MADDPG Collaboration and Competition project Tennis

Yunze Han

DEEP REINFORCEMENT LEARNING NANODEGREE
UDACITY

October 9, 2021

Content

Part 1

Project description

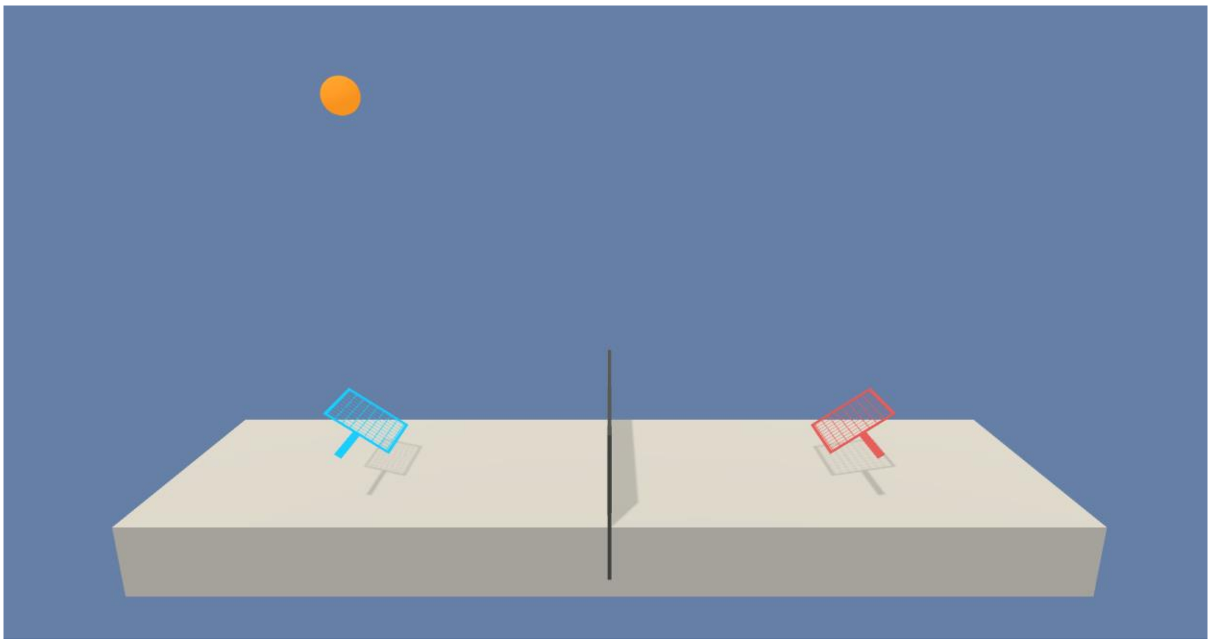
1.1 Environment

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, 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.

1.2 Learning algorithm

MADDPG, or Multi-agent DDPG, extends DDPG into a multi-agent policy gradient algorithm where decentralized agents learn a centralized critic based on the observations and actions of all agents. It leads to learned policies that only use local information (i.e. their own observations) at execution time, does not assume a differentiable model of the environment dynamics or any particular structure on the communication method between agents, and is applicable not only to cooperative interaction but to competitive or mixed interaction involving both physical and communicative behavior. The critic is augmented with extra information about the policies of other agents, while the actor only has access to local information. After training is completed, only the local actors are used at execution phase, acting in a decentralized manner.

Actor:

1.Fc1: input 24 output 256	activation function: ReLu
2.Fc2: input 256 output 128	activation function: ReLu
3.Fc3: input 128 output 4	activation function: tanh

Critic:

1.Fc1: input 48 output 256	activation function: ReLu
2.Fc2: input 260 output 128	activation function: ReLu
3.Fc3: input 128 output 1	

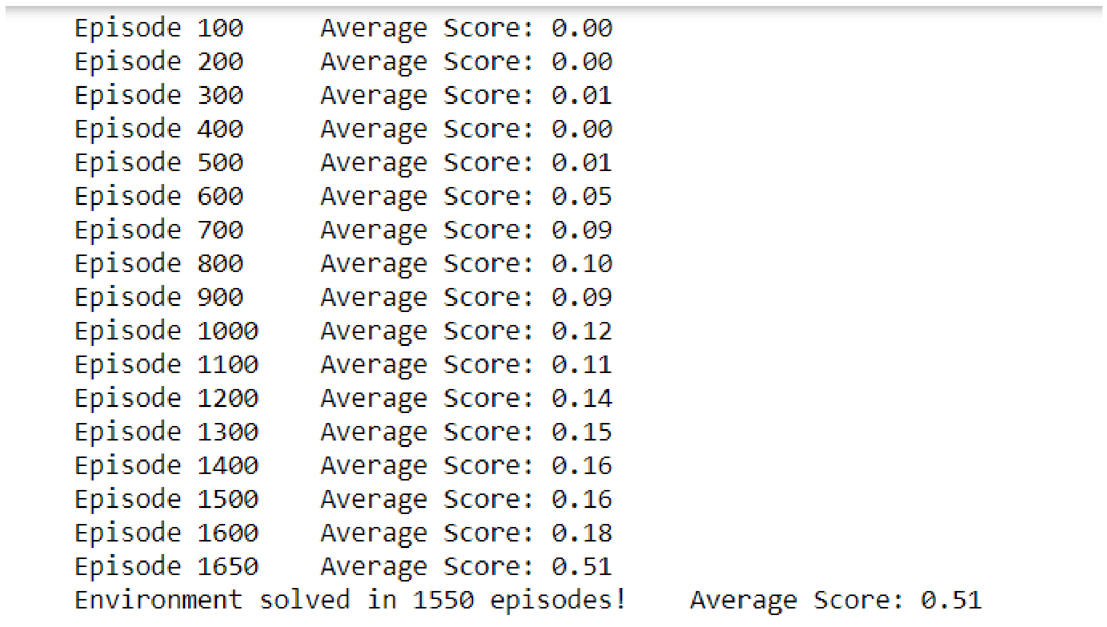
The training hyperparameters:

```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 128        # minibatch size
GAMMA = 0.99            # discount factor
TAU = 5e-3              # 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        # L2 weight decay
UPDATE_EVERY = 3
NOISE_DECAY = 0.99
BEGIN_TRAINING_AT = 500
NOISE_START = 1.0
NOISE_END = 0.1
```

Part 2

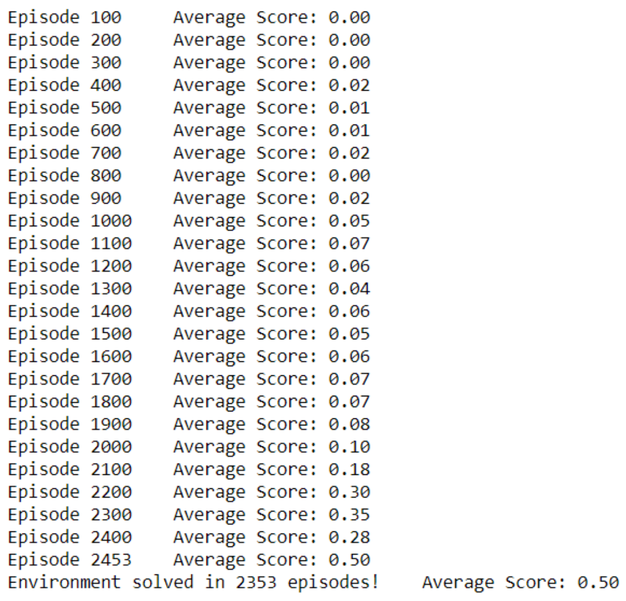
Plot of rewards

1. One result I obtained when trying different hyperparameters: buffersize 10e6, update_every = 1, so the training process is very slow (it takes more than one hours to train):
   Environment solved in 1550 episodes!   Average Score: 0.51



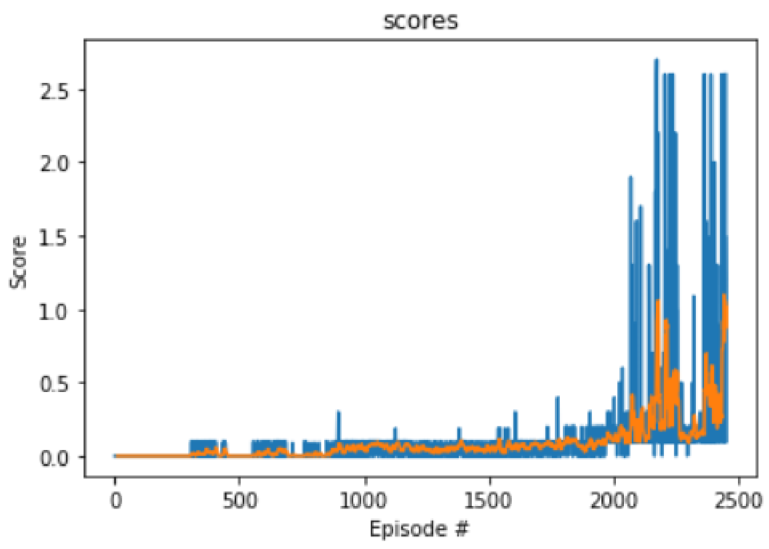
```
Episode 100     Average Score: 0.00
Episode 200     Average Score: 0.00
Episode 300     Average Score: 0.01
Episode 400     Average Score: 0.00
Episode 500     Average Score: 0.01
Episode 600     Average Score: 0.05
Episode 700     Average Score: 0.09
Episode 800     Average Score: 0.10
Episode 900     Average Score: 0.09
Episode 1000    Average Score: 0.12
Episode 1100    Average Score: 0.11
Episode 1200    Average Score: 0.14
Episode 1300    Average Score: 0.15
Episode 1400    Average Score: 0.16
Episode 1500    Average Score: 0.16
Episode 1600    Average Score: 0.18
Episode 1650    Average Score: 0.51
Environment solved in 1550 episodes!    Average Score: 0.51
```

2. Final result: (with hyperparameter: buffer_size= 10e5,and update_every=3, so the training prosses is quicker but it needs more episodes to train the network, it takes about half hour to finish the task)
   Environment solved in 2353 episodes!   Average Score: 0.50



```
Episode 100     Average Score: 0.00
Episode 200     Average Score: 0.00
Episode 300     Average Score: 0.00
Episode 400     Average Score: 0.02
Episode 500     Average Score: 0.01
Episode 600     Average Score: 0.01
Episode 700     Average Score: 0.02
Episode 800     Average Score: 0.00
Episode 900     Average Score: 0.02
Episode 1000    Average Score: 0.05
Episode 1100    Average Score: 0.07
Episode 1200    Average Score: 0.06
Episode 1300    Average Score: 0.04
Episode 1400    Average Score: 0.06
Episode 1500    Average Score: 0.05
Episode 1600    Average Score: 0.06
Episode 1700    Average Score: 0.07
Episode 1800    Average Score: 0.07
Episode 1900    Average Score: 0.08
Episode 2000    Average Score: 0.10
Episode 2100    Average Score: 0.18
Episode 2200    Average Score: 0.30
Episode 2300    Average Score: 0.35
Episode 2400    Average Score: 0.28
Episode 2453    Average Score: 0.50
Environment solved in 2353 episodes!    Average Score: 0.50
```

Here is the graph of the score evolution:



Part 3

Ideas of future work

1. To improve the efficiency of experience replay, we can replace the original uniform experience replay with prioritized experience replay.
2. We can consider different Noise function
3. Continue to adjust hyperparameters to get better training results