Udacity Collaboration and Competition Project

1.Learning Algorithm:

1.1.Introduction:

This project was solved using the MADDPG algorithm : The multi-agent version of the Deep Deterministic Policy Gradient, which is an Actor-Critic method.

So, it’s an actor-critic, model-free algorithm based on the deterministic policy gradient that can operate over continuous action spaces. Let’s see its key aspects, especially with this multi-agent version.

1.2.Continuous action-spaces:

The DDPG algorithm is based on two key components:

- **Actor:** approximate the optimal policy deterministically.
- **Critic:** used to approximate the maximizer over Q-values of the next state.

The main difference between DDPG algorithm and the other actor-critic algorithms is the role of the critic network, which is generally used as a learned baseline in other AC methods.

And in this way, we can deal with actions having a continuous range, instead of just discrete actions like in DQN.

1.3.Replay Buffer:

Like DQN, DDPG uses also a replay buffer to address the problem of **moving targets.**

Actually, the TD target from the update rule of network weights is dependent on our approximation (current weights). And this can lead to harmful correlations!

The solution here is to decouple the TD target from weights' update. For this we will need two networks with identical architectures:

- **Primary Q-network**: its weights will be updated at every learning step
- **Target Q-network**: weights won't be updated at every learning step. The frequency of update is a hyperparameter.

This will be applied to both actor and critic networks.

Be careful, when we use MADDPG, the replay buffer will be shared between the agents. So, an agent can learn from experiments done by other agents.

1.4.Soft Updates:

The difference between DDPG and DQN is that in DDPG the target network weights are updated using a soft update strategy: Every step, mix in your regular network weights into your target network weights.

In practice, this update strategy will result in a faster convergence.

2.Hyperparameters:

This is the list of hyperparams used along with their values and definitions:

- **BUFFER_SIZE**: indicates the replay buffer size for experience replay, value used 100 000
- **BATCH_SIZE:** Minibatch size for training, value used 256
- **GAMMA:** Discount rate, value used 0.99
- **TAU**: soft update of target parameters, value used 0.001
- **LR_ACTOR**: Learning rate for actor network, value used 0.0001
- **LR_CRITIC**: Learning rate for critic network, value used 0.001
- **WEIGHT_DECAY:** L2 weight decay, but in my experiment it was okay to set to 0.0 and the agent converged.

3.Network Architecture:

3.1 Actor network

- Input layer: Fully connected layer with 24 units (state size is 8 but we stack 3 observations)
- 1st Hidden layer: Fully connected layer of 200 units, with relu activation.

- 2nd Hidden layer: Fully connected layer of 150 units, with relu activation.
- Output layer: Fully connected layer of 2 units with tanh activation(action size is 2).

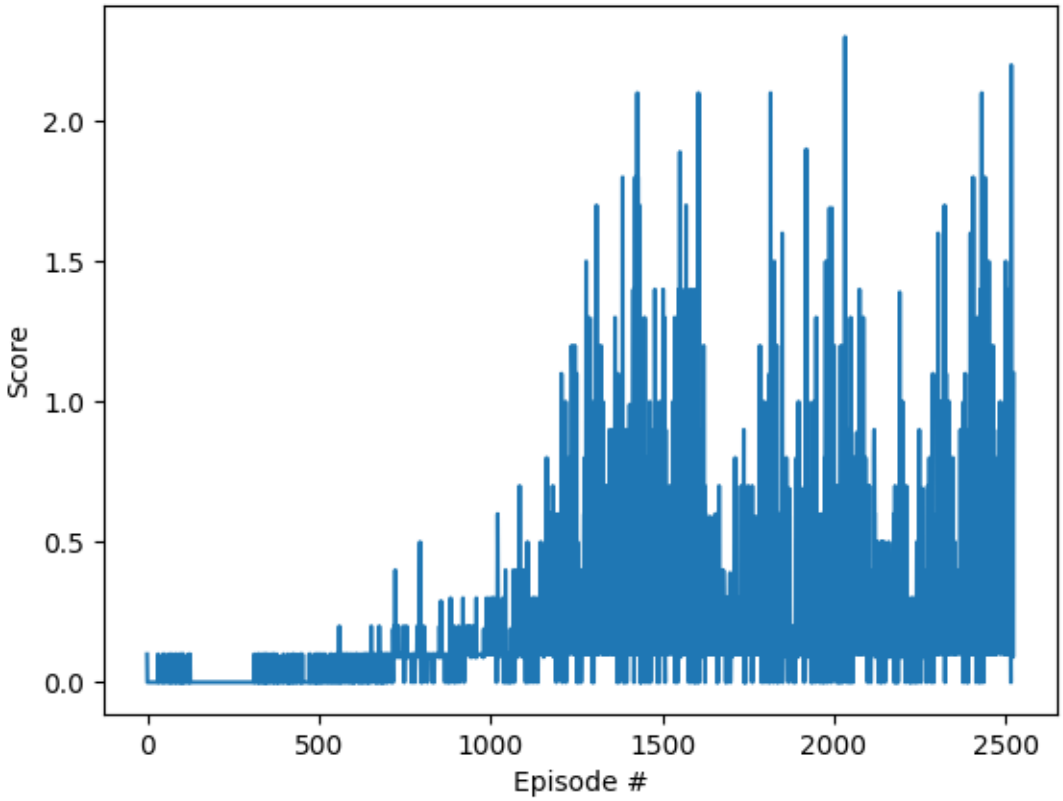
3.2 Critic network

- Input layer: Fully connected layer with 52 units ((state size(24) + action_size(2)) * num_agents(2))
- 1st Hidden layer: Fully connected layer of 200 units, with relu activation.
- 2nd Hidden layer: Fully connected layer of 150 units, with relu activation.
- Output layer: Fully connected layer of 1 unit (Maximizer of over Q-values)

4.Results:

The figure below shows the score evolution regarding the number of episodes done by the agent. As shown here, the problem was solved after about 2500 epsiodes.

PS: The environment is considered solved when the average score of an episode (maximum score of the two trained agents),over 100 consecutive episodes is at least +0.5



5.Ideas for future improvements:

DDPG algorithm can be improved via the use of a set of techniques like: The prioritized experience replay or extending DDPG to the bootstrapped actor-critic architecture to improve the efficiency of exploration. This applies also for the multi-agent version : MADDPG.