​

## 使用transform训练英德翻译模型

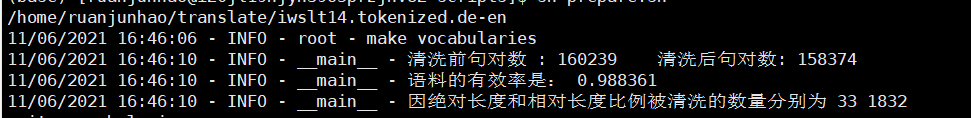
​

### 数据分析阶段：

在参考网址[[1]](#footnote-0)上下载IWSLT14 de-en数据集，检查数据集预处理情况，已经划分好了训练集，验证集和测试集，分词和小写也做了。BPE算法也做了，但是bpe可以分开两个语种学习code，也可以联合学习，前者的优点是在文本和词汇量方面更紧凑，而后者提高了源切分和目标切分之间的一致性，实践上后者的效果往往更好[2]，通过打开code检查发现语料库的提供者也是这么做的，这一步也不需要处理了。剩下如目标语言和源语言句长的ratio限制或者句子绝对词数的限制，不太容易肉眼检查，就留到之后的合并训练集部分过滤一下。

### 预处理阶段：

以前在fairseq里面看到过对这个数据集的预处理scripts[[2]](#footnote-1)，借鉴其中的参数设置，将长短比阈值也就是ratio设置为1.5，这意味着源语言或者目标语言的长度不会超过对方的1.5倍；同时将句子绝对长度设置在1-175之间。清洗结果如下:



这个过程会清洗掉1.2%的极端数据。

因为数据发生了变化，所以需要重新生成vocab词频文件，然后把双语语料合并为一个文件。这个过程在 prepare.py和scripts/prepare.sh中完成。

### 数据装载类设计：

需要一个Vocabs类，这个类可以支持token和index的相互转换，也就是词的独热码表示。同时可以在导入词频表的时候统计一个有效率（根据最小词频阈值洗掉一点词），主要需要维护的数据结构就是list类型的idx2token，dict类型的token2index和一个float类型的词有效率coverage。然后再写一个批量转换index和token的方法，转换token的还需要检查词是否存在，不存在返回UNK的下标。

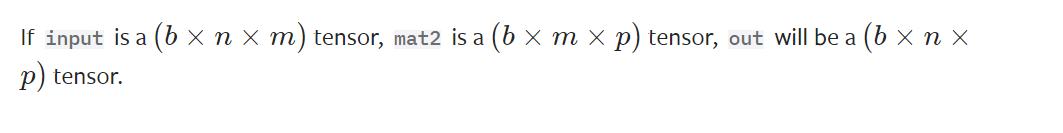
接着需要一个DataLoader类，需要一个bool类型的for\_train用来标记是不是训练用的，来决定是不是要进行shuffle操作。同时为了后面每个batch中padding的数量少一些，我们会把长度相近的句子放在一个batch里面，所以需要两个list来统计源语言和目标语言的句子长度。剩下就是保存一些后面要用到的参数，batch\_size和上面的Vocab类和数据本身的两个二维list。这个类主要的方法就是在\_\_iter\_\_里面为每源语言加上[EOS]，目标语言里加上[BOS][EOS]，然后根据上面shuffle的index取出对应的句子，然后加上[PAD]转成tensor。

### 模型实现：

transformer并没有限定独热编码怎么变成一个连续的分布向量，我们直接用lookup table的方式，也就是维护一个可学习的embedding参数矩阵，让这个矩阵和独热向量相乘，因为独热向量只有一维非0，就相当于按下标直接从embedding参数矩阵里取出一行。

标准的transformer论文中实现了两种位置编码，分别是LearnedPositionalEmbedding和SinusoidalPositionalEmbedding。正弦位置编码因为不依赖于特定的数据输入，只和位置和维度有关，所以我们可以预先生成一个第一维长度足够长（只要比我们设置的max\_seq\_len大一些就好，比如512），第二维是embedding\_size的二维位置编码矩阵，这样就可以和任意长度的句子相加了。

接下来是多头注意力的实现，在整个transformer中，多头注意力有三种存在形式。一个是encoder的多头自注意力，一个是decoder的多头自注意力，还有一个是decoder的交叉注意力。这三个注意力在输入和mask矩阵上有一点区别，我们可以根据多头自注意力的qkv相等，和交叉注意力的kv相等划分开这三者。这样做的好处在于不需要额外多写一个层，提高了一点点模型的优雅程度，这个思路是以前看fairseq源代码学到的。在这一层的前向传播中，得到了映射好的QKV，接下来是做个缩放。这里如果是高版本如1.9的pytorch，已经可以支持in-place的写法了（q\*=self.scaling），但如果是稍早一点的版本会报错。然后开始做点乘注意力，但是torch.bmm要求前一个张量的第二维度和后一个的第一维度相同，



所以我们得把qkv从seqLength\_first转成batchSize\_first，然后对k的第一维度和第二维度做转置，与q相乘。接下来是mask的区别，如果是encoder这边，只需要根据batch里的pad标记形成一个矩阵就好。然后对权重矩阵pad的位置做一个转换，替换成负无穷，在K也就是src\_len维度上做softmax。接着做dropout，这个可以在权重矩阵上做，也可以在和V相乘后的结果上做。然后再做个线性变换送走。

接下来是残差连接，但是这个可以在顶层的模块forward里组织，所以接下来先完成前馈神经网络。ffn就是两层全链接，第一层有个relu的激活函数，我用gelu[3]换了relu，高斯误差线性单元的工作在比较新的transformer类工作中得到了应用，像Bert和GPT-2，不过比较有意思的是这篇工作被ICLR2017拒了，但是现在已经有些影响力了。

在多头注意力和前馈神经网络中所有的线性变换参数W都用正态分布初始化，bias都初始化为0.在优化器的选择上除了被广泛使用的Adam[4]，我试了一下AdamW[5]。

最后额外写个输出层接在decoder的最后面，做个log\_softmax，加了标签平滑的nll loss算损失，然后直接取出softmax的最大值和reference的做torch.eq算acc。然后推断阶段写个beam类完成，再调一下sacrebleu算一下bleu分数，然后就完成了。

最后的实验结果是在验证集上bleu score 31.90,测试集上30.98.训练日志和模型文件在8.130.48.129服务器上可以获取，存放在/home/ruanjunhao/translate/mt.ckpts下。

比较遗憾的是，大部分的工作以transformer作为基线，在这个数据集上都有34+的bleu score，我用fairseq试了一下，甚至35.02。可能代码里有些地方没有实现好，之后再去读一些开源的项目学习一下。

### 一些工程tricks

#### 分布式训练

在pytorch上进行单机多卡的分布式训练，可以采取torch.nn.DataParallel或者torch.nn.parallel.DistributedDataParallel。

我将用表格简单介绍两者的一些优劣之处

|  |  |  |
| --- | --- | --- |
|  | DataParallel | DistributedDataParallel |
| 使用难度 | 非常简单，只需要把模型，可用的GPU和用于汇总的GPU传入DataParallel即可。  如model = nn.DataParallel(model.cuda(), device\_ids=gpus, output\_device=gpus[0]) | 比DataParallel多一步开启多线程，一般用torch.multiprocessing.spawn实现 |
| 多线(进)程 | 多线程 | 多进程 |
| 训练速度 | 受限于全局解释器锁GIL，维护同一个optimizer，梯度更新缓慢 | 比前者快很多 |
| 负载均衡 | 存在不均衡的现象 | 因为每个process管一个optimizer，所以不像前者要在一块主CPU上更新梯度，造成负载不均衡的情况。 |

经过分析和同时根据pytorch的使用建议，采用DistributedDataParallel来进行分布式训练设计。因为是gpu训练，backend采用nccl[[3]](#footnote-2)。

#### Time\_step的改动

训练过程中需要进行验证，但是在早期生成一个非常完整的句子去算bleu意义也不大，所以在current\_step到达两倍warmup\_step之前维持一个很小的时间步即可。

### 感想和收获：

抛开各种封装好的框架，从头实现transformer是一件很有意思的事情。在实验室暑期实习的时候反复读了很多遍attention is all you need，到如今从头写一遍这个流程，感觉对transformer又产生了更深入的理解。刚开始上手写的时候，debug比较轻松，到后期的时候随着模块的耦合繁杂，一环套一环的流程让debug变得不太容易。这时候怎么让整个工程组织的比较优雅，模块之间降低耦合程度，也是一个非常重要的环节，写的丑的代码不止折磨读者，还折磨了debug的自己。最后就是写好注释非常重要，因为这学期考试课和大作业不少，有时候写这个项目并不能投入大段的连续时间，写好注释可以让我快速回想起上次做到什么地方，还有什么实现需要注意，节约了大量的时间。

最后非常感谢肖老师和胡驰学长的课程讲授，王成龙学长和严钟响学长的认真指导。机器翻译这门课深入浅出地介绍了机器翻译的发展史，从SMT到NMT，从分词到翻译生成端到端式讲授，在实验阶段让我们实操了一些简单任务的复现，为刚接触nlp的萌新提供了友好的指引。

参考文献

1. Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems. 2017.
2. Sennrich, Rico, Barry Haddow, and Alexandra Birch. "Neural machine translation of rare words with subword units." arXiv preprint arXiv:1508.07909 (2015).
3. Hendrycks, Dan, and Kevin Gimpel. "Bridging nonlinearities and stochastic regularizers with gaussian error linear units." (2016).
4. Kingma D , Ba J . Adam: A Method for Stochastic Optimization[J]. Computer Science, 2014.
5. Loshchilov I , Hutter F . Fixing Weight Decay Regularization in Adam[J]. 2017.

1. <https://git.io/JPK9N> [↑](#footnote-ref-0)
2. https://github.com/pytorch/fairseq/blob/main/examples/translation/prepare-iwslt14.sh [↑](#footnote-ref-1)
3. https://pytorch.org/docs/stable/distributed.html [↑](#footnote-ref-2)