Project 2 - Continuous Control from the Deep Reinforcement Learning Nanodegree of Udacity

October 21, 2019

1 Introduction

This is the report for Project 2 - Continuous control from the Deep Reinforcement Learning Nanodegree of Udacity. The task is to train an agent to maintain the position of a double-jointed arm at a moving target location for as many time steps as possible. The task is continuous, although the episodes end after a certain time.

A reward of 0.1 is provided for each step that the agent's hand is in the goal location. In order to solve the environment, the agent must pick an average score of 30+ over 100 consecutive episodes. 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 must be a number between -1 and 1.

2 Learning Algorithm

At a high level the algorithm follows the usual approach of Reinforcement Learning, in which the agent interacts with the environment by choosing an action at every time step, where the environment is in a state; the environment in turn responds by providing a reward to the agent for this, and moving to the next state. The learning proceeds through a loss function, a reward function in this case, which the agent seeks to maximize by tweaking its internal parameters (weights), with the various techniques that we have seen in the course. The high level algorithm implemented in the notebook contains little more than a loop to implement this learning process.

More insteresting is the learning process itself, the internal configuration of the agent provided (*ddpg_agent.py*) and the structure of the neural networks (*model.py*). The first file implement the so-called Deep Deterministic Policy Gradient (DDPG) algorithm. This is an actor-critic architecture in which an actor network is used to approximate the policy function, and a critic is used for evaluating the policy function by using temporal differences (TD). This implementation of DDPG also has a replay buffer, and a noisy process (Ornstein-Uhlenbeck) is used to aid with the exploration of the observation space.

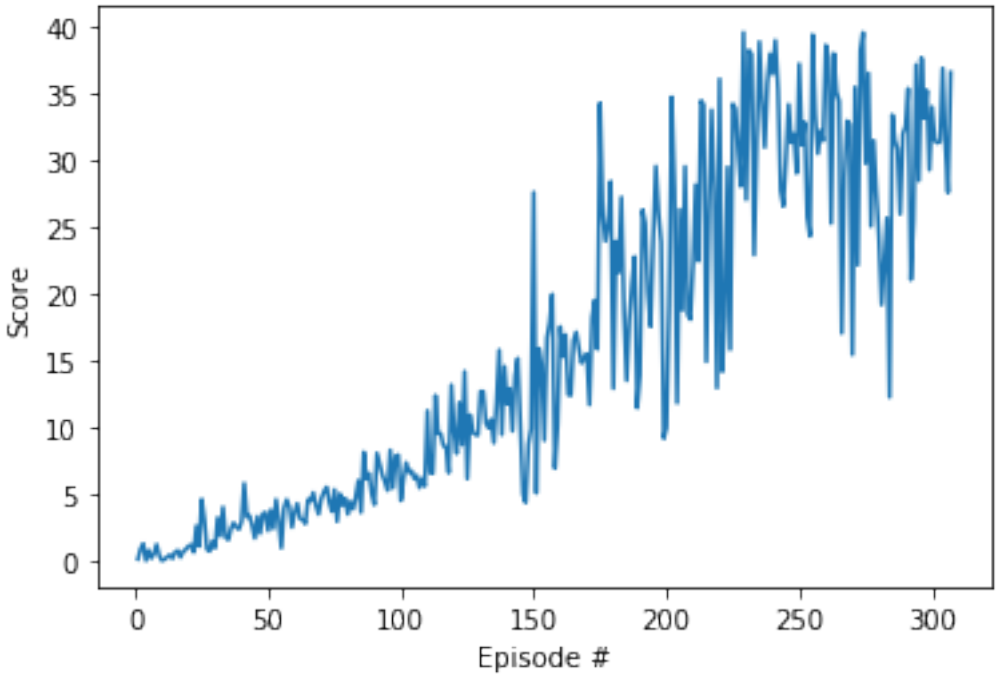
As mentioned, the underlying models for actor and critic have neural network architecture, in our case:

- an input layer
- a fully connected hidden layer with 128 units
- a relu activation function

- batch normalization
- a fully connected hidden layer with 128 units
- a relu activation function
- an output layer

Two of these networks (target and local) are used for both, actor and critic, with the updating of parameters proceeding via the soft update mechanism.

With this configuration the learning rate was already quite good, as can be seen in Figure 1.



although I have to admit that it took a fair bit of arbitrary tweaking of hyperparameters, for example I started with larger network sizes, learning rates, and shorter episodes. The full set of parameters for this final run is the one currently found in the files.

3 Future

Since the learnign rate is already quite good, and it has already been quite difficult to get to this point, I am happy enough and have not experimented more with the configuration. A possibility would be to use distributed learning and train several agents at the same time, as proposed in the exercise. I doubt that my computer would take it though, training this agent was already quite slow (about 3 hours).