Word2vector

word2vec的作用是将所有词语投影到K维的向量空间，每个词语都可以用一个K维向量表示。由于它简洁，高效的特点，引起了人们的广泛关注，并应用在很多NLP任务中，用于训练相应的词向量。

# 词向量的表示方式

## 传统的词表示--one-hot Representation

One-hot representation把每个词表示为一个向量。这个向量的维度是词表大小，只有一个维度的值为 1，这个维度就代表了当前的词。

这样的表示方法简单容易理解，而且编程也很容易实现，只需要取对应的索引就能够完成，已经可以解决相当一部分NLP的问题，但是仍然存在不足，即词向量与词向量之间都是相互独立的；我们都知道，词与词之间是有一定的联系的，我们无法通过这种词向量得知两个词在语义上是否相似，并且如果词表非常大的情况下，每个词都是茫茫 0。海中的一个 1，这种高维稀疏的表示也有可能引发维度灾难。为了解决上述问题，就有了词向量的第 二种表示方法。

假设一个词典，独热编码用一个高维度的向量来表示词典中的每个词，向量的维度为词 典的大小，向量的分量只有一个 1，其他全为 0， 1 的位置对应该词在词典中的位置。

比如：词典是{ 'I', 'have', 'an', 'apple' }，那么词典中各个单词的one-hot表示如下：

|  |
| --- |
| V(I) = [ 1, 0, 0, 0 ]  V(have) = [ 0, 1, 0, 0 ]  V(an) = [ 0, 0, 1, 0 ]  V(apple) = [ 0, 0, 0, 1 ] |

可以看到one-hot 编码非常简洁，就是一串比特串，而且仅有一个分量为1。

如果词典非常大的，那么 one-hot 维度将是非常庞大的，在DL一些算法中很容易产生维数灾难。而且 one-hot 编码可以认为没有距离的概念，很难发掘词与词之间的关系。

One-hot编码就是把每个样本中的单个特征用一个向量表示，该向量只有1个维度是1，其他的都是0。

One-hot的编码过程不需要学习和训练。但是One-hot编码的向量是高维度且极其稀疏的，大量的位置为0，计算效率较低，同时也不利于神经网络的训练。从语义角度来讲，One-hot 编码还有一个严重的问题，它忽略了单词先天具有的语义相关性。比如，对于单词“like”、“dislike”、“Rome”、“Paris”来说，“like”和“dislike”在语义角度就强相关，它们都表示喜欢的程度；“Rome”和“Paris”同样也是强相关，他们都表示欧洲的两个地点。对于一组这样的单词来说，如果采用One-hot编码，得到的向量之间没有相关性，不能很好地体现原有文字的语义相关度。

其缺点是：（1）向量的维度会随着句子的词的数量类型增大而增大；如果将世界所有城市名称对应的向量合为一个矩阵，那这个矩阵过于稀疏，并且会造成维度灾难。（2）编码是随机的，向量之间相互独立，无法表示语义层面上词汇之间的相关信息。

所以，提出了对独热编码的改进方法：（1）将vector每一个元素由整形改为浮点型，变为整个实数范围的表示；（2）转化为低维度的连续值，也就是稠密向量。将原来稀疏的巨大维度压缩嵌入到一个更小维度的空间。并且其中意思相近的词将被映射到向量空间中相近的位置。

简单说，要寻找一个空间映射，把高维词向量嵌入到一个低维空间。然后就可以继续处理。

## 分布式词向量编码

分布式编码（Distributed representation）最大的贡献就是让相关或者相似的词，在距离上更接近了。向量的距离可以用传统的欧氏距离来衡量，也可以用余弦相似度来衡量。Word2vec属于分布式词向量的一种。

将word映射到一个新的空间中，并以多维的连续实数向量进行表示叫做“Word Represention”或“Word Embedding”。自从21世纪以来，人们逐渐从原始的词向量稀疏表示法过渡到现在的低维空间中的密集表示。用稀疏表示法在解决实际问题时经常会遇到维数灾难，并且语义信息无法表示，无法揭示word之间的潜在联系。而采用低维空间表示法，不但解决了维数灾难问题，并且挖掘了word之间的关联属性，从而提高了向量语义上的准确度。

# 词向量模型类型

## LSA矩阵分解模型

采用线性代数中的奇异值分解方法，选取前几个比较大的奇异值所对应的特征向量将原矩阵映射到低维空间中，从而达到词矢量的目的。这种方法去除了原始向量空间中的一些“噪音”，但它无法保存词与词之间的linear regularities。

## PLSA潜在语义分析概率模型

从概率学的角度重新审视了矩阵分解模型，并得到一个从统计，概率角度上推导出来的和LSA相当 的词矢量模型。

## LDA文档生成模型

按照文档生成的过程，使用贝叶斯估计统计学方法，将文档用多个主题来表示。LDA不只解决了同义词的问题，还解决了一次多义的问题。目前训练LDA模型的方法有原始论文中的基于EM和差分贝叶斯方法以及后来出现的Gibbs Samplings 采样算法。

## Word2Vector模型

Word2vector通过神经网络机器学习算法来训练N-gram语言模型，并在训练过程中求出word所对应的vector的方法。

# Word2Vector理论

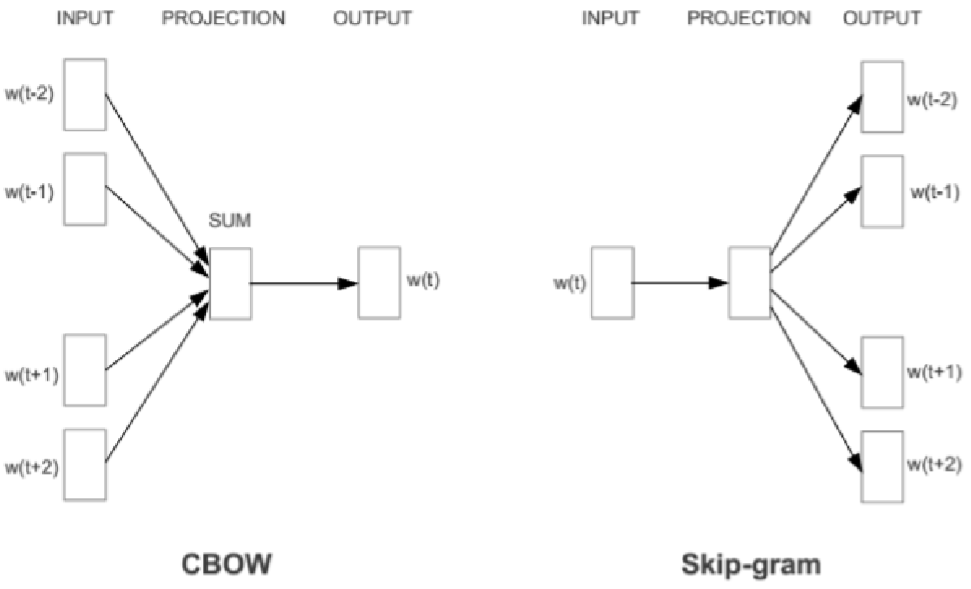
Word2vec就是将词表征为实数值向量的一种高效的算法模型，是一种分布式表示，其利用深度学习的思想，可以通过训练把文本中的每一个词映射成一个固定长度（如k维）的向量，将所有的向量放在一起形成一个词向量空间，每一向量为该空间中的一个点，向量空间上的相似度可以用来表示文本语义上的相似。word2vec的训练过程是有监督的。

Word2vec输出的词向量可以被用来做很多NLP相关的工作，比如聚类、找同义词、词性分析等。如果换个思路，把词当做特征，那么Word2vec就可以把特征映射到K维向量空间，可以为文本数据寻求更加深层次的特征表示 。

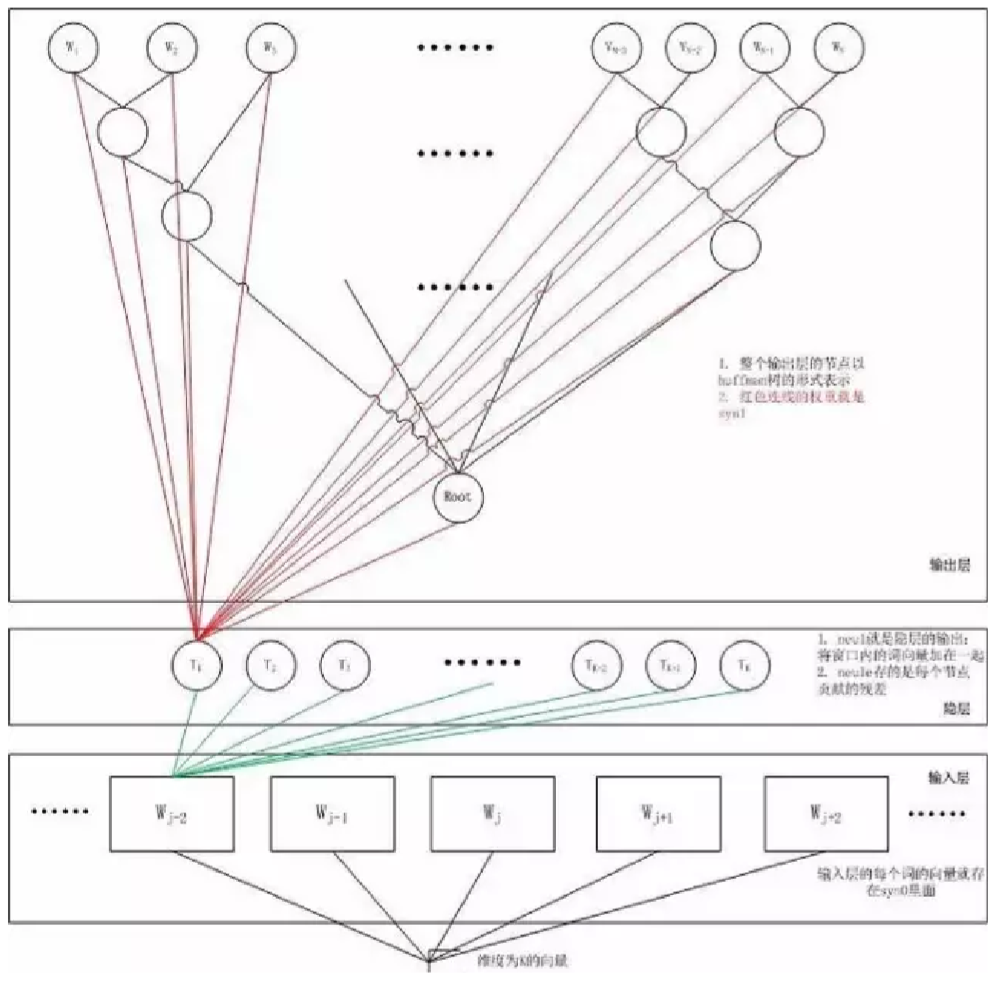
Word2vec属于分布式表示。其基本思想是：通过训练将每个词映射成K维实数向量（K一般为模型中的超参数），通过词之间的距离（比如 cosine 相似度、欧氏距离等）来判断它们之间的语义相似度。其采用一个三层的神经网络，输入层-隐层-输出层。有个核心的技 术是根据词频用Huffman编码，使得所有词频相似的词隐藏层激活的内容基本一致，出现频率越高的词语，他们激活的隐藏层数目越少，这样有效的降低了计算的复杂度。而Word2vec大受欢迎的一个原因正是其高效性，一个优化的单机版本一天可训练上千亿词。

与潜在语义分析（Latent SemanticIndex Analysis, LSA）、潜在狄立克雷分配（Latent Dirichlet Allocation，LDA）的经典过程相比，Word2vec利用了词的上下文，语义信息更加地丰富。

Word2Vec有两种不同的方法：Continuous Bagof Words (CBOW)和Skip-gram。CBOW的目标是根据上下文来预测当前词语的概率。Skip-gram刚好相反：根据当前词语来预测上下文的概率（如下图所示）。这两种方法都利用人工神经网络作为它们的分类算法。起初，每个单词都是一个随机N维向量。经过训练之后，该算法利用CBOW或者Skip-gram的方法获得了每个单词的优向量。



取一个适当大小的窗口当做语境，输入层读入窗口内的词，将它们的向量（K维，初始随机）加和在一起，形成隐藏层K个节点。输出层是一个巨大的二叉树，叶节点代表语料里所有的词（语料含有V个独立的词，则二叉树有|V|个叶节点）。而这整颗二叉树构建的算法就是Huffman树。这样，对于叶节点的每一个词，就会有一个全局唯一的编码，形如"010011"，不妨记左子树为1，右子树为0。接下来，隐层的每一个节点都会跟二叉树的内节点有连边，于是对于二叉树的每一个内节点都会有K条连边，每条边上也会有权值。



对于语料库中的某个词w\_t，对应着二叉树的某个叶子节点，因此它必然有一个二进制编码，如"010011"。在训练阶段，当给定上下文，要预测后面的词w\_t的时候，我们就从二叉树的根节点开始遍历，这里的目标就是预测这个词的二进制编号的每一位。即对于给定的上下文，我们的目标 是使得预测词的二进制编码概率大。形象地说，我们希望在根节点，词向量和与根节点相连经过logistic计算得到bit=1的概率尽量接近0，在第二层，希望其 bit=1 的概率尽量接近1，这么一 直下去，我们把一路上计算得到的概率相乘，即得到目标词w\_t在当前网络下的概率P(w\_t)，那么对于当前这个sample的残差就是1-P(w\_t)，于是就可以使用梯度下降法训练这个网络得到所有的参数值了。显而易见，按照目标词的二进制编码计算到后的概率值就是归一化的。

Hierarchical Softmax用Huffman编码构造二叉树，其实借助了分类问题中，使用一连串二分类近似多分类的思想。例如我们是把所有的词都作为输出，那么“桔子”、“汽车”都是混在一起。给定w\_t 的上下文，先让模型判断w\_t是不是名词，再判断是不是食物名，再判断是不是水果，再判断是不是“桔子”。

在训练过程中，模型会赋予这些抽象的中间结点一个合适的向量，这个向量代表了它对应的所有子结点。因为真正的单词公用了这些抽象结点的向量，所以Hierarchical Softmax方法和原始问 题并不是等价的，但是这种近似并不会显著带来性能上的损失同时又使得模型的求解规模显著上升。

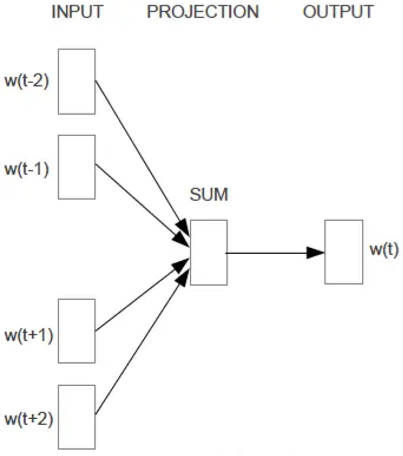
没有使用这种二叉树，而是直接从隐层直接计算每一个输出的概率——即传统的Softmax，就需要对|V|中的每一个词都算一遍，这个过程时间复杂度是O(|V|)的。而使用了二叉树（如Word2vec中 的Huffman树），其时间复杂度就降到了O(log2(|V|))，速度大大地加快了。

现在这些词向量已经捕捉到上下文的信息。我们可以利用基本代数公式来发现单词之间的关系（比 如，“国王”-“男人”+“女人”=“王后”）。这些词向量可以代替词袋用来预测未知数据的情感状况。该模型的优点在于不仅考虑了语境信息还压缩了数据规模（通常情况下，词汇量规模大约在300个单词左右而不是之前模型的100000个单词）。因为神经网络可以替我们提取出这些特征的信息，所以我们仅需要做很少的手动工作。但是由于文本的长度各异，我们可能需要利用所有词向量的平均值作为分类算法的输入值，从而对整个文本文档进行分类处理。

# 两种训练词向量的方法

## CBOW

CBOW（ Continuous Bagof-Words）也称为词袋法，不考虑word之间的先后顺序，可以理解为：用中心词的前后C个词（上下文）来计算中心词出现的概率。模型的输入不止用了历史词语，还用了未来词语。 即， 用t-n+1…t-1,t+1,…t+n-1的word作为输入，目标是正确分类得到第t个word。实验中得到的best n=4。



从图中能够看出，CBOW包含三层结构，分别是输入层，投影层，输出层；CBOW模型是在已知当前词上下文context的前提下预测当前词w(t)，类似阅读理解中的完形填空。

CBOW的复杂度为：

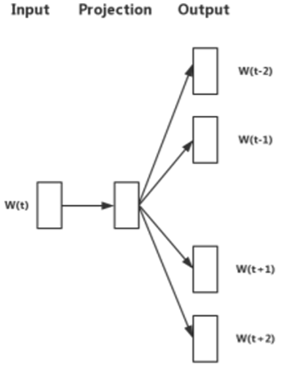


## Skip-gram

Skip-Gram是根据某个词，然后分别计算它前后C个词的各自出现的概率。即在已知当前词w(t)的前提下，预测上下文context。

Skip-gram不利用上下文。其输入为当前word，经过projection的特征提取去预测该word周围的c个词，其cost function为：

如下图所示。这里c增大有利于模型的完备性，但过大的c可能造成很多无关词语相关联，因此用随机采样方法，远的词少采，近的多采。

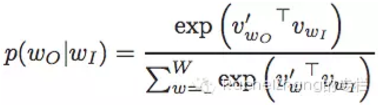


比如定义最大周围距离为C，则对于每个词w(t)，就选择距离为R=range(1,C)， 选前后各R 个词作为预测结果。

Skip-gram Model的复杂度为：



具体来说，最简单的情况下， P(wt+j|wt) 的表达式可以为：



其中v和v’分别为输入和输出中的word特征向量。所以说，word2vec方法本质上是一个动态的逻辑回归。

举例：假设有句子：I like to eat apple

假设：1）window=5 (window表示每次取词的窗口大小)；2）中心词为to

则对于CBOW，输入和输出分别是：

|  |
| --- |
| input : ( I, like, eat, apple )  target: ( to ) |

对于Skip-gram，输入和输出分别是：

|  |
| --- |
| ( input, target )  => ( to, I )  => ( to, like )  => ( to, eat )  => ( to, apple ) |

Word2Vec系统将遍历所有提供的gram和输入单词，并尝试学习适当的mapping vectors（embeddings），这些向量在给定输入单词的情况下为预测正确的上下文次产生高概率。

# 两种加速方法

## Negative Sampling

Negative Sampling（NEG）是Tomas Mikolov在Distributed Representations of Words and Phrasesand their Compositionality中提出的，它是噪声对比损失函数NCE(Noise Contrastive Estimation)的简化版本，用于提高训练速度和提升词向量质量。

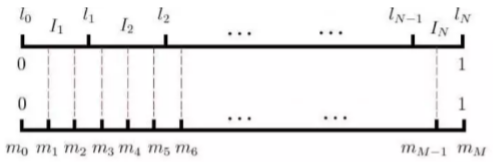
比如对于训练样本，中心词是w，它周围上下文共有2c个词，记为context(w)。由于这个中心词 w，的确和context(w)相关存在，因此它是一个真正例。通过Negative Sampling进行负采样， 得到neg（负采样的个数）个和w不同的中心词wi，i=1,2,..neg，这样context(w)和wi就组成了neg个并不真实存在的负例。利用这一个正例和neg个负例，进行二元逻辑回归（可以理解成一个二分类问题），得到负采样对应每个词wi对应的模型参数以及每个词的词向量。

## 负采样方法

word2vec采样的方法并不复杂，如果词汇表的大 小为V，那么我们就将一段长度为1的线段分成V份，每份对应词汇表中的一个词。当然每个词对应的 线段长度是不一样的，高频词对应的线段长，低频词对应的线段短（根据词频采样，出现的次数越 多，负采样的概率越大）。每个词w的线段长度由下式决定：

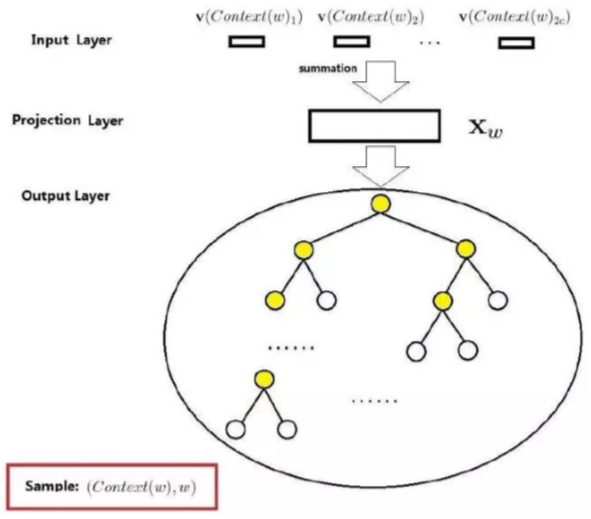


在采样前，我们将这段长度为1的线段划分成M等份，这里M»V，这样能够保证每个词对应的线段都会划分成对应的小块，而M份中每一份都会落在某一个词对应的线段上(如下图)，采样的时候，只需要随机生成neg个数，对应的位置就是采样的负例词。



## Hierarchical Softmax

如下图所示：网络结构包含三层网络结构，输入层，投影层，输出层。输入层到投影层是把输入层的所有向量进行加和给投影层，比如，输入的是三个4维词向量： (1,2,3,4)，(9,6,11,8)，(5,10,7,12)，那么我们word2vec映射后的词向量就是(5,6,7,8)，对CBOW模型来说，就是把上下文词向量加和；对于Skip-Gram模型来说就是简单的传值。最后的输出是构建一颗哈夫曼树，如何去构造简单的哈夫曼树。在这里不在累述；在这里，哈夫曼树的所有叶子节点是词表中的所有词，权值是每个词在词表中出现的次数，也就是词频。



一般得到哈夫曼树后我们会对叶子节点进行哈夫曼编码，由于权重高的叶子节点越靠近根节点，而权重低的叶子节点会远离根节点，这样我们的高权重节点编码值较短，而低权重值编码值较长。这保证的树的带权路径最短，也符合信息论，即希望越常用的词（词频越高的词）拥有更短的编码，一般的编码规则是左0右1，但是word2vec中正好采用了相反的编码规则，同时约定左子树的权重不小于右子树的权重。

如何“沿着哈夫曼树一步步完成”呢？ 在word2vec中，采用了二元逻辑回归的方法，即规定沿着左子树走，是负类(哈夫曼树编码1)，沿着右子树走，是正类(哈夫曼树编码0)。

使用哈夫曼树有什么好处呢？首先，由于是二叉树，之前计算量为V，现在变成了log2V。 其次，由于使用哈夫曼树是高频的词靠近树根，这样高频词需要更少的时间会被找到，这符合贪心优化思想。

# Word2vector实践

word2vec用于把单词映射为词向量，训练词向量的模型有Skip-gram和CBOW，通常采用Hierarchical Softmax和Negative Sampling两种加速方法。

因此，实现word2vec的方法主要有：

1）cbow + hierarchical softmax

2）cbow + negative sampling

3）skipgram + hierarchical softmax

4）skipgram + negative sampling

# Word2vec的特点

1. 不同于之前的计算cooccurrence次数方法，减少计算量；

2. 高效；

3. 可以轻松将一个新句子/新词加入语料库

# 总结

Wordvec在神经网络中学习将word映射成连续（高维）向量， 其实质是词语特征求取。

参考：

龙书，TensorFlow深度学习

<https://mp.weixin.qq.com/s/XZkQpAWZqFJWfqyIXlMmfg>

<https://mp.weixin.qq.com/s/un6e1KYhKlNJ-rGruazoXw>

<https://mp.weixin.qq.com/s/ElQE-TIqXbXN3Cn1ubvJmw>

<https://mp.weixin.qq.com/s/lE-kKDq6xBVgO35bEDnLVQ>

<https://mp.weixin.qq.com/s/xOzsFjqhEAKQnHW9TP5iJg>

<https://mp.weixin.qq.com/s/eFfa0GcbCk5MSw0t_3mN_w>