# 大模型（LLMs）微调面

来自： AiGC面试宝典

宁静致远

2023年12月24日 00:44





## 1. 如果想要在某个模型基础上做全参数微调，究竟需要多少显存?

一般 n B的模型，最低需要 16-20 n G的显存。（cpu offload基本不开的情况下）
vicuna-7B为例，官方样例配置为 4*A100 40G，测试了一下确实能占满显存。（global batch size 128，max length 2048）当然训练时用了FSDP、梯度累积、梯度检查点等方式降显存。

## 2. 为什么SFT之后感觉LLM傻了?

- 原版答案：

SFT的重点在于激发大模型的能力，SFT的数据量一般也就是万恶之源alpaca数据集的52k量级，相比于预训练的数据还是太少了。
**如果抱着灌注领域知识而不是激发能力的想法，去做SFT的话，可能确实容易把LLM弄傻。**

- 新版答案：

**指令微调是为了增强（或解锁）大语言模型的能力。**
其真正作用：
**指令微调后，大语言模型展现出泛化到未见过任务的卓越能力**，即使在多语言场景下也能有不错表现 。

## 3. SFT 指令微调数据 如何构建?

1. **代表性。**应该**选择多个有代表性的任务**；
2. **数据量。每个任务实例数量不应太多**（比如：数百个）否则可能会**潜在地导致过拟合问题并影响模型性能**；
3. **不同任务数据量占比。**应该**平衡不同任务的比例**，并且限制整个数据集的容量（通常几千或几万），**防止较大的数据集压倒整个分布。**

## 4. 领域模型Continue PreTrain 数据选取?

技术标准文档或领域相关数据是领域模型Continue PreTrain的关键。因为领域相关的网站和资讯重要性或者知识密度不如书籍和技术标准。

## 5. 领域数据训练后，通用能力往往会有所下降，如何缓解模型遗忘通用能力?

- 动机：仅仅使用领域数据集进行模型训练，模型很容易出现灾难性遗忘现象.
- 解决方法：通常在领域训练的过程中加入通用数据集

那么这个比例多少比较合适呢?
目前还没有一个准确的答案。主要与领域数据量有关系，当数据量没有那么多时，一般领域数据与通用数据的比例在1:5到1:10之间是比较合适的。

## 6. 领域模型Continue PreTrain ，如何 让模型在预训练过程中就学习到更多的知识?

领域模型Continue PreTrain时可以同步加入SFT数据，即MIP，Multi-Task Instruction PreTraining。
预训练过程中，可以加下游SFT的数据，可以让模型在预训练过程中就学习到更多的知识。

## 7. 进行SFT操作的时候，基座模型选用Chat还是Base?

仅用SFT做领域模型时，资源有限就用在Chat模型基础上训练，资源充足就在Base模型上训练。（资源=数据+显卡）
资源充足时可以更好地拟合自己的数据，如果你只拥有小于10k数据，建议你选用Chat模型作为基座进行微调；如果你拥有100k的数据，建议你在Base模型上进行微调。

## 8. 领域模型微调 指令&数据输入格式 要求?

在Chat模型上进行SFT时，请一定遵循Chat模型原有的系统指令&数据输入格式。
建议不采用全量参数训练，否则模型原始能力会遗忘较多。

## 9. 领域模型微调 领域评测集 构建?

领域评测集时必要内容，建议有两份，一份选择题形式自动评测、一份开放形式人工评测。
选择题形式可以自动评测，方便模型进行初筛；开放形式人工评测比较浪费时间，可以用作精筛，并且任务形式更贴近真实场景。

## 10. 领域模型词表扩增是不是有必要的?

领域词表扩增真实解决的问题是解码效率的问题，给模型效果带来的提升可能不会有很大。

## 11. 如何训练自己的大模型?

如果我现在做一个sota的中文GPT大模型，会分2步走：1. 基于中文文本数据在LLaMA-65B上二次预训练; 2. 加CoT和instruction数据, 用FT + LoRA SFT。
提炼下方法，一般分为两个阶段训练：

- 第一阶段：扩充领域词表，比如金融领域词表，在海量领域文档数据上二次预训练LLaMA模型；
- 第二阶段：构造指令微调数据集，在第一阶段的预训练模型基础上做指令精调。还可以把指令微调数据集拼起来成文档格式放第一阶段里面增量预训练，让模型先理解下游任务信息。

当然，有低成本方案，因为我们有LoRA利器，第一阶段和第二阶段都可以用LoRA训练，如果不用LoRA，就全参微调，大概7B模型需要8卡A100，用了LoRA后，只需要单卡3090就可以了。

## 12. 训练中文大模型有啥经验?

链家技术报告《Towards Better Instruction Following Language Models for Chinese: Investigating the Impact of Training Data and Evaluation》中，介绍了开源模型的训练和评估方法：

Table 1: A simple overview of public available chat models. More details could be found in 2.3.

| Project | Base model | Training | Training data | Evaluation data | Evaluation method |
|---|---|---|---|---|---|
| Stanford alpaca | LLaMA | Full-parameter finetuning | 52K text-davinci-003 generated instruction data | 252 samples from self-instruct evaluation dataset | Human evaluation |
| LLaMA-GPT4 | LLaMA | Full-parameter finetuning | 52K GPT-4 generated instruction data | 1. 252 user-oriented instructions<br>2. 80 vicuna test samples | 1. Human evaluation<br>2. Automatic GPT-4 evaluation |
| Vicuna | LLaMA | Full-parameter finetuning | 70K user-shared conversations with ChatGPT | 80 vicuna test samples | Automatic GPT-4 evaluation |
| Koala | LLaMA | Full-parameter finetuning | 1. Stanford alpaca<br>2. Anthropic HH<br>3. OpenAI webgpt<br>4. OpenAI summarization | 1. 180 samples from self-instruct evaluation dataset<br>2. 180 real user queries that were posted online | Human evaluation |
| Dolly | GPT-J | Full-parameter finetuning | Stanford alpaca | - | Case demonsrtration |
| Dolly 2.0 | Pythia | Full-parameter finetuning | 15k human-written instruction data | - | Case demonsrtration |
| Baize | LLaMA | LoRA | 15K ChatGPT generated multi-turn conversations | - | Case demonsrtration |



还对比了各因素的消融实验：

| Factor | Base model | Training data | Score_w/o_others |
|---|---|---|---|
| 词表扩充 | LLaMA-7B-EXT | zh(alpaca-3.5&4) + sharegpt | 0.670 |
| | LLaMA-7B | zh(alpaca-3.5&4) + sharegpt | 0.652 |
| 数据质量 | LLaMA-7B-EXT | zh(alpaca-3.5) | 0.642 |
| | LLaMA-7B-EXT | zh(alpaca-4) | 0.693 |
| 数据语言分布 | LLaMA-7B-EXT | zh(alpaca-3.5&4) | 0.679 |
| | LLaMA-7B-EXT | en(alpaca-3.5&4) | 0.659 |
| | LLaMA-7B-EXT | zh(alpaca-3.5&4) + sharegpt | 0.670 |
| | LLaMA-7B-EXT | en(alpaca-3.5&4) + sharegpt | 0.668 |
| 数据规模 | LLaMA-7B-EXT | zh(alpaca-3.5&4) + sharegpt | 0.670 |
| | LLaMA-7B-EXT | zh(alpaca-3.5&4) + sharegpt + BELLE-0.5M-CLEAN | 0.762 |
| - | ChatGPT | - | 0.824 |



消融实验结论：

- 扩充中文词表后，可以增量模型对中文的理解能力，效果更好
- 数据质量越高越好，而且数据集质量提升可以改善模型效果
- 数据语言分布，加了中文的效果比不加的好
- 数据规模越大且质量越高，效果越好，大量高质量的微调数据集对模型效果提升最明显。解释下：数据量在训练数据量方面，数据量的增加已被证明可以显著提高性能。值得注意的是，如

此巨大的改进可能部分来自belle-3.5和我们的评估数据之间的相似分布。评估数据的类别、主题和复杂性将对评估结果产生很大影响

- 扩充词表后的LLaMA-7B-EXT的评估表现达到了0.762/0.824=92%的水平

他们的技术报告证明中文大模型的训练是可行的，虽然与ChatGPT还有差距。这里需要指出后续RLHF也很重要，我罗列在这里，抛砖引玉。

## 13. 指令微调的好处?

有以下好处：

1. 对齐人类意图，能够理解自然语言对话（更有人情味）
2. 经过微调（fine-tuned），定制版的GPT-3在不同应用中的提升非常明显。OpenAI表示，它可以让不同应用的准确度能直接从83%提升到95%、错误率可降低50%。解小学数学题目的正确率也能提高2-4倍。（更准）

踩在巨人的肩膀上、直接在1750亿参数的大模型上微调，不少研发人员都可以不用再重头训练自己的AI模型了。（更高效）

## 14. 预训练和微调哪个阶段注入知识的?

预训练阶段注入知识的，微调是在特定任务训练，以使预训练模型的通用知识跟特定任务的要求结合，使模型在特定任务上表现更好。

## 15. 想让模型学习某个领域或行业的知识，是应该预训练还是应该微调?

可以使用预训练和微调相结合的方式，先用篇章数据进行预训练以获取广泛的知识，再用问答对数据进行微调，使模型更好的学习到特定领域的知识。

当然，GPT大模型的预训练和微调，从实现方式来讲是没有什么差别的，都是decoder only的语言模型训练并更新参数，如果样本集小，没有大量的篇章文档数据，我认为只进行微调也能注入知识的，不必太纠结预训练。而且特定领域跟预训练模型的分布差别不大，也不用二次预训练。

## 16. 多轮对话任务如何微调模型?

这里列举了 ChatGLM-6B 的生成对话的例子

```
    >>> from transformers import AutoTokenizer, AutoModel
    >>> tokenizer = AutoTokenizer.from_pretrained("THUDM/chatglm-6b",
trust_remote_code=True)
    >>> model = AutoModel.from_pretrained("THUDM/chatglm-6b",
trust_remote_code=True).half().cuda()
    >>> model = model.eval()
    >>> response, history = model.chat(tokenizer, "你好", history=[])
    >>> print(f"response:{response}")
    >>> print(f"history:{history}")
    response：你好👋!我是人工智能助手 ChatGLM-6B,很高兴见到你,欢迎问我任何问题。
    history：["你好", "你好👋!我是人工智能助手 ChatGLM-6B,很高兴见到你,欢迎问我任
何问题。"]
```

- response 为 ChatGLM-6B 模型的 当前反馈

- history 为 ChatGLM-6B 模型的 历史记录的保存

说白了，就是 ChatGLM-6B 模型简单的把上一轮对话扔进下一轮的input里，这种方法好处是简单，缺点是随着轮数的增加，history 存储的 对话会越来越多，导致 max_length 增加，从而 出现爆显问题。

- 解决方法：
    - 对 历史对话 做一层文本摘要，取其精华去其糟粕
    - 将 历史对话 做成一个 embedding
    - 如果是 任务型对话，可以将 用户意图 和 槽位 作为 上一轮信息 传递给 下一轮

## 17. 微调后的模型出现能力劣化，灾难性遗忘是怎么回事?

所谓的**灾难性遗忘**：即学习了新的知识之后，几乎彻底遗忘掉之前习得的内容。这在微调ChatGLM-6B模型时，有同学提出来的问题，表现为原始ChatGLM-6B模型在知识问答如“失眠怎么办”的回答上是正确的，但引入特定任务（如拼写纠错CSC）数据集微调后，再让模型预测“失眠怎么办”的结果就答非所问了。

我理解ChatGLM-6B模型是走完 “预训练-SFT-RLHF” 过程训练后的模型，其SFT阶段已经有上千指令微调任务训练过，现在我们只是新增了一类指令数据，相对大模型而已，微调数据量少和微调任务类型单一，不会对其原有的能力造成大的影响，所以我认为是不会导致灾难性遗忘问题，我自己微调模型也没出现此问题。

应该是微调训练参数调整导致的，微调初始学习率不要设置太高，lr=2e-5或者更小，可以避免此问题，不要大于预训练时的学习率。

## 18. 微调模型需要多大显存?

| 模型版本 | 7B | 13B | 33B | 65B |
| --- | --- | --- | --- | --- |
| 原模型大小（FP16） | 13 GB | 24 GB | 60 GB | 120 GB |
| 量化后大小（8-bit） | 7.8 GB | 14.9 GB | - | - |
| 量化后大小（4-bit） | 3.9 GB | 7.8 GB | 19.5 GB | 38.5 GB |



## 19. 大模型LLM进行SFT操作的时候在学习什么?

（1）预训练->在大量无监督数据上进行预训练，得到基础模型-->将预训练模型作为SFT和RLHF的起点。

(2) SFT-->在有监督的数据集上进行SFT训练，利用上下文信息等监督信号进一步优化模型-->将SFT训练后的模型作为RLHF的起点。

(3) RLHF-->利用人类反馈进行强化学习，优化模型以更好地适应人类意图和偏好-->将RLHF训练后的模型进行评估和验证，并进行必要的调整。

## 20. 预训练和SFT操作有什么不同

下面使用一个具体的例子进行说明。

```
问题：描述计算机主板的功能
```

```
回答：计算机主板是计算机中的主要电路板。它是系统的支撑。
```

进行预训练的时候会把这句话连接起来，用前面的词来预测后面出现的词。在计算损失的时候，问句中的损失也会被计算进去。
进行SFT操作则会构建下面这样一条训练语料。

```
输入：描述计算机主板的功能[BOS]计算机主板是计算机中的主要电路板。它是系统的支撑。[EOS]
标签：[......][BOS]计算机主板是计算机中的主要电路板。它是系统的支撑。[EOS]
```

其中[BOS]和[EOS]是一些特殊字符，在计算损失时，只计算答句的损失。在多轮对话中，也是一样的，所有的问句损失都会被忽略，而只计算答句的损失。
因此SFT的逻辑和原来的预训练过程是一致的，但是通过构造一些人工的高质量问答语料，可以高效地教会大模型问答的技巧。

## 21. 样本量规模增大，训练出现OOM错

- 问题描述：模型训练的样本数量从10万，增大300万，训练任务直接报OOM了。
- 解决方案，对数据并行处理，具体实现参考海量数据高效训练，核心思想自定义数据集本次的主要目标是使向量化耗时随着处理进程的增加线性下降，训练时数据的内存占用只和数据分段大小有关，可以根据数据特点，灵活配置化。核心功能分为以下几点:
    - 均分完整数据集到所有进程（总的GPU卡数）
    - 每个epoch训练时整体数据分片shuffle一次，在每个进程同一时间只加载单个分段大小数据集
    - 重新训练时可以直接加载向量化后的数据。

## 22. 大模型LLM进行SFT 如何对样本进行优化?

- 对于输入历史对话数据进行左截断，保留最新的对话记录。
- 去掉样本中明显的语气词，如嗯嗯，啊啊之类的。
- 去掉样本中不合适的内容，如AI直卖，就不应出现转人工的对话内容。
- 样本中扩充用户特征标签，如年龄，性别，地域，人群等

## 23. 模型参数迭代实验

验证历史对话轮次是否越长越好，通过训练两个模型，控制变量max_source_length | max_target_length，对训练好之后的模型从Loss、Bleu指标、离线人工评估等角度进行对比分析。
结论：从人工评估少量样本以及loss下降来看，历史对话长度1024比512长度好，后续如果训练可能上线模型，可以扩大到1024长度。

## 24. 微调大模型的一些建议

- 模型结构:
    - 模型结构+训练目标: Causal Decoder + LM。有很好的zero-shot和few-shot能力，涌现效应
    - layer normalization: 使用Pre RMS Norm
    - 激活函数: 使用GeGLU或SwiGLU
    - embedding层后不添加layer normalization，否则会影响LLM的性能

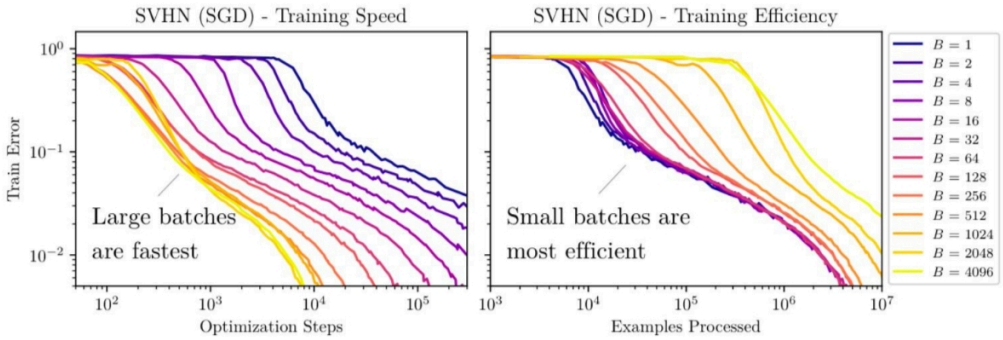
    - 位置编码: 使用ROPE或ALiBi。ROPE应用更广泛
    - 去除偏置项:去除dense层和layer norm的偏置项，有助于提升稳定性
- 训练配置:
    - batch: 选用很大的batch size; 动态地增加batch size的策略，GPT3逐渐从32K增加到3.2M tokens。
    - 学习率调度:先warmup再衰减。学习率先线性增长，再余弦衰减到最大值的10%。最大值一般在5e-5到1e-4之间。
    - 梯度裁剪:通常将梯度裁剪为1.0。
    - 权重衰减: 采用AdamW优化器，权重衰减系数设置为0.1Adamw相当于Adam加了一个L2正则项
    - 混合精度训练:采用bfloat16，而不是foat16来训练。
- 训练崩溃挽救:
    - 选择一个好的断点，跳过训练崩溃的数据段，进行断点重训。选择一个好的断点的标准:损失标度lossscale>0;梯度的L2范数<一定值 && 波动小

## 25. 微调大模型时，如果 batch size 设置太小 会出现什么问题?

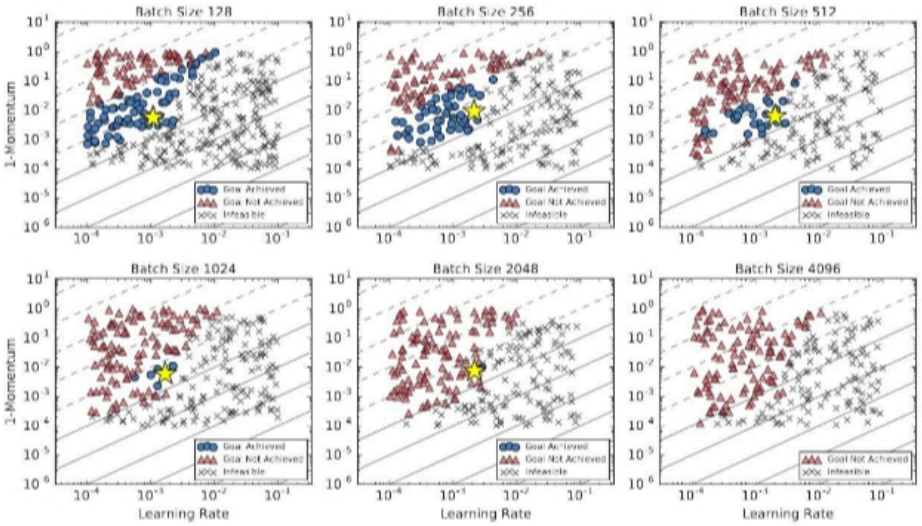
**当 batch size 较小时，更新方向（即对真实梯度的近似）会具有很高的方差，导致的梯度更新主要是噪声。**经过一些更新后，方差会相互抵消，总体上推动模型朝着正确的方向前进，但个别更新可能不太有用，可以一次性应用（使用更大 batch size 进行更新）。

## 26. 微调大模型时，如果 batch size 设置太大 会出现什么问题?

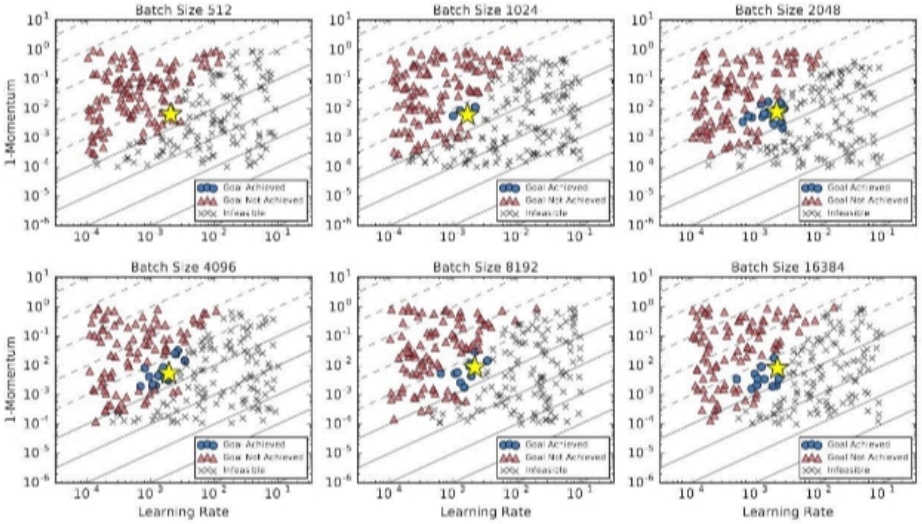
**当 batch size 非常大时，我们从训练数据中抽样的任何两组数据都会非常相似（因为它们几乎完全匹配真实梯度）**。因此，在这种情况下，增加 batch size 几乎不会改善性能，因为你无法改进真实的梯度预测。换句话说，你需要在每一步中处理更多的数据，但并不能减少整个训练过程中的步数，这表明总体训练时间几乎没有改善。但是更糟糕的是你增加了总体的 FLOPS。




通过观察这些线性图，我们可以发现使用更大的 batch size 通常需要较少的训练 step。然而，这将相应地增加需要处理的数据。当 batch size 从 2048 翻倍时，达到同样性能所需要的 step 几乎没有任何改善，但你需要花费两倍的计算资源。Google 的经验研究也有类似的观察，即在在固定的 epoch budget 下，当 batch size 达到临界值时，模型的性能会 batch size 的增加而降低。可以如下说明:




(a) Transformer on LM1B with a training budget of one epoch.




(b) Transformer on LM1B with a training budget of 25,000 steps.

## 27. 微调大模型时, batch size 如何设置问题?

各种结果表明似乎存在着一个关于数据并行程度的临界点，通过找到这个临界点，我们可以有效的平衡训练的效率和模型的最终效果。

OpenAI 发现最优步长:

$$\epsilon_{opt}(B) = argmin_{\epsilon}\mathbb{E}[L(\theta - \epsilon G_{est})] = \frac{\epsilon_{max}}{1+\mathcal{B}_{noise}/B}$$

注：B 为 batch size，*Bnoise*为 噪声尺度

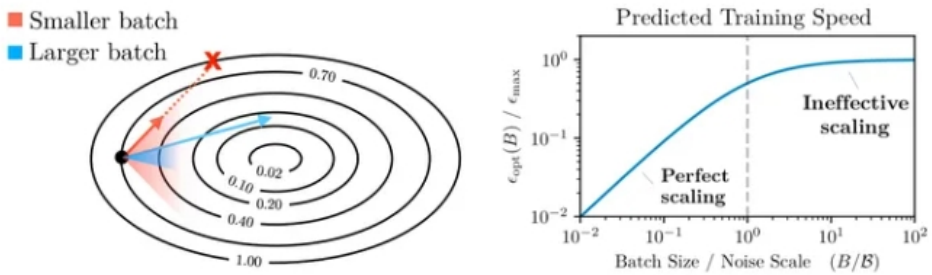
在采用最优 step size 时，从含有噪声的梯度中获得的损失的最优改进现在变为：

$$\Delta L_{opt}(B) = \frac{\Delta L_{max}}{1+\mathcal{B}_{noise}/B};\ \Delta L_{max} = \frac{|G|^4}{2G^THG}$$

从这些公式中我们可以得出两个结论：

1. 无论我们如何准确地估计真实梯度，总存在一个最大步长
2. 批处理大小越大，我们优化模型的步长就越大（有一个上限）

左侧的图表说明了为什么使用更大的批次模型可以取得更多提升。但是当 batch size 太大时，我们会遇到收益递减的问题（因为分母中的 1 开始占主导地位）。但是需要注意的事，这仅在学习率调整良好的情况下有效。因此，OpenAI 建议将学习率调整到一个相对接近最优值的数值是理论能有效的前提。



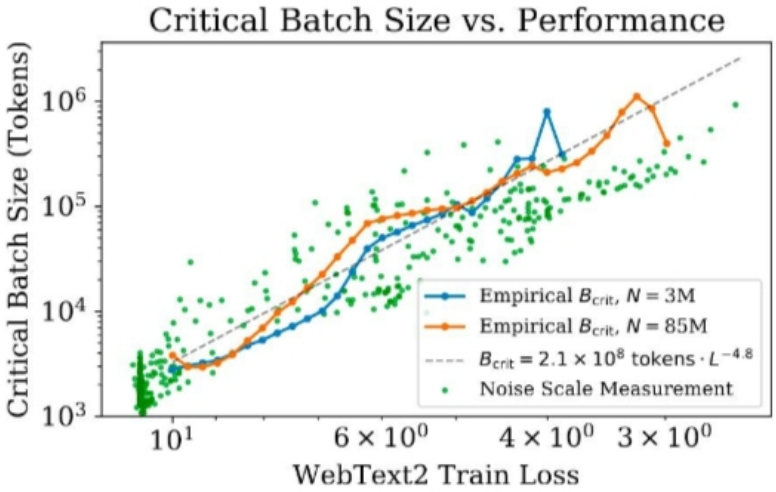

在进行一些其他数学计算后，OpenAI 发现噪声尺度可以通过以下方式估计：

$$\mathcal{B}_{noise} = \frac{tr(H\Sigma)}{G^T H G}$$

其中，H 是参数的真实 Hessian 矩阵，C 是相对于梯度的每个示例的协方差矩阵，g 是真实梯度。为了进一步简化这个方程，OpenAI作出了一个（不切实际的）假设，即优化是完全 well-conditioned 的。在这种情况下，Hessian 矩阵只是单位矩阵的倍数，噪声尺度简化就可以简化为以下形式：

$$\mathcal{B}_{simple} = \frac{tr(\Sigma)}{G^2}$$

他们经验上发现结果相当接近。该方程表明噪声尺度等于个别梯度分量的方差之和，除以梯度的 norm。OpenAI 使用以上结论在后续的 scaling law 工作中预测了模型的最优 batch size 大小。




- Learning rate as temperature

前面的结论有提到一个前提，就是模型的 LR 是调的比较好的。这是因为 OpenAI 发现噪声尺度基本符合以下规律

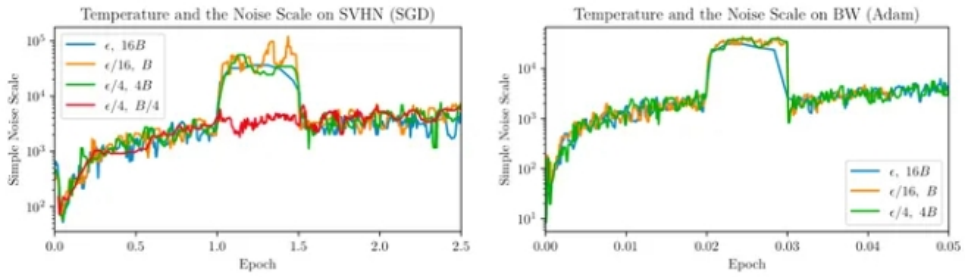
$$T(\epsilon, B) \equiv \frac{\epsilon}{\epsilon_{max}(B)}$$

在使用 SGD 和小 batch 进行更新时，可以大概近似为

$$T \approx \frac{\epsilon}{B}.$$

这表明

$$\mathcal{B}_{noise} \propto \mathcal{B}_{simple} \propto \frac{1}{T}$$




从以上内容，我们可以得知:

1. 高温度导致较小的噪声尺度。其中的直觉是在高温度下，相对于方差，梯度幅度较大。
2. 当学习率以一个常数因子衰减时，噪声尺度大致以相同的因子增长。因此，如果学习率太小，噪声尺度将被放大。

## 28. 微调大模型时, 优化器如何?

除了Adam和AdamW，其他优化器如Sophia也值得研究，它使用梯度曲率而非方差进行归一化，可能提高训练效率和模型性能。

## 29. 哪些因素会影响内存使用?

内存使用受到模型大小、批量大小、LoRA参数数量以及数据集特性的影响。例如，使用较短的训练序列可以节省内存。

## 30. 进行领域大模型预训练应用哪些数据集比较好?

通过分析发现现有的开源大模型进行预训练的过程中会加入书籍、论文等数据。主要是因为这些数据的数据质量较高，领域相关性比较强，知识覆盖率（密度）较大，可以让模型更适应考试。给我们自己进行大模型预训练的时候提供了一个参考。同时领域相关的网站内容、新闻内容也是比较重要的数据。

## 31. 用于大模型微调的数据集如何构建?

进行大模型微调时，数据是比较重要的，数据的高度决定模型效果的高度，因此数据的质量重要性大于数据的数量的重要性，因此对于构建微调数据时的几点建议如下所示：

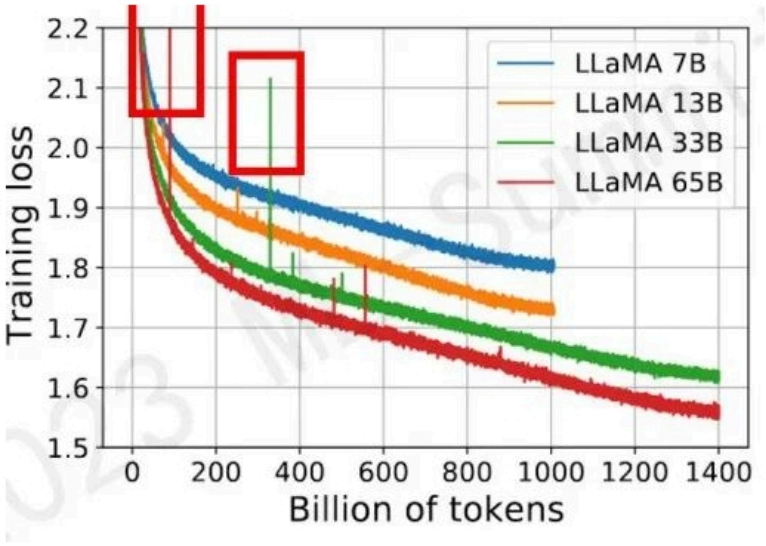
1. 选取的训练数据要干净、并具有代表性。
2. 构建的prompt尽量多样化，提高模型的鲁棒性。
3. 进行多任务同时进行训练的时候，要尽量使各个任务的数据量平衡。

## 32. 大模型训练loss突刺原因和解决办法

> 参考：A Theory on Adam Instability in Large-Scale Machine Learning

### 32.1 大模型训练loss突刺是什么?

loss spike指的是预训练过程中，尤其容易在大模型（100B以上）预训练过程中出现的loss突然暴涨的情况




如图所示模型训练过程中红框中突然上涨的loss尖峰 loss spike的现象会导致一系列的问题发生，譬如模型需要很长时间才能再次回到spike之前的状态（论文中称为pre-explosion），或者更严重的就是loss再也无法drop back down，即模型再也无法收敛
PaLM和GLM130b之前的解决办法是找到loss spike之前最近的checkpoint，更换之后的训练样本来避免loss spike的出现。

### 32.2 为什么大模型训练会出现loss突刺?

大模型训练使用的Adam优化器 会导致 loss突刺。
首先回顾一下Adam优化器的结构（这里介绍的是较为传统的Adam优化器，现在nlp任务更偏向于使用带有正则化项的Adamw变体）：

$$m_t = \frac{\beta_1}{1-\beta_1^t} m_{t-1} + \frac{1-\beta_1}{1-\beta_1^t} g_t$$

$$v_t = \frac{\beta_2}{1-\beta_2^t} v_{t-1} + \frac{1-\beta_2}{1-\beta_2^t} g_t^2$$

$$v_t = \frac{m_t}{\sqrt{v_t}+\varepsilon}$$

$\theta_{t+1} = \theta_t - \eta_t u_t$其中$\beta_1, \beta_2, \varepsilon$均为超参数( $\beta_1 = \beta_2$,$\varepsilon$防止除0)， $g_t$表示第t次更新的梯度， $m_t, v_t$，的初始值 $m_0, v_0$为0。

首先对Adam的有效性做了论述，其本质在于证明了Adam优化过程是对牛顿下降法（二阶导）的一个有效逼近，因此在收敛速度上大幅度领先传统SGD(一阶导)，证明过程不做赘述，可以参考本文和Adam系列相关论文
Adam算法是牛顿下降法的一个迭代逼近

that $m_t \approx \nabla f(\theta_t)$, and $v_t \gg \varepsilon$, would imply that the update of the Adam algorithm is an approximation of the update of the pure form of Newton's method, up to a constant multiplier:

$$u_t = \frac{m_t}{\sqrt{v_t}+\varepsilon} \approx \frac{m_t}{\sqrt{v_t}} \approx \left(\nabla^2 f(\theta_t)\right)^{-1} \nabla f(\theta_t).$$

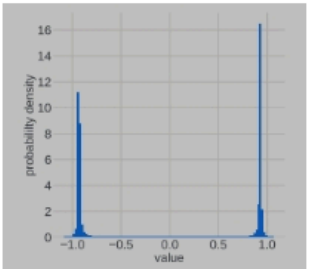
The quality of such an approximation would impact the convergence of the Adam dynamics and could explain the empirical success and popularity of the Adam algorithm.

Note that even if $\nabla f(\theta^\star) \neq 0$ but $\|\nabla f(\theta^\star)\nabla f(\theta^\star)^\top\| \ll \sigma^2$, the reasoning above goes through under the regularity assumption that $\mathbb{E}_\Delta \Delta = 0$ and slight modifications.

一切显得十分完美，但是理想很丰满，现实很骨感，收敛过程并不是一帆风顺的

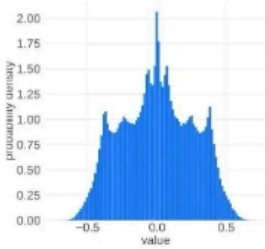
首先我们想象一下 ut 这个更新参数 的变化趋势

在 $t=1$的时候$m_1 = g_1$， $v_1 = g_1^2$， $u_1 = \frac{g_1}{\pm g1+\varepsilon}$， $u_1$ 集中在 ±1附近。而在训练到最后，假设模型收敛到某个最优点，此时 $u_n$应该集中在0附近。也就是说，我们似乎可以把更新参数的变化过程想象成为一个从两端（非稳态）向中间（稳态）收拢的过程。实际的观察现象也是如此：



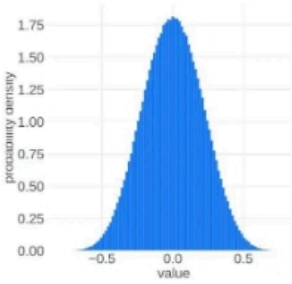

(d) $546b$

非稳态




(b) $u[G_0, t_2]$

中间态




稳态

进入正态分布的稳态之后，理想的更新参数变化趋势应该是方差越来越小，所有更新参数逐渐向0靠近。这应该是一个单向的过程，即稳定的单峰状态（unimodal）不会再次进入非稳定的双峰状态(bimodal)，但事实并非如此，更新参数会再次进入非稳定的双峰状态

本文在理论层面做了研究和解释，从中心极限定理（可以结合道尔顿板实验理解）出发，认为随机事件的叠加进入单峰的正态分布的必要条件之一是各个随机事件事件之间应该是相互独立的，但是

梯度变化以及更新参数的变化并不能特别好的满足独立性这一条件，而这一点恰恰是导致更新参数振荡，loss spike出现以及loss 不收敛的重要原因之一

4. The distribution of $g[i,t]$ for $i \in G$ becomes highly correlated across time domain for two reasons:
   - The model parameters remain unchanged ($\theta_{t+1} \approx \theta_t$) over the time steps as the update magnitude becomes close to zero.
   - The batch size for a large model is usually also very large, thus the gradient realizations have small time-domain variance.

造成梯度变化不独立的原因（1、浅层参数长时间不更新2、batch太大，后期梯度更新趋于平稳）
上述的理论有些晦涩，本文作者可能也了解这一点，之后开始直接点题，结合实验观察抛出了重要现象和结论
即训练过程中loss spike的出现与：梯度更新幅度，e 大小，batch大小这三个条件密切相关
本文作者对loss spike出现时模型的前后变化做了仔细拆解，发现下列一系列连续现象的出现导致了loss spike：

**Theory** Based on the experimental and theoretical results, we propose the following explanation for the origins and the behavior of the training instabilities observed in large-scale machine learning. It takes the form of a multi-stage process spanning the short period of a single model perplexity spike:

1. 当前模型处在稳态（健康状态），即单峰的正态分布状态，并且梯度值 $|g_t| \gg \varepsilon$，此时loss平稳，训练过程正常

   1. Healthy training: both $r[i,T]$ and $u[i,T]$ have uni-modal distributions, close to normal, with a standard deviation of the order of 1. This implies a low correlation between gradient estimates at consecutive steps and a high value of the gradient components relative to $\varepsilon$.

2.模型浅层(embedding层)梯度 $|g_t| \ll \varepsilon$，这一般是由于训练一段时间之后，浅层的语义知识表示此时一般已经学习的较好。但此时深层网络（对应复杂任务）的梯度更新还是相对较大

   2. The gradients of a group of parameters of the model (e.g. a layer, let us denote it with $G$) are vanishing ($\ll \varepsilon$) over the course of training. From our observations, this is most likely to happen in the earlier layers of the model, where a good feature representation has been learned, while the rest of the parameters further through the model keep gradients of high magnitude.

3.一段时间浅层(embedding层)梯度 $|g_t| \ll \varepsilon$之后会导致 $|m_t| \ll \varepsilon$， $v_t \ll \varepsilon$。此时$v_t = \frac{m_t}{\sqrt{v_t}+\varepsilon}$趋于0。因此导致浅层参数得不到更新（也对应于上述参数更新事件不独立的原因）

3. The optimizer state values $m[i,t]$ and $v[i,t]$ for $i \in G$ vanishing ($\ll \varepsilon$).
   - The update values $u[i,t]$ for $i \in G$ are vanishing due to $m[i,T]$ and $v[i,t]$ values dropping. The spatial distribution of $u[i,t]$ over $i \in G$ spikes at 0, dropping its variance. The distribution of $r[i,T]$ over $i \in G$ remains uni-modal for now, close to normal, variance of the order of 1.

4.此时虽然浅层(embedding层)参数长时间不更新，但是深层的参数依然一直在更新。长时间这样的状态之后，batch之间的样本分布变化可能就会直接导致浅层(embedding层)再次出现较大的梯度变化（可以想象成一个水坝蓄水太久终于被冲开了。至于小模型为什么不会出现这种情况，推测是小模型函数空间小，无法捕获样本的分布变化，越大规模的模型对样本之间不同维度的特征分布变化越敏感），此时 $|g_t| \gg \varepsilon$， $u_t \approx \frac{g_t}{\pm g t+\varepsilon}$再次集中在 $\pm 1$附近（此时 $|m_{t-1}| \ll \varepsilon, v_t - 1 \ll \varepsilon$），变成双峰的非稳定状态，本文提到了浅层(embedding层)这种突然的参数变化可能造成模型的连锁反应进而出现loss spike的现象（这也对应了更换样本重新训练有可能会减少loss spike的出现频率，实际上就是选择分布变化较小的样本，减小浅层梯度变换幅度）

5. The distribution of $r[i,t]$ over $i \in G$ changes from uni-modal to bi-modal. The distribution of $u[i,t]$ over $i \in G$ remains spiked at 0.

6. The model parameters $i$ outside the group $G$ slowly change their values because the values $u[i,t]$ corresponding to $i \notin G$ are still of order of 1. As the model changes, the probability that there would come a batch that would require a "reconsideration" of the feature maps learned in the earlier layers of the model, increases. Thus, increases the probability of a rare event (let's say it happens at time step $t^\star$), in which the $g[i,t^\star]$ for $i \in G$ becomes larger than $\varepsilon$. After this event, the distribution of $u[i,t]$ is going to depart from the spike form, approaching the bimodal distribution of $r[i,t]$. According to our analytical study described in Section 5, this implies divergence for the standard learning rate values, which means that the entries of the next gradient estimation $g[G,t^\star+1]$ must get an even larger magnitude, bringing the distribution of $u[i,t^\star+1]$ over $G$ even closer to the distribution of $r[i,t^\star+1]$. The process described here resembles a chain reaction, which we would expect to observe during a loss explosion. In Figures 4 it could be observed that the explosion of the gradient norm in the earlier layers of the model starts one training step earlier than the spike of perplexity metric.

5.这个阶段模型处于非稳态，梯度变化幅度较大，每一次的梯度变化和更新参数变化事件之间又出现了一定的独立性，因此经过一定的时间之后模型有可能再次进入稳态，loss再次drop back down（注意，本文着重提了这个再次drop back down并不是一定出现的，也很有可能loss长期处于flat状态，再也无法收敛）

因此我们得出一些结论，loss spike的出现和浅层的梯度更新幅度，e 大小密切相关（batch大小带来的相关性问题倒是显得没那么大说服力），实际上就是浅层网络参数突然进入到了之前长时间不在的状态与模型深层参数当前的状态形成了连锁反应造成了模型进入非稳态。同时一般情况即使出现loss spike也会自动回复到正常状态，但也有可能再也不会

### 32.3 大模型训练loss突刺 如何解决?

本文最后提到了防止loss spike出现的一些方法：

1. 如之前提到的PaLM和GLM130B提到的出现loss spike后更换batch样本的方法（常规方法，但是成本比较高）
2. 减小learning rate，这是个治标不治本的办法，对更新参数的非稳态没有做改进
3. 减小 e 大小。或者直接把 e 设为0，重新定义

$$v_t = \frac{m_t}{\sqrt{v_t}+\varepsilon}$$

在 vt 等于 0 时候的值（这应该是个值得尝试的办法）

值得一提的是智谱华章在本文发表之前，在去年的GLM130B训练时似乎也观察到了浅层梯度变化和loss spike相关这一现象（GLM-130B: An Open Bilingual Pre-trained Model），他采取的是把浅层梯度直接乘以缩放系数 a 来减小浅层梯度更新值

**Embedding Layer Gradient Shrink (EGS).** Our empirical search identifies that the gradient norm can serve as an informative indicator of training collapses. Specifically, we find that a training collapse usually lags behind a "spike" in gradient norm by a few training steps. Such spikes are usually caused by the embedding layer's abnormal gradients, as we observe that its gradient norm is often several magnitude larger that those of other layers in GLM-130B's early stage training (Cf. Figure 4 (a)). In addition, it tends to fluctuate dramatically in the early training. The problem is

出自130b

Finally, we find the gradient shrink on embedding layers could overcome loss spikes and thus stabilize GLM-130B's training. It is first used in the multi-modal transformer CogView (Ding et al., 2021). Let $\alpha$ be the shrinking factor, the strategy can be easily implemented via word_embedding $=$ word_embedding $* \alpha +$ word_embedding.detach() $* (1-\alpha)$. Figure 4 (b) suggests that empirically, setting $\alpha = 0.1$ wipes out most spikes we would have met, with negligible latency.

其实这块我有个自己的想法，e 和 a 是否也可以做衰减，随着训练过程逐渐减小，来避免loss spike的现象

另外假设我们能一次性加载所有样本进行训练（实际上不可能做到），是否还会出现loss spike的现象

最后目前流行的fp8，fp16混合训练，如果upscale设置的过小，导致梯度在进入优化器之前就下溢，是不是会增加浅层梯度长时间不更新的可能性，进而增加loss spike的出现的频率。（这么看来似乎提升upscale大小以及优化 e 大小是进一步提升模型效果的一个思路）