自然语言处理 | word2vector的原理及实例解析

DataGod 数据挖掘 2017-04-04

Word2vec是 Google 在 2013 年年中开源的一款将词表征为实数值向量的高效工具, 其利用深度学习的思想，可以通过训练，把对文本内容的处理简化为 K维向量空间中的向量运算，而向量空间上的相似度可以用来表示文本语义上的相似度。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 使用的是 Distributed representation 的词向量表示方式。其基本思想是通过训练将每个词映射成K维实数向量，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。

这个三层神经网络本身是对语言模型进行建模，但也同时获得一种单词在向量空间上的表示，而这个副作用才是Word2vec的真正目标。

与潜在语义分析（Latent Semantic Index, LSI）、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。

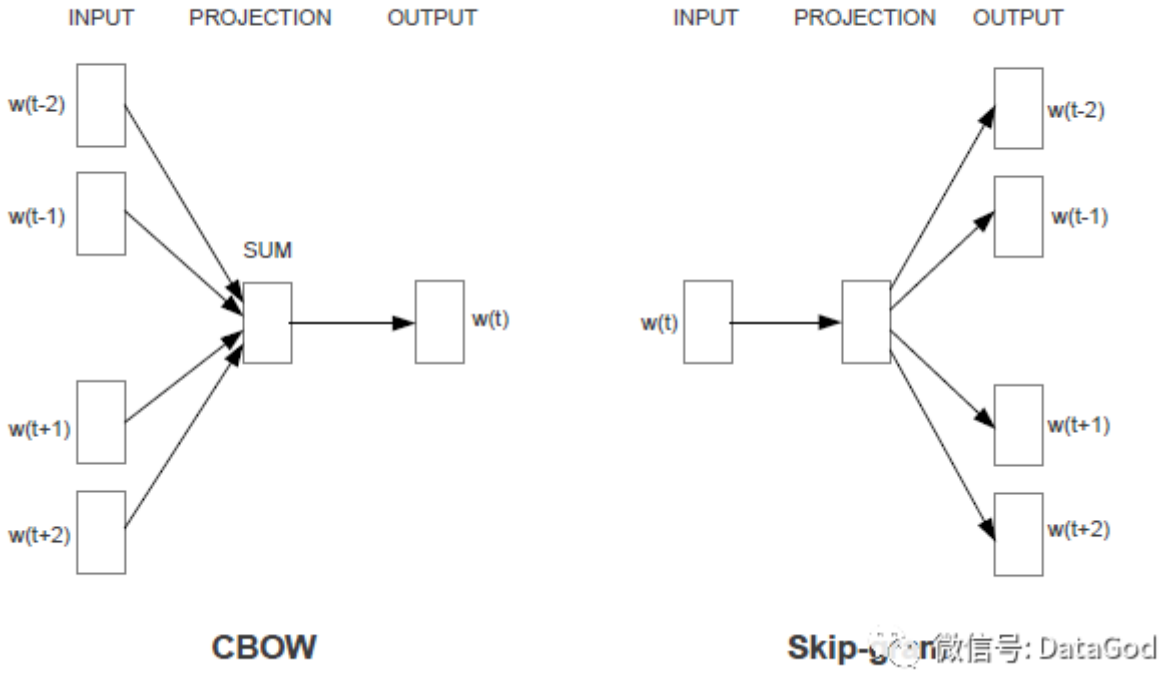
word2vec的两个模型CBOW和Skip-Gram

假设有这样一句话：今天 下午 2点钟搜索引擎组开组会。

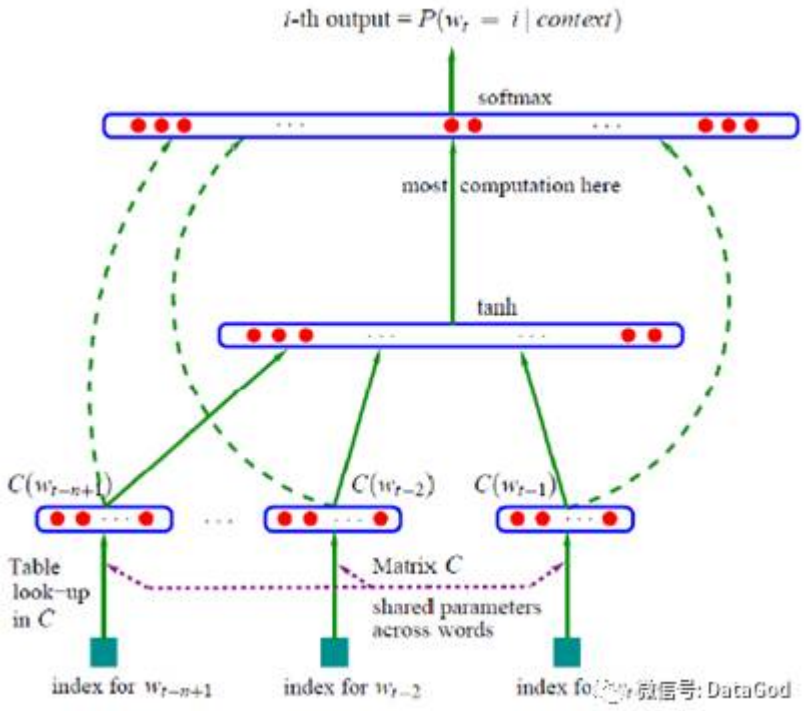
任务1：对于每一个word，使用该word周围的word来预测当前word生成的概率。如使用“今天、下午、搜索、引擎、组”来预测“2点钟”。

任务2：对于每一个word，使用该word本身来预测生成其他word的概率。如使用“2点钟”来生成“今天、下午、搜索、引擎、组”中的每个word。

两个任务共同的限制条件是：对于相同的输入，输出每个word的概率之和为1。



Word2vec的模型就是想通过机器学习的方法来达到提高上述任务准确率的一种方法。两个任务分别对应两个的模型（CBOW和skim-gram）。下面主要对CBOW进行分析。



上图是原始的未经过优化的用于训练上节提到的任务的神经网络模型。该模型有三层结构：输入层，隐藏层，输出层。

其中输入层和传统的神经网络模型有所不同，输入的每一个节点单元不再是一个标量值，而是一个向量，向量的每一个值为变量，训练过程中要对其进行更新，这个向量就是我们所关心的word所对应的vector，假设该向量维度为D。该层节点个数为整个语料库中的不同word的个数，设为V。隐藏层和传统

神经网络模型一样，使用激活函数为tanh或sigmoid均可。假设该层节点个数为H。输出层同样和传统神经网络模型一样，节点个数为整个语料库中的不同word的个数，即V。另外如果你认认真真的学过神经网络稀疏自编码的知识就会知道，如果将输入层和输出层之间再构建一层传递关系，更有利于提高该模型的准确度。

时间复杂度分析，对于上文中所举得例子，我们知道输入是N个word，输出是“2点钟”。它所对应的时间复杂度为N * D + N * D * H + N *D * V + H * V。其中D和H大约在100和500之间，N是3到8，而V则高达几百万数量级。因此该模型的瓶颈在后面两项。接下来我们要讨论的问题就是如何解决这个时间复杂度的问题。

(1)对于输入层到输出层的网络，我们可以直接将其剔除掉，实验证明，该措施并不会带来很多效果上的下降，反而省去了大部分计算时间。之所以隐藏层到输出层计算量最大是因为我们对于每一个输出的word均要进行验证并使用梯度下降更新。正确的word所对应的节点为正例，其他为反例，从而达到快速收敛的目的。

还可以使用：

(2)使用Hierarchical softmax或negativesampling。

(3)去除小于minCount的词。

(4)算出每个词被选出的概率，如果选出来则不予更新。此方法可以节省时间而且可以提高非频繁词的准确度。

(5)选取邻近词的窗口大小不固定。有利于更加偏重于离自己近的词进行更新。

(6)多线程，无需考虑互斥。

实例解析

从http://mattmahoney.net/dc/text8.zip下载了一个文件text8文件，为分好词的文件，词与词之间用空格隔开，将分好词的训练语料进行训练，这里数据量较小，几分钟就好了。以下是生成词向量的shell命令：

./word2vec -train text8 -output vectors.bin-cbow 0 -size 48 -window 5 -negative 0 -hs 1 -sample 1e-4 -threads 20 -binary 1-iter 100

以上命令参数解释：

-train text8 表示的是输入文件是text8，

-output vectors.bin 输出文件是vectors.bin，

-cbow 0表示不使用cbow模型，默认为Skip-Gram模型。

-size 48 每个单词的向量维度是48，

-window 5 训练的窗口大小为5就是考虑一个词前五个和后五个词语（实际代码中还有一个随机选窗口的过程，窗口大小小于等于5）。

-negative 0 -hs 1不使用NEG方法，使用HS方法。

-sampe指的是采样的阈值，如果一个词语在训练样本中出现的频率越大，那么就越会被采样。

-binary为1指的是结果二进制存储，为0是普通存储（普通存储的时候是可以打开看到词语和对应的向量的）除了以上命令中的参数，word2vec还有几个参数对我们比较有用比如-alpha设置学习速率，默认的为0.025.

–min-count设置最低频率，默认是5，如果一个词语在文档中出现的次数小于5，那么就会丢弃。

-classes设置聚类个数，看了一下源码用的是k-means聚类的方法。要注意-threads 20 线程数也会对结果产生影响。

经验说明：

模型：skip-gram慢，对罕见字有利；CBOW快。

训练算法：分层softmax对罕见字有利；负采样对常见词和低纬向量有利。

欠采样频繁词：可以提高结果的准确性和速度，适用范围1e-3到1e-5。

文本框(window)大小：skip-gram通常在10附近，CBOW通常在5附近。

计算词与词之间的余弦相似度

./distance vectors.bin

聚类

./word2vec -train text8 -output classes.txt -cbow 0 -size 200 -window 5 -negative 0 -hs 1 -sample 1e-3 -threads 12 -classes 100

按类别排序

sort classes.txt -k 2 -n > classes.sorted.txt

下面使用python的gensim包

>>> import gensim

输入句子，每个句子是一个词列表，中文的话，可以用jieba先分好词。

>>> sentences = [['first','sentence'], ['second', 'sentence']]

将输入视为Python的内置列表很简单, 但是在量很大时会占用大量的内存。所以Gensim只要求输入按顺序提供句子, 并不需要将这些句子存储在内存, 然后Gensim可以加载一个句子, 处理该句子, 然后加载下一个句子。因此我们可以写个迭代器

```
class MySentences(object):
    def __init__(self, dirname):
        self.dirname = dirname

    def __iter__(self):
        for fname in os.listdir(self.dirname):
            for line in open(os.path.join(self.dirname, fname)):
                yield line.split()
```

```
>>> sentences = MySentences('存文件的文件夹路径')
```

训练词向量

```
>>> model =gensim.models.Word2Vec(sentences, min_count=1)
```

训练过程可以加入很多参数，参数说明可以到这里查看：

http://blog.csdn.net/qq_26972303/article/details/53954103

注意：

```
model = Word2Vec(sentences, workers=4)
```

workers参数，是控制并线，只有在安装了Cython后才有效. 没有Cython的话, 只能使用单核.

模型存储和加载

```
>>> model.save('/tmp/mymodel')
>>> new_model =gensim.models.Word2Vec.load('/tmp/mymodel')
```

使用模型

Word2vec支持数种单词相似度任务:

```
>>> model.most_similar(positive=['woman','king'], negative=['man'], topn=1)
[('queen', 0.50882536)] 与woman，king接近，但是与man不接近的词
>>> model.doesnt_match("breakfastcereal dinner lunch".split())
'cereal'
>>> model.similarity('woman','man')
0.73723527
>>> model.similar_by_word('woman', topn=2, restrict_vocab=None)
```

可以通过以下方式来得到单词的向量:

```
>>> model['computer']  # raw NumPy vector of a word
array([-0.00449447, -0.00310097,  0.02421786, ...], dtype=float32)
```