残差卷积神经网络实现 cifar-10 数据集分类

陈鸿绪　少年班学院　PB21000224

2024 年 5 月 31 日

摘要

本次实验任务是微调大语言模型完成文本分类任务，本次微调模型采用 ChatGLM-6B-base 模型，微调数据集为 yelp_review_full 数据集，微调框架采取 LLaMA-Factory，微调方式为 Low-Rank Adaptation(Lora)，其中 lora 参数采取：lora_dropout 0.08, lora_rank 8, alpha 16。在设计 prompt、处理数据集后，笔者将其在 4 卡 3090 上进行了微调，微调数据量为 160k 条，其中验证集比例为 0.06，最后经过大约 9 个小时的训练得到 lora 适配器，最后微调的模型 F1-score 分数比原模型高出将近 0.3，所以我们认定微调对模型性能的提升是显著的。模型放置在 Hugging Face 上。

一、实验过程

1.1 设计 prompt 并处理 yelp_review_full 数据集

- 选择数据集为 Yelp-Full 数据集。设计 prompt，并根据原模型的输出进行改善微调。
- 划分数据集，这里随机均匀划分训练集和测试集，训练集中已经包含验证集 (因为在训练时，可以选定验证集的占比)，验证集占比 0.06。将数据集按照框架要求组织成 Supervised Fine-Tuning(SFT) 数据集格式，对应信息写入 data_info.json 文件中。

1.2 分布式 SFT Lora 微调

- 构建训练脚本 train_sh.sh，采取分布式训练，训练方式为 SFT Lora 微调，具体参数在后面的部分详释。
- 将模型、脚本导入节点，提交作业进行训练。训练时长大约 9 小时，4 核 GPU 占用大约都在 80-90%。

1.3 导出模型与微调前后模型性能测试

- 训练完毕，得到 loss 图像，利用 export_model.sh 脚本导出模型并本地加载模型查看效果。

- 分别构建训练前后的模型测试脚本：test-origin.sh, test-finetuned.sh。两者的推理时间大约为 1.5 个小时。得到推理输出，由于内置的评测指标并没有 F1-score，所以需要单独处理输出进行 F1-score 的计算。
- 对于微调后的模型，输出严格按照 prompt 进行，所以只需要将输出的一个数字提取即可。然而对于微调前的模型，对训练的 prompt 输出回答格式不稳定，所以采取基于规则式的答案提取，对于个别没有给出答案的测试例采取舍弃措施。最后进行两者的 F1-score 的计算。

二、实验详情

2.1 数据集构建

采用五分制数据集 yelp_review_full HuggingFace 数据集，该数据集训练集 650k 条，测试集 50k 条，由于数据量过大，所以采取随机均匀采样，训练条数大约在 160k 条数，验证集条数为 0.06×160k 条，推理用的测试集采用完整的 50k 条。

2.2 模型选择

国产开源模型中小型 LLM 中 ChatGLM-6B-base 模型表现较为优异，ChatGLM3-6B 的基础模型 ChatGLM3-6B-Base 采用了更多样的训练数据、更充分的训练步数和更合理的训练策略。在语义、数学、推理、代码、知识等不同角度的数据集上测评显示，ChatGLM3-6B-Base 具有在 10B 以下的预训练模型中最强的性能，所以本次实验采取该预训练模型。

2.3 Prompt 设计

在训练之前，由于我们并不知道采用什么形式的 prompt 可以达到较好的效果，所以只能在原模型上进行手动调整设计，设计原则遵循下面三条规则：

- Prompt 需要明确需要让模型完成的任务。
- Prompt 设计需要让模型的输出尽可能接近理想格式和合理答案。
- Prompt 设计需要在保证以上原则下尽可能减少 token 数。

最后经过多次实验调整，采取如下格式的 prompt:

> Instruction: The following is a user's review of a merchant. Please judge and output the corresponding five-star rating for this review.
> Review: {review content}
> Output: {0-4}

2.4 微调方法讲解

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，LOLA 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 query_key_value
- lora_dropout 0.08
- lora_rank 8
- 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 8。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。

三、关键代码讲解

3.1 训练脚本：train_sh.sh

```
#!/bin/bash

CUDA_VISIBLE_DEVICES=0,1,2,3 accelerate launch \
    --config_file LLaMA-Factory/LLaMA-Factory-main/examples/accelerate/master_config.yaml \
    LLaMA-Factory/LLaMA-Factory-main/src/train_bash.py \
    --stage sft \
    --do_train True \
    --model_name_or_path chatglm3-6b-base-model/models--THUDM--chatglm3-6b-base/snapshots/f91a1de587fdc692073367198e65369669a0b49d \
    --dataset yelp_review_full_train \
    --dataset_dir LLaMA-Factory/LLaMA-Factory-main/data \
    --template default \
    --finetuning_type lora \
    --lora_target query_key_value \
    --lora_dropout 0.08 \
```

```
        --lora_rank 8 \
        --output_dir LLaMA-Factory/LLaMA-Factory-main/saves/Chatglm-6B-base/lora/sft_1 \
        --overwrite_output_dir True \
        --cutoff_len 1024 \
        --preprocessing_num_workers 4 \
        --per_device_train_batch_size 4 \
        --per_device_eval_batch_size 4 \
        --gradient_accumulation_steps 2 \
        --lr_scheduler_type cosine \
        --logging_steps 10 \
        --warmup_steps 20 \
        --save_steps 1000 \
        --eval_steps 500 \
        --evaluation_strategy steps \
        --learning_rate 5e-5 \
        --num_train_epochs 1.0 \
        --max_samples 160000 \
        --val_size 0.06 \
        --ddp_timeout 1800000 \
        --plot_loss True\
        --fp16
```

Listing 1: 分布式训练脚本

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 accelerate launch 表示分布式在 4 个 GPU 设备上，采用 accelerate 分布式框架；stage 参数表示训练方式采取 SFT 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。

四、训练和测试结果展示

4.1 训练结果

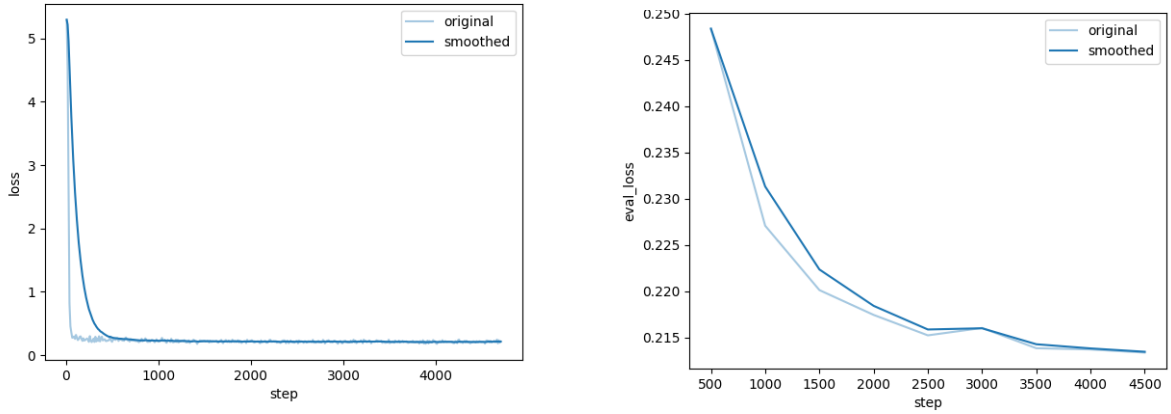

图 1: 左图：训练集上的 loss 曲线，右图：验证集上的 loss 曲线

我们可以看到训练时训练集上的 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。

4.2 测试结果

在测试集下，我们分别得到了原模型和微调后的模型 F1-score。展示如下图所示：

F1-score	原模型	微调模型
micro	0.406	0.695
macro	0.183	0.579
weighted	0.403	0.695

可以发现，无论哪一个指标，微调模型都比原模型高出将近 0.3-0.4，所以可以认定微调模型的性能有大幅度提升。