Project: Collaboration and Competition

The goal of this project is to train two Agents to play table tennis against each other.

In the environment of the project, 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, the agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents). Specifically,

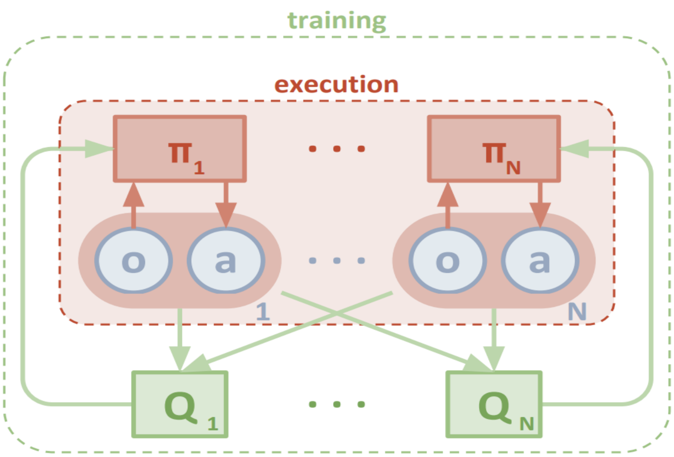
- After each episode, the rewards that each agent received are added up (without discounting), to get a score for each agent. This yields 2 (potentially different) scores. Then the maximum of these 2 scores is taken.
- This yields a single **score** for each episode.

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

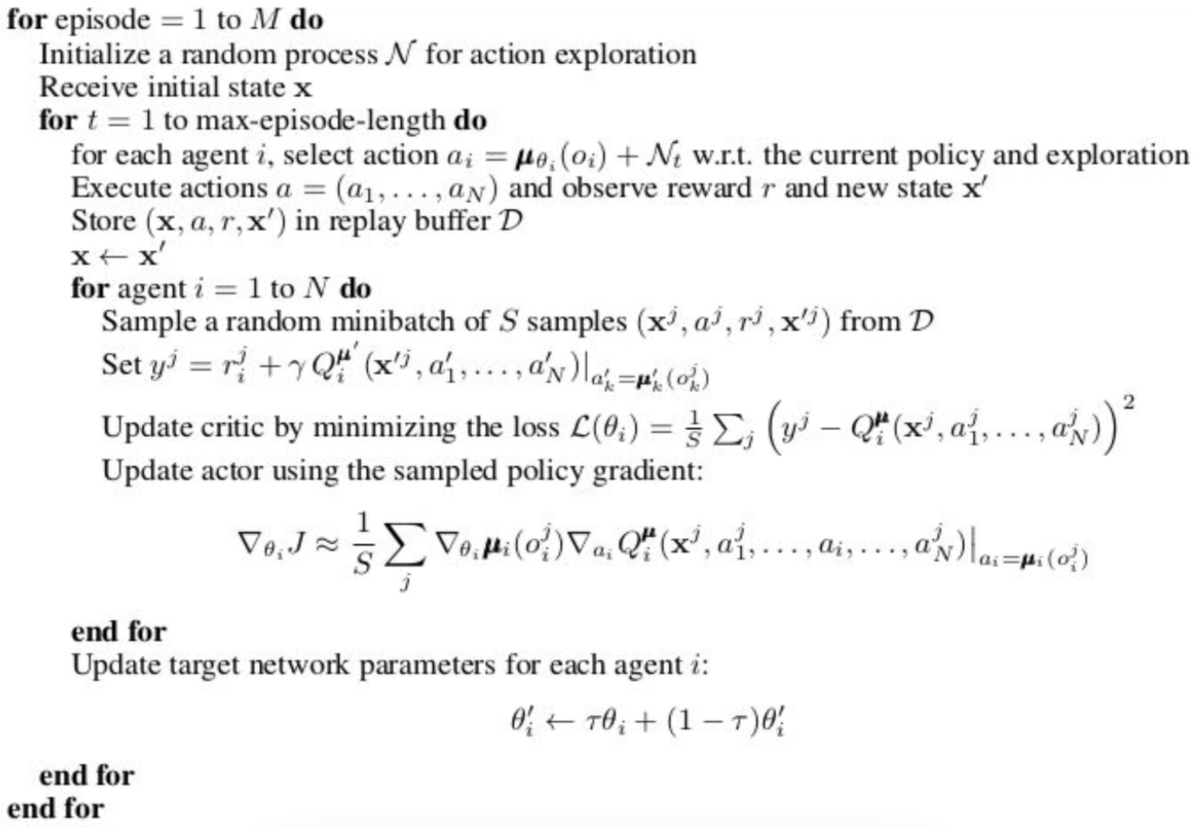
Environment of this project involves interaction between multiple (two) agents, where emergent behavior and complexity arise from agents co-evolving together. Unfortunately traditional reinforcement learning approaches such as Q-Learning or policy gradient are poorly suited to multi-agent environments. One issue is that each agent's policy is changing as training progresses, and the environment becomes non-stationary from the perspective of any individual agent. This presents learning stability issue and prevents the straightforward use of past experience replay, which is crucial for stabilizing deep Q-learning. Policy gradient methods, on the other hand, exhibit very high variance when coordination of multiple agents is required.

Learning Algorithm

Therefore to achieve the goal of this project I decided to use the a general-purpose multi-agent learning algorithm called Multi-Agent Deep Deterministic Policy Gradient method, which represents an extension of actor-critic policy gradient method (DDPG). The main idea of this method implies that the critic is augmented with extra information about the policies of other agents, while the actor only has access to local information. After training is completed, only the local actors are used at execution phase, acting in a decentralized manner



Algorithm that implements Multi-Agent Deep Deterministic Policy Gradient method has following structure:



```
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 $a = (a_1, \dots, a_N)$ 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 $y^j = r_i^j + \gamma Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \dots, a'_N)|_{a'_k = \boldsymbol{\mu}'_k(o_k^j)}$
            Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \dots, a_N^j)\right)^2$
            Update actor using the sampled policy gradient:
                $$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i}\boldsymbol{\mu}_i(o_i^j)\nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \dots, a_i, \dots, a_N^j)\big|_{a_i = \boldsymbol{\mu}_i(o_i^j)}$$
        end for
        Update target network parameters for each agent $i$:
                $$\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$$
    end for
end for
```

Implemented Actor's Neural Network has following architecture :

- 3 Fully Connected Layers (state pace, 256, 128, action space)
- First two fully connected layers are followed by ReLU activation
- The final output layer of the actor network is a tanh layer, to bound the resulting action values in a range (-1; +1).
- Optimizer type: Adam optimizer with a learning rate LR=1e-4.

The Critic's Neural Network has following architecture :

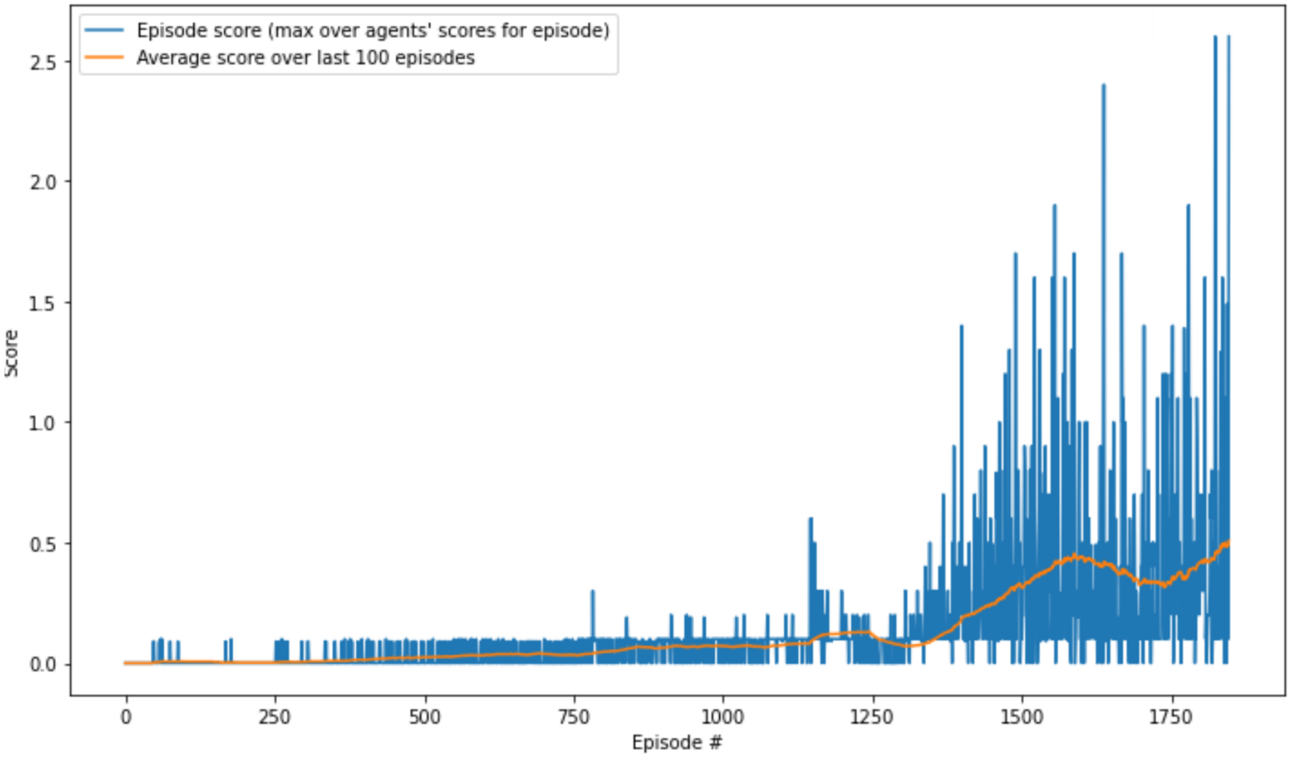
- 3 fully connected layers (state pace * num agents, 256, 128, 1)
- First two fully connected layers are followed by ReLU activation
- Optimizer type: Adam optimizer with a learning rate LR=2e-4.

Training process hyperparameters:

- Max episode length: max 10000 time steps or until is done
- Experience replay memory buffer size: 100000
- Batch size: 128
- discount factor for prioritizing future rewards (gamma): 0.99
- Fusion ratio for soft update of weights of target network (Tau): 0.3
- Frequency of updates for the weights of target networks: every 2 actions of Agent
- Number of learning passes per Agent's action: 4

Plot of Rewards

Following graph demonstrates the progression of rewards score per episode during the Agents' training phase. Additionally, the progression of mean score over the 100-episodes sliding window is displayed in orange color. The graph shows that the Agents successfully learned to reach an average reward of at least 0.5 over 100 consecutive episodes. The Agents managed to achieve this goal after 1847 episodes of learning.



Conclusions and ideas for Future Work

For solving the goal of this project I have used the Multi-Agent Deep Deterministic Policy Gradient method, which has shown to be quite efficient for a multi-agent environment.
The main challenge in making this implementation successful in solving the project goal was to find a proper combination of hyperparameters for all components of the learned Agents via multiple sessions of learning.

Following are my main observations from fine tuning of hyperparameters for implemented solution:

- Changing the number of nodes for the first hidden layer of actor and critic network from 256 to 128 improved the speed of learning process but could not allow to achieve a maximum average score 0.5 over the last 100 episodes. The maximum achieved score was 0.32.
- Batch size of 128 seems to show the most optimal learning rate
- Selected value of fusion ratio for soft update of target network weights from local network weights (TAU) seems to be quite high (3e-1), but setting smaller values (e.g common 1e-3) did not show good results during training process (project goal 0.5 could not be reached even after 3000 episodes)

One of the ways to improve the current solution could be introducing a batch normalization for the first layer of both actor and critic networks, since different components of the observation space have different physical units (for example, positions versus velocities). Introducing this technique into the algorithm might help to normalize each dimension across the samples in a minibatch to have unit mean and variance.
Another future improvement can be introduction of noise factor (epsilon) decay over network learning, to decrease the space for exploration the more 'experienced' the agent gets.