Mastering the Game of 3v3 Snakes with Rule-Enhanced Multi-Agent Reinforcement Learning 论文复现

曾湘宇

摘要

贪吃蛇作为一直以来的风靡全球的经典游戏，在以前有很多关于这个游戏的工作。在这篇 Mastering the Game of 3v3 Snakes with Rule-Enhanced Multi-Agent Reinforcement Learning 的论文中根据更复杂的 3v3 模式的贪吃蛇做了相关研究，并和之前的基于规则的最优 3v3 贪吃蛇算法上做了比较，实验结果证明，这篇论文的方法能够战胜之前的最优算法。其方法表现优于其他 132 个方法，连续 20 多天排名第一。本文是做了对此论文的复现的相关工作。

关键词：3v3Snakes；Reinforcement Learning

1 引言

游戏中的问题往往是现实问题的抽象，能够解决游戏中的问题对解决现实问题有一定参考作用。一直以来，游戏中的 AI 方法对视频动画游戏的效果并不是很好，随着深度强化学习 (DRL) 的发展，越来越多的研究人员投入到 DRL 在游戏中的工作中，并取得了不错的成绩，而在多人竞争性游戏中，也取得了令人鼓舞的进展，例如人工智能在麻将[1]、多人德州扑克[2]和格斗游戏[3]中的表现已经达到人类专业水平。同时更多的注意力也放在了更高性能的游戏中。

复现的这篇文章也致力于构建 3v3 贪吃蛇的程序框架。经典贪吃蛇的基本元素，贪吃蛇需要尽力吃更多的豆子来增加蛇身长度，同时避免碰到身体。3v3 贪吃蛇具备经典贪吃蛇的基本元素，此外还加入了竞争合作的元素，增加了难度。双方各操控三条贪吃蛇，在 10×20 的游戏中，双方不仅要保护自己，同时要防止让敌方贪吃蛇变长。在时间步长到达 200 步的时候，游戏结束并计算双方蛇的总长，长的一方判为胜利。此篇复现的文章希望让 agent 在竞争合作的环境中取得更好的性能。

2 相关工作

在这里将介绍强化学习的基本概念和强化学习在游戏中的应用。

强化学习 (RL) 旨在从智能体与环境的交互中寻找智能体的最优策略。一般研究者采用五元组 (S , A , P , R , γ) 马尔科夫决策过程 (MDP) 对学习过程进行建模，分别表示状态空间、动作空间、奖励函数、转移概率函数和折扣因子。在交互中，在每个时间步 t，智能体观察环境状态 st，然后根据当前策略 π (at | st) 选择一个 at 的动作。之后，环境根据状态转移概率函数 P (st + 1 | st , at) 转移到下一个状态 st + 1，智能体得到一个报酬 rt，整个过程定义为一条轨迹 τ，该轨迹的累积报酬为：

$$R_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k.$$

图 1: 累计报酬

在优化过程中，RL 代理的目标是寻找最优策略以最大化累积回报期望 J (π)：

$$J(\pi) = \mathbb{E}_{s\sim\mathcal{P},a\sim\pi}[\sum_{t=0}^{T}\gamma^t r_t].$$

图 2: 回报期望

为了寻找最优策略，许多 DRL 算法被提出，如信赖域策略优化 (TRPO)[4]、近似策略优化 (PPO)[5]、深度确定性策略优化 (DDPG)[6]、彩虹[7]等。

2.1 基于规则的 Territory Matrix 最优方法

目前在 3v3 贪吃蛇中表现最优的方法是基于规则的方法，该方法通过对原始贪吃蛇场景构造 Territory Matrix 领域矩阵来加入人类先验知识帮助训练 agent。构造领域矩阵的具体过程如下所示：

• 从原始环境中获得蛇的身体和头部的位置表示。

• 由于每个时间步蛇都会移动，故应删除蛇的尾部，同时将和蛇头相邻的网格标记为（蛇的 Index,1）。且标记的网格若是被其他蛇占据，则网格标记为 (1-冲突蛇的数目,1)。

• 重复上述过程，直到 10×20 的游戏环境都被被标记。至此，获得贪吃蛇的攻击 (范围) 矩阵。贪吃蛇的攻击矩阵如图 3 所示：

28	3	6	5	4	3	2	3	4	3	2	1	2	3	12	5	4	3	2	29
27	3	7	4	3	2	1	2	3	2	1	16	15	14	13	4	23	24	25	26
3	2	8	9	10	11	12	1	2	1	2	1	2	1	2	3	22	21	5	4
2	1	11	10	9	2	1	12	13	14	1	2	1	16	15	14	15	20	4	3
3	2	1	7	8	3	2	11	2	1	2	3	2	1	2	13	18	19	5	4
13	12	7	6	5	4	9	10	3	2	3	4	3	2	11	12	17	16	15	14
10	11	6	7	6	5	8	5	4	3	4	5	6	3	10	7	6	7	8	9
9	10	5	6	7	6	7	6	5	4	5	6	7	8	9	6	5	6	7	8
2	3	4	5	6	5	4	5	6	5	6	7	8	7	6	5	4	3	2	1
1	2	3	4	5	4	3	4	5	6	7	8	9	10	11	4	3	2	1	30

图 3: 攻击范围矩阵

得到攻击矩阵后，我们还需要将其编码为最终的领域矩阵，编码公式如图 4 所示：

$$N = \begin{cases} 6*j+i, & i \in [0,5] \\ -6*j+i, & i < 0 \end{cases}.$$

图 4: 领域矩阵编码公式

编码后的领域矩阵如图 5 所示：

168	18	41	35	29	23	17	23	29	21	15	9	15	21	77	34	24	18	12	174
162	19	47	29	23	17	11	17	23	14	9	101	95	89	83	28	138	144	150	156
19	13	53	59	65	71	77	11	17	8	14	9	-11	10	16	22	132	126	31	25
13	7	71	65	59	17	11	77	83	89	8	14	10	98	92	86	92	120	25	19
19	13	7	47	53	23	17	71	14	8	14	20	16	10	16	80	108	114	31	25
78	72	47	41	35	29	59	65	20	14	20	26	22	16	68	74	102	96	90	84
60	66	36	47	41	35	53	30	16	20	26	30	38	22	60	42	36	42	48	54
54	60	30	36	47	41	47	41	30	26	30	38	44	50	56	36	30	36	42	48
12	18	24	30	41	35	29	35	41	30	38	44	50	42	36	30	24	18	12	6
6	12	18	24	35	29	23	29	35	41	47	53	59	65	71	24	18	12	6	180

图 5: 领域矩阵

3 本文方法

接下来，将介绍复现论文的主要内容，其中主要框架，输入特征，奖励设计，训练方式。

3.1 输入特征

3v3 Snakes 中的映射被固定为一个 10 × 20 的矩阵。基于此，我们从环境中推导出一个 12 × 10 × 20 的矩阵作为输入特征。前 11 个通道是 one - hot 特性，其物理含义如下所示：

- [1]：六条蛇的全身位置。
- [2]：目前控制的蛇的头部位置。
- [3]：我们团队中三条蛇的头部位置。
- [4]：三条蛇在对手队伍中的头部位置。
- [5]：五个豆子的位置。
- [6]：当前控制蛇的完整身体位置。
- [7、8]：我们团队中另外两条蛇的完整身体位置。
- [9、10、11]：三条蛇在对手队伍中的完整身体位置

前 11 个通道主要是对原始环境的描述，包括豆子的位置，自身队伍和对手队伍的位置等。第 12 个通道则是利用根据原始环境计算出的领域矩阵，在利用矩阵前进行了归一化处理，整个矩阵除以最大可能值 240。领域矩阵能将人类的先验知识融入到训练中，达到类似状态增强的效果，帮助智能体更好的学习。

3.2 模型框架

文章采用构建了策略网络和价值网络，输入为 3.1 节提到的 12×10×20 的输入特征，输入特征输入到有 8 个残差块构成的网络中，其中每个残差块由两个 3×3 的卷积块构成。经过 8 个残差块后，其结果又两个全连接网络共享，第一个全连接网络输出策略，第二个全连接网络输出为价值，具体结构如图 6 所示。

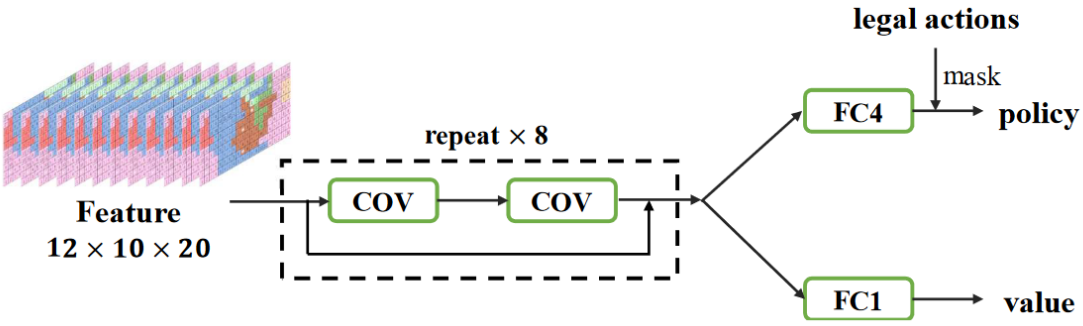

图 6: 模型框架

值得注意的是，在输出最终的策略前，需要进行合法行为的判断，因为对于贪吃蛇而言，任何时候都有一个非法的行为。例如在蛇向右走的时候，我们不能让它下一步立马向左走，对这种情况而言，我们就认为向左是非法行为，我们希望最终输出的策略中采取向左行为的概率是很小的。基于此，可以设置一个很大的值 NAN 来掩盖非法行为，具体如图 7 所示。其中当行为是合法的时候 legal 为 1，反之为 0。

$$P = softmax(PN(s) - (1 - l_{legal}) \times NaN),$$

图 7: 掩盖非法行为

3.3 奖励构造

对于奖励的构造，由于仅依靠环境给予的奖励不能很好的满足团队竞技目标，因此将额外进行奖励的设计。首先我们将每条蛇的奖励定义如图 8 所示，其中 n 表示第 n 步。

$$r_i = \begin{cases} l_n - l_{n-1}, & n < 200 \\ l_n - l_{n-1} + 10, & n = 200 \ and \ win \\ l_n - l_{n-1} - 10, & n = 200 \ and \ lose \end{cases}$$

图 8: 每条蛇的奖励

此外，我们希望在奖励的时候能考虑团队合作和团队竞技的因素，我们参考了博弈论的零和奖励，即奖励总和总是不变，当一方得到的奖励更多，另一方得到的奖励就会变少，希望以此来鼓励竞争，零和奖励被设计为：

$$\hat{r}_i = r_i - \bar{r}',$$

图 9: 零和奖励

其中减数表示对手奖励的平均。

我们在最终的奖励中加入了零和奖励，当然在竞争过程中，我们也希望当前队伍中每条蛇都能考虑队友的情况，于是我们也将奖励加入了合作奖励，并给零和奖励和合作奖励分配了参数，最终奖励表示如图 10 所示：

$$r_i^* = \hat{r}_i \times (1-\alpha) + \tilde{r} \times \alpha,$$

图 10: 最终奖励

3.4 训练方式

复现文章采用的是分布式 PPO 的方法进行训练，有 20 个 Actor 和 1 个 Learner。

Actor：开始的时候，Actor 获得 Learner 更新的参数。每个 Actor 负责和环境做交互并采集数据。Actor 先观察环境，之后将其传入构建的框架中获得相应策略，根据策略执行相应动作，获得下一个环境情况并得到表示是否结束情景的 done 和相应即时奖励。之后将表示轨迹的元组存放到缓冲区中给 Learner 学习并更新参数。

Learner: Learner 从 Actor 交互得到的轨迹中进行采样，将采样的轨迹通过 PPO 的策略价值网络更新方法进行 Loss 计算参数更新，并将更新的参数发送给相应的 Actor

4 复现细节

4.1 与已有开源代码对比

在复现的实现过程中，我参考了开源代码的其中一部分代码，并自己实现了大部分代码

参考部分 (伪代码)：

Procedure 1 预处理输入的轨迹数据

Input: trajectory

Output: pretrained data trajectory

$states, actions, rewards, dones, nstates, extras$ = trajectory

$Calculateapartof Gaedeltas$

$Obtainvalues, neglogp, legalacfromextras$

$nsteps$ = len(*states*)

$advs$ = np.zerosLike(*rewards*)

$lastgaelam$ = 0

for *t in reversed(*range(*nsteps))* **do**

  $nextnonterminal$ = 1.0 − dones[t]

  $advs[t]$ = lastgaelam = mb_deltas[t] + gamma ∗ lam ∗ nextnonterminal ∗ lastgaelam

end

$returns$ = advs + values[: −1]

$data$ = [arrforarrin[states, returns, actions, values[: −1], neglogp, legalac]]

$name$ = [′state′,′ return′,′ action′,′ value′,′ neglogp′,′ $legal_a$ction′]

Return dict(zip(*name,data*))

上面的伪代码主要是对 Actor 交互得到的轨迹进行处理，最后会作为字典形式返回，输入到缓冲区中给 Learner 使用。代码的含义为：首先从输入的 trajectory 中提取出对应的数据，因为做数据处理的时候已经完成的一个轨迹，而在采用 PPO 方式进行更新的时候，需要用到广义优势估计 Gae，所以在这里进行计算，注意这里的计算需要对 values 拼接一个 0 值，因为需要进行交叉运算。计算完毕后把处理好的 Gae 作为结果的一部分用 dict 形式存储并返回给函数调用者。

4.2 实验环境搭建

实验环境采用的是 Ubuntu，机器信息为 RTX2080Ti，cuda10，pytorch 版本不同可能对于一些方法的实现有所改动。

4.3 界面分析与使用说明

本实验无界面操作，使用方法即进入贪吃蛇文件目录，在 start_all.sh 的父目录下执行 bash start_all.sh 即可执行相应程序。

4.4 创新点

将并行 PPO 换位独立 PPO，但未取得较好的效果。

5 实验结果分析

本部分对实验所得结果进行分析，详细对实验内容进行说明，实验结果进行描述并分析。

实验结果如图 11 所示，图描绘的是到游戏结束时 20 个 Actor 中最长的平均蛇长, 每一千轮次求一个平均值。蛇在游戏开始时长度为 3，可以看到经过 2 万轮次的训练后，当游戏结束时，蛇长也仅仅在 4.2 左右。说明在训练过程中应该出现了差错，智能体并未从中学到很有用的信息。可能是在 IPPO 训练过程中梯度传播出错或者参数更新时并未成功更新，这涉及到 20 个 Actor 和 1 个 Learner 之间的交互。

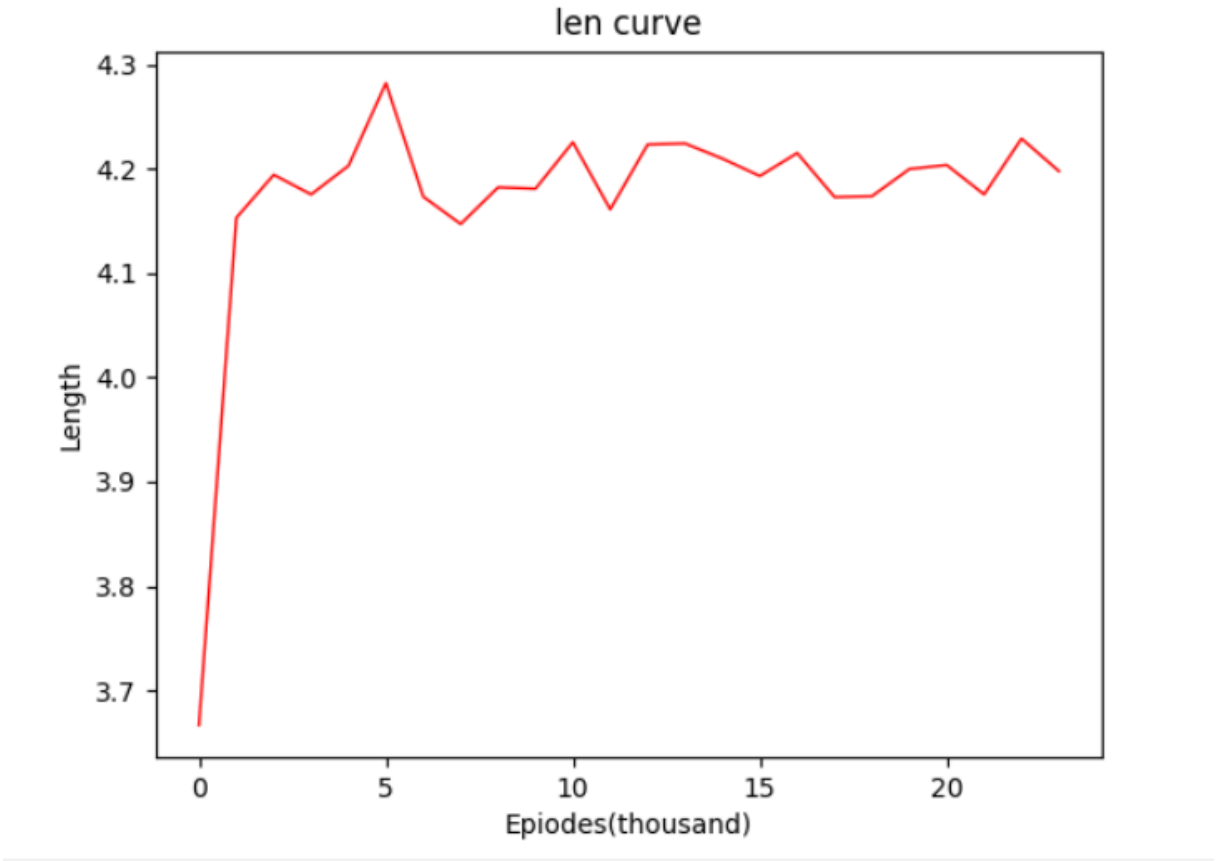

图 11: 实验结果示意

6 总结与展望

原文采用分布式 PPO 算法，而在实现的时候参考了类似独立 PPO(Independent PPO，IPPO) 的方法，独立创建多个 Agent，都分别采用 PPO 算法，但是在更新的时候仅用一个 Learner 进行更新，这在实现过程中可能会出现错误，这可能也是导致最后效果不佳的原因之一。

在实现代码的过程中，有部分论文细节暂未实现，例如在贪吃蛇移动超过边界时，会从另一个对

应边界出来，也就是虽然环境有 10×20 的大小限制，但是是没有边界的。还有数据处理的地方有部分并不到位，主要是一些小技巧没有熟练掌握。在执行 PPO 算法进行训练的时候，计算 Clip 也存在部分问题，都需要完善。

参考文献

[1] LI J, KOYAMADA S, YE Q, et al. Suphx: Mastering Mahjong with Deep Reinforcement Learning[J]. arXiv e-prints, 2020, arXiv:2003.13590: arXiv:2003.13590. arXiv: 2003.13590 [cs.AI].

[2] BROWN N, SANDHOLM T. Superhuman AI for multiplayer poker[J]. Science, 2019, 365: 885-890.

[3] KIM D W, PARK S, YANG S I. Mastering Fighting Game Using Deep Reinforcement Learning With Self-play[C]//2020 IEEE Conference on Games (CoG). 2020: 576-583. DOI: 10.1109/CoG47356.2020.9231639.

[4] SCHULMAN J, LEVINE S, MORITZ P, et al. Trust Region Policy Optimization[J]. arXiv e-prints, 2015, arXiv:1502.05477: arXiv:1502.05477. arXiv: 1502.05477 [cs.LG].

[5] SCHULMAN J, WOLSKI F, DHARIWAL P, et al. Proximal Policy Optimization Algorithms[J]. arXiv e-prints, 2017, arXiv:1707.06347: arXiv:1707.06347. arXiv: 1707.06347 [cs.LG].

[6] LILLICRAP T P, HUNT J J, PRITZEL A, et al. Continuous control with deep reinforcement learning[J]. arXiv e-prints, 2015, arXiv:1509.02971: arXiv:1509.02971. arXiv: 1509.02971 [cs.LG].

[7] HESSEL M, MODAYIL J, VAN HASSELT H, et al. Rainbow: Combining Improvements in Deep Reinforcement Learning[J]. arXiv e-prints, 2017, arXiv:1710.02298: arXiv:1710.02298. arXiv: 1710.02298 [cs.AI].