# Deep Reinforcement Learning - Collaboration and Competition

## AIM

To train a multi-agent scenario where two agents are interacting with each other and learning from their experiences. The score measured here at the end of each episode is the score of the agent who has scored higher.

The environment provided is that of a Unity ML Agents scenario.

In this environment, two agents control 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.

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.

## Implementation

### Algorithm

DDPG(Deep Deterministic Policy Gradient) Algorithm.

The base code of the project is derived from the solution provided in Udacity Deep Learning Nanodegree Github repo for solving the pendulum scenario from OpenAI Gym.
https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-pendulum

The same solution was modified and updated for the Unity ML Agents environment provided.

- **Tennis.ipynb** : contains the implementation for training and visualising the untrained agent. Then the training code is implemented.
- **ddpg_agent.py :** Contains the code to understand and determine how the agent interacts with the environment and learns to optimize the reward.
- **model.py :** Contains the architecture of the deep learning model used in this implementation.

## Algorithm

### Deep Deterministic Policy Gradient (DDPG)

The algorithm is outlined in this paper, Continuous Control with Deep Reinforcement Learning. In this paper, the authors present "a model-free, off-policy actor-critic algorithm using deep function approximators that can learn policies in high-dimensional, continuous action spaces." They highlight that DDPG can be viewed as an extension of Deep Q-learning to continuous tasks.

### Actor-Critic Method

Actor-critic methods leverage the strengths of both policy-based and value-based methods.

Using a policy-based approach, the agent (actor) learns how to act by directly estimating the optimal policy and maximizing reward through gradient ascent. Meanwhile, employing a value-based approach, the agent (critic) learns how to estimate the value (i.e., the future cumulative reward) of different state-action pairs. Actor-critic methods combine these two approaches in order to accelerate the learning process. Actor-critic agents are also more stable than value-based agents, while requiring fewer training samples than policy-based agents.

It also utilizes the replay buffer to learn from past experiences in the way DQN Algorithm does

## Agent

Deep Q-Learning agent which interacts with the environment. It references the local and target network from the model defined in model.py.

**Step**: Experiences in the replay memory. After a certain set of predefined intervals, it also causes the network to learn from the replay buffer a certain number of times.

**Act**: Returns the action determined by the local Actor Network. The output is of size 4 corresponding to each actions but in the range of -1 to 1 as expected by the network. Furthermore, noise is added via the Ornstein-Uhlenbeck process to encourage exploration.

**Learn**: Learns

**Critic network learning :** We randomly sample a batch from the experience buffer in the form of (states, actions, rewards, next_states, dones) and pass on the next_states to the actor target network to determine the next set of actions which in return is passed on to the critic target network.

The return from the critic target network is actually the Q values determined by the target network.

The expected Q values are computed from these next states and compute the Mean Squared Error loss between Q_targets and Q_expected and update the critic target network accordingly.

## ACTOR NETWORK LEARNING

Here we get the predicted actions from the local actor network based on the current states. The loss is computed as the mean of the Q values corresponding to the different state action pairs. We use the negative sign here because we want to maximise the gradients and hence using gradient ascent.

Then we run the soft_update function to update the target network with the local network parameters

**Soft_update**: Here we update the target networks with the local networks parameters using the formula

$$\theta\_target = \tau*\theta\_local + (1 - \tau)*\theta\_target$$

## Hyperparameters

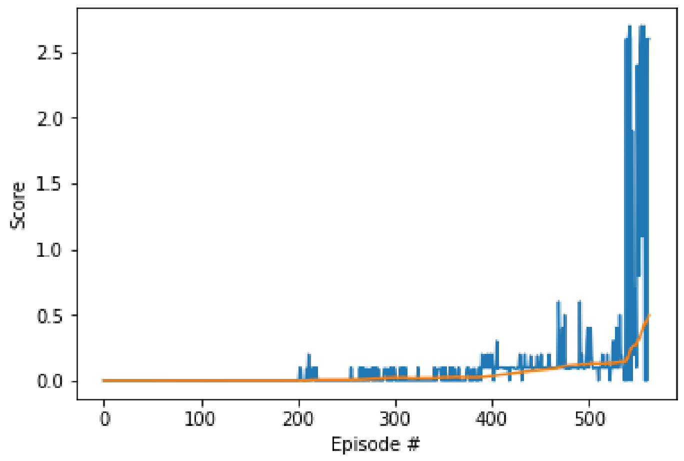
The hyperparameters used to train the agent are:

- BUFFER_SIZE = int(1e6) # replay buffer size
- BATCH_SIZE = 256 # minibatch size
- GAMMA = 0.99 # discount factor
- TAU = 1e-3 # for soft update of target parameters
- LR_ACTOR = 2e-4 # learning rate of the actor
- LR_CRITIC = 2e-4 # learning rate of the critic
- WEIGHT_DECAY = 0 # L2 weight decay
- UPDATE_EVERY= 1# Update interval
- NUM_UPDATES=1 # Number of learning steps after every predefined interval in UPDATE_EVERY

## Plot of Rewards

The plot of average rewards for every 100 episodes are listed below.

```
• Episode 100     Average Score: 0.00   Score: 0.00
• Episode 200     Average Score: 0.00   Score: 0.00
• Episode 300     Average Score: 0.02   Score: 0.10
• Episode 400     Average Score: 0.04   Score: 0.10
• Episode 500     Average Score: 0.13   Score: 0.40
• Episode 563     Average Score: 0.50   Score: 2.60
• Environment solved in 563 episodes!   Average Score: 0.50
```



## Ideas for Future Work

The Reinforcement Learning agent was trained using Deep Deterministic Policy Gradients extended to work over multiple agents which are interacting over multiple agents

- Prioritized Experience Replay might show better results in comparison.
- Code Optimizaion might be needed
- I stopped training when the target goal was reached, but further training for a longer period of time might lead to more insights on how long the current agent maintains its stability