# Robotic Arm Continuous Control Project

Tiago Montalvão

June 23, 2020

## 1 Introduction

This is a report of the second project of Udacity Deep Reinforcement Learning Nanodegree. It describes the model architecture with chosen hyperparameters used in the agent, the experiments, and potential improvements upon the current results.

The environment chosen in this project is the second one provided, in which there are 20 multiple arms interacting with the world simultaneously.

## 2 Implementation

### 2.1 Agent

The agent was implemented using Deep Deterministic Policy Gradient (DDPG). This algorithm is basically an enhancement of Deep Q-Networks (DQN) for dealing not only with a continuous state space but also with a continuous action space.

It is categorized as an Actor-Critic method, because there are two networks (Actor and Critic) collaborating to achieve the agents' results. Basically, the actor learns to pick actions from states, while the critic learns to calculate the state-action Q value function.

The DDPG was then implemented with:

- **Experience replay buffer** to handle data correlation between consecutive samples.
- **Two neural networks** with identical architecture (local and target networks) for each actor and critic, as described in the following section (so there were 4 networks in total). **Soft update** was also implemented to updated each target network, instead of updating the whole network every n-th iteration.
- **Ornstein–Uhlenbeck process** implemented to add noise to the actions, thus creating incentives to exploration.

It is worth mentioning that the implementation was heavily based on previously implemented DDPG on the Nanodegree, but a few adjustments were made in the hyperparameters and in the code itself. The adjustments were:

- Ornstein–Uhlenbeck process fixed to sample from Standard Normal distribution instead of [0, 1) uniform distribution.
- Replay buffer support for adding multiple samples (one for each agente) at once.
- Gradient clipping in the critic network update.
- Instead of sampling from the replay buffer and learning from these tuples every iteration, a counter was implemented to only update the network each UPDATE_EVERY iterations.

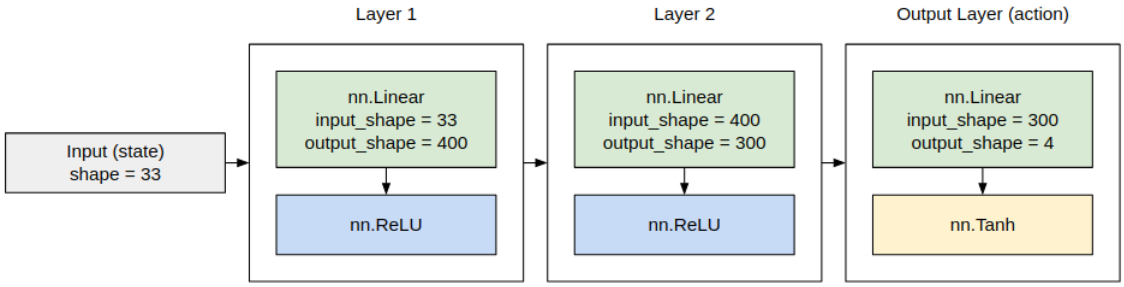
- Each UPDATE_EVERY iterations, tuples were sampled and the networks were updated a few times in a row (N_UPDATES_PER_STEP hyperparameter).
- The batch size of samples from the replay buffer was significantly increased from 128 to 1024. Before that, the learning was quite slow (environment solved in more than 400 episodes).

These adjustments gave a lot of stability for the learning process.
The chosen hyperparametes are described above:

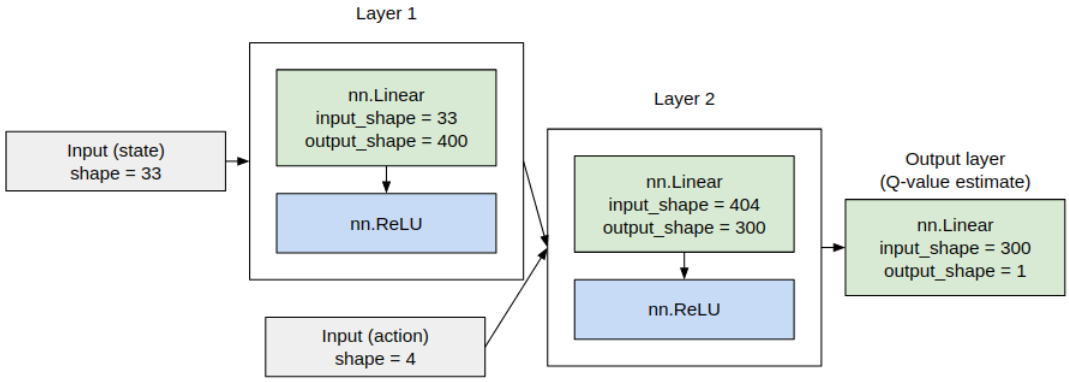
```
BUFFER_SIZE = int(1e6)     # replay buffer size
BATCH_SIZE = 1024          # minibatch size
GAMMA = 0.975              # discount factor
TAU = 1e-3                 # for soft update of target parameters
LR_ACTOR = 5e-4            # learning rate of the actor
LR_CRITIC = 1e-3           # learning rate of the critic
WEIGHT_DECAY = 0           # L2 weight decay
UPDATE_EVERY = 15          # how often to update the network
N_UPDATES_PER_STEP = 10    # number of updates to perform at each learning step
```

## 2.2 Model architecture

The architecture for the actor network is as follows:




The architecture for the critic network is as follows:



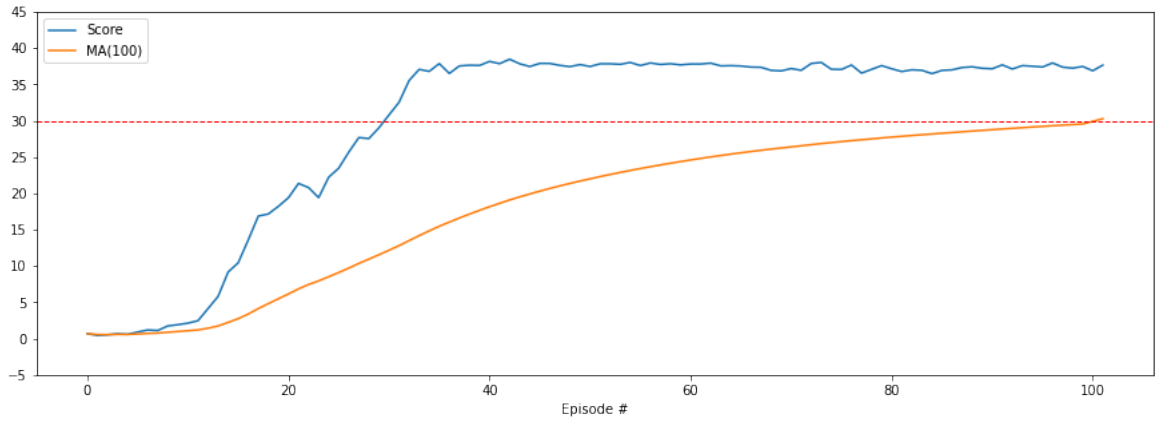

In the project notebook, there is a cell which executes a command that gives the model summary (the same info above with some numbers). Based on this, we can get the information that there

are 135104 trainable parameters in the actor network and 135401 trainable parameters in the critic network.

# 3 Results

The agent was able to solve the environment in only 102 episodes, and it took it about 70 minutes to train. All the details can be checked in the project notebook. It can be seen that the agent could learn quite well in early episodes and in a very stable way.

The following image shows the individual episode score (in blue), the moving average of the last 100 episodes' scores (in orange), and the threshold of 30 for the moving average (in red), so that the environment is considered solved.




# 4 Ideas for Future Work

DDPG is one of the most classic actor-critic algorithms. Other algorithms (like PPO, TRPO, A3C, D4PG) could be implemented for practicing purposes because the results obtained were satisfactory (maybe other algorithms could speed up training process).

Another improvement could be the addition of **Prioritized Experience Replay**, giving more weight in experience sampling from replay buffer for tuples that have larger TD errors.