REPORT OF IMPLEMENTATION

Overview

The goal of this project is to achieve an
The environment consists of two agents that control racquets to bounce a ball over a net.
average reward of at least +0.5 over 100 episodes in the environment. This is calculated by taking the maximum score from the two agents which is added up without discounting at the end of the episode.
A reward of +0.1 is given to an agent if it hits the ball over the net. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. The goal of each agent is to keep the ball in play.
The state space consists of eight variables corresponding to the position and velocity of the ball and racquet. Each agent receives its own, local observation.
The action space is two continuous actions, corresponding to movement toward (or away from) the net, and jumping.

Learning Algorithm

In this project was used Actor-Critic method, through the DDPG algorithm. DDPG means Deep Deterministic Policy Gradient and extend policy-based reinforcement learning methods to complex problems using deep neural networks.
DDPG is very similar to DQN method, using ReplayBuffer. A big difference is that DQN is used to discrete action spaces and DDPG is used to continuous action spaces.
In algorithms like DDPG that implement actor-critic method we have two neural networks, one is the actor and another is the critic.
Also, was used Experience Replay and Fixed Q-Targets methods to improve the agent's performance.
OUNoise Class is used to add noise to actions to promote exploration. It uses the Ornstein-

Uhlenback process to achieve this.

CODE

1- model.py
In this file was created the actor and critic classes, each one with your own neural network.
The neural network was created using PyTorch.

2- agent.py
In this file was created the Agent class.
Looking for learning and improving the results, the agent implements the Experience Replay and Fixed Q-Targets methods.

3- OUNoise.py
OUNoise Class is used to add noise to actions to promote exploration. It uses the Ornstein-

Uhlenback process to achieve this.

4- ReplayBuffer.py

Was created the ReplayBuffer class to implement the Experience Replay method.

4- Training the agent
The agent was trained in 1.500 episodes
The environment is considered solved when the agent achieve the mean score >= 0.5

ATTEMPT 1

Model.py
Actor model architecture:

- three fully connected layers, receiving the state as input and actions as output.
- 48(Input-state_size*2) x 256(hidden_layer) x 128(hidden_layer) x 2(output - action_size)
- Was applied ReLu activation at hidden layers and Tanh activation at output layer.

Critic model architecture:

- three fully connected layers, receiving the state as input and return as output only one unit, the Q-values.
- 48(Input-state_size*2) x 260(hidden_layer+action_size*2) x 128(hidden_layer) x 1(output - Q-values)
- Was applied ReLu activation at hidden layers.

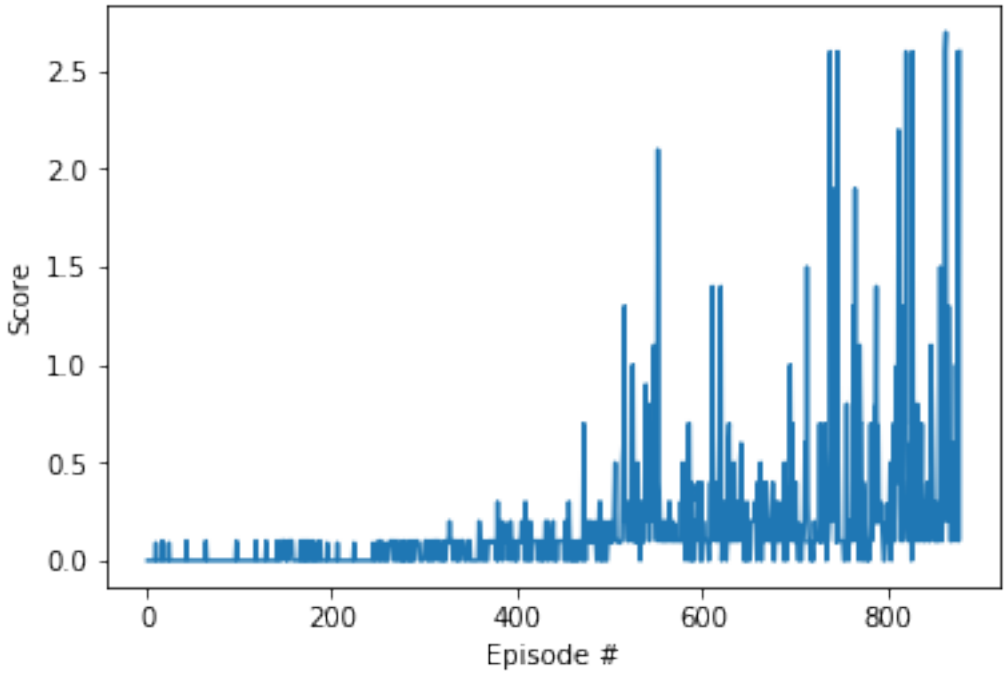
Hyperparameters used to train the agent:

```
BUFFER_SIZE = int(1e6)  # replay buffer size
BATCH_SIZE = 128        # minibatch size
GAMMA = 0.99            # discount factor
TAU = 6e-2              # for soft update of target parameters
LR_ACTOR = 1e-3         # learning rate of the actor
LR_CRITIC = 1e-3        # learning rate of the critic
WEIGHT_DECAY = 0        # L2 weight decay
UPDATE_EVERY = 1        # how often to update the network
N_EPISODES = 1500       # number of episodes
eps_start = 6           # Noise level start
eps_end = 0             # Noise level end
eps_decay = 250         # Number of episodes to decay over from start to end
```

RESULT

With those architecture and hyper parameters was possible achieve the Average Score of the 0.52 in only 878 episodes.
Below the progress of the agent's learning.

```
Episode:50 Score:0.00 Best Score:0.10       Average Score:0.01
Episode:100      Score:0.00 Best Score:0.10       Average Score:0.01
Episode:150      Score:0.00 Best Score:0.10       Average Score:0.01
Episode:200      Score:0.00 Best Score:0.10       Average Score:0.02
Episode:250      Score:0.09 Best Score:0.10       Average Score:0.02
Episode:300      Score:0.09 Best Score:0.10       Average Score:0.02
Episode:350      Score:0.00 Best Score:0.20       Average Score:0.05
Episode:400      Score:0.00 Best Score:0.30       Average Score:0.07
Episode:450      Score:0.10 Best Score:0.30       Average Score:0.08
Episode:500      Score:0.10 Best Score:0.70       Average Score:0.09
Episode:550      Score:0.40 Best Score:1.30       Average Score:0.20
Episode:600      Score:0.00 Best Score:2.10       Average Score:0.26
Episode:650      Score:0.00 Best Score:2.10       Average Score:0.24
Episode:700      Score:0.10 Best Score:2.10       Average Score:0.23
Episode:750      Score:0.10 Best Score:2.60       Average Score:0.30
Episode:800      Score:0.10 Best Score:2.60       Average Score:0.36
Episode:850      Score:0.20 Best Score:2.60       Average Score:0.42
Environment solved in 878 episodes2.70      Average Score:0.52
```



Future ideas for improving agent's performance

Looking for improving the agent's performance we suggest to try the following actions:

- Modifying the hyper parameters, looking for speed up training or increase the final score.
- Implement a differente algorithm like PPO or D4PG.
- Modifying the model architecture by changing the number of layers or neurons.
- Change the update frequency of the networks in the step function.