

在 Amazon SageMaker 上微调 Llama 2

2023 年 7 月 19 日，Meta 发布了开源的免费可商用大模型 LLaMA 2，一时撼动世界。本文将对话机器人作为目标，给出一套在单块 A10 GPU 上微调 Llama 2 的 LoRA [方案](<https://llm-sft-test-z6hw.notebook.us-east-1.sagemaker.aws/lab/tree/drx/solu-2.ipynb>)。

背景

Llama 2 简介

版本 Llama 2 是一系列模型的统称，按参数量可分为 70 亿、130 亿、700 亿三种规模。每种规模下，Meta 又根据是否进行了“面向聊天场景”的有监督微调（SFT），及“面向安全性、帮助性”的人类反馈强化学习（RLHF），提供了两个模型。访问它们的一个便利的接口是 [HuggingFace] (<https://huggingface.co/meta-llama>)。

特点

- 训练数据量庞大，可达 2T
 - 适于多轮对话场景
 - 使用分组查询注意力（GQA），组合处理多条查询
 - 使用 GAtt 方法，加强对==系统指令==的关注
- 系统指令例如：请仅仅使用 Emoji 回复未来的所有问题。
- 多目标，同时考量精炼性、安全性（以多样性或创造性为代价）、帮助性
 - 法律知识丰富，代码能力一般
 - 几乎不支持中文 :cn:

效果 在标准评估数据集中，Llama 2 基本能打败其他规模相当的开源模型；在人类评估中，Llama-2-70b-chat-hf 的能力介于 ChatGPT 和 GPT-3.5 之间。

LoRA 简介

预训练

大语言模型（Large Language Model, LLM）一般有十亿至千亿个参数，它们的预训练过程需要大量的数据和计算资源，成本高昂。

经过了预训练阶段的模型，被称为预训练语言模型（Pre-trained Language Model, PLM），在各种任务上的都具备一定能力，其最终目标是成为通用人工智能（Artificial General Intelligence, AGI，即可以像人类一样执行各种任务的 AI）。

微调

预训练的成本固然很高，但幸运的是，大多数时候我们并不需要自己预训练一个模型，而只需选择一个 PLM，微调（fine-tune）它的参数，以提高它处理某些下游任务（downstream task，如机器翻译、文本情感分类）的能力。

我们选择 Llama 2 等开源 PLM，就是因为其参数是公开可访问的，进而可以加载和调整。

LoRA

尽管不需要从零开始预训练一个大模型了，但如此多的参数，就算是做微调，也需要大量的存储和计算资源。

幸运的是，在资源有限时，我们亦不必进行全量微调参数高效微调（Full Fine-Tuning），而只需参数高效微调（Parameter-Efficient Fine-Tuning, PEFT），仅仅轻量级地调整少部分或额外的参数，并且不会显著影响微调的效果。

接下来即将使用的 Low-Rank Adaptation of Large Language Models (LoRA) 就是一种轻量级微调的技术。它会为原始的（不参与训练的）权重矩阵，叠加上一个更新矩阵，通过将更新矩阵分解为一个（可训练的）升维矩阵和一个（可训练的）降维矩阵的乘积，减少了参与训练的参数量。

1. 准备工作

在正式开始微调之前，我们需要安装代码中调用到的工具包。此外，我们还需要申请访问 Llama 的权限。

1.1 安装环境

在创建 Amazon SageMaker 笔记本后，我们可以为其配置一个单独的 kernel，以免和其他训练任务用到的环境发生版本冲突。具体步骤可以是：

创建一个新 Terminal，利用 conda 工具创建一个临时环境

```
1 conda create --name your_env_name python=3.10
2 conda install conda=23.7.2
3 conda activate base
4 conda activate your_env_name
5 conda install ipykernel
```

继续在 Terminal 中，利用 pip 工具安装需要使用的工具包

```
1 pip install huggingface_hub
2 pip install transformers
3 pip install accelerate peft bitsandbytes trl # 简化训练
4 pip install sentencepiece # 切词
5 pip install scipy
```

为避免歧义行为，不建议使用在笔记本代码块中 `!pip install xxx` 的方式

1.2 访问模型

Llama 2 虽然是开源模型，但 Meta 制定了专属的开源协议，我们需要通过官网申请访问权限，具体流程应该只需 1-2 天：

首先，在 [Meta](<https://ai.meta.com/resources/models-and-libraries/llama-downloads/>) 的网站上，为你的 :hugs: 邮箱申请访问 Llama 2 的权限

此后，在 [HuggingFace](<https://huggingface.co/meta-llama/Llama-2-7b-hf>) 上，申请通过 :hugs: 访问 Llama 2 的权限

最后，回到笔记本中，用 token 登录 HuggingFace 即可

```
1 !huggingface-cli login --token hf_xxx_your_token_xxx
```

2. 微调 Llama-2-7b

如简介所述，微调的目的是让模型在某些任务上表现更好，此处以对话助手为目标，使用公开数据集，采用 LoRA 技术，给出微调 Llama-2-7b 的一个示例，并以对比微调前后的对话效果。

2.1 数据集准备

严格选控数据集的质量、多样性，才能保证微调效果。

首先需要选用（和清洗）数据集。[LLaMA 2 论文](<https://arxiv.org/pdf/2307.09288.pdf>) 表明，原始模型在 SFT 阶段的数据集约为 27K 条高质量的“Prompt-Response”对。这里我们用的[数据集](<https://huggingface.co/datasets/timdettmers/openassistant-guanaco>)选自人类生成、人类标注的 OpenAssistant 数据集（具有主题和用语的多样性，并且有一定的[语言多样性](<https://huggingface.co/datasets/OpenAssistant/oasst1/blob/main/README.md#languages>))，是华盛顿大学用以训练 Guanaco 模型的，有 9.85K 条样本，每个样本包括两轮左右对话，形如

```
#### Human: Please write a poem about the 21st century in the style of comedy#### Assistant:
It's the 21st century, and I've got to say, things aren't all, in the right way. There's so much
sanitization, washing your hands is basic information, and toilets for all to see. Why oh why
can't things, be like they were before, with outhouses and sewage, foul smelly air and more?
Things should go back, to how they were before, with people in outhouses, and without
thought for anything more.#### Human: Well done.#### Assistant: Thank you. Is there
anything else that I can help you with?"
```

与原始论文 SFT 阶段的数据集体量相近、质量相当。读者可以根据自己的需求选用和清洗数据集。

openassistant-guanaco 数据集可以通过 HuggingFace 的接口简洁地加载

```
1 from datasets import load_dataset
2
3 dataset = load_dataset("timdettmers/openassistant-guanaco", split="train")
```

2.2 配置 LoRA 参数

本节将调用封装度较高的训练工具，配置 LoRA 训练相关的参数。具体步骤如下：

资源调度策略。示例选用了四块 A10 GPU 资源中的首块，读者可根据实际情况做调整

```
1 import os
2
3 os.environ["CUDA_VISIBLE_DEVICES"] = "0" # 仅允许调度首块 GPU
```

加载模型。需要特别注意的是，Llama 2 模型占空间较大，开发者可以根据实际情况，修改下载模型的位置（如 15 行所示），及其他参数

```
1 import torch
2 from transformers import AutoModelForCausalLM, BitsAndBytesConfig, AutoTokenizer
3
4 model_id = "meta-llama/Llama-2-7b-hf" # HuggingFace 上的空间名/数据集名
5
6 bnb_config = BitsAndBytesConfig(
7     load_in_4bit=True, # 线性层用 4 比特
8     bnb_4bit_use_double_quant=True, # 嵌套量化，节省内存，且几乎不损失效果
9     bnb_4bit_quant_type="nf4", # 用 4 比特表示浮点数
10    bnb_4bit_compute_dtype=torch.bfloat16 # 计算数据类型，可以不同于输入数据类型，加
11 )
12
13 model = AutoModelForCausalLM.from_pretrained( # Llama 2 为 casual LM, 相比 maske
14     model_id,
15     cache_dir="/your/path/to/cache", # 模型下载位置，约占 85 G
16     quantization_config=bnb_config,
17     device_map="auto" # 自动调配可用资源
18 )
```

```

19
20 tokenizer = AutoTokenizer.from_pretrained( # 处理输入 (将字符串映射进标识空间)
21     model_id,
22     trust_remote_code=True # 允许使用仓库中的代码
23 )

```

为 LoRA 训练做准备

```

1 from peft import prepare_model_for_kbit_training
2
3 model.gradient_checkpointing_enable() # 启用梯度检查点，节省内存，以训练时间为代价
4 model = prepare_model_for_kbit_training(model) # 为 LoRA 训练做准备

```

定义查看参数规模的函数

```

1 def print_trainable_parameters(model):
2     trainable_params = 0
3     all_param = 0
4     for _, param in model.named_parameters():
5         all_param += param.numel()
6         if param.requires_grad:
7             trainable_params += param.numel()
8
9     print(f"trainable params: {trainable_params} | | all params: {all_param} | |")

```

使用 LoRA 简化模型，并查看可训练参数规模，可以发现仅有 10% 左右的参数会参与训练，成功实现了轻量化

```

1 from peft import LoraConfig, get_peft_model
2
3 config = LoraConfig(
4     r=8, # 可更新的矩阵的维数
5     target_modules=["q_proj", "v_proj"], # 可更新的模块
6     lora_alpha=32, # LoRA 低秩矩阵的缩放系数，类似于学习率
7     lora_dropout=0.05, # LoRA 层的丢弃率
8     bias="none", # 每个节点的偏移是否可训练

```

```

9     task_type="CAUSAL_LM"
10 )
11 model = get_peft_model(model, config)
12
13 # 检验可训练参数量是否成功缩小了
14 print_trainable_parameters(model) # 应为 10% 左右

```

2.3 配置训练参数

在正式开始训练前，只剩为 `SFTTrainer` 配置训练参数这一步骤了，读者可以根据实际情况，调整 `per_device_train_batch_size`，`max_steps` 等参数

```

1 import transformers
2 from trl import SFTTrainer
3
4 tokenizer.pad_token = tokenizer.eos_token # 忽略句子结束符
5
6 trainer = SFTTrainer(
7     model=model,
8     train_dataset=dataset,
9     dataset_text_field="text",
10    args=transformers.TrainingArguments(
11        per_device_train_batch_size=16, # 每 GPU 分配的数据批数
12        gradient_accumulation_steps=4, # 累积 4 个批次数据的梯度
13        warmup_steps=2, # 热身过程中学习率递增
14        max_steps=100, # batch 数，会根据数据规模调整，以免数据耗尽
15        learning_rate=2e-4, # 学习率
16        fp16=True, # 计算更快，占内存更小
17        logging_steps=1,
18        output_dir="llama2-7b-hf-openassistant", # 输出文件夹的路径
19        optim="paged_adamw_8bit" # 分页内存管理，避免 OOM 错误
20    ),
21    data_collator=transformers.DataCollatorForLanguageModeling(
22        tokenizer,
23        mlm=False # 非 masked
24    ),
25
26 )
27
28 model.config.use_cache = False # 推理时再开启，详见下文

```

2.4 开始训练

正式开始训练。示例的训练耗时为分钟级别

```
1 trainer.train()
```

2.5 模型推理

如果说训练是模型获得能力的过程，那么推理（inference）就是模型运用能力的过程。

结束训练，开始推理时，我们通过以下配置，记录每个已输出标识的隐状态，避免重复计算

```
1 model.config.use_cache = True # 推理时重新启用
```

最后，创建 pipeline，输入 prompt，即可得到模型输出。示例的推理耗时亦为分钟级别

```
1 from transformers import pipeline
2
3 pipe = pipeline(
4     "text-generation",
5     model=model,
6     tokenizer=tokenizer,
7     torch_dtype=torch.bfloat16, # 16 比特，但范围相当于 float32
8     trust_remote_code=True, # 使用仓库中的代码
9     device_map="auto", # 自动调配可用资源
10 )
11
12 prompt = 'The actual entropy of rolling a dice is 2.584963. Please give the comp
13 sequences = pipe(
14     prompt,
15     max_length=1000,
16     do_sample=True,
17     top_k=10, # pipeline 返回的预测的标签数
18     num_return_sequences=1,
```



```

19         eos_token_id=tokenizer.eos_token_id, # 告知句子结束符
20         pad_token_id=tokenizer.eos_token_id # 告知占位符
21     )
22
23 # 输出 prompt 和 response
24 for seq in sequences:
25     print(seq['generated_text'])

```

以“计算投骰子的熵”为例，在微调之前，模型的输出为

```

... Result: The actual entropy of rolling a dice is 2.584963. Please give the computational steps that come to this answer.

Comment: The entropy of rolling a die is not 2.584963. That is the average entropy of a single outcome.

Comment: Why not? It is the average of the outcomes in a large sample of independent trials.

Comment: The average entropy is only the expected number of bits. This is not entropy.

Answer: The entropy is just  $\log_2 6$ . If you're rolling 56 sided dice and you are rolling 36 sided dice, then you have an entropy of  $0.3916$ .

Comment: The entropy is the average information content of a single outcome, not the average entropy of the entire system.

Comment: @AlecCass I don't get what you're saying. The average entropy is the information content per bit of each outcome. If it was a 60 sided die, the

Answer: The expected entropy of a random number of bits, where each bit is independently and uniformly generated, is the expected logarithm in base 2 of the number of bits.


$$\frac{5}{6} \log_2 \frac{36}{6} = \log_2 \left( \frac{12}{3} \right) = 2.221316 \dots$$


You should not confuse the entropy of the system with the entropy of each outcome. The expected entropy is what you would need to communicate to the rest of the world the outcome of the system.

As a side note: the entropy of the dice system is  $0.3916$  (which is what the expected entropy of each outcome).

Comment: I am not quite sure what your answer is saying.

Comment: @user3947581 I'm saying that the entropy of the system (5-sided die) is  $\log_2(12/3) = 2.2218$ , and the expected entropy of the

```

形式上更像网页评论区；内容上未能依据指令，正面给出计算步骤。

微调后，模型的输出为

```

... Result: The actual entropy of rolling a dice is 2.584963. Please give the computational steps that come to this answer.

Answer: The entropy of a random variable is a measure of the probability that it can take a specific value in the future.

In other words, the higher the entropy, the more likely a random variable can take a specific value in the future (assuming other things are unchanged).

In your question, you have a random variable,  $\text{rolling dice}$ , that can only take on two values,  $X = \{1, 2\}$ . This means the entropy for your variable is  $\log_2 2 = 1$ .

Comment: Thanks a lot for explaining it so clearly.

```

形式上更像问答助手；内容上先给出了熵的概念和意义，再依据定义式给出了熵的计算过程。虽然代入的值应为 6 而非 2，且对数计算结果应为 1 而非 2，但整体思路正确的，人类提问者只需稍加修改，即可得到想要的答案。

总结

我们认为大模型的知识是可压缩的，即其参数矩阵本质上是低秩的，因此相信采用 LoRA 技术微调，是有可能不显著降低模型能力的。此外，Llama 2 的预训练模型能力已经比较出众，openassistant-guanaco 数据集也具有很高的质量和多样性，这些都是微调后模型效果较好的基础。

<div style="page-break-after:always"></div>

参考材料

[FineTuning-and-Inference-Llama2/Llama_2_7b_chat_fine_tuning.ipynb at main · Rupesh-rkgit/FineTuning-and-Inference-Llama2 ([github.com](https://github.com/Rupesh-rkgit/FineTuning-and-Inference-Llama2))](https://github.com/Rupesh-rkgit/FineTuning-and-Inference-Llama2/blob/main/Llama_2_7b_chat_fine_tuning.ipynb)