sql模型微调测评

笔记本:	我的第一个笔记本		
创建时间:	2024/11/25 8:51	**更新时间:**	2024/11/26 15:38
作者:	耿介		
URL:	https://dsw-gateway-cn-shanghai.data.aliyun.com/dsw-467604/lab/tree/fine_tu...		

硬件需求
显存：24GB及以上（推荐使用30系或A10等sm80架构以上的NVIDIA显卡进行尝试） 内存：16GB RAM: 2.9 /16 GB GPU RAM: 15.5/16.0 GB

阿里云的服务器配置刚好足够

准备微调数据集并训练

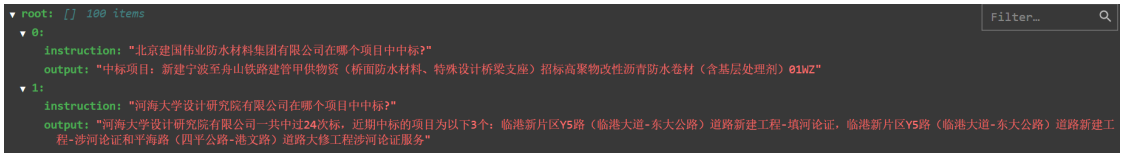
根据10:1的比例把1200个数据拆分成训练集和验证集，

根据我给的lora_finetune.ipynb进行微调，导出lora的model文件，按照之前的代码微调步骤，合并模型得到完整的微调后模型。

准备评估数据集

将评估的数据命名为questions.json

格式大概如下



并且分成两种情况进行测试，

1.一种是简单问题，用简单的sql语言就可以查询到答案。
2.另外一种是复杂问题，涉及统计计算，并且可能需要较复杂的sql语言才能查询答案，并且可能需要分析

SQL查询生成

然后调用本地微调好的模型进行sql查询生成，对于查询失败的问题，返回一个“没有查询成功”的字符串，以此作为召回率的评估，把生成的回答重新整理在一个新的.json文件

questions_with_generated_output.json

代码如下

```
import json
import pandas as pd
from langchain_community.utilities import SQLDatabase
from langchain_ollama import ChatOllama
import sqlite3


# 设置 SQLite 数据库路径
db_file = "/mnt/workspace/上海市交通系统交易问答框架/SQL_database/上海市交通系统交易情况.db"
conn = sqlite3.connect(db_file, check_same_thread=False)


# 定义 SQL 检索逻辑
def SQL_finetune_retrieve(question):
    try:
        llm = ChatOllama(model="jiesql_thousand")
        llm2 = ChatOllama(model="EntropyYue/chatglm3:6b")
        resized_question = question + "，请根据这个问题生成对应的SQL查询"
        response = llm.invoke(resized_question)
        sql_query = response.content  # 从 LLM 响应中提取 SQL 查询


        # 执行生成的 SQL 查询
        result = pd.read_sql_query(sql_query, conn)
        result_str = result.to_string(index=False)


        resized_question2 = (
            f"原问题是：{question}，检索结果是：{result_str}，"
            "请根据问题和检索结果整合成自然语言回答，只需要回答查询到的内容，不需要抱歉"
        )
        ans = llm2.invoke(resized_question2)
        return ans.content
    except Exception as e:
        return "没有查询成功"


# 读取 JSON 数据
input_file = "questions_thousand.json"
output_file = "questions_with_generated_output.json"


with open(input_file, "r", encoding="utf-8") as file:
    data = json.load(file)
```

```
# 遍历数据，生成新的键值对 generated_output，同时保留原来的 output
for entry in data:
    question = entry.get("Question", "")
    if question:
        generated_output = SQL_finetune_retrieve(question)
        entry["generated_output"] = generated_output  # 添加新的键值对


# 保存修改后的数据
with open(output_file, "w", encoding="utf-8") as file:
    json.dump(data, file, ensure_ascii=False, indent=4)


print(f"处理完成，结果已保存到 {output_file}")
```

召回率计算

统计查询失败的个数，以此作为召回率的计算，代码如下：

```
import json


# 加载 JSON 数据
with open('questions_with_generated_output.json', 'r', encoding='utf-8') as file:
    data = json.load(file)


# 初始化计数器
total_count = 0
none_count = 0


# 遍历数据计算
for item in data:
    total_count += 1
    if item.get('generated_output') == "没有查询成功":
        none_count += 1


# 计算召回率
recall_rate = 1 - (none_count / total_count) if total_count > 0 else 0


# 输出结果
print(f"总数: {total_count}")
print(f"generated_output 为 'None' 的数量: {none_count}")
print(f"召回率: {recall_rate:.2%}")
```

准确率计算:

设计提示词，比较ground_truth 和生成回答，对于正确回答的记录下true，反之则是false，最后统计true的频率

参数调整

一开始为了确保loss能够收敛，选择比较大的r来确保表达能力：

r= 16 , alpha = 32

这里只选择10个epoch：

```
{'loss': 0.0117, 'grad_norm': 2.368501663208008, 'learning_rate': 1.0000000000000002e-06, 'epoch': 10.21}
{'loss': 0.0109, 'grad_norm': 0.15307728946208954, 'learning_rate': 8.333333333333333e-07, 'epoch': 10.24}
{'loss': 0.0064, 'grad_norm': 0.4209514260292053, 'learning_rate': 6.666666666666667e-07, 'epoch': 10.28}
{'loss': 0.0047, 'grad_norm': 0.3458356559276581, 'learning_rate': 5.000000000000001e-07, 'epoch': 10.31}
{'loss': 0.0176, 'grad_norm': 0.7412222623825073, 'learning_rate': 3.3333333333333335e-07, 'epoch': 10.35}
{'loss': 0.0349, 'grad_norm': 1.5019413232803345, 'learning_rate': 1.6666666666666668e-07, 'epoch': 10.38}
{'loss': 0.0035, 'grad_norm': 0.5118281245231628, 'learning_rate': 0.0, 'epoch': 10.42}
```

100个难问题测试结果:

召回率 87%
准确率 86%

1000个简单问题测试结果：
召回率 97.32%
简单问题准确率基本等于召回率，不用专门计算

以下是修改完参数，r = 8 ， alpha = 16

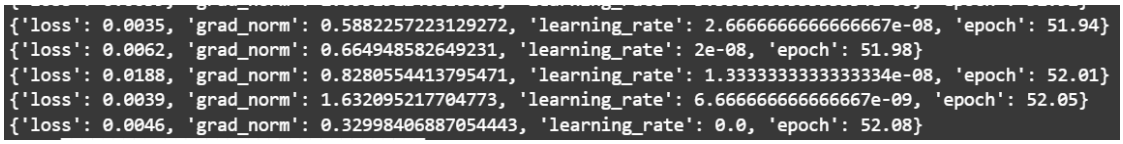
```
{'loss': 0.0065, 'grad_norm': 0.2488902509212494, 'learning_rate': 4e-08, 'epoch': 51.88}
{'loss': 0.0051, 'grad_norm': 0.7007434964179993, 'learning_rate': 3.333333333333334e-08, 'epoch': 51.91}
{'loss': 0.0033, 'grad_norm': 0.6595039963722229, 'learning_rate': 2.6666666666666667e-08, 'epoch': 51.94}
{'loss': 0.0049, 'grad_norm': 0.5619759559631348, 'learning_rate': 2e-08, 'epoch': 51.98}
{'loss': 0.0206, 'grad_norm': 0.8447806239128113, 'learning_rate': 1.3333333333333334e-08, 'epoch': 52.01}
{'loss': 0.0038, 'grad_norm': 2.048177719116211, 'learning_rate': 6.666666666666667e-09, 'epoch': 52.05}
{'loss': 0.0039, 'grad_norm': 0.24480192363262177, 'learning_rate': 0.0, 'epoch': 52.08}
```

100个难问题测试结果:

召回率 87%
准确率 65%

1000个简单问题测试结果：
召回率 95.42%

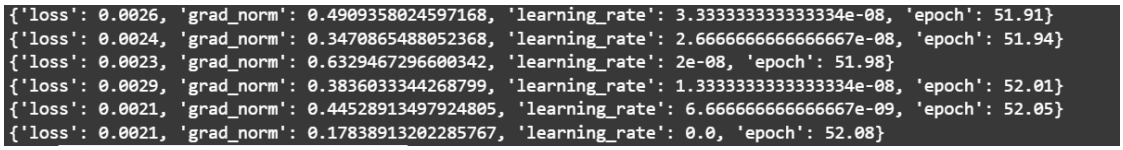
在增加drop_out,从0.1增加到0.2



100个难问题测试结果:
召回率 83
准确率 59

1000个简单问题测试结果：
召回率 95.66

修改参数r=32, alpha =64



100个难问题测试结果:
召回率 88
准确率 71

1000个简单问题测试结果：
召回率 95.66

后续做了更多实验，不再赘述，结果发现r<16的情况下容易欠拟合，而r>16的时候容易过拟合，学习率也应该保持较低的水平，防止过拟合。
dropout增大的原本目的是为了让模型去学习鲁棒性的特征，但是0.2反倒让模型产生欠拟合。

最终还是保留下列配置

```
data_config:
  train_file: train.json
  val_file: dev.json
  test_file: dev.json
  num_proc: 16
max_input_length: 256
max_output_length: 512
training_args:
  # see `transformers.Seq2SeqTrainingArguments`
  output_dir: ./output
  max_steps: 3000
  # needed to be fit for the dataset
  learning_rate: 5e-5
  # settings for data loading
  per_device_train_batch_size: 4
  dataloader_num_workers: 16
  remove_unused_columns: false
  # settings for saving checkpoints
  save_strategy: steps
  save_steps: 500
  # settings for logging
  log_level: info
  logging_strategy: steps
  logging_steps: 10
  # settings for evaluation
  per_device_eval_batch_size: 16
  evaluation_strategy: steps
  eval_steps: 500
  # settings for optimizer
  # adam_epsilon: 1e-6
  # uncomment the following line to detect nan or inf values
  # debug: underflow_overflow
  predict_with_generate: true
  # see `transformers.GenerationConfig`
  generation_config:
    max_new_tokens: 512
  # set your absolute deepspeed path here
  #deepspeed: ds_zero_2.json
  # set to true if train with cpu.
  use_cpu: false
peft_config:
  peft_type: LORA
  task_type: CAUSAL_LM
  r: 16
  lora_alpha: 32
  lora_dropout: 0.1
```