

法律声明

- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。



关注 小象学院

第10讲 多Agent强化学习

强化学习

主讲人：叶梓

上海交通大学博士

主要研究方向：机器学习、深度学习、人工智能

本章内容

- MARL的背景
- 博弈论简介
- 完全合作
- 完全竞争
- 混合任务
- MADDPG
- 案例

背景介绍

- 多智能体系统由一群有自主性的，可互相交互的实体组成，它们共享一个相同的环境，通过感知器感知环境并通过执行器采取行动。
- 多智能体在现实生活中已有应用，比如说，多个机器人的控制，语言的交流，多玩家的游戏，以及社会问题分析等。
- 也有很多重要的应用场景牵涉到多个智能体之间的交互，在这种共同的交互演化过程中，会有新的行为出现，问题也会变得更加复杂。

背景介绍

- 传统的基于值函数的Q-Learning或者policy gradient 都很不适用于多智能体环境。
- 值函数方法面临的困难是，在训练过程中，各智能体都在变化，因此每个智能体的角度来看，环境都会变得不稳定。
 - 这给训练稳定性带来了挑战，并且直接阻碍了利用 experience replay，这对于稳定DQN之类的算法至关重要。
- 对于policy gradient方法，面临的困难是在要求多智能体协作的时候通常会有更大的方差，因此导致算法更难以收敛。
 - 因为agent的reward通常取决于许多agent的动作，agent自己的动作的reward表现出更多的可变性，从而增加了其梯度的方差。

多Agent环境的复杂性

- 维度爆炸：在多体强化学习中，状态空间变大，联结动作空间随智能体数量指数增长，因此多智能体系统维度非常大，计算复杂。
- 联结动作是指每个智能体当前动作组合而成的多智能体系统当前时刻的动作。
- 联结动作 $A_t = [a_{1,t}, a_{2,t}, \dots, a_{n,t}]^T$ ， $a_{i,t}$ 指第 i 个智能体在时刻 t 选取的动作。
- 目标奖励确定困难：多智能体系统中每个智能体的任务可能不同，但是彼此之间又相互耦合影响。奖励设计的优劣直接影响学习到的策略的好坏。

多Agent环境的复杂性

- 不稳定性：在多智能体系统中，多个智能体是同时学习的。
 - 当同伴的策略改变时，每个智能体自身的最优策略也可能会变化，这将对算法的收敛性带来影响。
 - 最好的策略可能会随着其他Agent策略的改变而改变。
- 更为严重的探索-利用两难：探索不光要考虑自身对环境的探索，也要对同伴的策略变化进行探索，可能打破同伴策略的平衡状态。这将使算法很难稳定，学习速度慢。

囚徒困境

- 在单agent中只需要考虑将自己的task做到最好就可以了。
- 但是在MARL的环境设置中，有些环境是存在竞争与合作的。
- 如果一味追求最优自己的reward，最终可能会使得在同个环境中其他agent与你竞争，最终既导致自己回报受到损伤，同时使所有的agent的回报之和受到损伤。



MARL的目标

- 在MARL中，主要关注两方面学习目标：稳定性(或称为收敛性)和适应性。
- 稳定性指Agent的策略会收敛至固定，而适应性确保性能不会因为其他Agent改变策略而下降。
 - 收敛至均衡态是稳定性的基本要求，这要求所有Agent的策略收敛至协调平衡状态，最常用的是纳什均衡。
- 适应性又体现在合理性或无悔性两个方面上。
 - 合理性是指，当其他Agent稳定时，Agent会收敛于最优反馈。
 - 无悔性是说最终收敛的策略，其回报要不差于任何其他策略。

博弈论简介：矩阵博弈

- 介绍多智能体强化学习，首先需要用到的一些博弈论相关的概念及定义。
- 一个矩阵博弈可以表示为 $\langle n, A_1, \dots, A_n, R_1, \dots, R_n \rangle$:
 - n 表示智能体数量， A_i 是第 i 个智能体的动作集，
 - $R_i : A_1 \times \dots \times A_n \rightarrow R$ 表示第 i 个智能体的奖励函数，
- 从奖励函数可以看出每个智能体获得的奖励与多智能体系统的联结动作有关，联结动作空间为 $A_1 \times \dots \times A_n$ 。

博弈论简介：矩阵博弈

猜硬币	囚徒博弈	剪刀 - 石头 - 布
$R_1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$	$R_1 = \begin{bmatrix} 5 & 0 \\ 10 & 1 \end{bmatrix}$	$R_1 = \begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}$
$R_2 = -R_1$	$R_2 = (R_1)^T$	$R_2 = -R_1$
完全混合策略下纳什均衡	纯策略下纳什均衡	完全混合策略下纳什均衡

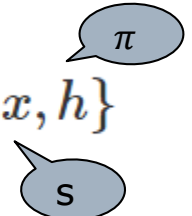
- 每个智能体的策略是一个关于其动作空间的概率分布，每个智能体的目标是最大化其获得的奖励值。
- 在博弈论的术语中，“纯策略”是指决定性策略；“混合策略”是指随机策略。

博弈论简介：随机博弈


- 随机博弈是将马尔科夫决策过程泛化到多Agent情形，定义如下：
- 随机博弈是一个多元组 $\langle n, S, A_1, \dots, A_n, f, R_1, \dots, R_n \rangle$ ， n 是Agent个数， S 是状态集合， A_i 是Agent i 的动作空间。
 - $A = A_1 \times \dots \times A_n$ ，是所有Agent的动作组成联合动作空间集。
 - $f: S \times A_1 \times \dots \times A_n \times S \rightarrow [0,1]$ 是状态转移函数；
 - $R_i: S \times A_1 \times \dots \times A_n \times S \rightarrow R, i=1, \dots, n$ 是每个Agent的回报函数。
 - 定义中概率转移和每个Agent的回报函数都是基于联合动作空间计算的。
 - 每个Agent的策略是： $\pi_i: S \times A_i \rightarrow [0,1]$ ，所有Agent的策略组合成联合策略 π 。

博弈论简介：随机博弈

□ 每个Agent在状态s下的期望回报为：

$$R_i^h(x) = E\left\{\sum_{k=0}^{\infty} \gamma^k r_{i,k+1} \mid x_0 = x, h\right\}$$


□ Q函数为：

$$Q_i^h(x, u) = E\left\{\sum_{k=0}^{\infty} \gamma^k r_{i,k+1} \mid x_0 = x, u_0 = u, h\right\}$$


□ 随机博弈可以分为静态博弈和动态博弈。

- 静态博弈对应系统中只有一个状态的情形。
- 多于一个状态的情形则为动态博弈。

博弈论简介：纳什均衡

- 在矩阵博弈中，如果联结策略 $(\pi_1^*, \dots, \pi_n^*)$ 满足

$$V_i(\pi_1^*, \dots, \pi_i^*, \dots, \pi_n^*) \geq V_i(\pi_1^*, \dots, \pi_i, \dots, \pi_n^*), \quad \forall \pi_i \in \Pi_i, i = 1, \dots, n$$

- 则它是一个纳什均衡。
- 总的来说，纳什均衡就是一个所有智能体的联结策略，而且在纳什均衡处，对于所有智能体而言，都不能在仅改变自身策略的情况下，来获得更大的奖励。
- 纳什均衡就是，其他玩家继续采用纳什均衡策略，而特定玩家无法通过改变自身的策略获得更大的回报的所有玩家的集合。
- 这时对应的策略就是纳什均衡策略（纳什策略）。

约翰-纳什

□ 从统计学看来，没有任何一个已经66岁的数学家或科学家能通过持续的研究工作，在他或她以前的基础上更进一步。但是，我仍然继续努力尝试。

□ 每次看（《美丽心灵》）的时候，我心里并不好受，但我还是认为这部电影有助于人们理解和尊重患有精神疾病的人。



博弈论简介：随机博弈

- 随机博弈(stochastic game / Markov game)是马尔可夫决策过程与矩阵博弈的结合，具有多个智能体与多个状态，即多智能体强化学习。
- 对于一个多智能体强化学习过程，就是找到每一个状态的纳什均衡策略，然后将这些策略联合起来。
- π_i 就是一个智能体 i 的策略，在每个状态选出最优的纳什策略。

博弈论简介：零和博弈与一般和博弈

□ 一般和博弈

- 一般和博弈是指任何类型的矩阵博弈，包括完全对抗博弈、完全合作博弈以及二者的混合博弈。
- 在一般和博弈中可能存在多个纳什均衡点。

□ 零和博弈

- 零和博弈中，两个智能体是完全竞争对抗关系，则 $R_1 = -R_2$ 。
- 在零和博弈中只有一个纳什均衡点，即使可能有很多纳什均衡策略，但是期望的奖励是相同的。

零和博弈与纳什策略

- 线性规划求解双智能体零和博弈
- 求解双智能体零和博弈的公式如下

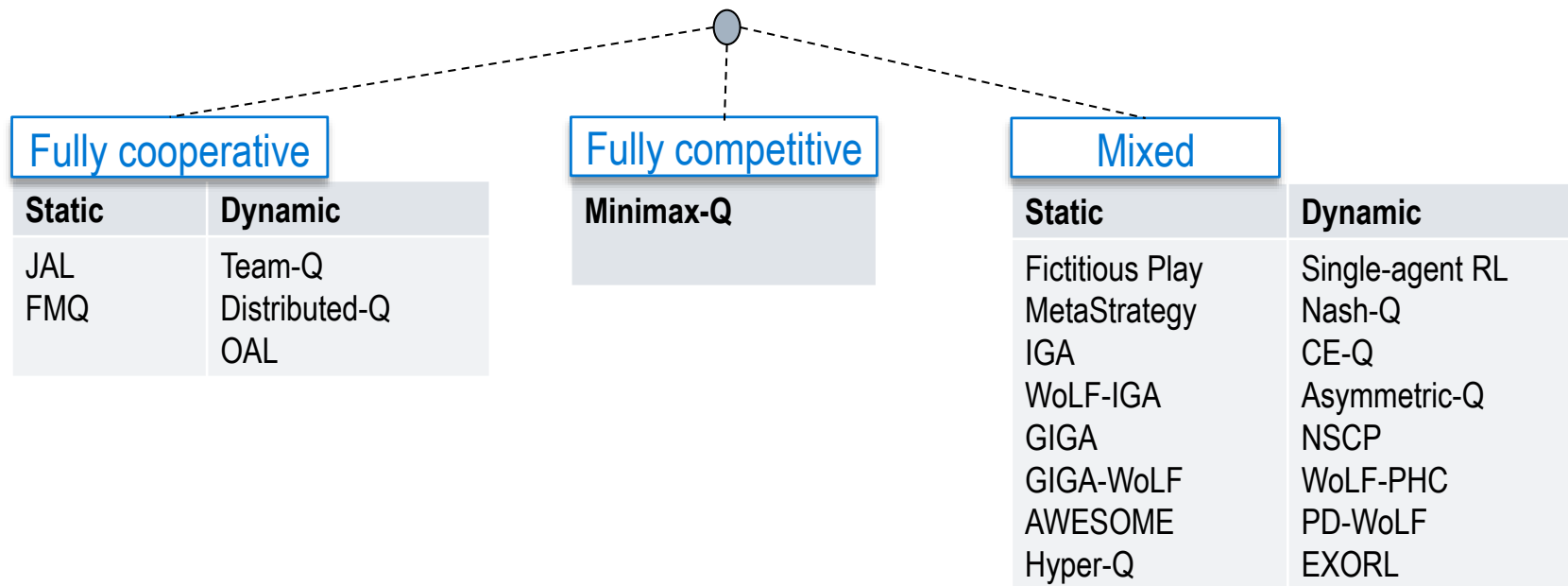
$$\max_{\pi_i} \min_{a_{-i} \in A_{-i}} \sum_{a_i \in A_i} Q_i^*(a_i, a_{-i}) \pi_i(a_i)$$

- 每个智能体最大化在与对手博弈中最差情况下的期望奖励值。
- MARL（多智能体强化学习）的纳什策略可以改写为：

$$\sum_{a_1, \dots, a_n \in A_1 \times \dots \times A_n} Q_i^*(s, a_1, \dots, a_n) \pi_1^*(s, a_1) \dots \pi_i^*(s, a_i) \dots \pi_n^*(s, a_n) \geq \sum_{a_1, \dots, a_n \in A_1 \times \dots \times A_n} Q_i^*(s, a_1, \dots, a_n) \pi_1^*(s, a_1) \dots \pi_i(s, a_i) \dots \pi_n^*(s, a_n)$$

MARL的分类

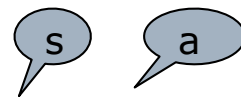
- 从不同维度来分类MARL算法，首先从任务的类型来分的话，MARL算法可以分成如下三大类：



- 从算法侧重的学习目标来分，关注稳定性的算法通常对其他Agent是独立无感知的，而侧重适应性的算法都需要能感知其他Agent，它们会为对手建模，来追踪对手的策略。

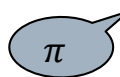
完全合作：team Q-learning

- 如果在随机博弈中，存在一个中心控制者，可以控制其他Agent的行动。那么就能求得最佳的联合动作值，随机博弈就退化成马尔科夫决策过程，并且可以用Q-learning算法求解。



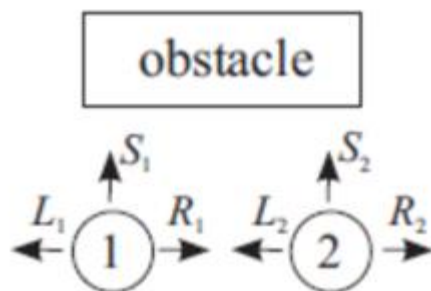
$$Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \alpha_k [r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1}, u') - Q_k(x_k, u_k)]$$

- 但是大部分系统是不存在中心控制者，那么可以假定其他Agent都是选择当前状态下最优的动作，在这种假定下，再选择对自己最优的动作，即：


$$\tilde{h}_i^*(x) = \operatorname{argmax}_{u_i} \max_{u_1, \dots, u_{i-1}, u_{i+1}, \dots, u_n} Q^*(x, u)$$

完全合作：team Q-learning

- 在特定状态下，一些组合动作才可能取得最优结果（此时Agent不一定是取自己最优的动作）。这时就需要协同机制，来协调Agent的动作。
- 在下图中，有两个Agent需要一起避开障碍物，并且保持之间一格的距离。
 - 如果两个同时往左或往右，那么有机会避开障碍物，回报值是10。
 - 如果Agent1往左，Agent2往右，虽然也能避开障碍物，但破坏了一格距离的限定，这时回报值是0。
 - 其他情况下，都会发送碰撞，回报值都是负的。



Q表	左2	直2	右2
左1	10	-5	0
直1	-5	-10	-5
右1	-10	-5	10

完全合作：team Q-learning

- 在没有负回报的确定性问题中，可以用分布式(distributed)Q-learning算法求解，这也不需要协调机制。
- 每个Agent保存一个本地的策略 $\tilde{h}_i(x)$ 和只取决于自身动作的本地的Q函数 $Q_i(x, u_i)$ 。Q函数的更新公式如下：

$$Q_{i,k+1}(x_k, u_{i,k}) = \max\{Q_{i,k}(x_k, u_{i,k}), r_{k+1} + \gamma \max_{u'_i} Q_{i,k}(x_{k+1}, u'_i)\}$$

- 本地的策略更新公式如下：

$$\bar{h}_{i,k+1}(x_k) = \begin{cases} u_{i,k} & \text{if } \max_{\bar{u}_i} Q_{i,k+1}(x_k, \bar{u}_i) > \max_{\bar{u}_i} Q_{i,k}(x_k, \bar{u}_i) \\ \bar{h}_{i,k}(x_k) & \text{otherwise} \end{cases}$$

完全合作：JAL

- 此外有一类间接协调算法，每个Agent通过为其他Agent建模，或统计不同动作的历史的回报值，来选择可能获得更高回报的动作。
- Joint Action Learners(JAL)算法中，每个Agent会为其其他Agent建模：

$$\sigma_j^i(u_j) = \frac{C_j^i(u_j)}{\sum_{\tilde{u}_j \in U_j} C_j^i(\tilde{u}_j)}$$

- $\sigma_j^i(u_j)$ 是Agent_i 对Agent_j的建模，
- $C_j^i(u_j)$ 统计了Agent_i观察到的Agent_j采取动作 u_j 的次数。

完全合作：FMQ

□ Frequency Maximum Q-value(FMQ)算法，统计每个动作取得最优回报的频率。

□ 然后进行玻尔兹曼探索，

$$\pi(a_t | s_t) \propto \exp(Q_\phi(s_t, a_t))$$

□ Q值计算：
$$\tilde{Q}_i(u_i) = Q_i(u_i) + \nu \frac{C_{max}^i(u_i)}{C^i(u_i)} \gamma_{max}(u_i)$$

□ 其中： $\gamma_{max}(u_i)$ 是采取动作 u_i 取得的最大回报值， $C_{max}^i(u_i)$ 是取得最大值的次数， $C^i(u_i)$ 是采取动作 u_i 的次数。

完全合作：FMQ

- 经过修改后，Agent就会倾向选择以往取得高回报的动作。
- 简单的说，假设在某次动作组合下，所有Agent取得了一个高回报值(完全合作下Agent的回报是一样的)，那么每个Agent在后面就会倾向选择在这个动作组合下各自采取的动作。
- 逐渐地就会增加这个最优动作组合对出现的概率。

完全合作

- ❑ 基于社会公约(social conventions)、互相交流等显式的协调机制。
- ❑ 在社会公约中，会规定每个Agent的先后顺序以及动作选择的先后顺序，这些信息是共享周知的。
- ❑ 例如在上面那两个Agent穿越障碍物的例子中，规定Agent 1优先于Agent 2，而且动作优先选择L。那么Agent 1在行动时，查Q表得知往左或往右都能取得最高收益，根据社会公约，采取L1。
- ❑ 轮到Agent 2的时候，根据社会公约推理可知Agent 1是选择L1的，这时候Agent 2就会选择L2，从而实现最大收益。

完全竞争： minimax-Q

□ 在完全竞争的随机博弈中，可以应用最小最大化(minimax)原则：在假定对手会采取使自己收益最小化的动作的情况下，采取使自己收益最大的动作。

■ 即“以最坏的恶意来揣测”对手。

□ 在两玩家零和随机博弈中，给定一个状态s，则定义第i个智能体的状态值函数为：

$$V_i^*(s) = \max_{\pi_i(s, \cdot)} \min_{a_{-i} \in A_{-i}} \sum_{a_i \in A_i} Q_i^*(s, a_i, a_{-i}) \pi_i(s, a_i), i = 1, 2$$

敌人会让我的收益最小

我要让自己收益最大

完全竞争：Minimax-Q

Minimax-Q 算法

1. 初始化 $Q_i(s, a_i, a_{-i}), V_i(s), \pi_i$

For iteration do

2. 第 i 个智能体根据当前状态 s 采用探索-利用策略得到动作 a_i 并执行

3. 得到下一个状态 s' ，以及智能体 i 获得的奖励 r_i ，并且观测智能体 $-i$ 在状态 s 执行的策略 a_{-i}

4. 更新 $Q_i(s, a_i, a_{-i})$:

$$Q_i(s, a_i, a_{-i}) \leftarrow Q_i(s, a_i, a_{-i}) + \alpha[r_i + \gamma V_i(s') - Q_i(s, a_i, a_{-i})]$$

5. 利用线性规划求解 $V_i^*(s) = \max_{\pi_i(s, \cdot)} \min_{a_{-i} \in \mathcal{A}_{-i}} \sum_{a_i \in \mathcal{A}_i} Q_i^*(s, a_i, a_{-i}) \pi_i(s, a_i), i=1,2$ 并更

新 $V_i(s)$ 与 $\pi_i(s, \cdot)$

End for

□ 其中， $-i$ 表示智能体 i 的对手。 $Q_i^*(s, a_i, a_{-i})$ 为联结动作状态值函数。

完全竞争： minimax-Q

- minimax-Q算法是对手独立的，不管对手如何选择，总能取得不差于minimax函数回报值。
- 但如果对手不是采取最优策略(即使自己的收益最小化)，而且能对对手建模，那么就可以取得更优的动作。
- 一种为对手建模的方式见公式：

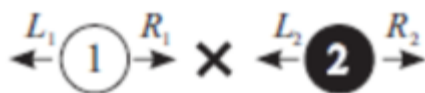
$$\sigma_j^i(u_j) = \frac{C_j^i(u_j)}{\sum_{\tilde{u}_j \in U_j} C_j^i(\tilde{u}_j)}$$

- 其中 $C_j^i(x, u_j)$ 是 Agent_i 观察到 Agent_j 在状态 x 下采取动作 u_j 的次数。

$$\hat{h}_j^i(x, u_j) = \frac{C_j^i(x, u_j)}{\sum_{\hat{u}_j \in U_j} C_j^i(x, \hat{u}_j)}$$

完全竞争： minimax-Q

- 在下面的例子中，Agent 1 需要到达X标志的位置，同时避免被Agent 2 捕抓到；Agent 2 的目标就是抓到Agent 1，两个Agent只能往左或往右移动。
- 右侧是Agent 1 的Q表。
 - 两个同时往左不产生任何收益。
 - 同时往右，则Agent 1达成目标且没被抓到，收益10。
 - Agent 1 往右而Agent 2 往左，则被抓到，收益-10。
 - Agent 1 往左，Agent 2 往右，虽然没达到指定位置，但远离Agent 2 所以收益1。
 - Agent 2 的Q表是Agent 1 的Q表取负。



Q表	左2	右2
左1	0	1
右1	-10	10

完全竞争

□ 根据minimax原则：

- Agent 1应该选择往左移动。因为往左的最小期望收益是0，往右的最小期望收益是-10。
- 对于Agent 2，其最优策略是往左移动保护目标位置。
- 但如果Agent 2不是采取最优策略而是往右走，并且Agent 1通过建模能预测到，那么Agent 1就可以往右移动，达成目标。

混合任务：Nash Q-learning

- Nash Q-Learning算法是将Minimax-Q算法从零和博弈扩展到多人一般和博弈的算法。
- 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。
- Nash Q-Learning算法在合作性均衡或对抗性均衡的环境中能够收敛到纳什均衡点，其收敛性条件是：
 - 在每一个状态 s 的阶段博弈中，都能够找到一个全局最优点或者鞍点，只有满足这个条件，Nash Q-Learning算法才能够收敛。

混合任务：Nash Q-learning

- Nash Q-learning 算法需要观测所有智能体的动作 a_i 与奖励值 r_i 。

Nash Q-Learning 算法

1. 初始化 $Q_i(s, a_1, \dots, a_n) = 0 \forall a_i \in A_i$

For iteration do

2. 第 i 个智能体根据当前状态 s 采用探索-利用策略得到动作 a_i 并执行。

3. 得到下一个状态 s' ，以及智能体 i 观测所有智能体的奖励 r_1, \dots, r_n ，并且观测所有智能体在状态 s 执行的策略 a_1, \dots, a_n 。

4. 更新 $Q_i(s, a_1, \dots, a_n)$:

$$Q_i(s, a_1, \dots, a_n) \leftarrow Q_i(s, a_1, \dots, a_n) + \alpha[r_i + \gamma \text{Nash}Q_i(s') - Q_i(s, a_1, \dots, a_n)]$$

5. 利用二次规划求解状态 s 处的纳什均衡策略并更新 $\text{Nash}Q_i(s)$ 与 $\pi_i(s, \cdot)$ 。

End for

混合任务：Nash Q-learning

- NashQ值是指当所有智能体从下一步开始执行联合纳什均衡策略时的未来折扣回报的预期总和。
- 与Minimax-Q算法相同，它也只满足收敛性，不满足合理性：只能收敛到纳什均衡策略，不能根据其他智能体的策略来优化调剂自身的策略。
- 同样地，Nash Q-Learning算法求解二次规划的过程也非常耗时，降低了算法的学习速度。

混合任务：FFQ

- ❑ Friend-or-Foe Q-Learning (FFQ) 算法也是从Minimax-Q算法拓展而来。
- ❑ 为了能够处理一般和博弈，FFQ算法对一个智能体 i ，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，因此对每个智能体而言都有两组。
- ❑ 这种情况下，一个 n 智能体的一般和博弈就转化为了一个两智能体的零和博弈。

混合任务：FFQ

Friend-or-Foe Q-Learning 算法

1. 初始化 $V_i(s) = 0, Q_i(s, a_1, \dots, a_{n_1}, o_1, \dots, o_{n_2}) = 0$, (a_1, \dots, a_{n_1}) 表示 i 所有 friend 的动作, (o_1, \dots, o_{n_2}) 表示 i 所有 foe 的动作,

For iteration do

2. 第 i 个智能体根据当前状态 s 采用探索-利用策略得到动作 a_i 并执行。
3. 得到下一个状态 s' , 以及智能体 i 观测自身的奖励 r_i , 并且观测所有 friend 的动作 (a_1, \dots, a_{n_1}) 与所有 foe 的动作 (o_1, \dots, o_{n_2}) 。
4. 更新 $Q_i(s, a_1, \dots, a_{n_1}, o_1, \dots, o_{n_2}) = 0$:
$$Q_i(s, a_1, \dots, a_{n_1}, o_1, \dots, o_{n_2}) \leftarrow Q_i(s, a_1, \dots, a_{n_1}, o_1, \dots, o_{n_2}) + \alpha[r_i + \gamma V_i(s') - Q_i(s, a_1, \dots, a_{n_1}, o_1, \dots, o_{n_2})]$$
5. 利用线性规划求解状态 s 处的纳什均衡策略并更新 $V_i(s)$ 与 $\pi_i(s, \cdot)$, 更新公式如下。

$$V_i(s) = \max_{\pi_1(s, \cdot), \dots, \pi_{n_1}(s, \cdot)} \min_{o_1, \dots, o_{n_2} \in O_1 \times \dots \times O_{n_2}} \sum_{a_1, \dots, a_{n_1} \in A_1 \times \dots \times A_{n_1}} Q_i(s, a_1, \dots, a_{n_1}, o_1, \dots, o_{n_2}) \pi_1(s, a_1), \dots, \pi_{n_1}(s, a_{n_1})$$

End for

混合任务：WoLF-PHC

- 上述方法都需要在学习过程中维护Q函数，假设动作空间 $|A|$ 与状态空间 $|S|$ 都是离散，且每个智能体的动作空间相同。
- 我们期望每个智能体只用知道自己的动作来维护Q值函数，这样空间就降到了 $|S||A|$ 。
- WoLF-PHC就是这样的算法，每个智能体只用保存自己的动作来完成学习任务。
- WoLF-PHC是将“Win or Learn Fast”规则与 policy hill-climbing 算法结合。

混合任务：WoLF-PHC

- WoLF是指，当智能体做的比期望值好的时候小心缓慢的调整参数，当智能体做的比期望值差的时候，加快步伐调整参数。
- PHC(Policy Hill-Climbing)是一种单智能体在稳定环境下的一种学习算法。
- PHC 算法的核心就是通常强化学习的思想，增大能够得到最大累积期望的动作的选取概率。
- PHC 算法具有合理性，能够收敛到最优策略。

混合任务：PHC算法

PHC 算法

1. 初始化 $Q(s, a) = 0, \pi(s, a) = \frac{1}{|A|}$

For iteration do

2. 智能体根据当前状态 s 采用探索-利用策略得到动作 a 并执行。
3. 观测下一个奖励值 r 以及下一个状态 s' 。
4. 更新 $Q(s, a)$:

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

5. 根据 $Q(s, a)$, 对每一个 $a \in A$ 更新 $\pi(s, a)$

$$\pi(s, a) \leftarrow \pi(s, a) + \Delta_{sa}$$

其中

$$\Delta_{sa} = \begin{cases} -\delta_{sa} & \text{若 } a \neq \operatorname{argmax}_{a'} Q(s, a') \\ \sum_{a' \neq a} \delta_{sa'} & \text{否则} \end{cases}$$

其他动作
的累加

$$\delta_{sa} = \min(\pi(s, a), \frac{\delta}{|A| - 1})$$

End for

□ PHC 算法由两部分组成，第一部分源自 RL，通过 Q 学习来计算 Q 值；第二部分源自博弈论，用于改进当前的策略。

□ PHC 的关键步骤是：选择最大行为值的概率，以一个较小的学习速率 $\delta \in (0, 1]$ 逐渐增大，使得策略不断改善。

混合任务：WoLF-PHC

- WoLF-PHC)是一种启发式算法。在WoLF-PHC算法中，定义了两种策略，即当前策略 $\pi_i(s,a)$ 和平均策略 $\bar{\pi}_i(s,a)$ 。
- 当前策略是一种概率分布函数，初始值为 $\pi_i(s,a) = \frac{1}{|A_i|}$ ，这个概率分布函数当Agent选择动作 a 时进行更新。
- 更新方法是：对于Q函数来说，最好的动作即 $a = \operatorname{argmax}_a Q(s,a')$ ，则增加概率，其他动作则降低概率。

混合任务：WoLF-PHC

- WoLF-PHC会不断更新平均策略，并和当前策略进行比较。如果当前策略平均奖励值大于平均策略的奖励值，即：

$$\sum_{a_i \in A_i} \pi_i(s, a_i) Q_i(s, a_i) > \sum_{a_i \in A_i} \bar{\pi}_i(s, a_i) Q_i(s, a_i)$$

- 则认为Agent是“winning”的，此时平均策略将采用 δ_{win} 速率来慢慢更新策略；
- 否则，认为当前Agent是“losing”的，用比较大的 δ_{lose} 来更快的自适应学习。

混合任务：WOLF-PHC

WoLF-PHC 算法

1. 初始化 $Q_i(s, a_i) = 0$, $\pi_i(s, a_i) = \frac{1}{|A_i|}$, $\bar{\pi}_i(s, a_i) = \frac{1}{|A_i|}$, $\delta_i > \delta_w$, $C(s) = 0$ 为状态 s 出现的次数。

For iteration do

2. 智能体 i 根据当前状态 s 采用探索-利用策略得到动作 a_c 并执行。
3. 观测下一个奖励值 r_i 以及下一个状态 s' 。
4. 更新 $Q_i(s, a_c)$:

$$Q_i(s, a_c) \leftarrow Q_i(s, a_c) + \alpha[r_i + \gamma \max_{a'} Q_i(s', a') - Q_i(s, a_c)]$$

5. 对每一个 $a_i \in A_i$, 更新平均估计策略 $\bar{\pi}_i(s, a_i)$:

$$C(s) = C(s) + 1$$

$$\bar{\pi}_i(s, a_i) = \bar{\pi}_i(s, a_i) + \frac{1}{C(s)} [\pi_i(s, a_i) - \bar{\pi}_i(s, a_i)]$$

6. 根据 $Q_i(s, a_i)$, 对每一个 $a_i \in A_i$ 更新 $\pi_i(s, a_i)$:

$$\pi_i(s, a_i) \leftarrow \pi_i(s, a_i) + \Delta_{sa}$$

其中

$$\Delta_{sa_i} = \begin{cases} -\delta_{sa_i} & \text{若 } a_i \neq \arg\max_{a'} Q(s, a') \\ \sum_{a' \neq a_i} \delta_{sa'} & \text{否则} \end{cases}$$

$$\delta_{sa} = \min(\pi_i(s, a_i), \frac{\delta}{|A_i| - 1})$$

$$\delta = \begin{cases} \delta_w & \text{若 } \sum_{a_i \in A_i} \pi_i(s, a_i) Q_i(s, a_i) > \sum_{a_i \in A_i} \bar{\pi}_i(s, a_i) Q_i(s, a_i) \\ \delta_l & \text{否则} \end{cases}$$

End for

WoLF-PHC 算法不用观测其他智能体的策略、动作及奖励值，需要更少的空间去记录 Q 值，

WoLF-PHC 算法通过 PHC 算法进行学习改进策略的，所以不需要使用线性规划或者二次规划求解纳什均衡，算法速度得到了提高。

Q学习

增量更新

win

lose

MADDPG

- MADDPG是一种actor-critic方法的变种，在考虑其他智能体的action policy的同时，能够成功地学到多智能体的协同policy。
- MADDPG还引入了一种训练规则，它集成了各个智能体的policy，来得到一个更加鲁棒的多智能体policy。
- MADDPG是一种通用的多智能体学习算法：
 - 可以学习到在执行时只使用本地信息（即它们自己的观察）的策略，
 - 不假设环境动力学的可微模型或智能体之间通信方法的任何结构，
 - 不仅适用于合作交互而且适用于涉及物质和信
息行为的竞争或混合交互环境（指合作和竞争的混合环境）

MADDPG

- MADDPG是对actor-critic policy gradient方法的简单扩展：
 - critic增加了其他智能体的policy的额外信息，
 - actor则只能接触到本地的信息（也就是该智能体自身的）。
- 提出MADDPG的一个主要动机是：如果知道所有智能体采取的行动，即使策略发生变化，环境也是稳定的，因为对于任何 $\pi_i = \pi'_i$ ，有 $P(s'|s, a_1, \dots, a_N) = P(s'|s, a_1, \dots, a_N, \pi'_1, \dots, \pi'_N)$ 。
- 只要没有像大多数传统的MARL方法那样，明确地以其他智能体的行为作为条件，那么就没关系。

MADDPG

- 具体地说，考虑policy由 $\theta=\{\theta_1, \dots, \theta_N\}$ 参数化的具有N个智能体的博弈，令所有智能体策略的集合为 $\pi=\{\pi_1, \dots, \pi_N\}$ 。那么可以写出智能体i的期望收益的梯度， $J(\theta_i)=E[R_i]$ 如下：

$$\nabla_{\theta_i} J(\theta_i) = E_{s \sim \rho^\pi, a_i \sim \pi_i} [\nabla_{\theta_i} \log \pi_i(a_i | o_i) Q_i^\pi(x, a_1, \dots, a_n)]$$

- 这里 $Q_i^\pi(x, a_1, \dots, a_N)$ 是一个集中的动作值函数，它将所有智能体的动作 a_1, \dots, a_N ，加上一些状态信息 x 作为输入，然后输出智能体i的Q值。
- 在最简单的情况下， x 可以包含所有智能体的观测值， $x=(o_1, \dots, o_N)$ ，也可以包含附加的状态信息。

MADDPG

- 扩展到确定性策略：现在考虑N个策略 μ_{θ_i} ，参数为 θ_i （缩写为 μ_i ），那么梯度写成：

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

- 经验重放缓冲区D包含元组 $(x, x', a_1, \dots, a_N, r_1, \dots, r_N)$ ，记录了所有智能体的经验。集中的动作值函数 Q_i^μ 按如下方式更新：

$$L(\theta_i) = E_{x,a,r,x'} [(Q_i^\mu(x, a_1, \dots, a_N) - y)^2], y = r_i$$

$$y = r_i + \gamma Q_i^{\mu'}(x', a'_1, \dots, a'_N) |_{a'_j = \mu'_j(o_j)}$$

- 其中 $\mu' = \{\mu_{\theta'_1}, \dots, \mu_{\theta'_N}\}$ 是具有延迟参数 θ'_i 的目标策略集合。

MADDPG

- 每个智能体可以额外保有一个与智能体j的真实策略 μ_j 有关的近似值 $\hat{\mu}_i^j$ 。
- 这个近似策略通过最大化智能体j的动作对数概率加上一个熵正则化项来进行学习：

$$L(\phi_i^j) = -E_{o_j, a_j} [\log \hat{\mu}_i^j(a_j | o_j) + \lambda H(\hat{\mu}_i^j)]$$

- 其中H是策略分布的熵。用近似的策略，上页中的y可以用如下计算的近似值 \hat{y} 代替：

$$\hat{y} = r_i + \gamma Q_i^{\mu'}(x', \hat{\mu}_i^{I1}(o_1), \dots, \mu_i'(o_i), \dots, \hat{\mu}_i^{IN}(o_N))$$

MADDPG

- 为了获得对竞争智能体策略变化更鲁棒的多智能体策略，可对K个不同的子策略进行汇总。
- 在每个回合的博弈中，为每个智能体随机地选择一个特定的子策略执行。
- 假设策略 μ_i 是K个不同子策略的集合，子策略k由 $\mu_i^{(k)}$ 表示。对于智能体i，最大化集成的目标值：

$$J_e(\mu_i) = E_{k \sim \text{unif}(1, K), s \sim p^\mu, a \sim \mu_i^{(k)}} [R_i(s, a)]$$

- 由于不同的子策略将在不同的博弈回合中执行，因此可为智能体i的每个子策略 $\mu_i^{(k)}$ 维护一个重播缓冲区 $D_i^{(k)}$ 。可以推出集成的目标值关于 $\theta_i^{(k)}$ 的梯度如下：

$$\nabla_{\theta_i^{(k)}} J_e(\mu_i) = \frac{1}{K} E_{x, a \sim D_i^{(k)}} [\nabla_{\theta_i^{(k)}} \mu_i^{(k)}(a_i | o_i) \nabla_{a_i} Q^{\mu_i}(x, a_1, \dots, a_N) |_{a_i = \mu_i^{(k)}(o_i)}]$$

MADDPG

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 $a = (a_1, \dots, a_N)$ 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 $y^j = r_i^j + \gamma Q_i^{\mu'}(\mathbf{x}'^j, a_1^j, \dots, a_N^j)|_{a_k^j = \mu_k'(o_k^j)}$

 Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S} \sum_j \left(y^j - Q_i^{\mu}(\mathbf{x}^j, a_1^j, \dots, a_N^j) \right)^2$

 Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S} \sum_j \nabla_{\theta_i} \mu_i(o_i^j) \nabla_{a_i} Q_i^{\mu}(\mathbf{x}^j, a_1^j, \dots, a_i, \dots, a_N^j)|_{a_i = \mu_i(o_i^j)}$$

end for

 Update target network parameters for each agent i :

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

end for

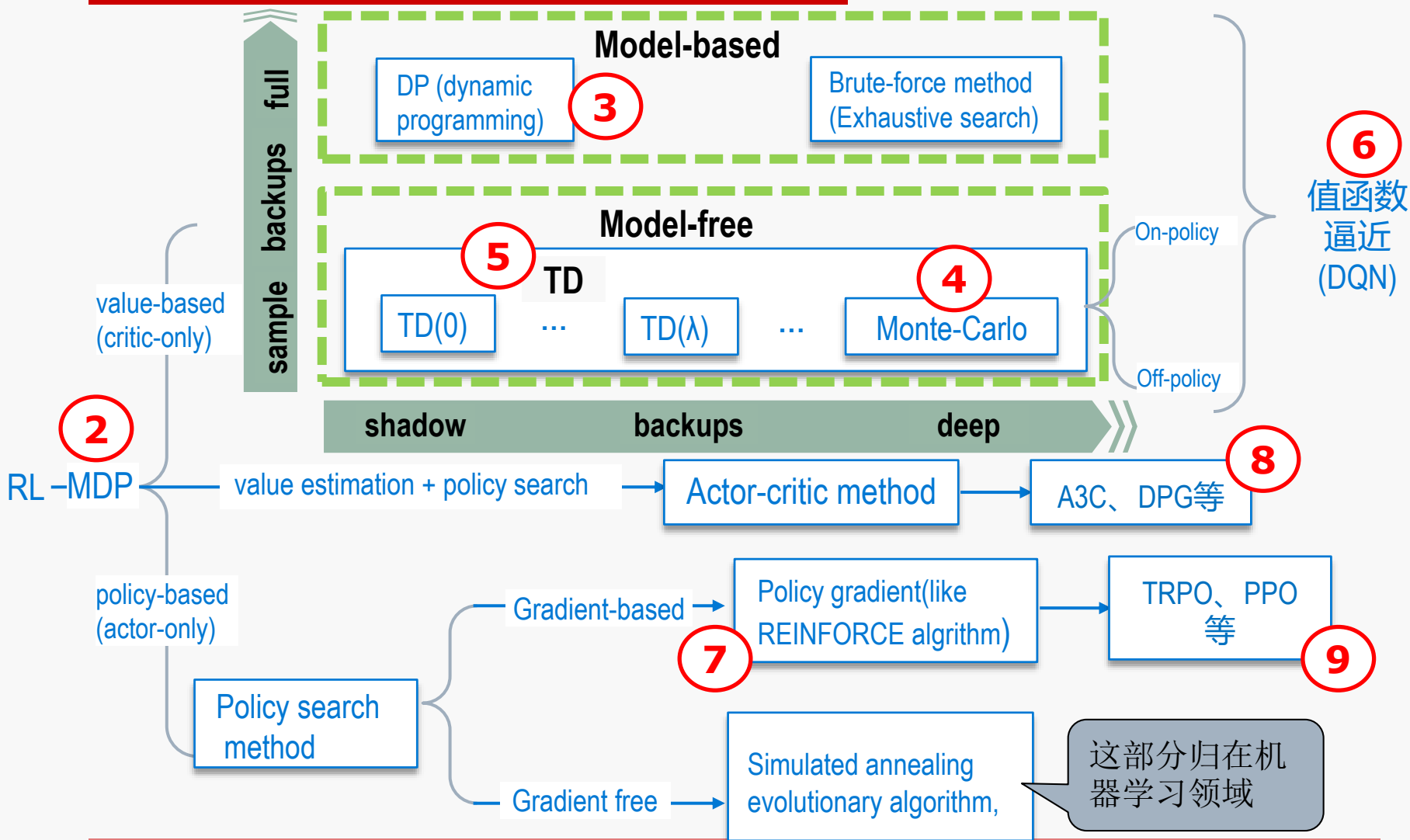
end for

参考资料

- ❑ <http://www.algorithmdog.com/multi-agent-rl>
- ❑ <https://blog.csdn.net/qiusuoxiaozi/article/details/79066612>
- ❑ <https://zhuanlan.zhihu.com/p/53474965>
- ❑ <https://blog.csdn.net/qiusuoxiaozi/article/details/79066612>
- ❑ <https://zhuanlan.zhihu.com/p/53811876>

RL学习回顾

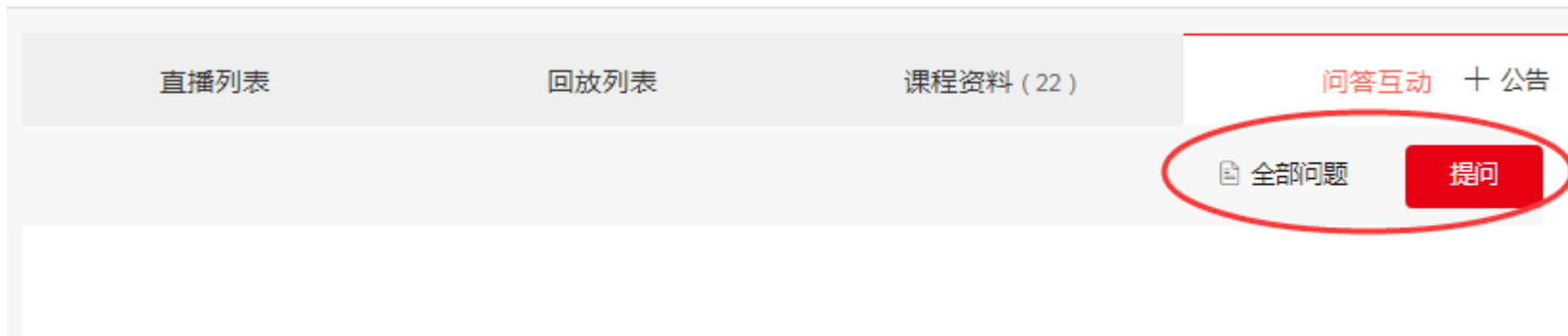
10 多Agent强化学习



问答互动

在所报课的课程页面，

- 1、点击“全部问题”显示本课程所有学员提问的问题。
- 2、点击“提问”即可向该课程的老师 and 助教提问问题。



联系我们

小象学院：互联网新技术在线教育领航者

— 微信公众号：**小象学院**

