PPO × Family 第六讲习题作业

本讲习题**共包含两部分：分别是算法理论题和代码实践题**。同学们可以**选择其一项**完成并提交。（**当然也欢迎大家将两部分全部完成**，这样能够加深对课程的理解）

- 算法理论题提交方式：
  发送邮件至 opendilab@pjlab.org.cn

  请同学们严格按照下方格式命名邮箱主题/标题：

  【PPO × Family】+ 学生名 + vol.6 (第几节课）+ 作业提交日期

  示例：【PPO × Family】+ 喵小DI + vol.6 +20230223
- 代码实践题提交方式：

  PPO × Family 官方GitHub 上发起 Pull Request
  - 地址：**PPOxFamily/chapter6_marl/**hw_submission
  - PR示例：https://github.com/opendilab/PPOxFamily/pull/5
  - 命名规范：

    hw_submission(学生名称): add hw6（第几节课）+作业提交日期

    示例：hw_submission(nyz): add hw6_20230104

提交 **截止时间为 2023.5.17 23:59 (GMT +8)**，逾期作业将不会计入证书考量。

如果其他问题请添加官方课程小助手微信（vx: OpenDILab)，备注「课程」，小助手将邀请您加入官方课程微信交流群；或发送邮件至 opendilab@pjlab.org.cn

算法理论题

题目1（HATRPO & HAPPO 推导）

TRPO 算法和 PPO 算法是两种重要的策略梯度强化学习算法。如何将 TRPO 和 PPO 算法延拓到多智能体的场景下，是一件十分值得研究的话题。下面我们将简化部分细节，推导这个拓展的核心内容。

具体来说，对于一个多智能体的协作场景，多个智能体之间可以彼此交流观察信息，了解各自的决策动作，协作完成团队决策，并共享相同的团队奖励。这里，我们可以记当前 n 个智能体的策略为 $\pi_{1:n}$，简写为 π，记当前状态为 s，记当前 n 个智能体的动作分别为 $a_{1:n}$。假如记折扣系数为 γ，每个时刻的奖励为 r_t，于是有此时策略 π 对应的多智能体动作价值函数为：

$$Q_\pi(s,a_{1:n})=\mathbb{E}_{s\sim\rho_\pi,a_{1:n}\sim\pi}[\sum_{t=0}^{\infty}\gamma^t r_t|s_{t=0}=s,a_{t=0}=a_{1:n}]$$

在单智能体环境中，我们可以使用优势函数（Advantage function），即动作价值函数和状态价值函数的差值，来评估，某个具体的动作 a 可以带来的，在当前状态 s 下超越平均动作价值的额外价值，记为 $A(s,a)$。那么在多智能体的场景中，我们应该如何评估某个动作 a_i，或某些动作 $a_{[i_1,i_2,...]}$，可以带来的超越平均价值的额外价值呢？

> 为了实现通用的定义，我们引入一个有序集合 $[i_1,i_2,...,i_p]$，代表多个智能体的其原本序号的集合，这样可以最大程度上避免重复枚举和歧义。比如对于有序集合 $[1,6,8]$，代表序号第 1 位，第 6 位和第 8 位的三个智能体构成的集合，而动作 $a_{[1,6,8]}$ 就是代表序号第 1 位，第 6 位和第 8 位的三个智能体的联合动作变量。
>
> 接下来我们需要评估这些动作的优势函数，比如序号第 1 位，第 6 位和第 8 位的三个智能体的联合动作 $a_{[1,6,8]}$ 相对于平均状态价值的额外价值，可以记为 $A(s,a_{[1,6,8]})$，又比如序号第 1 位，第 6 位和第 8 位的三个智能体的联合动作 $a_{[1,6,8]}$ 相对于序号第 2 位，第 7 位和第 9 位的三个智能体的动作价值函数 $Q(s,a_{[2,7,9]})$ 可以产生的额外价值，可以将其记为 $A(s,a_{[2,7,9]},a_{[1,6,8]})$。

为了完善上述定义，假如我们将多个智能体序号组成的有序集合 $[1,2,...,n]$ 分为两个有序集合 $[i_1,i_2,...,i_p]$ 与 $[j_1,j_2,...,j_q]$，其中 $p+q=n$。那么序号为 $[i_1,i_2,...,i_p]$ 的智能体集合，对应的动作价值函数为：

$$Q_\pi^{i_{1:p}}(s,a_{i_{1:p}})=\mathbb{E}_{a_{j_{1:q}}\sim\pi}[Q_\pi(s,a_{i_{1:p}},a_{j_{1:q}})]=\mathbb{E}_{a_{j_{1:q}}\sim\pi}[Q_\pi(s,a_{1:n})]$$

假如有序集合 $[j_1,j_2,...,j_q]$ 中有一个子集 $[j_1,j_2,...,j_k]$，即 $q\geq k\geq 1$，那么可以定义有序集合 $[j_1,j_2,...,j_k]$ 的智能体相对于有序集合 $[i_1,i_2,...,i_p]$ 的智能体的优势函数为：

$$\begin{aligned}A_\pi^{j_{1:k}}(s,a_{i_{1:p}},a_{j_{1:k}})&=Q_\pi^{i_{1:p},j_{1:k}}(s,a_{i_{1:p}},a_{j_{1:k}})-Q_\pi^{i_{1:p}}(s,a_{i_{1:p}})\\&=Q_\pi^{i_{1:p},j_{1:k}}(s,a_{i_{1:p}},a_{j_{1:k}})-\mathbb{E}_{a_{j_{1:k}}\sim\pi}[Q_\pi^{i_{1:p},j_{1:k}}(s,a_{i_{1:p}},a_{j_{1:k}})]\end{aligned}$$

假如上式中 $[i_1,i_2,...,i_p]$ 集合为空集，那么动作价值函数 $Q_\pi^{i_{1:p}}(s,a_{i_{1:p}})$ 退化为价值函数 $V_\pi(s)$，对应的优势函数的形式退化为：

$$A_{\pi}^{j_{1:k}}(s, a_{j_{1:k}}) = Q_{\pi}^{j_{1:k}}(s, a_{j_{1:k}}) - V_{\pi}(s)$$

1. **请证明多智能体优势函数分解公式 (Multi-Agent Advantage Decomposition)，即对于任意的有序集合 $[i_1, i_2, ..., i_m]$ ，都有如下分解公式成立：**

$$A_{\pi}^{j_{1:k}}(s, a_{j_{1:k}}) = \sum_{m=1}^{k} A_{\pi}^{j_m}(s, a_{j_{1:m-1}}, a_{j_m})$$

（提示：可以根据其定义式，尝试进行拆项）

2. **对于一个三个智能体的环境，已知当前的状态为 s ，智能体的动作为 $[a_1, a_2, a_3]$ ，$Q(s, a_1, a_2, a_3) = 10$ ，$Q(s, a_1, a_2) = 8$ ，$Q(s, a_1) = 9$ ，$V(s) = 5$ ，请尝试根据上面的定义式，计算 $A^{1:3}(s, a_{1:3})$ ，$A^{3}(s, a_{1:2}, a_3)$ ，$A^{2}(s, a_1, a_2)$ ，$A^{1}(s, a_1)$**

3. 类比 TRPO 的设计思路，我们可以用类似的做法，记策略 π 的累计回报为：

$$\eta(\pi) = \mathbb{E}_{s_{t=0}, a_{t=0}, \dots}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

将策略 π 替换为策略 $\tilde{\pi}$ 后，累计回报的偏离为下式，并可以在小变动范围内做一些近似：

$$\eta(\tilde{\pi}) - \eta(\pi) = \mathbb{E}_{s\sim\rho_{\tilde{\pi}}, a\sim\tilde{\pi}}[A_{\pi}(s, a)] \approx \mathbb{E}_{s\sim\rho_{\pi}, a\sim\tilde{\pi}}[A_{\pi}(s, a)]$$

请证明，将策略 π 替换为策略 $\tilde{\pi}$ 后，累计回报的偏离的数值可以被展开为：

$$\begin{aligned}\eta(\tilde{\pi}) - \eta(\pi) &\approx \mathbb{E}_{s\sim\rho_{\pi}, a\sim\tilde{\pi}}[A_{\pi}(s, a)] \\ &= \mathbb{E}_{s\sim\rho_{\pi}, a\sim\tilde{\pi}}[\sum_{m=1}^{n} A_{\pi}^{j_m}(s, a_{j1:m-1}, a_{j_m})]\end{aligned}$$

并请从算法执行层面分析和讨论上式的含义。

（提示：可以尝试使用上文中得到的多智能体优势函数分解公式）

完成问题2的证明后，添加一个 KL 散度作为每次策略梯度更新的幅度限制，那么可以得到多智能体版本的 TRPO 算法的目标函数 $J(\tilde{\pi})$ 的公式，其中 C 为常数：

$$J(\tilde{\pi}) \geq \eta(\pi) + \mathbb{E}_{s\sim\rho_{\pi}, a\sim\tilde{\pi}}[A_{\pi}(s, a)] - CD_{KL}^{\max}(\pi, \tilde{\pi})$$

对KL散度进行拆项，便能得到下述不等式，也就是多智能体版本的TRPO算法的目标函数：

$$\begin{aligned}J(\tilde{\pi}) &\geq \eta(\pi) + \mathbb{E}_{s\sim\rho_{\pi}, a\sim\tilde{\pi}}[A_{\pi}(s, a)] - CD_{KL}^{\max}(\pi, \tilde{\pi}) \\ &\geq \eta(\pi) + \mathbb{E}_{s\sim\rho_{\pi}, a\sim\tilde{\pi}}[\sum_{m=1}^{n} A_{\pi}^{j_m}(s, a_{j1:m-1, a_{j_m}}) - CD_{KL}^{\max}(\pi_{j_m}, \tilde{\pi}_{j_m})]\end{aligned}$$

题目2（值分解系列算法）

为了在多智能体的强化学习训练场景下，进行相对更为准确的价值函数的估计计算，我们需要将所有智能体的状态信息和动作信息都收集起来，并计算动作价值函数，可以进行中心化的训练模式。而在多智能体智能体的部署执行场景中，由于往往存在主观上的技术与博弈规则的限制（比如无法收集敌对智能体的决策信息），或是和客观技术条件的限制（比如通讯速率的限制，通讯数量的约束），单个智能体无法获取其它智能体的相关的信息，只能使用自身的有限信息进行动作价值函数的近似计算，称之为去中心化的部署模式。基于这种客观需求而产生的多智能体强化学习算法的范式就是 CTDE 范式，全称为 Centralised Training with Decentralised Execution [1]。

为了构建单个智能体的价值函数 $Q_i(o_i, a_i)$ 与全局多智能体价值函数 $Q_{tot}(s, a_{1:n})$ 之间的关联，近年来有许多著名的构建算法被提出。其中 o_i 为第 i 个智能体的局部观测，a_i 为第 i 个智能体的动作。

价值函数分解网络算法 VDN（Value-Decomposition Networks [2] ）是其中较为基础的一种，它的具体形式为：

$$Q_{tot}(s, a_{1:n}) = \sum_{i=1}^{n} Q_i(o_i, a_i)$$

即全局多智能体价值函数 $Q_{tot}(s, a_{1:n})$ 是所有单个智能体的价值函数 $Q_i(o_i, a_i)$ 之和。

1. 假如存在一个两个智能体的单步马尔科夫博弈，它是一个合作博弈，因此两个智能体共享相同的奖励数值，其奖励矩阵为：

		Player A	
		$a = 0$	$a = 1$
Player B	$a = 0$	$r = 0$	$r = 2$
	$a = 1$	$r = 2$	$r = 5$

请结合上述例子，分析为什么 VDN 提出的价值函数分解算法，在某些场景下无法将所有单个智能体的价值函数 $Q_i(o_i, a_i)$ 之和表示为全局多智能体价值函数 $Q_{tot}(s, a_{1:n})$ 。

2. QMIX 算法 [3] 是 VDN 算法的一个改进版本，它构建了一个以所有 $Q_i(o_i, a_i)$ 为参数的单调函数来表示 $Q_{tot}(s, a_{1:n})$ ，即模型满足：

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

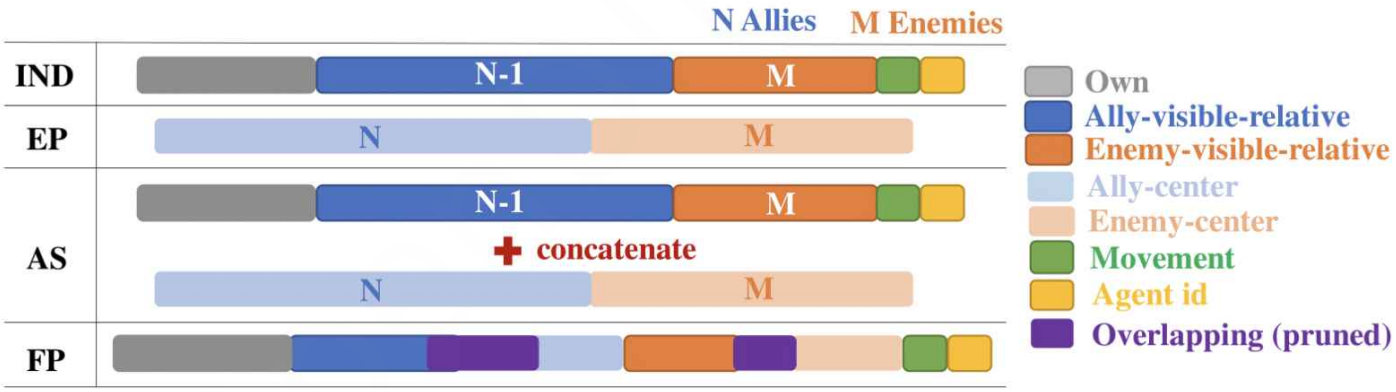
请简述，为什么 VDN 算法可以视为 QMIX 算法的一种特例，并基于问题1中的例子，简单分析为什么 QMIX 网络可以相比 VDN 网络更好地建模复杂的多智能体协作场景。

3. **基于 QMIX 算法的单调性特性，请构建一种两个智能体的单步马尔科夫合作博弈的简单情形，即设计一种对应的奖励矩阵，使得这种情形下， QMIX 算法无法完美的分解价值方程。**

		Player A	
		$a=0$	$a=1$
Player B	$a=0$	$r=?$	$r=?$
	$a=1$	$r=?$	$r=?$

代码实践题

题目1（多智能体协作全局状态信息设计）



（图1：MAPPO 原论文中各种全局状态的对比示意图。IND 指 Independent PPO 所使用的全局状态，仅仅包含单个智能体自身及视野中能获取到的信息。EP 指一些常见的多智能体环境默认提供的全局信息，一般为全局信息统计，相对来说比较简略，且对所有智能体相同。AS 是对于每个智能体特异的全局状态，即 Agent-Specific Global State，可以看作由 IND 和 EP 合成得来。FP 则是对 AS 做了进一步简化，将其中重叠重复的信息删掉，以减少不必要的全局状态信息维度。）

在 MAPPO 算法中，特殊的全局状态信息（Agent-Specific Global State）是重要的设计技巧之一，通过为每个智能体构造特异的全局状态，可以让 PPO 中的 Value Network 为不同的智能体给出不同的价值判断，指引它们更好地分工协作。具体地，在 MAPPO 算法中一共设计比较了上图中所示的4种全局状态信息，本次作业题就需要**根据图示补全下方的代码，完成其中2种（IND，AS）全局状态信息的实现。**

完整代码如下（也可从官网链接下载）：

```python
import numpy as np
import torch


def get_agent_id_feature(agent_id, agent_num):
    agent_id_feature = torch.zeros(agent_num)
    agent_id_feature[agent_id] = 1
    return agent_id_feature


def get_movement_feature():
    # for simplicity, we use random movement feature here
    movement_feature = torch.randint(0, 2, (8, ))
    return movement_feature


def get_own_feature():
    # for simplicity, we use random own feature here
    return torch.randn(10)


def get_ally_visible_feature():
    # this function only return the visible feature of one ally
    # for simplicity, we use random tensor as ally visible feature while zero tensor as ally invisible feature
    if np.random.random() > 0.5:
        ally_visible_feature = torch.randn(4)
    else:
        ally_visible_feature = torch.zeros(4)
    return ally_visible_feature


def get_enemy_visible_feature():
    # this function only return the visible feature of one enemy
    # for simplicity, we use random tensor as enemy visible feature while zero tensor as enemy invisible feature
    if np.random.random() > 0.8:
        enemy_visible_feature = torch.randn(4)
    else:
        enemy_visible_feature = torch.zeros(4)
    return enemy_visible_feature


def get_ind_global_state(agent_id, ally_agent_num, enemy_agent_num):
```

```python
    # You need to implement this function
    raise NotImplementedError


def get_ep_global_state(agent_id, ally_agent_num, enemy_agent_num):
    # In many multi-agent environments such as SMAC, the global state is the simplified version of the combination
    # of all the agent's independent state, and the concrete implementation depends on the characteris of environment.
    # For simplicity, we use random feature here.
    ally_center_feature = torch.randn(8)
    enemy_center_feature = torch.randn(8)
    return torch.cat([ally_center_feature, enemy_center_feature])


def get_as_global_state(agent_id, ally_agent_num, enemy_agent_num):
    # You need to implement this function
    raise NotImplementedError


def test_global_state():
    ally_agent_num = 3
    enemy_agent_num = 5
    # get independent global state, which usually used in decentralized training
    for agent_id in range(ally_agent_num):
        ind_global_state = get_ind_global_state(agent_id, ally_agent_num, enemy_agent_num)
        assert isinstance(ind_global_state, torch.Tensor)
    # get environment provide global state, which is the same for all agents, used in centralized training
    for agent_id in range(ally_agent_num):
        ep_global_state = get_ep_global_state(agent_id, ally_agent_num, enemy_agent_num)
        assert isinstance(ep_global_state, torch.Tensor)
    # get naive agent-specific global state, which is the specific for each agent, used in centralized training
    for agent_id in range(ally_agent_num):
        as_global_state = get_as_global_state(agent_id, ally_agent_num, enemy_agent_num)
        assert isinstance(as_global_state, torch.Tensor)


if __name__ == "__main__":
    test_global_state()
```

题目2（应用实践）

在课程第六讲（统筹多智能体）几个应用中任选一个

- Multi Particle Environment (MPE)（入门级多智能体粒子协作）
- Multi-Agent MuJoCo（进阶级机器人控制协作）

根据课程组给出的示例代码，训练得到相应的智能体。最终提交需要上传相关训练代码、日志截图或最终所得的智能体效果视频（replay），具体样式可以参考第六讲的示例 ISSUE。

参考文献

[1] Kraemer L, Banerjee B. Multi-agent reinforcement learning as a rehearsal for decentralized planning[J]. Neurocomputing, 2016, 190: 82-94.

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

[3] Rashid T, Samvelyan M, De Witt C S, et al. Monotonic value function factorisation for deep multi-agent reinforcement learning[J]. The Journal of Machine Learning Research, 2020, 21(1): 7234-7284.