# 多智能体学习

## 第二讲：多智能体强化学习

教师：张启超

中国科学院大学
中国科学院自动化研究所

Spring, 2023

# 回顾

矩阵博弈的三大元素：

博弈矩阵

- 玩家(智能体)集合 $N = \{1,2,\dots,n\}$
- 动作集合 $A = \{A_1, A_2, \dots, A_n\}$
- 收益（效用）函数 Payoff (utility)

$$u = (u_1, u_2, \dots, u_n)$$
$$u_1 : A_1 \times A_2 \rightarrow \mathbb{R}$$
$$u_2 : A_1 \times A_2 \rightarrow \mathbb{R}$$

| | | 囚徒2 | |
|---|---|---|---|
| | | 坦白 | 抵赖 |
| 囚徒1 | 坦白 | -4，-4 | 0，-10 |
| | 抵赖 | -10，0 | -1，-1 |

# 回顾

- 最优反应(Best-response)
  - Given $a_{-i} \in A_1 \times \cdots \times A_{i-1} \times A_{i+1} \times \cdots \times A_n$
  - $\boxed{a_i \text{ is best response to } a_{-i}} \Leftrightarrow u_i(a_i, a_{-i}) \geq u_i(a_i', a_{-i}), \forall a_i' \in A_i$

## Definition

- A joint strategy (or strategy profile) $a \in A$ is a Nash Equilibrium $\Leftrightarrow$ $a_i$ is best response to $a_{-i}$ holds for every player $i$

给定一个策略组合$a = (a_1, a_2, \dots, a_n) \in A_1 \times A_2 \times \cdots \times A_n$
若$V_i(a_1, a_2, \dots a_i, \dots a_n) \geq V_i(a_1, a_2, \dots a_i', \dots a_n), \forall a_i' \in A_i, \forall i \in N$
那么策略组合$a$是一个纳什均衡。

# 回顾

对于零和矩阵博弈 $R_1 = -R_2$

利用线性规划来求解纳什均衡

$$\max_x \min_y \quad x^T R_1 y$$

$$\text{s.t.} \quad 1^T x = 1, x_i \geq 0$$
$$1^T y = 1, y_i \geq 0$$

在对手对抗的最坏情况下最大化自身的期望回报

行玩家的LP问题

列玩家的LP问题

$$\max_x \ V_1$$

$$\text{s.t.} \quad x^T R_1 \geq V_1 1^T, \quad \text{(LP1)}$$
$$1^T x = 1, x_i \geq 0$$

$$\max_y \ V_2$$

$$\text{s.t.} \quad R_2 y \geq V_2 1, \quad \text{(LP2)}$$
$$y^T 1 = 1, y_i \geq 0$$

# 回顾

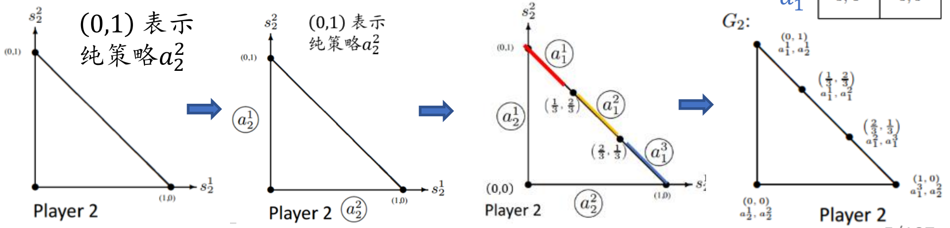
对于一般和矩阵博弈 $R_1$，$R_2$ *Lemke–Howson algorithm*

定理：若一对策略$(s_1, s_2)$是完全标记的，即满足$L(s_1) \cup L(s_2) = A_1 \cup A_2$
那么它就是双人一般和博弈的一组混合策略纳什均衡。

第1步：针对每个玩家找到被标记的策略$s_i$，并在图上进行动作标记$L(s_i)$

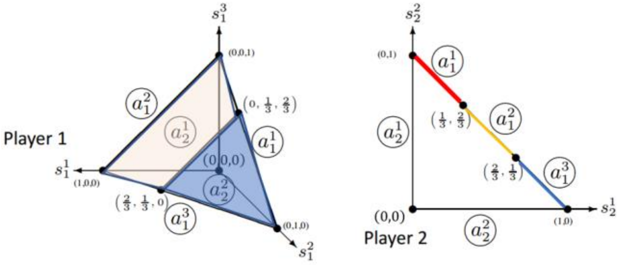
- 玩家$i$被标记的策略$s_i$：玩家的动作$a_i^j$选择概率为0，或者存在其他玩家的纯策略动作$a_{-i}^j$是玩家$i$策略的最优反应
- 玩家$i$被标记的动作：$a_i^j \cup a_{-i}^j$

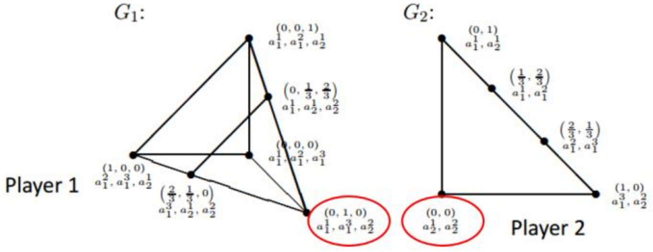
| | | |
|---|---|---|
| $a_1^1$ | 0, 1 | 6, 0 |
| $a_1^2$ | 2, 0 | 5, 2 |
| $a_1^3$ | 3, 4 | 3, 3 |



# 回顾

第2步：找到被完全标记的联合策略$(s_i, s_{-i})$，满足$L(s_i) \cup L(s_{-i})=A_i \cup A_{-i}$



# 回顾

IGA: 无穷小梯度上升

WoLF-IGA: 快速取胜无穷小梯度上升

PHC: 策略爬山

WoLF-PHC: 策略爬山

可用于处理
随机博弈

固定玩家$-i$策略，学习玩家$i$的最优反应策略

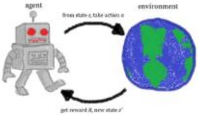
固定玩家$i$最优反应策略，学习玩家$-i$的最优反应策略

逼近纳什均衡策略

# 多智能体强化学习

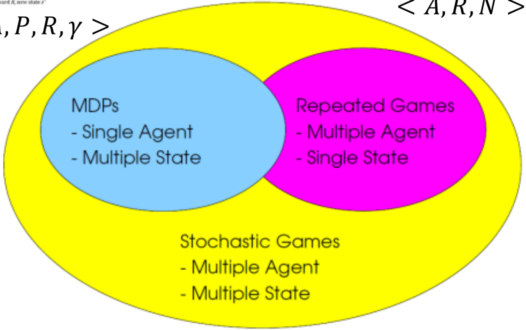
重复博弈
(Repeated games)

- 多个智能体
- 单步状态
- 矩阵博弈




$< S, A, P, R, \gamma >$

$< A, R, N >$




马尔可夫博弈/随机博弈
(Markov/Stochastic games)

- 多个智能体
- 多步状态
- 时序决策

$< S, A, P, R, \gamma, \mathbf{N} >$

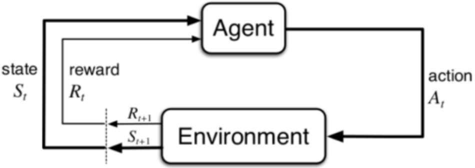
# 课程目录

- 1.1 多智能体强化学习基础
- 1.2 经典多智能体强化学习方法
  - 双人零和随机博弈 Minimax Q learning
  - 一般合随机博弈　Nash Q-learning/FFQ
  - 合作随机博弈　Joint action learner&对手模型
- 1.3 多智能体深度强化学习方法
  - COMA　　- 2018AAAI
  - QMIX　　- 2018ICML
  - MADDPG　- 2017NIPS

# 多智能体强化学习

单智能体强化学习

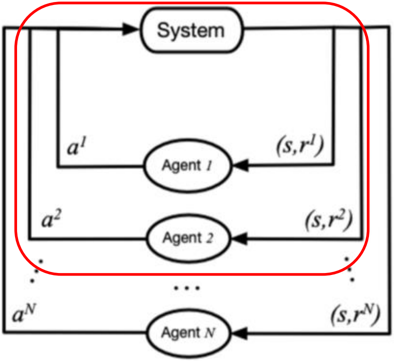
MDP




多智能体强化学习

环境全局状态的改变和所有智能体的联合动作相关

奖赏与全局状态和所有智能体的联合动作相关




多个智能体共享相同的环境

# 多智能体强化学习

MDP

**定义**

一个马尔可夫决策过程 由 $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ 组成

- $\mathcal{S}$ 是有限状态集
- $\mathcal{A}$ 是有限动作集
- $\mathcal{P}$ 是状态转移概率矩阵

$$\mathcal{P}_{ss'}^{a} = \mathbb{P}[s_{t+1} = s' | s_t = s, a_t = a]$$

- $\mathcal{R}$ 是奖励函数, $\mathcal{R}_s^a = \mathbb{E}[r_{t+1} | s_t = s, a_t = a]$
- $\gamma$ 是折扣因子 $\gamma \in [0, 1]$

- 强化学习研究的是序贯决策问题 (sequential decision)，智能体的状态会随时间发生转移

**马尔可夫性**

在给定现在状态及所有过去状态下，智能体未来状态的条件概率分布 仅依赖于当前状态;
换句话说，在给定现在状态时，未来状态与过去状态 (即智能体的历史轨迹) 是 条件独立的

$$\mathbb{P}[s_{t+1} | s_1, \ldots, s_t] = \mathbb{P}[s_{t+1} | s_t]$$

# 多智能体强化学习

**定义：马尔可夫博弈 (Markov Game)**

一个马尔可夫博弈 由 $\langle S;\ A;\ P;\ R;\ \gamma; N\rangle$ 组成

- $S$为有限状态集，s=$\{s_1, \dots, s_n\}$为全局状态，$s_i$第$i$个智能体状态
- $A$为所有智能体的联合动作集,$\{A_i\}_{i\in N}$,$\pi_i(s)$为第$i$个智能体策略
- $P$是状态转移概率矩阵

$$P_{ss'}^{a} = \mathbb{P}[s_{t+1} = s' | s_t = s, a_{(t)} = (a_1, a_2, \dots, a_n)]$$

- $R$为奖励函数，$\{R_i\}_{i\in N}$,$R_i$为第$i$个智能体的奖励函数

$$R_i = \mathbb{E}\left[r_i^{t+1} | s_t = s, a_{(t)} = (a_1, a_2, \dots, a_n)\right]$$

合作博弈：$R_1 = R_2 = \cdots = R_n$

零和博弈：$R_1 \propto -R_2$ . . . .

- $\gamma \in [0,1]$是折扣因子;N $= \{1,2, \dots, n\}$,$n$为智能体的数量
- $G_i$第$i$个智能体的回报 $$G_i^t = \sum_{k=0}^{\infty} \gamma^k r_i^{t+k+1}$$

$n$=1的马尔可夫博弈就是MDP!

# 多智能体强化学习

马尔可夫博弈中的第$i$个智能体的价值函数

$$
\begin{aligned}
&V_i^{\pi}(s) \\
&= \mathrm{E}_{\pi}\{\textstyle\sum_{k=0}^{+\infty} \gamma^k r_i(t+k+1)\,|s_t = s\} \\
&= \mathrm{E}_{\pi}\{r_i(t+1) + \gamma \textstyle\sum_{k=0}^{+\infty} \gamma^k r_i(t+k+2)\,|s_t = s\} \\
&= \sum_{a} \pi(a|s) \sum_{s'} p(s'|s,a)[r_i(s',a) + \gamma \mathrm{E}_{\pi}\{\textstyle\sum_{k=0}^{+\infty} \gamma^k r_i(t+k+2)\,|s_{t+1} = s'\}] \\
&= \sum_{a} \pi(a|s) \sum_{s'} p(s'|s,a)[r_i(s',a) + \gamma V_i^{\pi}(s')]
\end{aligned}
$$

- $\pi(a|s)$是在状态s下选择联合动作$a$的概率
- 价值函数依赖所有智能体的联合策略，而非该智能体策略
- 如果任意智能体改变其策略，则每个智能体的价值函数也会改变（耦合）

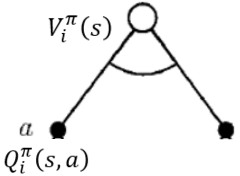
# 多智能体强化学习

马尔可夫博弈中的第$i$个智能体的动作-价值函数

$$
\begin{aligned}
& Q_i^{\pi}(s,a) \\
&= \mathrm{E}_{\pi}\{\textstyle\sum_{k=0}^{+\infty} \gamma^k r_i(t+k+1) \,|s_t = s, a_t = a\} \\
&= \mathrm{E}_{\pi}\{r_i(t+1) + \gamma \textstyle\sum_{k=0}^{+\infty} \gamma^k r_i(t+k+2) \,|s_t = s, a_t = a\} \\
&= \sum_{s\prime} p(s'|s,a)[r_i(s',a) + \gamma \mathrm{E}_{\pi}\{\textstyle\sum_{k=0}^{+\infty} \gamma^k r_i(t+k+2) \,|s_{t+1} = s', a_t = a\}] \\
&= \sum_{s\prime} p(s'|s,a)[r_i(s',a) + \gamma V_i^{\pi}(s')]
\end{aligned}
$$

与单智能体强化学习类似

$$
V_i^{\pi}(s) = \sum_{a} \pi(a|s)\, Q_i^{\pi}(s,a)
$$



# 多智能体强化学习

## 纳什均衡

- 随机博弈的纳什均衡是指一个策略组合$(\pi_1, \pi_2, \ldots, \pi_i, \ldots, \pi_n)$，对于组合中每个玩家的策略$\pi_i$，满足：

  -任意策略$\pi_i$都是针对其他策略$\pi_{-i}$的最优反应

- 没有任何一个玩家在改变其策略后表现得更好

$$\forall_{i=1\ldots n}, \pi_i^* \in BR_i(\pi_{-i}^*)$$

玩家$i$相对于其他玩家联合策略$\pi_{-i}$的最优反应为：

$\pi_i^* \in BR_i(\pi_{-i})$ 当且仅当：

$$\forall s \in S, V_i^{\langle \pi_i^*, \pi_{-i} \rangle} \geq V_i^{\langle \pi_i, \pi_{-i} \rangle}$$

# 多智能体强化学习

## 纳什均衡学习

学习均衡策略 $\pi^* = (\pi_i^*, \pi_{-i}^*)$，为下列方程的不动点

$$Q_i^*(s,a) = r_i(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i^*(s') \quad \forall_{i=1\ldots n}$$

- 智能体$i$学习的目标：学习一个策略$\pi_i$：$S \times A_i \rightarrow [0,1]$，即全局状态s到智能体$i$动作的分布

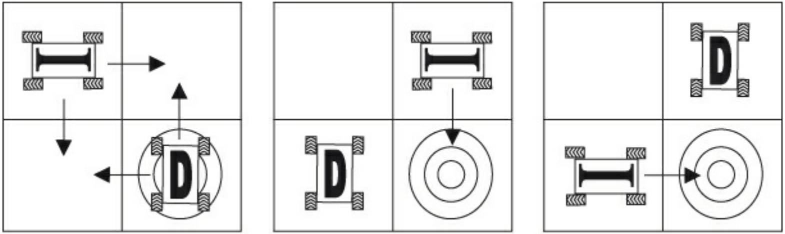
- 该策略可以最大化智能体的价值函数或动作价值函数

# 多智能体强化学习

例子：疆土防御博弈

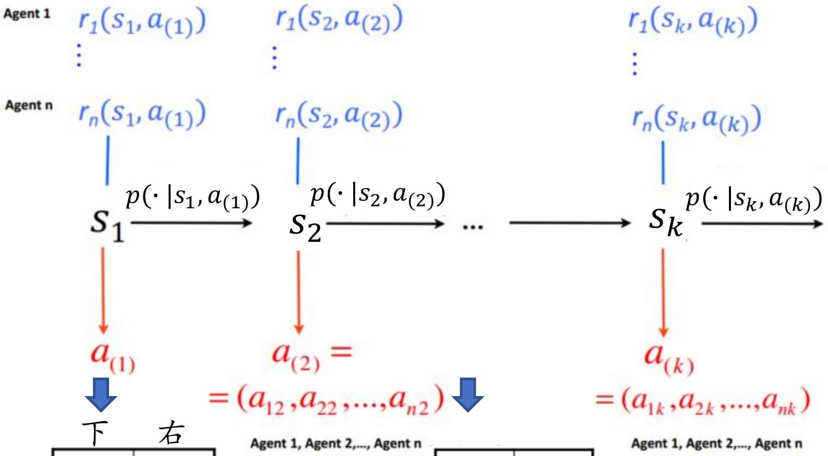
入侵者的动作{向下，向右}；防御者的动作{向上，向左}

入侵者到达领地(右下)，游戏结束；
入侵者与防御者达到非领地外的同一网格，则入侵者被抓，游戏结束。




(a) (b) (c)

# 多智能体强化学习

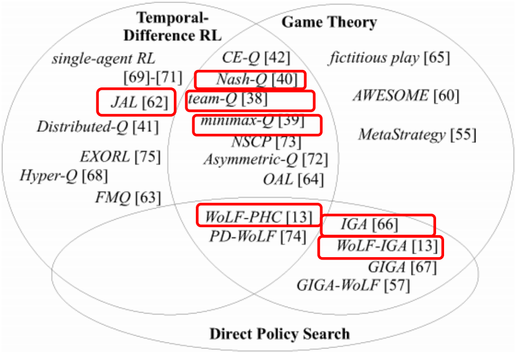



| | 下 | 右 |
|---|---|---|
| 上 | 1, 0 | 0, 1 |
| 左 | 0, 1 | 1, 0 |

| | | |
|---|---|---|
| | 1, 1 | 1, 0 |
| | 0, 1 | 0, 0 |

# 多智能体强化学习

## 方法角度




Busoniu L, Babuska R, De Schutter B. A comprehensive survey of multiagent reinforcement learning[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 2008, 38(2): 156-172.

# 课程目录

■ 1.1 多智能体强化学习基础

■ 1.2 经典多智能体强化学习方法

◆ 双人零和随机博弈 Minimax-Q learning

◆ 混合随机博弈 Nash-Q learning/FFQ

◆ 合作随机博弈 Joint action learner&对手模型

■ 1.3 多智能体深度强化学习方法

◆ COMA - 2018AAAI
◆ QMIX - 2018ICML
◆ MADDPG - 2017NIPS

# Minimax-Q learning

Littman M L. Markov games as a framework for multi-agent reinforcement learning[M]//Machine learning proceedings 1994. Morgan Kaufmann, 1994: 157-163.

## 2.1 Minimax-Q learning

### 双人零和随机博弈

针对双人零和随机博弈，Minimax-Q学习是最早提出的MARL方法。

纳什均衡$(\pi_1^*, \pi_2^*)$，满足最小最大条件 $V_1(\pi_1^*, \pi_2) \geq V_1(\pi_1^*, \pi_2^*) \geq V_1(\pi_1, \pi_2^*)$

Minimax Q函数 $Q(s, a_1, a_2) = r(s, a_1, a_2) + \gamma \sum_{s'} p(s'|s, a_1, a_2)\, V(s')$

在状态s下，玩家1的价值函数为

$$\boxed{V(s) = \max_{\pi_1(s,\cdot)} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} Q(s, a_1, a_2)\, \pi_1(s, a_1)}$$

$\pi_1(s, a_1)$为玩家1在状态s下采取动作$a_1 \in A_1$的概率，为随机策略

## 2.1 Minimax-Q learning

根据贝尔曼最优原理，纳什均衡点为求解**贝尔曼最小最大方程**

$$V_*(s) = \max_{\pi_1(s,\cdot)} \min_{a_2\in A_2} \sum_{a_1\in A_1} Q_*(s,a_1,a_2)\,\pi_1(s,a_1)$$

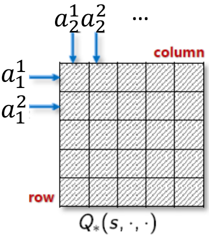
在概率策略空间上寻找最大

在动作（纯策略）空间上寻找最小解

若已知$Q_*$，可转化为线性规划问题求解$\pi_*$

$$\pi_{1*}(s) = \arg\max_{\pi_1(s,\cdot)} \min_{a_2\in A_2} \sum_{a_1\in A_1} Q_*(s,a_1,a_2)\,\pi_1(s,a_1)$$

矩阵博弈




$$\max_{\pi} \mathrm{c}$$

$$\text{s.t. } \textstyle\sum_{a_1\in A_1} \pi_{1*}(s,a_1) Q_*(s,a_1,a_2) \ge c,\ \forall a_2 \in A_2$$

$$\sum_{a_1\in A_1} \pi_{1*}(s,a_1) = 1,\ \pi_{1*}(s,\cdot) \ge 0$$

## 2.1 Minimax-Q learning

回顾： 利用线性规划(LP)来求解纳什均衡

| 玩家1 \ 玩家2 | 第1列 | 第2列 |
|---|---|---|
| 第1行 | -2， 2 | 3， -3 |
| 第2行 | 3， -3 | -4， 4 |

如果玩家1宣称自己的策略为 $<x_1, x_2>$, 则玩家2的期望回报为：

$$V_2(a_1) = 2x_1 - 3x_2$$

$$V_2(a_2) = -3x_1 + 4x_2$$

则玩家2对于玩家1策略 $<x_1, x_2>$ 的<u>最优反应</u>为 $\max(2x_1 - 3x_2, -3x_1 + 4x_2)$

由于是零和博弈， $\max(2x_1 - 3x_2, -3x_1 + 4x_2) = \min(-2x_1 + 3x_2, 3x_1 - 4x_2)$

对于玩家1策略的目标为:

$(x_1, x_2) = \text{argmax}_{(x_1, x_2)} \min(-2x_1 + 3x_2, 3x_1 - 4x_2)$

# 2.1.1 计算纳什均衡

回顾：利用线性规划(LP)来求解纳什均衡

| 玩家1 \ 玩家2 | 第1列 | 第2列 |
|---|---|---|
| 第1行 | -2，2 | 3，-3 |
| 第2行 | 3，-3 | -4，4 |

玩家1的策略通过求解如下线性规划问题：

$$\max_{x} \ V_1$$

$$\begin{aligned} \text{s.t.} \ -2x_1 + 3x_2 &\geq V_1, \\ 3x_1 - 4x_2 &\geq V_1, \\ x_1 + x_2 = 1, x_i &\geq 0 \end{aligned}$$

$\Rightarrow$

$$\max_{x} \ V_1$$

$$\begin{aligned} \text{s.t.} \quad x^T R_1 &\geq V_1 1^T, \\ 1^T x = 1, x_i &\geq 0 \end{aligned}$$

## 2.1 Minimax-Q learning

- Q learning

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r(s,a) + \gamma \max_{a'} Q(s',a'))$$

- Minimax-Q learning

$\alpha$为学习率

1. 初始化Q和V函数，策略$\pi$
2. 对于每次迭代
   3. 智能体根据当前状态s采用探索-利用策略得到动作$a_1$
   4. 观测对手智能体执行的动作$a_2$，奖励$r$和下一时刻状态s′
   5. 更新$Q(s,a_1,a_2)$:

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

6. 利用线性规划求解$V(s') = \max_{\pi(s',\cdot)} \min_{a_2' \in A_2} \sum_{a_1' \in A_1} Q(s',a_1',a_2')\pi_1(s',a_1')$，更新$\pi_1(s')$和$V(s')$

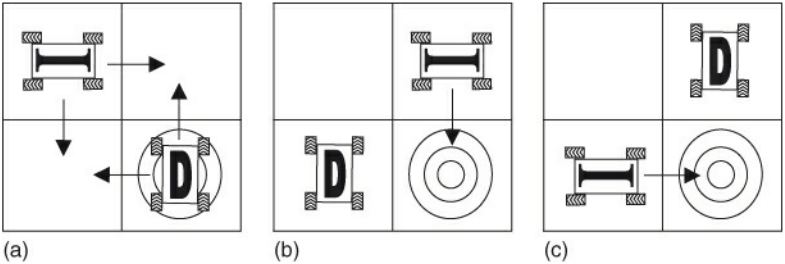
- 将单智能体动作空间扩展至多智能体联合动作空间
- 计算MiniMax均衡策略，而非最优解

## 2.1 Minimax-Q learning

例子：疆土防御博弈

入侵者的动作{向下，向右}；防御者的动作{向上，向左}

入侵者到达领地(右下)，游戏结束；
入侵者与防御者达到非领地外的同一网格，则入侵者被抓，游戏结束。



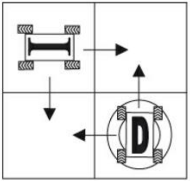

(a)玩家的初始位置：状态$s_1$;(b)入侵者位于右上角防御者在左下角：状态$s_2$；（c）入侵者位于左下角防御者在右上角：状态$s_3$

## 2.1 Minimax-Q learning

防御者的reward函数

$$R_D = \begin{cases} dist_{IT}, & \text{防御者抓获入侵者} \\ -10, & \text{入侵者到达疆土} \end{cases}$$

$$dist_{IT} = |x_I(t_f) - x_T| + |y_I(t_f) - y_T|$$




入侵者的reward函数

$$R_I = \begin{cases} -dist_{IT}, & \text{防御者抓获入侵者} \\ 10, & \text{入侵者到达疆土} \end{cases}$$

$$\boxed{\begin{aligned} \gamma &= 0.9 \\ V_D(s_2) &= -10 \\ V_D(s_3) &= -10 \end{aligned}}$$

$$Q_D^*(s_1, a_{left}, o_{right}) = \gamma V_D(s_2) = -9$$

$$Q_D^*(s_1, a_{up}, o_{down}) = \gamma V_D(s_3) = -9$$

$$Q_D^*(s_1, a_{left}, o_{down}) = 1$$

$$Q_D^*(s_1, a_{up}, o_{right}) = 1$$

状态$s_1$ 下的Q表

| | | Defender | |
|---|---|---|---|
| | $Q_D^*$ | Up | Left |
| Invader | Down | –9 | 1 |
| | Right | 1 | –9 |

## 2.1 Minimax-Q learning

假设防御者在状态$s_1$选取动作{向上，向左}的策略为<$p_1, p_2$>，收益值$V_D$

$$R_D(s_1) = \begin{bmatrix} -9 & 1 \\ 1 & -9 \end{bmatrix}$$

$$\max_{p_1,\, p_2} V_D$$

$$p_1 r_{11} + p_2 r_{21} \geq V_D$$
$$p_1 r_{12} + p_2 r_{22} \geq V_D$$
$$p_1 + p_2 = 1$$

利用线性规划求解可得 $p_1(s_1, a_{up}) = p_2(s_1, a_{left}) = 0.5$ $V_D(s_1) = -4$

类似地，对于入侵者而言，可以求出在状态$s_1$选取动作{向下，向右}的策略也为<0.5,0.5> $V_I(s_1) = 4$

# 2.1 Minimax-Q learning

1. 初始化$Q$和V函数，策略$\pi_D$

 对于$all\ s \in S, a_i \in A_i, a_{-i} \in A_{-i}$， $Q(s, a_i, a_{-i}) = 0, V(s) = 0$

 $\pi_D(a_{up}) = 1,\ \alpha = 0.1, \epsilon = 0.1, \gamma = 0.9, \pi_I(a_{right}) = 1;$

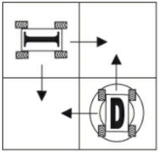
迭代训练

2. 智能体在$s_1$根据探索-利用策略采取动作，观察reward及下一时刻状态

 防御者$a_{up}$，入侵者$a_{right}$，$r_D = 1, r_I = -1$，下一时刻状态$s' = s_1$

3. 更新Q $Q(s_1, a_1, a_2) \leftarrow (1-\alpha)Q(s_1, a_1, a_2) + \alpha\big(r(s_1, a_1, a_2) + \gamma V(s')\big)$

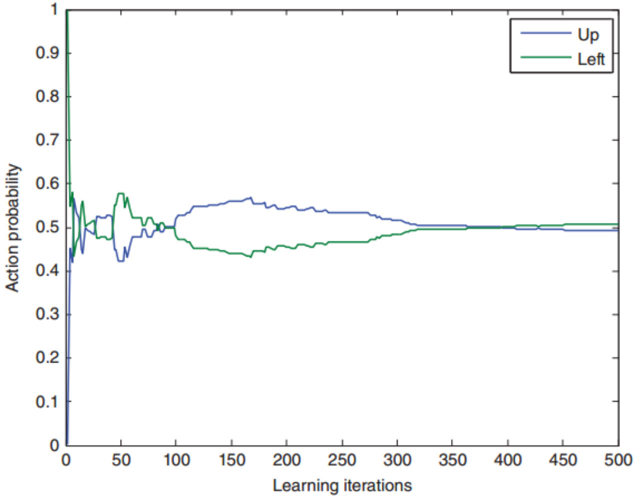
| D \ I | 向下 | 向右 |
|---|---|---|
| 向上 | 0，0 | 0.1,-0.1 |
| 向左 | 0，0 | 0，0 |




4. 利用线性规划求$V(s') = \max\limits_{\pi(s',\cdot)} \min\limits_{a_2' \in A_2} \sum_{a_1' \in A_1} Q(s', a_1', a_2')\pi_D(s', a_1')$

 得到$\pi_D(s')$，更新$V(s')$

## 2.1 Minimax-Q learning




防御者/入侵者博弈游戏的Minimax-Q学习结果：防御者策略

## 2.1 Minimax-Q learning

不足：

1. 需要不断求解一个线性规划，造成学习速度降低，增加计算时间。

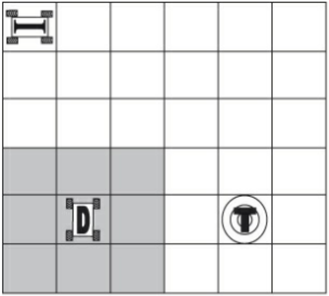
2.智能体需要事先知道所有智能体的动作空间。

3.满足收敛性，不满足合理性。

4.Minimax-Q算法是一个与对手策略无关的算法（opponent-independent algorithm），不论对手采取什么策略，都将收敛到该博弈的纳什均衡策略。

## ◆练习：利用Minimax Q learning学习疆土防御问题的均衡策略

6*6网格的疆土防御问题




两个玩家(防御者,入侵者)初始位置可随机选择，同时采取行动。

动作空间：向上，向下，向左，向右。如果选择动作使得其超出网格，则待在当前位置。

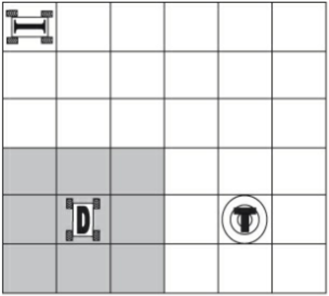
入侵者从左上角开始，试图在被捕获之前到达领土

防御者从底部开始，试图拦截入侵者

领土位置不变，保持在（5，5）的位置

# ◆练习：利用Minimax Q learning学习疆土防御问题的均衡策略

6*6网格的疆土防御问题




防御者周围的9个灰色单元格，是入侵者被捕获的区域。

当防御者捕获入侵者或入侵者成功到达领土时，一次游戏结束。然后玩家随机选择初始位置后，游戏重新开始。

# ◆练习：利用Minimax Q learning学习疆土防御问题的均衡策略

6*6网格的疆土防御问题

入侵者的目标：不被拦截的情况下到达领土

防御者的目标：尽可能在远离领土的地方拦截入侵者

奖赏的设定：入侵者与领土之间的距离

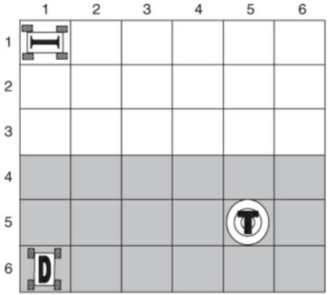
$$dist_{IT} = |x_I(t_f) - x_T| + |y_I(t_f) - y_T|$$

对两个玩家采用Minimax Q learning算法进行均衡策略学习

# ◆练习：利用Minimax Q learning学习疆土防御问题的均衡策略

训练：两个玩家初始位置随机选择

测试：两个玩家在如下图所示的位置进行1000次测试




计算1000次运行中每次终止时刻入侵者与领土之间的最终距离的平均值（平均距离）

# Nash-Q learning &

# Friend-or-Foe Q-Learning

Hu J, Wellman M P. Nash Q-learning for general-sum stochastic games[J]. Journal of machine learning research, 2003, 4(Nov): 1039-1069.

Littman M L. Friend-or-foe Q-learning in general-sum games[C]//ICML. 2001, 1: 322-328.

## 2.2 Nash-Q learning

Nash Q-Learning算法是将Minimax-Q算法从零和随机博弈扩展到多人一般和随机博弈的算法

Minimax Q函数

$$Q(s,a_1,a_2) = r(s,a_1,a_2) + \gamma \sum_{s'} p(s'|s,a_1,a_2)\ V(s')$$

$$V(s) = \max_{\pi_1(s,\cdot)} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} Q(s,a_1,a_2)\,\pi_1(s,a_1)$$

Nash-Q函数

$$Q_i^*(s,a_1,\ldots,a_n) = r_i(\mathrm{s},a_1,\ldots,a_n) + \gamma \sum_{s'} p(s'|s,a_1,\ldots,a_n) V_i^*(s')$$

智能体$i$的均衡价值$V_i^*(s) = V_i^{\pi^*}(s)$，表示在一般和博弈纳什均衡策略$\pi^*=(\pi_1^*,\ldots,\pi_n^*)$下的价值（期望收益）

## 2.2 Nash-Q learning

- Nash均衡

针对随机博弈问题，满足下列不等式的$(\pi_1^*,\dots,\pi_n^*)$是一组纳什均衡策略

$$V_i(s,\pi_1^*,\dots,\pi_n^*) \geq V_i(s,\pi_1^*,\dots,\pi_i,\dots,\pi_n^*), \ \forall \pi_i$$

**定义：Nash-Q函数[1]**

智能体$i$的Nash-Q函数定义为：对于$(s,a_1,\dots,a_n)$，当所有智能体执行纳什均衡策略，智能体$i$当前奖赏与智能体$i$下一时刻状态的均衡价值之和，即

$$\boxed{Q_i^*(s,a_1,\dots,a_n) = r_i(s,a_1,\dots,a_n)+\gamma\sum_{s'}p(s'|s,a_1,\dots,a_n)V_i(s',\pi_1^*,\dots,\pi_n^*)}$$

其中$(\pi_*^1,\dots,\pi_*^n)$为混合博弈纳什均衡策略。

在状态$s$执行联合动作$(a_1,\dots,a_n)$后基于纳什均衡策略所得到的$Q_i^*$

[1] Hu J, Wellman M P. Nash Q-learning for general-sum stochastic games[J]. Journal of machine learning research, 2003, 4(Nov): 1039-1069.

## 2.2 Nash-Q learning

**Algorithm 4.2** Nash Q-learning algorithm

1: Initialize $Q_i(s, a_1, \ldots, a_n) = 0, \forall a_i \in A_i, \boxed{i = 1, \ldots, n}$
2: **for** Each iteration **do**
3: Player $i$ takes an action $a_i$ from current state $s$ based on an exploration-exploitation strategy
4: At the subsequent state $s'$, player $i$ observes the rewards received from all the players $r_1, \ldots, r_n$, and all the players' actions taken at the previous state $s$.
5: Update $Q_i(s, a_1, \ldots, a_n)$:

$$\boxed{Q_i(s, a_1, \ldots, a_n) \leftarrow (1 - \alpha)Q_i(s, a_1, \ldots, a_n) + \alpha\left[r_i + \gamma \text{Nash}Q_i(s')\right]} \quad (4.15)$$

where $\alpha$ is the learning rate and $\gamma$ is the discount factor
6: Update Nash$Q_i(s)$ and $\pi_i(s)$ using quadratic programming
7: **end for**

$$NashQ_t^i(s') = \boxed{\pi^1(s') \cdots \pi^n(s')} \cdot Q_t^i(s')$$

## 2.2 Nash-Q learning

Our learning agent, indexed by $i$, learns about its Q-values by forming an arbitrary guess at time 0. One simple guess would be letting $Q_0^i(s,a^1,\ldots,a^n)=0$ for all $s \in S, a^1 \in A^1,\ldots,a^n \in A^n$. At each time $t$, agent $i$ observes the current state, and takes its action. After that, it observes its own reward, actions taken by all other agents, others' rewards, and the new state $s'$. It then calculates a Nash equilibrium $\pi^1(s')\cdots\pi^n(s')$ for the stage game $(Q_t^1(s'),\ldots,Q_t^n(s'))$, and updates its Q-values according to

$$Q_{t+1}^i(s,a^1,\ldots,a^n)=(1-\alpha_t)Q_t^i(s,a^1,\ldots,a^n)+\alpha_t\left[r_t^i+\beta NashQ_t^i(s')\right], \tag{6}$$

where

$$NashQ_t^i(s')=\pi^1(s')\cdots\pi^n(s')\cdot Q_t^i(s'), \tag{7}$$

Different methods for selecting among multiple Nash equilibria will in general yield different updates. $NashQ_t^i(s')$ is agent $i$'s payoff in state $s'$ for the selected equilibrium. Note that $\pi^1(s')\cdots\pi^n(s')\cdot Q_t^i(s')$ is a scalar. This learning algorithm is summarized in Table 2.

In order to calculate the Nash equilibrium $(\pi^1(s'),\ldots,\pi^n(s'))$, agent $i$ would need to know $Q_t^1(s'),\ldots,Q_t^n(s')$. Information about other agents' Q-values is not given, so agent $i$ must learn about them too. Agent $i$ forms conjectures about those Q-functions at the beginning of play, for example, $Q_0^j(s,a^1,\ldots,a^n)=0$ for all $j$ and all $s,a^1,\ldots,a^n$. As the game proceeds, agent $i$ observes other agents' immediate rewards and previous actions. That information can then be used to update agent $i$'s conjectures on other agents' Q-functions. Agent $i$ updates its beliefs about agent $j$'s Q-function, according to the same updating rule (6) it applies to its own,

$$Q_{t+1}^j(s,a^1,\ldots,a^n)=(1-\alpha_t)Q_t^j(s,a^1,\ldots,a^n)+\alpha_t\left[r_t^j+\beta NashQ_t^j(s')\right]. \tag{8}$$

Note that $\alpha_t=0$ for $(s,a^1,\ldots,a^n)\neq(s_t,a_t^1,\ldots,a_t^n)$. Therefore (8) does not update all the entries in the Q-functions. It updates only the entry corresponding to the current state and the actions chosen by the agents. Such updating is called *asynchronous updating*.

## 2.2 Nash-Q learning

Nash Q learning举例

确定性网格博弈

| | | |
|---|---|---|
| ❷ 6 | 7 | ❶ 8 |
| 3 | 4 | 5 |
| ⊡ 0 | 1 | ⧈ 2 |

⊡ 智能体1

❶ 1目的地

⧈ 智能体2

❷ 2目的地

动作空间

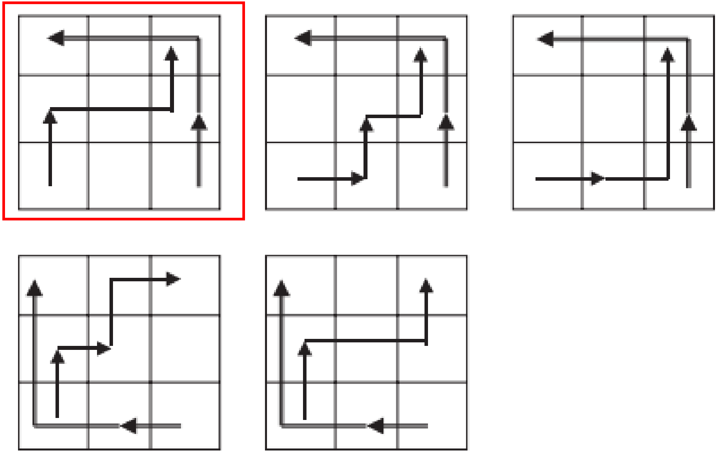
$A^i = \{$左，右，上，下$\}$

状态空间 s=$(l^1, l^2)$
左下角单元格状态为0，
右上角为8

当两个智能体移动到同一单元格，则返回，得到惩罚-1；
当一个智能体达到目标状态时，游戏结束，获得奖励100；
若两个智能体同时到达目标状态，均获得奖励100；移动到空网格奖励为0
智能体最初并不知道各自奖赏和目的地。

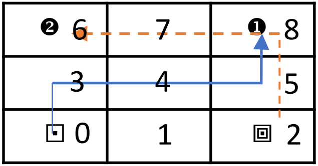
## 2.2 Nash-Q learning

Nash Q learning举例



## 2.2 Nash-Q learning

确定性网格博弈




一组纳什均衡策略

| 状态 | $\pi_1^*(s)$ | $\pi_2^*(s)$ |
|---|---|---|
| (0,2) | 上 | 上 |
| (3,5) | 右 | 上 |
| (4,8) | 右 | 左 |
| (5,7) | 上 | 左 |

初始状态$s_0$ =(0,2)下，$\gamma = 0.99$，纳什策略下的值函数为

$$V_1^*(s_0, \pi_1^*, \pi_2^*) = 0 + 0.99*0 + 0.99^2 * 0 + 0.99^3 * 100 = 97$$

进一步可推导智能体1在状态$s_0$的纳什$Q_1$值为

| 1 \ 2 | 左 | 上 |
|---|---|---|
| 右 | 95.1, 95.1 | 97，97 |
| 上 | 97，97 | 97，97 |

$$Q_1^*(s_0, 右, 左) = -1 + 0.99 * V_1^*(s_0)$$

$$Q_1^*(s_0, 上, 上) = 0 + 0.99 * V_1^*(3,5)$$

…

## 2.2 Nash-Q learning

在状态$s_0$下初始化每个智能体$i$的$Q_i$

20次迭代后状态$s_0$的$Q_1$值

| | *Left* | *Up* |
|---|---|---|
| *Right* | -1, -1 | 49, 0 |
| *Up* | 0, 0 | 0, 97 |

5000次迭代后状态$s_0$的$Q_1$值

| | *Left* | *Up* |
|---|---|---|
| *Right* | 86, 87 | 83, 85 |
| *Up* | 96, 91 | 95, 95 |

## 2.2 Nash-Q learning

不足：

需要维护多个Q表，双线性规划更耗时；

每个智能体需要知道其他智能体的策略计算Nash Q；

只满足收敛性，不满足合理性

Lemke C E, Howson, Jr J T. Equilibrium points of bimatrix games[J]. Journal of the Society for industrial and Applied Mathematics, 1964, 12(2): 413-423.

## 2.2 Friend-or-Foe Q-Learning (FFQ)

借鉴Minimax-Q算法来处理一般和博弈，将一个n智能体的一般和博弈就转化为了一个两类智能体的零和博弈。

对于智能体i而言，将其他所有智能体分为两组，一组为i的friend帮助i一起最大化其奖励回报，另一组为i的foe对抗i并降低i的奖励回报，

## 2.2 Friend-or-Foe Q-Learning (FFQ)

**Friend-or-foe Q-learning algorithm**

Initialize $V_i(s) = 0$ and $Q_i(s, a_1, \ldots, a_{n_1}, o_1, \ldots, o_{n_2}) = 0$ where $(a_1, \ldots, a_{n_1})$ denotes player $i$ and its friends' actions and $(o_1, \ldots, o_{n_2})$ denotes its opponents' actions.

**for** Each iteration **do**

Player $i$ takes an action $a_i$ from current state $s$ based on an exploration-exploitation strategy.

At the subsequent state $s'$, player $i$ observes the received reward $r_i$, its friends' and opponents' actions taken at state $s$.

Update $Q_i(s, a_1, \ldots, a_{n_1}, o_1, \ldots, o_{n_2})$:

$$\boxed{\begin{aligned} Q_i(s, a_1, \ldots, a_{n_1}, o_1, \ldots, o_{n_2}) \leftarrow (1-\alpha) Q_i(s, a_1, \ldots, a_{n_1}, o_1, \ldots, o_{n_2}) \\ + \alpha \left[ r_i + \gamma V_i(s') \right] \end{aligned}}$$

where $\alpha$ is the learning rate and $\gamma$ is the discount factor.

Update $V_i(s)$ using linear programming:

$$\boxed{\begin{aligned} V_i(s) = \max_{\pi_1(s,\cdot),\ldots,\pi_{n_1}(s,\cdot)} \min_{o_1,\ldots,o_{n_2} \in O_1 \times \cdots \times O_{n_2}} \sum_{a_1,\ldots,a_{n_1} \in A_1 \times \cdots \times A_{n_1}} \\ Q_i(s, a_1, \ldots, a_{n_1}, o_1, \ldots, o_{n_2}) \pi_1(s, a_1) \cdots \pi_{n_1}(s, a_{n_1}) \quad (4.58) \end{aligned}}$$

需要知道所有朋友的策略

**end for**

## 2.2 Friend-or-Foe Q-Learning (FFQ)

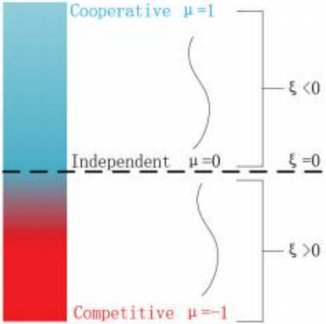
哪些是朋友，哪些是敌人呢？

# 2.2 Friend-or-Foe Q-Learning (FFQ)

**Definition 2.** *(competitive flag)* Let $u_r = Q(s_1, s_2, \pi_1, \pi_2)$ be a payoff function for player 1 in a two player game. Let $u_I = Q(s_1, \pi_1)$ be the payoff function for player 1 taking strategy $\pi_1$ without player 2. Agent 2 is cooperative if $u_r > u_I$ , competitive if $u_r < u_I$, or independent if $u_r = u_I$.

$$\xi = \text{sign}\left[Q_I(s_1, a_1; \theta_I) - Q_r^{\mu}(\vec{s}, \vec{a}; \theta_r)\right]$$

有你获得更高收益，则认为是队友；
有你获得更低收益，则认为是对手；




Zhou Z, Xu H. Deep Reinforcement Learning Based Intelligent Decision Making for Two-player Sequential Game with Uncertain Irrational Player[C]//2019 IEEE Symposium Series on Computational Intelligence (SSCI). IEEE, 2019: 9-15.

# Team Q learning &
# Joint action learner

Littman M L. Value-function reinforcement learning in Markov games[J]. Cognitive systems research, 2001, 2(1): 55-66.

Claus C, Boutilier C. The dynamics of reinforcement learning in cooperative multiagent systems[J]. AAAI/IAAI, 1998, 1998(746-752): 2.

## 2.3 Team Q-Learning

针对多智能体合作问题，期望学习到协作均衡

合作博弈：所有智能体期望最大化一个共同的目标

协作均衡(Coordination equilibria):

$$\sum_{a_1,\ldots,a_n} \pi_1(s,a_1)\cdots\pi_n(s,a_n)Q_i[s,a_1,\ldots,a_n] = \max_{a_1,\ldots,a_n} Q_i[s,a_1,\ldots,a_n]$$

## 2.3 Team Q-Learning

Team Markov games

完全合作博弈

Team reward： $R_1 = R_2 = \cdots = R_n$

Team Q function： $Q_1 = Q_2 = \cdots = Q_n = Q$

$$a_1^*, a_2^*, \ldots, a_n^* = \underset{a_1, a_2, \ldots, a_n}{\operatorname{argmax}} Q(s, a_1, a_2, \ldots, a_n)$$

$Q(s, a) = Q(s, (a_1, a_2, \ldots, a_n))$，将单智能体的动作扩展为团体联合动作

## 2.3 Team Q-Learning

Team Q learning

1. 初始化Team-Q函数
2. 对于每次迭代

$\alpha$为学习率

3.智能体$i$根据当前状态s采用探索-利用策略得到动作$a_i$

4.观测团队其他智能体执行的动作$a_{-i}$，团队奖励$r$和下一时刻状态s′

5.更新Team Q 函数$Q(s, a_1, a_2, \dots, a_n)$:

$$Q(s, a_1, a_2, \dots, a_n) \leftarrow (1-\alpha)Q(s, a_1, a_2, \dots, a_n) + \alpha\big(r(s, a_1, a_2, \dots, a_n) + \gamma V(s')\big)$$

其中 $V(s') = \boxed{\max_{a_1', a_2', \dots, a_n'} Q(s', a_1', a_2', \dots, a_n')}$

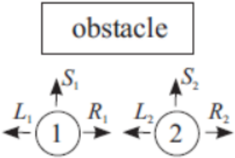
Q学习 $\boxed{V(s') = \max_{a} Q(s', a)}$

## 2.3 Team Q-Learning

定理. 对于完全合作博弈(所有智能体的reward均相同)，若存在唯一协作均衡，则Team Q learning方法中智能体采取a greedy in the limit with infinite exploration(GLIE) policy，可以确保收敛到唯一的协作均衡解。

GLIE：在有限的时间内进行无限可能的探索

但是对于存在多个协作均衡解的情况，Team Q learning难以确保收敛，Team Q learning是一种没有协同机制的方法。




| $Q$ | $L_2$ | $S_2$ | $R_2$ |
|---|---|---|---|
| $L_1$ | $\boxed{10}$ | -5 | 0 |
| $S_1$ | -5 | -10 | -5 |
| $R_1$ | -10 | -5 | $\boxed{10}$ |

## 2.3 Joint action learner (JAL)

Q学习bootstrap时需要用到下个状态的$V(s')$，对于联合动作而言，该值依赖于其他智能体的动作。

Team-Q learning

$$Q(s, a_1, a_2, \dots, a_n) \leftarrow (1-\alpha)Q(s, a_1, a_2, \dots, a_n) + \alpha\big(r(s, a_1, a_2, \dots, a_n) + \gamma V(s')\big)$$

其中 $V(s') = \boxed{\max_{a_1', a_2', \dots, a_n'} Q(s', a_1', a_2', \dots, a_n')}$

## 2.3 Joint action learner (JAL)

对于其他智能体非直接协作的情况，估计下一时刻状态$V(s')$时，其他智能体的动作如何确定更加合理呢？

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。

记录状态$s'$下其他智能体使用不同联合动作$a^{-i}$的次数，然后计算其出现的概率, 用统计方式维护其他智能体的策略模型，通常只能处理小规模的多智能体静态博弈问题。

$$EV(a^i) = \sum_{a^{-i} \in A_{-i}} Q(a^{-i} \cup \{a^i\}) \prod_{j \neq i} \{\mathrm{Pr}^i_{a^{-i}[j]}\}$$

$$V(s') = \max_{a_i'} EV(s', a_i')$$

# 2.3 Joint action learner (JAL)

对手建模：利用统计方法获得对手策略的估计

**Algorithm: Opponent Modeling Q-Learning for player $i$**

(1) Initialize $Q$ arbitrarily, and $\forall s \in \mathcal{S}, a_{-i} \in \mathcal{A}_{-i}$ $C(s, a_{-i}) \leftarrow 0$ and $n(s) \leftarrow 0$.

(2) Repeat,

(a) From state $s$ select action $a_i$ that maximizes,

$$\sum_{a_{-i}} \frac{C(s, a_{-i})}{n(s)} Q(s, \langle a_i, a_{-i} \rangle)$$

(b) Observing other agents' actions $a_{-i}$, reward $r$, and next state $s'$,

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(r + \gamma V(s')\right)$$

$$C(s, a_{-i}) \leftarrow C(s, a_{-i}) + 1$$

$$n(s) \leftarrow n(s) + 1$$

where,

$$a = (a_i, a_{-i})$$

$$V(s) = \max_{a_i} \sum_{a_{-i}} \frac{C(s, a_{-i})}{n(s)} Q(s, \langle a_i, a_{-i} \rangle).$$

对手建模将其他智能体视为一个具有联合行动能力的对手，并对其联合动作进行统计来估计对手策略，学习得到对手策略下的最佳对策

# 总结

- 计算均衡解
  - Minimax Q learning：零和随机博弈

    - 每一步利用线性规划来解零和矩阵博弈，迭代训练
  - Nash Q-learning/FFQ：一般和随机博弈

    - 每一步利用Lemke-Howson来解一般和矩阵博弈，迭代训练
  - Team Q-learning：合作随机博弈
- 计算最佳对策
  - Joint action learner：合作随机博弈

  -利用统计联合动作维护对手模型，学习针对该策略的最佳对策

# 课程目录

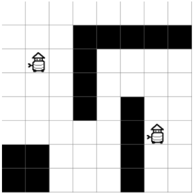
- 1.1 多智能体强化学习基础
- 1.2 经典多智能体强化学习方法
  - 双人零和随机博弈 Minimax Q learning
  - 混合随机博弈 Nash Q-learning/FFQ
  - 合作随机博弈 Team Q-learning/ Joint action learner
- 1.3 多智能体深度强化学习方法
  - COMA - 2018AAAI
  - QMIX - 2018ICML
  - MADDPG - 2017NIPS

# 三种训练架构

定义：分布式部分可观测MDP(Dec-POMDP) $\langle S,\{A_i\},\{R_i\},P,\{O_i\},\gamma;N\rangle$

- 第$i$个智能体的观测模型 $O_i$(s, a)， 局部观测状态为$o_i$~$O_i$(s,$a_i$)
- 个体动作－观测历史，$\tau \in T \equiv (O \times A)$ , $< o, a >$
- 分布式策略$\pi(a|\tau)$： $T \times A \rightarrow [0,1]$
- 部分可观原因：传感器限制；通信限制等

$$\tau_i = (a_{i,0}, o_{i,1}, \cdots, a_{i,t-1}, o_{i,t})$$



| | |
|---|---|
| MDP | 我确定我在什么位置 |
| POMDP | 我不确定我在什么位置 |
| Dec-POMDP | 我不确定我在什么位置同时不确定其他智能体的在什么位置 |

https://ideaheap.com/2014/05/multi-agent-systems-overview/

# 三种训练架构

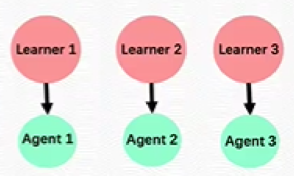
*decentralised policies*: each agent selects its own action conditioned only on its local action-observation history.

◆ 分布式训练&分布式执行

⇩

**独立式学习**

Independent learning

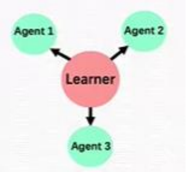



IQL+DQN

◆ 集中式训练&分布式执行

⇩

CTDE: Centralized training decentralized execution



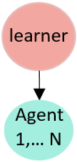

COMA
QMIX
MADDPG

◆ 集中式训练&集中式执行

⇩

**集中式学习**

Centralized learning




Team-Q

IQL+DQN

Tampuu A, Matiisen T, Kodelja D, et al. Multiagent cooperation and competition with deep reinforcement learning[J]. PloS one, 2017, 12(4): e0172395.

# IDL+DQN

每个智能体拥有独立的 Q network，独自采集数据并进行训练

- 完全合作环境：一方失球，则两方均获得 -1 的回报
- 完全竞争环境：一方失球，该方获得 -1 的回报；对方获得 +1 的回报

- □ 结构简单，容易实现，易扩展
- □ RL算法面临动态环境，环境复杂学习困难

# IDL+DQN

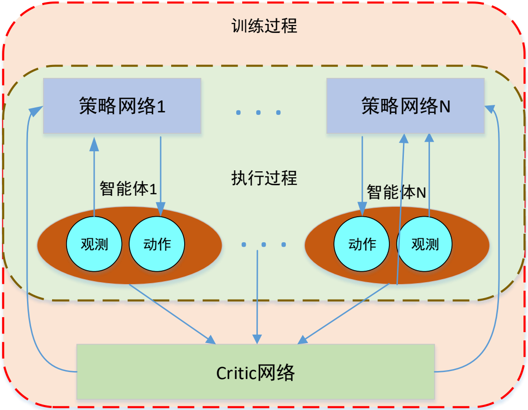

Multiagent Cooperation and Competition
with Deep Reinforcement Learning
Video: competitive mode of playing

# COMA

Foerster J, Farquhar G, Afouras T, et al. Counterfactual multi-agent policy gradients[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2018, 32(1).

## 3.2 COMA

### 集中式训练与分布式执行架构CTDE




集中式训练：
目前RL训练过程通常是在仿真器中进行的，仿真器中可以很容易地得到其他智能体地信息，且通信几乎没有资源消耗。

分布式执行：
实际执行过程会遇到通信问题或局部可观，导致智能体无法获取全局状态
同时联合动作空间过大会导致的维数灾难

# 3.2 COMA

针对合作博弈

1. 环境非静态的问题，如何确保训练稳定性。

2. 对于合作博弈而言，存在多智能体信誉分配问题

In cooperative settings, joint actions typically generate only global rewards, making it difficult for each agent to deduce its own contribution to the team's success. Sometimes it is possible to design individual reward functions for each agent. However, these rewards are not generally available in cooperative settings and often fail to encourage individual agents to sacrifice for the greater good.

1. 集中式Critic： 确保智能体在非静态环境下训练的稳定性

2. 反事实基线： 处理多智能体信誉分配问题

3. 高效的Critic表征： 可用于大规模NNs

## 3.2 COMA

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, G_t] && \text{REINFORCE} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, Q_{\mathrm{w}}(s,a)] && \text{Q Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, A_{\mathrm{w}}(s,a)] && \text{优势Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, \delta] && \text{TD Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, \delta e] && \text{TD}(\lambda)\ \text{Actor-Critic}
\end{aligned}$$

1. $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory.
2. $\sum_{t'=t}^{\infty} r_{t'}$: reward following action $a_t$.
3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula.
4. $Q^\pi(s_t, a_t)$: state-action value function.
5. $A^\pi(s_t, a_t)$: advantage function.
6. $\boxed{r_t + V^\pi(s_{t+1}) - V^\pi(s_t)\text{: TD residual.}}$

策略梯度PG (Policy Gradient) → REINFORCE算法 $\xrightarrow{\text{增加评价网络评估Q}}$ Actor-Critic (AC)

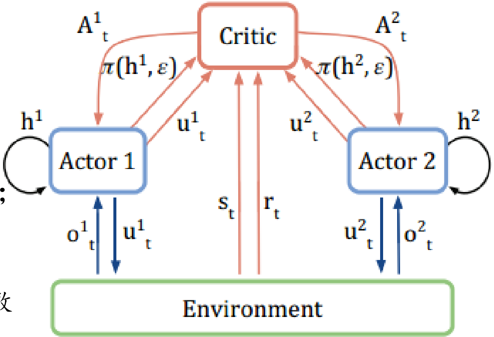
## 3.2 COMA

1. 集中式Critic

$$g = \nabla_{\theta^\pi} \log \pi(u|\tau_t^a) \boxed{(r + \gamma V(s_{t+1}) - V(s_t))}$$

只考虑全局回报，难以处理信誉分配问题

对于同构智能体，参数共享可以加速学习；




DRQN，处理Dec-POMDP

注:HAPPO提出参数共享也存在不足

## 2. 反事实Baseline

*difference rewards [Tumer & Agogino 2007]*

$$D^a = r(s, \mathbf{u}) - r(s, (\mathbf{u}^{-a}, c^a))$$

保持其他智能体动作不变的情况下，更改该智能体的动作，用默认动作$c^a$取代智能体的动作$u^a$后，对比其对团体reward的影响

局限性：

每个智能体都需要额外的仿真器来估计$r(s, (u^{-a}, c^a))$

需要预先定义的默认动作$c^a$，很多实际问题中很难设定

2. 反事实Baseline

- 利用$Q(s,a)$来估计difference rewards

$$g_k = \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_a \nabla_{\theta_k} \log \pi^a(u^a|\tau^a) A^a(s,\mathbf{u})\right]$$

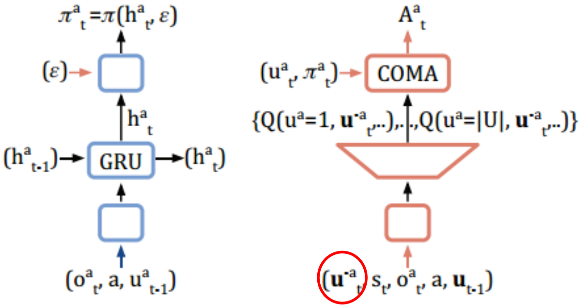
$$A^a(s,\mathbf{u}) = Q(s,\mathbf{u}) - \boxed{\sum_{u'^a} \pi^a(u'^a|\tau^a) Q(s,(\mathbf{u}^{-a},u'^a))}$$ 反事实baseline

评价当前动作的好坏不跟默认动作比了，而是跟当前策略的平均效果比，把平均策略当作默认策略

## 3.2 COMA

### 3.高效的Critic表征

Actor




Critic

其他智能体的动作

将其他智能体的动作当做输入，保留单个智能体动作数目的输出。

利用神经网络一次性计算出当前智能体所有动作的行为值函数，可以计算出各个动作的反事实基线

# QMIX

Rashid T, Samvelyan M, Schroeder C, et al. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning[C]//International Conference on Machine Learning. PMLR, 2018: 4295-4304.

# QMIX

当智能体数量变多，联合状态、动作空间维度增大，如果动作值函数的输入空间过大，则很难拟合出一个合适函数来表示真实的联合动作值函数。

能否设计一个集中式且可对每个智能体单独分解的$Q_{tot}$

$$Q_{tot} = \sum_{i=1}^{n} Q_i(\tau_i, a_i, ; \theta_i)$$

VDN中直接采用直接相加求和的方式得到$Q_{tot}$，未能有效利用全局状态信息

Sunehag P, Lever G, Gruslys A, et al. Value-decomposition networks for cooperative multi-agent learning[J]. arXiv preprint arXiv:1706.05296, 2017.

# QMIX

- 在训练学习过程中加入全局状态信息辅助，同样采用集中式学习，来提高算法稳定性。

- 针对每个智能体设计相应的局部值函数$Q_i$，输入为每个智能体的局部动作-观测历史,分布式执行，网络形式采用DRQN，处理Dec-POMDP

- 设计一个混合神经网络来整合每个智能体的局部值函数，考虑全局状态信息和非线性组合的方式得到联合动作值函数$Q_{tot}$，利用超参网络来学习混合网络参数

# QMIX

集中式且可分解的$Q_{tot}$

如果可以确保全局值函数与局部值函数的单调性约束

$$\frac{\partial Q_{tot}}{\partial Q_i} \geq 0, \forall i = \{1, \ldots, N\}$$

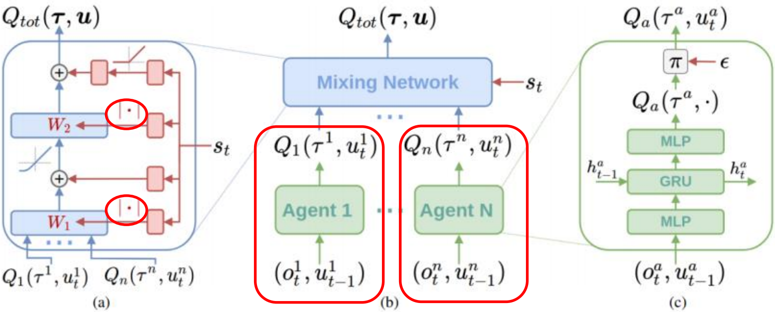
则有

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

只需要对每个局部Q最大化，即可确保全局Q最大化

从$Q_{tot}$中可以显示的提取分布式执行的各个智能体policy

## QMIX




智能体网络：利用DRQN构建局部$Q_i$

混合网络: 所有智能体局部$Q_i$作为输入，输出$Q_{tot}$

- 为了满足单调性约束，混合网络的权重限制为非负
- 权重和偏置由单独的超参数网络生成，网络输入为全局系统状态信息

Hausknecht M, Stone P. Deep recurrent q-learning for partially observable mdps[J]. arXiv preprint arXiv:1507.06527, 2015.

# QMIX

模型的训练

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

$$y^{tot}=r+\gamma\max_{\mathbf{u}'}Q_{tot}(\boldsymbol{\tau}',\mathbf{u}',s';\theta^-)$$

QMIX为端到端训练

损失函数为标准的DQN损失函数
b为batch size的数量，$\theta^-$为目标网络参数

由于满足单调性约束，对$Q_{tot}$的argmax操作不再随智能体数量呈指数增长，而是线性增长，极大提高了算法效率

# QMIX

1.；两步合作矩阵博弈 QMIX vs VDN

博弈矩阵

State 2A

| Agent 1 \ Agent 2 | A | B |
|---|---|---|
| A | 7 | 7 |
| B | 7 | 7 |

State 2B

| Agent 1 \ Agent 2 | A | B |
|---|---|---|
| A | 0 | 1 |
| B | 1 | 8 |

第一步

智能体1的动作决定下一步的矩阵博弈游戏

- 若智能体1采取动作A则跳转为State 2A游戏
- 若智能体1采取动作B则跳转为State 2B游戏

第二步

根据两个游戏的博弈矩阵，智能体1和2的动作决定合作博弈的全局回报

# QMIX

VDN学习结果

(a)

| | State 1 | | State 2A | | State 2B | |
|---|---|---|---|---|---|---|
| | *A* | *B* | *A* | *B* | *A* | *B* |
| *A* | 6.94 | 6.94 | 6.99 | 7.02 | -1.87 | 2.31 |
| *B* | 6.35 | 6.36 | 6.99 | 7.02 | 2.33 | 6.51 |

QMIX学习结果

(b)

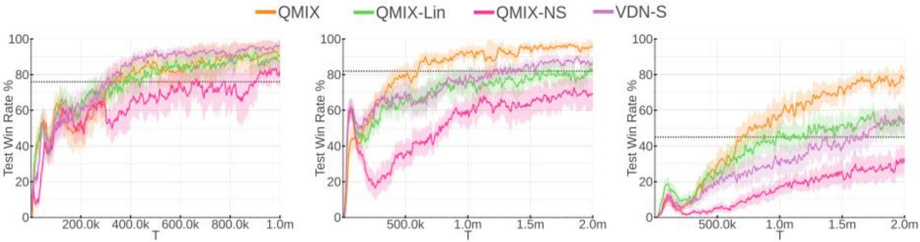
| | *A* | *B* | *A* | *B* | *A* | *B* |
|---|---|---|---|---|---|---|
| *A* | 6.93 | 6.93 | 7.00 | 7.00 | 0.00 | 1.00 |
| *B* | 7.92 | 7.92 | 7.00 | 7.00 | 1.00 | 8.00 |

VDN的$Q_{tot}$显示智能体1在第一步学习得到的是动作A，为局部最优策略

QMIX则可以学习到联合最优策略

# QMIX

星际争霸消融实验




(a) 3m (b) 2s_3z (c) 3s_5z

QMIX-LIN：移除混合网络的非线性

QMIX-NS： 不使用全局状态信息s

VDN-S：在VDN基础上加入全局状态信息

# QMIX

星际争霸微操实验

https://github.com/starry-sky6688/StarCraft

# QMIX

## 合作博弈问题- Q分解


- Sunehag, Peter, et al. "[Value-decomposition networks for cooperative multi-agent learning]." *arXiv preprint arXiv:1706.05296* (2017).
- Rashid, Tabish, et al. "[QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning]." *arXiv preprint arXiv:1803.11485* (2018).
- Son, Kyunghwan, et al. "[QTRAN: Learning to Factorize with Transformation for Cooperative Multi-Agent Reinforcement Learning]." *arXiv preprint arXiv:1905.05408* (2019).
- Peng, Bei, et al. "Facmac: Factored multi-agent centralised policy gradients." Advances in Neural Information Processing Systems 34 (2021): 12208-12221.

# MADDPG

Lowe R, Wu Y, Tamar Multi-agent actor-critic for mixed cooperative-competitive environments[J]. NIPS, 2017.

# MADDPG

COMA和QMIX适用于处理合作博弈问题

对于竞争博弈问题如何处理呢？

Multi-agent DDPG可用于处理合作或者竞争博弈问题

## MADDPG

### DDPG

- 结合了DQN和DPG，DQN用于高维输入离散动作空间，DPG用于低维输入连续动作空间

- $$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, G_t] && \text{REINFORCE} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, Q_w(s,a)] && \text{Q Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, A_w(s,a)] && \text{优势Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, \delta] && \text{TD Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, \delta e] && \text{TD}(\lambda)\ \text{Actor-Critic}
\end{aligned}$$

-

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left(r_{t+1} + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t)\right)$$

Continuous Control with Deep Reinforcement Learning (ICLR2016)

# MADDPG

## DDPG

- 结合了DQN和DPG，DQN用于高维输入离散动作空间，DPG用于低维输入连续动作空间

- 使用了DQN 的两种技术：Experience Replay 和Target Network

  - 对于critic和actor均有Target Network，采用软更新方式

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta' \qquad w' \leftarrow \tau w + (1-\tau)w' \qquad \tau \ll 1$$

$\theta'$为目标actor网络参数　　$w'$为目标critic网络参数

- 为了充分探索，利用添加噪声产生探索性动作

$$\pi'_\theta(s) = \pi_\theta(s) + \mathcal{N} \qquad \mathcal{N}\text{为噪声}$$

Continuous Control with Deep Reinforcement Learning (ICLR2016)

# MADDPG

## DDPG

Actor当前网络：负责策略网络参数θ的迭代更新，根据当前状态s选择当前执行的确定性动作$a$，用于和环境交互生成$s'$, r

Critic当前网络：负责价值网络参数w的迭代更新,计算当前Q值$Q_w(s,a)$

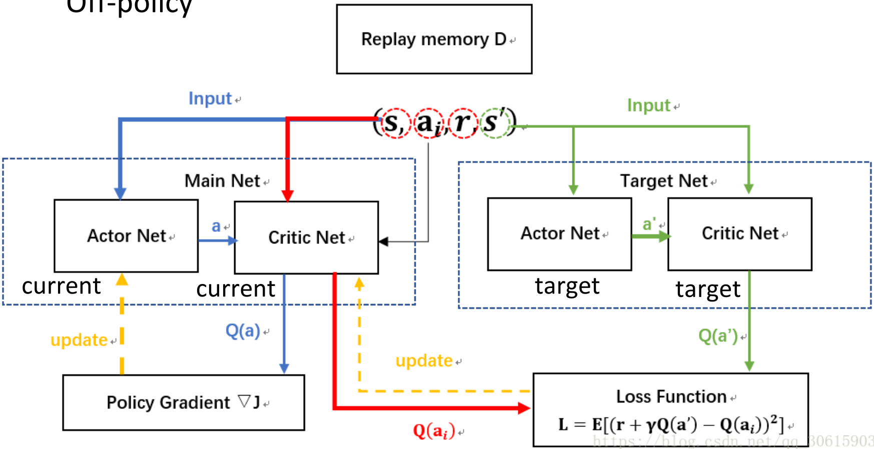
Critic目标网络：负责计算target Q值$Q(s',a')$, 网络参数 $w' \leftarrow \tau w+(1-\tau)w'$

Actor目标网络：负责根据经验回放池中采样的下一状态$s'$选择下一最优动作$a'$，估计target Q值，网络参数$\theta' \leftarrow \tau\theta+(1-\tau)\theta'$

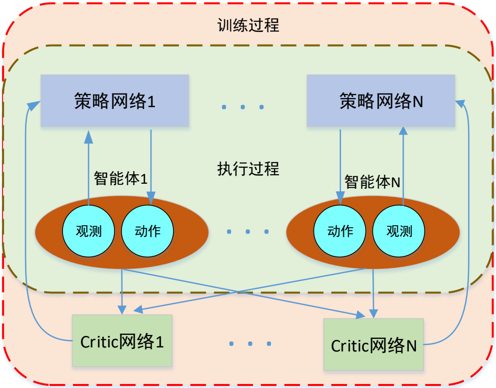
$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left(r_{t+1} + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t)\right)$$

# MADDPG

Off-policy



# MADDPG




- 集中式训练与分布式执行确保非静态环境下学习的稳定性
- 估计其他智能体策略，不再直接使用其他agent的策略
- 策略集合优化，提升策略网络的鲁棒性

## MADDPG

为了学习每个智能体i的策略，用DDPG的方法求该策略网络的梯度：

$$\nabla_{\theta_i} J(\mu_i) = E_{x,a\sim D}[\nabla_{\theta_i}\mu_i(a_i|o_i)\boxed{\nabla_{a_i} Q_i^{\mu}(x, a_1, \cdots, a_n)}|_{a_i=\mu_i(o_i)}]$$

经验池数据为 $(\mathbf{x}, \mathbf{x}', a_1, \ldots, a_N, r_1, \ldots, r_N)$

$Q_i^{\mu}$的更新方式为

Actor/Critic当前网络

$$\mathcal{L}(\theta_i) = \mathbb{E}_{\mathbf{x},a,r,\mathbf{x}'}[(Q_i^{\boldsymbol{\mu}}(\mathbf{x}, a_1, \ldots, a_N) - y)^2]$$

$$y = r_i + \gamma Q_i^{\boldsymbol{\mu}'}(\mathbf{x}', a_1', \ldots, a_N')|_{a_j'=\boldsymbol{\mu}_j'(o_j)}$$

Actor/Critic目标网络

$$\mu' = \{\mu_{\theta_1'}, \ldots, \mu_{\theta_N'}\}$$

$\theta_i'$为目标网络参数

# MADDPG

针对每个Critic网络，利用TD-error训练

Actor网络利用集中式Critic的DDPG更新

**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

2.估计其他智能体策略

$$y = r_i + \gamma Q_i^{\boldsymbol{\mu}'}(\mathbf{x}', a_1', \ldots, a_N')\big|_{a_j'=\boldsymbol{\mu}_j'(o_j)}$$

这里需要已知其他智能体的策略，可以通过对其他智能体的策略进行估计来实现

通过对每个智能体维护一个策略逼近器 $\hat{\boldsymbol{\mu}}_i^j$ 来逼近真实的策略 $\boldsymbol{\mu}_j$

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

极大似然估计与熵正则

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

## MADDPG

3. 策略集合优化

非静态环境，同时考虑其他智能体的动作，这种情况在竞争任务下经常会出现一个智能体针对其竞争对手过拟合出一个强策略。但是这个强策略是非常脆弱的。

第i个智能体的策略由一个具有K个子策略的集合构成，在每一个训练episode中只是用一个子策略，对每一个智能体，最大化其策略集合的整体奖励

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

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

# 实验

https://github.com/openai/multiagent-particle-envs

# 学习通信

[1] Sukhbaatar, Sainbayar, and Rob Fergus. "**Learning multiagent communication with backpropagation**." *Advances in Neural Information Processing Systems*. 2016.

[2] Foerster J N, Assael Y M, De Freitas N, et al. **Learning to communicate with deep multi-agent reinforcement learning[J]**. arXiv preprint arXiv:1605.06676, 2016.

[3] Peng, Peng, et al. "**Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games**." *arXiv preprint arXiv:1703.10069* 2 (2017).

[4] Jiang, Jiechuan, and Zongqing Lu. "**Learning attentional communication for multi-agent cooperation**." *Advances in Neural Information Processing Systems*. 2018.

[5] Kim, Daewoo, et al. "**Learning to Schedule Communication in Multi-agent Reinforcement Learning**." *arXiv preprint arXiv:1902.01554* (2019).

# 总结

## 马尔可夫博弈

| Minimax Q | Nash-Q | FOF | Team-Q/JAL |
|---|---|---|---|
| 零和博弈 | 一般和博弈 | 一般和博弈 | 完全合作 |

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

$$\boxed{V(s') = \max_a Q(s',a)}$$

$$\boxed{V(s') = \max_{\pi_1(s',\cdot)} \min_{a_2'\in A_2} \sum_{a_1'\in A_1} Q(s',a_1',a_2')\pi_1(s',a_1')}$$

$$\boxed{V(s') = NashQ_t^i(s') = \pi^1(s')\cdots\pi^n(s')\cdot Q_t^i(s')}$$

$$\boxed{V(s') = \max_{a_1',a_2',\dots,a_n'} Q(s',a_1',a_2',\dots,a_n')}$$

# 总结

分布式部分可观马尔可夫博弈

集中式训练分布式执行

- COMA
- QMIX
- MADDPG