机器学习与深度学习面试系列十七（Embedding初步）

什么是表示?

为了提高机器学习系统的准确率，我们需要将输入信息转换为有效的特征，这就是：表示。一般而言，一个好的表示具有以下几个优点:

1. 一个好的表示应该具有很强的表示能力，即同样大小的向量可以表示更多信息。
2. 一个好的表示应该使后续的学习任务变得简单，即需要包含更高层的语义信息。
3. 一个好的表示应该具有一般性，是任务或领域独立的。虽然目前的大部分 表示学习方法还是基于某个任务来学习，但我们期望其学到的表示可以比较容易地迁移到其他任务上。

局部表示和分布式表示?

以颜色表示为例 ，如果要在计算机中表示颜色，**局部表示**的方法是以不同名字来命名不同的颜色，也称为离散表示或符号表示。局部表示通常可以表示为one-hot向量形式。one-hot向量为有且只有一个元素为 1，其余元素都为0的向量。另一种表示颜色的方法是用 RGB 值来表示颜色，不同颜色对应到 R、G、B 三 维空间中一个点，这种表示方式叫做**分布式表示**。分布式表示叫做分散式表示可能更容易理解，即一种颜色的语义分散到语义空间中的不同基向量上。分布式表示通常可以表示为低维的稠密向量。

颜色	局部表示	分布式表示
琥珀色	$[1, 0, 0, 0]^T$	$[1.00, 0.75, 0.00]^T$
天蓝色	$[0, 1, 0, 0]^T$	$[0.00, 0.5, 1.00]^T$
中国红	$[0, 0, 1, 0]^T$	$[0.67, 0.22, 0.12]^T$
咖啡色	$[0, 0, 0, 1]^T$	$[0.44, 0.31, 0.22]^T$

颜色的表示

局部表示优点:

1. 这种离散的表示方式具有很好的解释性，有利于人工归纳和总结特征，并通过特征组合进行高效的特征工程。
2. 通过多种特征组合得到的表示向量通常是稀疏的二值向量，当用于线性模型时计算效率非常高。

局部表示不足:

1. one-hot 向量的维数很高且不能扩展。如果有一种新的颜色，我们就需要增加一维来表示。

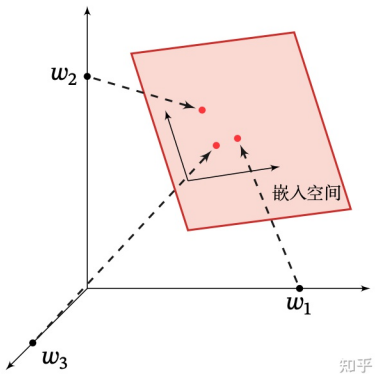
2. 不同颜色之间的相似度都为0（所以基向量都是正交的），即我们无法知道“红色”和“中国红”的相似度要高于“红色”和“黑色”的相似度。

分布式表示改进：

1. 分布式表示的向量维度一般都比较低。我们只需要用一个三维的稠密向量就可以表示所有颜色。
2. 分布式表示也很容易表示新的颜色名。
3. 不同颜色之间的相似度也很容易计算。

嵌入（Embedding）是什么?

我们可以使用神经网络来将高维的局部表示空间 $R^{|\mathcal{V}|}$ 映射到一个非常低维的分布式表示空间 $R^{|\mathcal{D}|}$ ，并且 $|\mathcal{D}| \ll |\mathcal{V}|$ 。 在这个低维空间中，每个特征不再是坐标轴上的点，而是分散在整个低维空间中。在机器学习中，这个过程称为嵌入(Embedding)。嵌入通常指将一个度量空间中的一些对象映射到另一个低维的度量空间中，并尽可能保持不同对象之间的拓扑关系。比如自然语言中词的分布式表示，也经常叫做词嵌入(Word Embedding)。推广开来，还有Sentence Embedding, Image Embedding, Video Embedding, Item Embedding都是一种将源数据映射到另外一个空间。下面我们重点介绍 Word Embedding。



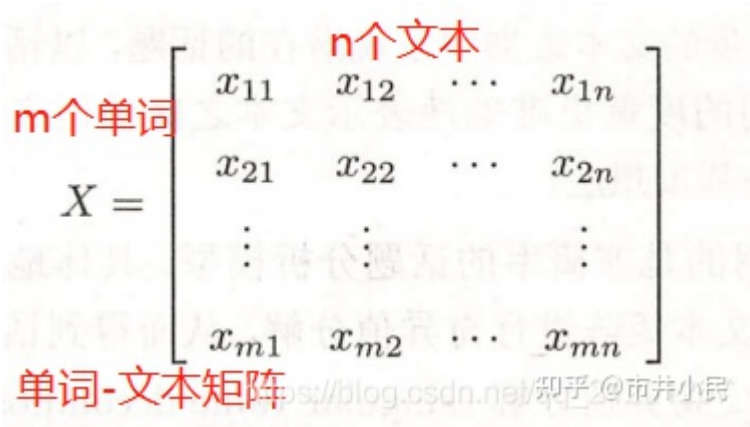
one-hot 向量空间与嵌入空间

常见的词嵌入（Word Embedding）模型有哪些?它们有什么联系和区别?

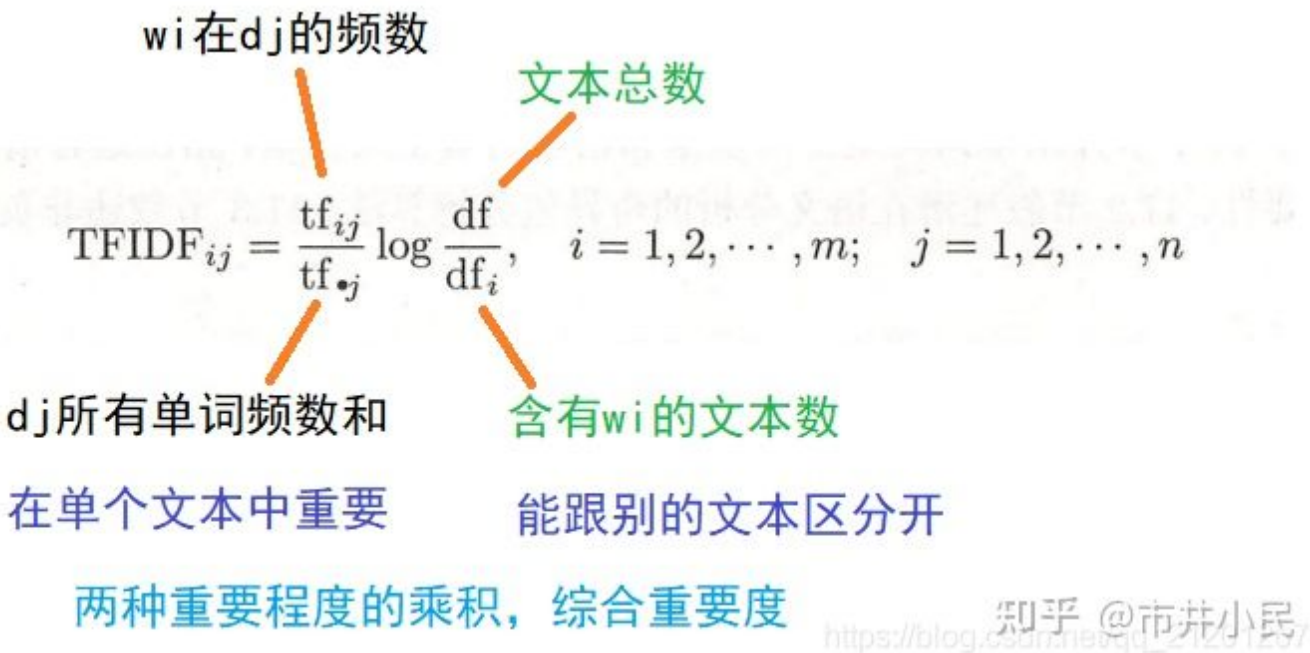
词嵌入模型基于的基本假设（分布式假设）是出现在相似的上下文中的词含义相似，以此为依据将词从高维稀疏的独热向量映射为低维稠密的连续向量， 从而实现对词的语义建模。词嵌入模型大致可以分为两种类型，一种是基于上下文中词出现的频次信息，包括：潜在语义分析(Latent Semantic Analysis，LSA)， GloVe等。另一种则是基于对上下文中出现的词的预测，包括：Word2Vec，FastText等。

潜在语义分析（LDA）是怎样的?

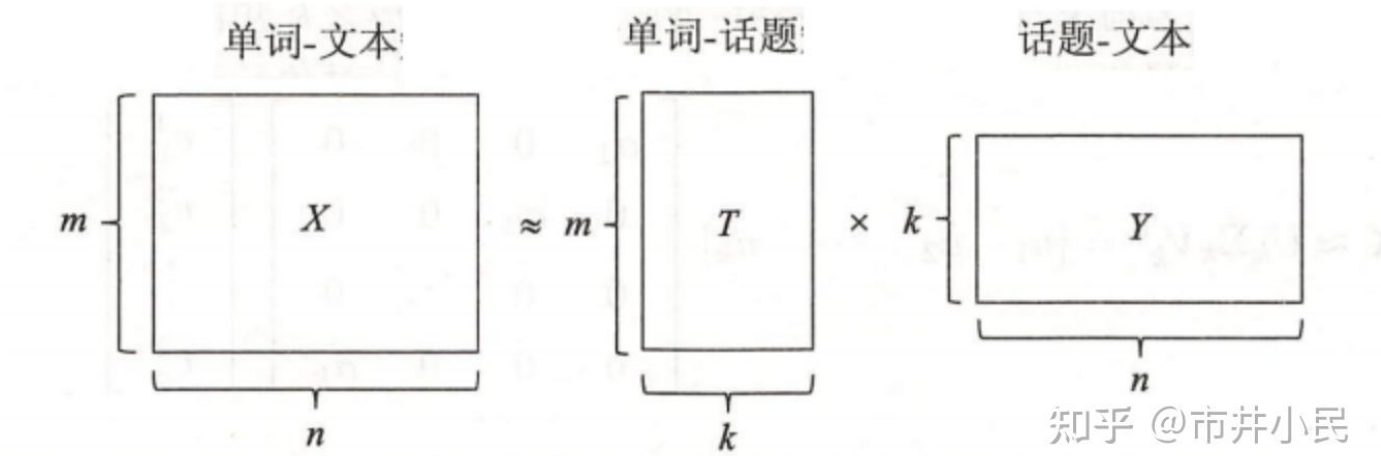
首先LDA使用单词向量空间矩阵对文本的语义内容进行表示。它的基本想法是，文本中所有单词的出现情况表示了文本的语义，所以我们可以用一个向量表示文本的“语义”，向量的每一维对应一个单词，其数值为该单词在该文本中出现的频数或权值。



其中，权值常用单词频率-逆文本频率（term frequency-inverse document frequency，TF-IDF）表示：



再考虑使用话题向量空间矩阵来对文本进行表示。它的基本想法是一个文本一般含有若干个话题，话题由若干个语义相关的单词表示。话题的个数通常远远小于单词的个数。那么单词向量空间矩阵和话题向量空间矩阵有什么关系呢？下图很好的诠释了LDA的思想：



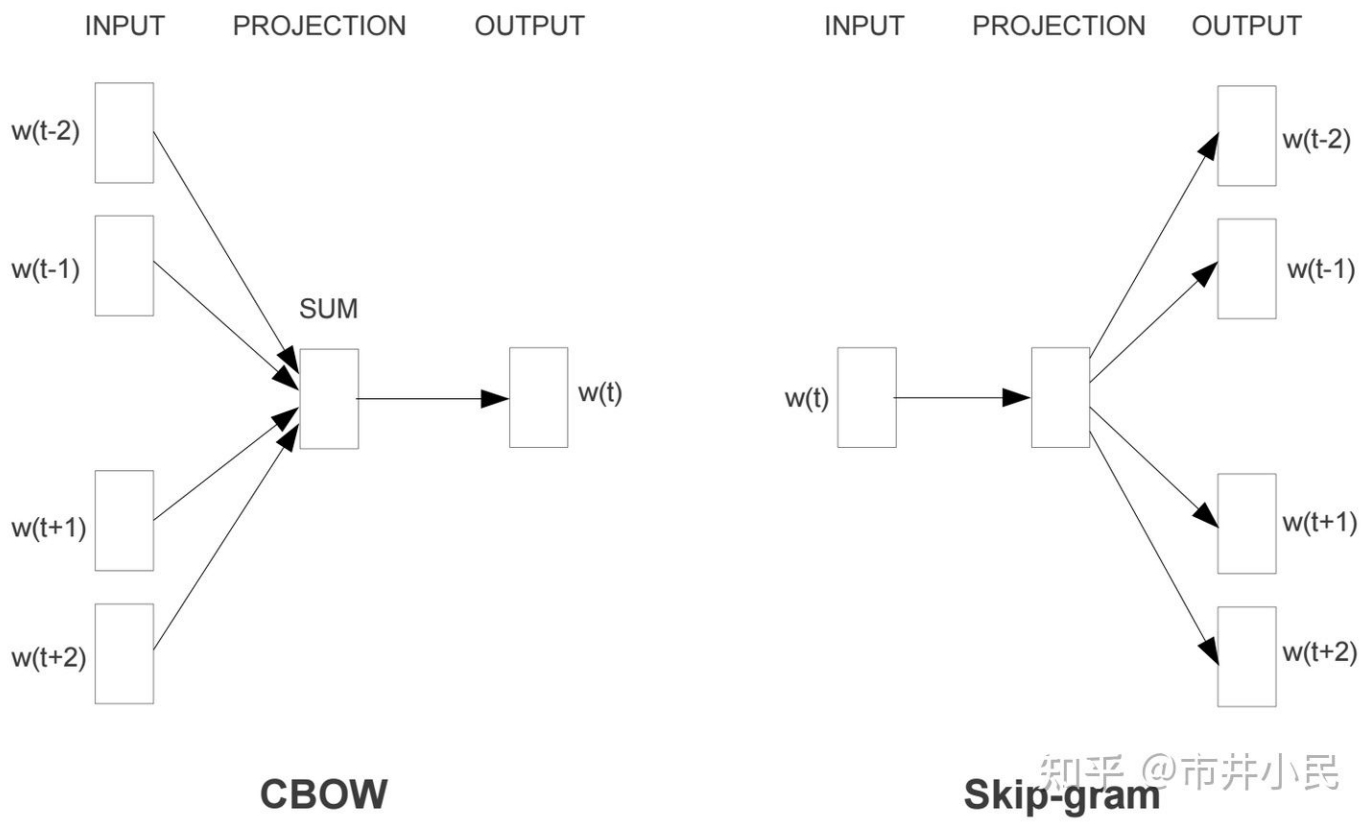
我们可以看到，单词-文本矩阵X 可以分解为 单词-话题矩阵T 和 话题-文本矩阵Y 的乘积，$\boldsymbol{X}=\boldsymbol{TY}$ 。如果我们能得到这个矩阵分解的结果，那么矩阵T 中的每一行都可以看作是对一个单词的向量表示（在话题空间中，而话题空间的维度k远小于文本维度n，是一种词嵌入思想）。要想做这个矩阵分解，可以使用截断SVD分解或者非负矩阵分解。

LDA的缺点：

1. SVD计算非常的耗时，尤其是我们的文本处理，词和文本数都是非常大的，对于这样的高维度矩阵做奇异值分解是非常难的。
2. 主题值的选取对结果的影响非常大，很难选择合适的k值。
3. LSA 缺乏统计基础，结果难以直观的解释。

Word2Vec模型是怎样的?

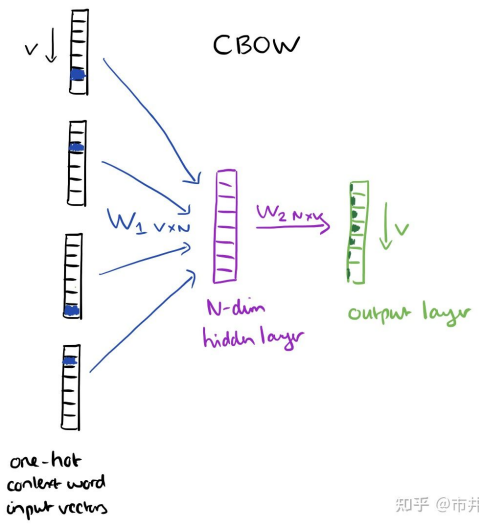
Word2Vec是在2013年被Google提出的，极具讽刺意味的是原始论文有着9700多次引用，却被ICLR 2013拒绝后从未正式发表。它提出了一对常用的模型：Skip-Gram（用一个词语作为输入，来预测它周围的上下文） 和 CBOW（用一个词语的上下文作为输入来预测这个词语本身），它们都是浅层的两层神经网络，但经历了大量数据的训练。



Word2Vec模型实际上分为了两个阶段，第一阶段建立语言模型，第二部分是通过模型获取词向量。Word2Vec的整个建模过程实际上与自编码器（auto-encoder）的思想很相似，即先基于训练数据构建一个神经网络，当这个模型训练好以后，我们并不会用这个训练好的模型处理新的任务，我们真正需要的是这个模型通过训练数据所学得的参数，即隐层的权重矩阵（word vectors）。业内戏称之为“Fake Task”，意味着建模并不是我们最终的目的。

Word2Vec这两个模型在隐藏层都不使用非线性激活函数，因为Word2Vec不是为了做语言模型，它不需要预测得更准。不使用非线性激活函数可以使网络求梯度更简单，加快网络训练。

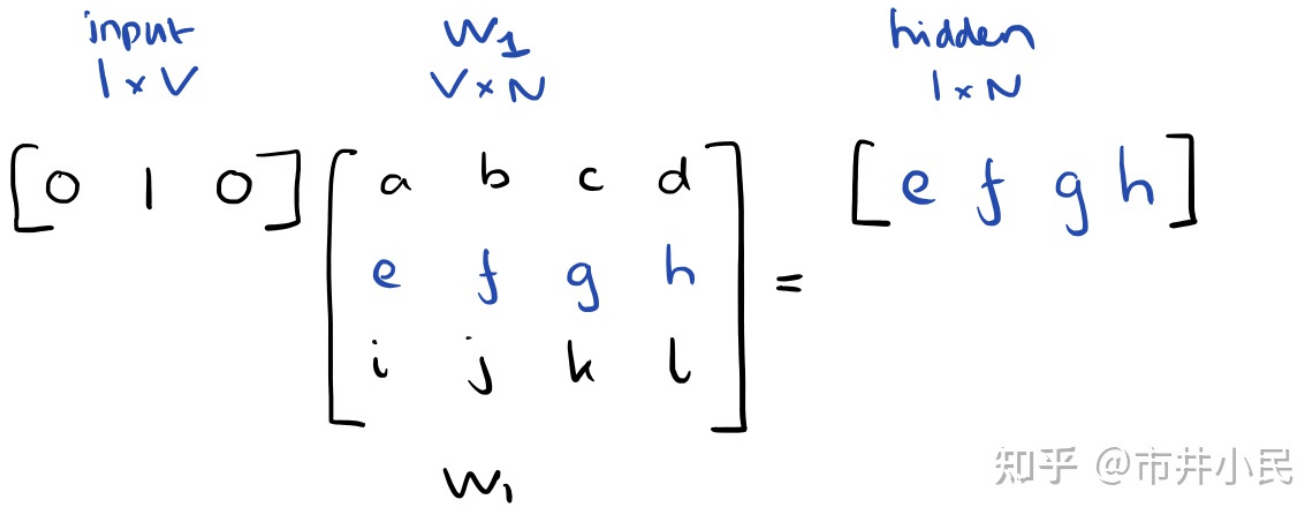
CBOW模型是怎样的?



输入上下文单词的one-hot编码（假设单词向量空间dim为V）所有one-hot分别乘以共享的输入权重矩阵 $\boldsymbol{W_1}$ ，所得的向量相加求平均得到1*N隐层向量。再乘以输出权重矩阵 $\boldsymbol{W_2}$ 得到 1*V 输出向量。然后将模型视作一个V分类问题，输出向量就是V个logits，利用softmax函数处理得到V-dim概率分布，概率最大的index所指示的单词为预测出的中间词（target word）。最后使用交叉熵损失函数对预测结果进行评估，使用梯度下降优化参数矩阵 $\boldsymbol{W_1}, \boldsymbol{W_2}$ 。[1]

举例来说，假设我们现在的训练语料是只有四个单词的：I drink coffee everyday。我们选coffee作为中心词，window size设为2。也就是说，模型的输入是：["I","drink"和"everyday"]三个单词的one-hot编码，期望输出是coffee。

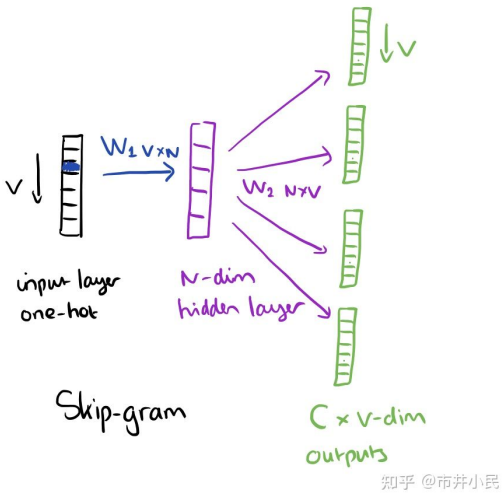
由于输入的单词进行了one-hot编码，而庞大的语料库中单词的数量是极大的，one-hot编码会浪费很多内存。我们可以用Lookup Table来优化从输入到隐藏层这个计算，其实这个Lookup Table就是 $\boldsymbol{W_1}$ 参数矩阵。由于输入是one-hot向量，所以输入向量乘以 $\boldsymbol{W_1}$ 矩阵，就是选择了 $\boldsymbol{W_1}$ 矩阵中对应one-hot中为1的index的那一行。



对于K个输入one-hot，即分别选择了 $\boldsymbol{W_1}$ 中对应的K行，然后求和即可。后来 word2vec 的工具包中将求和改进为求平均值，就是对求和结果除以K。

Skip-Gram模型是怎样的?

Skip-Gram 模型与 CBOW 模型相反，它以中心词作为单个输入向量，输出预测的目标上下文词。



训练目标是最小化输出层中所有上下文词的预测误差总和。 回到上面的例子，在Skip-Gram 模型中输入是coffee的one-hot编码，期望输出是 ["I","drink"和"everyday"]。值得注意的一点是，上面这幅图只是一个示意图，不是神经网络架构图。真实的网络结构输出层应该只有一个 1*V 的输出向量。然后我们分别计算"I","drink"和"everyday"三个单词对应的损失，将损失加和，然后使用反向传播进行优化。同时也注意，这暗含着 "I","drink"和"everyday" 三个单词，虽然距离中心词的距离不同，但是我们同等的对待它们。[2]

CBOW模型和Skip-Gram模型比较?

1. CBOW 训练速度比 Skip-Gram 更快，对同样一个样本窗口，CBOW 模型计算一次梯度，而 Skip-Gram 模型计算窗口大小次数的梯度。原文中同一个任务，CBOW训练花费几小时，而 Skip-Gram 花费了3天。
2. Skip-Gram 模型由于对低频词迭代更充分，因此对低频词的表示通常优于 CBOW 模型。例如，给定一个上下文，yesterday was a really [...] day。CBOW模型将会预测更高频的词汇如 beautiful或者nice, 像低频词delightful就会很少被模型关注到，因为 CBOW 模型设计来预测中心词，在这个场景下，delightful和beautiful是竞争关系，而大部分语料都会将模型往beautiful方向进行训练。但是 Skip-Gram 模型是设计用来预测上下文的，当给定delightful的时候，yesterday was a really [...] day就是它最大概率的上下文，delightful和beautiful之间不存在竞争关系，beautiful + context 被模型视作一个新的样本而已。[3]
3. CBOW 更好的学习单词之间的句法关系，而 Skip-gram 可以更好地捕捉更好的语义关系。例如，这意味着对于单词“cat”，CBOW 将检索词形相似的词，如复数形式“cats”，而 Skip-gram 会考虑词形上不同的词（但语义相关），如“dog”。

Word2Vec 为什么要用两个权重矩阵?

这个做法是遵循原文献的，原文没有给出动机。 一种可能的原因是：考虑单词 dog 和上下文 dog 共享相同向量 v 的情况。单词几乎不会出现在它们自身的上下文中，因此模型会为 $p(dog|dog)$ 分配一个极低的概率，这显然是违背我们直觉的。[4]

Word2Vec训练有哪些tricks?

Word pairs and "phases"

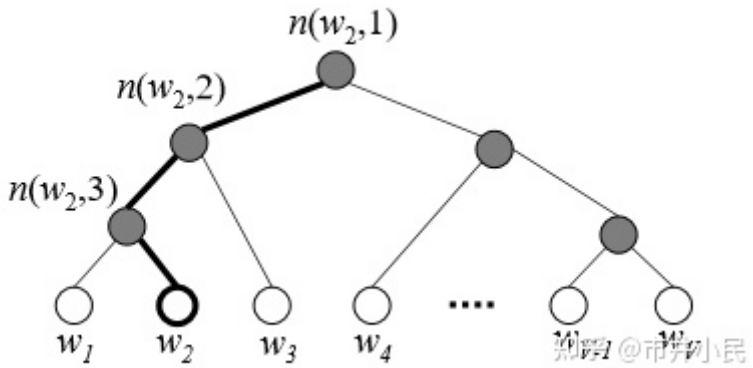
一些单词组合（或者词组）的含义和拆开以后具有完全不同的意义。比如“Boston Globe”是一种报刊的名字，而单独的“Boston”和“Globe”这样单个的单词却表达不出这样的含义。因此，在文章中只要出现“Boston Globe”，我们就应该把它作为一个单独的词来生成其词向量，而不是将其拆开。同样的例子还有“New York”，“United Stated”等。

Sub-sampling

对于“the”这种常用高频单词，我们将会有大量的（"the“，...）这样的训练样本，而这些样本数量远远超过了我们学习“the”这个词向量所需的训练样本数。例如：("fox", "the") 这样的训练样本并不会给我们提供关于“fox”更多的语义信息，因为“the”在每个单词的上下文中几乎都会出现。Sub-sampling就是对于我们在训练原始文本中遇到的每一个单词，它们都有一定概率被我们从文本中删掉，而这个被删除的概率与单词的频率有关。 删除单词 w_i 的概率为： $P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$ ，其中 $f(w_i)$ 为单词 w_i 出现的频率，t是阈值，通常取 10^{-5} 。

Hierarchical softmax

层次softmax的提出，是为了解决softmax函数在语料库单词量过大时，计算过于复杂的问题。当语料库中单词数量为 V 时，计算softmax的时间复杂度为 $O(V)$ 。



首先构造一颗二叉树，将原始字典D划分为两个子集D1和D2，然后对子集D1和D2进一步划分。重复这一过程直到集合中只剩下一个word。这样就将原始大小为V的字典D转换成一棵深度为logV的二叉树。树的叶节点与原始字典里的word一一对应，数量为V；非叶子节点对应着某一类word的集合，数量为V-1。

对于训练样本里的一个target word w_t ，其对应二叉树编码为 $\{1, 0, 1, \ldots, 1\}$ ，则构造的似然函数为： $p(w_t|context) = p(D_1 = 1|context)p(D_2 = 0|D_1 = 1)\ldots p(w_t|D_k = 1)$ 。其中每一项乘积因子都是一个逻辑回归函数。

可以通过最大化这个似然函数来求解二叉树的参数，即非叶子节点上的向量，用来计算游走到某一子节点的概率。Hierarchical softmax通过构造一棵二叉树将目标概率的计算复杂度从最初的V降低到了logV的量级。但是却增加了词与词之间的耦合性。比如一个word出现的条件概率的变化会影响到其路径上所有非叶子节点的概率变化。间接地对其他word出现的条件概率带来影响。并且如果使用哈夫曼树的话，对于一些生僻词，搜索的深度将会很深。

Negative Sampling

比如我们有一个训练样本，中心词是 w_c ,它周围上下文共有2c个词，记为context(w)。由于这个中心词 w_c 的确和context(w)相关存在，因此它是真实的正例。通过负采样(Negative Sampling)，我们得到neg个和 w_c 不同的中心词 w_i 。这样context(w)和 w_i 就组成了neg个并不真实存在的负例。利用这一个正例和neg个负例，我们进行二元逻辑回归，得到负采样对应每个词 w_i 对应的模型参数和每个词的词向量。对于小规模数据集，选择5-20个negative words会比较好，对于大规模数据集可以仅选择2-5个negative words。

一个单词被选作negative sample的概率跟它出现的频次有关，出现频次越高的单词越容易被选作negative words。

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n}(f(w_j)^{3/4})}$$

每个单词被赋予一个权重，即 $f(w_i)$ ， 它代表着单词出现的频次。公式中开3/4的根号是基于经验的，原文中提到这个公式的效果要比其它公式更加出色。[5]

Glove模型是怎样的?

对比Word2Vec模型是对一个滑动窗口内上下文词或者中心词进行预测，其输出本质是局部空间中（滑动窗口内）单词同时出现的概率分布。GLove模型认为全局单词同时出现的概率的比率能够更好地区分单词。例如，假设我们要表示“冰”和“蒸汽”这两个单词。对于和“冰”相关，和“蒸汽”无关的单词，比如“固体”，我们可以期望 $\frac{P(冰|固体)}{P(蒸汽|固体)}$ 值较大。类似地，对于和“冰”无关，和“蒸汽”相关的单词，比如“气体”，我们可以期望 $\frac{P(冰|气体)}{P(蒸汽|气体)}$ 较小。相反，对于像“水”之类同时和“冰”、“蒸汽”相关的单词，以及“时尚”之类同时和“冰”、“蒸汽”无关的单词，我们可以期望 $\frac{P(冰|水)}{P(蒸汽|水)}$ 、 $\frac{P(冰|时尚)}{P(蒸汽|时尚)}$ 应当接近于1。这就是Glove的朴素想法。至于式子中的各个概率P，都是可以根据语料直接计算得到的。

中心词	背景词
我	老，爱
老	我，爱，我
爱	我，老，我，老
我	老，爱，老，婆
老	爱，我，婆，了
婆	我，老，了
了	老，婆

	我	老	爱	婆	了
我	0	3	2	1	0
老	3	0	2	1	1
爱	2	2	0	0	0
婆	1	1	0	0	1
了	0	1	0	1	0

$$P_{i,j} = \frac{X_{i,j}}{\sum_{j=0}^{V} X_{i,j}}$$

	我	老	爱	婆	了
我	0	0.5	0.33	0.16	0
老	0.42	0	0.28	0.14	0.14
爱	0.5	0.5	0	0	0
婆	0.33	0.33	0	0	0.33
了	0	0.5	0.5	0	0

对于任意两个单词i和j，记他们的词向量表达为 w_i 和 u_j ，则它的代价函数为：$J = f(X_{ij})(w_i^T u_j + bw_i + bu_j — log(X_{ij}))^2$ 。这个代价函数的推导来源于物理博士出身的作者很多直觉，可以参考这篇文章[6]。

Glove模型和Word2Vec模型比较?

1. Word2Vec每次都是用一个窗口中的信息更新出词向量，但是Glove则是用了全局的信息（共现矩阵），也就是多个窗口进行更新。
2. Word2Vec是基于“predictive”的模型，而GloVe是“count-based”的模型。
3. 相比于Word2Vec，因为GloVe更容易并行化，所以速度更快。但因为要存全局信息，所以内存开销更大。

FastText模型和CBOW模型区别?

FastText在模型结构上和CBOW几乎是一样的，只是把预测中心词换成了预测类别，所以FastText是监督学习，需要标注语料类别，而CBOW是无监督学习。CBOW是将window_size的上下文词的向量做平均，提取的是上下文词特征，而FastText是将一个句子所有词做平均，提取的是句子的特征。FastText可以看作是对CBOW的一种拓展。

两点改进：

1. Subword n-gram。对于一个单词“google”，为了表达单词前后边界，我们加入<>两个字符，即变形为“<google>”。假设我们希望抽取所有的tri-gram信息，可以得到如下集合：G = { <go, goo, oog,ogl, gle, le>}。在实践中，我们往往会同时提取单词的多种n-gram信息，如2/3/4/5-gram。这样，原始的一个单词google，就被一个字符级别的n-gram集合所表达。这种操作对低频词/罕见词 和 一些在词典中都没有出现过的词或者带有某些拼写错误的词会有比较好的效果。
2. Hierarchical Softmax。这点和我们在word2vec中提到的层次softmax基本一样，只是将在word2vec中，树是基于单词建立的，这里的树是基于文本分类建立的，因为两者预测目标不同。

从静态Word Embedding到动态Word Embedding?

上面提及的这些做法是2018年以前业内采用预训练的典型做法，Word Embedding其实对于很多下游NLP任务是有帮助的，但是效果并没有期待中那么好。多义词问题是笼罩了Word Embedding头上好几年的乌云。比如多义词Bank，有两个常用含义，但是Word Embedding在对bank这个单词进行编码的时候，是区分不开这两个含义的，因为它们尽管上下文环境中出现的单词不同，但是在用语言模型训练的时候，不论什么上下文的句子经过word2vec，都是预测相同的单词bank，而同一个单词占的是同一行的参数空间，这导致两种不同的上下文信息都会编码到相同的word embedding空间里去。我们把这些早期不能区分多义词的词向量模型称作静态Word Embedding方法，后来陆续提出了ELMO、GPT、Bert等动态Word Embedding方法，它们根据上下文单词的语义去调整单词的Word Embedding表示，这样经过调整后的Word Embedding更能表达在这个上下文中的具体含义，自然也就解决了多义词的问题了。

万物皆Embedding?

Embedding的本质其实就是将一个高维空间的表达转换为一个低维的同构表达，使相似的东西在低维空间有着较高的相似度。理解了这一思想的基础上，我们可以发现，除了Word Embedding以外，其实很多类型的数据都可以用Embedding来进行处理，如：Sentence Embedding、Image Embedding、Item Embedding、Graph Embedding等。这些方法在很多跨模态的场景如Image caption，相关性搜索和推荐系统中得到了大量优秀的表现。我还在持续学习中。

参考

1. ^ https://www.zhihu.com/question/44832436/answer/266068967
2. ^ https://towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314
3. ^ https://stackoverflow.com/questions/38287772/cbow-v-s-skip-gram-why-invert-context-and-target-words
4. ^ https://arxiv.org/pdf/1402.3722v1.pdf
5. ^ https://zhuanlan.zhihu.com/p/27234078
6. ^ https://towardsdatascience.com/light-on-math-ml-intuitive-guide-to-understanding-glove-embeddings-b13b4f19c010#:~:text=GloVe%20is%20a%20word%20vector,by%20solving%20three%20important%20problems.&text=%2C%20and%20k