HATRPO & HAPPO

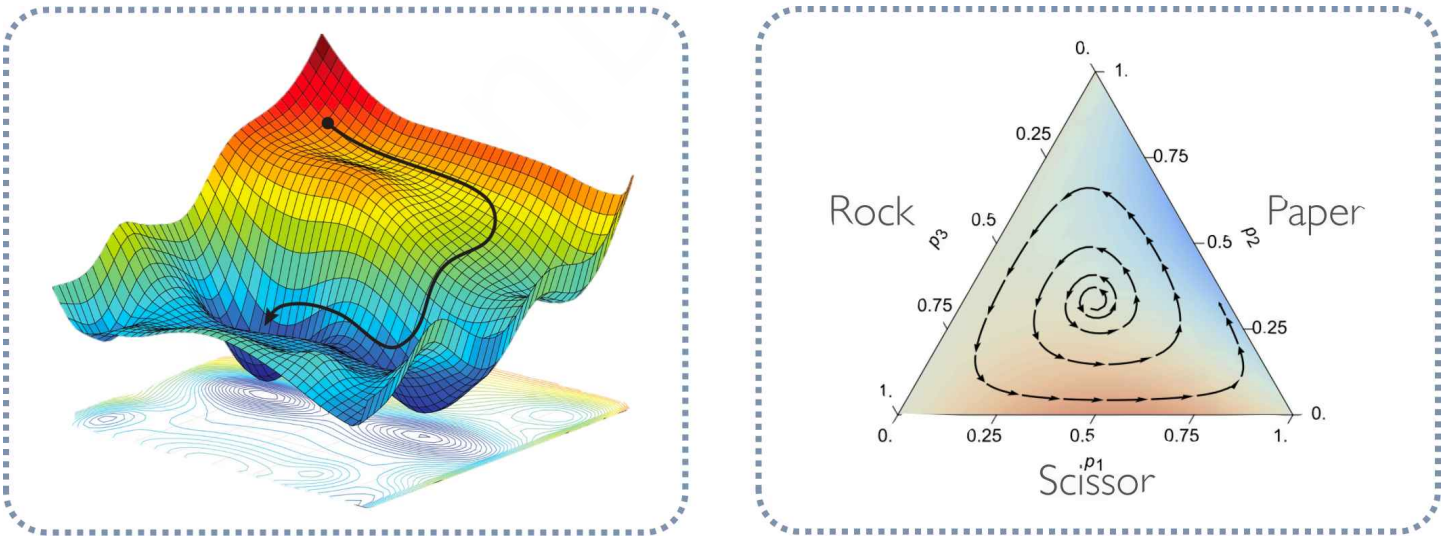
本文旨在介绍多智能体强化学习算法 HATRPO (Heterogeneous-Agent Trust Region Policy Optimisation) 与 HAPPO (Heterogeneous-Agent Proximal Policy Optimisation) [1] 的相关内容。本文涉及的公式符号及其解释可见符号表。

背景：从单智能体到多智能体

多智能体马尔可夫博弈

当决策智能场景中，出现多个自主独立的智能体，此时原本由单智能体的行为就能主导状态转移的简单的马尔可夫决策过程，会变一个多个智能体共同参与，所有行为的集合共同影响状态转移的马尔可夫博弈（Markov Game）[2] 。

一般而言，多个独立的智能体的博弈会导致局面变得更为复杂，因此也会给强化学习算法的训练过程带来额外的困难。如果说单智能体决策问题可以被视为一个仅与环境交互的最优策略搜索问题。而多智能体场景下的策略是否最优，需要同时考虑其它智能体的行为策略，即便是在一个最简单的双智能体的“剪刀石头布”博弈场景中：



(图1：单智能体与多智能体决策中最优策略的区别)

真实世界中，很多涉及到多个角色的复杂问题或复杂应用中都可以被抽象为一个理论中的多智能体马尔可夫博弈，如金融市场中的各类交易对手、交通控制场景中的各类行人和载具、机器人的各个组件或多个机器人的群体、分布式系统中的各个组件、资源分配里的多个竞争实体等等。诸如此类的场景里，一般充满着不确定性，而与单智能体的马尔可夫决策过程不同的是，对于其中一个智能体而言，

如果想要做出完美决策，除了需要应对环境本身自带的不确定性之外，还需要同时应对所有其它智能体的行动带来的状态的变化。而每个智能体在训练过程中更新自身的行为策略会影响整体的局势，从而让最优行为策略的搜索变得复杂。因此，为了解决这些问题，研究将强化学习算法推广并应用到多智能体的场景中是十分必要的。

> 多智能体马尔可夫博弈的符号可以参考课程符号表，在这里使用集合 $\mathcal{N}=\{1,2,3,..,n\}$ 来描述多智能体的全体的集合，集合中的数字为智能体的唯一识别序号，用于区分各个智能体的策略 π 、观测 o 、动作 a 、奖励 r 等等。

多智能体博弈类型

多智能体博弈中，任意两个智能体之间可以是合作博弈或竞争博弈。合作博弈中，智能体的奖励 r_i 往往相同或相近，而在竞争博弈中会相反。与单智能体的决策问题类似，如果奖励的折扣率记为 γ ，可以记目标函数为：

$$J_i(\pi_i)=\mathbb{E}_{s\sim\rho(s),a_{1:n}\sim\pi_{1:n}}(\sum_{t=0}^{\text{end}}\gamma^t r_{i,t})$$

如果多个智能体之间是完全协作的 (fully-cooperative)，一起实现共同的目标，那么它们的目标方程是同一个。

多智能体的策略梯度算法

MAPPO：多智能体策略梯度算法的早期实践

策略梯度 (Policy Gradient) 算法泛指那些使用策略梯度定理来提升智能体策略的强化学习算法，他们是当前强化学习领域的主干算法之一 [3,4,5] 。对于单智能体决策问题，TRPO (Trust Region Policy Optimisation) [4] 和 PPO (Proximal Policy Optimisation) [6] 是其中两种重要的信任域 (Trust-region) 策略梯度算法，在以往的强化学习实践中取得了很好的效果 [7]。

> 信任域方法的机制是通过在原本的目标函数之外添加一个度量函数 (metric function) 的惩罚项，从而将策略迭代的更新幅值限制在一个可信的区间内，它的优势是可以更好地保证策略梯度定理成立需要的条件，即需要更新前后的策略保持近似。而当价值函数的估计相对准确之后，策略梯度定理可以让智能体在策略更新后， π' ，相比策略更新前， π ，得到一个单调提升的稳定回报，即：
>
> $$J(\pi')\geq J(\pi)$$

MAPPO [8] 是 PPO 算法迁移到多智能体环境的一种早期实践，它的实现方法是让多个智能体使用各自独立的模型，并对每个智能体的模型独立使用 PPO 算法进行独立优化和参数更新。其策略优化的目标函数如下：

$$L(\pi_\theta')=\sum_{i=1}^{n}\mathbb{E}_{s\sim\rho_{\pi_\theta},a\sim\pi_\theta}[\min(\frac{\pi_\theta'(a_i|s)}{\pi_\theta(a_i|s)}A_{\pi_\theta}(s,a),\text{clip}(\frac{\pi_\theta'(a_i|s)}{\pi_\theta(a_i|s)},1\pm\epsilon)A_{\pi_\theta}(s,a))]$$

上式中， π_θ 为更新前的策略模型， π_θ' 为更新后的策略模型， ρ_{π_θ} 为更新前的策略对应的状态占用度， a 为所有智能体的动作 $a_{1:n}$ 的缩写， a_i 为第 i 个智能体的动作， $A_{\pi_\theta}(s,a)$ 为更新前的策略对应的优

势函数。

从 MAPPO 的目标函数中可以看到，虽然全局最优策略确实理论上存在于各个智能体共同构成的策略空间之内，即最优解存在，但该优化目标本身并不像原本的单智能体信任域策略优化算法 TRPO 和 PPO 的目标函数那样，可以保证训练过程的单调收敛性。

这是因为多智能体场景中的优势函数，$A_{\pi_\theta}(s,a)$，是所有智能体的策略共同决定的，即 $A_{\pi_\theta}(s,a_1,a_2,...,a_n)$。如果单独修正每个智能体的策略，然后将其它智能体视为环境的一部分，即保持其它智能体的策略不变，那么类似于单智能体 TRPO 算法，或许可以使累计回报相对单调增大；但如果对同一批数据同时修正所有智能体的策略，那么其它智能体将不能被视为环境的一部分，所以上述目标函数不具备策略梯度定理的单调提升特性，从而减弱了在训练中的收敛性和稳定性。

而训练过程的稳定可靠对于获取最优策略是至关重要的，但很可惜在 MAPPO 的实践中，原本属于 PPO 算法的这种特性并没有得到保留。那么，如何在多智能体决策场景下，实现单调的多智能体策略梯度算法呢，接下来将正式介绍 HATRPO 和 HAPPO 算法。

HATRPO & HAPPO： 单调的多智能体策略梯度算法应该如何实现

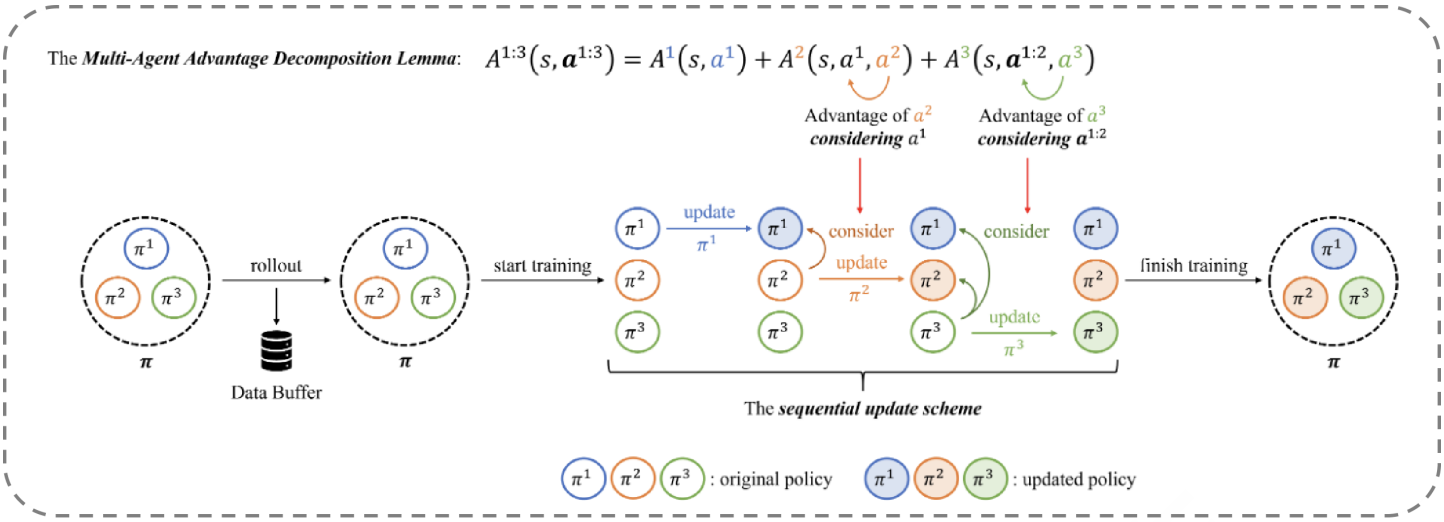
多智能体优势函数分解公式

为了保持多智能体决策场景下的策略梯度算法的单调特性 (Monotonicity)（即保持策略在梯度更新之后的累计回报的期望单调递增的特性），需要重新回顾一下该特性的成立条件，即某个智能体的策略梯度算法在更新参数的时候，需要保持其它智能体的策略不变，即当修改策略 π_1 的时候，$[\pi_2,\pi_3,...,\pi_n]$ 不变，而当修改策略 π_2 的时候，需要 $[\pi_1,\pi_3,...,\pi_n]$ 不变，以此类推。

最终，对于一组训练数据，我们需要使用策略梯度定理更新 $[\pi_1,\pi_2,...,\pi_n]$ 全体策略，那么毫无疑问，在修改第 i 个智能体的策略参数时，已经更新完的参数策略数为 $i-1$ 个，还未更新参数的策略为 $n-i$ 个，无论这些 $n-1$ 个策略的参数是否已经被更新，需要保证它们此时不被更新。因此，对于多智能体场景下的策略梯度算法，需要将原本的更新步骤拆解为 n 步，每一步都需要基于前几步的参数更新的结果进行评估：

$$\begin{aligned}\eta(\tilde{\pi})-\eta(\pi) &\approx \mathbb{E}_{s\sim\rho_\pi,a\sim\tilde{\pi}}[A_\pi(s,a)] \\ &= \mathbb{E}_{s\sim\rho_\pi,a\sim\tilde{\pi}}[\sum_{m=1}^{n}A_\pi^m(s,a_{1:m-1},a_m)]\end{aligned}$$

上式为策略 $\pi=[\pi_1,...,\pi_n]$ 更新为策略 $\tilde{\pi}=[\tilde{\pi}_1,...,\tilde{\pi}_n]$ 后，累计回报 $\eta(\tilde{\pi})$ 相比更新之前 $\eta(\pi)$ 的数值偏离，其数值在策略变化不大的情况下，约等于多智能体策略优势函数在策略更新前数据占用度分布和策略更新后动作分布下的期望。下图是当 $n=3$ 时，三个智能体在训练策略时，其优势函数 $A^{1:3}(s,a)$ 的分解示意：



(图2：多智能体优势函数分解公式的使用方法示意图)

它可以使用多智能体优势函数分解公式 (Multi-Agent Advantage Decomposition) 拆解为 n 个单一动作对应的优势函数的和。单一动作的优势函数 $A_\pi^m(s, a_{1:m-1}, a_m)$，它代表对于策略 π，在前 $m-1$ 个智能体的动作 $a_{1:m-1}$ 的基础上，第 m 个智能体的动作 a_m 的确定，让动作价值函数 $Q_\pi(s, a_{1:m})$ 相比动作价值函数 $Q_\pi(s, a_{1:m-1})$ 的产生的差值。

关于上式的详细证明，可见论文《**Trust Region Policy Optimisation in Multi-Agent Reinforcement Learning**》[1] 附录 C.2，上式的拆解可按照任意排列组合顺序进行，在具体算法的实践中，会采用一组随机生成的序号来确定拆解顺序。需要注意的是，在之后的公式中，智能体的顺序也是一样可以被随机排列和更换的。大家可以参考第六节课作业题，动笔实践，推导和理解这一公式，相信会对掌握 HATRPO 和 HAPPO 的核心方法大有裨益。

多智能体信任域策略梯度算法

在上节的基础上，我们使用信任域方法，约束拆解后的每一步策略梯度的更新幅值。

- HATRPO

与TRPO算法类似，HATRPO使用KL散度作为信任域的约束项，在第 m 个智能体的模型参数更新时，遵循下面的更新公式：

$$\tilde{\theta}^m = \arg\max_{\theta^m} \mathbb{E}_{s\sim\rho_\pi, a_{[1:m-1]}\sim\tilde{\pi}, a_m\sim\tilde{\pi}_{\theta^m}}[A_\pi^m(s, a_{1:m-1}, a_m)]$$

$$\text{subject to } \mathbb{E}_{s\sim\rho_\pi} D_{\mathrm{KL}}(\pi_{\theta^m}, \pi_{\tilde{\theta}^m})$$

> **重要性采样技巧**
>
> 为了在实际计算中更好地适用于蒙特卡洛法使用形式，这里需要对上式做重要性采样的形式变换，将一些集合智能体的优势函数，变换为全体智能体的优势函数，即可以将 $\mathbb{E}_{a_{[1:m-1]}\sim\tilde{\pi}, a_m\sim\pi_{\theta^m}}[A_\pi^m(s, a_{1:m-1}, a_m)]$ 变更为如下的形式：
>
> $$\mathbb{E}_{a_{[1:m-1]}\sim\tilde{\pi}, a_m\sim\tilde{\pi}_{\theta^m}}[A_\pi^m(s, a_{1:m-1}, a_m)] = \mathbb{E}_{a\sim\pi}[(\frac{\tilde{\pi}^m(a_m|s)}{\pi^m(a_m|s)} - 1)(\frac{\tilde{\pi}^{1:m-1}(a_{1:m-1}|s)}{\pi^{1:m-1}(a_{1:m-1}|s)})A_\pi(s, a)]$$

> 证明可详见于论文《**Trust Region Policy Optimisation in Multi-Agent Reinforcement Learning**》[1] 附录 D.1。

- HAPPO

与PPO算法类似，HAPPO使用 clip 函数作为信任域的约束项，从而简化了对KL散度的计算，并使用一个幅值 ϵ 来控制裁剪的程度，在第 m 个智能体的模型参数更新时的目标函数为：

$$\tilde{\theta}^m = \arg\max_{\theta^m} \mathbb{E}_{s\sim\rho_\pi, a\sim\pi}[\min((\frac{\tilde{\pi}^m(a_m|s)}{\pi^m(a_m|s)} - 1)(\frac{\tilde{\pi}^{1:m-1}}{\pi^{1:m-1}})A_\pi, \text{clip}(\frac{\tilde{\pi}^m(a_m|s)}{\pi^m(a_m|s)} - 1, 1 \pm \epsilon)(\frac{\tilde{\pi}^{1:m-1}}{\pi^{1:m-1}})A_\pi$$

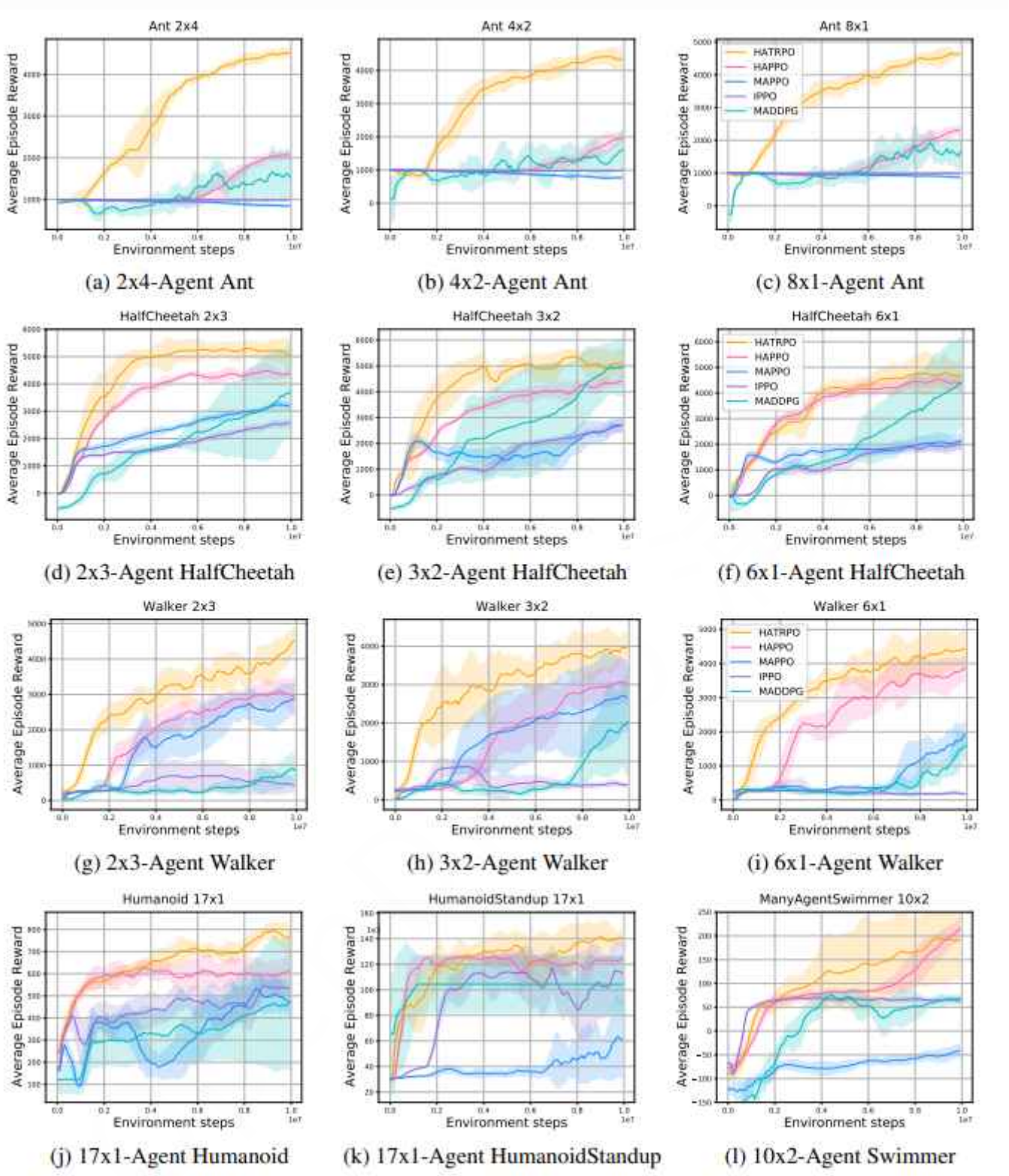
从上述目标函数中，可以看到，无论是HATRPO还是HAPPO算法，其第 m 个智能体的模型参数更新都是建立在前 $m-1$ 次更新的结果基础上进行计算的，这种多智能体模型按次序更新的安排，保证了 HATRPO 和 HAPPO 这两种多智能体策略梯度算法拥有，和单智能体策略梯度算法一样的单调收敛特性。完整的 HATRPO 与 HAPPO 伪代码可见于本文附录。

基线实验：优异稳定的实验性能

Benchmark 实验

在测试 benchmark 环境 Multi-Agent MuJoCo 中，HATRPO 和 HAPPO 的综合表现远远优于 MAPPO/MADDPG [9] 等其它多智能体算法 。

此外，各个任务下的实验结果，都表明HATRPO 算法的效果一般优于 HAPPO 算法，这在一定程度上可能说明，在多智能体环境下，使用 KL 散度控制的信任域，相比于使用 clip 函数控制的信任域，策略梯度算法可以拥有更好的单调提升特性：



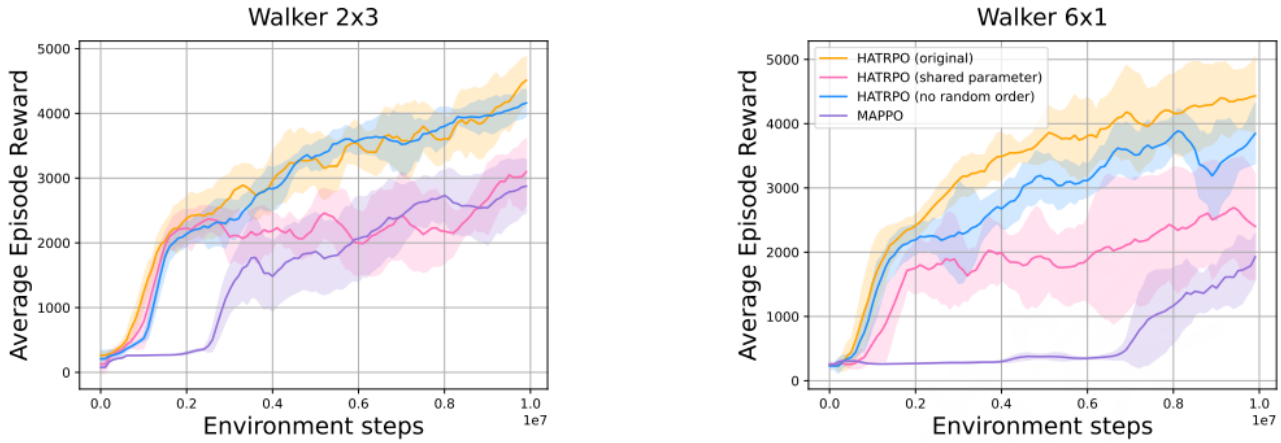
(图3：Multi-Agent MuJoCo 环境各任务 HATRPO / HAPPO / MAPPO / MADDPG / IPPO 算法平均累计回报的表现)

消融实验

在后续的 HATRPO 消融实验里，佐证了多智能体策略模型共享参数的模型设计，在那些需要异质的多智能体相互配合的决策场景中，会对整体性能带来较大的负面影响。

> 异质的智能体 (Heterogeneous-Agent) 代指那些多智能体场景中，具有不同属性和不同表现的智能体。它们在同个状态下，其最优策略各自对应着不同性质的动作。比如 Walker 场景中的左脚与右脚。

此外，为了研究策略更新顺序对算法性能的影响，如下图所示的 MuJoCo Walker 消融实验里，对比了随机生成多智能体策略的优化顺序，和固定多智能体策略优化的顺序，两种情形下的性能表现。值得注意的是，这个实验的结果表明，在 HAPPO 算法中，并不要求多个智能体的策略更新顺序必须使用某种固定顺序，且选用随机顺序的策略更新方式在一些场景下会有更利。



(a) 2-Agent Walker　　(b) 6-Agent Walker

(图4：Multi-Agent MuJoCo 环境 Walker 任务中 HATRPO 共享参数/固定顺序的消融实验)

附录

HATRPO 伪代码

Algorithm 2 HATRPO

Input: Stepsize α, batch size B, number of: agents n, episodes K, steps per episode T, possible steps in line search L, line search acceptance threshold κ.
Initialize: Actor networks $\{\theta_0^i,\ \forall i \in \mathcal{N}\}$, Global V-value network $\{\phi_0\}$, Replay buffer $\mathcal{B}$
for $k = 0, 1, \ldots, K-1$ **do**
  Collect a set of trajectories by running the joint policy $\boldsymbol{\pi}_{\boldsymbol{\theta}_k} = (\pi^1_{\theta^1_k}, \ldots, \pi^n_{\theta^n_k})$.
  Push transitions $\{(o_t^i, a_t^i, o_{t+1}^i, r_t), \forall i \in \mathcal{N}, t \in T\}$ into $\mathcal{B}$.
  Sample a random minibatch of B transitions from $\mathcal{B}$.
  Compute advantage function $\hat{A}(\mathbf{s}, \mathbf{a})$ based on global V-value network with GAE.
  Draw a random permutation of agents $i_{1:n}$.
  Set $M^{i_1}(\mathbf{s}, \mathbf{a}) = \hat{A}(\mathbf{s}, \mathbf{a})$.
  for agent $i_m = i_1, \ldots, i_n$ **do**
    Estimate the gradient of the agent's maximisation objective

$$\hat{\boldsymbol{g}}_k^{i_m} = \frac{1}{B} \sum_{b=1}^{B} \sum_{t=1}^{T} \nabla_{\theta_k^{i_m}} \log \pi_{\theta_k^{i_m}}^{i_m}\left(a_t^{i_m} \mid o_t^{i_m}\right) M^{i_{1:m}}(s_t, \boldsymbol{a}_t).$$

    Use the conjugate gradient algorithm to compute the update direction

$$\boldsymbol{x}_k^{i_m} \approx (\hat{\boldsymbol{H}}_k^{i_m})^{-1} \boldsymbol{g}_k^{i_m},$$

    where $\hat{\boldsymbol{H}}_k^{i_m}$ is the Hessian of the average KL-divergence

$$\frac{1}{BT} \sum_{b=1}^{B} \sum_{t=1}^{T} \mathrm{D}_{\mathrm{KL}}\left(\pi_{\theta_k^{i_m}}^{i_m}(\cdot|o_t^{i_m}), \pi_{\theta^{i_m}}^{i_m}(\cdot|o_t^{i_m})\right).$$

    Estimate the maximal step size allowing for meeting the KL-constraint

$$\hat{\beta}_k^{i_m} \approx \sqrt{\frac{2\delta}{(\hat{\boldsymbol{x}}_k^{i_m})^T \hat{\boldsymbol{H}}_k^{i_m} \hat{\boldsymbol{x}}_k^{i_m}}}.$$

    Update agent i_m's policy by

$$\theta_{k+1}^{i_m} = \theta_k^{i_m} + \alpha^j \hat{\beta}_k^{i_m} \hat{\boldsymbol{x}}_k^{i_m},$$

    where $j \in \{0, 1, \ldots, L\}$ is the smallest such j which improves the sample loss by at least $\kappa \alpha^j \hat{\beta}_k^{i_m} \hat{\boldsymbol{x}}_k^{i_m} \cdot \hat{\boldsymbol{g}}_k^{i_m}$, found by the backtracking line search.
    Compute $M^{i_{1:m+1}}(\mathbf{s}, \mathbf{a}) = \frac{\pi_{\theta_{k+1}^{i_m}}^{i_m}(\mathbf{a}^{i_m}|\mathbf{o}^{i_m})}{\pi_{\theta_k^{i_m}}^{i_m}(\mathbf{a}^{i_m}|\mathbf{o}^{i_m})} M^{i_{1:m}}(\mathbf{s}, \mathbf{a})$. //Unless $m = n$.
  end for
  Update V-value network by following formula:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{BT} \sum_{b=1}^{B} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t\right)^2$$

end for

HAPPO 伪代码

Algorithm 3 HAPPO

Input: Stepsize α, batch size B, number of: agents n, episodes K, steps per episode T.
Initialize: Actor networks $\{\theta_0^i,\ \forall i \in \mathcal{N}\}$, Global V-value network $\{\phi_0\}$, Replay buffer $\mathcal{B}$
for $k = 0, 1, \ldots, K-1$ **do**
  Collect a set of trajectories by running the joint policy $\boldsymbol{\pi}_{\boldsymbol{\theta}_k} = (\pi^1_{\theta^1_k}, \ldots, \pi^n_{\theta^n_k})$.
  Push transitions $\{(o_t^i, a_t^i, o_{t+1}^i, r_t), \forall i \in \mathcal{N}, t \in T\}$ into $\mathcal{B}$.
  Sample a random minibatch of B transitions from $\mathcal{B}$.
  Compute advantage function $\hat{A}(\mathbf{s}, \mathbf{a})$ based on global V-value network with GAE.
  Draw a random permutation of agents $i_{1:n}$.
  Set $M^{i_1}(\mathbf{s}, \mathbf{a}) = \hat{A}(\mathbf{s}, \mathbf{a})$.
  for agent $i_m = i_1, \ldots, i_n$ **do**
    Update actor i^m with $\theta_{k+1}^{i_m}$, the argmax of the PPO-Clip objective

$$\frac{1}{BT} \sum_{b=1}^{B} \sum_{t=0}^{T} \min\left(\frac{\pi_{\theta^{i_m}}^{i_m}\left(a_t^{i_m}|o_t^{i_m}\right)}{\pi_{\theta_k^{i_m}}^{i_m}\left(a_t^{i_m}|o_t^{i_m}\right)} M^{i_{1:m}}(s_t, \boldsymbol{a}_t),\ \mathrm{clip}\left(\frac{\pi_{\theta^{i_m}}^{i_m}\left(a_t^{i_m}|o_t^{i_m}\right)}{\pi_{\theta_k^{i_m}}^{i_m}\left(a_t^{i_m}|o_t^{i_m}\right)}, 1 \pm \epsilon\right) M^{i_{1:m}}(s_t, \boldsymbol{a}_t)\right).$$

    Compute $M^{i_{1:m+1}}(\mathbf{s}, \mathbf{a}) = \frac{\pi_{\theta_{k+1}^{i_m}}^{i_m}(\mathbf{a}^{i_m}|\mathbf{o}^{i_m})}{\pi_{\theta_k^{i_m}}^{i_m}(\mathbf{a}^{i_m}|\mathbf{o}^{i_m})} M^{i_{1:m}}(\mathbf{s}, \mathbf{a})$. //Unless $m = n$.
  end for
  Update V-value network by following formula:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{BT} \sum_{b=1}^{B} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t\right)^2$$

end for

参考文献

[1] Kuba J G, Chen R, Wen M, et al. Trust region policy optimisation in multi-agent reinforcement learning[J]. arXiv preprint arXiv:2109.11251, 2021.

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

[3] Silver D, Lever G, Heess N, et al. Deterministic policy gradient algorithms[C]//International conference on machine learning. Pmlr, 2014: 387-395.

[4] Schulman J, Levine S, Abbeel P, et al. Trust region policy optimization[C]//International conference on machine learning. PMLR, 2015: 1889-1897.

[5] Haarnoja T, Zhou A, Abbeel P, et al. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor[C]//International conference on machine learning. PMLR, 2018: 1861-1870.

[6] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms[J]. arXiv preprint arXiv:1707.06347, 2017.

[7] Berner C, Brockman G, Chan B, et al. Dota 2 with large scale deep reinforcement learning[J]. arXiv preprint arXiv:1912.06680, 2019.

[8] Yu C, Velu A, Vinitsky E, et al. The surprising effectiveness of ppo in cooperative multi-agent games[J]. Advances in Neural Information Processing Systems, 2022, 35: 24611-24624.

[9] Lowe R, Wu Y I, Tamar A, et al. Multi-agent actor-critic for mixed cooperative-competitive environments[J]. Advances in neural information processing systems, 2017, 30.