



## PPO训练方法

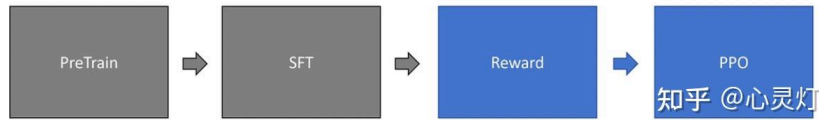


心灵的灯

关注

29 人赞同了该文章 &gt;

首先训练一个Reward模型来为模型的输出进行打分，提供奖励信号。

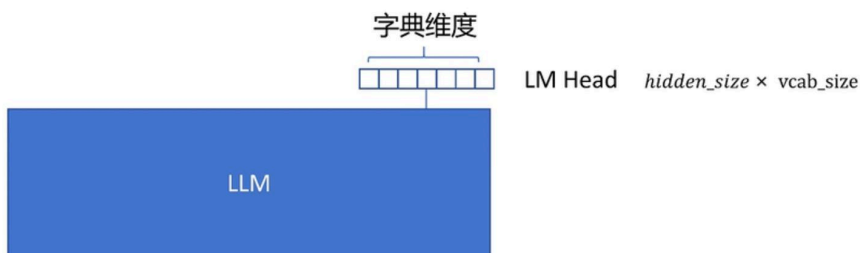


训练Reward模型需要的是偏好数据。格式如下：

```
{
  "question": "Python中的字典是什么？",
  "chosen": "Python中的字典是一种无序的可变容器，允许使用键-值对来存储数据。",
  "rejected": "Python中的字典用于存储数据。"
}
```

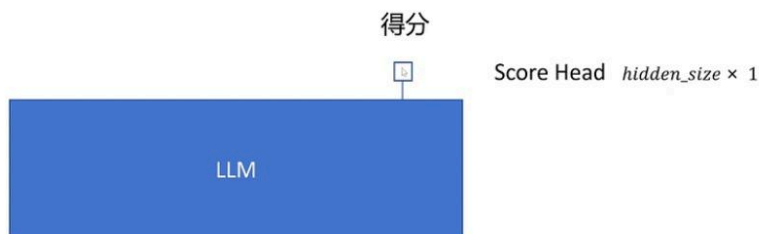
知乎 @心灵灯

当改造一个大语言模型成一个Reward model<sup>+</sup>，最后一层LM Head<sup>+</sup>（输出的维度是 vocab\_size）需要改造成Score Head（只有一维），并且只对序列的最后一个token调用score head（因为最后一个token能看到前面的所有序列，可以对整个序列进行评分，得到这个问答对的Reward值）



什么是数据库？数据库用于存储数据。

知乎 @心灵灯



什么是数据库？数据库用于存储数据。

知乎 @心灵灯

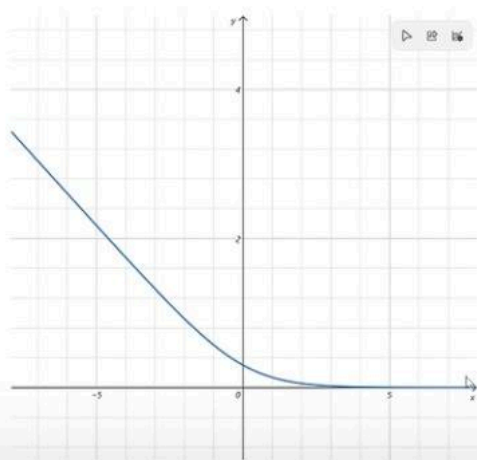
### Reward模型的Loss

需要调用两次Reward模型，得到模型对chosen和reject的得分。然后用chosen的得分减去reject的得分，输入一个sigmoid函数。



## 知乎 PPO训练方法

$$Loss = -\log\left(\frac{1}{1 + e^{-x}}\right) \quad x = chosen - rejected$$



知乎 @心灵灯

### 如何训练reward

分词，加载模型用的是一个分类模型，AutoModelForSequenceClassification，当分类（num\_labels）是1的时候就是一个回归模型，可以输出单个连续值，用量化模型。

```
import torch
from datasets import Dataset
import json

from peft import LoraConfig, TaskType, get_peft_model, prepare_model_for_kbit_training
from transformers import AutoTokenizer, BitsAndBytesConfig, AutoModelForSequenceClassification
from trl import RewardTrainer, RewardConfig

model_path = r'D:\work\models\Meta-Llama-3.1-8B-Instruct'
tokenizer = AutoTokenizer.from_pretrained(model_path, use_fast=False)
tokenizer.padding_side = "right"
tokenizer.pad_token = tokenizer.eos_token
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16
)
model = AutoModelForSequenceClassification.from_pretrained(model_path,
    num_labels=1,
    quantization_config=bnb_config)
model.config.pad_token_id = tokenizer.pad_token_id
peft_config = LoraConfig(
    r=8,
    target_modules=["q_proj",
        "v_proj",
        "k_proj",
        "o_proj",
        "gate_proj",
        "down_proj",
        "up_proj",
    ],
    task_type=TaskType.SEQ_CLS,
    lora_alpha=16,
    lora_dropout=0.05
)
```

知乎 @心灵灯

然后对数据进行处理，将问题和答案拼接起来，分词，生成一个新的数据样本，包含四项（tokenized\_chosen["input\_ids"]、tokenized\_chosen["attention\_mask"]、tokenized\_rejected["input\_ids"]、tokenized\_rejected["attention\_mask"]）。

## 知乎 PPO训练方法

```
rejected = example["question"] + example["rejected"]
```

```
tokenized_chosen = tokenizer(chosen)
tokenized_rejected = tokenizer(rejected)
```

```
new_example = {}
new_example["input_ids_chosen"] = tokenized_chosen["input_ids"]
new_example["attention_mask_chosen"] = tokenized_chosen["attention_mask"]
new_example["input_ids_rejected"] = tokenized_rejected["input_ids"]
new_example["attention_mask_rejected"] = tokenized_rejected["attention_mask"]
return new_example
```

```
dataset = dataset.map(process_func, remove_columns=['question', 'chosen', 'rejected'])
print(dataset)
```

```
config = RewardConfig(output_dir="./reward_model")
config.num_train_epochs = 1
config.per_device_train_batch_size = 1
```

知乎 @心灵灯

### 训练PPO+

训练PPO需要四个模型，

- 一个是基准模型（sft之后的大模型，新训练的模型不能和基础模型相差太大）
- 第二个训练模型（PPO训练的目标是优化训练模型，同时训练模型不能和基准模型相差太大）
- 第三个是Reward model（对问答序列进行评分，输出一个分数），
- 第四个是状态价值模型，对每个状态评估它的价值（即计算当前的token到预测序列结束之后，这个问答序列的期望回报是多少）。
- 这四个模型除了最后一层的head不同，底层都是大模型。

HS:hidden\_size  
VS:vocab\_size

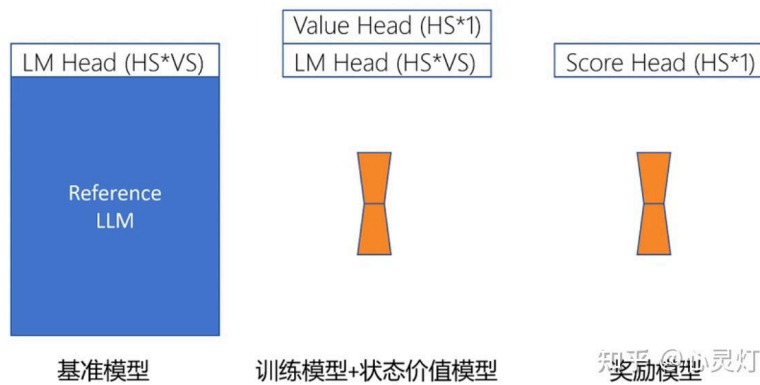
2v4indFan bilibili



同时加载四个大模型，显存就太大了。

这里可以加载一个大模型，多个adpater的技术，大大减少对显存的占用，并且将训练模型和状态价值模型合并，共用一套adpater参数，它同时有两个头，分别是LLM Head和 Value Head。这样就可以只加载一个大模型和两个Lora 参数的adpater来训练PPO。

## 知乎 PPO训练方法



对于大模型输出的每一步而言，

state就是截止到当前token的序列

action就是输出下一步的token

Reward是根据完整输出给出一个得分，也就是只给最后一个token一个得分，其他token为0。

可以给每个token增加一个奖惩项，使得输出的概率分布和基准模型的KL散度<sup>+</sup>相关。

针对每个当前token，大模型会通过LLM-head会输出字典里每个token作为输出的概率。训练模型的action分布如果和基准模型不一致，就会受到惩罚，分布越不一致，惩罚越大。这里如果把调整系数设置为0.2，训练模型输出每个action的  $\text{Reward} = \text{训练模型输出的概率分布相对于基准模型输出的概率分布的KL散度} * (-0.2) + \text{score}$

Reward										
Prompt+Response	天空	为什么	是	蓝色	？	因为	光	的	散射	。
训练模型	[0.11,0.02,0.0 2.0.32,0.04...]	[0.11,0.02,0.0 2.0.32,0.04...]	[0.11,0.02,0.0 2.0.32,0.04...]	[0.11,0.02,0.0 2.0.32,0.04...]	[0.11,0.02,0.0 2.0.32,0.04...]	[0.11,0.02,0.0 2.0.32,0.04...]	[0.11,0.02,0.0 2.0.32,0.04...]	[0.11,0.02,0.0 2.0.32,0.04...]	[0.11,0.02,0.0 2.0.32,0.04...]	[0.11,0.02,0.0 2.0.32,0.04...]
基准模型	[0.11,0.02,0.0 2.0.32,0.04...]	[0.11,0.02,0.0 2.0.32,0.04...]	[0.11,0.02,0.0 2.0.32,0.04...]	[0.11,0.02,0.0 2.0.32,0.04...]	[0.11,0.02,0.0 2.0.32,0.04...]	[0.11,0.02,0.0 2.0.32,0.04...]	[0.11,0.02,0.0 2.0.32,0.04...]	[0.11,0.02,0.0 2.0.32,0.04...]	[0.11,0.02,0.0 2.0.32,0.04...]	[0.11,0.02,0.0 2.0.32,0.04...]
KL	1.2	0.5	2.3	3.6	4.8	1.5	4.5	1.0	5.3	2.1
Score	0	0	0	0	0	0	0	0	0	3.8
Reward	-0.24	-0.1	-0.46	-0.72	-0.96	-0.3	-0.9	-0.2	0.106	3.38

GAE<sup>+</sup>优势函数怎么计算，GAE可以迭代进行计算，从后往前算。

# 知乎 PPO训练方法

$$\delta_t^V = r_t + \gamma * V_{\theta}(s_{t+1}) - V_{\theta}(s_t)$$

$$A_{\theta}^{GAE}(s_t, a) = \sum_{b=0}^{\infty} (\gamma \lambda)^b \delta_{t+b}^V$$

$$A_{\theta}^{GAE}(s_t, a) = \delta_t^V + \gamma \lambda A_{\theta}^{GAE}(s_{t+1}, a)$$

```
for t in reversed(range(gen_len)):
    nextvalues = values[:, t + 1] if t < gen_len - 1 else 0.0
    delta = rewards[:, t] + self.config.gamma * nextvalues - values[:, t]
    lastgaelam = delta + self.config.gamma * self.config.lam * lastgaelam
    advantages_reversed.append(lastgaelam)
    advantages = torch.stack(advantages_reversed[::-1]).transpose(0, 1)
```

知乎 @心灵灯

要知道我们是在训练两个模型一个是输出文本token，一个是输出每个token状态价值。

要计算状态价值的loss，首先需要知道状态价值的label是什么。生成状态价值的label有三种：

- 1、蒙特卡洛法
- 2、时序差分法
- 3、广义优势法（考虑了一步采样多步采样，平衡了偏差和方差）

## Loss

State Value Loss

Prompt+ Response	天空	为什么	是	蓝色	?	因为	光	的	散射	。
Reward	-0.24	-0.1	-0.46	-0.72	-0.96	-0.3	-0.9	-0.2	-0.106	3.38
State Value Predicted	-0.84	-0.91	-0.8	-0.76	-0.58	0.1	0.3	1.2	2.2	3.25

### Value Head Label:

蒙特卡洛法:  $V_{label}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^{T-t} r_T$  方差太大

时序差分法:  $V_{label}(s_t) = r_t + \gamma V(s_{t+1})$  偏差太大

广义优势法:  $V_{label}(s_t) = A_t^{GAE} + V(s_t)$  平衡方差和偏差

知乎 @心灵灯

代码:

advantages + values 作为label,

为了训练的稳定引入了一个clip的值，预测状态价值不能和重要性采样网络预测的状态价值变化太大，必须在一个范围内，超过这个范围就截断。loss1和loss2按位取最大。

```
returns = advantages + values
vpredclipped = clip_by_value(
    vpreds,
    values - self.config.cliprange_value,
    values + self.config.cliprange_value,
)

vf_losses1 = (vpreds - returns) ** 2
vf_losses2 = (vpredclipped - returns) ** 2
vf_loss = 0.5 * masked_mean(torch.max(vf_losses1, vf_losses2), mask)
```

知乎 @心灵灯

然后讨论一下PPO的loss

在实际的实现代码有两点不同：

# 知乎 PPO训练方法

2.KL散度已经在Reward中体现，已经通过Reward进入了Loss函数。

首先计算训练网络与重要性网络输出的比值ratio

然后ratio GAE 为第一种ppo\_loss, 为了训练未定，控制训练网络与重要性网络输出的比值在一定的范围内，有了clip版本的第一种loss，这两种loss取最大之后取均值，作为pg\_loss, pg\_loss + 调节系数 \* 状态价值的loss 作为最终loss。

## Loss

PPO Loss

$$Loss_{ppo} = -\frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} A_{\theta}^{GAE}(s_n^t, a_n^t) \frac{P_{\theta}(a_n^t | s_n^t)}{P_{\theta'}(a_n^t | s_n^t)} + \beta KL(P_{\theta}, P_{\theta'})$$

1. 进行重要性采样的模型可以与参考模型不一样。

2. KL散度已经在Reward中体现，已经通过Reward进入了Loss函数。

$\theta$  : 训练模型

$\theta'$  : 基准模型

$\theta''$  : 重要性采样模型，  
训练过程中逐步更新。  
始终保持和训练模型相差不大。

$$Loss_{ppo} = -\frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} A_{\theta}^{GAE}(s_n^t, a_n^t) \frac{P_{\theta}(a_n^t | s_n^t)}{P_{\theta'}(a_n^t | s_n^t)}$$

```
ratio = torch.exp(logprobs - old_logprobs)
```

```
pg_losses = -advantages * ratio
```

```
pg_losses2 = -advantages * torch.clamp(ratio, 1.0 - self.config.cliprange, 1.0 + self.config.cliprange)
```

```
pg_loss = masked_mean(torch.max(pg_losses, pg_losses2), mask)
```

```
loss = pg_loss + self.config.vf_coef * vf_loss
```

总的loss包含PPO loss 和 State Value loss

知乎 @心灵灯

## PPO训练循环

首先从prompt数据集中取出一个batch的prompt进行训练，然后利用重要性采样网络也就是训练网络进行回答的生成 (active\_model.generate), 然后把问题和回答合并生成训练文本 (batch\_data)，利用reward\_model模型对训练文本进行打分 (batch\_scores)

接着利用重要性采样网络计算针对整个字典所有tokens的概率分布 (batch\_all\_probs) 和作为输出token的概率 (batch\_probs) 以及每个token的状态价值 (batch\_all\_values)。

然后计算基准模型的ref\_all\_probs、ref\_probs、ref\_all\_values

然后计算KL散度，rewards，advantages，returns。

returns = advantages + batch\_all\_values(重要性采样的状态价值)

```
for batch_prompt in prompt_dataset:
    batch_response = active_model.generate(batch_prompt)
    batch_data = concat(batch_prompt, batch_response)
    batch_scores = reward_model(batch_data)

    batch_all_probs, batch_probs, batch_all_values = active_model.forward_pass(batch_data)
    ref_all_probs, ref_probs, ref_all_values = ref_model.forward_pass(batch_data)
    kls = compute_KL(batch_all_probs, ref_all_probs)
    rewards = compute_rewards(batch_scores, kls)
    advantages = compute_advantages(batch_all_values, rewards)
    returns = advantages + batch_all_values

    for i in range(epoch):
        active_all_probs, active_probs, active_all_values = active_model.forward_pass(batch_data)

        loss_state_value = torch.mean((returns - active_all_values) ** 2)
        ratio = active_probs / batch_probs
        loss_ppo = torch.mean(-advantages * ratio)
        loss = loss_ppo + value_loss_rate * loss_state_value
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
```

知乎 @心灵灯

每一个epoch，都用训练网络生成active\_all\_probs，active\_probs，active\_all\_values。然后计算状态价值的loss，loss\_state\_value = returns(状态价值的label) - 状态价值。

训练网络和重要性采样网络在每个token的比值，比值 (ratio) \* advantages 就是ppo的loss (loss\_ppo)

最后用ppo的loss + 状态价值的loss



## 知乎 PPO训练方法

总结：外层是在采样——>计算advantages——>利用advantages训练epoch次模型

如何利用trl库+来训练模型

首先看下训练数据，训练数据只需要query即可，不需要回答，回答是训练的时候生成的。

```
{
  "query": "请给出保持健康的三个方法。"
}
```

知乎 @心灵灯

首先定义Lora，任务类型为CAUSAL\_LM，模型类型为AutoModelForCausalLMWithValueHead，model\_path是基准模型，制定reward\_adapter共享基准模型的权重，生成reward模型，会根据peft\_config基于基准模型生成一个训练模型和状态价值模型合并的模型，最后是量化参数（quantization\_config）。

```
peft_config = LoraConfig(
    r=8,
    target_modules=["q_proj",
                    "v_proj",
                    "k_proj",
                    "o_proj",
                    "gate_proj",
                    "down_proj",
                    "up_proj"
    ],
    task_type=TaskType.CAUSAL_LM,
    lora_alpha=16,
    lora_dropout=0.05
)

model = AutoModelForCausalLMWithValueHead.from_pretrained(model_path,
    reward_adapter="./reward_model",
    peft_config=peft_config,
    quantization_config=bnb_config
)

model.to("cuda")
```

知乎 @心灵灯

然后加载数据，对数据tokenizer

```
items = []
with open("./data/queries.json", "r", encoding="utf8") as f:
    for line in f:
        items.append(json.loads(line))
queries_dataset = Dataset.from_list(items)

def collator(data):
    queries = []
    for item in data:
        queries.append(tokenizer(item["query"], return_tensors="pt")["input_ids"].squeeze())
    return queries
```

知乎 @心灵灯

接下来配置ppo\_config，如果kl\_penalty="full"表示标准的KL散度。ppo\_trainer中ref\_model不指定的话，用model的基准模型作为基准模型

```
ppo_config = PPOConfig(kl_penalty="full", ppo_epochs=3, batch_size=2, mini_batch_size=1)
ppo_trainer = PPOTrainer(config=ppo_config, model=model, ref_model=None, tokenizer=tokenizer, dataset=queries_dataset,
    data_collator=collator)
```

然后指定大模型的生成参数，训练时要在所有的概率空间进行采样，所以不设置top\_k，top\_p必须为1，do\_sample为True进行训练。

## 知乎 PPO训练方法

```
    "min_length": -1,  
    "top_k": 0.0,  
    "top_p": 1.0,  
    "do_sample": True,  
    "pad_token_id": tokenizer.pad_token_id,  
    "max_new_tokens": 32,  
}
```

知乎 @心灵灯

首先取一个batch的数据，batch的数据中只有问题没有回答，用ppo\_trainer来生成回答，接着用reward模型对回答进行打分，最重要的一步用ppo\_trainer的step方法，传入问题，回答，得分即可

```
for batch in ppo_trainer.dataloader:  
    query_tensors = batch  
  
    response_tensors = ppo_trainer.generate(  
        query_tensors, return_prompt=False, **generation_kwargs)  
    scores = []  
    for query, response in zip(query_tensors, response_tensors):  
        input_ids = torch.concat([query, response], dim=0)  
        input_ids = torch.unsqueeze(input_ids, dim=0)  
        score = ppo_trainer.model.compute_reward_score(input_ids=input_ids)[0],  
        scores.append(score)  
    stats = ppo_trainer.step(query_tensors, response_tensors, scores)  
ppo_trainer.save_pretrained("./rl_model")
```

知乎 @心灵灯

编辑于 2024-12-23 23:32 · 北京

PPO算法



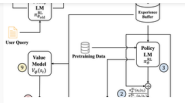
发首评



还没有评论，发表第一个评论吧

推荐阅读





**【RLHF】怎样让 PPO 训练更稳定？早期人类征服 RLHF ...**

何枝      发表于何小枝与N...

1.主要内容参考自：图解大模型 RLHF系列之：人人都能看懂的PPO 原理与源码解读,如标题所示，没有抄袭，只是作者自己太菜，作一下注释。2.以下内容主要是标明细节：2.1 代码来自：...

水样浓烈

Published October 24, 2023

update on GitHub

Shengyi Costa Huang   Tianlin Liu   Leandro von Werra

**解读RLHF-PPO实现细节及代码复现**

时空猫的问...   发表于深度强化学...