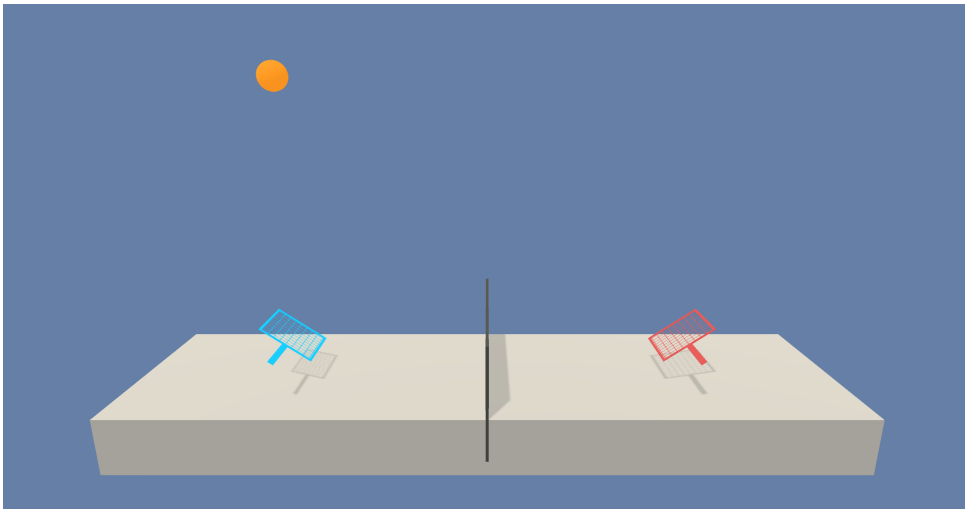
Project 2 - Continuous Control in the Reacher Environment



By José Ilberto Fonceca Junior

Abstract

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

Introduction

This project aims to train two tennis players agents at the same time capable of bounce the ball back and forth to each other using the Unity Tennis environment and getting as many rewards as possible. In this environment, the agent only collects a 0.1 positive reward when it hits the ball and it crosses the net otherwise it receives a -0.01 reward.

The agents are both rackets that can move and jump to hit the ball. The state vector in each time for each agent has 8 elements representing:

- positions of the racket and the ball in the xy plane respectively;
- velocities of the racket and the ball in the xy plane respectively.

In each time step, the agent must take an action that has 2 inputs: the first one make the racket move forward or backward and the another one makes the racket jump. Here, we present the solution the environment using Multi-agent Deep Deterministic Policy Gradients (MADDPG) algorithm.

Environment

States

We are interested in training the agents to be able to collect the highest rewards in the tennis environment where very little information is given to the agent. In this scenario, the agents are rackets that are aware of their positions and velocities as well as the same information regarding the ball for each time step.

Actions

So in this world, the agents can move forward and backwards and jump to get in touch with the ball in a higher position.

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:

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

Algorithm

The agents learn how to collect higher rewards in the environment using a modified version Multi-Agent Deep Deterministic Policy Gradients (MADDPG) algorithm presented in the paper Multi-Agent Actor Critic for Mixed Cooperative-Competitive Environments. Each 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.

Also, they store their experiences in a shared Replay Buffer, this shared memory easens the necessary communication between the agents to coordinate on keeping the ball in game as long as possible and achieve higher rewards. This added feature allowed us to achieve a set of agents able to play the game for rewards higher than 2.5 in the last steps (it represents that a single agent was able to hit the ball 25 times in a single match).

- The specifications of the neural networks of the actor

Layer	Dimensions single arm	Type	Activation	Information

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

- The specifications of the neural networks of the critic

Layer	Dimensions single arm	Type	Activation	Information
Input	33			tensor with dimensions (batch_size, 33)
Hidden	(33, 128)	Linear	ReLU	
Batch_norm	128	Linear	--	Batch Normalization layer
Concatenation	4	Linear		Concatenation from Batch_norm layer with the action vector
Hidden	(132, 1)	Linear	ReLU	
Output	1	Linear		tensor with dimensions (batch, 1)

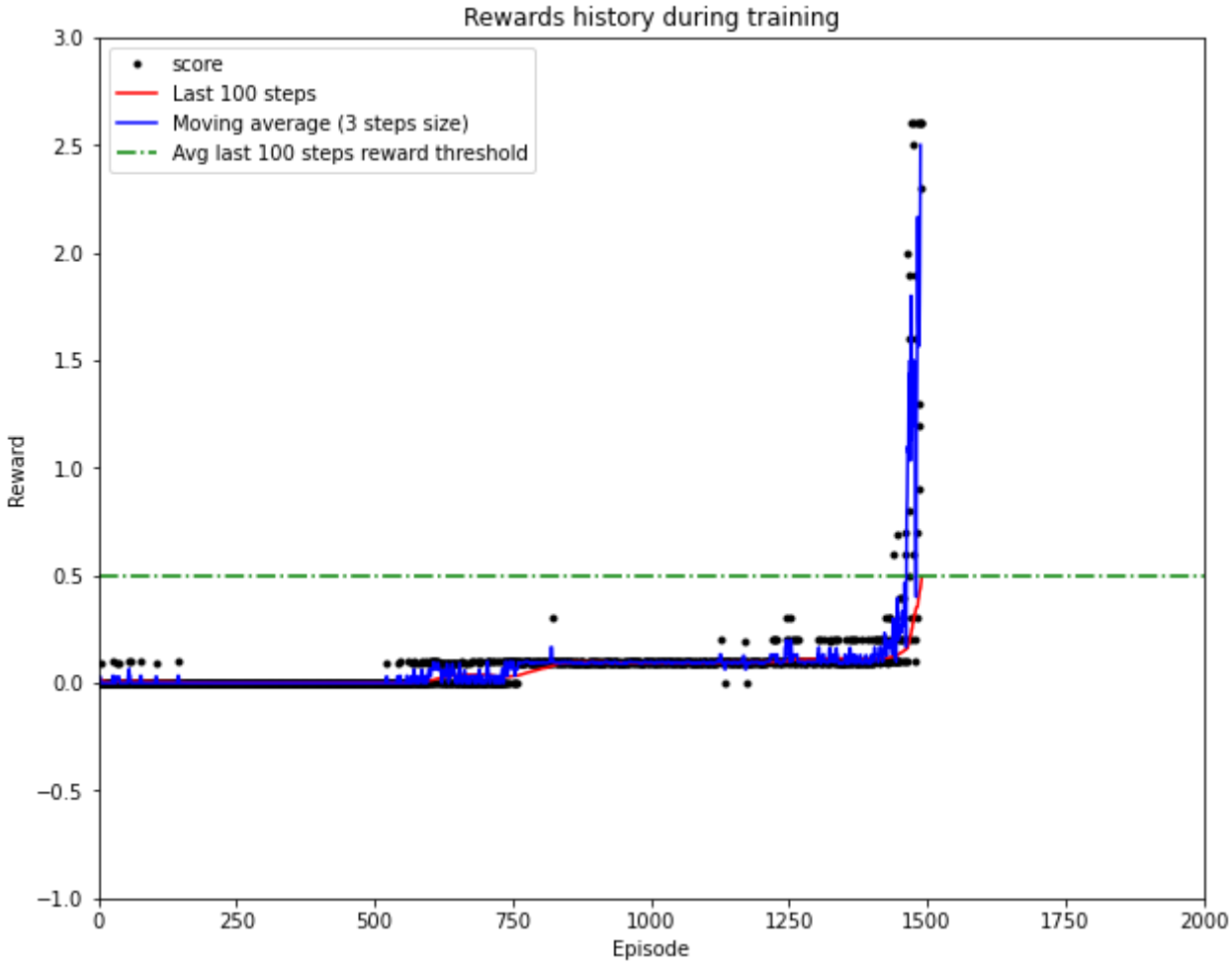
- The variables in the algorithm

Parameter Name	Value	Description
Shared Replay buffer size	100000	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
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

Parameter Name	Value	Description
Reward threshold	0.5	The number of rewards obtained during the last 100 episodes for truncating the training

Results

Using this set of parameters, our agents were able to follow the hit the ball as many time as possible collecting the most rewards they could, needing less than 200 episodes to obtain an average reward of 0.5 over the last 100 episodes. This result is shown in the figure below.



Conclusions and future work

The agents were 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 to keep their performance for larger time step scenarios.

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 Actor-Attention-Critic for multi-agent reinforcement learning and Multi-agent actor centralized-critic with

communication which have been shown to perform better with environments where agents are cooperating and competing to individually collect as many rewards as possible.