Deep Reinforcement Learning
Project 3
Collaboration and Competition

Submitted by: Avinash Kaur
Date: 29th June 2019

Objective
The purpose of this project is to apply RL algorithms to train two agents simultaneously that are capable of collaborating (by controlling rackets to bounce a ball over a net) to keep a table tennis ball on the table. This is for an episodic task and the overall score depends on the collaboration of both the agents.

Problem Statement
In this project, we will work with the Unity ML Agents: Tennis Environment.

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.

Learning Algorithm
The algorithm chosen for implementation in this project is the Deep Deterministic Policy Gradient (DDPG).

DDPG (Deep Deterministic Policy Gradient)
DDPG, as the name suggests, is a policy gradient method that directly optimizes parameterized policies instead of estimating a value function and then obtaining an optimal policy. All Policy gradient methods have this as their underlying philosophy. They also provide a robust solution to the intractability caused by continuous action spaces.

In particular, DDPG utilizes two neural networks: an actor and a critic. The Actor tunes the parameter θ of the policy function to decide the best action for a given state, such that:

$$\pi_\theta(s,a) = \mathbb{P}[a \mid s,\theta]$$

Whereas, the critic evaluates the policy function estimated by the actor following the same old Bellman equation such that:

$$r_{t+1} + \gamma V^v(S_{t+1}) - V^v(S_t)$$

Here the terms carry the same meaning as in the TD setting with nothing new until this point (only in critic).

We utilize the same advances we made in the Q learning algorithm to stabilize training i.e. the usage of a replay buffer and a fixed Q Network. This is done in both actor and critic. So, in essence, we have local and target networks for both the actor and the critic. Another change here from the regular Q learning algorithm is the usage of 'soft' updates instead of directly copying the weights of the local network to target after a fixed number of timesteps. It has been observed that this provides more stabilized training than directly updating our target networks. This is expressed by the following equation:

$\theta_{target} = \tau * \theta_{local} + (1 - \tau) * \theta_{target}$

Where 'tau' is the hyperparameter that determines the extent to which we utilize local network to update target network at each time step. It is typically set to 0.001.

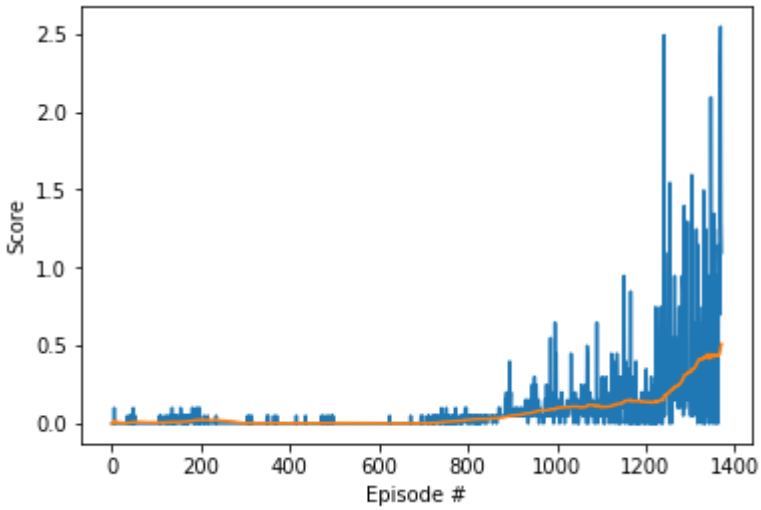
Chosen Hyperparameters

PARAMETER	VALUE
REPLAY BUFFER SIZE	1e6
BATCH SIZE	256
GAMMA	0.99
TAU	0.001
LEARNING RATE ACTOR	2e-4
LEARNING RATE CRITIC	2e-4
WEIGHT DECAY	0.0

Neural Network Architecture

ITEM	ACTOR	CRITIC
INPUT LAYER	1 input layer of size 8 (size of state space)	1 input layer of size 8 (size of state space)
NUMBER OF FC LAYERS AND UNITS	2 FC layers, 256 units each	2 FC Layers, 256 units each. Concatenation of 2 nodes of action space to the output of the first layer before sending to the next layer.
HIDDEN LAYER ACTIVATION FUNCTION	RELU for both layers	RELU for both layers
BATCH NORMALIZATION	Only the output of the first layer	Only the output of the first layer
OUTPUT LAYER	2 nodes (size of action space) with Tanh non linearity to clamp output in the range of -1 to 1.	1 node with RELU non linearity

Plot of rewards per episode



Results

As can be seen from the above plot, the number of episodes needed to solve the environment: 1370 episodes.

Summary and Future ideas of improvements

- DDPG was chosen for this project, following the second project. During training I observed that the algorithm is quite sensitive to initialization. In one of the experiments, at ~1400 episodes the average score of both the agents reached 0.49 and it hovered around that for a while. It steadily decreased after that and stayed around 0.35 for ~5k episodes. With no architectural changes, after re-initializing it, I got the above results. Likewise, for several random runs to confirm this, I got a different result every time. This tells me that the current implementation should not be considered robust.
- During training, it was observed that one agent was clearly 'weaker' (if that's a term) than the other agent. This brought the average score down during the first few hundreds of episodes. However, with time, I could observe the improvement and hence the gradual increase in score. This behavior was also sensitive to initialization. Therefore, in future I'd like to investigate more into precisely what factors lead to the gradual improvement of one agent over the other in a collaborative setting.
- In future, I'll also look into using other algorithms to train this network and for carrying out effective comparison among all algorithms.
- As suggested by several students in the Slack channel, it is worth trying prioritized action replay.