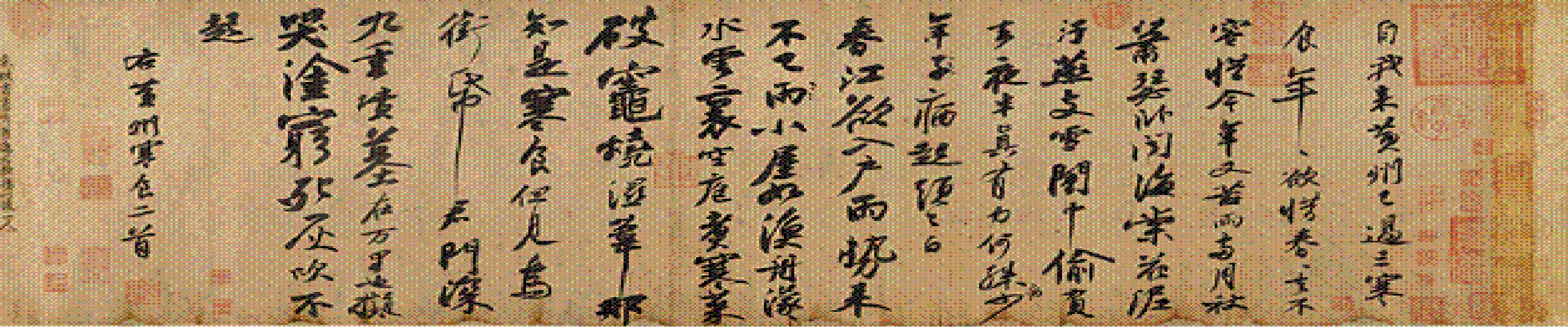


信息检索
Information Retrieval

教师：孙茂松

Tel:62781286

Email:sms@tsinghua.edu.cn

TA：胡锦毅

Email:hu-jy21@mails.tsinghua.edu.cn

郑重声明

● 此课件仅供选修清华大学计算机系本科生课《信息检索》的学生个人学习使用，所以只允许学生将其下载、存贮在自己的电脑中。未经孙茂松本人同意，任何人不得以任何方式扩散之（包括放到9#服务器上）。否则，由此可能引起的一切涉及知识产权的法律责任，概由该人负责。

● 此课件仅限孙茂松本人讲课使用。除孙茂松本人外，凡授课过程中，PTT文件显示此《郑重声明》之情形，即为侵权使用。

第三章 文本分析及自动标引
(Part 4)

3.5 Thesaurus及term自动关联(Word2Vec)

Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols:
hotel, conference, motel – a localist representation

Means one 1, the rest 0s

Such symbols for words can be represented by one-hot vectors:

motel = [0 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 0]

Vector dimension = number of words in vocabulary (e.g., 500,000)

3.5 Thesaurus及term自动关联(Word2Vec)

Problem with words as discrete symbols

Example: in web search, if user searches for “Seattle motel”, we would like to match documents containing “Seattle hotel”

But:

motel = [0 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 0]

These two vectors are orthogonal

There is no natural notion of **similarity** for one-hot vectors!

Solution:

- Could try to rely on WordNet’s list of synonyms to get similarity?
  - But it is well-known to fail badly: incompleteness, etc.
- **Instead: learn to encode similarity in the vectors themselves**

3.5 Thesaurus及term自动关联(Word2Vec)

Representing words by their context

- Distributional semantics: **A word's meaning is given by the words that frequently appear close-by**
  - *"You shall know a word by the company it keeps"* (J. R. Firth 1957: 11)
  - One of the most successful ideas of modern statistical NLP!
- When a word *w* appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- Use the many contexts of *w* to build up a representation of *w*

…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***

3.5 Thesaurus及term自动关联(Word2Vec)

Word vectors

We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts

$$\textit{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Note: word vectors are also called word embeddings or (neural) word representations

They are a distributed representation

3.5 Thesaurus及term自动关联(Word2Vec)

Word meaning as a neural word vector – visualization

$$expect = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

3.5 Thesaurus及term自动关联(Word2Vec)

Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

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

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

Distributed Representations of Words and Phrases and their Compositionality

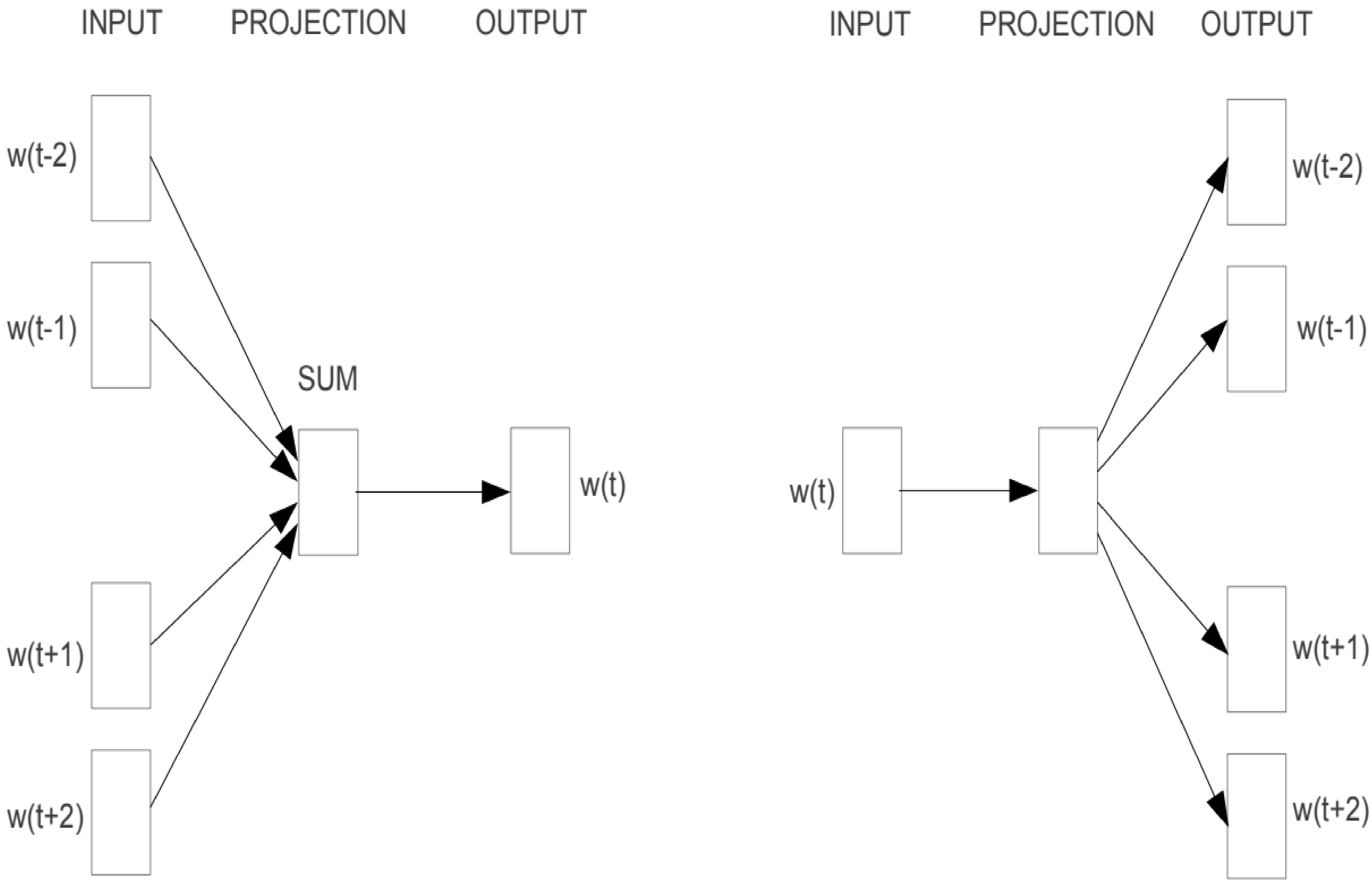
Tomas Mikolov
Google Inc.
Mountain View
mikolov@google.com

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

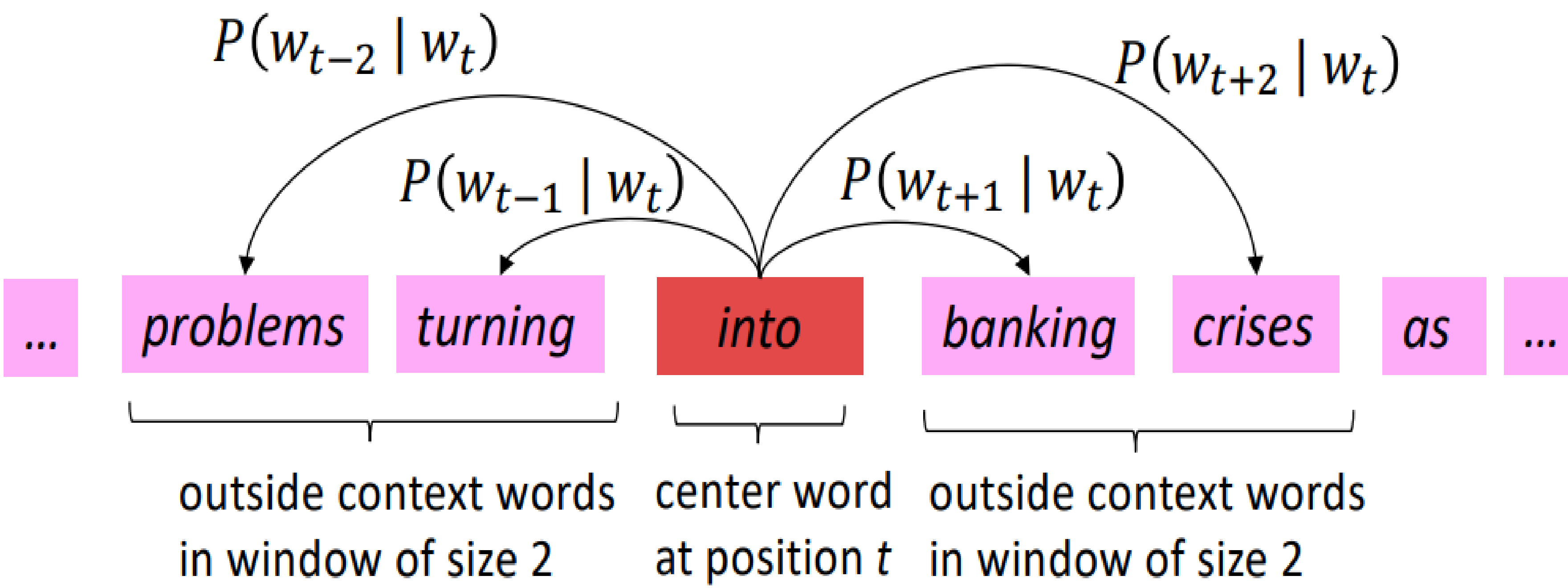


CBOW

Skip-gram

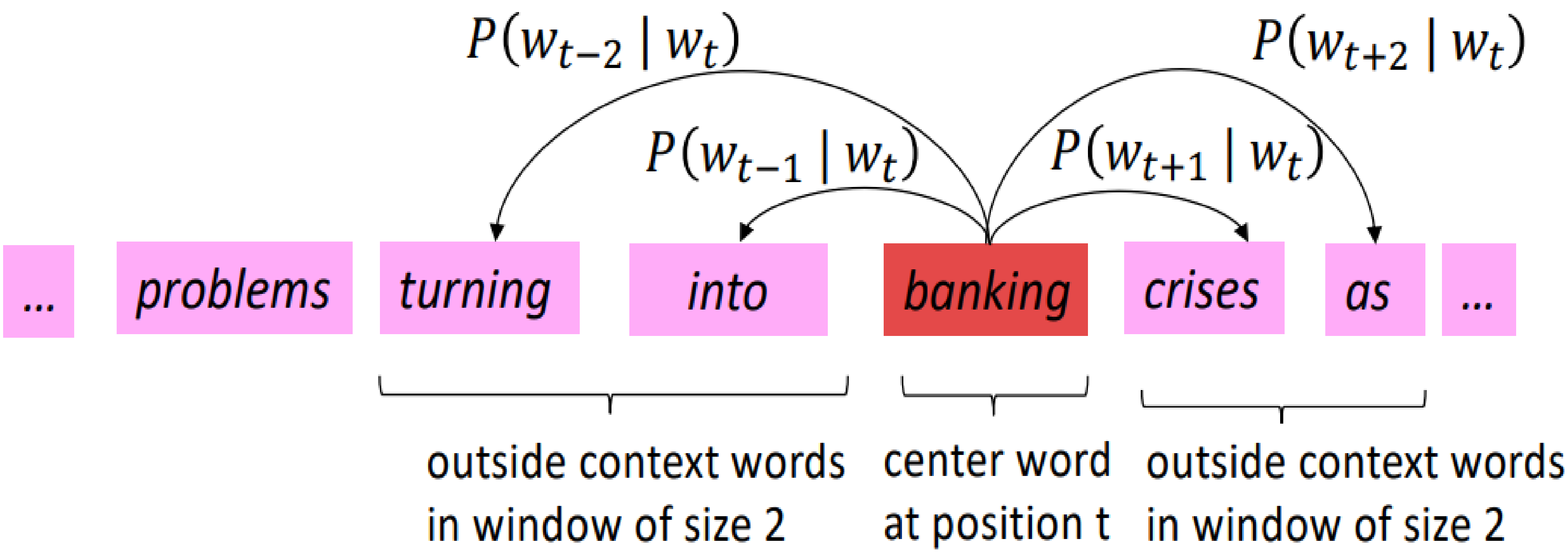
3.5 Thesaurus及term自动关联(Word2Vec)

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



3.5 Thesaurus及term自动关联(Word2Vec)

Example windows and process for computing $P\left(w_{t+j} \mid w_t\right)$



Word2vec: objective function

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size *m*, given center word w_j. Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} P\left(w_{t+j} \mid w_t; \theta\right)$$

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

Minimizing objective function ⟺ Maximizing predictive accuracy

Word2vec: objective function

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m\\ j\ne 0}} \log P\left(w_{t+j} \mid w_t;\theta\right)$$

- **Question:** How to calculate $P\left(w_{t+j} \mid w_t;\theta\right)$?

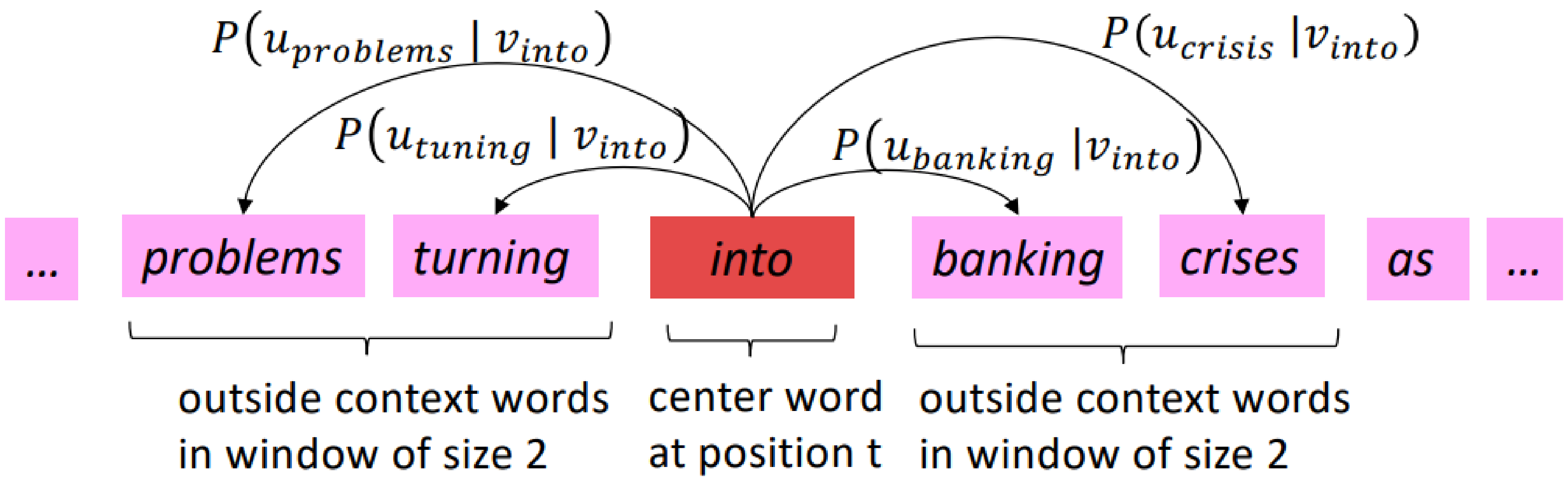
- **Answer:** We will *use two* vectors per word *w*:
    - v_w when *w* is a center word
    - u_w when *w* is a context word

- Then for a center word *c* and a context word *o*:

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

3.5 Thesaurus及term自动关联(Word2Vec)

- Example windows and process for computing $P(w_{t+j} \mid w_t)$
- $P(u_{problems} \mid v_{into})$ short for $\mathrm{P}(problems \mid into \,; u_{problems}, v_{into}, \theta)$



3.5 Thesaurus及term自动关联(Word2Vec)

Word2vec: prediction function

② Exponentiation makes anything positive

① Dot product compares similarity of *o* and *c*.

$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$

Larger dot product = larger probability

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

③ Normalize over entire vocabulary to give probability distribution

- This is an example of the **softmax function** $\mathbb{R}^n \to (0,1)^n$

Open region

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

- The softmax function maps arbitrary values x_i to a probability distribution p_i
  - "max" because amplifies probability of largest x_i
  - "soft" because still assigns some probability to smaller x_i
  - Frequently used in Deep Learning

But sort of a weird name because it returns a distribution!

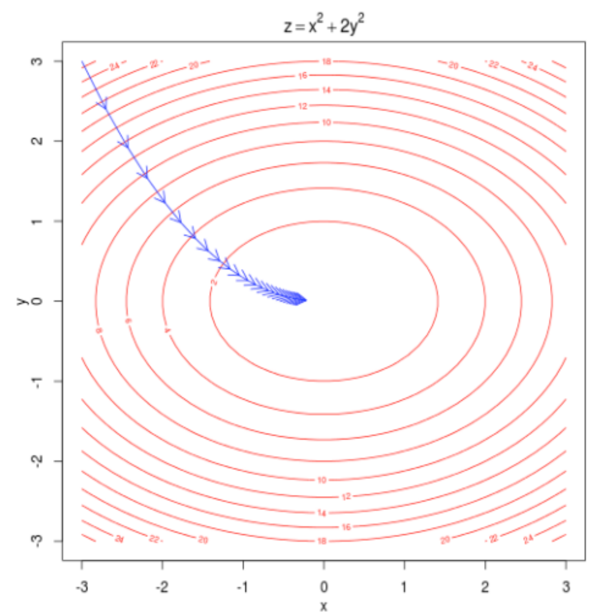
3.5 Thesaurus及term自动关联(Word2Vec)

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

To train a model, we gradually adjust parameters to minimize a loss

- Recall: θ represents **all** the model parameters, in one long vector
- In our case, with *d*-dimensional vectors and *V*-many words, we have:
- Remember: every word has two vectors

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



- We optimize these parameters by walking down the gradient (see right figure)
- We compute all vector gradients!

Chain Rule

- Chain rule! If $y = f(u)$ and $u = g(x)$, i.e., $y = f(g(x))$, then:

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx} = \frac{df(u)}{du}\frac{dg(x)}{dx}$$

- Simple example: $\frac{dy}{dx} = \frac{d}{dx}5(x^3+7)^4$

$$y = f(u) = 5u^4 \qquad u = g(x) = x^3 + 7$$

$$\frac{dy}{du} = 20u^3 \qquad \frac{du}{dx} = 3x^2$$

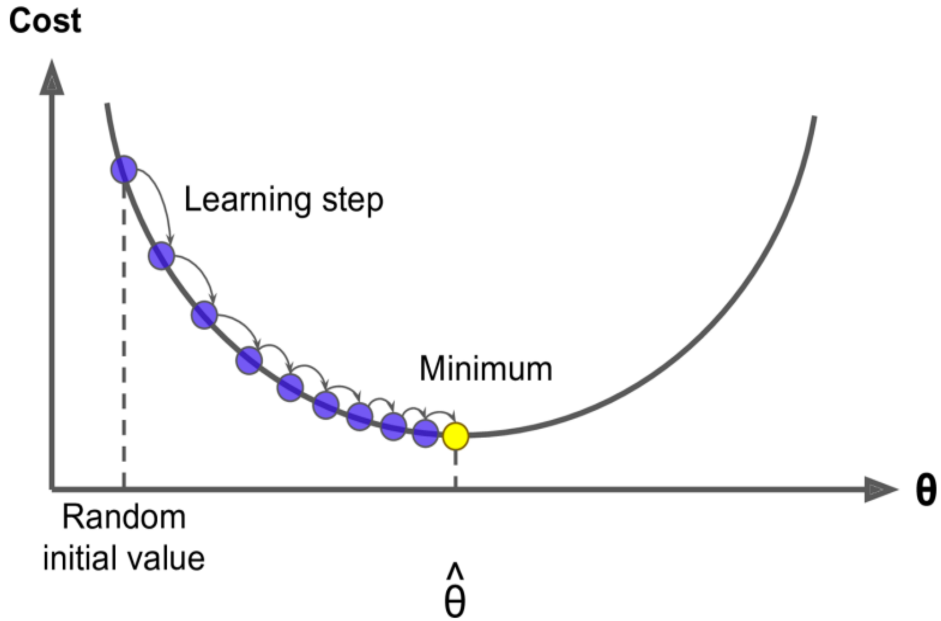
$$\frac{dy}{dx} = 20(x^3+7)^3 . 3x^2$$

Useful basic fact: $\frac{\partial \mathbf{x}^T\mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^T\mathbf{x}}{\partial \mathbf{x}} = \mathbf{a}$

3.5 Thesaurus及term自动关联(Word2Vec)

Gradient Descent

- We have a cost function $J(\theta)$ we want to minimize
- **Gradient Descent** is an algorithm to minimize $J(\theta)$
- **Idea:** for current value of θ, calculate gradient of $J(\theta)$, then take small step in direction of negative gradient. Repeat.



Note: Our objectives may not be convex like this ☹

But life turns out to be okay ☺

3.5 Thesaurus及term自动关联(Word2Vec)

Gradient Descent

- Update equation (in matrix notation):

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

α = *step size* or *learning rate*

- Update equation (for single parameter):

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$$

3.5 Thesaurus及term自动关联(Word2Vec)

- Iteratively take gradients at each such window for SGD
- But in each window, we only have at most 2*m* + 1 words, so $\nabla_\theta J_t(\theta)$ is very sparse!

$$\nabla_\theta J_t(\theta) = \begin{bmatrix} 0 \\ \vdots \\ \nabla_{v_{like}} \\ \vdots \\ 0 \\ \nabla_{u_I} \\ \vdots \\ \nabla_{u_{learning}} \\ \vdots \end{bmatrix} \in \mathbb{R}^{2dV}$$

$$J(\theta) = -\frac{1}{T}\log L(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m \\ j\ne 0}} \log P\left(w_{t+j} \mid w_t;\theta\right)$$

则对其中两个词o和c, 令：

$$\mathrm{P}(\mathrm{O}=\mathrm{o}|\mathrm{C}=\mathrm{c}) = \frac{\exp\left(u_o^T v_c\right)}{\sum_{x=1}^{V}\exp\left(u_x^T v_c\right)}$$

现计算其导数，即：

$$\frac{\partial}{\partial v_c}\log\frac{\exp\left(u_o^T v_c\right)}{\sum_{x=1}^{V}\exp\left(u_x^T v_c\right)}$$

注意：向量右上角*T*表示转置，*J*中*T*表示语料库规模。下同

可分解为两部分。其中前半部分：

$$\frac{\partial}{\partial v_c}(\log(\exp(u_o^T v_c))) = \frac{\partial}{\partial v_c}(u_o^T v_c) = u_o$$

其中后半部分：

$$-\frac{1}{\sum_{x=1}^{V} \exp(u_x^T v_c)} \sum_{x=1}^{V} \frac{\partial}{\partial v_c} \exp(u_x^T v_c)$$

$$= -\sum_{x=1}^{V} \frac{\exp(u_x^T v_c)}{\sum_{x=1}^{V} \exp(u_x^T v_c)} * u_x$$

$$= -\sum_{x=1}^{V} P(x|c) * u_x$$

于是有：

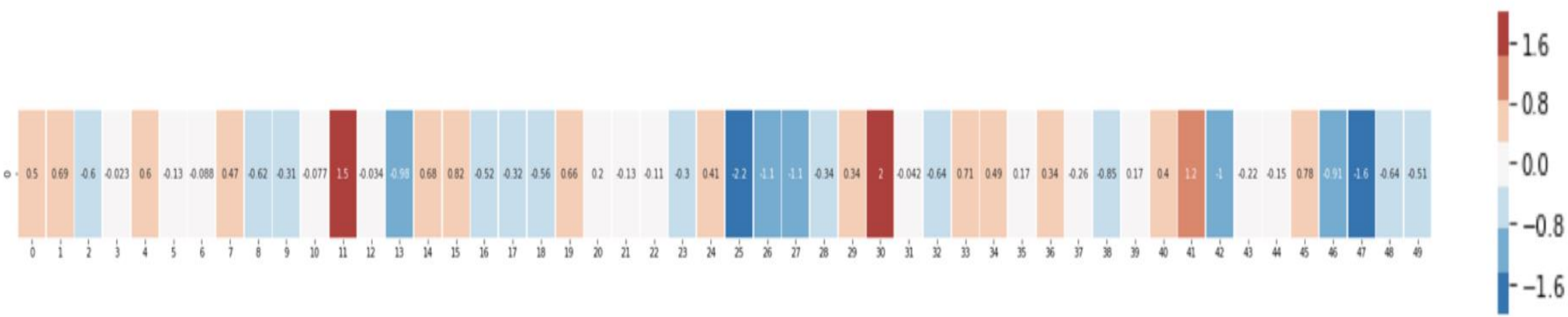
$$\frac{\partial}{\partial v_c} J_t(\theta) = -\sum_{o\text{在}c\text{窗口内}} \left(u_o - \sum_{x=1}^{V} P(x|c) * u_x \right)$$

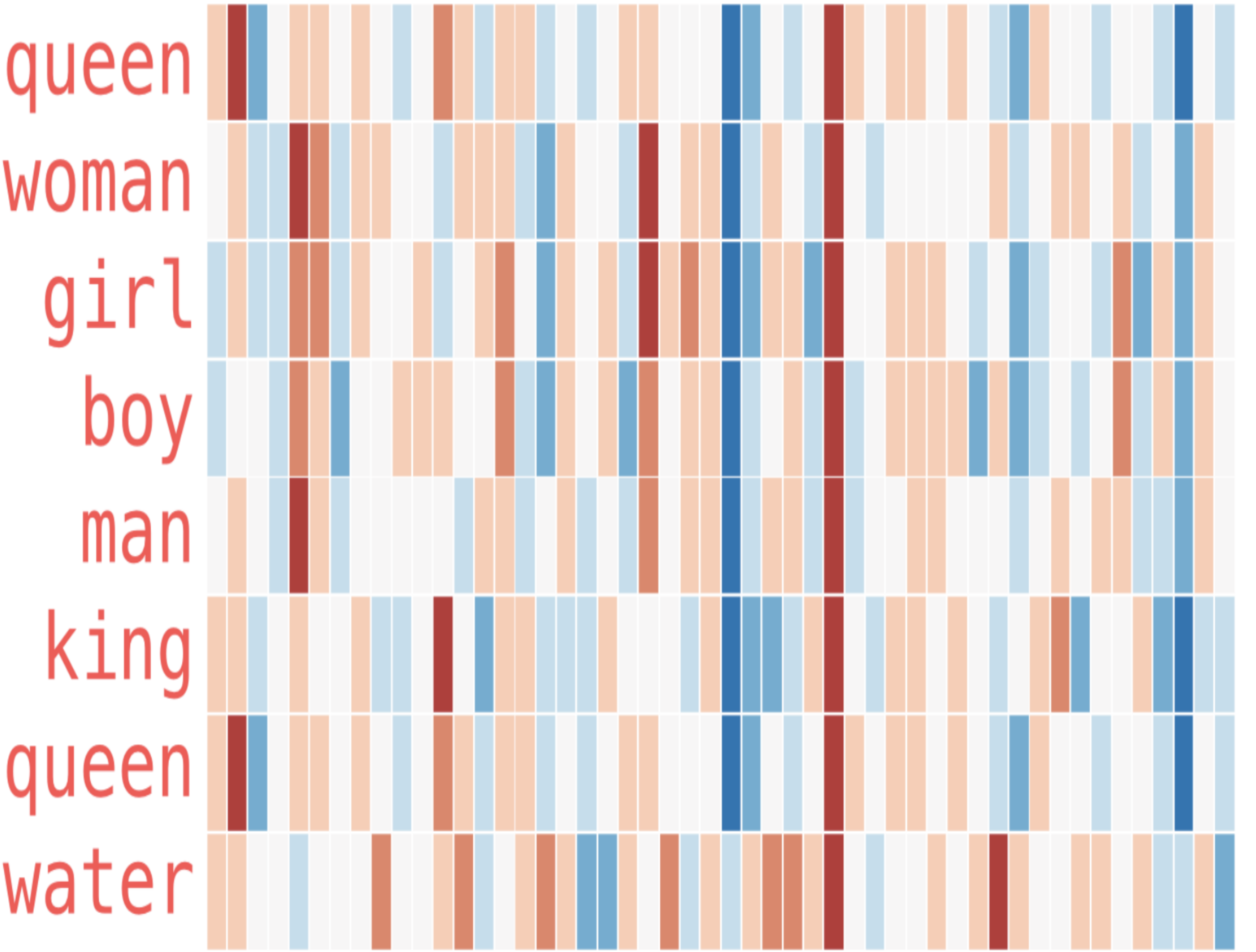
类似地，可得：

$$\frac{\partial}{\partial u_o} J_t(\theta) = -\sum_{o\text{在}c\text{窗口内}} (v_c - P(x|c) * v_c)$$

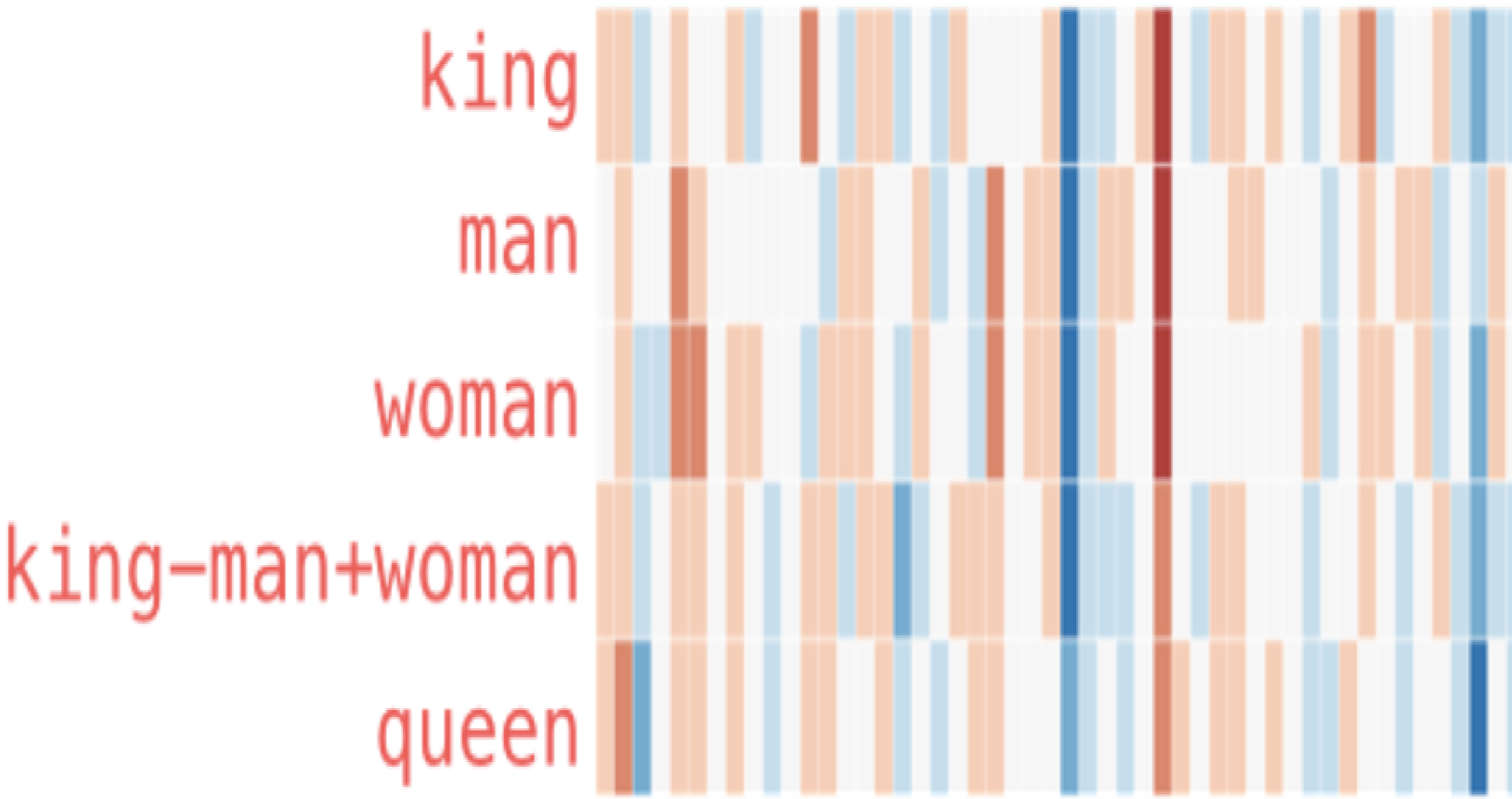
This is a word embedding for the word “king” (GloVe vector trained on Wikipedia):

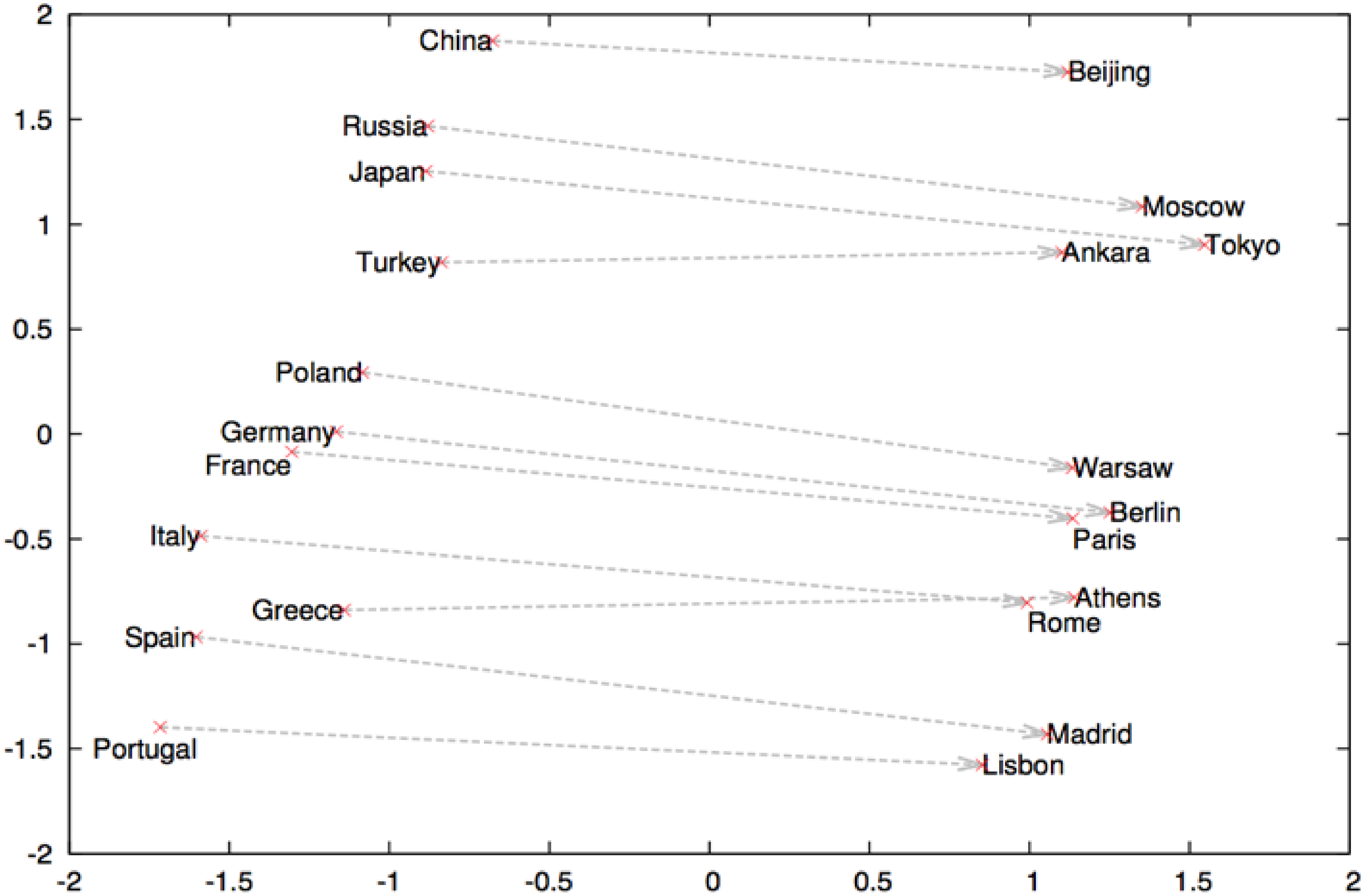
```
[ 0.50451 , 0.68607 , -0.59517 , -0.022801, 0.60046 , -0.13498 , -0.08813 , 0.47377 , -0.61798 , -0.31012 ,
-0.076666, 1.493 , -0.034189, -0.98173 , 0.68229 , 0.81722 , -0.51874 , -0.31503 , -0.55809 , 0.66421 , 0.1961
, -0.13495 , -0.11476 , -0.30344 , 0.41177 , -2.223 , -1.0756 , -1.0783 , -0.34354 , 0.33505 , 1.9927 ,
-0.04234 , -0.64319 , 0.71125 , 0.49159 , 0.16754 , 0.34344 , -0.25663 , -0.8523 , 0.1661 , 0.40102 , 1.1685 ,
-1.0137 , -0.21585 , -0.15155 , 0.78321 , -0.91241 , -1.6106 , -0.64426 , -0.51042 ]
```




queen
woman
girl
boy
man
king
queen
water

king - man + woman ~= queen





$$W(\text{“China“}) - W(\text{“Beijing”}) \simeq W(\text{“Japan“}) - W(\text{“Tokyo"})$$

Intrinsic word vector evaluation

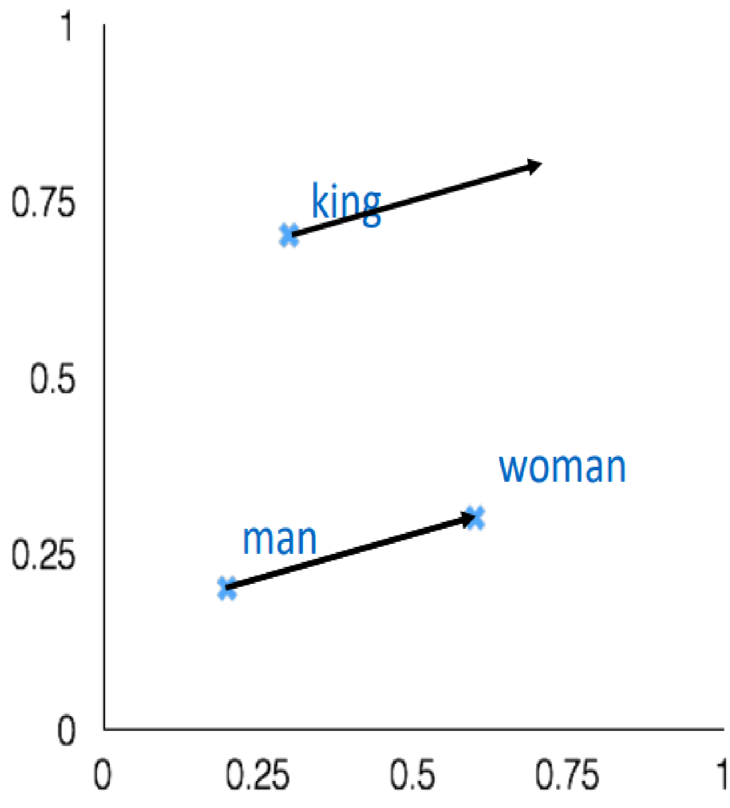
- Word Vector Analogies

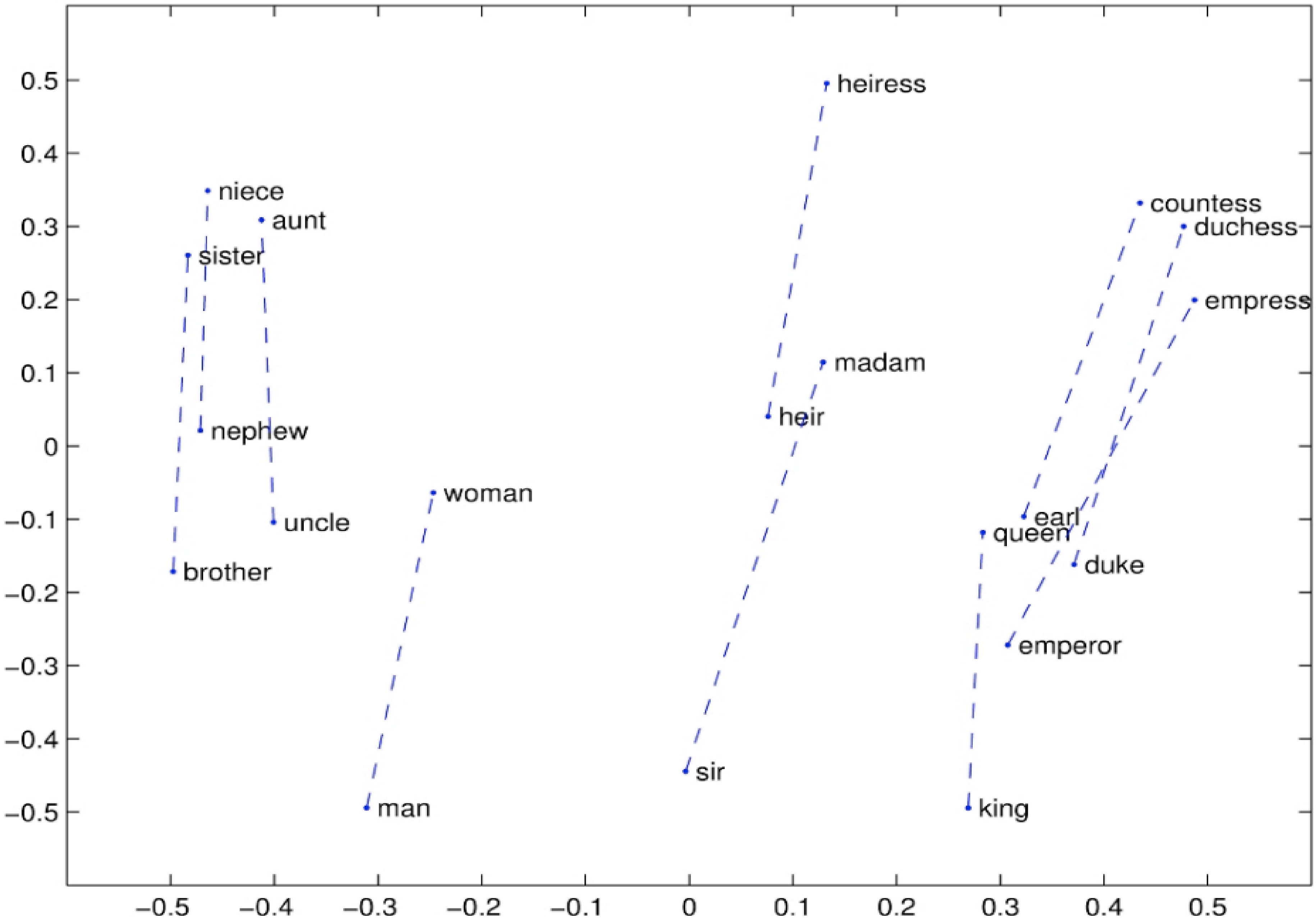
a:b :: c:?

man:woman :: king:?

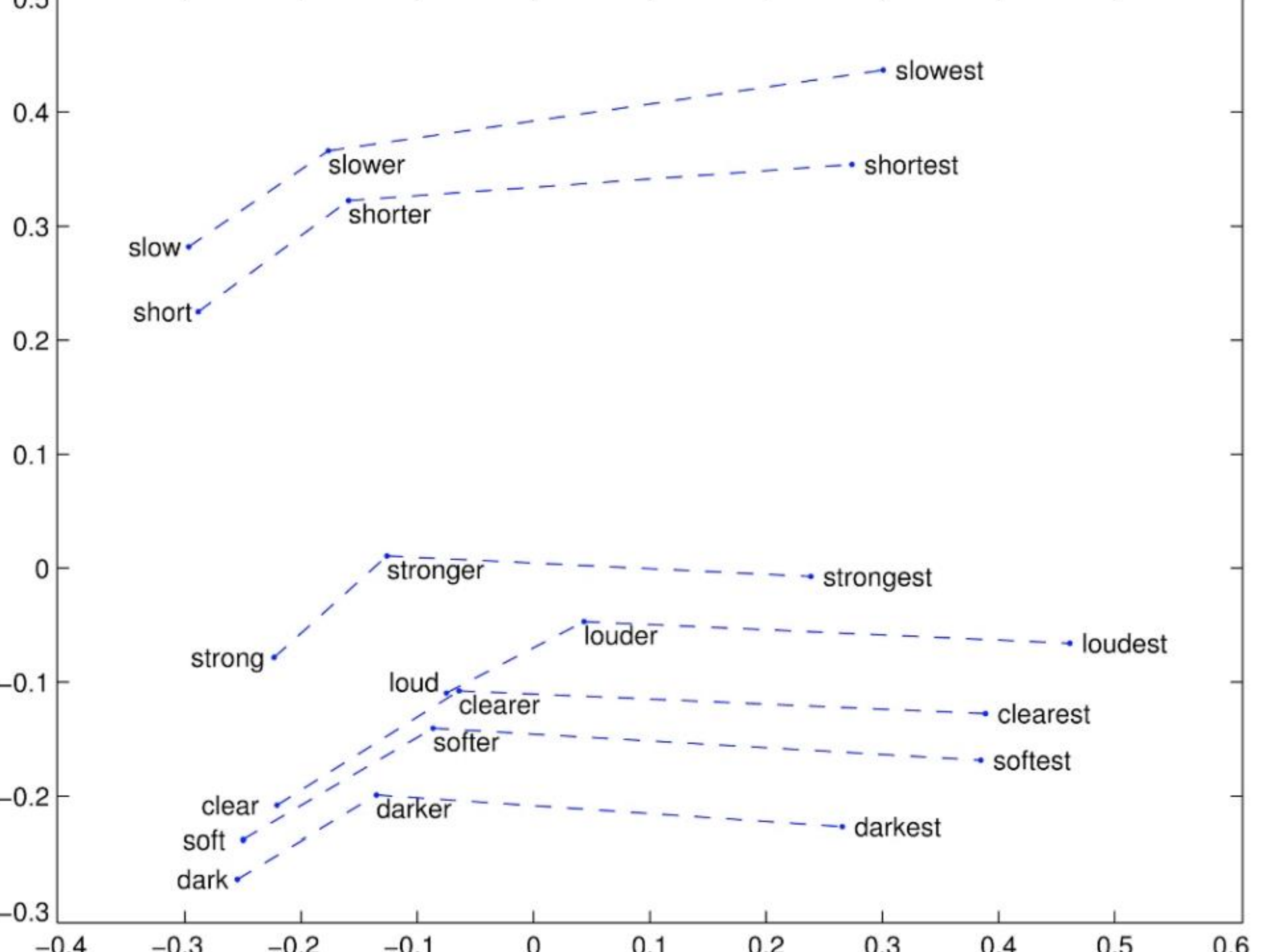
$$d = \arg\max_i \frac{(x_b - x_a + x_c)^T x_i}{||x_b - x_a + x_c||}$$

- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions
- Discarding the input words from the search!
- Problem: What if the information is there but not linear?

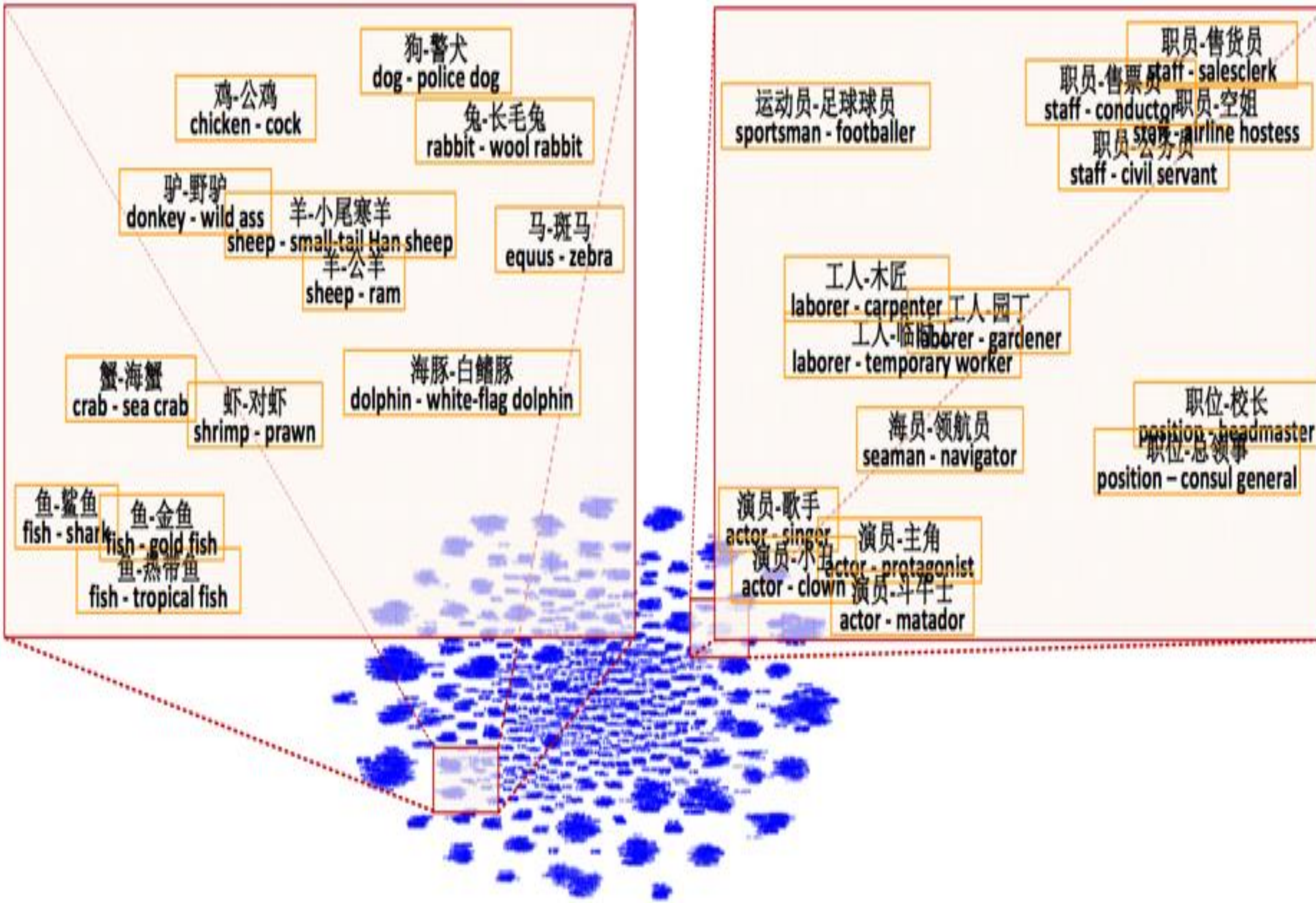




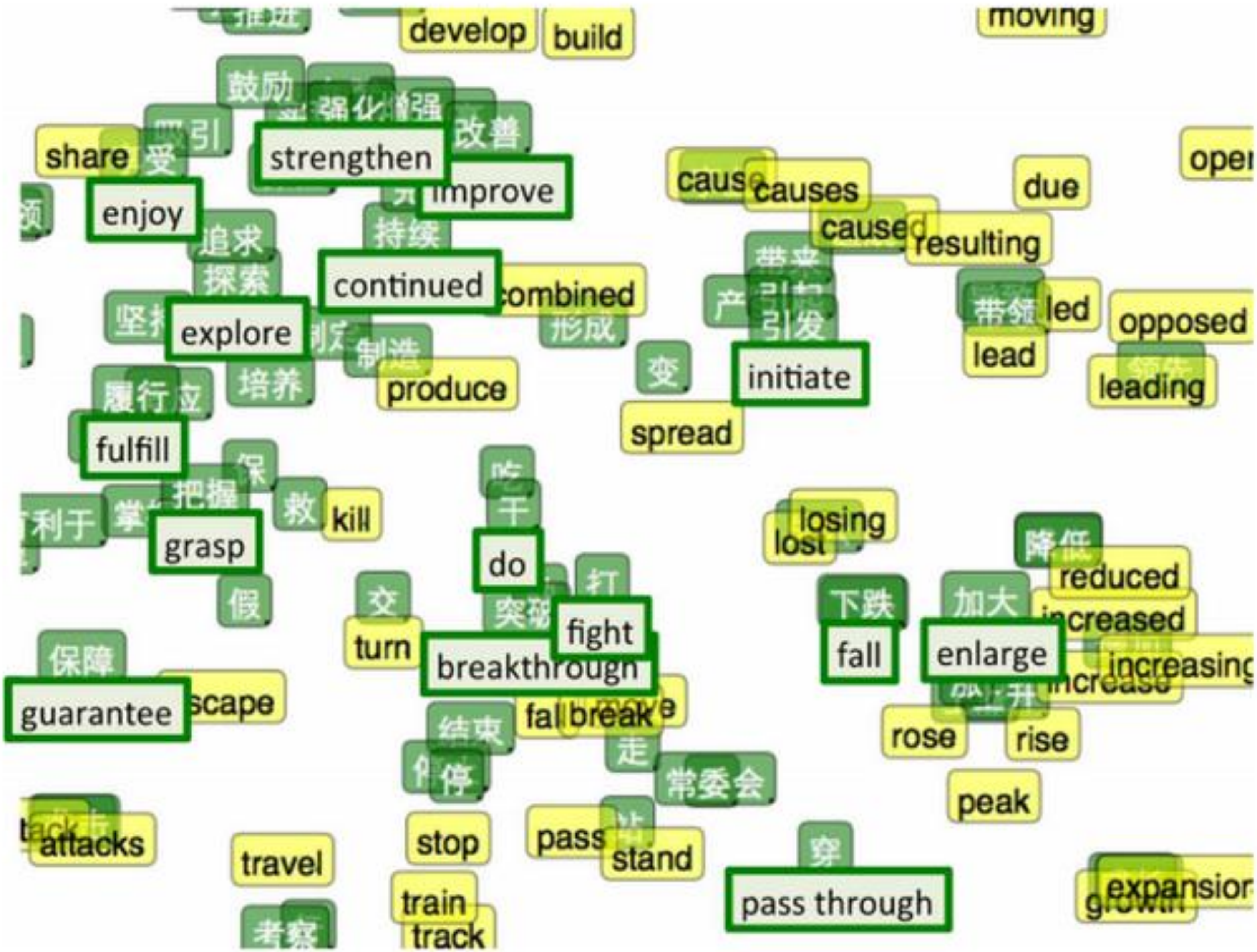
heiress
niece
aunt
sister
countess
duchess
empress
madam
heir
nephew
woman
uncle
earl
queen
duke
brother
emperor
sir
man
king
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5



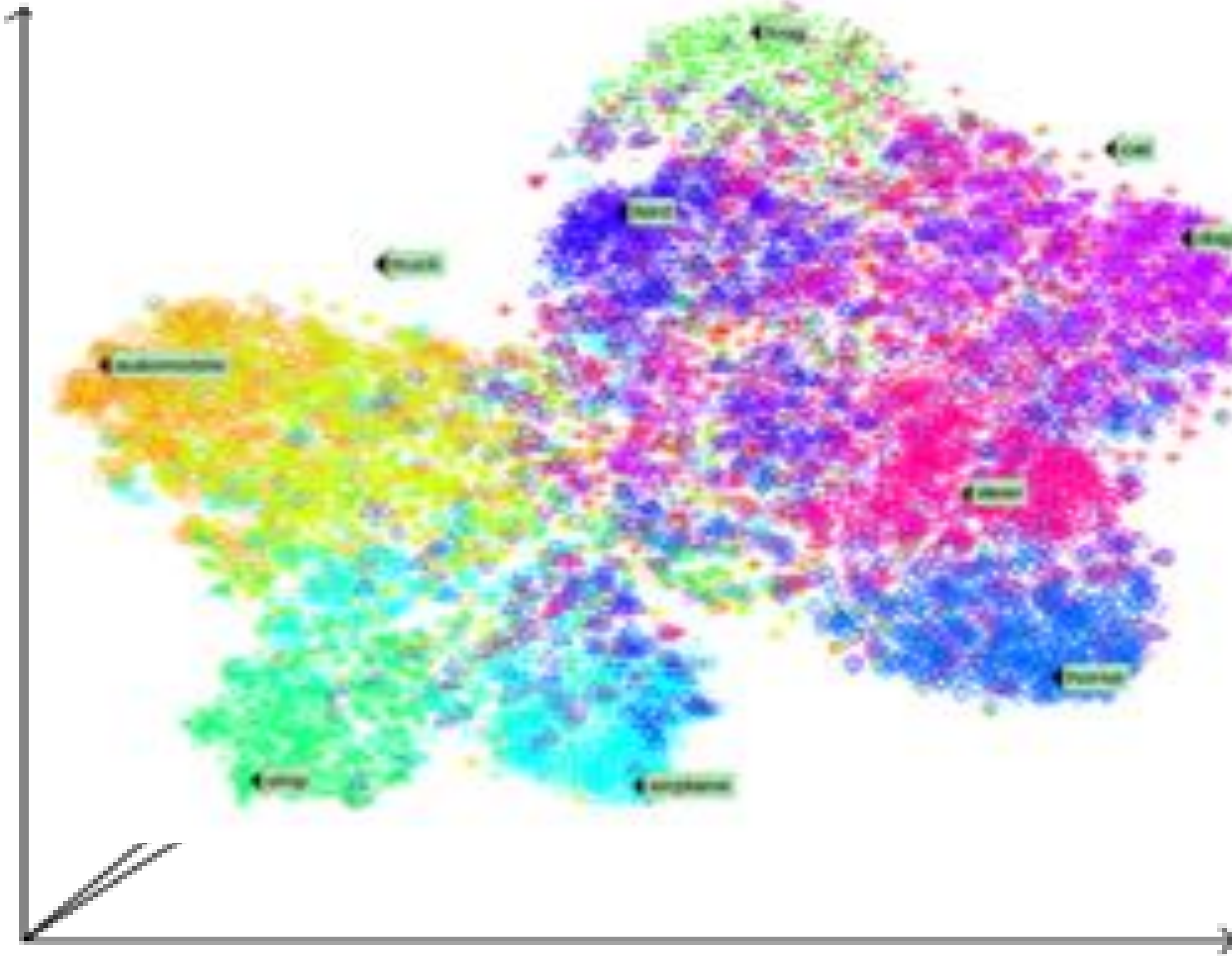
slowest
slower
shortest
shorter
slow
short
stronger
strongest
louder
loudest
strong
loud
clearer
clearest
softer
softest
clear
soft
darker
darkest
dark
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
0.6



Fu, Ruiji, et al. Learning semantic hierarchies via word embeddings. ACL 2014.

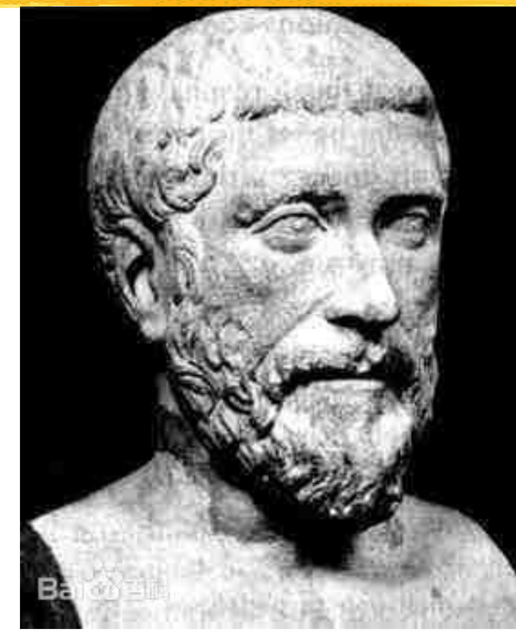


Zou, Will Y., et al. Bilingual word embeddings for phrase-based machine translation. EMNLP 2013.



3.5 Thesaurus及term自动关联(Word2Vec)

- 毕达哥拉斯：
  “万物皆数”
- 深度学习：
  “万物皆数组”（向量）！



（约前572—约前500）

Yann LeCun

Yoshua Bengio

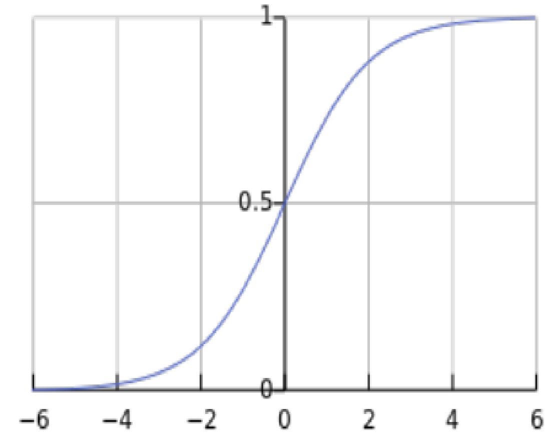
Geoffrey Hinton

The skip-gram model with negative sampling (HW2)

- From paper: "Distributed Representations of Words and Phrases and their Compositionality" (Mikolov et al. 2013)
- Overall objective function (they maximize): $J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\theta)$

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

- The logistic/sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$
  (we'll become good friends soon)
- We maximize the probability of two words co-occurring in first log and minimize probability of noise words



$$
\begin{aligned}
f'(z) &= (\frac{1}{1+e^{-z}})' \\
&= \frac{e^{-z}}{(1+e^{-z})^2} \\
&= \frac{1+e^{-z}-1}{(1+e^{-z})^2} \\
&= \frac{1}{(1+e^{-z})}(1-\frac{1}{(1+e^{-z})}) \\
&= f(z)(1-f(z))
\end{aligned}
$$

- Notation more similar to class and HW2:

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

- We take *k* negative samples (using word probabilities)
- Maximize probability that real outside word appears,
  minimize probability that random words appear around center word

- Sample with P(*w*)=U(*w*)$^{3/4}$/Z, the unigram distribution U(w) raised to the 3/4 power (We provide this function in the starter code).
- The power makes less frequent words be sampled more often

$$\frac{\partial J(\theta)}{\partial v_c} = -\sigma(-u_o^T v_c)u_o + \sum_{k=1}^{K} \sigma(u_k^T v_c)u_k$$

$$\frac{\partial J(\theta)}{\partial u_o} = -\sigma(-u_o^T v_c)v_c$$

$$\frac{\partial J(\theta)}{\partial u_k} = \sum_{k=1}^{K} \sigma(u_k^T v_c)v_c$$

证明说明见：https://medium.com/analytics-vidhya/maths-behind-word2vec-explained-38d74f32726b

此外：

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$