Collaboration and Competition udacity deep RL project report

Overview:

This report illustrates the conducted work to pass the collaboration and competition final udacity deep reinforcement learning project. We will present the implementation, results, and future work.
Link to the project repository: github link

1- Implementation details:

The project has the following structure:

- **.Isort.cfg:** config file that specifies some code setting (line_length, packages,...).
- **.pre-commit-config.yaml:** config file to specify the hooks used when running `pre-commit`.
- **core.py:** Serves as a wrapper to encapsulate the Tennis unity environment.
- **environment.yml:** contains the dependencies and packages to be able to run the code.
- **train.py:** Train the agent.
- **ddpg.py**: ddpg implementation inspired from Udacity deep RL repository.
- **Maddpg.py**: multi agent ddpg that handles the shared buffer and the learning of the two agents.
- **model.py**: contains the actor and critic architecture.
- **checkpoints**: directory where to store the checkpoints.
- **plots:** directory where to store the plots.
- **report.pdf**: A summary of the conducted work.

2- Learning algorithm:

In this project we used **ma*ddpg*** algorithm to train our agent.
Following is the pseudo-code of the algorithm that we want to implement:

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
```

Taken from "Medium blog: https://medium.com/@amitpatel.gt/maddpg-91caa221d75e

Actor architecture:

- 3 fully connected layers. [state_size, 256, 128. action_size]
- The first layers are followed by the ***Relu*** activation function and the last one with ***tanh***.

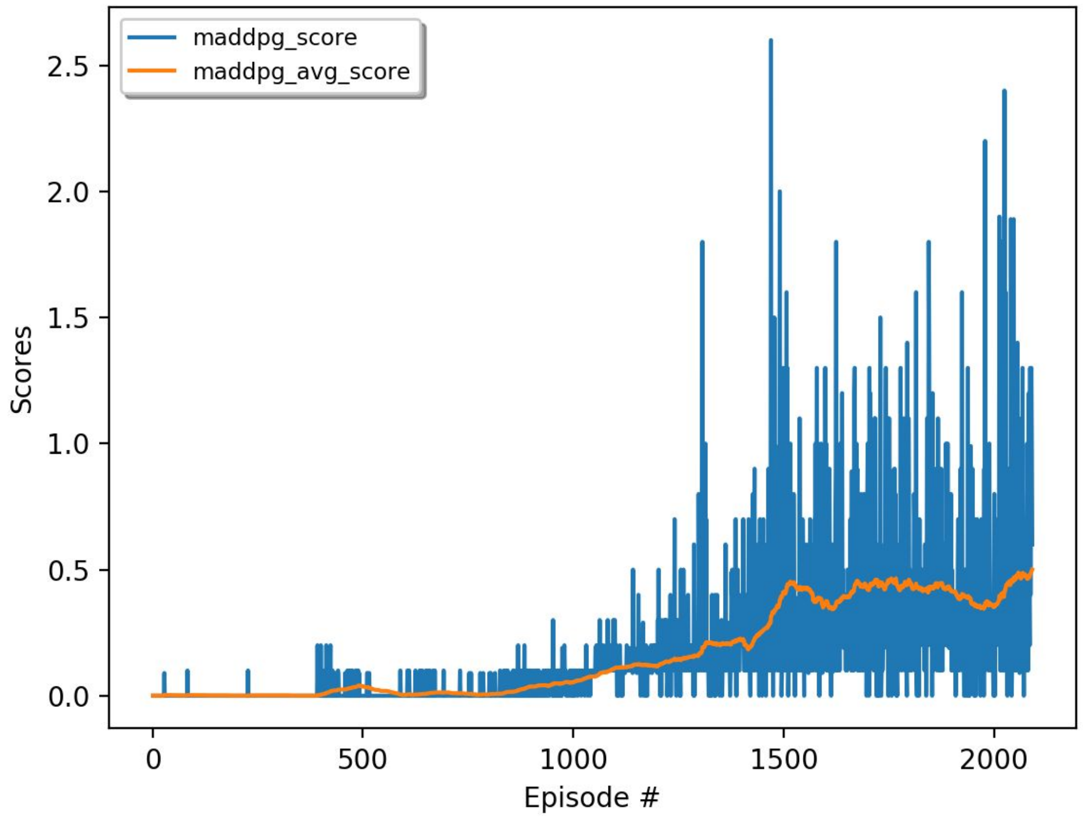
Critic architecture:

- 3 fully connected layers. [(state_size+action_size)*num_agents, 256, 128, 1]
- The first two layer are followed by ***Relu*** activation function

Features added to be able to learn properly:

- Don't fix the horizon since the task is episodic.
- Tweak the Sigma from the Ornstein-Uhlenbeck process. (**sigma=0.2**) and make sure that the noise sampling uses the standard normal distribution.
- Learning rate tuning. (**actor_lr= 1e-4** and **critic_lr=1e-3**)
- Make the Buffer shareable between agents.

3- Results:



We were able to solve the environment and reach the score of **0.5** over 100 episodes after almost **2091 episodes.**

4- Future work:

- Some fancy ideas can be applied using the state of the art research work in multi agent reinforcement learning. Those methods propose communication protocol that helps the agent to communicate and synchronize between them for better and faster learning.
- Further tweaking of the hyperparameters. (Deeper networks for Actor and critics as well as using learning rate annealing)
- Adding prioritized experience replay for less correlation.