Project Details

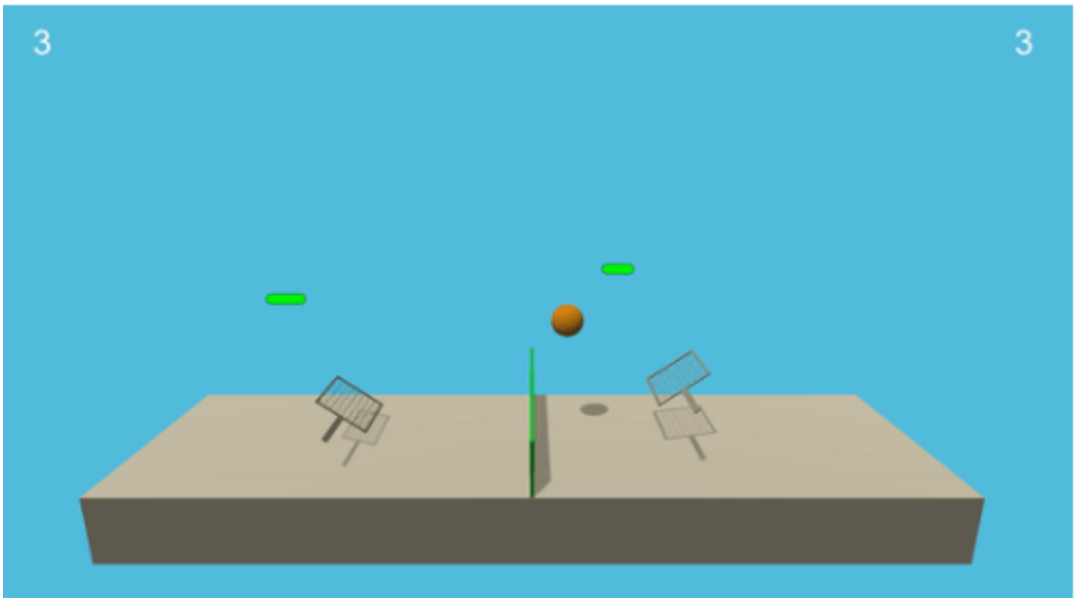
In this environment, two agents control rackets to bounce a ball over a net. If an agent hits the ball over the net, it receives a reward of +0.1. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. Thus, the goal of each agent is to keep the ball in 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).

- 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.**



Multi-agent DDPG Actor-Critic Architecture

To achieve the goal score, a multi-agent DDPG (deep deterministic Policy Gradient) Actor-Critic architecture was chosen.

For completeness, we provide the MADDPG algorithm below.

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

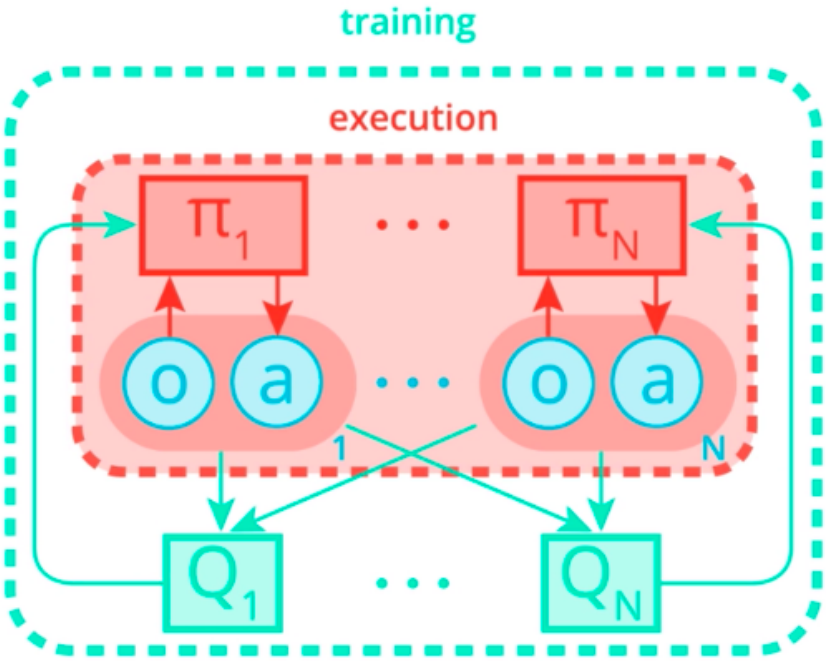
for episode $= 1$ to M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state x
  for $t = 1$ to max-episode-length **do**
    for each agent i, select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration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

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

    end for
    Update target network parameters for each agent i:

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

  end for
end for



Similar to single-agent Actor Critic architecture, each agent has it's own actor and critic network. The actor network takes in the current state of agent and output a recommended action for that agent. However the critic part is slightly different from ordinary single-agent DDPG. Here the critic network of each agent has full visibility on the environment. It not only takes in the observation and action of that particular agent, but also observations and actions of all other agents as well. Critic network has much higher visibility on what is happening while actor network can only access to the observation information of the respective agent. The output of the critic network is, nevertheless, still the Q value estimated given a full observation input(all agents) and a full action input(all agents). The output of the actor network is a recommended action for that particular agent.

- Actor - It proposes an action given a state.
- Critic - It predicts if the action is good (positive value) or bad (negative value) given a state and an action.

Why two networks? Because it adds stability to training. In short, we are learning from estimated targets and Target networks are updated slowly, hence keeping our estimated targets stable.

Goal

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

Reacher environment used: UnityEnvironment(file_name="/data/Tennis_Linux_NoVis/Tennis")

Getting Started:

What you need to install
Pytorch
Python 3x

Instructions:

How to Run, simple agent:

Github Link:
https://github.com/ChristianET-DS/Multi-Agent-Final-Project/blob/main/TennisFinal.ipynb

Model Architecture

Actor Model Architecture

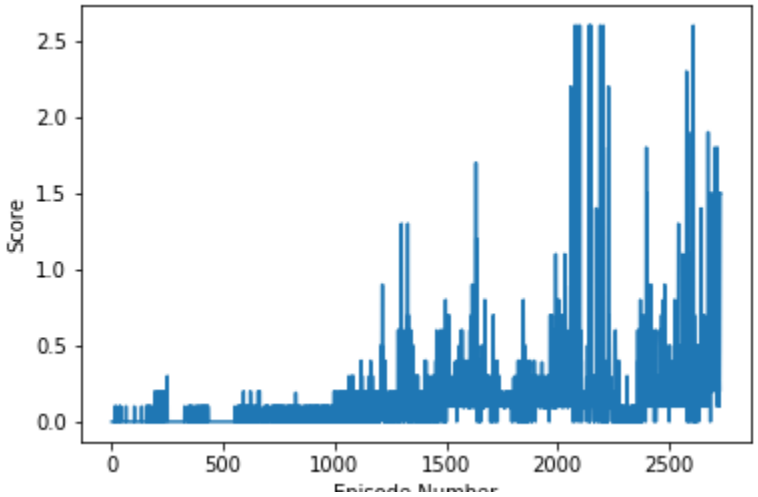
```
 # Dense layers ##########################################################

        self.fc1 = nn.Linear(input_size, fc1_units)
        self.fc2 = nn.Linear(fc1_units, fc2_units)
        self.fc3 = nn.Linear(fc2_units, output_size)

 # Normalization layers ##################################################
        self.bn = nn.BatchNorm1d(fc1_units)

# Forward ##############################################################

        x = F.leaky_relu(self.fc1(state))
        x = F.leaky_relu(self.fc2(x))
        x = F.tanh(self.fc3(x))
```

Critic Model Architecture

```
 # Dense layers ##########################################################

        self.fc1 = nn.Linear(input_size, fc1_units)
        self.fc2 = nn.Linear(fc1_units, fc2_units)
        self.fc3 = nn.Linear(fc2_units, 1)
        self.bn1 = nn.BatchNorm1d(fc1_units)


 # Normalization layers ##################################################
        self.bn1 = nn.BatchNorm1d(fc1_units)

# Forward ##############################################################
       xs = torch.cat((states, actions), dim=1)
        x = F.leaky_relu(self.fc1(xs))
        x = self.bn1(x)
        x = F.leaky_relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Results:

Around 5 Hours training with GPU.
Episode 2731 Average Score: 0.506
Environment solved in 2731 episodes! Average Score: 0.506



Ideas for Future Work:

- Play with forward functions and model parameters
- Improve the learning, getting the average under 2000 Episodes.
- Try to improve the training time. This took more than 5 Hours with GPU.