# Word Tensors 词张量

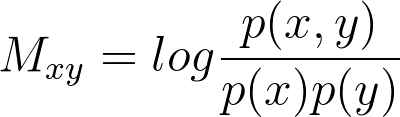
原文链接：  
原文链接：

To see the first part of this article, check out “Stop Using word2vec”  
要查看本文的第一部分，请查看“停止使用word2vec”

In the , we saw that we can get word vectors by factorizing a 2D matrix of word co-occurrences. But what do we get if we factorize a 3D tensor?  
在中，我们看到可以通过分解单词共现的二维矩阵来获得单词向量。但是如果我们把一个三维张量分解，会得到什么呢？

If our tensor is the association between word X, word Y within document Z, it turns out that like doc2vec, we can get word and document vectors!  
如果我们的张量是文档Z中单词X，单词Y之间的关联，那么就像doc2vec一样，我们可以得到单词和文档向量！

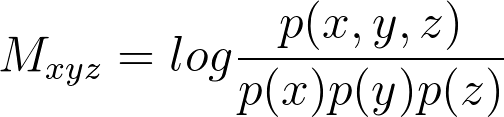
In skipgram matrix factorization, we SVD factorize a matrix like:  
在skipgram矩阵分解中，我们对矩阵进行SVD分解，如下所示：



rank2 PMI  
兰克2 PMI

This says: construct a large, potentially very sparse, matrix M where we’ve counted how frequently word x cooccurrs near word y and normalized that count to get the probability p(x, y). Then we divide by the probabilty of each word by itself (instead of the cooccurrence), where p(x) and p(y) are essentially the popularity of word xand word y. If this ratio is far above or below 1.0, then there’s something special about the relationship between token x and y, but if it’s near 1.0, then they co-occcur at the ‘usual’ rate. The log of this ratio is called the and has deep connections to information theory. Read our previous blog post for more details.  
这就是说：构造一个大的，可能非常稀疏的矩阵M，在这里我们计算了单词x在单词y附近出现的频率，并对其进行了规范化，得到了概率p（x，y）。然后我们用每个单词的概率除以它本身（而不是共现），其中p（x）和p（y）本质上是单词x和y的受欢迎程度。如果这个比率远高于或低于1.0，那么记号x和y之间的关系就有一些特殊之处，但如果它接近1.0，那么它们以“通常”的速率共现。这个比率的对数称为，与信息论有着深刻的联系。阅读我们以前的博客文章了解更多细节。

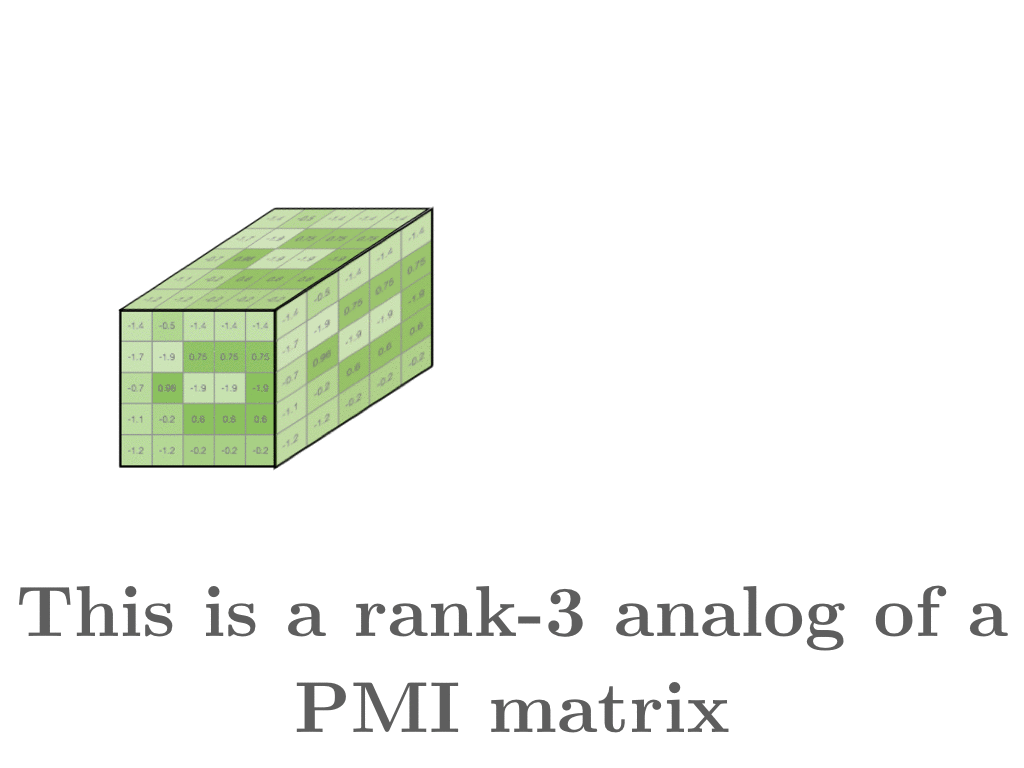
But in tensor factorization we can choose to factorize a tensor like this:  
但是在张量因式分解中，我们可以选择这样的张量因式分解：



rank3 PMI  
rank3采购经理

This is similar to the previous matrix, but with that extra dependence on z this tensor is now indexed by three variables instead of two. And like in the 2D case, we measure how often the three objects x, y, z cooccur (instead of just x and y) and then divide by p(x), p(y) and p(z) that measure the individual word probabilities. In this example, we’ll count how frequently word x cooccurs near word y in document z. At Stitch Fix, z is an index over all comments about a single piece of clothing, so z typically encodes all written information about a particular style or item of clothing.  
这与前一个矩阵类似，但是由于对z的额外依赖，这个张量现在由三个变量而不是两个变量索引。就像在二维情况下，我们测量三个对象x，y，z cooccur（而不是x和y）的频率，然后除以p（x），p（y）和p（z）来测量单个单词的概率。在本例中，我们将计算文档z中单词x在单词y附近的出现频率。在Stitch Fix中，z是一个索引，覆盖了关于一件衣服的所有注释，因此z通常会对关于一种特定样式或一件衣服的所有书面信息进行编码。

Having formed this large 3D tensor, we can then decompose it into three 2D modes: one mode for the word index x, one mode for the word index y, and another for document index z.  
在形成了这个大的三维张量之后，我们可以将其分解为三种二维模式：一种模式用于单词索引x，一种模式用于单词索引y，另一种模式用于文档索引z。



Tensor  
张量

In this example, the first and second modes contain word vectors, which are word representations our clients use to describe their fixes. The third mode represents document vectors which yields a summary about everything said about one style.  
在本例中，第一和第二个模式包含词向量，这是我们的客户机用来描述其修复的词表示。第三种模式表示文档向量，它生成关于一种样式的所有内容的摘要。

Interpreting these matrices is easy. Out of one of the two word matrices, we can extract the row vector corresponding to the word spandex and see what other row vectors are similar. And voilà! spandex turns out to be similar to stretchy\_fabric and jeggings. Just like the original word2vec, we get that these tokens are similar because they cooccur in similar contexts. Spandex on occasion can squeeze your body into awkward ways, which explains why the token sausage ends up in similar contexts.  
解释这些矩阵很容易。从两个单词矩阵中的一个中，我们可以提取与单词spandex对应的行向量，并查看其他哪些行向量相似。还有，喂！氨纶类似于弹性纤维和胶布。就像最初的word2vec一样，我们得到这些标记是相似的，因为它们在相似的上下文中共存。有时氨纶会让你的身体陷入尴尬的境地，这就解释了为什么这种象征性的香肠会出现在类似的环境中。

Our documents are composed of all comments written about a single style. So if we look at what is similar to a denim jacket, the results are populated with sensible items: denim jackets, jackets with similar cuts, and jackets with similarly prominent brass buttons. If our input query is a salmon-colored Dolman top, the closest items are other Dolman tops with varying colors.  
我们的文档由所有关于单一样式的注释组成。因此，如果我们看看什么是类似于牛仔夹克，结果是充满了明智的项目：牛仔夹克，夹克与类似削减，夹克与类似突出的黄铜按钮。如果我们的输入查询是一个鲑鱼色的Dolman top，那么最接近的项是其他颜色不同的Dolman top。

We aren’t limited to exploring vectors within a single mode – we can compare vectors from words matrices to vectors within the document matrix. This cross-modality allows us to find interesting relationships between styles and their descriptions. For one of our most snuggle-worthy styles, we can see that it’s closest word vector is cocoon\_cardigan!  
我们不仅限于探索单一模式下的向量，我们还可以将词矩阵中的向量与文档矩阵中的向量进行比较。这种交叉情态使我们能够发现风格与其描述之间有趣的关系。对于我们最值得依偎的款式之一，我们可以看到它最接近的文字载体是cocoon\_开襟衫！

## Tensor Decomposition 张量分解

There’s only three steps to computing word tensors. Counting word-word-document skipgrams, normalizing those counts to form the PMI-like M tensor and then factorizing M into smaller matrices.  
计算单词时态只有三个步骤。对word文档技巧进行计数，规范化这些计数以形成类似PMI的M张量，然后将M分解为更小的矩阵。

But to actually perform the factorization we’ll need to generalize the SVD to higher rank tensors . Unfortunately, tensor algebra libraries aren’t very common . We’ve written one for non-negative , but because the PMI can be both positive and negative it isn’t applicable here. Instead, for this application I’d recommend HOSVD as implemented in . I’ve also heard good things about .  
但要实际执行分解，我们需要将SVD推广到更高阶的张量。不幸的是，张量代数库并不常见。我们已经为非负编写了一个，但是因为PMI可以是正的也可以是负的，所以这里不适用。相反，对于此应用程序，我建议使用中实现的HOSVD。我也听说过一些好消息。

## Conclusion 结论

Counting and tensor decompositions are elegant and straightforward techniques. But these methods are grossly underepresented in business contexts. In this post we factorized an example made up of word skipgrams occurring within documents to arrive at word and document vectors simultaneously. This kind of analysis is effective, simple, and yields powerful concepts.  
计数和张量分解是优雅而直接的技术。但这些方法在商业环境中的代表性严重不足。在这篇文章中，我们将一个由文档中出现的word skipgrams组成的示例分解为同时到达word和文档向量。这种分析是有效的，简单的，并产生强大的概念。

Look to your own data, and before throwing black-box deep learning machines at them, try out tensor factorizations!  
看看你自己的数据，在向它们扔黑匣子深度学习机器之前，试试张量分解！

## Footnotes 脚注

1. 1 For a very approachable introduction and review of tensor decomposition methods, check out   
   1对于非常平易近人的张量分解方法的介绍和回顾，请查看
2. 2 And no, despite it’s name, and to my chagrin, Tensorflow does not intrinsically support tensor decompositions.   
   不，尽管它的名字，让我懊恼的是，Tensorflow本质上不支持张量分解。