



# Word2vec



讲师：曾江峰



E-mail: [jfzeng@ccnu.edu.cn](mailto:jfzeng@ccnu.edu.cn)

# Background

Before ...

## One-hot Representation

- NLP 相关任务中最常见的第一步是创建一个词表库并把每个词顺序编号。这实际就是词表示方法中的 One-hot Representation，这种方法把每个词顺序编号，每个词就是一个很长的向量，向量的维度等于词表大小，只有对应位置上的数字为 1，其他都为 0。当然在实际应用中，一般采用稀疏编码存储，主要采用词的编号。
- 三大缺点：**无法捕捉词与词之间的相似度**，即**语义鸿沟**，就算是近义词也无法从词向量中看出任何关系；**维数灾难**；**强稀疏性**。

# 离散表示: One-hot表示

## 语料库

John likes to watch movies. Mary likes too.

John also likes to watch football games.

## 词典

```
{"John": 1, "likes": 2, "to": 3, "watch": 4, "movies": 5, "also":  
6, "football": 7, "games": 8, "Mary": 9, "too": 10}
```

## One-hot表示

John: [1, 0, 0, 0, 0, 0, 0, 0, 0]

likes: [0, 1, 0, 0, 0, 0, 0, 0, 0]

...

too : [0, 0, 0, 0, 0, 0, 0, 0, 1]

- ❖ 词典包含10个单词，每个单词有唯一索引
- ❖ 在词典中的顺序和在句子中的顺序没有关联

先贤的  
智慧

“You shall know a word by the company it keeps.”

——J. R. Firth 1957: 11

一种重要的语言学思想是 Firth 在 1957 年提出的，一个词语的意思应该由它的上下文来表示。

现代统计自然语言处理中最有创见的想法之一。

government debt problems turning into banking crises as has happened in  
saying that Europe needs unified banking regulation to replace the hodgepodge

banking附近的词将会代表banking的含义

Middle ...

## Frequency based vector

- Counter vector
- TF-IDF
- Co-Occurrence matrix

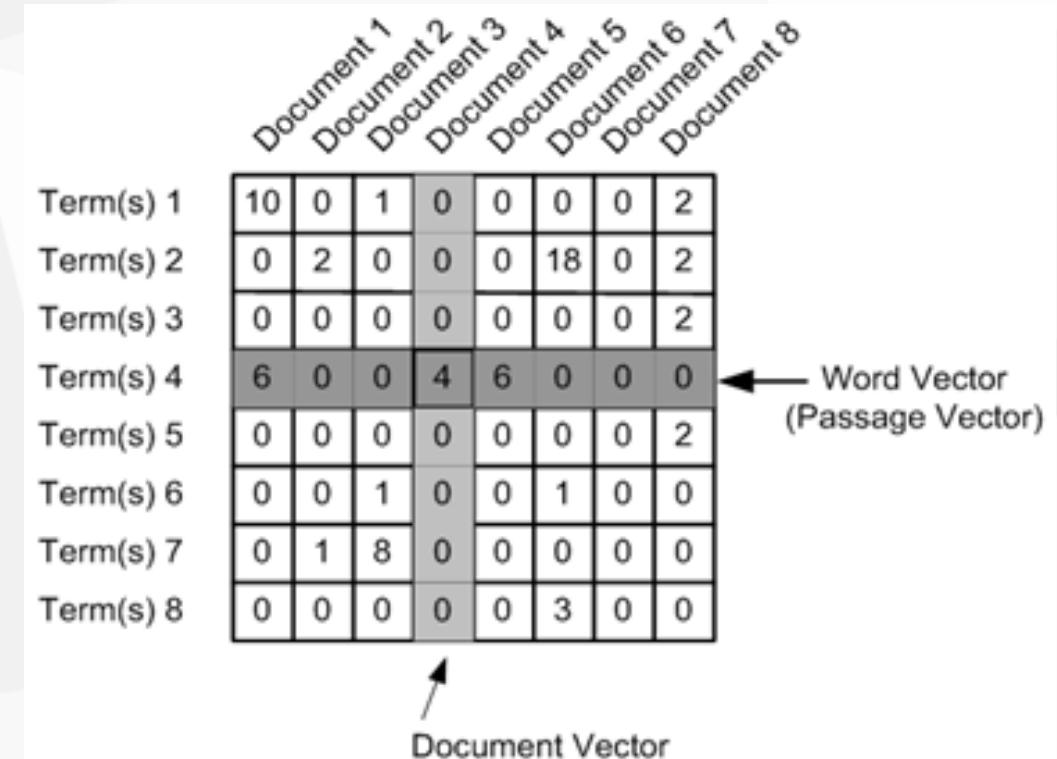
# Counter Vector

D1: He is a lazy boy. She is also lazy.

D2: Neeraj is a lazy person.

The count matrix M of size 2 X 6 will be represented as –

	He	She	lazy	boy	Neeraj	person
D1	1	1	2	1	0	0
D2	0	0	1	0	1	1



# TF-IDF

- ❖ 在信息检索中，tf-idf（词频-逆文档频率）是一种统计方法，用以评估一个单词在一个文档集合或语料库中的重要程度。
- ❖ 经常被用作信息检索、文本挖掘以及用户模型的权重因素。
- ❖ tf-idf 的值会随着单词在文档中出现的次数的增加而增大，也会随着单词在语料库中出现的次数的增多而减小。
- ❖ tf-idf 是如今最流行的词频加权方案之一。

# TF-IDF

- ❖ TF-IDF值的计算方式： $TF * IDF$
- ❖ **主要思想**是：如果某个词或短语在一篇文章中出现的频率高（即TF高），并且在其他文章中很少出现（即IDF高），则认为此词或者短语具有很好的类别区分能力，适合用来分类。
- ❖ 通俗理解TF-IDF就是：TF刻画了词语t对某篇文档的重要性，IDF刻画了词语t对整个文档集的重要性。

# TF-IDF

- ❖ TF (Term Frequency, 词频) 表示一个给定词语t在一篇给定文档d中出现的频率。TF越高，则词语t对文档d来说越重要，TF越低，则词语t对文档d来说越不重要。
- ❖ 那是否可以以TF作为文本相似度评价标准呢？----》 No
  - 常用词（“我”、“是”）的TF值在每篇文档中都具有非常高的词频
- ❖ 对于某一文档  $d_j$  里的词语  $t_i$  来说， $t_i$  的词频可表示为：

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

- ❖ 其中  $n_{i,j}$  是词语  $t_i$  在文档  $d_j$  中的出现次数，分母则是在文件  $d_j$  中所有词语的出现次数之和。

# TF-IDF

- ❖ IDF (Inverse Document Frequency, 逆向文档频率) 的主要思想是：如果包含词语t的文档越少，则IDF越大，说明词语t在整个文档集层面上具有很好的类别区分能力。
- ❖ 对于整个文档集而言，评价词语重要性的标准就是IDF。
- ❖ 某一特定词语的IDF，可以由总文件数除以包含该词语的文件数，再将得到的商取对数得到：

$$\text{idf}_i = \log \frac{|D|}{|\{j : t_i \in d_j\}|}$$

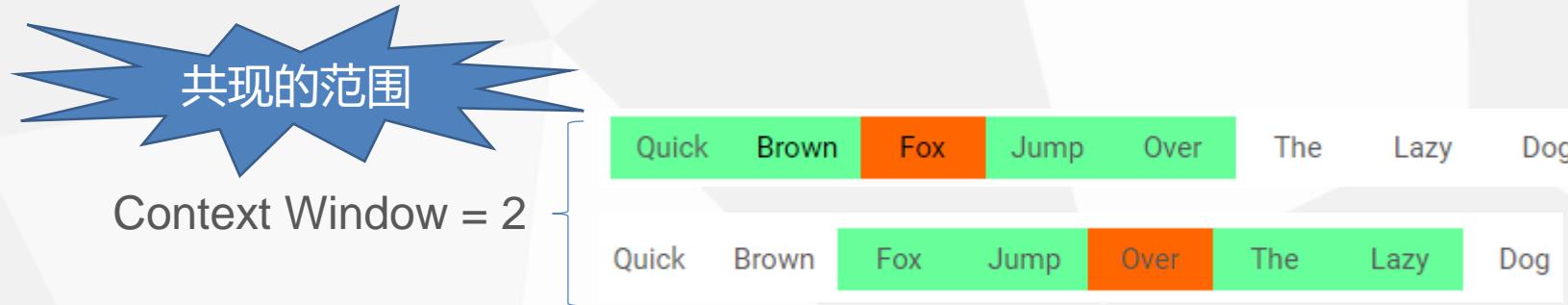
- ❖ 其中 |D| 是语料库中所有文档总数，分母是包含词语  $t_i$  的所有文档数。

# TF-IDF

- ❖ TFIDF算法是建立在这样一个假设之上的：对区别文档最有意义的词语应该是那些在文档中出现频率高，而在整个文档集合的其他文档中出现频率少的词语。显然这并不是完全正确的。
- ❖ IDF的简单结构并不能有效地反映单词的重要程度和特征词的分布情况，使其无法很好地完成对权值调整的功能，所以TF-IDF法的精度并不是很高。
- ❖ 在TF-IDF算法中并没有体现出单词的位置信息。

# Co-Occurrence Matrix

- ❖ The big idea – Similar words tend to occur together and will have similar context.
  - Apple is a fruit.
  - Mango is a fruit.



# Co-Occurrence Matrix

Corpus = He is not lazy. He is intelligent. He is smart.

	He	is	not	lazy	intelligent	smart
He	0	4	2	1	2	1
is	4	0	1	2	2	1
not	2	1	0	1	0	0
lazy	1	2	1	0	0	0
intelligent	2	2	0	0	0	0
smart	1	1	0	0	0	0

He	is	not	lazy	He	is	intelligent	He	is	smart
He	is	not	lazy	He	is	intelligent	He	is	smart
He	is	not	lazy	He	is	intelligent	He	is	smart
He	is	not	lazy	He	is	intelligent	He	is	smart

➤ 将共现矩阵行(列)作为词向量

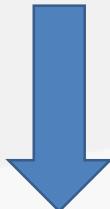
- 向量维数随着词典大小线性增长
- 存储整个词典的空间消耗非常大
- 一些模型如文本分类模型会面临稀疏性问题
- 模型会欠稳定

# Co-Occurrence Matrix

- ❖ But, remember this co-occurrence matrix is not the word vector representation that is generally used.
- ❖ Instead, this Co-occurrence matrix is decomposed using techniques like PCA, SVD etc. into factors and combination of these factors forms the word vector representation.

离散表示

Co-currence矩阵的行(列)向量作为词向量



分布式表示

Co-currence矩阵的SVD降维的低维词向量表示

# *Co-Occurrence Matrix*

## Advantages

- ❖ It preserves the semantic relationship between words.
  - man and woman tend to be closer than man and apple.

## Disadvantages

- ❖ It requires huge memory to store the co-occurrence matrix.



Now ...

## Distributed Representation

- 基本思想是通过训练将每个词映射成  $K$  维实数向量 ( $K$  一般为模型中的超参数)，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。
- Distributed representation 最大的贡献：具有相似语义的词语的词向量更接近，缩小语义鸿沟；避免维数灾难。
- Word2vec 使用的就是这种 Distributed representation 的词向量表示方式。



# What is Word Embedding?

The process of transforming words into vectors (numbers) is called Word Embedding.

## 理解：

- In language processing, the vectors are derived from textual data, in order to reflect various linguistic properties of the text.
- Word2vec is the most popular WE model.
- 将词用具有语义相关性的“词向量”的方式表示，可谓是将DL算法引入NLP领域的核心技术。

# 词编码需要保证词的相似性

Nearest words to  
**frog**:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus



**litoria**



**rana**



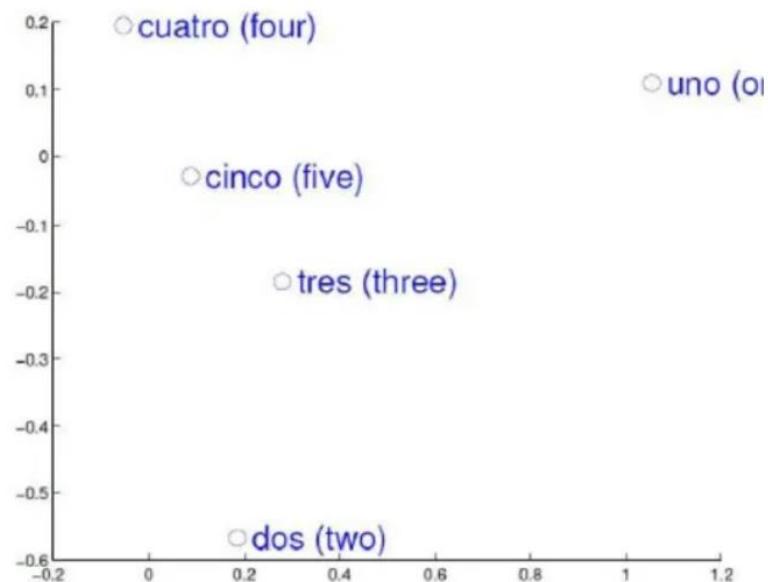
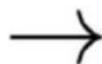
**leptodactylidae**



**eleutherodactylus**

# 简单词/短语翻译

## □ 向量空间分布的相似性



左： 英语

右： 西班牙语

## 向量空间子结构

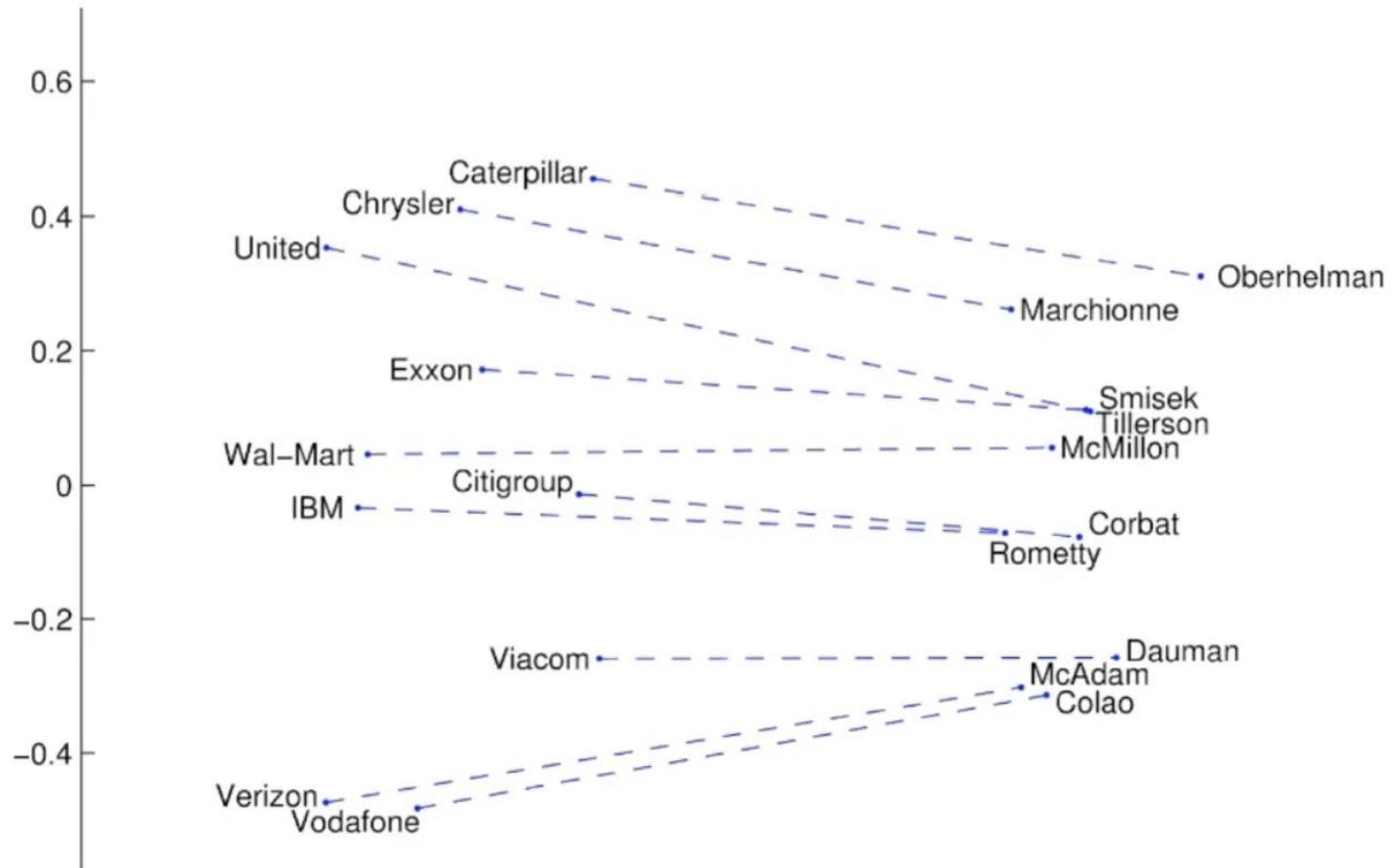
---

$$V_{King} - V_{Queen} + V_{Women} = V_{Man}$$

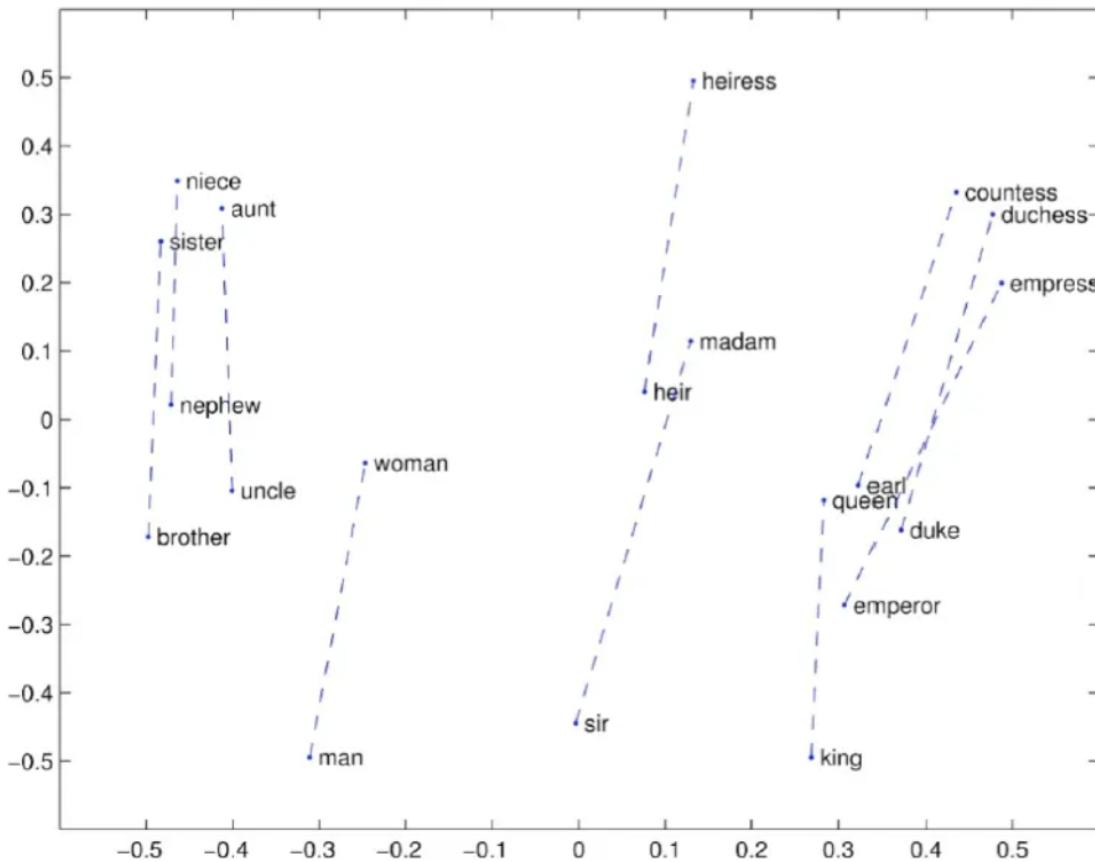
$$V_{Paris} - V_{France} + V_{German} = V_{Berlin}$$

最终目标：词向量表示作为机器学习、特别是深度学习的输入和表示空间

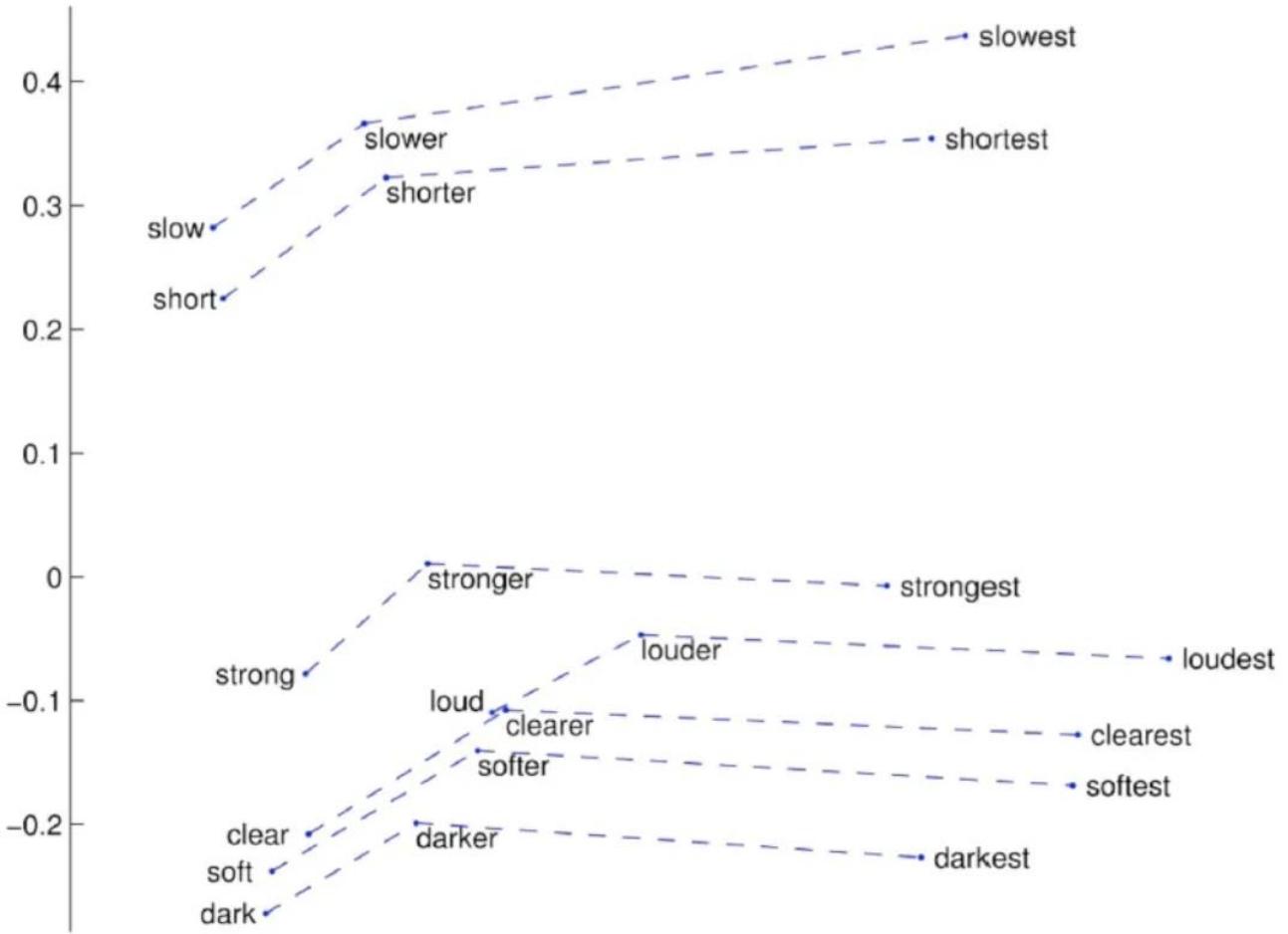
# 词嵌入可视化: 公司 — CEO



# 词嵌入可视化：词向



# 词嵌入可视化：比较级和最高级





Language Model



**语言模型**目标是建模自然语言的概率分布，在自然语言处理研究中具有重要的作用，是机器翻译、语音识别、输入法、句法分析等任务的支撑。语言模型是自然语言处理基础任务之一，大量的研究从n元语言模型和神经语言模型等不同角度开展了系列工作。

由于语言模型可以为自然语言的表示学习提供天然的自监督训练目标，近年来，**预训练语言模型**做为通用的基于深度神经网络的自然语言处理算法的基础工具，受到越来越多的重视。大规模的预训练语言模型对于提升各类自然语言处理任务的效果起到了重要作用。

# *Language Model* 的副产物——Word Embedding

- ❖ 大胆想法：如何从大量未标注的普通文本数据中无监督地学习出词向量
- ❖ 训练好语言模型的同时，顺便得到词向量
- ❖ 语言模型其实就是在一句话是不是正常人说出来的。形式化的描述就是给定一个字符串，计算其对应的自然语言的概率 $P(w_1, w_2, \dots, w_t)$ .

$$P(w_1, w_2, \dots, w_t) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_1, w_2) \times \dots \times P(w_t|w_1, w_2, \dots, w_{t-1})$$

要从一段无标注的自然文本中学习出一些东西，无非就是统计出词频、词的共现、词的搭配之类的信息。而要从自然文本中统计并建立一个语言模型，无疑是要求最为精确的一个任务。既然构建语言模型这一任务要求这么高，其中必然也需要对语言进行更精细的统计和分析，同时也会需要更好的模型，更大的数据来支撑。目前最好的词向量都来自于此，也就不难理解了。

# 传统的统计语言模型

- 传统的统计语言模型是表示语言基本单位（一般为句子）的概率分布函数，这个概率分布也就是该语言的生成模型。一般语言模型可以使用各个词语条件概率的形式表示：

$$p(s) = p(w_1^T) = p(w_1, w_2, \dots, w_T) = \prod_{t=1}^T p(w_t | \text{Context})$$

# 上下文无关模型

- 上下文无关模型（Context=NULL）
  - 该模型仅仅考虑当前词本身的概率，不考虑该词所对应的上下文环境。这是一种最简单，易于实现，但没有多大实际应用价值的统计语言模型。

$$p(w_t | \text{Context}) = p(w_t) = \frac{N_{w_t}}{N}$$

- 这个模型不考虑任何上下文信息，仅仅依赖于训练文本中的词频统计。它是n-gram模型中当 n=1 的特殊情形，所以有时也称作 Unigram Model（一元文法统计模型）。实际应用中，常被应用到一些商用语音识别系统中。

# n-gram模型

n-1阶的Markov假设

- n-gram模型利用前n-1个词的信息来预测当前词出现的概率，其Context是前n-1个词，对当前词的预测具有很强的约束力。
- 因为只看前n-1个词而不是所有词使得模型的效率较高。
- 缺点：
  - 无法建模更远的关系，语料的不足使得无法训练更高阶的语言模型。大部分研究或工作都是使用Trigram，若使用高阶的模型，其统计到的概率可信度就大打折扣。
  - 无法建模出词之间的相似度。理论上，两个具有某种相似性的词，如果一个词经常出现在某段词之后，那么也许另一个词出现在这段词后面的概率也比较大。比如“白色的汽车”经常出现，那完全可以认为“白色的轿车”也可能经常出现。但n-gram无法得出做出这样的判断，因为对它而言，“汽车”和“轿车”是两个完全不同概念。
  - 缺乏平滑性。训练语料里面有些n元组没有出现过，其对应的条件概率就是0，导致计算一整句话的概率为0。

$$\hat{P}(w_t | w_1^{t-1}) \approx \hat{P}(w_t | w_{t-n+1}^{t-1}).$$

$$p(w_k|w_1, \dots, w_{k-1}) \approx p(w_k|w_{k-n+1}, \dots, w_{k-1})$$

$$p(w_k|w_1, \dots, w_{k-1}) \approx \frac{\text{count}(w_{k-n+1}, \dots, w_k)}{\text{count}(w_{k-n+1}, \dots, w_{k-1})}$$

$$p(w_k|w_1, \dots, w_{k-1}) \approx \frac{\text{count}(w_{k-1}, \dots, w_k)}{\text{count}(w_{k-1})}$$

那么，n-gram中的参数n取多大比较合适呢？一般来说，n的选取需要同时考虑计算复杂度和模型效果两个因素。

n	模型参数的数量
1(unigram)	$2 \times 10^5$
2(bigram)	$4 \times 10^{10}$
3(trigram)	$8 \times 10^{15}$
4(4-gram)	$16 \times 10^{20}$

表1：模型参数数量与n的关系

# bigram case

- 语料库总词数为13,748
- 计算结果： $P(I \text{ want to eat Chinese food})$   
 $=P(I)P(want|I)P(to|want)P(eat|to)P(Chinese|eat)P(food|Chinese)$   
 $=0.25*1087/3437*786/1215*860/3256*19/938*120/213 =0.000154171$

I ↗	3437 ↗
want ↗	1215 ↗
to ↗	3256 ↗
eat ↗	938 ↗
Chinese ↗	213 ↗
food ↗	1506 ↗
lunch ↗	459 ↗

表1 词和词频 ↗

↗	I ↗	want ↗	to ↗	eat ↗	Chinese ↗	food ↗	lunch ↗
I ↗	8 ↗	1087 ↗	0 ↗	13 ↗	0 ↗	0 ↗	0 ↗
want ↗	3 ↗	0 ↗	786 ↗	0 ↗	6 ↗	8 ↗	6 ↗
to ↗	3 ↗	0 ↗	10 ↗	860 ↗	3 ↗	0 ↗	12 ↗
eat ↗	0 ↗	0 ↗	2 ↗	0 ↗	19 ↗	2 ↗	52 ↗
Chinese ↗	2 ↗	0 ↗	0 ↗	0 ↗	0 ↗	120 ↗	1 ↗
food ↗	19 ↗	0 ↗	17 ↗	0 ↗	0 ↗	0 ↗	0 ↗
lunch ↗	4 ↗	0 ↗	0 ↗	0 ↗	0 ↗	1 ↗	0 ↗

表2 词序列频度 ↗

# 神经网络语言模型NNLM

- NNLM 是 Neural Network Language Model 的缩写，即神经网络语言模型。NNLM 采用的是 Distributed Representation，即每个词被表示为一个浮点向量。

➤ 直接从语言模型出发，将模型最优化过程转化为求词向量表示的过程

目标函数  $L(\theta) = \sum_t \log P(w_t | w_{t-n+1}, \dots, w_{t-1}).$

# NNLM: 结构



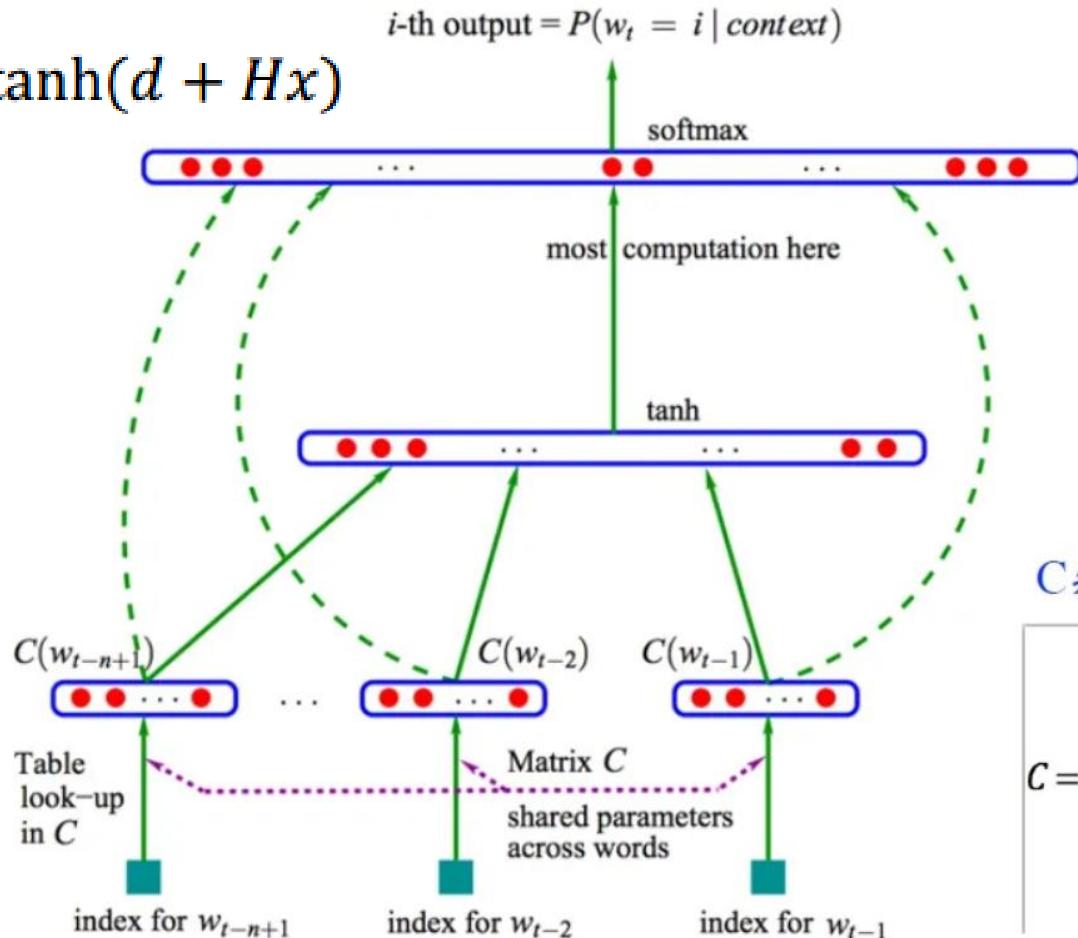
$$\hat{P}(w_t | w_1^{t-1}) \approx \hat{P}(w_t | w_{t-n+1}^{t-1}).$$

$$y = b + Wx + Utanh(d + Hx)$$

$U: V * h$   
 $W: V * (n-1)m$   
 $b: V$

$H: (n-1)m * h$   
 $d: h$

$X: (n-1)m$



- (N-1)个前向词: one-hot 表示
- 采用线性映射将 one-hot 表示投影到稠密 D 维表示
- 输出层: Softmax
- 各层权重最优化: BP + SGD

C矩阵是投影矩阵，也是稠密词向量表：

$$C = (w_1, w_2, \dots, w_v) = \begin{pmatrix} (w_1)_1 & (w_2)_1 & \dots & (w_v)_1 \\ (w_1)_2 & (w_2)_2 & \dots & (w_v)_2 \\ \vdots & \vdots & \ddots & \vdots \\ (w_1)_D & (w_2)_D & \dots & (w_v)_D \end{pmatrix}$$

# 神经网络语言模型NNLM

- 目标是要学到一个好的模型：

$$f(w_t, w_{t-1}, \dots, w_{t-n+2}, w_{t-n+1}) = p(w_t | w_1^{t-1})$$

- 需要满足的约束为：

$$f(w_t, w_{t-1}, \dots, w_{t-n+2}, w_{t-n+1}) > 0$$

$$\sum_{i=1}^{|V|} f(i, w_{t-1}, \dots, w_{t-n+2}, w_{t-n+1}) = 1$$

- 每个输入词都被映射为一个向量，该映射用  $C$  表示。  
 $f$  为一个前馈或递归神经网络，其输出是一个含有  $|V|$  个元素的向量，向量中的第  $i$  个元素表示当前词被预测为词汇中的第  $i$  个单词的概率

$$p(w_t = i | w_1^{t-1})$$

# 神经网络语言模型NNLM

- 训练的目标依然是最大似然加正则项，即：

$$\text{Max Likelihood} = \max \frac{1}{T} \sum_t \log f(w_t, w_{t-1}, \dots, w_{t-n+2}, w_{t-n+1}; \theta) + R(\theta)$$

- 输出层采用 softmax 函数：

$$p(w_t | w_{t-1}, \dots, w_{t-n+2}, w_{t-n+1}) = \frac{e^{y_{wt}}}{\sum_i e^{y_i}}$$

$$y = b + Wx + Utanh(d + Hx)$$

- Softmax 模型使得概率取值为(0,1)，因此不会出现概率为 0 的情况，即自带平滑，无需传统 n-gram 模型中那些复杂的平滑算法。
- 需要注意的是，一般神经网络的输入层只是一个输入值，而在这里，输入层  $x$  也是参数（存在  $C$  中），也是需要优化的。优化结束之后，词向量有了，语言模型也有了。

与n-gram模型相比，神经概率语言模型有什么优势呢？  
主要有以下两点：

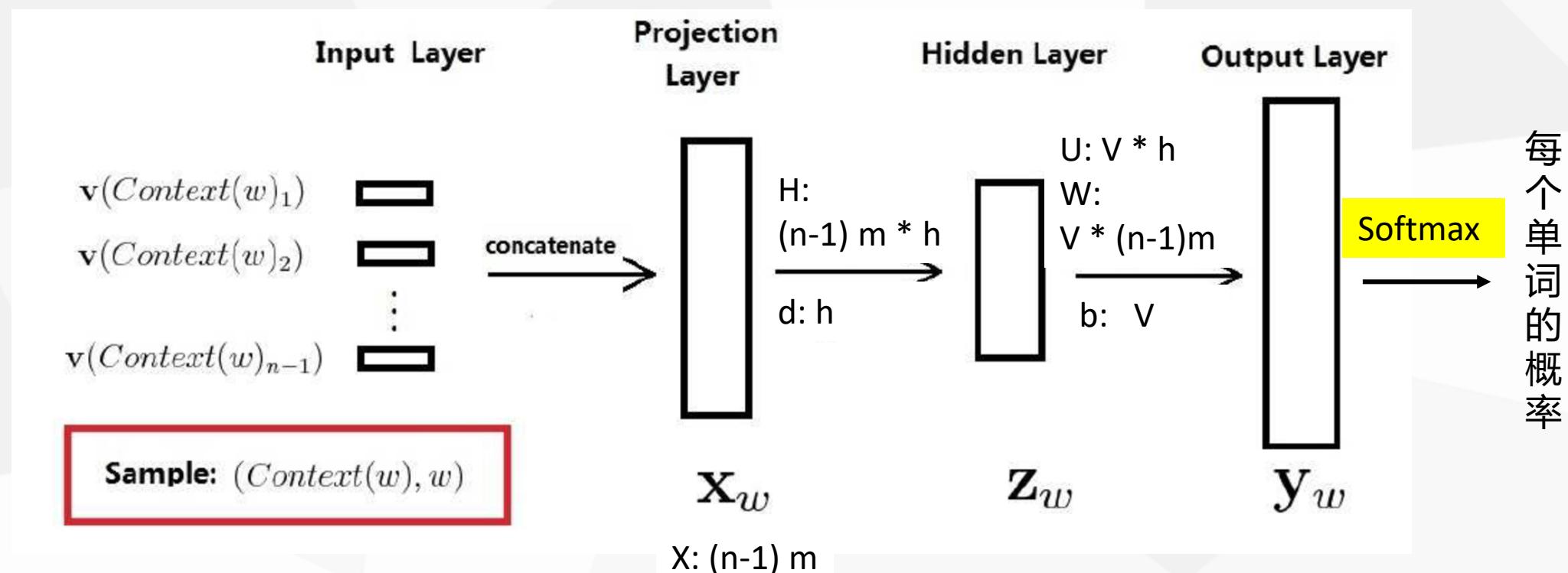
- 词语之间的相似性可以通过词向量来体现。
- 基于词向量的模型自带平滑化功能，不再需要像n-gram那样进行额外处理了。

# N-gram NNLM的缺陷



Word2vec的产生

- 计算量太大。主要的计算集中在『**隐层和输出层之间的矩阵运算**』和『**输出层上的Softmax**』归一化运算。



# Word2vec

# 主要参考文献

- ❖ Efficient Estimation of Word Representations in Vector Space (ICLR 2013 workshop)
- ❖ Distributed Representations of Words and Phrases and their Compositionality (NIPS 2013)
- ❖ A Neural Probabilistic Language Model (JMLR 2003)
- ❖ <http://migsena.com/build-and-visualize-word2vec-model-on-amazon-reviews/> (code: <https://github.com/MiguelSteph/word2vec-with-gensim>)

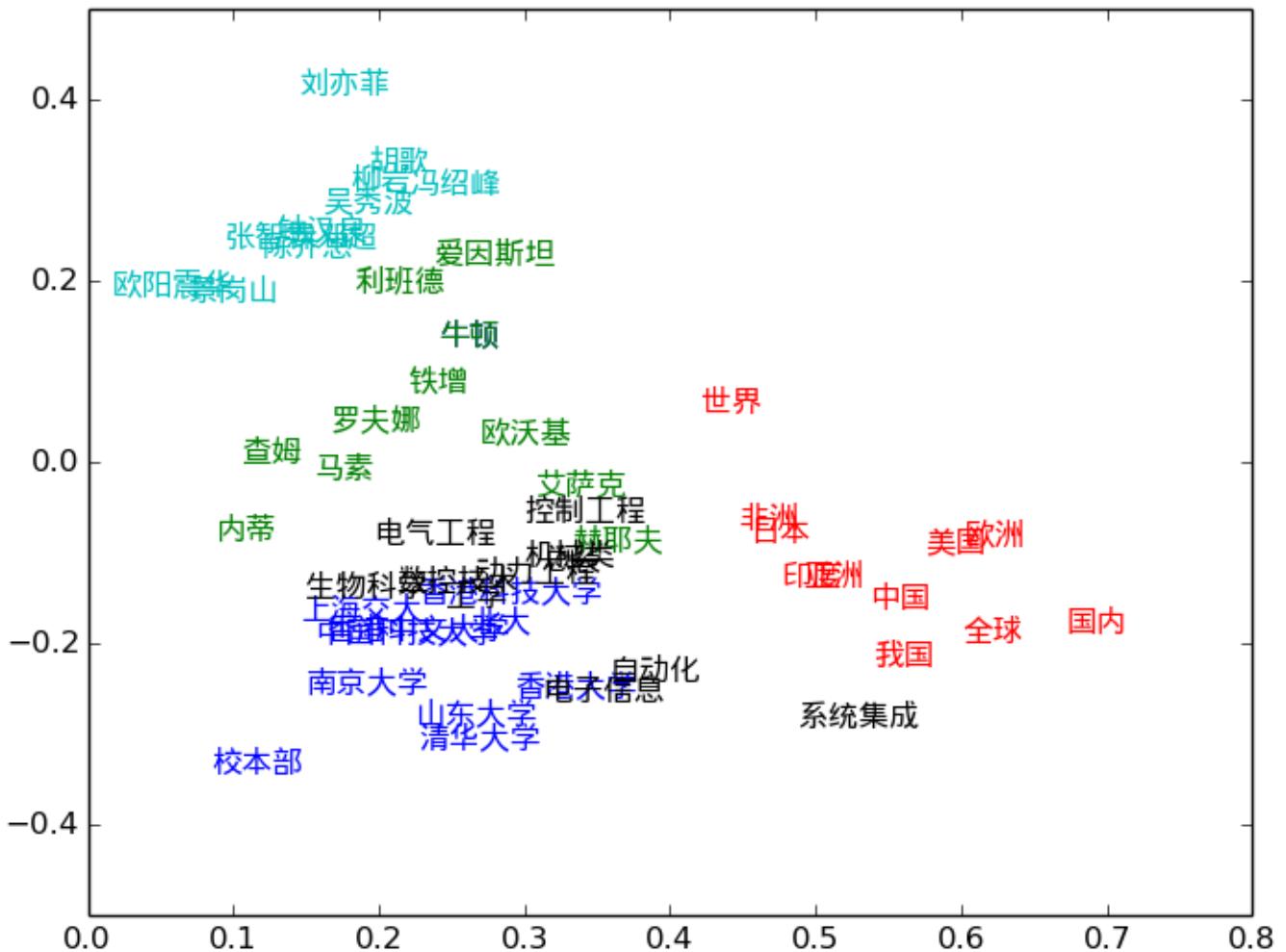


# Word2vec

- Word2vec is a **two-layer neural network** that processes text.
  - Input: a text corpus
  - Output: a set of feature vectors for words in that corpus
- Word2vec is a prediction based model rather than frequency. It uses predictive analysis to make a weighted guess of a word co-occurring with respect to its neighboring words.



语料库中拥有相同上下文的词映射到向量空间中的距离会更近。



# Word2vec

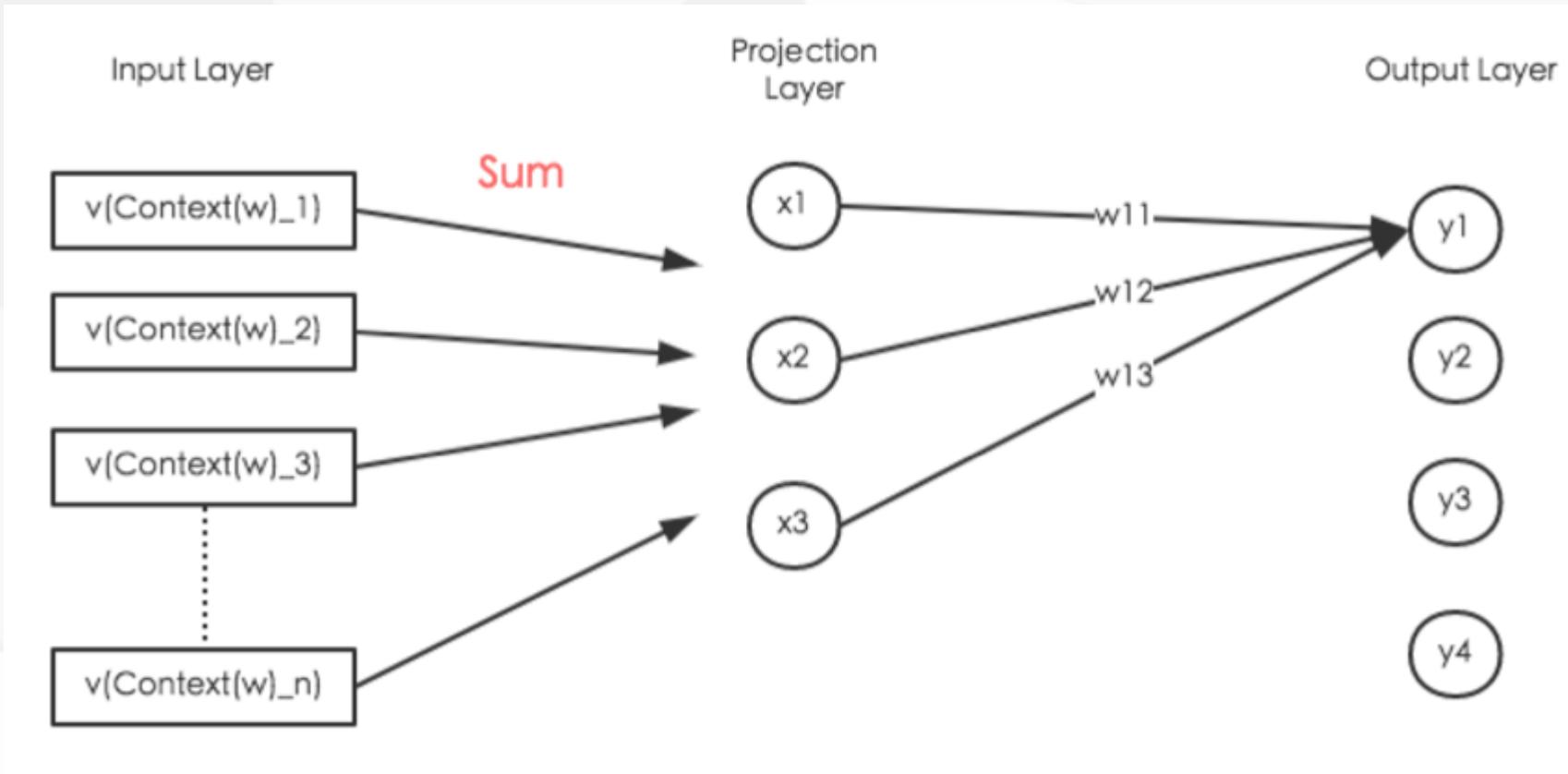
- 在神经网络语言模型中, word embeddings 只是副产品; word2vec 等模型则以生成 word embeddings 为直接目标。
- 两者的主要区别在于:
  - 计算复杂度 computational complexity: 用深度神经网络来生成 word embeddings 开销太大; word2vec 提出了训练 word embeddings 的简单模型, 计算开销大大减小。  
(计算复杂度是 word embeddings 模型的关键指标之一)
  - word2vec 和 GloVe 能将语义关系编码进最终的 word embeddings, 这对于需要这一层关系的后续任务是很帮助的; 常规的神经网络生成 task-specific embeddings, 不适用于其他任务。

# Word2vec

- Motivations:
  - 加快模型训练速度
    - 简化模型结构
    - 优化Softmax归一化
  - 应用于更大规模语料集，提升word vector的质量
    - 得益于模型结构简单，训练速度非常快

# 简化模型结构

- 针对『隐层和输出层之间的矩阵运算』，  
word2vec选择删去隐藏层。



# 优化Softmax归一化

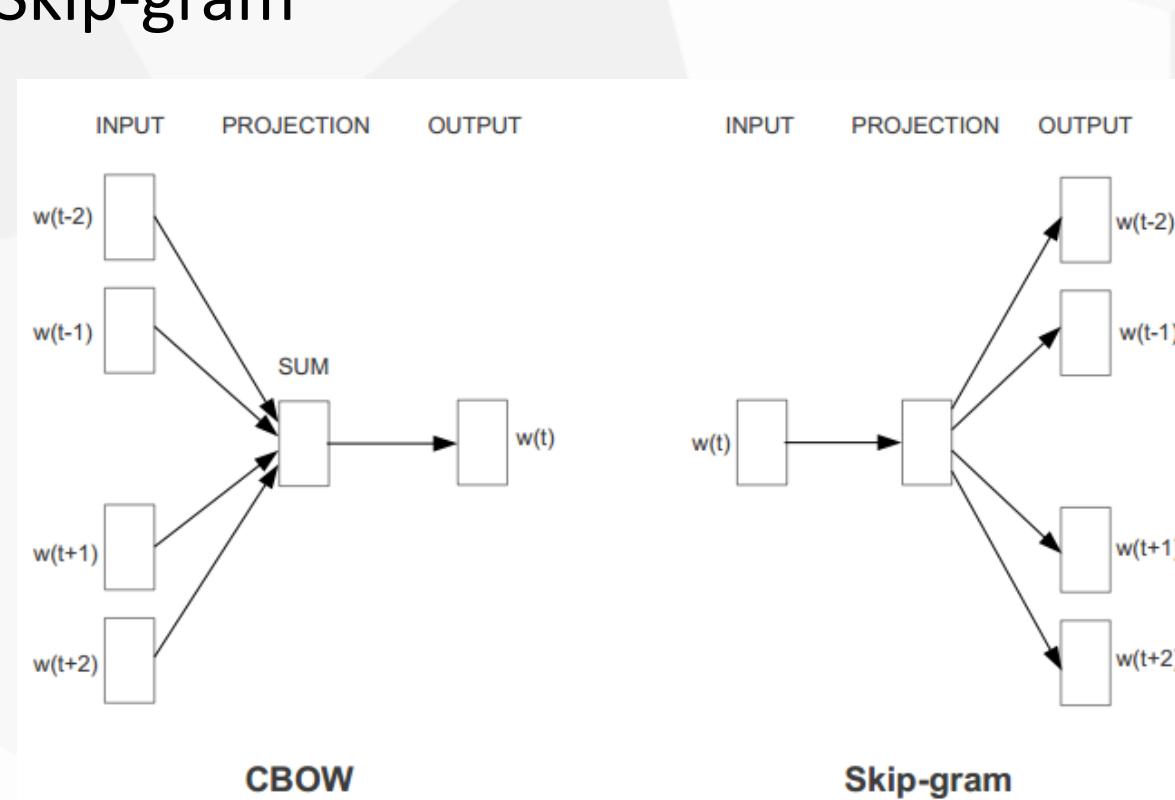
- 下面式子的计算瓶颈在于分母。

$$\begin{aligned} p(y_i | \text{Context}(w)) &= \frac{\exp(y_i)}{\sum_{k=1}^4 \exp(y_k)} \\ &= \frac{\exp(\vec{w}_i^T \vec{x})}{\sum_{k=1}^4 \exp(\vec{w}_k^T \vec{x})} \end{aligned}$$

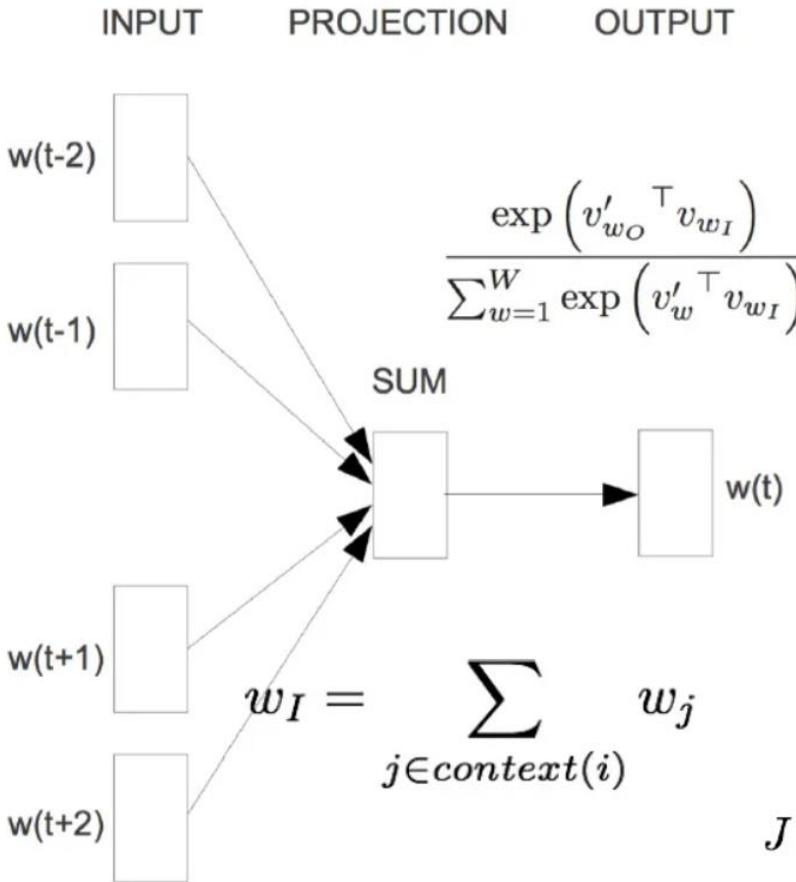
- Word2vec提出了两种优化Softmax计算过程的方法，同样也对应着Word2vec的两种框架，即：Hierarchical Softmax和Negative Sampling。

# Word2vec

- Two variants:
  - CBOW (Continuous Bag of Words)
  - Skip-gram



# word2vec: CBOW(连续词袋)



- ✓ 无隐层
- ✓ 使用双向上下文窗口
- ✓ 上下文词序无关 (BoW)
- ✓ 输入层直接使用低维稠密表示
- ✓ 投影层简化为求和(平均)

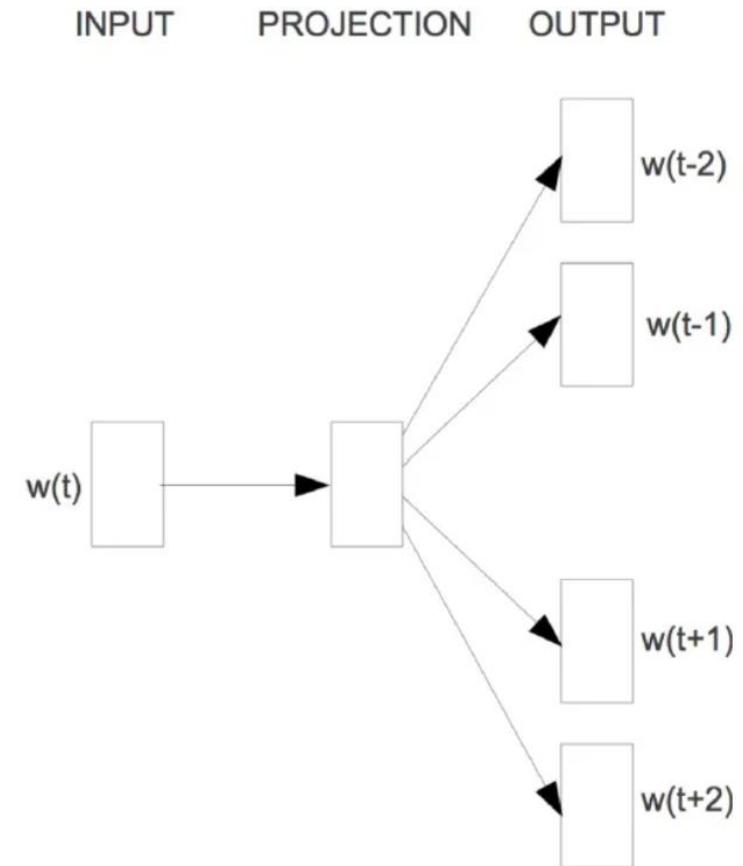
目标函数

$$J = \sum_{w \in corpus} P(w | context(w))$$

$$J = \sum_{i \in corpus} \log \left( \frac{\exp(w_i^T \tilde{w}_I)}{\sum_{k=1}^V \exp(w_i^T \tilde{w}_k)} \right)$$

$$J = \sum_{i \in corpus, j \in context(i)} \log \left( \frac{\exp(w_i^T \tilde{w}_j)}{\sum_{k=1}^V \exp(w_i^T \tilde{w}_k)} \right)$$

# Word2Vec: Skip-Gram 模型



- 无隐层
- 投影层也可省略
- 每个词向量作为log-linear模型的输入

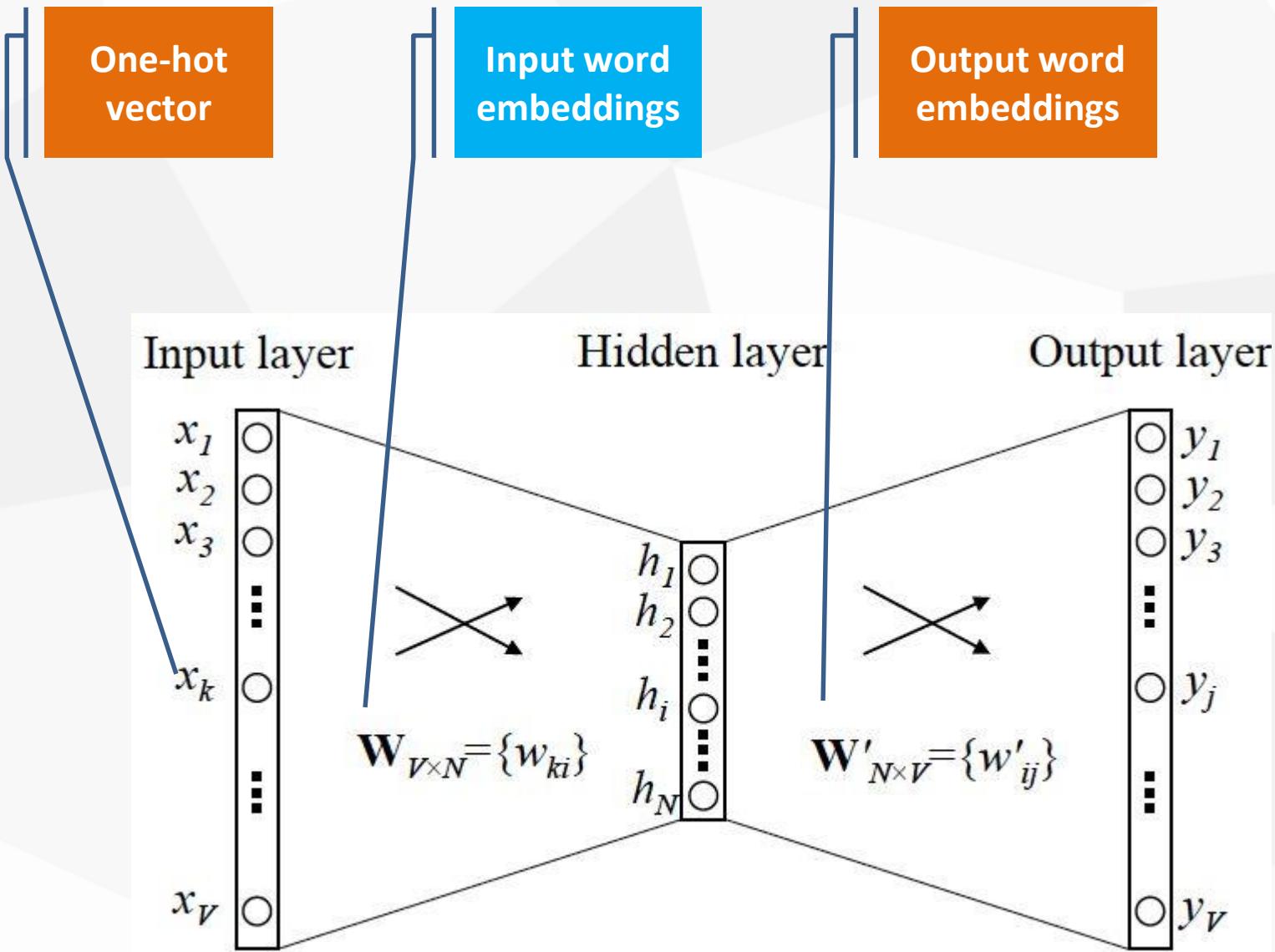
目标函数：

$$\frac{1}{T} \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log(p(w_{t+j}|w_t))$$

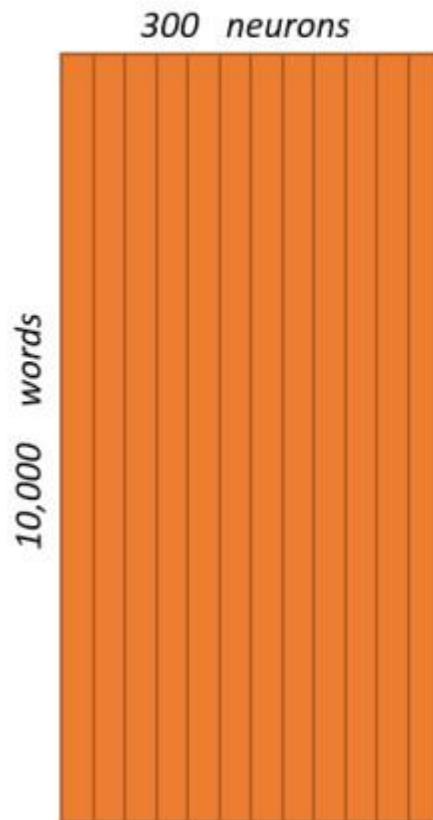
- 概率密度由Softmax给出

$$p(w_k|w_t) = \frac{\exp(\tilde{w}_k^T w_t)}{\sum_{m=1}^V \exp(\tilde{w}_m^T w_t)}$$

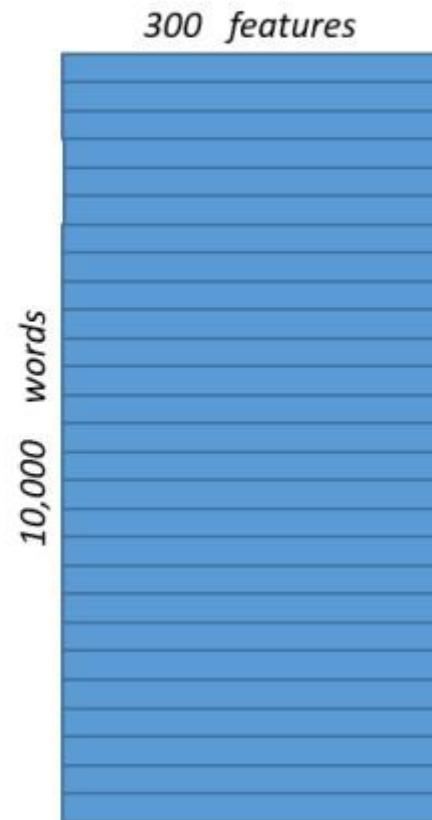
# 特例理解



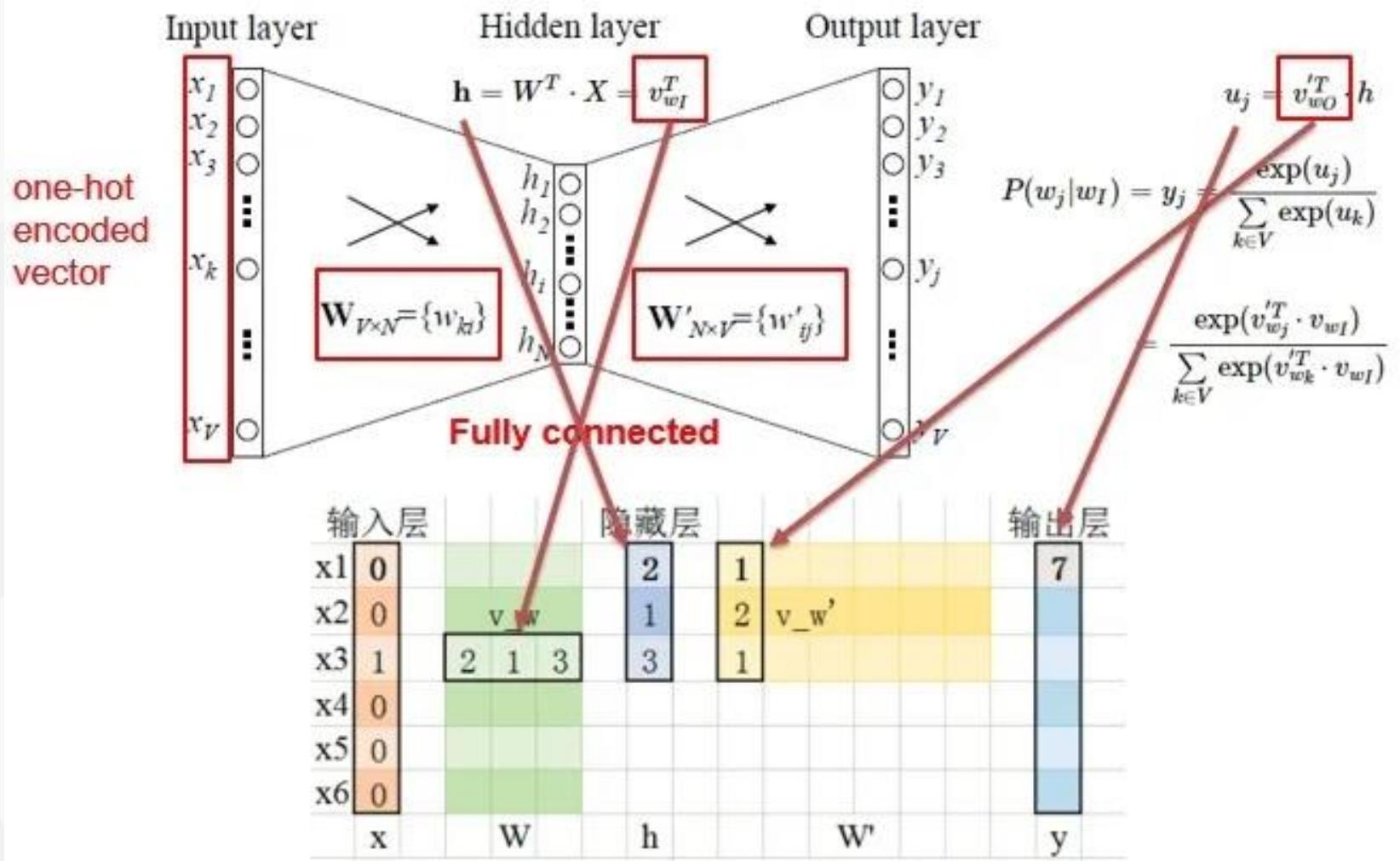
Hidden Layer  
Weight Matrix



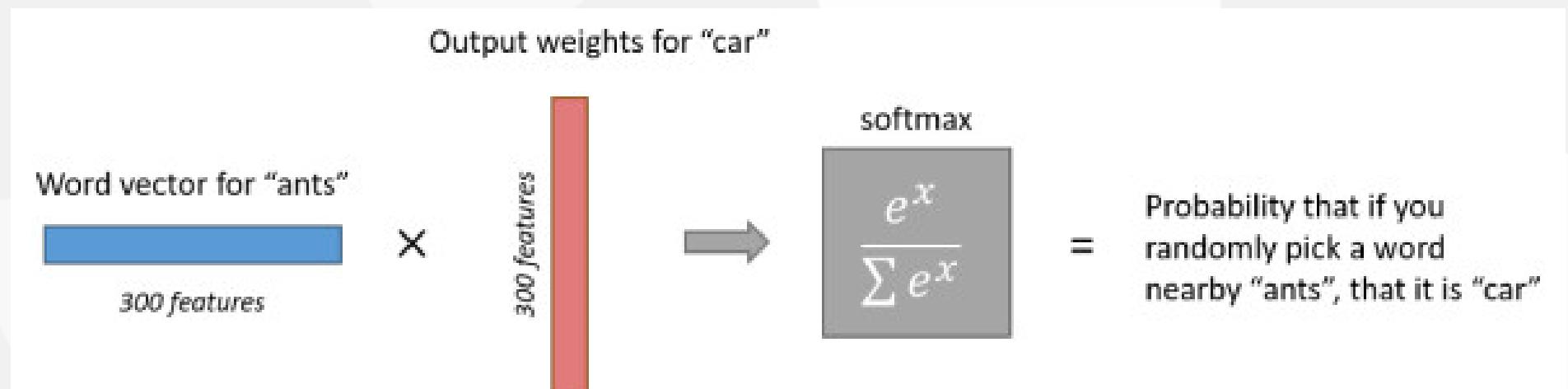
Word Vector  
Lookup Table!



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



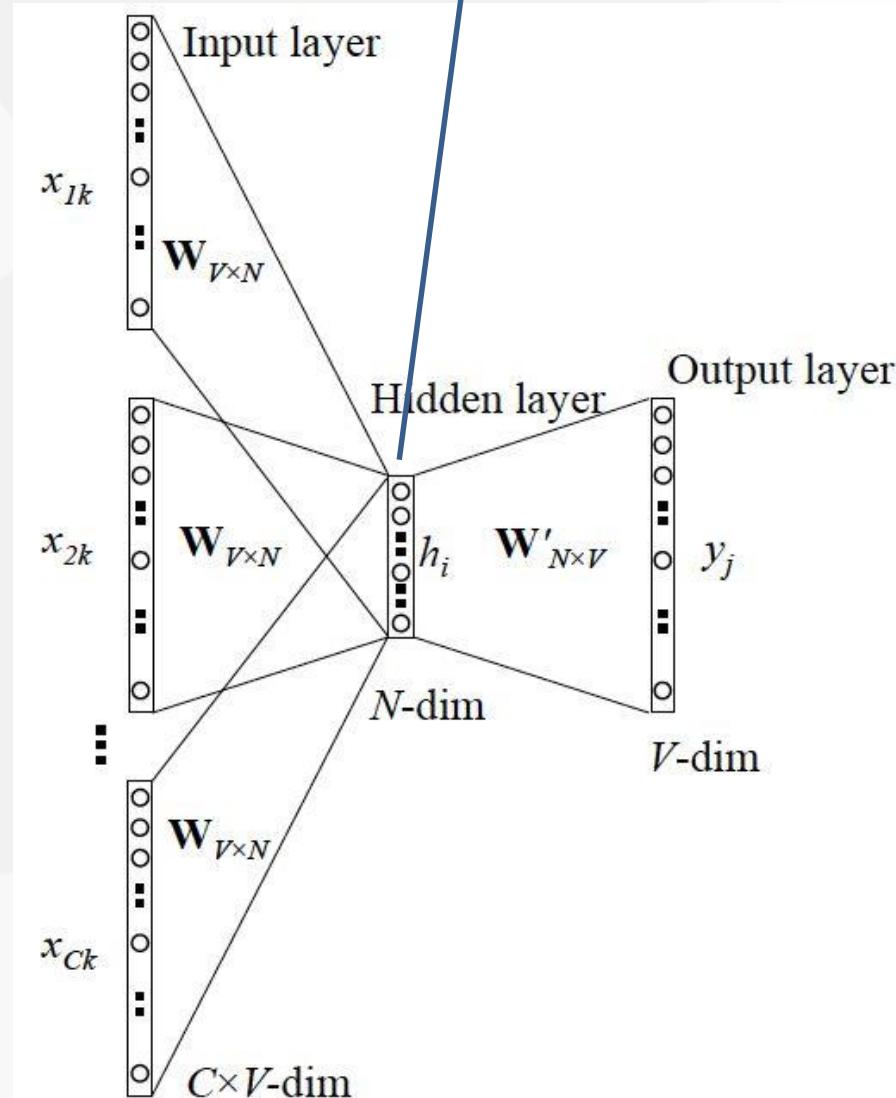
- 训练样本为 (input word: “ants”, output word: “car”) 的计算示意图。



# CBOW

Average of  
the context

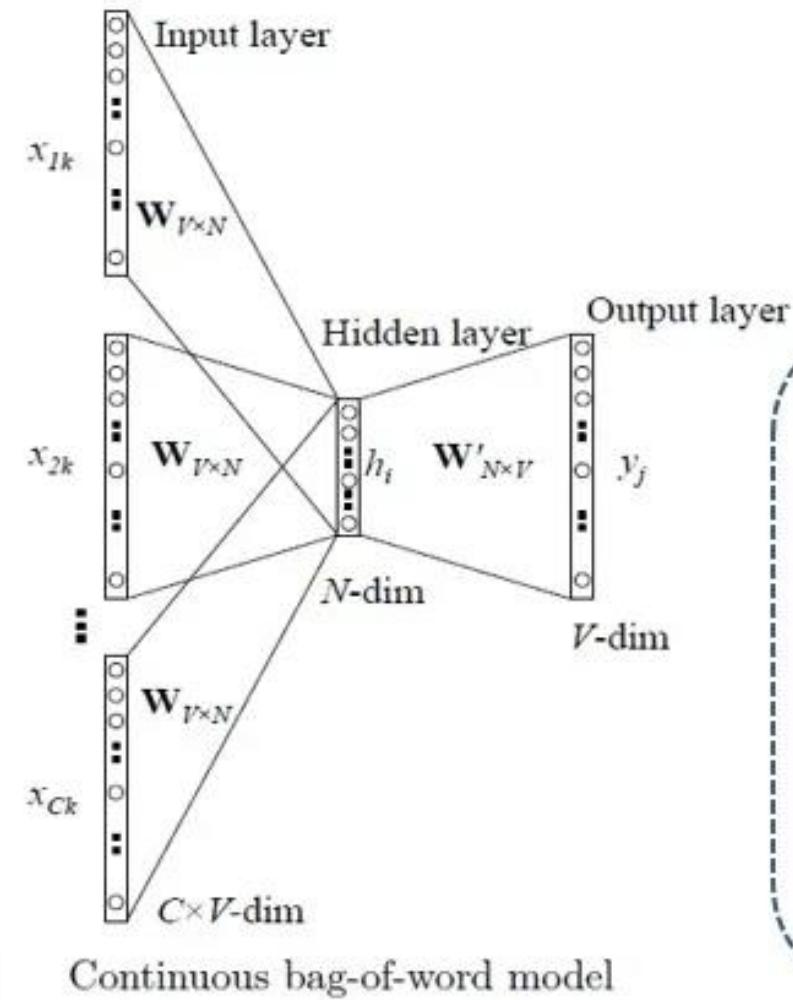
- CBOW tends to find the probability of a word occurring in a neighborhood (context).



# CBOW

- CBOW 的 objective function 和 LM objective 略有不同, 它预测的是中心单词:

$$J_{\theta} = \frac{1}{T} \sum_{t=1}^T \log p(w_t \mid w_{t-n}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+n}).$$



### Simple CBOW model

$$\mathbf{h} = \mathbf{W}^T \cdot \mathbf{x} = \mathbf{v}_{wI}^T \quad u_j = \mathbf{v}_{wO}^T \cdot \mathbf{h}$$

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

### Continuous bag-of-word model

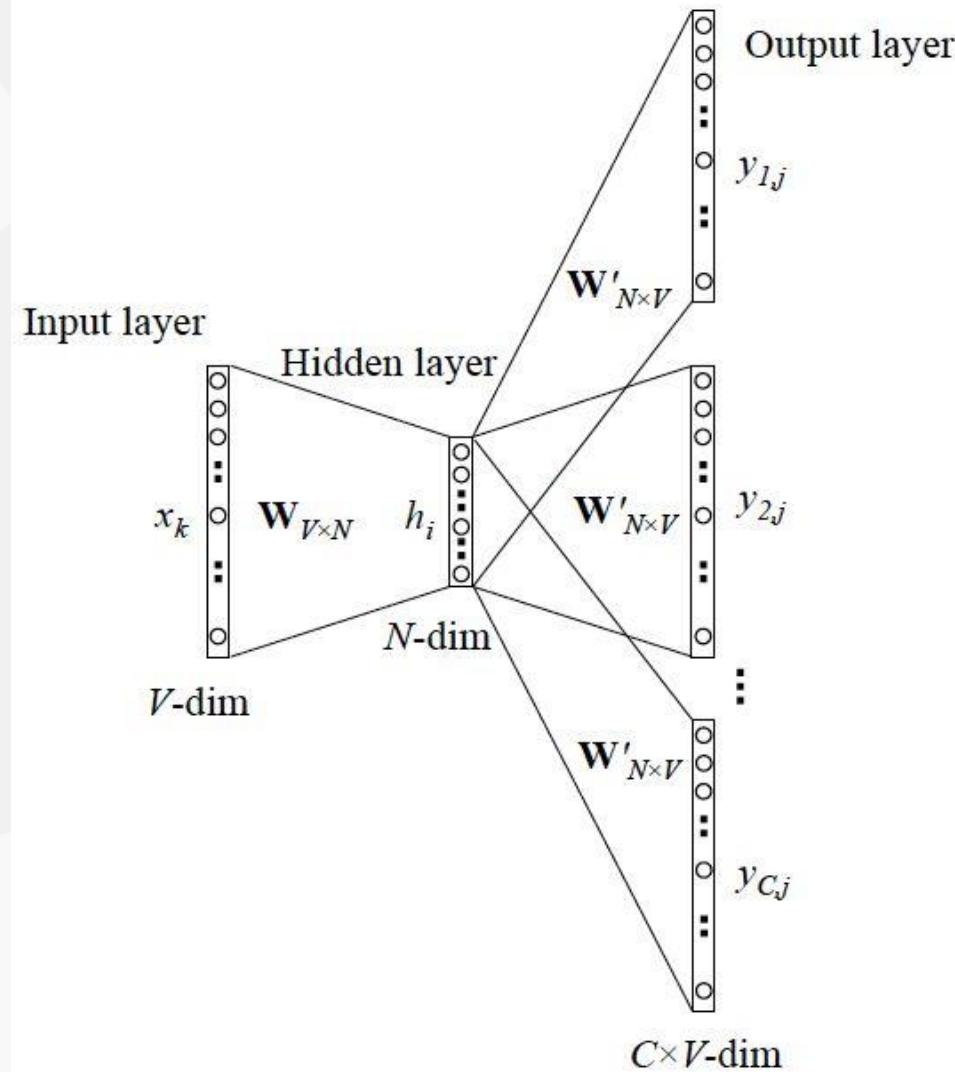
$$\begin{aligned} \mathbf{h} &= \frac{1}{C} \mathbf{W}^T (\mathbf{x}_1 + \mathbf{x}_2 + \cdots + \mathbf{x}_C) \\ &= \frac{1}{C} (\mathbf{v}_{w_1} + \mathbf{v}_{w_2} + \cdots + \mathbf{v}_{w_C})^T \end{aligned}$$

$$\begin{aligned} E &= -\log p(w_O | w_{I,1}, \dots, w_{I,C}) \\ &= -u_{j^*} + \log \sum_{j'=1}^V \exp(u_{j'}) \end{aligned}$$

$$= -\mathbf{v}_{wO}^T \cdot \mathbf{h} + \log \sum_{j'=1}^V \exp(\mathbf{v}_{w_{j'}}^T \cdot \mathbf{h})$$

# Skip-gram

- Skip-gram tends to learn the different contexts separately.
- Skip-gram requires more data to train and also contains more knowledge about the context.

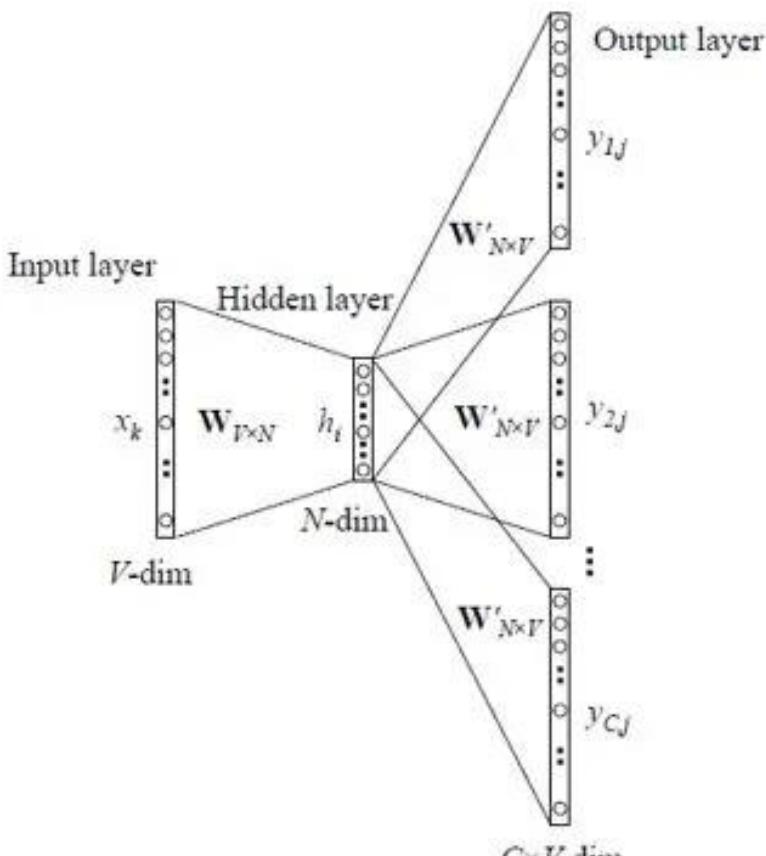


# Skip-gram

- Skip-gram 的 objective function 很自然地表示为所有邻近单词的概率和:

$$J_\theta = \frac{1}{T} \sum_{t=1}^T \sum_{-n \leq j \leq n, j \neq 0} \log p(w_{t+j} | w_t).$$

$$p(w_O | w_I) = \frac{\exp \left( {v'_{w_O}}^\top v_{w_I} \right)}{\sum_{w=1}^W \exp \left( {v'_{w}}^\top v_{w_I} \right)}$$



The skip-gram model

**input → hidden**

$$\mathbf{h} = \mathbf{W}_{(k,\cdot)}^T := \mathbf{v}_{w_I}^T$$

**hidden → output**

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

**Share the same weights:**

$$u_{c,j} = u_j = \mathbf{v}_{w_j}^T \cdot \mathbf{h}, \text{ for } c = 1, 2, \dots, C$$

**Loss function:**

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

# 训练技巧

- hierarchical softmax
  - 本质是把  $N$  分类问题变成  $\log(N)$  次二分类
  - H-Softmax 能将 word prediction 任务加速至少 50 倍!
- negative sampling
  - 本质是预测总体类别的一个子集

# Hierarchical Softmax

- Hierarchical Softmax用树形结构替代了输出层的结构，计算更加高效。
- Hierarchical Softmax采用是Huffman最优二叉树，叶子节点分配给词典里的词。赋予高频词短编码，低频词长编码。高频词离根节点距离更近，从而使得训练速度加快。
- full softmax需要一次计算所有的 $V$ 个词，而 hierarchical softmax却只需要计算大约 $\log_2(V)$ （即树根到该叶子节点的路径长度）个词，大大减少了计算的复杂度。

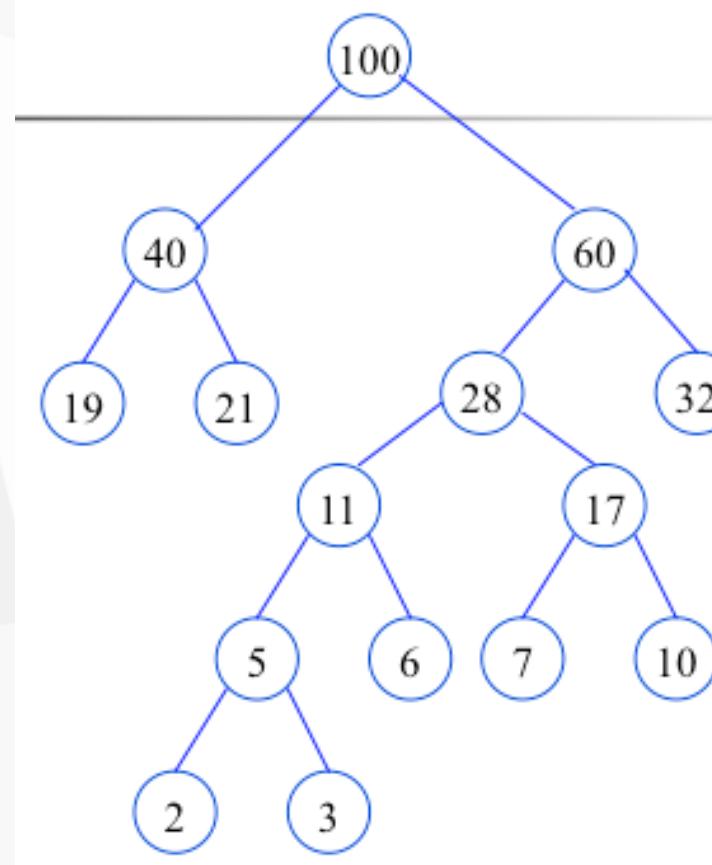
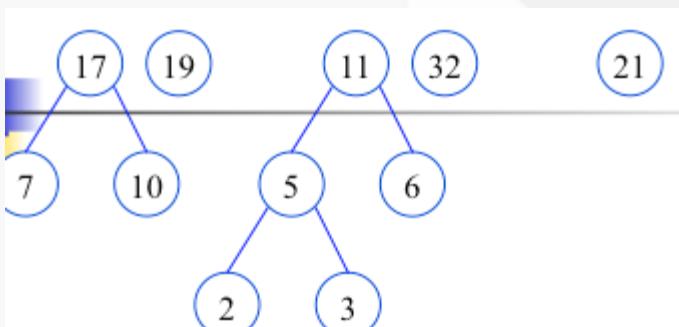
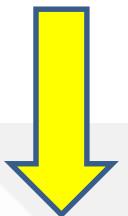
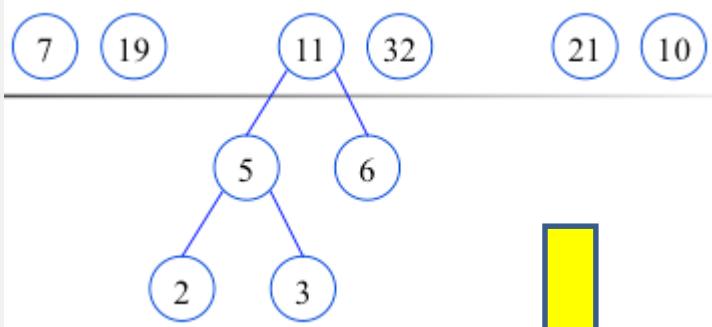
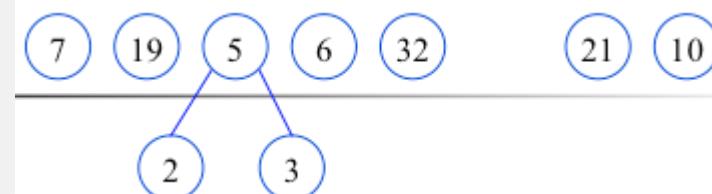
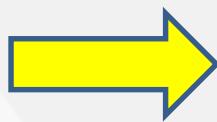
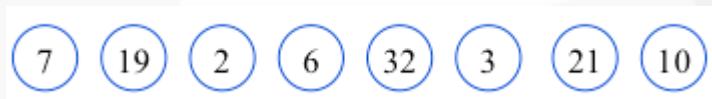
# Huffman树

- 给定n个权值 $w_1, w_2, \dots, w_n$ 作为二叉树的n个叶子节点，则以此构造Huffman树的算法如下所示：
  - 将n个权值看成是n棵树的森林（每棵树仅有一个节点）。
  - 从森林中选择两个根节点权值最小的树合并，作为一棵新树的左右子树，且新树的根节点权值为其左右子树根节点权值之和。
  - 从森林中删除被选中的两棵树，并且将新树加入森林。
  - 重复2-3步，知道森林中只有一棵树为止，则该树即所求的Huffman树。



让权值小的叶子节点层次更深，权值大的叶子节点层次更浅。

# case

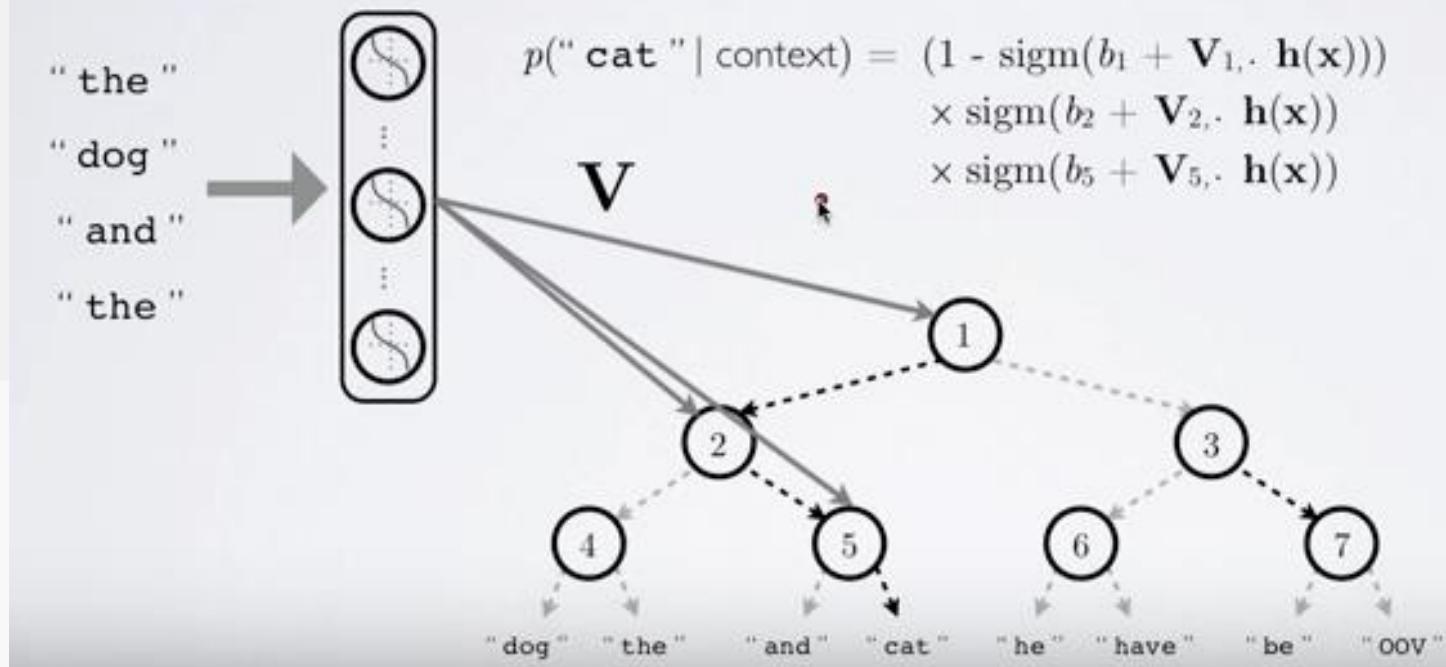


# Huffman编码

- 利用Huffman树设计的二进制前缀编码，就被称为**Huffman**编码。
- Word2vec算法用了Huffman编码，它把训练语料中的词当成叶子节点，其在语料中出现的次数当做权值，通过构造Huffman树来对每一个词进行Huffman编码。

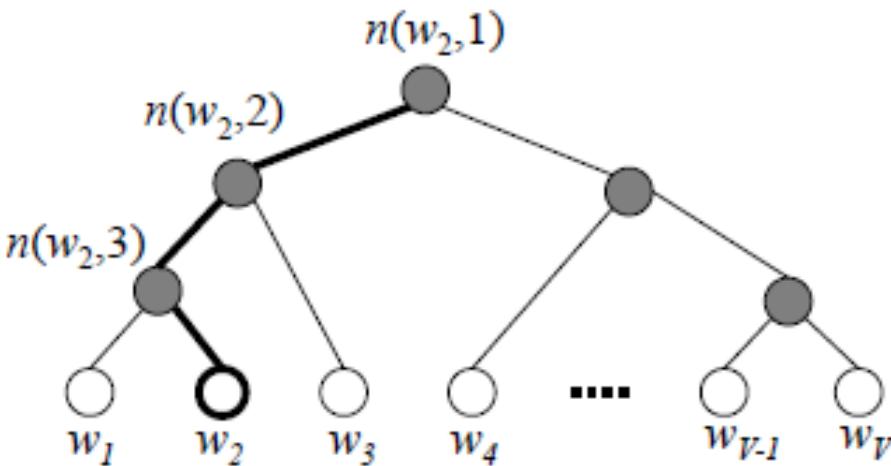
# Hierarchical Softmax 案例

- Example: ["the", "dog", "and", "the", "cat"]



# Hierarchical Softmax 最大似然函数

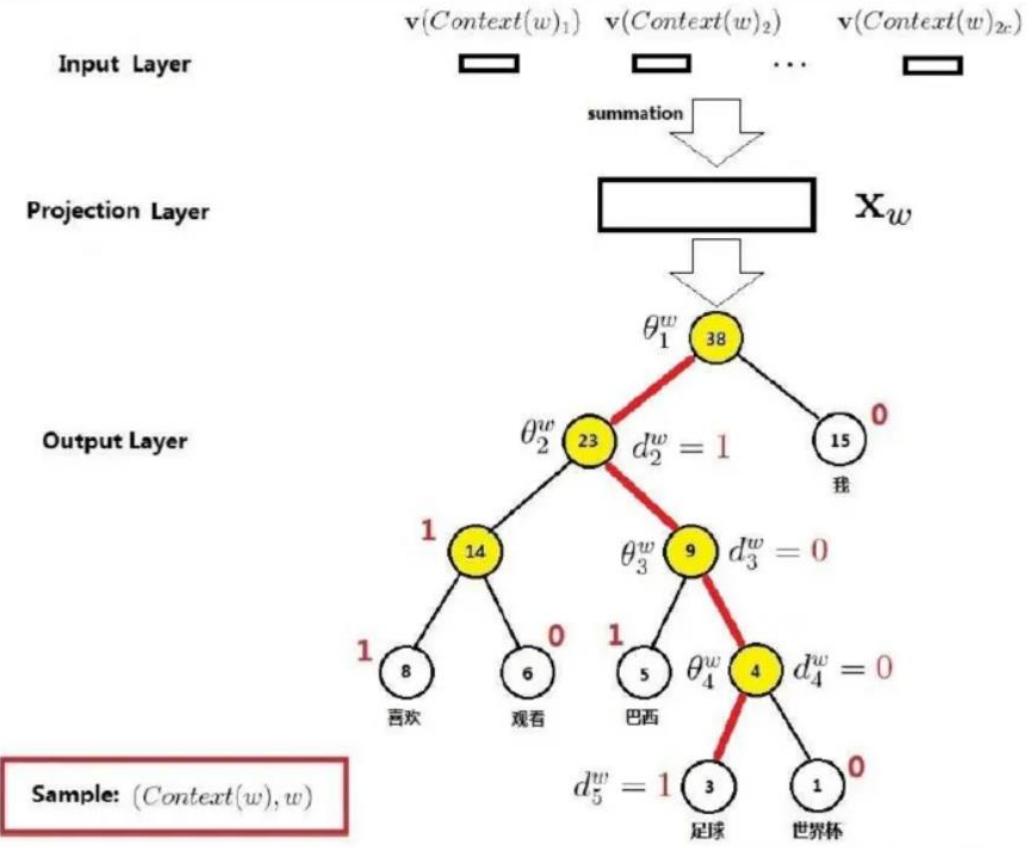
$$\prod_{i=1}^3 P(n(w_i), i) = \left(1 - \frac{1}{1 + e^{-x_0^T \theta_1}}\right) \left(1 - \frac{1}{1 + e^{-x_0^T \theta_2}}\right) \frac{1}{1 + e^{-x_0^T \theta_3}}$$



# Hierarchical Softmax 总结

- 在 H-softmax 中，不再需要 output word embeddings，取而代之的是，每个赫夫曼树中的节点都有 embeddings，各不相同。因此实际上 H-softmax 和 softmax 拥有差不多的参数量。
- 树结构对于 h-softmax 很重要，比如为相似的概率分配相似的路径。
- 注意：只能在训练阶段获得加速，因为此时事先知道了 target word，也就知道了其在树中的路径。测试时，要计算最有可能的单词，仍需要计算所有单词的概率。
- 词库庞大，赫夫曼树非常复杂，如果训练样本里的中心词  $w$  是一个很生僻的词，那么就得在霍夫曼树中辛苦的向下走很久。

# CBOW：层次Softmax



- 使用 Huffman Tree 来编码输出层的词典
- 只需要计算路径上所有非叶子节点词向量的贡献即可
- 计算量降为树的深度  $V \Rightarrow \log_2(V)$

W=“足球”时的相关记号示意图

## CBOW：层次Softmax

$$p(w|w_I) = \prod_{j=1}^{L(w)-1} \sigma \left( \llbracket n(w, j+1) = \text{ch}(n(w, j)) \rrbracket \cdot v'_{n(w,j)}^\top v_{w_I} \right)$$

- Sigmoid函数  $\sigma(x) = 1/(1 + \exp(-x))$
- $n(w,j)$ : Huffman数内部第j层的节点
- $\text{ch}(n(w,j))$ :  $n$ 节点的child节点
- $[[n(w,j+1)=\text{ch}(n(w,j))]]$  是选择函数，表明只选择从根节点到目标叶节点路径上的内部节点

$$\sum_{w=1}^W p(w|w_I) = 1$$

# Negative Sampling

- 比如我们有一个训练样本，中心词是 $w$ ，它的上下文共有 $2c$ 个词，记为 $\text{context}(w)$ 。由于这个中心词 $w$ ，的确和 $\text{context}(w)$ 相关，因此它是一个真实的正例。通过 Negative Sampling 采样，我们得到 $M$ 个和 $w$ 不同的中心词  $w_i, i=1,2,\dots,M$ ，这样 $\text{context}(w)$ 和 $w_i$ 就组成了 $M$ 个并不真实存在的负例。利用这一个正例和 $M$ 个负例，我们进行二元逻辑回归，得到负采样对应每个词 $w_i$ 对应的模型参数 $\theta_i$ ，和每个词的词向量。
- Negative Sampling 由于没有采用霍夫曼树，每次只是通过采样 $M$ 个不同的中心词做负例，就可以训练模型，因此整个过程要比 Hierarchical Softmax 简单。
- 不过有两个问题还需要弄明白：
  - 1) 如果通过一个正例和 $M$ 个负例进行二元逻辑回归呢？
  - 2) 如何进行负采样呢？

# Negative Sampling最大似然函数

正例和负  
例的概率

$$P(\text{context}(w_0), w_i) = \sigma(x_{w_0}^T \theta^{w_i}), y_i = 1, i = 0$$

$$P(\text{context}(w_0), w_i) = 1 - \sigma(x_{w_0}^T \theta^{w_i}), y_i = 0, i = 1, 2, \dots, neg$$

期望

$$\prod_{i=0}^{neg} P(\text{context}(w_0), w_i) = \sigma(x_{w_0}^T \theta^{w_0}) \prod_{i=1}^{neg} (1 - \sigma(x_{w_0}^T \theta^{w_i}))$$

似然和对  
数似然

$$\prod_{i=0}^{neg} \sigma(x_{w_0}^T \theta^{w_i})^{y_i} (1 - \sigma(x_{w_0}^T \theta^{w_i}))^{1-y_i}$$

$$L = \sum_{i=0}^{neg} y_i \log(\sigma(x_{w_0}^T \theta^{w_i})) + (1 - y_i) \log(1 - \sigma(x_{w_0}^T \theta^{w_i}))$$

# 负采样方法

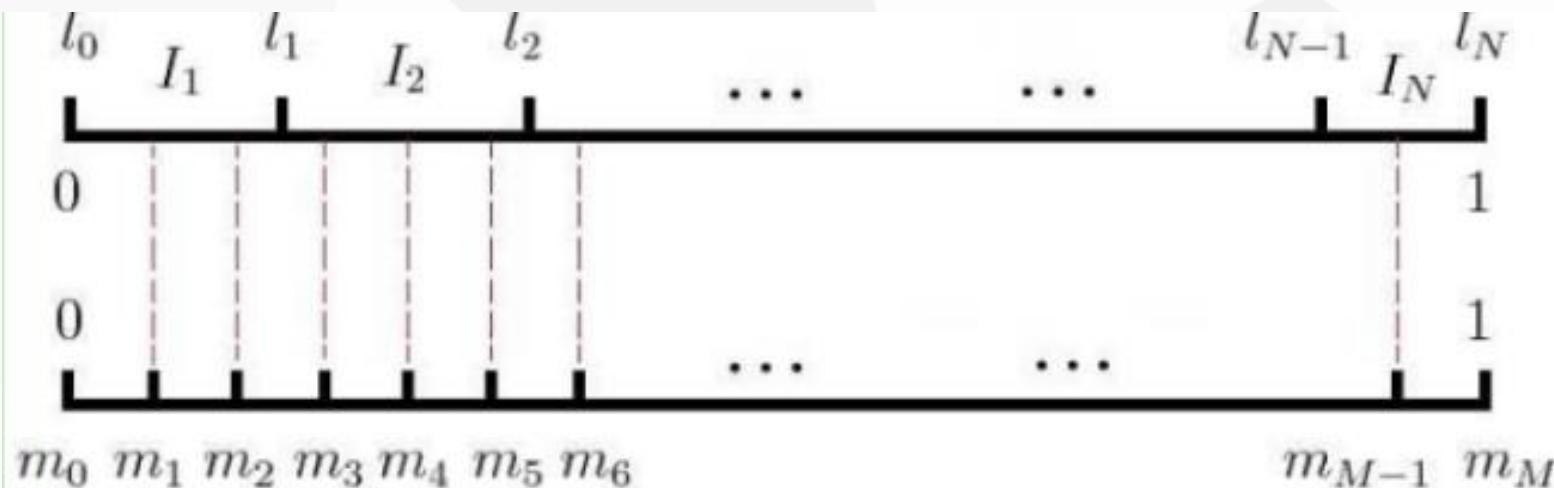
- 如果词汇表的大小为 $V$ ,那么我们就将一段长度为1的线段分成 $V$ 份,每份对应词汇表中的一个词。当然每个词对应的线段长度是不一样的,高频词对应的线段长,低频词对应的线段短。每个词 $w$ 的线段长度由下式决定:

$$\text{len}(w) = \frac{\text{count}(w)}{\sum_{u \in \text{vocab}} \text{count}(u)}$$

$$\text{len}(w) = \frac{\text{count}(w)^{3/4}}{\sum_{u \in \text{vocab}} \text{count}(u)^{3/4}}$$

# 负采样方法

- 在采样前，我们将这段长度为1的线段划分成 $M$ 等份，这里 $M \gg V$ ，这样可以保证每个词对应的线段都会划分成对应的小块。而 $M$ 份中的每一份都会落在某一个词对应的线段上。在采样的时候，我们只需要从 $M$ 个位置中采样出 $neg$ 个位置就行，此时采样到的每一个位置对应到的线段所属的词就是我们的负例词。在word2vec中， $M$ 取值默认为 $10e8$ 。



## CBOW：负例采样

$P(w|\text{context}(w))$ : 一个正样本,  $V-1$ 个负样本, 对负样本做采样

$$g(w) = \sigma(\mathbf{x}_w^\top \theta^w) \prod_{u \in \text{NEG}(w)} [1 - \sigma(\mathbf{x}_w^\top \theta^u)]$$

$X_w$  是  $\text{context}(w)$  中词向量的和

$\theta^u$  是词  $u$  对应的一个(辅助)向量

$\text{NEG}(w)$  是  $w$  的负样本采样子集

损失函数: 对语料库中所有词  $w$  求和

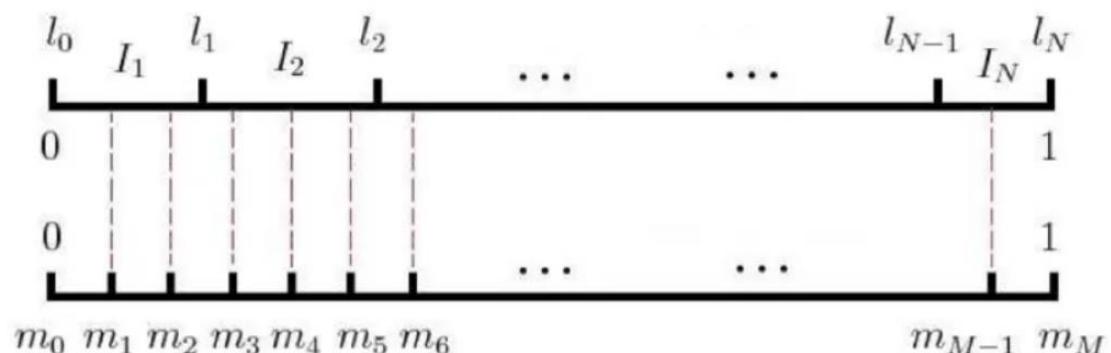
$$L = \sum_{w \in \mathcal{C}} \log(g(w))$$
$$\sum_{w \in \mathcal{C}} \left\{ \log [\sigma(\mathbf{x}_w^\top \theta^w)] + \sum_{u \in \text{NEG}(w)} \log [\sigma(-\mathbf{x}_w^\top \theta^u)] \right\}$$

## CBOW：负例采样

词典中的每一个词对应一条线段，所有词组成了 $[0, 1]$ 间的剖分

$$len(w) = \frac{\text{counter}(w)}{\sum_{u \in \mathcal{D}} \text{counter}(u)}, \quad \text{实际使用中取} \text{counter}(w)^{(3/4)} \text{效果最好}$$

$l_1, l_2, \dots, l_N$  组成了 $[0, 1]$ 间的剖分



将 $[0, 1]$ 划分为 $M=10^8$ 等分，每次随机生成一个 $[1, M-1]$ 间的整数，看落在那个词对应的剖分上。

# Word2vec的缺陷

- Word2Vec只考虑到了局部上下文信息，而忽略了全局信息；
- Word2vec对每个local context window单独训练，没有利用包含在global co-currence矩阵中的统计信息；
- Word2Vec只考虑了上下文的共现性，而忽略了彼此之间的顺序性；
- Word2vec生成的是静态词向量，不能应对多义词问题，即无法处理词向量多义问题。

# Word2vec的应用

- Word2Vec主要原理是根据上下文来预测单词，一个词的意义往往可以从其前后的句子中抽取出来。
- 而用户的行为也是一种相似的时间序列，可以通过上下文进行推断。当用户浏览并与内容进行交互时，我们可以从用户前后的交互过程中判断行为的抽象特征，这就使得我们可以把词向量模型应用到推荐、广告领域当中。

# NLP领域的应用

- Word2Vec学习到的词向量代表了词的语义，可以用来做分类、聚类、也可以做词的相似度计算。
- 把Word2Vec生成的向量直接作为深度神经网络的输入，可以做sentiment analysis等工作。

# 图嵌入领域的应用

- 基于Word2Vec这一类的Graph Embedding方法有很多，具体可以参考论文：DeepWalk（是引入Word2Vec思想比较经典的图嵌入算法），node2vec，struc2vec 等等。

# 推荐领域的应用

- Airbnb在论文《Real-time Personalization using Embeddings for Search Ranking at Airbnb》中提出将用户的浏览行为组成List，通过Word2Vec方法学习item的向量，其点击率提升了21%，且带动了99%的预定转化率。该论文主要是在Skip-gram 模型的基础上做了改进。
- Yahoo在论文《E-commerce in Your Inbox: Product Recommendations at Scale》中提出Yahoo邮箱从发送到用户的购物凭证中抽取商品并组成List，通过Word2Vec学习并为用户推荐潜在的商品。

# 广告领域的应用

- Yahoo在论文《Scalable Semantic Matching of Queries to Ads in Sponsored Search Advertising》中提出将用户的搜索查询和广告组成List，并为其学习特征向量，以便对于给定的搜索查询可以匹配适合的广告。



# THANKS!