基于对抗价值分解的弹性多智能体强化学习

Thomy Phan

摘要

我们关注协作多智能体系统中的弹性，在这种系统中，智能体可能由于硬件和软件组件的升级或故障而改变其行为。目前最先进的合作多智能体强化学习 (MARL) 方法要么专注于没有任何变化的理想化设置，要么专注于非常专业的场景，其中变化智能体的数量是固定的，例如，在只有一个生产智能体的极端情况下。因此，我们提出了基于对手比率的弹性对抗价值分解 (RADAR)。RADAR 提供了一个价值分解方案来训练不同规模的竞争团队，以提高对任意代理变化的弹性。我们在两个协作多智能体领域中评估了 RADAR，并表明在任意智能体变化时，RADAR 比最先进的 MARL 取得了更好的最坏情况性能。

关键词：随机对抗训练；弹性价值分解

1 引言

1.1 选题背景

分布式系统由多个独立的组件组成，它们协作完成一个共同的任务[1]。分布式自治系统可以表述为协作多智能体系统 (MAS)，该系统可通过强化学习 (RL) 方法实现[2-3]。分布式自治系统可以表述为协作多智能体系统 (MAS)，该系统可通过强化学习 (RL) 方法实现[2]。

与单代理 RL 相比，多代理 RL 具有更好的可伸缩性和抗变化代理的弹性。我们将代理更改定义为更新或失败。例如，由于维护的原因，一些代理可能会被新软件更新或暂时被其他版本替换。在这两种情况下，我们期望剩下的 MAS 与这些新代理合作。另一方面，代理可能由于硬件或软件故障而行为错误。在这种情况下，我们希望剩余的 MAS 能够优雅地降级，而不会完全失效[4]。直观地说，由于有更多可用的补偿资源，弹性应该随着代理数量的增加而提高[5]。

尽管弹性长期以来被认为是实现合作 MAS 的主要动机[4,6-7]，大多数最先进的合作 MARL 方法都专注于优化理想场景，在这种情况下，代理只面对与训练期间相同或类似的代理[2-3,8]。这有过拟合的风险，当一些代理显著改变其行为时，MAS 可能完全失败，这在安全关键环境中可能是致命的，在这种失败可能会产生灾难性的后果[9]。

基于对抗学习的弹性 MARL 研究[10-12]专注于具有固定数量对手 agent 的专门设置，例如，在一个生产 agent 仍然存在的情况下。当 MAS 的任意部分可能发生变化时，这些方法缺乏所需的灵活性。此外，它们引入了新的可调超参数，如对手的比例或对抗行为的程度，这进一步增加了对意外情况的敏感性。

1.2 相关概念

1.2.1 集中训练分散执行 (CTDE, 也即中心化训练分布式化执行)

对于许多问题，培训通常在实验室或模拟环境中进行，在那里可以获得全局信息。最新的 MARL 利用这一事实来近似集中值函数 $\hat{Q}_i$，它以全局状态 S_t 和联合行动 A_t 为条件，并将它们作为公式 *

$g = A_i^{\pi}(s_t, a_t)\nabla_\theta \log \hat{\pi}_{i,\theta}(a_{t,i}|\tau_{t,i})$ 中的 critic。而 Q_i 只需要在训练中学习当地的策略，$_i$ 本身只需要当地历史 $_t, i$ 为条件，因此它可以以分布式的方式执行。这种范式被称为集中训练和分散执行 (CTDE)。

与 IAC 相反，在整合全局信息时，可以对每个 agent i 分别逼近 Q_i，以学习最佳对策[13]。这种方法, 缺乏训练代理团队的多代理信用分配机制，所有代理都遵守相同的奖励信号。

COMA 近似于每个团队的单一价值函数 $\hat{Q}_i$，用于计算个体信用分配的代理反事实基线 $V_i^{\pi}(s_t) = \sum_{a_{t,i}\in A_i} \hat{\pi}_i(a_{t,i}|\tau_{t,i})\hat{Q}(s_t, < a_{t,i}, a_{t,-i} >)$[2]。

集中的$\hat{Q}_i$ 可以分解成每个 agent i 的单个 Q_i，以便根据公式 1 协调地更新$\hat{\pi}_i$。价值分解网络 (VDN) 是最简单的分解方法，其中 Q 定义为 $\sum_{i\in D} \hat{Q}_i(\tau_{t,i}, a_{t,i})$[14]。另外，也存在非线性因子分解方法，如 QMIX 或 QTRAN[3,15]。

虽然 CTDE 利用状态的马尔可夫性质缓解了独立学习的非平稳性问题，但由于$\hat{Q}(s_t, a_t)$ 需要预定义的输入维数 s_t 和 a_t，大多数基于深度学习的方法需要固定数量的代理 N[2-3,13,15]。

1.2.2 对抗强化学习

在零和游戏中，有 N = 2 个具有相反目标的代理。agent i 和 j 的值函数 (和类似的奖励) 定义为 Q_i^{π} =−Q_j^{π}。agent i 的最小最大平衡策略定义为 π_i^*=argmaxπ_i minπ_j Q_i^*，它对应于最坏情况下的最佳响应，用 π_j^* 表示[10]。

对抗性 RL 方法试图通过对每个代理应用标准 RL 技术来交替优化或重新制定极小极大目标，从而逼近 π_i^*[10-11,16]。

1.3 选题依据

因此本文中，基于当前问题，找到能帮助解决的最新提出方法，加以尝试。文章中提出了 Resilient Adversarial value Decomposition with Antagonist-Ratios（基于对手比率弹性对抗值分解方法）(RADAR)。RADAR 提供了一个价值分解方案来训练不同规模的竞争团队，以提高对任意代理变化的弹性。

1.4 选题意义

在训练过程中训练具有可变团队规模的对抗代理的简单机制，这对于创建能够应对任意代理变化的 MAS 是必要的。与之前在弹性 MARL 上的工作不同，RADAR 没有引入任何新的超参数，因此可以很容易地集成到现有的 RL 框架中。

文章中提出了一种代理测试方案，以公平的方式持续评估在合作的 MAS 中对不断变化的代理的性能和弹性，其灵感来自于先前对单代理 RL 的研究[17-18]

在两个合作的多智能体领域的 RADAR 的经验评价，并与先进的 MARL 的比较提出的测试方案。虽然在合作环境中与最先进的 MARL 竞争，但在测试时面对数量不定的未知对手代理时，RADAR 获得了更好的最坏情况性能。以较为简单，易集成的形式，实现了对抗可变 agent 弹性的提高，正是本文的核心意义所在。

2 相关工作

2.1 对抗性强化学习

对抗学习是一种流行的范式，它交替训练两个对手，以提高彼此的性能和鲁棒性[16,19]。自我 RL 是对抗性 RL 的最简单形式，其中单个代理被训练与自己对抗，以确保足够的难度水平和稳健政策的稳定收敛[20-22]。在 (单代理)RL 中，环境可以被建模为对手，通过添加干扰来对抗最坏情况下的原始代理[16,23-24]。这些对抗性干扰可以通过 RL 或共同进化方法实现[24-25]。

我们的工作主要基于对抗性学习。与单 agent RL(外部变化只能发生在环境内部) 相比，我们关注的是协同 MAS 中的 agent 变化。为此，我们将对手 agent 整合到训练过程中，以提高应变能力。

2.2 多智能体强化学习

MARL 是一个长期存在的人工智能研究领域，有多种方法[2,6,15,26]。虽然合作 MARL 在具有挑战性的领域取得了令人印象深刻的结果，但大多数方法都只在训练中遇到的相同或类似的代理上进行了评估。因此，尚不清楚这些方法是否提供了对任意代理更改的弹性，那些在现实世界中是可预期的。

对于弹性 MARL 已有一些先前的研究:Minimax-Q[10]作为零和游戏 Q-Learning 的一种改编中被提出。在保证对最坏情况下的对手收敛到安全策略的同时，如果对手 j 的 (联合) 行动空间很大，Minimax-Q 将变得难以处理。Li 提出的 M3DDPG[11], 它考虑了极端情况，其中每个 agent i 认为自己是唯一的生产 agent，而所有其他 agent 都被建模为试图最小化 Q_i 的对手。M3DDPG 可能导致糟糕的政策，如果问题太难，甚至无法解决的单一生产代理，导致训练信号不足。Phan[12]提出了 ARTS，其中生产代理和对手代理根据固定的对手比例同时进行训练，因为大多数 CDTE 方法需要一个预定义的输入维度来近似 $Q \approx Q^{\pi}$。通过适当选择对手比率，ARTS 可以提高对抗代理失败的弹性。然而，一个理想的比率需要先验的知道，这是一个不现实的假设。此外，当 N 足够大时，固定的比率可以导致敏感的政策。

我们提出了一种对抗价值分解方案，在训练过程中生产代理和对手代理的数量可以动态变化。此外，我们还提出了一个代理测试方案，以公平的方式评估 MARL 方法的性能和弹性。

3 本文方法

3.1 方法概述

本文首先通过引入的 RAT 方法，对从均匀分布中随机采样的数据进行循环迭代，并且使用两个池 (pro 和 ant) 分别维护正方和对抗方。在迭代中，进行零和博弈，交替轮换正方和对抗方的回合，对应一方的回合从该次迭代中获得机房的经验，从当前迭代的当前放维护池中提取出 Q 值集合，放入对应的 VDN(RADAR 引用的思想) 得到对应不同的结果，并将 π 集合存入策略 π 中。RAT 和 RADAR 的工作结构如图 1所示：

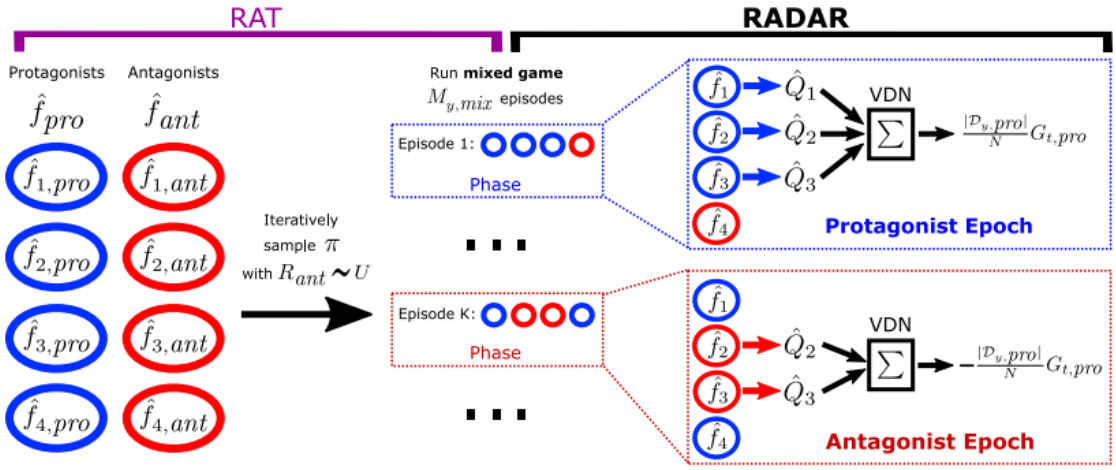

图 1: RADAR 模型图

3.2 随机对抗训练

大多数针对弹性 MARL 的方法都专注于具有固定 R_{ant} 的特定故障场景[10-12]，它有几个缺点: 首先，R_{ant} 必须是先验的或广泛调优的，这通常是不可行的。其次，在面对数量变化的代理时，训练期间固定的 R_{ant} 会导致不灵活的行为，这在现实场景中是可以预期的。第三，R_{ant} 会对训练质量本身产生巨大的影响，例如，如果 R_{ant} 太大，MAS 问题就会变得很难学习任何有意义的政策。

因此，我们考虑使用随机对抗训练 (RAT) 方案。由于在 MAS 中可以发生任意的代理更改，我们为每个代理提供一个主角和对手表示。我们维护一个主角池 $\hat{f}_{pro} =< \hat{f}_{1,pro}, ..., \hat{f}_{N,pro} >$ 和一个对手池 $\hat{f}_{ant} =< \hat{f}_{1,ant}, ..., \hat{f}_{N,ant} >$, 它们和在 T 阶段训练类似[16]。在每个阶段 x，我们从均匀分布 U 中随机抽样 $R_{ant} \in [0, 1)$ ，以运行 Ne 时期不同的混合博弈 $M_{y,mix}$，其中 [(1−R_{ant})N] 主体策略 $\pi_{i,pro}$ 代表 $D_{y,pro}$ 和 [R_{ant}N] 对抗策略 $\pi_{j,ant}$ 代表 $D_{y,ant}$ 伴随 $i \neq j$ 的随机选取。每一次 y 的阶段都可以看作是 $D_{y,pro}$ 和 $D_{y,ant}$ 之间的零和博弈。在每个阶段之后，f_{pro} 或 f_{ant} 会根据生成的经验 $e_{y,t}$= (s_t, z_t, a_t, $r_{t,proi}$) 交替更新，而另一个池则保持固定。

Algorithm 1 Randomized Adversarial Training (RAT)

1: **procedure** $RAT(\mathcal{D}, N, \hat{f}_{pro}, \hat{f}_{ant}, \Psi)$
2: Initialize parameters of $\hat{f}_{pro}$ and $\hat{f}_{ant}$
3: **for** phase $x = 1, T$ **do**
4: Sample $R_{ant} \sim U$ ▷ uniform sampling
5: **for** episode $y = 1, N_e$ **do**
6: $\mathcal{D}_{y,ant} \leftarrow$ sample $\lceil R_{ant} N \rceil$ agents from $\mathcal{D}$
7: $\mathcal{D}_{y,pro} \leftarrow \{i \in \mathcal{D} | i \notin \mathcal{D}_{y,ant}\}$
8: **for** $i = 1, N$ **do** ▷ Create $M_{y,mix}$
9: **if** $i \in \mathcal{D}_{y,ant}$ **then**
10: $\pi_i \leftarrow \hat{\pi}_{i,ant}$ ▷ from $\hat{f}_{i,ant}$
11: **if** $i \in \mathcal{D}_{y,pro}$ **then**
12: $\pi_i \leftarrow \hat{\pi}_{i,pro}$ ▷ from $\hat{f}_{i,pro}$
13: $\pi \leftarrow \langle \pi_1, ..., \pi_N \rangle$
14: Run one $M_{y,mix}$ episode with joint policy π
15: Store $e_{y,t} = \{\langle s_t, z_t, a_t, r_{t,pro} \rangle\}$ and $\mathcal{D}_{y,pro}$
16: **if** $x \bmod 2 = 1$ **then**
17: Update $\hat{f}_{i,pro}$ with Ψ $\forall i \in \mathcal{D}_{y,pro}$ w.r.t. $e_{y,t}$
18: **else**
19: Update $\hat{f}_{i,ant}$ with Ψ $\forall i \in \mathcal{D}_{y,ant}$ w.r.t. $e_{y,t}$

图 2: RAT 算法示意图

RAT(算法 1) 的完整公式在图 2中给出，其中 D 为原 MAS 的 agent 集合 (给定 R_{ant} = 0)，N = |D| 为 agent 数量，f_{pro} 和 f_{ant} 分别为可学习的主角和对手表示，Ψ 为最优化或 MARL 算法。除了 RAT 以外，文章还有一个主要贡献，也就是后续介绍的介绍的 RADAR，而 RAT 是 RADAR 的一个必要的基础和基线。

3.3 弹性对抗价值分解

算法 1 中的 Ψ 可以很容易地设置为 IAC 或其他独立学习算法，因为 RAT 要求 Ψ 在主角和对手的数量方面具有灵活性，每个阶段之间可以根据 R_{ant} 的不同而变化。在 RAT 中使用独立学习会带来非平稳性的问题，并且对 (主角和对手) 代理团队缺乏 credit 分配。大多数 CTDE 方法都需要一个固定的团队规模，因为 $\hat{Q} \approx Q^{\pi}$ 的预定义输入维度取决于 s_t 和 a_t 的联合作用[2,13,27]。

因此，我们提出 RADAR，一个基于 VDN 的具有可变 Rant 的主敌策略近似 CTDE 方案[14]。合作 MAS 的 VDN 用 $\hat{Q}(\tau_t, a_t) = \sum_{i \in D} \hat{Q}_i(s_t, a_t)$ 近似为 $Q^{\pi}(s_t, a_t)$，其中 $\tau = < \tau_{t,1}, ..., \tau_{t,N} >$ 为联合历史。虽然我们关注的是混合对策 $M_{y,mix}$，但显然 Q 只能近似于合作主体。因此，我们分别使用独立的 VDN 实例为主角和对手近似 Q_{pro} 和 Q_{ant}。

给定算法 1 中的 RAT，我们可以用以下关系近似主角迭代 (第 17 行) 的 Q_{pro}:

$$\sum_{i\in D_{y,pro}} \hat{Q}_{i,pro}(\tau_{t,i}, a_{t,i}) = \mathbb{E}_{y,\pi}[\frac{D_{y,pro}}{N} G_{t,pro} | s_t, a_t] \qquad (1)$$

其中 $G_{t,pro}$ 是主角 return，用于规范化 $G_{t,pro}$ 与迭代 y 中参与的主角数量有关，因为 $G_{t,pro}$ 的规模可以赋予 R_{ant} 较小的场景更多的权重。

类似地, 我们可以用以下术语来近似 Q_{ant} 在对抗时期 (算法 1 的第 19 行):

$$\sum_{i\in D_{y,ant}} \hat{Q}_{i,ant}(\tau_{t,i}, a_{t,i}) = \mathbb{E}_{y,\pi}[-\frac{D_{y,pro}}{N} G_{t,pro} | s_t, a_t] \qquad (2)$$

它近似于 Q_{ant} =−Q_{pro}。

Eq. 1 和 2 的项可以通过反向传播对 $\hat{Q}_{i,pro}$ 和 $\hat{Q}_{i,ant}$ 进行端到端训练逼近[14]。$\hat{Q}_{i,pro}$ 和 $\hat{Q}_{i,ant}$ 可以用来推导出相应的本地政策，可以通过应用于值的多臂老虎机问题，也可以根据公式 1 通过策略梯度方法。RADAR 的主要部件及其与 RAT 的集成如图 1所示。

尽管非线性因式分解方法已经被提出[3,15]，VDN 在我们的背景下提供了一些优势: 对于 Eq. 1 和 2,VDN 只是近似一个不受特定代理数量限制的和，因此能够处理 $D_{y,pro}$ 和 $D_{y,ant}$ 的可变团队规模。由于不同于非线性情况下的线性分解，VDN 中针对 R_{ant} 的归一化返回是直接的。此外，VDN 既不引入额外的可学习参数 (例如额外的神经网络) 也不引入超参数，这提高了计算和调优的效率。我们仍然意识到，将非线性因子分解方法应用于 RAT 可能会带来更强大的弹性 MARL 方法，这将推迟到未来的工作。

4 实现细节

4.1 与已有开源代码对比

复现时有部分参考代码但无法运行，无 Readme 可读文件，代码无注释。本人按照文章思路，引用了 VDN 的代码框架，pytorch 的神经网络模型框架，参考作者的神经网络结构设计进行构建，参考了从均匀分布中随机取样的代码实现，并结合了对抗强化学习的思想进行改进，循环迭代进行零和博弈。通过上述改进，实现代码复现。

4.2 实验环境

我们实现了一个具有 N 个 agent 的捕食者-被捕食者 (PP)。

PP[K,N] 包含一个 K×K 网格，其中有 N 个学习捕食者代理和 N/2 个随机移动的被捕食者代理。每个代理从一个随机的位置开始，可以向北、向南、向西、向东移动，或者什么都不做，并且拥有 5 × 5 的视野。

当至少一个捕食者 i 与主捕食者处于同一位置，另一个捕食者 $j \neq i$, 在 i 的视线范围内，记录为 κ =<h_i, j_i> 时，捕获猎物时的全局奖励 +1。捕获的猎物在随机位置重生。

环境使用 gym 库、box2d 和 pygame 实现。创建 python 虚拟环境。使用 conda create -n your_env_name python=X.X（2.7、3.6 等) 命令创建 python 版本为 X.X、名字为 your_env_name 的虚拟环境。your_env_name 文件可以在 Anaconda 安装目录 envs 文件下找到。conda info 或 conda config –show 查看 conda 配置发现 envs directories : $C:\backslash Users\backslash lenov\backslash conda\backslash envs$ 导致安装的环境全跑到 c 盘下面，所以需要修改。conda config –add envs_dirs D:/ProgramData/Anaconda3/envsconda config –remove envs_dirs xxx。

RAT 和 RADAR 的工作结构如图 3所示：

图 3: 环境安装成功的测试 demo

4.3 输入格式

以下对训练和测试的程序输入格式，以表格形式展示。

表 1. 测试集表

可用域	标签	描述
PP[5,2]	PredatorPrey-2	5x5 网格中有 2 个捕食者和 1 个猎物的捕食者猎物域
PP[7,4]	PredatorPrey-4	网格中有 4 个捕食者和 2 个猎物的捕食者猎物域
PP[10,8]	PredatorPrey-8	10x10 网格中有 8 个捕食者和 4 个猎物的捕食者猎物域

输入格式：

python generate_test.py RADAR_X D A

D 为可用域环境

A 为对手比 (adversary ratio)

表 2. 训练集表

Method	Value-based	Standard Actor-Critic	Proximal Policy Optimization
Independent Learning	DQN	IAC*	PPO*
IndependentRAT variant	RAT_DQN	RAT_IAC	RAT_PPO
Linear Value Decomposition	VDN*	RADAR or RADAR_X**	RADAR_PPO
Non Linear Value Decomposition	QMIX*	AC-QMIX	PPO-QMIX

4.4 创新点

文章创新点主要在于：引入了新的测试方法，基于零和博弈对抗学习，从均匀分布中做随机抽样，正方和对抗方轮流在迭代中输入对应的经验，更新对应的 Q 值，重复迭代训练，达到理想的弹性效果。同时，针对已有的 ARTS 和 CTED 加以改进，沿用 CTED 的执行思想，在 ARTS 上引入 VDN 网络分别对正方 (pro) 和对抗方 (ant) 进行价值分解。引入对抗比率概念，并动态改变该值进行训练，使得表现相比于 ARTS 更加具有灵活性。最终，实验证明在最坏情况下表现最优。

5 实验结果分析

对于选中的一种 MARL 方法，我们执行了 20 次 40,000 集的训练运行 (总共 200 万时间步)。在 Ne = 10 阶段的每个时期之后，在 f_pro 上每个 $c \in T$ 运行 50 次，进行完整测试。

经过程程测试用例和训练后，在 200w 次迭代后输出结果，程序结束。执行完成后，将每次迭代结果和一定步长的经验写入 output 文件下。程序结束如图 4所示。

```
    domain statistics: 0.0
-- R = 0.75 | algorithm-RADAR_X_ratio-0.75 | 0.75
PredatorPrey-2 episode Protagonist Test AC-QMIX vs. RADAR_X (0.75) finished:
    discounted return: 0.0
    undiscounted return: 0.0
    domain statistics: 0.0
DONE
```

图 4: 执行完成

同时，迭代期间的对抗数据和结果数据自动写入的给定文件下。

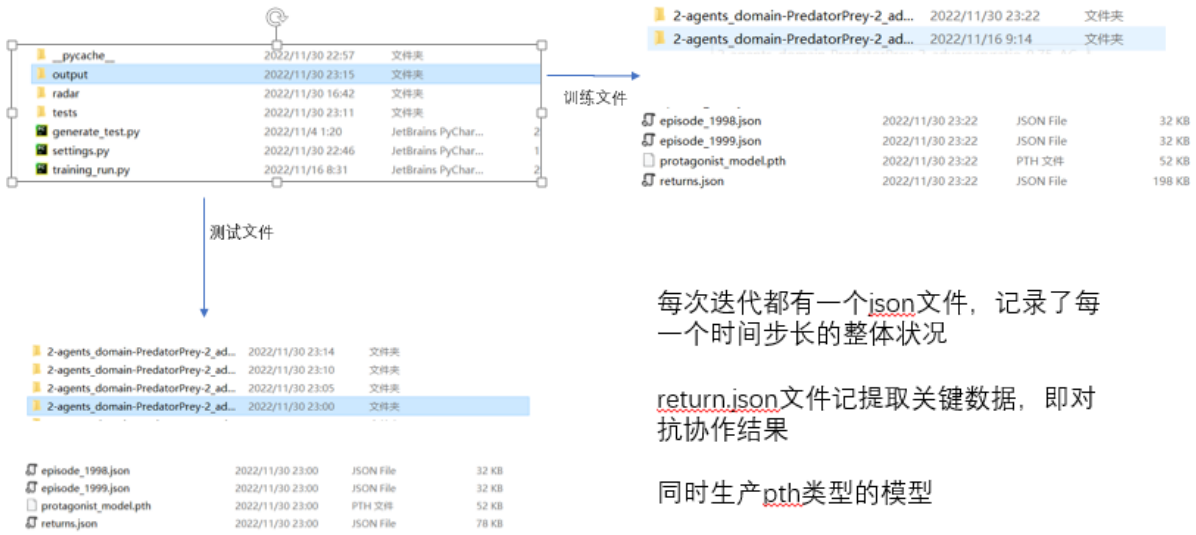

图 5: 导出文件

RADAR、IAC、COMA、AC-QMIX 和 M3DDPG 的最坏情况性能程序结果如图 6所示。所示。除了 PP[7,4]，RADAR 在所有设置下显然都达到了最佳的最差情况性能，在 PP[7,4] 中，COMA、AC-QMIX 和 IAC 具有竞争力。M3DDPG 在所有设置中表现最差。

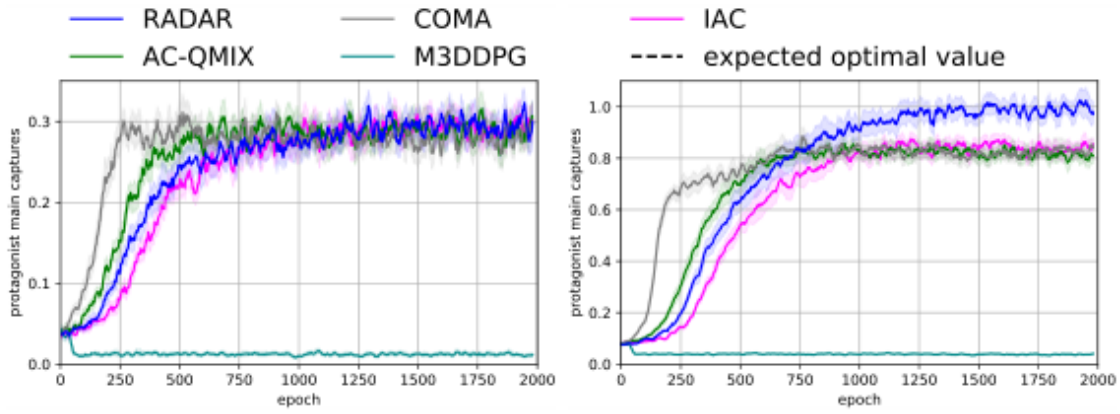

图 6: 最坏情况性能

6 总结与展望

文章引用最新提出的一种弹性 MAS 对抗值分解方案 RADAR。RADAR 训练不同大小的主角和对手代理的竞争团队，提高对任意代理 (agent) 更改的弹性 (适应性)。

与 IAC 比：根据我们的消融实验，价值分解方案比独立学习具有显著的优势:RADAR 和大多数固定的对手比 (antagonist-ratio) 变量明显优于 RAT ($\Psi = \text{IAC}$)，因为 RAT ($\Psi = \text{IAC}$) 缺乏价值分配机制，这对学习协调的主角和对手策略很重要。

与最先进的 MARL 如 COMA、AC-QMIX 和 IAC 相比，RADAR 能够实现具有竞争力的合作性能。我们假设随机集成对抗比率的额外训练会导致一些开销，对于合作测试代理的特殊情况，这会牺牲一些性能。然而，对于包括故障场景在内的任意 agent 更改，RADAR 能够获得优越的最坏情况性能。

虽然 RADAR (N−1/N) 关注的是极端情况，但在所有设置下的表现都很差，这表明如果一个领域对单个主角来说太困难 (或根本无法解决)，这种专门化 (极端情况) 不足以学习弹性行为。尽管 RADAR(N−1/N) 在一开始提高最快 (因为，对抗强化学习训练的特点，对手就做为自己的训练，对手多训练强度大，则主角成长速度更快，但后面对手速度上来，且数量多，足以牵制住单一主角)，对手最终学会了拖住 (牵制)CPPS 中的单一主角，从而导致性能下降。

RADAR 的显著优势是算法的简洁性和团队规模的灵活性。由于它使用统一采样和线性值分解 (VDN 是简单的线性分解)，所以它不引入任何新的超参数来进行调优 (集成的对抗工具有与主角完全相同的超参数)，因此在面对各种对抗设置时，它比先进的 MARL 更不敏感。与其他 CTDE 方法一样，RADAR 在预期和最坏情况下线性伸缩，因此在弹性方面为现有的 RL 方法提供了一种可行且简单的扩展。这也是使用该算法能在最坏情况下保持较好表现的原因。

同时，文章中使用的价值分解网络是 VDN，他是线性的，选择他是因为结构简单，主要为了验证该思想。后期考虑使用非线性价值分解方式，预计表现上会得到进一步提升。

参考文献

[1] TANENBAUM M, Andrew S;Van Steen. Distributed Systems: Principles and Paradigms[J]. Prentice-Hall, 2007: 686.

[2] KUBA J G, WEN M, MENG L, et al. Settling the variance of multi-agent policy gradients[J]. Advances in Neural Information Processing Systems, 2021, 34: 13458-13470.

[3] 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. 2018: 4295-4304.

[4] STONE P, VELOSO M. Multiagent systems: A survey from a machine learning perspective[J]. Autonomous Robots, 2000, 8(3): 345-383.

[5] VAN STEEN M, TANENBAUM A. Distributed systems principles and paradigms[J]. Network, 2002, 2: 28.

[6] PANAIT L, LUKE S. Cooperative multi-agent learning: The state of the art[J]. Autonomous agents and multi-agent systems, 2005, 11(3): 387-434.

[7] BUŞONIU L, BABUŠKA R, SCHUTTER B D. Multi-agent reinforcement learning: An overview[J]. Innovations in multi-agent systems and applications-1, 2010: 183-221.

[8] GUPTA J K, EGOROV M, KOCHENDERFER M. Cooperative multi-agent control using deep reinforcement learning[C]//International conference on autonomous agents and multiagent systems. 2017: 66-83.

[9] UESATO J, KUMAR A, SZEPESVARI C, et al. Rigorous agent evaluation: An adversarial approach to uncover catastrophic failures[J]. arXiv preprint arXiv:1812.01647, 2018.

[10] LITTMAN M L. Markov games as a framework for multi-agent reinforcement learning[G]//Machine learning proceedings 1994. Elsevier, 1994: 157-163.

[11] LI S, WU Y, CUI X, et al. Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient[C]//Proceedings of the AAAI Conference on Artificial Intelligence: vol. 33: 01. 2019: 4213-4220.

[12] PHAN T, GABOR T, SEDLMEIER A, et al. Learning and testing resilience in cooperative multi-agent systems[C]//Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems. 2020: 1055-1063.

[13] 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.

[14] SUNEHAG P, LEVER G, GRUSLYS A, et al. Value-decomposition networks for cooperative multi-agent learning[J]. arXiv preprint arXiv:1706.05296, 2017.

[15] SON K, KIM D, KANG W J, et al. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning[C]//International conference on machine learning. 2019: 5887-5896.

[16] PINTO L, DAVIDSON J, SUKTHANKAR R, et al. Robust adversarial reinforcement learning[C]//International Conference on Machine Learning. 2017: 2817-2826.

[17] BADIA A P, PIOT B, KAPTUROWSKI S, et al. Agent57: Outperforming the atari human benchmark [C]//International Conference on Machine Learning. 2020: 507-517.

[18] JORDAN S, CHANDAK Y, COHEN D, et al. Evaluating the performance of reinforcement learning algorithms[C]//International Conference on Machine Learning. 2020: 4962-4973.

[19] GOODFELLOW I, POUGET-ABADIE J, MIRZA M, et al. Generative adversarial networks[J]. Communications of the ACM, 2020, 63(11): 139-144.

[20] SAMUEL A L. Some stuies on Machine Learning Using the Game of Checkers[J]. IBM Journal on Research and Development, 1959, 3: 210-229.

[21] TESAURO G, et al. Temporal difference learning and TD-Gammon[J]. Communications of the ACM, 1995, 38(3): 58-68.

[22] SILVER D, HUANG A, MADDISON C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. nature, 2016, 529(7587): 484-489.

[23] MORIMOTO J, DOYA K. Robust reinforcement learning[J]. Advances in neural information processing systems, 2001: 1061-1067.

[24] GABOR T, SEDLMEIER A, KIERMEIER M, et al. Scenario co-evolution for reinforcement learning on a grid world smart factory domain[C]//Proceedings of the Genetic and Evolutionary Computation Conference. 2019: 898-906.

[25] WANG R, LEHMAN J, CLUNE J, et al. Poet: open-ended coevolution of environments and their optimized solutions[C]//Proceedings of the Genetic and Evolutionary Computation Conference. 2019: 142-151.

[26] MING T. Multi-Agent Reinforcement Learning: Independent versus Cooperative Agents[C]//Proceedings of the Tenth International Conference on Machine Learning (ICML 1993): 330-337.

[27] 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.