大语言模型的微调

LLM的开发流程

大规模语言模型的开发流程大致可分为四个关键阶段：预训练、有监督微调、奖励建模和强化学习。

每个阶段都针对特定的目标，采用不同的数据集和算法策略。

1. 预训练（Pretraining）

预训练构成了LLMs开发的基础，涉及处理大规模数据集，如互联网文本、百科全书、文学作品、开源代码库、学术论文等，以构建一个包含数千亿到数万亿词汇的语料库。此阶段的目标是让模型掌握广泛的语言规则、语境理解和文本生成技能。

- **模型规模与资源需求**：预训练通常采用具有大量参数的模型，例如，GPT-3拥有1750亿参数，其预训练的计算量可达数千PFlops，需要使用多达1000个GPU，并耗时近一个月完成。其他类似规模的模型包括LLaMA 等。鉴于预训练对计算资源的巨大需求和超参数的敏感性，优化分布式计算效率并确保模型的稳定收敛是研究的关键点。

2. 有监督微调（Supervised Finetuning）

有监督微调是在预训练基础上进行的，目的是让模型更加精准地理解和执行特定任务。

这个阶段通过使用标注好的数据集，如用户提问和期望的答案，来进一步训练模型。

- **数据样例**：
  - **Prompt**：纽约大学有几个校区？
  - **Desired Output**：纽约大学在美国和全球共有多个校区，包括但不限于纽约校区、阿布扎比校区和上海校区。

通过这些标注数据的训练，可以得到一个有监督微调模型（SFT Model），它具备任务理解和上下文理解的能力，能够执行如开放领域问答、阅读理解、翻译和代码生成等任务。SFT模型的训练相对较快，通常使用数十个GPU，只需数天时间。

3. 奖励建模（Reward Modeling）

奖励建模的目的是创建一个评价不同输出质量的模型，即奖励模型（RM Model），它通过比较不同输出的优劣来为它们排序。

- **数据规模与挑战**：为了训练RM模型，需要大量的标注对比数据。这些数据的标注工作往往耗时且复杂。保持标注的一致性和提升RM模型的泛化能力是此阶段的主要挑战。

4. 强化学习（Reinforcement Learning from Human Feedback, RLHF）

在强化学习阶段，利用上一阶段训练的RM模型来评估SFT模型的输出，并据此调整SFT模型的参数，以生成更高质量的文本。

- **计算需求与优势**：与预训练相比，强化学习的计算量较小，通常使用数十个GPU，训练周期为数天。研究表明，强化学习在相同参数量下比有监督微调能获得更好的效果，但其稳定性较差，且超参数众多，加之RM模型的准确性问题，使得强化学习在大规模语言模型中的应用颇具挑战。

经过强化学习训练的RL模型是最终提供给用户使用的系统，如ChatGPT。强化学习虽然能显著提升模型效果，但也可能导致模型输出多样性的减少。因此，如何在提升输出质量和保持多样性之间取得平衡，是强化学习阶段需要考虑的问题。

LLM模型构建流程3阶段划分

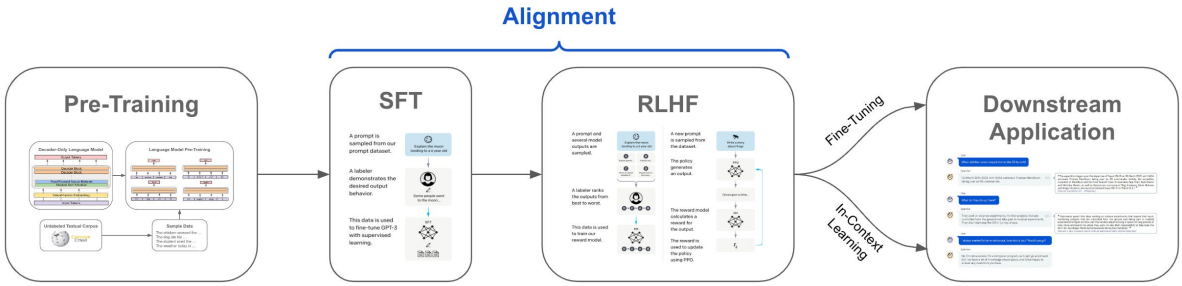
1. 预训练阶段（Pretraining）
在此阶段，模型通过大量的非标注文本数据进行训练，掌握基本的语言理解与生成能力，构建基础语言模型。

2. 微调阶段（Finetuning）
通过有监督数据或少样本学习，让模型更好地适应特定任务或领域。

3. 对齐阶段（Alignment）
模型经过奖励建模和强化学习阶段对齐人类的目标和价值观，提高模型输出的合理性与质量。

LLM模型构建流程2阶段划分

1. **预训练阶段（Pretraining）**
2. **对齐阶段（Alignment）：SFT + RLHF**



从语言模型到聊天智能: 基座模型到对话模型的转变

以GPT和ChatGPT为例, 原始的GPT模型被设计为一个通用的语言模型，它的主要目标是在给定的上文情况下，预测下一个单词或继续生成文本。

而ChatGPT是在GPT基础上进行微调，特别优化以应对对话场景，其目标是生成连贯、自然且上下文相关的对话回复。

基座模型到对话模型的转变的步骤如下：

第一步: 预训练LLM基础模型

首先, 我们需要在大规模语料库上预训练一个LLM基础模型。这个模型通常是一个大型的Transformer模型, 如GPT-3、T5、LLaMA等。

预训练的目标是让模型学习到语言的基本结构、语法、语义等一般性知识。

预训练通常采用自监督学习的方式，即让模型在大量无标签的文本数据上学习预测下一个词或恢复被遮盖的词。通过这种方式，模型可以学习到词汇、短语、句法、语义等多层次的语言表征。

预训练得到的LLM基础模型具有强大的语言理解和生成能力，但它们通常是通用的，没有针对特定任务或领域进行优化。

第二步：使用SFT进行任务适应

为了让LLM基础模型适应特定的任务，如对话、问答、摘要等，我们需要在特定任务的标注数据集上对其进行有监督微调(SFT)。

在对话任务中，我们通常准备一个由人类对话组成的数据集，其中每个样本包括一个对话历史(上下文)和一个人类回复(目标)。

然后，我们将这些样本输入到LLM基础模型中，让其学习根据给定的对话历史生成恰当的回复。

SFT的过程类似于监督学习，即最小化模型生成的回复与真实人类回复之间的差异(通常使用交叉熵损失)。

通过在大量对话样本上训练，模型可以学习到对话的一般模式、礼貌用语、常见话题等。

SFT得到的模型已经可以进行基本的对话交互，但其生成的回复可能仍然存在一些问题，如不够流畅、不够贴切、缺乏逻辑等。

第三步：使用RLHF提高对话质量

为了进一步提高对话模型的质量，使其生成更加自然、贴切、符合人类偏好的回复，我们可以使用RLHF对SFT模型进行优化。

RLHF的核心思想是让模型根据人类对其生成回复的反馈(奖励)来学习和改进其行为策略。

具体来说，RLHF的过程如下：

1. 收集一批SFT模型生成的回复样本，并让人类对每个回复打分(如按相关性、流畅度、逻辑性等维度)。
2. 训练一个奖励模型，用于根据对话历史和回复内容预测人类的评分。这个奖励模型本质上是一个回归模型，可以用人类打分的样本进行监督学习。
3. 使用奖励模型对SFT模型的生成过程进行引导。具体来说，在生成每个词时，我们使用奖励模型评估不同词选择对应的未来奖励，并鼓励模型选择能获得更高奖励的词(通常使用策略梯度算法)。
4. 重复步骤1-3,不断收集新的人类反馈数据，优化奖励模型和对话模型，直到对话质量满足要求。

通过RLHF，对话模型可以学习到更加符合人类偏好的对话策略，生成更加自然、贴切、有逻辑的回复。同时，RLHF也可以帮助模型学习避免生成不恰当、有害或偏见的内容。

第四步：模型部署和持续优化

经过SFT和RLHF优化的LLM聊天模型已经可以投入实际应用，为用户提供智能对话服务。但我们的工作并没有就此结束，还需要持续监控模型的性能，收集用户反馈，发现和解决存在的问题。

同时，随着时间的推移，用户的需求和偏好可能会发生变化，新的话题和任务也会不断出现。因此，我们需要持续收集新的对话数据，定期对模型进行重新训练和优化，以适应不断变化的应用环境。

此外，我们还需要重视聊天模型的伦理性、安全性和可解释性。这需要在数据收集、模型训练、应用部署的各个环节进行全面考虑，并与用户、社会各界保持积极沟通和互动。

针对特定领域的SFT微调

大语言模型(LLM)如GPT-3、LLaMA等，在通用自然语言处理任务上已经展现出了非凡的能力。

然而, 当我们希望将LLM应用到特定领域时,如医疗、法律、金融等, 仅仅依靠通用的语言能力往往是的。

我们还需要对LLM进行针对性的微调, 使其能够掌握领域专业知识,遵循行业规范,适应特定的任务需求。

这就是特定领域的SFT(Supervised Fine-Tuning, 有监督微调)。

为什么需要特定领域SFT微调

特定领域SFT微调之所以必要, 主要有以下几个原因:

1. **领域知识的获取**: 通用LLM的训练数据主要来自互联网, 对特定领域的知识覆盖有限。通过在领域数据上进行微调, 可以让LLM快速习得专业术语、概念、理论等领域知识, 从而生成更加专业和准确的内容。
2. **行业规范的遵循**: 许多特定领域都有其独特的行业规范和伦理准则, 如医疗中的隐私保护,法律中的程序正义等。仅凭通用LLM很难完全遵循这些规范。通过SFT微调, 我们可以在训练数据中明确地体现这些规范, 引导LLM形成合规的行为模式。
3. **任务形式的适配**: 不同领域的任务形式千差万别, 如医疗诊断、合同审核、股票分析等。通用LLM可能缺乏对这些具体任务形式的适配能力。SFT微调让我们可以用领域任务的输入输出样本来训练LLM, 使其习得任务的流程、规则和约束。
4. **性能的进一步提升**: 即使对于通用LLM已经能够处理的领域任务, SFT微调也可以在特定数据集上进一步提升其性能。微调后的模型往往能够生成更加流畅、连贯、符合领域特点的内容, 给用户更好的体验。

特定领域SFT微调的一般流程

进行特定领域SFT微调通常包括以下几个关键步骤:

1. **领域数据准备**:
    - 收集高质量的领域文本数据, 包括各种形式的结构化、半结构化和非结构化数据。
    - 对原始数据进行清洗、脱敏,去除噪音、错误和敏感信息。
    - 根据任务需求, 将数据标注为特定的输入输出格式, 如问答对、摘要对、翻译对等。
2. **微调样本构建**:
    - 根据LLM的特点和领域任务的要求, 设计输入输出的样本格式和提示模板。
    - 利用领域知识图谱、行业规范文档等, 丰富输入样本的背景信息和约束条件。
    - 将格式统一的输入输出样本整理为可直接用于微调的数据集。
3. **微调策略设计**:
    - 选择一个与领域任务相近的通用LLM作为初始化模型, 以提高微调的效率和效果。
    - 设计微调的目标函数, 除了基本的语言建模损失, 还可加入特定任务的评估指标。
    - 确定微调的超参数, 如学习率、BatchSize、训练轮数等, 以平衡性能和成本。
4. **模型微调训练**:
    - 利用构建好的微调样本集, 在初始化的LLM上进行梯度下降训练。
    - 动态调整超参数, 监控模型在验证集上的收敛情况。
    - 对训练过程中的checkpoints进行测试, 选择性能最优的模型进行后续优化。
5. **微调模型评估**:
    - 在独立的测试集上全面评估微调后的模型, 既要考察通用的语言质量, 也要重点评测领域任务的关键指标。
    - 邀请领域专家对模型输出进行人工评审, 提供改进反馈。
    - 针对评估结果, 有针对性地扩充数据、调整模型, 进行多轮迭代优化。
6. **模型部署应用**:
    - 将微调优化后的模型部署到生产环境, 为特定领域任务提供智能辅助。
    - 构建人机交互界面, 让领域用户可以方便地使用LLM的能力。

- 持续收集用户反馈数据, 定期审核模型性能, 并根据领域知识的更新迭代模型。

特定领域SFT微调的典型应用

通过SFT微调, LLM可以被赋予领域智能,在许多行业领域发挥重要作用, 例如:

1. **医疗健康领域**:
   - 医疗对话助手: 提供智能问诊、医疗咨询、健康指导等服务。
   - 医学报告生成: 自动撰写病历、影像报告、出院总结等医学文档。
   - 药物研发辅助: 协助分析药物分子结构、预测药物性质、优化药物设计等。
2. **法律司法领域**:
   - 法律咨询助手: 为非专业人士提供法律知识普及、案例解析、诉讼指引等服务。
   - 合同审核助手: 自动审查合同条款,识别风险条款,提供修改建议。
   - 判决文书分析: 对大量判决文书进行要素提取、类案检索、统计分析等处理。
3. **金融经济领域**:
   - 金融资讯摘要: 自动提取财经新闻要点,生成新闻摘要和市场评论。
   - 股票趋势预测: 结合财报数据、市场行情,对个股未来走势进行预测分析。
   - 经济报告撰写: 根据经济数据和分析模型,自动生成宏观经济报告、行业分析报告等。
4. **教育培训领域**:
   - 智能教学助手: 为学生提供针对性的学习指导、答疑解惑、知识点总结等服务。
   - 题库生成系统: 根据知识点和难度要求,自动生成试题和答案解析。
   - 论文评改助手: 对学生提交的论文进行自动评阅,检查语法错误、逻辑漏洞,并给出修改意见。

从哪个模型开始做SFT微调？基座模型和对话模型的选择

究竟是在预训练的基础模型(foundation model)上还是在聊天模型上进行特定领域的SFT微调, 取决于多个因素,包括任务需求、数据可用性、计算资源等。让我们分别来看看这两种选择的优缺点。

在预训练的基础模型上进行领域SFT微调

优点:

1. 通用语言能力: 预训练模型已经在大规模语料库上学习了丰富的语言知识, 具有强大的语言理解和生成能力, 可以为领域微调提供一个很好的起点。
2. 更多的可塑性: 预训练模型相对于聊天模型而言更加"原始", 没有被优化到特定的对话任务, 因此在领域微调时可能有更大的可塑性和适应性。
3. 更低的计算成本: 如果使用同样的模型架构, 从预训练模型开始微调通常比从聊天模型开始微调需要更少的计算资源和时间, 因为聊天模型已经在预训练模型的基础上进行了额外的训练。

缺点:

1. 领域适应性差: 预训练模型虽然具有通用的语言能力, 但对特定领域的知识和对话模式的理解可能比较欠缺, 需要更多的领域数据和训练时间来适应。
2. 对话连贯性不足: 预训练模型擅长处理独立的文本片段, 但可能缺乏对话所需的上下文理解和一致性维持能力, 需要在微调时着重优化。

在聊天模型上进行领域SFT微调

优点:

1. 更好的领域适应性: 聊天模型在预训练的基础上, 已经在通用对话数据上进行了微调, 学习了对话的一般模式和策略, 可以更快地适应特定领域的对话任务。
2. 更强的对话连贯性: 聊天模型已经具备一定的上下文理解和一致性维持能力, 在领域微调时可以更好地保持对话的连贯性和逻辑性。
3. 更优的用户体验: 基于聊天模型微调得到的领域对话模型, 可以为用户提供更加自然、流畅、贴近人类交互方式的对话体验。

缺点:

1. 对话偏见 :聊天模型在通用对话数据上训练时,可能已经学习了一些不适用于特定领域的对话偏见, 如果不加以校正, 可能影响领域对话的质量。
2. 知识覆盖不足: 聊天模型的知识主要来自于对话数据 ,对特定领域的深度知识覆盖可能不够, 需要在微调时引入更多的领域知识。
3. 计算成本较高: 如果聊天模型的规模很大, 或者使用了不同于预训练模型的架构, 则在聊天模型基础上进行领域微调可能需要更多的计算资源和时间。

总结建议

综合来看, 如果你的目标是开发一个高度专业化的领域对话系统, 并且有足够的领域数据和计算资源, 那么在聊天模型上进行领域SFT微调可能是更好的选择。这样可以在已有的对话能力基础上, 快速适应领域特点, 提供更优质的用户体验。

但如果你的领域数据比较有限, 或者希望开发一个更加通用的领域对话系统, 那么从预训练的基础模型开始微调可能更加经济高效。你可以利用预训练模型的通用语言能力, 在更少的数据和计算资源下, 获得一个基本合格的领域对话模型。

当然, 这两种选择并不是绝对的。你也可以将它们结合起来, 先在预训练模型上进行领域微调, 然后再在此基础上进行对话优化。或者, 你可以使用一些最新的技术, 如提示工程(prompt engineering),通过设计良好的提示模板和少样本学习, 在预训练模型上实现快速的领域适应。

总之, 选择哪种模型作为领域SFT微调的起点, 需要综合考虑任务需求、数据条件、资源约束等因素, 权衡不同方案的优缺点, 找到最适合自己的方案。同时, 我们也要持续关注语言模型领域的最新进展, 学习和尝试新的技术和范式, 不断提升领域对话系统的性能和效率。

LLM微调方法

训练微调LLM是一项复杂而昂贵的任务, 需要巨大的计算资源和数据支持。

为了在不同的应用场景中高效地利用和优化LLM, 研究者们提出了多种训练方法, 包括Full-tuning、Freeze-tuning、LoRA(Low-Rank Adaptation)和QLoRA(Quantized Low-Rank Adaptation)。

Full-tuning (全量微调)

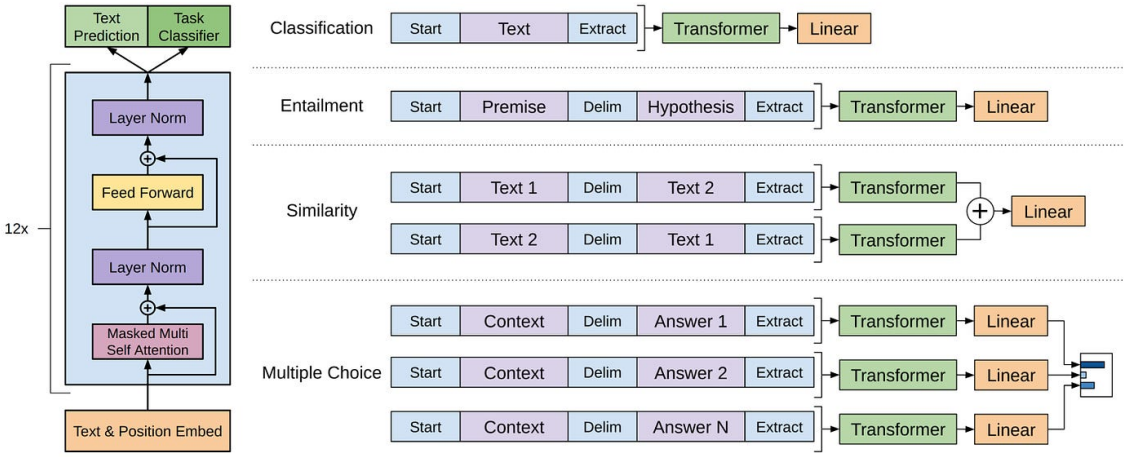


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

Full-tuning是最传统、最直接的LLM微调方法。

顾名思义, 它对预训练模型的所有参数进行全面调整, 以适应特定的下游任务。

具体而言,Full-tuning的流程如下:

1. 加载预训练的LLM模型及其所有参数。
2. 在下游任务的训练数据上,使用任务特定的损失函数(如交叉熵、平方损失等)对模型进行端到端的微调。
3. 所有模型参数都得到更新,以最小化任务损失。
4. 微调后的模型可以直接应用于目标任务的推理。

Full-tuning的优势在于其灵活性和表现力。通过调整所有参数, 模型可以充分适应目标任务的特点,学习任务特定的知识和模式。这种全面的微调通常能够取得较好的性能表现。

然而, Full-tuning也存在一些局限性。

首先, 它需要大量的计算资源和时间成本。对于大型LLM,全参数微调需要训练数亿甚至上千亿个参数,对计算设备提出了很高的要求。

其次, Full-tuning可能面临过拟合的风险。当下游任务的训练数据有限时, 全参数微调可能过度适应训练集的特点, 导致在测试集上泛化能力下降。

Freeze-tuning (冻结部分参数微调)

Freeze-tuning是一种轻量化的LLM微调方法。

与Full-tuning不同, Freeze-tuning只调整模型的一部分参数, 而将其余参数"冻结"为预训练的初始值。

通常, Freeze-tuning只微调LLM的最后一层或几层, 而保持底层的表示学习层不变。

其流程如下:

1. 加载预训练的LLM模型及其参数。
2. 选择要微调的顶层(如最后一个Transformer块), 并解冻其参数。
3. 冻结其余底层的参数, 使其保持预训练的初始值不变。
4. 在下游任务的训练数据上, 只对解冻的顶层参数进行微调, 以最小化任务损失。

Freeze-tuning的主要优势在于其计算效率和泛化能力。

由于只微调部分参数, Freeze-tuning大大减少了训练时间和资源消耗。

同时, 冻结底层参数有助于保留预训练模型学到的通用语言知识, 降低过拟合风险, 提高模型在下游任务上的泛化表现。

但Freeze-tuning的表现力相对有限。

由于大部分参数被冻结, 模型的适应能力受到限制, 可能难以充分捕捉目标任务的特定模式。

在数据充足、任务复杂度高的场景下, Freeze-tuning可能难以达到Full-tuning的性能水平。

LoRA 低秩适应的参数高效微调

LoRA(Low-Rank Adaptation)是一种更加参数高效的LLM微调方法。

与Freeze-tuning固定底层参数不同, LoRA在每个Transformer层中引入了一组低秩分解矩阵, 并只训练这些新增的低秩矩阵, 而保持原有的预训练权重不变。

LoRA的流程如下:

1. 加载预训练的LLM模型及其参数。
2. 在每个Transformer层的自注意力机制和前馈神经网络模块中, 并行地插入一组低秩分解矩阵(秩r<<d,d为隐藏层维度)。
3. 冻结预训练模型的所有原有权重, 只微调新增的低秩矩阵。
4. 在前向传播时,将低秩矩阵与原有权重相乘, 得到适应后的隐藏层表示。

LoRA的核心思想是, 通过少量的额外参数(低秩矩阵)来捕捉下游任务的特定模式, 而不改变预训练权重。

这种方法大大减少了可训练参数的数量(比Full-tuning少2-3个数量级), 显著提高了训练效率。同时, 冻结预训练权重, LoRA在很大程度上保留了原有模型的泛化能力, 降低了过拟合风险。

在实践中, LoRA已经在多个NLP任务上取得了与Fine-tuning相当甚至更优的性能,如文本分类、问答、命名实体识别等。这表明, 低秩适应可以在参数高效性和性能表现之间取得很好的平衡。

但LoRA也有其局限性, 如在一些特定任务上可能达不到Full-tuning的性能水平, 以及低秩假设可能并不总是成立。

QLoRA: 低秩适应的量化优化

QLoRA(Quantized Low-Rank Adaptation)是LoRA的一种量化优化变体。

QLoRA 的核心思想是在应用 LoRA 进行模型调优之前, 先对预训练的基础模型进行量化。

量化是一种将模型参数从高精度浮点数转换为低精度整数的技术, 可以在保持模型性能的同时, 大幅降低模型的存储和计算开销。

具体来说, QLoRA 的训练过程可以分为以下几个步骤:

1. **对基础模型进行量化**：使用量化技术, 如整数量化或二值化, 将预训练模型的权重从浮点数转换为低精度整数。这一步可以显著减小模型的体积和内存占用。
2. **在量化模型上应用 LoRA**：在量化后的基础模型上, 通过添加低秩矩阵对模型进行微调。这一步与标准的 LoRA 类似, 通过训练额外的低秩矩阵来适应特定任务, 而无需调整原始模型的权重。
3. **训练和部署**：使用任务特定的数据对 LoRA 矩阵进行训练, 得到适应后的模型。在推理阶段, 量化后的基础模型和训练好的 LoRA 矩阵可以高效地结合, 以生成所需的输出。

QLoRA 结合了量化和参数高效微调的优点,具有以下特点:

1. **显著减小模型体积**：通过量化技术,QLoRA 可以将基础模型的体积减小到原来的几分之一, 甚至更小。这使得 QLoRA 模型更容易存储和部署在资源受限的设备上。
2. **加速推理**：量化后的模型可以使用整数运算, 而整数运算通常比浮点运算更快。因此, QLoRA 模型在推理阶段可以获得显著的加速。
3. **参数高效**：与标准的微调方法相比,QLoRA 只需训练额外的低秩矩阵, 而不需要调整整个模型的参数。这使得 QLoRA 的参数开销非常小, 非常适合在参数量受限的情况下进行模型调优。

小结

Full-tuning、Freeze-tuning、LoRA和QLoRA代表了LLM训练的不同思路和方法。

它们在参数效率、计算成本、性能表现等方面各有优劣,适用于不同的任务场景和资源限制。

- Full-tuning是最传统、最灵活的微调方法, 通过调整所有参数来充分适应下游任务, 但计算成本高, 过拟合风险大。
- Freeze-tuning通过冻结底层参数, 在减少计算开销的同时保留了预训练知识, 提高了泛化能力, 但表现力有限。
- LoRA通过引入少量低秩分解矩阵, 在保留预训练权重的同时高效地适应下游任务, 取得了参数效率和性能的良好平衡。
- QLoRA在LoRA的基础上进行量化优化, 进一步压缩模型尺寸, 加速计算过程,特别适用于资源受限场景。

LoRA 微调

大规模语言模型（Large Language Models，LLMs）的训练与微调是人工智能领域的核心技术。

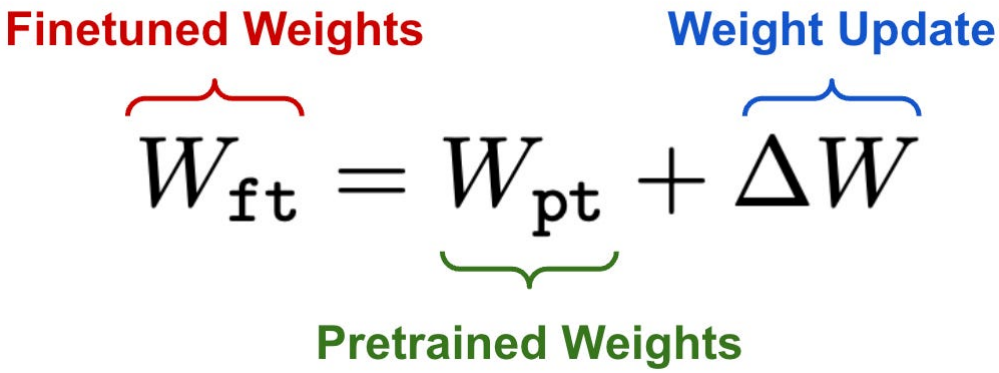
传统的微调方法常因计算资源和存储需求过高而受到限制。

为了应对这些挑战，LoRA（Low-Rank Adaptation）作为一种轻量级的微调技术被引入，它能够以低成本、高效地适应模型。

LoRA 简介

LoRA 是一种将模型权重矩阵分解为低秩矩阵并进行训练的微调技术。

其主要目标是通过引入低秩分解矩阵，使得大型模型在微调时只需调整少量参数，从而显著降低计算与存储成本。



LoRA 的数学原理

权重分解

假设我们有一个原始的权重矩阵 $W_0 \in \mathbb{R}^{d \times k}$，通常在预训练模型中它被用于线性变换。

LoRA 的核心思想是将该矩阵表示为两个低秩矩阵的乘积：

$\Delta W = AB$

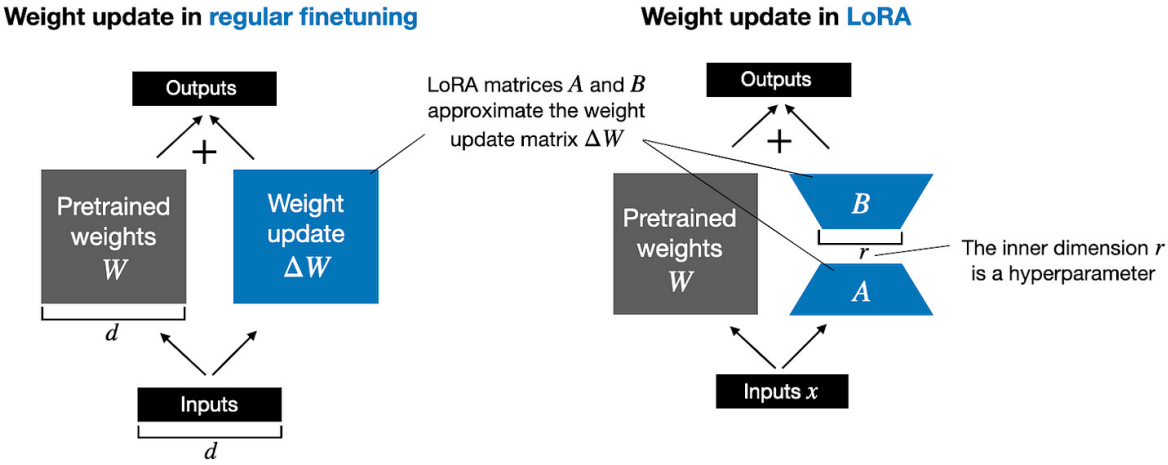
其中：

- $A \in \mathbb{R}^{d \times r}$
- $B \in \mathbb{R}^{r \times k}$

参数 r 为分解矩阵的秩，且 $r \ll \min(d, k)$，因此可以显著减少需要训练的参数量。

经过 LoRA 改动的线性层计算如下：

$W = W_0 + \Delta W = W_0 + AB$



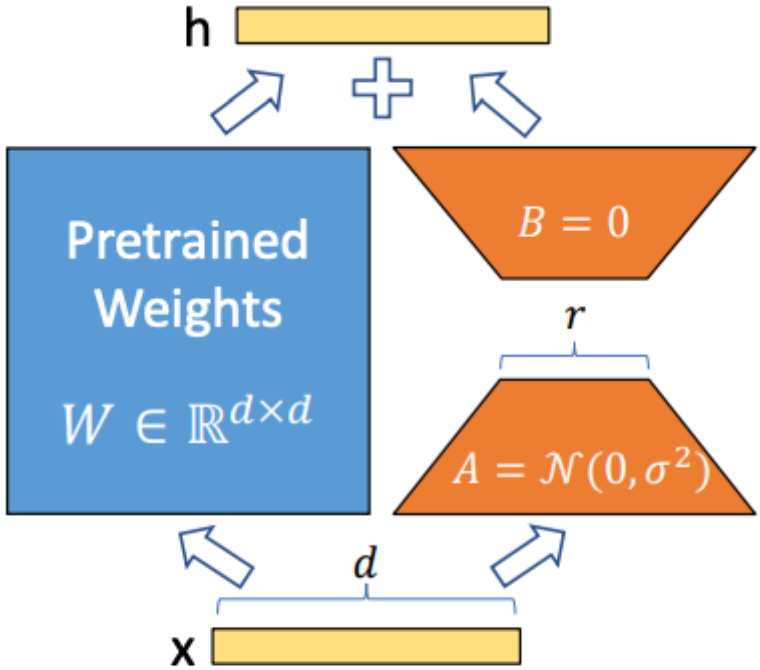


Figure 1: Our reparametrization. We only train A and B.

插入位置

在实际应用中，LoRA 通常插入在 Transformer 架构的注意力模块和前馈网络中。

以注意力模块中的投影矩阵为例，假设我们需要对查询矩阵 W_q 进行 LoRA 微调。

- 原始投影矩阵：
  $Q = XW_q$
- 应用 LoRA 微调后的投影矩阵：
  $Q = X(W_q + AB) = XW_q + XAB$

其中：

- X 是输入的特征矩阵。
- W_q 是预训练模型中的查询矩阵。

在微调过程中，我们只训练 A 和 B，保持原始的 W_q 不变。

Rank Decomposition Matrix

$$W_{\text{ft}} = W_{\text{pt}} + \underbrace{\Delta W}_{\text{Approximation}} = W_{\text{pt}} + \overbrace{AB}^{\text{Rank Decomposition Matrix}}$$

$$\text{where} \quad W_{\text{ft}}, W_{\text{pt}}, \Delta W, AB \in \mathbb{R}^{d \times d}$$

$$\text{and} \quad \underbrace{A \in \mathbb{R}^{d \times r}, B \in \mathbb{R}^{r \times d}}_{\text{Low Rank}}$$

参数量对比

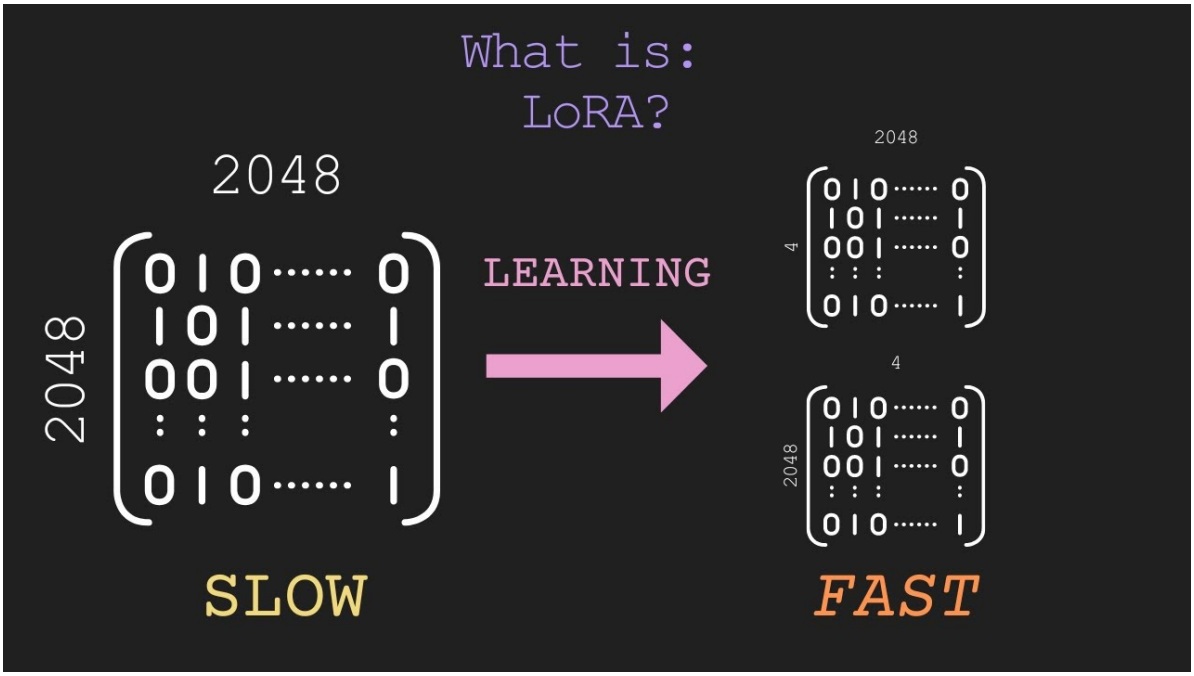
假设原始矩阵 W_0 的大小为 $d \times k$，而 LoRA 引入的矩阵 A 和 B 的大小分别为 $d \times r$ 和 $r \times k$。此时：

- 原始模型的参数量：
  $d \times k$
- LoRA 增加的参数量：
  $d \times r + r \times k$

由于 r 通常远小于 d 和 k，因此 LoRA 增加的参数量很小。例如，若 $d = 4096$，$k = 4096$，而 $r = 8$，则：

- 原始参数量：
  $4096 \times 4096 = 16,777,216$
- LoRA 增加的参数量：
  $4096 \times 8 + 8 \times 4096 = 65,536$

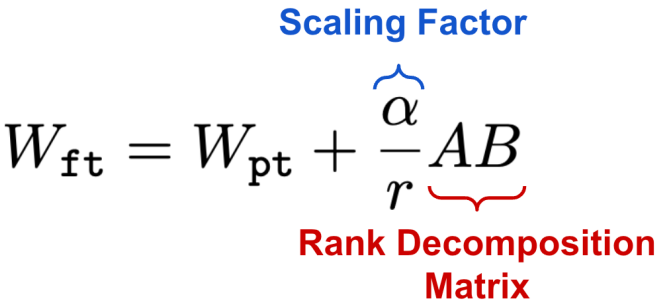
仅为原始参数量的 0.39%。



在低秩适应（LoRA）技术中，缩放因子α的使用是一个关键的调整步骤，它允许我们精细控制预训重与新任务适应性之间的平衡。

缩放因子的作用

在LoRA中，通过在模型的权重矩阵上应用低秩更新，可以实现对模型的微调。这个更新是通过乘以一个低秩矩阵R来完成的，该矩阵是通过奇异值分解得到的。然而，为了确保微调后的模型权重不会过度影响模型的原始预训练权重，我们引入了一个缩放因子α：



默认值与调整

- **默认值**：α的默认值通常设置为1，这意味着在模型的前向传播计算中，预训练权重和低秩权重更新将被平等地考虑。
- **重要性调整**：通过调整α的值，我们可以增加或减少低秩权重更新对最终权重的贡献，从而更好地适应新任务。

实证分析与应用

- **秩的影响**：实证分析表明，对于较高秩的LoRA（即更大的r值），可能需要更大的α值来实现更好的性能。这是因为较高秩的更新可能在权重空间中引入更大的变化，需要通过缩放因子来进行适当的调整。
- **任务特定调整**：不同的任务可能需要不同的α值来达到最优性能。例如，在一些任务中，保持预训练权重的较大影响力可能是有益的，而在其他任务中，可能需要更多地依赖于新任务的适应性。

缩放因子α在LoRA中提供了一种灵活的机制，使我们能够根据不同任务的需求调整预训练模型的权重。

通过精心选择α的值，我们可以在保持预训练模型知识的同时，有效地引入新任务的特定适应性，从而提高模型在特定任务上的表现。

这种方法不仅提高了微调的效率，而且增加了模型的适用性和灵活性。

LORA为什么有效?

LORA通过低秩分解构造了从微调得到的权重更新, 其中只包含非常少的可训练参数。

考虑到这一点,我们可能会想: 即使只用如此少的参数进行微调, 模型为什么能表现良好? 是不是使用更多可训练参数会更有利?

为了回答这个问题, 我们可以看一下之前的研究, 其表明大型模型(如LLM)往往具有低内在维度。

虽然这个想法听起来很复杂, 但它只是意味着超大型模型的权重矩阵往往是低秩的。

换句话说,并非所有这些参数都是必需的!

我们可以通过将这些权重矩阵分解为一个具有更少可训练参数的表示来实现可比的性能,如上所述。

"Aghajanyan等人表明, 学习到的过参数化模型实际上驻留在一个低内在维度上。我们假设模型自适应过程中权重的变化也具有低内在秩。"

鉴于大型模型的参数具有低内在维度, 我们可以合理地推断经过微调的模型也是如此——从微调得到的重更新也应该具有低内在维度。因此, 像LORA这样使用低秩分解来近似微调更新的技术应该能够高效有效地学习, 从而产生出即使只有少量可训练参数也能表现出色的模型。

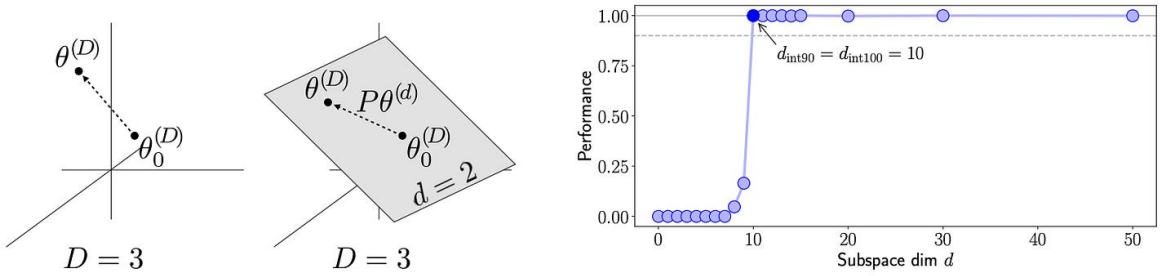


Figure 1: **(left)** Illustration of parameter vectors for direct optimization in the $D = 3$ case. **(middle)** Illustration of parameter vectors and a possible random subspace for the $D = 3, d = 2$ case. **(right)** Plot of performance *vs.* subspace dimension for the toy example of toy example of Sec. 2. The problem becomes both 90% solvable and 100% solvable at random subspace dimension 10, so d_{int90} and d_{int100} are 10.

这张图阐释了在低维子空间中优化问题的一些关键概念。

左侧图示说明了当问题维度 D=3 时,参数向量的直接优化情况。

中间图示展示了当 D=3 且子空间维度 d=2 时, 参数向量和一个可能的随机子空间的关系。

右侧的图表显示了一个玩具示例问题的性能与子空间维度之间的关系。当随机子空间维度达到 10 时,该问题变为 90% 可解和 100% 可解。因此,d int90 和 dint100 (分别表示 90% 和 100% 可解所需的内在维度)的值都是 10。

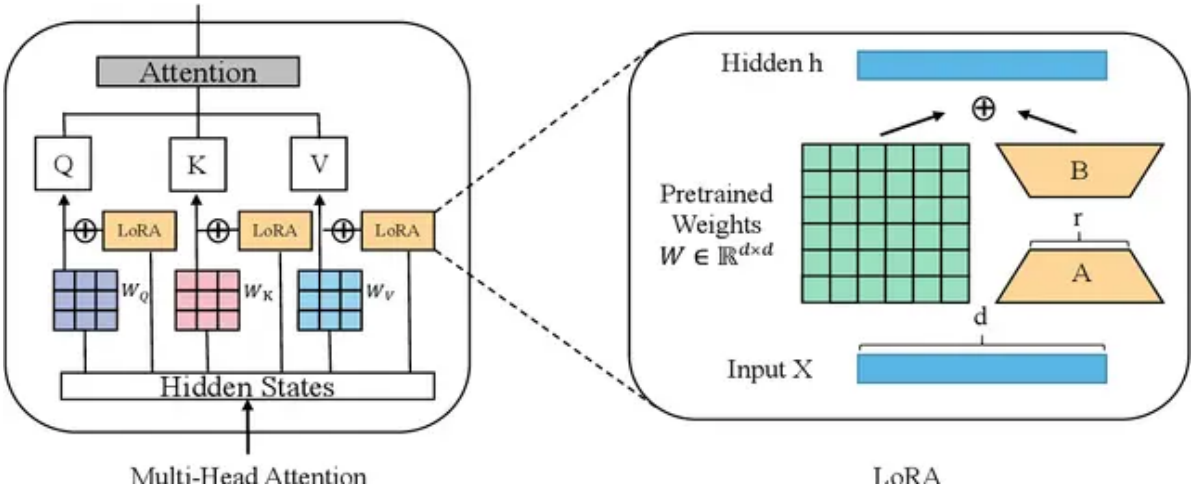
这个图总结了在低维度子空间中优化时, 存在一个内在维度(或临界维度)可以使问题在该子空间内高效求解。

一旦子空间的维度达到或超过这个临界值, 问题就可以在低秩条件下得到令人满意的解。

这个概念为像 LoRA 这样基于低秩分解的微调技术提供了理论基础, 解释了为什么在极少的参数下也能取得良好性能。

可选择只对Q和V的线性映射进行微调

LoRA可以选择只对Q和V的线性映射进行微调, 而不对K的线性映射进行微调。



LoRA（Low-Rank Adaptation）是一种用于微调大型预训练模型的方法。

它通过在原始模型参数矩阵中添加低秩矩阵来进行微调，从而减少需要调整的参数数量。

这种方法通常用于减少微调的计算和存储成本。

在Transformer架构中，Q（查询）、K（键）和V（值）是自注意力机制的关键组成部分。它们分过线性变换从输入中得到。

LoRA可以选择只对Q和V的线性映射进行微调，而不对K的线性映射进行微调，主要原因有以下几点：

1. **减少计算复杂度**：Q和V是注意力机制中的两个核心部分，Q用于生成注意力权重，V则是经过加权求和的值。通过微调Q和V，可以有效地影响注意力机制的行为，而不需要对K进行调整，从而减少了计算复杂度。
2. **重点在于提高表示能力**：Q和V的线性变换主要负责生成注意力权重和加权求和的值，调整这两个部分可以更直接地影响模型的输出结果。K的线性变换则主要用于计算注意力权重，它的调整对最终输出的影响相对间接。因此，只微调Q和V可以更有效地提高模型的表示能力和性能。
3. **避免过拟合**：在某些情况下，微调过多的参数可能会导致模型过拟合。通过只微调Q和V，可以减少需要调整的参数数量，从而降低过拟合的风险。
4. **经验性策略**：在实践中，研究者们发现只微调Q和V可以在保证模型性能的前提下，显著减少微调所需的资源和时间。这种经验性策略也得到了很多实验的验证，成为一种常用的做法。

总之，LoRA选择只对Q和V进行微调，而不对K进行微调，是为了在保证模型性能的前提下，尽量减少微调的计算复杂度和资源消耗，同时降低过拟合的风险。

QLoRA量化微调

随着大型语言模型(LLM)在自然语言处理任务中取得了卓越表现, 如何高效微调这些参数庞大的模型成为一个关键挑战。

LoRA(Low-Rank Adapation)技术的出现为解决这个难题提供了新思路, 但仍有进一步优化的空间。

这时, QLoRA(Quantized-LoRA)应运而生, 成为最受欢迎的LoRA变体之一。

QLoRA 是什么?

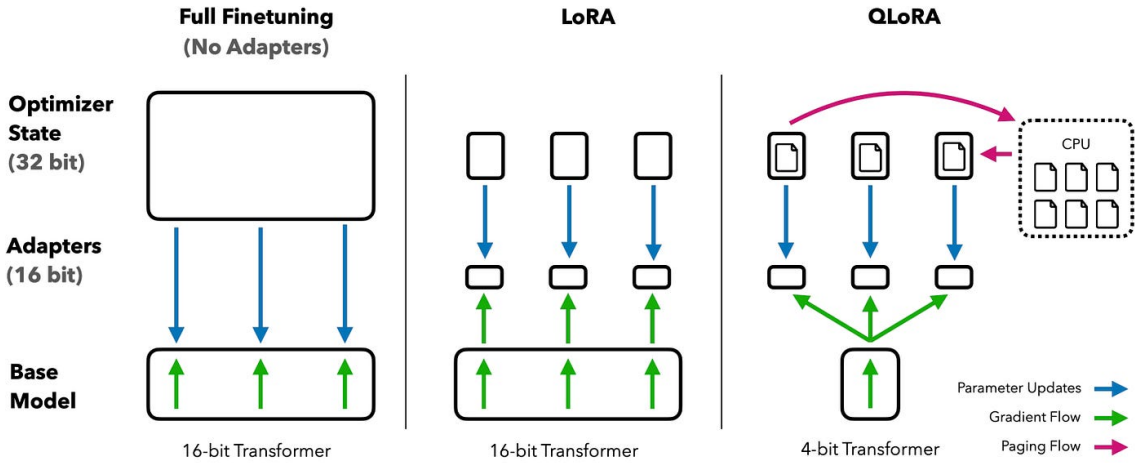


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

QLoRA 是一种结合了 LoRA 低秩微调和量化技术的创新方法。

它的核心思想是: 在 LoRA 基础上,通过对预训练模型权重进行量化, 从而进一步降低微调时的内存占用, 同时(大致)保持与完全精度微调相当的性能表现。

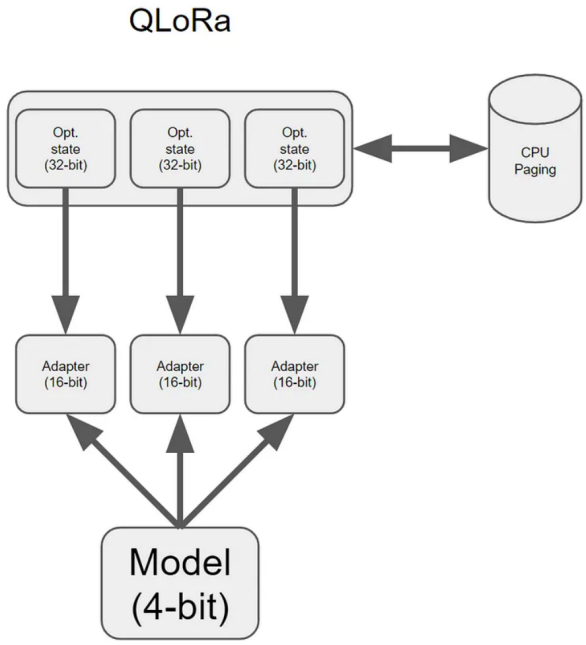
具体而言, QLoRA 采用 4 位精度对预训练模型权重进行量化, 然后在量化模型之上训练 LoRA 模块。

与此同时,QLoRA 还引入了多种创新性的量化技术, 以实现更高效的内存利用:

1. **4 位 NormalFloat (NF4)数据格式**: 一种新颖的量化数据类型, 专门针对服从正态分布的权重优化而设计。
2. **双重量化**: 不仅量化模型权重, 还量化其对应的量化常数, 进一步减小内存占用。

3. **分页优化器**: 防止梯度校验导致的内存峰值, 避免长序列处理或大模型训练时出现内存不足错误

通过上述措施 ,QLoRA 能以最小的内存代价执行高质量的 LLM 微调。



QLoRA的工作流程是:

1) 首先对预训练的大型语言模型权重进行量化, 通常采用4位精度量化。

2) 在量化后的模型基础上, 再使用LoRA的方式进行微调, 即注入两个小的低秩矩阵来适配注意力和前馈网络。

所以QLoRA是先量化预训练模型的"主体"权重, 降低模型本身的内存占用, 然后再在量化模型之上使用LoRA技术进行精细微调和适配,进一步减少微调所需的额外内存。

通过这种"量化+LoRA"的两步走策略, QLoRA能最大程度节省内存, 同时还能较好地保留模型性能。

量化降低了预训练模型本身的内存占用, 而LoRA则只需引入很少的可训练参数, 避免了完全微调的巨大内存代价。

该方法的关键在于合理权衡: 通过适度量化牺牲一些精度, 换来了极大的内存节省空间, 使得大模型微调在有限资源下成为可能。

而LoRA则提供了一种高效的"矫正"机制,弥补了量化带来的性能损失。二者合力,实现了内存高效和性能均衡。

QLoRA 的优势

QLoRA 最关键的优势在于极大地节省了内存使用, 使得在个人电脑或小型 GPU 上微调 LLM 也成为可能。

除了内存节省之外,QLoRA 还具有以下优点:

1. 性能影响有限: 尽管采用量化, 但 QLoRA 能(大致)保持与完全精度微调相当的性能水平。
2. 部署高效: QLoRA 最终得到的量化模型在推理时也更加高效, 对于边缘设备部署很有潜力。
3. 与 LoRA 兼容: QLoRA 可与 LoRA 无缝集成, 能够继承 LoRA 诸多优点。
4. 灵活可扩展: QLoRA 的底层量化技术可移植到其他场景, 从而推进量化深度学习的发展。

根据论文, **4位NormalFloat (NF4)**数据格式是这样介绍的:

NF4是一种新颖的量化数据类型, 专门针对服从正态分布的权重进行优化设计。它的构造过程如下:

1. 估计理论上一个标准正态分布N(0,1)的2^4+1=17个分位数, 得到一个4位精度的量化数据类型, 将正态分布中的值均匀分配到每个量化bin中。
2. 将这个数据类型的值归一化到[-1,1]的范围内。
3. 通过绝对最大值缩放, 将需要量化的权重张量也归一化到[-1,1]范围。
4. 使用步骤1中得到的量化数据类型对步骤3中的归一化权重张量进行量化。

该数据类型在理论上是对于正态分布数据进行量化的最优方案, 每个量化bin中的值数量是相等的。论文中还提到, 为了使量化值中存在0这个特殊值, 会对正负值分别估计分位数, 并去掉一个0值。

NF4数据类型的确切取值在论文的附录中给出。通过这种方式, NF4能高效地对预训练模型的正态分布权重进行4位量化,并保留较高的信息量。

另外，论文中介绍了**"双重量化"(Double Quantization)**的技术, 目的是进一步降低量化常数(quantization constants)的内存占用:

先来看看普通的量化方法。当我们想要把一个模型从32位浮点数精度压缩到更小的比特位(比如4位或8位)时, 需要一个"量化常数"来帮助完成从高精度到低精度的转换。这个量化常数本身也需要用32位浮点数来存储, 所以会占用额外的内存。

双重量化的诀窍就在于: 它对这个"量化常数"本身也进行了量化!

具体来说:

1) 首先用普通方法对模型权重进行第一次量化, 得到一个32位精度的"量化常数"。

2) 然后对这个32位的"量化常数"再进行一次量化, 压缩成8位精度, 就得到了"量化过的量化常数"。

3) 存储的时候, 就只需要存8位的"量化过的量化常数"和一个32位均值即可。

通过这个小技巧, 双重量化把原本32位的"量化常数"压缩成了只需8位存储, 从而进一步减小了模型的整体内存占用。

总的来说, 双重量化就是给"量化常数"也做了一次量化, 是一个降低内存需求的妙招, 使得大模型在有限显存条件下也能被微调训练。

下面给出量化和反量化的数学公式, 以说明"量化常数"是如何帮助实现精度转换的。

假设我们有一个32位浮点数张量 X, 想将它量化为k位精度的张量 X_quantized。

量化过程可以表示为:

X_quantized = round(c * X)

其中 c 是"量化常数", 它的作用是将X这个32位浮点数张量缩放到量化数据类型的值域范围内, 通常是[-127, 127]这样的整数区间。

c的计算公式是:

c = (最大量化值 - 最小量化值) / (X的最大绝对值)

例如, 如果我们要量化为8位整数, 即k=8, 那么量化值域就是[-127, 127]。

假设X的绝对最大值是10.0, 那么:

c = (127 - (-127)) / 10.0 = 254 / 10.0 = 25.4

量化后得到 X_quantized 就是一个8位整数张量了。

反量化过程为:

X_dequantized = X_quantized / c

可以看出, 量化常数c扮演了一个缩放的角色, 将32位浮点数张量缩放到量化值域范围内, 使得能够用较少的比特位表示。

这里给出了最基本的量化方法, 实际操作中通常会对数据分块并分别量化、或使用其他高级量化技术, 本质上都需要利用量化常数c来完成精度转换。

根据论文, **分页优化器(Paged Optimizers)**是QLORA提出的一项创新技术, 目的是防止梯度校验(gradient checkpointing)导致的内存峰值问题。

具体来说, 在训练大型语言模型时, 处理长序列或者使用大批量时, 计算激活梯度所需的内存可能会突然剧增,导致超出GPU显存容量而出现内存不足错误。传统的解决方案是减小批量大小或序列长度, 但这会影响训练效率。

QLORA的**分页优化器**利用了NVIDIA的统一内存(Unified Memory)特性, 通过在CPU和GPU之间自动分页传输数据, 从而避免了内存不足错误。

分页优化器的工作原理是:

1. 为优化器状态(如momentums)分配一块可分页的统一内存。
2. 当GPU显存不足时, 统一内存会自动将部分数据页从GPU内存转移至CPU内存。
3. 在优化器更新步骤时, 需要的数据页会自动从CPU内存传输回GPU进行计算。
4. 通过这种分页机制, 大型模型和长序列的训练不再受GPU显存限制。

分页优化器让QLORA能够在单GPU上微调包括65B参数在内的超大规模语言模型, 而避免了传统梯度校验中内存峰值所带来的内存不足问题, 极大提高了训练的可行性和效率。