# Project-3 : Deep Reinforcement learning - Collaboration and Competition

## Background

Objective of the project is to train two agents to play tennis using Deep Deterministic Policy Gradients. The environment has 2 agents that control rackets to bounce balls over the 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.

| **States** | **Reward** |
|---|---|
| Hit Over the Net | +0.1 |
| Ball Hits the Ground | -0.01 |

The observation space has 8 variable correspondings to position and velocity of the ball and racket. Each agent receives its local observation

There are two continuous actions available: 1) moving away and towards the net, 2) jumping.

The task is episodic, and the environment is solved when the agents get an average score of +0.50 over 100 consecutive episodes).The score of an episode is defined to be the maximum score over the two agents on the episode.

Algorithm used in the project is an *off-policy method* called **Multi Agent Deep Deterministic Policy Gradient (MADDPG)** algorithm.

## Model Architecture

The code is written in Python 3.6 and is relying on PyTorch 0.4.0 framework.
The project implemented an Actor-critic method to leverage the strengths of both policy-based and value-based methods. The code consist of :

- Model.py : Implement the **Actor** and the **Critic** classes,each implement a *Target* and a *Local* Neural Networks used for training.
- Agent.py: Implement the MADDPG algorithm uses a **decentralized actor with centralized critic** approach

## Hyperparameters

```
BUFFER_SIZE = int(1e6)  # replay buffer size
BATCH_SIZE = 128        # minibatch size
LR_ACTOR = 1e-3         # learning rate of the actor
LR_CRITIC = 1e-3        # learning rate of the critic
WEIGHT_DECAY = 0        # L2 weight decay
LEARN_EVERY = 5         # learning timestep interval
LEARN_NUM = 5           # number of learning passes
```

GAMMA = 0.99 # discount factor
TAU = 7e-2 # for soft update of target parameters
OU_SIGMA = 0.2 # Ornstein-Uhlenbeck noise parameter, volatility
OU_THETA = 0.12 # Ornstein-Uhlenbeck noise parameter, speed of mean reversion
EPS_START = 5.5 # initial value for epsilon in noise decay process in Agent.act()
EPS_EP_END = 250 # episode to end the noise decay process
EPS_FINAL = 0 # final value for epsilon after decay

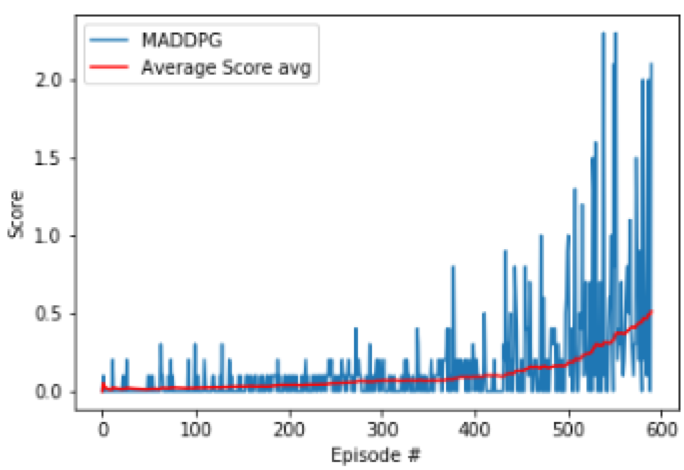
Noise parameters used in the model is as follows:
OU_SIGMA = 0.2 # Ornstein-Uhlenbeck noise parameter, volatility
OU_THETA = 0.11 # Ornstein-Uhlenbeck noise parameter, speed of mean reversion

Decay parameters used in the model is as follows:
EPS_START = 5.5 # initial value for epsilon in noise decay process in Agent.act()
EPS_EP_END = 250 # episode to end the noise decay process
EPS_FINAL = 0 # final value for epsilon after decay

## Training Results:

The figure below show the performance and results of the model. The model performs well. The agent takes 490 episodes to achieve the goal.

```
Episodes 0490-0500      Highest Reward: 0.900   Lowest Reward: 0.000    Average Score: 0.175
Episodes 0500-0510      Highest Reward: 1.300   Lowest Reward: 0.000    Average Score: 0.206
Episodes 0510-0520      Highest Reward: 1.200   Lowest Reward: 0.000    Average Score: 0.238
Episodes 0520-0530      Highest Reward: 1.600   Lowest Reward: 0.000    Average Score: 0.294
Episodes 0530-0540      Highest Reward: 2.300   Lowest Reward: 0.000    Average Score: 0.310
Episodes 0540-0550      Highest Reward: 2.100   Lowest Reward: 0.000    Average Score: 0.340
Episodes 0550-0560      Highest Reward: 2.300   Lowest Reward: 0.100    Average Score: 0.367
Episodes 0560-0570      Highest Reward: 1.100   Lowest Reward: 0.200    Average Score: 0.410
Episodes 0570-0580      Highest Reward: 1.500   Lowest Reward: 0.000    Average Score: 0.441
Episodes 0580-0590      Highest Reward: 2.100   Lowest Reward: 0.000    Average Score: 0.511
<-- Environment solved in 490 episodes!
<-- Average Score: 0.511 over past 100 episodes
```



## Future Improvements

There could be further improvement in applied Multi Agent Deep Deterministic Policy Gradient (MADDPG) algorithm known as Batch. The Google DeepMind paper talks about the benefits of using this approach.

- Similar to the exploding gradient, running computations on large input values makes learning inefficient. Batch normalization addresses this problem by scaling the features to be within the same range throughout the model and across different environments and units. The range of values is often much smaller, typically between 0 and 1.