Report for Project 2: Continuous Control

Udacity Deep Reinforcement Learning Nanodegree

April 14, 2020

1 The Implementation

The main part of the code is written in the Jupyter Notebook `Continuous_Control.ipynb`, which guides the user when executing the code. The intelligent agent has been implemented in the `agent.py` file, it uses two neural networks, for its actor and its critic, as defined in `model.py`.

1.1 The Neural Networks

The networks as defined in `model.py` by default have two hidden fully-connected layers with 64 neurons each. The goal of the actor is to provide the best action given a state s: $a = \mu(s; \theta_\mu)$. The critic's goal is to estimate Q for the best-believed action, i.e. $Q(s, \mu(s; \theta_\mu); \theta_Q)$. For the critic, the actions are fed into the network as inputs to the second hidden layer, while states are directly input to the first layer.

1.2 The Agent Class

The agent class is capable of learning according to the Deep Deterministic Policy Gradient (DDPG) algorithm as discussed in class and in this paper.

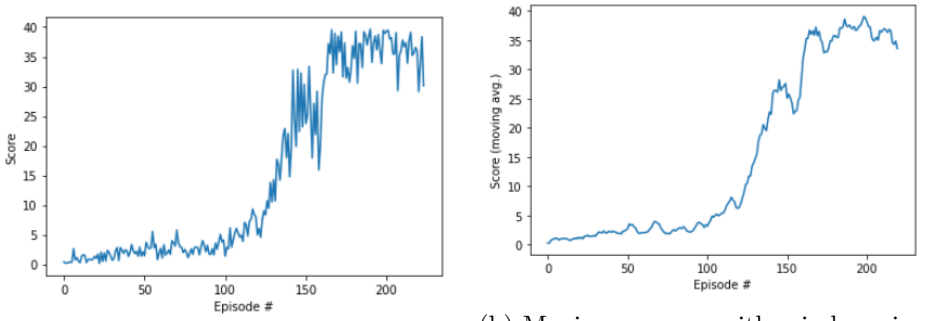
Within its `__init__` method, several internal variables are defined: state size, action size, and the `ReplayBuffer` for experience replay. Besides, the network weights (local w and target w^-) and optimizer to be used (Adam) are defined for the agent's actor and critic network.

The `step` method adds the current experience to the replay buffer and triggers a learning step. To do this, it triggers the `learn` method, which backpropagates the errors made from sampled experiences through the network to update weights, once for the critic, and once for the actor. The actual (soft) update is done in `soft_update`. Hyperparameters used are mentioned in Table 1.

The `act` method is used to return the action suggested by the actor for the current state. This is a maximized, thus deterministic choice, not a stochastic policy.

Replay Buffer	Value	Reward discounting	Value	Weight updates	Value
Buffer Size	10k	γ	0.9	τ	0.002
Batch Size	256				

Table 1: The hyperparameters used for the learning algorithm.



(a) Original.

(b) Moving average with window size 5.

Figure 1: The scores reached by the intelligent agent.

The `save` and `load_weigths` functions are used to save trained agents for further use and to load previously-trained agents, respectively. They are quite straightforward, only some checks for stringent dimensions of the networks are done in the `load` method.

The `ReplayBuffer` class is taken from the lessons and comprises an initialization method as well as methods for adding experiences, sampling from the stored experiences, and a function for returning the current length of the buffer.

2 Results

As Figure 1 shows, the agent is capable of learning, the environment has been solved within **124 episodes**.

3 Ideas for Improvements

One of the main challenges after having a working code basis was to find suitable hyperparameters. This involved extensive searching on the web plus lengthy simulation runs. Other approaches like Proximal Policy Optimization (PPO) may achieve results of similar quality while having less need for parameter tuning.

Potentially, learning itself could also be speeded up by reducing the model size.