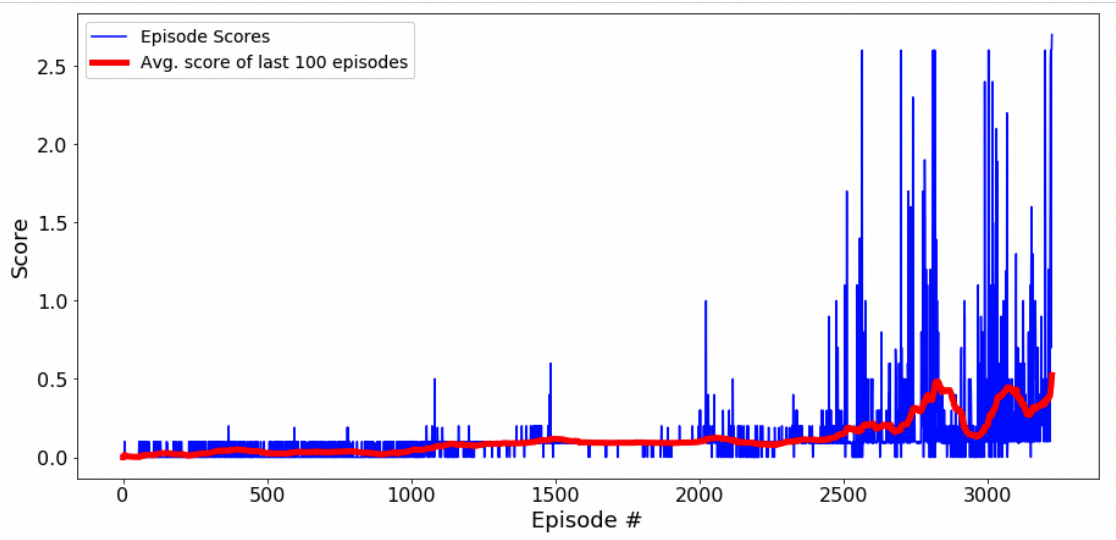

Figure 1: Example training run of the agent. The environment was solved in around 3200 episodes.

For the noise I chose to start with an initial amount and keep it constant until a certain number of overall time steps (30000) had been done throughout training. (Looking at an average number of 30 – 40 of time steps in an episode to begin with the noise would have been removed from training after 800 – 1000 episodes).

A hint that was made by a colleague in the Slack channel was to adapt the τ parameter used in the soft target model update. I eventually chose to use 10^{-3} and found it to work (even larger values might have been tried).

Further parameter choices include: batch size = 256, replay buffer size = 10000 and weight decay = 0.

Results

As outlined above the environment could be solved in ~ 3200 episodes, significantly more than what I expected or what I observed in the last project (somehow the collaborative aspect seems to introduce difficulties into the process). Below, we have the results of a training run with the final setup (individual episode scores are shown). Progress is very slow to begin with, which is not unexpected due to the added noise in the process. The agent then stalls at an average score of ~ 0.1 for more than 1000 episodes starting at episode ~ 1200. Only after around 2500 episodes there is some dynamic in the training. The agent's performance improves drastically starting at episode ~ 2700, then drops back down and rises with one short dip to reach average score 0.5 at episode 3221.

Another observation from looking at the individual episode scores: Even later on in training the performance is rather unstable. There are episodes with scores as high as 2.5 but also episodes with scores close to 0.

Further work

One of the most obvious directions for additional research would be the change of the model. In particular, one could try Proximal Policy Optimization (PPO) on this task, which seems to have

worked for other people in the nanodegree. I could imagine it to work quite well on a problem like this, that is not very high-dimensional.

There are, however, also potential improvements to be looked at within the approach I chose. As usual with experience replay, the sampling from the buffer may be a crucial element of the model that could offer some levers of improvement. I did not systematically look into that. In an environment like this with sparse positive rewards to begin with, optimized sampling could speed up learning.

Another direction of work I imagine valuable is the choice and decay of noise introduced through- out training. I chose to stop noise after a certain number of training steps. But looking at better ways of initializing and gradually reducing the noise could be interesting.