UDACITY DEEP REINFORCEMENT NANODEGREE
PROJECT 3
TENNIS SIMULATOR: COLLABORATION AND COMPETITION

INTRODUCTION

This project was carried out as a part of the deep reinforcement learning nanodegree from Udacity. For this project the environment provided is Tennis made in unity3D.

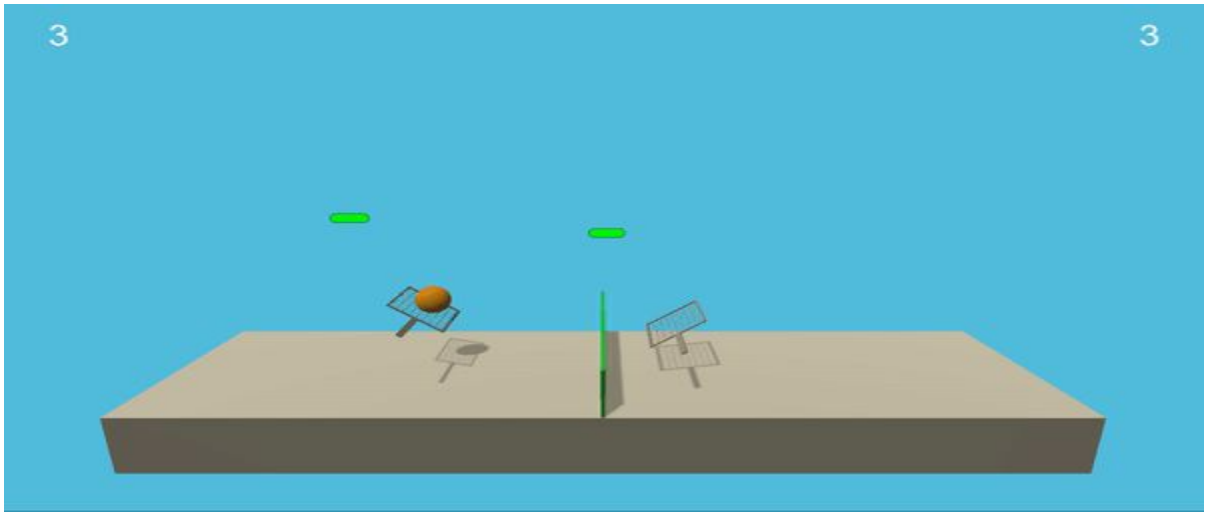
In this environment, two agents control rackets to bounce a ball over a net. If an agent hits the ball over the net, it receives a reward of +0.1. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. Thus, the goal of each agent is to keep the ball in 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.



IMPLEMENTATION

The implementation consists of files ddgn_agent.py, model.py and ddpg_trainer.py. The code is organized as follows:

1. tennis.ipynb notebook contains code to initiate training and testing of the model
2. model.py contains the Actor and Critic neural network code that is used by the agent.
3. ddpg_agent.py contains code that is used to train the agent. This code was adopted for the OpenAI Baseline.
4. Maddgp.py module creates agents using MADDPG algorithm, replay buffer

For information about the project and the environment can be found in the file README.md.

LEARNING ALGORITHM

The algorithm chosen to solve the tennis problem is Multi Agent Deep Deterministic Policy Gradient (MADDPG), an extension of DDPG Algorithm applicable for multiple agents. The algorithm for MADDPG for N agents is given below

Algorithm 1: Multi-Agent Deep Deterministic Policy Gradient for N agents

for episode $= 1$ to M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state $\mathbf{x}$
  for $t = 1$ to max-episode-length **do**
    for each agent i, select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration
    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state $\mathbf{x}'$
    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    for agent $i = 1$ to N **do**
      Sample a random minibatch of S samples $(\mathbf{x}^j, a^j, r^j, \mathbf{x}'^j)$ from $\mathcal{D}$
      Set $y^j = r_i^j + \gamma\, Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)\big|_{a'_k = \boldsymbol{\mu}'_k(o_k^j)}$
      Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_N^j)\right)^2$
      Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i^j) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)\big|_{a_i = \boldsymbol{\mu}_i(o_i^j)}$$

    end for
    Update target network parameters for each agent i:

$$\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$$

  end for
end for

Network hyperparameters

Parameter and its value	Comments
BUFFER_SIZE = int(1e5)	# Replay buffer size
BATCH_SIZE = 256	# Minibatch size
GAMMA = 0.99	# Discount factor
TAU = 0.001	# For soft update of target parameters
LR_ACTOR = 0.0001	# Learning rate of the actor
LR_CRITIC = 0.0003	# Learning rate of the critic

MODEL SUMMARY

The architecture of the actor and critic neural network is given below

```
Actor(
  (fc1): Linear(in_features=24, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=2, bias=True)
  (bn1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
```

```
Critic(
  (fcs1): Linear(in_features=24, out_features=256, bias=True)
  (fc2): Linear(in_features=258, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=1, bias=True)
  (bn1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
```

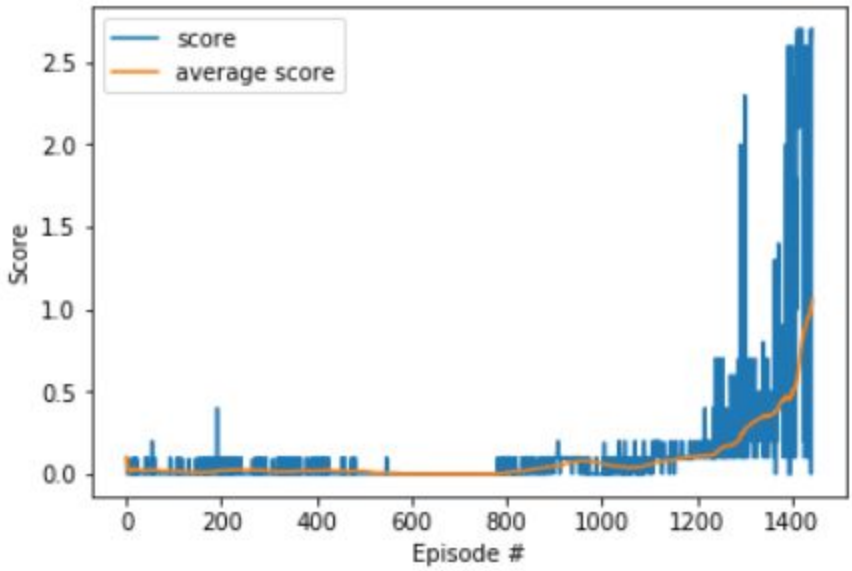
PLOT OF REWARDS

Below is the summary of the training with average score and episode count

```
0 episode       avg score 0.10000       max score 0.10000
500 episode     avg score 0.01560       max score 0.00000
1000 episode    avg score 0.06170       max score 0.00000
1398 episode    avg score 0.50360       max score 2.60000
Environment solved after 1398 episodes with the average score 0.5036000075750053

1441 episode    avg score 1.05770       max score 2.70000
```

A plot of the score Vs Episode is given below



The training ended when the target of an average score of 0.5 was reached as shown in the plot.

After training the weight of the networks are stored as

1. local_critic.pth
2. Local_actor.pth
3. target_critic.pth
4. target_actor.pth

These files are available as part of the repo

IDEAS FOR FUTURE WORK

So far the agent is trained only using MADDPG which can also be implemented using the Distributed Distributional Deterministic Policy Gradients(D4PG) algorithms

The performance can also be improved by usage of prioritized experience buffer instead of replay buffer, using other variations of noise and different model architecture.

Also in this attempt the agent interacted with the environment but the agent can be trained using raw pixels from the environment as input.

REFERENCE

[1] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, Igor Mordatch, Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, Proximal Policy Optimization Algorithms

[2] Juliani, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Mattar, M., Lange, D. (2018). Unity: A General Platform for Intelligent Agents. [arXiv preprint arXiv:1809.02627.] (https://github.com/Unity-Technologies/ml-agents)

[3] R. S. Sutton and A. G. Barto, Introduction to Reinforcement Learning, 2nd ed. Cambridge, MA, USA: MIT Press, 2017

[4] Deterministic Policy Gradient Algorithms, Silver et al. 2014

[5]Continuous Control With Deep Reinforcement Learning, Lillicrap et al. 2016