Project: Collaboration and Competition

Sang-Ho Kim

This is the report submitted for the Udacity project "Collaboration and Competition". This project is to solve the Tennis environment using MA-DDPG (Multi Agent Deep Deterministic Policy Gradient). In Tennis 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 solution was started by modifying ddpg-pendulum solution in the Udacity GitHub repo of deep-reinforcement-learning: https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-pendulum. The class of multiagent was added with the functions of action and step, which call the ddpg agents and states of the two players. The number of agents is 2; the size of each action is 2; the number of states for each agent is 24.

Learning Algorithm

The solution was started by modifying ddpg-pendulum solution in the Udacity GitHub repo of deep-reinforcement-learning: https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-pendulum.

Critic model have 1 input layer, 1 hidden layer and 1 output layer. The number of nodes in the hidden layer were slightly changed from the original values. DDPG learns a Q-function using off policy data and Bellman equation and a policy using the Q-function.

In DDPG, the optimal action-value function defined by

$$Q^*(s,a) = \mathop{\mathrm{E}}_{s' \sim P}\left[r(s,a) + \gamma \max_{a'} Q^*(s',a')\right]$$

This is used to define the mean squared Bellman error (MSBE) function,

$$L(\phi,\mathcal{D}) = \mathop{\mathrm{E}}_{(s,a,r,s',d)\sim\mathcal{D}}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)\max_{a'} Q_\phi(s',a')\right)\right)^2\right]$$

DDPG is to minimize the MSBE loss function.

In the solution, the following hyper parameter values were used:

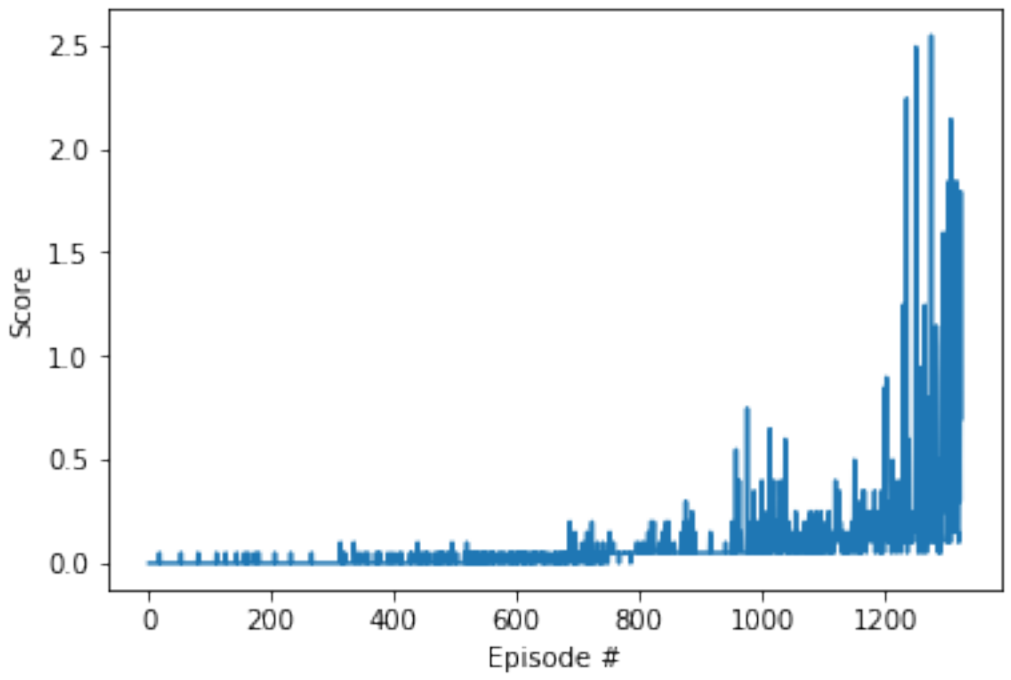
maximum time step for each episode = 2000
mu, theta, sigma in OUNoise = 0., 0.12, 0.2
BUFFER_SIZE = int(1e6) # replay buffer size
BATCH_SIZE = 512 # minibatch size
GAMMA = 0.99 # discount factor
TAU = 1e-1 # for soft update of target parameters
LR_ACTOR = 1e-4 # learning rate of the actor
LR_CRITIC = 3e-3 # learning rate of the critic
WEIGHT_DECAY = 0 # L2 weight decay

The deep learning model in both Actor and Critic contains input layer of (state size x 512 nodes), 1 hidden layer of (512 + action size) x 256, and 1 output layer (256 x 1). The first 2 layers in Actor contains the relu activation function and tanh in the output layer. The Critic takes state as the input, which is fed to the hidden layer and concatenated with action obtained from Actor as the input.

It should be noted that TAU in the soft update was taken large (e.g., 0.1) while it was 1e-3 in the previous project, continuous control. This means it is now closer to the hard update than the soft update. With a lower TAU, the model did not learn much. It is believed that the hard update is more effective for this problem, where hitting a ball conditions (or states) are rather of small variation in terms of ball incoming angle, velocity, etc., except the dropping positions.

Plot of Rewards

The following rewards plot was obtained. In the beginning, the model did not learn much until about 800 episodes and it suddenly increased above the score of 0.3. Although the average score hit above 0.5 at the end, the score fluctuates up and down showing unstable performance. A further study on the hyper parameters would have improved the performance.



Ideas for Future Work

The better result would be obtained with playing around with the hyper parameters tuning. A further study is desirable.