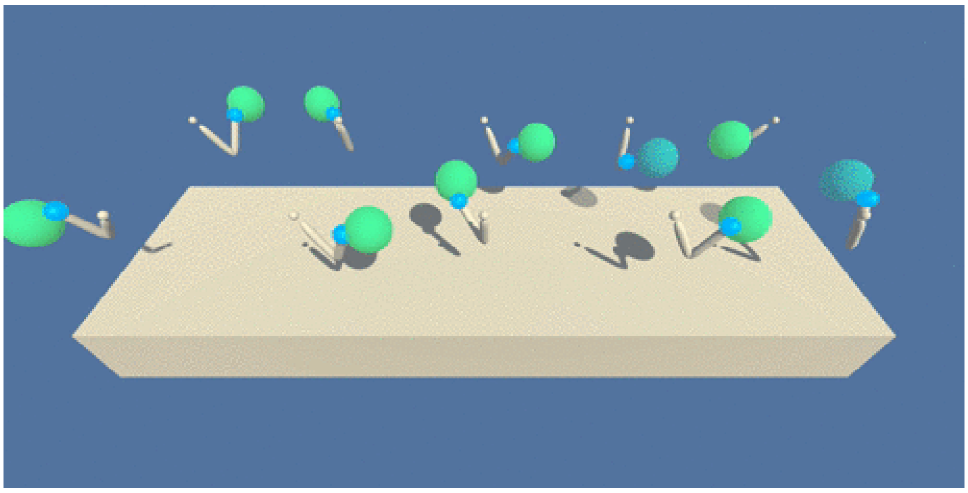
Project 2 - Continuous Control in the Reacher Environment



By José Ilberto Fonceca Junior

Abstract

This document summarizes the procedures to train a deep reinforcement learning agent with Deep Deterministic Policy Gradients algorithm to solve the reacher environment brought by Udacity.

Introduction

This project aims to train an armed agent capable of follow an object that moves in the Reacher environment and getting as many rewards as possible. In this environment, the agent only collects a 0.1 positive reward when it touches the object, otherwise it receives no reward.

The agent is a double-jointed arm that can translate and rotate in the space. The state vector in each time has 33 elements representing positions, rotations, velocities and angular velocities. In each time step, the agent should take an action that has 4 inputs which apply torques to the joints and change the state of the arm in the next time step.

Here, we present the solution for 2 different versions of the same environment: one with a single arm trying to reach the object and another one with 20 arms each one attempting to reach the nearest object.

Environment

States

We are interested in training an agent able to collect the highest rewards in the reacher environment where very little information is given to the agent. In this scenario, the agent is a double-jointed arm that is aware of the position, rotation, velocity and angular velocity of each of its joints in every time step.

Actions

So in this world, the arm applies torques to the joints in two different directions during each time step. These torques have values between -1 and 1.

Installation and requirements

Plese, visit the Deep Reinforcement Learning repository maintained by Udacity in order to install all dependencies to work with the code used here. All the steps presented there can be break down into the steps in file requirements.txt:

```
youruse@yourcomputer:~$ conda env create -f environment.yml
youruse@yourcomputer:~$ python -m ipykernel install --user --name drlnd --display-
name "drlnd"
```

Finally we should download and extract the reacher environment in the same folder of your project. Please, refer to the list below to download it:

Version with one agent:

- Linux: link
- Mac OSX: link
- Windows (32-bit): link
- Windows (64-bit): link

Versionn with multiple agents:

- Linux: link
- Mac OSX: link
- Windows (32-bit): link
- Windows (64-bit): link

Algorithm

The agent learns how to collect higher rewards in the environment using modifier version of the Deep Deterministic Policy Gradients (DDPG) algorithm presented in the paper Continuous Control with deep reinforcement learning. The agent has 4 densely connected neural networks:

- the local actor and critic: it improves at every accumulated time step (k, since we are not improving the networks at every time step, but we are waiting k time steps to do so). By calculating the action-value function of N_u batches of size (b) using both the local and the target networks from actor and critic, we obtain 2 different action-value functions that are then used to update the local critic and then the local actor.

- the target actor and critic: comparing it with the local one every N batches of size k, we update it using a separate weight (tau) called the soft update rate of the neural network.

These improvements over the DDPG algorithm with the use of N_u updates every k time steps allow us to learn at small steps and from the past experiences multiple times. This small improvement has shown to improve the results of DDPG specially in the scenario with 20 arms.

- The specifications of the neural networks of the actor

Layer	Dimensions single arm	Dimensions multiple arms	Type	Activation	Information
Input	33	33			tensor with dimensions (batch_size, 33)
Hidden	(400, 300)	(256, 128)	Linear	ReLU	
Batch_norm	400	256	Linear	--	Batch Normalization layer
Hidden	(300, 4)	(256, 128)	Linear	ReLU	
Output	4	4	Tanh		tensor with dimensions (batch, 4)

- The specifications of the neural networks of the critic

Layer	Dimensions single arm	Dimensions multiple arms	Type	Activation	Information
Input	33	33			tensor with dimensions (batch_size, 33)
Hidden	(33, 400)	(33, 256)	Linear	ReLU	
Batch_norm	400	256	Linear	--	Batch Normalization layer
Concatenation	4	4	Linear		Concatenation from Batch_norm layer with the action vector
Hidden	(404, 300)	(260, 128)	Linear	ReLU	
Output	1	1	Linear		tensor with dimensions (batch, 1)

- The structure of the single arm environment agent

Parameter Name	Value	Description
Replay initial size	0	Replay buffer initial size
Replay size	10000	Replay buffer maximum size
Batch size (b)	128	Batch size for each update
Gamma	0.99	Discount rate
Tau	0.001	Soft update rate
Time steps until update (k)	1	Number of time steps until the actor and critic networks are updated

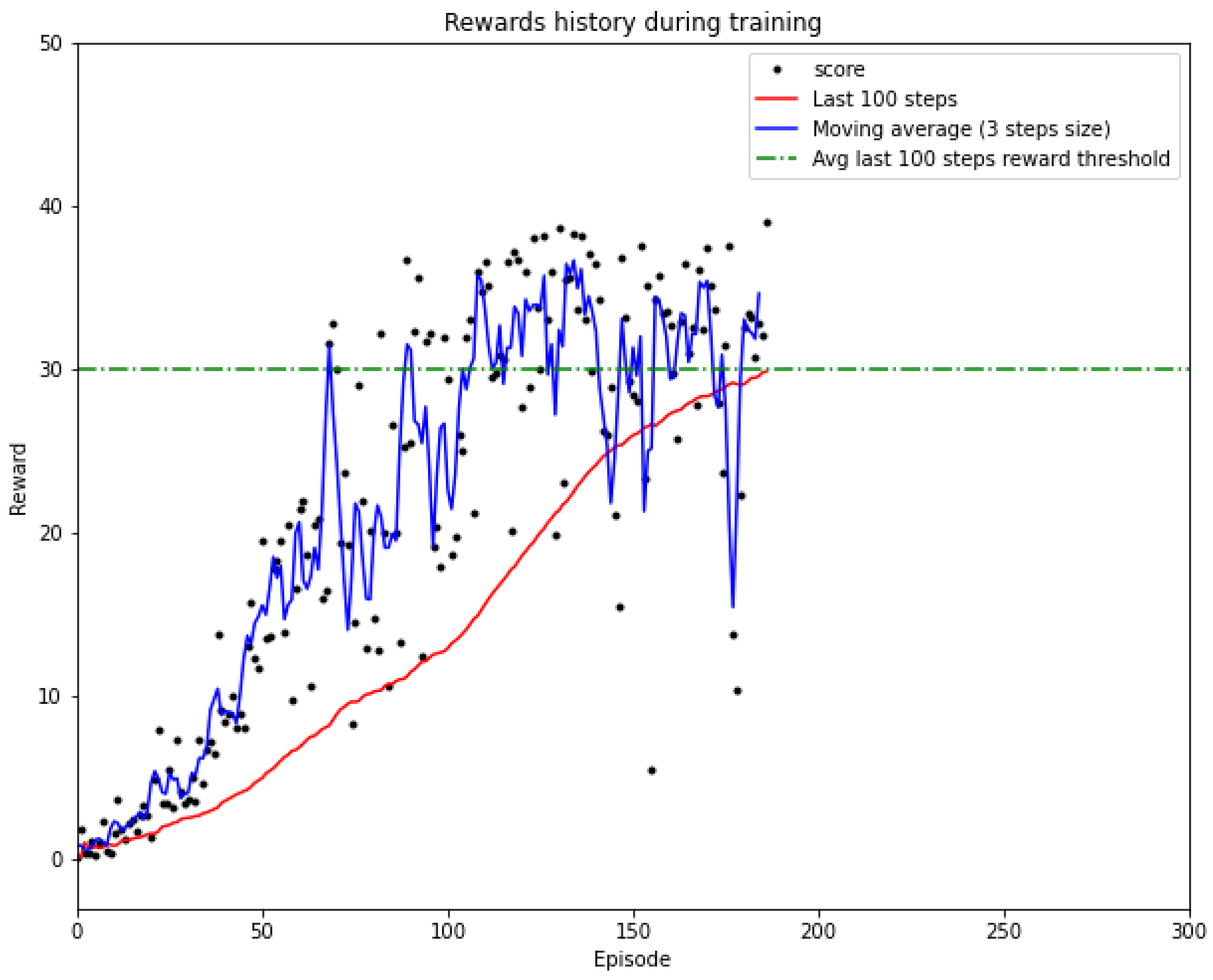
Parameter Name	Value	Description
Number of updates (N_u)	1	Number of times the neural networks will be updated at once
Weight decay (Omega)	0	Weight decay in the critic's updates
Learning rate actor	0.0002	The learning rate of the actor local neural network
Learning rate critic	0.0002	The learning rate of the critic local neural network
Reward threshold	30	The number of rewards obtained during the last 100 episodes for truncating the training

- The structure of the 20 arms environment agents

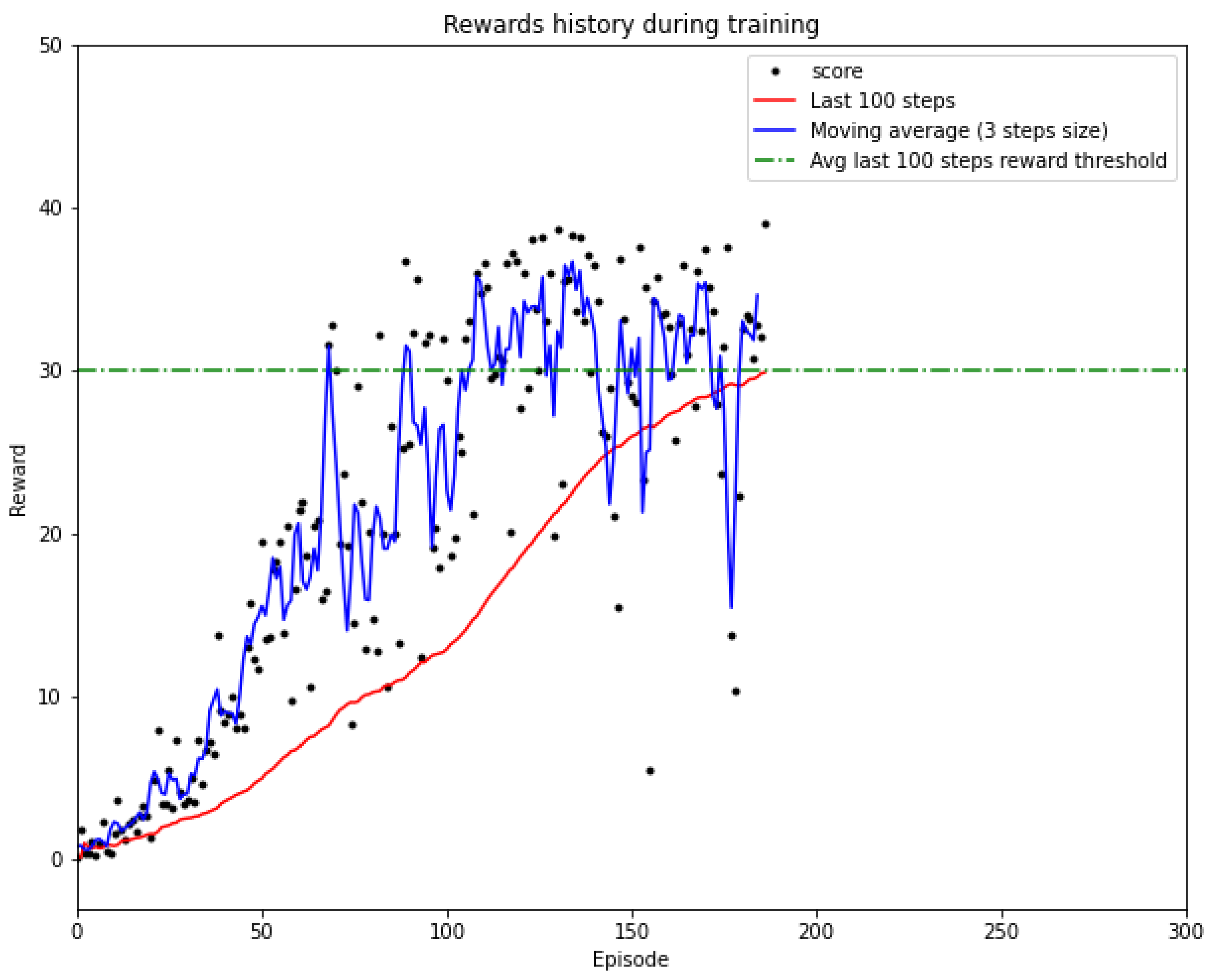
Parameter Name	Value	Description
Replay initial size	0	Replay buffer initial size
Replay size	10000	Replay buffer maximum size
Batch size (b)	128	Batch size for each update
Gamma	0.99	Discount rate
Tau	0.001	Soft update rate
Time steps until update (k)	20	Number of time steps until the actor and critic networks are updated
Number of updates (N_u)	10	Number of times the neural networks will be updated at once
Weight decay (Omega)	0	Weight decay in the critic's updates
Learning rate actor	0.0002	The learning rate of the actor local neural network
Learning rate critic	0.0002	The learning rate of the critic local neural network
Reward threshold	30	The number of rewards obtained during the last 100 episodes for truncating the training

Results

Using this set of parameters, our agent was able to follow the object in the environment collecting the most rewards it could, needing less than 300 episodes in both environments to obtain an average reward of 30 over the last 100 episodes. These results are shown in the figure below for the single arm environment.



And below we have the results for the 20 arms environments.



Conclusions and future work

The agent was able to generalize the environment enough to obtain the most rewards in the given maximum time steps. Nevertheless, we do not know how it would perform in a scenario where there is no limitation in the training of the agent and if it would ever be able keep its performance in a large time step scenario.

We should also consider that it is a continuous control task and the agent has a very specific selection of parameters that works with the environments provided. It implies that slight variations in the set of parameters lead to nonconverging scenarios.

The natural prospects of the present work is to explore environments longer and implement algorithms such as Distributed Distributional Deterministic Policy Gradients (D4PG) and Proximal Policy Optimization (PPO) which have been shown to perform better with continuous tasks including the Reacher environment.