2025-2026-1 学期强化学习课程 - 心得体会

1120231863　左逸龙

November, 28 2025

1 引言

说实话，在开始这个实验之前，我是一头雾水的。PPO 算法？虽然上课讲过，但是具体怎么工作的，代码该要怎么写，完全不知道。但就是这样一个几乎一无所知的我 (加上半斤八两的队友)，最终却在这个实验里拿到了第二名的成绩——50 胜 8 负，胜率 86.2%。

这个结果让我有点哭笑不得。我们组训练了 110 个小时，而第一名只用了 100 个小时，而且他们什么代码都没改，就是单纯地把腾讯提供的原始 PPO 代码一直训。我们费尽心思修改代码、调整奖励函数，结果反而证明了——**有时候，简单粗暴的方法可能最有效**。

当然，抛开结果不谈，要说我在这次实验中收获了什么，我认为还是不少的，这些在后面我会详细展开。

2 实验概况与结果

2.1 最终成绩

我们组的最终战绩是 **50 胜 8 负**，胜率 **86.2%**，在 30 个小组中排名第二。距离第一名（52 胜 6 负）只差了两场，而且我们和第一名打的两场都输了。如果能打平或赢一场，我们就也能拿第一了。

更让我觉得“幽默”的是，第一名的模型是原始代码训练 100 小时的结果。而我们训练了 110 小时，还做了奖励函数的优化，结果还是打不过。当然据我所知，也有组训练得比我们还久 (点名“国一鲁大”队)，结果表现还不如我们。所以说，训练时间也并非越长越好，整个实验感觉有点“玄学”，最终的评测也挺看运气。

2.2 训练策略

我们的训练分为两个阶段：

1. **原始代码训练 80 小时**：直接使用腾讯开悟平台提供的 PPO Baseline 代码，没有任何修改
2. **强化推塔奖励训练 30 小时**：基于 80 小时的模型，继续训练一个修改了奖励函数的版本

为什么要在第二阶段修改奖励函数？因为我们在评估过程中发现，智能体的推塔积极性比较低，很多时候都是站在角落刷兵线，即使有优势也不愿意去推塔。所以我们决定强化推塔相关的奖励。

具体的修改包括：

- 当英雄处于敌方防御塔下方时，对塔造成伤害的奖励乘以 3
- 当己方小兵在敌方防御塔下时，推塔奖励额外增加 3 倍（鼓励在有小兵掩护时推塔）
- 推塔奖励使用平方根函数，鼓励攻击残血的塔（因为塔残血后一鼓作气推掉就直接赢了，可以考虑牺牲一些血量去强推）

这些修改的思路其实很简单：我们希望智能体学会在有优势的时候推塔，而不是一味地刷兵线和击杀。

3 实验过程与反思

3.1 最初的误解

实验刚开始的时候，我对实验目标的理解是：训练一个智能体，让它学会游戏中的各种操作和技巧。所以我花了很多时间去研究代码，思考如何修改奖励函数、如何改进特征工程等等，让智能体“学得更好”。

但现实给了我一个响亮的耳光。我们发现，即便代码不做任何修改，只要训练时间足够长，智能体就能自行学会这些操作，甚至觉醒了一些我们意想不到的技巧。比如，智能体会借助己方防御塔的掩护来击杀对方，实现反败为胜。这种“自主学习”的能力让我对 PPO 算法有了新的认识，意识到了其强大之处。

更令人沮丧的是，我们尝试了很多代码修改，结果发现大部分都是无效的。这些修改包括但不限于：

- 修改特征向量，将 725 维的后 20 维自定义特征改为野怪相关特征
- 直接修改 `action_process` 函数，覆盖原有策略，用规则限制智能体行为
- 修改奖励函数，强化普攻、鼓励回泉水等等
- ……

这些修改在相同训练时间下，评估时甚至还打不过原始 Baseline。这让我深刻体会到了类似于科研中“踩坑”的痛苦，也意识到了实验检验的重要性。

3.2 策略的转变

随着实验的进行，我们对实验目标的理解也逐渐发生了变化。我们意识到，因为大规模的修改十分困难，因此大部分竞争对手应该都是 Baseline 模型或者 Baseline 的微小修改版本。于是，我们的目标从“让智能体学得更好”变成了“击败我们之前训好的 Baseline 模型”，因为其他组的模型应该差不多。

最终的策略也演变成了：**尽可能多堆训练时长**。因为我们发现，训练时长越长的 Baseline，在面对之前的 Baseline 时，往往有压倒性的优势。

这个转变让我意识到：在比赛环境中，有时候“卷时长”可能比“卷算法”更有效。这当然不是说算法不重要，而是在时间有限、资源有限的情况下，充分利用已有的、已经验证过的方案，可能是更务实的选择。

3.3 唯一有效的修改

虽然大部分修改都没用，但有一个修改确实起到了一定效果：强化推塔奖励。我们在第二阶段训练时使用了这个修改，虽然最终效果并不明显，但至少没有负面效果。这让我意识到，修改奖励函数确实需要非常谨慎，一个小的改动可能会影响整个学习过程。

4 技术细节与收获

4.1 对腾讯开悟框架的理解

虽然我对 PPO 算法的具体实现并不了解（毕竟代码都是腾讯提供的），但我对腾讯开悟框架的整体设计印象深刻。无论是 `gorge_walk`、`back_to_the_realm` 还是 `hok1v1` 实验，都遵循了相同的代码架构，并提供了详细的文档。这种模块化的设计体现了软件工程的最佳实践，让我明白了真正好的代码应该是什么样的。

框架的核心模块包括：

- **特征处理**（`feature/`）：定义数据结构（`ObsData`、`ActData`、`SampleData`）和处理观测数据
- **奖励设计**（`reward_process.py`）：设计多层次奖励函数
- **模型**（`model/`）：神经网络模型，包含 LSTM 处理时序信息
- **算法**（`algorithm/`）：PPO 算法实现，包括采样、更新、优化等
- **工作流**（`workflow/`）：训练循环，协调环境交互、样本收集、模型更新

这种清晰的分层架构让我理解了复杂系统是如何组织和实现的，是软件工程的最佳实践，值得学习。

4.2 代码同步方案

在实验过程中，我遇到了一个实际问题：腾讯开悟平台的 IDE 基于 code-server，功能比较简陋，很多插件都无法下载，部分快捷键与浏览器快捷键冲突，而且无法访问常用的代码托管平台。为了能在我更加熟悉的本地开发环境中写代码，我设计了一套通过 Base64 编码同步代码的方案。

具体做法是：

1. 在本地使用脚本将代码打包并 Base64 编码
2. 在开悟平台的 IDE 中创建编码文件
3. 通过脚本解码并解压到代码目录

详情可以参考我在 Github 上的 README

这个方案虽然看起来有点“原始”，但在实际使用中效果很好，特别是在小组实验中，我们通过 Git 仓库 管理代码，然后每个人手动同步到各自的开发空间，实现了小组协同开发。

同时，由于一个账号只能开一个训练，如果有人一口气训 10 小时 (对 Baseline 而言是家常便饭)，那他接下来即便修改了代码，也没法再训练并检测效果，进而导致了“拖延症”：反正改了也得 10 小时后才能看到效果，那我晚点改也一样。后果就是代码推进缓慢。但是有了这样一种同步方案后，我们就可以方便地将本地代码同步到空闲组员的账号上来训练，从而避免了拖延症。

这个经历让我意识到 “工欲善其事，必先利其器”，有时候掌握好一些工具与技巧，可以做到事半功倍。

4.3 调试技巧的提升

虽然我没有深入阅读代码（主要是让 AI 帮忙读），但在调试过程中，我学会了使用断点调试来截取变量值，从而分析数据结构 (主要还是因为腾讯给的文档这一块不是特别详细)。有时还需要对断点添加条件，来截取特定情况下的变量值。这些调试技巧在实际开发中非常有用。

5 实验反思与启发

5.1 关于“有效”和“无效”的思考

回顾整个实验过程，我认为唯一有效的做法就是：**不停地训练 Baseline 模型**。只要训练时间压其他组的模型一头，我们就能有更大概率战胜他们。而其他的做法——研究代码、修改奖励函数、改进特征工程——大部分都无效，甚至可能有害。

这让我反思：写代码的目的不仅仅是写出正确的代码，还要写出**有效**的代码，在当今 Vibe Coding 盛行的时代，一些强大的 AI 能够一口气生成成百上千行正确且看起来很厉害的代码，但这些代码是否真的有效，是否真的能解决问题，是否真的能带来收益，这些都是值得怀疑的。就我个人的使用经历来说，有的 AI 会生成复杂的错误处理逻辑，但事实上这些情况根本不会发生；有的 AI 会攥写大量的注释，但有的地方明明一眼就能看懂，注释反而显得画蛇添足。很多人其实什么也不懂，但是看着 AI 生成的代码觉得很高大上，就以为自己也会了，便开始沾沾自喜，殊不知这在他人眼中可能只是一些“花架子”，更

会对团队协作造成负面影响。只有真正理解代码与蕴含在其中的逻辑与思想，才能写出真正有效的代码。

因此，即便在 AI 愈发强大的今天，我们仍然需要保持学习，不能盲目依赖 AI，更不能因为 AI 而放弃自我提升。

5.2 对实验设计的思考

我认为腾讯开悟平台提供的代码框架、开发文档、训练平台都设计得非常合理。唯一的“缺点”就是实现得太完美了，导致我们拿到手之后只管训练就完事了，感觉没有太多可以改进的地方，没能很好地实践我们在课堂上所学的强化学习知识。

不过，这也许不是一个缺点，而是一个优点。对于教学实验来说，让学生能够快速上手，把精力集中在理解强化学习的核心概念上，可能比让学生“造轮子”更有价值。

5.3 时间管理的经验

我们的运气算是比较好的，实验一开放就开始训练 Baseline 模型，因此最终能够训练 110 小时。而有些组等到实验快结束、代码看懂了、算法改完了再想开始训练，肯定来不及。毕竟能训 100 小时的前提就是要跑满 100 小时，怎么的也得马不停蹄地跑个四五天。而实验本身就一两周时间，前一周还要做峡谷漫步实验，后一周还要做重返秘境实验，总体上时间非常紧张。

这也让我意识到时间管理的重要性。在实验开始时就要有全局规划，知道哪些任务需要长时间运行，哪些任务可以并行进行。

5.4 如果有更多时间与资源

本次实验只开放了两个周，且各种资源都十分有限，使得我们没法施展开手脚。但如果有更多时间和资源，我可能会考虑更多的改进，比如：

- **课程学习**：分阶段训练，先学会基本操作，再学习高级策略
- **更精细的奖励设计**：添加更多奖励项，平衡不同目标
- **特征工程**：添加更有意义的特征
- **后置规则**：在特定条件下规范化智能体的行为

但这些改进都需要大量的实验验证，在时间有限的情况下，可能还是“堆时长”更保险。

5.5 我的两个启发

这个实验给我最大的启发是：**有时候，最简单的方法可能最有效**。

我们花费了大量时间去研究代码、思考如何修改，结果发现最有效的策略就是“什么都不改，只训练”。这让我想起了奥卡姆剃刀原理——在多个可能的解释中，应该选择最简单的那个。

当然，这并不意味着算法改进不重要。在科研环境中，在工业应用中，算法改进往往是突破的关键。但在比赛环境中，在时间有限、资源有限的情况下，充分利用已有的、验证过的方案，可能是更务实的选择。

另一个启发是：**要尽早开始**。训练模型需要时间，而时间是有限的。如果等到“准备好”了再开始，可能已经来不及了。实验一开放就立刻开始训练，可能是最好的策略。

6 对未来的建议

对于将来要做这个实验的同学，我的建议很简单：

1. **实验一开放就立刻开始训练 Baseline**。不要等到代码看懂了、算法改完了再开始，时间不够。
2. **改代码不如堆时长**。除非你非常有把握，否则不要轻易修改代码。原始代码已经足够好了。
3. **充分利用评估功能**。每训练一段时间就评估一下，根据胜率判断模型是否有进步。
4. **做好代码同步**。如果有条件，可以在本地开发，然后同步到平台。这会大大提高开发效率。
5. **不要过分追求完美**。有时候，80% 的效果用 20% 的努力就能达到，剩下的 20% 可能需要 80% 的努力。

7 结语

这个实验让我对强化学习有了初步的认识，虽然我对 PPO 算法的具体实现仍然不太了解，但至少我知道了它是什么，以及它是怎么运行的。更重要的是，这个实验让我学会了如何在有限的时间和资源下，做出最有效的决策。

虽然我们没有拿到第一名，但我对这个结果已经很满意了。毕竟，在这样一个“玄学”的实验中，能够稳定地拿到第二名，已经说明我们的策略是有效的。

最后，我想说：有时候，承认“我不知道”、“我不会”并不是什么丢人的事情。重要的是，在面对这样的情况下，我们仍然能够找到解决问题的方法，即使这个方法看起来“简单粗暴”。

写完这篇心得体会，我突然意识到，也许这就是这个实验最大的价值——不是让我们成为强化学习专家，而是让我们学会了如何在不确定的环境中，做出最有效的决策。而这，可能是比任何算法知识都更重要的能力。