万字长文帮你彻底搞定词向量

公众号：NLP从入门到放弃

1. 灵魂20问帮你彻底搞定词向量

1. 有没有使用自己的数据训练过Word2vec，详细说一下过程。包括但是不限于：语料如何获取，清理以及语料的大小，超参数的选择及其原因，词表以及维度大小，训练时长等等细节点。
2. Word2vec模型是如何获得词向量的？聊一聊你对词嵌入的理解？如何理解分布式假设?
3. 如何评估训练出来的词向量的好坏
4. Word2vec模型如何做到增量训练
5. 大致聊一下 word2vec这个模型的细节，包括但不限于：两种模型以及两种优化方法（大致聊一下就可以，下面会详细问）
6. 解释一下 hierarchical softmax 的流程(CBOW and Skip-gram)
7. 基于6，可以展开问一下模型如何获取输入层，有没有隐层，输出层是什么情况。
8. 基于6，可以展开问输出层为何选择霍夫曼树，它有什么优点，为何不选择其他的二叉树
9. 基于6，可以问该模型的复杂度是多少，目标函数分别是什么，如何做到更新梯度（尤其是如何更新输入向量的梯度）
10. 基于6，可以展开问一下 hierarchical softmax 这个模型 有什么缺点
11. 聊一下负采样模型优点（为什么使用负采样技术）
12. 如何对输入进行负采样（负采样的具体实施细节是什么）
13. 负采样模型对应的目标函数分别是什么（CBOW and Skip-gram）
14. CBOW和skip-gram相较而言，彼此相对适合哪些场景
15. 有没有使用Word2vec计算过句子的相似度，效果如何，有什么细节可以分享出来
16. 详细聊一下Glove细节，它是如何进行训练的？有什么优点？什么场景下适合使用？与Word2vec相比，有什么区别（比如损失函数）？
17. 详细聊一下Fasttext细节，每一层都代表了什么？它与Wod2vec的区别在哪里？什么情况下适合使用Fasttext这个模型?
18. ELMO的原理是什么？以及它的两个阶段分别如何应用？（第一阶段如何预训练，第二阶段如何在下游任务使用）
19. ELMO的损失函数是什么？它是一个双向语言模型吗？为什么?
20. ELMO的优缺点分别是什么？为什么可以做到一词多义的效果?

2. W2C模型篇--一个词通过Word2vec训练之后，可以得到几个词向量?

先问大家个问题：一个词通过Word2vec训练之后，可以得到几个词向量?

如果您有点懵，说明我写对了，您慢慢看下去，码字不易，**请多多点赞，让更多人看到**，谢谢。

词向量一般被认为是一个词的特征向量，也就是说可以代表一个词的含义。一个中文词，比如"中国"这个词，是很难被神经网络直接作为输入，我们需要将词向量化（或者说数字化），才能更好的喂进神经网络。

这个过程很类似于计算机在处理我们输入的文字的时候，会转化为二进制进行处理。

只不过这个二进制表达规则我们会自己人为规定，而词向量这个向量表达根据不同的方式会有着不同的结果。**想一下，两者是不是很类似。**

谈到向量表达单词，一般首先想到的都是One-Hot编码。但是它是有缺点的。我这里谈两个缺点。

首先是**维度灾难**：如果我们有1万个单词，那么你的One-hot去表示每个单词的的时候，会转变为一个1万维度的向量。

那么在计算的时候会带来巨大的不便，而且向量矩阵极其稀疏，占据了太多不必要的内存。当然对于维度灾难，我们一般可以使用PCA等降维手段来缓解。

其次是**语义表达不足**。这一点很简单，”娱乐“与”八卦“两个词，通过One-hot编码得到的向量，计算Consine得到相似度为0。

这显然是不合适的。**One-Hot编码表示出来的词向量是两两正交的**，从余弦相似度的角度来看，是互不相关的，所以 One-Hot 不能很好的表达词语的相似性。

这个时候，我们再来看 Word2vec。首先，要明确一点，**Word2vec 不是一个模型，而是一个工具**。这个点虽然无伤大雅，但是每每看到有的文章说它是个模型，就有点...

Word2vec 从整体上理解，就是包含两个模型（CBOW and Skip-gram）和两个高效的优化训练方式（负采样和层序softmax）

这个文章主要谈一下两种模型。

先假定我们的窗口大小为2，也就是说中心词前面有两个词，后面有两个词，比如"我/永远/爱/中国/共产党"，抽象出来就是：

$$W_{t-2}W_{t-1}W_tW_{t+1}W_{t+2}$$

对于Skip-gram，中文我们叫跳字模型，使用中心词预测背景词。也就是用"爱"去预测其他的四个词。

对于CBOW，中文我们叫连续词袋模型，使用背景词预测中心词，也就是用"我"，"永远"，"中国"，"共产党" 去预测"爱"。

CBOW的模型架构，从整体上，我们可以分为三层：输入层，投影层，和输出层。

对于输入层，对应的是窗口中的单词，也就是例子中"我"，"永远"，"中国"，"共产党" 四个词的词向量，在投影层，将四个词的词向量进行相加求平均，输出层在没有优化的前提下，维度为词表大小，随后做 Softmax即可。

Skip-gram模型架构很类似，只不过可以省去投影层，因为输入为中心词，是一个单词的词向量，无从谈起求和与平均了。

接下来，我们来详细谈一下Skip-gram。

对于一个神经网络模型，我们需要确定我们的优化目标或者说目标函数是什么?

对于Skip-gram，其实很容易理解，就是我要最大化在给出中心词的条件下背景词出现的概率，公式如下

$$\prod_{t=1}^{T} \prod_{-m \leq j \leq m, j \neq 0} P(w^{t+j} | w^{t})$$

这个公式对应着两个连乘，第一个连乘是对应 T 个输入样本，也就是说我们文本中的每个单词都要做中心词。第二个连乘代表着给定一个中心词的情况下，窗口中的单词出现的概率，内含相互独立的假设。

在这里，**有一个细节点想要提醒大家，在词汇表中的每个单词，都是对应两个词向量的，一个是在作为中心词的时候的中心词向量，一个是在作为背景词时候的背景词向量。**

优化目标函数的时候，为了减少复杂度（也就是忽略 T），我们可以使用随机梯度下降，针对每一个样本我们都更新一次参数。

通过公式推导（略），我们能够观察到一个特点，就是每次更新参数，都会涉及到词典中的全部词汇，这是因为我们在做 Softmax 的时候，是分母是针对所有词汇的操作。

这样的操作的复杂度是 O(|V|)，其中|V|就是我们词汇表的大小。CBOW 其实是很类似的情况，每次更新都会涉及到我们的全部词汇。

于是，我们就需要对此进行优化，使用了两种近似的训练方式来高效的训练Word2vec，降低训练的复杂度。

3. W2C优化方式篇

Word2vec 涉及到两种优化方式，一种是负采样，一种是层序Softmax

先谈一下负采样，以跳字模型为例。中心词生成背景词可以由两个相互独立事件的联合组成来近似（引自李沐大神的讲解）。

第一个事件是，中心词和背景词同时出现在窗口中。第二个事件是，中心词和K个噪声词不同时出现在窗口数据中，其中噪声词由噪声分布随机生成。

这里我们就可以知道上一个文章开头说到的，负采样是一种等价操作还是近似操作？我们在第二个事件中，使用了K个噪声词。但是实际上呢？应该远远大于K。

还是那个例子，句子为"我/永远/爱/中国/共产党"，中心词为'爱'，我们在选择噪声词的时候，选择了K个，但是实际上，在词汇表中，排除掉'我'，'永远'，'中国'，'共产党' 这四个词汇的其他词都可以算做我的噪声词，然而为了减少复杂度，我只选择了其中的K个，所以当然应该是近似了。

接下来，我们看层序Softmax。

层序Softmax 对应的就是在输出层使用一个霍夫曼树，代替了原本在输出层统一进行的softmax。

首先，我们需要了解霍夫曼树在这里是如何构建的。

简单讲，霍夫曼树是一个二叉树，以语料中出现过的词当做叶子节点，以各词在语料中出现的次数当做权值进行构造。其中叶子节点有N个，就是词典的大小，非叶子节点有N-1个（包括根节点）。

比如说我的所有文章中，“共产党”这个词出现了 100次，是最大的，那么根节点的左分支（或者右分支）就对应着”共产党“这个词，另一个分支做与根节点相同的操作，找到排除”共产党“这个词之外的所有词中最大的词，比如”中国“作为其中的左分支（或者右分支），以此类推，一个霍夫曼树就成功构建。

霍夫曼树中，我们需要注意的是，每个非叶子节点对应一个向量，每个叶子节点对应一个向量。两种向量都会随着模型的训练进行更新。

其中叶子节点的向量就是我们的词向量，而非叶子节点上的向量就是没有什么实际含义，它的作用就是帮助我们计算模型在霍夫曼树上不断的进行二分类时候的概率。

以上面那句话为例，我们现在中心词为‘爱’，然后，我要预测背景词‘中国’。首先我们要确定的是我的叶子节点是包含所有单词的，也就是包含了我这个简单句子的五个单词（不考虑前期数据清洗低频率词的情况）。

也就是说，在这个霍夫曼树上，有且仅有一条路径，让我从根节点出发，经过多次判断（也就是说走过了多个非叶子节点），最终走到了“中国”这个叶子节点，对应的概率就是每个节点概率的连乘。

然后这个时候，我们想一下霍夫曼树是不是一种近似?

当然，我们每更新一个词向量，只是涉及到了可以到达叶子节点的这一条路径上节点。所以复杂度就是树的高度，也就是 O(log|V|)

4. W2C-负采样/霍夫曼之后模型是否等价

私下有几个朋友看完之后还是有点懵，又问了一下具体细节。基于此，我重新写了一个简短的文章，希望能让大家明白，大家可以结合上一个文章看。

我们再看一下题目：W2V经过霍夫曼或者负采样之后，模型与原模型相比，是等价的还是相似的?

首先，我们要明确，这里的原模型指的是什么？原模型就是我们的没有经过优化的W2V（当然我们也说过它是一个工具不是一个模型）。

也就是只是使用Skip-gram模型或者CBOW模型而没有进行优化的原始版本。对于这个原始版本，是在最后一层进行了Softmax。

我们的目标函数中，最核心的一个部分就是在给定中心词的条件下生成正确背景词的概率，我们要最大化这个东西，公式如下：

$$P(w_o \mid w_c) = \frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{i \in \mathcal{V}} \exp(\boldsymbol{u}_i^\top \boldsymbol{v}_c)}.$$

仔细看，在分母涉及到了一个V，这里的V就是我们的词典大小。也就是说，为了计算这个条件概率，我们需要对整个词典进行操作，复杂度就是O(|V|)

所以，负采样和霍夫曼就是针对这一个计算开销大的地方进行了优化。当然W2V为了减少计算量，还是去掉了隐层。比如CBOW直接是输入向量求和平均然后接霍夫曼树。比如Skip-gram直接是中心词的词向量接霍夫曼树。

这不是我这个文章的重点，就不细细展开了。

我们先说负采样。负采样的本质在于生成K个噪声。它的本质是基于中心词生成正确的背景词概率为1，生成噪声词概率为0，这个是我们的优化方向。公式如下：

仔细看这个公式，V已经消失，取而代之的是K，也就是我们的噪声词的数量，换句话讲，我们的复杂度被K这个大小限制住了，降低为了O(|K|)

然后我们再来看层序Softmax。它的核心本质是在一条路径上不停的做二分类，概率连乘就会得到我们的条件概率。公式如下：

$$P(w_o \mid w_c) = \prod_{j=1}^{L(w_o)-1} \sigma\left([\![n(w_o, j+1) = \text{leftChild}(n(w_o, j))]\!] \cdot \boldsymbol{u}_{n(w_o,j)}^{\top} \boldsymbol{v}_c\right),$$

注意看，这个公式中，V也已经消失了，被霍夫曼树中到达背景词的路径限制住了，这也就是上个文章中说到的，复杂度变成了二叉树的高度: O(log|V|)

既然只是针对的部分节点，那么与原始版本相比，当然是近似。

简单的总结一下：

其实可以这样理解，以跳字模型为例，条件概率是中心词生成背景词的概率，也就是我们优化函数中最核心的部分。没有使用优化的，分母涉及到全部词汇，训练开销大。负采样近似训练，把复杂度限制在了k个噪声词，层序softmax也属于近似训练，在它的条件概率中，不断的二分类，涉及到的是能够达到背景词的那个路径上的非叶子结点，也就是没涉及到其他节点，这一点和负采样很类似，都是从全部词汇降低复杂度，只不过负采样是被k限制，层序是被路径编码限制(0,1,1,1,0)这种限制住。

不知道大家有没有注意到，负采样和霍夫曼都是讲Softmax转化为二分类的问题从而降低了复杂度。负采样是针对是不是背景词做二分类，霍夫曼是在对是不是正确路径上的节点做二分类。这么说有点不严谨，但是意思就是这么个意思，大家理解一下。

5. Word2vec训练参数的选定?

首先根据具体任务，选一个领域相似的语料，在这个条件下，语料越大越好。然后下载一个 word2vec 的新版（14年9月更新），语料小（小于一亿词，约 500MB 的文本文件）的时候用 Skip-gram 模型，语料大的时候用 CBOW 模型。最后记得设置迭代次数为三五十次，维度至少选 50，就可以了。（引自《How to Generate a Good Word Embedding》）

6. W2C为什么需要二次采样?

说到 Word2vec 的采样，首先会想起来的是负采样，属于对Word2vec的一个近似训练方法。

其实它还涉及到一个采样方法，就是subsampling，中文叫做二次采样。 用最简单的一句话描述二次采样就是，对文本中的每个单词会有一定概率删除掉，这个概率是和词频有关，越高频的词越有概率被删掉。 二次采样的公式如下所示：

$$P(w_i) = \max\left(1 - \sqrt{\frac{t}{f(w_i)}}, 0\right),$$

注意: t为超参数，分母 f(w) 为单词w的词频与总词数之比

首先说一下，我们需要对文本数据进行二次采样？ 举个简单例子，“他/是/个/优秀/的/学生”。如果此时中心词为"学生"，背景词为"的"。

那么，我们的背景词对于我们这个中心词其实是没有什么作用的，并没有什么语义信息上的补充。

但是像“的”这种高频词，出现的机会还很大，所以对于这一句话信息是存在冗余的。 也就是说，在一个背景窗口中，一个词和较低频词同时出现比和较高频词同时出现对训练词嵌入模型更有益。

举个生活中的例子，现实生活中自律优秀的人比较少，堕落不努力人的人比较多，当然是优秀的人出现在我们身边会对我们自身的成长更加的有益。

所以我们的想法就是减少和堕落的人出现的次数，远离他们，让优秀的人出现在我们生活中的概率上升。

那么二次采样之后文本数据变成了什么样子?

还是上面那句话，“他/是/个/优秀/的/学生”，在这个时候，就变成了“他/是/个/优秀/学生”。也就是说高频词“的”在我们的训练数据中消失了。

当然这个消失正如上文所说，是一个概率，可能在之后的另一个句子中，它还是存在的，只不过它出现在文本中的词频肯定是降低了的。

7. Word2vec的负采样

负采样的特点

首先对基于负采样的技术，我们更新的权重只是采样集合，减少了训练量，同时效果上来说，中心词一般来说只和上下文有关，更新其他词的权重并不重要，所以在降低计算量的同时，效果并没有变差。

负采样具体实施细节

我自己的总结就是创建两个线段，第一个线段切开词表大小的份数，每个份数的长度和频率正比。

第二个线段均分M个，然后随机取整数，整数落在第二个线段那里，然后取第一个线段对应的词，如果碰到是自己，那么就跳过。

欢迎拍砖

8. W2C模型究竟是如何获得词向量的

问大家一个问题：Word2vec模型是如何获得词向量的?

很多文章在解释的时候，会说对一个词通过One-hot编码，然后通过隐层训练，得到的输入到隐层的矩阵就对应的词表词向量。

我不能说这么解释是不对的，但是我认为是不准确的。

在前面文章也说过了，Word2vec是不涉及到隐层的，CBOW有投影层，只是简单的求和平均，Skip-gram没有投影层，就是中心词接了一个霍夫曼树。

所以，很多文章涉及到的隐层的权重矩阵也就无从谈起。

在此情况下，词向量是怎么来的?

从源码的角度来看，我们是对每个词都初始化了一个词向量作为输入，这个词向量是会随着模型训练而更新的，词向量的维度就是我们想要的维度，比如说200维。

以Skip-gram为例，我们的输入的中心词的词向量其实不是One-hot编码，而是随机初始化的一个词向量，它会随着模型训练而更新。

需要注意的一个细节点是，每个单词都会对应两个词向量，一个是作为中心词的时候的词向量，一个是作为背景词的时候的词向量。大家一般选择第一种。

这一点需要注意区别Glove中的中心词向量和背景词向量。Glove中的中心词向量和背景词向量从理论上来说是等价的，只不过由于初始化的不同，最终结果会略有不同。

9. CBOW和skip-gram相较而言，彼此相对适合哪些场景

先用一句话来个结论：CBOW比Skip-gram 训练速度快，但是Skip-gram可以得到更好的词向量表达。

为什么这么说?

因为我们知道两种优化方式只是对softmax的近似优化，不会影响最终结果，所以这里，我们讨论的时候，为了更加的清晰讲解，不考虑优化的情况。

使用一句话作为一个例子：“我/永远/爱/中国/共产党”

先说CBOW，我们想一下，它的情况是使用周围词预测中心词。如果“爱”是中心词，别的是背景词。对于“爱”这个中心词，只是被预测了一次。

对于Skip-gram，同样，我们的中心词是“爱”，背景词是其他词，对于每一个背景词，我们都需要进行一次预测，每进行一次预测，我们都会更新一次词向量。也就是说，相比CBOW，我们的词向量更新了2k次（假设K为窗口，那么窗口内包含中心词就有2k+1个单词）

想一下是不是这么回事？Skip-gram被训练的次数更多，那么词向量的表达就会越丰富。

如果语料库中，我们的的低频词很多，那么使用Skip-gram就会得到更好的低频词的词向量的表达，相应的训练时长就会更多。

简单来说，我们视线回到一个大小为K的训练窗口（窗口内全部单词为2k+1个），CBOW只是训练一次，Skip-gram 则是训练了2K次。当然是Skip-gram词向量会更加的准确一点，相应的会训练的慢一点。

欢迎大佬拍砖

10. Fasttext解读-文本分类

我先说一个小问题，估计很多人也有疑惑。

看了很多文章，有的说是fasttext是CBOW的简单变种，有的说是Skip-gram的变种。究竟哪个是对的?

带着这个问题，我们来聊一聊Fasttext。首先Fasttext涉及到两个论文：

1. 第一个是Bag of Tricks for Efficient TextClassification(201607)。它解决的问题是使用Fasttext进行文本分类
2. 第二个是Enriching Word Vectors with Subword Information(201607) 。它解决的是使用Fasttext训练词向量。

今天这个文章，主要谈一下Bag of Tricks for Efficient Text Classification 这个论文 ，主要涉及到的就是文本分类的问题。

Fasttext用作文本分类，做到了速度和精读的一个平衡：标准多核CPU情况下，不到十分钟，可以训练超过十亿个单词。不到一分钟，可以对50万个句子在312千个类别中进行分类。

这么说，其实不太明显，简单算一下。假设每个句子含有20个单词，那么十亿个单词对应就是五千万个句子，换句话讲在多核CPU的条件下，一分钟左右可以训练500万个句子。

和Bert比较一下，在GPU条件下，8个小时训练300万条数据左右。相比之下Fasttext的这个速度是真的太快了。

在这个论文中，也就是使用做文本分类的Fasttext，使用的是CBOW的架构。

注意哦，强调一遍，Fasttext用在文本分类，模型架构使用的是CBOW的变种。（我这句话的意思不是说使用Skip-gram不可以，而是CBOW在理解文本分类的时候更加的容易理解）

这里和Word2vec的CBOW有两个区别：

1. 第一，使用类别标签替换了中心词。
2. 第二，使用句子中所有单词作为输入，而不再是单单的针对滑动窗口中的单词。

这两个点如果我们自己考虑，也很容易想到。

为什么这么说呢？先说第二点。我现在要做的是针对文本进行分类，所以对于我的输入需要转换到整体这个句子来看，才能使对一个句子的特征表达。

再说第一点，我们知道在Wrod2vec中，我们使用的是中心词作为输出，而且使用了霍夫曼作为输出层。

非叶子点上的向量为了我的二分类提供计算，叶子节点为整个词汇表中所有词汇的向量。两个向量都会随着模型而训练。

如果要做分类，我们可以想一下叶子节点和非叶子节点的变化。

首先叶子节点对应的是所有类别。如果说我们的类别有5000个，那么对应到Word2vec，我们就有着5000个词汇。想一下是不是这么对应。

非叶子节点其实没什么变化，因为它没有什么实际含义，只是为二分类提供计算。

在这里还想说一下，word2vec中的叶子节点也就是词向量更新之后我们最后是要的，但是对于fasttext其实不会用到这个，因为我们是对文本进行分类，只需要保存了模型权重在预测的时候可以预测就可以了。

还想谈一下词向量初始化的问题，模型训练开始的时候，词向量随机初始化就可以，模型训练结束之后，我们在预测阶段直接使用这个词向量就可以（就是随着模型训练而更新的这个词向量）。

对这个论文还有一个很有意思的点，就是N-gram就是fasttext的模型的输入不仅仅针对的是每个单词，为了加入词序信息，还加入了n-gram信息。

需要注意的一个细节是，这里的n-gram针对的是word，而不是char。对应到中文，应该对应的是分词之后的词，而不是字。但是我自己认为这么对应过来不太好理解。

中文的字做n-gram貌似也有词序信息。但是英文的char-level的n-gram很难说针对这个句子提供一个语序信息。大家理解一下就好。

还有一个问题想说一下，使用了n-gram信息之后，词表肯定是变大了的。你需要的n的内容越多，比如你想要n=1orn=2orn=3等等吧，n的取值范围越大，你的词表越大。

这就会出现一个问题训练非常缓慢。这点很容易理解，参数越多训练当然越慢。

针对这个问题，怎么解决呢？使用哈希。

我举个简单的例子，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，不知道大家有咩有想到和albert很类似。哈希这个过程我自己感觉有点共享参数的意思。

11. Fasttext解读-获取词向量

这个文章主要是谈一下Enriching Word Vectors with Subword Information 这个论文。

有了上一个文章的打底，（上一个文章点击这里）这个论文理解起来就比较简单，所以我写的也比较短。

对于这个论文，我先给出它最核心的部分： 使用负采样的skip-gram的基础上，将每个中心词视为子词的集合，并学习子词的词向量。

这句话涉及到的一个最关键的部分就是子词subword，也是这个论文的核心。

举个例子，现在我们的中心词是"where"，设定子词大小为3，那么子词集合分为两个部分，注意是两个部分。

第一部分形如这样：“<wh”，“whe”，“her”，“ere”，“re>”，第二部分就是特殊子词，也就是整词“”。

那么对应到模型是，原来我的输入是“where”的词向量，现在在Fasttext就是所有子词的词向量的和。

注意哦，这里是所有子词，是包含特殊子词，也就是整词的。

对于背景词，直接使用整词就可以。

简单来说，就是输出层使用子词（普通子词加上整词），输出层使用整词。

如果遇到了OOV怎么办？使用普通子词的向量和来表示就可以。

其实这里的子词，在名字上和上一个文章的ngram很类似，不过，这里使用的是就char的n-gram，缓解的问题并不是语序，而是利用了词序形态的规律。

对应到中文，其实就是偏旁部首。 我记得阿里好像有发一个关于fasttext的中文版本，训练的就是偏旁部首。大家有兴趣可以去看一看。

写完了，我对两个文章做个小总结，顺便对文章开头的问题做个回答: fasttext 训练词向量的时候一般是使用Skip-gram模型的变种。在用作文本分类的时候，一般是使用CBOW的变种。

在这里，我想要强调一下，上一段我说的是一般情况，是为了方便大家了解，并不代表说CBOW架构不能训练词向量，skip-gram不能用作文本分类，需要注意这一点哦。

12. Glove细节详解解读

本文大概需要阅读 4.75 分钟 先问大家两个问题，看能不能解答

1. Glove 中词向量的表达是使用的中心词向量还是背景词向量还是有其他方法?
2. 能不能分别用一句话概括出Glove和Fasttext 的核心要点?

先来谈Glove。中文全称 Global Vectors for Word Representation。它做的事情概括出来就是：基于全局语料，获得词频统计，学习词语表征。

我们从语料之中，学习到X共现词频矩阵，词频矩阵中的每个元素x_{ij}，代表的是词

x_j出现在x_i的环境中的次数。注意，对于共现词频矩阵来说，它是一个对称矩阵。

这一点非常的重要，也很容易理解，词A出现在词B周围的次数肯定是等价于词B出现在词A周围的次数的。

类比于Word2vec，对于词x_i，就是中心词，对于词x_j也就是背景词。

理论上，一个词作为中心词向量和一个词作为背景学到的两种向量应该是完全相同的。

但是现实中，由于我们初始化的不同，所以我们最终学习到的两种词向量是有些许不同。

为了增加模型的鲁棒性，在Glove中，使用两个词向量的和作为我们一个词的词向量的表达。

这一点是区别于Word2vec，对于Word2vec，中心词向量和背景词向量是不等价的，我们一般使用中心词向量代表一个词最终的语义表达。

Glove 论文中的推导过程其实不是很严谨，大致流程就是从大语料中发现了一个规律，即条件概率的比值可以比较直观的表达出词与词之间的关系。

随后可以构建词向量函数去拟合条件概率的比值。基于此一步步的进行延伸推导，在我看了就是在基于一些假设，寻找出一种可能性的存在。

在这里就不细说，直接给出Glove的损失函数：

$\sum_{i \in V} \sum_{j \in V} h(x_{ij})(u_j^T v_i + b_i + b_j - log(x_{ij}))^2$

详细讲一下这个损失函数，它是一个平方损失函数，很值得琢磨。

我把它分为了三个部分，权重函数h(x),词向量表达是$u_j^T v_i + b_i + b_j$，共现词频的对数 $log(x_{ij})$

从这里，我插一句我的理解，就是GLove基于的是无标签的语料，属于无监督的训练过程，但是从损失函数看，我觉得可以认为它是一个有监督的过程。

标签就是$log(x_{ij})$，这个是我们从语料之中计算出来的，换句话说，在模型训练之前，我们可以对语料的计算，得到相应的标签数据，所以我自己认为这个可以看做是一个有监督的过程。

我们使用一句话去描述这个损失函数可以这么去说：随着模型不停的优化，词向量的表达在不断的拟合共现词频的对数。

h(x)是权重函数，代表的含义是表达一个词对的重要性，在值域[0,1]上单调递增。直观上理解就是一对词语共现的词频越多，那么它的重要性也就越大。

论文中给出的函数是这样的，在x<c(比如c=100)的情况下，h(x)=(x/c)^\alpha (\alpha=0.75)，在其余情况下，h(x)=1。也就是重要度不能无限增大，给了一个上限。

我们看这个损失函数，有一个很有意思的点，就是h(0)=0。想一下，这个代表着什么？ 也就是说，当我们一对词的共现词频为0的时候，损失函数是为0，换句话讲，我们的损失函数的复杂度是和共现词频矩阵中的非零元素是线性关系。

所以在训练的时候，我们只需要对非零元素采样，使用随机梯度就可以对词向量和两个偏置项进行学习更新。

这里还有一个细节点，需要注意，偏置项是不可以省略的。为什么呢？因为如果去掉，在公式推导中的对称性假设就不满足，感兴趣的同学可以自己推导一系。