# Report on Collaboration and Competition project

## Multi-Agent Deep Deterministic Policy Gradient (MADDPG)

Ryan et al. (2018) proposed a general-purpose multi-agent learning algorithm that is a simple extension of **actor-critic policy gradient methods** (specifically DDPG) where the critic is augmented with **extra information about the policies of other agents** while the actor **only has access to local information (i.e., its own observations)**. This framework adopts the use of the centralized training with the decentralized execution. This allows the policies to use extra information to ease training but this information is not used at test time. The primary motivation behind MADDPG is that, if the actions taken by all agents are known, then the environment is stationary even as the policies change [1].

MADDPG is comprised of two phases as shown in the diagram below [1]:

- **Training phase:** During training, the critic for each agent uses extra information like state's observed and actions taken by all the other agents. As for the actor, each actor has access to only its agent's observation and actions.
- **Execution phase:** After training is completed, only the local actors are used at execution phase, acting in a decentralized manner and equally applicable in cooperative and competitive settings. Learning critic for each agent allows us to use a different reward structure for each. Hence the algorithm can be used in all cooperative, competitive, and mixed scenarios.

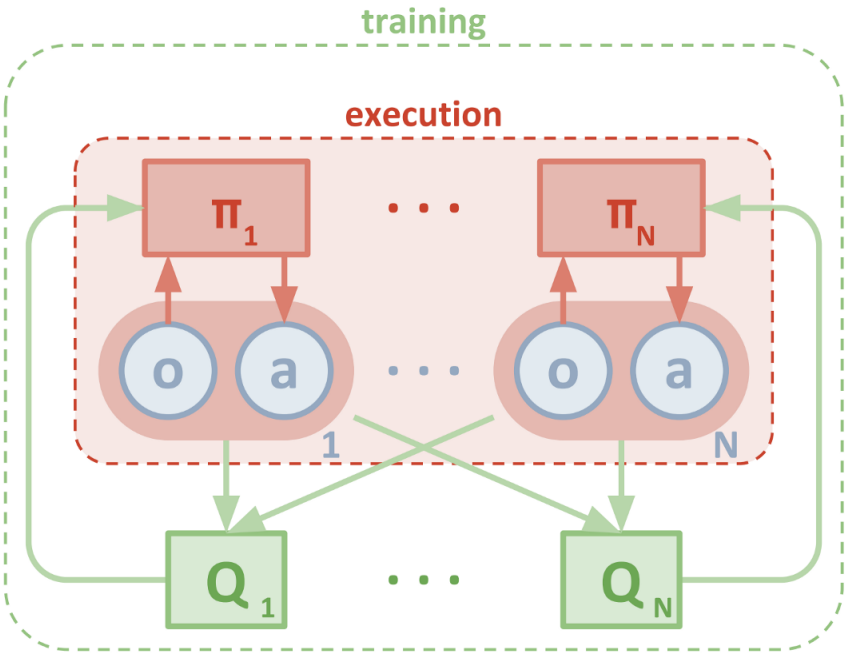



**Figure 1 Overview of multi-agent decentralized actor, centralized critic approach [1]**

## MADDPG Algorithm

MADDPG algorithm can be shown as below:

**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 $y^j = r_i^j + \gamma\, Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)\big|_{a'_k = \boldsymbol{\mu}'_k(o_k^j)}$
      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**

## Model Architecture

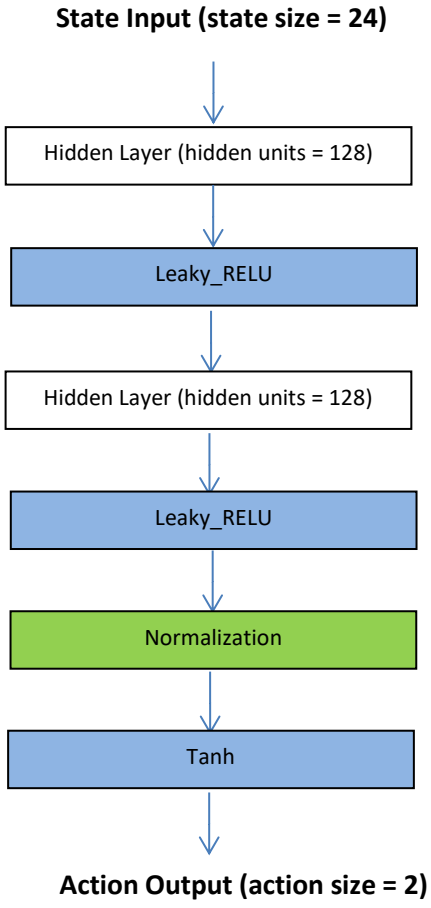
MADDPG is Actor-Critic model. It contains 2 networks each.

- The Actor network contains 2 hiddens layers with 128 hidden units each and leaky_relu activations.
- The Critic network also contains 2 hidden layers. However, state and action inputs to this network are merged between the first and second hidden layers. The intermediate activation is a leaky_relu activation unit, while the output activation is Linear. In case of Critic network, MADDPG should takes the states and actions of all agents.

### Actor Network

This represents the model architecture of the Actor network.

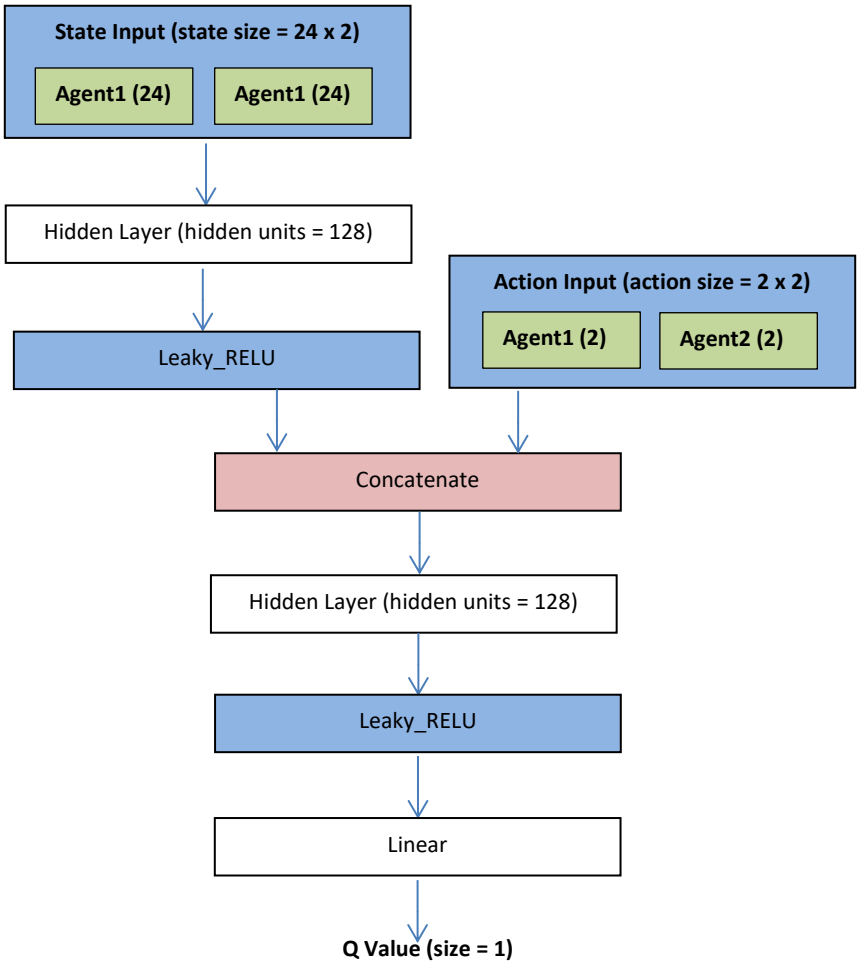
```
Actor(
  (fc1): Linear(in_features=24, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=2, bias=True)
)
```



## Critic Network

This represents the model architecture of the Critic network.

```
Critic(
  (fcs1): Linear(in_features=48, out_features=128, bias=True)
  (fc2): Linear(in_features=132, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=1, bias=True)
)
```

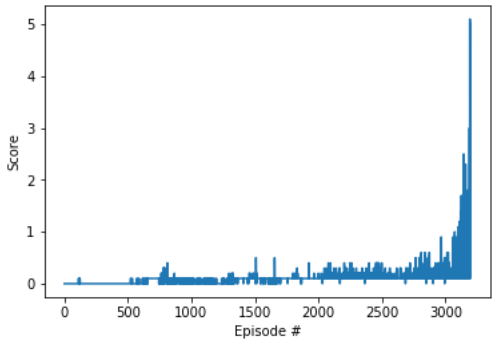


## Hyperparameters

| | | |
|---|---|---|
| BUFFER_SIZE | 1e5 | replay buffer size |
| BATCH_SIZE | 128 | minibatch size |
| GAMMA | 0.99 | Discount factor |
| TAU | 1e-3 | for soft update of target parameters |
| LR_ACTOR | 1e-5 | learning rate of the actor |
| LR_CRITIC | 1e-4 | learning rate of the critic |
| WEIGHT_DECAY | 0 | Weight Decay |
| Theta | 0.15 | how "fast" the variable reverts towards to the mean |
| Sigma | 0.2 | It is the degree of volatility of the process |
| UPDATE_NETWORK_FREQUENCY | 3 | how many steps the network is updated |

## Plot of Rewards

A plot of rewards per episode is included to illustrate the outcome at which the agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents).




**Environment solved in 3198 episodes! Average Score: 0.51**

## Ideas for Future Work

- Add more hidden layers into the network architecture of both actor and critic.
- Change the parameters' values of theta and sigma of the Ornstein-Uhlenbeck noise.
- Check several options for noise within MADDPG framework to make the policies exploration better:
  - Uncorrelated, mean-zero Gaussian noise can work perfectly well as it is also simple [3].
  - Scale the noise with an Epsilon hyper-parameter, and even reduce this parameter over time to implement a version of the epsilon-greedy algorithm.
- Use of multiple-agent with the implementation of using Proximal Policy Optimization (PPO) [4].

## References


1. https://arxiv.org/pdf/1706.02275.pdf
2. http://www.rohansawhney.io/multi-agent-rl.pdf

3. https://spinningup.openai.com/en/latest/algorithms/ddpg.html
4. https://arxiv.org/abs/1707.06347