

LLM实践--Tokenizer训练



真中合欢

真中合欢爱好者

[关注他](#)

刷月亮的滴仙人、y bq 等 84 人赞同了该文章

[上一篇：LLM实践--数据去重：Simhash&Minhash 原理分析&代码实现](#)[下一篇：LLM实践--训练准备：训练硬件估算、软件环境搭建、评估集合](#)

引子

经过了数据收集、筛选、去重，马上就可以开始训练实验了。但是在实验之前，我们还需要先获取一个语言模型的基石：分词器（Tokenizer）。Tokenizer 的作用是对一条文本数据进行切分、词表映射，得到这条文本的token序列。

用开源 Tokenizer 还是自己训练

Tokenizer可以自己训练，也可以从目前开源的模型中扒一个来用，用开源Tokenizer有几个点需要着重关注：

- 压缩率：压缩率决定了文本向量化后的长度，压缩率越高，向量后数据越短，训练和推理效率越高，但是对训练数据的数量要求也越大，主流的tokenizer对汉字的压缩率都在1.5-1.6之间，也就是1.5-1.6个汉字划分为一个token。
- token覆盖率：token覆盖率不用纠结细节，只需要关注是否有你的目标语种的token，比如llama的tokenizer中文就很少，相应地中文上压缩率就比较低，token向字节流的退化率比较高，也一定程度的反应了中文训练数据不多。
- 预留token数量：预留token也叫特殊token，一般写作reserved_token、unused_token，paded_token，都是一个意思。这些token是指不会出现在自然语料中，仅保留为后续post train阶段的一些特殊用途使用。比如任务隔离、角色隔离、function call的特殊指令、agent特殊指令等等。预留token最好足够，100-1000为佳。如果下载的tokenizer预留token不够，可以手动添加。
- 词表大小：目前开源的tokenizer词表大小一般在8万或15万左右。词表越大，压缩率一般也越高，同时模型的embedding层和logits层也会更大，对显存资源敏感的需要注意一下。

有开源tokenizer，训练自己的tokenizer意义何在？用自己的数据训练的 tokenizer，在同词表大小的情况下，会比开源tokenizer有更高的压缩率（也不会高太多），可以一定程度降低训练和推理成本。另外更主观一点的原因是，训tokenizer是个很基础的工作，训不训一定程度上反映了团队的技术栈是否全面。还有一点需要注意的是，不同压缩率的tokenizer训练的模型loss可能有差别，但是性能不会有太大差别。

经典的分词器有WordPiece、subword-nmt、Unigram等等，但是这些并不是本文的重点，本文主要说目前最流行的两种tokenizer：BPE和BBPE，去年很多家用的是BPE，今年都转到BBPE上。

BPE (Byte Pair Encoding)

BPE是目前大模型主流的两种分词法之一，训练过程总结成一句话就是：迭代合并当前最高频的token对，直到到达预设词表大小上限。举个具体的例子来说，假设我要训练一个词表大小是12的tokenizer，训练语料就是下面这句话：

“海水潮潮潮潮潮落,浮云长长长长长长消”

1.这句话首先会按字拆分成：海 水 潮 潮 潮 潮 潮 落 , 浮 云 长 长 长 长 长 长 消。

我没数错的话算上逗号应该是总共有9个不重复的字。这些字会被当作初始token，加入词表。现在我们离目标词表大小还差12-9=3个token，下面开始迭代合并。

这里有两点注意，首先是“当前”，也就是说每次迭代加入新token后，下一次统计两两组合要算上这个新token。其次是“所有token两两组合”，也就是既要统计A token与B token的组合，也要统计A token与自身的组合。比如上面这个句子，我们要统计“海水”、“水潮”、“潮潮”、“潮落”....这些所有两两组合出现的次数。

3. 取两两组合中出现次数最高的那一个，作为新token加入词表，同时记录下这个token的合成路径。

比如上面“潮潮”和“长长”是出现次数最高的组合，都出现了5次，那么我们取更早出现的“潮潮”作为本次要加入词表的新token，同时记录下“潮”+“潮”=“潮潮”。现在词表大小为10。

4. 如果词表没有达到设定的上限12，那么就迭代执行2-3步。

再一次统计两两组合出现次数，这一次最多的就是刚才并列第一的“长长”。当然也不要忘记上一步刚加入的“潮潮”这个token，它可以和前面的token组成“水潮潮”，也可以和后面的token组成“潮潮潮”，不过次数都不如“长长”。所以这一次加入词表的是“长长”。

再迭代一次，再统计组合，此时次数最多的是“潮潮”+“潮”组成的“潮潮潮”，以及“长长”+“长”组成的“长长长”，分别出现4次。按照之前的原则，取次数最多且更靠前的“潮潮潮”加入词表，此时词表大小为12，训练停止，我们已经得到了大小为12，在“海水潮潮潮潮潮潮落，浮云长长长长长长消”上训练的分词器。

BPE的训练过程还是很简单很好理解的，但是还是有一些需要注意的地方。上面的解释中有两个我刻意严谨表达的点，一个是「取次数最多且更靠前的“潮潮潮”加入词表」，另一个是「“潮潮”+“潮”组成的“潮潮潮”，以及“长长”+“长”组成的“长长长”」。前者想说明token加入词表的顺序是有先后的，是有优先级的，后者说明token的合成路径方式需要严格遵循，改变词表可能导致错误的合成路径。

特意强调这个是因为我看到一些改词表的开源工作其实是有问题的。举个我看到的实际的例子，有一个地名“乌鲁木齐”，假设词表中包含“乌鲁”和“鲁木”两个token。首先说可不可能出现这两token？完全可能，如果我的训练语料是下面这样的就会出现这两个token：

乌鲁
乌鲁
鲁木
鲁木
齐

这个语料训出来的tokenizer词表大概率是这样的：「乌」「鲁」「木」「齐」「乌鲁」「鲁木」

如果词表里乌鲁这个token在前，分词结果就是「乌鲁」「木」「齐」，如果是鲁木这个token在前，分词结果就是「乌」「鲁木」「齐」。分词结果是不一样的。如果拿到一个训练过的模型，改一改词表顺序，肯能会导致分词结果的不一致，模型可能完全没有见过这样的token，导致一些无法理解的怪异生成结果。

再说合成路径，如果我的词表是「乌」「鲁」「木」「齐」「乌鲁」「鲁木」，后来扩增了词表，增加了「乌鲁」+「木」=「乌鲁木」，和「乌鲁木」+「齐」=「乌鲁木齐」两个token，「乌鲁木齐」这个token是合成不出来的，因为「木齐」在「乌鲁木」之前，所以优先合成「木齐」，而不是「乌鲁木」，那么没有「乌鲁木」，自然无法合成「乌鲁木齐」。如果要进行词表删减、扩增，或者两个tokenizer进行合并，尤其要注意这个问题。

BBPE (Byte-Level Byte Pair Encoding)

BBPE和BPE大体上是一样的，区别在于BPE把文本看作字符集合，第一步是按照字符切分获得初始token，BBPE把文本看作是二进制编码，按照8bit切分获得原始token。比如还是上面那句话，会先转成utf8编码：

“海水潮潮潮潮潮潮落,浮云长长长长长长消” =>

然后取每一个2位16进制数作为初始token，也就是「\xe6」「\xb5」「\xb7」...这些。剩下的统计两两组合、合成路径都和BPE是一样的，不过都是在二进制层面去合并。我们知道utf8是变长编码，ascii字符在utf8中的编码长度是1，也就是刚好一个2位16进制数。比如我上面句子里的逗号对应的utf8编码是“\x2c”。所以ascii字符一定会作为一个基础字符加入词表，而且也不会被拆分，所以英文单词、数字这种ascii字符组成的词，一定是整数个token表示的。但是汉字的编码长度大部分是3，比如“海”的编码是“\xe6\xb5\xb7”，这就导致汉字在bbpe的词表中并不一定是1个、2个字这种整数个token组成。可能是3/2个token表示一个汉字。

BBPE与BPE的对比

从流行度来说，BPE是去年、前年大家普遍使用的方法，而BBPE是去年底到今年的模型主要使用的方法。GPT2使用的tokenizer也是BBPE。

从编码的角度，有一些文章说BBPE对比BPE的优势是不存在OOV问题，字符只要能转utf8，就一定能被BBPE表示，但是实际执行起来并不是，因为大部分BPE的库也支持bytes退化，遇到超出词表范围的字符，也会退化到二进制表示。这么看来BBPE剩下的优点就是多语种下、token切分更均匀。毕竟一个中文能占3/2个token了。

从实现的角度，BPE的tokenizer用sentencepiece库的居多，BBPE用huggingface的tokenizers库的居多，但是sentencepiece库产出的tokenizer.model本质是一个protobuf文件，可以用protobuf库读出这个tokenizer原始的训练参数，甚至带着训练语料的磁盘路径，不太安全。

训练参数

除了最基本的词表大小外，实际训练的tokenizer还有一些可配置关键参数。我比较喜欢读google的文档，就拿sentencepiece的训练参数来介绍了，两个库的可配置参数其实差不多，可以类比。

--vocab_size

tokenizer预设的词表大小。最终模型训练的时候，我们一般会确保embedding层的shape可以被128整除，这个一方面是为了量化考虑，一方面是为了序列并行考虑。所以可以在这一步直接设置一个能被128整除的词表大小，也可以这里不设置，等tokenizer训练完了加一些特殊token，补到128的倍数。另外词表大小也决定了压缩率。

--character_coverage

字符覆盖率。这个表示在最一开始，初始单字token要覆盖训练语料中全部token的百分之多少。如果是1，表示所有token只要出现就加入初始词表，这会导致词表的单字token过多。一般可以设置0.9998、0.9999，表示初始单字token要覆盖训练集字符的99.99%

--max_sentencepiece_length

单一token最多多少个字符组成，一般设为2、4、6、8或以上就比较大了，不太推荐，可能会出现低频超长token。

--split_digits

是否做数字的一致性切分，说白了就是数字和其他token是否可以组合成新token，还是数字必须一个字符一个token不做合并。早期一些工作认为开了对数字任务有好处，但是从我的实验上来看，是可以打开，但不是必须。数字在自然界是均匀分布的，也就是说1、2、3还是111、222、132、863数量其实是差不多的，所以就算不开一致性切分，这些token也都应该在词表里。只要模型训练充分，模型是能分辨111和1 1 1是一个东西的。我一直认为不应该对语言模型的泛化抱有太乐观的态度，它就只能说出现过的话。所以我不指望在欠训练的模型上，依赖一致性切分提高数学能力。当然这是理想情况，如果数据集准备的不充分，不能保证所有这些数字都出现在词表里，可以手动把0-9999添加到词表里，这样既保持一致性，又能提高压缩率。说个题外话，前段时间知乎上流行讨论为什么9.11>9.8，我认为就是token欠训练，又刚好没有泛化过来。为什么没有泛化过来呢？因为真的有9.11大于9.8的时候。python3.11就大于python3.8

--allow_whitespace_only_pieces

-user_defined_symbols

这个主要就是为了配置之前说的特殊token，可以预留几百比如 <reserved_0> <reserved_1> ...，如果要手动添加数字的一致切分和连续空格token，也可以在这里加

--byte_fallback

之前说的二进制退化，让BPE当半个BBPE用

--remove_extra_whitespace

是否删除多余空格，这个一般改为False，不要让tokenizer随便动我们的空格。

--unk_id、--bos_id、--eos_id、--pad_id、--unk_piece、--bos_piece、--eos_piece、--pad_piece

指定控制字符和ID，这里面现在我们一般只用pad和eos。在训练的时候文档或者一个turn的末尾增加一个eos token。需要做padding补齐的时候拼pad token，也可以直接用eos token当补齐token。不过建议四个都设置上，也算是致敬一下之前的NLPer

番外篇1:tokenizer与loss

不同tokenizer的压缩率，每个token的信息量是不同的，这就导致不同tokenizer在同一份数据下训练出来的模型loss不一样。假设不考虑训练tokenizer的语料质量差异，那么tokenizer的压缩率越高，同一个文本分词后产生的token越少，整句话的平均loss就越高。相反压缩率越低，一句话的loss通常也越低。但是实际推理起来，两种tokenizer训出来的模型效果差异不大，所以更多的还是从效率和成本的角度考虑压缩率的取舍吧。

再引申一下，从这一点上不禁让我们升起一个疑惑，loss能代表模型性能吗？答案是不行。其实同loss不同tokenizer、同loss不同参数量的模型性能都不一样。这个具体可以等写到scaling law拟合、模型性能预测或者continue pretraining的时候再讨论。

番外篇2：看一看真实的tokenizer

再拿qwen2的tokenizer举例，qwen2的tokenizer是BBPE，重要文件有这么几个：

tokenizer

qwen

merges.txt

tokenizer_config.json

vocab.json

draft.py

知乎 @真中合欢

merges.txt就是保存合成路径的文件，里面看上去会有一些像乱码的东西：

G G
 Ġ Ġ Ġ Ġ
 i n
 Ġ t
 Ġ Ġ Ġ Ġ Ġ Ġ Ġ Ġ
 e r
 Ġ Ġ Ġ
 o n
 Ġ a
 r e

知乎 @真中台欢

这些其实是转义后的控制字符。前面也说了BBPE的初始token是2位16进制数。如果直接存这些二进制字符，可能被翻译成ascii码中的控制字符，比如换行制表符之类的，所以这里坐了转义。

vocab.json是每次token的映射id，tokenizer_config.json里面可以配置一下控制字符。tokenizer训练完之后如果想加特殊字符，也可以在这里配置。

番外篇3:词表增减问题

词表的修改最好发生在模型训练之前，包括tokenizer合并、添加特殊token、自定义token等等，这其中还尤其要注意增加词表。语言模型训练的时候，计算logits时是hidden_states和所有token的logits_weight的乘积，softmax也是所有token的归一，删除词表相当于减小归一化的分母，会导致最后sample时的概率发生变化。添加token则跟要小心，如果添加的token未经训练，可能导致归一化后乱掉。如果添加的token初始化的不好，比如正常token是 $1e-2$ ，新增加的token是 $1e-23$ 量级，rms norm会回传给embedding层一个大概在 $1e21$ 量级的梯度。这个时候如果你开了梯度裁切，在求梯度的模长的时候 $1e21$ 求平方直接变成inf，再归一化其他token，所有token梯度都会变成0，这样你不太看得出来报错，但是embedding实际一点没训。

再有就是前文强调过的，注意处理合成路径和token顺序的冲突问题。



理性发言，友善互动

5 条评论

默认 最新



老北京鸡肉卷

我们最近的一个工作(arxiv.org/pdf/2407.1362...)讨论了词表的设置如何影响语言模型的性能。我们通过对scaling law公式进行修改和引入词表大小无关的评价指标系统研究了这个问题。关键结论: (1) 大模型比想象中要更“渴望”大的词表 (2)词表部分的参数要比非词表部分的参数增长得慢, 增长系数比是0.83 (3)现有大多数大模型的词表大小远低于理论最优值, 仍有较大的提升空间

10-16 · 新加坡

回复 喜欢



老北京鸡肉卷

另一个角度看, 大词表很大程度上代表了一个“更难的预训练任务”, 更适合大参数量的模型来做。所以0.5B可能100K词表就够用了, 70B就得需要500K词表。

10-16 · 新加坡

回复 喜欢



LindgeWAI

目前哪种tokenization是主流??

10-15 · 天津

回复 喜欢



真中合欢 作者

BBPE

10-15 · 北京

回复 喜欢



罗小黑

感谢博主梳理, 讲得非常清楚!

10-09 · 辽宁

回复 喜欢

推荐阅读

[sentencepiece]Tokenizer的原理与实现

由来 无论在使用LLM大模型时, 还是使用bert等传统的模型, 对字符串进行编码都是必要的, 只有经过编码后的字符串才能参与到后面的模型计算。 以下是在transformers库下的编码方式, 无论是什...

平平无奇小熊猫

Zhenghao Lin^{1,2*} Zhilin Guo^{1,2*} Boyan Gong^{1,2*} Xiao Li^{1,2*} Xiang Shi
Renshen Xie¹ Chen Liu¹ Yujie Yang¹ Jian Zhai¹ Xian Zhang¹ Microsoft
¹Xiamen University ²Tongshan University ^{*}Microsoft
<https://arxiv.org/abs/2407.1362>

Abstract

Previous language model pre-training methods have uniformly applied a word token prediction loss to all training tokens. Challenging this norm, we posit that “for all tokens in a corpus are equally important for language model training”. Our initial analysis delves into token-level training dynamics of language models, revealing distinct loss patterns for different tokens. Leveraging these insights, we introduce a new language model called Rho-1. Unlike traditional LMs that learn to predict every word token in a corpus, Rho-1 employs Selective Language Modeling (SLM), which selectively trains on useful tokens that aligned with the desired distribution. This approach involves scoring pre-training tokens using a reference model, and then retains the top-k tokens with a threshold loss on tokens with higher scores.

【LLM】RHO-1: 不是所有的token都是你所需要的

无影寺