Deep Reinforcement Learning - Project 2

Peter Toke Heden Ahlgren[1]
[1]Mjølner Power ApS

Keywords: Continuous Control, DDPG 10th March, 2021

1 Introduction

In the following report we solve the Unity Environment "Reacher" with reinforcement learning agents. The environment consists of either 1 or 20 agents, maneuvering an arm, the end of which is supposed to follow a moving target. For every timestep where the end of the arm is inside the target, a reward of 0.1 is given. For the current project an average score of 30 over 100 episodes of the game is considered a success for the single agent in the 1 agent environment and on average over all agents in the 20 agent environment. The agent gets 33 different inputs as its state and has 4 continuous action variables all ranging between -1 and 1.

2 Reinforcement Learning Agent

As agent we use a deep reinforcement learner, *i.e.*, a reinforcement learner agent where the Q-table is replaced by a neural network. For this structure to successfully learn, a few tricks has to be implemented: instead of only one network replacing the Q-table we use a *local* and *target* network, such that it is not the same network that for every step is updated and take an action - we also utilize a replay buffer, giving the agent memory, such that also old experiences can still be used. The structure with both *local* and *target* networks are used for both an *actor* and a *critic*, working together in order to solve the environment. The particular implementation chosen for this project is a *Deep Deterministic Policy Gradient* (DDPG) in which the actor learns a policy and the critic learns a Q-value function. DDPG can be thought of as deep Q-learning for continuous actions space. As such, DDPG is an off-policy algorithm. While the actor seeks to find the best policy for the given observation space, the critic tries to estimate the Q-value of the actions taken by the actors, given the current observations.

For the environment containing 20 agents, we construct a DDPG agent with 20 actors, common replay buffer and one common critic. This is heavily inspired by the OpenAI multi agent DDPG, MADDPG*, which shares these traits but also has further details to it. We use a common critic, based on the idea that there must be an advantage of estimating the Q-function based on all data points. The idea behind having as many actors as agents is that it might give rise to slightly different exploratoty paths. Also, just using the same actor for all 20 agents, seems like cheating by only counting every 20th episode as an episode in terms of how fast the model trains.

3 Parameters

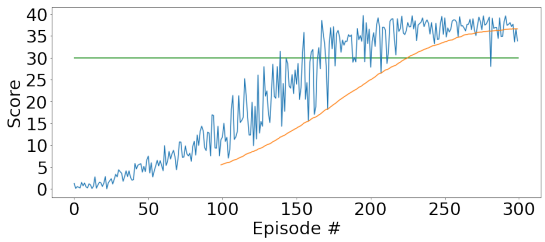
For the basic structure, we choose a network having 2 hidden layers for the actor and 3 for the critic. The actor has with 128 nodes in each layer, while the critic has 128 nodes in all layers but the second; here the number of nodes is 128+the size of the action space. Activation functions are ReLU between all layers. The actor(s) ends with a $\tanh$ activation function, mapping the output to the $[-1, 1]$ interval. The critic ends in a fully connected layer.
For all examples we use the same values of the following variables:

- Replay buffer size: $1e6$
- Mini-batch size: 128
- Discount factor, γ: 0.9
- Parameter for soft updating of target, τ: $1e{-3}$
- Learning rate, actor: $5e{-4}$
- Learning rate, critic: $1e{-3}$

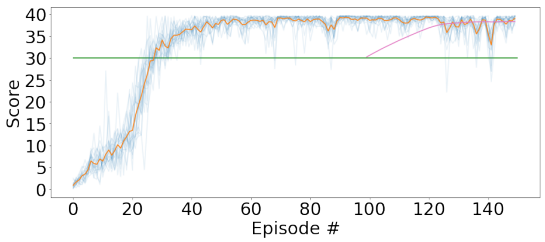
4 Results

The resulting rewards for the environments are depicted on figure 1.

*Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments, by Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, Igor Mordatch, https://arxiv.org/abs/1706.02275



(a) Single agent environment performance (blue), its running mean (orange) and the threshold for succes (30).



(b) 20 agent environment performance for individual agents (blue), their step wise average (orange), their running mean (pink) and the succes threshold (green).

Fig. 1. The reward dynamics for the examined environments. Notice that the x-axis on the right plot is half the length of the x-axis on the left. The 20 actors sharing a common critic learns rather quickly and the average running over 100 episodes begins at a level just above the success threshold.

For the 1 agent environment, figure 1a, it takes around 225 episodes before the running average crosses the accepted level of 30. We let it run a bit further and accept the agent at step 300 as the final solution. Along the training, there is a somewhat high variability of the rewards compared to the benchmark solution. Quite some work has been done to change this for the better, but no results better than the here presented have been found.

For the 20 agent environment, figure 1b, learning seems to start almost immediately. The individual agents does not see more data per episode than the agent in the environment with only 1 agent. Hence it must be the common critic who is responsible for the way faster learning. The episode average already exceeds the accepted level around episode 30 and since the running average begins at a value just above the acceptance threshold, we accept the agent at step 100 as the final solution for this setup.

General findings for both environments are that the learning rate for the critic should be higher than the actor(s), the discount factor, γ, should be relatively small and the training is relatively sensitive to the mini-batch size.

5 Future Work

First, it would be interesting to investigate what it takes to lower the variability towards convergence for the 1 agent environment. Since many tweeks on input parameters have been tried it is hard to come up with concrete ideas of what could do the trick, but playing more with τ and γ or looking into the update frequency of actor and critic networks could be suggestions.

For the 20 agent environment it would be interesting to utilize the 20 different actors in a better way. For example, it would be great to see if copying weights from the most succesfull actor to the rest every N'th episode would make the model even better. Also, it would be great to experiment with other agent types than the DDPG and learn the differences in training characteristics. For example using PPO or D4PG as suggested in the problem formulation of this project.