Learning Algorithm

I used Multi Agent Deep Deterministic Policy Gradient agent for this environment because multiple agent exist in the environment and it can be used for continuous action space, we can apply some of the DQN techniques to improve performance. Local and target networks have been used for avoiding over-estimation. Also, e-greedy Ornstain-Uhlenbeck noised action selection and experience replay is used for proper exploration. MADDPG agent is updated every 4 episodes. At every 4 episodes, agent updates its networks 3 times with different experience samples. This increased chance of getting rewarded states to the learning batch which increased learning.
Shared experience replay is used for critic networks but every actor network uses its own observations from the same buffer to update itself.

Hyperparameters

I tried several combinations of hyperparameters as;
Actor Learning Rate: [0.0001, 0.0005, 0.001, 0.005]
Critic Learning Rate: [0.0001, 0.0005, 0.001, 0.005]
Epsilon: [0.3, 1]
Epsilon Decay: [1, 0.999]
Minimum Epsilon : [0.01, 0.02]
Batch Size : [200, 256, 500, 1024]
Gamma : [0.99, 0.995]
TAU : [0.001]
Buffer Size : 100000
Weight Decay : 0
Update Every N Episodes: [1, 2, 4]
Update Replay: [1, 3]

Most successful parameters are;

Actor Learning Rate: 0.0005
Critic Learning Rate: 0.001
Epsilon: 1
Epsilon Decay: 1
Minimum Epsilon : 0.01
Batch Size : 200
Gamma : 0.995
TAU : 0.001
Buffer Size = 100000
Weight Decay = 0
Update Every N Episodes: 4
Update Replay: 3

Model Structure

DDPG model structure shown below;

Actor Network has;
- Fully Connected Layer (state_size, 256)
- ReLU
- Batch Normalization
- Fully Connected Layer (256, 128)
- ReLU
- Fully Connected Layer (128, 128)
- ReLU
- Fully Connected Layer (128, action_size)
- Tanh

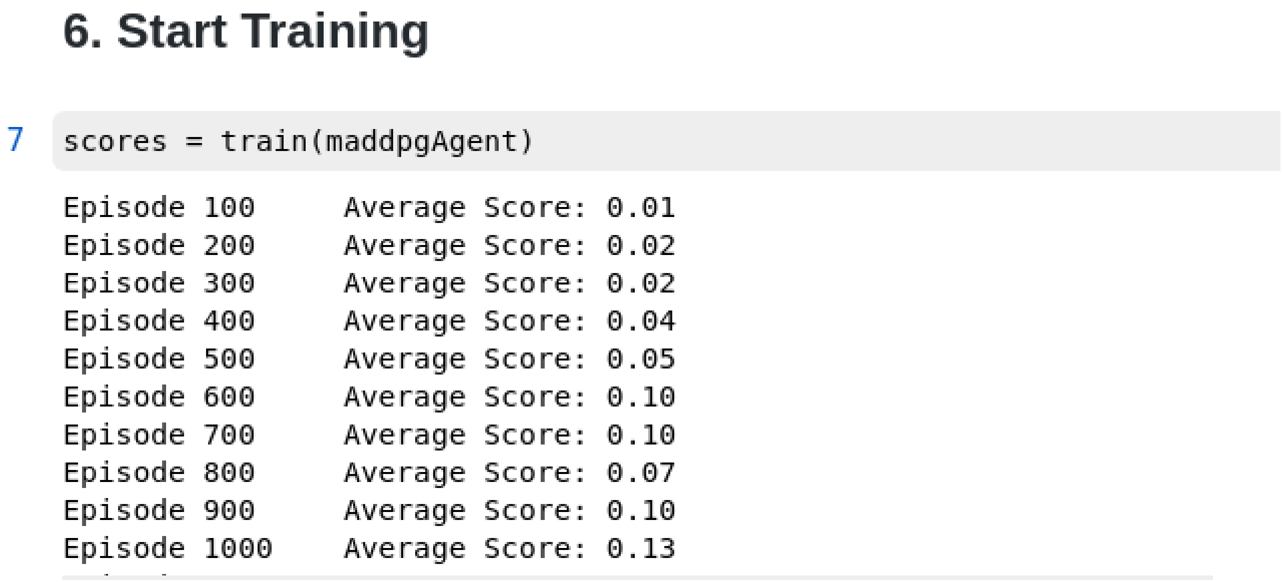
Critic Network has;

- Fully Connected Layer (state_size + action_size, 300)
- ReLU
- Batch Normalization
- Fully Connected Layer (300, 400)
- ReLU
- Fully Connected Layer (400, 1)

In the beginning, I tried smaller sized networks but learning is not sufficient. Also, actor critic network had 3 FC layer. Then I increased the sizes of the networks both as hidden layer size and hidden layer number. After some point, agent started to slow down for learning. I figured out the boundaries of network structure.
I used Adam optimizer for both actor and critic networks.
MSE loss function is used for critic model.
Min(-Q value from critic with the action that is generated from actor model) loss function is used for actor model.
Gradient clipping is used for both critic and actor models training in order to stabilize learning.

Plots

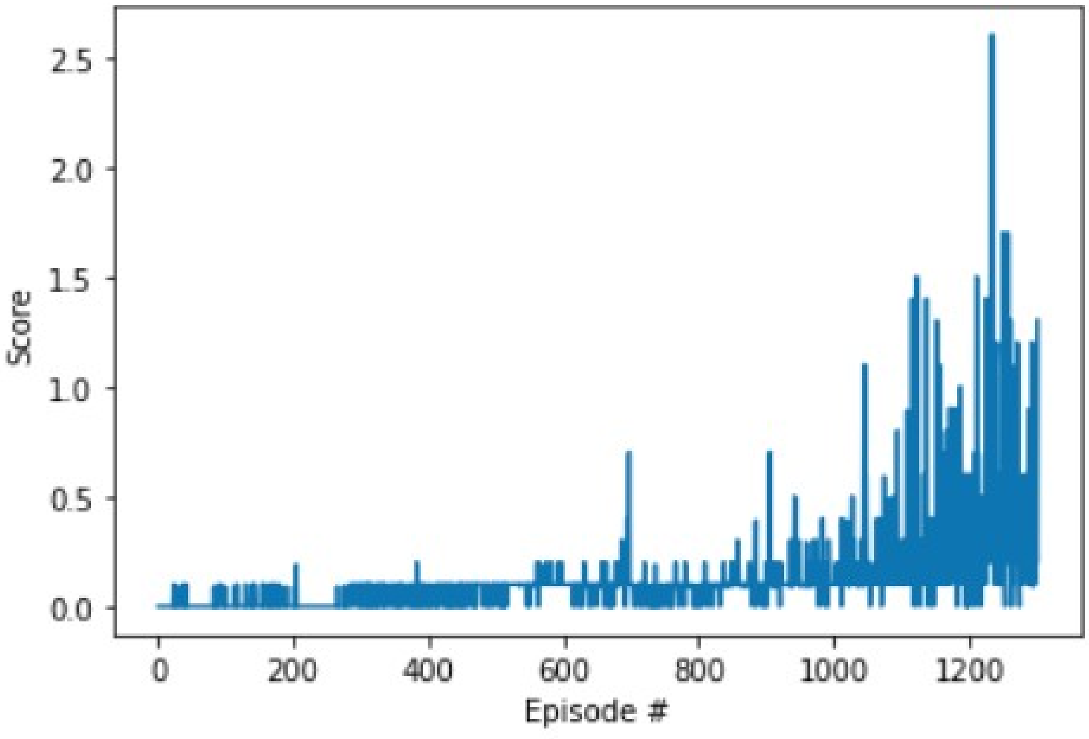
Training process result:



6. Start Training

```
7   scores = train(maddpgAgent)
```

```
Episode 100     Average Score: 0.01
Episode 200     Average Score: 0.02
Episode 300     Average Score: 0.02
Episode 400     Average Score: 0.04
Episode 500     Average Score: 0.05
Episode 600     Average Score: 0.10
Episode 700     Average Score: 0.10
Episode 800     Average Score: 0.07
Episode 900     Average Score: 0.10
Episode 1000    Average Score: 0.13
```



Future Work

Rewards are very rare at the beginning of the training. Therefore, states that have rewards cannot be included to the randomly sampled batch very frequently. That makes training slower. I tried to exclude some of the 0-reward states (%33 - %20) in order to increase the possibility of having non-zero-reward states. However, that does not worked very well because some of the zero-states are also important. Prioritized Experience Replay with Sum Tree may increase learning speed, especially in the early-mid episodes.
Also I can try other extensions of DDPG like TD3. It has been reported that TD3 can give really good results.
Distributed learning for pair of agents can be a fast way to train agents.