DRLND 3 Project Report

Ariel Kwiatkowski

May 2020

1 Methods and results

The approach I used is the PPO algorithm using an MLP model as the policy and value networks. Including the input and output sizes, the neural network layers are: 24, 32, 32, 32, 32 (followed by 4 for the policy network, and 1 for the value network). The network uses Leaky ReLU activations in all intermediate layers. It's optimized using the Adam algorithm with the learning rate $3 \cdot 10^{-4}$, performing updates in 10 minibatches across a batch of 500 transitions each step. PPO performs 5 full sweeps through the entire batch.

The process can be summarized as follows:

1. Gather a batch of 500 environment transitions
2. Compute discounted rewards to go with TD estimation, and advantages using GAE, for each agent separately
3. Perform 5 runs over the that data divided in 10 minibatches, performing a gradient update each time, again separately for each agent

The graph of rewards can be seen in Figures 1 and 2, showing the mean return of respectively the first and the second agent. It' pretty clear that the training is quite unstable, reaching the 0.5 threshold and then falling below it again. In the end, it stayed above it, although even if it didn't, I kept all the previous agents to be able to choose the best ones at the end.

After the training, I evaluated the final agents for 100 episodes to check whether they managed to solve the environment. Indeed, the mean return of that, computed the way it's described in the exercise, was $1.004 > 0.5$ which means the goal has been achieved.

2 Possible improvements

The first thing that comes to mind is a more extensive hyperparameter search – reinforcement learning in general is quite sensitive to that, so it's possible that e.g. a deeper network or a larger learning rate would perform better.

More interestingly, my approach didn't actually use the fact that the environment is multiagent. The agents don't actually interact that much, since

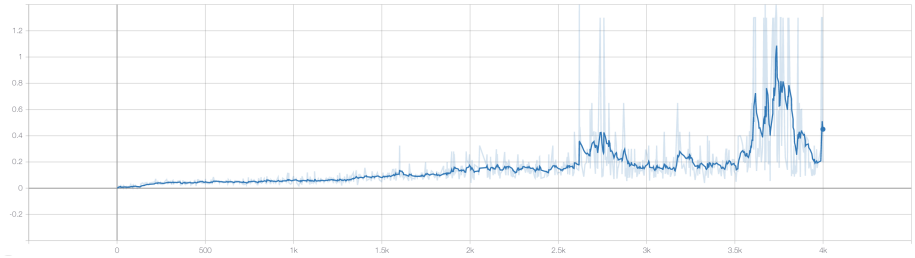

Figure 1: Rewards obtained by the first agent, along with a smoothed out line (taken from TensorBoard)

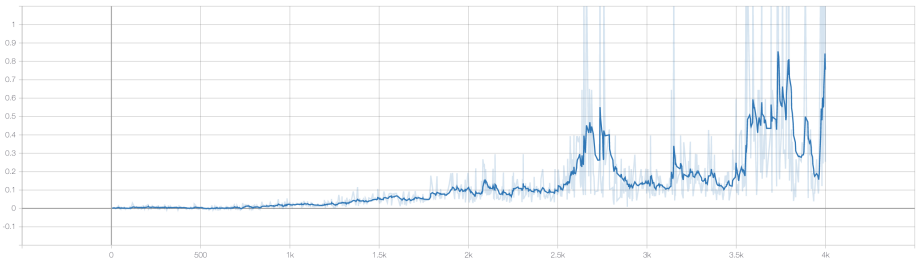

Figure 2: Rewards obtained by the second agent, along with a smoothed out line (taken from TensorBoard)

only one really needs to be active at the same time, but through applying the appropriate symmetry, the experience could be shared between the two agents. Then, the agents could be updated to the better one between the two of them every few iterations, to ensure one doesn't lag behind the other.