ChatGLM3微调部署

有关ChatGLM3部署的文档。

1. 下载代码

```
$ git clone https://github.com/THUDM/ChatGLM3.git
```

2. 基础环境搭建

```
$ conda create -n chatglm_test python==3.10
$ conda activate chatglm_test
```

3. ChatGLM3环境搭建

```
$ pip3 install torch torchvision torchaudio -i
https://pypi.tuna.tsinghua.edu.cn/simple --default-timeout=10000
# 可以先不安装，后续有说明
$ pip install peft -i https://pypi.tuna.tsinghua.edu.cn/simple
$ cd ChatGLM3
$ pip install -r requirements.txt -i https://pypi.tuna.tsinghua.edu.cn/simple
```

4. 模型下载（魔塔社区）

```
$ sudo apt update
$ sudo apt-get install git-lfs
$ git lfs install
$ git clone https://www.modelscope.cn/ZhipuAI/chatglm3-6b.git
```

可以删除部分模型（可选）：

```
$ rm -rf *.safetensors
$ rm -rf model.safetensors.index.json
```

5. Web APP (gradio)

修改路径：文件在**./basic_demo/web_demo_gradio.py**

代码片段：

```
MODEL_PATH = os.environ.get('MODEL_PATH', '你的路径')
```

运行Web Demo

```
$ python web_demo_gradio.py
```

第一次加载模型时间很长：

```
Loading checkpoint shards: 100%|                                                | 7/7 [00:10<00:00,  1.44s/it]
Setting eos_token is not supported, use the default one.
Setting pad_token is not supported, use the default one.
Setting unk_token is not supported, use the default one.
Running on local URL:  http://127.0.0.1:7870

To create a public link, set `share=True` in `launch()`.
```

完成后出现：

6. 微调模型

可能perf出现版本问题，导致模型微调后不能推理，可以改为0.7.0。

进入finetune_demo文件夹:

```
$ finetune_demo
$ sudo apt install mpich
$ sudo apt-get install openmpi-bin openmpi-doc libopenmpi-dev
$ env LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/usr/lib64:/usr/lib" MPICC=$(which mpicc) python -m pip install -v --no-cache-dir --upgrade mpi4py
$ export OMPI_CC=x86_64-conda_cos6-linux-gnu-cc
$ conda install gcc_linux-64 -c conda-forge
$ conda install openmpi -c conda-forge
$ pip install typer
$ pip install sentencepiece
$ pip install -r requirements.txt -i https://pypi.tuna.tsinghua.edu.cn/simple
```

（1）下载与解压数据

```
$ wget https://cloud.tsinghua.edu.cn/f/b3f119a008264b1cabd1/?dl=1 -O AdvertiseGen.tar.gz
$ tar -xzvf AdvertiseGen.tar.gz
```

（2）查看数据

```
{"content": "类型#裤*材质#牛仔布*材质#蕾丝*颜色#黑色*风格#性感*图案#蕾丝*裤长#短裤", "summary": "牛仔短裤，黑色调，散发出性感撩人的迷人风情；简洁的腰头，显得腰线特别好看；裤脚边沿顺着蕾丝睫毛花边，微微蜿蜒的弧线婉约浪漫，整套look更显得精致了几分。"}
```

（3）数据转换

查看ChatGLM3所需要的数据格式，在官方GitHub链接中：
https://github.com/THUDM/ChatGLM3/tree/main/finetune_demo

如果您仅希望微调模型的对话能力，而非工具能力，您应该按照以下格式整理数据。

```
[
  {
    "conversations": [
      {
        "role": "system",
        "content": "<system prompt text>"
      },
      {
        "role": "user",
        "content": "<user prompt text>"
      },
      {
        "role": "assistant",
        "content": "<assistant response text>"
      },
      // ... Muti Turn
      {
        "role": "user",
        "content": "<user prompt text>"
      },
      {
        "role": "assistant",
        "content": "<assistant response text>"
      }
    ]
  }
  // ...
]
```

创建数据格式转换文件:

```
$ touch data_process.py
```

将如下程序复制进创建的Python脚本中：

```python
import json
from typing import Union
from pathlib import Path

def _resolve_path(path: Union[str, Path]) -> Path:
    return Path(path).expanduser().resolve()

def _mkdir(dir_name: Union[str, Path]):
    dir_name = _resolve_path(dir_name)
    if not dir_name.is_dir():
        dir_name.mkdir(parents=True, exist_ok=False)

def convert_adgen(data_dir: Union[str, Path], save_dir: Union[str, Path]):
    def _convert(in_file: Path, out_file: Path):
        _mkdir(out_file.parent)
        with open(in_file, encoding='utf-8') as fin:
            with open(out_file, 'wt', encoding='utf-8') as fout:
```

```
                for line in fin:
                    dct = json.loads(line)
                    sample = {'conversations': [{'role': 'user', 'content': dct['content']},
                                                {'role': 'assistant', 'content': dct['summary']}]}
                    fout.write(json.dumps(sample, ensure_ascii=False) + '\n')

    data_dir = _resolve_path(data_dir)
    save_dir = _resolve_path(save_dir)

    train_file = data_dir / 'train.json'
    if train_file.is_file():
        out_file = save_dir / train_file.relative_to(data_dir)
        _convert(train_file, out_file)

    dev_file = data_dir / 'dev.json'
    if dev_file.is_file():
        out_file = save_dir / dev_file.relative_to(data_dir)
        _convert(dev_file, out_file)

convert_adgen('data/AdvertiseGen', 'data/AdvertiseGen_fix')
```

运行代码：

```
$ python data_process.py
```

得到一个fix文件夹：

```
.
├── AdvertiseGen
│   ├── dev.json
│   └── train.json
├── AdvertiseGen_fix
│   ├── dev.json
│   └── train.json
```

里面是转换好的文件。

（5）微调

修改peft>=0.10.0为peft=0.7.1。

mpi4py可以删除，但是执行上面的步骤也可以的。

需要安装nltk：

```
$ pip install nltk -i https://pypi.tuna.tsinghua.edu.cn/simple
```

查看并修改yaml文件（注释版本）：

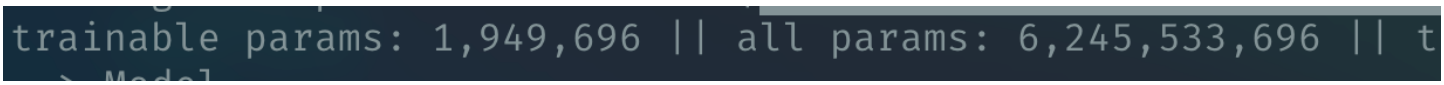
```yaml
data_config:  # 数据配置部分
  train_file: train.json  # 训练数据文件
  val_file: dev.json  # 验证数据文件
  test_file: dev.json  # 测试数据文件
  num_proc: 16  # 处理数据的进程数
max_input_length: 256  # 输入最大长度
max_output_length: 512  # 输出最大长度
training_args:  # 训练参数配置
  # 查看 `transformers.Seq2SeqTrainingArguments` 了解更多信息
  output_dir: ./output  # 输出目录
  max_steps: 3000  # 训练的最大步数
  # 根据数据集调整的学习率
  learning_rate: 5e-5  # 学习率
  # 数据加载设置
  per_device_train_batch_size: 4  # 每个设备的训练批量大小
  dataloader_num_workers: 16  # 数据加载的工作线程数
  remove_unused_columns: false  # 是否移除未使用的列
  # 保存检查点的设置
  save_strategy: steps  # 保存策略：按步数
  save_steps: 500  # 每500步保存一次
  # 日志设置
  log_level: info  # 日志等级
  logging_strategy: steps  # 日志记录策略：按步数
  logging_steps: 10  # 每10步记录一次日志
  # 评估设置
  per_device_eval_batch_size: 16  # 每个设备的评估批量大小
  evaluation_strategy: steps  # 评估策略：按步数
  eval_steps: 500  # 每500步进行一次评估
  # 优化器设置
  # adam_epsilon: 1e-6  # Adam优化器的epsilon参数（未启用）
  # 以下行用于检测nan或inf值（未启用）
  # debug: underflow_overflow
  predict_with_generate: true  # 使用生成模式进行预测
  # 查看 `transformers.GenerationConfig` 了解更多信息
  generation_config:  # 生成配置
    max_new_tokens: 512  # 生成的最大新令牌数
```

```yaml
  # 在此处设置你的deepspeed配置路径（未启用）
  #deepspeed: ds_zero_2.json
  # 如果使用CPU训练，设置为true。
  use_cpu: false
peft_config:  # PEFT配置
  peft_type: LORA  # PEFT类型：LORA
  task_type: CAUSAL_LM  # 任务类型：因果语言模型
  r: 8  # LORA的r参数
  lora_alpha: 32  # LORA的alpha参数
  lora_dropout: 0.1  # LORA的dropout率
```

执行微调：

```
$ CUDA_VISIBLE_DEVICES=0 python3 finetune_hf.py data/AdvertiseGen_fix
/home/asic-zty/LLM/model_path/chatglm3-6b ./configs/lora.yaml
```

使用lora微调的参数：



训练中：

7. 测试

训练后模型文件夹如下：

```
.
├── adapter_config.json             # 适配器配置文件，包含适配器模块的结构和参数设置
├── adapter_model.safetensors       # 适配器模型的权重文件，使用 SafeTensors 格式存储，用于确保数据的一致性和安全
├── optimizer.pt                    # 优化器状态文件，存储优化器的当前状态，以便恢复训练时使用
├── README.md                       # 一般包含模型和训练过程的说明或其他重要信息
├── rng_state.pth                   # 随机数生成器的状态文件，用于确保模型训练可复现
```

```
├── scheduler.pt                # 学习率调度器状态文件，存储调度器的当前状态，用于控制学习率的变化
├── trainer_state.json          # 训练器状态文件，包含训练过程中的各种统计信息和指标
└── training_args.bin           # 训练参数的二进制文件，存储了训练过程中使用的参数设置
```

我们在 ./output/checkpoint-500/adapter_config.json文件中看下配置文件：

```
{
  "alpha_pattern": {},                    // alpha参数的模式配置，用于调整LORA层的特定参数
  "auto_mapping": null,                   // 自动映射配置，通常用于自动识别并映射模型的部分配置
  "base_model_name_or_path": "/model_path/chatglm3-6b",  // 基础模型的名称或路径，指定预训练模型的存放位置
  "bias": "none",                         // 偏置配置，指定是否在LORA适配中添加偏置项
  "fan_in_fan_out": false,                // 是否启用fan-in和fan-out初始化方法
  "inference_mode": true,                 // 推理模式，指示模型是否处于推理状态
  "init_lora_weights": true,              // 是否初始化LORA权重，开启后将按照指定方法初始化权重
  "layers_pattern": null,                 // 层模式配置，用于定义哪些层应用特定的模式
  "layers_to_transform": null,            // 需要转换的层列表，指定哪些层应用LORA或其他PEFT技术
  "loftq_config": {},                     // LOFT-Q技术的配置，用于定量化和优化训练过程
  "lora_alpha": 32,                       // LORA的alpha参数，控制学习率的调整
  "lora_dropout": 0.1,                    // LORA适配器中的dropout比率，有助于防止过拟合
  "megatron_config": null,                // Megatron配置，用于大型模型的特定优化设置
  "megatron_core": "megatron.core",       // 指定Megatron库的核心模块路径
  "modules_to_save": null,                // 指定需要保存的模块，用于模型持久化时选择性保存
  "peft_type": "LORA",                    // 指定PEFT（参数高效微调）的类型，这里使用的是LORA
  "r": 8,                                 // LORA的rank参数，决定了适配器矩阵的秩大小
  "rank_pattern": {},                     // rank参数的模式配置
  "revision": null,                       // 版本控制，用于模型的版本管理
  "target_modules": [
    "query_key_value"                     // 目标模块列表，指定LORA适配器应用于哪些模块
  ],
  "task_type": "CAUSAL_LM"                // 任务类型，这里是因果语言模型（Causal Language Model）
}
```

测试：

```
$ python inference_hf.py ./output/checkpoint-500 --prompt "牛仔裤是什么？"
```

效果如下：