Report

9 April 2023

Introduction

This project is on the Tennis environment.

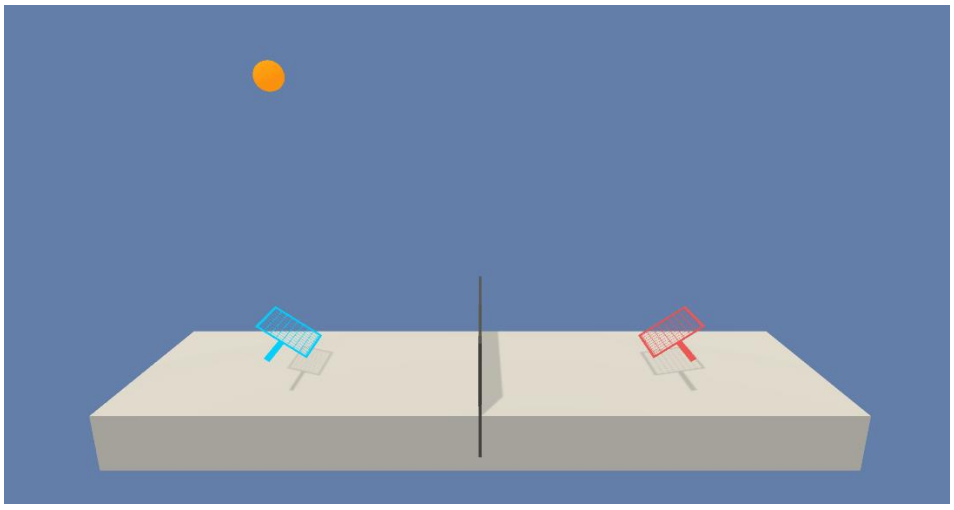
In this environment, two agents control rackets to bounce a ball over a net. If an agent hits the ball over the net, it receives a reward of +0.1. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. Thus, the goal of each agent is to keep the ball in 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, the 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 used in this project is a Multi-Agent Deep Deterministic Policy Gradient (MADDPG), an algorithm in actor-critic method. It is based on DDPG agent used in the previous Continuous Control project.

A DDPG has two networks, actor and critic. The goal of the actor is to learn a policy that maximizes the expected return over time. The critic is responsible for estimating the value function of the policy. The value function estimates the expected return for a given state. The goal of the critic is to learn an accurate value function that can guide the actor towards actions that lead to higher expected returns.

Both actor and critic have two networks, local and target. They are to stabilize the learning process and prevent divergence. The local networks are the main networks used for action selection and value estimation. Target networks are used to estimate the target values during the learning process. The weights of the target networks are periodically updated with a fraction of the weights of the corresponding local networks, providing a more stable target for learning. This reduces the correlation between successive observations and results in a more stable learning process.

The MADDPG algorithm has a decentralized actor with centralized critic networks. It uses a critic network with actions and state observations as inputs.

The hyperparameter are as follows:

```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 256        # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # 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
```

The network:

The input to the actor network is the observation space consisting of 24 variables and the output of 2 numbers is the predicted best action for that observed state.

The input to the critic network is also the observation space. The input to the critic's second hidden layer is the output of the critic's first hidden layer and the output of the actor's network. The output of the overall network is the prediction of the target value based on the given state and the predicted best action.

In short, critic calculates the optimal action-value function *Q(s, a)* by using the action from actor's deterministic optimal policy.

Actor:
First layer: input size = **24**, output size = 256
Second layer: input size = 256, output size = 128
Third layer: input size = 128, output size = **2**

Critic:
First layer: input size = 24, output size = 256
Second layer: input size = **258**, output size = 128
Third layer: input size = 128, output size = **1**

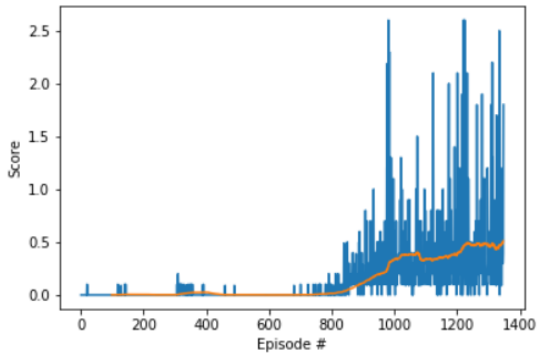
Plot of Rewards

The average score over 100 episodes is **0.509** achieved in **1348 episodes**.

```
with active_session():
    # do long-running work here
    scores_all = ddpg()
```

```
Episode 100, Average Score: 0.00, Max: 0.10, Min: 0.00, Avg: 0.00
Episode 200, Average Score: 0.00, Max: 0.10, Min: 0.00, Avg: 0.00
Episode 300, Average Score: 0.00, Max: 0.00, Min: 0.00, Avg: 0.00
Episode 400, Average Score: 0.03, Max: 0.20, Min: 0.00, Avg: 0.03
Episode 500, Average Score: 0.00, Max: 0.09, Min: 0.00, Avg: 0.00
Episode 600, Average Score: 0.00, Max: 0.00, Min: 0.00, Avg: 0.00
Episode 700, Average Score: 0.00, Max: 0.09, Min: 0.00, Avg: 0.00
Episode 800, Average Score: 0.02, Max: 0.20, Min: 0.00, Avg: 0.02
Episode 900, Average Score: 0.11, Max: 0.60, Min: 0.00, Avg: 0.11
Episode 1000, Average Score: 0.33, Max: 2.60, Min: 0.00, Avg: 0.33
Episode 1100, Average Score: 0.34, Max: 1.50, Min: 0.00, Avg: 0.34
Episode 1200, Average Score: 0.37, Max: 2.10, Min: 0.00, Avg: 0.37
Episode 1300, Average Score: 0.48, Max: 2.60, Min: 0.00, Avg: 0.48
Episode 1348, Average Score: 0.51, Max: 2.50, Min: 0.00, Avg: 0.51

Environment solved in 1348 episodes!    Average Score: 0.509

Saving model ... done.
```

```
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(np.arange(1, len(scores_all)+1), scores_all)

rolling_window = 100
rolling_mean = pd.Series(scores_all).rolling(rolling_window).mean()
plt.plot(rolling_mean);

plt.ylabel('Score')
plt.xlabel('Episode #')
plt.show()
```



Future Works

To improve the performance of the agent, below are some of the possible future works.

- Hyper-parameters
  To investigate optimal parameters to solve a given environment.
- Prioritized Experience Replay
  This technique learns more efficiently by replaying important transitions more frequently unlike in the current implementation where all transitions are sampled uniformly.
- To implement MADDPG with a private actor-critic network for each agent.