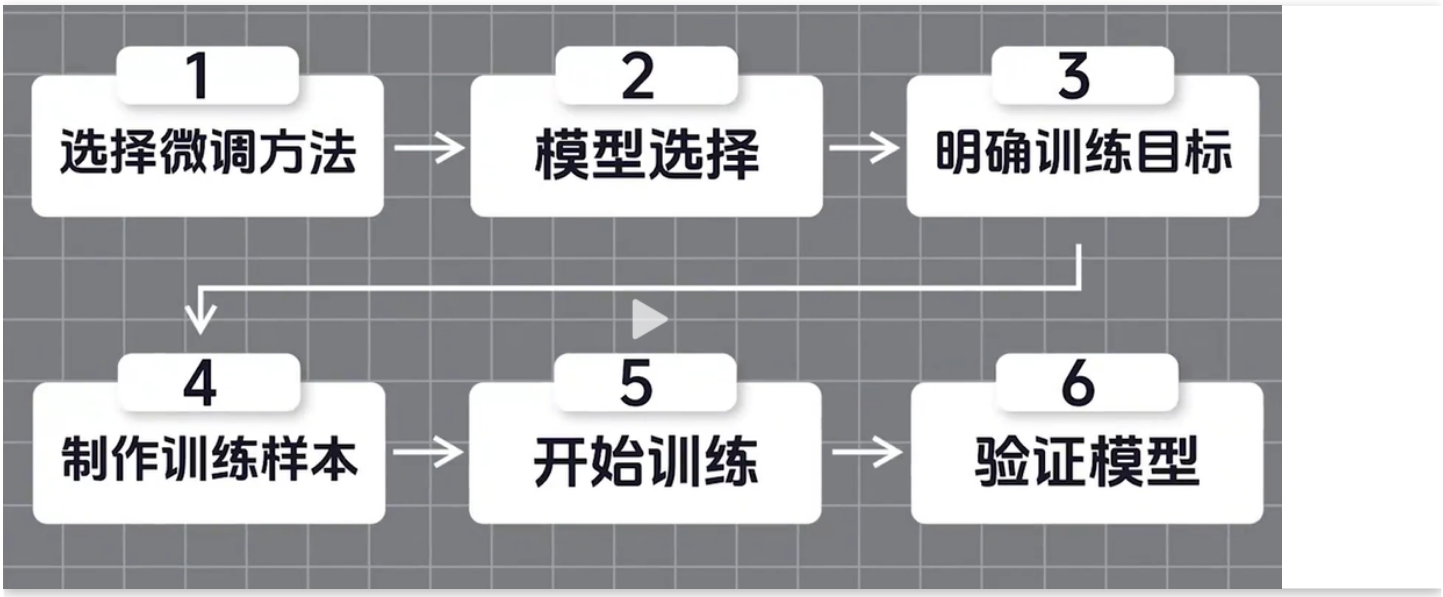
一、基本逻辑



二、训练方法

学习率调度

LORA

QLORA

全模型微调

逐层解冻-双阶段训练

正则化方法

多任务学习

双阶段训练

持续学习

Lora 和 QLoRA 是近年来在大模型微调领域出现的高效方法。它们旨在通过降低微调过程中的计算和内存开销，使得大规模预训练模型能够在资源有限的情况下进行有效微调。这两种方法的核心思想是避免对整个大模型进行全面的微调，而是仅对模型的一部分进行训练，减少了微调时所需的计算资源和内存消耗。

1. LoRA（Low-Rank Adaptation）

LoRA 是一种通过引入低秩矩阵来进行微调的技术，它不直接改变预训练模型的参数，而是通过引入可训练的低秩矩阵来“适应”目标任务。这种方法能够减少训练的计算复杂度，同时保证大模型的效率和表现。

LoRA的工作原理：

LoRA的核心思想是，在原始模型的参数中插入低秩适配器层。在每一层的线性变换（如全连接层）中，引入一个低秩矩阵 A 和 B，这些矩阵的维度相对较小（比原始参数矩阵要小）。这种方式的关键优势是通过低秩矩阵来逼近原始参数矩阵的变化，从而在保持效果的同时大大减少了训练时的计算量。

具体步骤：

添加低秩适配器：对于每一层的权重矩阵，LoRA 会为其增加低秩矩阵 A 和 B，通过 A·B 来进行微调。

冻结原有参数：保持预训练模型的权重不变，只训练新引入的低秩矩阵。

低秩矩阵优化：训练过程中仅优化低秩矩阵 A 和 B，而不触及原始模型参数。

LoRA的优势：

计算开销小：相比全面微调，LoRA只训练少量的参数，因此显著降低了计算和内存需求。

高效的适配：适用于大规模预训练模型，尤其是在计算资源有限的情况下。

无需修改原始权重：LoRA通过适配器进行微调，避免了对原始模型权重的修改，使得微调过程更加稳定。

应用场景：

LoRA适用于需要对非常大的模型进行微调的场景，特别是在 NLP、计算机视觉等领域的大型预训练模型中，能够减少计算资源的消耗，并保持较高的任务性能。

2. QLoRA（Quantized LoRA）

QLoRA 是 LoRA 的一种扩展，它结合了量化技术进一步降低了内存和计算开销。通过对低秩矩阵进行量化，可以显著减少微调过程中对内存的需求，特别适用于具有严格资源限制的硬件设备。

QLoRA的工作原理：

QLoRA 在 LoRA 的基础上，增加了量化步骤。量化的核心思想是将模型的浮点数权重转换为低精度（如 8 位或 4 位）的整数，从而降低模型在训练时占用的内存空间和计算量。具体来说：

量化低秩矩阵：在LoRA中加入的低秩矩阵 A 和 B 会进行量化处理，通常使用较低的位宽（如 8-bit）。

量化优化：在训练时，不仅微调低秩矩阵的参数，还通过量化后的表示来进行优化。这使得内存消耗和计算成本进一步降低。

反量化：在微调完成后，通过反量化将低精度矩阵转换回较高精度的格式，从而恢复模型的精度。

QLoRA的优势：

进一步减少内存需求：通过量化技术，QLoRA显著降低了模型的内存占用，使得即使在资源有限的硬件上也可以微调大规模模型。

适应硬件资源限制：在一些内存较小的设备（如GPU显存较低的环境）中，QLoRA尤其具有优势。

高效性：量化减少了计算成本，从而提高了训练效率。

应用场景：

QLoRA适用于在内存和计算资源受限的环境中进行大规模模型微调，尤其是在高效的硬件平台（如移动设备或嵌入式系统）上。

3. LoRA与QLoRA的比较

LoRA：引入低秩矩阵适配器，保持原始模型不变，仅训练适配器。适用于大模型微调，能够显著降低计算和内存开销。

QLoRA：在LoRA的基础上增加了量化技术，进一步减少内存占用。适用于内存受限或计算资源有限的环境。

对比总结：

计算效率：QLoRA 优于 LoRA，特别是在需要量化以减少内存消耗的场景中。

内存占用：QLoRA 的内存占用进一步减少，尤其适合硬件限制较大的设备。

训练速度：由于减少了内存带宽和计算负载，QLoRA 通常在训练速度上优于 LoRA。

4. LoRA 和 QLoRA 在实际中的应用

这些方法已经在多个领域取得了良好的效果，尤其是在NLP（如GPT、BERT等语言模型）和计算机视觉（如视觉Transformer）等领域。

NLP：在各种预训练语言模型的微调中，LoRA和QLoRA帮助加速了微调过程，减少了GPU内存消耗。例如，使用LoRA微调大型Transformer模型（如GPT、BERT等）时，通过引入低秩适配器显著提高了微调效率。

计算机视觉：在视觉Transformer（ViT）等大模型的微调中，LoRA和QLoRA同样起到了减少计算量、提高训练效率的作用。

总结

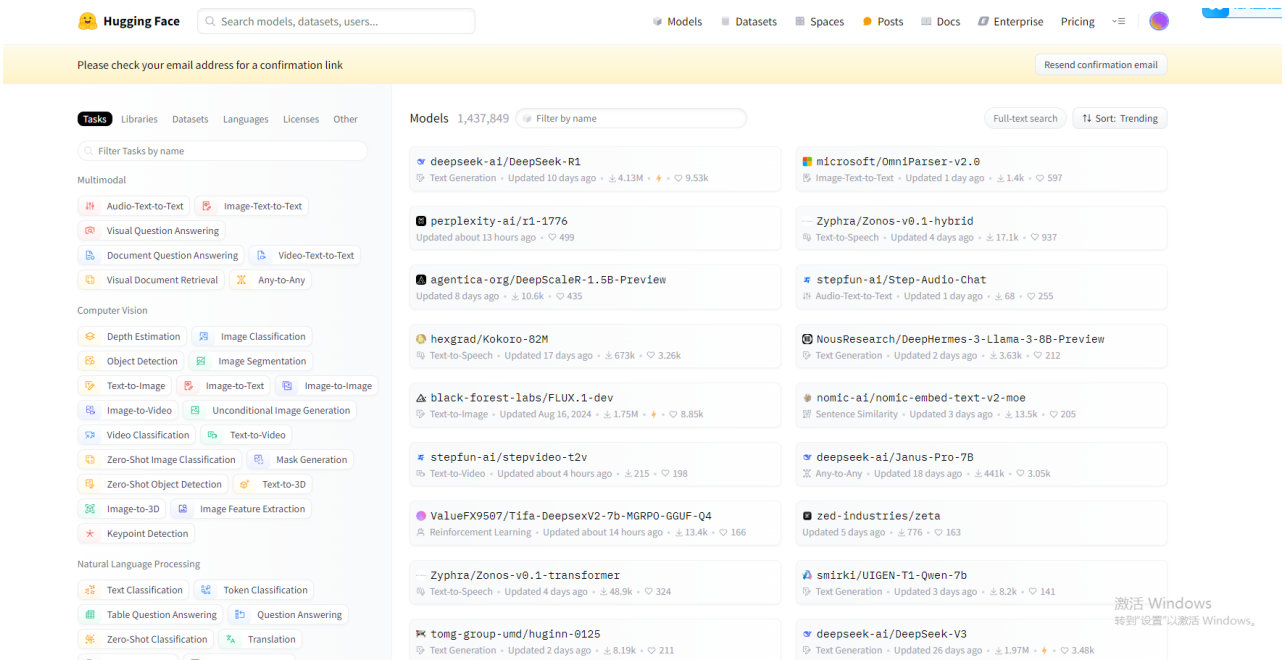
LoRA 和 QLoRA 是针对大模型微调的高效方法，特别适合资源有限的场景。LoRA通过低秩矩阵引入适配器，减少了计算和内存开销，而QLoRA则在此基础上通过量化进一步优化了内存和计算效率。这些方法为大规模预训练模型的微调提供了更高效、节省资源的方案。

三、模型仓库、选用微调模型

AI-LLM 库 AI-github

https://huggingface.co/

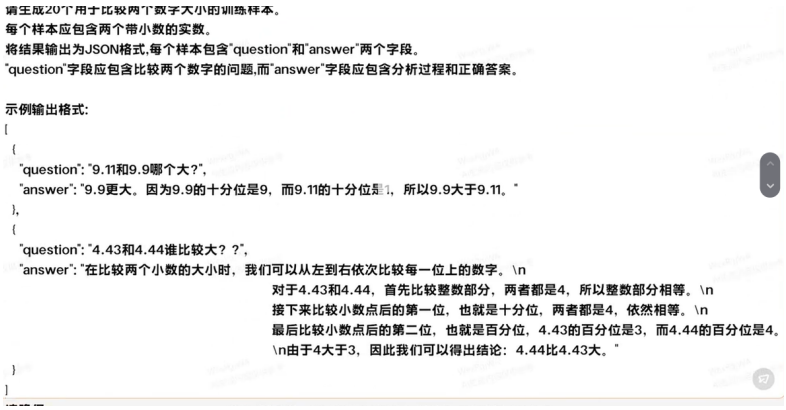
https://huggingface.co/models



四、数据集制备

训练目标-需要解决什么问题、什么数据

训练样本-制作训练样本、注意训练样本格式、保存为json格式



请确保:

1. 数字的范围应该多样化。
2. 有时两个数字可能非常接近,以增加难度。
3. 问题的表述应该有所变化,如"哪个数更大"、"哪个数比较大"、"哪个数更高"等。
4. 答案应该包含分析的过程

请生成符合上述要求的20个训练样本。

五、环境安装

1、安装python环境

2、安装各种依耐库

3、下载预训练模型

4、部署训练代码

训练环境安装-可能用到的环境：

微调大模型通常需要一个适合深度学习任务的Python环境。以下是一些基本的环境设置和依赖库，确保你能够顺利进行大模型微调：

1. Python 环境

Python 版本：建议使用 Python 3.7 及以上版本。大多数深度学习库都支持 Python 3.7、3.8、3.9 等较新版本。

2. 常用的 Python 库

以下是一些在微调大模型时常用的库：

1. 深度学习框架

PyTorch：当前最受欢迎的深度学习框架之一，广泛应用于大模型微调。

bashCopy Code

```
pip install torch
```

TensorFlow：另一个流行的深度学习框架，也支持大模型的训练和微调。

bashCopy Code

```
pip install tensorflow
```

2. Transformer 和预训练模型相关库

Transformers（由Hugging Face提供）：这个库非常适合加载和微调各种预训练语言模型（如BERT、GPT、T5等）。

bashCopy Code

```
pip install transformers
```

Accelerate：用于优化训练过程，简化分布式训练，适用于多GPU或TPU。

bashCopy Code

```
pip install accelerate
```

3. 优化器和训练工具

DeepSpeed：提供了高效的训练工具，支持模型并行化，可以加速大规模模型的训练。

bashCopy Code

```
pip install deepspeed
```

Fairscale：提供分布式训练和优化功能，能够帮助加速大模型的训练过程。

bashCopy Code

```
pip install fairscale
```

4. 数据加载和预处理

Dataloader（PyTorch中的数据加载工具）：用于高效加载数据。

datasets（Hugging Face）：一个很方便的工具，用于加载和处理大量标准化的NLP数据集。

bashCopy Code

```
pip install datasets
```

5. 混合精度训练

Apex（NVIDIA）：用于支持混合精度训练，减少计算负载。

bashCopy Code

```
pip install apex
```

6. LoRA 和 QLoRA

如果使用 LoRA 或 QLoRA，你可能需要额外的库来实现低秩适配器或量化：

peft：一个简洁的库用于处理 LoRA 和其它轻量化微调方法。

bashCopy Code

```
pip install peft
```

7. 其他常用库

NumPy：数值计算库，几乎所有深度学习框架都需要。

bashCopy Code

```
pip install numpy
```

scikit-learn：用于数据处理、评估和优化的工具。

bashCopy Code

```
pip install scikit-learn
```

3. 硬件支持和环境

CUDA

：如果你使用NVIDIA GPU进行训练，确保已安装与PyTorch或TensorFlow兼容的CUDA版本（通常支持CUDA 11.x或更高版本）。

CUDA和cuDNN可以通过NVIDIA的官方网站下载并安装。

如果你没有GPU，PyTorch和TensorFlow也能在CPU上运行，但速度会慢很多。

GPU驱动

：确保你安装了适合你的GPU型号的驱动程序。NVIDIA的CUDA工具包包含了与GPU兼容的驱动。

NVIDIA APEX

：对于需要在混合精度训练时加速的用户，可以使用NVIDIA的Apex库，来提升训练速度和效率。

4. 虚拟环境管理

为了避免库的版本冲突，建议使用虚拟环境来管理项目的依赖。你可以使用 conda 或 venv 来创建和管理虚拟环境。

使用 conda 创建虚拟环境：

bashCopy Code

```
conda create --name myenv python=3.8
conda activate myenv
```

使用 venv 创建虚拟环境：

bashCopy Code

```
python -m venv myenv
source myenv/bin/activate  # Linux/macOS
myenv\Scripts\activate     # Windows

5. 一些附加的配置
配置文件：如果你进行多GPU训练，可能需要设置分布式训练的配置，PyTorch支持torch.distributed，而TensorFlow则使用tf.distribute.Strategy来进行分布式训练。
训练日志：建议使用如 TensorBoard 或 WandB 这样的工具来监控训练过程。
bashCopy Code
pip install tensorboard
pip install wandb

```

六、训练设备

1、GPU服务器

迈克生物

2、租赁GPU云服务器

https://gpuez.com/zscloud/gpus/intro

https://www.funhpc.com/?bd_vid=11794289404561290342#/

七、其他

数据上传-训练

微调模型、验证

八、附录：

代码参考：懒人包

~~**云盘：**~~通过网盘分享的文件：just_train.zip

链接: https://pan.baidu.com/s/1To_5H4-E69Mwv2sD5J5t2w 提取码: 9b3x