论文｜万物皆可Vector之Word2vec：2个模型、2个优化及实战使用

原创 Thinkgamer 搜索与推荐Wiki 2020-12-04

收录于话题
#论文笔记

19个

Word2vec的出现改变了OneHot的高维稀疏的困境，自此之后各种xxx2vec如雨后春笋般冒了出来，用来解决各种嵌入式编码，包括后来的各种Embedding方式其实很多本质上都是Word2vec的延伸和优化。在本公众号「搜索与推荐Wiki」上也发布了不少Embedding相关的文章，后续也会持续的发布相关文章，欢迎关注。

本主题文章将会分为三部分介绍，每部分的主题为：

- word2vec的前奏-统计语言模型（点击阅读）
- word2vec详解-风华不减
- 其他xxx2vec论文和应用介绍

后续会更新Embedding相关的文章，可能会单独成系列，也可能会放到《特征工程-Embedding系列中》，欢迎持续关注「**搜索与推荐Wiki**」

1、背景介绍

word2vec 是Google 2013年提出的用于计算词向量的工具，在论文Efficient Estimation of Word Representations in Vector Space中，作者提出了Word2vec计算工具，并通过对比NNLM、RNNLM语言模型验证了word2vec的有效性。

word2vec工具中包含两种模型：CBOW和skip-gram。论文中介绍的比较简单，如下图所示，CBOW是通过上下文的词预测中心词，Skip-gram则是通过输入词预测上

下文的词。

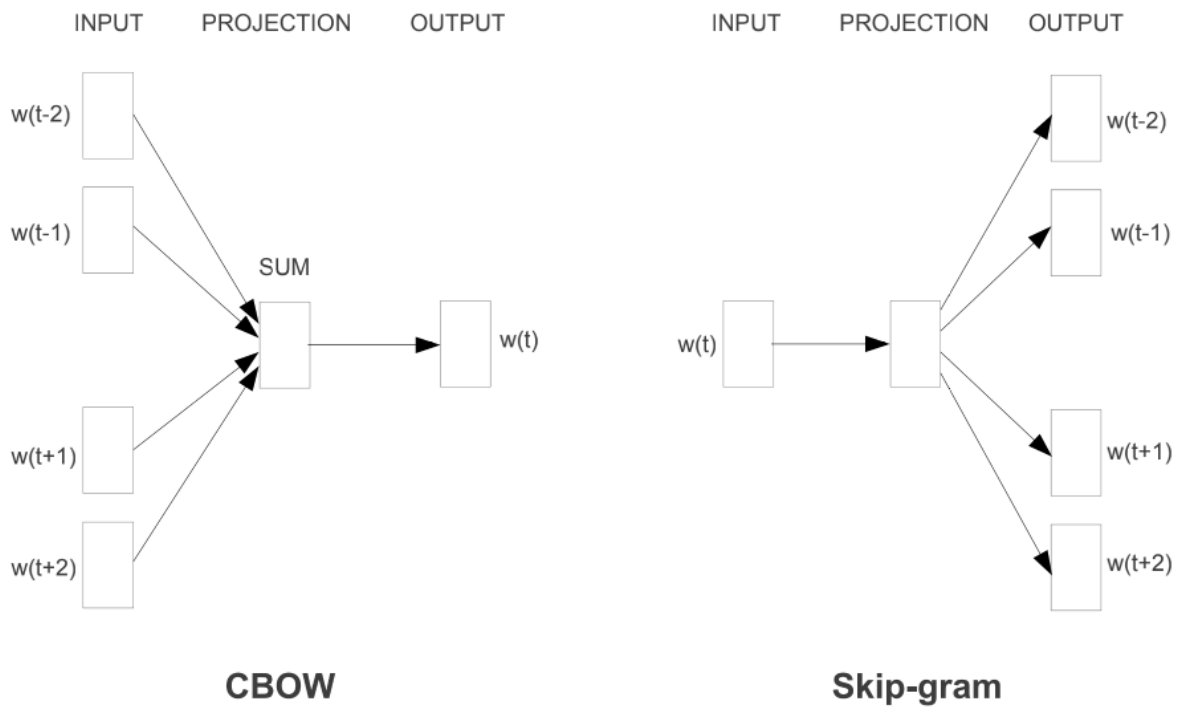


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

CBOW和skip-gram

2、CBOW 和 Skip-gram

原论文对这两种模型的介绍比较粗略，在论文《word2vec Parameter Learning Explained》中进行了详细的解释和说明，接下来我们详细看下CBOW和Skip-gram。

a）CBOW

One-word context

首先看一下只有一个上下文词的情况

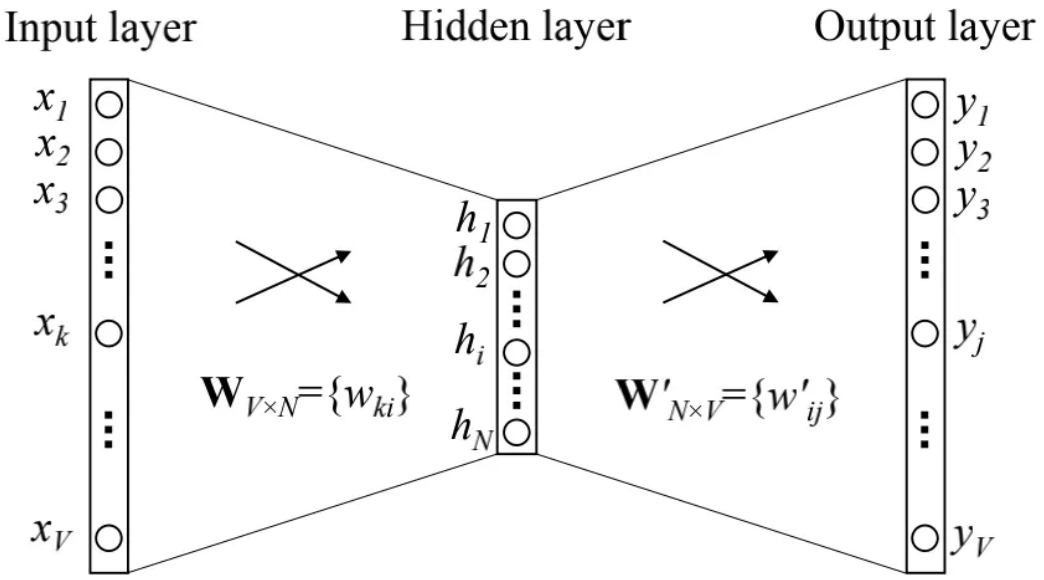


Figure 1: A simple CBOW model with only one word in the context

其中单词的总个数为V，隐藏层的神经元个数为N，输入层到隐藏层的权重矩阵为W_{V*N}，隐藏层到输出层的权重矩阵为W'_{N*V}。

输入层是单词的One-hot编码。从输入层到隐藏层：

$$h = W^T x := v_{w_I}^T$$

v_{w_I} 表示的就是输入单词w_I的向量表示（注意和输入层x进行区分，输入向量即W中的向量表示，输出向量即W'中的向量表示），其维度为$[N,1]$，转置后维度变成了$[1,N]$，用来表示向量的输入表述，要注意**这里不是$[1,N]$，否则容易在往下的第二个公式中相乘时维度搞混**，符号:=表示的**定义为**。

从隐藏层到输入层：

$$u_j = (v'_{w_j})^T h$$

其中v'_{w_j} 表示的是矩阵W'的第j 列，其维度为$[N,1]$，计算出来的u_j为一个具体的值，表示的是第j个输入的词在输出层第j个位置对应的值。

最后使用$softmax$进行归一化处理（**因为输出层是固定的V个单词，所以可以看作是多分类，因此使用$softmax$进行归一化**），得到输入单词w_I所属词库中每个单词的概率：

$$p(w_j|w_I) = y_j = \frac{exp(u_j)}{\sum_{j'=1}^{V} exp(u'_j)}$$

其中 y_j 表示的是输出层的第 j个神经元的值。

联合上面三个公式可得：

$$p(w_j|w_I) = \frac{exp((v'_{w_j})^T * v^T_{w_I})}{\sum_{j'=1}^{V} exp((v'_{w_{j'}})^T * v^T_{w_I})}$$

其中v_w可以理解为单词的输入向量表示，v'_w为单词的输出向量表示。

此种情况下的损失函数为：

$$max\ p(w_O|w_I) = max\, y_{j*} = max\, log\, y_{j*} = u_{j*} - log \sum_{j'=1}^{V} exp(u_{j'}) := -E$$

上述公式可以转化为：

$$E = -u_{j*} + log \sum_{j'=1}^{V} exp(u_{j'})$$

其中$j*$ 表示 实际输出层输出值对应的下标。因为u_j^* 是固定的，因此$max\, log\, y_{j*}$时，只需对分母求log即可

multi-word context

当有多个上下文单词时对应的图为：

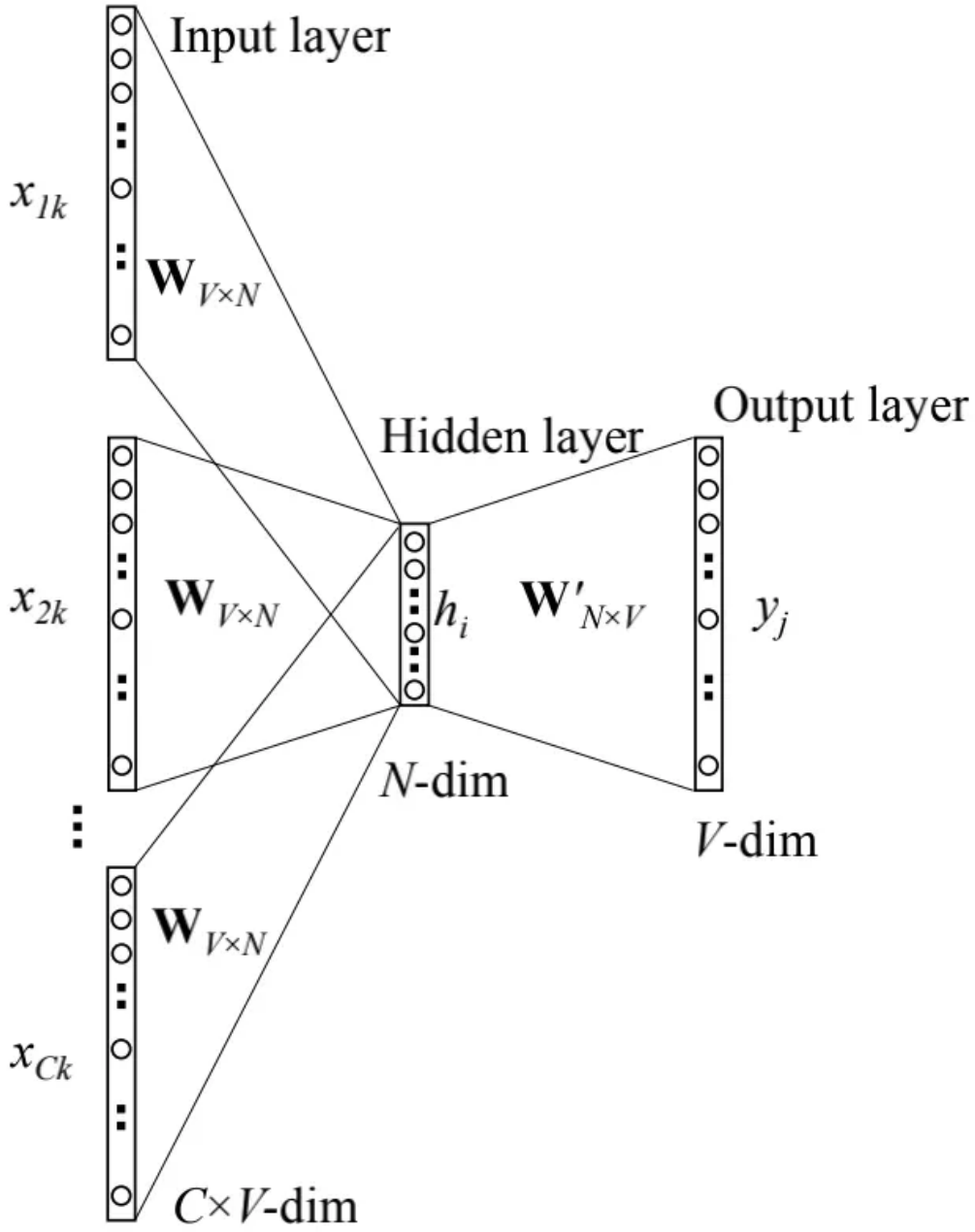


Figure 2: Continuous bag-of-word model

此时的h 表达式为：

$$h = \frac{1}{C}W^T(x_1 + x_2 + \ldots + x_C) = \frac{1}{C}(v_{w_1} + v_{w_2} + \ldots + v_{w_C})^T$$

其中 C 表示上下文单词的个数，$w_1, w_2, \ldots, w_C$ 表示上下文单词，v_w 表示单词的输入向量（注意和输入层x区别）。

目标函数为：

$$E = -log\, p(w_O|w_{I_1}, w_{I_2}, \ldots, w_{I_C}) = -u_j * log \sum_{j'=1}^{V} exp(u'_j) = -(v'_{w_O})^T * h + log \sum_{j'=}^{V}$$

b）Skip-gram

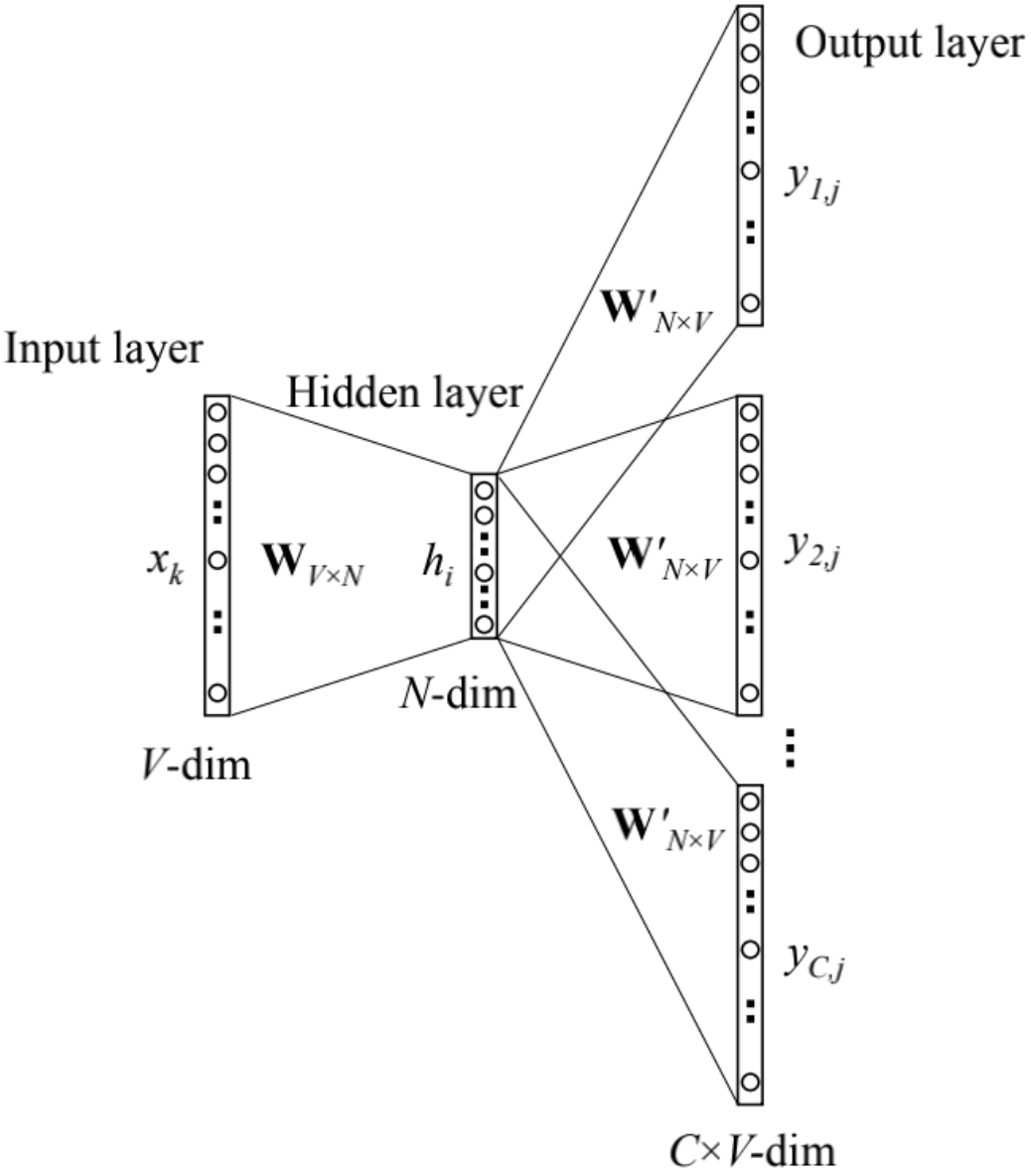


Figure 3: The skip-gram model.

Skip-gram

从输入层到隐藏层：

$$h = W_{k,\cdot}^T := v_{w_I}^T$$

从隐藏层到输出层：

$$p(w_{c,j} = w_{O,c}|w_I) = y_{c,j} = \frac{exp(u_{c,j})}{\sum_{j'=1}^{V} exp(u_{j'})}$$

其中：

- w_I 表示的是输入词
- $w_{c,j}$ 表示输出层第c个词实际落在了第j个神经元
- $w_{O,c}$ 表示输出层第c个词应该落在第O个神经元
- $y_{c,j}$ 表示输出层第c个词实际落在了第j个神经元上归一化后的概率
- $u_{c,j}$ 表示输出层第c个词实际落在了第j个神经元上未归一化的值

因为输出层共享权重，所以：

$$u_{c,j} = u_j = (v'_{w_j})^T * h, for\, c = 1, 2, \ldots, C$$

其中 v'_{w_j} 表示第j个单词的输出向量，其值为输出权重矩阵W'的第j 列。

损失函数变为：

$$E = -log\, p(w_{O,1}, w_{O,2}, .., w_{O,C}|w_I) = -log \prod_{c=1}^{C} \frac{exp(u_{c,j_c^*})}{\sum_{j'=1}^{V} exp(u_{j'})} = -\sum_{c=1}^{C} u_{j_c^*} + C *$$

注意⚠

- **经验上一般选择使用skip-gram模型，因为效果较好**
- **在Word2vec模型中，如果选择使用CBOW时，最终产出的word embedding为 单词的输出向量（W'_{N*V}）表示，如果选择使用skip-gram时，最终产出的word embedding为单词的输入向量（W_{N*V}）表示，因为更倾向于选择靠近中心词一端的权重矩阵。**

3、hierarchical softmax 和negative sampling

因为基于word2vec框架进行模型训练要求语料库非常大，这样才能保证结果的准确性，但随着预料库的增大，随之而来的就是计算的耗时和资源的消耗。那么有没有优化的余地呢？比如可以牺牲一定的准确性来加快训练速度，答案就是 hierarchical softmax 和 negative sampling。

在论文《Distributed Representations of Words and Phrases and their Compositionality》中介绍了训练word2vec的两个技（同样在论文《word2vec Parameter Learning Explained》中进行了详细的解释和说明），下面来具体看一下。

a）霍夫曼树和霍夫曼编码

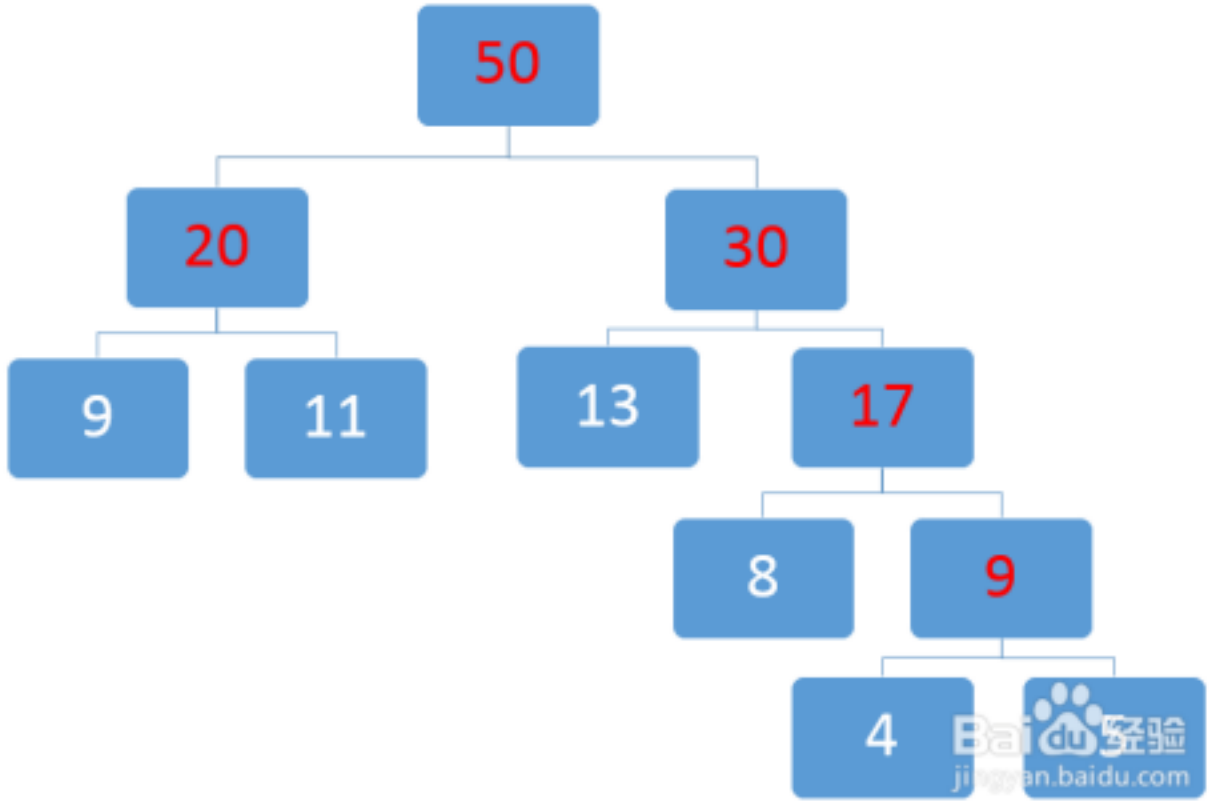
在了解层次softmax（hierarchical softmax）之前，先来理解一下什么是霍夫曼树和霍夫曼编码。

霍夫曼树本质上是一棵最优二叉树，是指对于一组带有确定权值的叶子节点所构造的具有带权路径长度最短的二叉树。

那么针对一组权重值，如何构造一棵霍夫曼树呢？根据**权值大的结点尽量靠近根**这一原则，给出了一个带有一般规律的算法，称为**霍夫曼算法**，其描述如下：

- 1、根据给定n个权值$w_1, w_2, \ldots, w_n$构成n棵二叉树的集合$F = T_1, T_2, \ldots, T_n$;其中,每棵二叉树$T_i(1 <= i <= n)$只有一个带权值$w_i$的根结点,其左、右子树均为空
- 2、在F中选取两棵根结点权值最小的二叉树作为左、右子树来构造一棵新的二叉树,且置新的二叉树根结点权值为其左右子树根结点的权值之和
- 3、在F中删除这两棵树,同时将生成新的二叉树加入到F中
- 4、重复2、3，直到F中只剩下一棵二叉树加入到F中

例如一组数据其对应的权重为：[9,11,13,8,4,5]，其生成的霍夫曼树为（图来源于百度经验）：



霍夫曼树构建示例

注意⚠：

- **在构造哈夫曼树时，叶子节点无左右之分，只需约定好一个规则，从头到尾遵守这个规则执行即可。习惯上左节点比右节点小。**

那什么又是霍夫曼编码呢？霍夫曼编码是一种基于霍夫曼树的编码方式，是可变长编码的一种。

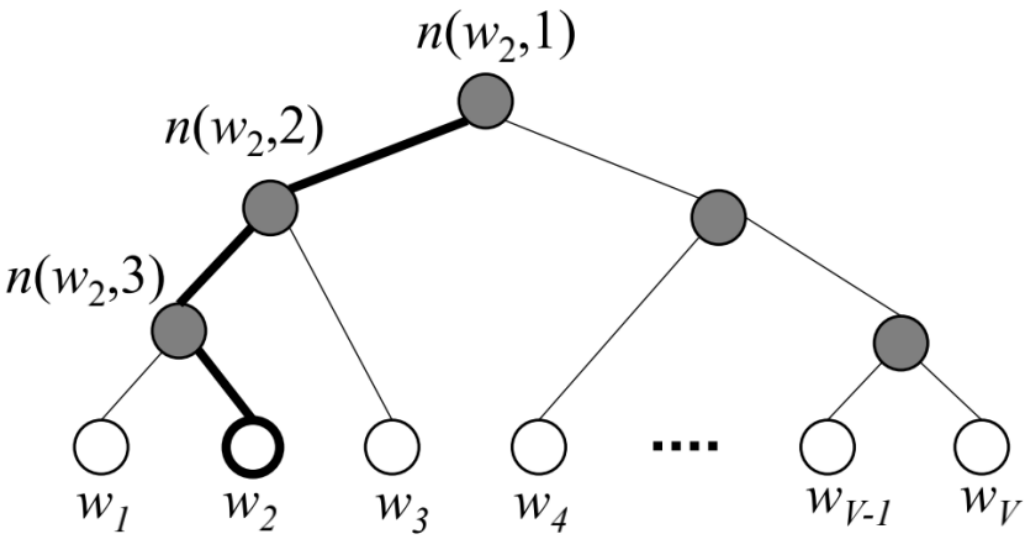
对于构造好的霍夫曼树进行0/1编码，左子树为0，右子树为1，则针对上图构造好的霍夫曼树，其各个叶子节点的霍夫曼编码分别为：

- 9 -> 00
- 11 -> 01
- 13 -> 10
- 8 -> 110
- 4 -> 1110
- 5 -> 1111

注意⚠：

- **同样针对霍夫曼树的编码也没有明确规定说左子树为1或者左子树为0**
- **在word2vec中，针对霍夫曼树的构建和编码和上边说的相反，即约定左子树编码为1，右子树编码为0（论文中说的是-1，含义一致），同时约定左子树的权重不小于右子树的权重**

b）hierarchical softmax



hierarchical softmax

上图为一棵霍夫曼编码树，其中白色结点表示词库中的所有单词，黑色结点表示内部的隐藏结点，单词w_2 对应的路径编码如图中黑色线连接所示，其路径长度为4，

$n(w,j)$表示的是针对单词w，其所在路径上的第j个结点。

基于霍夫曼树进行优化的word2vec，移除了从隐藏层到输出层的权重矩阵（即输出向量），使用的是霍夫曼树中的隐藏结点编码代替（如上图中的黑色结点），那么输出结点是输入单词w的概率可以表示为：

$$p(w=w_O)=\prod_{j=1}^{L(w)-1}\sigma([n(w,j+1)=ch(n(w,j))]\cdot(v'_{n(w,j)})^T*h)$$

其中：

- $ch(n)$ 表示路径中的隐藏的左结点
- $v'_{n(w,j)}$ 表示 隐藏结点的向量表示（整个算法优化过程中的辅助向量）
- $n(w,j)$ 表示单词w所在路径上的第j个结点
- h 表示隐藏层的输出（skip-gram模型中其等于v_{w_I}，cbow模型中其等于 $\frac{1}{C}\sum_{c=1}^{C}v_{w_c}$，即输入词向量求平均）
- $L(w)$ 表示叶子结点是w的最短路径中长度，减1表示的是到达该结点之前的隐藏结点
- $[x]$ 的定义如下（即如果走的是左子树路径为1，右子树路径为-1）

$$[x]=\begin{cases}1 & if\,x\,is\,true\\ -1 & otherwise\end{cases}$$

在上图中，我们定义是左结点的概率为：

$$p(n,left)=\sigma((v'_n)^T\cdot h)$$

其中σ 表示的是$sigmoid$函数

右结点的概率为：

$$p(n,right)=1-\sigma((v'_n)^T\cdot h)=\sigma(-(v'_n)^T\cdot h)$$

那么针对图中的单词w_2，其概率为：

$$p(w_2=w_O)=p(n(w_2,1),left)\cdot p(n(w_2,2),left)\cdot p(n(w_2,3),right)=\sigma((v'_{n(w_2,1}$$

针对所有的单词有：

$$\sum_{i=1}^{V}p(w_i=w_O)=1$$

此时其损失函数为：

$$E=-log\,p(w=w_O|w_I)=-\sum_{j=1}^{L(w)-1}log\,\sigma([n(w,j+1)=ch(n(w,j))]\cdot(v'_{n(w,j)})^T*$$

c) negative sampling

除了hierarchical softmax，另外一种优化方法是Noise Contrasive Estimation（NCE），在论文《**Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics**》中有详细的解释和说明，但因为NCE的逻辑有些复杂，所以这里使用的是简化版的，称之为：**Negative Sampling**。

因为每次计算全量的负样本计算量比较大，因此进行了负采样，负采样之后对应的损失函数为：

$$E=-log\sigma((v'_{w_O})^Th)-\sum_{w_j\in W_{neg}}log\,\sigma(-(v'_{w_i})^Th)=-log\sigma((v'_{w_O})^Th)+\sum_{w_j\in W_{neg}}log\,\sigma($$

其中：

- w_O 表示输出的单词
- v'_{w_O} 表示w_O的输出词向量
- h 表示隐藏层的输出，当模型为CBOW时为$\frac{1}{C}\sum_{c=1}^{C}v_{w_c}$，如果时skip-gram模型时为：$v_{w_I}$
- W_{neg} 表示的是负采样的样本数

注意⚠：

- **基于层次softmax或者negative sampling优化的cbow或者skip-gram模型，输出的词向量应该是输入层到隐藏层之间的词向量（之所以说应该，是因为论文中没有进行特意说明，也没有在公开的资料中看到，可能是我看的不够认真）**
- **猜想：能否根据最短路径节点的平均向量来表示叶子结点，即词向量？**
- **以上两个问题有读者明白了可以在评论区进行留言，感谢！**

4、Gensim中Word2vec的使用

关于gensim的文档可以参考：https://radimrehurek.com/gensim/auto_examples/index.html#documentation

使用之前需要先引入对应的模型类

```
from gensim.models.word2vec import Word2Vec
```

创建一个模型

```
model = Word2Vec(sentences=topics_list, iter=5, size=128, window=5, min_count=0, workers=10, sg=1,
```

其对应的模型参数有很多，主要的有：

- sentences：训练模型的语料，是一个可迭代的序列
- corpus_file：表示从文件中加载数据，和sentences互斥
- size：word的维度，默认为100，通常取64、128、256等
- window：滑动窗口的大小，默认值为5
- min_count：word次数小于该值被忽略掉，默认值为5
- seed：用于随机数发生器
- workers：使用多少线程进行模型训练，默认为3
- min_alpha=0.0001
- sg：1 表示 Skip-gram 0 表示 CBOW，默认为0
- hs：1 表示 hierarchical softmax 0 且 negative 参数不为0 的话 negative sampling 会被启用，默认为0
- negative：0 表示不采用，1 表示采用，建议值在 5-20 表示噪音词的个数，默认为5

更多参数可以参考模型中的注释

保存模型

```
model.save(model_path)
```

加载模型

```
model = Word2Vec.load(model_path)
```

输出loss值

```
model.get_latest_training_loss()
```

计算相似度

```
model.wv.similarity(word1, word2)
```

如果觉得文章不错，点个赞、在看，或者分享给更多人看到吧（戳【**阅读原文**】触达更多精彩内容）！

收录于话题 #论文笔记·19个

上一篇
论文｜Item2vec中值得品味的8个经典tricks

下一篇
论文｜万物皆可Vector之语言模型：从N-Gram到NNLM、RNNLM

阅读原文

喜欢此内容的人还喜欢

独家｜利用Python实现主题建模和LDA 算法（附链接）

数据派THU