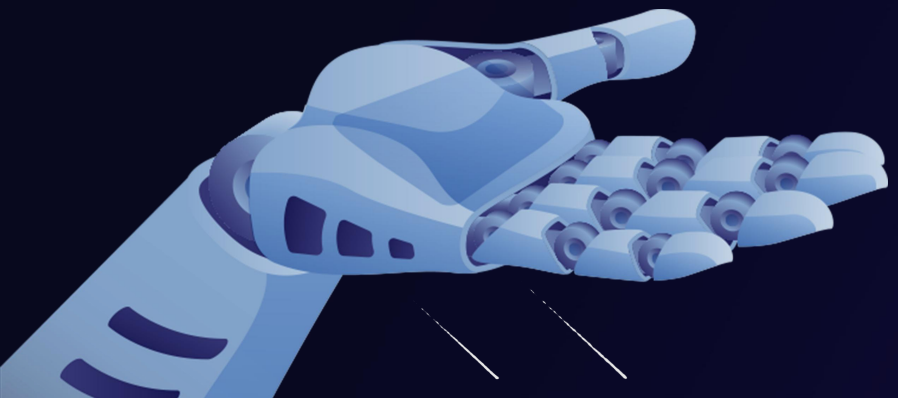
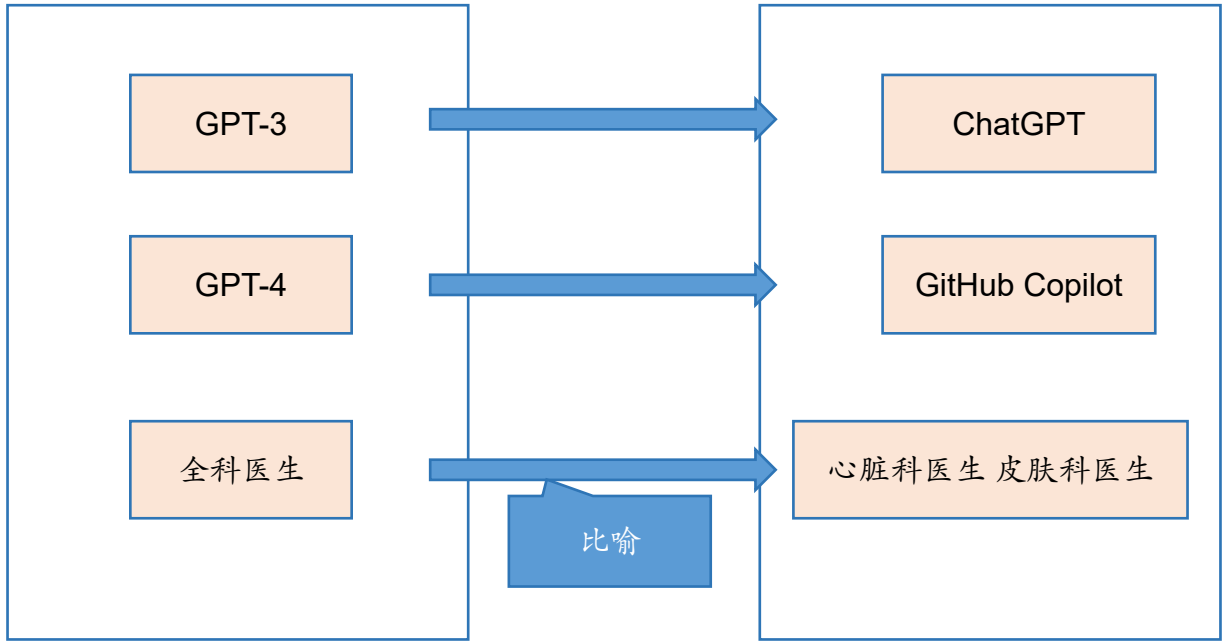
第七章：大模型微调技术及医疗问诊机器人实战



CONTENTS

7.1 大模型微调相关原理：定义

- 预训练是用大量数据训练模型，使其具备广泛的知识。
- 微调是用特定任务的数据进一步训练，优化模型表现。



7.1 大模型微调相关原理：分类

自然语言处理领域的微调分为两大部分，一是非生成类微调，常见的以BERT为预训练模型进行分类、回归、命名实体识别（NER）任务的微调。二是生成类微调，常见以GPT等大语言模型为预训练模型进行自监督、问答、多轮会话等微调。

非生成类微调

分类：如情感分析、文本分类等。

回归：如预测数值、分数等。

命名实体识别（NER）：识别和分类文本中的实体。

前面已学过

生成类微调

自监督：利用无标签数据进行模型训练，提升模型的泛化能力。

问答：如机器阅读理解、开放域问答等。

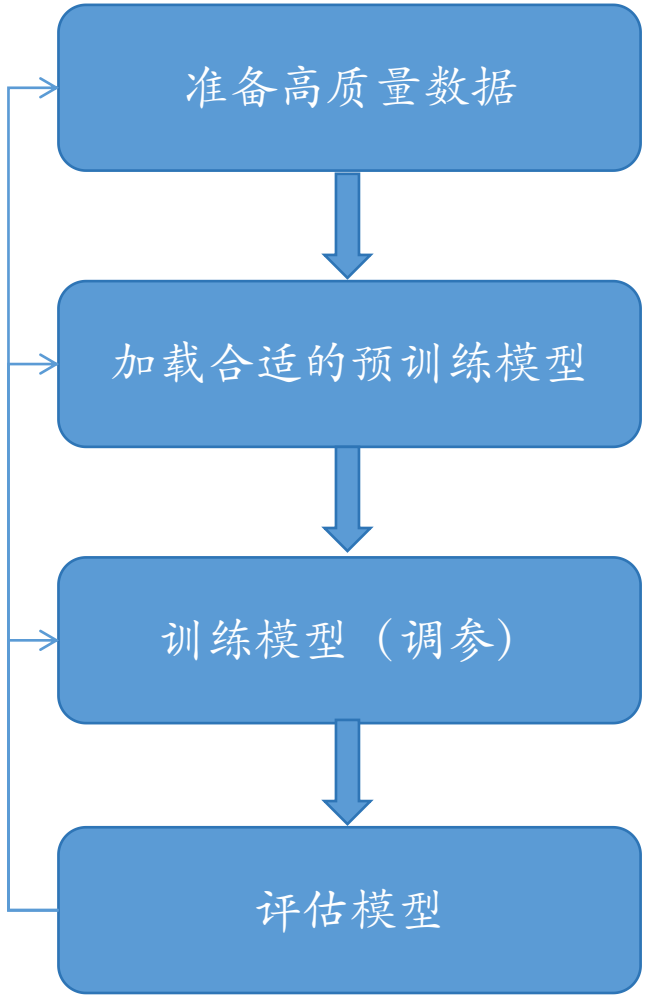
多轮会话：实现自然语言对话系统的连续对话能力。

重点学习

7.1 大模型微调相关原理：作用

	Prompting	VS	Finetuning
优势	直接使用，不用数据 前期成本低 不需要技术知识 使用检索增强（RAG）		可用数据不限 学习到新内容 减少幻觉 增加一致性 减少无用输出 小模型后期成本低 使用检索增强（RAG）
劣势	可使用数据少 存在数据遗忘 存在幻觉 rag未检索到有用信息		需要高质量数据 前期计算成本高 需要知识储备，尤其是领域数据背景知识

7.1 大模型微调相关原理：过程



- 数据充足（1000+）
- 覆盖全面
- 内容准确

- 自监督模式：一段文本。
- 问答模式：问答对。
- 多轮会话模式：多轮对话。

- Llama系列：Llama3.1 405B, 70B & 8B; Llama3.2 1B、3B、11B & 90B。
- 通义千问系列：Qwen2.5 7B & 72B。
- mistral系列：Mistrai 7B、8 X 7B、8 X 22B; Pixtral12B; Mathstral；Codestral Mamba。

GPU
资源紧张

数据量小
模型参数少
量化精度
LORA调参

数据量大
模型参数多
正常精度
全量调参

GPU
资源丰富

- 人工评估准确性
- 评估数据覆盖全面
- 避免与训练数据重复

7.2 模型量化和LORA技术

模型量化

什么是量化

大模型的量化（Quantization）是减少模型参数的表示精度的一种技术。这种技术可以显著减少模型的存储空间和计算资源需求，同时保持模型性能。能够适应资源受限情况下的推理和微调需求。

量化的分类

FP32 (32-bit Floating Point)：标准的浮点数表示方式，精度高，但存储和计算成本较大。

FP16 (16-bit Floating Point)：半精度浮点数，存储和计算成本比FP32低一半，适用于模型训练和推理时的计算加速，同时可以保持较高的精度。

INT8 (8-bit Integer)：8位整数表示，大大减少了存储和计算需求。适用于推理阶段，但可能会引入一定的量化误差，需要通过校准和优化技术来减小误差对模型性能的影响。

INT4 (4-bit Integer)：进一步降低存储和计算需求，适用于资源非常受限的设备，但量化误差可能更明显，需要谨慎应用。

BFloat16：与FP16类似，但在表达范围上与FP32更接近，适用于特定的深度学习任务，能在保持较好精度的同时提供加速效果。

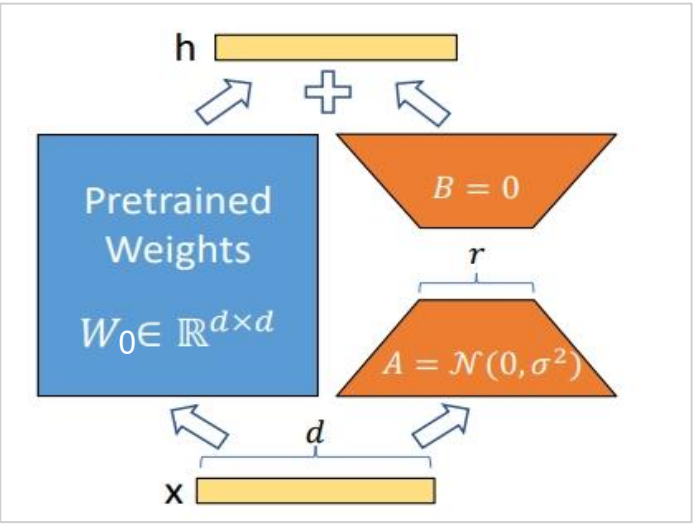
LORA微调

LORA微调是指将预训练模型参数W分解到低秩空间A和B，通过仅更新低秩空间的参数，在降低微调参数数量的情况，最大程度地接近全参数微调的效果。

$h=W_0x+Wx=W_0x+ABx$

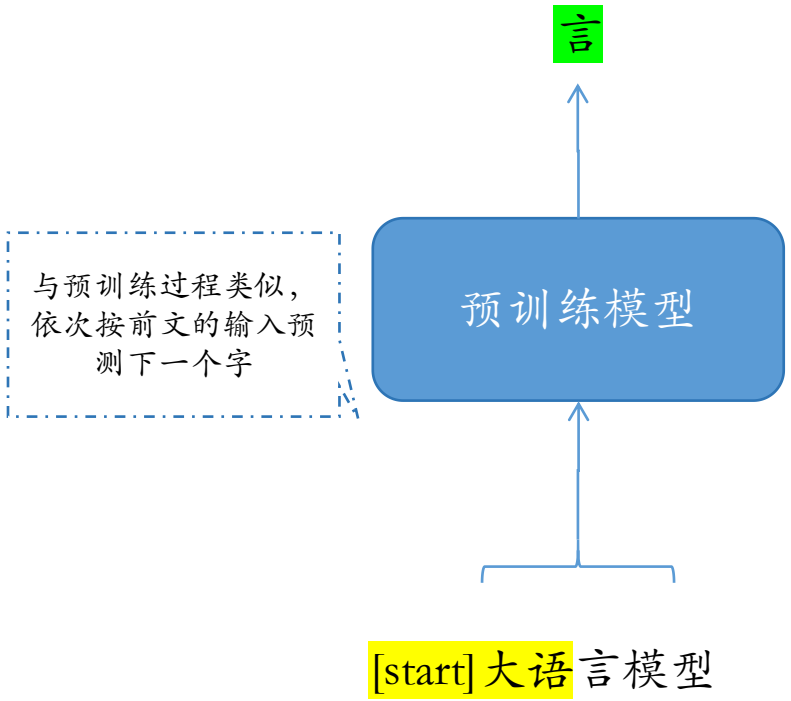
$W^{(d\times d)}=A^{(d\times r)}\times B^{(r\times d)}$

其中$r << d$。



右图为几种常见的微调方法对比，与全参数微调相比，LORA微调达到了和后者一样的结果。

7.3 自监督模式微调



安装并导入相关包

```
!pip install datasets
!pip install trl
```

自监督模式微调方法：输入一段文字，对每个进行预测。

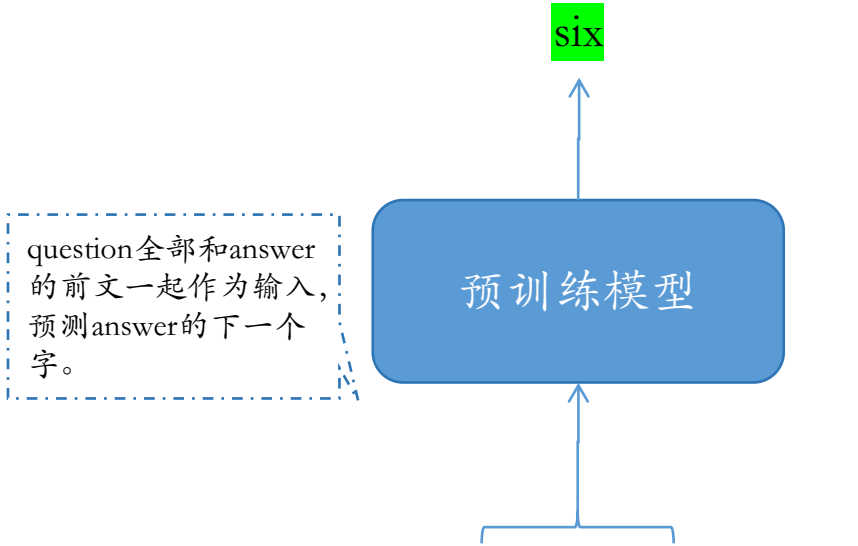
```
from datasets import load_dataset
from transformers import AutoModelForCausalLM
from trl import SFTConfig, SFTTrainer
```

```
dataset = load_dataset('/imdb')
```

```
small_train_dataset = dataset["train"].select(range(10))
small_eval_dataset = dataset["test"].select(range(10))
```

```
model = AutoModelForCausalLM.from_pretrained("facebook/opt-350m")

training_args = SFTConfig(
        eval_strategy="epoch",
        dataset_text_field="text",
        max_seq_length=512,
        output_dir="/tmp",
        logging_steps=1,
)
trainer = SFTTrainer(
        model,
        train_dataset=small_train_dataset,
        eval_dataset=small_eval_dataset,
        args=training_args,
)
trainer.train()
```

7.4 问答模式微调



问答模式微调方法：question: text1, answer: text2, 模型对text2的每个字进行预测。

```
[2] from transformers import AutoModelForCausalLM, AutoTokenizer
    from datasets import load_dataset
    from trl import SFTConfig, SFTTrainer, DataCollatorForCompletionOnlyLM
```

```
[3] dataset = load_dataset("lucasmccabe-lmi/CodeAlpaca-20k")
```

```
[8] dataset1 = dataset['train'].train_test_split(test_size=0.3)
    small_train_dataset = dataset1['train'].select(range(20))
    small_val_dataset = dataset1['test'].select(range(20))
```

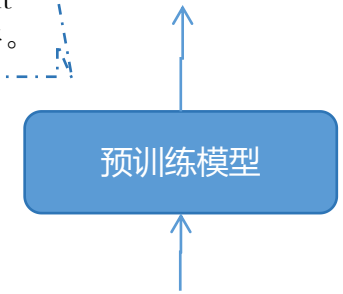
```
[9] model = AutoModelForCausalLM.from_pretrained("facebook/opt-350m")
    tokenizer = AutoTokenizer.from_pretrained("facebook/opt-350m")
```

```
def formatting_prompts_func(example):
        output_texts = []
        for i in range(len(example['instruction'])):
                text = f"### Question: {example['instruction'][i]}\n ### Answer: {example['output'][i]}"
                output_texts.append(text)
        return output_texts

response_template = " ### Answer:"
collator = DataCollatorForCompletionOnlyLM(response_template, tokenizer=tokenizer)

training_args = SFTConfig(
        eval_strategy="epoch",
        output_dir="/tmp",
        logging_steps=1,
)

trainer = SFTTrainer(
        model,
        train_dataset=small_train_dataset,
        eval_dataset=small_val_dataset,
        args=training_args,
        formatting_func=formatting_prompts_func,
        data_collator=collator,
)
trainer.train()
```

7.5 多轮会话模式微调

前面所有会话内容作为输入，预测当前assistant部分的每个字。

six



user: What is your name?
assistant: My name is john.
user: How old are you?
assistant: I am six years old.

对话模式语言模型：输入human: text1, assistant: text2, human: text3, assistant: text4, human: text5, assistant: text6, ..., 模型对assistant对应的text的每个字进行预测。

```
[10] from transformers import AutoModelForCausalLM, AutoTokenizer
     from datasets import load_dataset, DatasetDict
     from trl import SFTConfig, SFTTrainer, DataCollatorForCompletionOnlyLM
     import re
     import random
     from multiprocessing import cpu_count
```

```
[11] from huggingface_hub import login
     login(token="here is your auth token")
```

```
[12] tokenizer = AutoTokenizer.from_pretrained('meta-llama/Llama-3.2-1B-Instruct')
     model = AutoModelForCausalLM.from_pretrained('meta-llama/Llama-3.2-1B-Instruct')
```

```
     # 加载数据
     dataset = load_dataset("HuggingFaceH4/ultrachat_200k")
     dataset = DatasetDict({
         "train": dataset["train_sft"],
         "test": dataset["test_sft"]
     })
```

```
[15] # 处理数据
     chat_template =
     "{% for message in messages %}\n{% if message['role'] == 'user' %}\n{{ '<|user|>\n' + message['content'] + eos_token }}\n
     {% elif message['role'] == 'system' %}\n{{ '<|system|>\n' + message['content'] + eos_token }}\n
     {% elif message['role'] == 'assistant' %}\n{{ '<|assistant|>\n' + message['content'] + eos_token }}\n
     {% endif %}\n{% if loop.last and add_generation_prompt %}\n
     {{ '<|assistant|>' }}\n{% endif %}\n{% endfor %}"
     tokenizer.chat_template = chat_template
     def apply_chat_template(example, tokenizer):
         messages = example["messages"]
         # We add an empty system message if there is none
         if messages[0]["role"] != "system":
             messages.insert(0, {"role": "system", "content": ""})
         example["text"] = tokenizer.apply_chat_template(messages, tokenize=False)
         return example
     dataset1 = dataset.map(
                         apply_chat_template,
                         num_proc=cpu_count(),
                         fn_kwargs={"tokenizer": tokenizer},
                         remove_columns=list(dataset["train"].features),
                         desc="Applying chat template",
                 )
     small_train_dataset = dataset1["train"].select(range(10))
     small_test_dataset = dataset1["test"].select(range(10))
```

```
[ ] tokenizer.pad_token = tokenizer.eos_token
```

```
[ ] # 训练数据
    training_args = SFTConfig(
            eval_strategy="epoch",
            output_dir="/tmp",
            overwrite_output_dir=True,
            dataset_text_field="text",
            packing=True,
            logging_steps=1,
            per_device_train_batch_size=1,
            per_device_eval_batch_size=1,
    )

    trainer = SFTTrainer(
                    model=model,
                    args=training_args,
                    train_dataset=small_train_dataset,
                    eval_dataset=small_test_dataset,
                    tokenizer=tokenizer,
    )

    trainer.train()
```

7.6 医疗问诊机器人项目实战

安装与导入必要包

```
!pip install datasets
!pip install trl
!pip install -U bitsandbytes
```

```
from transformers import AutoModelForCausalLM, AutoTokenizer
from datasets import load_dataset, DatasetDict
from trl import SFTConfig, SFTTrainer
import re
import random
from multiprocessing import cpu_count
from huggingface_hub import login
from google.colab import drive
import shutil
import torch
from transformers import pipeline
from peft import LoraConfig
```

加载预训练模型

```
login(token="here is your token")
tokenizer = AutoTokenizer.from_pretrained('meta-llama/Llama-3.1-8B-Instruct')
model = AutoModelForCausalLM.from_pretrained('meta-llama/Llama-3.1-8B-Instruct', torch_dtype=torch.float16)
```

加载与处理数据

```
data = load_dataset("Flmc/DISC-Med-SFT")
data1 = data.rename_column('conversation', 'messages')
data1 = data1['train'].filter(lambda x: x['source'] == 'meddial')
data1 = data1.train_test_split(test_size=0.3)
data2 = DatasetDict({
    "train": data1['train'],
    "test": data1['test']
})
data2
```

```
# 按模板调整
chat_template = "
{% for message in messages %}\n{%
if message['role'] == 'user' %}\n{{ '<|user|>\n' + message['content'] + eos_token }}\n
{% elif message['role'] == 'system' %}\n{{ '<|system|>\n' + message['content'] + eos_token }}\n{%
elif message['role'] == 'assistant' %}\n{{ '<|assistant|>\n' + message['content'] + eos_token }}\n{%
endif %}\n{% if loop.last and add_generation_prompt %}\n{{ '<|assistant|>' }}\n{% endif %}\n{% endfor %}"
tokenizer.chat_template = chat_template
def apply_chat_template(example, tokenizer):
    messages = example["messages"]
    # We add an empty system message if there is none
    if messages[0]["role"] != "system":
        messages.insert(0, {"role": "system", "content": ""})
    example["text"] = tokenizer.apply_chat_template(messages, tokenize=False)
    return example

data3 = data2.map(
                    apply_chat_template,
                    num_proc=cpu_count(),
                    fn_kwargs={"tokenizer": tokenizer},
                    remove_columns=list(data2["train"].features),
                    desc="Applying chat template",
                )

small_train_dataset = data3["train"].select(range(1000))
small_test_dataset = data3["test"].select(range(1000))
```

训练模型

```
tokenizer.pad_token = tokenizer.eos_token
# 训练数据
training_args = SFTConfig(
    eval_strategy="epoch",
    output_dir="/content/model/",
    overwrite_output_dir=True,
    dataset_text_field="text",
    packing=True,
    logging_steps=1,
    per_device_train_batch_size=1,
    per_device_eval_batch_size=1,
    save_strategy="epoch",
)
peft_config = LoraConfig(
    r=16,
    lora_alpha=32,
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM",
)
trainer = SFTTrainer(
            model=model,
            args=training_args,
            train_dataset=small_train_dataset,
            eval_dataset=small_test_dataset,
            tokenizer=tokenizer,
            peft_config=peft_config,
)
trainer.train()
```

7.6 医疗问诊机器人项目实战

微调前效果

```
# sft前
model_id = "meta-llama/Llama-3.2-1B-Instruct"
pipe_before = pipeline(
        "text-generation",
        model=model_id,
        torch_dtype=torch.bfloat16,
        device_map="auto",
)
```

```
[17] messages = [
        {"role": "system", "content": "您是一个专业的医生，回答我提出的医疗问题。"},
        {"role": "user", "content": "医生，我全身疼痛，发烧38.5，还有咳嗽，请问该咋办？"},
    ]
    outputs = pipe_before(
        messages,
        max_new_tokens=256,
    )
    print(outputs[0]["generated_text"][-1])
```

微调后效果

```
# sft后
model_id = "/content/model/checkpoint-1521/"
pipe = pipeline(
        "text-generation",
        model=model_id,
        torch_dtype=torch.bfloat16,
        device_map="auto",
)
```

```
[15] messages = [
        {"role": "system", "content": "您是一个专业的医生，回答我提出的医疗问题。"},
        {"role": "user", "content": "医生，我全身疼痛，发烧38.5，还有咳嗽，请问该咋办？"},
    ]
    outputs = pipe(
        messages,
        max_new_tokens=256,
    )
    print(outputs[0]["generated_text"][-1])
```

感谢您的观看