Udacity Reinforcement Learning Nanodegree Project 3 - Collaboration and Competition

Jayanth Nair, October 2020

Project Details:

Objective: Train two competing agents to play a game of tennis against each other in the Tennis Unity ML environment using Deep Reinforcement Learning using Deep Reinforcement Learning

Reward System:

The reward system is as follows

- +0.1 if an agent hits the ball over the net
- -0.01 if an agent lets a ball hit the ground or hits the ball out of bounds

State Space:

- 24 variables for each agent corresponding to position and velocity of the ball and racket
- Each agent receives its own local observation.

Action Space:

2 continuous actions corresponding to movement toward (or away from) the net and jumping

Solution Criterion:

The task is considered solved when the maximum averaged score between the 2 agents is +0.5 over 100 episodes after taking the maximum over both agents

Learning Algorithm(s)

The Deep Deterministic Policy Gradient (DDPG) and a modification of the Multi Agent DDPG algorithm (MADDPG) are used for this project

1) ***DDPG Algorithm:***

   The DDPG algorithm consists of two separate networks, namely the actor and the critic. The actor converts a state into an action. This mapping will be deterministic. Given a state and action, the critic will provide a Q-value.

   "The gradients are calculated from Q which use actions produced by the actor, so the whole system is differentiable and could be optimized end to end with stochastic gradient descent (SGD)" - Maxim Lapan, Deep Reinforcement Learning Hands-on

Since the policy learnt is deterministic, in order to encourage exploration, noise is added to action selection using the Ornstein-Uhlenbeck process. To improve stability, a large replay buffer is used along with target networks for both actor and critic.

DDPG Network Architecture:

The actor network consists of 3 fully connected layers. The input layer consists of 128 units, the hidden layer consists of 64 units and the output layer consists of 4 units corresponding to the action_size parameter. The ReLU activation function is used for the input and hidden layer. The tanh activation function is used on the output layer -> clips actions between -1 and 1

The critic network also consists of 3 fully connected layers with a similar number of units as the actor network. However, in the hidden layer, an additional 4 units are connected to link the actions. The activation functions are the same as the actor network.

The weights for the actor and critic network are initialized as per Lillicrap et al. (2016)

Hyper-parameters:

- Replay Buffer Size - 10^6
- Batch Size - 512
- Discount Factor (Gamma) - 0.99
- Soft Update Annealing Factor (Tau) - 0.001
- Learning Rate for Actor - 0.0001
- Learning Rate for Critic - 0.0001
- L2 Weight Decay for Critic Optimizer - 0
- Time Steps Before Update - 20
- Number of Updates - 10

For this implementation both agents share the same actor and critic networks and a common experience replay buffer containing experiences of both agents.

2) ***MADDPG Algorithm:***

The implementation of MADDPG algorithm for this project is essentially the same as the DDPG algorithm, but is modified so that each competing agent uses its own actor and critic networks. Each agent also uses its own experience replay buffer and learns solely from its own experiences. This is a modification of the original MADDPG algorithm where the critic for each agent has access to extra information (such as the actions of all other

agents) during training. The critic networks in this implementation does not have access to this extra information.

The same hyper-parameters were used as the DDPG implementation

Results

The two plots below show the performance of both algorithms. The DDPG algorithm solved the environment in 3636 episodes while the MADDPG implementation solved the environment in 440 episodes.

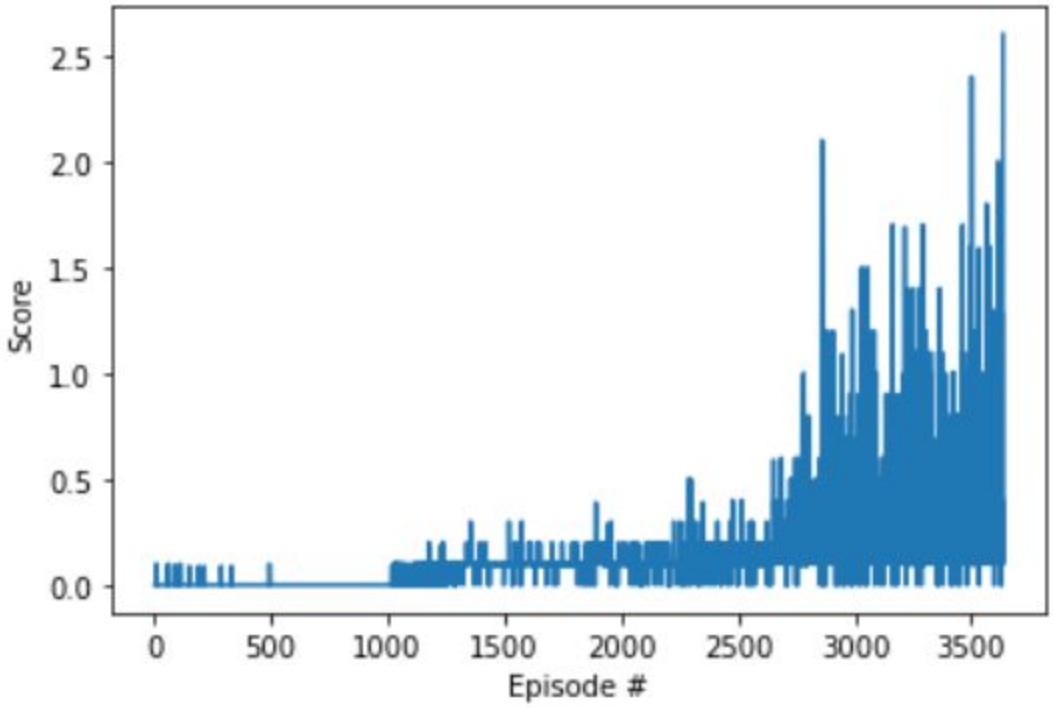

Figure 1 : DDPG Algorithm Performance

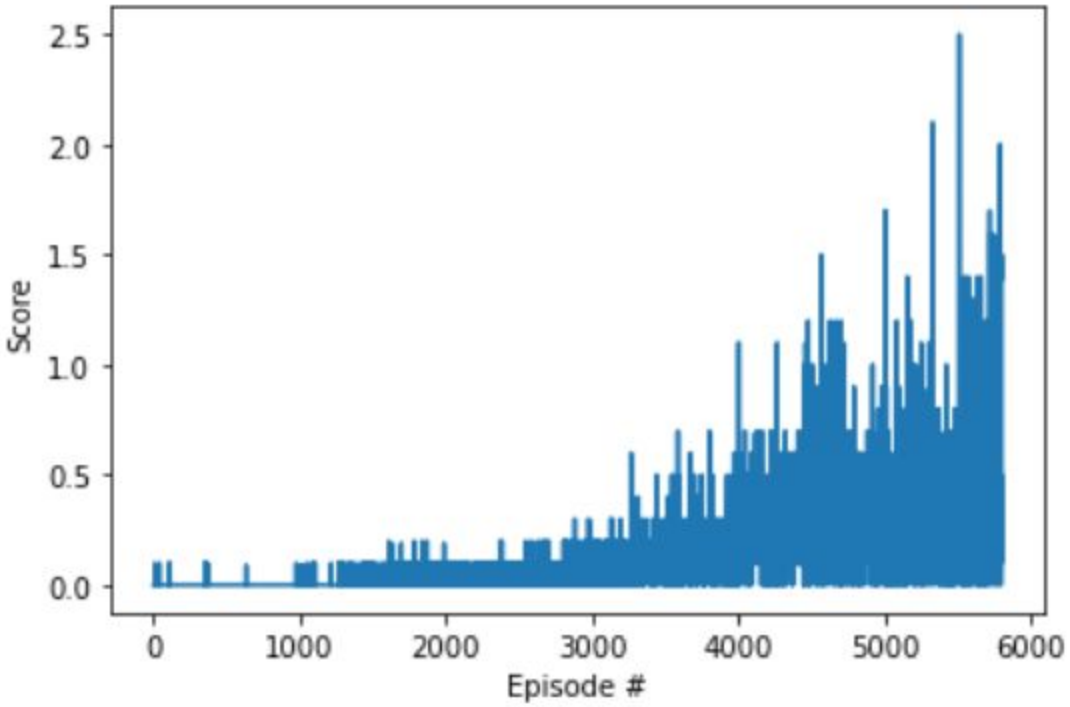

Figure 2 : MADDPG Algorithm Performance

Lessons Learnt, Next Steps and Room for Improvement

Both implementations require a large number of episodes to make an meaningful progress. My original hypothesis was that the MADDPG algorithm would make the agents more robust as two separate actor-critic networks will be training against each other. However, it seems for the set of hyper-parameters I've selected, using two separate networks did not offer an advantage. It is possible that with the critic accessing the actions of all agents, the training might be faster. The next steps I see as helpful are:

1. Modifying critic to access actions of all agents during training
2. Hyper-parameter optimization
3. D4PG implementation

References:

1. Udacity Course Material, Deep Reinforcement Learning Nanodegree
2. Deep Reinforcement Learning Hands-on by Maxim Lapan