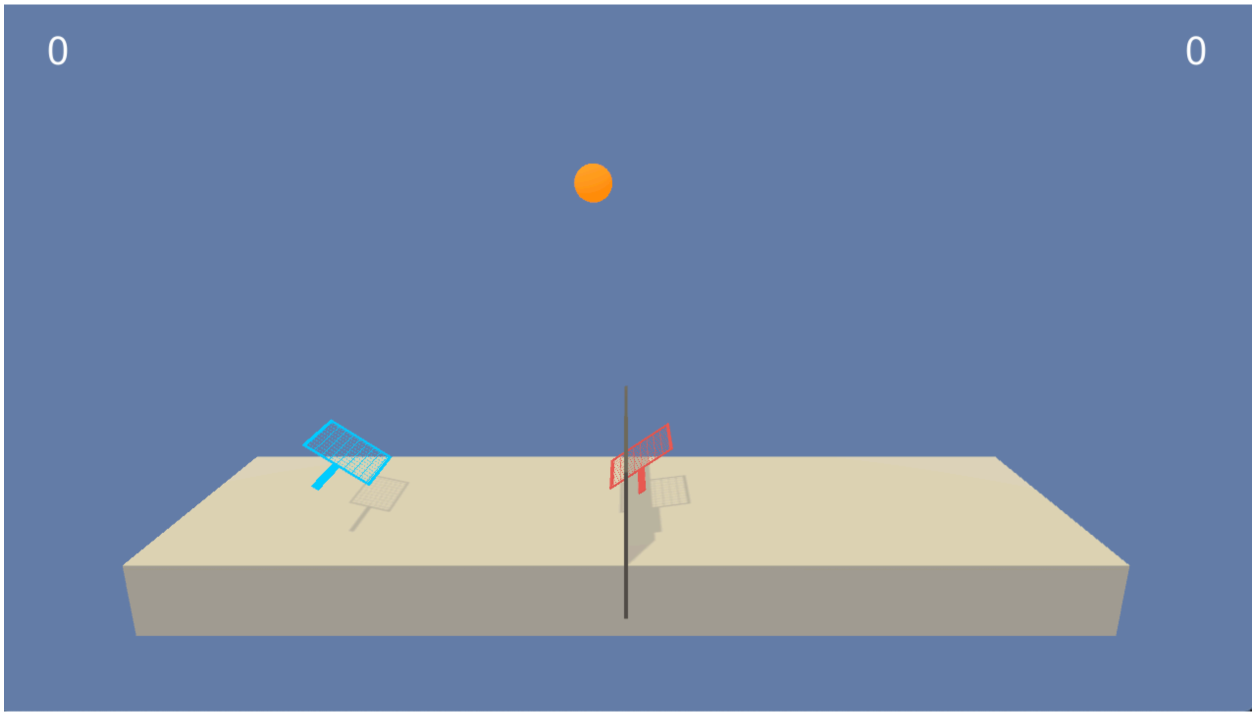
Tennis – Collaboration and Competition

Using Deep reinforcement learning for training multiple agent to compete.

Mohamed Dawod
dawod.moh@gmail.com



Context:

In this project we train two agents to play tennis by controlling a racket to bounce a ball over net.

The environment state and reward scheme

a- Reward:
The agent is reward +0.1 for hitting the ball over the net, and -0.01 if the agent let the ball hit the ground or hit the ball out of bounds.
Thus, the goal for each agent is to keep the ball in the play.

b- State:
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.

c- Score:

This task is episodic, and the score for each episode is calculated by taking the maximum score of both agents. The agent is considered successfully trained is it manged to get an average score of +0.5 (over 100 consecutive episodes).

The Solution:

To solve this project, we used the method of Multi Agent Deep Deterministic Policy Gradient (MADDPG) to solve this project, where each agent player has its own actor and critic network. However both agents are sharing the same random replay buffer. Sharing the same replay buffer, increase the speed of learning as it collects a massive amount of experiences really quickly.

Deep Neural Network Model

The algorithm uses two deep neural networks (actor-critic) with the following structure:

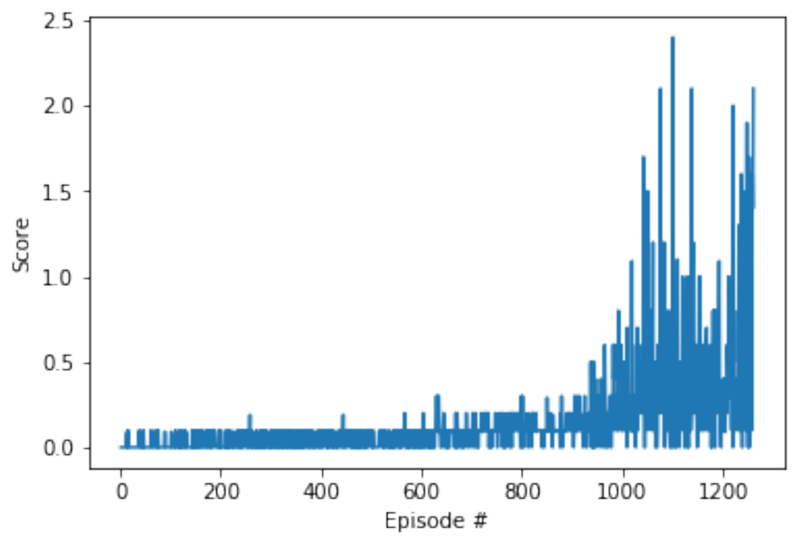
- Actor
    - Hidden: (State_size, 256) - ReLU
    - Hidden: (256, 128) - ReLU
    - Output: (128, 4) - TanH
- Critic
    - Hidden: (State_size, 256) - ReLU
    - Hidden: (256 + action_size, 128) - ReLU
    - Output: (128, 1) - Linear

Hyperparameters

```
BUFFER_SIZE = int(1e6)              # replay buffer size
BATCH_SIZE = 512                    # minibatch size
GAMMA = 0.99                        # discount factor
TAU = 1e-3                          # for soft update of target parameters
LR_ACTOR = 1e-4                     # learning rate of the actor
LR_CRITIC = 3e-4                    # learning rate of the critic
WEIGHT_DECAY = 0                    # L2 weight decay
ACTOR_HL_SIZE= [256, 128]          #Actor Hidden layers
CRITIC_HL_SIZE= [256, 128]         #Critic Hidden layers
n_episodes= 3000, max_t=2000  #Number of episodes and times steps
```

Results

```
Episode 100   Current Score: 0.00   Average Score: 0.01
Episode 200   Current Score: 0.10   Average Score: 0.03
Episode 300   Current Score: 0.09   Average Score: 0.03
Episode 400   Current Score: 0.00   Average Score: 0.04
Episode 500   Current Score: 0.00   Average Score: 0.05
Episode 600   Current Score: 0.00   Average Score: 0.04
Episode 700   Current Score: 0.09   Average Score: 0.07
Episode 800   Current Score: 0.00   Average Score: 0.10
Episode 900   Current Score: 0.10   Average Score: 0.11
Episode 1000  Current Score: 0.10   Average Score: 0.18
Episode 1100  Current Score: 0.20   Average Score: 0.36
Episode 1200  Current Score: 0.30   Average Score: 0.38
Episode 1261  Current Score: 2.10   Average Score: 0.51
Environment solved in 1261 episodes! Average Score: 0.51
Environment solved in 3182.52 seconds
```



Future Work

I'm willing to try to improve the algorithm by implementing the following concepts:

1- Prioritized Experience Replay
2- Exploration with Parameter Noise