MADDPG for Unity Tennis Environment (Multi-Player)

Report of Results

I. 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 generated actions should be two continuous values, corresponding to movement to & fro 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.

II. Results

In this implementation, the MADDPG methodology is used to train the agents (players). The actions of agents are shared by both the agents. A replay buffer in implemented as part of the MADDPG agent to save the states & actions of both the agents. The actions are given as input to the first hidden layer of the critic network. No other information is shared in this implementation.

The DDPG network was an MLP with 2 hidden layers. The input layer has 24 neurons as there were 24 observations from the environment and the output layer has 4 neurons for actor and 1 for critic. The Hidden layers each had 256 neurons. Please refer to architecture for details. The network achieved an average of 0.5 over 100 consecutive episodes within 7000 iterations of training. To converge gradient clipping was needed. This is in the learn function before critic_optimizer.step().

First 2000 episodes were run with random generated actions and these actions are saved only if there was any reward. The training is initiated once the Replay buffer has more than 7 times the batch size of data.

Parameters :

Mini batch size :128.

TAU for soft updates : 1e-3

Hidden layers : 2 layers of 256 neurons

Discount rate gamma = 0.99

Replay buffer max size : 1e5

Noise parameters:

Theta: 0.10 and Sigma = 0.10 for Ornstein-Uhlenbeck process.

The training started converging only after many tweaks. The training report screen shots are below.

1) Results with learning update once as the score reaches 5.0 fig 1.

Starting learning rate (actor and critic) : 1e-4

Ending learning rate : (actor & critic) : 1e-4

The performance graphs are below

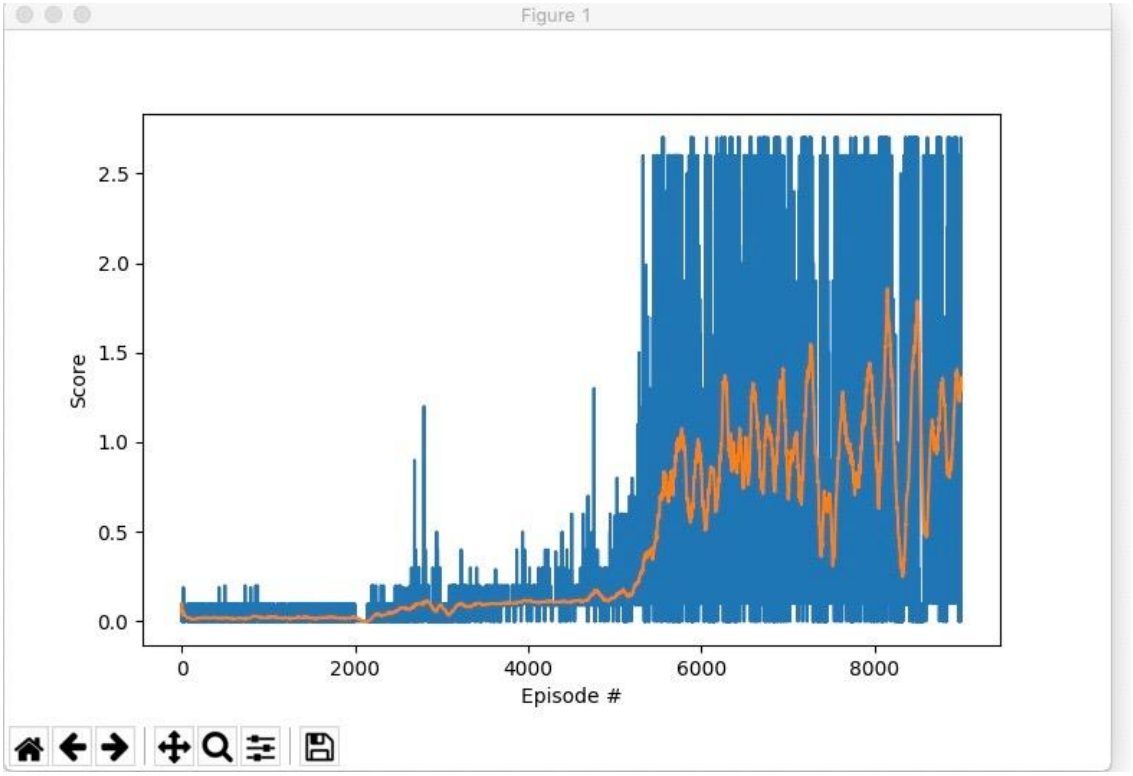
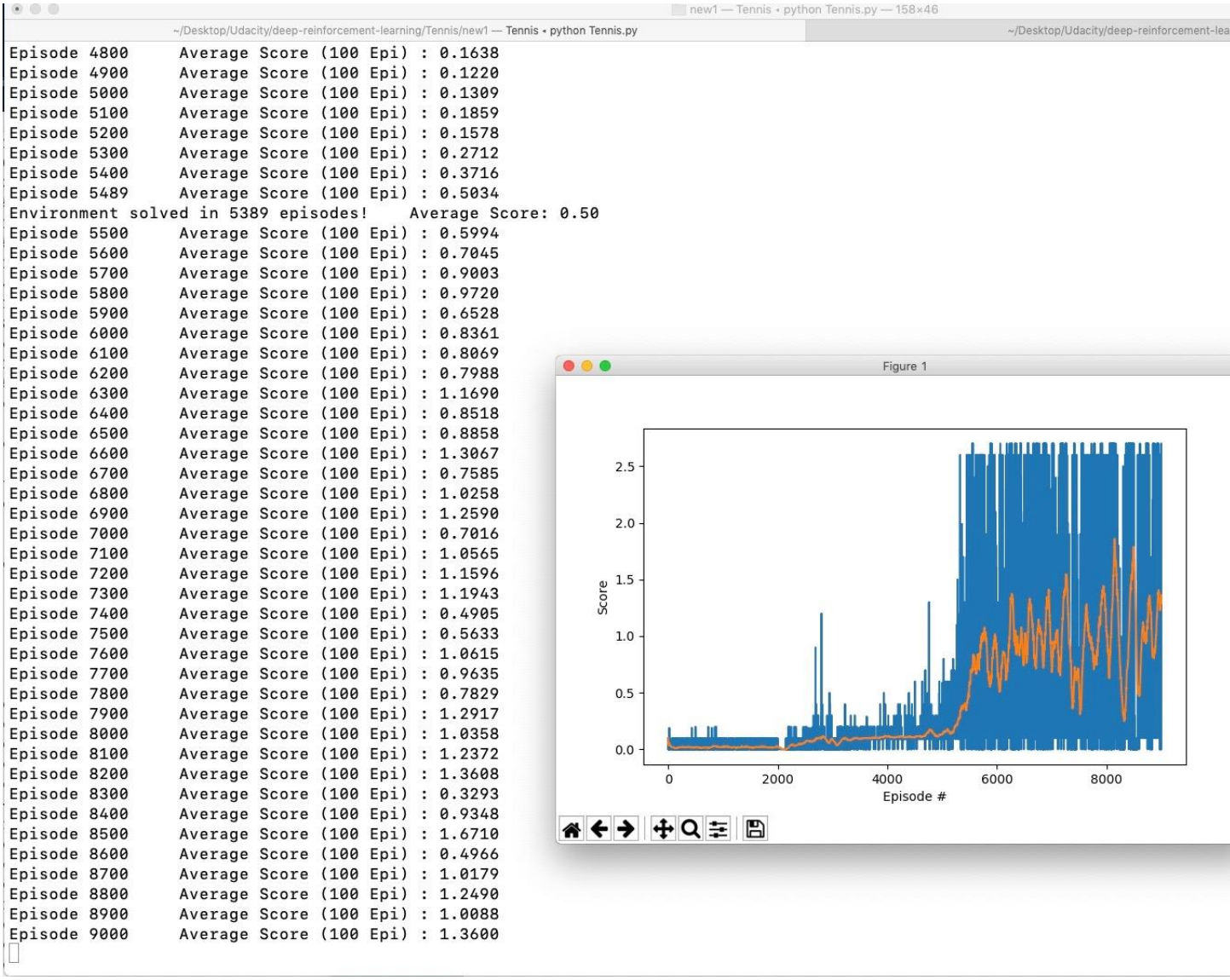


Fig. 1 Score with average scores over 100 episodes (orange line)

~/Desktop/Udacity/deep-reinforcement-learning/Tennis/new1 — Tennis ◂ python Tennis.py

```
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.         -6.65278625 -1.5
  -0.          0.          6.83172083  6.         -0.          0.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.         -6.4669857  -1.5
   0.          0.         -6.83172083  6.          0.          0.        ]]
================================================================================
================================================================================
=======================Network Training=========================================
        First 2000 episodes, actions are random generated...
        ------------------------------------------------------------
Episode 1000     Average Score (100 Epi) : 0.0231
Episode 1999     Average Score (100 Epi) : 0.0249
        Switiching to Network generated action...
        ----------------------------------------
Episode 2000     Average Score (100 Epi) : 0.0249
Episode 2100     Average Score (100 Epi) : 0.0000
Episode 2200     Average Score (100 Epi) : 0.0216
Episode 2300     Average Score (100 Epi) : 0.0309
Episode 2400     Average Score (100 Epi) : 0.0503
Episode 2500     Average Score (100 Epi) : 0.0642
Episode 2600     Average Score (100 Epi) : 0.0781
Episode 2700     Average Score (100 Epi) : 0.0924
Episode 2800     Average Score (100 Epi) : 0.1173
Episode 2900     Average Score (100 Epi) : 0.0705
Episode 3000     Average Score (100 Epi) : 0.0980
Episode 3100     Average Score (100 Epi) : 0.0407
Episode 3200     Average Score (100 Epi) : 0.1051
Episode 3300     Average Score (100 Epi) : 0.0830
Episode 3400     Average Score (100 Epi) : 0.1069
Episode 3500     Average Score (100 Epi) : 0.0944
Episode 3600     Average Score (100 Epi) : 0.1099
Episode 3700     Average Score (100 Epi) : 0.1090
Episode 3800     Average Score (100 Epi) : 0.1025
Episode 3900     Average Score (100 Epi) : 0.1181
Episode 4000     Average Score (100 Epi) : 0.1134
Episode 4100     Average Score (100 Epi) : 0.1171
Episode 4200     Average Score (100 Epi) : 0.1122
Episode 4300     Average Score (100 Epi) : 0.1156
Episode 4400     Average Score (100 Epi) : 0.1139
Episode 4500     Average Score (100 Epi) : 0.1162
Episode 4600     Average Score (100 Epi) : 0.1259
Episode 4700     Average Score (100 Epi) : 0.1394
Episode 4800     Average Score (100 Epi) : 0.1638
Episode 4900     Average Score (100 Epi) : 0.1220
Episode 5000     Average Score (100 Epi) : 0.1309
Episode 5100     Average Score (100 Epi) : 0.1859
Episode 5200     Average Score (100 Epi) : 0.1578
Episode 5300     Average Score (100 Epi) : 0.2712
Episode 5400     Average Score (100 Epi) : 0.3716
Episode 5489     Average Score (100 Epi) : 0.5034
Environment solved in 5389 episodes!    Average Score: 0.50
Episode 5500     Average Score (100 Epi) : 0.5994
Episode 5600     Average Score (100 Epi) : 0.7045
Episode 5700     Average Score (100 Epi) : 0.9003
Episode 5800     Average Score (100 Epi) : 0.9720
Episode 5900     Average Score (100 Epi) : 0.6528
Episode 6000     Average Score (100 Epi) : 0.8361
Episode 6100     Average Score (100 Epi) : 0.8069
Episode 6200     Average Score (100 Epi) : 0.7988
Episode 6300     Average Score (100 Epi) : 1.1690
Episode 6400     Average Score (100 Epi) : 0.8518
Episode 6500     Average Score (100 Epi) : 0.8858
Episode 6600     Average Score (100 Epi) : 1.3067
Episode 6700     Average Score (100 Epi) : 0.7585
Episode 6800     Average Score (100 Epi) : 1.0258
Episode 6900     Average Score (100 Epi) : 1.2590
Episode 7000     Average Score (100 Epi) : 0.7016
Episode 7100     Average Score (100 Epi) : 1.0565
Episode 7200     Average Score (100 Epi) : 1.1596
Episode 7300     Average Score (100 Epi) : 1.1943
Episode 7400     Average Score (100 Epi) : 0.4905
Episode 7500     Average Score (100 Epi) : 0.5633
Episode 7600     Average Score (100 Epi) : 1.0615
Episode 7700     Average Score (100 Epi) : 0.9635
Episode 7800     Average Score (100 Epi) : 0.7829
Episode 7900     Average Score (100 Epi) : 1.2917
Episode 8000     Average Score (100 Epi) : 1.0358
Episode 8100     Average Score (100 Epi) : 1.2372
Episode 8200     Average Score (100 Epi) : 1.3608
Episode 8300     Average Score (100 Epi) : 0.3293
Episode 8400     Average Score (100 Epi) : 0.9348
Episode 8500     Average Score (100 Epi) : 1.6710
Episode 8600     Average Score (100 Epi) : 0.4966
Episode 8700     Average Score (100 Epi) : 1.0179
Episode 8800     Average Score (100 Epi) : 1.2490
Episode 8900     Average Score (100 Epi) : 1.0088
Episode 9000     Average Score (100 Epi) : 1.3600
```

Fig.2 Output Logs



III. Future Improvements

1. Adding priority to experience replay

I ran many experiments I ran to get MADDPG converge it seem to be a good option to use prioritized experience replay with MADDPG for faster and stable convergence. To use

prioritization based on TD-Error or Stochastic prioritization is better for DDPG is something than need to be explored. Also, I haven't shared the state information between agents while training, only the actions. This might be the reason of slow convergence while training. I am working on a version that does the same.

1.1 Prioritizing with TD-error

The central component of prioritized replay is the criterion by which the importance of each transition is measured. One idealized criterion would be the amount the RL agent can learn from a transition in its current state (expected learning progress). While this measure is not directly accessible, a reasonable proxy is the magnitude of a transition's TD error δ, which indicates how 'surprising' or unexpected the transition is: specifically, how far the value is from its next-step bootstrap estimate. The TD-error can be a poor estimate in some circumstances as well, e.g. when rewards are noisy.

1.2 Stochastic prioritization

In Stochastic prioritization, a stochastic sampling method that interpolates between pure greedy prioritization and uniform random sampling is used. It ensures that the probability of being sampled is monotonic in a transition's priority, while guaranteeing a non-zero probability even for the lowest-priority transition. Concretely, the probability of sampling transition i is defined as

$$P(i) = \frac{p_i^{\propto}}{\sum_k P_k^{\propto}}$$

2.Asynchronous Methods for Deep Reinforcement Learning.

I had been training MADDPG on a single CPU machine on single thread. This takes extremely long time (many hours) to experiment with different parameters. An additional improvement would be to use A3C to optimize the training and debug times.