Lora 微调 Yi-9B 模型增强模型购物概念理解能力

陈鸿绪–PB21000224　黄祈铭–PB21000196　林博文–PB20000068

2024 年 6 月 28 日

一、摘要

本次实验的选题为”Amazon KDD Cup 24: Understanding Shopping Concepts”，我们通过微调大语言模型强化模型在多任务下购物概念理解能力。本次微调模型采用 Yi-9B 模型，具体分为两阶段进行微调训练：分别为 Post Pre-tran(即在预训练模型的基础上进行特定领域知识的学习)，与 Supervised fine-tuning(对于特定下游任务进行有监督微调训练) 得到最终的模型。训练配置为 6 张 3090 显卡，采用 LLaMA-factory 微调框架进行分布式微调。微调后的模型开源地址：模型 URL

这次实验我们也花费很多精力在微调数据集生成、收集和清洗上，以实现高质量、多种类的微调数据。数据集类型分为互联网收集的数据和 GPT 生成的数据，Post Pre-traning 阶段的微调数据集构成为亚马逊商品信息数据和消费评论等，共计 25M，SFT 阶段的微调数据集构成为 GPT 生成特定数据和 Shoppingai 任务数据集，共计 3M。数据集开源地址：SFT 数据集; Post Pre-traning 数据集。

最后在六种不同类型的任务上的测试结果表明，微调模型显著优于预训练模型，且相应指标也较高 (具体测试结果见后续结果分析)。我们的项目开源在：项目地址。分工见报告附录，项目简要说明请见 README.md。

二、实验过程

1.1 模型选择

在国产中小型 LLM 中 Yi-9B 模型表现优异，全面综合能力（Mean-All）强劲，同时在常识和推理能力方面（Mean-Text）也达到了同类模型的前列。模型既提供了浮点数版本 (FP16)，也提供了 int8 量化版本，使其能够轻松部署在包括 RTX 4090 和 RTX 3090 在内的消费级显卡上，大大降低了使用门槛和成本。为了得到最佳的模型性能，我们在实验中加载训练模型并没有采用量化版本，而是使用全精度加载模型参数，fp16 进行计算。

1.2 数据集构建

这部分我们会构建我们自己的微调数据集。共有两类数据集，一类为 Post Pre-traning 数据集，另一类为 SFT 数据集。下面是训练数据集语料、任务的具体组成表格，其中 GPT 生成数据由 chatglm,

Ernie40, qwen 三个大模型生成，随机种子来源于 Development 数据集。实验中使用的现成数据集均可以在 Huggging Face 上下载。

表 1: **构建的训练数据组成**

	Post Pre-traning Data	SFT Data
Composition	Amazon product description, Red Dot Design Award product description, Shopping Review	GPT generate data, Develpoment data, Shoppingai data
total	25MB,55393	3MB,8851

同时我们也基于规则选取或人工筛选标注了六种不同任务类型的测试数据集，测试数据集基本信息为下：

表 2: **六种不同类型任务测试集基本信息**

CLS	NRE	generation	multi-choice	ranking	retrieval
7kb,58	97kb,277	177kb,312	68kb,171	132kb,248	129kb,241

在以上构建数据集的同时，我们进行了大量的数据清洗工作：

- 数据去重：这部分主要是在关心数据集中是否会存在内容高度重合的数据，例如在 GPT 生成数据时可能会由于随机种子类似导致输出内容相似，这时我们会采用 Rouge 分数去衡量生成数据是否和之前某一条数据高度相似 (设置阈值为 0.7)，若大于阈值则丢弃。
- 数据去空：对于互联网上现成的一些数据有些必要字段为空，或者由于 GPT 输出不稳定导致没有答案。此时就需要我们将这些必要字段为空的数据进行鉴别与去除。
- 剔除低质量数据：这里由于数据量过大，我们只能采取抽样的方式进行数据质量人工检验，检验主要关注数据的问题合理性、答案的正确性等，将不符合以上条件的数据进行去除。

1.3 Post Pre-traning 分布式微调

这部分我们使用预训练的原始模型通过上述构造好的 Post Pre-traning 数据集再次进行预训练，目的是为了让模型可以在原本已掌握的知识上获取到更多关于购物概念理解的基本知识。该部分的微调基本参数设置参见后续的代码讲解。训练配置为 6 卡 3090，训练时长大致在 2-3h。

1.4 SFT 分布式微调

这部分我们使用上面已经学到更多关于购物概念理解的微调模型再次进行有监督微调，由于我们的 SFT 数据集包含了近 9000 条多种类高质量的关于购物概念理解的任务，模型性能在这些具体的下游任务上获得相应的提升。该部分的微调基本参数设置参见后续的代码讲解。训练配置为 6 卡 3090，训练时长大致在 4-5h。

1.5 推理与模型测试

这部分我们将模型合并导出后，在六种不同任务类型 (CLS, NRE, ranking, generation, multi-choice, retrieval) 上进行评估测试，评估测试指标与实验要求一致。测试时，在得到微调模型的输出后，我们需要进行答案析取操作，这部分主要依靠基于规则的答案析取与人工手动析取。同时对原预训练模型也需要进行同样操作。最后两者进行指标比较。推理配置为 2 卡 3090，单样本推理时长在 4s 左右。

三、项目模块简介

这部分在项目 README.md 亦可见，主要介绍项目各个部分组成与用处。

- bash：存放 linux 系统实验过程中训练、测试、导出模型的 sh 脚本。
- data：存放 post-pre-train 数据集 (三个不同语料数据的下载地址、合并之后最终数据集)，sft 数据集 (不同 GPT 生成数据集代码、shoppingai 数据集、合并之后最终数据集)，测试数据集 test-data（包括 6 种不同大类型任务）。
- fig：训练时的测试集与验证集 loss 图像。
- LLaMA-Factory：微调框架，主要工作区。
- report：存放报告
- slurm-out：训练、导出操作提交作业后的计算节点输出
- test-result：测试输出结果处理，其中有两个子文件夹，分别存储原模型和微调后模型在六种不同类型的任务上的测试输出与相应指标。其中 metric.txt 会存放模型在测试集的评测指标。

四、Lora 微调原理

Lora 是一种微调预训练语言模型的方法，它通过在原始模型的基础上添加一个低秩矩阵来引入一些新的参数，以便更好地适应特定的任务或领域。与传统的微调方法相比，Lora 具有参数效率高、训练成本低等优点。在 Lora 方法中，原始模型的参数被分为两部分：一部分是基础模型参数，另一部分是新引入的低秩矩阵。在微调过程中，我们只更新低秩矩阵的参数，而保持基础模型参数不变。这样，我们就可以通过较小的参数更新来达到较好的适应效果。在本次实验中，我们采取 lora 的参数具体为如下：

- lora_target q_proj,v_proj
- lora_dropout 0.08
- lora_rank 4
- lora_alpha 16

其中 lora_target 参数代表需要 lora 适配器加上的目标全连接层 (矩阵)，上面的脚本表示：lora 适配器加载在模型中 transformer 模块 q,k,v 矩阵上。lora_rank 参数代表 lora 适配器加上的矩阵的秩，秩越高，添加的参数就越多，模型的复杂度也越高。因此，选择合适的秩对于 Lora 方法的效果至关重要，该脚本中认定秩为 4。lora_dropout 在 lora 适配器中起着 dropout 作用，该参数起着正则化的作用，加强模型的泛化性能，我们设置大小为 0.08。lora_alpha 参数定义了 LoRA 适应的学习率缩放因子，这个参数影响了低秩矩阵的更新速度。

五、实验关键代码讲解

1.1 Post Pre-traning 微淘

```
#!/bin/bash

CUDA_VISIBLE_DEVICES=0,1,2,3,4,5 accelerate launch --main_process_port 8888 \
--config_file LLaMA-Factory/LLaMA-Factory-main/examples/accelerate/master_config
    .yaml \
LLaMA-Factory/LLaMA-Factory-main/src/train_bash.py \
--stage pt \
--do_train True \
--model_name_or_path Yi-9B \
--dataset post-pre-train-data \
--dataset_dir LLaMA-Factory/LLaMA-Factory-main/data \
--template default \
--finetuning_type lora \
--lora_target q_proj,v_proj \
--lora_dropout 0.08 \
--lora_rank 4 \
--output_dir LLaMA-Factory/LLaMA-Factory-main/saves/Yi-9B/post-pt-1 \
--overwrite_output_dir True \
--cutoff_len 1024 \
--preprocessing_num_workers 6 \
--per_device_train_batch_size 1\
--per_device_eval_batch_size 4 \
--gradient_accumulation_steps 2 \
--lr_scheduler_type cosine \
--logging_steps 10 \
--warmup_steps 20 \
--save_steps 150 \
--eval_steps 100 \
--evaluation_strategy steps \
--learning_rate 2e-5 \
--num_train_epochs 5 \
--max_samples 160000 \
--val_size 0.05 \
--ddp_timeout 1800000 \
```

```
    --plot_loss True\
    --fp16
```

Listing 1: Post Pre-traning 训练脚本

我们挑出其中较为重要的几个参数，lora 的基本参数在上面部分已经详细阐述，这里不再赘述。其它的参数是深度学习常采用的一些平凡参数。对于其他 sh 脚本参数设置类型，基本上与 train_sh.sh 一致。CUDA_VISIBLE_DEVICES=0,1,2,3,4,5 accelerate launch 表示分布式在 6 个 GPU 设备上，采用 accelerate 分布式框架；stage 参数表示训练方式采取 pre-train 训练微调；cutoff_len 表示训练输入 token 如果过长采取截断措施 (超过 1024 token 截断)；preprocessing_num_worker 代表训练并行进程数 (对应于分布式进程)；per_device_train_batch_size 与 per_device_eval_batch_size 为训练、验证时分别跑在 GPU 上的 batch size 个数；lr_scheduler_type 表示学习率调节器，这里采取了 cosine 调节器；fp16 表示在计算的时候采取混合精度。

对于 SFT 训练代码其实与之类似，只不过 stage 会改成 sft 模式。

1.2 调用 LLM 生成数据集部分代码

```
    def split_string(gpt_output: str):
        parts = gpt_output.split('###')
        return parts

    def delete_repeat(threshold, dataset_list: list, tar):
        if len(dataset_list) == 0:
            return True
        for item in dataset_list:
            rouge = Rouge()
            scores = rouge.get_scores([item], [tar])
            if scores[0]['rouge-l']['f'] > threshold:
                return False
        return True

    flag = 0
    final_seed_len = 5
    generate_time = 1
    num_list = list(range(len(df_seed)))
    selected_set = []
    prefix_prompt ="""
    You are asked to come up with a set of 5 diverse tasks. These task instructions
        will be given to a GPT model and we will evaluate the GPT model for
        completing the instructions.

    Here are the requirements:
    0. The theme of this task is about Shopping Concept Understanding, aims to
        evaluate the model's ability to understand shopping entities in the form of
        texts, such as product names, product categories, attributes, product
        descriptions, reviews, etc
    1. Try not to repeat the verb for each instruction to maximize diversity.
```

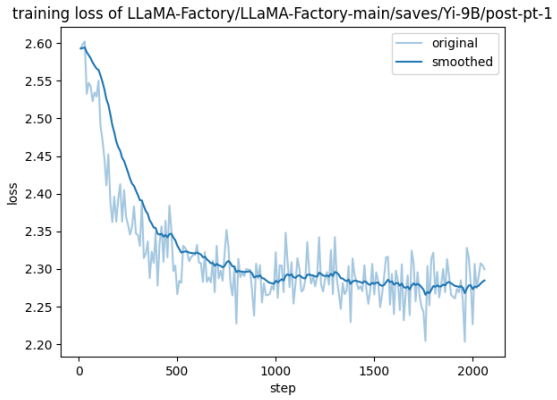
```
        2. The way of expressing words in the input_field needs to be diversified.
        3. In order to make the type of task be diverse, trying to generate task types
            different from the examples. The list should include diverse types of tasks
            like open-ended generation, classification, ranking, multi-choice etc.
        4. A GPT language model should be able to complete the output_field based on the
             input_field.
        5. The output_field should be an appropriate and correct response to the
            input_field. Make sure the output_field is less than 100 words.
        6. Each task needs to be enclosed with “###” just like the examples. Note that
             the task type should be at the beginning of the example.
        7. Input_field must end with 'Answer ' or 'Output: '. Make sure the input_field
            is more than 200 words!!!
        List of 5 tasks:
        """

        prompt_gpt_input_list = []

        for i in range(generate_time):
            while True:
                selected_numbers = set(random.sample(num_list, final_seed_len))
                if selected_numbers not in selected_set:
                    selected_set.append(selected_numbers)
                    break

        for set_item in selected_set[-generate_time:]:
            df_final_seed = pd.DataFrame([], columns = ["input_field", "output_field", "
                task_name", "task_type", "metric", "is_multiple_choice"])
            item_list = ""
            for j in set_item:
                df_final_seed.loc[len(df_final_seed)] = df_seed.loc[j]
            for j in range(len(df_final_seed)):
                temp_string = '\n###\n task_type:' + df_final_seed['task_type'][j] + "\n
                     input_field: " + str(df_final_seed['input_field'][j]) + '\n
                    output_field: ' + str(df_final_seed['output_field'][j]) + '\n'
                item_list += temp_string
            prompt_gpt_input = prefix_prompt + item_list
            prompt_gpt_input_list.append(prompt_gpt_input)
```

Listing 2: GPT 生成数据集

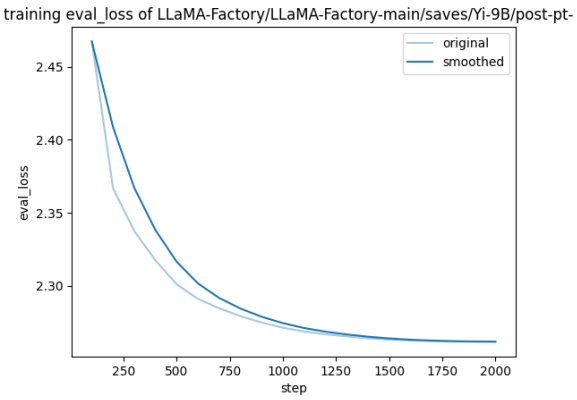
以上代码主要是通过随机抽取 develop 数据集的若干条数据，然后通过实验构造出较好的 prompt，让大语言模型输出多样化、高质量、关于购物概念理解、并且符合我们划分任务格式的文本，这时我们可以使用 split_string 函数对大模型的输出进行分割，利用 delete_repeat 函数进行 Rouge 分数的计算并进行数据去重。对于数据发送给大模型 API 的代码部分具体见代码文件 generate_new_dataset.ipynb，采用的发送方式为异步并发请求，由于每个大模型 API 代码不一样，所以不同大模型异步发送请求的代码部分也会不一致。

六、实验结果分析

1.1 训练损失图像



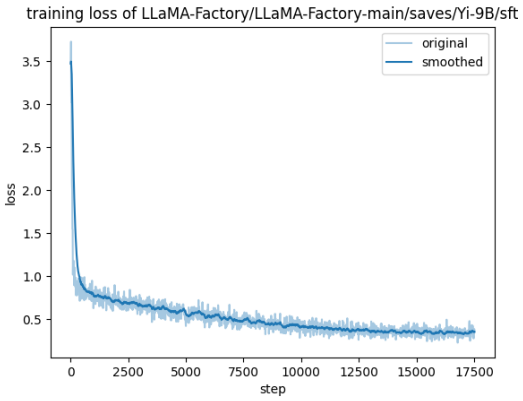
(a) post pt 训练的损失曲线



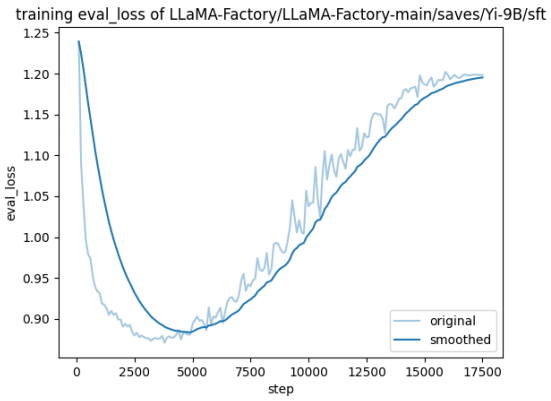
(b) post pt 验证集损失曲线

图 1: Post Pre-traning 损失曲线情况

我们可以看到做 Post Pre-traning 时训练集上的 loss 是为减少趋势的，最后趋于平缓，但验证集上的 loss 一直保持减少趋势，所以可以得出模型的性能是一直在被优化的，没有出现过拟合、欠拟合现象。



(a) sft 训练的损失曲线



(b) sft 验证集损失曲线

图 2: SFT 损失曲线情况

我们可以看到做 SFT 时训练集上的 loss 是为减少趋势的，最后趋于平缓，但验证集上的 loss 先降后升，大致在 3000-4000 范围内模型在验证集上达到最优性能，之后就开始过拟合，为了避免过拟合情况的发生，我们选取 3750 时的 checkpoint 文件，将模型导出作为最终微调完毕的模型。

1.2 测试结果

在对微调后模型和原模型进行 6 中不同种类任务数据集上的测试后，我们得到下表。(注：在下表中，原模型对 ranking 数据集问题输出格式及其不稳定，答案析取无论是人工还是基于规则难度都极大，费时费力，所以用'-' 表示不参与，原模型在 NRE 上的表现输出非常糟糕，即使有极少答对其输出格式也不稳定 (可以从输出文件看出)，所以可以近似为 0.00 的 Micro-F1。)

表 3: 微调模型在各类任务测试上的指标展示

	CLS	NRE	ranking	generation		multi-choice	retrieval
Metric	ACC	Micro-F1	NDCG@4	bleu-4	rouge-l	ACC	Hit@3
Yi-9B	15.63%	0.00	—	13.03	10.55	34.66%	1.028
Fine-tuned Yi-9B	75.00%	0.73	1.082	40.59	36.24	80.11%	2.271

对于除了 ranking 任务以外的数据集，我们得到的结果可以发现 Classification 任务的 ACC 指标涨点近 60%，NRE 任务 NDCG 指标涨点约 0.73，对于 generation 的任务 bleu-4 上涨近 27，rouge-l 上涨近 26，multi-choice 的 ACC 提高了近 56%，retrieval 任务的 Hit@3 指标多出了近 1.24。所以几乎所有任务相对于原模型都有显著的提升，而且对我们微调模型的输出进行人工检查，其实主观效果已经非常不错了。所以我们认定我们微调后的模型在理解购物概念的任务上的表现是比较卓越的。

七、任务分工

陈鸿绪：负责 GPT 生成数据异步并发实现、构建微调数据集、数据处理、模型训练微调与测试、项目构建、实验报告撰写

黄祈铭：负责生成多任务 GPT 数据、构建微调数据集、数据处理、实验报告检查

林博文：构建微调数据集、数据处理、实验报告撰写