Reach Continuous Control Project Report

Patrick Sowinski

This project implements a Deep Deterministic Policy Gradient agent (DDPG) to solve the Reacher robot arm environment from Unity ML Agents.
The project is part of the Udacity Nanodegree program on Deep Reinforcement Learning.

The Environment

The environment is implemented within Unity ML Agents. More info about these environments can be found here: https://github.com/Unity-Technologies/ml-agents

In this environment, a double-jointed arm can move to target locations. A reward of +0.1 is provided for each step that the agent's hand is in the goal location. Thus, the goal of the agent is to maintain its position at the target location for as many time steps as possible.

The observation space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm. Each action is a vector with four numbers, corresponding to torque applicable to two joints. Every entry in the action vector should be a number between -1 and 1.

To accelerate training for the agent, we use a multi-agent version of the environment that renders 20 robot arms, each with a different target, simultaneously. The scores of the different arms are averaged.

The environment is considered solved, when the average (over 100 episodes) of those average scores is at least +30.

The Solution

The environment was solved with a Deep Deterministic Policy Gradient (DDPG) agent.
The agent has an “act” function, which determines the next action to take given the environment’s state. It also has a “step” function, which adds experiences to the replay buffer. This replay buffer is used for learning, which includes updating the network weights.
The agent uses an actor and a critic network, each of which has two versions, a local and a target network with the same architecture. So in total there are 4 networks. The target network weights are updated after every learning step with a soft update from the corresponding local network.

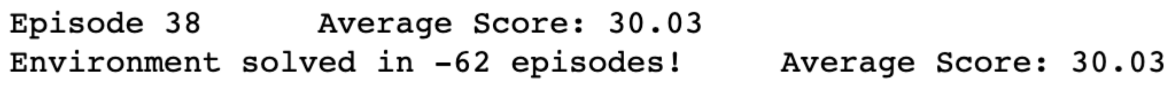
The actor networks each consist of 4 fully connected layers with ReLU activations between them (input and output layers + 2 hidden layers). The final layer output is fed into a tanh activation function to match the defined action value intervals from -1 to +1.
The input size corresponds to the size of the state space, given by the environment, which in this case has 33 dimensions. The output size of the networks corresponds to the number of dimensions in the action space, which is 4 here.
The hidden sizes of the layers are 256 and 128.

The critic networks each consist of 4 fully connected layers with ReLU activations between them (input and output layers + 2 hidden layers). The critic networks receive the environment’s state and the selected actions as inputs, which are fed into the network at different points. The state is fed into the first input layer directly. The actions are concatenated with the outputs of the first hidden layer (after ReLU activation).
The input size corresponds to the size of the state space, given by the environment, which in this case has 33 dimensions. The output size of the network is 1, because it is only meant to approximate a scalar value. There is no activation function for the output layer.
The hidden sizes of the layers are 256 and 128. We need to remember that the 256 output values of the first hidden layer are concatenated with the given action values to form inputs for the second hidden layer with size 128.

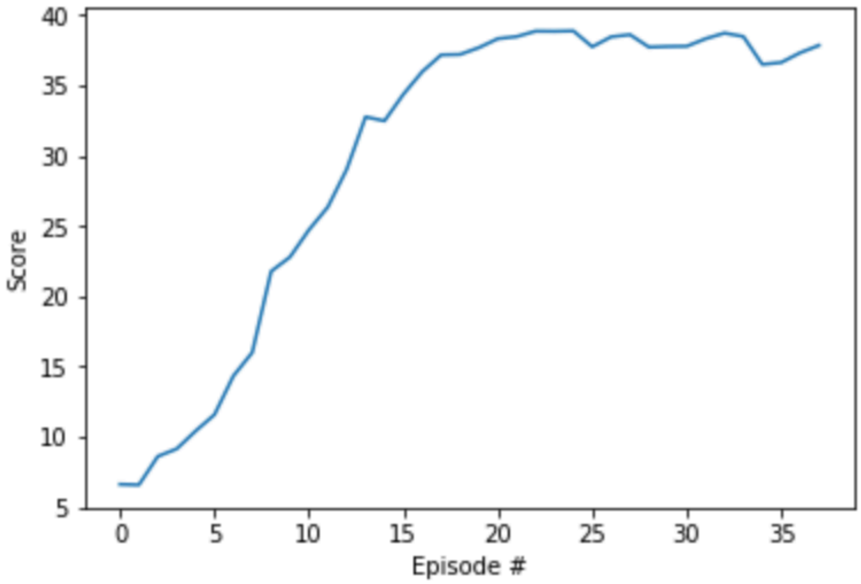
The replay buffer has a size of 10,000, so it can store a variety of different experiences.
The batch size for a learning step is 32, which ensures a training curve without too much noise.
The factor gamma used for discounting rewards is set to 0.99, which is probably irrelevant in this case. 1.0 should work as well.
The soft update factor tau is set to 0.001, which means it takes roughly 1000 steps to completely overwrite the target network weights.
The learning rate for the actor network is set to 0.0001 and for the critic network it is 0.0003. We update the critic “more strongly”, since we want to have a good feeling for the value of some actions before changing the actions we are making.

Results

Below is the plot of rewards per episode during training. The environment was solved as soon as the agent reached an average reward of +30 over the previous 100 episodes.
The score per episode has consistently exceeded 30 after just 12 episodes. Our training loop aborted the training after just 37 episodes, since the average over all existing episodes was already over 30 by then.



```
Episode 38	Average Score: 30.03
Environment solved in -62 episodes!	Average Score: 30.03
```



Ideas for future work

Another improvement would be to use prioritized experience replay, so we learn more often from uncommon experiences that provide a lot of information.