baichuan7B/13B的原理与微调：从baichuan的SFT实现到baichuan2的RLHF实现

前言

国内众多开源模型中，现在比较有影响力的除了ChatGLM，就是baichuan了(当然，还有其他，后续我再关注)

而baichuan也在不断迭代，23年7月份出了第一代，23年9月份出了第二代，分别对应本文的第一部分、第二部分(即本文第一部分更新于7月份，第二部分更新于9月份)

第一部分 baichuan-7B/13B：与LLaMA的结构相同且表现优秀可商用

1.1 baichuan-7B

1.1.1 基于Transformer/RoPE/RMSNorm/SwiGLU + 1.2万亿训练数据/上下文窗口4096

baichuan-7B 是由百川智能(CEO为原搜狗创始人王小川)开发的一个开源可商用的大规模 预训练　语言模型

- 基于 Transformer 结构，采用了和 LLaMA 一样的模型设计，比如
  位置编码：用的现阶段被大多模型采用的 rotary-embedding 方案，具有更好的外延效果
  激活层：SwiGLU, Feedforward 变化为 8/3 倍的隐含层大小，即 11,008
  Layer-Normalization: 基于 RMSNorm 的 Pre-Normalization

  关于LLaMA结构的解读，请参见：LLaMA的解读与其微调：Alpaca-LoRA/Vicuna/BELLE/中文LLaMA/姜子牙/LLaMA 2
- 在大约 1.2 万亿 tokens 上训练的 70 亿参数模型，支持中英双语
- 上下文窗口长度为 4096
- 在标准的中文和英文权威 benchmark（C-EVAL/MMLU）上均取得同尺寸最好的效果
  具体而言，C-Eval 数据集是一个全面的中文基础模型评测数据集，涵盖了 52 个学科和四个难度的级别
  我们使用该数据集的 dev 集作为 few-shot 的来源，在 test 集上进行了 5-shot 测试
  除了中文之外，作者团队也测试了模型在英文上的效果，MMLU 是包含 57 个多选任务的英文评测数据集，涵盖了初等数学、美国历史、计算机科学、法律等，难度覆盖高中水平到专家水平，是目前主流的LLM评测数据集

  一句话总结，即是在C-EVAL/MMLU等数据集上的表现好于ChatGLM-6B (当然，ChatGLM2-6B又变更强了)

1.1.2 baichuan-7B相比LLaMA-7B的优势

虽然baichuan-7B采用了和LLaMA一样的模型设计，但他们在原本的 LLaMA 框架上进行诸多修改

比如为提升模型的效果以及解码效率，做了

- **分词改进**
  词表大小为64K ，而LLaMA词表大小为32K

  具体而言，参考学术界方案使用 SentencePiece 中的 Byte-Pair Encoding (BPE) 作为分词算法，并且进行了以下的优化：
  目前大部分开源模型主要基于英文优化，因此对中文语料存在效率较低的问题，使用 2000 万条以中英为主的多语言语料训练分词模型，显著提升对于中文的压缩率

  对于数学领域，他们参考了 LLaMA 和 Galactica 中的方案，对数字的每一位单独分开，避免出现数字不一致的问题，对于提升数学能力有重要帮助
  对于罕见字词（如特殊符号等），支持 UTF-8 characters 的 byte 编码，因此做到未知字词的全覆盖
- **数据集改进**
  使用了大约 1.2T 中英 tokens 进行训练(基于开源的中英文数据和自行抓取的中文互联网数据以及部分高质量知识性数据进行的数据清洗)，而 LLaMA 7B 使用 1T 英文 tokens 进行训练

比如为提升训练时的吞吐，做了以下优化

- 算子优化技术：采用更高效算子，如 Flash-Attention，NVIDIA apex 的 RMSNorm 等。
- 算子切分技术：将部分计算算子进行切分，减小内存峰值。
- 混合精度技术：降低在不损失模型精度的情况下加速计算过程。
- 训练容灾技术：训练平台和训练框架联合优化，IaaS + PaaS 实现分钟级的故障定位和任务恢复。
- 通信优化技术，具体包括：
  采用拓扑感知的集合通信算法，避免网络拥塞问题，提高通信效率
  根据卡数自适应设置 bucket size，提高带宽利用率
  根据模型和集群环境，调优通信原语的触发时机，从而将计算和通信重叠

基于上述的几个优化技术，使得在千卡 A800 显卡上达到了 7B 模型 182 TFLOPS 的吞吐， GPU　峰值算力利用率高达 58.3%

1.1.3 baichuan-7B的微调

本次微调参考项目：https://github.com/wp931120/baichuan_sft_lora

由于baichuan没有 supervised finetune 这一步，没有和人类意图进行对齐，经常听不懂你下达的指令。该项目遂利用belle 0.5M 指令微调数据，采用qlora的量化微调的方式对百川大模型进行人类意图对齐训练

如何训练呢？简单而言

1. 首先，从huggingface 中下载baichuan7b 的模型权重 ,然后将 belle 数据集 train_0.5M_CN 下载到本地并放到项目目录下的dataset文件夹下，最后运行sft_lora.py 脚本
2. 接着，将百川LLM 采用qlora的 nf4 和双重量化方式进行量化
3. 最后，采用lora进行指令微调

更具体而言，本次微调baichuan-7B的步骤如下

1. 微调之前的准备
   下载项目仓库

   ```
   git clone https://github.com/wp931120/baichuan_sft_lora.git
   cd baichuan_sft_lora
   ```

   配置环境

   ```
   conda create -n baichuan-7b python=3.9
   conda activate baichuan-7b
   pip install -r requirements.txt
   ```

   数据集下载
   sft 数据集采用的是belle 0.5M
   下载地址：https://huggingface.co/datasets/BelleGroup/train_0.5M_CN/tree/main
   将 belle 数据集 train_0.5M_CN 下载到本地并放到项目目录下的dataset文件夹下
2. 将百川LLM 采用qlora的 nf4 和双重量化方式进行量化
3. 再采用lora进行指令微调
   wp931120x/baichuan_4bit_lora · Hugging Face
4. 修改并运行sft_lora.py文件
   将sft_lora.py中的模型路径设置为自己的模型路径
   执行python sft_lora.py (代码如下所示，来源：baichuan_sft_lora /sft_lora.py)

   ```
   import os  # 导入os模块，这个模块提供了一种方便的使用操作系统依赖功能的方式
   os.environ['CUDA_VISIBLE_DEVICES'] = '0'  # 设置CUDA可见设备，'0'表示仅使用第一块GPU

   from datasets import load_dataset  # 导入load_dataset函数，用于加载数据集
   import transformers                # 导入transformers库，这是一个常用的NLP库

   # 导入Trainer和TrainingArguments，分别用于模型的训练和训练参数的设置
   from transformers import Trainer, TrainingArguments
   # 导入AutoTokenizer和AutoModelForCausalLM，分别用于自动化地从预训练模型中获取Tokenizer和模型
   from transformers import AutoTokenizer, AutoModelForCausalLM
   # 导入BitsAndBytesConfig，用于设置模型的量化配置
   from transformers import BitsAndBytesConfig

   # 导入一些特定的函数和配置类
   from peft import (
       LoraConfig,
       get_peft_model,
       prepare_model_for_kbit_training,
       set_peft_model_state_dict,
   )
   import torch  # 导入PyTorch库，这是一个常用的深度学习库


   # 定义一些配置信息
   CUTOFF_LEN = 1024
   VAL_SET_SIZE = 2000
   DATA_PATH = "./dataset/Belle_open_source_0.5M.json"
   OUTPUT_DIR = "baichuansft"
   resume_from_checkpoint = "baichuansft"

   # 设置设备映射，""表示默认设备，0表示设备编号
   device_map = {"": 0}
   # 使用AutoTokenizer从预训练模型中获取Tokenizer
   tokenizer = AutoTokenizer.from_pretrained("./baichuan-7B",trust_remote_code=True)
   # 使用AutoModelForCausalLM从预训练模型中获取模型，并设置量化配置
   model = AutoModelForCausalLM.from_pretrained("./baichuan-7B",
                                                trust_remote_code=True,
                                                quantization_config=BitsAndBytesConfig(
                                                    load_in_4bit=True,
   ```

```
                                                bnb_4bit_compute_dtype=torch.bfloat16,
                                            bnb_4bit_use_double_quant=True,
                                            bnb_4bit_quant_type='nf4'
                                        ),
                                        device_map=device_map)
model = prepare_model_for_kbit_training(model)  # 准备模型进行kbit训练

# 导入bitsandbytes模块
import bitsandbytes as bnb

# 定义一个函数，用于找到模型中所有的线性层的名称
def find_all_linear_names(model):
    cls = bnb.nn.Linear4bit
    lora_module_names = set()
    for name, module in model.named_modules():  # 遍历模型中的所有模块
        if isinstance(module, cls):  # 如果模块是线性层
            names = name.split('.')
            lora_module_names.add(names[0] if len(names) == 1 else names[-1])  # 添加到线性层名称集合中

    if 'lm_head' in lora_module_names:  # 如果'lm_head'在名称集合中，需要移除
        lora_module_names.remove('lm_head')
    return list(lora_module_names)  # 返回线性层名称列表

# 获取所有的线性层的名称
modules = find_all_linear_names(model)

# 设置LoRA配置
config = LoraConfig(
    r=8,
    lora_alpha=16,
    lora_dropout=0.05,
    bias="none",
    target_modules=modules,
    task_type="CAUSAL_LM",
)

# 获取用于训练的模型
model = get_peft_model(model, config)
tokenizer.pad_token_id = 0  # 设置tokenizer的pad_token_id为0

# 如果有设置从检查点恢复
if resume_from_checkpoint:
    # 检查可用的权重并加载
    checkpoint_name = os.path.join(
        resume_from_checkpoint, "pytorch_model.bin"
    )  # 完整的检查点
    # 如果完整的检查点不存在，则加载LoRA模型的检查点
    if not os.path.exists(checkpoint_name):
        checkpoint_name = os.path.join(
            resume_from_checkpoint, "adapter_model.bin"
        )  # 仅LoRA模型 - 上面的LoRA配置必须匹配
        resume_from_checkpoint = (
            False  # 所以训练器不会尝试加载状态
        )
    if os.path.exists(checkpoint_name):
        print(f"Restarting from {checkpoint_name}")
        adapters_weights = torch.load(checkpoint_name)
        set_peft_model_state_dict(model, adapters_weights)  # 设置模型的状态字典
    else:
        print(f"Checkpoint {checkpoint_name} not found")

# 加载数据集
data = load_dataset("json", data_files=DATA_PATH)

# 定义tokenize函数，用于将输入进行tokenize
def tokenize(prompt, add_eos_token=True):
    # 这里是tokenize的具体操作
    result = tokenizer(
        prompt,
        truncation=True,
        max_length=CUTOFF_LEN,
        padding=False,
        return_tensors=None,
    )
    # 添加EOS token
    if (
            result["input_ids"][-1] != tokenizer.eos_token_id
            and len(result["input_ids"]) < CUTOFF_LEN
            and add_eos_token
```

```
    ):
        result["input_ids"].append(tokenizer.eos_token_id)
        result["attention_mask"].append(1)

    if add_eos_token and len(result["input_ids"]) >= CUTOFF_LEN:
        result["input_ids"][CUTOFF_LEN - 1] = tokenizer.eos_token_id
        result["attention_mask"][CUTOFF_LEN - 1] = 1

    # 输入和标签都是input_ids
    result["labels"] = result["input_ids"].copy()

    return result

# 定义generate_and_tokenize_prompt函数，用于生成并tokenize输入
def generate_and_tokenize_prompt(data_point):
    instruction = data_point['instruction']
    input_text = data_point["input"]
    input_text = "Human: " + instruction + input_text + "\n\nAssistant: "
    input_text = tokenizer.bos_token + input_text if tokenizer.bos_token != None else input_text
    target_text = data_point["output"] + tokenizer.eos_token
    full_prompt = input_text + target_text
    tokenized_full_prompt = tokenize(full_prompt)
    return tokenized_full_prompt

# 划分训练集和验证集，并进行shuffle和map操作
if VAL_SET_SIZE > 0:
    train_val = data["train"].train_test_split(
        test_size=VAL_SET_SIZE, shuffle=True, seed=42
    )
    train_data = train_val["train"].shuffle().map(generate_and_tokenize_prompt)
    val_data = train_val["test"].shuffle().map(generate_and_tokenize_prompt)
else:
    train_data = data['train'].shuffle().map(generate_and_tokenize_prompt)
    val_data = None

# 创建Trainer对象，用于进行训练
trainer = Trainer(
    model=model,
    train_dataset=train_data,
    eval_dataset=val_data,
    args=TrainingArguments(
        num_train_epochs=1,
        per_device_train_batch_size=1,
        per_device_eval_batch_size=1,
        learning_rate=3e-4,
        gradient_accumulation_steps=4,
        evaluation_strategy="steps" if VAL_SET_SIZE > 0 else "no",
        save_strategy="steps",
        eval_steps=2000 if VAL_SET_SIZE > 0 else None,
        save_steps=2000,
        output_dir=OUTPUT_DIR,
        report_to = "tensorboard",
        save_total_limit=3,
        load_best_model_at_end=True if VAL_SET_SIZE > 0 else False,
        optim="adamw_torch"
    ),
    data_collator=transformers.DataCollatorForSeq2Seq(tokenizer,
                                                      pad_to_multiple_of=8,
                                                      return_tensors="pt",
                                                      padding=True),
)

# 进行训练
trainer.train(resume_from_checkpoint=False)
# 保存预训练模型
model.save_pretrained(OUTPUT_DIR)
```

最终，显存占用为7G左右

1.2 baichuan-13B：1.4万亿tokens/上下文长度4096/可商用

1.2.1 模型的介绍、下载、推理

2023年7月11日，百川智能发布Baichuan-13B(这是其GitHub地址)

Baichuan-13B 是继 Baichuan-7B 之后开发的包含 130 亿参数的开源可商用的大规模 语言模型 ，本次发布包含以下两个版本

- 预训练(Baichuan-13B-Base)
- 对齐(Baichuan-13B-Chat，July注：我看了下代码，这里的对齐指的是通过对话数据对齐，即只做了SFT，没做RLHF)

Baichuan-13B 有如下几个特点：

1. 更大尺寸、更多数据
   Baichuan-13B 在 Baichuan-7B 的基础上进一步扩大参数量到 **130** 亿，并且在高质量的语料上训练了 **1.4** 万亿 tokens，他们声称超过 LLaMA-13B 40%，是当前开源 13B 尺寸下训练数据量最多的模型
   支持中英双语，使用 ALiBi 位置编码，上下文窗口长度为 **4096**
2. 同时开源预训练和对齐模型
   预训练模型是适用开发者的『 基座 』，而广大普通用户对有对话功能的对齐模型具有更强的需求。因此本次开源我们同时发布了对齐模型(Baichuan-13B-Chat)，具有很强的对话能力，开箱即用，几行代码即可简单的部署
3. 更高效的推理
   为了支持更广大用户的使用，我们本次同时开源了 int8 和 int4 的量化版本，相对非量化版本在几乎没有效果损失的情况下大大降低了部署的机器资源门槛，可以部署在如 Nvidia 3090 这样的消费级显卡上
4. 开源免费可商用
   Baichuan-13B 不仅对学术研究完全开放，开发者也仅需邮件申请并获得官方商用许可后，即可以免费商用

模型名称	隐藏层维度	层数	注意力头数	词表大小	总参数量	训练数据（tokens）	位置编码	最大长度
Baichuan-7B	4,096	32	32	64,000	7,000,559,616	1.2 万亿	RoPE	4,096
Baichuan-13B	5,120	40	40	64,000	13,264,901,120	1.4 万亿	ALiBi	4,096

由于Baichuan-13B-Base、Baichuan-13B-Chat 推理所需的模型权重、源码、配置已发布在 Hugging Face，故可以使用下面命令将模型下载到本地，方便使用时直接加载（/data/sim_chatgpt/）

```
git clone https://huggingface.co/baichuan-inc/Baichuan-13B-Base
git clone https://huggingface.co/baichuan-inc/Baichuan-13B-Chat
```

至于模型推理，可以直接用 LLaMA-Efficient-Tuning 仓库中的环境

```
conda activate baichuan-7b
```

```
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer
from transformers.generation.utils import GenerationConfig
tokenizer = AutoTokenizer.from_pretrained("/data/sim_chatgpt/Baichuan-13B-Chat", use_fast=False, trust_remote_code=True)
model = AutoModelForCausalLM.from_pretrained("/data/sim_chatgpt/Baichuan-13B-Chat", torch_dtype=torch.float16, trust_remote_code=True)
model = model.quantize(4).cuda()
model.generation_config = GenerationConfig.from_pretrained("/data/sim_chatgpt/Baichuan-13B-Chat")
messages = []
messages.append({"role": "user", "content": "世界上第二高的山峰是哪座"})
response = model.chat(tokenizer, messages)
print(response)
```

1.2.2 基于LoRA 微调Baichuan-13B-Chat过程(单轮)

本次微调参考：LLaMA-Efficient-Tuning

1. 下载仓库

   ```
   git clone https://github.com/hiyouga/LLaMA-Efficient-Tuning.git
   ```

2. 配置环境

   ```
   # 这里使用和 baichuan-7b 相同的环境
   conda create -n baichuan-7b python=3.10
   conda activate baichuan-7b
   cd LLaMA-Efficient-Tuning
   pip install -r requirements.txt
   ```

3. 微调数据
   这里直接使用 alpaca_gpt4_zh，也就是由GPT4生成的中文alpaca数据，在LLaMA-Efficient-Tuning /data目录下
   数据格式如下：
   instruction：任务指令，不能为空。
   input：任务输入，可为空。如果不为空，项目内部处理训练数据时，会将 instruction、input 拼接在一起作为任务的输入
   output：任务输出，不能为空

数据示例如下：

```
 {
  "instruction": "保持健康的三个提示。",
  "input": "",
  "output": "以下是保持健康的三个提示：\n\n1. 保持身体活动。每天做适当的身体运动，如散步、跑步或游泳，能促进心血管健康，增强肌肉力量，并有助于减少体重。\n\n2. 均衡饮食。每天食用新鲜的蔬菜、水果、全谷物和脂肪含量低的蛋白质食物，避免高糖、高脂肪和加工食品，以保持健康的饮食习惯。\n\n3. 睡眠充足。睡眠对人体健康至关重要，成年人每天应保证 7-8 小时的睡眠。良好的睡眠有助于减轻压力，促进身体恢复，并提高注意力和记忆力。"
 },

 {
  "instruction": "三原色是什么？",
  "input": "",
  "output": "三原色通常指的是红色、绿色和蓝色（RGB）。它们是通过加色混合原理创建色彩的三种基础颜色。在以发光为基础的显示设备中（如电视、计算机显示器、智能手机和平板电脑显示屏），三原色可混合产生大量色彩。其中红色和绿色可以混合生成黄色，红色和蓝色可以混合生成品红色，蓝色和绿色可以混合生成青色。当红色、绿色和蓝色按相等比例混合时，可以产生白色或灰色。\n\n此外，在印刷和绘画中，三原色指的是以颜料为基础的红、黄和蓝颜色（RYB）。这三种颜色用以通过减色混合原理来创建色彩。不过，三原色的具体定义并不唯一，不同的颜色系统可能会采用不同的三原色。"
 },
```

4. 微调的各个参数如下

```
CUDA_VISIBLE_DEVICES=0  # 设定CUDA设备ID为0，只使用第一块GPU进行训练
nohup python src/train_bash.py \   # 使用nohup命令在后台运行训练脚本train_bash.py，保证终端关闭后任务仍在运行
    --do_train \                   # 指定执行训练操作
    --model_name_or_path /data/sim_chatgpt/Baichuan-13B-Chat \  # 指定模型的名称或者模型文件的路径
    --template baichuan \          # 模型模板为"baichuan"
    --dataset alpaca_gpt4_zh \     # 使用名为"alpaca_gpt4_zh"的数据集进行训练
    --output_dir baichuan_lora_checkpoint \  # 训练过程中的输出目录
    --max_source_length 256 \      # 输入文本的最大长度
    --max_target_length 512 \      # 输出文本的最大长度
    --per_device_train_batch_size 1 \      # 每个设备的训练批次大小
    --gradient_accumulation_steps 1 \      # 梯度累计步骤，用于模拟更大的批次大小
    --lr_scheduler_type cosine \   # 使用余弦退火策略进行学习率调整
    --logging_steps 10 \           # 每10步记录一次日志
    --save_steps 10000 \           # 每10000步保存一次模型
    --learning_rate 5e-5 \         # 设置学习率
    --num_train_epochs 1.0 \       # 训练1个epoch
    --plot_loss \                  # 绘制损失图
    --fp16 \                       # 使用半精度浮点数进行训练，可以加速训练并减少内存使用
    --lora_target W_pack \         # LoRA的目标设置
    --lora_rank 8 \                # LoRA的秩设置
    --padding_side right \         # 在序列的右侧进行填充
    --quantization_bit 4 \         # 量化位数设置为4
    >> qlora_log.out 2>&1 &        # 将标准输出和错误输出都重定向到qlora_log.out文件中，并在后台运行
```

显存占用约9个G，运行时间大概60小时

```
1. 保持身体活动。每天做适当的身体运动，如散步、跑步或游泳，能促进心血管健康，增强肌肉力量，并有助于减少体重。

2. 均衡饮食。每天食用新鲜的蔬菜、水果、全谷物和脂肪含量低的蛋白质食物，避免高糖、高脂肪和加工食品，以保持健康的饮食习惯。

3. 睡眠充足。睡眠对人体健康至关重要，成年人每天应保证 7-8 小时的睡眠。良好的睡眠有助于减轻压力，促进身体恢复，并提高注意力和记忆力。</s>
  2%||         | 1000/48818 [1:24:26<57:28:20,  4.33s/it]{'loss': 1.9935, 'learning_rate': 4.9999995806904065e-05, 'epoch': 0.0}
```

微调后生成的日志文件

```
{'loss': 1.3176, 'learning_rate': 4.0087961802326434e-10, 'epoch': 1.0}
{'loss': 1.267, 'learning_rate': 3.149474423147503e-10, 'epoch': 1.0}
{'loss': 1.3224, 'learning_rate': 2.3936845940930597e-10, 'epoch': 1.0}
{'loss': 1.2906, 'learning_rate': 1.7414270060689408e-10, 'epoch': 1.0}
{'loss': 1.1922, 'learning_rate': 1.1927019291646525e-10, 'epoch': 1.0}
{'loss': 1.2969, 'learning_rate': 7.475095906706031e-11, 'epoch': 1.0}
{'loss': 1.3823, 'learning_rate': 4.058501749115706e-11, 'epoch': 1.0}
{'train_runtime': 161449.7236, 'train_samples_per_second': 0.302, 'train_steps_per_second': 0.302,
n_loss': 1.2808091177095076, 'epoch': 1.0}
***** train metrics *****
  epoch                    =               1.0
  train_loss               =            1.2808
  train_runtime            = 1 day, 20:50:49.72
  train_samples_per_second =             0.302
  train_steps_per_second   =             0.302
```

在 baichuan_lora_checkpoint 路径下会生成微调后的模型文件，如下：

```
adapter_config.json  checkpoint-10000  checkpoint-40000      runs                training_args.bin
adapter_model.bin    checkpoint-20000  finetuning_args.json  trainer_log.jsonl   training_loss.png
all_results.json     checkpoint-30000  README.md             trainer_state.json  train_results.json
```