1. Learning algorithm

1.1. Overview of the technique

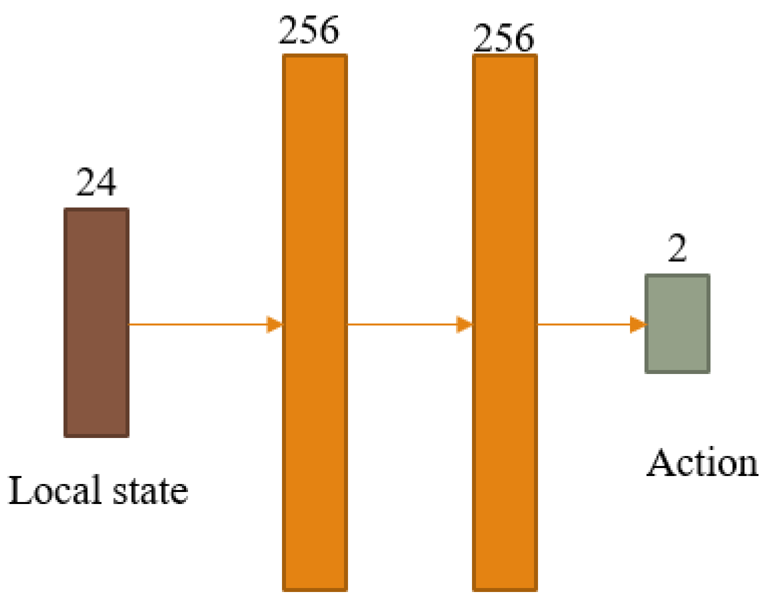
The agent is implemented using the Multi-agent Deep Deterministic Policy Gradient (MADDPG for short) method. The paper for this algorithm can be found here:
[1706.02275v4] Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments (arxiv.org)
This algorithm is a multi-agent version of the Deep deterministic policy gradient (DDPG). A quick recap, DDPG is an algorithm which concurrently learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the Q-function, and uses the Q-function to learn the policy. DDPG is an off-policy algorithm and can only be used in continuous action space.
MADDPG, or Multi-agent DDPG, extends DDPG into a multi-agent policy gradient algorithm where decentralized agents learn a centralized critic based on the observations and actions of all agents. It leads to learned policies that only use local information (i.e. their own observations) at execution time, does not assume a differentiable model of the environment dynamics or any particular structure on the communication method between agents, and is applicable not only to cooperative interaction but to competitive or mixed interaction involving both physical and communicative behavior. The critic is augmented with extra information about the policies of other agents, while the actor only has access to local information. After training is completed, only the local actors are used at execution phase, acting in a decentralized manner.

The algorithm in detail:

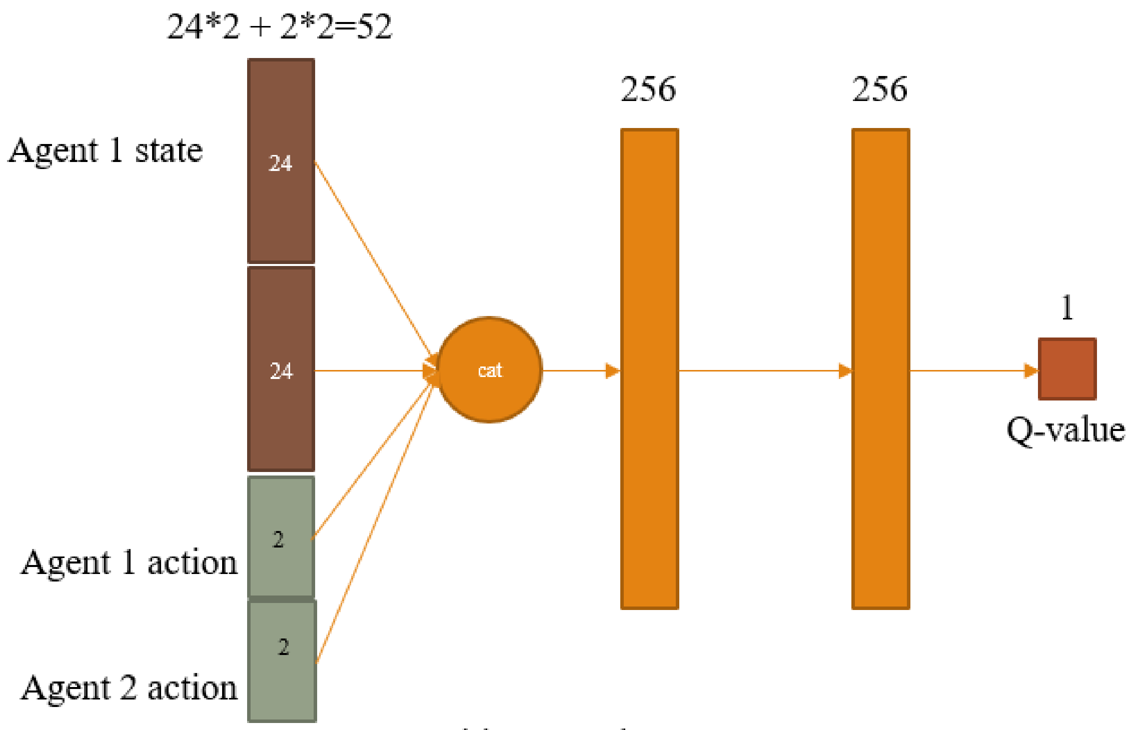
```
Algorithm 1: Multi-Agent Deep Deterministic Policy Gradient for $N$ agents
for episode $= 1$ to $M$ do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial state $\mathbf{x}$
    for $t = 1$ to max-episode-length do
        for each agent $i$, select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ 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
```

1.2. Network architecture

The network architecture for actor and critic are described below (the number is # neurons at each hidden layers)



Actor network structure



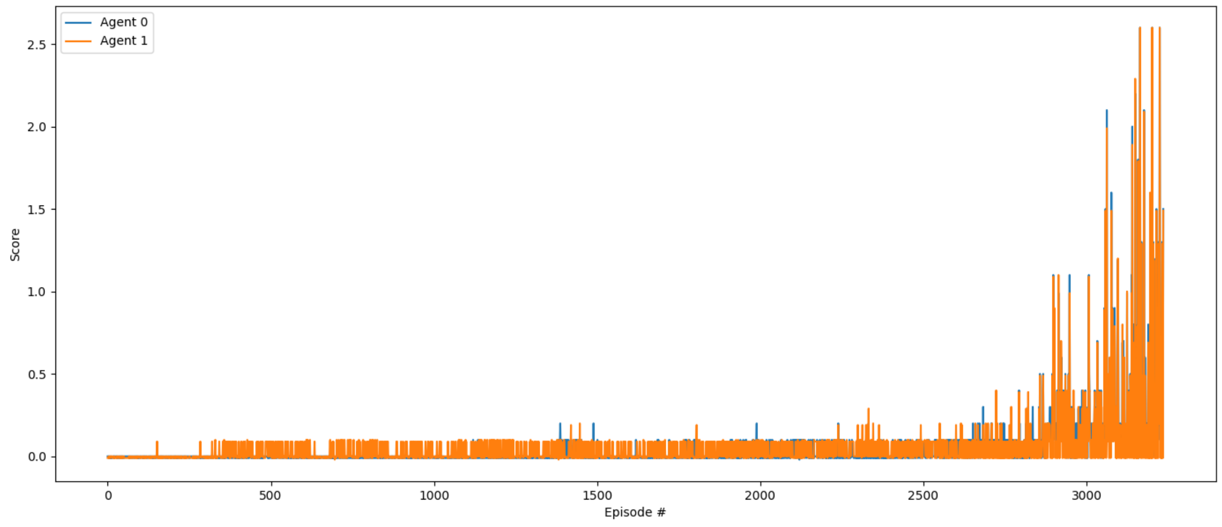
Critic network structure

: Concatnate

1.3. Hyperparameters select

Name	Value	Meaning
seed	37	Random seed for stable result
actor_lr	0.00001	Learning rate for actor’s Adam optimizer
critic_lr	0.0001	Learning rate for critic’s Adam optimizer
gamma	0.95	Discount factor
tau	0.01	Soft update target network hyperparameter
buffer_size	1000000	Number of transitions stored in replay buffer
batch_size	128	Batch size for learning phase of the agent

2. Plot of rewards



Plot of rewards in training. Blue line indicates accumulate reward of the first agent while the orange is for the other.

3. Ideas for future works

The result I have shown is acceptable, which solve the problem in around 3000 episodes. However, the result is a bit unstable during training. So with that said, I think some modifications will make the result even better:

- Tuning hyperparameters such as learning rate and the architecture of the actor and critic network probably enhance the learning ability of the agent
- Maybe I can do a multi-agent with this centralized-critic and decentralized-actor framework for other better single algorithms than DDPG such as TD3, PPO or SAC.