



北京航空航天大學  
BEIHANG UNIVERSITY

# 自然語言處理

人工智能研究院

主讲教师 沙磊



第三课

# 词向量概述

How do we represent the meaning of a word?

## 定义：含义 (meaning)

- 用一个词、词组等表示的概念
  - 一个人想用语言、符号等来表达的想法
  - 表达在作品、艺术等方面的思想

从语言方式(**linguistic way**)来看含义(**meaning**)：语言符号与语  
言意义(想法、事情)的相互对应

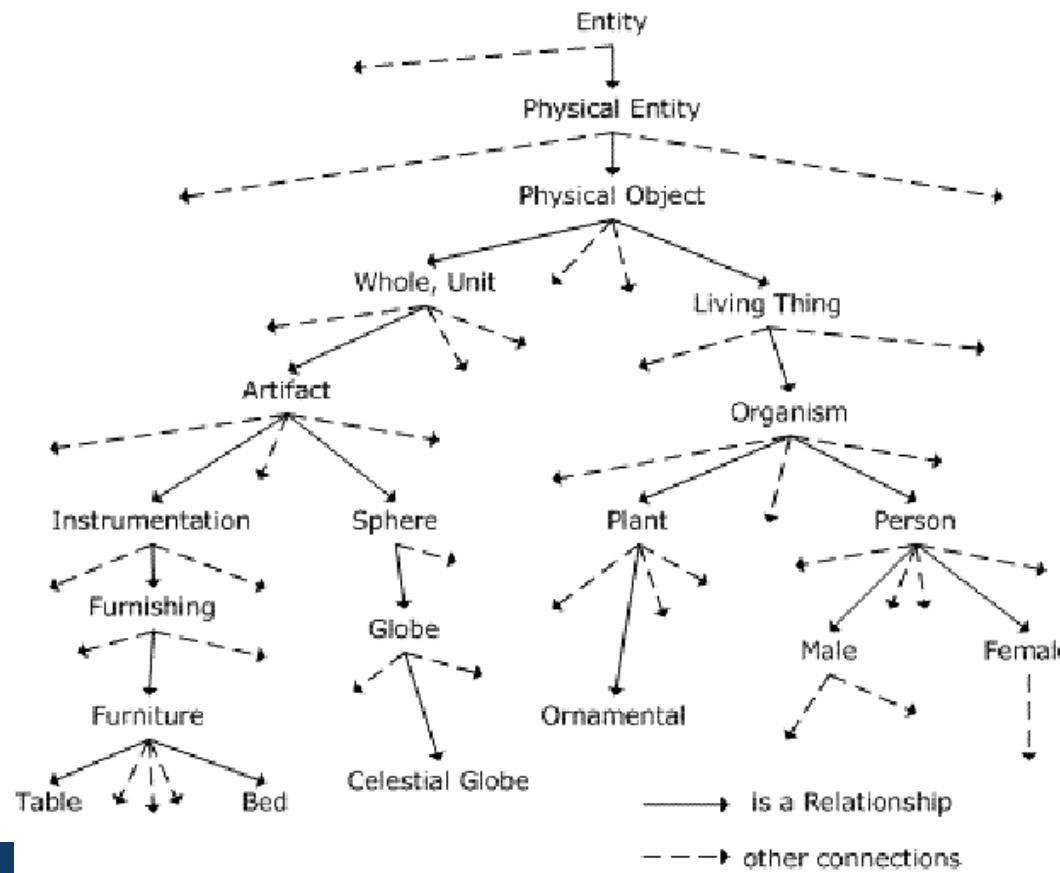
signifier (symbol)  $\Leftrightarrow$  signified (idea or thing)

= denotational semantics

tree  $\iff \{ \text{}, \text{}, \text{}, \dots \}$

# How do we have usable meaning in a computer?

- 要使用计算机处理文本词汇，一种处理方式是**WordNet**: 即构建一个包含同义词(synonym)集和上位词(hypernym) (“is a”关系)的列表的辞典。



# WordNet?

e.g., synonym sets containing “good”:

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
        ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
...
adverb: well, good
adverb: thoroughly, soundly, good
```

e.g., hypernyms of “panda”:

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(pandaclosure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

# Problems with resources like WordNet

- 忽略了词汇的细微差别
  - 例如“proficient”被列为“good”的同义词。这只在某些上下文中是正确的。
- 缺少单词的新含义
  - 难以持续更新！
  - 例如：wicked、badass、nifty、wizard、genius、ninja、bombast
- 因为是小部分专家构建的，有一定的主观性
- 构建与调整都需要很多的人力成本
- 无法定量计算出单词相似度

# Representing words as discrete symbols

- 在传统的自然语言处理中，我们会把词语看作离散的符号：例如 hotel、conference、motel 等。
- 一种文本的离散表示形式是把单词表征为 one-hot 向量的形式
  - One-hot：只有一个1，其余均为0的稀疏向量
- 在 one-hot 向量表示中，向量维度 = 词汇量（如 500,000），以下为一些 one-hot 向量编码过后的单词向量示例：

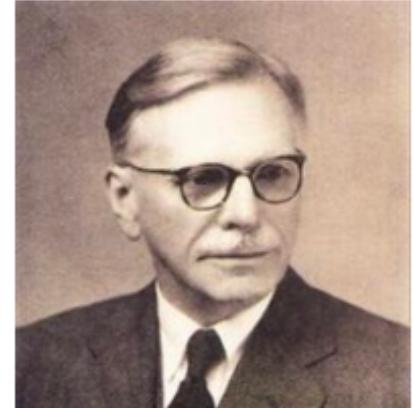
**motel = [0 0 0 0 0 0 0 0 0 1 0 0 0 0]**

**hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0]**

# Problem with words as discrete symbols

- 例子：用户搜索“Seattle motel”，我们想要找到包含“Seattle hotel”的文档，
- 然而：  
 $\text{motel} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0]$   
 $\text{hotel} = [0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0]$
- 所有词向量是正交的。对于one-hot向量，没有关于相似性概念。
- 解决思路：
  - ① 使用类似WordNet的工具中的同义词表？
    - 会因不够完整而失败
  - ② 通过大量数据学习词向量本身相似性，获得更精确的稠密词向量编码

# Representing words by their context



- **分布式语义：**一个单词的意思是由经常出现在它附近的单词给出的

英国语言学家 J.R. Firth

- “You shall know a word by the company it keeps” (J. R. Firth 1957.11)
- 这是现代统计NLP最成功的理念之一，总体思路有点物以类聚，人以群分的感觉
- 当一个单词w出现在文本中时，它的上下文是出现在其附近的一组单词(在一个固定大小的窗口中)
- 基于海量数据，使用w的许多上下文来构建w的表示
- 如图所示，banking的含义可以根据上下文的内容来表示

*...government debt problems turning into banking crises as happened in 2009...*  
*...saying that Europe needs unified banking regulation to replace the hodgepodge...*  
*...India has just given its banking system a shot in the arm...*



These context words will represent **banking**

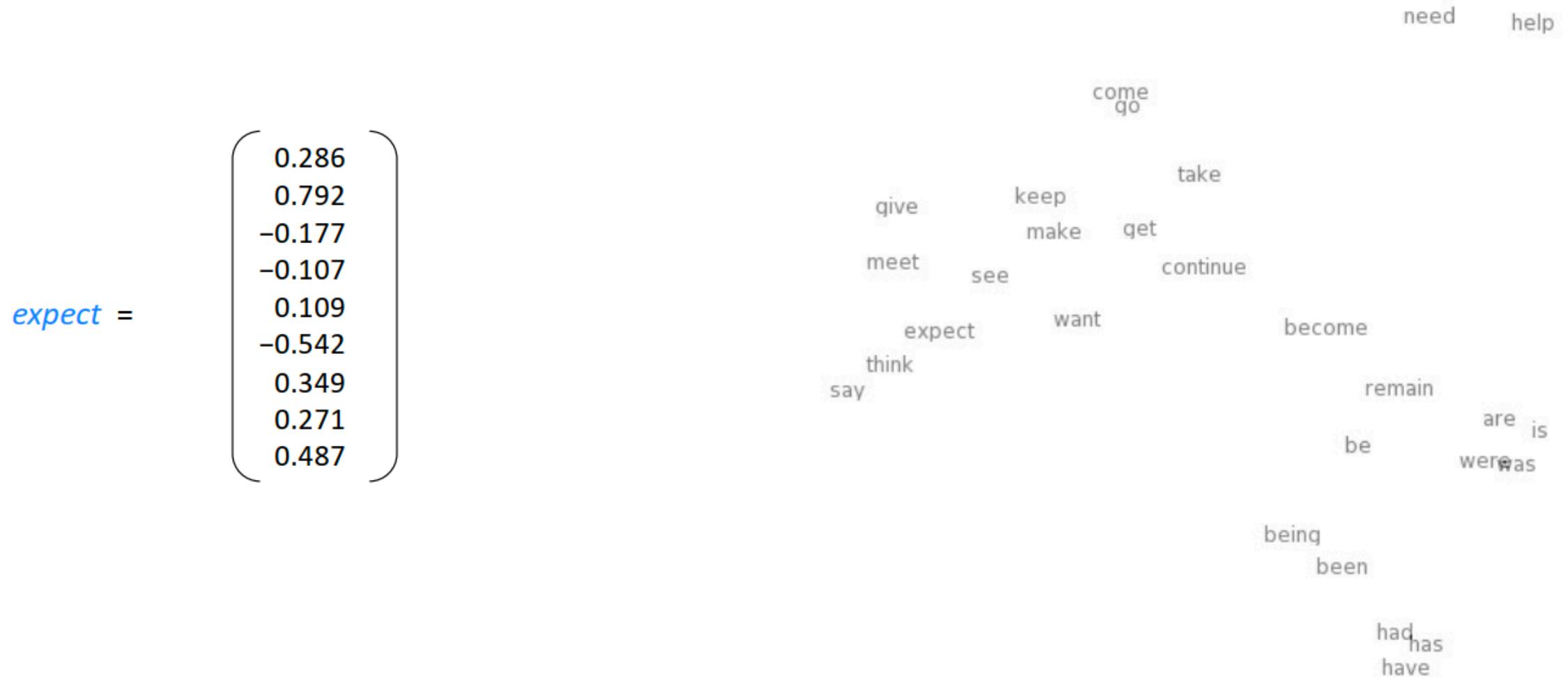
# 词向量

- 下面我们要介绍词向量的构建方法与思想，我们希望为每个单词构建一个稠密表示的向量，使其与出现在相似上下文中的单词向量相似。

$$\begin{aligned} \textcolor{magenta}{banking} &= \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix} & \textcolor{magenta}{monetary} &= \begin{pmatrix} 0.413 \\ 0.582 \\ -0.007 \\ 0.247 \\ 0.216 \\ -0.718 \\ 0.147 \\ 0.051 \end{pmatrix} \end{aligned}$$

- 词向量 (word vectors) 有时被称为词嵌入 (word embeddings) 或词表示 (word representations)，是一种分布式表示 (distributed representation)。

# Word meaning as a neural word vector – visualization



# Word2vec: Overview

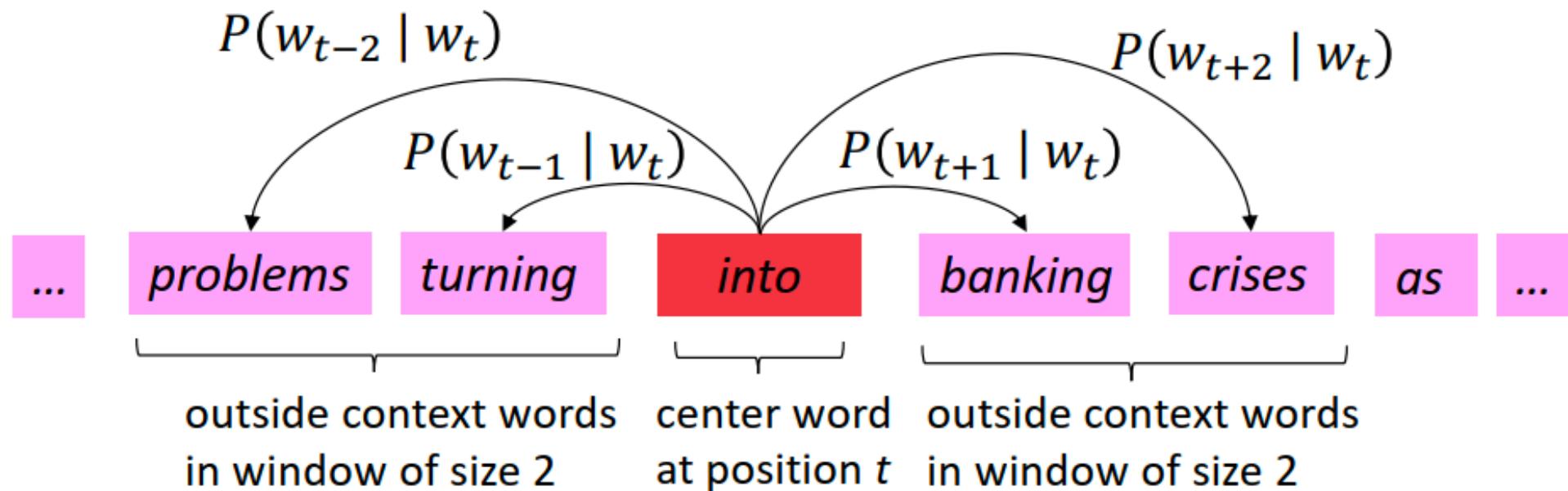
**Word2vec** (Mikolov et al. 2013)是一个学习词向量表征的框架。

Idea:

- We have a large corpus (“body”) of text: a long list of words
- Every word in a fixed vocabulary is represented by a vector
- Go through each position  $t$  in the text, which has a center word  $c$  and context (“outside”) words  $o$
- Use the similarity of the word vectors for  $c$  and  $o$  to calculate the probability of  $o$  given  $c$   $p(o|c)$  (or vice versa  $p(c|o)$ )
- Keep adjusting the word vectors to maximize this probability

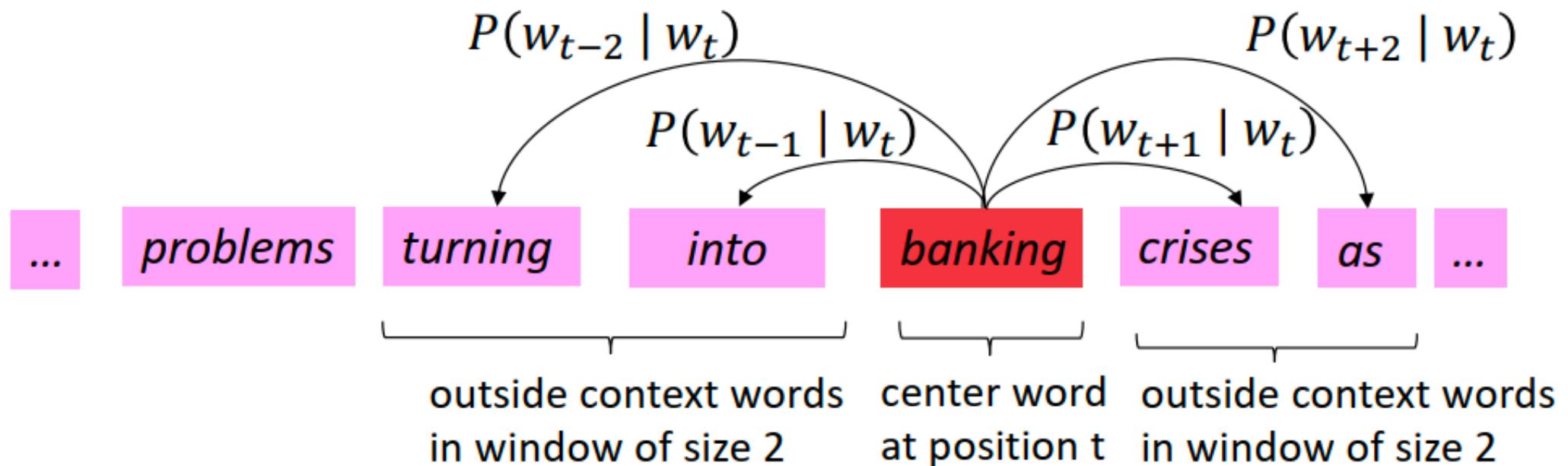
# Word2vec: Overview

Example windows and process for computing  $P(w_{t+j} | w_t)$



# Word2Vec Overview

Example windows and process for computing  $P(w_{t+j} | w_t)$



# Word2vec: objective function

- 对于每个位置  $t = 1, \dots, T$ , 在大小为  $m$  的固定窗口内预测上下文单词, 给定中心词  $w_t$ , 极大似然函数可以表示为:

$$Likelihood = L(\theta) = \prod_{t=1}^T \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$$

$\theta$  代表所有要优化的变量

- 对应上述似然函数的目标函数 (objective/cost/loss function)  $J(\theta)$  可以取作 (平均) 负对数似然

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

Minimizing objective function  $\Leftrightarrow$  Maximizing predictive accuracy

# Word2vec: objective function

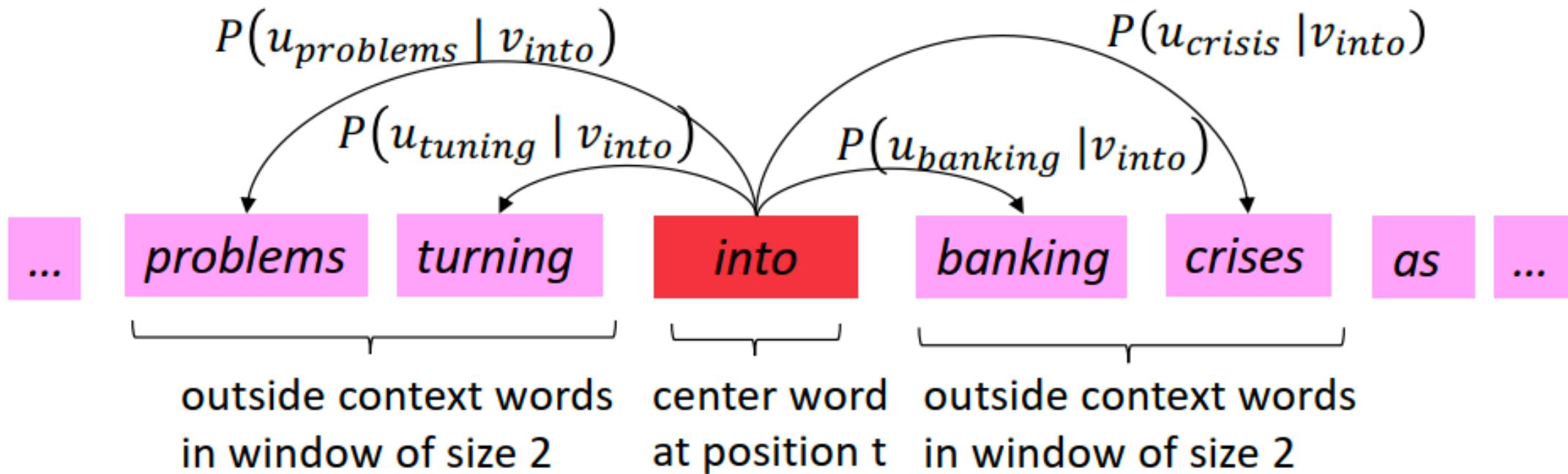
- We want to minimize the objective function

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

- Question: How to calculate  $P(w_{t+j} | w_t; \theta)$  ?
- Answer: We will use two vectors per word w.
  - $v_w$  when w is a center word
  - $u_w$  when w is a context word
- Then for a center word c and a context word o:  $P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$

# Word2Vec with Vectors

- 下图为计算  $P(w_{t+j} | w_t)$  的示例，这里把  $P(problems | into ; u_{problems}, v_{into}, \theta)$  简写为  $P(u_{problems} | v_{into})$ ，例子中的上下文窗口大小2，即“左右2个单词+一个中心词”。



# Word2vec: prediction function

② Exponentiation makes anything positive

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

① Dot product compares similarity of  $o$  and  $c$ .  
 $u^T v = u \cdot v = \sum_{i=1}^n u_i v_i$   
Larger dot product = larger probability

③ Normalize over entire vocabulary  
to give probability distribution

- Softmax 函数举例： $\mathbb{R}^n \rightarrow (0,1)^n$

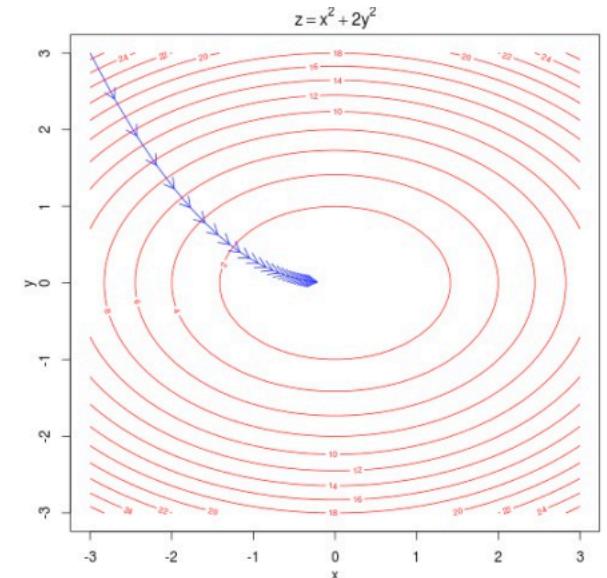
$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)} = p_i$$

- Softmax函数将任意实数向量  $x_i$  ( $i=1, \dots, n$ ) 映射为一个概率分布  $p_i$  ( $i=1, \dots, n$ )
  - Max: 最大元素拥有最大的概率值
  - Soft: 小的元素依然拥有概率值，不会是0
  - 在深度学习中普遍应用

# To train the model: Optimize value of parameters to minimize loss

- 模型训练过程中，我们需要逐步调整参数来最小化loss
- $\theta$ 代表所有模型的参数
- 此处，我们每个词向量长度为d，一共V个词，那么我们有→
- 每个词有两个向量
- 整个优化过程顺着梯度的方向下降，直到找到最低点
- 所有向量的梯度都要计算！

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$



# 具体的loss function求梯度的流程

$$\min J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log p(w'_{t+j} | w_t)$$

↑                      ↑  
文本长度              窗口长度

负对数似然

此处：

$$p(o|c) = \frac{\exp(u_o^\top v_c)}{\sum_{w=1}^V \exp(u_w^\top v_c)}$$

接下来，看看梯度怎么求。

# 具体的loss function求梯度的流程

此处是对向量求偏导

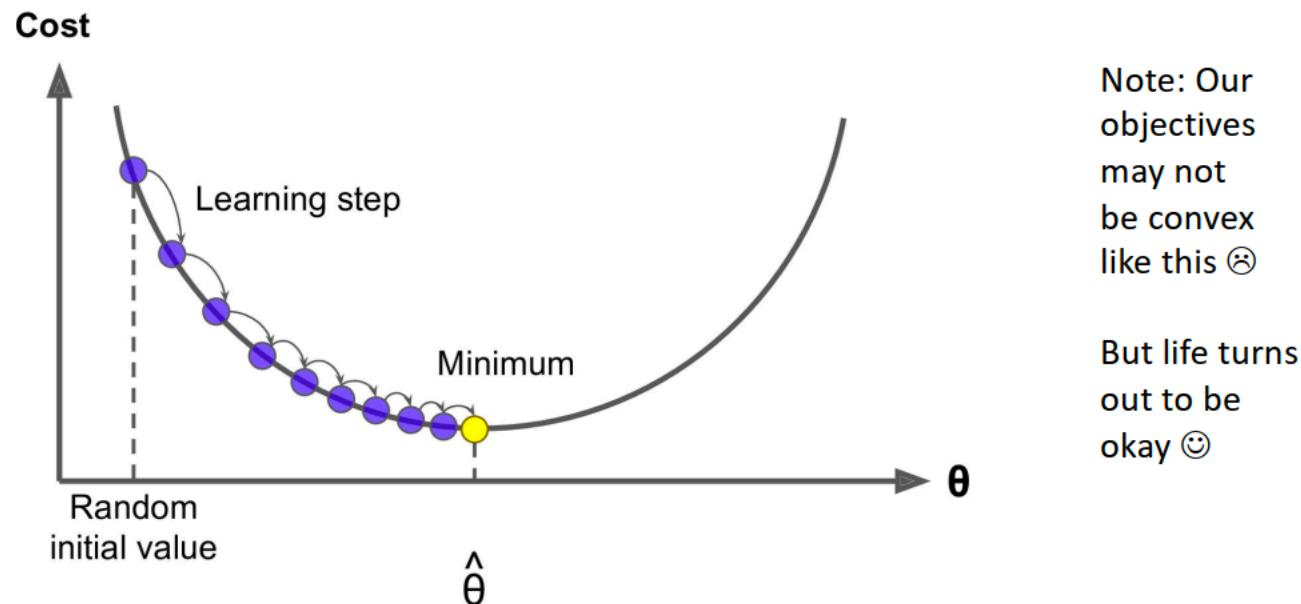
$$\begin{aligned}\frac{\partial}{\partial v_c} \log P(o|c) &= \frac{\partial}{\partial v_c} \log \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)} \\ &= \frac{\partial}{\partial v_c} \left( \log \exp(u_o^T v_c) - \log \sum_{w \in V} \exp(u_w^T v_c) \right) \\ &= \frac{\partial}{\partial v_c} \left( u_o^T v_c - \log \sum_{w \in V} \exp(u_w^T v_c) \right) \\ &= u_o - \frac{\sum_{w \in V} \exp(u_w^T v_c) u_w}{\sum_{w \in V} \exp(u_w^T v_c)} \\ &= u_o - \sum_{w \in V} \frac{\exp(u_w^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)} u_w \\ \text{Observed - expected} &= u_o - \boxed{\sum_{w \in V} P(w|c) u_w}\end{aligned}$$

把所有上下文向量用其概率加权求和

- 这一段是对中心向量  $v_c$  的梯度求法
- 输出向量  $u_o$  的求法类似

# Optimization: Gradient Descent

- 我们现在需要最小化
- 利用梯度下降法
- 基本思路：对于当前的参数 $\theta$ , 计算 $J(\theta)$  的梯度  $\nabla J(\theta)$ , 然后在负梯度的方向不断地小步迭代



# Gradient Descent

- 更新方程

$$\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J(\theta)$$

$\uparrow$   
 $\alpha = \text{step size or learning rate}$

- Python代码

```
while True:  
    theta_grad = evaluate_gradient(J,corpus,theta)  
    theta = theta - alpha * theta_grad
```

# Stochastic Gradient Descent

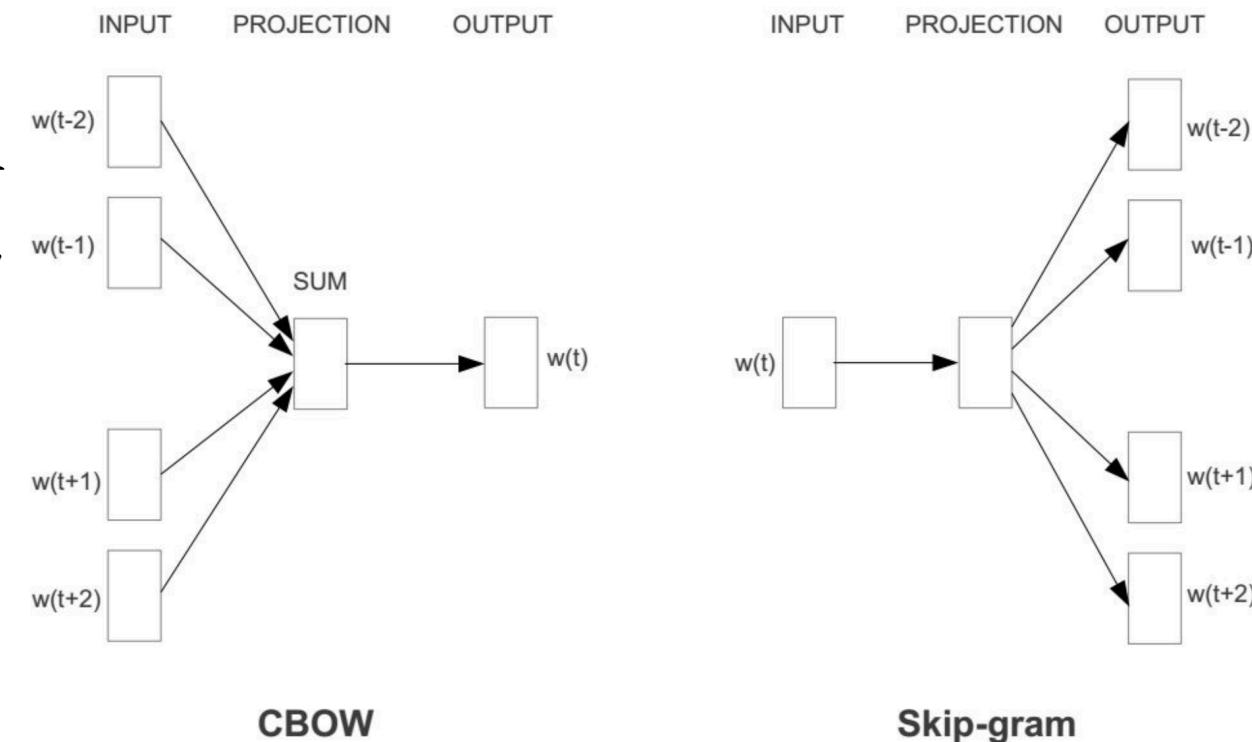
- 不足之处： $J(\theta)$  是语料库中所有窗口的函数（上亿）
  - $\nabla J(\theta)$  的计算需要花费大量资源
- 对于任意神经网络都不是好主意
- 解决方案： Stochastic gradient descent (SGD) 随机梯度下降
  - 不断的采样窗口，每采样一次做一次更新
- 代码示例

```
while True:  
    window = sample_window(corpus)  
    theta_grad = evaluate_gradient(J,window,theta)  
    theta = theta - alpha * theta_grad
```

# Word2vec algorithm family: More details

- 为什么每个单词一定要对应两个向量?
  - 最终两个向量的平均值代表词向量
  - 实际操作中, 每个词只对应一个向量。。。而且效果更好一些。
- 其他可能的模型结构
  - Skip-gram: 利用中心词预测上下文词  $p(o|c)$
  - Continuous Bag of Words(CBOW): 利用上下文词预测中心词  $p(c|o)$

我们刚刚讲过的是skip-gram model



# Training efficiency

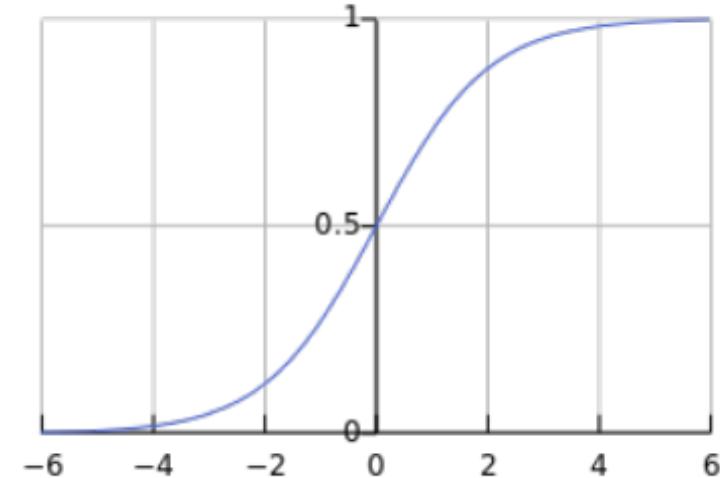
- 为了提高训练效率
  - Hierarchical Softmax (在算力足够的条件下不实用, 略)
  - Negative sampling
- Why? 归一化项的计算太费资源 
$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$
- 所以在Word2vec的标准实现中用的是负采样方法
- 主要思想: 使用一个true pair(中心词及其上下文窗口中的词)与几个noise pair(中心词与随机词搭配)形成的样本, 训练二元逻辑回归。

# The skip-gram model with negative sampling

- From paper: “Distributed Representations of Words and Phrases and their Compositionality”(Mikolov et al. 2013)
- 目标函数（最大化）  $J(\theta) = \frac{1}{T} \sum_{t=1}^T J_t(\theta)$

$$J_t(\theta) = \log \sigma(u_o^T v_c) + \sum_{i=1}^k \mathbb{E}_{j \sim P(w)} [\log \sigma(-u_j^T v_c)]$$

- Sigmoid函数  $\sigma(x) = \frac{1}{1+e^{-x}}$
- 我们要最大化2个词共现的概率，  
最小化与噪音词的共现概率



# The skip-gram model with negative sampling

- 实际操作中：

$$J_{\text{neg-sample}}(\mathbf{u}_o, \mathbf{v}_c, \mathbf{U}) = -\log \sigma(\mathbf{u}_o^T \mathbf{v}_c) - \sum_{k \in \{K \text{ sampled indices}\}} \log \sigma(-\mathbf{u}_k^T \mathbf{v}_c)$$

- 我们取 k 个负例采样
- 最大化出现在窗口中的“中心词”之外的词语出现的概率，而最小化其他没有出现在窗口中的随机词的概率
- 对词进行采样的时候使用概率分布： $P(w) = U(w)^{3/4} / Z$ ，其中 U() 代表一元的分布 (w 出现的概率)
- $3/4$  次幂减少了单词出现频率之间的差异，从而提高了低频词被抽中的概率

# Why not capture co-occurrence counts directly?

- 与其一遍一遍的遍历语料库优化模型，为什么不直接统计一下单词与附近单词的共现情况呢？
- 建立共现矩阵X：两种形式：window和全文档
  - Window：与word2vec类似，在每个单词周围都使用Window，捕捉一些语法和语义信息
  - Word-document：共现矩阵的基本假设是在同一篇文章中出现的单词更有可能相互关联。假设单词 $i$ 出现在文章 $j$ 中，则矩阵元素 $X_{ij}$ 加一，当我们处理完数据库中的所有文章后，就得到了矩阵 $X$ ，其大小为 $|V| \times M$ ，其中 $|V|$ 为词汇量，而 $M$ 为文章数。这一构建单词文章co-occurrence matrix的方法也是经典的Latent Semantic Analysis (LSA) 所采用的

# Example: Window based co-occurrence matrix

- Window length 1 (more common: 5–10)
- Symmetric (irrelevant whether left or right context)
- Example corpus:
  - I like deep learning
  - I like NLP
  - I enjoy flying

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

# Co-occurrence vectors

- 共现向量的问题
  - 用共现次数衡量单词的相似性，但是会随着词汇量的增加而增大矩阵的大小。
  - 需要很多空间来存储这一高维矩阵。
  - 后续的分类模型也会由于矩阵的稀疏性而存在稀疏性问题，使得效果不佳。
- 低维向量
  - 如何降维，获得低维稠密向量？
  - 25–1000维，类似Word2vec？

# Classic Method: Dimensionality Reduction on $X$

- SVD 分解 (Singular Value Decomposition)

$$\underbrace{\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}}_{X^k} = \underbrace{\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}}_U \underbrace{\begin{bmatrix} \bullet & & \\ & \ddots & \\ & & \bullet \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}}_{V^T}$$

Retain only  $k$  singular values, in order to generalize.

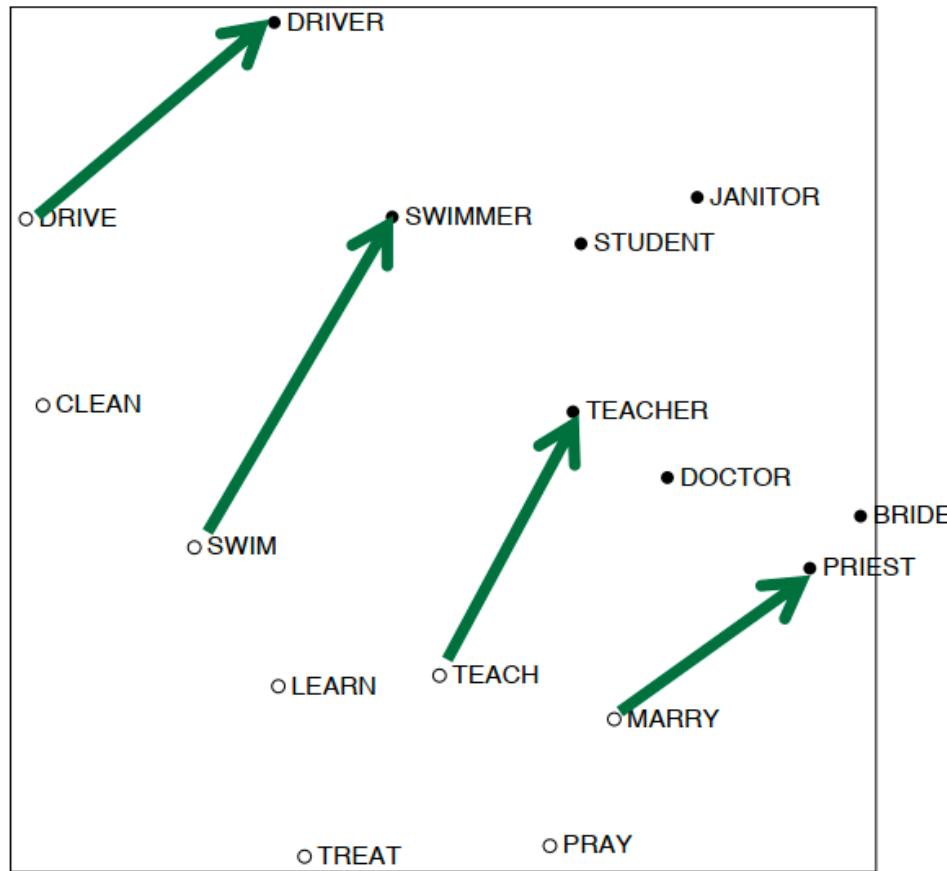
$\hat{X}$  is the best rank  $k$  approximation to  $X$ , in terms of least squares.

Classic linear algebra result. Expensive to compute for large matrices.

# Hacks to X (several used in Rohde et al. 2005 in COALS)

- 在原始计数的矩阵上运行SVD往往不会得到很好的结果！！
- 问题：一些功能性词汇（the, he, has）出现的过多，语法有太多的影响
  - 使用log进行缩放
  - 超过某个值（比如100），一律截断
  - 直接全部忽视功能性词汇
- 在基于window的计数中，提高距离近的单词的计数
- 使用Pearson相关系数代替直接计数，负值直接设为0

Interesting semantic patterns emerge in the scaled vectors



COALS model from

Rohde et al. ms., 2005. An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence

# Towards GloVe: Count based vs. direct prediction

**基于计数**：使用整个矩阵的全局统计数据来直接估计

- LSA, HAL (Lund & Burgess),
- COALS, Hellinger-PCA (Rohde et al, Lebret & Collobert)

**优点**：训练快速；统计学信息高效利用

**缺点**：主要用于捕捉单词相似性；对大量数据给予比例失调的重视

**基于预估模型**：定义概率分布并试图预测单词

- Skip-gram/CBOW (Mikolov et al)
- NNLM, HLBL, RNN (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton)

**优点**：提高其他任务的性能；能捕获除了单词相似性以外的复杂的模式

**缺点**：随语料库增大会增大规模；统计数据的低效使用（采样是对统计数据的低效使用）

# Encoding meaning components in vector differences [Pennington, Socher, and Manning, EMNLP 2014]

**Crucial insight:** Ratios of co-occurrence probabilities can encode meaning components

	$x = \text{solid}$	$x = \text{gas}$	$x = \text{water}$	$x = \text{random}$
$P(x \text{ice})$	large	small	large	small
$P(x \text{steam})$	small	large	large	small
$\frac{P(x \text{ice})}{P(x \text{steam})}$	large	small	~1	~1

# Encoding meaning components in vector differences [Pennington, Socher, and Manning, EMNLP 2014]

**Crucial insight:** Ratios of co-occurrence probabilities can encode meaning components

	$x = \text{solid}$	$x = \text{gas}$	$x = \text{water}$	$x = \text{fashion}$
$P(x \text{ice})$	$1.9 \times 10^{-4}$	$6.6 \times 10^{-5}$	$3.0 \times 10^{-3}$	$1.7 \times 10^{-5}$
$P(x \text{steam})$	$2.2 \times 10^{-5}$	$7.8 \times 10^{-4}$	$2.2 \times 10^{-3}$	$1.8 \times 10^{-5}$
$\frac{P(x \text{ice})}{P(x \text{steam})}$	8.9	$8.5 \times 10^{-2}$	1.36	0.96

# Encoding meaning components in vector differences

- 问题：
  - 我们如何在词向量空间中以线性含义成分的形式捕获共现概率的比值？
- 解决方案：
  - log-bilinear 模型： $w_i \cdot w_j = \log P(i|j)$
  - 向量之间作差的话：

$$w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$$

# Combining the best of both worlds

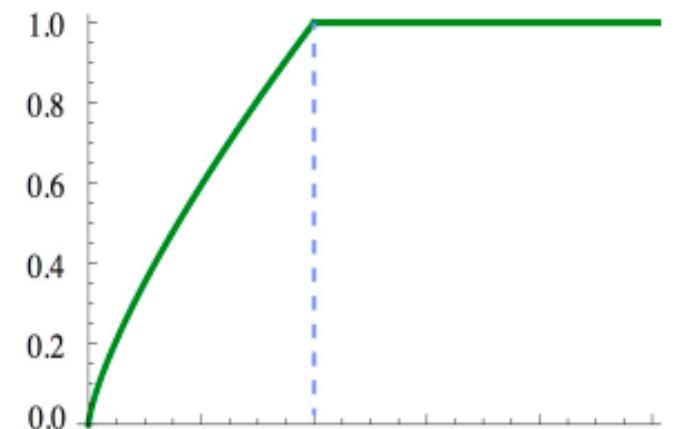
## GloVe [Pennington, Socher, and Manning, EMNLP 2014]

$$w_i \cdot w_j = \log P(i|j)$$

$$J = \sum_{i,j=1}^V f(X_{ij}) \left( w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2$$

- 训练快速
- 可以扩展到大型语料库
- 即使是小语料库和小向量，性能也很好

$f \sim$



# GloVe results

Nearest words to  
[frog](#):

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



[litoria](#)



[rana](#)



[leptodactylidae](#)



[eleutherodactylus](#)

# How to evaluate word vectors?

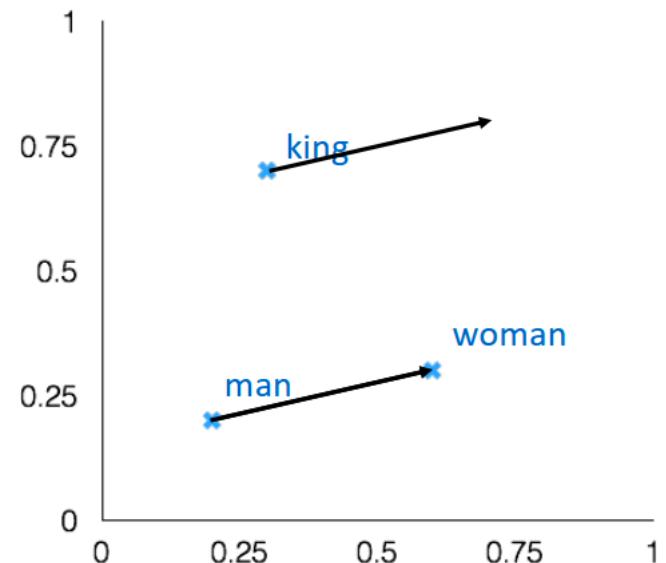
- 我们如何评估词向量呢，有内在和外在两种方式：
- 内在评估方式
  - 对特定/中间子任务进行评估
  - 计算速度快
  - 有助于理解这个系统
  - 不清楚是否真的有用，除非与实际任务建立了相关性
- 外部任务方式
  - 对真实任务（如下游NLP任务）的评估
  - 计算精确度可能需要很长时间
  - 不清楚子系统问题所在，是交互还是其他子系统问题
  - 如果用一个子系统替换另一个子系统可以提高精确度 → winning

# Intrinsic word vector evaluation

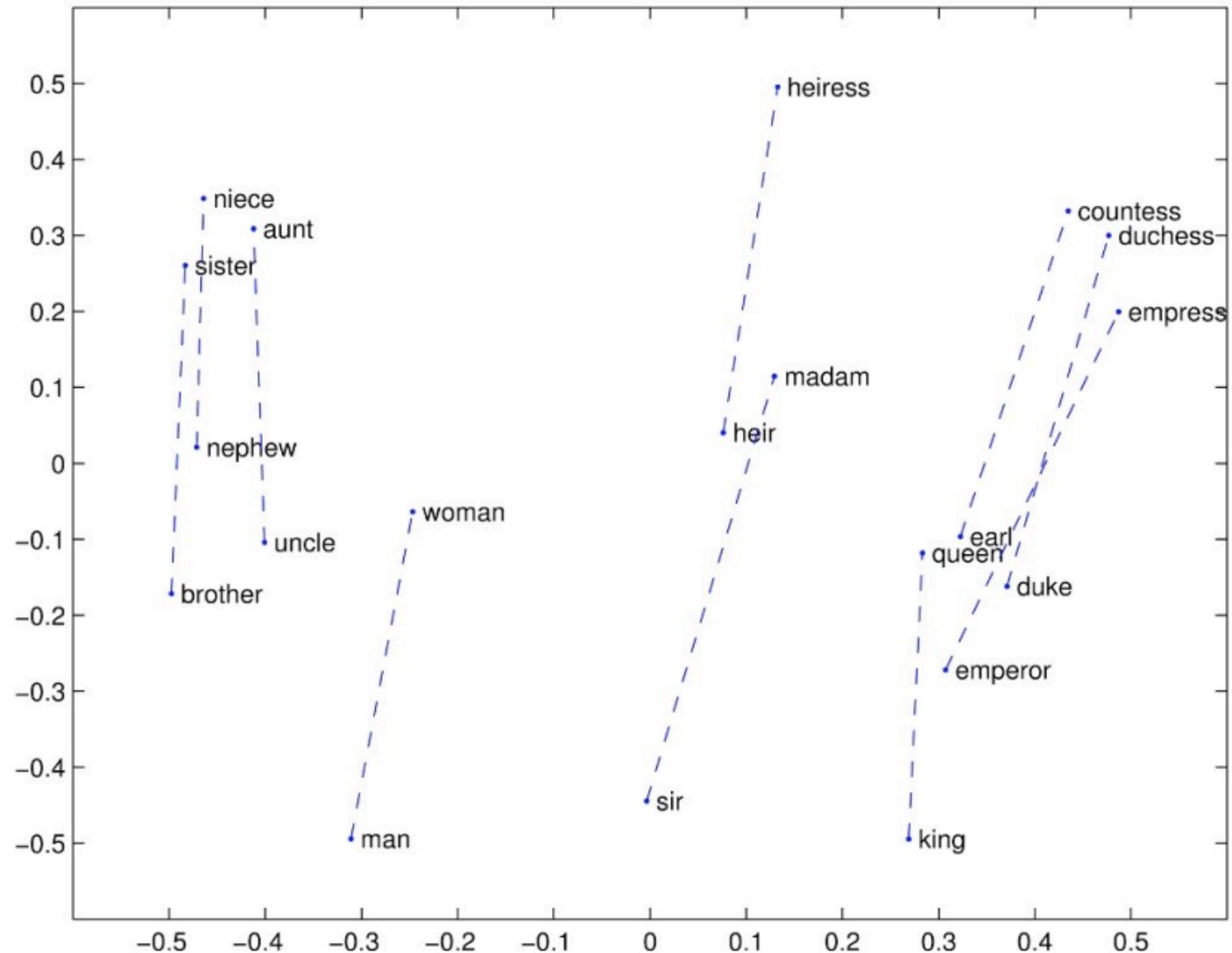
- 一种内在词向量评估方式是“词向量类比”：对于具备某种关系的词对a,b，在给定词c的情况下，找到具备类似关系的词d

$$\begin{array}{c} \boxed{a:b :: c:?} \\ \text{man:woman :: king: ?} \end{array} \longrightarrow \boxed{d = \arg \max_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}}$$

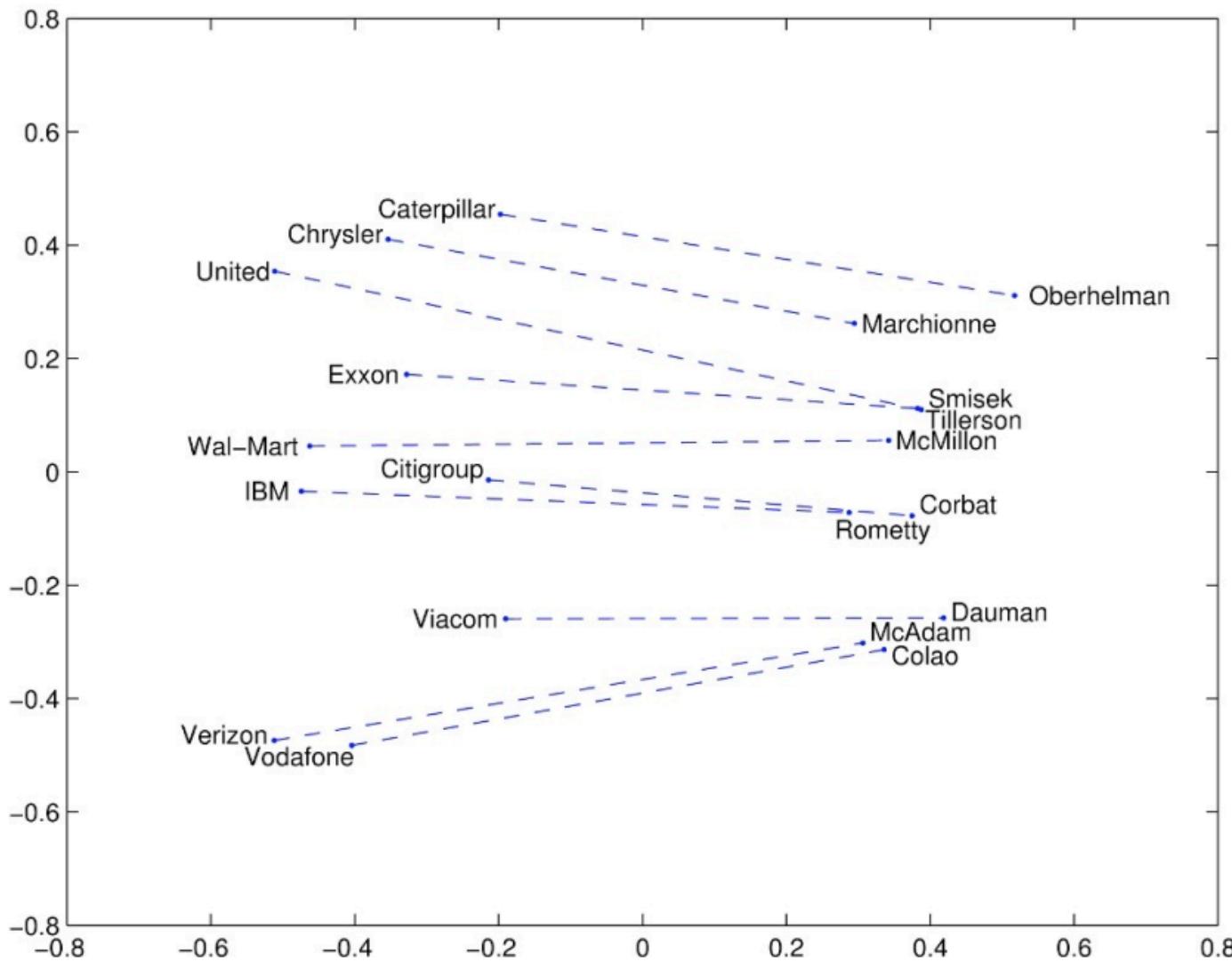
- 通过加法后的余弦距离是否能很好地捕捉到直观的语义和句法类比问题来评估单词向量
- 从搜索中丢弃输入的单词
- 问题：如果有信息但不是线性的怎么办？



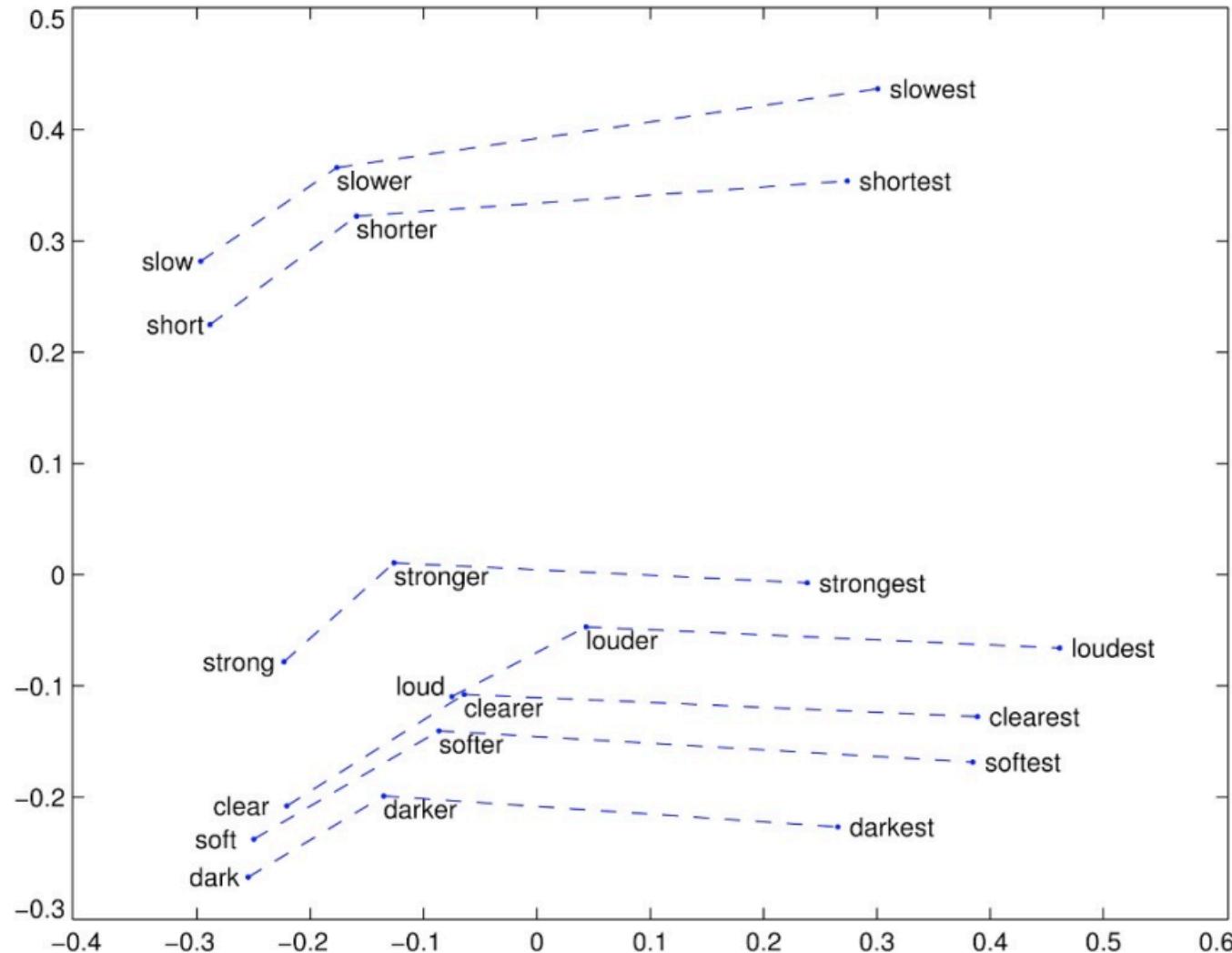
# Glove Visualizations



# Glove Visualizations: Company - CEO



# Glove Visualizations: Comparatives and Superlatives



# Analogy evaluation and hyperparameters

- 在一个包含各种关系种类的测试集上测试（包括语法和语义关系）

## Glove word vectors evaluation

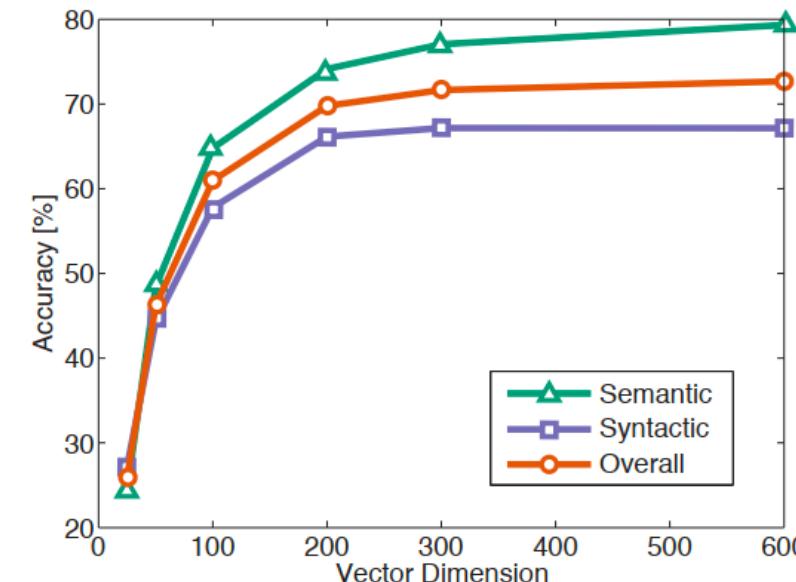
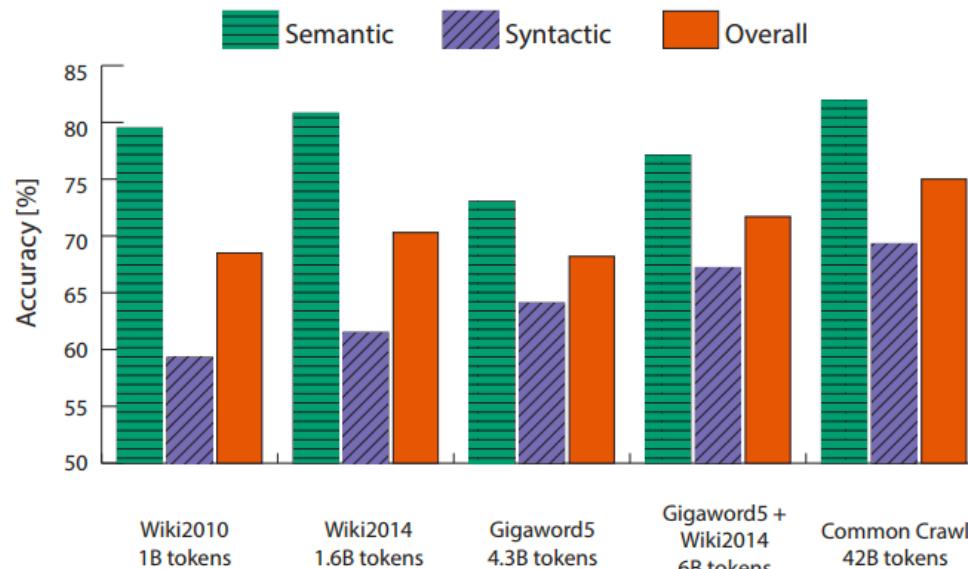
Model	Dim.	Size	Sem.	Syn.	Tot.
SVD	300	6B	6.3	8.1	7.3
SVD-S	300	6B	36.7	46.6	42.1
SVD-L	300	6B	56.6	63.0	60.1
CBOW <sup>†</sup>	300	6B	63.6	<u>67.4</u>	65.7
SG <sup>†</sup>	300	6B	73.0	66.0	69.1
GloVe	300	6B	<u>77.4</u>	67.0	<u>71.7</u>

# Analogy evaluation and hyperparameters

- 其他的一些实验发现

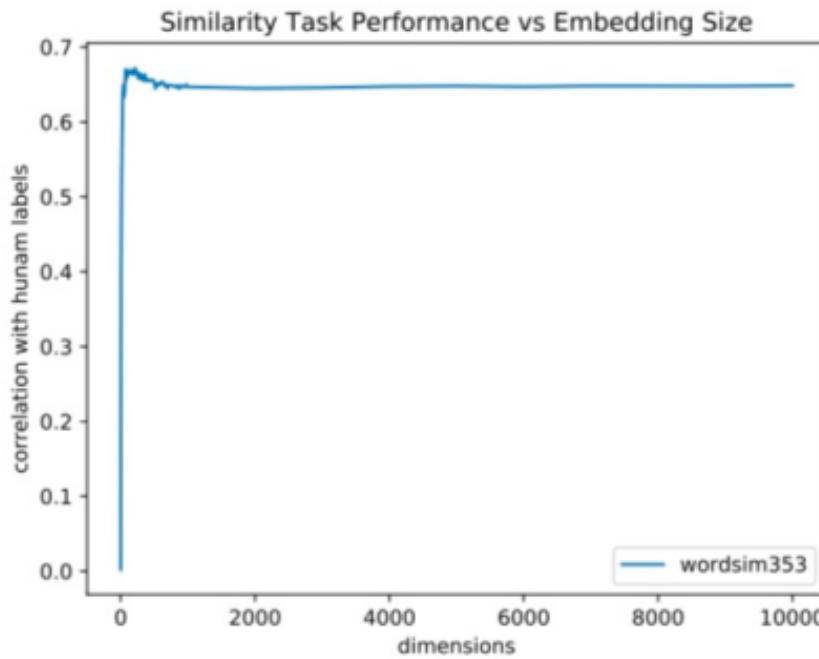
- More data helps
- Wikipedia is better than news text!

- Dimensionality
- Good dimension is  $\sim 300$



# On the Dimensionality of Word Embedding [Zi Yin and Yuanyuan Shen, NeurIPS 2018]

- <https://papers.nips.cc/paper/7368-on-the-dimensionality-of-word-embedding.pdf>



(b) WordSim353 Test

- 利用矩阵摄动理论，揭示了词嵌入维数选择的基本的偏差与方法的权衡
- 补充说明：当持续增大词向量维度的时候，词向量的效果不会一直变差并且会保持平稳

# Another intrinsic word vector evaluation

- 使用 cosine similarity 衡量词向量之间的相似程度并与人类评估比照
- Example dataset: WordSim353  
<http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/>

Word 1	Word 2	Human (mean)
tiger	cat	7.35
tiger	tiger	10
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

# Extrinsic word vector evaluation

- NER (Named Entity Recognition) : 识别实体中的人名，地名，机构名：Apple locates in Cupertino

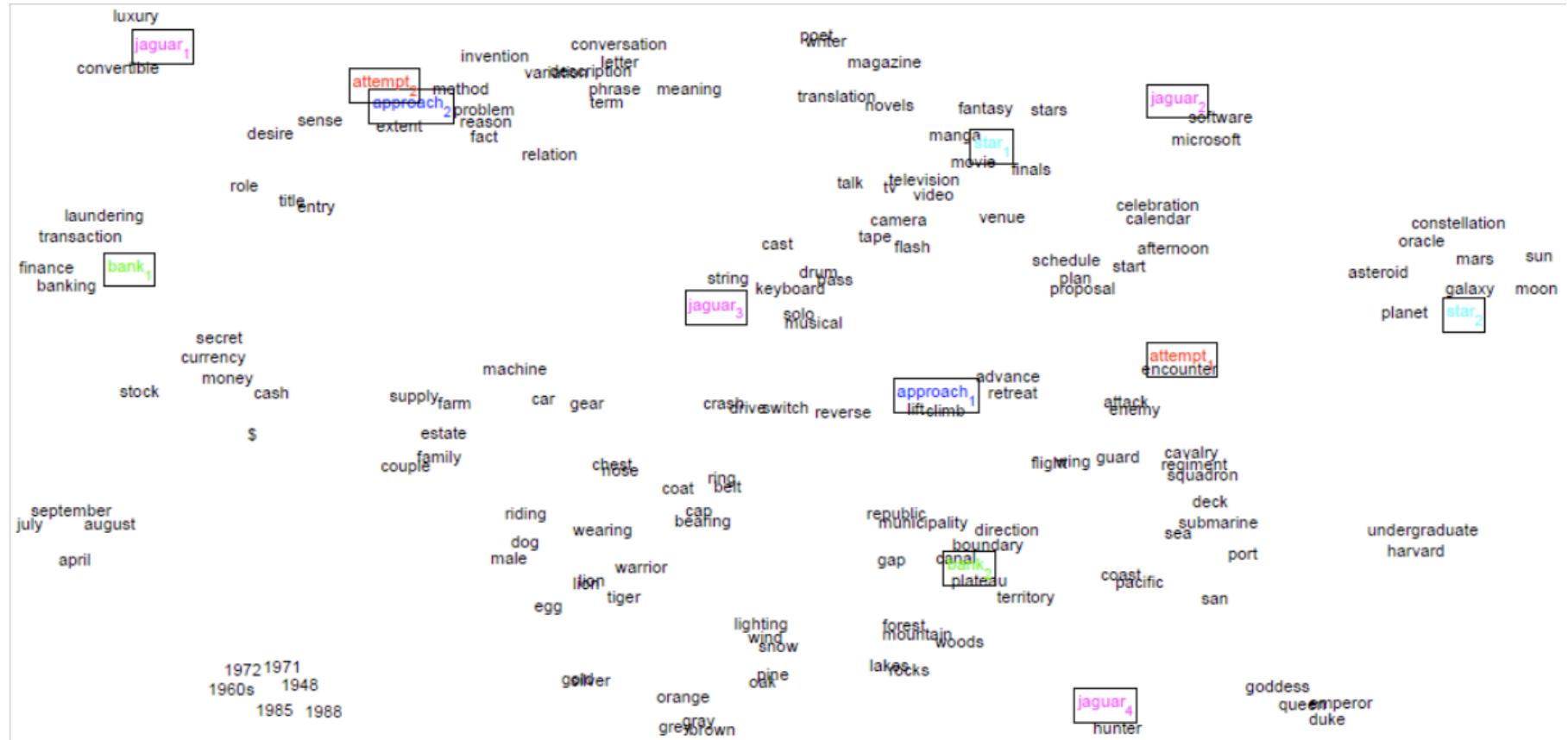
Model	Dev	Test	ACE	MUC7
Discrete	91.0	85.4	77.4	73.4
SVD	90.8	85.7	77.3	73.7
SVD-S	91.0	85.5	77.6	74.3
SVD-L	90.5	84.8	73.6	71.5
HPCA	92.6	<b>88.7</b>	81.7	80.7
HSMN	90.5	85.7	78.7	74.7
CW	92.2	87.4	81.7	80.2
CBOW	93.1	88.2	82.2	81.1
GloVe	<b>93.2</b>	88.3	<b>82.9</b>	<b>82.2</b>

# Word senses and word sense ambiguity

- 大多数单词都是多义的
  - 特别是常见单词
  - 特别是存在已久的单词
- 例如： pike
  - A sharp point or staff 矛
  - A type of elongated fish 梭子鱼
  - A railroad line or system
  - A type of road
  - The future (coming down the pike)
  - A type of body position (as in diving)
  - To kill or pierce with a pike
  - To make one's way (pike along)
  - In Australian English, pike means to pull out from doing something: I reckon he could have climbed that cliff, but he piked!
- 那么，词向量是总体捕捉了所有这些信息，还是杂乱在一起了呢？

# Improving Word Representations Via Global Context And Multiple Word Prototypes (Huang et al. 2012)

- 将常用词的所有上下文进行聚类，通过该词得到一些清晰的簇，从而将这个常用词分解为多个单词，例如bank-<sub>1</sub>, bank-<sub>2</sub>等



# Linear Algebraic Structure of Word Senses, with Applications to Polysemy (Arora, ..., Ma, ..., TACL 2018)

- 此方法中，单词在标准单词嵌入(如word2vec)中的不同含义以线性叠加(加权和)的形式存在

$$v_{\text{pike}} = \alpha_1 v_{\text{pike}_1} + \alpha_2 v_{\text{pike}_2} + \alpha_3 v_{\text{pike}_3} \quad \alpha_1 = \frac{f_1}{f_1 + f_2 + f_3}$$

- f是出现频率
- 结果
  - 只是加权平均值就已经可以获得很好的效果
  - 由于里面用到了稀疏编码的方法，实际上可以将不同词义分离出来  
(前提是它们相对比较常见)

tie				
trousers	season	scoreline	wires	operatic
blouse	teams	goalless	cables	soprano
waistcoat	winning	equaliser	wiring	mezzo
skirt	league	clinching	electrical	contralto
sleeved	finished	scoreless	wire	baritone
pants	championship	replay	cable	coloratura

Thank you