4. FastText

1．什么是fasttext

FastText是Facebook开发的一款快速文本分类器，提供简单而高效的文本分类和表征学习的方法。.将整篇文档的词及n-gram向量叠加平均得到文档向量，然后使用文档向量做softmax多分类。

2．fastText结构

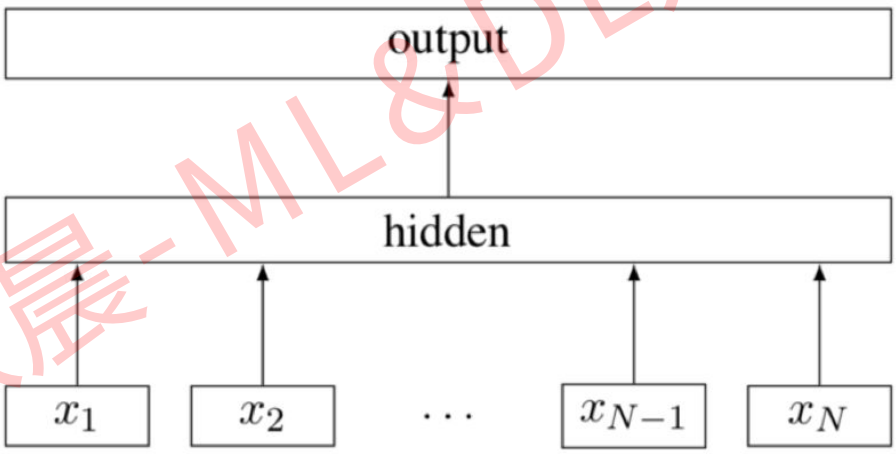

Figure 1: Model architecture of `fastText` for a sentence with N ngram features $x_1, \ldots, x_N$. The features are embedded and averaged to form the hidden variable.

Fasttext模型与CBOW的模型结构有点像，都有输入层、隐藏层、输出层，但二者还是有挺大区别的，二者的区别如下：

输入层：

- CBOW：输入的是每个词的One-hot向量；
- Fasttext：输入的是每个词的词嵌入向量 + 字符级别的ngram向量

隐藏层：

- CBOW：先将输入层的每个One-hot向量乘以词嵌入矩阵，得到每个词的词嵌入向量，再求和取平均；
- Fasttext：直接对输入层的词向量求和取平均。

输出层：

- CBOW：原始的CBOW使用Softmax方法，输出的类别是词表中的每个词的概率；
- Fasttext：使用Hiererachical softmax方法简化softmax计算，提升训练速度，Fasttext的输出是文章的类别。

3．损失函数

无监督：语言模型

首先**定义损失函数**，objective是最大化给定输入上下文，target单词的条件概率。因此，损失函数为：

$$
\begin{aligned}
E &= -\log p(w_O|w_I) \\
&= -u_{j^*} - \log \sum_{j'=1}^{V} \exp(u_{j'}) \\
&= -v_{w_O}{}^{T} \cdot h - \log \sum_{j'=1}^{V} \exp\left(v_{w_{j'}}{}^{T} \cdot h\right)
\end{aligned}
$$

有监督：文本分类问题

FastText用负对数似然作为损失函数：

$$
-\frac{1}{N}\sum_{n=1}^{N} y_n \log(f(BAx_n))
$$

其中N是文档数，xn是文档中的词特征。yn是标签，A和B是权重矩阵，A用于转换到文本表示，B用于线性变换计算类别，f是一个softmax函数用于计算最终分类的概率。

4．fasttext的输入是什么

fastText的输入是多个单词及其n-gram特征，这些特征组合起来用来表示单个文档

5．fasttext的字符级别的n-gram如何实现的

word2vec把语料库中的每个单词当成原子的，它会为每个单词生成一个向量。这忽略了单词内部的形态特征，比如：“apple” 和“apples”，“达观

数据”和“达观”，这两个例子中，两个单词都有较多公共字符，即它们的内部形态类似，但是在传统的word2vec中，这种单词内部形态信息因为它们被转换成不同的id丢失了。为了克服这个问题，fastText使用了字符级别的n-grams来表示一个单词。对于单词“apple”，假设n的取值为3，则它的trigram有“<ap”，“app”，“ppl”，“ple”，“le>”。其中，<表示前缀，>表示后缀。于是，我们可以用这些trigram来表示“apple”这个单词，进一步，我们可以用这5个trigram的向量叠加来表示“apple”的词向量。

这带来两点好处：

- **对于低频词生成的词向量效果会更好。因为它们的n-gram可以和其它词共享。**
- **对于训练词库之外的单词，仍然可以构建它们的词向量。我们可以叠加它们的字符级n-gram向量。**

6．n-gram带来的参数增多训练变慢的问题

使用了n-gram信息之后，词表肯定是变大了的。这就会出现问题：参数越多训练越慢。针对这个问题，怎么解决呢？使用哈希。

将n-gram进行hash，hash到同一个位置的多个n-gram是会共享一个embedding的。举个简单例子，不一定准确，"我/爱/中国/共产党"，我在更新的时候，把'我','爱','中国','共产党'我们都使用同一个参数来代表（这种情况很难遇见，理解一下就好），那么在更新训练参数的时候，我只需要更新一个参数就把这个四个词都更新了，当然会快一点。但是会出现一个问题，就是精度的问题。这个过程，不知道大家有没有想到和albert很类似。哈希这个过程我自己感觉有点共享参数的意思。

7．为什么训练的模型非常大

fastText对字和字符串使用hash表，hash表的大小将直接影响模型的大小；另一个影响模型大小重要的因素是训练向量的维度大小(-dim)，如果维度缩小模型将大大减小，但同时也会很大程度影响模型的性能，因为向量维度越大则捕获的信息越多。

8．模型中使用单词短语而不是单个单词最佳方式是什么

目前使用单词短语或句子最好的方式是使用词向量的bow(bag of words)，另一种方式例如New York，我们可以将其处理成New_York也会有帮助

9．为什么fastText甚至可以为语料库中未出现的单词产生词向量

fastText一个重要的特性便是有能力为任何单词产生词向量，即使是未出现的，组装的单词。主要是因为fastText是通过包含在单词中的子字符substring of character来构建单词的词向量，正文中也有论述，因此这种训练模型的方式使得fastText可以为拼写错误的单词或者连接组装的单词产生词向量。

10．为什么分层softmax在效果上比完全softmax略差

分层softmax是完全softmax的一个近似，分层softmax可以让我们在大数据集上高效的建立模型，但通常会以损失精度的几个百分点为代价。

11．可以在GPU上运行fastText项目吗

目前fastText仅仅可运行在CPU上，但这也是其优势所在，fastText的目的便是要成为一个高效的CPU上的分类模型，可以允许模型在没有GPU的情况下构建。

12．可以在连续的数据集上使用fastText吗

不可以，fastText仅仅是用于离散的数据集，因此无法直接在连续的数据集上使用，但是可以将连续的数据离散化后使用fastText。

13．数据中存在拼写错误，我们需要对文本进行规范化处理吗

如果出现的频率不高，没有必要，对模型效果不会有什么影响。因为训练数据足够多，几个错的不足以掩盖对的。

14．在模型训练时遇到了NaN，为什么会这样

这种现象是可能出现的，很大原因是因为你的学习率太高了，可以尝试降低一下学习率直到不再出现NaN。

15．如何完全重现fastText的运行结果，为什么每次运行的结果都有些差异

当多次运行fastText时，因为优化算法**异步随机梯度下降算法**或Hogwild!,所以每次得到的结果都会略有不同，如果想要fastText运行结果复现，则必须将参数thread设置为1，这样你就可以在每次运行时获得完成相同的性能。

16．fastText的优点

- fastText引入了subword n-gram的概念，解决了词形变化(morphology)的问题、低频词、未登录词的问题。
- fastText更适用于样本数量大、类别标签多的任务，一般能够得到很好的效果，大多数情况下强于传统的BOW + LR/SVM分类器。更重要的是，训练效率非常之高。

17．fastText的缺点

subword n-gram信息的加入，不但解决了低频词未登录词的表达的问题，而且对于最终任务精度一般会有几个百分点的提升。唯一的问题就是由于需要估计的参数多，模型可能会比较膨胀。不过，Facebook也提供了几点压缩模型的建议：

- 采用hash-trick。由于n-gram原始的空间太大，可以用某种hash函数将其映射到固定大小的buckets中去，从而实现内存可控；
- 采用quantize命令，对生成的模型进行参数量化和压缩；
- 减小最终向量的维度。

需要注意的是以上几种方法都会以一定的精度损失为代价，尤其是维度的压缩。

18．fastText与CBOW的不同之处在于

- 目标不同，fasttext是分类，CBOW是预测中心词
- fastText还可以进行有监督学习进行文本分类
- fastText引入N-gram，考虑词序特征
- fastText引入subword来处理长词，处理未登陆词问题
- fastText分层Softmax的叶子结点（类别）相对w2v（所有词汇）少很多，样本中标签多的类别被分配短的搜寻路径

19．fastText与Word2Vec的相同点

- fasttext与CBOW结构相似，多加了字符级别的n-gram丰富词向量
- 都可以去做无监督学习
- 都可采用两种加速方法
- 损失函数都是交叉熵损失

https://blog.csdn.net/ningyanggege/article/details/88971615 <https://blog.csdn.net/ningyanggege/article/details/88971615>

https://blog.csdn.net/budaibetter/article/details/106025518 <https://blog.csdn.net/budaibetter/article/details/106025518>

https://www.cnblogs.com/elisha/p/14033109.html

<https://www.cnblogs.com/elisha/p/14033109.html>