Project 3: Collaboration and Competition

Soroosh Rezazadeh

Introduction:

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

Given below are the characteristics of the environment:

- Unity brain name: TennisBrain
- Number of Visual Observations (per agent): 0
- Vector Observation space type: continuous
- Vector Observation space size (per agent): 8
- Number of stacked Vector Observation: 3
- Vector Action space type: continuous
- Vector Action space size (per agent): 2

Learning Algorithm:

We use the DDPG and the MADDPG (Multi-Agent DDPG) algorithm is this project. DDPG is an actor-critic algorithm that extends DQN to work in continuous space. DDPG concurrently learns a Q-function and a policy.
In MADDPG, two separate agents are trained, and the agents need to collaborate (to prevent the ball hit the ground) and compete (to gather points as many as possible). In MADDPG, each agent's critic is trained using the observations and actions from both agents, whereas each agent's actor is

trained using just its own observations. In MADDPG, the function *step* collects all current info for both agents into the common variable memory of the type ReplayBuffer. Then we get the random sample from memory into the variable experience.

Architecture of Neural Networks:

Overall there are 8 neural networks in our architecture. MADDPG agent creates 2 DDPG agents. Each of two agents creates 4 neural networks: *actor_local*, *actor_target*, *critic_local*, and *critic_target*. Actor and Critic classes are provided by model.py

```
actor_target(state) => next_actions
actor_local(states) => actions_pred
critic_target(state, action) => Q-value
-critic_local(states, actions_pred) => actor_loss
```

Both the actor and critic classes implement the neural network with 3 fully-connected layers.

Actor:
Layer 1: # of units: state_size × fc1_units, activation = relu
Layer 2: # of units: fc1_units × fc2_units, activation = relu
Layer 3: # of units: fc2_units × action_size, activation = tanh

Critic:
Layer 1: # of units: (state_size + action_size) × n_agents × fcs1_units, activation = relu
Layer 2: # of units: fcs1_units × fc2_units, activation = relu
Layer 3: # of units: fc2_units × 1

We set state_size = 24, action_size = 2, and the input parameters fc1_units = fc2_units = fcs1_units = 64

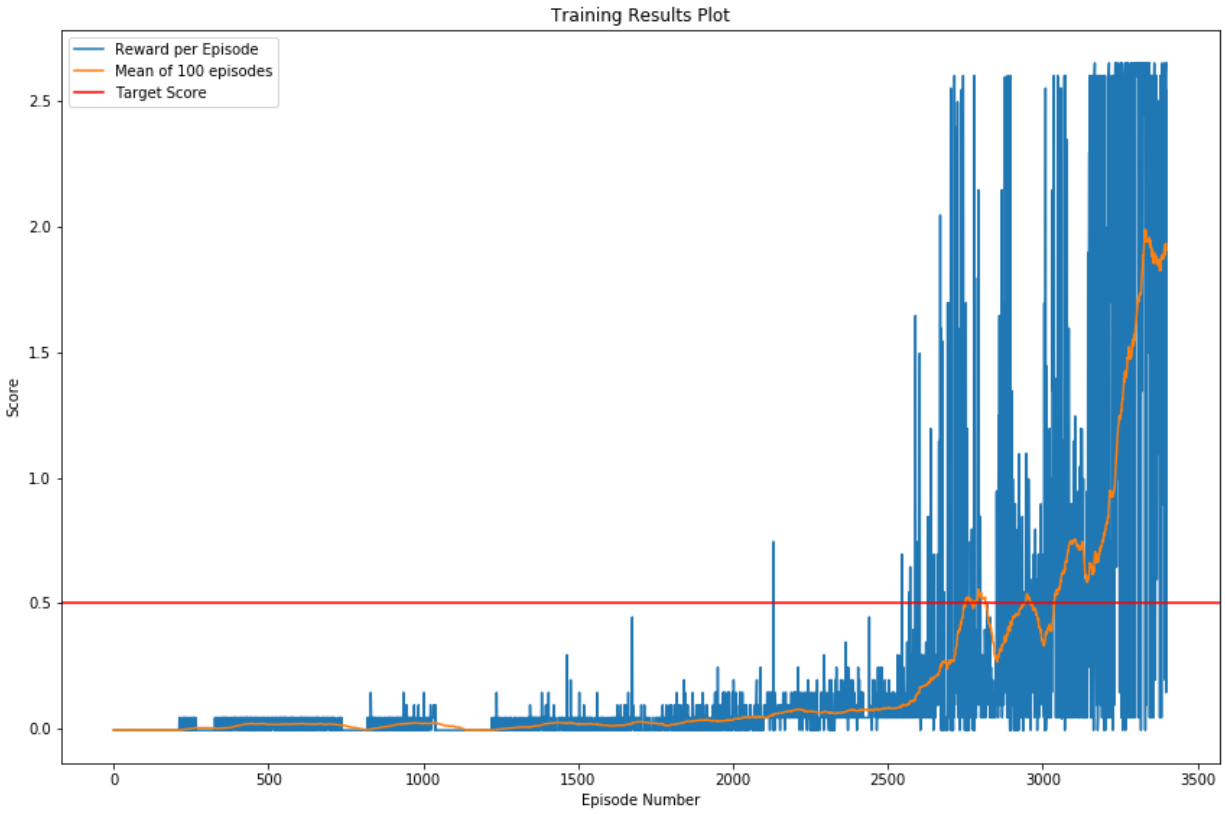
Hyperparameters:

```
GAMMA = 0.99                    # discount factor
TAU = 5e-2                      # for soft update of target
parameters
LR_ACTOR = 5e-4                 # learning rate of the actor
LR_CRITIC = 5e-4                # learning rate of the critic
WEIGHT_DECAY = 0.0              # weight decay
NOISE_AMPLIFICATION = 1         # exploration noise
amplification
NOISE_AMPLIFICATION_DECAY = 1   # noise amplification decay
BUFFER_SIZE = int(1e6)          # replay buffer size
BATCH_SIZE = 512                # minibatch size
```

```
LEARNING_PERIOD = 2                 # weight update frequency
```

Training:

Using the udacity workspace GPU, the average reward +0.5 could be achieved in 2751 episodes. If we continue training, average reward of 1.91 is obtained in 3400 episodes.



Future Works:

- Investigate different hyper-parameters: more non-linear layers, different hidden units
- Add batch norm to accelerate training
- Train agents with different algorithms like multi-agent PPO