Introduction

We tried to solve the tennis environment in project 3 of Deep Reinforcement learning course. In this environment, two agents are trained to use 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.

Algorithm

Multi-Agent Deep Deterministic Policy Gradients (MADDPG) is used to solve this environment since this is has a continuous action space. At a high level DDPG algorithm can be broken into: Experience replay,

Actor and critic network updates, target network updates and exploration

Experience replay: Since this activity is sequential but to optimize, we need independently distributed data. So, we store them in a replay buffer and take random batches for training. the sample experience is used to update neural network parameters. we save all the experience tuples (state, action, reward, next state) and store them during each trajectory the next-state Q values are calculated with the target value network and target policy network. Then, we minimize the mean-squared loss between the updated Q value and the original Q value

Actor and critic network updates: the target value network and target policy network as used to update the next state values. we try to minimize the mean-squared loss between the updated Q value and the original Q value

Target Network Updates: We make a copy of the target network parameters and have them slowly track those of the learned networks via soft updates Exploration: since the problem is in continuous action spaces, exploration is done by adding noise to the action. In the DDPG paper mentioned in the reference, the authors use Ornstein-Uhlenbeck Process to add noise to the action output

On top of the DDPG algorithm, multi agents are initiated to fill the replay buffer with experiences periodically. Multi agents are preferred in complex tasks to gain knowledge and learn faster using collaboration or competition. Also, the environment becomes dynamic in this case

Network Architecture

The network architecture for actor and critic has two fully connected hidden layers of 256 and 128 units. Adam was used as an optimizer for both actor and critic networks in the DDPG pendulum code. Both have ReLU activations and batch normalization on the hidden layers to avoid getting stuck in local minimum.

Hyperparameters

I started my work by fixing the network architecture based on my learnings from project 2, also introduced gradient clipping for critic learning to add stability to the algorithm. The spent time on tweaking the hyper parameters to achieve the objective

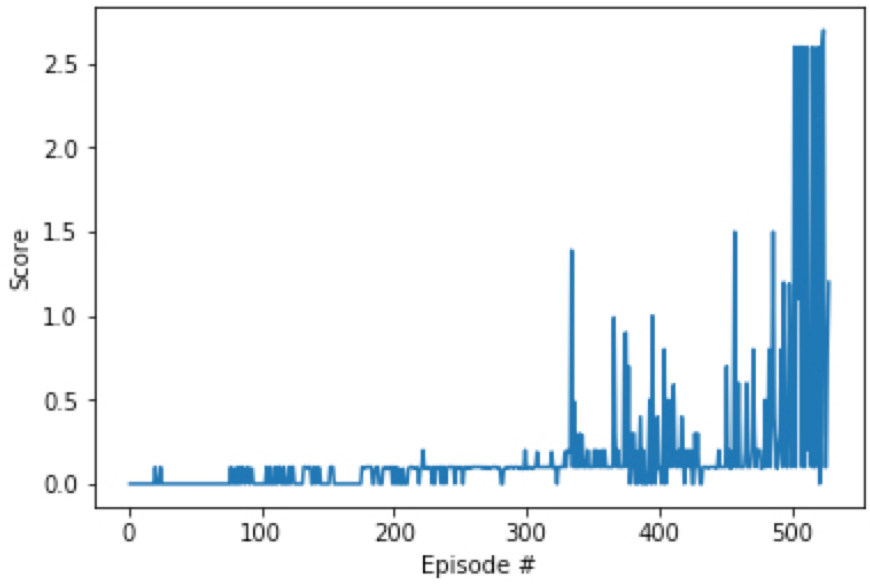
```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 256        # minibatch size
GAMMA = 0.990           # discount factor
TAU = 1e-2              # for soft update of target parameters
LR_ACTOR = 1e-3         # learning rate of the actor
LR_CRITIC = 3e-3        # learning rate of the critic
WEIGHT_DECAY = 0        # L2 weight decay
LEARNING_STEPS = 5.   # periodic updates
LEARNING_ITER = 10.    # number of updates at once
```

As can be seen I used the default buffer size, doubled the batch sized to accommodate 2 agents. used the default GAMMA , I reduced the TAU as it gave me better performance when I tried to increase the critic learning rate to 3 times that of the actor since the actor uses the information from the critic to change its policy

Plot:

As can be seen, this problem is solved in 528 episodes with an average score of 0.51

```
Episode 520     Average Score: 0.44
Episode 528     Average Score: 0.51
Environment solved in 528 episodes!     Average Score: 0.51
```



Future Improvements

I would try work on this problem with prioritized buffer to see the improvements compared to replay buffer. I would also like to try the problem with recommended such as TRPO, TNPG and own version of PPO