Background and requirements:

The goal in this project is to train two agents controlling rackets to bounce a ball over a net. By taking advantage of the Unity Machine Learning Agents (ML-Agents), which is an open-source Unity plugin that enables games and simulations to serve as environments for training intelligent agents, this project would be straightford on focusing on the model development and performance improving.

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.

The task is episodic, and in order to solve the environment, your agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents). Specifically,

- After each episode, we add up the rewards that each agent received (without discounting), to get a score for each agent. This yields 2 (potentially different) scores. We then take the maximum of these 2 scores.
- This yields a single score for each episode.

The environment is considered solved, when the average (over 100 episodes) of those scores is at least +0.5.

Learning Algorithm

After briefly reviewing the project requirement, a straightforward idea using the solutions from the project 2 but with MADDPG algorithm (Multi-Agent Deep Deterministic Policy Gradient) looking very promising.

Policy-based, Value-based and Actor-Critic:

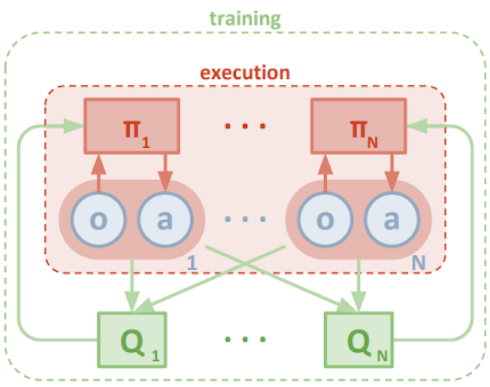
RL algorithms using neural networks as function approximators can take approaches with things they are estimating. Firstly, given a state as input they can estimate the expected future rewards for different possible actions. This approach is known as a **value-based method**. However, for tasks that involve continuous actions, a function approximator to estimate the actual policy directly could be applied. This approach is known as a **policy-based method**.By applying the DDPG (Deep

Deterministic Policy Gradient, Continuous Action-space) algorithm as an "Actor-Critic" method is introduced.

MADDPG:

As presented in in the paper Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments (https://arxiv.org/abs/1706.02275), this project adapts the actor-critic method mentioned above to consider action policies of other agents by adopting a centralized training with decentralized execution.

Decentralized: any agent has its private Actor network and takes actions considering only its observation of the environment. **Centralized training:** although each agent has its private Critic network, the critic takes as input the observations of any agent present in the environment as well as their actions in order to ease training while learning the action-value function *Q(s,a)*.

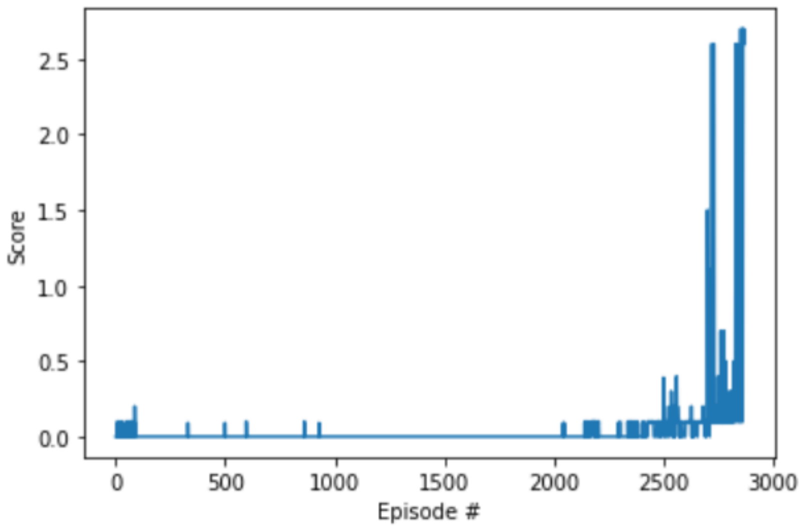


On top of taking advantage of the MADDPG algorism, this solution also had several adjustments has been done to improve the performance,

1. On top of the Ornstein-Uhlenbeck process, introduce the epsilon parameter for noise decay
2. Fine-tune the theta (speed of mean reversion) and sigma (volatility) parameters for adding noise
3. Applying gradient clipping and normalization
4. Adjusting learning interval, by learn every 20 steps with 10-time multiplication.

After these adjustments, the performance improved and the environment is solved as shown in the plot, the agent is able to receive an average (over 100 episodes) of those scores is at least +0.5.

```
[8]: import matplotlib.pyplot as plt
     fig = plt.figure()
     ax = fig.add_subplot(111)
     plt.plot(np.arange(1, len(scores)+1), scores)
     plt.ylabel('Score')
     plt.xlabel('Episode #')
     plt.show()
```



```
Episode 100     Average Score: 0.02rage maximum score over the last 10 episodes: 0.00
Episode 200     Average Score: 0.00rage maximum score over the last 10 episodes: 0.00
Episode 300     Average Score: 0.00rage maximum score over the last 10 episodes: 0.00
Episode 400     Average Score: 0.00rage maximum score over the last 10 episodes: 0.00
Episode 500     Average Score: 0.00rage maximum score over the last 10 episodes: 0.01
Episode 600     Average Score: 0.00rage maximum score over the last 10 episodes: 0.01
Episode 700     Average Score: 0.00rage maximum score over the last 10 episodes: 0.00
Episode 800     Average Score: 0.00rage maximum score over the last 10 episodes: 0.00
Episode 900     Average Score: 0.00rage maximum score over the last 10 episodes: 0.00
Episode 1000    Average Score: 0.00rage maximum score over the last 10 episodes: 0.00
Episode 1100    Average Score: 0.00rage maximum score over the last 10 episodes: 0.00
Episode 1200    Average Score: 0.00rage maximum score over the last 10 episodes: 0.00
Episode 1300    Average Score: 0.00rage maximum score over the last 10 episodes: 0.00
Episode 1400    Average Score: 0.00rage maximum score over the last 10 episodes: 0.00
Episode 1500    Average Score: 0.00rage maximum score over the last 10 episodes: 0.00
Episode 1600    Average Score: 0.00rage maximum score over the last 10 episodes: 0.00
Episode 1700    Average Score: 0.00rage maximum score over the last 10 episodes: 0.00
Episode 1800    Average Score: 0.00rage maximum score over the last 10 episodes: 0.00
Episode 1900    Average Score: 0.00rage maximum score over the last 10 episodes: 0.00
Episode 2000    Average Score: 0.00rage maximum score over the last 10 episodes: 0.00
Episode 2100    Average Score: 0.00rage maximum score over the last 10 episodes: 0.00
Episode 2200    Average Score: 0.03rage maximum score over the last 10 episodes: 0.05
Episode 2300    Average Score: 0.01rage maximum score over the last 10 episodes: 0.03
Episode 2400    Average Score: 0.01rage maximum score over the last 10 episodes: 0.00
Episode 2500    Average Score: 0.06rage maximum score over the last 10 episodes: 0.13
Episode 2600    Average Score: 0.07rage maximum score over the last 10 episodes: 0.01
Episode 2700    Average Score: 0.09rage maximum score over the last 10 episodes: 0.08
Episode 2800    Average Score: 0.22rage maximum score over the last 10 episodes: 0.10
Episode 2866    max score: 2.60 average maximum score over the last 10 episodes: 1.46
Environment solved in 2766 episodes!    Average Score: 0.51
```

Model architecture

# ACTOR NETWORK

Batch Norm Layer -> Fully Connected Layer -> Leaky Relu(leakiness=0.01) -> Fully Connected Layer -> Leaky Relu(leakiness=0.01) Fully Connected Layer ->Tanh Activation()

# CRITIC NETWORK

Batch Norm Layer -> Fully Connected Layer -> Leaky Relu(leakiness=0.01) -> Fully Connected Layer -> Leaky Relu(leakiness=0.01) Fully Connected Layer

Hyperparameters and values

Continuous_control.ipynb Notebook:

rewards	# get the rewards
next_states	# get the resulting states
dones = env_info.local_done	# check whether episodes have finished

ddpg_agent.py:

BUFFER_SIZE = int(1e6)	# replay buffer size
BATCH_SIZE = 128	# minibatch size
GAMMA = 0.99	# discount factor
TAU = 1e-3	# for soft update of target parameters
LR_ACTOR = 1e-3	# learning rate of the actor
LR_CRITIC = 1e-3	# learning rate of the critic
WEIGHT_DECAY = 0.	# L2 weight decay
OU_SIGMA = 0.2	# Ornstein-Uhlenbeck noise parameter
OU_THETA = 0.15	# Ornstein-Uhlenbeck noise parameter
EPSILON = 1.0	# explore->exploit noise process added to act step
EPSILON_DECAY = 1e-6	# decay rate for noise process

model.py:

state_size (int):	State dimension
action_size (int):	Action dimension
random_seed (int):	Random seed
fc1_units (int):	First hidden layer nodes
fc2_units (int):	second hidden layer nodes
hidden_size(int):	Hidden layers node
leak	Leakiness in leaky relu

Future ideas

As this project is done in tight timeline, there still plenty to improve when time allows, that includes:

- **Hyperparameters tuning**, that would give a better performance and solve the environment faster.
- Improve the **noise process algorithms**.
- Training regimen utilizing an **ensemble of policies** for each agent.
- Further investigation of introducing **negative rewards**.