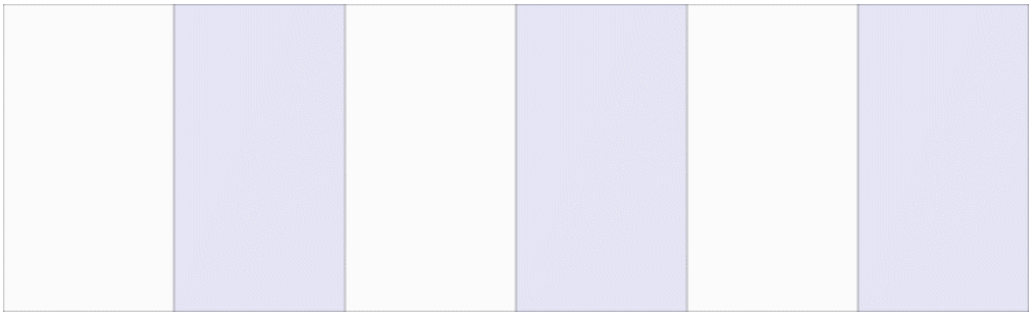
面试面经 | 大模型面试八股含答案

极市平台 2023-08-17 22:00:30 发表于广东 手机阅读

以下文章来源于包包算法笔记，作者包包闭关修炼

包包算法笔记
包大人的算法，程序，机器学习，职场，理财闲谈。



作者 | 包包算法笔记

来源 | 包包算法笔记

编辑 | 极市平台

极市导读

大模型面试问题总结。 >>加入极市CV技术交流群，走在计算机视觉的最前沿

大模型训练的一些坑点和判断在前面的文章写了大模型面试八股，原文没有写答案，后续有很多朋友要答案，这里一位朋友整理了部分答案，对于没回答的部分我做了一些补充。原作者：花甘者浅狐感谢这位朋友辛苦整理，访问知乎原文https://zhuanlan.zhihu.com/p/643560888

基础知识

1.transformer 八股文

a.Self-Attention的表达式

$$\text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

b.为什么上面那个公式要对QK进行scaling

scaling后进行softmax操作可以使得输入的数据的分布变得更好，你可以想象下softmax的公式，数值会进入敏感区间，防止梯度消失，让模型能够更容易训练。

c.self-attention一定要这样表达吗?

不一定，只要可以建模相关性就可以。当然，最好是能够高速计算（矩阵乘法），并且表达能力强（query可以主动去关注到其他的key并在value上进行强化，并且忽略不相关的其他部分），模型容量够。

d.有其他方法不用除根号$\sqrt{d_k}$吗?

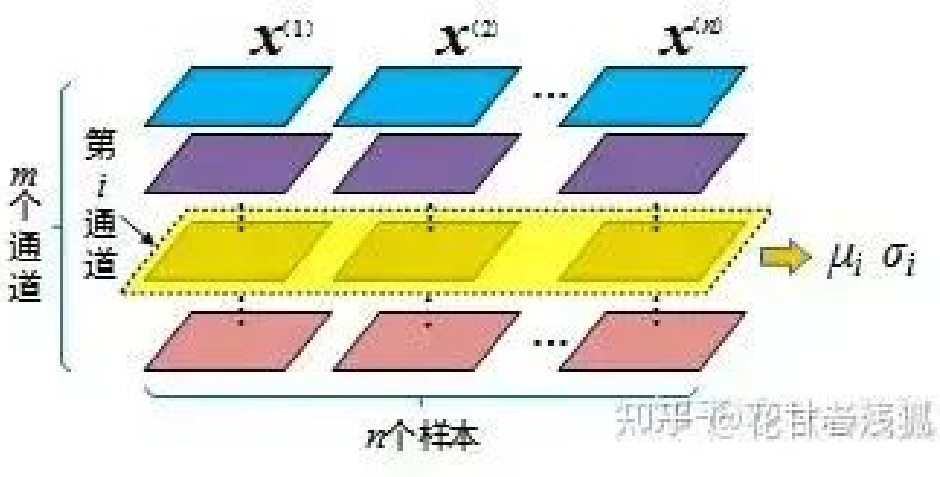
有，只要能缓解梯度消失的问题就可以。详情可以了解Google T5的Xavier初始化。

e.为什么transformer用Layer Norm？有什么用？

任何norm的意义都是为了让使用norm的网络的输入的数据分布变得更好，也就是转换为标准正态分布，数值进入敏感度区间，以减缓梯度消失，从而更容易训练。当然，这也意味着舍弃了除此维度之外其他维度的其他信息。为什么能舍弃呢？请看下一题。

f.为什么不用BN？

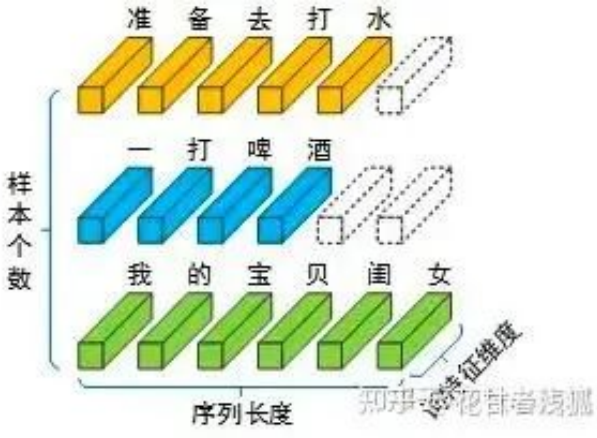
可以参考这个回答：为什么Transformer要用LayerNorm？我在这里做下总结。首先要明确，如果在一个维度内进行normalization，那么在这个维度内，相对大小有意义的，是可以比较的；但是在normalization后的不同的维度之间，相对大小这是没有意义的



这个图片很好，转自上面链接，BN相当于要做m次normalization

BN(batch normalization)广泛应用于CV，针对同一特征，以跨样本的方式开展归一化，也就是对不同样本的同一channel间的所有像素值进行归一化，因此不会破坏不同样本同一特征之间的关系，毕竟“减均值，除标准差”只是一个平移加缩放的线性操作。在“身高体重”的例子中，这就意味着“归一化前是高个儿的归一化后仍然是高个儿，归一化前胖的归一化后也不会变瘦”。这一性质进而决定了经过归一化操作后，样本之间仍然具有可比较性。但是，特征与特征之间的不再具有可比较性，也就是上一个问题中我所说的“舍弃了除此维度之外其他维度的其他信息”。既然前面说了是CV中用BN，那为什么NLP中不用BN，而用LN呢？道理一样，因为NLP中：

- 对不同样本同一特征的信息进行归一化没有意义：
- 三个样本（为中华之崛起而读书；我爱中国；母爱最伟大）中，“为”、“我”、“母”归一到同一分布没有意义。
- 舍弃不了BN中舍弃的其他维度的信息，也就是同一个样本的不同维度的信息：
- “为”、“我”、“母”归一到同一分布后，第一句话中的“为”和“中”就没有可比性了，何谈同一句子之间的注意力机制？



对比上面的图片，同转自上面链接，LN相当于要做3次normalization

加强一下，我们再回顾CV中：

- 对不同样本同一特征（channel）的信息进行归一化有意义；
- 因为同一个channel下的所有信息都是遵循统一规则下的大小比较的，比如黑白图中越白越靠近255，反之越黑越靠近0
- 可以舍弃其他维度的信息，也就是同一个样本的不同维度间（channel）的信息；
- 举例来说，RGB三个通道之间互相比较意义不大

g.Bert为什么要搞一个position embedding?

因为仅仅有之前提到的self-attention无法表达位置信息（对位置信息不敏感），比如说“1+1=2”和“1+2=1”是一样的，因此需要增强模型针对位置的表达能力。

h.Bert为什么三个embedding可以相加?

【深度玄学】为何Bert三个Embedding可以相加

这里的三个embedding是指token embedding，segment embedding，position embedding。如果感兴趣，还是来看Rethinking Positional Encoding in Language Pre-training原文，不过为了理解也可以看下邱老师的回答：为什么 Bert 的三个 Embedding 可以进行相加？如果你是在质疑加法会导致”信息损失“，但是本质上神经网络中每个神经元收到的信号也是“权重”相加得来。详细想想，在实际场景中，叠加是一个更为常态的操作。比如声音、图像等信号。一个时序的波可以用多个不同频率的正弦波叠加来表示。只要叠加的波的频率不同，我们就可以通过傅里叶变换进行逆向转换。一串文本也可以看作是一些时序信号，也可以有很多信号进行叠加，只要频率不同，都可以在后面的复杂神经网络中得到解耦（但也不一定真的要得到解耦）。在BERT这个设定中，token，segment，position明显可以对应三种非常不同的频率。由此可以再深入想一想，在一串文本中，如果每个词的特征都可以用叠加波来表示，整个序列又可以进一步叠加。哪些是低频信号（比如词性？），哪些是高频信号（比如语义？），这些都隐藏在embedding中，也可能已经解耦在不同维度中了。说不定可以是一种新的表示理论

i.transformer为什么要用三个不一样的QKV?

前面提到过，是为了增强网络的容量和表达能力。更极端点，如果完全不要project_q/k/v，就是输入x本身来做，当然可以，但是表征能力太弱了（x的参数更新得至少会很拧巴）

j.为什么要多头？举例说明多头相比单头注意力的优势

和上一问一样，进一步增强网络的容量和表达能力。你可以类比CV中的不同的channel（不同卷积核）会关注不同的信息，事实上不同的头也会关注不同的信息。假设我们有一个句子"the cat, which is black, sat on the mat"。在处理"sat"这个词时，一个头（主语头）可能会更关注"cat"，因为"cat"是"sat"的主语；另一个头（宾语头）可能会更关注"on the mat"，因为这是"sat"的宾语；还有一个头（修饰头）可能会关注"which is black"，因为这是对"cat"的修饰。当然，这只是为了方便你理解，事实上就和卷积核一样，不同头关注的内容是很抽象的。你当然可以就用一个头同时做这个事，但是还是这个道理，我们的目的就是通过增加参数量来增强网络的容量从而提升网络表达能力。经过多头之后，我们还需要att_out线性层来做线性变换，以自动决定（通过训练）对每个头的输出赋予多大的权重，从而在最终的输出中强调一些头的信息，而忽视其他头的信息。这是一种自适应的、数据驱动的方式来组合不同头的信息。

k.为什么Bert中要用WordPiece/BPE这样的subword Token?

避免OOV（Out Of Vocabulary），也就是词汇表外的词。在NLP中，通常会预先构建一个词汇表，包含所有模型能够识别的词。然而，总会有一些词没有出现在预先构建的词汇表中，这些词就是 OOV。传统的处理方式往往是将这些 OOV 映射到一个特殊的符号，如 <UNK>，但这种方式无法充分利用 OOV 中的信息。例如，对于词汇表中没有的词 "unhappiness"，如果直接映射为 <UNK>，则模型就无法理解它的含义。WordPiece/Byte Pair Encoding (BPE) 等基于子词的分词方法提供了一种解决 OOV 问题的方式。现在更多的语言大模型选择基于BPE的方式，只不过BERT时代更多还是WordPiece。BPE 通过将词分解为更小的单元（子词或字符），可以有效地处理词汇表外的词。对于上面的 "unhappiness" 例子，即使 "unhappiness" 本身不在词汇表中，但是它可以被分解为 "un"、"happiness" 等子词，而这些子词可能在词汇表中。这样，模型就可以通过这些子词来理解 "unhappiness" 的含义。另一方面就是，BPE本身的语义粒度也很合适，一个token不会太大，也不会小到损失连接信息（如一个字母）。

l.Bert中为什么要在开头加个[CLS]?

sliderSun：关于BERT中的那些为什么 其实这个回答也写了一些为什么，其中就包含这个题目。为了文章的完整性我再输出一点自己的观点。具体来说，我们想让[CLS]做的事情就是利用好BERT强大的表示能力，这个表示能力不仅限于token层面，而且我们尝试要得到整个seqence的表示。因此，[CLS]就是做这个事情的。具体来说，整个encoder的最后一层的[CLS]学到的向量可以很好地作为整句话的语义表示，从而适配一些setence层面的任务，如整句话的情感分类。那关键点就在于，为什么[CLS]可以建模整句话的语义表征呢？简单来说也很好理解，因为“这个无明显语义信息的符号会更“公平”地融合文本中各个词的语义信息，从而更好的表示整句话的语义。” ——为什么无明显语义？因为训练的时候BERT发现每个句子头都有，这样他能学到什么语义呢？——为什么要公平？因为控制变量，我们不希望做其他下游任务的时候用于区分不同样本间特征的信息是有偏的。当然，不放在句子开头的其他位置是否可行？一个未经考证的臆想是，任何其他位置的position embedding都无法满足放在开头的一致性。所以不同句子间可能会有一定的不同，这并不是我们做一些句间的分类问题想要的。

m.不用[CLS]的语义输出，有其他方式可以代替吗?

这个问题还是考察到了[CLS]的核心内涵，也就是如何获得整个sentence的语义表示。既然不让使用特意训好的[CLS]，那我们就从每个token得到的embedding入手，把所有的token弄到一起。很直观的思路，就是对BERT的所有输出词向量（忽略[CLS]和[SEP]）应用MaxPooling和AvgPooling，然后将得到的两个向量拼接起来，作为整个序列的表示。这样做的话可以同时保留序列中最显著的特征（通过MaxPooling）和整体的，均衡的特征（通过AvgPooling）。当然这种做法我本人并没有尝试过，或许也不是一个很好做的研究/工作方向。

n.Bert中有哪些地方用到了mask?

预训练任务Masked Language Model (MLM) self-attention的计算 下游任务的decoder

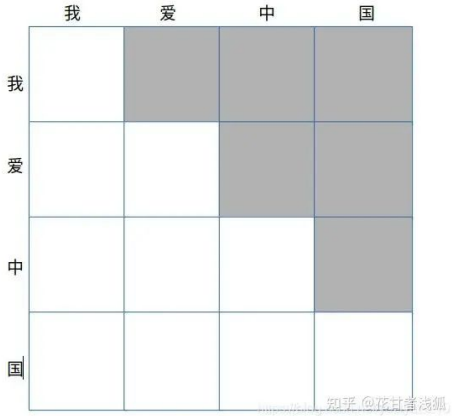
o.预训练阶段的mask有什么用?

虽然MLM现在被主流LLM抛弃了，但是也是一项很重要的任务。主要的思想是，把输入的其中一部分词汇随机掩盖，模型的目标是预测这些掩盖词汇。这种训练方式使得每个位置的BERT都能学习到其上下文的信息。

p.attention中的mask有什么用？（BERT中）

这是nlp任务很重要的问题，就是不同样本的seq_len不一样。但是由于输出的seq_len需要一致，所以需要通过补padding来对齐。而在attention中我们不希望一个token去注意到这些padding的部分，因为实际场景下它们是不存在的，所以attention中的mask就是来处理掉这些无效的信息的。具体来说就是在softmax前每个都设为-inf（或者实际的场景一个很小的数就可以），然后过完softmax后"padding"部分的权重就会接近于零，query token就不会分配注意力权重了。

q.decoder中的mask有什么用?



转自https://blog.csdn.net/yoziyezi210

这就是decoder-only的模型（自回归模型）架构问题，一般称为future mask，通常为一个上三角矩阵。简单来说，就是模拟inference的过程。比如，在做next token prediction的时候，模型是根据前面已有的tokens来做的，也就是看不到未来的tokens的信息。而在我们训练的过程中，通常采用teacher forcing的策略，也就是我们当然会把完整的标签喂给模型，但是由于在一个一个生成next token的过程中，模型应该是一个一个往外“蹦”字的过程（想想chatgpt回复你的样子）所以我们会遮盖掉seqence中当前位置之后信息，以防止模型利用未来信息，也就是信息泄露。mask掉后模型的注意力只会集中在此前的序列上。

r.Bert中self attention 计算复杂度如何?

$O(d_L^2)$，你可以参考上面那张图，因为输入的序列的每一个token都要对这个序列上的所有token去求一个attention score。

s.有什么技术降低复杂度提升输入长度的?

比如Sparse Attention，放弃对全文的关注，只关心局部的语义组合，相当于self-attention上又加了一些mask，这样的话就可以降低复杂度，而且下游任务的语义关联性的体现往往是局部/稀疏的。

t.Bert是如何处理传统方法难以搞定的溢出词表词(oov)的语义学习的?

前面提到了，关键词是subword。

u.中文是如何处理溢出词表词(oov)的语义学习的?

subword处理中文都是字符级别的，所以就不会有词级别oov的问题了。

v.为什么以前char level/subword level的NLP模型表现一般都比较差，但是到了bert这里就比较好?

还是归功于Transformers，因为对于字符级别（char-level）或者子词级别（subword-level）的NLP模型，挑战在于需要模型能够理解字符或者子词组合起来形成词语和句子的语义，这对模型的能力有很高的要求。然而，以前NLP模型没办法做到很深，两层lstm基本就到极限了，非线性成长路线过分陡峭，所以增加网络容量的时候，降低了泛化能力。Bert降低了输入的复杂度，提升了模型的复杂度。模型多层产生的非线性增长平滑，可以加大网络容量，同时增强容量和泛化能力。解释一下：——什么叫非线性成长路线过分陡峭？如果我们将模型的深度看作是 x 轴，模型的复杂度或训练难度看作是 y 轴，那么随着深度的增加，y 值的增长可能会变得非常快。——BERT为什么降低了输入复杂度？WordPiece这种subword的做法不至于像char level那样基本完全抛弃了自身的语义信息（因为切的太细就会太复杂），也不至于像word leve那样，因此可以减小词汇表大小。当然也避免了OOV的问题。——BERT为什么提升了模型复杂度？Transformers可以把网络做深，本身内部网络容量也很够。

w.Bert为什么要使用warmup的学习率trick

主要是考虑到训练的初始阶段params更新比较大，可能会使模型陷入local minima或者overfitting。warmup就是把lr从一个较小的值线性增大到预设，以减缓参数震荡，让训练变得比较smooth，当模型参数量上来后这种策略变得更重要了。

x.为什么说GPT是单向的Bert是双向的?

这也是decoder-only和encoder-only的区别。decoder-only架构的生成模型在输出的时候只能看到当前位置前的tokens，也就是屏蔽了序列后面的位置，以适配NTP任务。encoder-only架构的编码模型在输出的时候可以利用前后位置的tokens，以适配MLM任务。具体的做法是self-attention加不加casual mask，也就是遮不遮住序列后面的内容。

y.Bert如何处理一词多义?

一词多义指的是在不同句子中token有不同的含义。这正是self-attention解决的，搭配上MLM的任务，就可以让每个token会注意到上下文的其他token来得到自己的embedding。

z.Bert中的transformer和原生的transformer有什么区别?

其实很多，如果我们只讨论模型架构，也就是对比Attention is All You Need的encoder和BERT的话，最重点的区别在于位置编码。原生的transformer是最经典的Sinusoidal绝对位置编码。而BERT中变成了可以学习的参数，也就是可学习位置编码。变得可学了的话，只要模型学习能力强，数据量够，确实不会差。可以类比卷积核从手工变成了模型自己学。关于位置编码，如果你有时间的话，建议从下面的链接一直往后看，苏神的内容质量都很高。位置编码确实大有可为，最近RoPE+NTK的方法来外推context length也挺让人热血沸腾的。Transformer升级之路：1、Sinusoidal位置编码追根溯源 - 科学空间|Scientific Spaces

z+.Albert是通过什么方法压缩网络参数的？有什么问题?

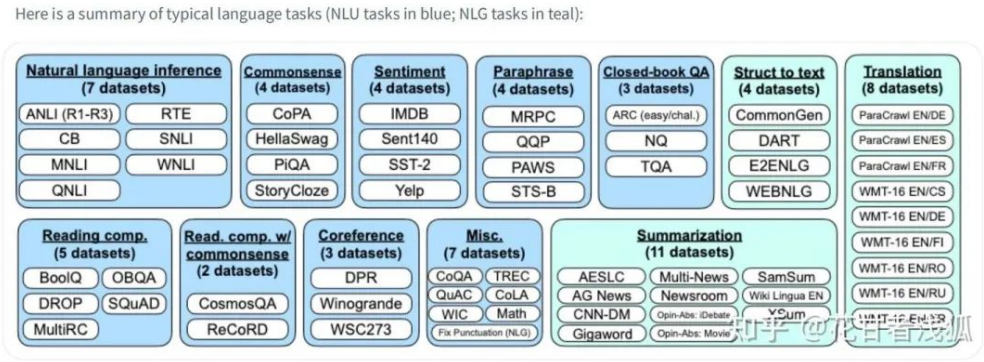
两个技巧，其一是参跨层数共享，其二是对嵌入参数化进行因式分解，也就是“不再将 one-hot 向量直接映射到大小为 H 的隐藏空间，先映射到一个低维词嵌入空间 E，然后再映射到隐藏空间”。问题也是“模型压缩”通用的问题，网络表达能力和容量下降。然后推理速度也不会有很直观的提升。

2.attention计算方式以及参数量，attention layer手写，必考。

如果你找的工作是比较基础的，比如说本科生找llm相关实习，那基本会让你手写多头。如果你想比较方便地一站对比各个Transformer模型的源码，可以来这个库：GitHub - OpenBMB/ModelCenter

3.NLU以及NLG各种任务的差异。

NLU（自然语言理解）& NLG（自然语言生成），“望文生义”，其实只是在任务目标上有区别，本人认为不用太重点区分。



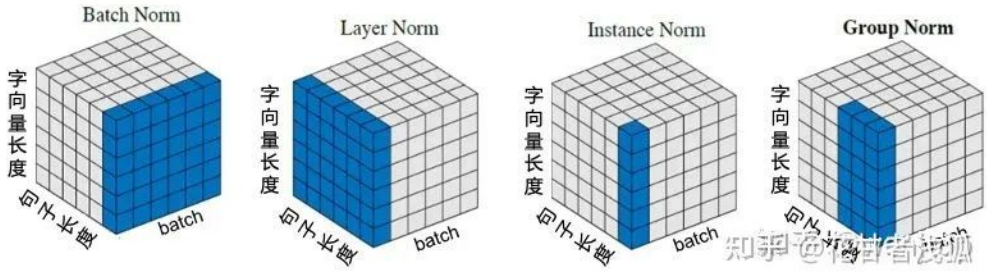
https://arxiv.org/pdf/2109.01652.pdf

4.tokenizer的细节，tokenizer的计算方式，各种tokenizer的优缺点。

tokenizer真又是一大块工作，而且也很值得优化，因为相当于模型的眼睛，它怎么去看待文字。【这两天更下这里】

5.各种norm方式的优缺点。

我们常说的norm有以下四种：LN/BN/IN/GN，分别对应layer/batch/instance/group NLP下很直观的一个图：



图源https://zhuanlan.zhihu.com/p/540695633

具体来说：

- Batch Norm：把每个Batch中，每句话的相同位置的字向量看成一组做归一化。
- Layer Norm：在每一个句子中进行归一化。
- Instance Norm：每一个字的字向量的看成一组做归一化。
- Group Norm：把每句话的每几个字的字向量看成一组做归一化。

- 其实只要仔细看上面的例子，就很容易能想到NLP中每一种norm的优缺点：

Batch Normalization（Batch Norm）：

缺点：在处理序列数据（如文本）时，Batch Norm可能不会表现得很好，因为序列数据通常长度不一，并且一次训练的Batch中的句子的长度可能会有很大的差异；此外，Batch Norm对于Batch大小也非常敏感。对于较小的Batch大小，Batch Norm可能会表现得不好，因为每个Batch的统计特性可能会有较大的波动。

Layer Normalization（Layer Norm）：

优点：Layer Norm是对每个样本进行归一化，因此它对Batch大小不敏感，这使得它在处理序列数据时表现得更好；另外，Layer Norm在处理不同长度的序列时也更为灵活。

Instance Normalization（Instance Norm）：

优点：Instance Norm是对每个样本的每个特征进行归一化，因此它可以捕捉到更多的细节信息。Instance Norm在某些任务，如风格迁移，中表现得很好，因为在这些任务中，细节信息很重要。

缺点：Instance Norm可能会过度强调细节信息，忽视了更宏观的信息。此外，Instance Norm的计算成本相比Batch Norm和Layer Norm更高。

Group Normalization（Group Norm）：

优点：Group Norm是Batch Norm和Instance Norm的折中方案，它在Batch的一个子集（即组）上进行归一化。这使得Group Norm既可以捕捉到Batch的统计特性，又可以捕捉到样本的细节信息。此外，Group Norm对Batch大小也不敏感。

缺点：Group Norm的性能取决于组的大小，需要通过实验来确定最优的组大小。此外，Group Norm的计算成本也比Batch Norm和Layer Norm更高。

大模型算法

适合入门的OverView: Current Best Practices for Training LLMs from Scratch 以及RUC AI Box的：A Survey of Large Language Models

1.在指令微调中，如何设置、选择和优化不同的超参数，以及其对模型效果的影响?

这里的超参数感觉有一些宽泛，我在这里做一下分类总览，并且仅讨论全参数微调的情况。不过说实话，数据集才是最重要的，所以建议先看问题2 「这个问题可能得搁置一阵，有点太宽泛了，没想好怎么回答」

- 数据集
- drop-last
- 模型本身
- 训练设置
- batchsize
- clip grad

- lr rate
- weight decay
- loss scale
- warm up iters
- lr decay style
- lr decay iters

2.在指令微调中，如何选择最佳的指令策略，以及其对模型效果的影响?

这里最佳指令策略我称作「指令微调方法论」，是各种做LLM微调的实验室/企业都会去探索的很重要的一个方向，主要讨论的是微调中最重要的部分——数据该如何组织甚至制造的问题，当然，由于在讨论"最佳"的指令微调方式，所以如何评测sft后的model在这里也会谈及。这里很建议大家去阅读机器之心在2023-06-29由符尧撰写的文章：《过去三个月，LLaMA系模型发展如何？指令微调的核心问题又是什么？》 对我本人有很大的启发，下面的内容一部分是这篇文章的重点内容的摘要，另一部分是我本人阅读论文和实验的经验。总的来说，你需要先明确自己需要向什么方面微调：是一个更对齐人类价值观的gossip bot？还是更具备解决问题能力的helper？是更健谈并回答内容更丰富？还是人狠话不多？是单领域？还是多领域？a.数据需要什么样的？b.该如何组织数据集来微调？首先是一些已知的要点：Flanv2提到的一些tricks【待补充】

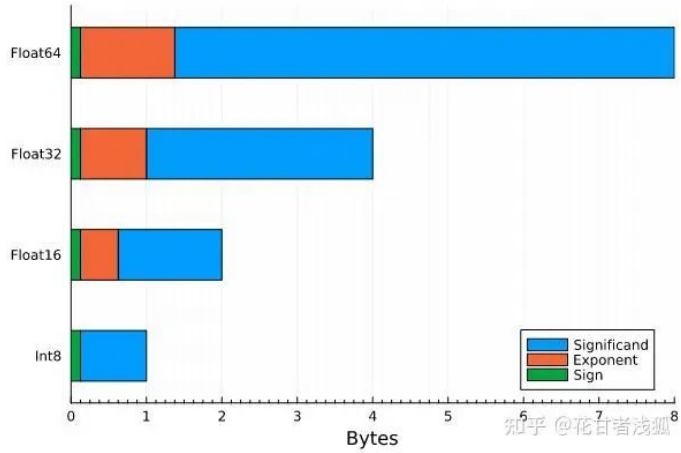
- 更多样的任务-使用不同的提示模板
- 混合few-shot和zero-shot learning
- 考虑输入反转
- 加入CoT
- 任务混合权重按照经验考虑即可

下面是数据集组织的方法论：c.该如何评估sft后的model的能力？首先，不要用gpt4去做评测，因为你要知道题目中所谓的最佳一定是

3.llama, glm，bloom等现有大模型的数据处理，训练细节，以及不足之处模型架构的优化点，包括但不限于attention, norm, embedding

4.解决显存不够的方法有哪些?

这真是一个很广泛的问题，在这里做一下总结。不过我非常建议把这篇文章看一下：Transformer Math 10 会分推理or训练来讨论，而且讨论具体方法前会有些铺垫。



不同dtype类型params的bytes分布，浮点为fp系列

首先如果是只做 **推理** 推理中显存主要被parms占用。就parms而言，对于float32的数据类型，大致估算的话就是每1B的参数需要4G的显存。float16/int8/int4对应除以2即可。具体来说：

- int8:
- fp16/bf16:
- fp32:

除了存储模型权重所需的内存外，实际前向传递过程中还有一小部分额外的开销。根据经验，这种开销小于等于20%，通常对于确定能适应你的GPU的最大模型来说是不相关的。总的来说，关于“这个模型是否适合进行推理”的一个好的启发式答案是： 一些更细节的内容在这里，写得很好：Transformer Inference Arithmetic 下面就是怎么去对params做文章了：【待完成】 a. 模型压缩 可以看下OpenBMB的BMCook，具体来说就是在保持性能的前提下把大模型压缩成小模型，比如CPM-Bee在10B的版本上基于BMCook可以压缩成1B/3B/5B的版本。方法有量化、蒸馏、剪枝、Moefication。b.Memory Offload 部分显存挪去内存，只是内存确实跑不动。c.并行 然后如果是 **微调** 的话，考虑的就很多了。微调中显存的占用除了在推理中提到的params，此外还有 forward activations, gradients, 以及optimizer的state 下面是具体的分析：**就params而言**，模型可以使用纯fp32或fp16进行训练：

- fp16/bf16:
- fp32:

除了推理中讨论的常见模型权重数据类型，训练可以引入混合精度训练，例如AMP（Automatic Mixed Precision）。这种技术旨在在保持收敛性的同时最大化GPU张量核心的吞吐量。现代深度学习训练中经常使用混合精度训练的原因是：1）fp32训练是稳定的，但内存开销较大(avtivations和gradients)，不能充分利用NVIDIA GPU的tensor cores；2）fp16训练是稳定的，但（可能）很难收敛。关于混合精度训练的更多信息，建议阅读tunib-ai团队的这篇Jupyter Notebook Viewer（韩文，加油）或者Accelerating Large Language Models with Mixed-Precision Techniques。请注意，混合精度训练需要在内存中存储模型的fp16/bf16和fp32版本，因此需要：Mixed-precision (fp16/bf16 and fp32)： 加上一份放在optimizer state中的fp32的拷贝： **就optimizer state而言**，如果开了混合精度训练占用内存计算如下（未开的话去掉第一项）：

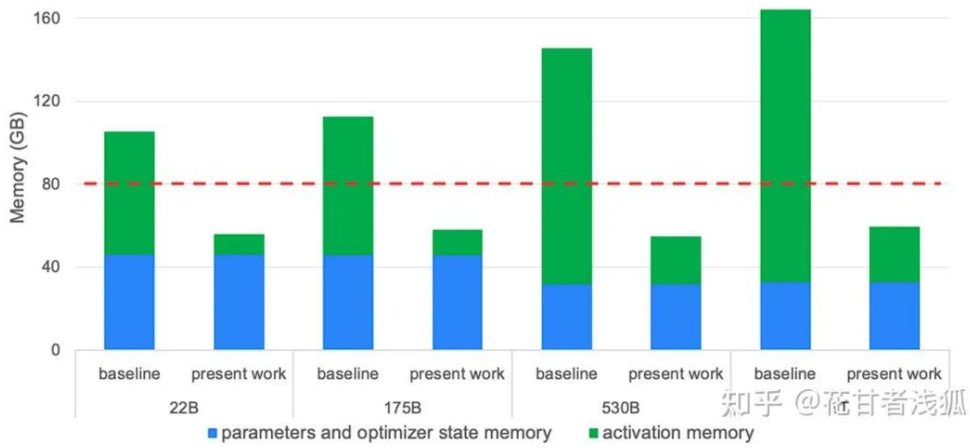
- For vanilla AdamW,
- fp32 copy of parameters:

- Momentum:
- Variance:
- For 8-bit optimizers like bitsandbytes,
- fp32 copy of parameters:
- Momentum:
- Variance:
- For SGD-like optimizers with momentum,
- fp32 copy of parameters:
- Momentum:

就gradients而言，可以存储为fp32/fp16/bf16（注意，梯度的数据类型通常与模型的数据类型相匹配。因此在fp16混合精度训练中，通常以fp16存储）。具体如下：

- In fp16,
- In fp32,

就activations而言，现代GPU在进行LLM的训练时，通常遇到的瓶颈是显存，而不是FLOPs。因此，activation recomputation/checkpointing或者叫gradient checkpointing就变得很重要。它通过牺牲额外的计算成本来减少内存成本。其工作原理是反传算梯度的时候重新计算某些层的activations，而不是将它们存储在显存中。显存减少的程度取决于选择哪些层做checkpointing。Megatron的选择性重计算方案如下图所示：



红线表示A100-80GB显卡的内存容量，“present work”表示应用selective activation recomputation后的内存需求。更多细节请参阅https://arxiv.org/abs/2205.05198

用于存储Transformer模型激活值所需内存的基本方程为：

- int4:
- int8:
- fp16:

where:

- : sequence length, in tokens
- : batch size per GPU
- : hidden dim
- : Layers层数
- : 多头数
- : the degree of tensor parallelism being used (1 if not)
- no sequence parallelism
- 假设activations是fp16

这一部分要减少显存，有以下方法：【待完成】 a.分布式训练&张量共享 考虑到是讨论节省内存，所以主要讨论FSDP。b.gradient checkpointing c.PEFT(Parameter Efficient Tuning)也就是Delta-Tuning，这一部分也可以写一篇不短的综述出来了，我在这里也大概总结一下：d.仍然是各种Offloading 比如说你的优化器可以用AdamOffload等等 e.混合精度训练 同样也是量化 f.QLoRA 强推，基本上单卡玩这个就足够了，前一阵实现了CPM-Bee 10B模型的QLoRA适配，在单卡3090上就可以十分轻松地完成微调，而且loss降得很舒服。g.optimizer的选择前面讲过了，不同的optimizer占用显存也不同，比如可以选offloading的，或者选SGD：比如邱锡鹏团队也提出了LOw-Memory Optimization（LOMO），其中在做了一些蛮有趣的设定的前提下，搞了很省的全参数微调的方法论，8块3090全参数微调了Llama65B，就很夸张。h.减少批大小 很直观的方法 i.梯度累积与微批 梯度累积是一种在训练过程中虚拟增加批大小的方法，当可用的 GPU 内存不足以容纳所需的批量大小时，这是非常有用的。并且这种方法只会在运行时产生影响，建模性能并不会受到影响。梯度累积中，每批计算的量较小，并在多次迭代中累积梯度（通常求和或求平均），而不是在每个批次之后立刻更新模型权重。一旦累积的梯度达到目标「虚拟」批大小，模型权重就会用累积的梯度更新。

5.请解释P-tuning 的工作原理，并说明它与传统的 fine-tuning方法的不同之处。

6.介绍一下Prefix-tuning的思想和应用场景，以及它如何解决一些NLP任务中的挑战

7.Lora的原理和存在的问题讲一下?

前面在4.中提到了Lora，我也曾在别的回答中提过Lora真是本世纪最美女名，因为真的是个人微调的一大福音。如果你想细致了解，可以看看这篇CW不要無聊的風格：当红炸子鸡 LoRA，是当代微调 LLMs 的正确姿势？我在这里也具体来讲讲：首先肯定是要搬上来这张图的：

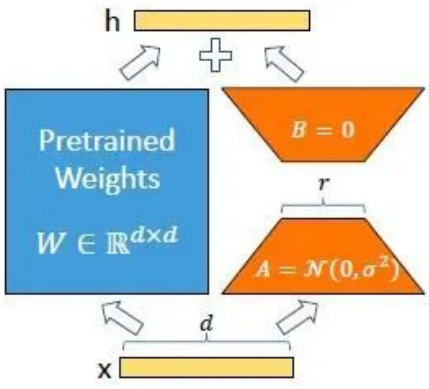


Figure 1: Our reparametrization. We only train A and B.

lora之前的PEFT方法是adapter/prefix/promp/P-tuning，但是Adapter会引入很强的推理延迟（只能串行），prefix/prompt/P-tuning很难练，而且要占用context length，变相的降低模型能力——所以，根本不改原来的model，这就引出了女主人公lora：Low-Rank Adaptation 具体来说，就是考虑到语言模型（LLM尤其如此）的参数的低秩属性（low intrinsic dimension），或者说过参数化，在做finetune的时候不做full-finetune，而是用一个降维矩阵A和一个升维矩阵B去做finetune。如果我们认为原来的模型的某个参数矩阵为 ，那么可以认为原来经过全微调的参数矩阵为 ，但考虑到前面的低秩属性，在lora中我们可以简单认为 ，其中 的秩相当于是你认为的模型实际的秩。这样的话在做推理的时候， ，根本不会引入推理延迟，因为你只需要把训好的parms 加进 的params就可以了。在Transformer中self-attention和mlp模块都有对应的params矩阵，对应加上lora即可。总的来说，这样就很好地把微调参数和原模型参数进行了解耦，因此甚至一些基于lora微调的模型只需要公布lora参数即可（大概就几M）下面是lora的问题，说实话，这点我刚看还真愣了一下，一时间没想出来有什么缺点——都这么完美了，还要什么自行车？但是，PEFT总是有局限性的吧，就算她是lora，为此调研了一下，主要如下（其实有点鸡蛋里挑骨头的意味）：a.基于低秩的微调可能并不always work，比如finetune与pretrain的gap过大的时候，比如中英差异。当然，这一点在LLM时代可能并不突出，我们认为LLM在预训练阶段已经get了所有基本的知识，finetune只是格式微调，因此可能不会有上述gap过大的情况。b.用lora也需要设置r和target module等，这部分超参的设置需要考虑

8.bf16，fp16半精度训练的优缺点

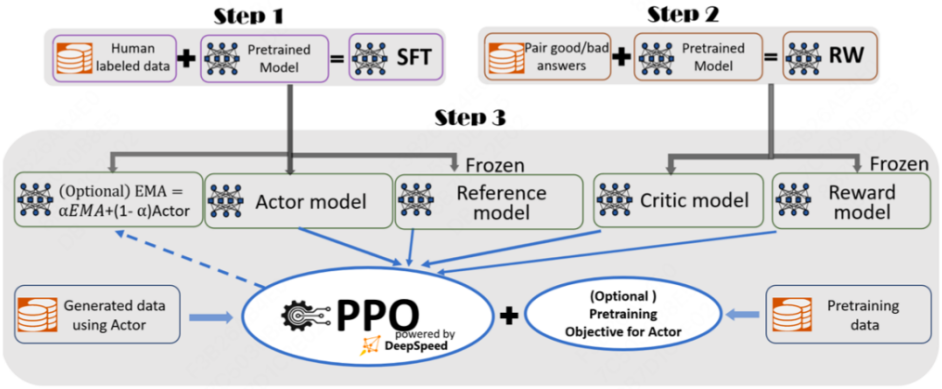
这个问题我们先来谈谈这两类半精度类型的区别，然后谈下半精度训练的优缺点，以及为什么现在用混合精度训练居多。a.首先是bf16&fp16: bf16 用8bit 表示指数，7bit 表示小数；fp16 用5bit 表示指数，10bit 表示小数。也就是说bf16 可表示的整数范围更广泛，但是精度较低；fp16 表示整数范围较小，但是精度较高。事实上，bf16就是google brain team为了深度学习而设计的数据类型，因为在深度学习中，我们更关心范围，而不是精度（这也是为什么量化很火），因为过参数本身就可以一定程度上弥补精度的损失。因此，尽管BF16的精度较低，但是它的表示范围较大，因此在深度学习中通常是更好的选择。此外，也是由于精度没有那么高，BF16在操作时需要的硬件资源也会较少。当然，bf16在一些较老的显卡上可能并不支持，不过一般的建议是能适应bf16就使用bf16。b.然后是半精度训练的优缺点及混合精度训练 半精度训练优点：很直接的优势，就是跑得快+省显存 半精度训练缺点：当然还是精度（下溢+舍入误差）的问题。当然，使用了半精度训练，一般会采用一些捆绑的技术来弥补半精度的缺点，所谓扬长避短。这些方法的合体可以叫做混合精度训练，具体如下：混合精度训练（有的时候叫fake quantization）总体的策略是，同时有低精度和高精度的权重（现在讨论的是float16和float32），前向传播用低精度来算，反向传播的gradients也用低精度来算，但是在更新参数的时候更新的是高精度的参数。然后在下一次的前向传播之前，对这个更新后的高精度参数量化为低精度参数，再开启下一次前向传播。为什么要这么做呢，其中用到下面几个技术：- FP32 权重备份：字面意思，对权重备份一份float32版本的版本，在梯度更新的时候避免float16精度不够而发生舍入误差导致的无效梯度更新，但是这样会占用额外的权重的内存，不过这些显存在一些情况下并不致命也就是了。- loss scale：由于下溢的问题，也就是训练后期，梯度会很小，float16 容易产生 underflow，所以可以对loss做scale操作，毕竟loss的scale也会作用在梯度上（链式法则），这样一个大的scale比每个梯度都scale下要划算很多。那么混合精度训练的优势是否能继承半精度训练的优势？当然可以：在显存方面，额外存了一份params的float32版本，但是forward activations都是float16的，实际上这些才是大头，所以还是有显著的显存节省的。在训练推理速度方面，实际上不会更慢，虽然不同数据类型要对齐，但是计算平台会有优化，也就是有专门的硬件加速计算，所以反而会快（仅限于大batch下，因为小batch下的计算速度瓶颈本来在IO，引入混合精度训练可能变成新瓶颈） 这样的话，训练完后推理的时候就用float16就可以了，也就是节省了推理时候前向传播的显存占用和推理时间。另外就是，layer norm的层可能会完全使用float32，因为需要计算一组值的均值和方差，而这需要进行加法和除法运算，所以float16可能会出岔子。不过针对还要保存一份float32参数这很不舒服的一点，我们题外话一下就是，你完全可以用QLoRA，也就是量化线性层+lora，反正线性层不会更新参数，我只用它进行前向传播，因此也不用存一份额外的参数；而lora的部分可以采用float1

6精度来训练，这一部分就算额外保存参数也不会占用很多显存。这样的话相当于把量化+lora的各自优点都拿出来。

9.如何增加context length 模型训练中节约显存的技巧。

10.RLHF完整训练过程是什么？RL过程中涉及到几个模型？显存占用关系和SFT有什么区别？

具体可以参考图



11.RLHF过程中RM随着训练过程得分越来越高，效果就一定好吗？有没有极端情况？

参考大模型开源社区的原子弹Llama2提到的的reward hacking以及大模型RLHF的trick

12.encoder only，decoder only，encoder-decoder 划分的具体标注是什么？典型代表模型有哪些？

参考苏神：https://kexue.fm/archives/9529

训练框架

1.Megatron以及deepspeed实现原理，各种参数以及优化策略的作用

2.模型训练以及推理中的显存占用各种混合精度训练的优劣

3.deepspeed的特点是什么？各个zero stage都有什么用？

参考https://my.oschina.net/u/5682856/blog/5539626

评测

1.除了loss之外，如何在训练过程中监控模型能力？

参考大模型RLHF的trick中提到

2.如果想全面的评测模型能力，有哪些维度以及数据集？评测指标等评测中比较重要的部分要了解。

在谈论评测大模型之前，我们需要了解评测的核心逻辑是什么？一个模型的评测需要和模型被期望获得的能力紧紧相关。具体来说，如果是评价base model，考虑到一般认为pretrain过程是模型获得99%能力的阶段，我们需要关注的是具有一般通用的核心能力，尤其是一些具有划分度的推理/泛化能力；如果是评价chat bot，那自然就是考量提升用户的体验的相关能力，比如说3H(helpful,honesty,harmless)等等；如果是领域微调的model，那自然有行业的基准去评测。因此，说在前面的是，刷榜是不必要的，因为我们更多地是考量模型的能力，只不过能力的衡量是体现在这些预设的指标上的，或者说，当你刷榜模型在一些方面的能力时，我们并不清楚但很可能模型在其他方面的能力会有缺失。换句话说，也就是 素质教育>应试教育，因此在一些指标的衡量上，你确保你的模型在这方面的能力equip，你的training strategy works

就好。我们的期望当然是所谓的评测能够证明模型的能力强，只是相比通过刷榜来证明，你可能更多关注的是为什么在某些题上面表现不好，是与训练的数据质量有关，还是你的评测本身就没有激发出模型的能力等等（你的模型能力够强，你的评测一定不会差；你的评测效果很好，未必说明你的模型就是ok的）。毕竟刷榜本身也和model的实际应用场景并不契合，是有偏的。既然清楚了核心逻辑，这对我们的评测过程是有一定指导意义的，那么回归正题:

Benchmark	Focus	Domain	Evaluation Criteria
SOCKET (Choi et al., 2023)	Sociability	Specific downstream task	Coherence, Contextual understanding
MMLU (Hendrycks et al., 2020)	Text model	General language task	Multitask accuracy
C-Eval (Huang et al., 2023b)	Chinese evaluation	General language task	52 Exams in a Chinese context
OpenLLM (HuggingFace, 2023)	Language model evaluation	General language task	Task-specific metrics, Leaderboard rankings
DynaBench (Kiela et al., 2021)	Dynamic evaluation	General language task	NLI, QA, Sentiment, Hate speech
Chatbot Arena (LMSYS, 2023)	Response quality	General language task	Crowdsourcing, Elo rating system
AlpacaEval (Li et al., 2023c)	Automated evaluation	General language task	Metrics, Robustness, Diversity
HELM (Liang et al., 2022)	Comprehensive evaluation	General language task	Task-specific metrics
API-Bank (Li et al., 2023a)	Tool-augmented	Specific downstream task	Metrics
Big-Bench (Srivastava et al., 2022)	Language model evaluation	General language task	Various metrics, Model comparisons
MultiMedQA (Singhal et al., 2022)	Medical QA	Specific downstream task	Accuracy, Medical Knowledge, Reasoning ability
ToolBench (ToolBench, 2023)	Software tools	Specific downstream task	Execution success rate
PandaLM (Wang et al., 2023g)	Automatic evaluation	General language task	Winrate judged by PandaLM
GLUE-X (Yang et al., 2022)	Natural language understanding	General language task	OOD robustness
KoLA (Yu et al., 2023)	Knowledge-oriented evaluation	General language task	Self-contrast metrics
AGIEval (Zhong et al., 2023)	Human-centered foundational models	General language task	General
PromptBench (Zhu et al., 2023)	Adversarial prompt resilience	General language task	Adversarial robustness
MT-Bench (Zheng et al., 2023)	Multi-turn conversation	General language task	[illegible]
M3Exam (Zhang et al., 2023c)	Multilingual, Multimodal, Multilevel	Specific downstream task	Task-specific metrics

a.对于base model的评测如下

- 找有区分度，能体现核心能力的数据集，下面可以参考：
- 英文知识 — MMLU
- 中文知识 — C-Eval
- 推理 — GSM8k / BBH
- 代码 — HumanEval / MBPP
- 数学 — MATH
- 用体现这些能力的数据集，来做automatic eval：
- 可以参考chain-of-thought-hub，但是评测中的难点就在于不同prompt的设计对于不同模型的能力挖掘是不一样的，所以可能相对公平的方法就是用实际业务可能的prompt来eval（除非是想公开自己研发的llm的评测board，那自然是需要做一些prompt engineering的方法来努力提点）

b.对于sft后的chat model评测如下

3.如何评测生成，改写等开放性任务?

还是根据指导思想，开放性任务的写作能力可以被看作核心能力之一，但由于这类任务本身就很主观，我们不太方便用Rouge或者BLEU这样的评价指标，因为它本身就不能体现模型的核心能力，而且与人类基准就是不对齐的（偏离实际需求） 所以，从更贴近实际需求的角度来说，Elo的方式还是最合理的；或者如果你的模型的核心业务就是生成/改写/总结，那你本身就应该有一套业务逻辑的评价指标来评测你的模型——以你的业务需求为导向。

4.zeroshot和Fewshot具体做法的区别?

数据 把数据放在最后，数据的重要性实际上远超前面的所有（或许是因为模型开源，数据闭源）。有的时候怎么训都提不了点的时候，建议回头看下你的数据，高质量的数据就是可以随便训一个不差的模型——事实就是这样。

LLM数据处理概述 - 放在最前面，如果你刚入门，可以看看

综述blog：Processing Data for LLM，下面是这个blog整理后的内容。这里更多讨论的是对预训练中如何处理整个互联网语料的讨论，一些行业头部公司自然会有很高的行业数据壁垒，但是从整个互联网的语料得到供模型学习的“高质量”的数据就是一项很广泛且重要的议题了。LLMs之所以强大，有很大一部分源自其在超大规模数据集上的训练，使得它们各方面能力超越小模型，这就是Scaling的魔力。通常来说，数据量越大，模型效果通常越佳。一些数据集如C4、The Pile、The Bigsicence Roots Corpus和OpenWebText，甚至于近期的The Refined Web Dataset、RedPajama或者说The Stack，都是通过网络爬取并清洗大量文本以提升训练规模。然而，超大规模数据集的手动审查和筛选的价格高昂且难以保证质量，导致训练出的模型可能带有训练数据的偏见，也就是效果不够好。同时，对超大规模数据集进行定量和定性研究本就很难。再加上训练中可能还会涉及到Curriculum Learning，以及一些研究表明Loss spike也和训练时batch内数据分布有很大关系，这些都要求我们对互联网数据本身有很清楚的认识，以及一定程度上指导训练的policy。为处理这些问题，第一步就是需确保用于训练LLMs的数据是高质量的。方法包括但不限于：

- 处理无效数据

一些无效数据，如意义空泛或模板化的文本（例如HTML代码、Lorem ipsum等）。甚至于在多语言语料库的构建过程中，从网站提取文本用于语言建模也极具挑战性。但这是我们必然要做到的，因为NTP(Next Token Prediction)的方式注定训练模型使用的数据本身就是真实语言世界很好的映射。数据清洗工具，如justext、trafilatura等，能有效剔除HTML模板文本，同时在减少噪音（提高精度）与保留所有有效部分（提高召回率）之间取得平衡。另外一点是，处理网页语料库中无效数据的有效方法之一是利用元数据进行筛选。例如，OpenAI在构建GPT-2用的WebText语料库时，抓取了reddit上点赞数至少为3的所有外部链接，这种启发式方法有助于减少数据集中的噪音，同时确保数据质量。

- Document Length Considerations

一方面，考虑到NTP，从语料库中移除非常短的文档（包含少于约100个标记的文本）可以帮助通过创建连续的文本来建模文本中的依赖关系，从而去除噪音。另一方面，由于大多数语言模型如今都基于Transformer架构，对非常大的文档进行预处理并将其分成所需长度的连续片段是很有用的。

- Machine Generated Text

训练语言模型的目标之一是捕捉人类语言的分布。然而，网络爬取的数据集包含大量机器生成的文本，例如现有语言模型生成的文本、OCR文本和机器翻译文本。例如，来自http://patents.google.com的数据构成了C4语料库的大部分。该语料库使用机器翻译将来自世界各地专利机构的专利翻译成英语。此外，网络语料库中的数据还包含来自扫描书籍和文档的OCR生成文本。OCR系统并不完美，因此生成的文本与自然英语的分布不同（通常OCR系统会在拼写错误和完全遗漏单词等方面产生可预测的错误）——这点很重要，也很难搞，pdf扫描文档怎么去做还真挺头疼的。虽然很难识别机器生成的文本，但有一些工具，如ctrl-detector，可用于识别和检测机器生成的文本。在为语言建模预处理语料库时，重要的是对语料库中机器生成文本的存在进行表征和记录。

- 去重

从互联网上爬取原始文本创建的数据集往往会导致相同的序列被多次重复出现。例如，在论文《Deduplicating Training Data Makes Language Models Better》中，作者发现在C4数据集中，一个50个单词的序列被重复出现了60000次。事实上，在去重的数据集上训练模型速度更快，并且不太容易导致记忆效应——很不好。最近，研究人员还表明，在重复数据上训练的语言模型容易受到隐私攻击，其中对手从训练模型中生成序列并检测哪些序列来自训练集的记忆。在论文《Deduplicating Training Data Mitigates Privacy Risks in Language Models》中，作者展示了语言模型重新生成训练序列的速率与序列在训练集中的出现次数超线性相关。例如，一个在训练数据中出现10次的序列平均会比一个只出现一次的序列生成1000倍多。去重可以在不同粒度级别上执行。从精确匹配去重到模糊去重工具（例如deduplicate-text-datase

ts和datasketch），可以帮助减少和去除正在处理的语料库中的冗余文本。正如许多研究人员所指出的，需要理解去重过程需要大量计算资源（CPU和RAM），因为网页爬取数据集的大小，因此建议在分布式环境中运行此类计算。

- 清洗污染数据

这部分还挺保受争议的，可能还没有很细致的标准，不少公司也都挺功利的，就不好说。在NLP领域，我们常说的数据清洗，主要指的是训练数据和测试数据的区分和处理。在大型语言模型的情况下，由于训练和测试数据集都源于互联网，确保二者不发生交叉，这个过程可能颇具挑战。大型语言模型的评估通常会用到基准数据，如问答对，如果这些基准数据在训练数据中出现，可能会导致基准性能的高估。因此，需要进行去污染操作，也就是从训练数据中去除和基准数据集有重叠的部分，保证训练数据集的完整性。OpenAI的研究人员在创建WebText数据集时，就通过剔除所有维基百科内容来实现数据去污染，因为维基百科数据在他们的基准数据集中被广泛使用。另一个案例是EleutherAI的研究人员，他们开发了名为lm-eval harness的软件包，用以实现对基准数据集的去污染。在具体操作中，我们需要关注两类数据污染：

1. 输入与输出污染：这种情况下，预训练语料库中存在与下游任务标签相同的数据。对于语言建模等任务，任务标签就是目标文本。如果目标文本在预训练语料库中出现，模型可能会倾向于复制文本，而非真正解决任务。

2. 输入污染：这指的是评估样本中并未包含标签的情况，这也可能导致下游任务的性能高估。在进行零样本和少样本评估时，如果预训练数据集中存在与热门基准任务重叠的数据，我们必须重视数据去污染。

- 毒性和偏见控制

尽管网络语料库具有丰富的多样性，但其中也常常弥漫着毒性和偏见内容。如，《RealToxicityPrompts》一文中作者使用PerspectiveAPI指出，OpenWebText与WebText的内容中分别有2.1%与4.3%存在毒性分数超过50%。因此，在训练语言模型时，必须警觉并借助PerspectiveAPI等工具筛选掉预训练数据集中的毒性内容，以防止模型表现出偏见或在下游应用中产生有害内容。一种解决策略是过滤掉"bad words"名单中的文本，比如C4的作者们就采用了这种策略。另一个例子是，PILE数据集的研究者利用spamscanner来对有害内容进行分类。然而，执行此类过滤步骤必须极为谨慎，并需考虑到下游应用，以免过滤器保留下更可能坚持霸权观点的声音。在利用数据进行预训练语言模型之前，对贬损内容和性别/宗教偏见进行深度分析是必要的。

- 个人身份信息控制

在收集大型数据集时，理解与数据集实例相关的法律问题至关重要，尤其是在处理个人身份信息（PII）时，如真实姓名、组织名称、医疗记录、社会安全号码等。根据不同的应用，对这些信息进行遮蔽或删除在预训练语言模型之前是必要的。像presidio和pii-codex这样的工具提供了检测、分析和处理文本数据中个人身份信息的流程，这些工具能帮助确保数据集中的个人信息得到合理处理，以遵守相关隐私法规并保护用户隐私。

1.bloom, llama，glm等开源模型的数据来源，配比，以及不足之处

在谈论各个base model之前，我们得先清楚pretrain的数据来源的分类，大致如下

- 自然语言
- 书籍/文章
- Book
- 网页

- Arxiv
- Wikipedia
- C
- OSCAR
- Common Crawl
- StackExchange
- 其他数据集
- 编程语言
- 网页
- Github（Google BigQuery）
- StackOverflow
- 其他数据集

注意这里所谓的数据集就是有人替你爬好了而已。其实语料的来源并不那么重要，因为总的来说各个模型用的源语料基本上都是这些（多样性较易达成） 重点在于怎么去清洗，因此当前这一部分大致了解即可。a.bloom(**B**igScience**L**arge**O**pen-science**O**pen-access**M**ul-tilingual Language Model) bloom更多信息指南：白强伟：【自然语言处理】【大模型】BLOOM：一个176B参数且可开放获取的多语言模型 在这里仅谈数据部分。

- ROOTS语料https://arxiv.org/pdf/2303.03915.pdf：
- 341B tokens+25B tokens or 1.61TB的文本
- 是一个由498个Hugging Face数据集组成的语料
- 46种自然语言和13种编程语言

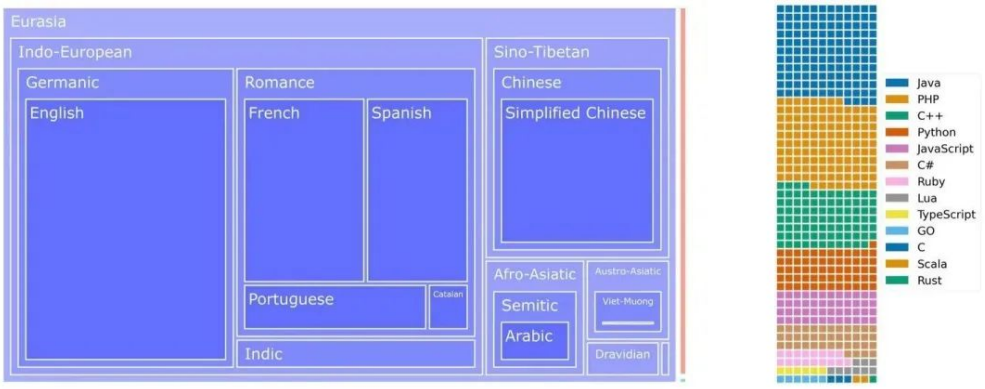


Figure 1: Overview of ROOTS. Left: A treemap of natural language representation in number of bytes by language family. The bulk of the graph is overwhelmed by the 1321.89 GB allotted to Eurasia. The orange rectangle corresponds to the 18GB of Indonesian, the sole representative of the Papunesia macroarea, and the green rectangle to the 0.4GB of the Africa linguistic macroarea. Right: A waffle plot of the distribution of programming languages by number of files. Each square corresponds approximately to 30,000 files.

b.llama v1 & v2 c.glm d.baichuan

2.cot以及ic能力是如何涌现的？与预训练数据有何关系？

3.数据处理的重要步骤，如何保证预训练以及sft时候的数据多样性，数据质量，数据数量等,包括但不限于去重，质量筛选，敏感及有害信息过滤，各种来源数据配比对于模型能力的影响。

4.多轮对话数据如何组织？

大模型微调样本构造的trick

极市平台

YOLOv5帮助母猪产仔？南京农业大学研发母猪产仔检测模型并部署到Jetson Nano开发板

极市平台

ICCV 2023｜Pixel-based MIM: 简单高效的多级特征融合自监督方法

极市平台