

# 【NLP】doc2vec原理及实践

Johnson0722 深度学习自然语言处理 2019-09-21

点击上方，选择星标或置顶，每天给你送干货🤖!

阅读大概需要7分钟🤖

跟随小博主，每天进步一丢丢🤖

作者：Johnson0722

链接：

[https://blog.csdn.net/John\\_xyz/article/details/79208564](https://blog.csdn.net/John_xyz/article/details/79208564)

## 1.“句向量”简介

word2vec提供了高质量的词向量，并在一些任务中表现良好。

关于**word2vec**的原理可以参考这几篇论文：

<https://arxiv.org/pdf/1310.4546.pdf>

<https://arxiv.org/pdf/1301.3781.pdf>

关于如何使用第三方库gensim训练word2vec可以参考这篇博客：

[http://blog.csdn.net/john\\_xyz/article/details/54706807](http://blog.csdn.net/john_xyz/article/details/54706807)

尽管word2vec提供了高质量的词汇向量，仍然没有有效的方法将它们结合成一个高质量的文档向量。对于一个句子、文档或者说一个段落，怎么把这些数据投影到向量空间中，并具有丰富的语义表达呢？过去人们常常使用以下几种方法：

- bag of words
- LDA
- average word vectors
- tfidf-weighting word vectors

就**bag of words**而言，有如下缺点：**1.没有考虑到单词的顺序，2.忽略了单词的语义信息**。因此这种方法对于短文本效果很差，对于长文本效果一般，通常在科研中用来做baseline。

**average word vectors**就是简单的对句子中的所有词向量取平均。是一种简单有效的方法，但**缺点也是没有考虑到单词的顺序**

**tfidf-weighting word vectors**是指对句子中的所有词向量根据tfidf权重加权求和，是常用的一种计算sentence embedding的方法，在某些问题上表现很好，相比于简单的对所有词向量求

平均，考虑到了tfidf权重，因此句子中更重要的词占得比重就更大。但**缺点也是没有考虑到单词的顺序**

**LDA模型**当然就是**计算出一片文档或者句子的主题分布**。也常常用于文本分类任务，后面会专门写一篇文章介绍LDA模型和doc2vec的本质不同

## 2. doc2vec原理

doc2vec是google的两位大牛Quoc Le和Tomas Mikolov在2014年提出的，原始论文地址如下：

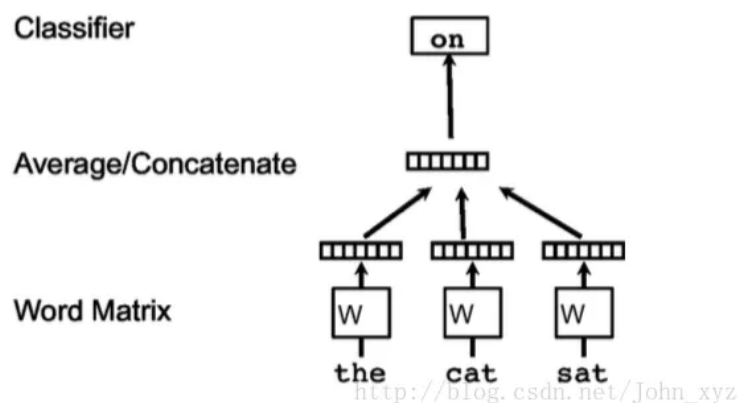
[https://cs.stanford.edu/~quocle/paragraph\\_vector.pdf](https://cs.stanford.edu/~quocle/paragraph_vector.pdf)

**Doc2Vec** 或者叫做 paragraph2vec, sentence embeddings，是一种非监督式算法，可以获得 sentences/paragraphs/documents 的向量表达，是 word2vec 的拓展。学出来的向量可以通过计算距离来找 sentences/paragraphs/documents 之间的相似性，可以用于文本聚类，对于有标签的数据，还可以用监督学习的方法进行文本分类，例如经典的**情感分析**问题。

在介绍doc2vec原理之前，先简单回顾下word2vec的原理

### word2vec基本原理

熟悉word2vec的同学都知道，下图是学习词向量表达最经典的一幅图。在下图中，任务就是给定上下文，预测上下文的其他单词。



其中，每个单词都被映射到向量空间中，将上下文的词向量级联或者求和作为特征，预测句子中的下一个单词。一般地：给定如下训练单词序列 $w_1, w_2, w_3, \dots, w_T$ ，目标函数是

$$\frac{1}{T} \sum_{t=k}^{T-k} \log p(w_t | w_{t-k}, \dots, w_{t+k})$$

当然，预测的任务是一个多分类问题，分类器最后一层使用softmax，计算公式如下：

$$p(w_t | w_{t-k}, \dots, w_{t+k}) = \frac{e^{k_t}}{\sum_i e^{k_i}}$$

这里的每一个 $y_i$ 可以理解为预测出每个word的概率。因为在该任务中，每个词就可以看成一个类别。计算 $y_i$ 的公式如下：

$$y = b + Uh(w_{t-k}, \dots, w_{t+k}; W)$$

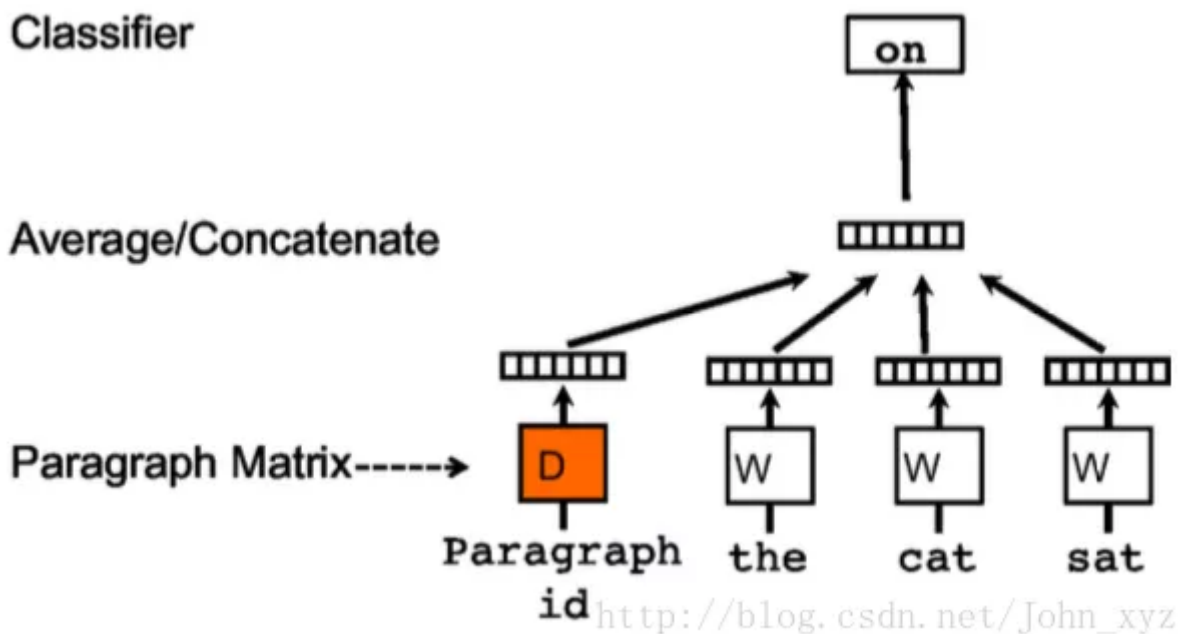
这里U和b都是参数，h是将 $w_{t-k}, \dots, w_{t+k}$ 级联或者求平均。

因为每个单词都是一类，所以类别众多，在计算softmax归一化的时候，效率很低。因此使用hierarchical softmax加快计算速度，其实就是huffman树，这个不再赘述，有兴趣的同学可以看word2vec的paper。

## doc2vec基本原理

### 1. A distributed memory model

训练句向量的方法和词向量的方法非常类似。**训练词向量的核心思想就是可以根据每个单词 $w_i$ 的上下文预测 $w_i$** ，也就是说上下文的单词对 $w_i$ 是有影响的。那么同理，可以用同样的方法训练doc2vec。例如对于一个句子s: i want to drink water，如果要去预测句子中的单词want，那么不仅可以根据其他单词生成feature，也可以根据其他单词和句子ss来生成feature进行预测。因此doc2vec的框架如下所示：



每个段落/句子都被映射到向量空间中，可以用矩阵DD的一列来表示。每个单词同样被映射到向量空间，可以用矩阵WW的一列来表示。然后将段落向量和词向量级联或者求平均得到特征，预测句子中的下一个单词。

这个段落向量/句向量也可以认为是一个单词，它的作用相当于是上下文的记忆单元或者是这个段落的主题，所以我们一般叫这种训练方法为Distributed Memory Model of Paragraph Vectors(PV-DM)

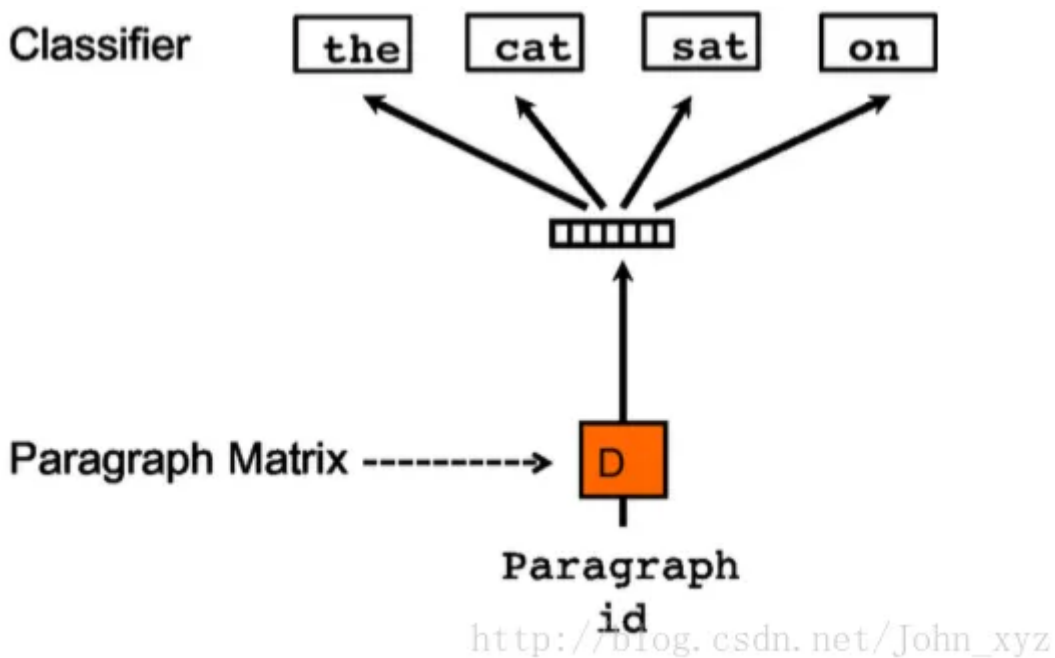
在训练的时候我们固定上下文的长度，用滑动窗口的方法产生训练集。段落向量/句向量 在该上下文中共享。

### 总结doc2vec的过程, 主要有两步:

- 训练模型，在已知的训练数据中得到词向量 $W$ , softmax的参数 $U$ 和 $b$ ,以及段落向量/句向量 $D$
- 推断过程 (inference stage) , 对于新的段落，得到其向量表达。具体地，在矩阵 $D$ 中添加更多的列，在固定 $WW, UU, bb$ 的情况下，利用上述方法进行训练，使用梯度下降的方法得到新的 $D$ ,从而得到新段落的向量表达。

## 2. Paragraph Vector without word ordering: Distributed bag of words

还有一种训练方法是忽略输入的上下文，让模型去预测段落中的随机一个单词。就是在每次迭代的时候，从文本中采样得到一个窗口，再从这个窗口中随机采样一个单词作为预测任务，让模型去预测，输入就是段落向量。如下所示：



我们称这种模型为 Distributed Bag of Words version of Paragraph Vector(PV-DBOW)

在上述两种方法中，我们可以使用PV-DM或者PV-DBOW得到段落向量/句向量。对于大多数任务，PV-DM的方法表现很好，但我们也强烈推荐两种方法相结合。

## 3. 基于gensim的doc2vec实践

我们使用第三方库gensim进行doc2vec模型的训练

```
1 # -*- coding: utf-8 -*-
2 import sys
3 import logging
4 import os
```

```
5 import gensim
6 # 引入doc2vec
7 from gensim.models import Doc2Vec
8 curPath = os.path.abspath(os.path.dirname(__file__))
9 rootPath = os.path.split(curPath)[0]
10 sys.path.append(rootPath)
11 from utilities import ko_title2words
12
13 # 引入日志配置
14 logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s',
15
16 # 加载数据
17 documents = []
18 # 使用count当做每个句子的“标签”，标签和每个句子是一一对应的
19 count = 0
20 with open('../data/titles/ko.video.corpus','r') as f:
21     for line in f:
22         title = unicode(line, 'utf-8')
23         # 切词，返回的结果是列表类型
24         words = ko_title2words(title)
25         # 这里documents里的每个元素是二元组，具体可以查看函数文档
26         documents.append(gensim.models.doc2vec.TaggedDocument(words, [s
27 count += 1
28 if count % 10000 == 0:
29     logging.info('{} has loaded...'.format(count))
30
31 # 模型训练
32 model = Doc2Vec(documents, dm=1, size=100, window=8, min_count=5, worke
33 # 保存模型
34 model.save('models/ko_d2v.model')
```

接下来看看训练好的模型可以做什么

```
1 def test_doc2vec():
2     # 加载模型
3     model = doc2vec.Doc2Vec.load('models/ko_d2v.model')
4     # 与标签‘0’最相似的
5     print(model.docvecs.most_similar('0'))
6     # 进行相关性比较
```

```
7     print(model.docvecs.similarity('0','1'))
8     # 输出标签为'10'句子的向量
9     print(model.docvecs['10'])
10    # 也可以推断一个句向量(未出现在语料中)
11    words = u"여기 나오는 팀 다 가슴"
12    print(model.infer_vector(words.split()))
13    # 也可以输出词向量
14    print(model[u'가슴'])
```

以上都是一些比较常见的用法，更多的用法请参  
<https://radimrehurek.com/gensim/models/doc2vec.html>

方便交流学习，备注：**昵称-学校（公司）-方向**，进入DL&NLP交流群。

方向有很多：**机器学习、深度学习，python，情感分析、意见挖掘、句法分析、机器翻译、人机对话、知识图谱、语音识别等。**



记得备注哟

#### 推荐阅读：

【ACL 2019】腾讯AI Lab解读三大前沿方向及20篇入选论文

【一分钟论文】IJCAI2019 | Self-attentive Biaffine Dependency Parsing

【一分钟论文】NAACL2019-使用感知句法词表示的句法增强神经机器翻译

【一分钟论文】Semi-supervised Sequence Learning半监督序列学习

【一分钟论文】Deep Biaffine Attention for Neural Dependency Parsing

详解Transition-based Dependency parser基于转移的依存句法解析器

经验 | 初入NLP领域的一些小建议

学术 | 如何写一篇合格的NLP论文

干货 | 那些高产的学者都是怎样工作的？

一个简单有效的联合模型

近年来NLP在法律领域的相关研究工作