Collaboration and Competition

Nontawat Pattanajak
Deep Reinforcement Learning Nanodegree
16 January 2021

1. Introduction

The purpose of this project is to implement algorithm to control agents to play tennis. In this task, there are two agents which they locate in opposite side of each other. The rule of this task is each agent must hit the ball over the net. If the agent can do successfully, it will receive reward +0.1. On the other hand, if the agent cannot do (the ball falls to the ground), the agent will receive penalty (reward -0.01). The goal of this task is to keep the ball in play as long as possible, which the average score needs to exceed 0.5 for over 100 episodes.

This task can be classified as episode. Each episode will finish when the ball falls to the ground. There are 8 variables to consider as the state of environment which they are related to position and velocity of ball and rocket. Each agent will receive its own local states. There are two actions such as moving (forward or backward) and jumping. The action space can be considered as continuous action space. This is because the action value can vary (not discreet value).

2. Algorithm

The proposal algorithm is using Multi-Agent Deep Deterministic Policy Gradient (MADDPG) which is introduced by Ryan Lowe et al in 2017 [1]. MADDPG is developed from DDPG. It is improved by letting each agent learning the experiences by its own. However, they use the same model structure.

For original DDPG, it collects experiences from environment online and store the experiences to Reply Buffer. Then Reply Buffer generates batch data (in sampled uniformly random) to feed to Artificial Neural Networks which maps state to action. However, DDPG does not use argmax to provide a selected action, but it uses policy function. Therefore, this is the reason to use for continuous action space.

The policy of DDPG is deterministic. For the untrained policy, the action is not accurate enough and the policy will not explore on-policy. The agent might select the same action all the time. To solve this problem, Gaussian noise is injected to the action.

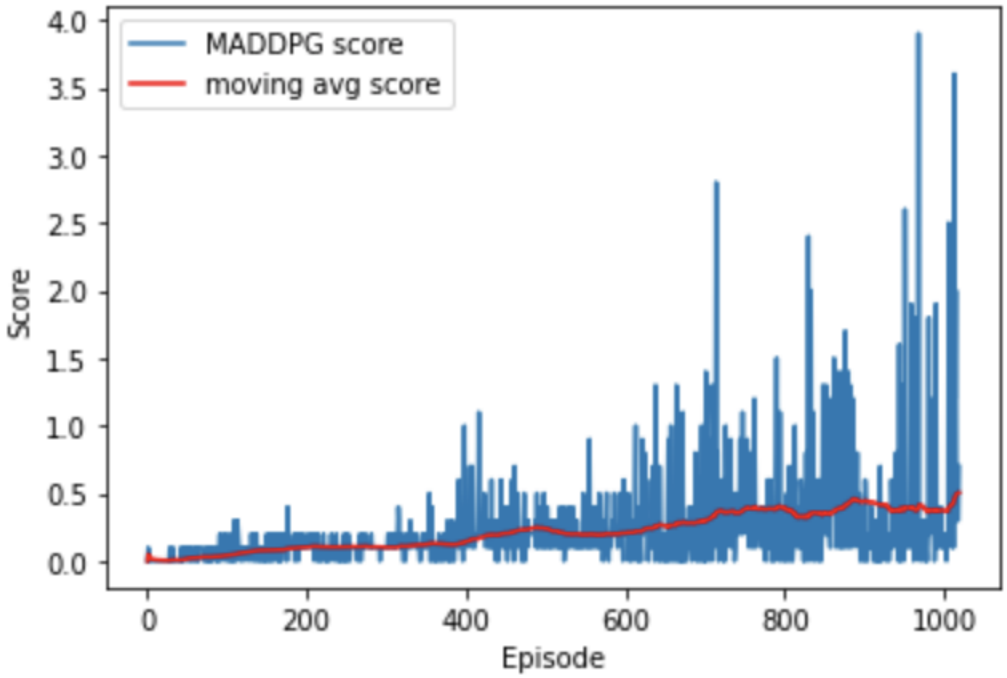
3. Model Structure and Hyperparameters

The neural network implemented in this task consists of two hidden layers. The first hidden layer consists of 512 nodes while the second hidden layer contains 128 nodes. The hyperparameters used in this work are shown below.

```
BUFFER_SIZE = int(1e6)     # replay buffer size
BATCH_SIZE = 128           # minibatch size
LR_ACTOR = 1e-3            # learning rate of the actor
LR_CRITIC = 1e-3           #learning rate of critic
WEIGHT_DECAY = 0           # L2 weight decay
LEARN_EVERY = 1            # learning teimstep
LEARN_NUM = 5              # number of learning passes
GAMMA = 0.99               # discount factor
TAU = 8e-3                 # for soft update of target parameters
OU_SIGMA = 0.2             # Ornstein-Uhlenbeck noise parameter, volatility
OU_THETA = 0.15            # Ornstein-Uhlenbeck noise parameter, speed of
mean reversion
EPS_START = 5.0            # initial value for epsilon in noise decay
process in Agent.act()
EPS_EP_END = 300           # episode to end the noise decay process
EPS_FINAL = 0              # final value for epsilon after decay
```

4. Result

The agents consume 1020 episodes to train to make average score 0.505. The result is shown below.



5. Conclusion and Future Work

The agents developed in this project are implemented from MADDPG. The architecture of algorithm consists of Artificial Neural Networks (ANN), Reply Buffer, Ornstein-Uhlenbeck Noise. The agents learn experience from environment and store in Buffer Reply. Then, Buffer Reply produces batch data and feeds to ANN. ANN learns the data and provides action. However, DDPG is a deterministic policy. In the early episode, the action provided by DDPG will not accurate enough. Adding Ornstein-Uhlenbeck Noise to the action can improve accuracy. For this task, each agent learns the experience be its own. Even though they use the same model architecture and algorithm, their trained parameters are different. This is because each agent learns from different experience.

The ANN structure two hidden layers. Each layer contains 512 nodes and 128 nodes respectively. The result of training the agents in this project found that the agents can achieve score more than 0.5 in 1020 episodes.

There are still many ways to improve the accuracy by adding some techniques such as adding prioritized experience replay, or by using different algorithms such as Trust Region Policy Optimization (TRPO), Proximal Policy Optimization (PPO), and Distributed Distributional Deterministic Policy Gradients (D4PG).

Reference

[1] Ryan Lowe et al (2017), “Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments”, https://arxiv.org/abs/1706.02275