DRL Collaboration and Competition

Introduction

This project aims to train two agents to learn how to play tennis collaboratively. Two agents control rackets to bounce a ball over a net in this environment. 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 local observation. Two continuous actions are available, corresponding to movement toward (or away from) the net and jumping.

The task is episodic, and to solve the environment, the agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents). This problem will be solved using the MADDPG using PyTorch and Python.

Methodology

DDPG is an example of an actor-critic method. It stands for Deep Deterministic Policy Gradient. It is an extension of the DQN (Deep Q-Network) algorithm used for discrete action spaces.

The key idea behind DDPG is to use deep neural networks to approximate the actor and critic functions. The actor-network takes the current state as input and outputs the best action to take. The critic network takes both the state and action as input and estimates the Q-value, which represents the expected return of taking that action in that state.

The actor update equation is used to update the weights of the actor-network. It aims to maximize the expected return by adjusting the actor's policy. The equation is as follows:

$$\nabla \theta\mu \approx 1/N \sum(\nabla aQ(s, a|\theta Q)|s=s, a=\mu(s|\theta\mu)) \ \nabla \theta\mu(s|\theta\mu)$$

Here, $\nabla\theta\mu$ represents the gradients of the actor network's weights, $\theta\mu$, with respect to the expected return. N is the batch size, s is the state, a is the action, $Q(s, a|\theta Q)$ is the Q-value estimated by the critic network, and $\mu(s|\theta\mu)$ is the action selected by the actor-network.

The critic update equation is used to update the weights of the critic network. It aims to minimize the mean squared error between the estimated and target Q values. The equation is as follows:

$$L = 1/N * \sum(y - Q(s, a|\theta Q))^2$$

Here, L represents the loss function, N is the batch size, y is the target Q-value, and $Q(s, a|\theta Q)$ is the Q-value estimated by the critic network.

The target Q-value, y, is calculated using the target actor and target critic networks. It is given by:

$$y = r + \gamma * Q'(s', \mu'(s'|\theta\mu')|\theta Q')$$

Here, r is the reward received, γ is the discount factor, s' is the next state, μ' is the action selected by the target actor network, and $Q'(s', \mu'(s'|\theta\mu')|\theta Q')$ does the target critic network estimate the Q-value.

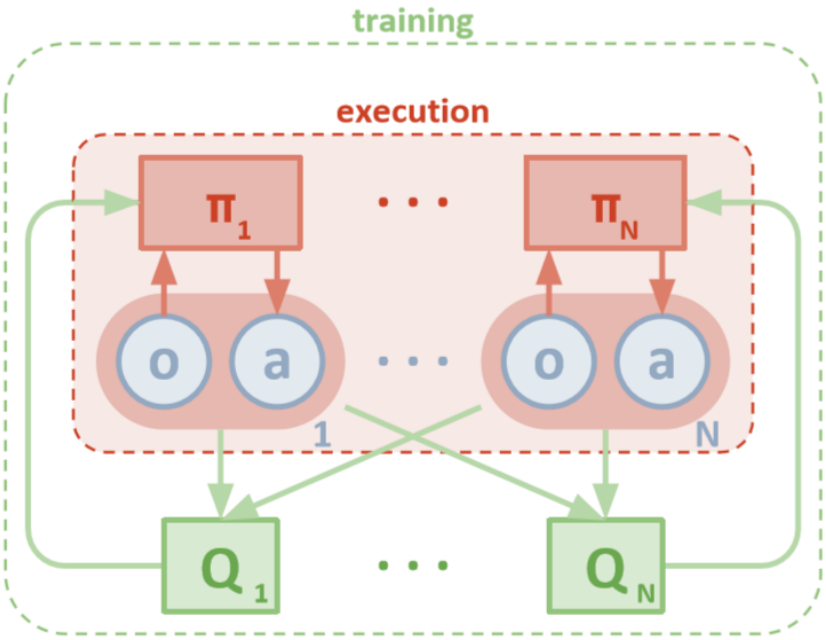
To improve the performance of the DDPG algorithm, two techniques are used: replay buffer and soft updates to the target networks.

1. The **replay buffer** is a memory that stores the agent's experiences. It randomly samples experiences from the buffer to break the correlation between consecutive experiences and improve training stability. This allows the agent to learn from a diverse set of experiences.
2. **Soft updates** to the target networks are used to update the target actor and critic networks slowly. Instead of copying the weights from the regular networks to the target networks abruptly, DDPG blends the weights gradually. This helps stabilize the learning process and prevents sudden changes that could lead to instability.

To adopt a DDPG codebase to implement the MADDPG algorithm, the following several changes where made :-

1. Multiple Agents: First and foremost, the code was extended to handle multiple agents. This means maintaining separate actor and critic networks for each agent.

2. Centralized Critic: In MADDPG, I have implemented a centralized critic network that takes as input the joint actions of all agents.



3. Experience Sharing: During training, each agent should share their experiences with the centralized critic. This means that the experiences collected by one agent should be made available to all other agents when updating the critic.

4. Decentralized Execution: During execution (when agents are interacting with the environment), each agent was coded to use its own local observations to select actions.

5. Target Networks: Since there are multiple agents. Separate target networks for each agent's actor and critic where maintained.

6. Multi-Agent Training Loop: The training loop was modified to accommodate multiple agents. This involves sampling experiences from each agent's replay buffer and updating their respective actor and critic networks.

The neural network architecture for the actor has two hidden layers and an output layer. The ReLu activation function was used for both layers and TanH for the output layer. The critic also has two hidden layers and an output layer, ReLu and Linear.

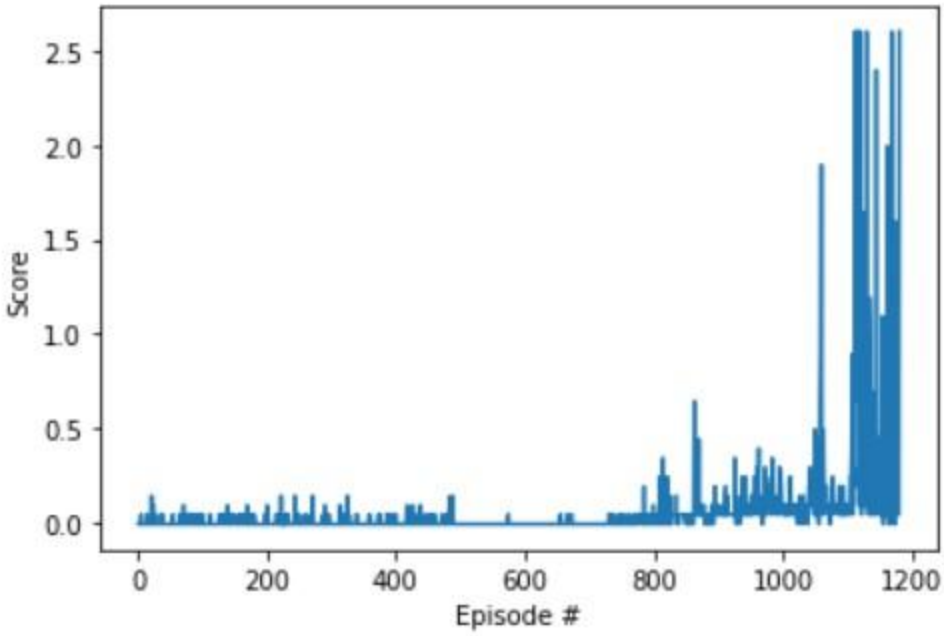
Hyperparameters

For the training, the following hyperparameters were used.

Parameter	Value
Number of Episodes	2000
Maximum Time Steps	1000
Print Every	100
Num_agents	2
Random Seed	38
Replay buffer size	int(1e5)
Batch Size	256
Discount Factor	0.99
Soft update of target Parameters	1e-3
Learning Rate Actor	1e-4
Learning Rate Critc	3e-4
Average value of the noise	0
standard deviation of the noise	1.0

Results

The following reward plot was obtained when the agent achieved an average reward of 0.52 for 100 continuous cycles. The environment was solved in 1180.



Future Work:

I intend to do the following:-

1. Implement Prioritized experience replay and try different algorithms to improve the results obtained in this project.
2. To complete the ‘Play Soccer’ challenge, where the goal is to train a small team of agents to play soccer.