你可能不需要BERT-flow：一个线性变换媲美BERT-flow

11 Jan By 苏剑林 | 2021-01-11 | 17347位读者

BERT-flow来自论文《On the Sentence Embeddings from Pre-trained Language Models》，中了EMNLP 2020，主要是用flow模型校正了BERT出来的句向量的分布，从而使得计算出来的cos相似度更为合理一些。由于笔者定时刷Arixv的习惯，早在它放到Arxiv时笔者就看到了它，但并没有什么兴趣，想不到前段时间小火了一把，短时间内公众号、知乎等地出现了不少的解读，相信读者们多多少少都被它刷屏了一下。

从实验结果来看，BERT-flow确实是达到了一个新SOTA，但对于这一结果，笔者的第一感觉是：不大对劲！当然，不是说结果有问题，而是根据笔者的理解，flow模型不大可能发挥关键作用。带着这个直觉，笔者做了一些分析，果不其然，笔者发现尽管BERT-flow的思路没有问题，但只要一个线性变换就可以达到相近的效果，flow模型并不是十分关键。

余弦相似度的假设

一般来说，我们语义相似度比较或检索，都是给每个句子算出一个句向量出来，然后算它们的夹角余弦来比较或者排序。那么，我们有没有思考过这样的一个问题：余弦相似度对所输入的向量提出了什么假设呢？或者说，满足什么条件的向量用余弦相似度做比较效果会更好呢？

我们知道，两个向量$\boldsymbol{x},\boldsymbol{y}$的内积的几何意义就是“各自的模长乘以它们的夹角余弦”，所以余弦相似度就是两个向量的内积并除以各自的模长，对应的坐标计算公式是

$$\cos(\boldsymbol{x},\boldsymbol{y}) = \frac{\sum\limits_{i=1}^{d} x_i y_i}{\sqrt{\sum\limits_{i=1}^{d} x_i^2}\sqrt{\sum\limits_{i=1}^{d} y_i^2}} \tag{1}$$

然而，别忘了一件事情，上述等号只在“标准正交基”下成立。换句话说，向量的“夹角余弦”本身是具有鲜明的几何意义的，但上式右端只是坐标的运算，坐标依赖于所选取的坐标基，基底不同，内积对应的坐标公式就不一样，从而余弦值的坐标公式也不一样。

因此，假定BERT句向量已经包含了足够的语义（比如可以重构出原句子），那么如果它用公式(1)算余弦值来比较句子相似度时表现不好，那么原因可能就是此时的句向量所属的坐标系并非标准正交基。那么，我们怎么知道它具体用了哪种基底呢？原则上没法知道，但是我们可以去猜。猜测的依据是我们在给向量集合选择基底时，会尽量地用好每一个基向量，从统计学的角度看，这就体现为每个分量的使用都是独立的、均匀的，如果这组基是标准正交基，那么对应的向量集应该表现出“各项同性”来。

当然，这不算是什么推导，只是一个启发式引导，它告诉我们如果一个向量的集合满足各向同性，那么我们可以认为它源于标准正交基，此时可以考虑用式(1)算相似度；反之，如果它并不满足各向同性，那么可以想办法让它变得更加各向同性一些，然后再用式(1)算相似度，而BERT-flow正是想到了“flow模型”这个办法。

flow模型的碎碎念

依笔者来看，flow模型真的是一种让人觉得一言难尽的模型了，关于它的碎碎念又可以写上几页纸，这里尽量长话短说。2018年中，OpenAI发布了Glow模型，效果看起来很不错，这吸引了笔者进一步去学习flow模型，

甚至还去复现了一把Glow模型，相关工作记录在《细水长flow之NICE：流模型的基本概念与实现》和《细水长flow之RealNVP与Glow：流模型的传承与升华》中，如果还不了解flow模型的，欢迎去看看这两篇博客。简单来说，flow模型是一个向量变换模型，它可以将输入数据的分布转化为标准正态分布，而显然标准正态分布是各向同性的，所以BERT-flow就选择了flow模型。

那么flow模型有什么毛病吗？其实之前在文章《细水长flow之可逆ResNet：极致的暴力美学》就已经吐槽过了，这里重复一下：

> （flow模型）通过比较巧妙的设计，使得模型每一层的逆变换比较简单，而且雅可比矩阵是一个三角阵，从而雅可比行列式很容易计算。这样的模型在理论上很优雅漂亮，但是有一个很严重的问题：由于必须保证逆变换简单和雅可比行列式容易计算，那么每一层的非线性变换能力都很弱。事实上像Glow这样的模型，每一层只有一半的变量被变换，所以为了保证充分的拟合能力，模型就必须堆得非常深（比如256的人脸生成，Glow模型堆了大概600个卷积层，两亿参数量），计算量非常大。

看到这里，读者就能理解为什么笔者开头说看到BERT-flow的第一感觉就是“不对劲”了。上述吐槽告诉我们，flow模型其实是很弱的；然后BERT-flow里边所用的flow模型是多大呢？是一个level=2、depth=3的Glow模型，这两个参数大家可能没什么概念，反正就是很小，以至于整个模型并没有增加什么计算量。所以，笔者的“不对劲”直觉就是：

> flow模型本身很弱，BERT-flow里边使用的flow模型更弱，所以flow模型不大可能在BERT-flow中发挥至关重要的作用。反过来想，那就是也许我们可以找到更简单直接的方法达到BERT-flow的效果。

标准化协方差矩阵

经过探索，笔者还真找到了这样的方法，正如本文标题所说，它只是一个线性变换。

其实思想很简单，我们知道标准正态分布的均值为0、协方差矩阵为单位阵，那么我们不妨将句向量的均值变换为0、协方差矩阵变换为单位阵试试看？假设（行）向量集合为$\{\boldsymbol{x}_i\}_{i=1}^N$，我们执行变换

$$\tilde{\boldsymbol{x}}_i = (\boldsymbol{x}_i - \boldsymbol{\mu})\boldsymbol{W} \tag{2}$$

使得$\{\tilde{\boldsymbol{x}}_i\}_{i=1}^N$的均值为0、协方差矩阵为单位阵。了解传统数据挖掘的读者可能知道，这实际上就相当于传统数据挖掘中的白化操作（Whitening），所以该方法笔者称之为BERT-whitening。

均值为0很简单，让$\boldsymbol{\mu} = \frac{1}{N}\sum\limits_{i=1}^N \boldsymbol{x}_i$即可，有点难度的是$\boldsymbol{W}$矩阵的求解。将原始数据的协方差矩阵记为

$$\boldsymbol{\Sigma} = \frac{1}{N}\sum_{i=1}^N (\boldsymbol{x}_i - \boldsymbol{\mu})^\top (\boldsymbol{x}_i - \boldsymbol{\mu}) \tag{3}$$

那么不难得到变换后的数据协方差矩阵为$\tilde{\boldsymbol{\Sigma}} = \boldsymbol{W}^\top \boldsymbol{\Sigma} \boldsymbol{W}$，所以我们实际上要解方程

$$\boldsymbol{W}^\top \boldsymbol{\Sigma} \boldsymbol{W} = \boldsymbol{I} \quad \Rightarrow \quad \boldsymbol{\Sigma} = \left(\boldsymbol{W}^\top\right)^{-1} \boldsymbol{W}^{-1} = \left(\boldsymbol{W}^{-1}\right)^\top \boldsymbol{W}^{-1} \tag{4}$$

我们知道协方差矩阵$\boldsymbol{\Sigma}$是一个正定对称矩阵，正定对称矩阵都具有如下形式的SVD分解

$$\boldsymbol{\Sigma} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^{\top} \tag{5}$$

其中$\boldsymbol{U}$是一个正交矩阵，而$\boldsymbol{\Lambda}$是一个对角阵，并且对角线元素都是正的，因此直接让$\boldsymbol{W}^{-1} = \sqrt{\boldsymbol{\Lambda}}\boldsymbol{U}^{\top}$就可以完成求解：

$$\boldsymbol{W} = \boldsymbol{U}\sqrt{\boldsymbol{\Lambda}^{-1}} \tag{6}$$

Numpy的参考代码为：

```
def compute_kernel_bias(vecs):
    """计算kernel和bias
    vecs.shape = [num_samples, embedding_size],
    最后的变换：y = (x + bias).dot(kernel)
    """
    mu = vecs.mean(axis=0, keepdims=True)
    cov = np.cov(vecs.T)
    u, s, vh = np.linalg.svd(cov)
    W = np.dot(u, np.diag(1 / np.sqrt(s)))
    return W, -mu
```

可能会有人问答大语料怎么办的问题。首先，上述算法只需要知道全体句向量的均值向量$\boldsymbol{\mu} \in \mathbb{R}^d$和协方差矩阵$\boldsymbol{\Sigma} \in \mathbb{R}^{d\times d}$（$d$是词向量维度），$\boldsymbol{\mu}$是全体句向量$\boldsymbol{x}_i$的均值，均值是可以递归计算的：

$$\boldsymbol{\mu}_{n+1} = \frac{n}{n+1}\boldsymbol{\mu}_n + \frac{1}{n+1}\boldsymbol{x}_{n+1} \tag{7}$$

同理，协方差矩阵$\boldsymbol{\Sigma}$也只不过是全体$(\boldsymbol{x}_i - \boldsymbol{\mu})^{\top}(\boldsymbol{x}_i - \boldsymbol{\mu})$的均值，自然也是可以递归计算量：

$$\boldsymbol{\Sigma}_{n+1} = \frac{n}{n+1}\boldsymbol{\Sigma}_n + \frac{1}{n+1}(\boldsymbol{x}_{n+1} - \boldsymbol{\mu})^{\top}(\boldsymbol{x}_{n+1} - \boldsymbol{\mu}) \tag{8}$$

既然可以递归，那么就意味着我们是可以在有限内存下计算$\boldsymbol{\mu}, \boldsymbol{\Sigma}$的，因此对于大语料来说BERT-whitening也不成问题的。

相比于BERT-flow

现在，我们就可以测试一下上述BERT-whitening的效果了。为了跟BERT-flow对比，笔者用bert4keras在STS-B任务上进行了测试，参考脚本在：

> **Github链接：https://github.com/bojone/BERT-whitening**

效果比较如下：

	STS-B
$\text{BERT}_{\text{base}}$-last2avg (论文结果)	59.04
$\text{BERT}_{\text{base}}$-flow (target, 论文结果)	70.72
$\text{BERT}_{\text{base}}$-last2avg (个人复现)	59.04
$\text{BERT}_{\text{base}}$-whitening (target, 个人实现)	71.20
$\text{BERT}_{\text{large}}$-last2avg (论文结果)	59.56
$\text{BERT}_{\text{large}}$-flow (target, 论文结果)	72.26
$\text{BERT}_{\text{large}}$-last2avg (个人复现)	59.59
$\text{BERT}_{\text{large}}$-whitening (target, 个人实现)	71.98

可以看到，简单的BERT-whitening确实能取得跟BERT-flow媲美的结果。除了STS-B之外，笔者的同事在中文业务数据内做了类似的比较，结果都表明BERT-flow带来的提升跟BERT-whitening是相近的，这表明，flow模型的引入可能没那么必要了，因为flow模型的层并非常见的层，它需要专门的实现，并且训练起来也有一定的工作量，而BERT-whitening的实现很简单，就一个线性变换，可以轻松套到任意的句向量模型中。（当然，非要辩的话，也可以说whitening是用线性变换实现的flow模型...）

> 注：这里顺便补充一句，BERT-flow论文里边说的last2avg，本来含义是最后两层输出的平均向量，但它的代码实际上是“第一层+最后一层”输出的平均向量，相关讨论参考ISSUE。

降维效果还能更好

现在我们知道BERT-whitening的变换矩阵$\boldsymbol{W}=\boldsymbol{U}\sqrt{\boldsymbol{\Lambda}^{-1}}$可以将数据的协方差矩阵变换成单位阵，如果我们不考虑$\sqrt{\boldsymbol{\Lambda}^{-1}}$，直接用$\boldsymbol{U}$来变换，结果如何呢？不难得出，如果只用$\boldsymbol{U}$来变换，那么数据的协方差矩阵就变成了$\boldsymbol{\Lambda}$，它是个对角阵。

前面说了，$\boldsymbol{U}$是一个正交矩阵，它相当于只是旋转了一下整体数据，不改变样本之间的相对位置，换句话说它是完全“保真”的变换。而$\boldsymbol{\Lambda}$的每个对角线元素，则衡量了它所在的那一维数据的变化幅度。如果它的值很小，说明这一维特征的变化很小，接近一个常数，那么就意味着原来句向量所在可能只是一个更低维的空间，我们就可以去掉这一维特征，在降维的同时还可以使得余弦相似度的结果更为合理。

事实上，SVD出来的对角矩阵$\boldsymbol{\Lambda}$已经从大到小排好序了，所以我们只需要保留前面若干维，就可以到达这个降维效果。熟悉线性代数的读者应该清楚，这个操作其实就是PCA！而代码只需要修改一行：

```
def compute_kernel_bias(vecs, n_components=256):
    """计算kernel和bias
    vecs.shape = [num_samples, embedding_size],
    最后的变换：y = (x + bias).dot(kernel)
    """
    mu = vecs.mean(axis=0, keepdims=True)
    cov = np.cov(vecs.T)
    u, s, vh = np.linalg.svd(cov)
    W = np.dot(u, np.diag(1 / np.sqrt(s)))
    return W[:, :n_components], -mu
```

效果如下：

	STS-B
$\text{BERT}_{\text{base}}$-last2avg (论文结果)	59.04
$\text{BERT}_{\text{base}}$-flow (target, 论文结果)	70.72
$\text{BERT}_{\text{base}}$-last2avg (个人复现)	59.04
$\text{BERT}_{\text{base}}$-whitening (target, 个人实现)	71.20
$\text{BERT}_{\text{base}}$-whitening-256 (target, 个人实现)	71.42
$\text{BERT}_{\text{large}}$-last2avg (论文结果)	59.56
$\text{BERT}_{\text{large}}$-flow (target, 论文结果)	72.26
$\text{BERT}_{\text{large}}$-last2avg (个人复现)	59.59
$\text{BERT}_{\text{large}}$-whitening (target, 个人实现)	71.98
$\text{BERT}_{\text{large}}$-whitening-384 (target, 个人实现)	72.66

从上表可以看出，我们将base版本的768维只保留前256维，那么效果还有所提升，并且由于降维了，向量检索速度肯定也能大大加快；类似地，将large版的1024维只保留前384维，那么降维的同时也提升了效果。这个结果表明，无监督训练出来的句向量其实是“通用型”的，对于特定领域内的应用，里边有很多特征是冗余的，剔除这些冗余特征，往往能达到提速又提效的效果。

相比之下，flow模型是可逆的、不降维的，这在某些场景下是好处，但在不少场景下也是缺点，因为它无法剔除冗余维度，限制了性能，比如GAN的研究表明，通过一个256维的高斯向量就可以随机生成1024×1024的人脸图，这表明这些人脸图其实只是构成了一个相当低维的流形，但是如果用flow模型来做，因为要保证可逆性，就得强行用$1024 \times 1024 \times 3$那么多维的高斯向量来随机生成，计算成本大大增加，而且效果还上不去。

（注：后续实验结果，请看《无监督语义相似度哪家强？我们做了个比较全面的评测》。）

所以最终结论就是

所以，目前的结果就是：笔者的若干实验表明，通过简单的线性变换（BERT-whitening）操作，效果基本上能媲美BERT-flow模型，这表明往句向量模型里边引入flow模型可能并非那么关键，它对分布的校正可能仅仅是浅层的，而通过线性变换直接校正句向量的协方差矩阵就能达到相近的效果。同时，BERT-whitening还支持降维操作，能达到提速又提效的效果。