Environmental Details

An environment (of two-player game) is being set-up where agents control rackets to bounce ball over a net. As for the goal, the agents must keep bouncing ball between one another while not dropping or sending ball out of bounds.

Agent's **Reward Function** (independent):
+0.1 To agent when hitting ball over net.
-0.1 To agent who let ball hit their ground, or hit ball out of bounds.

Observation space: 8 variables corresponding to position and velocity of ball and racket. In the Udacity provided environment, 3 observations are stacked (8 *3 = 24 variables)

Action space: (Continuous) Size of 2, corresponding to movement toward net or away from net, and jumping.

Learning Algorithm

To solve this given environment, the algorithm of MADDPG is being used. MADDPG (Multi-Agent Deep Deterministic Policy Gradient) is a deep reinforcement learning algorithm designed for multi-agent systems. It is an extension of the DDPG (Deep Deterministic Policy Gradient) algorithm, which is a model-free, off-policy algorithm that is used for continuous control tasks. MADDPG is specifically designed to handle the challenges that arise when multiple agents are present in an environment, where each agent's actions can impact the state of the environment and the actions of other agents. The goal of the algorithm is to learn a decentralized policy for each agent that maximizes its own expected cumulative reward while taking into account the actions of the other agents.

The key idea behind MADDPG is to use a centralized critic network to estimate the value function for each agent, which takes as input the states and actions of all agents, and outputs the expected cumulative reward for each agent. The actor network for each agent then uses the output from the centralized critic network as a baseline, and learns a decentralized policy using the DDPG algorithm. During training, each agent interacts with the environment and receives observations of the state of the environment, along with the actions and rewards of other agents. Each agent then updates its actor and critic networks using the DDPG algorithm, with the additional step of using the critic network to estimate the value function of the other agents.

MADDPG has been shown to be effective in a range of multi-agent environments, including cooperative, competitive, and mixed scenarios.

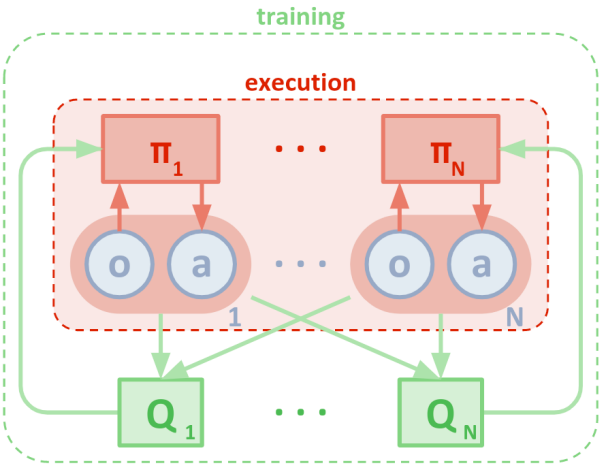

Figure 1: Overview of the multi-agent decentralized actor, centralized critic approach.

Implementation

The concrete execution can be found in the directory of "p3_collab-compet". The reinforcement learning agent of the MADDPG algorithm can be found in "maddpg_agent.py," while "model.py" contains the neural networks utilized as the estimators. The original code was sourced from the "ddpg-pendulum" exercise from Udacity but was modified to suit the requirements of this particular problem.

Model architectures

The algorithm employs two deep-neural-networks (which consist of 1 actor and 1 critic). The structure is as follow:

Actor model:
Input layer: 24 (state size) neurons
1st hidden layer: 256 neurons, [relu activation function]
2nd hidden layer: 128 neurons, [relu activation function]
Output layer: 2 (action size) neurons, [tanh activation function]

Critic model:
Input layer: 24 (state size) neurons
1st hidden layer: (256 + 2) neurons (where the additional 2 are the action size), [relu activation function] -> [batch normalization]
2nd hidden layer: 128 neurons, [relu activation function]
Output layer: 1 neuron

Hyper-parameters

Setting 1

- num of episodes allowed: 10000
- max timestep allowed: 10000
- replay buffer size: 10000
- minibatch size: 256
- gamma, discount factor: 0.99
- tau, soft update for target networks factor: 3e-1
- Learning rate of the actor: 1e-4
- Learning rate of the critic: 1e-4
- OU (Ornstein-Uhlenbeck process) noise decay: 0.999

In deep reinforcement learning, the Ornstein-Uhlenbeck (OU) process is often used as a source of exploration noise. This stochastic process generates temporally correlated noise, which can encourage exploration by the agent to try different actions over time. During training, it's important to balance exploration and exploitation to achieve better learning outcomes. Exploration involves trying out new actions to find better strategies, while exploitation involves using the current best strategy to maximize rewards.

To balance exploration and exploitation, the OU process noise is added to the action selection process during training. At the beginning of training, the exploration noise is high (0.999) to encourage the agent to explore a wide range of actions. As the agent learns and improves, the exploration noise is gradually reduced or decayed, allowing the agent to focus more on exploiting the current best strategy.

Plot of Rewards

The result of training shows that the agent is able to receive an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents). As indicated in the Figure 2, the number of episodes needed to solve the environment was reported as near the 350th.

```
Episode 100     Average Score: 0.02rage maximum score over the last 10 episodes: 0.03
Episode 200     Average Score: 0.06rage maximum score over the last 10 episodes: 0.08
Episode 300     Average Score: 0.13rage maximum score over the last 10 episodes: 0.23
Episode 350     max score: 2.60 average maximum score over the last 10 episodes: 1.14
Environment solved in 250 episodes!     Average Score: 0.51
```

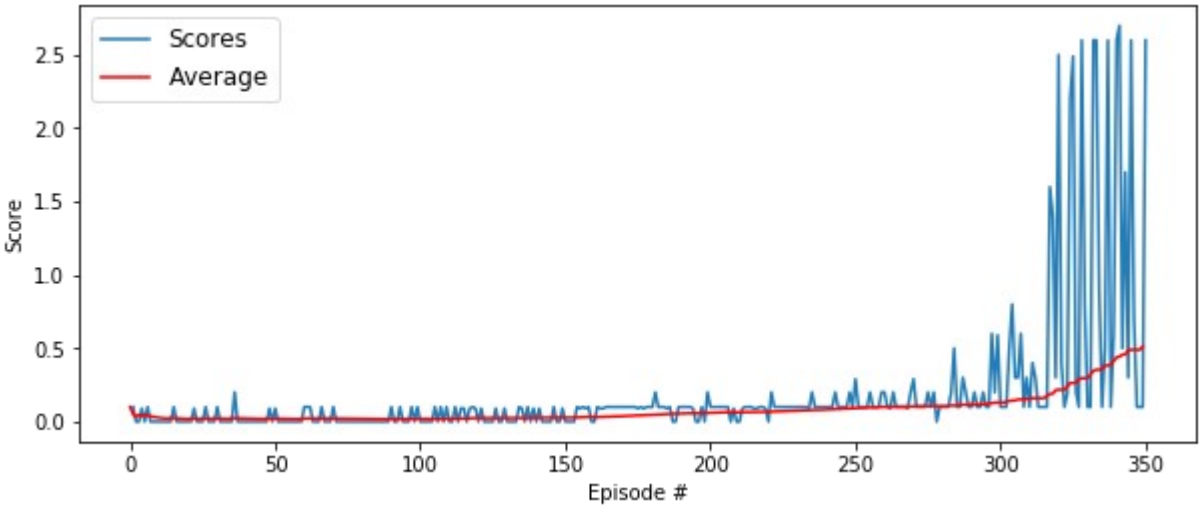

Figure 2: Plot of the training scores

Idea for future work

1. Implement the Prioritized Experience Replay, so as to replay important transitions more frequently, and therefore let the agents learn more efficiently [2].
2. Can further improve the performance of MADDPG method by training agents with an ensemble of policies, an approach which is believed to be generally applicable to any multi-agent algorithm [1].

Reference

1. https://arxiv.org/abs/1706.02275
2. https://arxiv.org/abs/1511.05952