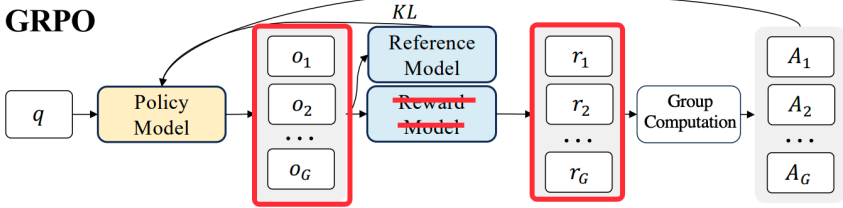




GRPO



【手撕LLM-GRPO】你只管给Reward, 剩下的交给RL (附代码)



小冬瓜AIGC

原创课程 公众号: 手撕LLM

关注他

来自专栏 · 手撕LLM >

568 人赞同了该文章 >

起

我是小冬瓜AIGC，原创超长文知识分享，原创课程已帮助多名同学速成上岸LLM赛道。
研究方向：LLM、RL、RLHF

关于作者



小冬瓜AIGC

原创课程 公众号: 手撕LLM

回答

8

文章

37

关注者

4,854

关注他

发私信

你只管给Reward, 剩下的交给RL

前置阅读

我们在前三篇分别从实现方法和实现思想解读了DeepSeek-R1:

小冬瓜AIGC: 【解读】DeepSeek-R1: RL前真的不需要SFT了吗???

小冬瓜AIGC: 再深挖DeepSeek-R1: Reward is Enough

小冬瓜AIGC: X-R1: 不到50元, 人人都能复现0.5B Aha Moment

本文涉及到的GRPO-loss代码开源, X-R1复现Aha Moment开源

github.com/dhcode-cpp/g...

[GitHub - dhcode-cpp/X-R1: minimal-cost for training 0.5B R1-Zero](https://github.com/dhcode-cpp/X-R1)

1. 问题设置

1.1 问题定义和评价标准

在DeepSeek-R1-Zero中, 仅通过RL算法就能提升模型的数学推理能力。我们给定数学问题 q , 让模型进行回答 o , 再将回答与标签 y 判别正确性 $\mathbb{I}(o = y)$, 给定 n 个问题, 模型能够答对 k 个, 则准确率为 $\text{acc} = k/n$, 那么我们的评判模型推理能力的标准是有更高的准确率。

对于选择题来说, 模型的回答 o 可以直接输出 $\{A, B, C, D\}$ 之中的限定一个, 但是对于数学解答题来说, 模型要生成一系列的指示函数的判别应等于1

赞同 568

141 条评论

分享

喜欢

收藏

申请转载

...

$$\mathbb{I}(1/2 = 0.5) = 1$$



以上的数学问题是特定形式的，即一个问题对应于一个客观的答案，但是客观的答案有多种表达形式，所生成答案与标签的数学含义是相等的，才能算是正确，这种特性我们简称为答案的一致性。并非严格意义上要求字符串匹配或者是token序列匹配，比如0.5和1/2 判别为错误，那么将误导模型的学习，造成收敛困难。

为了方便我们提取最终答案，我们可以要求模型输出正确答案时，以 $\boxed{1/2}$ latex 格式进行包裹。

1.2 CoT⁺与LongCoT

对于一个算术问题，我们通常有如下“提示词工程”手段来推断出模型的答案：

1. 直接问答：让模型直接输出预测的最终答案，但通常准确率低。
2. In-context Learning：在提示词增加example，来进行引导，比如在MMLU选择题中，让base模型参考例子来生成 $\{A, B, C, D\}$ 中的一个选项
3. 提示词工程Chain-of-Thought(CoT)：在提示词中，加入一种结构化的解答形式的例子，来帮助模型进行推断，使得模型能够生成相似的结构化解答形式，增加回答正确答案的可能性。
4. Auto-CoT：在3中我们针对不同类型的问题需要特定的示例，Auto-CoT提出一种“Let's think step-by-step”的提示词，让模型自动化生成结构化的解答。

上述都是基于提示词工程的角度，不同程度的增强模型的推理能力。另外

1. 我们也可以收集问题和结构化解答的回答(CoT形式的回答), 对Base模型进行SFT训练，将结构化推理能力训入到模型参数里，在实际使用模型时，我们可以使用zero-shot的方式，得到CoT式的解答。

我们比较方法(2,3,4)与(5)的方法最大的差异在于模型的参数是否发生了改变，前者是从输入提示词里进行即时的学习，后者是将解答形式永久的学习(模型参数发生了改变)。

综上我们希望，能够让模型内在的学会推理，即面对一个数学问题时，inference时能够CoT形式推导出正确的答案；

当问题较为复杂，模型生成的内容可能具有反复的猜想、推理和验证的推理属性，通常会产生大量的token，这里的量级我们用长度来衡量，即是Long-CoT⁺。二者实际上没有本质的区别，都是为了推导出最终答案而服务的，缺陷在于Long-CoT需要耗费更多的inference decoding资源，需要注意：

1. CoT的推理过程可能具有错误
2. CoT推理过程有错误也可能回答到正确答案。
3. CoT推理过程可能不能回答到任何答案。
4. Long-CoT/CoT并无明确的长度定义的区别标准
5. Long-CoT实际上依然考究模型的long-context能力，要在复杂的上下文中推导出正确答案
6. 模型智能到达一定程度，或许可以跳过CoT直接给出答案

好的模型，应该有更简短的CoT和高的准确率。一味的追求Long-CoT并不是最理想的，其带来的成本增长是有差异的，比如生成1w长度的Token和生成2w长度的Token，推理的成本并非是非线性增长的。

1.3 输入和输出

我们在进行训练时，通常需要准备问题集，和每个问题对应的答案标签。

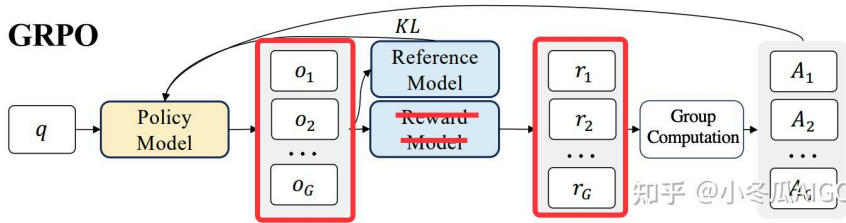
模型输入：问题

模型输出：CoT解答和最终答案

判别：最终答案和标签的一致性。

1.4 GRPO优化算法⁺

GRPO优化算法概述如下, GRPO优化过程仅需要加载两个模型, 一个是目标优化模型, 一个是ref模型。由于有规则reward的判别, 并不需要reward model。(如果判别是由奖励模型, 那么仍需要, 比如用GRPO做RLHF需要peference reward model)



$$\mathcal{L}_{\text{GRPO}}(\theta) = -\frac{1}{G} \sum_{i=1}^G \frac{1}{|o_i|}$$

$$\sum_{t=1}^{|o_i|} \left[\min \left(\frac{\pi_{\theta}(o_{i,t} | q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} | q, o_{i,<t})}, \text{clip} \left(\frac{\pi_{\theta}(o_{i,t} | q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} | q, o_{i,<t})}, 1 - \epsilon, 1 + \epsilon \right) \hat{A}_{i,t} \right) - \beta \mathbb{D}_{\text{KL}} [\pi_{\theta} \| \pi_{\text{ref}}] \right],$$

$$\mathbb{D}_{\text{KL}} [\pi_{\theta} \| \pi_{\text{ref}}] = \frac{\pi_{\text{ref}}(o_{i,t} | q, o_{i,<t})}{\pi_{\theta}(o_{i,t} | q, o_{i,<t})} - \log \frac{\pi_{\text{ref}}(o_{i,t} | q, o_{i,<t})}{\pi_{\theta}(o_{i,t} | q, o_{i,<t})} - 1,$$

$$\hat{A}_{i,t} = \frac{r_i - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r})}$$

1. 对于模型 π_{θ} 给定一个问题 q 进行采样生成多个回答 $o_i, i = 1, 2, 3, \dots, G$, G 为group数量, 而每个回答有不同的长度为 $|o_i|$
2. $\pi_{\theta}(o_{i,t} | q, o_{i,<t})$ 为在1的采样解答 $o_{i,t}$ 解码的第 t 个词元的策略概率。
3. KL项约束 π_{θ} 策略分布与原始策略 π_{ref} 分布不能差异太大, 这里采用一种优化的KL项, 具有无偏且方差小的特性。
4. $\mathbf{r} = \{r_1, r_2, \dots, r_G\}$, 而 $\hat{A}_{i,t}$ 为组相对的优势。 t 意味着一个回答的每个token的优势都是一样的。

2. 在线采样

2.1 格式化prompt

我们给定常规的问答模版, 比如我们问询: “如何写python程序”, 通常需要进行格式化处理, 并附上模型发言的前缀 #ASSISTANT:

```
#SYSTEM: 你需要友善的回答用户的问题
#USER: 如何写python程序
#ASSISTANT:
```

对于数学问题我们可以如下定义模版, 在问题后面附上 <THINK> . 所以我们输入到模型里面的内容, 也声明了回答的格式, 仍要求模型具备一定的指令跟随能力。

```
#SYSTEM: 你需要在<THINK>标签后面思考以<\THINK>结尾,
答案前后分别使用<ANSWER>和<\ANSWER>, 最终答案以$\boxed{\}$来处理。
#USER: 求解f(x)=x^3+e^x+1的最小值
<THINK>
```

基于上述的提示词, 我们可以对模型 π_{θ} 进行采样出多条回答 o_i 。回答的类型可能有

1. 没有 <THINK>
2. 没有 <ANSWER>

3. 没有 `<\ANSWER>`
4. 答案没有 `$\boxed{\}$` 框住

上述四种情况都为未遵循指令要求的格式输出。正确的模型输出格式为：

```
<THINK>{SOLUTION} <\THINK>
<ANSWER> the final answer is $\boxed{1/2}$ <\answer>
```

2.2 说明

这里 π_θ 是on-policy采样, 在GRPO相关论文里, 采样的数量 $G = 64$, 采样过程实际上为一种探索, 如果能够探索到正确答案, 得到正的奖励。

所以在数学问题的GRPO优化里, 我们不需要提前准备好解答过程 {SOLUTION}, 只需要让模型去采样(合成) 数据。

在上述生成过程里如果能够生成 `<THINK> {SOLUTION} <\THINK>`, {SOLUTION} 指代模型的 CoT 推理。

我们讨论两种情况, 对于GRPO算法来说, 哪个是好的采样?

1. 生成的解答过程是对的, 但是答案错误, 即 `$\boxed{1/3}$`
2. 生成的解答过程是错的, 但是答案正确, 即 `$\boxed{1/2}$`

答案是2, 因为该形式的数学问题, 分数是结果导向的, 而非过程。

那么又产生了新的疑问:

Q1: 上述采样1判别是否合理?

Q2: 采样2的解答过程是否会随着RL训练, 采样的解答过程越来越严谨?

3. 奖励函数和优势

奖励通常有两种:

1. model-base: 通常需要有神经网络或带学习参数的算法, 对数据进行打分, 如preference reward, PRM
2. ruled-base: 指的是按照一定的规则就能确认, 如选择题的ABCD选项、字符串匹配和算术题数字、以及围棋, 均可以通过确定的规则进行判别。

第二种, 并没有参数其判别更加直接且客观。

3.1 规则奖励

在模型的回答 o_i 里我们要提取 $1/2$, 与最终的结果比较 0.5

```
{SOLUTION} <\THINK>
<ANSWER> the final answer is $\boxed{1/2}$ <\answer>
```

这里涉及到答案的一致性判别, 为了方便实现我们可以严格的要求字符串等同, 但效率太差。

我们看open-r1早期实现判别:

```
def accuracy_reward(completions, ground_truth, **kwargs):
    """Reward function that checks if the completion is the same as the gr
    # Regular expression to capture content inside \boxed{}
    contents = [completion[0]["content"] for completion in completions]
    answers = [extract
```

```
# Reward 1 if the content is the same as the ground truth, 0 otherwise
return [1.0 if answer == gt else 0.0 for answer, gt in zip(answers, gr
```

当前实现hf写了个库做一致性判别 [huggingface/Math-Verify](#)⁺, 可以通过调包来判别。但是具体的math_verify未探究

```
from math_verify import LatexExtractionConfig, parse, verify
def accuracy_reward(completions, solution, **kwargs):
    """Reward function that checks if the completion is the same as the ground truth"""
    contents = [completion[0]["content"] for completion in completions]
    rewards = []
    for content, sol in zip(contents, solution):
        gold_parsed = parse(sol,
                             extraction_mode="first_match",
                             extraction_config=[LatexExtractionConfig()])
        if len(gold_parsed) != 0:
            # We require the answer to be provided in correct latex (no math mode)
            answer_parsed = parse(
                content,
                extraction_config=[
                    LatexExtractionConfig(
                        # ...
                        boxed=True, # 提取boxed
                        # ...
                    ),
                    # Ensures that boxed is tried first
                    boxed_match_priority=0,
                    try_extract_without_anchor=False,
                ]
            ),
            extraction_mode="first_match",
        )
        # Reward 1 if the content is the same as the ground truth, 0 otherwise
        reward = float(verify(answer_parsed, gold_parsed)) # 进行正确性判别
```

3.2 SimpleEval判别

那么我们也可以用LLM-As-Judge⁺的形式进行判别, 参照OpenAI的SimpleEval的实现, 判别模版如下, 即让GPT-4模型in-context learning来判别。

```
# simple-evals/common.py

EQUALITY_TEMPLATE = r"""
Look at the following two expressions (answers to a math problem) and judge if they are equivalent.

Examples:

Expression 1: $2x+3$
Expression 2: $3+2x$

Yes

Expression 1: 3/2
Expression 2: 1.5

Yes

Expression 1: $x^2+2x+1$
Expression 2: $y^2+2y+1$
.....

No
(these are actually equal, don't mark them equivalent if you need to do no
```

```
Expression 1: 2/(-
```

Expression 2: -2/3

Yes

(trivial simplifications are allowed)

Expression 1: 72 degrees

Expression 2: 72

Yes

(give benefit of the doubt to units)

....

YOUR TASK

Respond with only "Yes" or "No" (without quotes). Do not include a rationale

Expression 1: %(expression1)s

Expression 2: %(expression2)s

"".strip()

根据上述的LLM-As-Judge, 我们得到YES或者NO, 那么对应的奖励分数为1和0.

那么一组采样数据里, 我们得到对应的规则奖励为:

$$\mathbf{r} = \{r_1, r_2, \dots, r_G\} \in \{0, 1\}$$

奖励是sentence-level的, 奖励是标量值。

3.3 手撕Advantage

在上述的奖励函数所得到的奖励, 可以直接使用, 我们根据GRPO的优势函数公式计算

$$\hat{A}_{i,t} = \frac{r_i - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r})}$$

其中 $\hat{A}_{i,t}$ 里的 t 对应token-level优势, 即一个句子中, 每个token对应的优势是一样的。这种方式的好处在于, 估计都是从真实的环境reward计算得来, 而不是通过价值估计计算而得。

对应的代码实现为:

```
def grpo_advantage(rewards):
    epsilon = 0.00001
    rewards = torch.tensor(rewards, dtype = torch.float)
    A = (rewards - rewards.mean()) / (rewards.std() + epsilon)
    return A
```

测试, 我们假设采样6条输出, 对应6个奖励, 我们看case2里, 正确项相对上的优势估计更大。而错误的回答将得到负的值。当全错和全对时, 优势是无效的, 那么可以skip掉, 如果base模型差无法采样到正确反馈, 那么优化将会非常缓慢和不稳定。

```
A = grpo_advantage([0,0,0,0,0,0])
# A tensor([-0., -0., -0., -0., -0., -0.])

A = grpo_advantage([1,0,0,0,0,0])
# A tensor([ 2.0468, -0.4094, -0.4094, -0.4094, -0.4094, -0.4094])

A = grpo_advantage([1,0,0,0,1,0])
# A tensor([ 1.2905, -0.6452, -0.6452, -0.6452,  1.2905, -0.6452])

A = grpo_advantage([1,
```



```
def grpo_kl(pi_logprob, pi_ref_logprob):
    return pi_ref_logprob.exp() / pi_logprob.exp() - (pi_ref_logprob - pi_l

pi = torch.randn(3, 5) # batch, sequence
pi_ref = torch.randn(3, 5) # batch, sequence
pi_logprob = torch.nn.functional.log_softmax(pi, dim = 1)
pi_ref_logprob = torch.nn.functional.log_softmax(pi_ref, dim = 1)
print(grpo_kl(pi_logprob, pi_ref_logprob))
```

实现上，我们通常存储logprob，而不需要完整的输出分布来计算KL，减少存储占用。

输出

```
tensor([[0.1465, 0.0457, 0.5321, 0.0793, 0.0728],
        [1.1576, 3.0678, 0.5285, 0.3712, 0.0817],
        [0.0031, 0.8179, 2.4575, 0.1597, 0.0221]])
```

均为正数。与PPO不同，不产生reward shaping: $R = r - KL$

4.3 无偏小方差KL

根据John Schulman: [Approximating KL Divergence](#)

- k1 (naive estimator) : $\log \frac{q(x)}{p(x)} = -\log r$, 其中 $r = \frac{p(x)}{q(x)}$ 。
 - 这个估计器是无偏的，但方差很大，因为它对一半的样本是负的即 $q(x) < p(x)$ ，将导致估计值波动非常大。
 - 当 $q(x)$ 和 $p(x)$ 差异大时，对数值也可能产生很大的变化。
- k2 (low variance estimator) : $\frac{1}{2} (\log \frac{p(x)}{q(x)})^2 = \frac{1}{2} (\log r)^2$ 。
 - 这个估计器是有偏的，但方差较低。
 - 并且实证偏差很小。因为保证所有样本的值都是正的
- k3 (unbiased low variance estimator) : $(r - 1) - \log r$ 。这个估计器是无偏的，并且方差较低。
 - 在k1基础上增加 $(r - 1)$ 项，减少方差
 - $r - 1$ 是无偏的
 - $r - 1 > \log r$, 保证 k3 都是正值

4.4 手撕KL

```
import torch.distributions as dis
p = dis.Normal(loc=0, scale=1)
q = dis.Normal(loc=0.1, scale=1)
x = q.sample(sample_shape=(10_000_000,))
truekl = dis.kl_divergence(p, q)
print("true", truekl)
logr = p.log_prob(x) - q.log_prob(x)
k1 = -logr
k2 = logr ** 2 / 2
k3 = (logr.exp() - 1) - logr
for k in (k1, k2, k3):
    print((k.mean() - truekl) / truekl, k.std() / truekl)
```

打印

Now let's compare the bias and variance of the three estimators for $KL[q,p]$.
Suppose $q=N(0,1)$, $p=N(0.1,1)$. Here, the true KL is 0.005.

结果为如下，常规的KL有较

	bias/true	stdev/true
K1	0	20
K2	0.002	1.42
K3	0	1.42

Now let's try for a larger true KL divergence. $p=N(1,1)p=N(1,1)$ gives us a true KL divergence of 0.5.

	bias/true	stdev/true
K1	0	2
K2	0.25	1.73
K3	0	1.7

4.5 无KL版本的GRPO

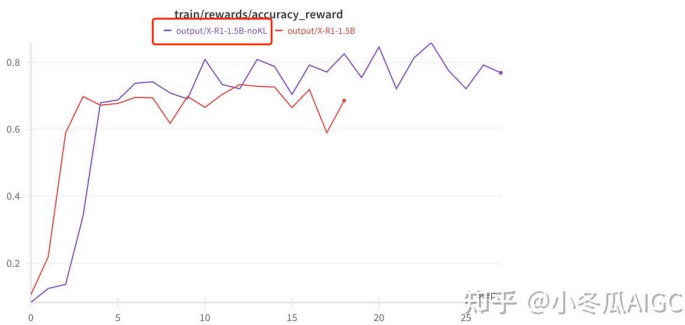
小冬瓜AIGC：GRPO简化Tricks, 性能暴涨10%, 只改一个参数?

170 赞同 · 34 评论 文章

在 trl #2806里, @ingambe 大胆的提出要把GRPO里的KL项去掉, 去掉ref-model, 并且实验work了。可以快速验证就是把KL项的\beta置为0

$$\mathcal{L}_{\text{GRPO_trl}}(\theta) = -\frac{1}{G} \sum_{i=1}^G \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \left[\frac{\pi_{\theta}(o_{i,t} \mid q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} \mid q, o_{i,<t})} \hat{A}_{i,t} - \beta \text{D}_{\text{KL}}[\pi_{\theta} \parallel \pi_{\text{ref}}] \right],$$

我在X-R1上测试是可行的, 对比有KL和无KL的版本, 无KL竟然高了10个点左右



4.6 KL项分析

- 1. 去除KL项, 意味着不需要ref-model, 减少一个模型的显存, 减少一次前向ref_policy的计算。
- 2. 没有KL的约束, 那么可以将过大的梯度进行裁剪(max_grad_norm), 避免优化的不稳定性 (这也是另一种层面的clip)。
- 3. 没有KL的约束, 参数的优化更加自由, 更容易探索到好的回答

5. GRPO损失

5.1 GRPO损失项分析

GRPO loss看起来复杂, 实际上, 仅包含三部分:

- 1. 第一个连加的G 为一个样本的采样数量, 第二个|o_i| 是第i条输出的采样长度
- 2. 在min(.,.) 里, 与标准PPO差异不大, 这里的advantage我们已经提前计算好了, 在一条采样回答数据中对于不同的t 优势值都一样的。另外这里的ratio对比的是新旧策略。这个式子是token-level的。
- 3. KL项\beta因子控制约束力度,

$$\mathcal{L}_{\text{GRPO}}(\theta) = -\frac{1}{G} \sum_{i=1}^G \frac{1}{|\mathcal{O}_i|} \sum_{t=1}^{|\mathcal{O}_i|} \left[\min \left(\frac{\pi_{\theta}(\mathcal{O}_{i,t} | q, \mathcal{O}_{i,<t})}{\pi_{\theta_{\text{old}}}(\mathcal{O}_{i,t} | q, \mathcal{O}_{i,<t})} \hat{A}_{i,t}, \text{clip} \left(\frac{\pi_{\theta}(\mathcal{O}_{i,t} | q, \mathcal{O}_{i,<t})}{\pi_{\theta_{\text{old}}}(\mathcal{O}_{i,t} | q, \mathcal{O}_{i,<t})}, 1 - \epsilon, 1 + \epsilon \right) \hat{A}_{i,t} \right) - \beta \mathbb{D}_{\text{KL}} [\pi_{\theta} \| \pi_{\text{ref}}] \right]$$

上述的优化目标是最小化，那么实际训练过程会产生

- 初始阶段Loss为0
- 训练过程Loss存在负值的情况
- 训练过程由于目标优化模型的策略会远离ref模型，还可能导致loss上升的趋势，那么进一步分析

小冬瓜AIGC: GRPO的Loss为什么会有负值?
47 赞同 · 3 评论 文章

5.2 手撕GRPO

我从自己实现了全流程的GRPO里，摘选出关键的Loss函数，帮助大家理解算法，代码能对应原公式。

<https://github.com/dhcode-cpp/grpo-loss>
github.com/dhcode-cpp/grpo-loss

```
def grpo_loss(pi_logprob, pi_old_logprob, pi_ref_logprob, advantage, input,
              epsilon = 0.2,
              beta = 0.01):

    bs, seq_len = pi_logprob.shape
    # skip计算采样的每条采样长度
    len_oi = torch.tensor([len_oi] * group_num, dtype = torch.long)
    # 设定mask, 仅对response 为 1, 算loss
    mask = torch.zeros(bs, seq_len)
    mask[:, input_len:] = 1

    # GRPO loss
    ratio = torch.exp(pi_logprob - pi_old_logprob)
    ratio_clip = torch.clamp(ratio, 1 - epsilon, 1 + epsilon)
    advantage = advantage.unsqueeze(dim = 1) # [a, b, c] -> [[a], [b], [c]]
    policy_gradient = torch.minimum(ratio * advantage, ratio_clip * advantage)
    kl = grpo_kl(pi_logprob, pi_ref_logprob)

    loss = (policy_gradient - beta * kl) * mask
    loss = (-1 / group_num) * (1 / len_oi.unsqueeze(dim = 1)) * loss
    loss = loss.sum()

    return loss
```

调用输出

```
# 输出分布
pi_logits = torch.randn(3, 5, 32) # batch, seq_len, vocab_size
pi_ref_logits = torch.randn(3, 5, 32)
pi_old_logits = torch.randn(3, 5, 32)

# 获取log prob
pi_logprob = F.log_softmax(pi_logits, dim = -1)
pi_ref_logprob = F.log_softmax(pi_ref_logits, dim = -1)
pi_old_logprob = F.log_softmax(pi_old_logits, dim = -1)

# group data
token_ids = torch.tensor([
    [1, 2, 3, 4, 5],
    [6, 7, 8, 9, 10],
    [11, 12, 13, 14, 15]
```

```
# 获取policy
pi_logprob = torch.gather(pi_logprob, dim=-1, index=token_ids.unsqueeze(-1))
pi_ref_logprob = torch.gather(pi_ref_logprob, dim=-1, index=token_ids.unsqueeze(-1))
pi_old_logprob = torch.gather(pi_old_logprob, dim=-1, index=token_ids.unsqueeze(-1))
loss = grpo_loss(pi_logprob, pi_old_logprob, pi_ref_logprob, A, 3, 2)
print(loss)
```

输出

```
tensor(0.0033)
```

可见, GRPO仅需通过在线采样和规则奖励, 就能进行RL训练。

5.3 TRL GRPO算法实现

截止到 Trl-0.15.0 GRPO当前未实现clip ratio, 如下所示:

- 如果不clip, 那么直接算新旧ratio, 这里的新旧ratio的 π_θ 和 $\pi_{\theta_{old}}$ 的参数是一样的。
- 这里的新旧policy的 ratio 恒为 1

$$\mathcal{L}_{GRPO}(\theta) = -\frac{1}{G} \sum_{i=1}^G \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \left[\min \left(\frac{\pi_\theta(o_{i,t} | q, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t} | q, o_{i,<t})} \hat{A}_{i,t}, \text{clip} \left(\frac{\pi_\theta(o_{i,t} | q, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t} | q, o_{i,<t})}, \text{ratio} \right) \right]$$

$$\mathcal{L}_{GRPO-TRL}(\theta) = -\frac{1}{G} \sum_{i=1}^G \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \left[\left(\frac{\pi_\theta(o_{i,t} | q, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t} | q, o_{i,<t})} \right) \hat{A}_{i,t} \right]$$

我们先看trl的实现

```
advantages = inputs["advantages"]
# x - x.detach() allows for preserving gradients from x
# 这段代码能否计算出梯度?
per_token_loss = torch.exp(per_token_logps - per_token_logps.detach()) * advantages
per_token_loss = -(per_token_loss - self.beta * per_token_kl)
loss = ((per_token_loss * completion_mask).sum(dim=1) / completion_mask.sum(dim=1))
```

首先模型得到的是 per_token_logps , 这里概率是 logprob 形式的, 工程上不容易溢出。

$$r = \frac{\pi_\theta}{\pi_{\theta_{old}}} = \exp \log \left(\frac{\pi_\theta}{\pi_{\theta_{old}}} \right) = \exp(\log(\pi_\theta) - \log(\pi_{\theta_{old}})) = \exp(0) = 1$$

```
# 以下恒为0
per_token_logps - per_token_logps.detach() == 0

# torch.exp(0) == 1 恒为 1
ratio = torch.exp(per_token_logps - per_token_logps.detach())

# 等价于
per_token_prob = torch.exp(per_token_logps)
per_token_prob_old = torch.exp(per_token_logps.detach())
ratio = per_token_prob / per_token_prob_old
```

这里我们分析关键代码 `x-x.detach()` , 是否无法更新梯度。

1. 前向时: ratio恒为1, 那么前向如果advantage不为0, 那么 ratio*advantage 不为0
2. 反向时: 由于 per_token_prob_old 已经detach, 那么如果有梯度的话, 我们可以参考 [GitHub - dhcode-cpp/grpo-loss](https://github.com/dhcode-cpp/grpo-loss) , 可以计算出 per_token_logps 是有梯度的。

```
policy = torch.tensor([0.5])
old_policy = torch.tensor([0.5])
ratio = policy/old_policy
```

```

print(ratio)

ratio = torch.exp( policy.log() - old_policy.log())
print(ratio)

gradient = -0.2 # 假设反传误差
policy_gradient = - gradient * ( 1 / old_policy)
print(policy_gradient)

```

那么我们可以看见，在`x=x.detach()`是能正常计算梯度的。但是相当于new-policy只更新一次。

总的来说，标准的写法是在：

- $t=1$ 时刻，根据 $\theta_{t=1}$ 获取old policy和new policy，这两者ratio为1，参数更新 $\theta_{t=2}$
- $t=2$ 时刻，获取 $\theta_{t=2}$ 获取new policy，这两者ratio可能不为1, ... 以此类推

而TRL的实现在 $t=1$ 更新完梯度，就完成了当前批次的参数优化。

总结：Trl简化了GRPO的loss计算，但是仍然能work，我们在X-R1实验中，用这一套方法也能训练并且训出效果。

5.4 GRPO算法分析

我们分析GRPO的流程：

1. 模型在线采样时，实际上是生成CoT用于推断最终的答案
2. GRPO则通过规则反馈，引导模型参数调整

两者经过不断的迭代，能够产生越来越强的CoT，或者说是更长的Long-CoT

我们回到问题part2遗留问题：

1. 生成的解答过程是对的，但是答案错误，即 $\boxed{1/3}$
2. 生成的解答过程是错的，但是答案正确，即 $\boxed{1/2}$

Q1: 上述采样1判别是否合理？

A：合理，在规则奖励的设计就是辨别最终答案与标签的一致性。解答过程不参与规则奖励的判别

Q2: 采样2的解答过程是否会随着RL训练，采样的解答过程越来越严谨？

A：采样2的解答最终答案正确，错误的推断初期虽然会被强化，随着RL训练模型权重调整，CoT的推理的模式就会越来越严谨。但是初期可能会造成非常大的不稳定性。

6. 总结

1. 当前的数学推理问题形式，可以端到端的进行学习中间的reasoning能力，即通过RL调整模型参数，生成高质量的CoT，从而能够推导出最终正确的答案。
2. 在采样时，如果一个问题难以采样到正确的答案，由于没有正反馈，将导致难以有效优化，初期RL可能存在训练缓慢和不稳定问题。
3. 早期有ReMax和RLOO等方法均有相似去除value-model的思想，GRPO虽然减少Reward Model，但在线采样仍然考验系统效率。
4. 我们按照数学的问题形式，可以思考在垂直领域中，收集问题和答案，也可以无脑的做RL，也就是Reinforcement Fine-Tune(ReFT)

7. 手撕GRPO & R1-Zero复现

本文涉及到的GRPO-loss代码开源📄

<https://github.com/dhcode-cpp/grpo-loss>
github.com/dhcode-cpp/grpo-loss
github.com/dhcode-cpp/grpo-loss

在彻底理解GRPO算法后，我们可以通过以下仓库，进行R1-Zero复现能够出现



GitHub - dhcode-cpp/X-R1: minimal-cost for training 0.5B R1-Zero
github.com/dhcode-cpp/X-R1

拓展阅读

Reference

DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models

DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning

Back to Basics: Revisiting REINFORCE Style Optimization for Learning from Human Feedback in LLMs

ReMax: A Simple, Effective, and Efficient Reinforcement Learning Method for Aligning Large Language Models

Chain-of-Thought Prompting Elicits Reasoning in Large Language Models

John Schulman: Approximating KL Divergence

OpenAI's Reinforcement Fine-Tuning Research Program

git: huggingface/open-r1

git: openai/simple-evals

《手撕RLHF》解析如何系统的来做LLM对齐工程

[小冬瓜AIGC：GRPO的Loss为什么会有负值？](#)

[小冬瓜AIGC：GRPO简化Tricks, 性能暴涨10%, 只改一个参数？](#)

[小冬瓜AIGC：再深挖DeepSeek-R1: Reward is Enough](#)

[小冬瓜AIGC：【解读】DeepSeek-R1：RL前真的不需要SFT了吗???](#)

[小冬瓜AIGC：【OpenAI o3安全对齐方案】坏消息：RLHF里的HF无了!!](#)

[小冬瓜AIGC：【o1推理】Scaling LLM Test-Time：谁说类o1推理一定要用RL?!](#)

[小冬瓜AIGC：为什么DPO里Chosen和Rejected概率会同时下降???](#)

[小冬瓜AIGC：【手撕RLHF-DPO】step-by-step公式推导及实验分析](#)

[小冬瓜AIGC：【手撕RLHF-](#)

[小冬瓜AIGC：【手撕RLHF_Weak-to-Strong】OpenAI超级对齐新思路 \(含代码解析\)](#)

[小冬瓜AIGC：【手撕RLHF-Safe RLHF】带着脚镣跳舞的PPO](#)

[小冬瓜AIGC：【手撕RLHF-Rejection Sampling】如何优雅的从SFT过渡到PPO](#)

[小冬瓜AIGC：【手撕RLHF-LLaMA2】 Reward Model PyTorch实现](#)

《手撕LLM》系列文章+原创课程：LLM原理涵盖Pretrained/PEFT/RLHF/高性能计算

[小冬瓜AIGC：【手撕NSA】DeepSeek新作-原生稀疏注意力-超长文\(附代码\)](#)

[小冬瓜AIGC：【手撕DualPipe】让我们一步步把 MoE EP 通信 "消除" \(附代码\)](#)

[小冬瓜AIGC：【手撕LLM_Nv-Embed】英伟达LLM-as-Embedding, ICLR高分佳作, RAG检索有救了!!!](#)

[小冬瓜AIGC：【手撕LLM-Cut Cross Entropy】ICLR高分：LLM训练交叉熵的Memory-Efficient优化](#)

[小冬瓜AIGC：【手撕online softmax】Flash Attention前传，一撕一个不吱声](#)

[小冬瓜AIGC：【手撕LLM-FlashAttention2】只因For循环优化的太美](#)

[小冬瓜AIGC：【手撕LLM-Flash Attention】从softmax说起，保姆级超长文！！](#)

[小冬瓜AIGC：【手撕LLM】长文本的Position Encoding的衰减性证明](#)

[小冬瓜AIGC：【手撕LLM-NTK RoPE】长文本“高频外推、低频内插”从衰减性视角理解](#)

[小冬瓜AIGC：【手撕LLM - Mixtral-8x7B】Pytorch 实现](#)

[小冬瓜AIGC：【手撕LLM-Medusa】并行解码范式: 美杜莎驾到, 通通闪开！！](#)

[小冬瓜AIGC：【手撕LLM-Speculative Decoding】大模型迈向"并行"解码时代](#)

[小冬瓜AIGC：【手撕LLM-Generation】Top-K+重复性惩罚](#)

[小冬瓜AIGC：【手撕LLM-KVCache】显存刺客的前世今生--文末含代码](#)

我是小冬瓜AIGC，原创超长文知识分享，原创课程已帮助多名同学速成上岸LLM赛道。
研究方向：LLM、RL、RLHF

送礼物

还没有人送礼物，鼓励一下作者吧

所属专栏 · 2025-06-12 10:32 更新



手撕LLM

小冬瓜AIGC

36 篇内容 · 5174 赞同

订阅

最热内容 · 【手撕LLM-Flash Attention】从softmax说起，保姆级超长文！！

编辑于 2025-05-30 14:48 · 广东

LLM

大模型

DeepSeek



理性发言，友善互动

141 条评论

默认

最新



Llama跑不通

grpo如果解决通用问题，也就是不是代码题或者数学题目，这里的r应该咋设计呀，此外，A是所有的Reward的值做的加权之后的结果嘛（一堆0或者1 减去均值除方差后的结果）

02-19 · 北京

回复

4



小冬瓜AIGC

作者

用独立训练好的rewardmodel(由偏好数据训练而来)，2. 是

02-19 · 广东

回复

2



Llama跑不通

小冬瓜AIGC

相当于grpo这里也得需要有一个reward model嘛

02-19 · 北京

回复

1

展开其他 3 条回复



着急的人生

纠正下GRPO foregoes the critic model,而不是去掉了reward 模型。

02-12 · 浙江

回复

4



坐忘道

这里应该是不需要通过模型来计算reward，但是reward本身还是需要的，变成了通过规则计算，基于格式给出reward以及基于答案给出reward

02-20 · 北京

回复

2



小冬瓜AIGC

作者

坐忘道

合理，用reward function来统一定义会更好。

02-20 · 广东

回复

喜欢

展开其他 1 条回复



沐黎

请教一下大佬，grpo的adv是根据组内的平均reward计算出来的，放在rule based reward这个情况下，假如在一个sample上，模型采样的结果都是对的、都是不对的，那么单个sample减去均值之后就变成0了，adv是0的情况下，就没有训练loss了吗？我看了下ray的代码，应该是允许adv直接算出来是0的情况发生的。

03-20 · 北京

回复

2



小冬瓜AIGC

作者

沐黎

是只剩KL loss

03-20 · 美国

回复

1



沐黎

小冬瓜AIGC

感谢回复 已三连

03-20 · 北京

回复

喜欢

展开其他 1 条回复



寒山

大佬您好，想请教一下我关于KL项计算的理解对不对：GRPO的KL计算是在一个回答生成结束后发生的，一次性对句子中的每个token计算KL散度，并参与最终loss的计算；PPO的KL计算是在每个token的生成过程中发生的，不断计算当前token和ref_model的KL散度。谢谢您！

03-05 · 英国

回复

2



小冬瓜AIGC

作者

你的理解是对的，补充下PPO是逐token是可以通过forward并行算reward+KL的。区别在于：PPO的KL塞到reward里（reward shaping），GRPO的KL是独立的损失项。再从advantage角度来看，PPO的KL惩罚是token-level的，GRPO的KL惩罚是sentence-level（但是也是逐个token算kl再取mean）的

03-05 · 广东


回复 3

 **寒山** · **小冬瓜AIGC** ...

明白了，谢谢回复！

03-05 · 英国

回复 喜欢

 **雪怪**爱上火炉 ...

大佬，想请教下，在policy_logprob, ref_logprob都知道的情况下，为什么不直接计算kl散度，而要用 $(r - 1 - \log r)$ 去估计kl呢，看kl散度的计算量，似乎 $\exp^{\text{ref_logprob} * (\text{ref_logprob} - \text{policy_logprob})}$ ，也不是很大。

07-06 · 上海


回复 喜欢

 **小冬瓜AIGC** 作者 ...

可以直接计算 KL； 2. $r-1-\log r$ 方差小，无偏。3：计算量不是主因

07-07 · 美国


回复 喜欢

 **酒涤尘** ...

请教下大佬，如果前面ratio=1，adv又是r-mean算的，那无论怎么采样 Loss=sum(ratio*adv) 不都0吗？这样能训练出东西吗

05-26 · 陕西


回复 喜欢

 **小冬瓜AIGC** 作者 ...

那么ratio*adv都为0，但是grpo loss有kl约束，还是会优化的。只不过当adv全0的时候也有改进的方法来跳过优化

05-26 · 澳大利亚

回复 喜欢

 **某乎**不太行 ...

大佬，我想问一下，理论上是不是batch越大，grpo的adv的效果越好呢？

05-22 · 浙江

回复 喜欢

 **小冬瓜AIGC** 作者 ...

是

05-23 · 广东


回复 喜欢

 **jungle** ...

既然是on-policy采样，那么在5.2寿司GRPO中，为什么要分为pi_logprob, pi_old_logprob两项呢？

05-20 · 江苏

回复 喜欢

 **小冬瓜AIGC** 作者 ...

为了代码可扩展，因为还可以进一步实现多步的ratio计算。

05-21 · 广东

回复 喜欢

 **天枢未来** ...

reward_func的值可以是浮点吗

04-24 · 北京


回复 喜欢

 **小冬瓜AIGC** 作者 ...

可以。哪怕是{-1, 0, 1}，都是表示成{-1.0, 0.0, 1.0}的

04-25 · 广东

回复 喜欢

 **HelloMonica** ...

大佬，可以用GRPO来优化SFT模型的效果吗，看中了其中定义规则奖励来优化模型，不需要输出CoT，这种情况用GRPO能work嘛

04-09 · 北京

回复 喜欢

 **小冬瓜AIGC** 作者 ...

能用。输出cot的目的是为了提高预测最终答案的正确率，仍然需要采样cot的。

04-16 · 美国

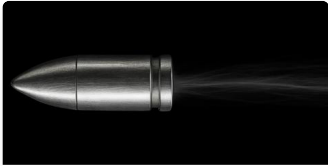
回复 1

[点击查看全部评论 >](#)



理性发言，友善互动

推荐阅读



RL 是 LLM 性能提升的“银弹”吗？

LukeL...

发表于Codin...

如何理解 LLM 中的 RL 算法？

随着最近 R1 爆火，我经常刷到一些有意思的话题，例如： SFT 无用，RL 才是通往智能化的正解； R1 并不像是传统的强化学习，更像是监督学习；这些话题，或多或少有我曾经的疑惑在里面， ...

ybq

谈LLM的数据合成与近期热议的RL范式【2024.9】

TL;DR最近由拾象科技文章讨论的RL和MCTS思路有一些用处，但它们并不是新的，也并非万能。拾象科技 的文章 LLM的范式转移：RL带来新的 Scaling Law 0、前言本文包括以下内容： LLM模型数据...

孔某人

发表于孔某人的低...



LLM Re...

Jarle...