Collaboration and Competition MARL
Udacity Deep RL Project Report

Overview:

Through this report we will introduce the work done in the third project of the Udacity Deep RL course.
We will present the project structure, after that the learning algorithm with the neural network and the hyper-parameters and finishing with showing the results and showing the future work that will be done.

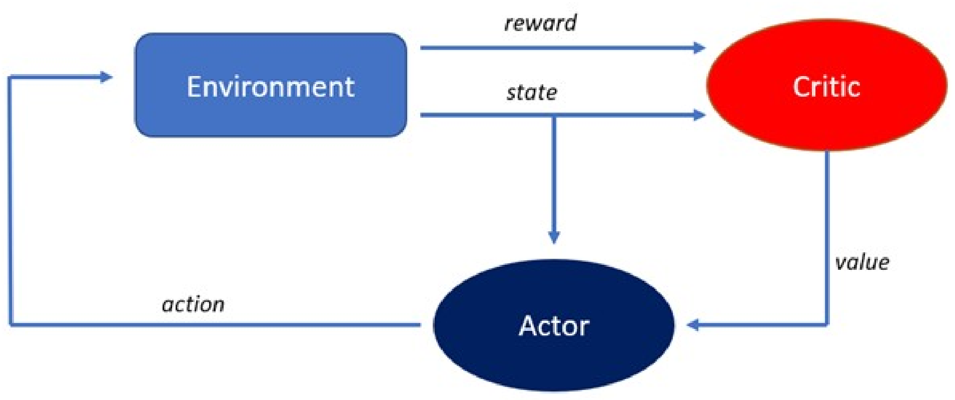
Project Structure:

- config.yaml: a configuration file where you could change the parameters.
- main.py: the main file which is used to run the project.
- trainer.py: the trainer file which is used to train the agent.
- evaluator.py: the evaluator file which is used to evaluate the agent.
- actor.py: contains the deep network actor created with pytorch.
- critic.py: contains the deep network critic created with pytorch.
- agent.py: contains the class Agent.
- replay_buffer.py: contains the ReplayBuffer class used in the DQN algorithm.
- utils.py: contains the utils used in the project.
- model.pth: the saved weights of the trained model.
- .pre-commit-config.yaml: contains pre-commit hoocks.

Learning Algorithm:

To solve this environment we have chosen the Deep Deterministic Policy Gradient **DDPG** Algorithm which is a reinforcement learning technique that combines both **Q-learning** and **Policy gradients**.
DDPG being an actor-critic technique consists of two models: Actor and Critic.

The Architecture of **DDPG** is illustrated in this figure:



DDPG Agent Hyper Parameters:

- BUFFER_SIZE (int=1e5): replay buffer size
- BATCH_SIZ (int=128): mini batch size
- GAMMA (float=0.99): discount factor
- TAU (float=1e-3): for soft update of target parameters
- LR_Actor (float=2e-4): learning rate for the actor optimizer
- LR_Critic (float=2e-4): learning rate for the critic optimizer
- n_episodes (int=5000): the maximum number of episodes
- device (str=cuda): the training is done on this device could be cuda (on gpu)

Actor Neural Network:

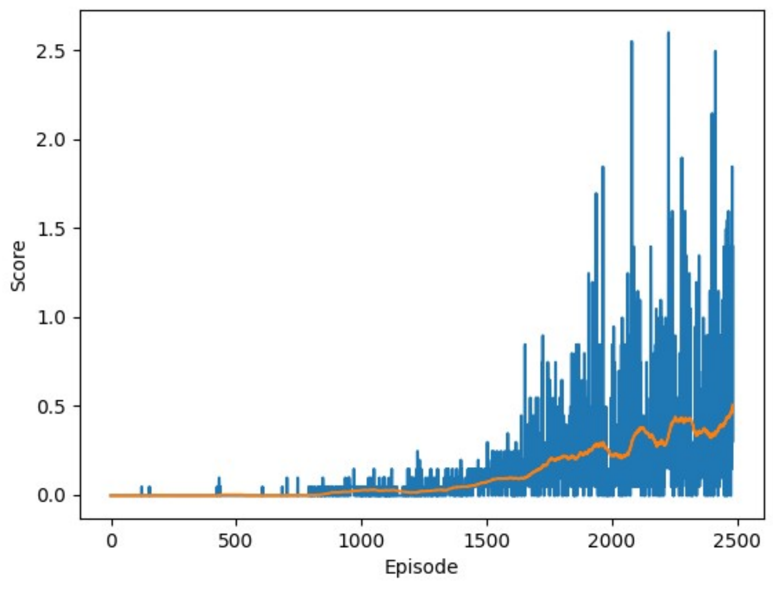
For the Critic we used a basic architecture consisting of **2 fully connected layer** (256,128) as number of units with **Relu** as an activation function for each one and followed by a final fully connected layer as an output with number of units equal to the number of actions.

Critic Neural Network:

For the Critic we used a basic architecture consisting of **2 fully connected layers** (256,128) as number of units with **Relu** as an activation function for each one and followed by a final fully connected layer as an output with number of units equal to 1.

Results:

This environment was solved after 2500 episodes and you could see the plot in the figure below:



Future work:

In order to improve the results we could try other algorithms like Distributed Distributional Deterministic Policy Gradients **(D4PG)**
Also we could make more hyper-parameter tuning by implementing some search algorithms.