Report Navigation Project

In this report I describe how I solved the second project. I focused on solving the Reacher environment with 20 agents. I never trained the model with a single agent, however, the solution model is applicable to a single agent and the training code should also work with the single agent environment (in fact with any number of agents >= 1). I define for this report the 20 agents to be called an "agent collective".

Learning Algorithm

The learning algorithm was based on the template DDPG model from the Pendulum example environment. As suggested in the project sections Benchmark implementation, I implemented 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. Since I assumed that it was the fault of the model/agent implementation that my agent did not learn at all, I started to implement tricks which could make the agent learn.
First, I added batch normalization layers after the first fully connected layer in both the actor and the critic. There I was hoping that the "not-learning" effect was due to high noise across all the agents. It did not help.
Second, I applied gradient clipping on the critic with an upper threshold of 1. This also did not seem to help.
As a third solution idea, I implemented an epsilon decay on the Ornstein-Uhlenbeck-noise to decrease exploration of the learning process with time.
Finally, I found out that the issue was a copy-paste error from the pendulum project. I checked at each training timestep if the array "dones", which contains the flags if any of the robots was done with its episode, was simply checked as an exit condition for the episode. Since doing so always took the address of the array as condition, which always yielded a positive value, each episode was immediately aborted after a single timestep. Wrapping the array "dones" with the numpy method "any()" did the trick to check if any of the robots was done with its episode and the training went on very well from that point on.

Hyperparameters

I defined some parameters in the "agents/ddpg_agent.py" file. An episode was defined by the variable TIMESTEPS_PER_EPISODE which I set to be 1000. 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 263 episodes of training. However, the agent was programmed to always finish an Epoch before aborting the training. The agent trained for 3 epochs, means 300 episodes.

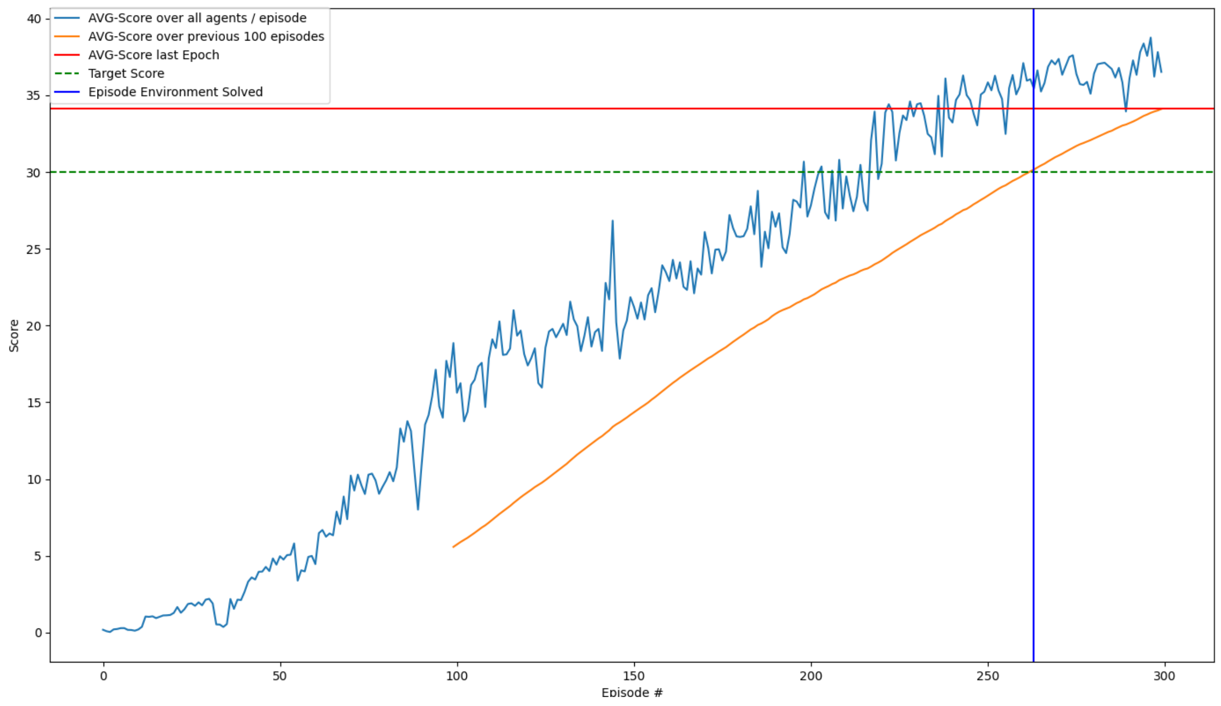

Figure 1: The green dashed line shows the target score for the agent which it must be above for 100 episodes to solve the environment. The red dashed line shows the final average score of the agent collective over 100 episodes. The blue solid vertical line marks the episode at which the agent collective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

Ideas for Future Work

As future work one could also implement the idea of Hou et al.[1], which suggests that a prioritized experience replay buffer can reduce the training time, improve the stability of the training process and is less sensitive to the change in hyperparameters. Another idea would be to test the proposed technique of Fujimoto et al.[2], which states that overestimated value estimates can lead to suboptimal policies and therefore are a problem to the DDPG algorithm. As a solution they extended the DDPG critic with a Double Q-Learning model.