强化微调完全指南

陈星强

亿铸智能
中国杭州

2025 年 4 月 25 日

内容概述

使用 DeepSeek-R1 和强化学习重塑 AI

强化学习简介

自我提升范式

- 强化学习引入了一种基于反馈驱动的 AI 训练机制
- 相比依赖标记样本，强化学习代理通过以下方式学习：
  - 探索：模型尝试多种策略或行动
  - 奖励：每个行动产生指导未来选择的奖励信号
- 更接近人类通过尝试、错误和反馈自然学习的方式

强化学习简介

自我提升范式

- 为持续学习和更深层的推理能力打开了大门
- AI 不再是静态的实践—模型可以在部署后进化和适应

从静态到动态学习

- 传统方法依赖大量标记数据集进行记忆
- 强化学习将焦点转移到学习策略和推理模式
- “环境”可以是任何提供反馈信号指导改进的场景

DeepSeek-R1 的重要性

DeepSeek-R1：一个为推理而设计的模型

- 自适应奖励结构：多重奖励函数聚焦于准确性、效率性和创造性
- 迭代精化：基于奖励的反馈循环强调实践中最有效的方法
- 突破性能障碍：持续学习使其能够超越传统大语言模型
- 开源优势：与 OpenAI 闭源的 O1 不同，DeepSeek-R1 共享模型权重和训练方法

DeepSeek-R1: 技术实现

主动探索机制

- 使用主动探索而非被动学习
- 采用基于奖励的反馈循环
- 平衡探索与利用

推理模型民主化

- 开源设计便于自定义
- 发布模型权重保证透明度
- 社区驱动的创新

DeepSeek-R1 与传统模型对比

关键差异：

- 传统模型：静态训练，固定参数
- DeepSeek-R1：动态学习方法

DeepSeek-R1 与传统模型对比 (继续)

DeepSeek-R1 优势:

- 动态学习，响应需求变化
- 通过奖励学习，减少对标记数据的依赖
- 通过持续学习降低停滞风险

关键见解:

- “一次性训练完成”模式正在成为过去式

强化微调与监督微调对比

什么是强化微调 (RFT)?

将微调与强化学习相结合

- RFT 结合了预训练大语言模型与基于反馈的强化学习的优势
- 核心过程:
  1. 从具有通用知识的预训练模型开始
  2. 为目标指标定义奖励函数
  3. 使用强化学习技术进行迭代微调
- 使用最少的数据实现开源大语言模型的自定义
- 将通用模型转变为针对特定任务的强大推理模型

RFT 与 SSFT 对比

因素	RFT	SFT
数据需求	最少的标记数据	需要 1,000+ 行数据
适应性	持续改进	受标记数据限制
探索能力	主动尝试新策略	依赖固定示例
性能	持续进步	达到平台期
错误处理	从错误中学习	重复数据中的错误
训练复杂度	较高（奖励函数）	较低（仅需示例）

Table: RFT 和 SFT 方法的比较

RFT 的优势场景

RFT 在数据稀缺时表现出色

- 无需标记数据，依靠客观正确性
- 在小型数据集（几十个示例）上优于 SFT
- 通过学习稳健策略抵抗过拟合
- 最佳应用场景：
  - 代码转译（如 Java 到 Python）
  - 游戏策略（国际象棋，Wordle）
  - 医疗诊断（在决策点从反馈中学习）

定量性能比较

按数据集大小比较 RFT 与 SFT

- 10 个示例：RFT 使基础模型提升 18%，SFT 仅显示最小增益
- 50 个示例：RFT 相比基准线显示 42% 的提升
- 100 个示例：RFT 提升跃升至 60%，比 SFT 好 3 倍
- 数据集越小，RFT 相比 SFT 的优势越明显

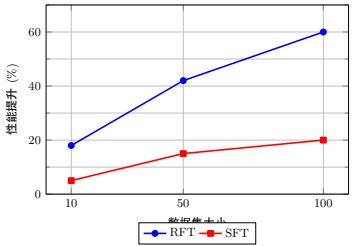


Figure: 性能提升与数据集大小的关系

何时使用 RFT 与 SFT

决策因素:

- 数据可用性: 数据有限时选择 RFT；有大量标记数据时选择 SFT
- 任务复杂性: 对于有明确成功标准的任务，选择 RFT
- 性能目标: 需要持续改进时选择 RFT；需要稳定结果时选择 SFT
- 可验证性: 当结果可以客观衡量时，RFT 表现更佳
- 资源限制: SFT 初期实施更简单，但需要更多数据

实际应用:

- 编程助手: RFT 训练模型编写可编译并通过测试的代码
- 数据分析: RFT 改进生成准确结果的查询能力
- 推理任务: RFT 增强逐步解决问题的能力

使用 Turbo LoRA 加速推理模型

推理模型的挑战

处理量问题:

- 推理模型推动了 AI 的解决问题能力
- 但高级推理需要付出代价：处理速度缓慢
- 推理模型通过生成大量标记来“思考”
- 多重中间计算增加了处理时间
- 生产部署需要解决这些挑战

实际性能瓶颈

- 推理模型比类似的非推理模型慢 2-3 倍
- 更高的延迟显著影响用户体验和成本效率
- 传统加速方法常常会损害推理质量
- 需要特殊解决方案，在提高速度的同时保持推理能力

LoRA 和 Turbo LoRA

LoRA: 低秩适应

- 使用小型可训练参数集微调大型模型
- 保留原始权重，维持预训练知识
- 显著减少微调所需的内存要求
- 支持模型高效适应专业任务

Turbo LoRA: 推理速度提升 2-4 倍

主要特点

- 使用推测解码以及专有优化
- 并行预测多个标记，然后验证它们
- 在更快生成文本的同时保持输出质量

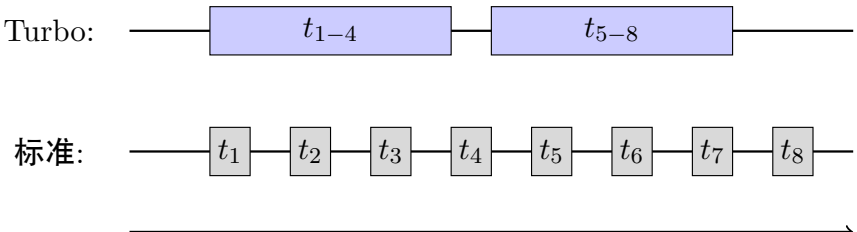
Turbo LoRA 如何工作

技术实现

1. 小型、快速的“推测器”模型并行预测多个标记
2. 主模型验证预测的标记
3. 正确的标记立即被使用
4. 只有不正确的标记需要重新计算

实际效益

- 最终输出质量零差异
- 应用于 DeepSeek-R1-distill-qwen-32b 模型
- 实现了 2-3 倍的处理量提升
- 可应用于任何推理模型



Turbo 对大型推理模型的益处

实现实时 AI 可行性

- 2-3 倍的速度提升使推理模型适用于:
  - AI 驱动的客户支持

Turbo 对大型推理模型的益处（续）

实现实时 AI 可行性（续）

- 2-3 倍的速度提升使推理模型适用于:
  - 开发者的 AI 副驾驶
  - 医疗保健 AI 助手
- 降低 GPU 成本：相同工作量需要更少的 GPU

大型推理模型 Turbo 的优势

实际影响

- 使几乎任何组织都能负担推理模型的运行
- 典型推理设置：所需 GPU 容量减少 2-3 倍
- 成本节约随部署规模扩大而增加

实施方法

- 可使用更小、更高效的模型而不牺牲能力
- 详细实施教程：https://predibase.com/blog/turbo-lora

教程: 使用 RFT 编写 CUDA 核心

GPU 代码生成为何困难

GPU 编程的挑战:

- 并行架构: 多核心
- 内存层次: 多个层级
- 线程同步: 复杂
- 小错误 $\rightarrow$ 大后果

传统方法的失败:

- 高质量 CUDA 示例稀少
- SFT 需要成千个配对
- 需要处理多种边缘情况
- 细微错误导致失败

RFT 为何适合代码生成

关键优势:

- 无需大型数据集
- 代码可验证

实现:

- 从 13 个示例开始
- 强化学习智能探索

设置 CUDA 任务: 最小数据集

数据集组成

- 每个示例包含:
  - 一个 PyTorch 函数（如矩阵乘法或激活函数）
  - 一组用于验证正确性的测试用例
- 示例函数包括矩阵运算、激活函数和元素级运算
- 测试用例涵盖边缘情况、不同大小和边界条件

PyTorch 到 Triton 示例

- PyTorch 函数:

```
def add(x, y): return x + y
```

Setting Up the CUDA Task: Triton Implementation

Target Triton Kernel

```
@triton.jit
def add_kernel(x_ptr, y_ptr, output_ptr, n_elements):
    pid = tl.program_id(0)
    block_size = 128
    offsets = pid * block_size + tl.arange(0, block_size)
```

设置 CUDA 任务: Triton 实现（继续）

目标 Triton 核心（继续）

```
mask = offsets < n_elements
x = tl.load(x_ptr + offsets, mask=mask)
y = tl.load(y_ptr + offsets, mask=mask)
output = x + y
tl.store(output_ptr + offsets, output, mask=mask)
```

为代码生成定义奖励

多层次奖励结构 (1)

奖励 1: 格式化 (0.1-0.3)

- 代码结构、导入和标签
- 为良好 Triton 语义给予部分分数
- 合理的变量名和代码组织

多层次奖励结构 (2)

奖励 2: 编译 (0.3-0.6)

- 无错误执行的代码
- 无运行时异常或语法错误
- 正确导入所需依赖

为代码生成定义奖励（继续）

多层次奖励结构 (3)

奖励 3: 正确性 (0.6-1.0)

- 输出与测试输入上的 PyTorch 函数匹配
- 反奖励黑客措施（检查硬编码输出）
- 正确处理边缘情况和不同输入形状

该奖励结构的益处

- 为模型提供清晰的进阶路径
- 允许为部分解决方案给予部分分数
- 模拟人类学习编码的方式

奖励函数的示例实现

奖励函数逻辑 - 第 1 部分

1. 格式检查:
   - 如果代码有正确的 Triton 语法 $\rightarrow$ 奖励 $= 0.2$
2. 编译检查:
   - 如果代码成功编译 $\rightarrow$ 奖励 $= 0.5$
   - 如果编译失败 $\rightarrow$ 返回当前奖励

奖励函数的示例实现（继续）

奖励函数逻辑 - 第 2 部分

3. 正确性检查:
   - 对每个测试用例，比较 PyTorch 和 Triton 结果
   - 如果输出匹配 $\rightarrow$ 奖励 $= 1.0$
   - 否则 $\rightarrow$ 奖励 $= 0.6$

关键要素

- 多阶段评估过程
- 每个阶段建立在前一阶段之上
- 部分成功给予部分奖励
- 通过测试用例进行确定性验证
- 数值从 0.0 到 1.0 缩放
- 多个测试用例确保稳健性

训练循环和结果

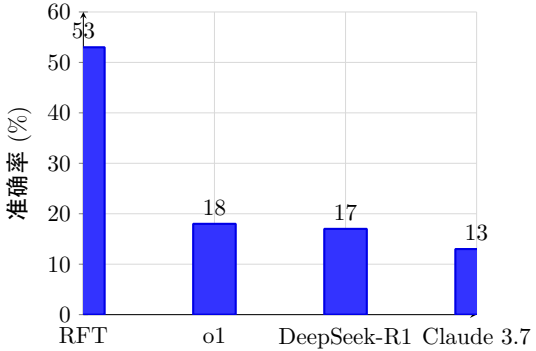
GRPO（基于梯度的奖励策略优化）如何工作:

- 生成: 使用采样为每个提示生成多个完成
- 评估: 对每个完成运行奖励检查
- 更新: 计算优势并反向传播信号
- 重复: 模型精细策略以最大化奖励

结果:

- 在 5,000 步后对保留示例的准确率为 53%
- 比 OpenAI o1 和 DeepSeek-R1 高出 3 倍的正确率
- 比 Claude 3.7 Sonnet 性能好 4 倍

性能比较: RFT 与领先模型对比



RFT 训练进展

- 起始点: 5% 准确率
- 早期训练 (1,000 步): 22%
- 中期训练 (3,000 步): 41%
- 最终结果: 53% 准确率
- 模型开发了超越有限训练示例的可泛化模式

与领先模型的性能比较

结果:

- 在 5,000 步后对保留示例的准确率为 53%
- 比 OpenAI o1 和 DeepSeek-R1 高出 3 倍的正确率
- 比 Claude 3.7 Sonnet 性能好 4 倍

Key Success Factors

- 奖励设计: 多级奖励提供了清晰的学习信号
- 测试多样性: 多样化的测试用例防止了过拟合
- 反奖励黑客: 防止模型仅仅记忆输出

使用 Unsloth 实现实用的 RFT

Unsloth 实用 RFT 工作流程

Unsloth 是什么?

- 用于高效 LLM 微调的开源库
- 可在有限硬件上运行（3GB+ 显存）
- 支持 QLoRA、LoRA、RLHF、GRPO
- 比标准方法快 2-4 倍

开发工作流程

1. 设置: 安装依赖项
2. 模型: 选择基础模型
3. 数据: 格式化训练数据
4. 配置: 设置参数
5. 训练: 运行微调
6. 部署: 保存并服务

步骤 1: 环境设置

安装 Unsloth
<MINTED>

步骤 1: 所需库

<MINTED>

性能优化

- 适用于显存有限设备的 CPU 卸载
- 用于更快训练的 Flash Attention 2（在支持的 GPU 上）
- 梯度检查点以计算换内存

步骤 2: 选择模型与方法

加载基础模型

<MINTED>

选择微调方法

- QLoRA: 与 LoRA 结合的 4 位量化（最少资源）
- LoRA: 低秩适应（质量/资源平衡）
- GRPO: 用于 DeepSeek 风格的强化学习
- 全量微调: 在高端硬件上获得最佳质量

步骤 2: 为 RFT 配置 LoRA

RFT 特定配置

- 针对键值注意力层以获得最高效的适应
- 对于强化学习等复杂任务使用秩 16-32
- 添加 dropout 防止对奖励信号过拟合
- 启用梯度检查点以在训练期间节省内存

代码实现

<MINTED>

步骤 3: 数据集准备

RFT 的数据集格式

- 格式化包含输入和预期输出的数据集
- 包含奖励信号或成功标准
- 确保不同场景下的多样化示例

<MINTED>

步骤 3: 数据集格式化为训练

应用适当的模板

- 使用模型的特定聊天模板格式
- 确保正确应用特殊标记
- 为您的用例设置适当的序列长度

<MINTED>

步骤 4: 配置训练参数

训练参数

<MINTED>

步骤 4: RFT 的超参数选择

关键超参数

- LoRA 秩 (r): 在 8-32 之间；越高提供更强的表达能力
- LoRA Alpha: 通常与秩相同
- 学习率: QLoRA 为 2e-4 至 5e-4
- 训练轮数: 小数据集为 2-5 轮
- Dropout: 0.05-0.1 用于正则化

RFT 中避免过拟合

- 如果出现过拟合迹象，增加 dropout
- 通过监控损失实现早停
- 使用验证集评估泛化能力
- 添加权重衰减进行正则化

步骤 5: 执行训练

执行训练
<MINTED>

步骤 5: 训练过程与监控

运行训练

<MINTED>

监控关键指标

- 训练损失: 应当稳定下降，但不应过快达到平台期
- 验证损失: 监控过拟合迹象（验证损失上升）
- 学习率: 记录学习率变化对性能的影响
- GPU 利用率: 验证资源使用效率

步骤 6: 评估

评估模型性能

- 加载微调后的模型进行推理
- 创建结构化测试提示
- 在相同输入上与基线模型进行比较
- 测试多样化场景和边缘案例
- 测量推理速度和资源使用情况

<MINTED>

步骤 6: 部署

保存和部署模型

- 保存训练好的模型和分词器
- 优化推理性能（例如，ONNX 转换）
- 配置部署环境
- 设置监控和日志记录

<MINTED>

步骤 6: 持续改进

持续的精细化

- 在生产环境中收集用户反馈和模型输出
- 根据真实世界性能更新奖励函数
- 使用部署中发现的边缘案例扩展训练数据集
- 定期重新训练以纳入改进

长期维护

- 监控随时间推移的性能退化
- 跟踪强化微调领域的新最佳实践
- 评估更新到新的基础模型的收益

可直接使用的 Unsloth 笔记本

可用的实现资源

- Unsloth 提供了适用于各种模型和任务的现成笔记本
- 可通过 Google Colab 或 Kaggle 访问（免费 GPU 资源）
- 包含监督微调和 GRPO（强化微调）实现

流行模型

- Llama 3.1 (8B)
- Phi-4 (14B)
- Mistral (7B, 22B)
- Qwen 2.5 (3B, 14B)
- Gemma 2 (2B, 9B)

专用变体

- Qwen2.5-Coder (14B)
- CodeGemma (7B)
- Llama 3.2 Vision
- Qwen2-VL (7B)
- Phi-3 Medium

笔记本可在以下地址获取: https://docs.unsloth.ai/get-started/unsloth-notebooks

RFT 的数据集构建

入门指南

- 确定数据集的目的: 聊天对话、结构化任务或领域特定数据
- 定义期望的输出风格: JSON、HTML、文本、代码或特定语言
- 确定数据来源: Hugging Face 数据集、维基百科或合成数据

常见数据格式

- 纯文本格式
- 指令-输入-输出
- ShareGPT 格式（多回合）
- ChatML（OpenAI 风格）

数据集要求

- 最少: 100 个示例
- Optimal: 1,000+ examples
- Quality over quantity
- Can combine multiple datasets

RFT Implementation: Common Pitfalls (1)

- Problem: Model produces generic or refuses to generate responses
  Solution: Increase LoRA rank and alpha; ensure diverse training data
- Problem: Catastrophic forgetting (model loses pre-trained capabilities)
  Solution: Lower learning rate; add regularization
- Problem: High training loss that doesn't converge
  Solution: Check dataset formatting; reduce sequence length

RFT 实施：常见问题 (2)

- 问题：训练期间内存不足错误
  解决方案：启用梯度检查点；减小批量大小
- 问题：模型生成幻觉
  解决方案：实施惩罚虚构内容的奖励函数
- 问题：使用奖励信号时训练不稳定
  解决方案：归一化奖励；使用带有适当裁剪的 PPO

结论

关键要点

RFT 的力量:

- 自我提升: 模型超越静态方法
- 数据效率: 使用更少数据超越 SFT
- 速度: Turbo LoRA 将处理量提高 2-4 倍
- 实际应用: 超越学术领域

范式转变:

- 传统方式: 大数据 $\rightarrow$ 静态模型
- RFT: 最小数据 + 奖励 $\rightarrow$ 成长
- 新循环: 持续改进

实施策略 - 核心组件

所需核心组件

1. 基础模型（开源 LLM）
2. 奖励函数定义
3. 提示数据集（可以很小）
4. 强化学习算法（RLHF、GRPO 等）
5. 评估框架

实施策略 - 最佳实践

最佳实践

- 从小型、清晰的奖励函数开始
- 构建全面的验证测试
- 实施反奖励黑客措施
- 监控演发行为
- 逐步增加复杂性

RFT 的后续步骤

开始使用:

- 强化微调标志着 LLM 发展的重大飞跃
- 通过奖励信号而非标记示例进行训练
- 端到端平台使开发者能够轻松使用这种方法
- 专注于创新而非基础设施复杂性

RFT 应用的成熟领域

- 专业编程: 领域特定的代码生成（嵌入式系统、高性能计算）
- 科学研究: 提出并验证假设的模型
- 推理任务: 复杂的逻辑和数学问题解决
- 教育: 能够理解学生知识空白的自适应辅导系统

Thank You!

Questions?