多智能体强化学习实训
Multi-agent Reinforcement Learning

Lecture 9.1:
MARL with Actor Critic

教师：张寅 zhangyin98@zju.edu.cn

助教：邓悦 devindeng@zju.edu.cn

王子瑞 ziseoiwong@zju.edu.cn

李奕澄 yichengli@zju.edu.cn

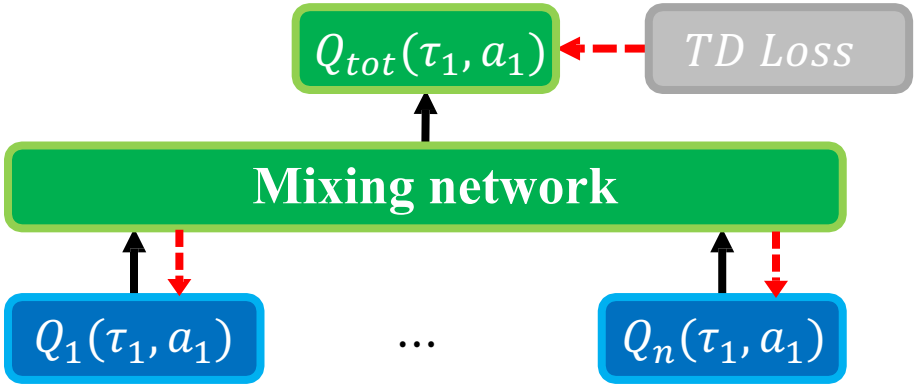
浙江大学计算机学院

回顾

- 智能体无法获得全局状态，只能看到全局状态的投影：
  - 观测：$o_i \in \Omega$
  - 观测函数：$o_i \in \Omega \sim O(s, i)$
- 智能体i的分布式策略：$\pi_i(\tau_i): T \to A$
  - 行为-观测历史：$\tau_i \in T = (\Omega \times A)^*$
- 集中训练分布式执行（CTDE）：
  - 训练过程中智能体可以获取全局状态
  - 测试过程中智能体智能看到局部观测

回顾

- 可扩展的集中训练与分布式执行过程



$$V(\tau';\theta^-) = \max_{a'} Q_{tot}(\tau', a';\theta^-)$$

$$L(\theta) = \mathbb{E}[\left(r + \gamma V(\tau';\theta^-) - Q_{tot}(\tau, a;\theta)\right)^2]$$

$$Q_{tot}(\tau, a) = f(Q_1(\tau_1, a_1), \dots, Q_n(\tau_n, a_n))$$

- 满足IGM原则：智能体个体最优决策即全局最优联合决策。

$$\operatorname{argmax}_a Q_{tot}(\tau, a) = \begin{pmatrix} \operatorname{argmax}_{a_1} Q_1(\tau_1, a_1) \\ \dots \\ \operatorname{argmax}_{a_n} Q_n(\tau_n, a_n) \end{pmatrix}$$

价值？策略？价值+策略

- Value-based的算法的核心是要估准每个状态-行为价值函数$Q_i(\tau, a)$，再根据最大值 $Q_i(\tau, a)$ 选择最优的行为。即 $Q_i(\tau, a)$ 的准确性可以影响策略的优劣与收敛。大多数提升算法在解决连续空间、过高估计等问题。
- Policy-based方法建立输入和输出之间的可微参数模型，然后通过梯度优化搜索合适的参数，其输出为动作的分布而不是状态动作价值。多数提升算法在于解决MC采样带来的大方差问题以及奖励裁剪问题。
- AC-based方法中Actor前身是Policy Gradient，可以轻松地在连续空间内选择合适动作。但是Actor根据每个episode的累积奖励更新，所以效率比较慢。用一个value-based的算法作为Critic就可以使用TD方法实现单步更新，这其实可以看做是拿偏差换方差。

课程大纲

- **MADDPG**
- **COMA**
- **LICA**
- **MAPPO**

回顾：确定性策略方法

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled gradient:

$$\nabla_{\theta^\mu}\mu|_{s_i} \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

动作上的噪声

离线策略

更新critic网络（a_i带有噪声）

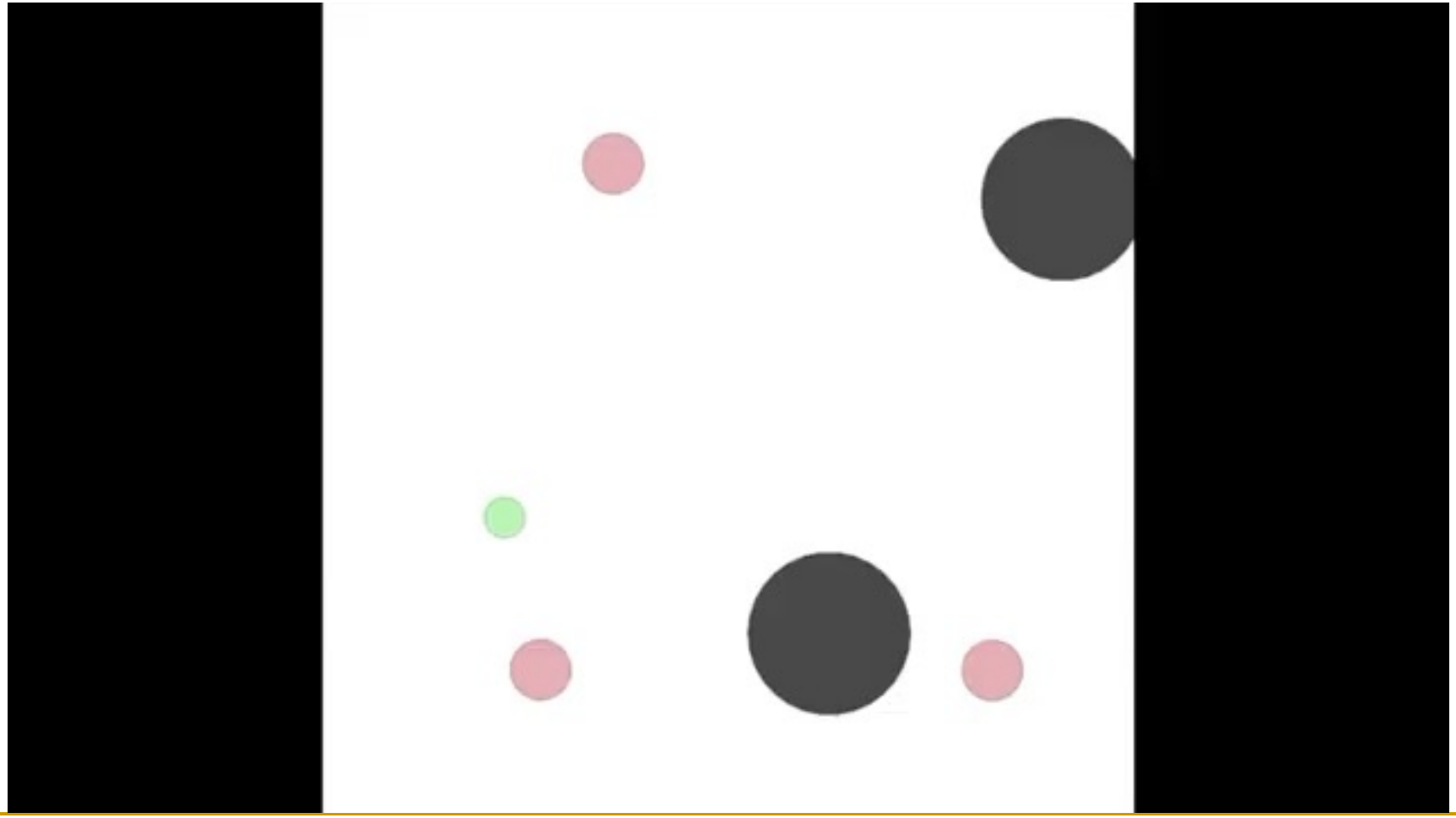
目标critic网络

目标actor网络

更新actor网络

MADDPG

- 三个慢捕食者追一个快猎物，尽量避免障碍物



传统算法在多智能体环境中有以下几个困境

- 环境的变化由所有智能体共同影响，对于单个智能体，环境是不稳定的，这违反了Q-learning所需的马尔可夫假设；
- 由于环境的不稳定，策略不同时，状态转移矩阵也不同，因此不能直接将过去经验(s, a_i, r_i, s')进行回放；
- 策略梯度方法中大方差的问题加剧。基于这样的局限性，使用集中式训练、分布式执行的方式。

MADDPG

- 学习到的策略可以分布式执行，即智能体根据自己的观察结果来决策；
- 不需要假定环境动态系统是可微的，也不需要假设智能体之间的通讯方式有任何特性结构，即世界模型和通信模型都不要求是可微的；
- 因为每个智能体最大化各自的累积奖励，MADDPG不仅可以应用于具有明确通信渠道的合作博弈，还可以应用于竞争博弈。

MAAC架构

- 随机离散Multi-Agent的AC思路：
  - N个智能体，策略参数分别为：$\theta = \{\theta_1, \dots, \theta_N\}$，策略：$\pi = \{\pi_1, \dots \pi_N\}$
  - 针对智能体i的策略梯度公式：

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{s \sim p^{\pi}, a \sim \pi_\theta}[\nabla_{\theta_i} log \pi_i(a_i|o_i) Q_i^{\pi}(x, a_1, \dots a_N)]$$

  - 其中$Q_i^{\pi}(x, a_1, \dots a_N)$是一个集中的动作价值函数，它将所有智能体的动作$a_1, \dots a_N$，以及一些状态信息$x$作为输入，输出每个智能体的动作价值；
  - x中包含所有智能体的观测信息$x = (o_1, \dots o_N)$，以及其他额外信息，例如通信信息等；
  - 由于Q_i^{π}都是独立学习到，因此每个智能体可以有任何的奖励结构，包括合作或竞争以及混合奖励。

MADDPG

- MADDPG算法思路基于以下原理：
  - 每个智能体都只输出一个确定性动作，而不是基于概率分布π采样的随机变量。则：

$$P(s'|a_1,\dots a_N,\pi_1,\dots \pi_N)=P(s'|a_1,\dots a_N)=P(s'|a_1,\dots a_N,\pi_1',\dots \pi_N'),\qquad \pi_i'\neq\pi_i$$

  - 即不论策略是否相同，只要其产生的动作a_i相同，那么其状态转移可以视为不变。
  - 如果已知各个智能体的动作，即便生成的策略不同，环境依旧是稳定的。
  - 可以直接将DDPG的目标损失拓展到多智能体版本。

MADDPG

- 采用经验回放缓存的模式，将交互数据存入$\mathcal{D}$中，即$(x, a_1, \dots a_N, r_1, \dots, r_N, x')$，从缓存中采样数据，根据如下公式训练：

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

- 其中全局中心化的Q_i^{μ}函数采用如下公式更新：

$$L(\theta_i) = \mathbb{E}_{x,a,r,x'}\left[\left(Q_i^{\mu}(x, a_1, \dots, a_N) - y\right)^2\right]$$

$$y = r_i + \gamma Q_i^{\mu'}(x', a_1', \dots a_N')|a_j' = \mu_j'(o_j)$$

其他智能体策略

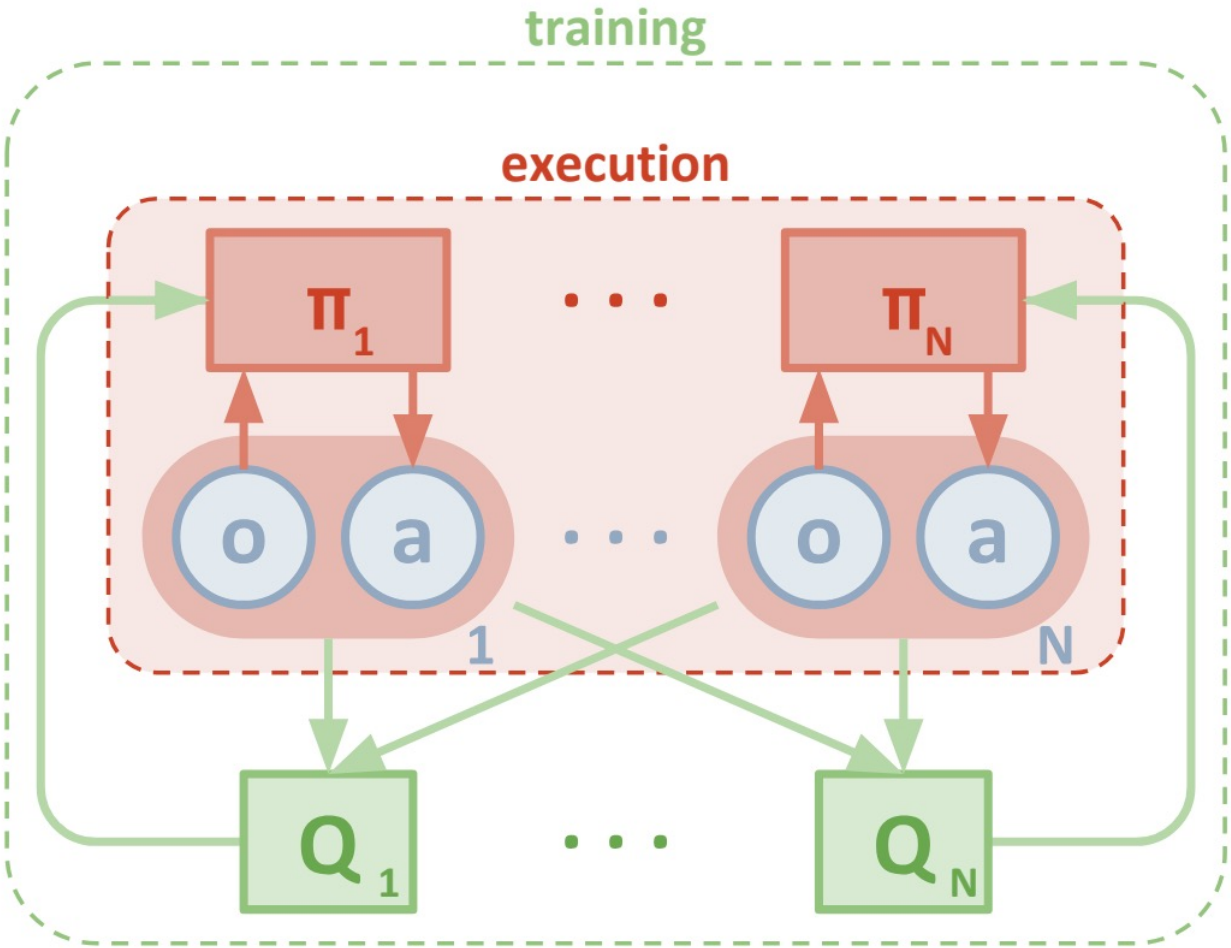
- 由于Q网络的输入不仅包含自身观测、通讯信息等x，还包含自身的和其他智能体的行为$(a_1, \dots a_N)$。因此每个智能体i维护一个对智能体j策略的一个近似$\hat{\mu}_{\phi_i^j}$推断。智能体j的真实策略为μ_j，该策略可以通过智能体j行为对数概率以及熵正则的方式进行学习：

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

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

- 其中H为策略分布的熵， $\hat{\mu}_i'^j$为$\hat{\mu}_i^j$的目标网络。

MADDPG算法图



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

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

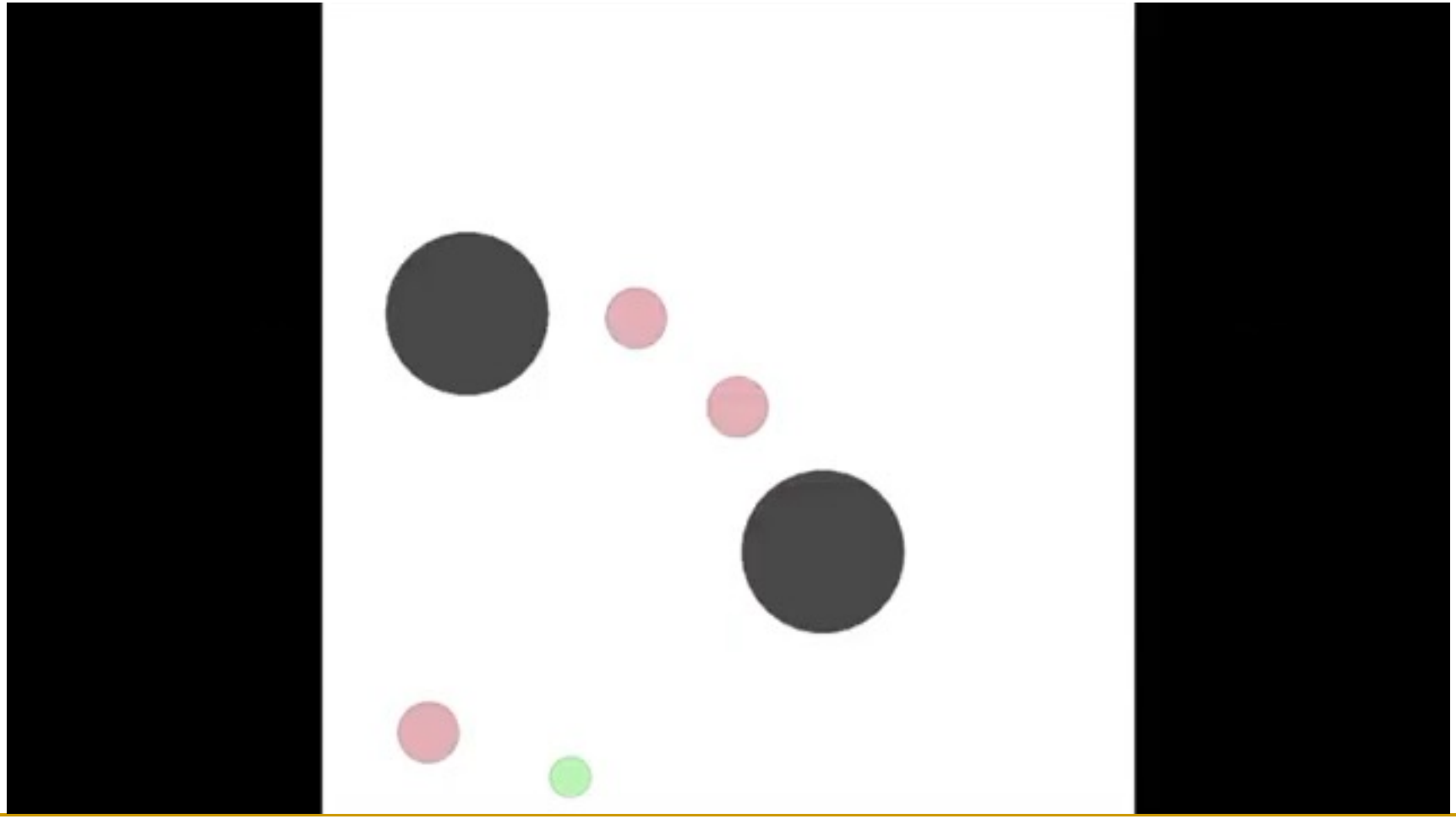
    end for
    Update target network parameters for each agent i:

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

  end for
end for

MADDPG

- 三个慢捕食者追一个快猎物，尽量避免障碍物



课程大纲

- MADDPG
- **COMA**
- LICA
- MAPPO

COMA

- 系统的联合动作空间(joint action space)将会随智能体数量指数性地扩大。因此，直接从这么大的联合动作空间中学习出一个比较好用的联合策略会非常困难。
- 考虑一种分布式策略，让每个智能体根据自己的观测，输出各自的动作，使得该分布式策略对全局性能来说是最优的。尤其在当每个智能体的观测是局部观测并且互相之间的通信受到限制时，这种分布式策略更是必须要考虑的。
- 假设CTDE，智能体执行动作的时候策略是分布式的，但是在学习的过程中，我们还是假设能够获取更多的全局信息。
- CTDE存在多智能体信用分配，COMA提出了一种方法用于学习非中心式的、部分可观测的多智能体协同的控制策略。

COMA

- 使用一个集中式critic网络和分布式的actor网络，在训练的过程中可以获取所有智能体的信息；
- 采用反事实基线（counterfactual baseline）来解决信用分配的问题；为了应对多智能体信用分配的挑战，COMA使用了一个反事实基线将单个智能体的行动边缘化，而其他智能体的行动保持不变。“如果智能体不采用当前的行为，那会不会有更多的收益”；
- 相比于在联合动作空间搜索最优值，Critic网络要能够对反事实基线进行高效的计算。

COMA

- 对于智能体i，用a_i表示其动作，τ_i表示其历史观测-动作序列，a表示所有智能体的联合动作，a_{-i}表示除智能体i之外的联合动作，s表示系统全局状态，则智能体i的反事实基线可由以下计算：

$$D_i = r(s,a) - r(s,(a_{-i},c_i))$$

- 其中D_i为智能体i的独立行为回报，c_i为**基线默认行为**；
- COMA提出使用Critic网络来计算D_i的值，取所有行为的均值作为默认行为的效用值，符合baseline的思路；

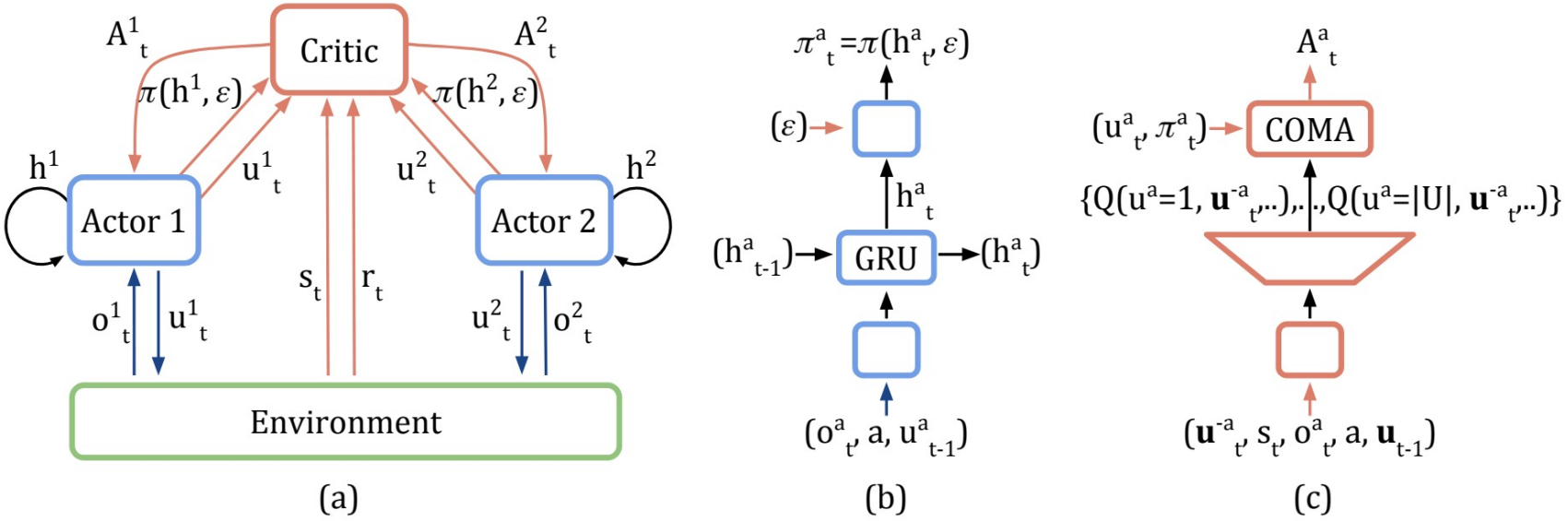
$$Q(s,c_i) = \sum_{a_i'} \pi_i(a_i'|\tau_i) Q(s,(a_{-i},a_i'))$$

$$A_i(s,a) = Q(s,a) - \sum_{a_i'} \pi_i(a_i'|\tau_i) Q(s,(a_{-i},a_i'))$$

- 维度诅咒：A^N计算消耗太大。

COMA

- 为了简化计算，Critic不会计算所有智能体动作组合的Q值（输出维度是A^N个），而是针对每一个智能体，将其他智能体的动作结合其他历史信息当作输入，输出该智能体每个动作对应的Q值（输出维度降为 A ）



COMA

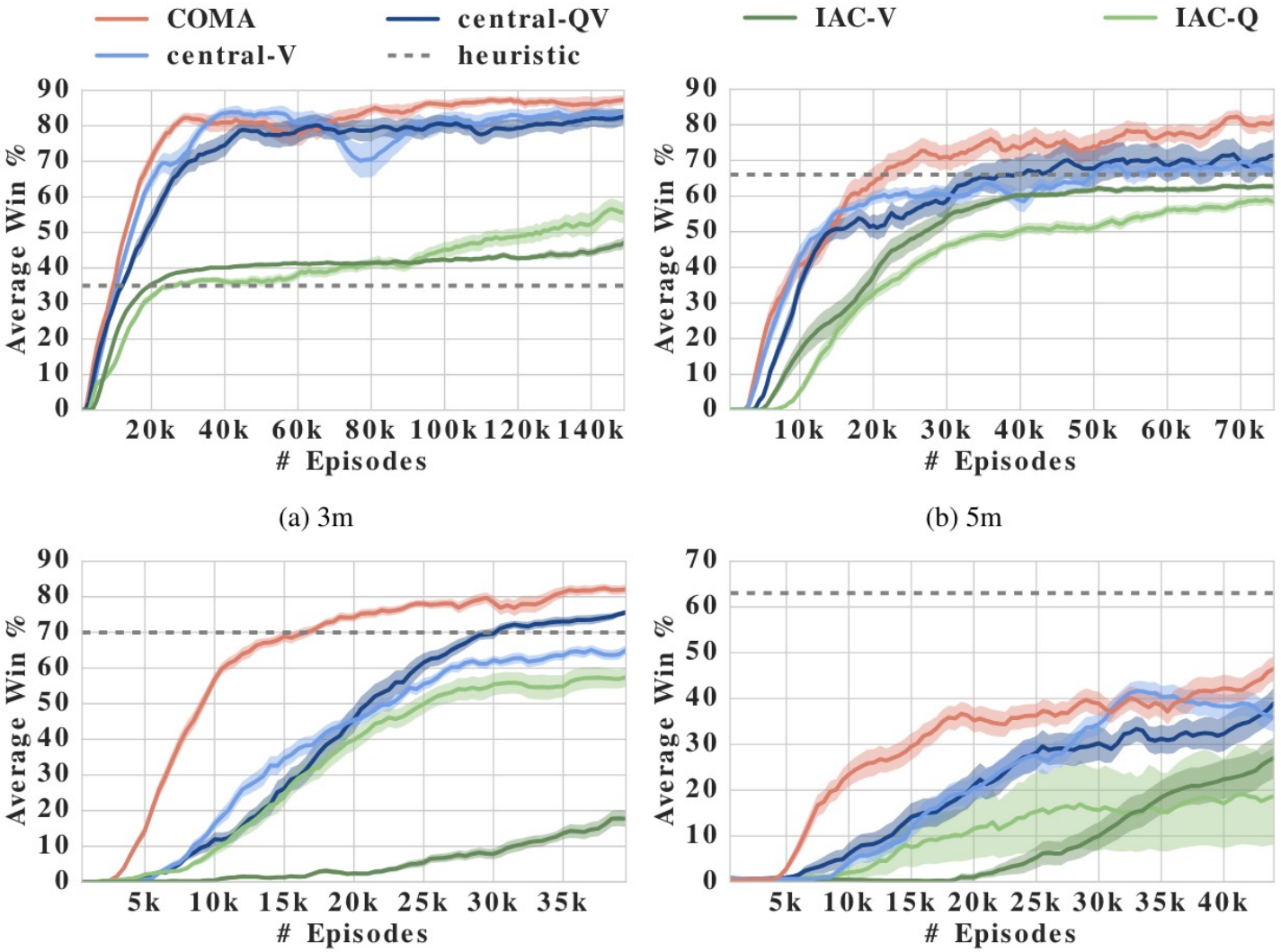
Algorithm 1 Counterfactual Multi-Agent (COMA) Policy Gradients

Initialise θ_1^c, $\hat{\theta}_1^c$, θ^π
for each training episode e **do**
  Empty buffer
  for $e_c = 1$ **to** $\frac{\text{BatchSize}}{n}$ **do**
    $s_1 =$ initial state, $t = 0$, $h_0^a = \mathbf{0}$ for each agent a
    while $s_t \neq$ terminal **and** $t < T$ **do**
      $t = t + 1$
      for each agent a **do**
        $h_t^a = \text{Actor}\left(o_t^a, h_{t-1}^a, u_{t-1}^a, a, u; \theta_i\right)$
        Sample u_t^a from $\pi(h_t^a, \epsilon(e))$
      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(λ) targets y_t^a using $\hat{\theta}_i^c$
  for $t = T$ **down to** 1 **do**
    $\Delta Q_t^a = y_t^a - Q\left(s_j^a, \mathbf{u}\right)$
    $\Delta\theta^c = \nabla_{\theta^c}(\Delta Q_t^a)^2$ // calculate critic gradient
    $\theta_{i+1}^c = \theta_i^c - \alpha\Delta\theta^c$ // update critic weights
    Every C steps reset $\hat{\theta}_i^c = \theta_i^c$
  for $t = T$ **down to** 1 **do**
    $A^a(s_t^a, \mathbf{u}) = Q(s_t^a, \mathbf{u}) - \sum_u Q(s_t^a, u, \mathbf{u}^{-a})\pi(u|h_t^a)$ // calculate COMA
    $\Delta\theta^\pi = \Delta\theta^\pi + \nabla_{\theta^\pi} \log \pi(u|h_t^a) A^a(s_t^a, \mathbf{u})$ // accumulate actor gradients
  $\theta_{i+1}^\pi = \theta_i^\pi + \alpha\Delta\theta^\pi$ // update actor weights

COMA

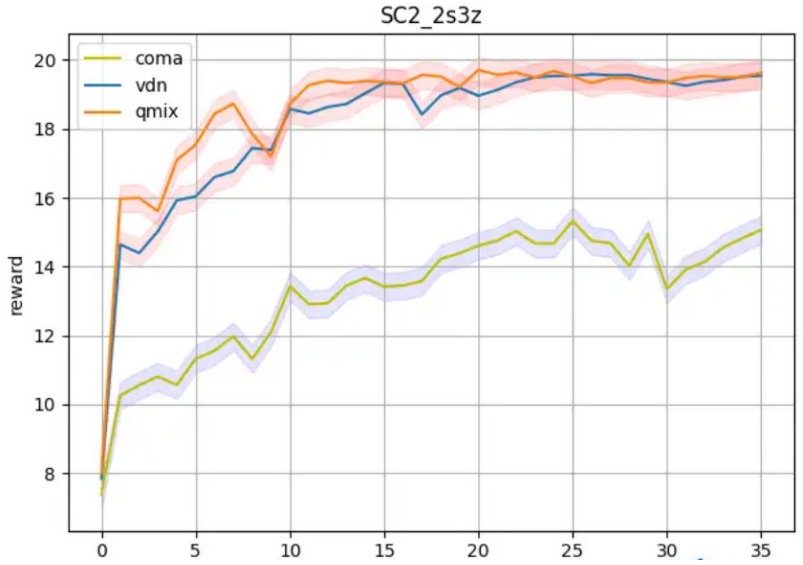
- COMA针对离散动作，学习的是随机策略。而MADDPG针对连续动作，学习的是确定性策略。这在它们策略梯度函数的表达式上能够体现出区别。
- COMA主要针对多智能体协作任务，因此只有一个critic评价团队整体的回报。MADDPG既可以做协作任务，也可以做竞争任务，每个智能体都对应一个奖励函数，因此每个智能体对应一个critic。
- 在原文中，COMA使用了历史观测、动作序列作为网络的输入，而MADDPG没有使用历史序列。因此COMA的网络结构中包含了GRU层，而MADDPG的网络均为2-3个隐层的MLP。
- COMA中所有智能体的actor网络共享参数，输入端加上智能体ID以示区别。而MADDPG则没有共享参数；
- COMA使用反事实基线作为actor网络的优化目标，而MADDPG直接使用Q函数作为每个智能体actor网路的优化目标。

COMA


COMA
central-QV
IAC-V
IAC-Q
central-V
heuristic
Average Win %
Episodes
(a) 3m
(b) 5m
(c) 5w
(d) 2d_3z

COMA

- 采用AC架构，易于并行化，训练起时间比较快；
- 真实表现相对低迷，是AC领域算法的baseline；
- 使用时易于实现，增加实现技巧可以有表现提升。



课程大纲

- MADDPG
- COMA
- **LICA**
- MAPPO

- MARL 中解决信用分配问题的方法大致可分为两类：第一类为显式方法，第二类的方法为隐式方法。
  - 显式方法利用反事实推理、沙普利值、积分梯度等众多其他领域解决信用分配的的方法迁移到 MARL，显式地学习不同智能体的贡献大小；
  - 隐式方法，例如值分解算法，这类算法不需要显式计算不同智能体的信用，而是使用神经网络自动学习，因此在一些无法用反事实推理或沙普利值等方法显式计算各智能体贡献的复杂场景下更优优势。
  - 值分解一般都是应用到基于值的 MARL 算法上，LICA将值分解应用到 AC 算法框架下。

LICA

- LICA值分解算法无需满足IGM约束。之前介绍的值分解算法需要令全局Q函数和局部Q函数满足IGM的约束，LICA将值分解迁移到AC算法框架后，因为Actor是使用来自Critic的梯度更新的，所以基于局部Actor选出来的最优动作也必然是全局Critic的最优动作。
- LICA的Critic使用了超网络，将动作输入到Critic网络，其 MLP 的权重则由基于当前全局状态的超网络确定，此时全局状态和动作实际上属于相乘关系，求动作的偏导时梯度中会包含当前全局状态的信息，从而更准确地计算各智能体具体信用。

$$\frac{\partial Q_\phi^\pi}{\partial u} = \frac{\partial Q_\phi^\pi}{\partial v}\frac{\partial v}{\partial u}, with\ v = \begin{cases} f_s(s) + f_u(u) \\ f_s(s) f_u(u) \end{cases}, and\ \frac{\partial v}{\partial u} = \frac{\partial v}{\partial f_u}\frac{\partial f_u}{\partial u} = \begin{cases} \frac{\partial f_u}{\partial u}, & for\ MLP \\ f_s(s)\frac{\partial f_u}{\partial u}, & for\ MIX \end{cases}$$

LICA

$$\frac{\partial Q_\phi^\pi}{\partial u} = \frac{\partial Q_\phi^\pi}{\partial v}\frac{\partial v}{\partial u}, with\ v = \begin{cases} f_s(s) + f_u(u) \\ f_s(s) f_u(u) \end{cases}, and\ \frac{\partial v}{\partial u} = \frac{\partial v}{\partial f_u}\frac{\partial f_u}{\partial u} = \begin{cases} \frac{\partial f_u}{\partial u}, & for\ MLP \\ f_s(s)\frac{\partial f_u}{\partial u}, & for\ MIX \end{cases}$$

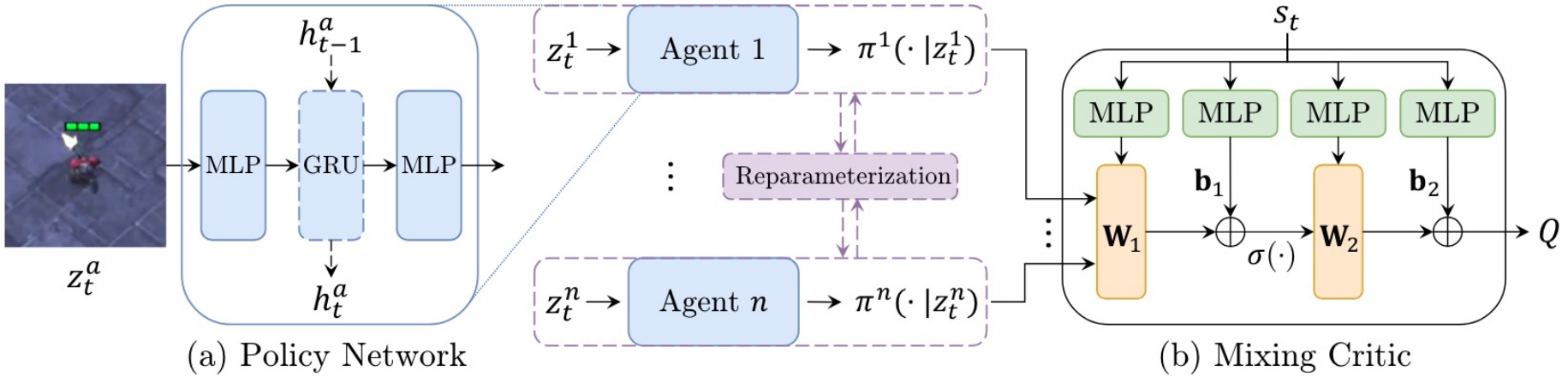


Figure 1: **Overview of LICA**. (a) Architecture for the decentralized policy networks. (b) Architecture for the centralized mixing critic network. Dashed components are situational. Best viewed in color.

LICA-Critic

- 回顾：TD(λ)是一种比较常见的资格迹方法，不同时序差分估计值的权重随着其时间步的增加而衰减

$$G_t^n = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(s_{t+n})$$

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^n$$

- 对于Critic部分，LICA采用了GAE和TD(λ)的变体，其更新为：

$$\mathcal{L}(\phi) = \min_{\phi} \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\left(y_t^{(\lambda)} - Q_\phi^\pi(s_t, u_t) \right)^2 \right]$$

$$y_t^{(\lambda)} = r_t + \gamma(\lambda y_{t+1}^{(\lambda)} + (1-\lambda) Q_\phi^\pi(s_{t+1}, u_{t+1}))$$

LICA-Actor

- LICA 学习的是随机策略而非确定策略。另外，由于基于策略的强化学习算法往往由于探索能力不足而过早地陷入局部最优，因此LICA在更新 Actor 时引入了熵项。

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[Q_\phi^\pi(s_t, u_t) + \mathbb{E}_a \left[\mathcal{H}\left(\pi_{\theta_a}^a(.|z_t^a) \right) \right] \right]$$

$$\frac{\partial J(\theta_a)}{\partial \theta_a} \approx \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\frac{\partial Q_\phi^\pi(s_t, \pi_t^1, \dots \pi_t^n)}{\partial \pi_t^a} \frac{\partial \pi_t^a}{\partial \theta_a} + \frac{\partial \mathcal{H}(\pi_t^a)}{\partial \theta_a} \right]$$

$$\mathcal{H}\left(\pi^a(.|z^a)\right) = \beta H\left(\pi^a(.|z^a)\right) = \beta \mathbb{E}_{u^a \sim \pi^a}[-log\pi^a(u^a|z^a)]$$

LICA-Actor

- 超参数β对智能体探索能力的影响十分敏感，并且β并不能随智能体的训练不断调整，因此论文提出了自适应熵正则化（Adaptive Entropy Regularization）。
- 假设智能体a的某个动作概率为p，对熵项进行求导：

$$p^a = \pi^a(.|z^a)$$

$$d\mathcal{H} = \left[\frac{\partial \mathcal{H}}{\partial p_1^a}, \ldots, \frac{\partial \mathcal{H}}{\partial p_k^a}\right] = [-\beta(logp_1^a + 1), \ldots, -\beta(logp_k^a + 1)]$$

- 当选择某个动作概率较大时，熵项的约束较小；而当选择某个动作概率较小时，熵项的约束反而较大。

$$d\mathcal{H}_i := -\xi . \frac{logp_i^a + 1}{H(p_a)}$$

LICA

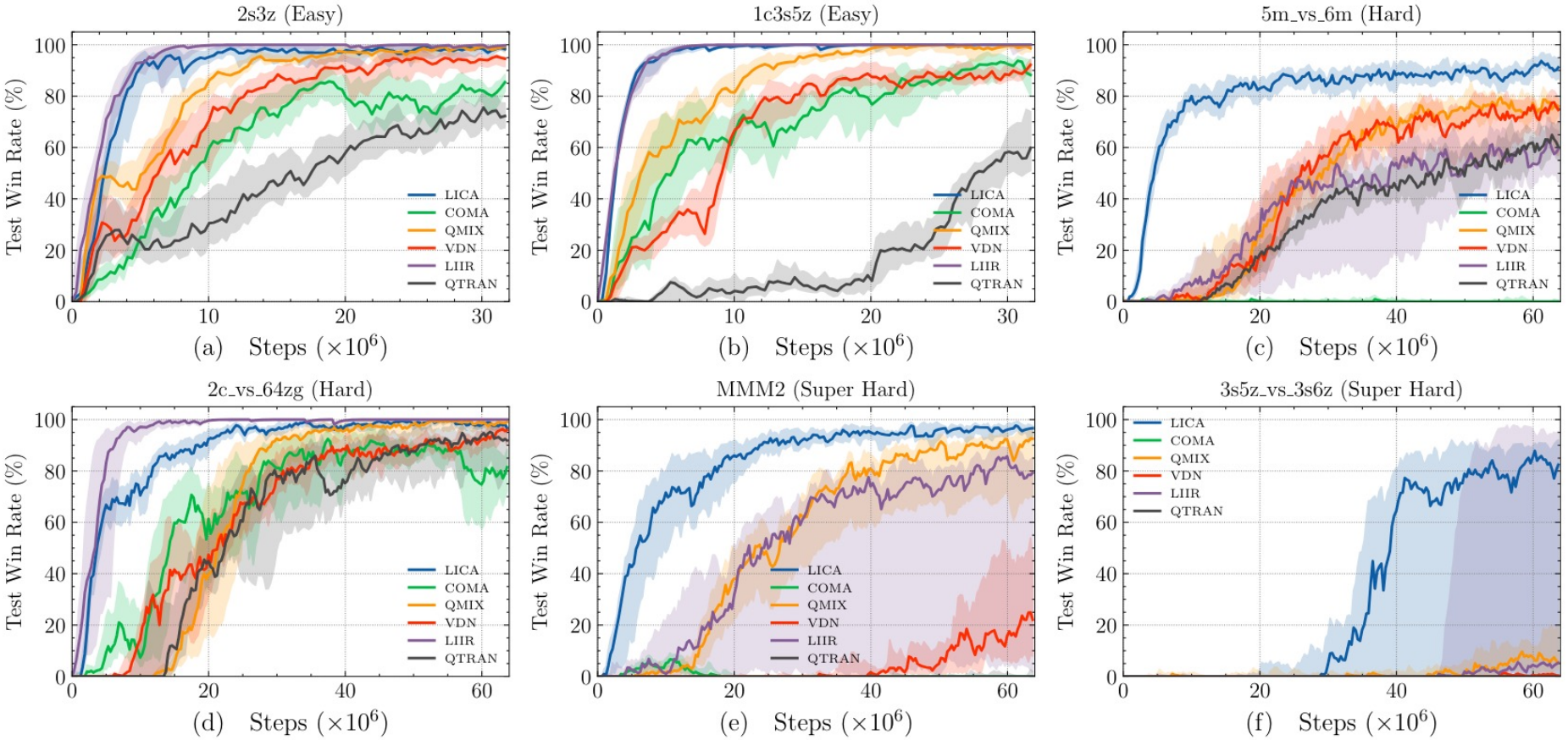


Figure 4: Performance comparison across various scenarios for **StarCraft II micromanagement**.

课程大纲

- MADDPG
- COMA
- LICA
- **MAPPO**

MAPPO

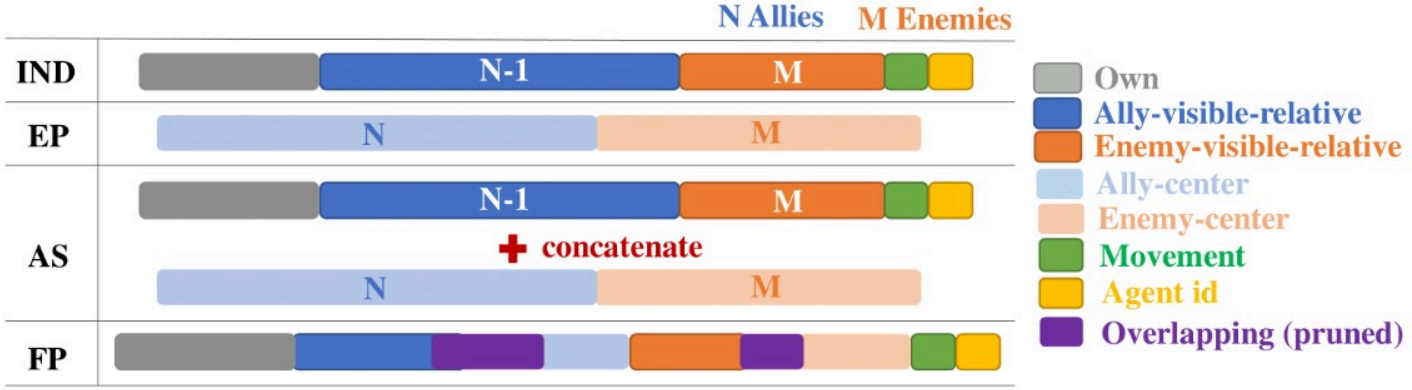
- The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games
  - Surprising
  - Effectiveness
  - PPO
  - Cooperative
- PPO is surprisingly effective in cooperative multi-agent games.

MAPPO-价值归一化

- 在 MAPPO 的训练过程中，由于实际回报的差异，价值目标可能会发生巨大变化，这会导致价值学习的不稳定。
- 通过使用价值目标的平均值和标准差的动态估计来标准化价值函数的目标值。即，在价值学习时，价值网络会以归一化的目标值进行学习。在计算广义优势估计（GAE）时，使用动态平均值来调整价值网络的输出，确保价值输出的正确比例。
- 作者发现使用价值归一化从不会对训练产生负面影响，并且通常能显著提升 MAPPO 的最终性能。

MAPPO-价值函数的输入

- CL：所有本地观测的串联（concatenation of local observations, CL）形成的全局状态。
- EP：采用了一个包含环境状态概况信息的环境提供的全局状态（Environment-Provided global state, EP）
- AS（EP+特定 agent 的观测）：特定智能体的全局状态（Agent-Specific Global State, AS），它通过连接 EP 状态和 o_i 为智能体 i 创建全局状态。这为价值函数提供了对环境状态的更全面描述。
- FP：为了评估这种增加的维度对性能的影响，MAPPO通过移除 AS 状态中重复的特征，创建了一个特征剪枝的特定智能体全局状态（Featured-Pruned Agent-Specific Global State, FP）。



MAPPO-训练数据利用

- PPO 的一个重点特性是使用重要性采样（importance sampling）进行非策略（off-policy）校正，这允许重复使用样本数据。也就是将收集的大量样本分成小批次，并进行多个训练周期（epochs）的训练。
- 在单智能体连续控制领域，常见做法是将大量样本分成大约 32 或 64 个小批次，并进行数十个训练周期。然而，在多智能体领域，我们发现当样本被过度重用时，MAPPO 的性能会降低（也就是不能重复使用次数太多）。
- 这可能是由于多智能体强化学习中的非平稳性（non-stationarity）：减少每次更新的训练周期数可以限制智能体策略的变化，这可能有助于提高策略和价值学习的稳定性。

MAPPO-Clip

- PPO 的另一个核心特征是利用剪切的重要性比例（clipped importance ratio）和价值损失（value loss），以防止策略和价值函数在迭代过程中发生剧烈变化。剪切的强度由超参数ϵ控制：较大的ϵ值允许对策略和价值函数进行更大的更新。
- 与训练周期数类似，假设策略和价值的剪切可以限制由于智能体策略在训练中的变化而引起的非平稳性。对于较小的ϵ，智能体的策略在每次更新中的变化可能会更小，因此可以在可能牺牲学习速度的情况下，提高整体学习的稳定性。

MAPPO-Batchsize

- 在训练更新过程中，PPO 会采样一系列符合策略的轨迹（on-policy trajectories），这些轨迹被用来估计策略和价值函数目标的梯度。由于在我们的训练中小批次的数量是固定的，通常较大的批次会导致更准确的梯度，从而更好地改进价值函数和策略。
- 但是，批次的积累受到可用计算资源和内存的限制：收集大量的轨迹需要有效的并行性，并且这些批次需要存储在 GPU 内存中。因此，使用过大的批次大小可能会在所需的计算资源和样本效率（sample-efficiency）方面造成浪费。
- 为了获得 MAPPO 的最佳任务性能，请使用较大的批次大小。然后，调整批次大小以优化样本效率

MAPPO

Algorithm 1 Recurrent-MAPPO

Initialize θ, the parameters for policy π and ϕ, the parameters for critic V, using Orthogonal initialization (Hu et al., 2020)
Set learning rate α
while $step \leq step_{\max}$ **do**
  set data buffer $D = \{\}$
  for $i = 1$ **to** $batch\_size$ **do**
    $\tau = []$ empty list
    initialize $h_{0,\pi}^{(1)}, \dots h_{0,\pi}^{(n)}$ actor RNN states
    initialize $h_{0,V}^{(1)}, \dots h_{0,V}^{(n)}$ critic RNN states
    for $t = 1$ **to** T **do**
      for all agents a **do**
        $p_t^{(a)}, h_{t,\pi}^{(a)} = \pi(o_t^{(a)}, h_{t-1,\pi}^{(a)}; \theta)$
        $u_t^{(a)} \sim p_t^{(a)}$
        $v_t^{(a)}, h_{t,V}^{(a)} = V(s_t^{(a)}, h_{t-1,V}^{(a)}; \phi)$
      end for
      Execute actions $\boldsymbol{u_t}$, observe $r_t, s_{t+1}, \boldsymbol{o_{t+1}}$
      $\tau += [s_t, \boldsymbol{o_t}, \boldsymbol{h_{t,\pi}}, \boldsymbol{h_{t,V}}, \boldsymbol{u_t}, r_t, s_{t+1}, \boldsymbol{o_{t+1}}]$
    end for
    Compute advantage estimate $\hat{A}$ via GAE on τ, using PopArt
    Compute reward-to-go $\hat{R}$ on τ and normalize with PopArt
    Split trajectory τ into chunks of length L
    for l = 0, 1, ..., T//L **do**
      $D = D \cup (\tau[l : l + T, \hat{A}[l : l + L], \hat{R}[l : l + L])$
    end for
  end for
  for mini-batch $k = 1, \dots, K$ **do**
    $b \leftarrow$ random mini-batch from D with all agent data
    for each data chunk c in the mini-batch b **do**
      update RNN hidden states for π and V from first hidden state in data chunk
    end for
  end for
  Adam update θ on $L(\theta)$ with data b
  Adam update ϕ on $L(\phi)$ with data b
end while

https://arxiv.org/pdf/2103.01955

提问环节

谢 谢