# Collaboration and Competition with MADDPG

In order to solve the collaboration and competition project[1], we have trained a MADDPG (Multi-agent Deep Deterministic Policy Gradient) model using Pytorch. In this report, we will elaborate the learning algorithm, training results and the potential for future work.

## I. Learning Algorithm

### 1. *Deep Deterministic Policy Gradient (DDPG)*

DDPG is a model free, off-policy actor-critic algorithm that can operate over high-dimensional, continuous action spaces (Lillicrap, Timothy P., et al, 2015). In DDPG, we use two deep neutral networks, a local and a target neural network for actor and critic respectively. The local network is the most up to date network because it is the one that is trained, while the target network is the one used for prediction to stabilize training. The actor is used to approximate the optimal policy deterministically and the critic learns to evaluate the optimal action value function by using the actors best believed action.

The parameterized actor function $\mu(s|\theta^{\mu})$ specifies the current policy by deterministically mapping states to a specific action. The critic $Q(s,a)$ is learned using the Bellman equation.

The actor weights $\theta^{\mu}$ are updated as follows,

$$\Delta_{\theta^{\mu}} J = -\mathbb{E}[\Delta_{\theta^{\mu}} Q(s,a|\theta^{Q})|_{s=s_t,a=\mu(s_t|\theta^{\mu})}]$$

$$\theta^{\mu} = \theta^{\mu} + \alpha\Delta_{\theta^{\mu}} J$$

where $J = -\mathbb{E}[Q(s,a|\theta^{Q})|_{s=s_t,a=\mu(s_t|\theta^{\mu})}]$ is the loss function and α is the learning rate.

In other words, the action predicted by the actor should be able to maximize the action value function estimated by the critic. Therefore, we use the local critic neural network to evaluate and update the weights of the local actor neural network.

The critic weights $\theta^{Q}$are updated as follows,

$$\Delta_{\theta^{Q}} L = -\mathbb{E}\{[Q(s_t,a_t|\theta^{Q}) - (r(s_t,a_t) + \gamma Q(s_{t+1},\mu(s_{t+1})|\theta^{Q})]^2\}$$

$$\theta^{Q} = \theta^{Q} + \beta\Delta_{\theta^{Q}} L$$

where $Q(s_t,a_t|\theta^{Q})$ represents the action value for $s_t$ estimated by the local critic neural network with the action $a_t$ outputted by the local actor neural network; And $r(s_t,a_t) + \gamma Q(s_{t+1},\mu(s_{t+1})|\theta^{Q})$ is the action value for $s_t$ estimated using temporal difference method by the target critic neural network with $\mu(s_{t+1})$ outputted by the target actor neural network. $\gamma$ is the discount rate and $\beta$ is the learning rate of the critical network.

[1] The introduction of the project can be found in the README.md file.

In other words, the local critic neural network should learn to converge to the target gradually. In the meantime, the iterative leaning process will push the local actor network to converge to the target actor network.

Additionally, DDPG employs experience replay to disentangle the correlation of consecutive actions. In our case, since we have multiple agents, each agent adds its experience to the replay buffer that is shared by all agents. Afterwards, $(N, M)$ experience tuples $(s_t, a_t, r_t, s_{t+1})$ are sampled from the replay buffer for training, where $N$ is the minibatch size and $M$ is the number of agents.

Furthermore, a soft update strategy is implemented in DDPG to stabilize the algorithm. This strategy can be described using the following equation:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

where $\theta'$ are the target network weights, $\theta$ are the local network weights and $\tau$ is the update rate.

Intuitively, the soft update strategy slowly blends the local network weights with the target network weights.

2. *MADDPG*

DDPG proves to be efficient in single agent domains or independent multi-agent domains, where modelling or predicting the behavior of other actors in the environment is unnecessary. However, when it comes to applications that involve interaction between multiple agents, DDPG appears to be less competent due to the following issues:

- The environment is non-stationary since each agent's policy is changing during training;
- Policy gradient methods suffer from high variance when coordination of multiple agents is required;
- Model based policy optimization is prone to instability when applied to competitive environments.

Our application involves both collaborative and competitive behaviors of two agents, therefore we resort to MADDPG. MADDPG is a framework of centralized training with decentralized execution. It extends DDPG by allowing the critic to use extra information about the policies of other agents, while the actor only has access to local information.

Technically, the input of the critic should contain the full state information and actions of both agents. Additionally, experience replay buffer should record experience of all agents instead of each agent individually.

3. *Neural Network*

As shown in Fig1, we have built a 3-layer plain neural network using Pytorch for the actor and the critic respectively[2]. The structure of the local and target network are the same, since the target network is only a copy of the local network at the beginning. The first pair of actor and critic is created for the first agent and the second pair if generated for the second agent.

The input size of the actor is 24, corresponding to the state size of each individual agent. The input size of the critic is 52, including the states (24 x 2 = 48) and actions (2 x 2 = 4) of both agents.

```
Actor network:
 Actor(
  (fc1): Linear(in_features=24, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=2, bias=True)
  (bn1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)

Critic network:
 Critic(
  (fc1): Linear(in_features=52, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=1, bias=True)
  (bn1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)

Actor network:
 Actor(
  (fc1): Linear(in_features=24, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=2, bias=True)
  (bn1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)

Critic network:
 Critic(
  (fc1): Linear(in_features=52, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=1, bias=True)
  (bn1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
```

Fig1 Neural network structures of the actor and the critic

For the actor network, there are 256 units (or nodes) in each of the two hidden layers. We have implemented the linear transformation followed by a normalization and Relu activation function in the first hidden layer. In the second hidden layer, we have only implemented linear transformation and Relu activation function. The output layer contains 2 units, corresponding to the action size. Instead of Relu, we employed tanh function after the linear transformation in the output layer, since the action values must be within (-1, 1).

For the critic network, there are also 256 units in each of the two hidden layers. Similar to the actor network, linear transformation, normalization and Relu activation function are utilized in the first hidden layer and normalization is omitted in the second hidden layer. The output layer contains only 1 unit, corresponding to the state action value.

[2] Note that the input layer is not counted.

Additionally, the weights in the hidden layers are initialized from uniform distributions $[-\frac{1}{\sqrt{f}}, \frac{1}{\sqrt{f}}]$, where $f$ is the fan-in of the layer. The weights of the output layers are initialized uniformly from $[-0.003, 0.003]$.

## II. Empirical Results and Technical Details

We have solved the project[3] with MADDPG and the following set of hyperparameters within 213 episodes as shown in Fig2.

Hyperparameters:

```
'BATCH_SIZE': 256          # minibatch size
'BUFFER_SIZE': 100000      # replay buffer size
'GAMMA': 0.99              # discount factor
'LR_ACTOR': 0.0001         # learning rate of the actor
'LR_CRITIC': 0.0001        # learning rate of the critic
'NOISE_DECAY': 0.99        # decay rate of the noise
'TAU': 0.01                # soft update rate of the target parameters
'UPDATE_EVERY': 1          # how often to update the network
'WEIGHT_DECAY': 1e-07      # L2 weight decay for critic network
```

In our case, to further stabilize the algorithm, we decayed the Ornstein-Uhlenbeck noise gradually with a factor of 0.99.

```
Training performance:
Episode 500     Max Score: 0.0900       Moving Average Score (lastest 100): 0.0423
Episode 1000    Max Score: 0.1000       Moving Average Score (lastest 100): 0.2229
Episode 1029    Max Score: 2.6000       Moving Average Score (lastest 100): 0.5162
Environment solved in 1029 episodes!    Moving Average Score (lastest 100): 0.5162
CPU times: user 19min 14s, sys: 25.1 s, total: 19min 39s
Wall time: 21min 9s
```

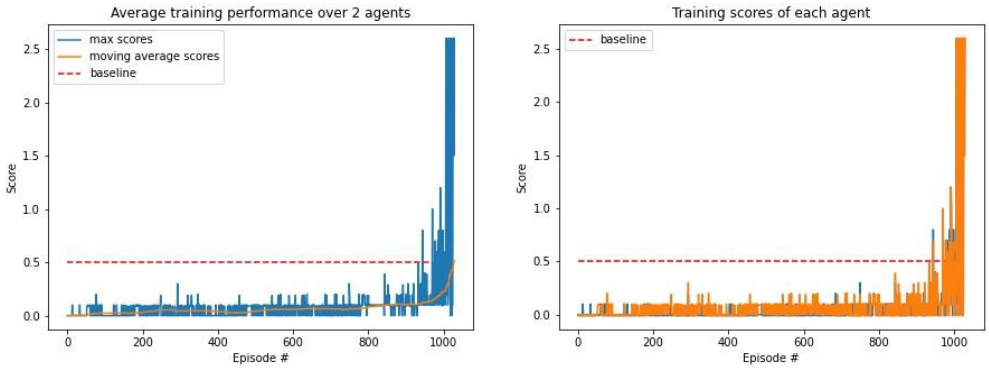


Fig2 Training performance

[3] The criterion to solve the project is to achieve at least 0.5 average scores over 100 consecutive episodes after taking the maximum over the two agents.

With 1 GPU, we have solved the project within 21 minutes. For users who only use CPU, the processing time could be longer. Please note that the max score shown in Fig2 represents the maximum score of two agents per episode and the moving average score stands for the average of the max score over different episodes (max. 100 consecutive episodes). We can observe from the subplot on the right hand side, which shows the training scores for each agent, that the agents share similar learning patterns.

## III. Outlook

In order to build a more robust model, one can train the agents with an ensemble of environments, requiring the average score across the parallel environments to fulfill the solving criterion. Due to the scope of current work, we would like to left it for future research.

## Reference


Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." *arXiv preprint arXiv:1509.02971* (2015).

Lowe, Ryan, et al. "Multi-agent actor-critic for mixed cooperative-competitive environments." *arXiv preprint arXiv:1706.02275* (2017).