一. chat-glm 模型微调

1. 数据集构建规范

1. 使用自定义数据集，请按照以下格式提供数据集定义。

```
"数据集名称": {
  "hf_hub_url": "Hugging Face 的数据集仓库地址（若指定，则忽略 script_url 和 file_name）",
  "ms_hub_url": "ModelScope 的数据集仓库地址（若指定，则忽略 script_url 和 file_name）",
  "script_url": "包含数据加载脚本的本地文件夹名称（若指定，则忽略 file_name）",
  "file_name": "该目录下数据集文件的名称（若上述参数未指定，则此项必需）",
  "file_sha1": "数据集文件的 SHA-1 哈希值（可选，留空不影响训练）",
  "subset": "数据集子集的名称（可选，默认：None）",
  "folder": "Hugging Face 仓库的文件夹名称（可选，默认：None）",
  "ranking": "是否为偏好数据集（可选，默认：False）",
  "formatting": "数据集格式（可选，默认：alpaca，可以为 alpaca 或 sharegpt）",
  "columns": {
    "prompt": "数据集代表提示词的表头名称（默认：instruction，用于 alpaca 格式）",
    "query": "数据集代表请求的表头名称（默认：input，用于 alpaca 格式）",
    "response": "数据集代表回答的表头名称（默认：output，用于 alpaca 格式）",
    "history": "数据集代表历史对话的表头名称（默认：None，用于 alpaca 格式）",
    "messages": "数据集代表消息列表的表头名称（默认：conversations，用于 sharegpt 格式）",
    "role": "消息中代表发送者身份的键名（默认：from，用于 sharegpt 格式）",
    "content": "消息中代表文本内容的键名（默认：value，用于 sharegpt 格式）",
    "system": "数据集代表系统提示的表头名称（默认：None，用于两种格式）"
  }
}
```

添加后可通过指定 `--dataset 数据集名称` 参数使用自定义数据集。

支持的数据集：**alpaca** 和 **sharegpt**，其中 alpaca 格式的数据集按照以下方式组织：

```
[
  {
    "instruction": "用户指令（必填）",
    "input": "用户输入（选填）",
    "output": "模型回答（必填）",
    "system": "系统提示词（选填）",
    "history": [
      ["第一轮指令（选填）", "第一轮回答（选填）"],
      ["第二轮指令（选填）", "第二轮回答（选填）"]
    ]
  }
]
```

对于上述格式的数据，`dataset_info.json` 中的 `columns` 应为：

```
"数据集名称": {
  "columns": {
    "prompt": "instruction",
    "query": "input",
    "response": "output",
    "system": "system",
    "history": "history"
  }
}
```

其中 `prompt` 和 `response` 列应当是非空的字符串，分别代表用户指令和模型回答。`query` 列的内容将会和 `prompt` 列拼接作为模型输入。

`system` 为模板中的系统提示词。`history` 列是由多个字符串二元组构成的列表，分别代表历史消息中每轮的指令和回答。注意每轮的模型回答**均会被用于训练**。

对于预训练数据集，仅 `prompt` 列中的内容会用于模型训练。

对于偏好数据集，`response` 列应当是一个长度为 2 的字符串列表，排在前面的代表更优的回答，例如：

```
{
  "instruction": "用户指令",
  "input": "用户输入",
  "output": [
    "优质回答",
    "劣质回答"
  ]
}
```

sharegpt 格式的数据集按照以下方式组织：

```
[
  {
    "conversations": [
      {
        "from": "human",
        "value": "用户指令"
      },
      {
        "from": "gpt",
        "value": "模型回答"
      }
    ],
    "system": "系统提示词（选填）"
  }
]
```

对于上述格式的数据，`dataset_info.json` 中的 `columns` 应为：

```
"数据集名称": {
  "columns": {
    "messages": "conversations",
    "role": "from",
    "content": "value",
    "system": "system"
  }
}
```

其中 `messages` 列必须为偶数长度的列表，且符合 `用户/模型/用户/模型/用户/模型` 的顺序。

预训练数据集和偏好数据集尚不支持 sharegpt 格式。

2. chaglm 模型微调(参数讲解)

```
CUDA_VISIBLE_DEVICES=0 python src/train_bash.py \
    --stage sft \ # 训练阶段
    --do_train True \
    --model_name_or_path ./THUDM/chatglm3-6b-base \
    --finetuning_type lora \
    --template default \
    --dataset_dir data \
    --dataset function_call,alpaca_gpt4_en,alpaca_gpt4_zh \
    --cutoff_len 2048 \
    --learning_rate 0.002 \
    --num_train_epochs 10.0 \
    --max_samples 8000 \
    --per_device_train_batch_size 4 \
    --gradient_accumulation_steps 4 \
    --lr_scheduler_type cosine \
    --max_grad_norm 1.0 \
    --logging_steps 5 \
    --save_steps 200 \
    --warmup_steps 0 \
    --lora_rank 8 \
    --lora_dropout 0.1 \
    --lora_target all \
    --output_dir saves\ChatGLM3-6B-Base\lora\train_full \
    --fp16 True \
    --plot_loss True
```

- stage:训练阶段,分为预训练（Pre-Training）、指令监督微调（Supervised Fine-Tuning）、奖励模型训练（Reward Modeling）、PPO 、DPO 五种，建议选择指令监督微调（Supervised Fine-Tuning）。
  - Pre-Training：在该阶段，模型会在一个大型数据集上进行预训练，学习基本的语义和概念。
  - Supervised Fine-Tuning：在该阶段，模型会在一个带标签的数据集上进行微调，以提高对特定任务的准确性。

    - Reward Modeling：在该阶段，模型会学习如何从环境中获得奖励，以便在未来做出更好的决策。
    - PPO Training：在该阶段，模型会使用策略梯度方法进行训练，以提高在环境中的表现。
    - DPO Training：在该阶段，模型会使用深度强化学习方法进行训练，以提高在环境中的表现。
- model_name_or_path：预训练模型路径
- finetuning_type:微调方法
    - full：将整个模型都进行微调。
    - freeze：将模型的大部分参数冻结，只对部分参数进行微调。
    - lora：将模型的部分参数冻结，只对部分参数进行微调，但只在特定的层上进行微调。
- template：模板
- Data dir:数据路径。
- Dataset:数据集。
- Learning rate：学习率，学习率越大，模型的学习速度越快，但是学习率太大的话，可能会导致模型在寻找最优解时**跳过**最优解，学习率太小的话，模型学习速度会很慢，所以这个参数需要根据实际情况进行调整，这里我们使用默认值`5e-5`。
- Epochs：训练轮数，训练轮数越多，模型的学习效果越好，但是训练轮数太多的话，模型的训练时间会很长，因为我们的训练数据比较少，所以要适当增加训练轮数，这里将值设置为`30`。
- Max samples：最大样本数，每个数据集最多使用的样本数，因为我们的数据量很少只有 80 条，所以用默认值就可以了。
- Compute type：计算类型，这里的`fp16`和`bf16`是指数字的数据表示格式，主要用于深度学习训练和推理过程中，以节省内存和加速计算，这里我们选择`bf16`
- LR Scheduler：学习率调节器，有以下选项可以选择，建议选择默认值`cosine`
    - linear（线性）: 随着训练的进行，学习率将以线性方式减少。
    - cosine（余弦）: 这是根据余弦函数来减少学习率的。在训练开始时，学习率较高，然后逐渐降低并在训练结束时达到最低值。
    - cosine_with_restarts（带重启的余弦）: 和余弦策略类似，但是在一段时间后会重新启动学习率，并多次这样做。
    - polynomial（多项式）: 学习率会根据一个多项式函数来减少，可以设定多项式的次数。
    - constant（常数）: 学习率始终保持不变。
    - constant_with_warmup（带预热的常数）: 开始时，学习率会慢慢上升到一个固定值，然后保持这个值。
    - inverse_sqrt（反平方根）: 学习率会随着训练的进行按照反平方根的方式减少。
    - reduce_lr_on_plateau（在平台上减少学习率）: 当模型的进展停滞时（例如，验证误差不再下降），学习率会自动减少。
- Gradient accumulation：梯度累积和最大梯度范数,这两个参数通常一起使用，以保证在微调大型语言模型时，能够有效地处理大规模数据，同时保证模型训练的稳定性。梯度累积允许在有限的硬件资源上处理更大的数据集，而最大梯度范数则可以防止梯度爆炸，保证模型训练的稳定性，这里我们使用默认值即可。

3. 基于本地知识库问答应用

实现流程：加载文件 -> 读取文本 -> 文本分割 -> 文本向量化 -> 问句向量化 -> 在文本向量中匹配出与问句向量最相似的 `top k`个 -> 匹配出的文本作为上下文和问题一起添加到 `prompt`中 -> 提交给 `LLM`生成回答。

第一阶段：加载文件-读取文件-文本分割(Text splitter)

加载文件：这是读取存储在本地的知识库文件的步骤

读取文件：读取加载的文件内容，通常是将其转化为文本格式

文本分割(Text splitter)：按照一定的规则(例如段落、句子、词语等)将文本分割

第二阶段：文本向量化(embedding)-存储到向量数据库

文本向量化(embedding)：这通常涉及到NLP的特征抽取，可以通过诸如TF-IDF、word2vec、BERT等方法将分 割好的文本转化为数值向量

存储到向量数据库：文本向量化之后存储到数据库vectorstore

说明：可以网上下载embedding模型，也可以使用具体向量化功能的大模型对文本向量化，如shibing624/text2vec-base-chinese 、ernie-3.0-base-zh等

第三阶段：问句向量化

这是将用户的查询或问题转化为向量，应使用与文本向量化相同的方法，以便在相同的空间中进行比较

第四阶段：在文本向量中匹配出与问句向量最相似的top k个

这一步是信息检索的核心，通过计算余弦相似度、欧氏距离等方式，找出与问句向量最接近的文本向量

第五阶段：匹配出的文本作为上下文和问题一起添加到prompt中

这是利用匹配出的文本来形成与问题相关的上下文，用于输入给语言模型

第六阶段：提交给LLM生成回答

将这个问句问题和上下文一起提交给语言模型(大模型)，让它生成回答，比如知识查询

二. 本地知识库

实现流程：加载文件 -> 读取文本 -> 文本分割 -> 文本向量化 -> 问句向量化 -> 在文本向量中匹配出与问句向量最相似的 `top k`个 -> 匹配出的文本作为上下文和问题一起添加到 `prompt`中 -> 提交给 `LLM`生成回答。

1. 文本分割

> 文本切割的准确性对于后续问题具有至关重要的地位，

2. 文本向量化

ssh -p 34420 root@connect.westb.seetacloud.com

3. 问句向量化

xtuner train ./internlm2_chat_7b_qlora_oasst1_e3_copy.py