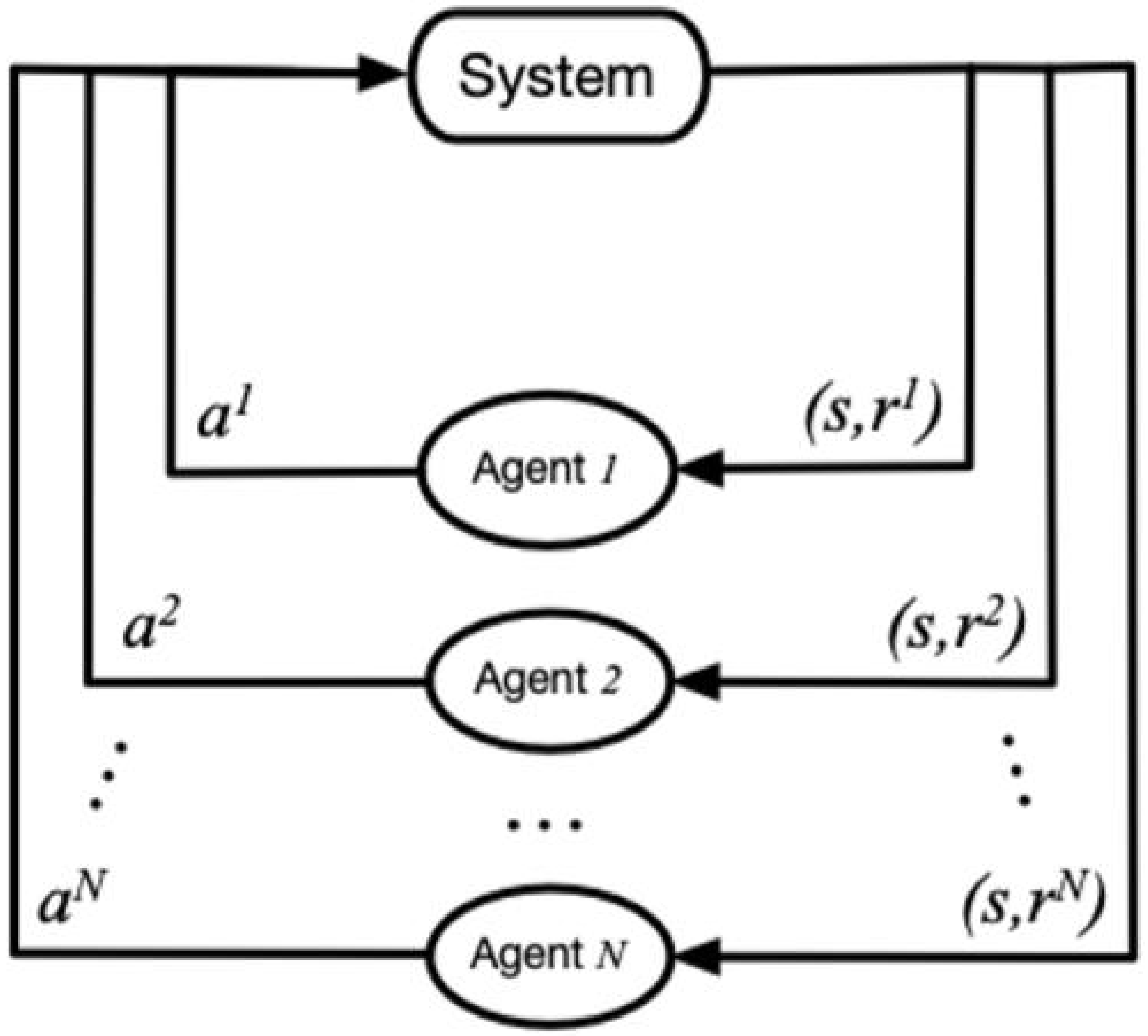
多智能体强化学习

金德才

2023

多智能体强化学习定义及建模



马尔科夫决策过程拓展到多智能体系统，被定义为马尔科夫博弈（又称为随机博弈，Markov/stochastic game）

- 棋类对弈
- 出租车派单
- 魔兽世界
- 自动驾驶
- 。。。

多智能体强化学习分类

- 完全合作式、完全竞争式、混合关系式
- 表格强化学习和函数近似（深度）多智能体强化学习
- 基于值的多智能体强化学习和基于策略的多智能体强化学习

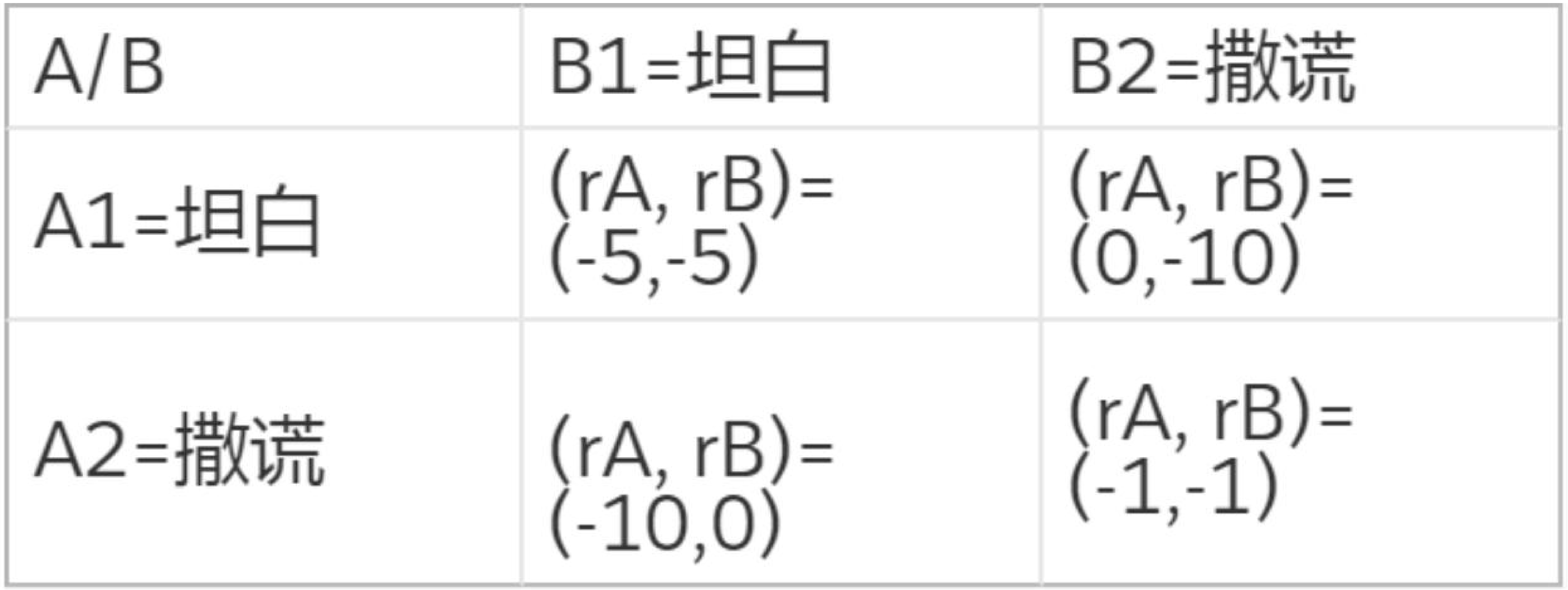
纳什均衡

纳什均衡（Nash equilibrium）:在多个智能体中达成的一个不动点，对于其中任意一个智能体来说，无法通过采取其他的策略来获得更高的累积回报

- 纳什均衡不一定是全局最优，但它是在概率上最容易产生的结果，是在学习时较容易收敛到的状态，特别是如果当前智能体无法知道其他智能体将会采取怎样的策略

- 如何对他人行为建模假设:
  - 最保险的策略意味着把别人想象的最坏，这样其他智能体策略的改变只会使自己变得更好
  - 认为别人选择对其自身最有利的动作，这种思想不如上一种稳定，但是有可能达到最优点

囚徒困境



A/B	B1=坦白	B2=撒谎
A1=坦白	(rA, rB)=(-5,-5)	(rA, rB)=(0,-10)
A2=撒谎	(rA, rB)=(-10,0)	(rA, rB)=(-1,-1)

很简单的矩阵博弈，也很能说明一些问题，比如纳什均衡不一定是帕累托最优

注：帕累托最优：帕累托最优（Pareto Optimality），也称为帕累托效率（Pareto efficiency），是指资源分配的一种理想状态，假定固有的一群人和可分配的资源，从一种分配状态到另一种状态的变化中，在没有使任何人境况变坏的前提下，使得至少一个人变得更好，这就是帕累托改进或帕累托最优化。 [1]
帕累托最优状态就是不可能再有更多的帕累托改进的余地；换句话说，帕累托改进是达到帕累托最优的路径和方法。 帕累托最优是公平与效率的“理想王国”。是由帕累托提出的

多智能体强化学习的问题和挑战

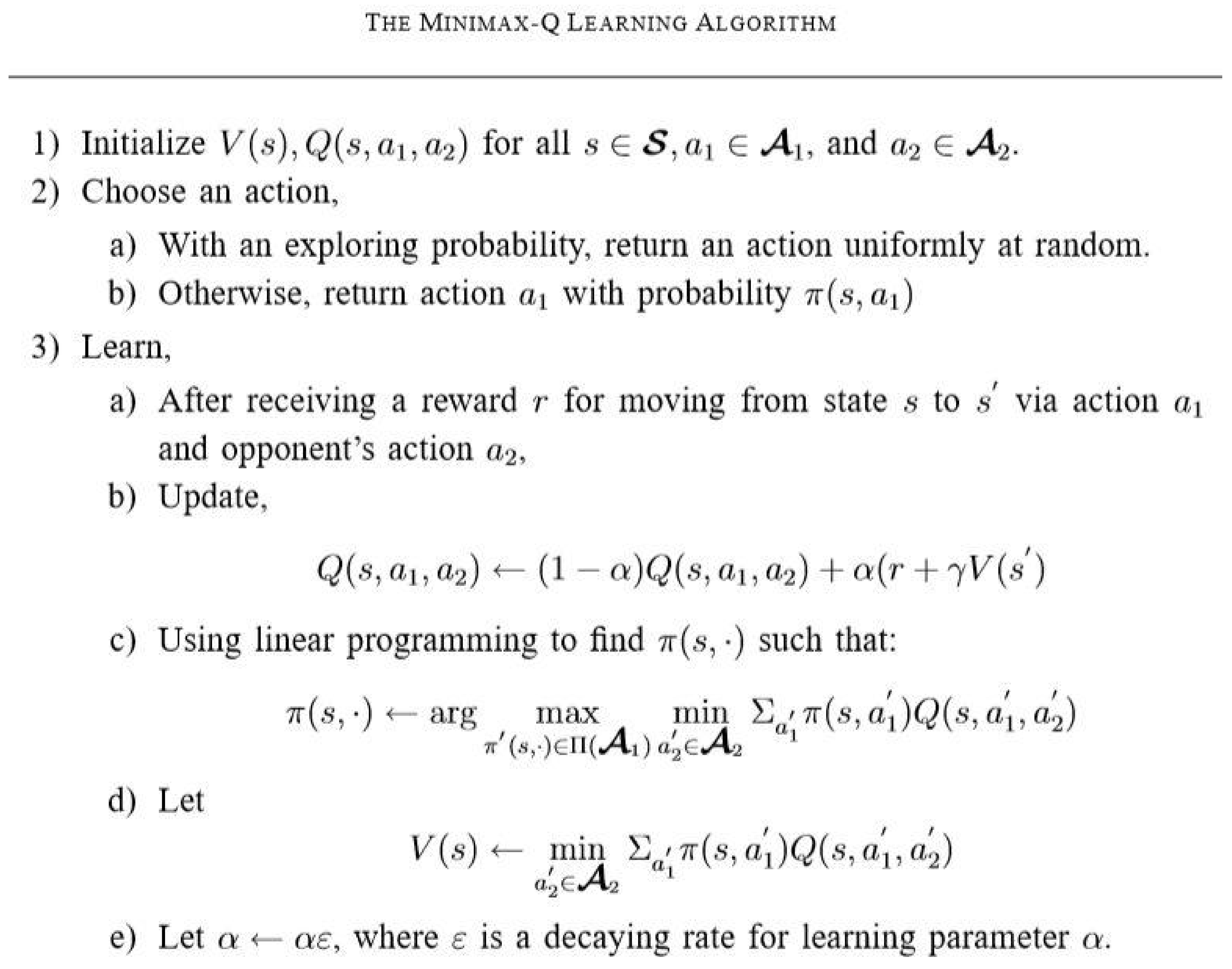
◆环境的不稳定性：智能体在做决策的同时，其他智能体也在采取动作；环境状态的变化与所有智能体的联合动作相关

◆智能体获取信息的局限性：不一定能够获得全局的信息，智能体仅能获取局部的观测信息，但无法得知其他智能体的观测信息、动作和奖励等信息

◆个体的目标一致性：各智能体的目标可能是最优的全局回报；也可能是各自局部回报的最优

◆可拓展性：在大规模的多智能体系统中，就会涉及到高维度的状态空间和动作空间，对于模型表达能力和真实场景中的硬件算力有一定的要求。对于大量的多个智能体，强化学习与规划算法的结合已经应用到实际中，此外智能优化算法（群体智能）也是一个很好的方向

minimax Q-Learning



THE MINIMAX-Q LEARNING ALGORITHM

1) Initialize $V(s), Q(s, a_1, a_2)$ for all $s \in \boldsymbol{\mathcal{S}}, a_1 \in \boldsymbol{\mathcal{A}}_1$, and $a_2 \in \boldsymbol{\mathcal{A}}_2$.
2) Choose an action,
    a) With an exploring probability, return an action uniformly at random.
    b) Otherwise, return action a_1 with probability $\pi(s, a_1)$
3) Learn,
    a) After receiving a reward r for moving from state s to $s^{'}$ via action a_1 and opponent's action a_2,
    b) Update,

$$Q(s, a_1, a_2) \leftarrow (1 - \alpha)Q(s, a_1, a_2) + \alpha(r + \gamma V(s^{'})$$

    c) Using linear programming to find $\pi(s, \cdot)$ such that:

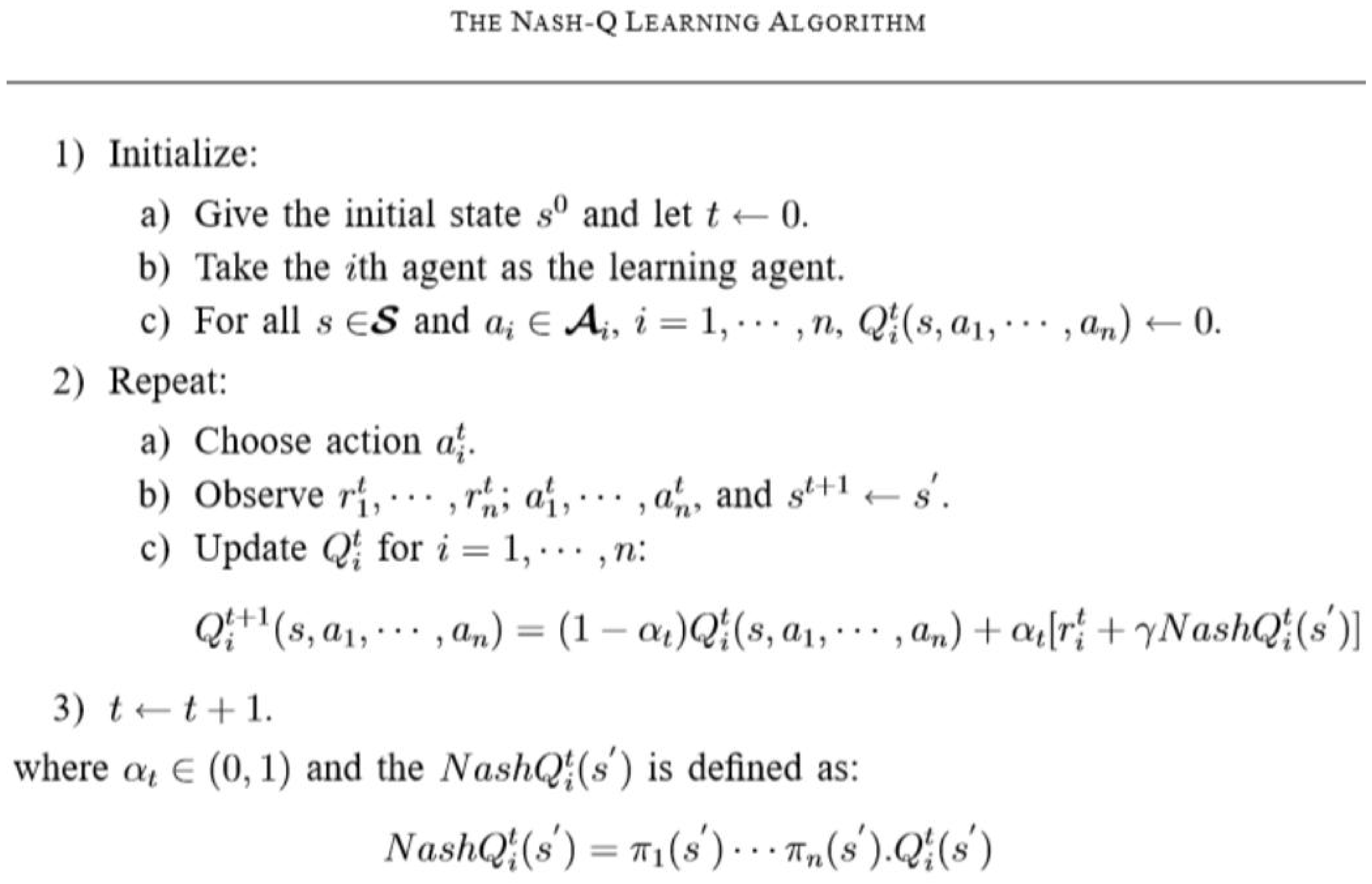
$$\pi(s, \cdot) \leftarrow \arg \max_{\pi^{'}(s,\cdot) \in \Pi(\boldsymbol{\mathcal{A}}_1)} \min_{a_2^{'} \in \boldsymbol{\mathcal{A}}_2} \Sigma_{a_1^{'}} \pi(s, a_1^{'}) Q(s, a_1^{'}, a_2^{'})$$

    d) Let

$$V(s) \leftarrow \min_{a_2^{'} \in \boldsymbol{\mathcal{A}}_2} \Sigma_{a_1^{'}} \pi(s, a_1^{'}) Q(s, a_1^{'}, a_2^{'})$$

    e) Let $\alpha \leftarrow \alpha\varepsilon$, where ε is a decaying rate for learning parameter α.

Nash Q-learning



THE NASH-Q LEARNING ALGORITHM

```
1) Initialize:
    a) Give the initial state $s^0$ and let $t \leftarrow 0$.
    b) Take the $i$th agent as the learning agent.
    c) For all $s \in \boldsymbol{\mathcal{S}}$ and $a_i \in \boldsymbol{\mathcal{A}}_i$, $i = 1, \cdots, n$, $Q_i^t(s, a_1, \cdots, a_n) \leftarrow 0$.
2) Repeat:
    a) Choose action $a_i^t$.
    b) Observe $r_1^t, \cdots, r_n^t$; $a_1^t, \cdots, a_n^t$, and $s^{t+1} \leftarrow s'$.
    c) Update $Q_i^t$ for $i = 1, \cdots, n$:
        $Q_i^{t+1}(s, a_1, \cdots, a_n) = (1 - \alpha_t) Q_i^t(s, a_1, \cdots, a_n) + \alpha_t [r_i^t + \gamma NashQ_i^t(s')]$
3) $t \leftarrow t + 1$.
```

where $\alpha_t \in (0, 1)$ and the $NashQ_i^t(s')$ is defined as:

$$NashQ_i^t(s') = \pi_1(s') \cdots \pi_n(s') . Q_i^t(s')$$

在Minimax-Q算法中需要通过Minimax线性规划求解阶段博弈的纳什均衡点，拓展到Nash Q-Learning算法就是使用二次规划求解纳什均衡点

Team Q-learning

$$h_i^*(x) = argmax_{u_i} max_{u_1,\ldots,u_{i-1},u_{i+1},\ldots,u_n} Q^*(x,u)$$

当所有智能体的联合最优动作是唯一的时候，完成该任务是不需要协作机制的，考虑全局最优动作组合就是可行的

联合动作学习与频率最大 Q 值

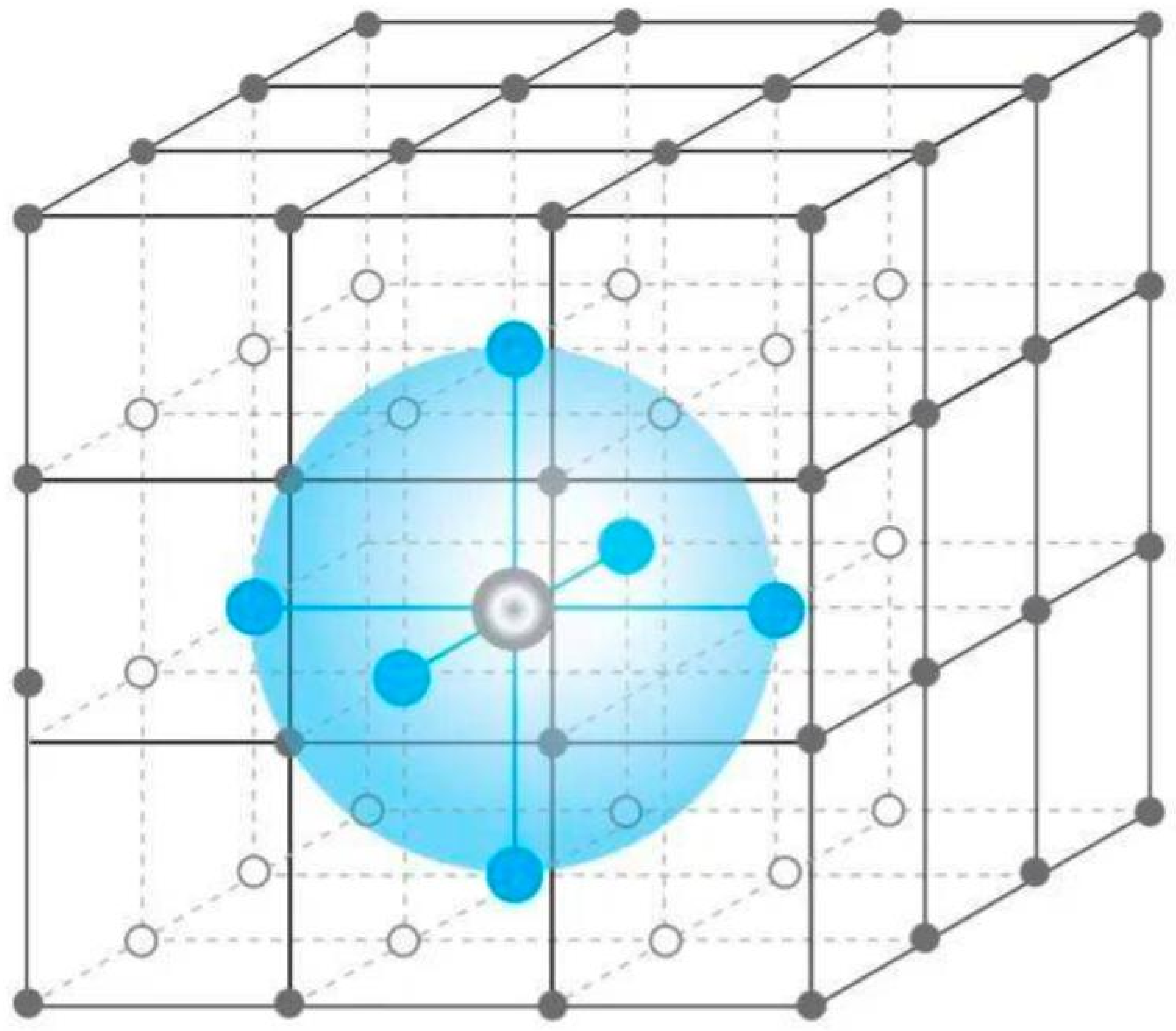
联合动作学习（joint action learner，JAL）

智能体 i 会基于观察到的其他智能体 j 的历史动作、对其他智能体 j 的策略进行建模。

频率最大 Q 值（frequency maximum Q-value, FMQ）

动作组合结果最优的概率，在个体 Q 值的定义中引入了个体动作所在的联合动作取得最优回报的频率，从而在学习过程中引导智能体选择能够取得最优回报的联合动作中的自身动作，那么所有智能体的最优动作组合被选择的概率也会更高。而Team Q-learning则是对应一个确定的动作组合

Mean Field MARL -- 大规模多智能体学习



当智能体数量非常多的时候，简化智能体相互之间的关系是十分重要的，MFRL将传统强化学习方法（Q-learning）和平均场理论（mean field theory）相结合。将所有其他智能体或着周围的智能体看作一个“平均场”，个体与其他智能体之间的关系可以描述为个体和平均场之间的相互影响，从而简化了后续的分析过程

MFMARL实现

$$Q_{t+1}^j(s,a^j,\bar{a}^j) = (1-\alpha)Q_t^j(s,a^j,\bar{a}^j) + \alpha[r^j + \gamma v_t^j(s')] \,, \tag{9}$$

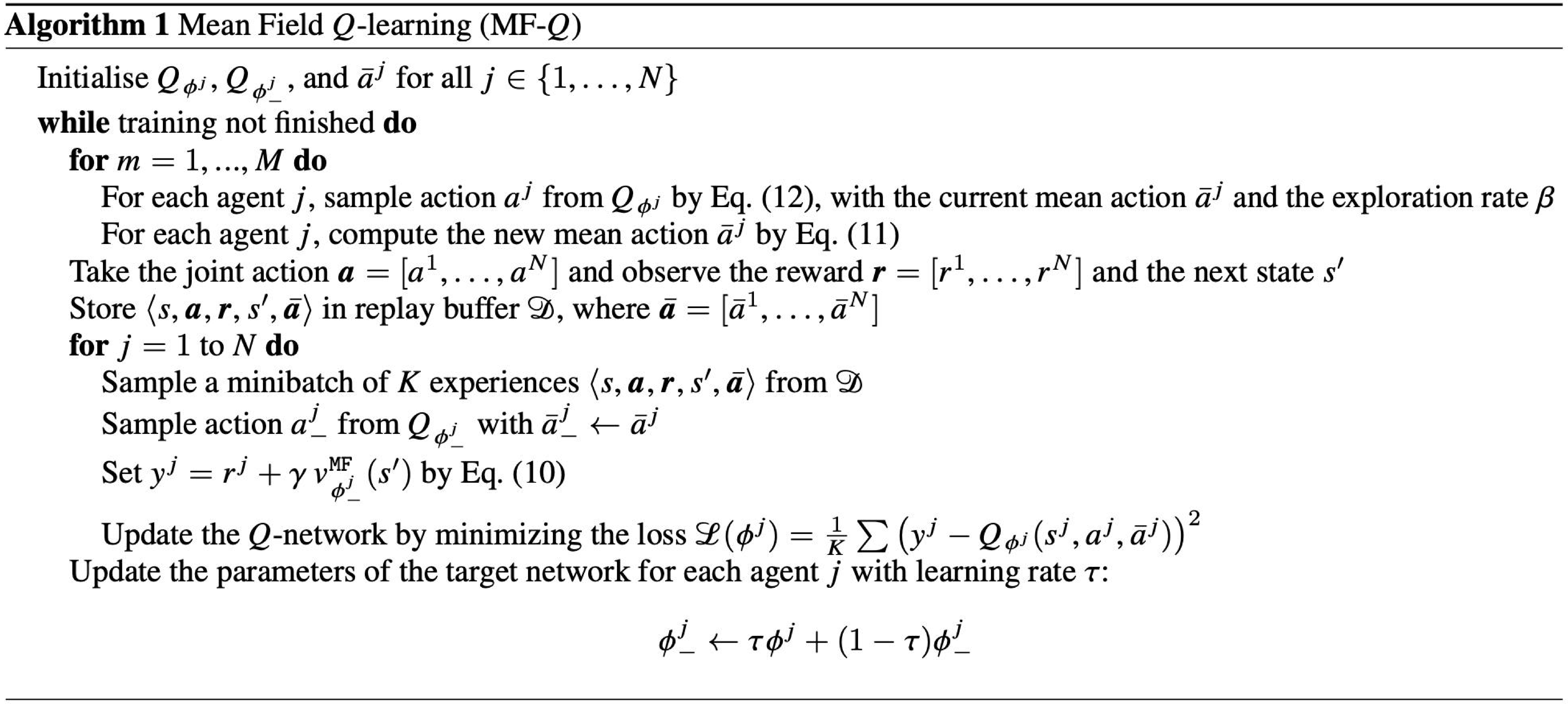
$$v_t^j(s') = \sum_{a^j} \pi_t^j(a^j|s',\bar{a}^j)\mathbb{E}_{\bar{a}^j(\boldsymbol{a}^{-j})\sim\boldsymbol{\pi}_t^{-j}}\Big[Q_t^j(s',a^j,\bar{a}^j)\Big], \tag{10}$$

$$\bar{a}^j = \frac{1}{N^j}\sum_k a^k,\ a^k \sim \pi_t^k(\cdot|s,\bar{a}_-^k)\,, \tag{11}$$

$$\pi_t^j(a^j|s,\bar{a}^j) = \frac{\exp\big(\beta Q_t^j(s,a^j,\bar{a}^j)\big)}{\sum_{a^{j'}\in\mathcal{A}^j}\exp\big(\beta Q_t^j(s,a^{j'},\bar{a}^j)\big)}\,. \tag{12}$$

- 将各个智能体的动作建模成one-hot 向量，这样就可以求平均。这里智能体邻居动作的均值是根据上次动作后算出来的
- 这里一个假设就是周边智能体的动作每次变化不是很大，所以平均才不会很大，能够收敛到纳什均衡策略。

MFMARL - Q



Algorithm 1 Mean Field Q-learning (MF-Q)

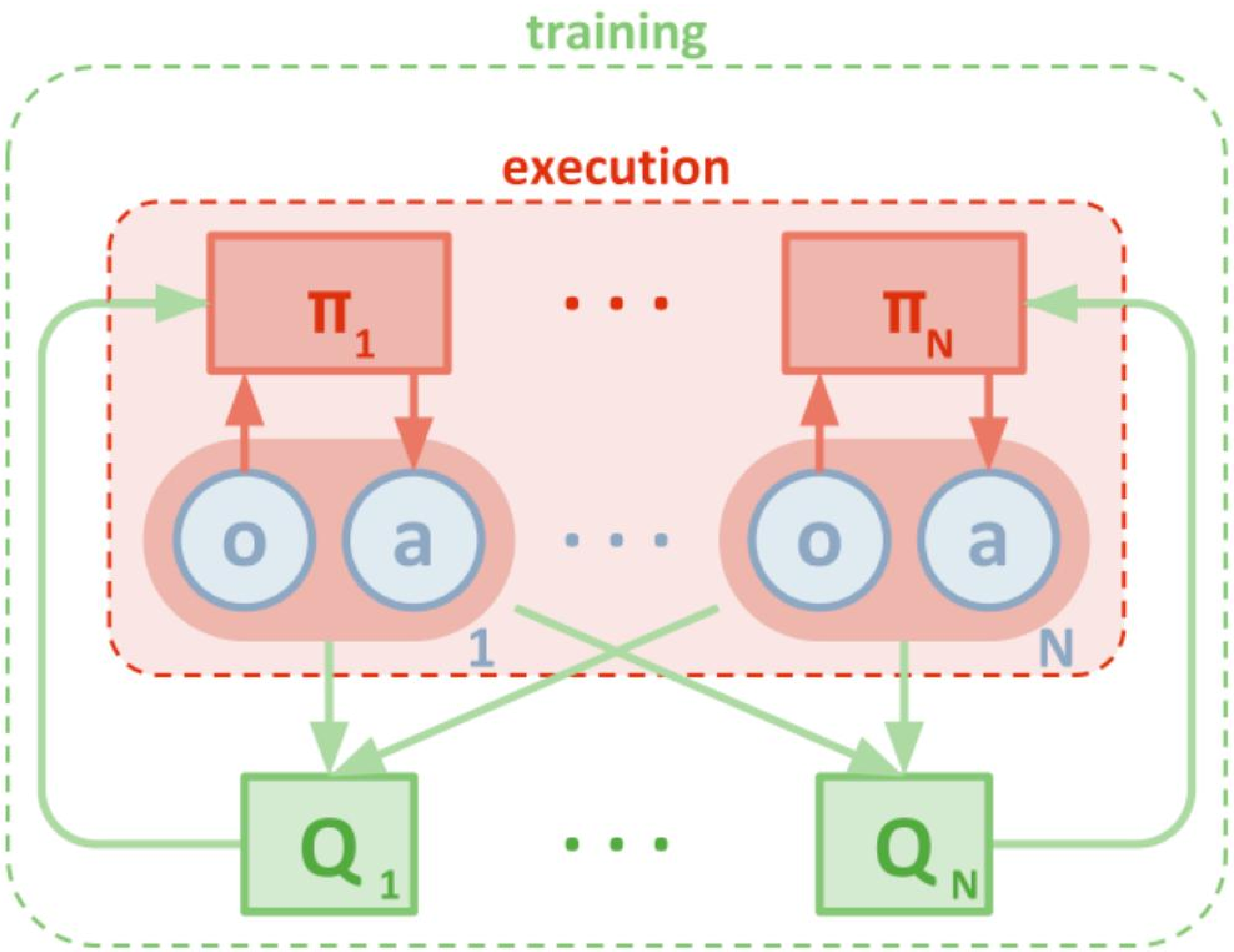
```
Initialise $Q_{\phi^j}$, $Q_{\phi_-^j}$, and $\bar{a}^j$ for all $j \in \{1, \ldots, N\}$
while training not finished do
    for $m = 1, \ldots, M$ do
        For each agent $j$, sample action $a^j$ from $Q_{\phi^j}$ by Eq. (12), with the current mean action $\bar{a}^j$ and the exploration rate $\beta$
        For each agent $j$, compute the new mean action $\bar{a}^j$ by Eq. (11)
    Take the joint action $\boldsymbol{a} = [a^1, \ldots, a^N]$ and observe the reward $\boldsymbol{r} = [r^1, \ldots, r^N]$ and the next state $s'$
    Store $\langle s, \boldsymbol{a}, \boldsymbol{r}, s', \bar{\boldsymbol{a}} \rangle$ in replay buffer $\mathcal{D}$, where $\bar{\boldsymbol{a}} = [\bar{a}^1, \ldots, \bar{a}^N]$
    for $j = 1$ to $N$ do
        Sample a minibatch of $K$ experiences $\langle s, \boldsymbol{a}, \boldsymbol{r}, s', \bar{\boldsymbol{a}} \rangle$ from $\mathcal{D}$
        Sample action $a_-^j$ from $Q_{\phi_-^j}$ with $\bar{a}_-^j \leftarrow \bar{a}^j$
        Set $y^j = r^j + \gamma \, v_{\phi_-^j}^{\mathrm{MF}}(s')$ by Eq. (10)
        Update the $Q$-network by minimizing the loss $\mathcal{L}(\phi^j) = \frac{1}{K} \sum \left( y^j - Q_{\phi^j}(s^j, a^j, \bar{a}^j) \right)^2$
    Update the parameters of the target network for each agent $j$ with learning rate $\tau$:

        $$\phi_-^j \leftarrow \tau \phi^j + (1 - \tau) \phi_-^j$$
```

MFRL - AC

Algorithm 2 Mean Field Actor-Critic (MF-AC)

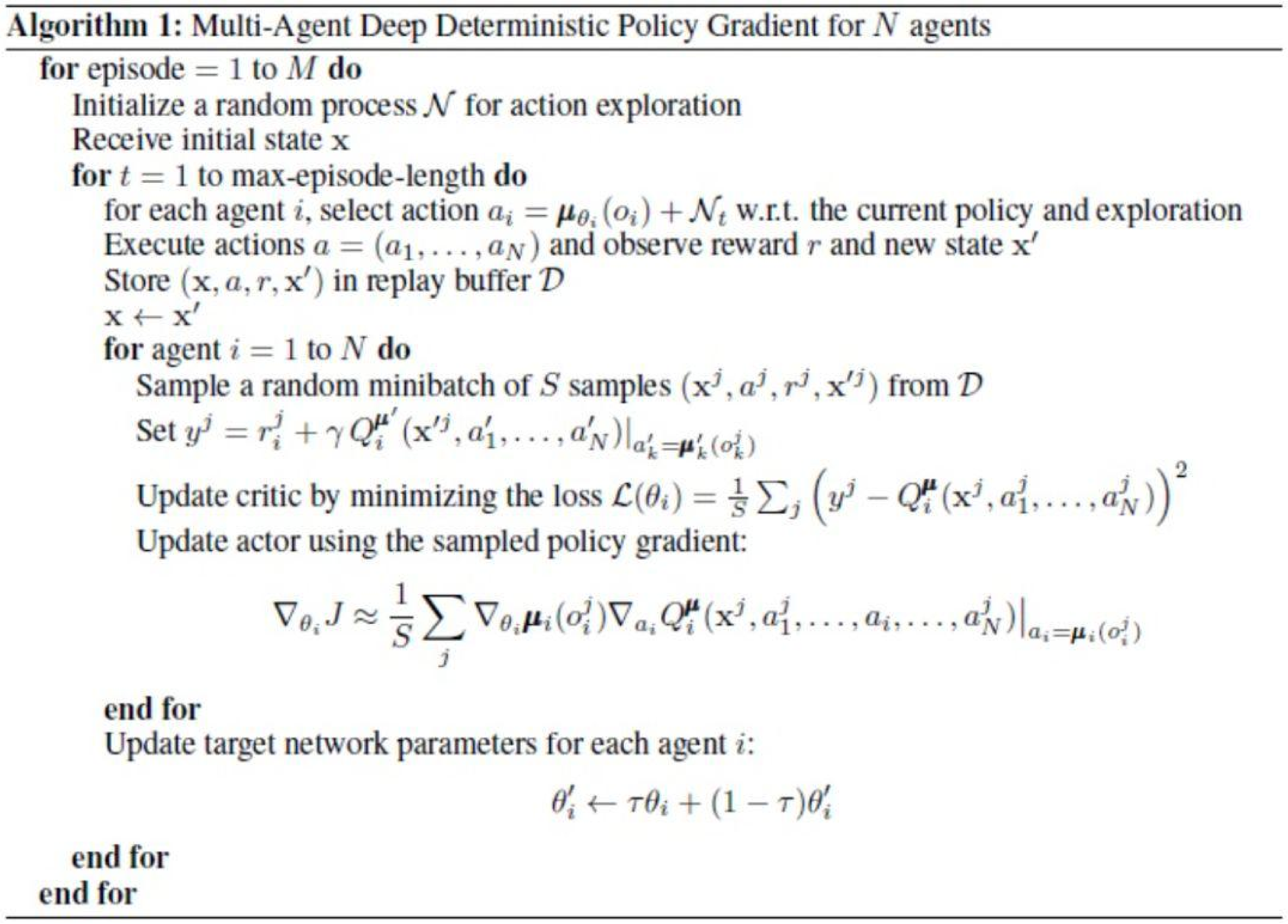
```
Initialize $Q_{\phi^j}, Q_{\phi_-^j}, \pi_{\theta^j}, \pi_{\theta_-^j}$, and $\bar{a}^j$ for all $j \in \{1, \ldots, N\}$
while training not finished do
  For each agent $j$, sample action $a^j = \pi_{\theta^j}(s)$; compute the new mean action $\bar{\boldsymbol{a}} = [\bar{a}^1, \ldots, \bar{a}^N]$
  Take the joint action $\boldsymbol{a} = [a^1, \ldots, a^N]$ and observe the reward $\boldsymbol{r} = [r^1, \ldots, r^N]$ and the next state $s'$
  Store $\langle s, \boldsymbol{a}, \boldsymbol{r}, s', \bar{\boldsymbol{a}} \rangle$ in replay buffer $\mathcal{D}$
  for $j = 1$ to $N$ do
    Sample a minibatch of $K$ experiences $\langle s, \boldsymbol{a}, \boldsymbol{r}, s', \bar{\boldsymbol{a}} \rangle$ from $\mathcal{D}$
    Set $y^j = r^j + \gamma\, v^{\mathrm{MF}}_{\phi_-^j}(s')$ by Eq. ($\boxed{10}$)
    Update the critic by minimizing the loss $\mathcal{L}(\phi^j) = \frac{1}{K} \sum \left( y^j - Q_{\phi^j}(s, a^j, \bar{a}^j) \right)^2$
    Update the actor using the sampled policy gradient:
      $$\nabla_{\theta^j} \mathcal{J}(\theta^j) \approx \frac{1}{K} \sum \nabla_{\theta^j} \log \pi_{\theta^j}(s') Q_{\phi_-^j}(s', a_-^j, \bar{a}_-^j) \Big|_{a_-^j = \pi_{\theta_-^j}(s')}$$
  Update the parameters of the target networks for each agent $j$ with learning rates $\tau_\phi$ and $\tau_\theta$:
      $$\phi_-^j \leftarrow \tau_\phi \phi^j + (1 - \tau_\phi) \phi_-^j$$
      $$\theta_-^j \leftarrow \tau_\theta \theta^j + (1 - \tau_\theta) \theta_-^j$$
```

MADDPG



CentralizedTraining Decentralized Execution($\boldsymbol{C}$ $\boldsymbol{D}$ $\boldsymbol{E}$) 缓解多智能体系统环境不稳定的问题

MADDPG算法



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
```

MADDPG 行为建模

$$\mathcal{L}(\phi_i^j) = -\mathbb{E}_{o_j, a_j}\left[\log \hat{\boldsymbol{\mu}}_i^j(a_j|o_j) + \lambda H(\hat{\boldsymbol{\mu}}_i^j)\right]$$

$$\hat{y} = r_i + \gamma Q_i^{\boldsymbol{\mu}'}(\mathbf{x}', \hat{\boldsymbol{\mu}}_i'^1(o_1), \ldots, \boldsymbol{\mu}_i'(o_i), \ldots, \hat{\boldsymbol{\mu}}_i'^N(o_N))$$

MADDPG 策略集成

$$J_e(\boldsymbol{\mu}_i) = \mathbb{E}_{k\sim\text{unif}(1,K), s\sim p^{\boldsymbol{\mu}}, a\sim\boldsymbol{\mu}_i^{(k)}}\left[R_i(s,a)\right]$$

$$\nabla_{\theta_i^{(k)}} J_e(\boldsymbol{\mu}_i) = \frac{1}{K}\mathbb{E}_{\mathbf{x},a\sim\mathcal{D}_i^{(k)}}\left[\nabla_{\theta_i^{(k)}}\boldsymbol{\mu}_i^{(k)}(a_i|o_i)\nabla_{a_i}Q^{\boldsymbol{\mu}_i}(\mathbf{x},a_1,\ldots,a_N)\Big|_{a_i=\boldsymbol{\mu}_i^{(k)}(o_i)}\right]$$

- 为了应对单个智能体的策略对其他智能体的策略过拟合以及提升训练稳定性，论文中应用了策略集成
- 对于单个智能体 i，它的策略μi 是由多个子策略μi^k 构成的集合。在一个 episode 中，只使用一种从集合中采样得到的子策略进行决策和完成交互。在学习过程中最大化的目标是所有子策略的期望回报

Counterfactual Multi-Agent Policy Gradients(COMA)

反事实多智能体策略梯度法方法

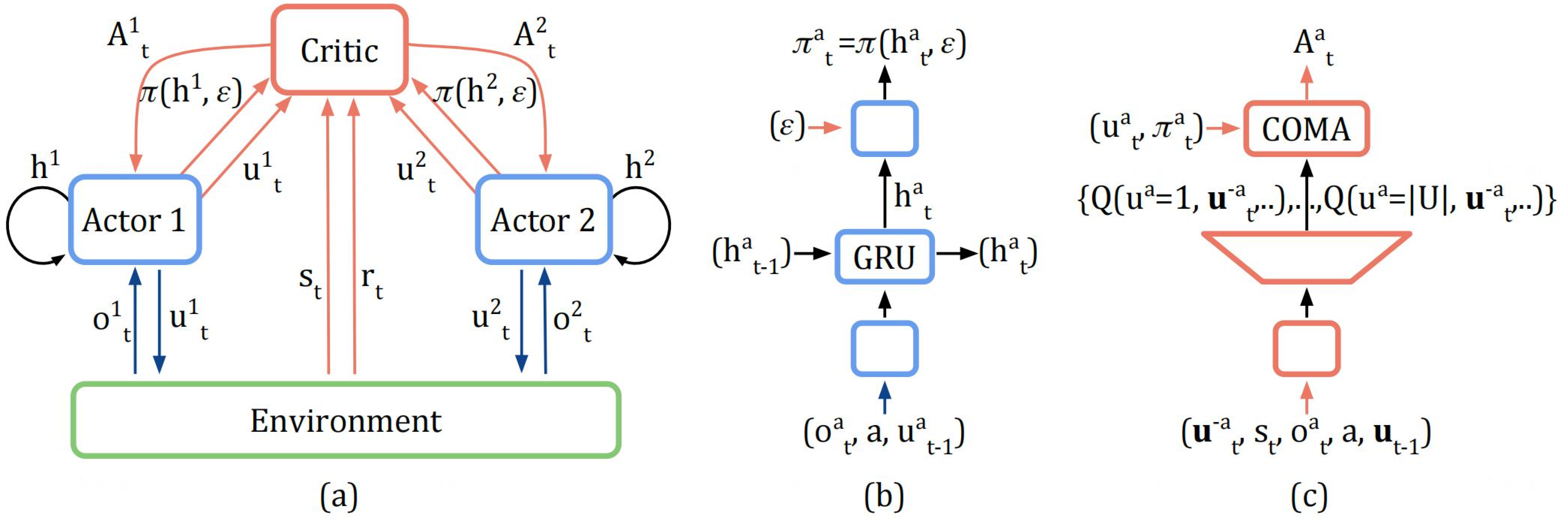
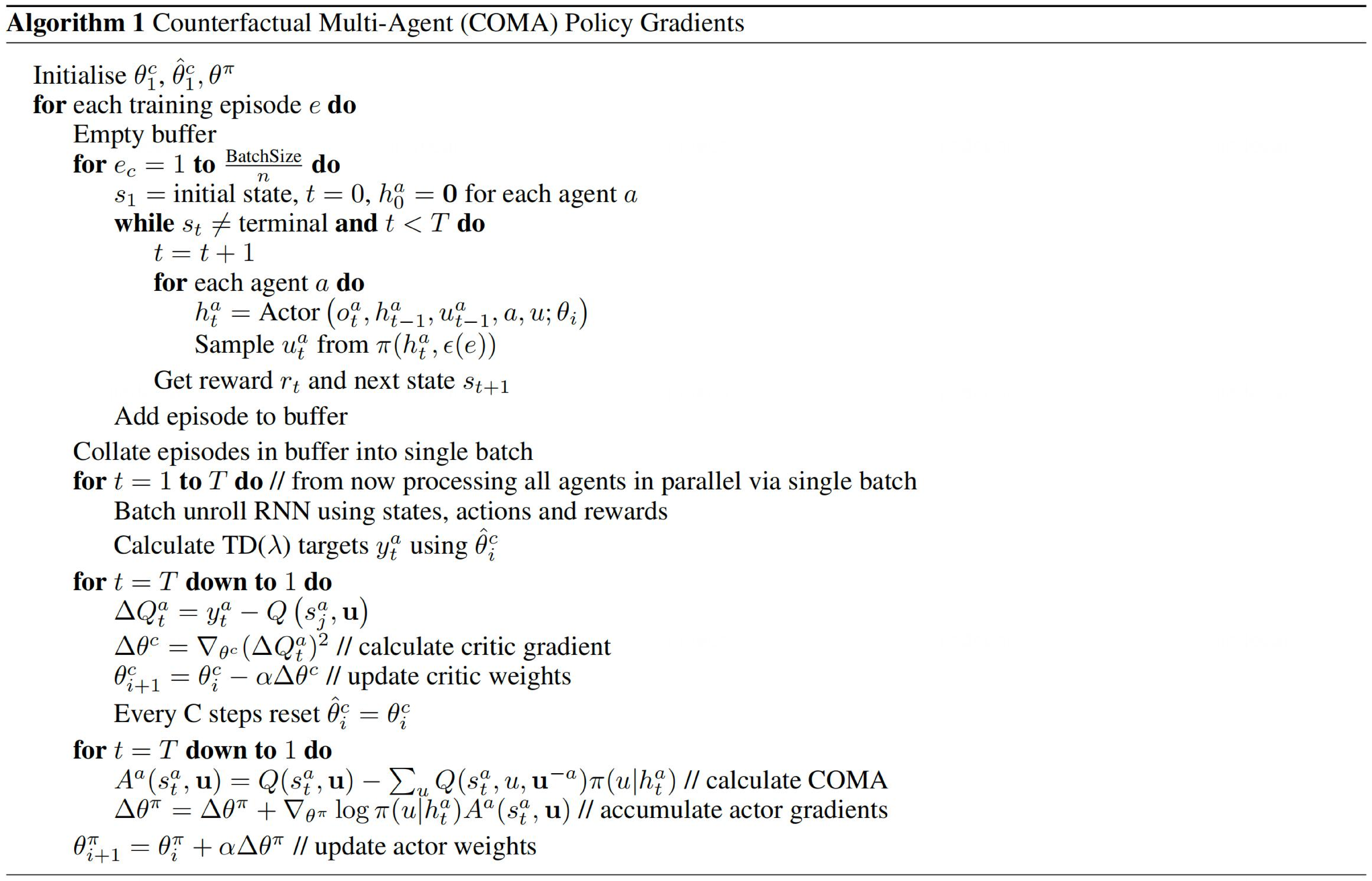


Figure 1: In (a), information flow between the decentralised actors, the environment and the centralised critic in COMA; red arrows and components are only required during centralised learning. In (b) and (c), architectures of the actor and critic.

COMA算法



Algorithm 1 Counterfactual Multi-Agent (COMA) Policy Gradients

```
Initialise $\theta^c_1, \hat{\theta}^c_1, \theta^\pi$
for each training episode $e$ do
    Empty buffer
    for $e_c = 1$ to $\frac{\text{BatchSize}}{n}$ do
        $s_1 =$ initial state, $t = 0$, $h^a_0 = \mathbf{0}$ for each agent $a$
        while $s_t \neq$ terminal and $t < T$ do
            $t = t + 1$
            for each agent $a$ do
                $h^a_t = \text{Actor}\left(o^a_t, h^a_{t-1}, u^a_{t-1}, a, u; \theta_i\right)$
                Sample $u^a_t$ from $\pi(h^a_t, \epsilon(e))$
            Get reward $r_t$ and next state $s_{t+1}$
        Add episode to buffer
    Collate episodes in buffer into single batch
    for $t = 1$ to $T$ do // from now processing all agents in parallel via single batch
        Batch unroll RNN using states, actions and rewards
        Calculate TD($\lambda$) targets $y^a_t$ using $\hat{\theta}^c_i$
    for $t = T$ down to $1$ do
        $\Delta Q^a_t = y^a_t - Q\left(s^a_j, \mathbf{u}\right)$
        $\Delta\theta^c = \nabla_{\theta^c}(\Delta Q^a_t)^2$ // calculate critic gradient
        $\theta^c_{i+1} = \theta^c_i - \alpha\Delta\theta^c$ // update critic weights
        Every C steps reset $\hat{\theta}^c_i = \theta^c_i$
    for $t = T$ down to $1$ do
        $A^a(s^a_t, \mathbf{u}) = Q(s^a_t, \mathbf{u}) - \sum_u Q(s^a_t, u, \mathbf{u}^{-a})\pi(u|h^a_t)$ // calculate COMA
        $\Delta\theta^\pi = \Delta\theta^\pi + \nabla_{\theta^\pi} \log \pi(u|h^a_t) A^a(s^a_t, \mathbf{u})$ // accumulate actor gradients
    $\theta^\pi_{i+1} = \theta^\pi_i + \alpha\Delta\theta^\pi$ // update actor weights
```

VDN -- value decomposition networks

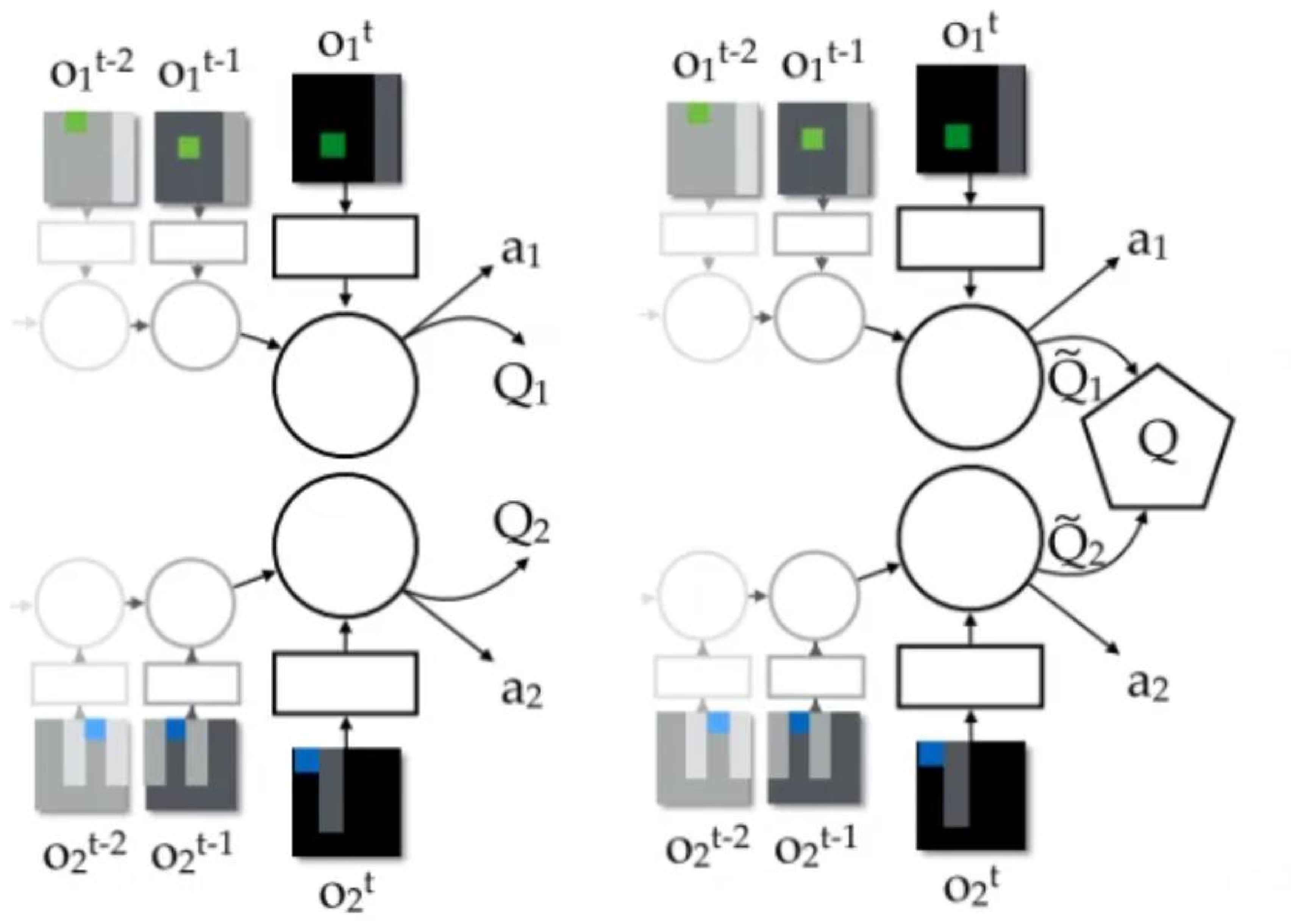


图 9：左图是完全分布式的局部 Q 值网络结构，右图是 VDN 的联合动作 Q 值网络结构。考虑两个智能体，它们的联合动作 Q 值由个体的 Q1 和 Q2 求和得到，在学习时针对这个联合 Q 值进行迭代更新，而在执行时个体根据各自的 Qi 值得到自身的动作 ai。图源：[11]

$$Q((h^1, h^2, ..., h^d), (a^1, a^2, ..., a^d)) \approx \sum_{i=1}^{d} \tilde{Q}_i(h^i, a^i)$$

这个算法考虑了历史的动作，依据共通的状态，或者自身的动作序列，隐含的对其他智能体的Q值进行了建模和预估，也是说得通的

MA-RDPG介绍

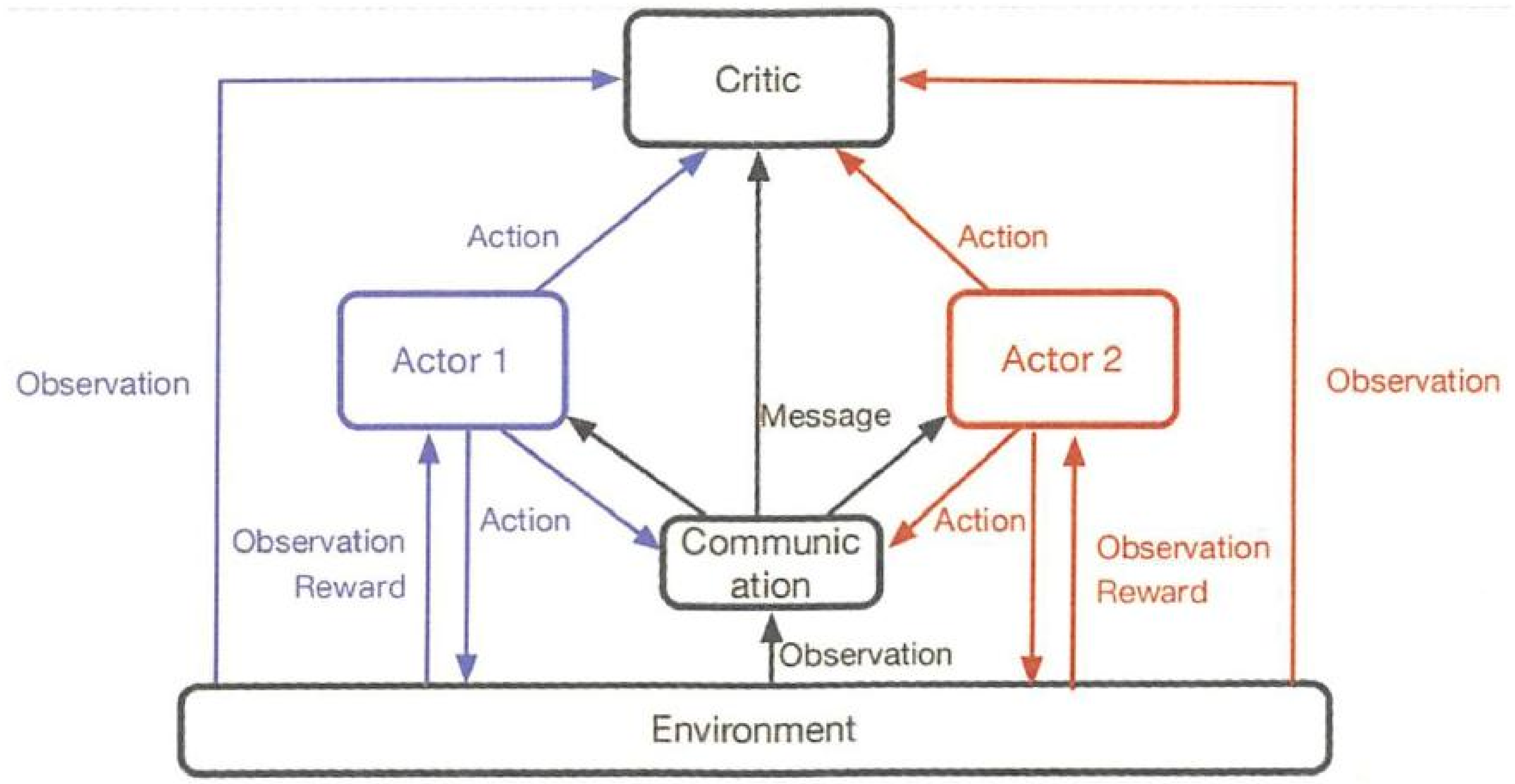


图 4.2　模型的整体结构

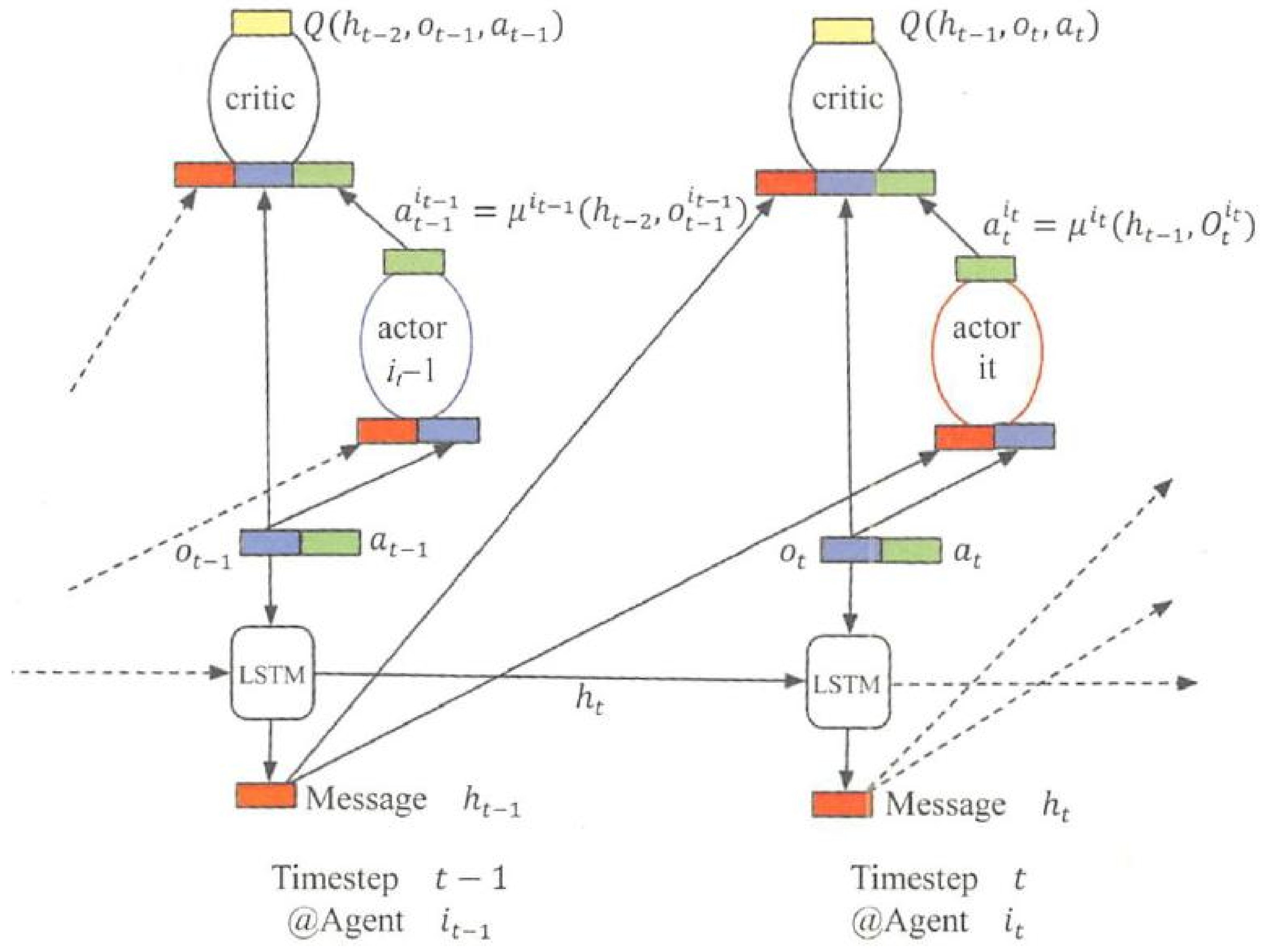


图 4.3　MA-RDPG 算法的详细结构

Q-MIX

$$\underset{\mathbf{u}}{\operatorname{argmax}}\, Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \begin{pmatrix} \operatorname{argmax}_{u^1} Q_1(\tau^1, u^1) \\ \vdots \\ \operatorname{argmax}_{u^n} Q_n(\tau^n, u^n) \end{pmatrix}. \quad (4)$$

$$\frac{\partial Q_{tot}}{\partial Q_a} \geq 0, \ \forall a \in A.$$

QMIX is trained end-to-end to minimise the following loss:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \left[\left(y_i^{tot} - Q_{tot}(\boldsymbol{\tau}, \mathbf{u}, s; \theta) \right)^2 \right], \quad (6)$$

$$y^{tot} = r + \gamma \max_{a'} \overline{Q}(\tau', a', s'; \bar{\theta})$$

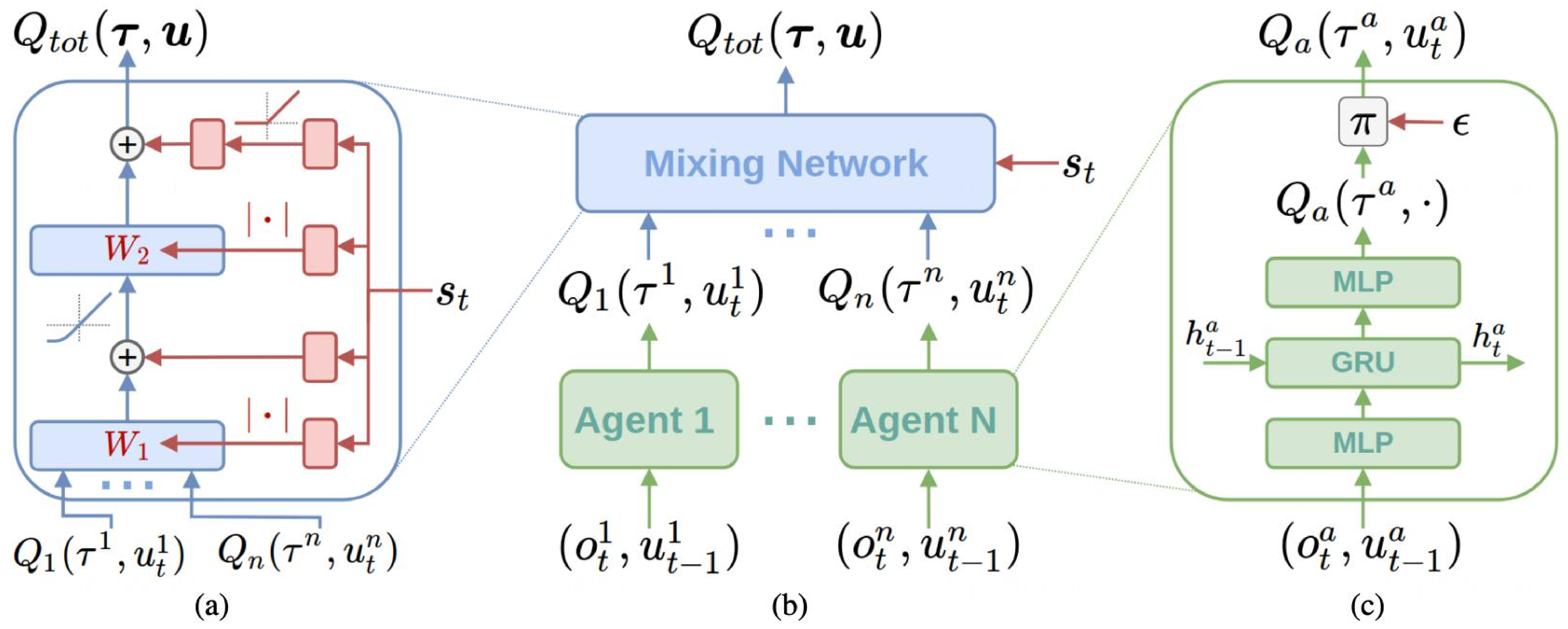


Figure 2. (a) Mixing network structure. In red are the hypernetworks that produce the weights and biases for mixing network layers shown in blue. (b) The overall QMIX architecture. (c) Agent network structure. Best viewed in colour.

智能体强化学习应用

- ◆AlphaGo google deepmind 围棋
- ◆OpenAI Five openai Dota 2
- ◆AlphaStar OpenAI 星际争霸 2
- ◆淘宝主搜索与店铺内搜索
- ◆滴滴派单