Udacity DRLND Project 3: Multiagent

Introduction

For this project, I have worked with a Unity Environment called Tennis. In this environment, two agents control rackets to bounce a ball over a net. If an agent hits the ball over the net, it receives a reward of +0.1. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. Thus, the goal of each agent is to keep the ball in play.

The observation space consists of 8 variables corresponding to the position and velocity of the ball and racket. Each agent receives its own, local observation. Two continuous actions are available, corresponding to movement toward (or away from) the net, and jumping.

The task is episodic, and in order to solve the environment, your agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents). Specifically,

- After each episode, we add up the rewards that each agent received (without discounting), to get a score for each agent. This yields 2 (potentially different) scores. We then take the maximum of these 2 scores.

- This yields a single score for each episode.

The environment is considered solved, when the average (over 100 episodes) of those **scores** is at least +0.5.

Algorithm

The code in the Jupyter Notebook is very similar to the previous projects. The main difference is this time, the agent is itself a multi-agent which receives states and rewards for both players it controls inside. After that, it outputs the actions inferred for both the players as well.

The algorithm selected is Deep Deterministic Policy Gradient (DDPG), an actor-critic algorithm which concurrently learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the Q-function, and uses the Q-function to learn the policy. The same DDPG agent actually learns to play as both players by transforming and understanding its differences and similitudes on states, actions and rewards.

A common replay buffer is used to store the steps of both players. From that replay buffer are periodically retrieved the experiences to train the agent. Some noise is added to the selected action in order to foment exploration. As in prior projects, both actor and critic present different networks. For both, separate target and local weights are maintained in order to avoid instability. To update the target networks, soft updates are accomplished periodically. Adam optimizer is employed for actor and critic.

References for most of the architectures and code used and first parameters tested was taken from previous or later lessons in the nanodegree program.

Model

Architecture

Actor:

1. Normalization. Fully connected layer – input: state size. Output: 128. Leaky ReLu
2. Fully connected layer – input: 128. Output: 128. Leaky ReLu
3. Fully connected layer – input: 128. Output: 128. Leaky ReLu
4. Fully connected layer – input: 128. Output: action size. Hyperbolic tangent

Critic:

1. Normalization. Fully connected layer – input: state size. Output: 128. Leaky ReLu
2. Fully connected layer – input: 128. Output: 128 + action size (concat). Leaky ReLu
3. Fully connected layer – input: 128 + action size. Output: 128. Leaky ReLu
4. Fully connected layer – input: 128. Output: 1.

Parameters

replay buffer size: int(1e 5)

minibatch size: 128

discount factor (gamma) = 0.99

soft update of target parameters from local to target networks (tau) = 1e-3

learning rate of the actor = 1e- 5

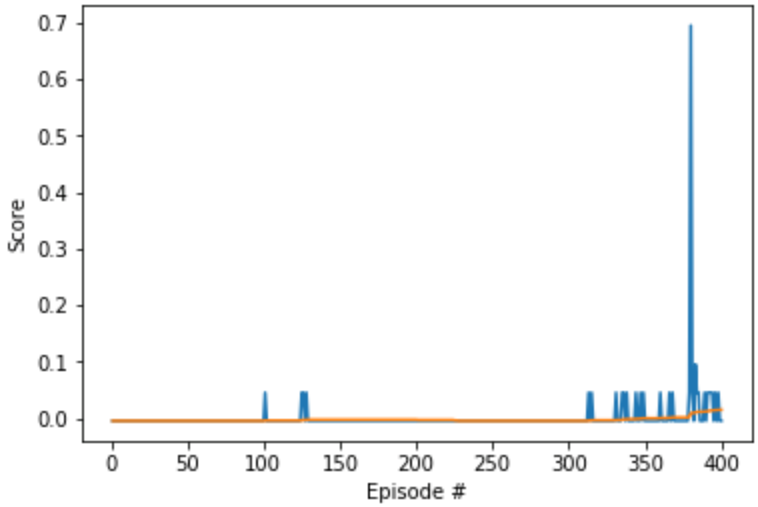
learning rate of the critic = 1e- 4

weight decay = 0

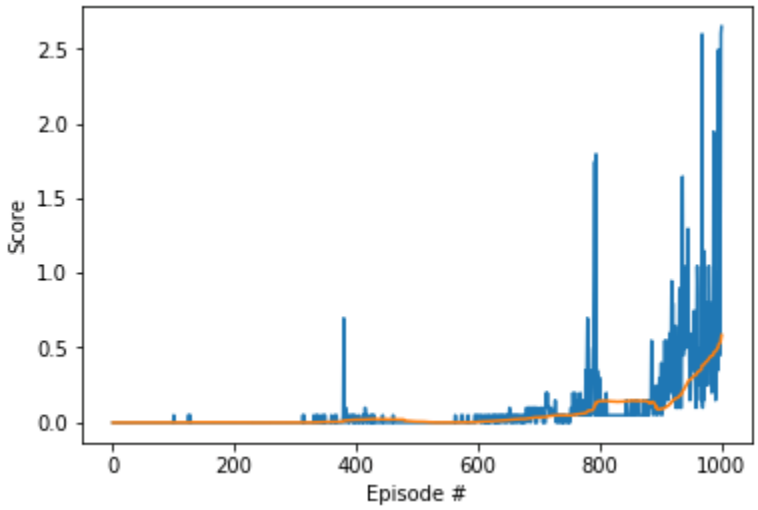
number of steps before each learning (LEARN_EVERY) = 1

Results

Results show a long period of not learning during the firs 300 episodes, in which the agents is still acting randomly anda recovering information of the environment.



Eventually, a series of succesful episodes allow it to gain useful information that leas to a slow increasing in the score. Progresively the performance improvisación untill the agent solves the environment over the episode 990.



Future work

Results show that more training time leads to a significant improvement in the performance, so waiting up to a plateau would be interesting to improve current algorithm's score. Besides, the architecture of actor and critic would be increased with more neurons per layer and maybe on depth level.

Another promising way is the use of prioritized replay PER, that makes use of the idea that when we sample experiences to feed the Neural Network, we assume that some experiences are more valuable than others. If we sample with weights, we can make it so that some experiences which are more

beneficial get sampled more times on average. The benefit of each sample is valued based on the absolute value of the TD Error, which is later smoothed and normalized to calculate the probabilities.

The Soccer environment will be tested to afford a harder challenge. For sure, the time required to train the agent will increase as well the initial period of instability and non-learning.