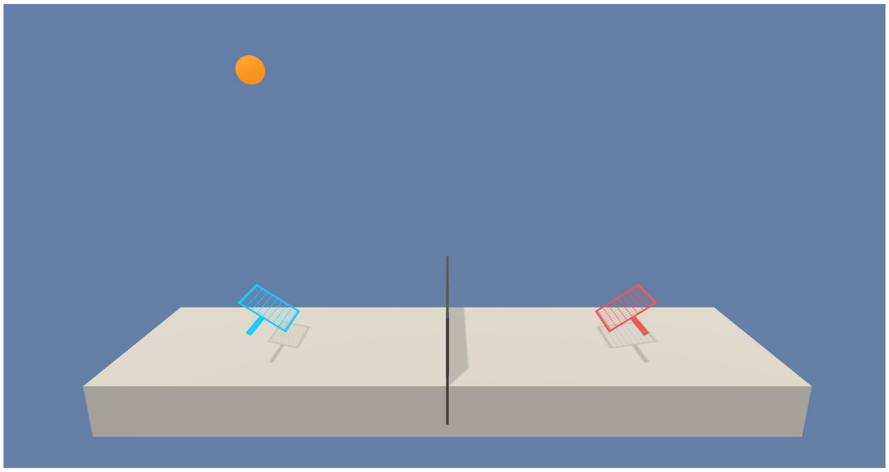
Collaboration and Competition



Project's Goal

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.

Environment and Project Information

- The project environment is similar to, but not identical to the Tennis environment on the Unity ML-Agents
- 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).
- 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, Network Structure and Implementation

In this project, MADDPG algorithm used for learning algorithm.

According to Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments paper, the motivation behind this algorithm is;

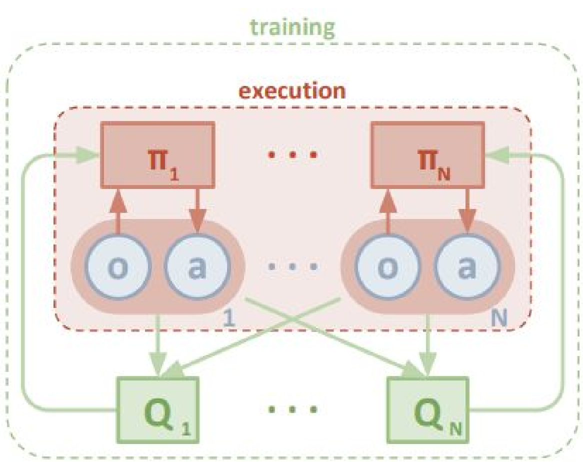


Figure 1: Overview of our multi-agent decentralized actor, centralized critic approach.

- If we know the actions taken by all agents, the environment is stationary even as the policies change

Why MADDPG? According to the paper;

- leads to learned policies that only use local information (i.e. their own observations) at execution time
- does not assume a differentiable model of the environment dynamics or any particular structure on the communication method between agents
- is applicable not only to cooperative interaction but to competitive or mixed interaction involving both physical and communicative behavior

In the project;

- Each agent has own actor and critic model
- Training agents all use common experience replay buffer.
- Hyperparameters set by experimental results and research.
- For the Continuous Control project i used DDPG algorithm, then for this project i want to try MADDPG (basically; every agent modeled by DDPG but they are also sharing some information. It is an extended version of DDPG. MADDPG allows agents to compete each other and to collaboration)

MADDPG Algorithm

Multi-Agent Deep Deterministic Policy Gradient Algorithm

For completeness, we provide the MADDPG algorithm below.

Algorithm 1: Multi-Agent Deep Deterministic Policy Gradient for N agents

---

for episode $= 1$ to M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state $\mathbf{x}$
  for $t = 1$ to max-episode-length **do**
    for each agent i, select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration
    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state $\mathbf{x}'$
    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    for agent $i = 1$ to N **do**
      Sample a random minibatch of S samples $(\mathbf{x}^j, a^j, r^j, \mathbf{x}'^j)$ from $\mathcal{D}$
      Set $y^j = r_i^j + \gamma Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)|_{a'_k = \boldsymbol{\mu}'_k(o_k^j)}$
      Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_N^j)\right)^2$
      Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i}\boldsymbol{\mu}_i(o_i^j)\nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)\big|_{a_i = \boldsymbol{\mu}_i(o_i^j)}$$

    end for
    Update target network parameters for each agent i:

$$\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$$

  end for
end for

Architecture of Networks:

1. **Architecture of Actor Network** : 2 fully connected hidden layer (256), Relu activation function and batch normalization layer and tanh

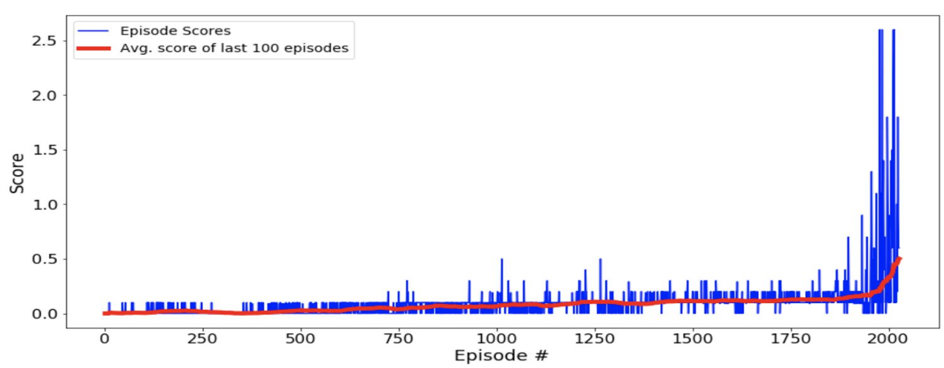
2. **Architecture of Critic Network:** 2 fully connected hidden layer (256), Activation is Relu and batch normalization

Parameters and Results:

```
_(self, action_size=2, seed=0,
  n_agents=2,
  buffer_size=10000,
  batch_size=256,
  gamma=0.99,
  update_every=2,
  noise_start=1.0,
  noise_decay=1.0,
  t_stop_noise=30000):
```

```
Episode 200     Average Score: 0.025
Episode 400     Average Score: 0.007
Episode 600     Average Score: 0.024
Episode 800     Average Score: 0.051
Episode 1000    Average Score: 0.064
Episode 1200    Average Score: 0.091
Episode 1400    Average Score: 0.094
Episode 1600    Average Score: 0.118
Episode 1800    Average Score: 0.128
Episode 2000    Average Score: 0.326
Episode 2027    Average Score: 0.501
Environment solved in 2027 episodes!    Average Score: 0.501
```

Plot of Rewards



Ideas of Future Work:

Proximal Policy Optimization Algorithms

- PPO strikes a balance between ease of implementation, sample complexity, and ease of tuning, trying to compute an update at each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small.
- Detailed about PPO algorithm can be found in this paper

Here is the algorithm

So that's it! We can finally summarize the PPO algorithm

1. First, collect some trajectories based on some policy π_θ, and initialize theta prime $\theta' = \theta$
2. Next, compute the gradient of the clipped surrogate function using the trajectories
3. Update θ' using gradient ascent $\theta' \leftarrow \theta' + \alpha \nabla_{\theta'} L_{\text{sur}}^{\text{clip}}(\theta', \theta)$
4. Then we repeat step 2-3 without generating new trajectories. Typically, step 2-3 are only repeated a few times
5. Set $\theta = \theta'$, go back to step 1, repeat.