大模型的使用与微调方法
——以 Qwen大模型、Lora微调方法 为例

有问题可咨询：孙健（2281766692@qq.com / 15379525328）

一、为什么要使用和微调开源大模型?

- 调用 Chatgpt 的问题（成本、稳定性）
- 预训练大模型的缺陷（幻觉、价值观）
- 领域适配（准确率）
- 企业级数据（隐私性）

二、如何部署开源大模型?

1、所需配置

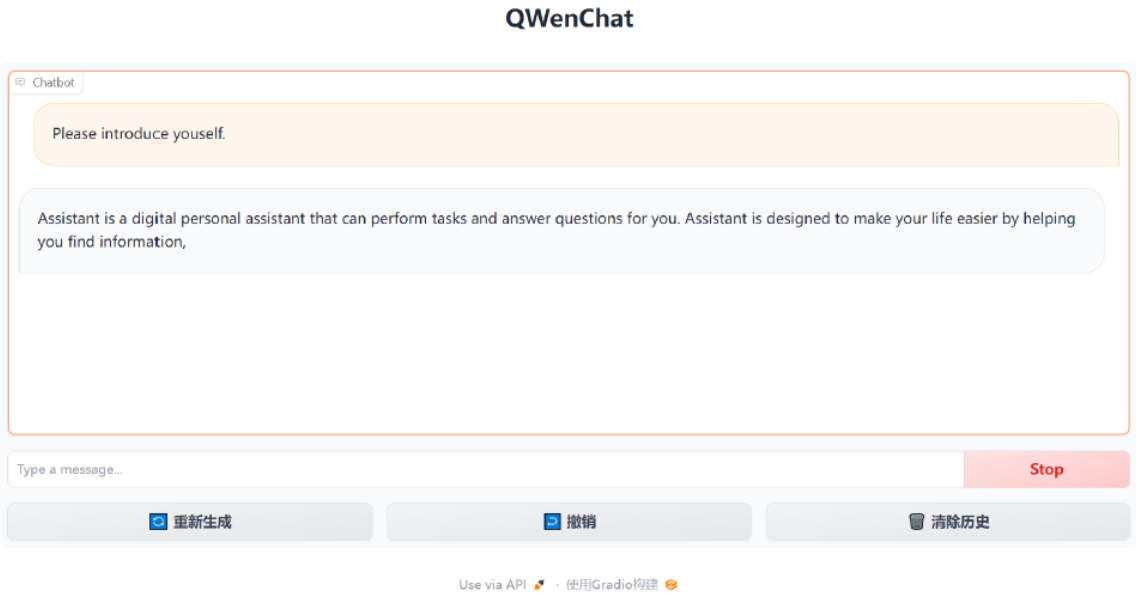
- 硬件：GPU/CPU 服务器
- 软件：Linux系统、python环境

2、部署方式

2.1 接口调用

```
# 示例：调用openai的gpt3.5接口

import openai
response = openai.ChatCompletion.create(
        engine="nova-gpt-35-16",
        temperature=0,
        max_tokens=1000,
        **kwargs,
        request_timeout=60,
    )
```

2.2 web应用



3、开源大模型社区

魔搭社区：https://www.modelscope.cn/my/overview

huggingface：https://huggingface.co/

三、如何微调开源大模型?

1、微调方法、原理（略）、特点

- 全参数微调（效果？成本？适用情形？）
- Lora（优势？）
- QLora（优势？）

2、如何用Lora/QLora进行微调？（工具介绍）

2.1 开源模型的加载

工具:

- modelscope：魔搭社区提供的开源工具，可用于开源模型下载
- transformers：大模型开源工具库， 提供了模型加载、训练、生成等工具和方法

```
from modelscope import snapshot_download
from transformers import AutoModelForCausalLM, AutoTokenizer

# 从魔搭社区中下载开源模型
model_dir = snapshot_download('qwen/Qwen-7B', revision='v1.1.4',
cache_dir='/data/sunjian/Model/Qwen_7B')

# 加载分词器和模型参数
```

```
tokenizer = AutoTokenizer.from_pretrained(model_dir, trust_remote_code=True)
         # 分词器
model = AutoModelForCausalLM.from_pretrained(
    model_dir,
    device_map="auto",      # 自动将模型参数加载到GPU上
    offload_folder="offload",
    trust_remote_code=True
).eval()
       # 模型

# 配置模型生成参数（开源模型一般会附带generation-config文件）
model.generation_config = GenerationConfig.from_pretrained(model_dir,
trust_remote_code=True)    # 生成参数
# 调用模型进行对话生成
response, history = model.chat(tokenizer, "你好", history=None)
response, history = model.chat(tokenizer, "浙江的省会在哪里？", history=history)
response, history = model.chat(tokenizer, "它有什么好玩的景点", history=history)
```

2.2 模型微调

工具:

- transformers.BitsAndBytesConfig：模型参数类型配置

量化方式介绍：https://juejin.cn/post/7291931852800524329

```
from transformers import BitsAndBytesConfig
from transformers import AutoModelForCausalLM, AutoTokenizer
# 配置量化相关参数
quantization_config = BitsAndBytesConfig(
            load_in_4bit=True,              # 模型量化加载方式
            bnb_4bit_compute_dtype=torch.bfloat16,
            bnb_4bit_use_double_quant=True,
            bnb_4bit_quant_type="nf4",
)
# 使用量化方式加载模型
model = model_cls.from_pretrained(
        model_name_or_path, #模型地址
        device_map=device_map,  #模型参数与GPU映射方式
        torch_dtype=model_dtype,  #模型参数类型
        quantization_config=quantization_config,  #量化配置参数
        trust_remote_code=True,
)
```

优势？劣势？

工具：

- peft：用于Lora、QLora训练方法的工具库，简单易用

```
from peft import PeftModel
from peft import prepare_model_for_kbit_training

# 使用Qloar方式，对模型原始参数做量化处理
model = prepare_model_for_kbit_training(model)                # QLora方式加载
# 配置Lora的训练参数
lora_config = LoraConfig(
    r=8,                                    #控制Lora参数的维度k
    lora_alpha=16,
    lora_dropout=0.05,
    target_modules=target_modules,        #控制Lora参数的类型数量n
    task_type=lora_task_type,
)
# 生成Lora参数，加载model-lora模型
model = get_peft_model(model, lora_config)
```

工具：

- datasets：训练数据预处理、批处理工具

```
from datasets import load_dataset
# 数据预处理，处理成特殊格式
def generate_and_tokenize_prompt(data_point):
    input_text = '你现在是一个要素提取机器人，请根据要求从句子中提取出对应的关键元素。未发现对应的元素则不返回。\n<eoh>\n'
    input_text += '<|instruction|>:\n' + data_point['instruction'] + '\n<eoh>\n'
    input_text += '<|input|>:\n' + data_point['input'] + '\n<eoh>\n'
    # input_text += '<|Output|>:\n' + data_point['output'] + '\n<eoa>\n'
    input_text += '<|Output|>:\n' + data_point['removed_output'] + '\n<eoa>\n'

    full_prompt = input_text[:max_len]
    tokenized_full_prompt = tokenize(full_prompt)
    return tokenized_full_prompt

data = load_dataset("json", data_files=data_path, split='train')
data = data.map(generate_and_tokenize_prompt, num_proc=num_proc)
```

工具：

- transformers.Trainer：深度学习模型训练器，自动完成模型训练过程

```
from transformers import Trainer
#加载trainer训练器，自动配置优化器，数据集等
trainer = Trainer(
    model=model,
    train_dataset=train_data,
    eval_dataset=val_data,
```

```
    args=training_args,
    data_collator=DataCollatorForSeq2Seq(tokenizer,
                                         pad_to_multiple_of=8,
                                         return_tensors="pt",
                                         padding=True)
)
#开启训练过程
trainer.train(resume_from_checkpoint=False)
#保存模型参数
model.save_pretrained(training_args.output_dir)                    # 仅保存Lora参数
```

2.3 微调模型的使用

工具:

- PeftModel：Lora模型加载工具，自动完成原模型参数、Lora参数的merge处理

```
from peft import PeftModel
#加载量化后的模型。（以什么方式训练，就要以什么方式加载）
model = prepare_model_for_kbit_training(model)
base_model = AutoModelForCausalLM.from_pretrained(
    model_name_or_path,
    device_map=device_map,
    torch_dtype=model_dtype,
    quantization_config=quantization_config,
    trust_remote_code=True,
)
# 加载model-lora。先加载模型原始参数，然后加载lora参数。
model = PeftModel.from_pretrained(base_model, load_lora_parameter_dir,
device_map='auto')

model.eval()                                                       # 测试模式  与
model.train()的差异
```

工具:

- model.generate、model.stream_generate：大模型生成调用工具，自动完成文本编码、模型调用生成、特殊token处理、编码解码等过程

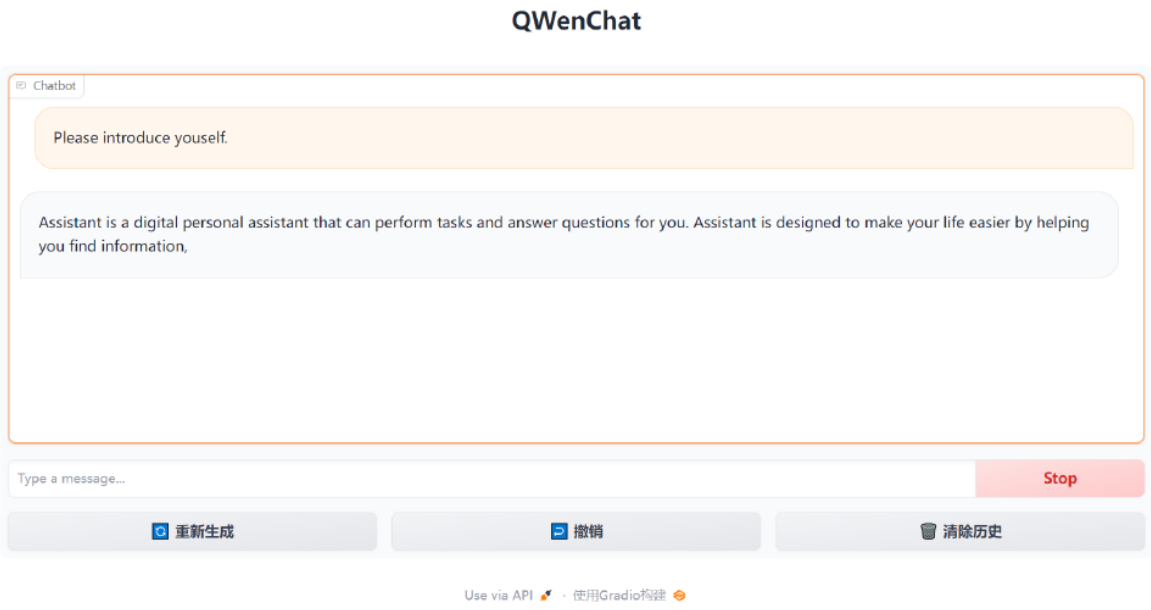
```
from transformers import GenerationConfig
# 将输入文本进行编码
def tokenize_for_inference(prompt):
    inputs = tokenizer(prompt, return_tensors='pt')input_ids = inputs[
input_ids'].cuda()
    return input ids
# 按照训练方式，处理输入文本数据

def generate_and_tokenize_prompt_forinference(data_point):
    input_text "你现在是一个要素振取机器人，请根据要求从句子中报取出对应的关键元素。未发现对应
的元素则不返回。"
    input_text +="<|Output|>:\n"
    tokenized_prompt = tokenize_for_inference(input_text)
```

```
    return tokenized_prompt
# 配置生成参数
generate_args = GenerationConfig(do_sample=True)
one_data="这是一个要案提取的示例"
input_ids = generate_and_tokenize-prompt_for_inference(one_data)
# 模型生成并进行解码操作
with torch.no_grad():
    s = model.generate(input_ids=input_ids, generation-config=generate_args)
    output = tokenizer.decode(s[0],skip_special_tokens=True)
```

工具:

- gradio：大模型对话工具，自动搭建前后端，可用于构建简易的大模型交互界面，也可自动生成大模型调用接口。

```
import gradio as gr
# 配置gradio demo示例
demo = gr.ChatInterface(
    predict, title=tile,
    description=description
).queue()
# 运行demo示例
demo.launch(
    server_port=server_port,
    server_name=server_name
)
```



3、如何提升模型微调的效果?

- 选用参数量更大的模型，或者新的模型（qwen-7B、qwen-14B、qwen-72B、qwen1.5）
- 选用效果更好的微调方法（Lora、Qlora、全参数微调）
- 提升数据质量、多样性（数据收集、清洗）
- 调整微调参数

```
learning_rate                   # 学习率
gradient_accumulation_steps         # 参数更新频次
num_train_epochs                    # 训练轮数

quantization                        # 量化方式
max_len                             # 最大长度

lora_r                             # Lora参数-维度
lora_alpha                         # Lora参数
lora_dropout                       # Lora参数-过拟合
```

- 改进提示模板（step-by-step）

```
prompt = '你现在是一个信息抽取模型，请你帮我从所给句子中，抽取出所有的关系三元组。'


prompt = '你现在是一个信息抽取模型，请你帮我从所给句子中，抽取出所有的关系三元组。' \
         '请按步骤来完成。第一步是找出句子中所有的实体，并分析出实体对应的类型。' \
         '第二步是结合句子，找出实体之间对应的关系类型，并得到关系三元组。用逗号分隔开。\n' \
```

- 调整训练策略（读论文） （多任务混合训练、多轮问答）
  OpenCodeInterpreter：https://arxiv.org/abs/2402.14658

4、其他内容

- 显存消耗参考

GPU数量	加载方式	lora_r	para_size	max_len	GPU消耗	s/it
1	4bit	8	700MB	1800	30G	420
1	4bit	8	700mb	3600	43.89G	900
1	8bit	8	700mb	900	42G	128
1	4bit	8	700mb	900	25.6G	225
1	4bit	32	700mb	900	26.9G	227
1	4bit	8	248MB	1300	26.51	300

- 节省显存的技巧
  1. 直接使用量化版本的开源模型
  2. 筛除冗余数据，减少 tokens 数量（即max_len）
  3. 检查点 checkpoint 策略

- 如何提高训练速度
  1. 使用多GPU，并行训练。
  2. 调整参数 gradient_accumulation_steps
  3. 使用 FlashAttention （开源微调框架 llama-factory 等）
- 如何评估模型的效果

生成式任务： BLEU score（文本翻译），Perplexity（问答任务），**modelEval**

分类任务：准确率（关系抽取）

专项任务评估：准确率，比如代码能力评估（humaneval数据集）

四、微调模型？ OR 微调提示模板？

- 是否有足够的微调数据？
- 能否通过微调提示模板解决问题？
- 其他策略：RAG

- 学习参考

  github：https://github.com/ （找训练框架）

  huggingface：https://huggingface.co/ （找模型、数据）

  modelscope：https://www.modelscope.cn/models （找模型、数据）

  AutoDL：https://www.autodl.com/home （租用服务器）

demo演示：https://finchat.sufe.edu.cn/t_expert/