MA-DDPG Agent for Collaboration and Competition:

Learning Algorithm:

I have solved Unity's Tennis environment by building and training a Multi Agent Deep Deterministic Policy Gradient (DDPG) agent.

Model Architecture:

Actor:

Feedforward network with 2 hidden layers. First hidden layer has 200 units. Second hidden layer has 150 units. Both Hidden layers have used ReLu activation function. Output layer uses a Tanh activation function.

Critic:

Feedforward network with 2 hidden layers. First hidden layer has 200 units. Second hidden layer has 150 units. Both Hidden layers have used ReLu activation function. Output layer doesn't use any activation function.

DDPG Agent:

Agent interacts with the environment using current policy of actor_local and learns using Adam optimizer. DDPG agent comprises of both the Actor Network and Critic Network defined above which hold the learnable parameters.

This agent is capable of taking action in the environment using current policy of the Actor along with some noise. The action is clipped to be between -1 and 1. Once the agent takes an action in a given state - environment returns reward and the next_state for that time step. This experience tuple of (state, action, reward, next_state) is saved in Memory/Replay Buffer. These experiences are then sampled from the Memory for the learning step. This learning only happens once the available experiences in the memory are more the Batch size.

In the learning step - states, actions, rewards and next_states are first unpacked from the experiences tuple where we first update critic and then we update actor and then update target networks.
To update critic -> first next_states are used in actor_target to produce next actions. And then next_states and next actions are used to compute Q_targets_next using the critic_target network. Then Q_targets are computed using rewards + (gamma * Q_targtes_next) update.

Tehn Q_exptecd are calculated using critic local network. Now using Q_expected and Q_targets we calculate the critic loss. Using this loss we optimize critic network.
To update actor -> First actor network is used to compute actions_predicted in the current states. Then we take the key step where we equate loss of the actor to be equal to the negative of value computed from the citic network in the current states and for actions predicted by the actor network. With this loss we update/optimize the actor network.

To update target networks -> Target networks of both critic and actor are updated using a soft_update method. As part of this soft update - parameters of local actor and critic networks are not directly copied to the target networks but they are soft updated using a tau interpolation parameter.

Multi Agent DDPG:

Uses MA-DDPG with centralized training and decentralized. 2 Agents share a common replay buffer. Each agent modelled as a DDPG(described above) with some information being shared between the agents. Each agent has its own actor and critic. Each actor of each agent receives as input the individual state observations of the agent and outputs a two-dimensional action. The critic model of each agent however receives the states and actions of both actors concatenated. This allows information sharing between the two agents.

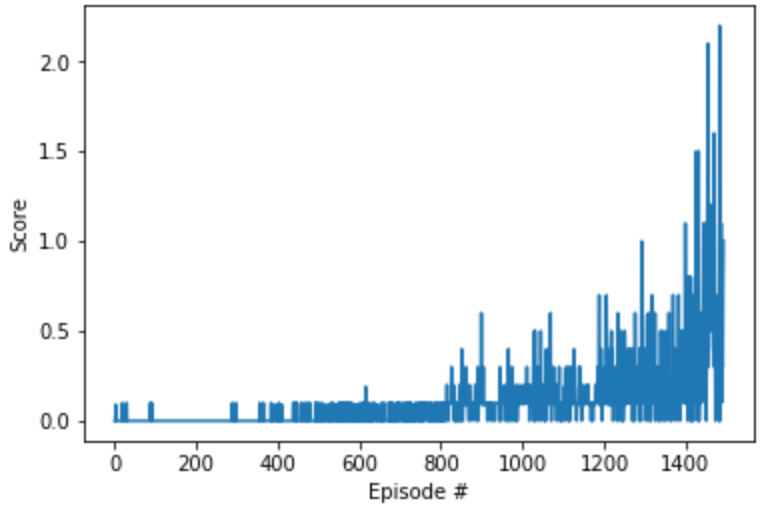
Below are the chosen hyperparameters
```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 250        # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR_ACTOR = 1e-4         # learning rate of the actor
LR_CRITIC = 1e-3        # learning rate of the critic
WEIGHT_DECAY = 0        # L2 weight decay
```

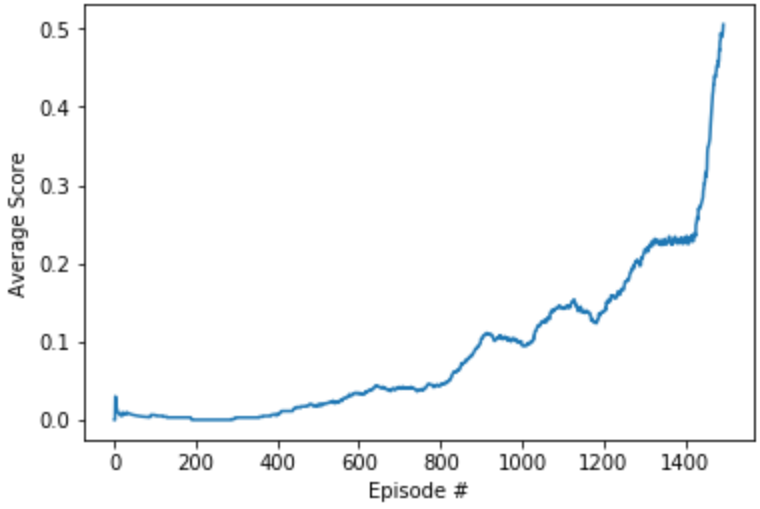
Training:

Agent is trained in episodes and there are steps in each episode. Below are the chosen parameters for training:
n_episodes=5000

Plot of Rewards:

Environment solved in 1493 Episodes! Plots of rewards below.





Ideas for future Work:

We have used a Multi Agent DDPG agent to learn to solve the Tennis environment. In future we can explore using Multi Agent PPO and Multi Agent DQN architecture to see if it can improve the performance of the RL agent further.

Hyperparameter search methods like Bayesian optimization can also be used to further fine tune hyperparamaters which can also improve RL agent's performance.

Batch normalization and Prioritized experience replay can be tried as well to further improve performance.