P3 Collaboration and Competition

For this project, the provided solution is an adaptation of Multi-Agent Deep Deterministic Policy Gradients (MADDPG) algorithm[1]

Goal : In this project, Multi-Agent Deep Deterministic Policy Gradient (DDPG) model was used to solve the Unity Tennis environment. The two agents in the environment control rackets to bounce a ball over a net.

Reward: A reward of +0.1 is provided for each step that the the agent hits the ball over the net. If the agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01.

Environment details: The observation space consists of 8 variables corresponding to the position and velocity of the ball and racket. Each agent receives its own, local observation. Two continuous actions are available, corresponding to movement toward (or away from) the net, and jumping. To solve the environment, the agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents)

Learning Algorithm:

The learning algorithm for the Tennis environment is implemented in *Tennis.ipynb* notebook. MADDPG algorithm implemented in this project uses an Actor-Critic model which can learn policies in high-dimensional, continuous action space. The multi-agent algorithm is an extension to actor-critic model, where the critic is augmented with extra-information about the policies of other agent.

As explained in [1], the model maintains a parameterized actor function $\mu(s|\theta\mu)$ which specifies the current policy by mapping states to a specific action. The critic $Q(s, a; \theta)$ is learned using the Bellman equation as in Q-learning and is used to evaluate the actions chosen by the actor. The resulting evaluation is used to compose a baseline (or advantage function) that is then used to train the actor model.

In the Navigation project, DQN algorithm allows the agent to maintain a trade off between exploration and exploitation. The agent "explores" by picking a random action with some epsilon `$\epsilon$` probability. Meanwhile, the agent continues to "exploit" its knowledge of the environment by choosing actions based on the deterministic policy with probability $(1-\epsilon)$.However, this apporach wont work for controlling the tennis agent. As the action space is not discrete and movement of the arm is completely continuous driving the arm to various degrees of angles and position.

Hyperparameters:

As mentioned in [4], the Ornstein-Uhlenbeck process generates temporally correlated exploration for exploration efficiency in physical control problems. The Ornstein-Uhlenbeck process itself has three hyperparameters that determine the noise characteristics and magnitude: (mu: the long-running mean, theta: the speed of mean reversion, sigma: the volatility parameter).Fine tuning these parameters resulted in efficient learning process for the agent.

Experience Replay:

Experiences are stored in a single replay buffer as each agent interacts with the environment. These experiences are then sampled by the central critic, therefore allowing the agents to learn from each others' experiences. The replay buffer contains a collection of experience tuples with the state, action, reward, and next state `(s, a, r, s')`. The critic samples from this buffer as part of the learning step. Experiences are sampled randomly, so that the data is uncorrelated. This prevents action values from oscillating or diverging catastrophically, since a naive algorithm could otherwise become biased by correlations between sequential experience tuples.

Also, experience replay improves learning through repetition. By doing multiple passes over the data, our agents have multiple opportunities to learn from a single experience tuple. This is particularly useful for state-action pairs that occur infrequently within the environment.

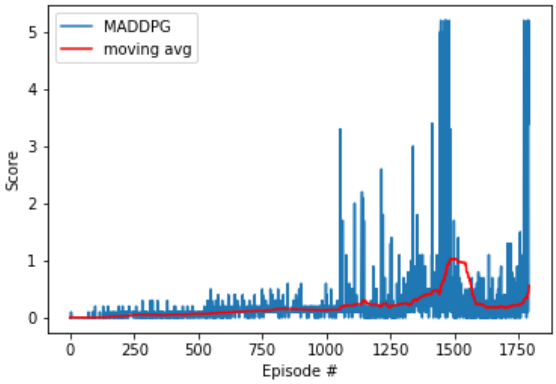
Plot:

The agent was able to solve the environment by achieving score average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents). There is a drop in reward after acheiving the expected reward when the agent was trained in GPU with Ornstein-Uhlenbeck- speed of mean reversion, theta-value of 0.20. However, when the same agent was trained in CPU with with Ornstein-Uhlenbeck process theta value of 0.05) the agent performance was stabilised and provide more consistent rewards.

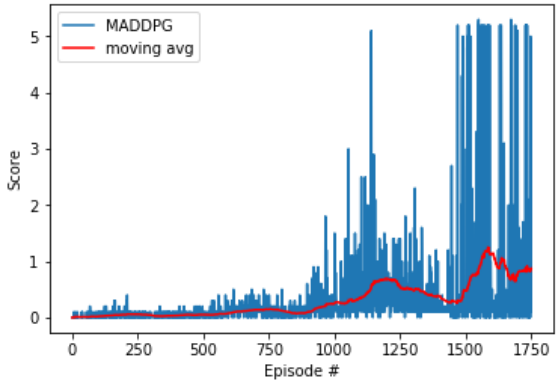
Plot of rewards can be seen after the environment has been solved.

Tuning Ornstein-Uhlenbeck process:

Experiment 1: (Theta:0.20 , GPU)



Experiment 2: (Theta:0.05 , CPU)



Results:

Once all of the above components were in place, the agents were able to solve the Tennis environment. Again, the performance goal is an average reward of at least +0.5 over 100 episodes, taking the best score from either agent for a given episode.

```
<-- Best episode so far!
Episode 1551    Max Reward: 5.300       Moving Average Score: 0.871
Episodes 1560   Max Reward: 5.300       Moving Average Score: 0.975
Episodes 1570   Max Reward: 5.200       Moving Average Score: 1.120
Episodes 1580   Max Reward: 5.200       Moving Average Score: 1.177
Episodes 1590   Max Reward: 5.200       Moving Average Score: 1.227
```

```
Episodes 1600   Max Reward: 5.200       Moving Average Score: 1.131
Episodes 1610   Max Reward: 0.500       Moving Average Score: 1.097
Episodes 1620   Max Reward: 0.400       Moving Average Score: 0.917
Episodes 1630   Max Reward: 1.200       Moving Average Score: 0.873
Episodes 1640   Max Reward: 5.200       Moving Average Score: 1.059
Episodes 1650   Max Reward: 3.100       Moving Average Score: 0.972
Episodes 1660   Max Reward: 0.500       Moving Average Score: 0.831
Episodes 1670   Max Reward: 2.000       Moving Average Score: 0.699
<-- Best episode so far!
Episode 1675    Max Reward: 5.300       Moving Average Score: 0.700
Episodes 1680   Max Reward: 5.300       Moving Average Score: 0.752
Episodes 1690   Max Reward: 2.000       Moving Average Score: 0.647
Episodes 1700   Max Reward: 5.200       Moving Average Score: 0.790
Episodes 1710   Max Reward: 1.600       Moving Average Score: 0.821
Episodes 1720   Max Reward: 1.400       Moving Average Score: 0.824
Episodes 1730   Max Reward: 5.200       Moving Average Score: 0.879
Episodes 1740   Max Reward: 5.200       Moving Average Score: 0.831
Episodes 1750   Max Reward: 5.000       Moving Average Score: 0.871
<-- Training stopped. Best score acheived in less than 200 episodes
```

Ideas for Future Work

- Fine tuning the optimal values for hyperparameters using grid search, beam search or bayesian optimization or genetic algorithm.
- Better control of the exploration-exploitation trade-off by implementing a decreasing schedule for the sigma parameter of the Ornstein-Uhlenbeck process .

- Prioritized experience replay : As mentioned in the paper [2], When using experience replay separates online learning agents and help the agent to process the state transitions in the exact order they are learned. While the prioritized replay further liberates agents from considering the state transitions with random order and experiences are learned by the magnitude of their temporal-differences error.

References

[1] Lowe, Ryan, et al. Multi-agent actor-critic for mixed cooperative-competitive environments
[2] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., … Wierstra, D. (2016). Continuous control with deep reinforcement learning. 4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings.
[3] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., & Mordatch, I. (2017). Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. 4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings. Retrieved from http://arxiv.org/abs/1706.02275