

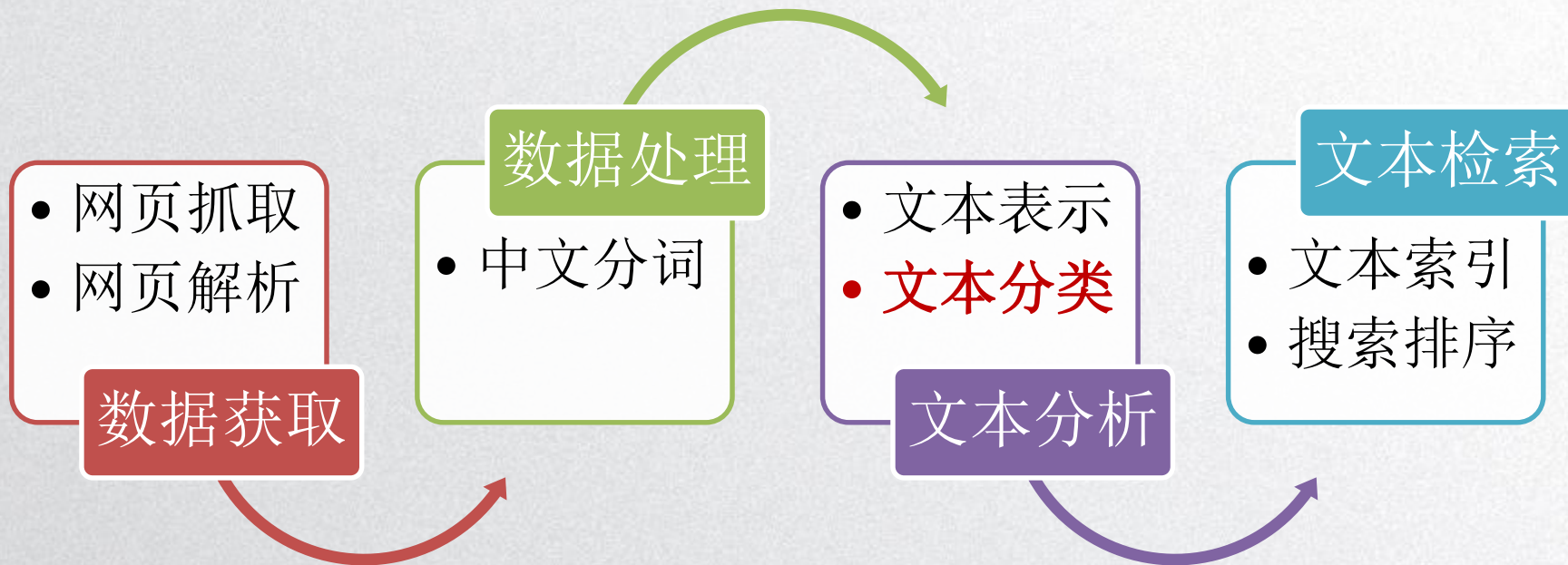


# 文本表示2: Word2Vector



覃雄派

# 文本模块涉及的内容





# 提纲

- Word2Vec
  - CBOW
  - Skip-gram
- Word2Vec的效果



文本表示2:  
Word2Vector

# 文本表示2: Word2Vector

- Word2Vec
  - 关注于**单词的分布式表达** (后来的doc2vec考虑了文档表达)
  - 核心想法:
    - 两个单词的相似性 = 其对应的**词向量的相似度**
    - 对文档集中的每一个词 (中心词), 计算它周围的词 (上下文) 的相似度
      - (1) 用中心词去**→预测上下文所出现的单词**
      - (2) 或者, **用上下文出现的词→预测中心词**

## Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov  
Google Inc.  
Mountain View  
mikolov@google.com

Ilya Sutskever  
Google Inc.  
Mountain View  
ilyasu@google.com

Kai Chen  
Google Inc.  
Mountain View  
kai@google.com

Greg Corrado  
Google Inc.  
Mountain View  
gcorrado@google.com

Jeffrey Dean  
Google Inc.  
Mountain View  
jeff@google.com

papers.nips.cc › paper › 5021-distributed-representatio... ▼ PDF

## Distributed Representations of Words and Phrases and their ...

In this **paper** we present several extensions of the original Skip-gram model. ...

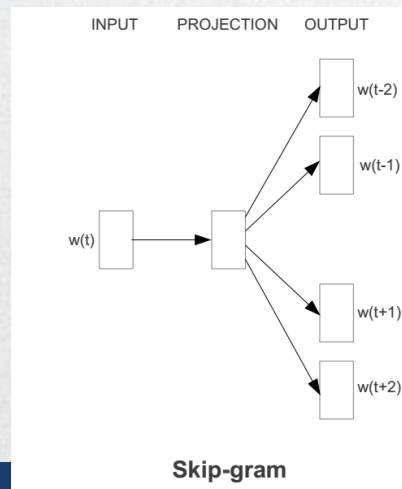
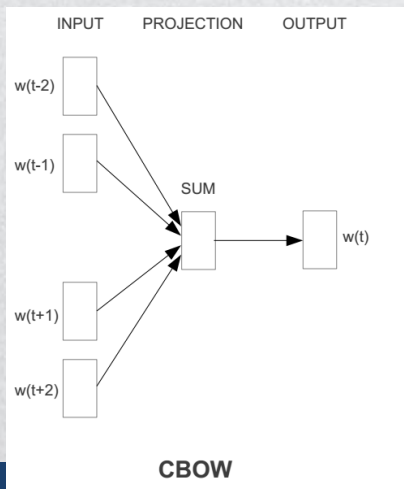
1code.google.com/p/**word2vec**/source/browse/trunk/questions-words.txt. 5 ...

by T Mikolov - 2013 - Cited by 18862 - Related articles



# 文本表示2: Word2Vector

- Word2vec两大类基本模型
  - 两大基本模型:
    - 用上下文出现的单词→去预测中心词
      - Continuous Bag of Word (CBOW)
    - 用中心词→去预测上下文出现的单词
      - Skip-gram (SG)

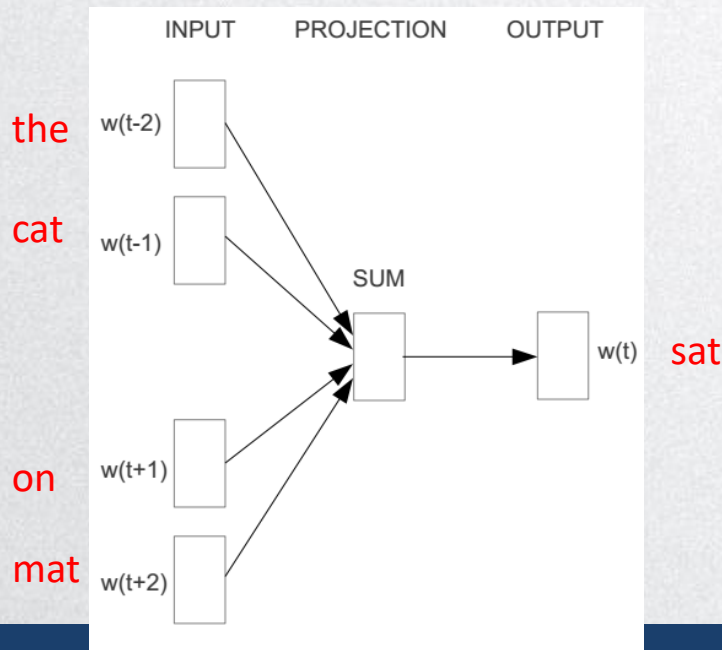


# 文本表示1: TF-IDF、SVD



## 文本表示2: Word2Vector

- 以Continuous Bag of Word(CBOW)为例
  - 例句: "The cat sat on mat"
  - 上下文窗口大小 = 2 (上文和下文均取同一句话中的两个单词)

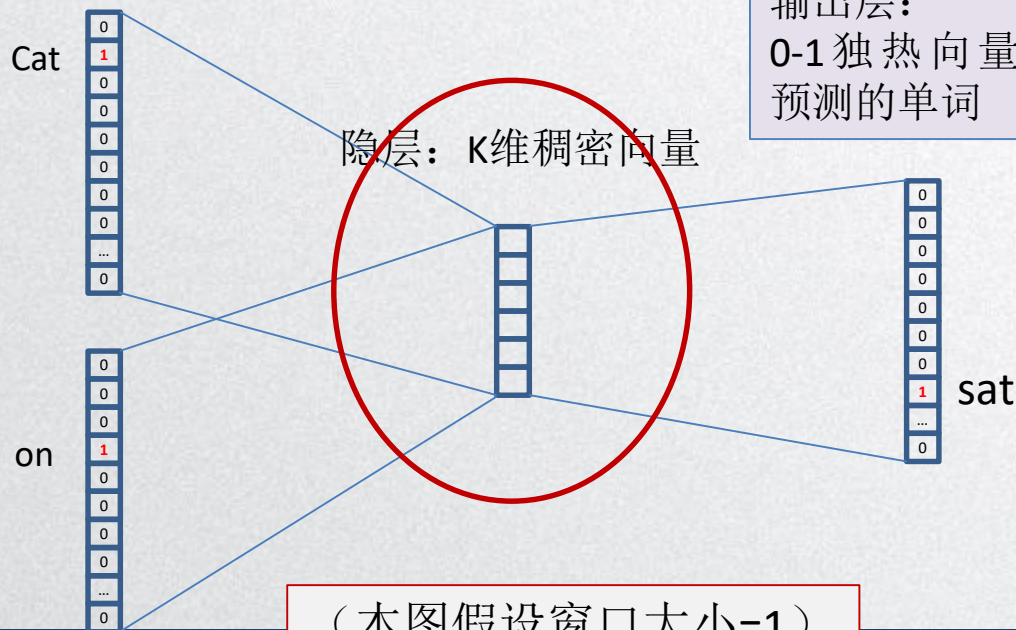


# 文本表示2: Word2Vec

- Word2Vec
  - CBOW预测网络结构

输入层：独热表示

0-1 独热向量，每一个单词占据一个维度



输出层：  
0-1 独热向量，表示  
预测的单词

(本图假设窗口大小=1)

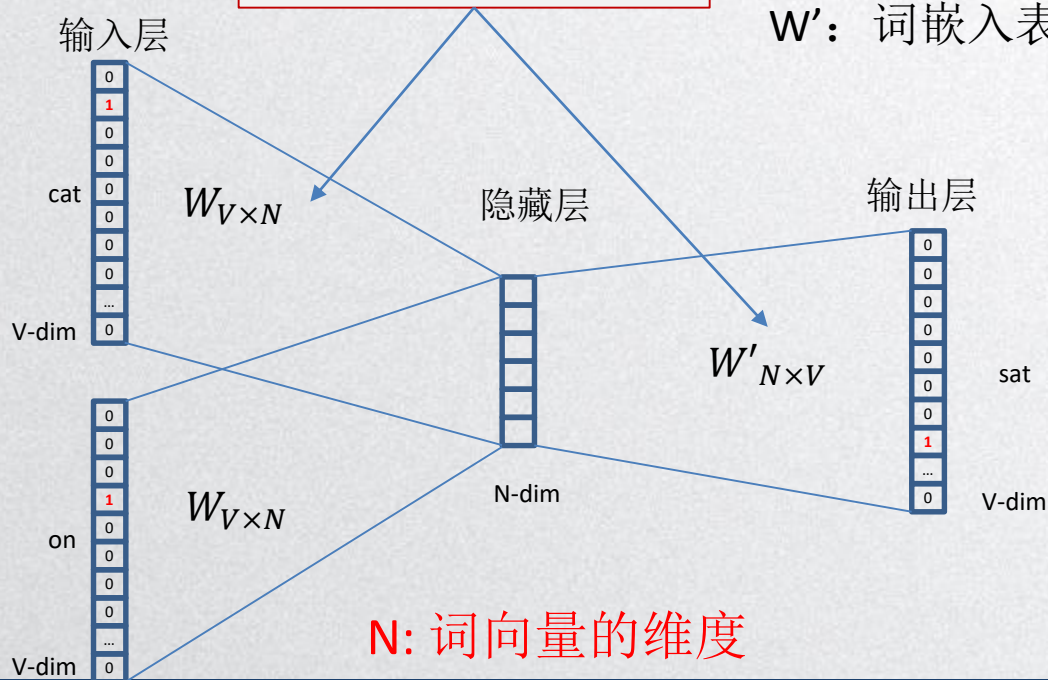


# 文本表示2: Word2Vector

- Word2Vec
  - CBOW参数

模型参数:  $W$ 和 $W'$

$W$ : 独热表示  $\rightarrow$  词嵌入表示  
 $W'$ : 词嵌入表示  $\rightarrow$  独热表示

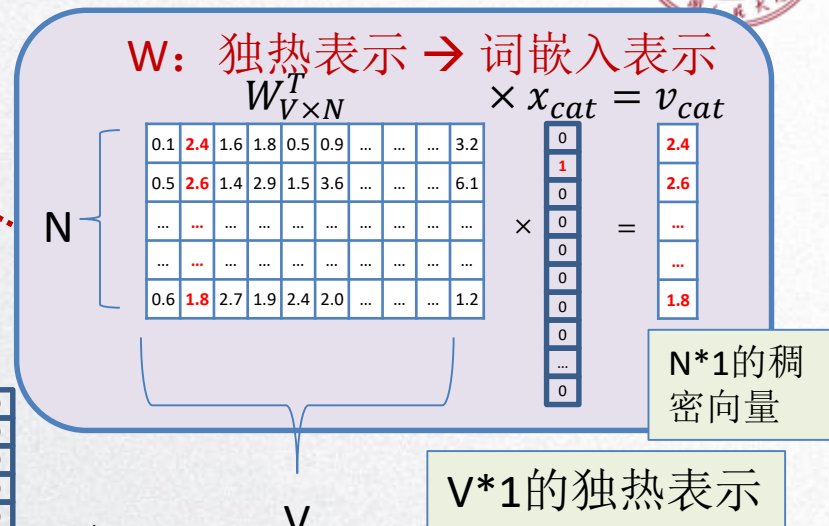
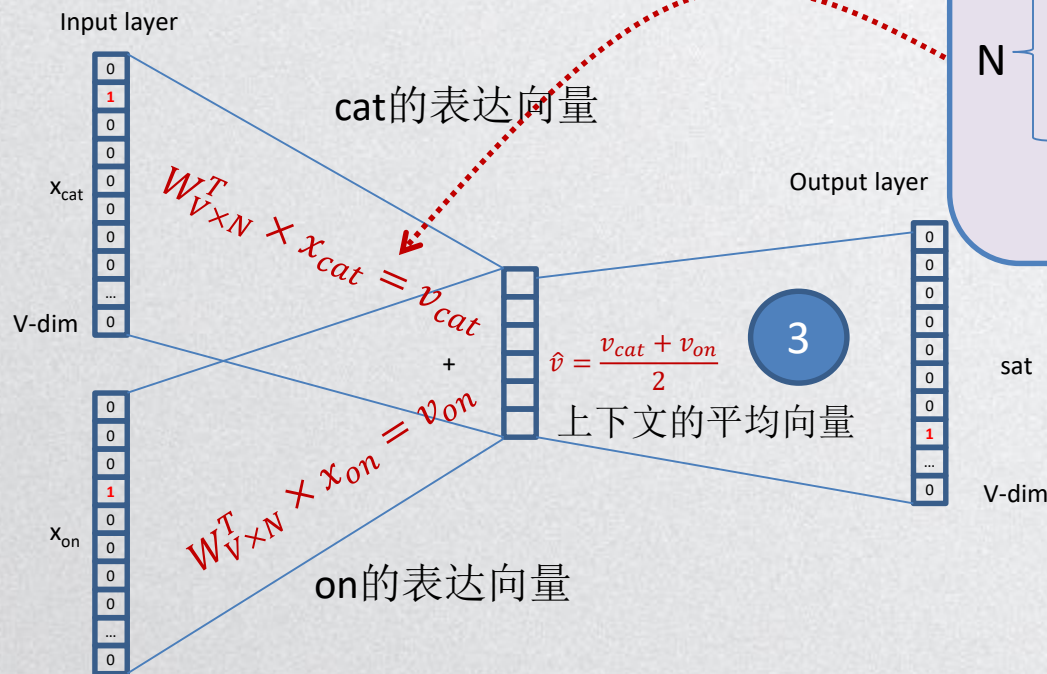




# 文本表示2: Word2Vector

## • Word2Vec

- 隐层的计算: 上下文的平均向量

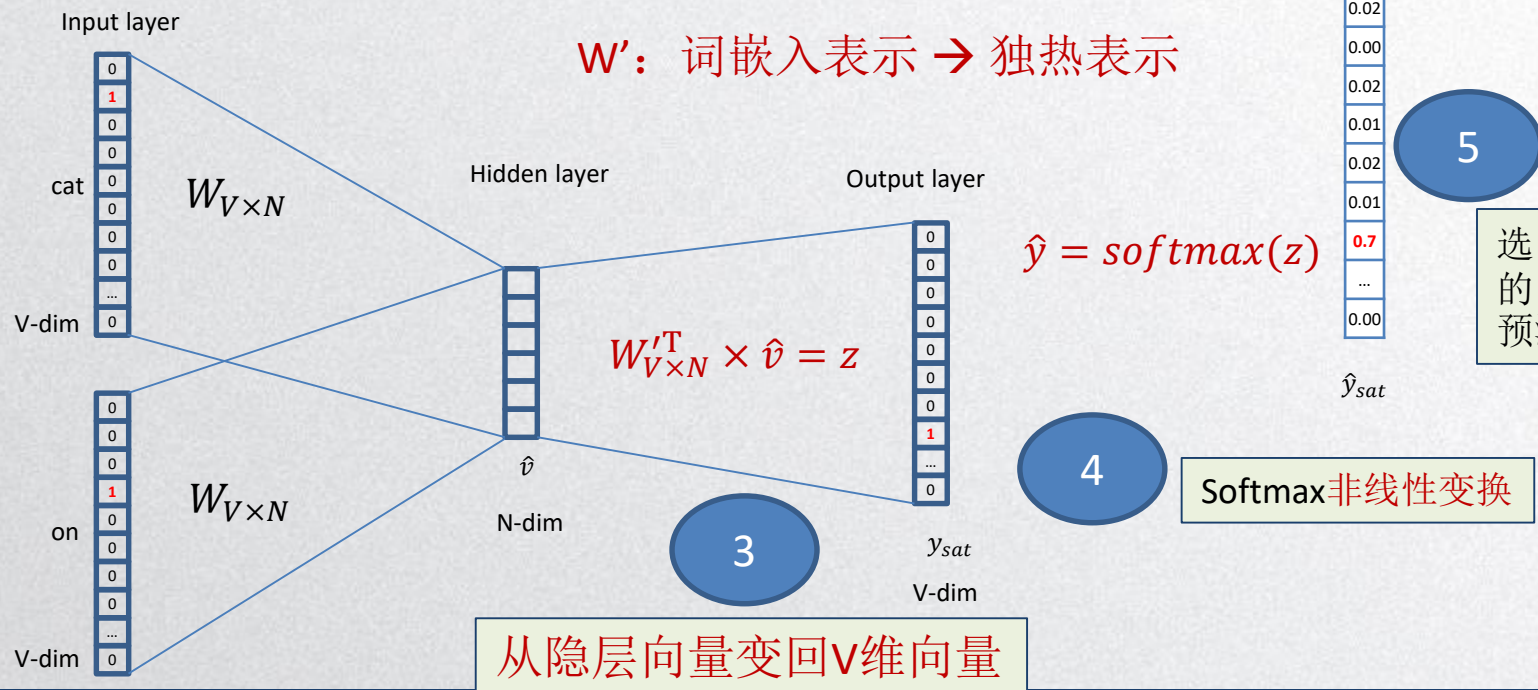


# 文本表示2: Word2Vector

## • Word2Vec

- 从隐层到输出层: 选择一个预测单词

训练目标: 选择合适的 $W$ 和 $W'$ 参数, 使得预测到sat的概率最大





# 文本表示2: Word2Vec

- Word2Vec
  - CBOW学习过程: 上下文训练数据构造

原始文本

生成的训练数据

The quick brown fox jumps over the lazy dog. →

(the, quick)  
(the, brown)

The quick brown fox jumps over the lazy dog. →

(quick, the)  
(quick, brown)  
(quick, fox)

窗口大小=2

The quick brown fox jumps over the lazy dog. →

(brown, the)  
(brown, quick)  
(brown, fox)  
(brown, jumps)

The quick brown fox jumps over the lazy dog. →

(fox, quick)  
(fox, brown)  
(fox, jumps)  
(fox, over)



# 文本表示2: Word2Vector

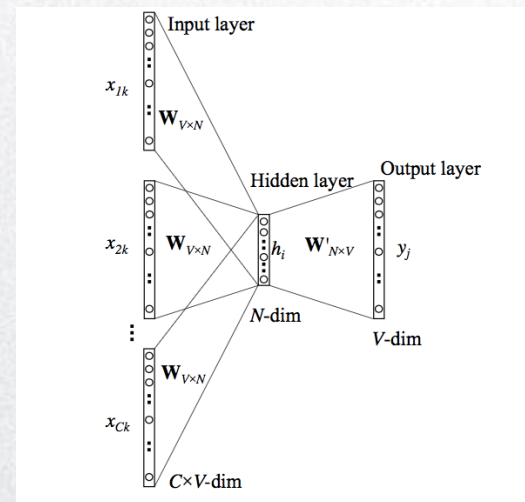
- Word2Vec

- CBOW学习过程

- 举个例子

- "... an efficient method for learning high quality distributed vector ..."

- 对于如下的句子, 如果我们取上下文窗口大小为4,
      - 那么对于Learning这个词项来讲, 它就是要输出的词项
      - 而输入的词项有8个, 即前面有4个, 后面有4个



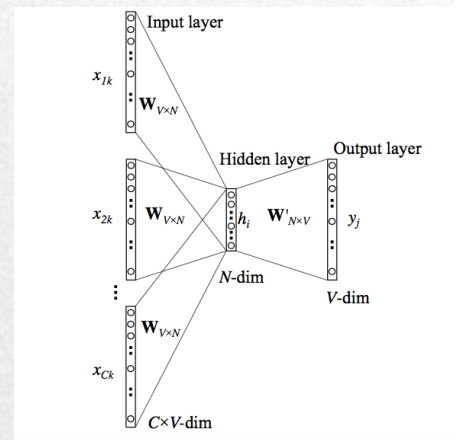
# 文本表示2: Word2Vector

- Word2Vec

- CBOW学习过程
- 举个例子

- “... an efficient method for learning high quality distributed vector ...”

- CBOW的计算过程如下, 注意 $|V|$ 的大小有可能达到10万, 甚至100万级别。
- (1) 输入数据是8个 $V \times 1$ 的向量, 首先求均值 $\bar{x}$ 。
- $W$ 的大小是 $V \times N_{\text{dim}}$ ,  $N_{\text{dim}}$ 是一个为了创建稠密的向量表示而设定的一个维度,  $V$ 可能是100,000, 而 $N_{\text{dim}}$ 可能是300,  $N_{\text{dim}} \ll V$ 。
- $$h = \frac{1}{C} W^T \sum_{c=1}^C x^{(c)} = W^T \bar{x}$$
 (注:  $N_{\text{dim}} * V * V * 1$ )



# 文本表示2: Word2Vector

- Word2Vec

- CBOW学习过程
- 举个例子

- “... an efficient method for learning high quality distributed vector ...”

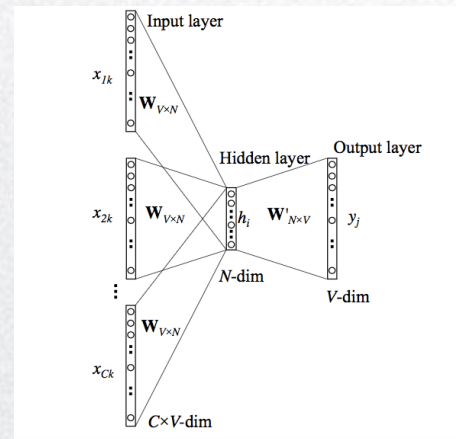
- (2) 在h基础上计算u

- $$u = W'^T h = \frac{1}{C} \sum_{c=1}^C W'^T W^T x^{(c)} = W'^T W^T \bar{x} \quad (\text{注:}$$

$V * N_{\text{dim}} * N_{\text{dim}} * 1$ )

- $W'$ 是一个  $N_{\text{dim}} * V$  的权重矩阵

- 可以注意到,  $W'^T W^T \bar{x}$  向量计算的维度关系为  $(V * N_{\text{dim}}) * (N_{\text{dim}} * V) * (V * 1)$ , 所以输出是一个  $V * 1$  的向量



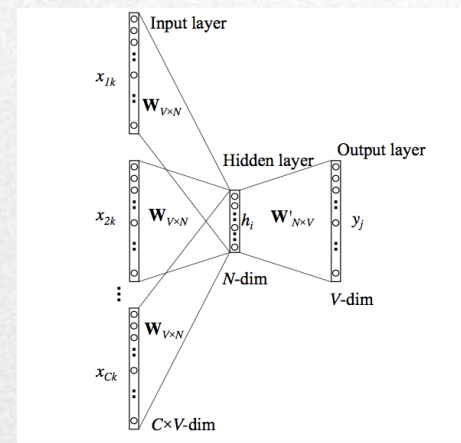
# 文本表示2: Word2Vector

- Word2Vec


- CBOW学习过程
- 举个例子

- "... an efficient method for learning high quality distributed vector ..."

- (3) 最后进行Softmax规范化
- $y = \text{Softmax}(u) = \text{Softmax}(W'^T W^T \bar{x})$
- 输入是Learning的8个上下文词项对应的One Hot Encoding, 要求Softmax输出和Learning的One Hot Encoding的误差尽可能小, 即只有对应Learning的位置为1, 其余位置为0, 作为优化目标, 来训练该网络




采用交叉熵损失函数, 请参考这2个文档

 2021-new-Word2Vector详解 (skip-gram)的正向传导和反向传播.docx

0307-神经网络入门、反向传播算法 (基于一个简单的神经网络)

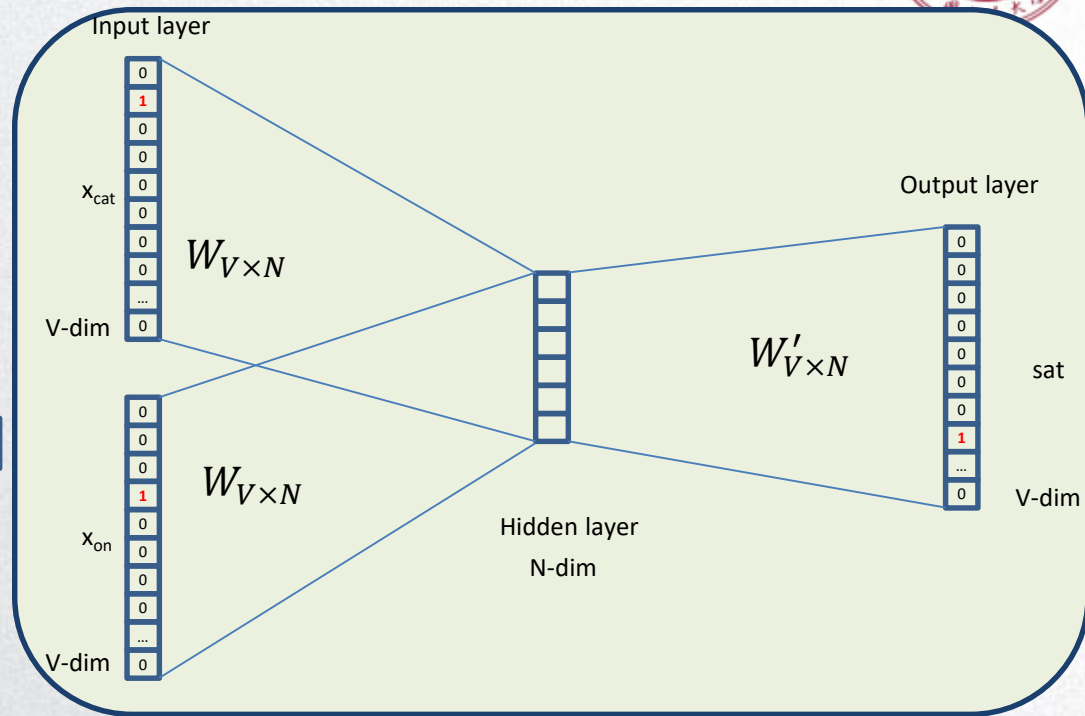
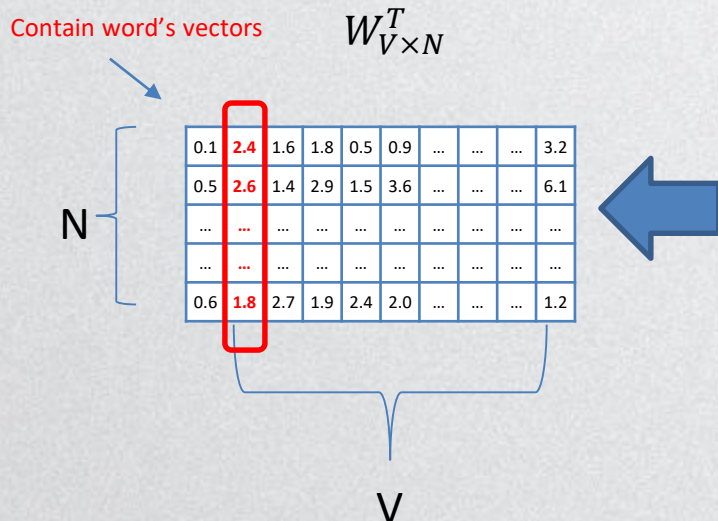
名称

 misc交叉熵.pptx



# 文本表示2: Word2Vector

- Word2Vec
  - CBOW
  - 学习到的词向量:  $W$ 或者 $W'$



$W$ 和 $W'$ 中的向量都可以作为词向量（或者它们的平均值）


# 文本表示1: TF-IDF、SVD





# 文本表示1: TF-IDF、SVD

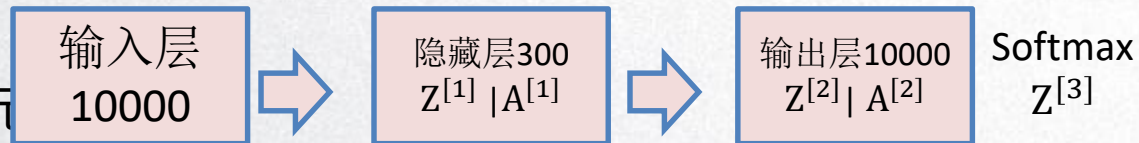
- CBOW的前向传导和反向传播
  - 准备工作, 请复习

名称	类型	大小	修改日期
 0308-神经网络MLP二值分类、多类别分类、回归（不同损失函数，反向传播算法）.pptx	Microsoft Pow...	2,930 KB	2021/11/25 13:25

- 假设词汇表的单词数量 $V$ 为10000
- 低维空间的维度为300
- 上下文大小为4, 即用前边4个单词和后边4个单词, 预测当前单词

## 文本表示1: TF-IDF、SVD

- 一个针对CBOW的多层MLP
  - 神经网络为输入层有10000个神经元
  - 隐藏层有300个神经元
  - 输出层为10000个神经元

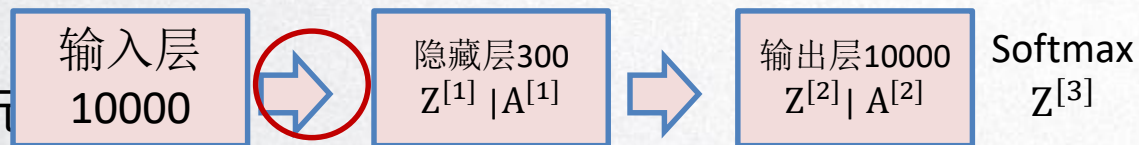




## 文本表示1: TF-IDF、SVD

- 一个针对CBOW的多层MLP

- 神经网络为输入层有10000个神经元
- 隐藏层有300个神经元
- 输出层为10000个神经元
- Layer1的前向传导过程具体如下



$X=(1, 10000)$  8个独热编码样本，求平均，得到一个样本，这个样本是10000维的

$$W^{[1]}=(300, 10000)$$

$$A^{[0]} = X^T=(10000, \mathbf{1})$$

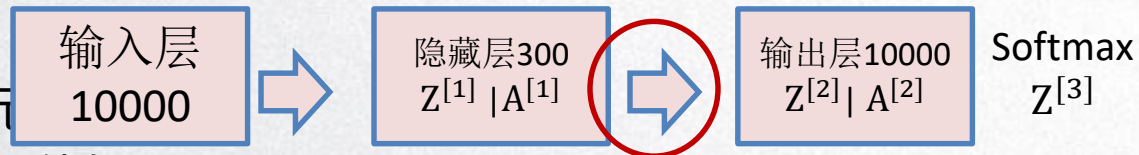
$$Z^{[1]} = W^{[1]}X=W^{[1]}A^{[0]}=(300,10000) \times (10000, \mathbf{1})=(300, \mathbf{1})$$

$$A^{[1]} = \sigma(Z^{[1]})=Z^{[1]}=(300, \mathbf{1})$$

## 文本表示1: TF-IDF、SVD

- 一个针对MNIST数据集二值分类的多层MLP

- 神经网络为输入层有10000个神经元
- 隐藏层有300个神经元
- 输出层为10000个神经元
- Layer2的前向传导过程具体如下



$$W^{[2]} = (\text{10000}, 300)$$

$$Z^{[2]} = W^{[2]}A^{[1]} = (\text{10000}, 300) \times (300, \text{1}) = (\text{10000}, \text{1})$$

$$A^{[2]} = \sigma(Z^{[2]}) = Z^{[2]} = (\text{10000}, \text{1})$$

接着进行Softmax转换

$$Z^{[3]} = \text{Softmax}(A^{[2]}) = (\text{10000}, \text{1})$$

$$L = -\sum_{i=1}^c y_i \log z_i^{[3]}$$

根据Softmax的导数

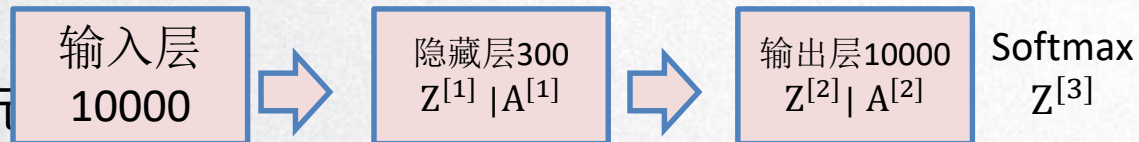
$$\text{有 } \frac{dL}{dA^{[2]}} = Z^{[3]} - Y = (\text{10000}, \text{1})$$



## 文本表示1：TF-IDF、SVD

- 一个针对MNIST数据集二值分类的多层MLP

- 神经网络为输入层有10000个神经元
- 隐藏层有300个神经元
- 输出层为10000个神经元
- Layer2的反向传播过程具体如下



$$Y^T = (10000, 1)$$

$$dA^{[2]} = \frac{dL}{dA^{[2]}} = Z^{[3]} - Y = (10000, 1), \text{ 注意, 矩阵的各个位置相减 (} A^{[2] \text{ 参考上页)}$$

$$dZ^{[2]} = dA^{[2]} g'(Z^{[2]}) = (10000, 1) * (10000, 1) = (10000, 1), \text{ 注意, 矩阵的各个位置相乘}$$

$$dW^{[2]} = dZ^{[2]} (A^{[1]})^T = (10000, 1) \times (1, 300) = (10000, 300), \text{ 注意, 是矩阵乘法}$$

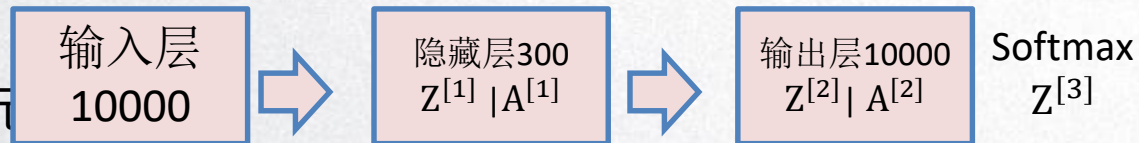
$$dA^{[1]} = (W^{[2]})^T dZ^{[2]} = (300, 10000) (10000, 1) = (300, 1)$$

10000行1列, 所有元素都是1

## 文本表示1: TF-IDF、SVD

- 一个针对MNIST数据集二值分类的多层MLP

- 神经网络为输入层有10000个神经元
- 隐藏层有300个神经元
- 输出层为10000个神经元
- Layer1的反向传播过程具体如下



$dZ^{[1]} = dA^{[1]}g'(Z^{[1]}) = (300, \mathbf{1}) * (300, \mathbf{1}) = (300, \mathbf{1})$ , 注意, 矩阵的各个位置相乘  
 $dW^{[1]} = dZ^{[1]}(A^{[0]})^T = (300, 1) \times (\mathbf{1}, 10000) = (300 * 10000)$ , 注意, 是矩阵乘法

300行1列, 所有  
元素都是1

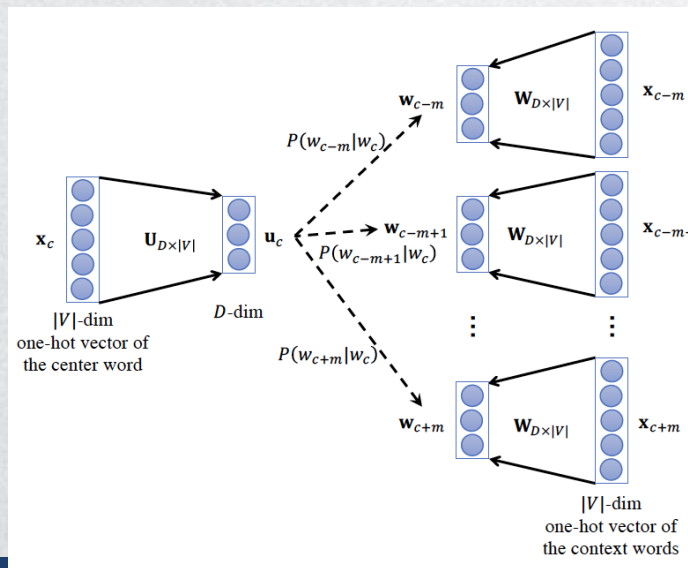


# 文本表示1: TF-IDF、SVD



# 文本表示2: Word2Vector

- Word2Vec
  - Skip-Gram
    - **优化目标构建**: 扫描文本数据集, 对遇到的每一个单词
    - 通过当前单词预测从 **-c 到 c** 的窗口中的单词的概率



## 文本表示2: Word2Vector

- Word2Vec
  - Skip-Gram的优化

由中心词 $w_c$ 的表达  
逐个预测其 $2m$ 个上下文词

$$\begin{aligned}\arg \min_{\mathbf{U}, \mathbf{W}} \ell &= -\log \prod_c \prod_{j=0; j \neq m}^{2m} P(w_{c-m+j} | w_c) \\ &= -\log \prod_c \prod_{j=0; j \neq m}^{2m} \frac{\exp\{\mathbf{w}_{c-m+j}^T \mathbf{u}_c\}}{\sum_{k=1}^{|V|} \exp\{\mathbf{w}_k^T \mathbf{u}_c\}}.\end{aligned}$$



# 文本表示1: TF-IDF、SVD





# 文本表示1: TF-IDF、SVD

- Skip-Gram的前向传导和反向传播
  - 和CBOW相比, Skip的样本构造不一样, 如下所示
  - 比如 "... an efficient method for learning high quality distributed vector ..."

## CBOW的样本

前4个单词+后4个单词, 预测当前单词

X: an efficient method for high quality distributed vector的独热表示(求平均)

Y: learning的独热表示

## Skip-Gram的样本

当前单词, 预测前4个单词+后4个单词

X: learning的独热表示

Y: an efficient method for high quality distributed vector的独热表示

# 文本表示1: TF-IDF、SVD





# 文本表示1: TF-IDF、SVD

- 通过一个小实例了解Skip Gram
  - 现有如下的句子
  - The man who passes the sentence should swing the sword
  - 有10个词项, 8个唯一的词项
    - 于是字典表大小 $|V|$ 为8
    - 每个词项编码为one hot encoding的时候, 是一个8维向量

Man
Passes
Sentence
Should
swing
sword
The
who

需要首先学习神经网络的2个PPT

本实例的具体细节请参考如下文档



2021-new-Word2Vector详解 (skip-gram)的正向传导和反向传播.docx





# 文本表示1: TF-IDF、SVD

- 通过一个小实例了解Skip Gram
  - 现有如下的句子
  - The man who passes the sentence should swing the sword
  - 如果我们设定 $c=1$ , 那么我们针对passes这个词项
  - 就可以把它之前的词项who和它之后的词项the提取出来, 和passes一起构成2个训练样本
    - Passes who
    - Passes the





# 文本表示1: TF-IDF、SVD

- 通过一个小实例了解Skip Gram
  - 现有如下的句子
  - The man who passes the sentence should swing the sword
    - 我们准备利用Skip-Gram进行训练, 把8维的one hot encoding通过训练, 降维为3维

W初始化

Man	-0.078	0.018	0.033
Passes	0.068	0.170	-0.109
Sentence	-0.158	-0.081	-0.151
Should	0.150	0.064	0.145
swing	-0.097	-0.055	0.188
sword	0.036	0.071	0.059
The	0.168	-0.060	-0.058
who	0.098	0.015	0.096

W转置为 $W^{[1]}$

-0.078	0.068	-0.158	0.150	-0.097	0.036	0.168	0.098
0.018	0.170	-0.081	0.064	-0.055	0.071	-0.060	0.015
0.033	-0.109	-0.151	0.145	0.188	0.059	-0.058	0.096

$3 \times 8$

词项passes对应的one hot encoding如下, 它作为输入X, 即

Man	0
Passes	1
Sentence	0
Should	0
swing	0
sword	0
The	0
who	0

$8 \times 1$

# 文本表示1: TF-IDF、SVD

- 通过一个小实例了解Skip Gram
  - 现有如下的句子
  - The man who passes the sentence should swing the sword
    - 我们准备利用Skip-Gram进行训练, 把8维的one hot encoding通过训练, 降维为3维

$$A^{[1]} = Z^{[1]} = W^{[1]} A^{[0]}$$

-0.078	0.068	-0.158	0.150	-0.097	0.036	0.168	0.098
0.018	0.170	-0.081	0.064	-0.055	0.071	-0.060	0.015
0.033	-0.109	-0.151	0.145	0.188	0.059	-0.058	0.096

3\*8

Man	0
Passes	1
Sentence	0
Should	0
swing	0
sword	0
The	0
who	0



8\*1

$A^{[1]}$

0.068
0.170
-0.109

3\*1

# 文本表示1: TF-IDF、SVD

- 通过一个小实例了解Skip Gram
  - 现有如下的句子
  - The man who passes the sentence should swing the sword
    - 我们准备利用Skip-Gram进行训练, 把8维的one hot encoding通过训练, 降维为3维

$W'$ 初始化

0.192	0.070	-0.066	0.014	-0.012	0.013	0.016	-0.028
0.176	0.061	0.117	0.006	0.067	0.111	0.175	-0.016
0.012	-0.046	0.083	-0.044	0.147	-0.097	-0.198	0.148

$W'$  转置为 $W^{[2]}$

0.192	0.176	0.012
0.070	0.061	-0.046
-0.066	0.117	0.083
0.014	0.006	-0.044
-0.012	0.067	0.147
0.013	0.111	-0.097
0.016	0.175	-0.198
-0.028	-0.016	0.148

$8 \times 3$



# 文本表示1：TF-IDF、SVD

- 通过一个小实例了解Skip Gram
  - 现有如下的句子
  - The man who passes the sentence should swing the sword
    - 我们准备利用Skip-Gram进行训练，把8维的one hot encoding通过训练，降维为3维

$$A^{[2]} = Z^{[2]} = W^{[2]} A^{[1]}$$

0.192	0.176	0.012
0.070	0.061	-0.046
-0.066	0.117	0.083
0.014	0.006	-0.044
-0.012	0.067	0.147
0.013	0.111	-0.097
0.016	0.175	-0.198
-0.028	-0.016	0.148

8\*3

0.068
0.170
-0.109

3\*1



0.042
0.020
0.006
0.007
-0.005
0.030
0.052
-0.021

$A^{[2]}$

8\*1



# 文本表示1：TF-IDF、SVD

- 通过一个小实例了解Skip Gram
  - 现有如下的句子
  - The man who passes the sentence should swing the sword
    - 我们准备利用Skip-Gram进行训练，把8维的one hot encoding通过训练，降维为3维

$$A^{[2]} = Z^{[2]} = W^{[2]} A^{[1]}$$

0.192	0.176	0.012
0.070	0.061	-0.046
-0.066	0.117	0.083
0.014	0.006	-0.044
-0.012	0.067	0.147
0.013	0.111	-0.097
0.016	0.175	-0.198
-0.028	-0.016	0.148

8\*3

0.068
0.170
-0.109

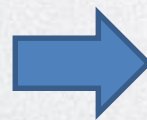
3\*1



0.042
0.020
0.006
0.007
-0.005
0.030
0.052
-0.021

$A^{[2]}$

8\*1



$$Z^{[3]} \hat{=} y$$

*Softmax*

0.128
0.125
0.124
0.124
0.122
0.127
0.130
0.120

# 文本表示1: TF-IDF、SVD

- 通过一个小实例了解Skip Gram
  - 现有如下的句子
  - The man who passes the sentence should swing the sword
    - 由于Skip-Gram针对passes的输出是the和who, 它们的one hot encoding 具体如下

Man	0
Passes	0
Sentence	0
Should	0
swing	0
sword	0
The	0
who	1

who

Man	0
Passes	0
Sentence	0
Should	0
swing	0
sword	0
The	1
who	0

the

实际y值

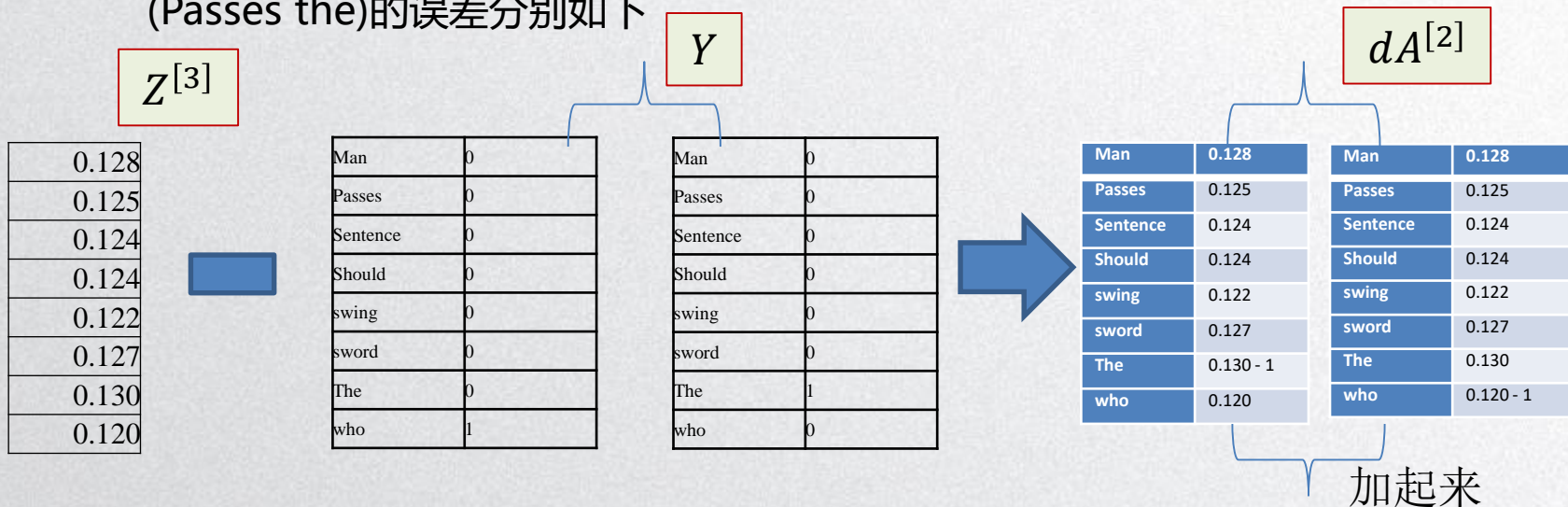
# 文本表示1：TF-IDF、SVD

- 通过一个小实例了解Skip Gram

- 现有如下的句子

- The man who passes the sentence should swing the sword

- 现在开始进行反向传播。根据  $dA^{[2]} = Z^{[3]} - Y$ ，有样本(Passes who)和样本(Passes the)的误差分别如下





# 文本表示1: TF-IDF、SVD

- 通过一个小实例了解Skip Gram
  - 现有如下的句子
  - The man who passes the sentence should swing the sword
    - 两个样本的总的误差如下, 根据这个误差进行反向传播
    - $dZ^{[2]} = dA^{[2]} = dA^{[2]}g'(Z^{[2]})$ , 因为g为直接传导,g' 为全是1的矩阵

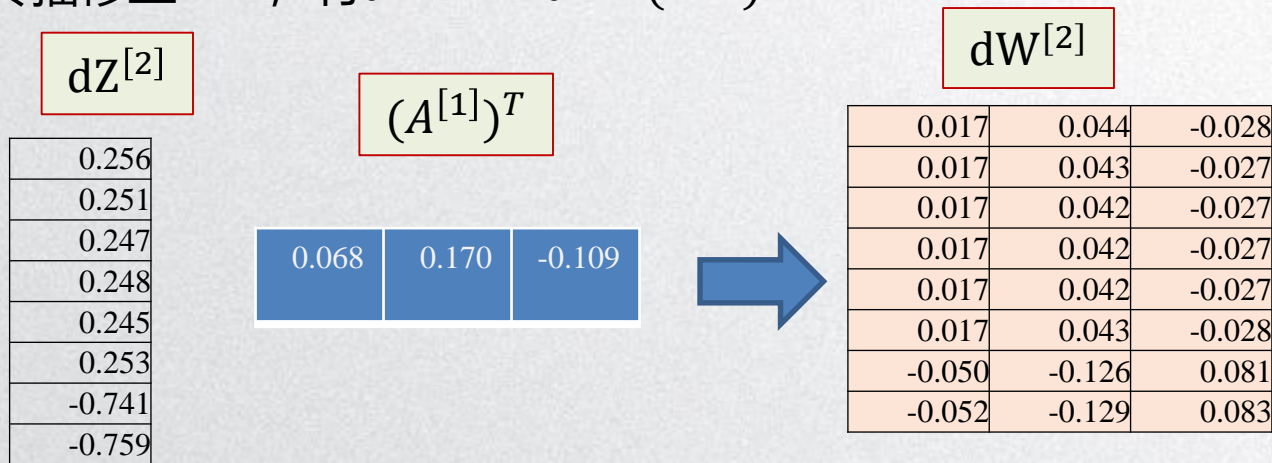
$$dZ^{[2]} = dA^{[2]} = dA^{[2]}g'(Z^{[2]})$$

0.256
0.251
0.247
0.248
0.245
0.253
-0.741
-0.759



# 文本表示1: TF-IDF、SVD

- 通过一个小实例了解Skip Gram
  - 现有如下的句子
  - The man who passes the sentence should swing the sword
  - 反向传播修正 $W^{[2]}$ , 有 $dW^{[2]} = dZ^{[2]}(A^{[1]})^T$



# 文本表示1: TF-IDF、SVD

- 通过一个小实例了解Skip Gram
  - 现有如下的句子
  - The man who passes the sentence should swing the sword
  - 继续  $dA^{[1]} = (W^{[2]})^T dZ^{[2]}$
  - $dZ^{[1]} = dA^{[1]} = dA^{[1]}g'(Z^{[1]})$  , 因为  $g$  为直接传导,  $g'$  为全是1的矩阵

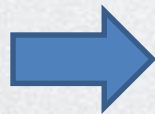
$$(W^{[2]})^T$$

0.192	0.070	-0.066	0.014	-0.012	0.013	0.016	-0.028
0.176	0.061	0.117	0.006	0.067	0.111	0.175	-0.016
0.012	-0.046	0.083	-0.044	0.147	-0.097	-0.198	0.148

$$dZ^{[2]}$$

0.256
0.251
0.247
0.248
0.245
0.253
-0.741
-0.759

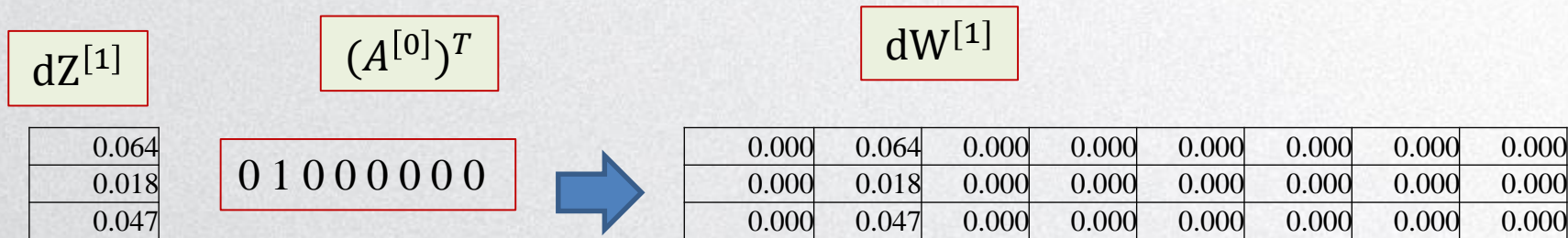
$$dZ^{[1]} = dA^{[1]} = dA^{[1]}g'(Z^{[1]})$$



0.064
0.018
0.047

# 文本表示1: TF-IDF、SVD

- 通过一个小实例了解Skip Gram
  - 现有如下的句子
  - The man who passes the sentence should swing the sword
  - $dW^{[1]} = dZ^{[1]}(A^{[0]})^T$





# 文本表示1: TF-IDF、SVD

- 通过一个小实例了解Skip Gram
  - 现有如下的句子
  - The man who passes the sentence should swing the sword
  - $W^{[1]} = W^{[1]} - \eta dW^{[1]}$

$$W^{[1]}$$

-0.078	0.068	-0.158	0.150	-0.097	0.036	0.168	0.098
0.018	0.170	-0.081	0.064	-0.055	0.071	-0.060	0.015
0.033	-0.109	-0.151	0.145	0.188	0.059	-0.058	0.096



0.2\*

$$dW^{[1]}$$

0.000	0.064	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.018	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.047	0.000	0.000	0.000	0.000	0.000	0.000

$$\text{new } W^{[1]}$$


-0.078	0.055	-0.158	0.150	-0.097	0.036	0.168	0.098
0.018	0.166	-0.081	0.064	-0.055	0.071	-0.060	0.015
0.033	-0.118	-0.151	0.145	0.188	0.059	-0.058	0.096



# 文本表示1: TF-IDF、SVD

- 通过一个小实例了解Skip Gram

- 现有如下的句子

- The man who passes the sentence should swing the sword

- $W^{[2]} = W^{[2]} - \eta dW^{[2]}$

0.192	0.176	0.012
0.070	0.061	-0.046
-0.066	0.117	0.083
0.014	0.006	-0.044
-0.012	0.067	0.147
0.013	0.111	-0.097
0.016	0.175	-0.198
-0.028	-0.016	0.148

$W^{[2]}$

$dW^{[2]}$

0.2\*

0.189	0.167	0.018
0.067	0.052	-0.041
-0.069	0.109	0.088
0.011	-0.002	-0.039
-0.015	0.059	0.152
0.010	0.102	-0.091
0.026	0.200	-0.214
-0.018	0.010	0.131


0.017	0.044	-0.028
0.017	0.043	-0.027
0.017	0.042	-0.027
0.017	0.042	-0.027
0.017	0.042	-0.027
0.017	0.043	-0.028
-0.050	-0.126	0.081
-0.052	-0.129	0.083

New  $W^{[2]}$



# 文本表示1：TF-IDF、SVD

- 通过一个小实例了解Skip Gram
  - 现有如下的句子
  - The man who passes the sentence should swing the sword
  - 利用新的权重进行前向传导，误差就会减小一点点
    - 不断迭代，误差继续降低
    - 请打开Excel文件进行实验
    - 验证这个结果

我的电脑 > Application (D:) > 2021-07-18 《数据科学概论》 new plan > 2022newPPT > 0404-文本表示2: Word2Vector					搜索"0404-文
名称	类型	大小	修改日期		
 2021-11-20SkipGram-正向传播、反向传导、在正向传播.xlsm	Microsoft Excel 启用...	31 KB	2021/11/27 13:15		

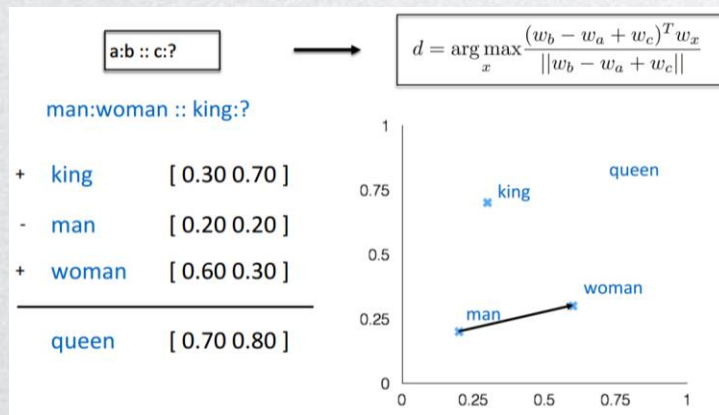
# 文本表示1: TF-IDF、SVD





# 文本表示2: Word2Vector

- Word2Vec: Word2Vec在单词类比上的结果
  - 单词类比任务:
    - 语义关系建模: "Beijing is to China as Paris is to \_\_\_\_"
    - 语法关系建模: "big is to bigger as deep is to \_\_\_\_"



$$\mathbf{v}(\text{king}) - \mathbf{v}(\text{man}) + \mathbf{v}(\text{woman}) = \mathbf{v}(\text{queen})$$



# 文本表示2: Word2Vector

- Word2Vec: 更多单词类比的结果
  - 降维和可视化

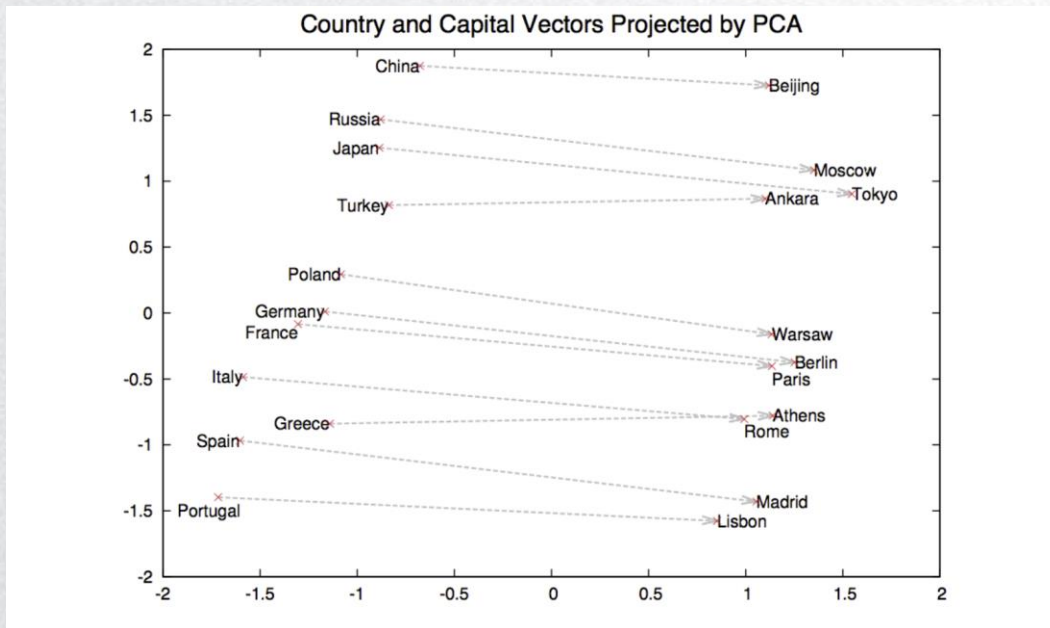
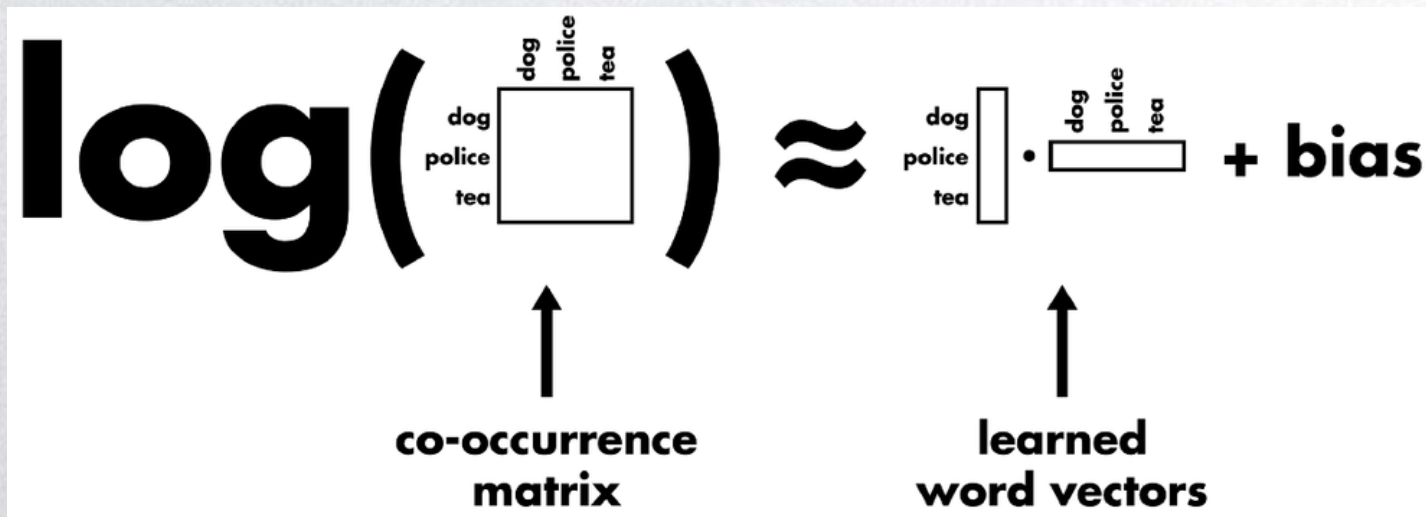


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

## 文本表示2: Word2Vector

- Word2Vec: 从矩阵分解的角度理解词嵌入
  - 词嵌入近似于分解单词-单词共现矩阵


$$\log\left(\begin{array}{c} \text{dog} \\ \text{police} \\ \text{tea} \end{array} \begin{array}{cc} \text{dog} & \text{police} \\ \text{police} & \text{tea} \\ \text{tea} & \end{array}\right) \approx \begin{array}{c} \text{dog} \\ \text{police} \\ \text{tea} \end{array} \cdot \begin{array}{ccc} & \text{dog} & \text{police} \\ & \text{police} & \text{tea} \\ & \text{tea} & \end{array} + \text{bias}$$

↑  
co-occurrence matrix

↑  
learned word vectors

# 文本表示2: Word2Vector

- Word2Vec

- 词嵌入已经成为深度学习时代文本处理的基础, 广泛应用于
  - 文本分类、聚类
  - 搜索、推荐
  - 自然语言处理, 如: 问答、机器翻译、NLI(自然语言推理inference)等

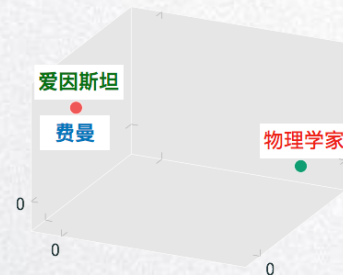
词嵌入建模分布式假设中的聚合关系

d1 爱因斯坦 是一个 物理学家  
 d2 费曼 是一个 物理学家

聚合



	爱因斯坦	费曼	物理学家
爱因斯坦	0	0	1
费曼	0	0	1
物理学家	1	1	0



# 文本表示1: TF-IDF、SVD





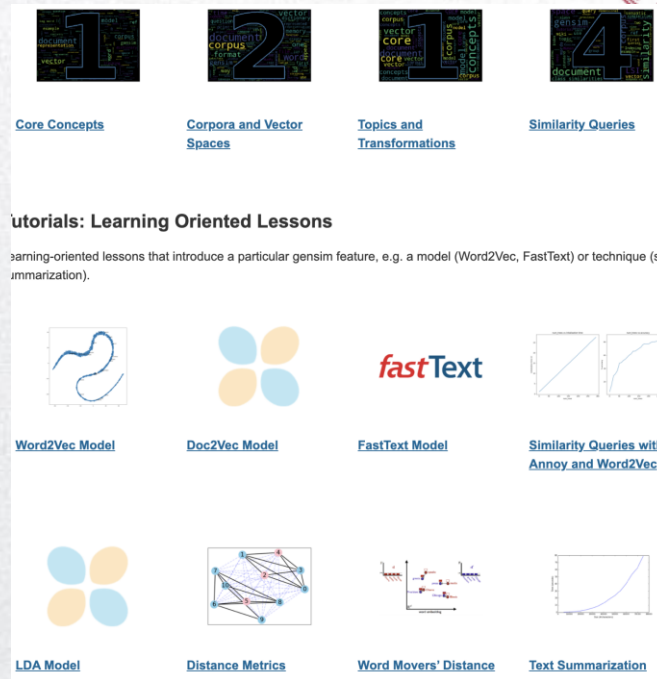
# 文本表示2: Word2Vector

- 文本表示实践: 基于gensim
- 安装:
  - `pip install gensim`



# 文本表示2: Word2Vector

- 文本表示实践: Gensim的功能
  - 基本文本处理
  - 文本的Vector Space表达 (如: TF-IDF)
  - 话题模型: LSI、PLSI、LDA等
  - 词嵌入模型:
    - Word2Vec、Doc2Vec、fastText等
  - .....



[https://radimrehurek.com/gensim/auto\\_examples/index.html](https://radimrehurek.com/gensim/auto_examples/index.html)



# 文本表示2: Word2Vector

- 文本表示实践
  - 案例: 构建TF-IDF、LSI、Word2vec的表达

- Step1: 定义文档集合
  - 以LSI论文中的数据集合为例
  - 去除停用词和出现次数太少的词
  - 处理成Bag-of-words表达

```
from collections import defaultdict
from gensim import corpora

documents = [
    "Human machine interface for lab abc computer applications",
    "A survey of user opinion of computer system response time",
    "The EPS user interface management system",
    "System and human system engineering testing of EPS",
    "Relation of user perceived response time to error measurement",
    "The generation of random binary unordered trees",
    "The intersection graph of paths in trees",
    "Graph minors IV Widths of trees and well quasi ordering",
    "Graph minors A survey",
]

# 去除停用词
stoplist = set('for a of the and to in'.split())
texts = [
    [word for word in document.lower().split() if word not in stoplist]
    for document in documents
]

# 去除只出现一次的词
frequency = defaultdict(int)
for text in texts:
    for token in text:
        frequency[token] += 1

#处理后的Bag-of-words文本
texts = [
    [token for token in text if frequency[token] > 1]
    for text in texts
]
```

# 文本表示2: Word2Vector

- 文本表示实践
- Step 2: 建立字典
  - 字典是从单词到ID的映射
  - 文档处理成ID后, 方便后续处理

```
#建立字典
dictionary = corpora.Dictionary(texts)
print(dictionary.token2id)
print()
#基于上述字典建立corpus
corpus = [dictionary.doc2bow(text) for text in texts]
print(corpus)
```

```
{'computer': 0, 'human': 1, 'interface': 2, 'response': 3, 'survey': 4, 'system': 5, 'time': 6, 'user': 7, 'eps': 8, 'trees': 9, 'graph': 10, 'minors': 11}
```

```
[[ (0, 1), (1, 1), (2, 1) ], [ (0, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1) ], [ (2, 1), (5, 1), (7, 1), (8, 1) ], [ (1, 1), (5, 2), (8, 1) ], [ (3, 1), (6, 1), (7, 1) ], [ (9, 1) ], [ (9, 1), (10, 1) ], [ (9, 1), (10, 1), (11, 1) ], [ (4, 1), (10, 1), (11, 1) ]]
```





# 文本表示2: Word2Vector

- 文本表示实践
- Step 3: 构建文档的TF-IDF表达

```
from gensim import models  
#tf-idf表达
```

```
#初始化tf-idf模型, 主要是计算IDF  
tfidf = models.TfidfModel(corpus)  
print(tfidf)
```

```
TfidfModel(num_docs=9, num_nnz=28)
```

```
#得到每一个文档的TF-IDF表达, 以稀疏矩阵的形式存储  
corpus_tfidf = tfidf[corpus]  
for doc in corpus_tfidf:  
    print(doc)
```

```
[(0, 0.5773502691896257), (1, 0.5773502691896257), (2, 0.5773502691896257)]  
[(0, 0.44424552527467476), (3, 0.44424552527467476), (4, 0.44424552527467476), (5, 0.3244870206138555), (6, 0.44424552527467476), (7, 0.3244870206138555)]  
[(2, 0.5710059809418182), (5, 0.4170757362022777), (7, 0.4170757362022777), (8, 0.5710059809418182)]  
[(1, 0.49182558987264147), (5, 0.7184811607083769), (8, 0.49182558987264147)]  
[(3, 0.6282580468670046), (6, 0.6282580468670046), (7, 0.45889394536615247)]  
[(9, 1.0)]  
[(9, 0.7071067811865475), (10, 0.7071067811865475)]  
[(9, 0.5080429008916749), (10, 0.5080429008916749), (11, 0.695546419520037)]  
[(4, 0.6282580468670046), (10, 0.45889394536615247), (11, 0.6282580468670046)]
```

# 文本表示2: Word2Vector

- 文本表示实践
- Step 4: 基于TF-IDF表达训练LSI
  - 可以得到: 1. topic的定义; 2. 基于topic的文档表达

```
#运行LSI, 以TFIDF表达为输入, 2个话题
lsi_model = models.LsiModel(corpus_tfidf, id2word=dictionary, num_topics=2) # 初始化LSI模型参数, K=2
corpus_lsi = lsi_model[corpus_tfidf] #基于corpus_tfidf训练LSI模型

#打印出学习到的latent topic, 一共两个
lsi_model.print_topics(2)

[(0,
  '0.703*trees" + 0.538*"graph" + 0.402*"minors" + 0.187*"survey" + 0.061*"system" + 0.060*"time" + 0.060*"respons
e" + 0.058*"user" + 0.049*"computer" + 0.035*"interface"',
  (1,
    '-0.460*"system" + -0.373*"user" + -0.332*"eps" + -0.328*"interface" + -0.320*"time" + -0.320*"response" + -0.293
*"computer" + -0.280*"human" + -0.171*"survey" + 0.161*"trees"')])

# 打印文档的topic表达, 每一个文档表示成2维的topic向量
for doc, as_text in zip(corpus_lsi, documents):
    print(doc, as_text)

[(0, 0.06600783396090383), (1, -0.5200703306361846)] Human machine interface for lab abc computer applications
[(0, 0.19667592859142569), (1, -0.7609563167700049)] A survey of user opinion of computer system response time
[(0, 0.08992639972446467), (1, -0.7241860626752505)] The EPS user interface management system
[(0, 0.07585847652178195), (1, -0.6320551586003422)] System and human system engineering testing of EPS
[(0, 0.10150299184980208), (1, -0.5737308483002957)] Relation of user perceived response time to error measurement
[(0, 0.7032108939378314), (1, 0.1611518021402594)] The generation of random binary unordered trees
[(0, 0.8774787673119835), (1, 0.16758906864659567)] The intersection graph of paths in trees
[(0, 0.3098624686818582), (1, 0.1408655362071016)] Graph minors IV Widths of trees and well quasi ordering
[(0, 0.6165825350569284), (1, -0.05392907566389303)] Graph minors A survey
```

# 文本表示2: Word2Vector

- 文本表示实践
- Step 5: 基于TF-IDF表达训练NMF
  - 1. 训练模型
  - 2. 得到特定词的表达 (输入为ID)
  - 3. 得到特定文档的表达 (输入为BOW: 即(单词ID, TF-IDF)列表)

```
In [107]: from gensim.models import nmf
          corpus_nmf = nmf.Nmf(corpus_tfidf, num_topics=2)
          print(corpus_nmf)
          corpus_nmf.print_topics(2)

<gensim.models.nmf.Nmf object at 0x1a236a73d0>
```

```
Out[107]: [(0,
            '0.183*"5" + 0.137*"7" + 0.134*"6" + 0.134*"3" + 0.127*"8" + 0.069*"4" + 0.059*"1" + 0.053*"11" + 0.052*"10" + 0.
            027*"2"'),
            (1,
            '0.405*"9" + 0.173*"10" + 0.115*"0" + 0.113*"2" + 0.089*"11" + 0.070*"1" + 0.036*"4" + 0.000*"3" + 0.000*"8" + 0.
            000*"7"')]
```

```
In [113]: print(corpus_nmf.get_term_topics(word_id=0))

          print(corpus_nmf.get_document_topics([(0, 1), (1, 1), (2, 1)]))

          [(0, 0.18426264063010644), (1, 0.8157373593698936)]
          [(0, 0.3392047279046352), (1, 0.6607952720953647)]
```

这里是单词编号，应  
该显示单词

Term → topic  
Document → topic

# 文本表示2: Word2Vector

- 文本表示实践
- Step 6: 基于上述文档集合训练Word2Vec模型
  - 得到每一个单词的嵌入向量
    - 注意: Word2Vec不能直接输出文档表达, Doc2Vec可以

```
# 基于texts数据集训练一个word2vec模型, 隐维度为2
import gensim
print(texts)
w2v = gensim.models.Word2Vec(texts, min_count=1, size=2)
print(w2v)
```

```
 [['human', 'interface', 'computer'], ['survey', 'user', 'computer', 'system', 'response', 'time'], ['eps', 'user',
 'interface', 'system'], ['system', 'human', 'system', 'eps'], ['user', 'response', 'time'], ['trees'], ['graph', 't
 rees'], ['graph', 'minors', 'trees'], ['graph', 'minors', 'survey']]
Word2Vec(vocab=12, size=2, alpha=0.025)
```

```
# 查询词向量
for i, word in enumerate(w2v.wv.vocab):
    if i == 20:
        break
    print(word, ":", w2v[word])
```

```
human : [-0.12686646  0.23246695]
interface : [-0.14137153 -0.14618981]
computer : [-0.04327307 -0.03931544]
survey : [ 0.09973007 -0.1505802 ]
user : [ 0.24788894 -0.2236827 ]
system : [-0.06096118  0.21712142]
response : [-0.18140024 -0.13812009]
time : [-0.17781538  0.14191036]
eps : [-0.18355471 -0.1860176 ]
trees : [ 0.24606217 -0.03211486]
graph : [-0.1818891  0.15232858]
minors : [0.08371314  0.06142422]
```



## 文本表示2: Word2Vector

- 文本表示实践
- Step 7: 基于词嵌入的单词相似度计算

```
# 计算两个词的相似度
pairs = [
    ('human', 'system'),
    ('system', 'eps'),
]
for w1, w2 in pairs:
    print('%r\t%r\t%.2f' % (w1, w2, w2v.wv.similarity(w1, w2)))
```

'human'	'system'	0.97
'system'	'eps'	-0.50