5.词向量

- 文本是由词和字组成的，想将文本转化为向量，首先要能够把词和字转化为向量

- 所有向量应该有同一维度n，我们可以称这个n维空间是一个语义空间

我 [0.78029002 0.77010974 0.07479124 0.4106988]
爱 [0.14092194 0.63690971 0.73774712 0.42768218]
北京 [0.95780568 0.51903789 0.76615855 0.6399924]
天安门 [0.73861383 0.49694373 0.13213538 0.41237077]

1. 方式

a. one-hot编码

- 首先统计一个字表或词表，选出n个字或词

- 今天 [1, 0, 0, 0, 0]
- 天气 [0, 1, 0, 0, 0]
- 真 [0, 0, 1, 0, 0]
- 不错 [0, 0, 0, 1, 0]
- 。 [0, 0, 0, 0, 1]

- 今天 不错 [1, 0, 0, 1, 0] 今天 真 不错 [1, 0, 1, 1, 0]

- 在对文本向量化时，也可以考虑词频
- 不错 [0, 0, 0, 1, 0]
- 不错 不错 [0, 0, 0, 2, 0]

- 有时也可以不事先准备词表，临时构建
- 如做文本比对任务，成对输入，此时维度可随时变化

- 例1: 你好吗心情 例2: 我不知道谁呀
- A: 你好吗 [1, 1, 1, 0, 0] A:我不知道 [1, 1, 1, 1, 0, 0]
- B: 你心情好吗 [1, 1, 1, 1, 1] B:谁知道呀 [0, 0, 1, 1, 1, 1]

本质上是数出现的公共词

缺点:

1. 有很多词的话，向量编码维度高，且十分稀疏，计算负担大
2. 不能返回语义相似性

b. word2vec – 词向量

- 我们希望得到一种词向量，使得向量关系能反映语义关系，比如：
- cos（你好， 您好） > cos(你好，天气）
- 即词义的相似性反映在向量的相似性

- 国王 - 男人 = 皇后 -女人
- 即向量可以通过数值运算反映词之间的关系

- 同时，不管有多少词，向量维度应当是固定的

word2vec / word embedding 一个意思

- 将整个embedding矩阵看作一个线性层
- Onehot 编码作为输入

word embedding 权重矩阵

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

$W \in R^{wordLen \times embedSize}$

$wordLen$ 表示词表大小

这时所有词的维度都会映射到embedSize，不会太大

我们需要训练词向量权重矩阵 W （以前成为词查找表look-up table），这里难点在于没有标记好的label

i. 训练方法：基于语言模型

- 做出假设：
- 每段文本中的某一个词，由它前面n个词决定

- 例如：
- 今天 天气 不错 我们 出去 玩
- 今天 -> 天气
- 今天 天气 -> 不错
- 今天 天气 不错 -> 我们
- 今天 天气 不错 我们 -> 出去

语言模型实际上是设计了一个任务：由前n个词，训练下一个词（不需要人为标注

同时学习**词向量中的参数**和最后输出的**概率**

ii. 基于窗口训练

- 做出假设：
- 如果两个词在文本中出现时，它的前后出现的词相似，则这两个词语义相似。

- 如：
- 你 想 明白 了 吗　　今天 北京 很热
- 你 想 清楚 了 吗　　今天 南京 很热

窗口长度 = 2
目标词 = 明白 / 清楚

窗口长度 = 1
目标词 = 北京 / 南京

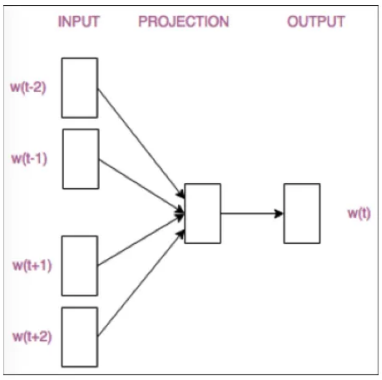
窗口长度是人为设定的

1. 窗口长度大，会导致向量维度大，计算复杂
2. 窗口长度小，丢失语义信息

- 基于前述思想，我们尝试用窗口中的词（或者说周围词）来表示（预测）中间词

CBOW模型

窗口：你 想 明白 了 吗
输入：你 想 了 吗
输出：明白



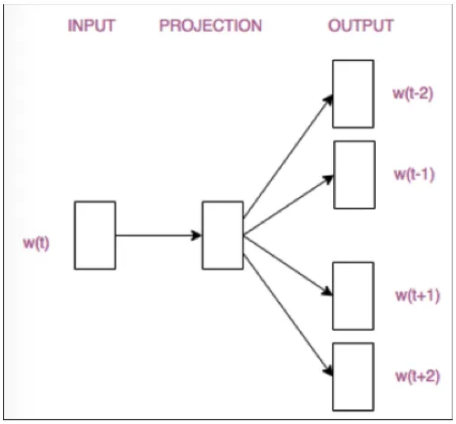
CBOW：周围词 ---> 中间词

- 或用中间词来表示周围词

SkipGram模型

窗口：你 想 明白 了 吗

(输入,输出)：（明白,你）
（明白,想）
（明白,了）
（明白,吗）



CBOW：周围词 -----> 中间词

SkipGram：中间词 ----> 周围词

CBOW：有两处可训练 1. embedding层 2. 线性层

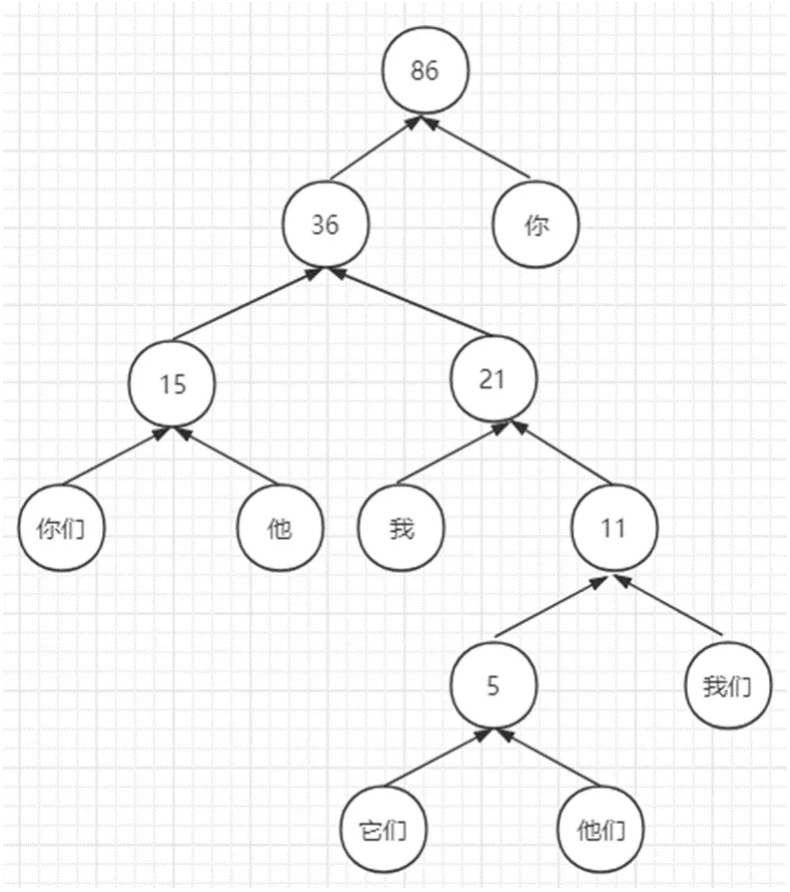
2. Huffman树编码

- 1）不同词编码不同

- 2）每个词的编码不会成为另一个词编码的前缀
- 即如果某个词编码为011，则不能有词的编码是0111或0110或011001等

- 3）构造出的词编码总体长度最小，且越高频词编码越短

- 你 50
- 我 10
- 他 8
- 你们 7
- 我们 6
- 他们 3
- 它们 2



每次选取最小的两个频次的字组合起来

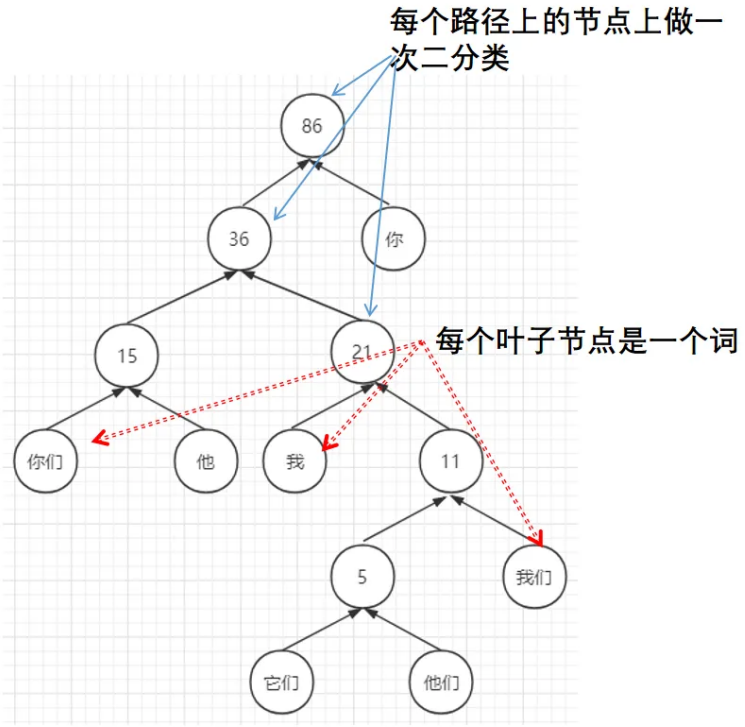
然后基于one-hot的词向量训练会导致维度灾难！解决：huffman树

这样，带分类的维度将大大降低！从而使得线性分类器的参数大大减少

huffman树的分类过程其实可以用多个二分类器来实现：

Huffman树

- 最终编码
- 你 1
- 我 010
- 他 001
- 你们 000
- 我们 0111
- 他们 01101
- 它们 01100



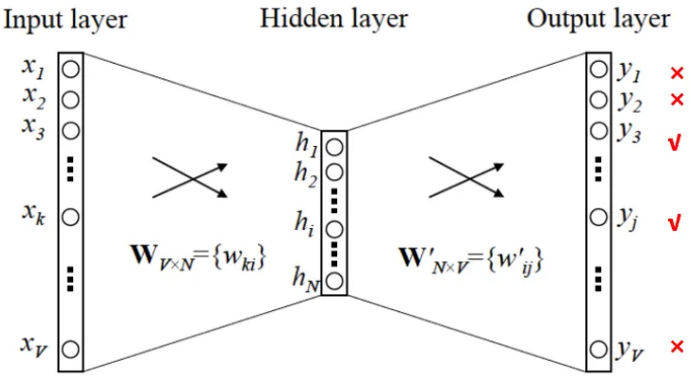

```python
self.cls1 = nn.Linear(embedding_size, 2)
self.cls2 = nn.Linear(embedding_size, 2)
self.cls3 = nn.Linear(embedding_size, 2)
self.cls4 = nn.Linear(embedding_size, 2)
self.cls5 = nn.Linear(embedding_size, 2)
#其实具体可以这样做
for i in range(100):
    layer.append(nn.Linear(embedding_size, 2))
layer[1]
layer[2]
```

1. 基于语言模型和基于窗口都是基于不合理的假设，但仍然能解决很多问题

3. 负采样

在最后的输出层中，不对输出层全部的元素进行softmax和求交叉熵，只选取一部分进行反向传播。

- 使用sigmoid函数逐个计算概率，代替softmax
- 只更新选择的部分词的权重矩阵



4. Glove 基于共现矩阵

NLP学习语义的重要部分是找到一个训练目标

a. 共现矩阵

- 基于共现矩阵

- 语料:
- 今天 天气 不错
- 今天 天气 很 好
- 天气 很 好
- 天气 不错

- 选取窗口长度1

	今天	天气	很	好	不错
今天	0	2	0	0	0
天气	2	0	2	0	2
很	0	2	0	2	0
好	0	0	2	0	0
不错	0	2	0	0	0

今天 & 天气共同出现的次数为2

b. 共现概率

- 共现概率

$$P_{ij} = P(j|i) = \frac{x_{ij}}{x_i}$$

- 词j出现在词i周围
- 的概率，被称为
- 词i和词j的
- 共现概率
- P(天气|今天) =
- 2 / 2 = 1

	今天	天气	很	好	不错
今天	0	2	0	0	0
天气	2	0	2	0	2
很	0	2	0	2	0
好	0	0	2	0	0
不错	0	2	0	0	0

P(天气|今天):

表示天气出现在今天的次数占天气本身出现次数的比例

c. 共现概率比

- 共现概率比
- 两个共现概率的比值

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

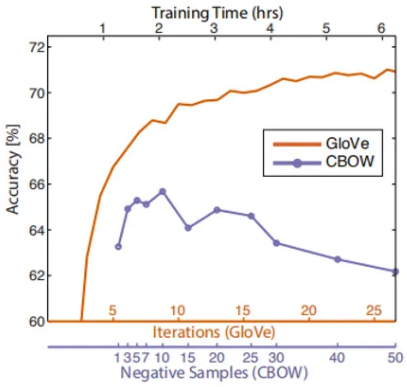
P(solid|ice) / P(solid|steam)

- 如果词A与词B的相关性，大于词A与词C的相关性，
- 则共享概率比 P(A|B)/P(A|C) 会较高，反之亦然

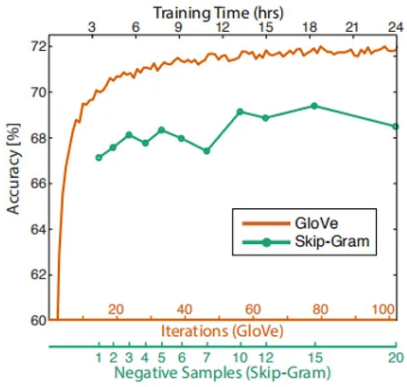
solid（固体）跟ice（冰）更接近，因此共现概率比更高。反之，gas（气体）跟steam（蒸汽）更接近。

反应的是词之间的相关性

- Glove通过共现矩阵，让模型看到了整个文本的信息，而word2vec模型一直在看某个窗口



(a) GloVe vs CBOW



(b) GloVe vs Skip-Gram

d. 损失函数

- 问题转化：
- 给定三个词的词向量，V_a, V_b, V_c三者的通过某个函数映射后，其比值应接近ABC的共现概率比
- 即目标为找到向量使得 $f(V_a, V_b, V_c) = P(A|B)/P(A|C)$
- 预测数值，属于回归问题，损失函数使用均方差
- f的设计论文中给出的是$f(V_a, V_b, V_c) = (V_a - V_b)\cdot V_c$

5. 词向量训练总结

- 一、根据词与词之间关系的某种假设，制定训练目标
- 二、设计模型，以词向量为输入
- 三、随机初始化词向量，开始训练
- 四、训练过程中词向量作为参数不断调整，获取一定的语义信息
- 五、使用训练好的词向量做下游任务

训练过程其实是一厢情愿的事情，因为实际上是拿着公式（网络框架），去拟合里面的参数

a. 词向量存在的问题

- 1)词向量是“静态”的。每个词使用固定向量，没有考虑前后文
- 2)一词多义的情况。西瓜 - 苹果 - 华为
- 3）影响效果的因素非常多

  维度选择、随机初始化、skip-gram/cbow/glove、分词质量、词频截断、未登录词、窗口大小、迭代轮数、停止条件、语料质量等
- 4）没有好的直接评价指标。常需要用下游任务来评价

1. “静态”指的是，在词向量中，每个词的向量是固定的，但其实词在不同语境下会出现不同的含义
2. 词向量需要下游任务评价

词向量现在有成熟的框架进行训练，下面demo用来找词的相似词

word2vec_train Python

```python

import json
import jieba
import numpy as np
import gensim
from gensim.models import Word2Vec
from collections import defaultdict

'''
词向量模型的简单实现
'''

#训练模型
#corpus: [["cat", "say", "meow"], ["dog", "say", "woof"]]
#corpus: [["今天", "天气", "不错"], ["你", "好", "吗"]]
#dim指定词向量的维度，如100
def train_word2vec_model(corpus, dim):
    model = Word2Vec(corpus, vector_size=dim, sg=1)
    model.save("model.w2v")
    return model

#输入模型文件路径
#加载训练好的模型
def load_word2vec_model(path):
    model = Word2Vec.load(path)
    return model

def main():
    sentences = []
    with open("corpus.txt", encoding="utf8") as f:
        for line in f:
            sentences.append(jieba.lcut(line))
    model = train_word2vec_model(sentences, 100)
    return model

if __name__ == "__main__":
    model = main()  #训练

    model = load_word2vec_model("model.w2v")  #加载

    print(model.wv.most_similar(positive=["男人", "母亲"], negative=["女人"]
)) #类比

    while True:  #找相似
        string = input("input:")
```

```
        try:
            print(model.wv.most_similar(string))
        except KeyError:
            print("输入词不存在")
```

6. 词向量的应用

a. 近义词

- 输入：红烧肉

- 注意：
- 依赖分词正确
- 与A最接近的词是B，
  不代表B最接近的是A
- 有时也会有反义词很相似
- 总会有很多badcase

b. 相似句

- 1）将一句话或一段文本分成若干个词

- 2）找到每个词对应的词向量

- 3）所有词向量加和求平均或通过各种网络模型，得到文本向量

- 4）使用文本向量计算相似度或进行聚类等

7. K-Means

- KMeans

- 随机选择k个点作为初始质心
- repeat
- 将每个点指派到最近的质心，形成k个簇
- 重新计算每个簇的质心
- until
- 质心不发生变化

1. K过大，本应是同一类被分为不同类（重复的比较多
2. K过小，本应是不同类，被分为同一类（热点问题被覆盖

NLP中应该优先选择更多的类别进行聚类

- KMeans一些使用技巧：
- 先设定较多的聚类类别
- 聚类结束后计算类内平均距离
- 排序后，舍弃类内平均距离较长的类别

- 计算距离时可以尝试欧式距离、余弦距离或其他距离

- 短文本的聚类记得先去重，以及其他预处理

8. 词向量总结

1. 质变：将离散的字符转化为连续的数值
2. 通过向量的相似度代表语义的相似度
3. 词向量的训练基于很多不完全正确的假设，但是据此训练的词向量是有意义的
4. 使用无标注的文本的一种好方法

后来的NLP任务实际上是将词向量和下游任务一起训练