Continuous Control Project Report

Udacity Reinforcement Learning Nanodegree Program

Author: Guilherme Sales Santa Cruz

11 may 2020

Summary

This document describes the navigation project. It is divided in 4 sections:

Problem description

The problem consists of creating an **Policy-Learning** agent capable of controlling a robotic arm to reach a moving target on a 3-dimensional space.

The agent is reward with +0.1 for each step that the agent's hand is in the target location. The goal of the agent is to achieve an average of 30+ by episode over a 100 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 should be a number between -1 and 1.

Algorithm description

The algorithm used to solve this problem is called **DDPG (Deep Deterministic Policy Gradient)**. It is an off-policy method that consists of an “actor-critic” agent that receive continuous informations as input and take continuous decisions based on it.

Components

It has 4 different components.

The first is a ***memory buffer***. It has a fixed size and saves the experiences of the agent along the steps. One *experience* consists of: the state before the action, the action taken, the reward for that action, the new state and the information on weather the episode is over or not (done). Once it is full, it discards its oldest experiences to add new ones.

The second is a ***noise generator***, which was implemented using the Ornstein-Uhlenbeck process, a stationary Gauss-Markov process, originally created to model the velocity of a massive Brownian particle.

The other 2 components are the ***Actor*** *and the* ***Critic***. These agents, break the task of creating a policy for an environment into evaluating the current state (*Critic*) and choosing an action that will maximize the value for the next state (*Actor*) instead of picking an action that will maximize your final score.

Each of these agents have 2 internal main components a ***local network*** *and a* ***target network***. Similarly to the DQN algorithm, two networks are used to reduce the chances that the agent overestimates the parameters while learning.

Methods

The main agent has 2 public methods.

The first is ***act***. This method receives as input one state and outputs an action for this state.

The second method is ***step***. This method saves an experience and from time to time triggers the learning routine. This method receives as input one *experience* (see the previous section for definition) and stores it in the internal *memory buffer*. At every N-*steps* the learning routine is triggered. It takes a random sample of *experiences* from the *memory buffer* and trains the sub-agents.

To train the *Critic* it uses the sequence of states given by the actions taken by the *Actor* and train on the discounted value of the future state value. To train the *Actor* it uses the *Critic*'s state value estimation difference.

Hyperparameters

Hyperparameters are parameters that should be tuned to improve the performance of the model. The following image shows the hyperparameters used to build the final agent.

```
BUFFER_SIZE = int(1e6)  # replay buffer size
BATCH_SIZE = 128        # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR_ACTOR = 1e-3         # learning rate of the actor
LR_CRITIC = 1e-3        # learning rate of the critic
WEIGHT_DECAY = 1e-8     # L2 weight decay

ACTOR_FC1_UNITS = 200
ACTOR_FC2_UNITS = 150
CRITIC_FC1_UNITS = 400
CRITIC_FC2_UNITS = 300

n_agents = num_agents
update_every = n_agents
```

- *BUFFER_SIZE* : Configures the size of the internal *memory buffer.*
- *BATCH_SIZE* : Number of experiences randomly sampled from the *memory buffer*.
- *GAMMA* : Gives the discounted value of future rewards.
- *TAU* : Configures the update of the second network by interpolation.
- *LR_ACTOR* : Configures the learning rate of the actor network optimizer (Adam).
- *LR_CRITIC* : Configures the learning rate of the critic network optimizer (Adam).
- ACTOR_FC1_UNITS : Gives the number of neurons at the *Actor*'s 1st hidden layer.
- ACTOR_FC2_UNITS : Gives the number of neurons at the *Actor*'s 2nd hidden layer.
- CRITIC_FC1_UNITS : Gives the number of neurons at the *Critic*'s 1st hidden layer.
- CRITIC_FC2_UNITS : Gives the number of neurons at the *Critic*'s 2nd hidden layer.
- *update_every* : Describes how often to update the network.

Network architecture

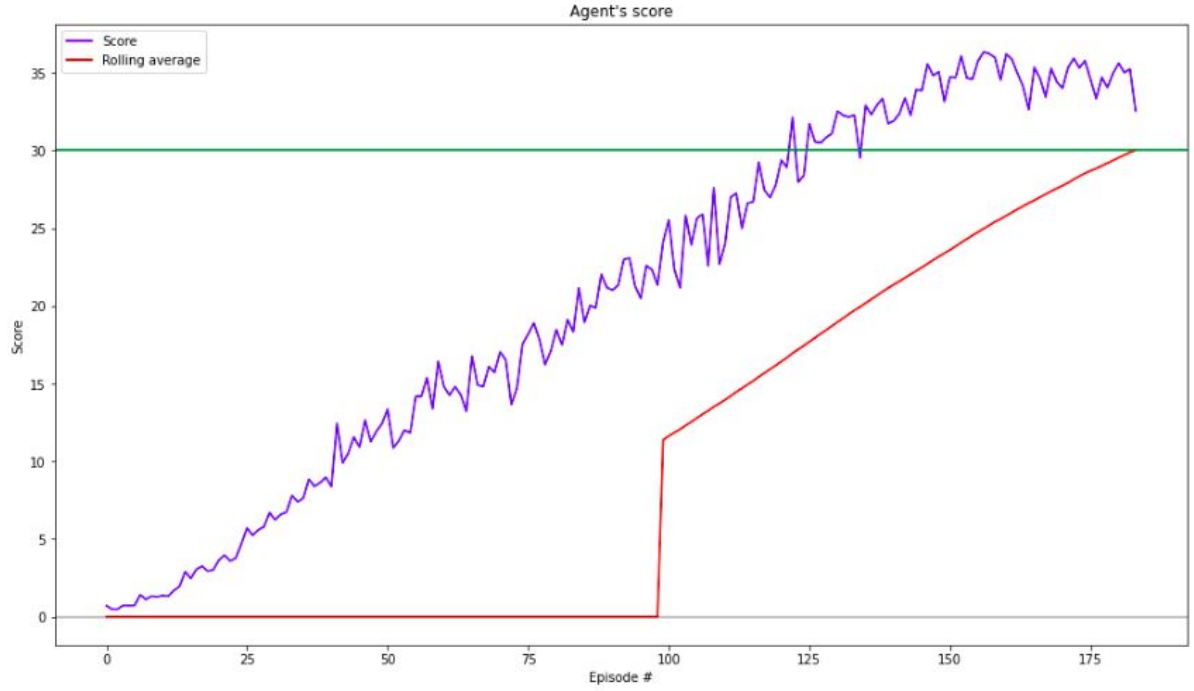
This algorithm has two sets of networks, one for the *Actor* and one for the *Critic*.

The *Actor*'s network is composed of 2 fully connected hidden layers 200 and 150 neurons respectively. The last layer has a hyperbolic tangent activation for an output limited between 1 and -1 and the previous ones have ReLU (rectified linear unit) activation. The input layer has one neuron for each state dimension. The output layer has 4 neurons, one for each torque direction of the arm.

The *Critic*'s network is slightly more complex. It is composed of one input layer, one fully connected hidden layer, one hybrid hidden layer and one output layer respectively. The input layer has one neuron for each state dimension; the hidden layer has 400 neurons and has a ReLU activation; the hybrid layer concatenates an action vector with the output from the previous layer (204 neurons) and applies ReLU activation; the output layer has 1 neuron and has a linear activation.

Results

The agent was able to solve the problem in 184 episodes. The image below plots the score and the rolling average over the number of episodes.



The rolling average grew monotonically and steadily. The variation of the scores increased on the middle but seemed more stable in the end.

Future work

There are multiple options for potentially improving the agent such as:

- Fine tune the hyperparameters;
- Use prioritized memory replay;
- Use Generalized Advantage Estimation (GAE).

Additionally, as a future work, a comparison between this algorithm and a on-policy algorithm for this task would be interesting.