Word Embedding

Jaegul Choo (주재걸)
고려대학교 컴퓨터학과
https://sites.google.com/site/jaegulchoo/

Slides mostly made by my student, Yunjey Choi, Cheonbok Park

Contents

1. Word Embedding

1-1. Word Embedding이란**?**
1-2. Word2Vec 모델 및 알고리듬
1-3. Word2Vec 적용분야

01

기존의 Word Representation 방법

- 가장 쉬운 방법으로는 “one-hot” representation
- e.g., horse = [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
- zebra = [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]

- horse와 zebra 간의 유사도를 구하는 방법 중 하나를 생각해보면

  horse ∩ zebra = 0 (nothing)
- 그러나, 우리는 horse와 zebra가 최소한 포유동물이라는 공통점을 가진다는 것을 알고 있음.

- 만약 단어의 개수가 증가한다면??

Word Embedding 이란?

- 한 단어를 **벡터 형태**로 표현
- ‘cat’과 ‘kitty’는 **비슷한 단어**이므로 **비슷한 단어 표현형**을 가짐
- ‘hamburger’는 ‘cat’, ‘kitty’와 비슷하지 않으므로 다른 표현형 = 거리가 멀다

Word의 기존의 다른 Vector Representation 방법들

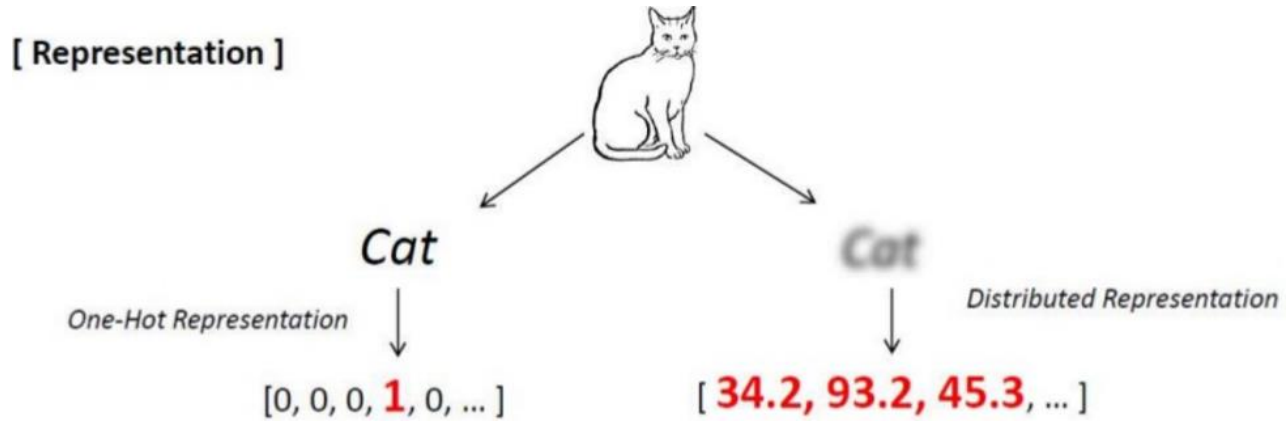
- Distributional semantics: **A word's meaning is given by the words that frequently appear close-by**
  - *"You shall know a word by the company it keeps"* (J. R. Firth 1957: 11)
  - One of the most successful ideas of modern statistical NLP!
- When a word *w* appears in a text, its **context** is the 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…*

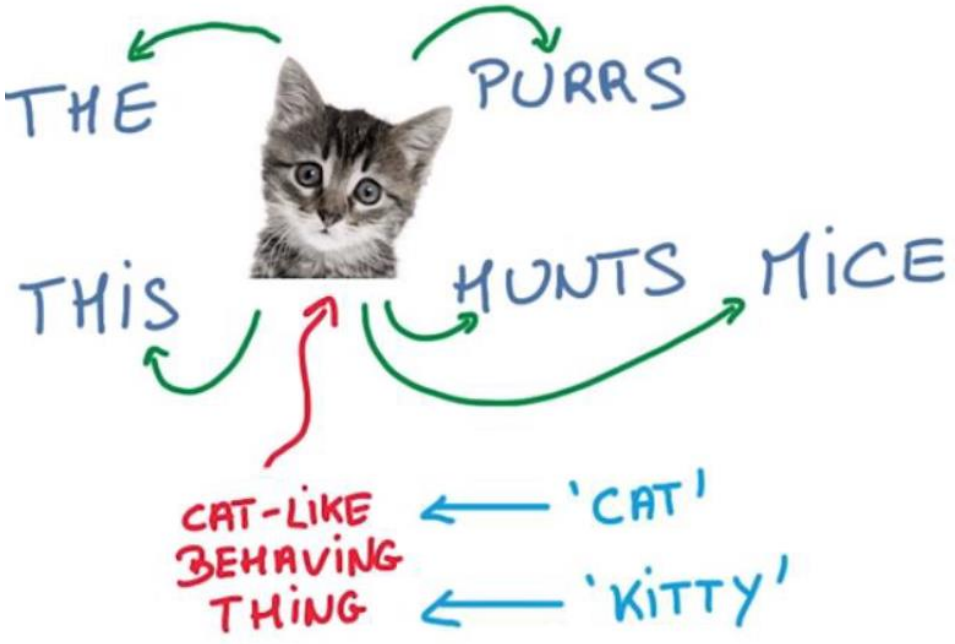
These context words will represent ***banking***

Word2Vec

- Word의 distributed vector representation 학습기법



- **주변 단어**로 해당 단어를 학습시키는 알고리즘
  - 주변에 있으면 **단어의 의미가 더 비슷**할 것이라 가정
- **예시)**
  - The **cat** purrs.
  - This **cat** hunts mice.



Distributed Representation : Word Vector

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

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

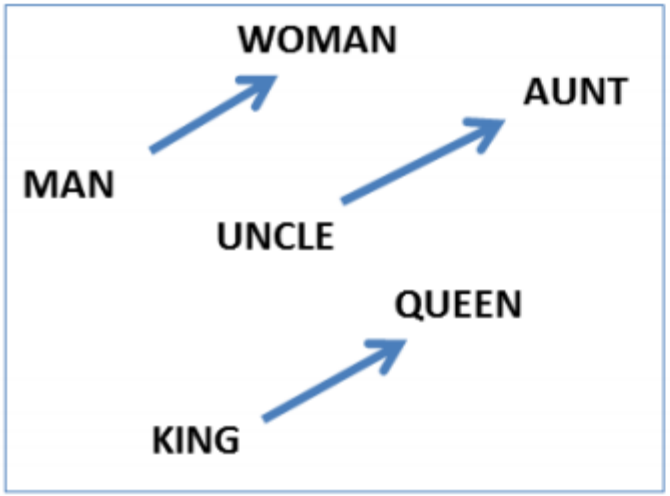
Note: word vectors are sometimes called word embeddings or word representations. They are a distributed representation.

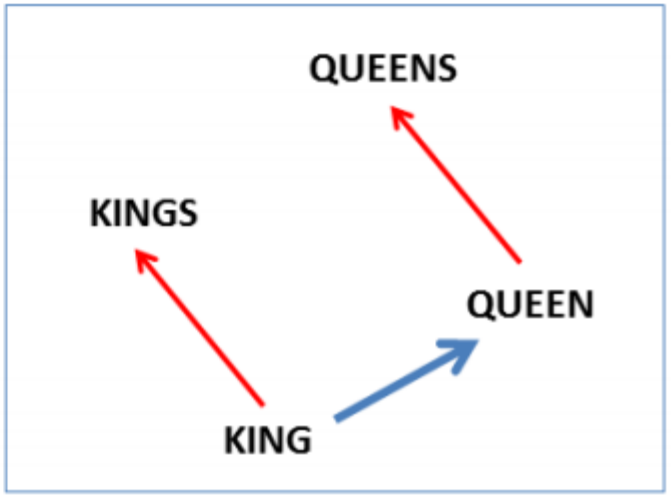
Word meaning as a neural word vector – visualization

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

Word2Vec의 특성

- 워드 벡터, 즉 공간에서의 벡터 포인트 간의 관계는 해당 단어들 간의 관계를 나타냄.
- 같은 관계는 같은 벡터로 나타남.





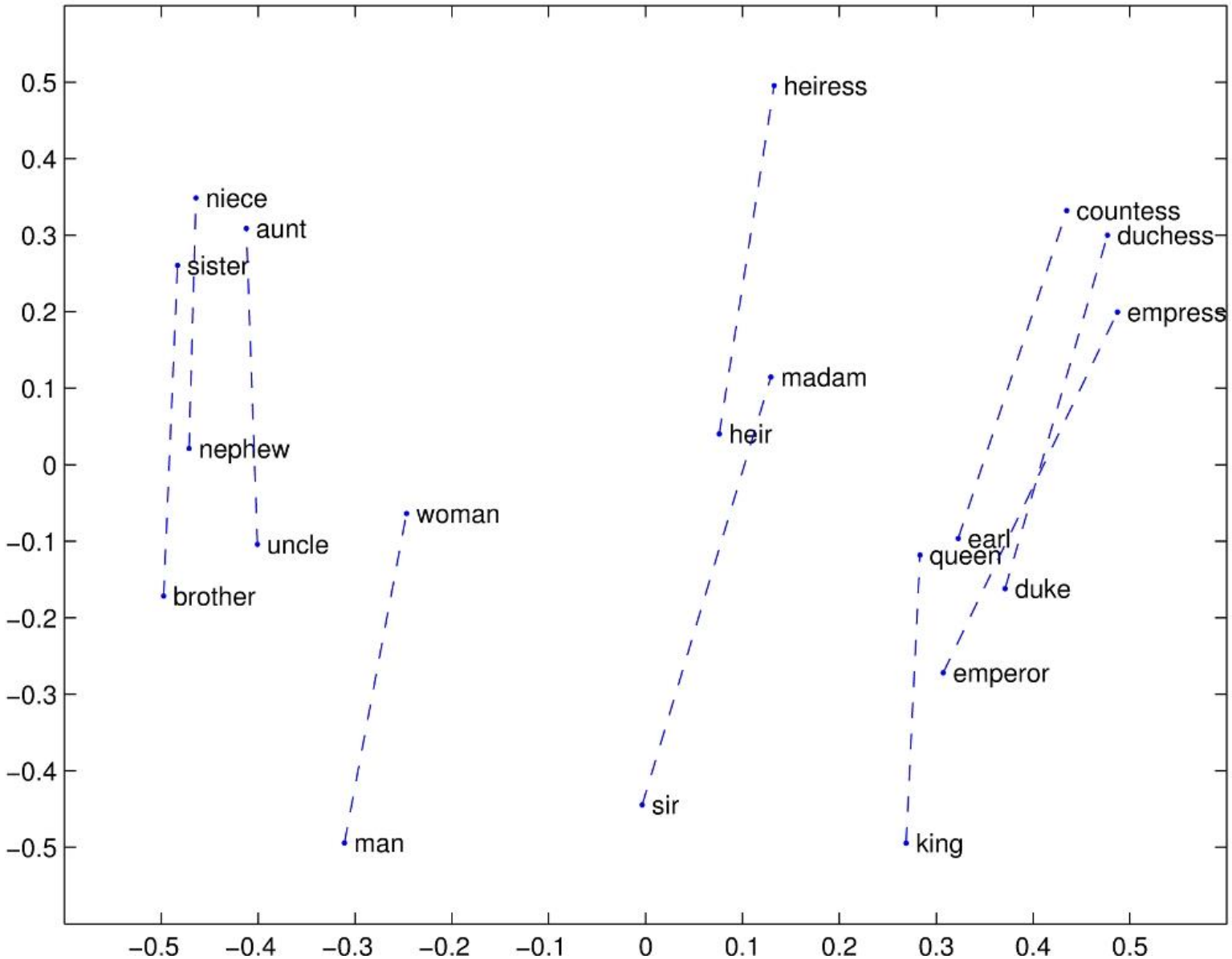
(Mikolov et al., NAACL HLT, 2013)

- e.g.,

vec[queen] - vec[king] = vec[woman] - vec[man]

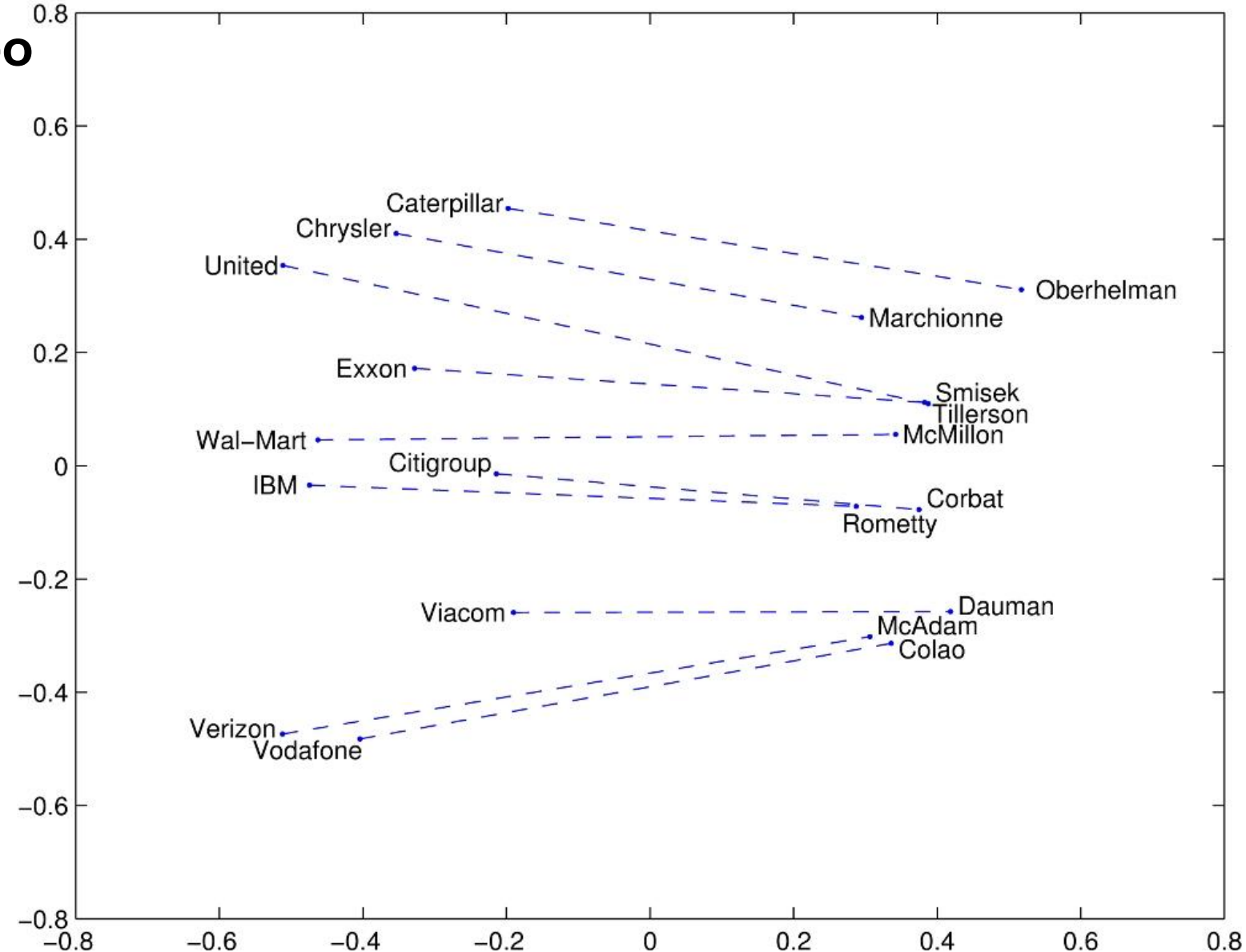
Word2Vec의 특성 – Linear Substructure

man - woman



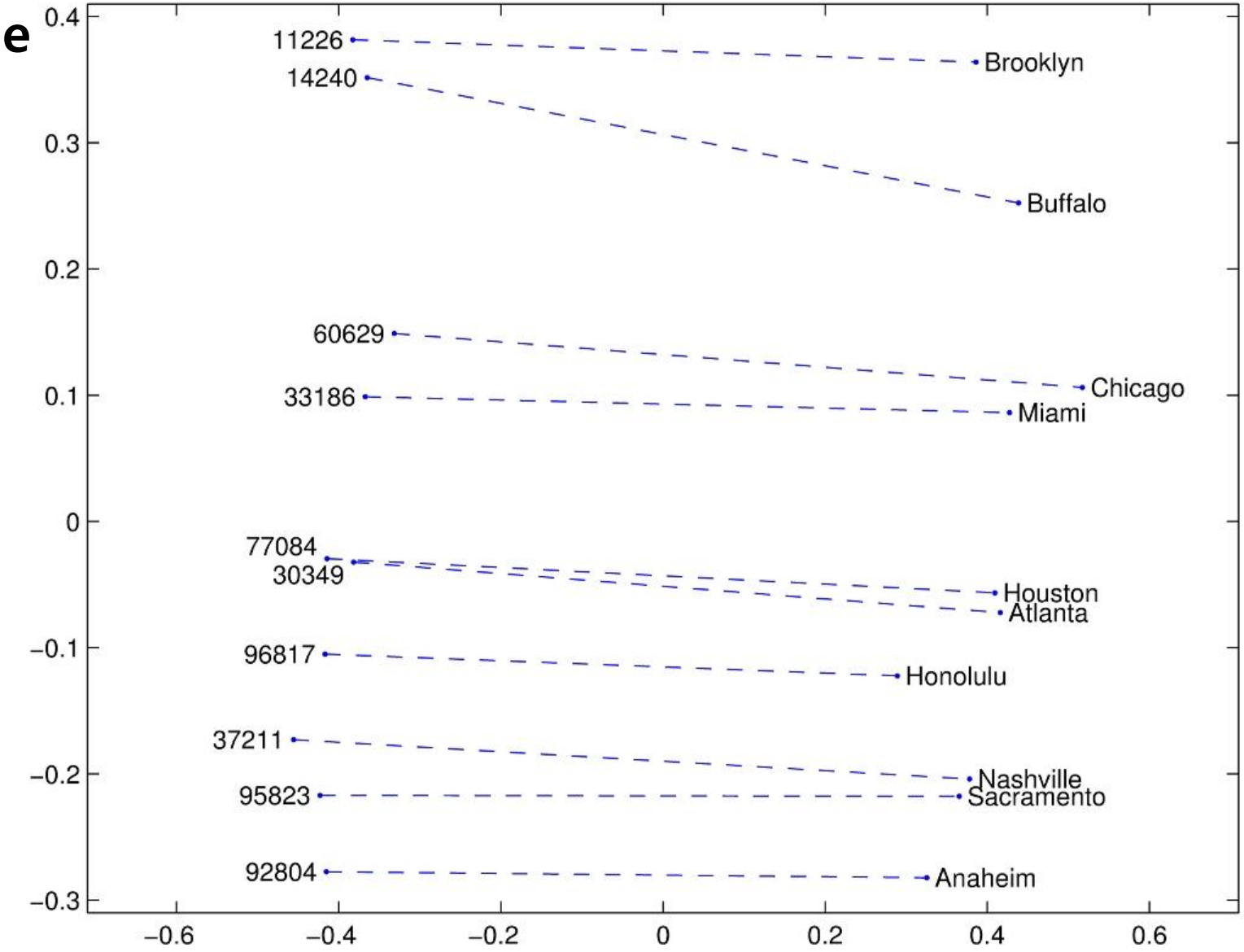
Word2Vec의 특성 – Linear Substructure

company - ceo



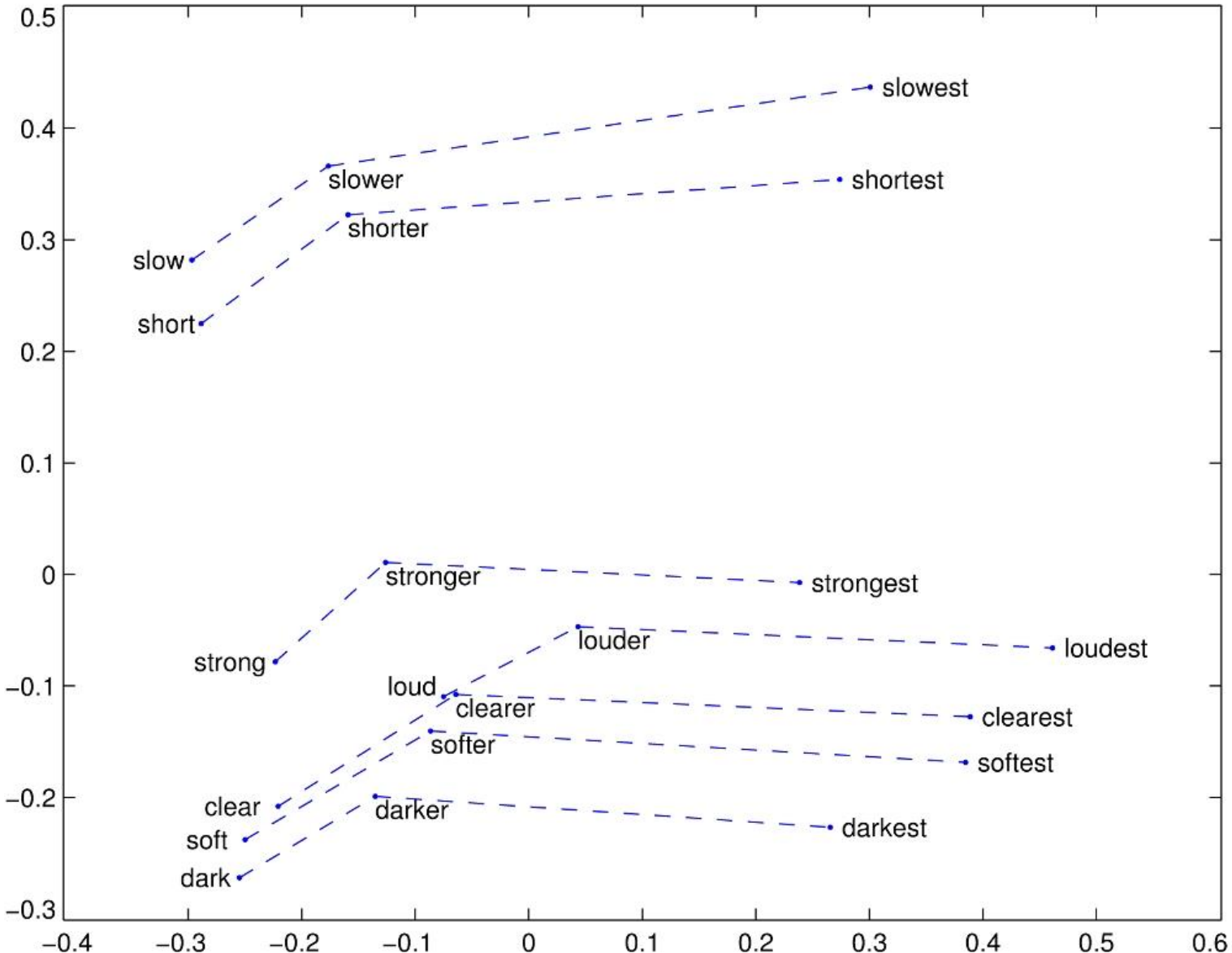
Word2Vec의 특성 – Linear Substructure

city - zip code



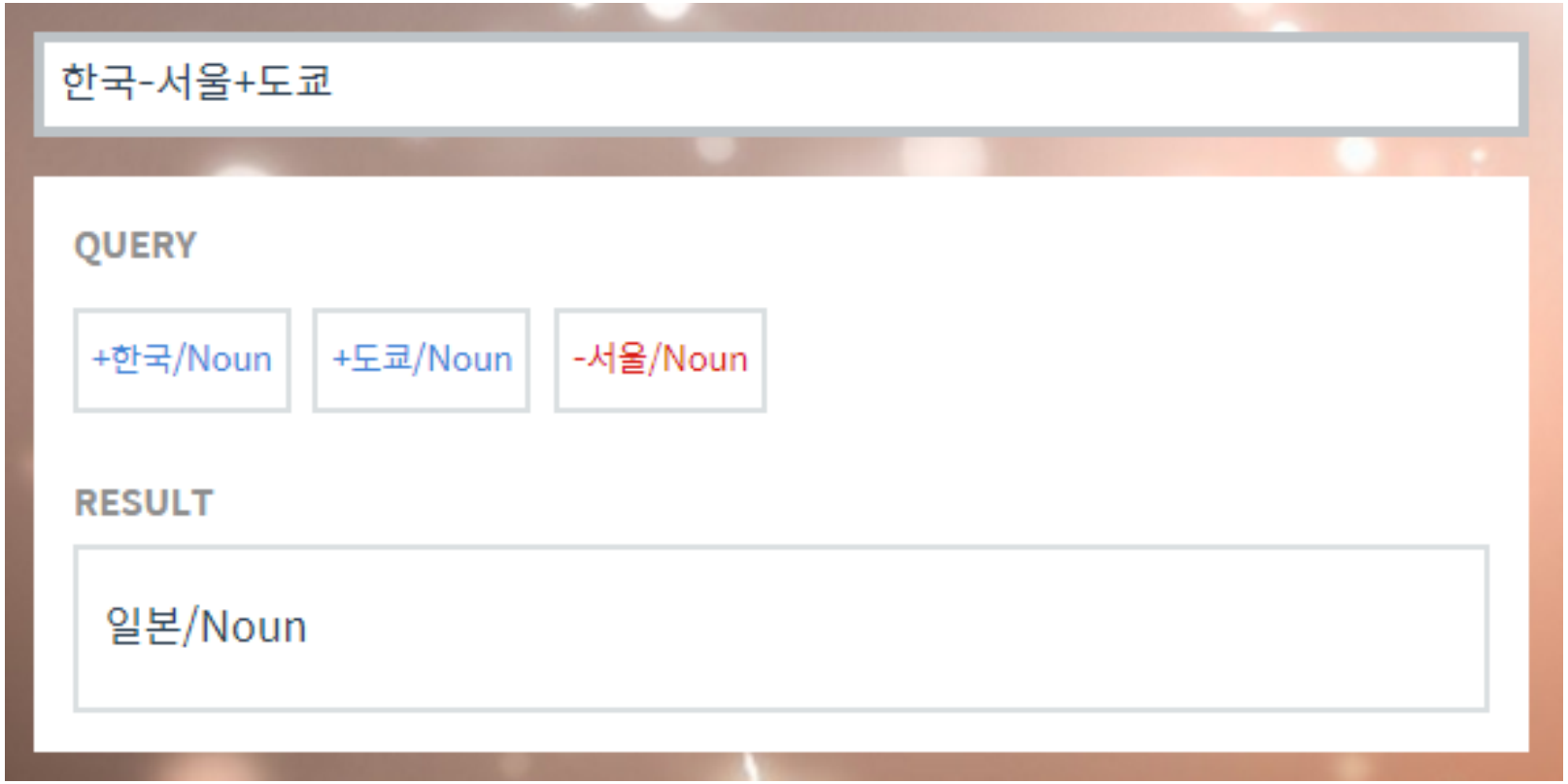
Word2Vec의 특성 – Linear Substructure

comparative
- superlative



Word2Vec의 특성 – Analogy Reasoning

- 예제: 한글 word2vec
  - http://w.elnn.kr/search/



Word2Vec의 특성 – Analogy Reasoning

- 다른 예제들: http://wonjaekim.com/archives/50

- 여기서 wor2vec이 만들지 않은 예제는?

- 1) 컴퓨터-기계+인간 = 일반인
- 2) 인간 – 생각 = 악마
- 3) 사랑 + 이별 = 추억
- 4) 사랑 – 이별 = 미움
- 5) 나+운명 = 당신
- 6)이명박 – 돈 = 박근혜

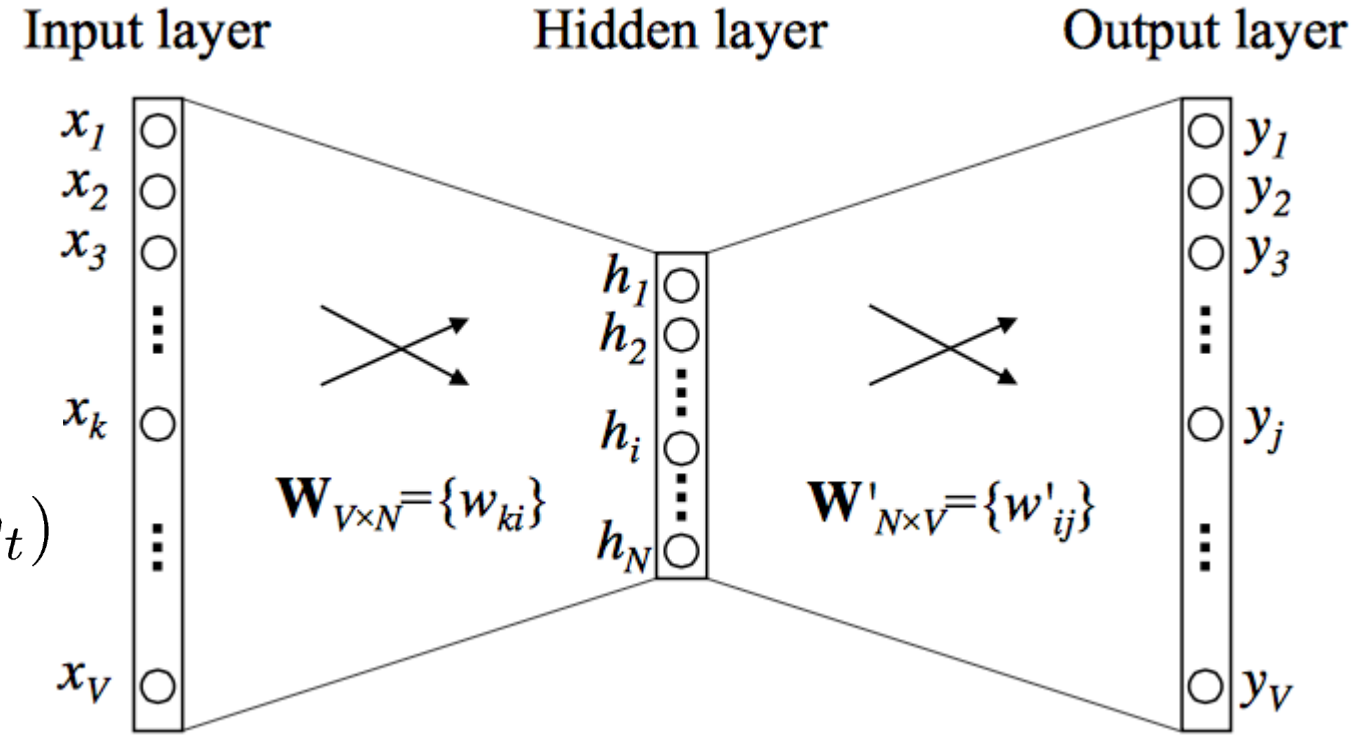
Word2Vec의 특성 – Semantic Similarity

- 예제: https://github.com/dhammack/Word2VecExample

- Word intrusion detection
  - staple hammer saw drill
  - math shopping reading science
  - rain snow sleet sun
  - eight six seven five three owe nine
  - breakfast cereal dinner lunch
  - england spain france italy greece germany portugal australia

How Word2Vec Algorithm Works

- V : vocabulary size
- N : embedding dimension



- Objective criterion

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

where

$$\log p(o|c) = \log \frac{\exp\left(u_o^T v_c\right)}{\sum_{w=1}^{W} \exp\left(u_w^T v_c\right)}$$, which is a softmax function.

Word2vec : overview

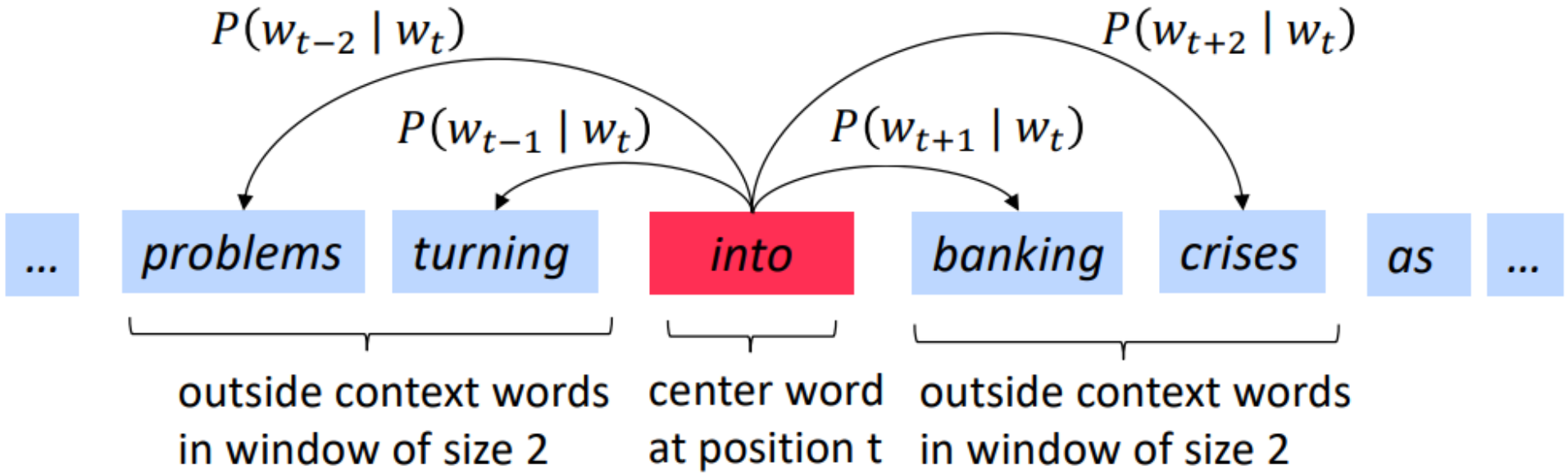
Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

Idea:

- We have a large corpus of text
- 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* (or vice versa)
- Keep adjusting the word vectors to maximize this probability

Word2vec : overview

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



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.

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

θ is all variables to be optimized

sometimes called *cost* or *loss* function

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

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

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Word2vec : overview with Vectors

- We want to minimize the objective function:

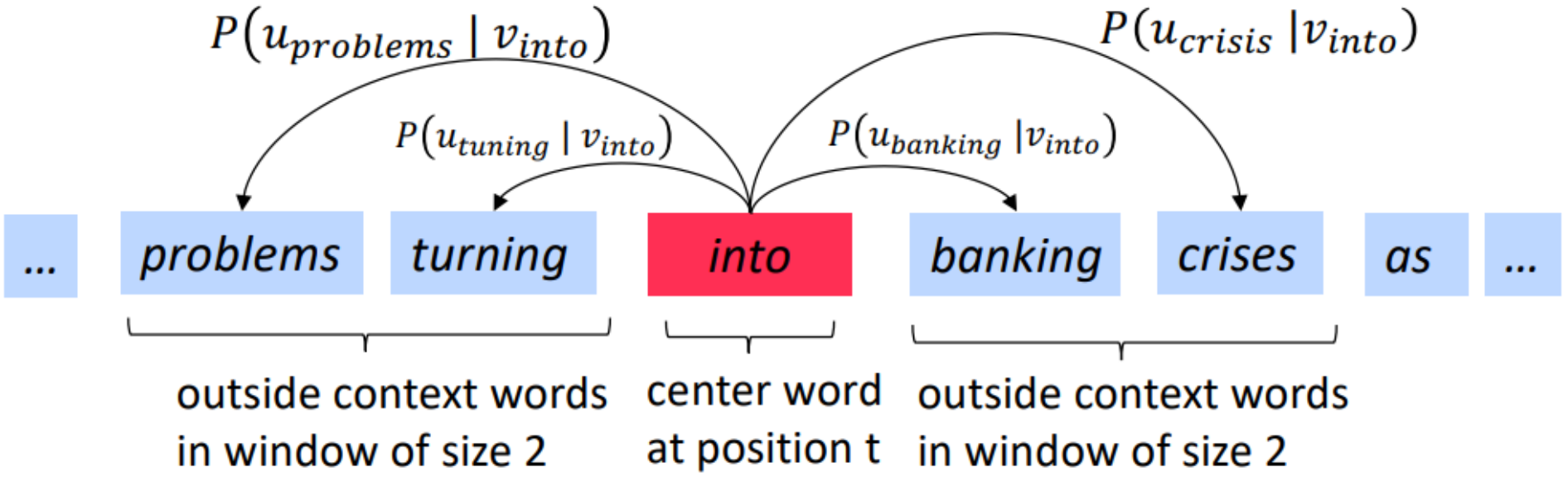
$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m \\ j\neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

- 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*:

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

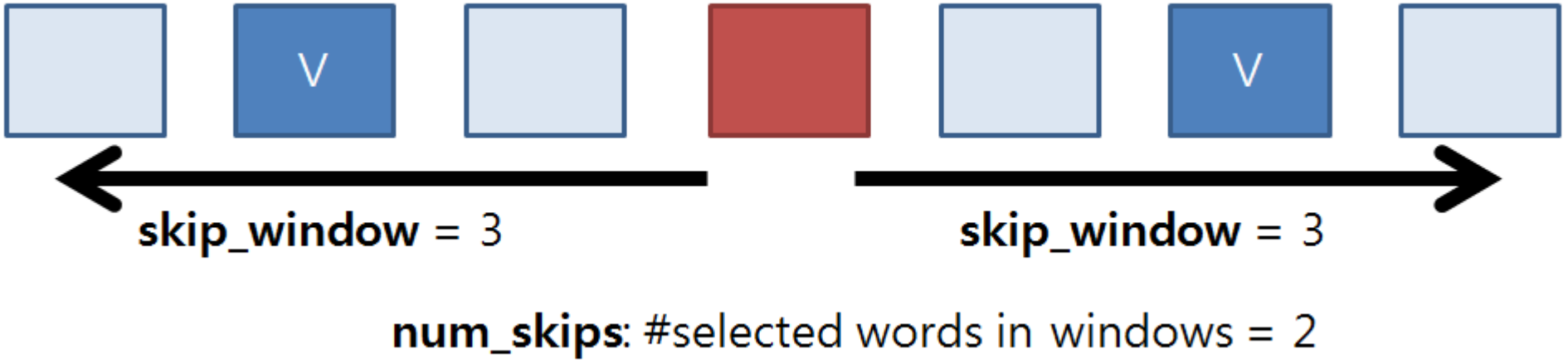
Word2vec : overview

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



How Word2Vec Algorithm Works

- 먼저 window 크기를 정하고



- Skipgram: 중심 단어로 window 내의 주변 단어 (context words)를 예측
  - Semantic task에 장점을 보이나, 학습이 느림.
- CBOW: 주변 단어들로 중심 단어를 예측
  - Syntactic task에 장점을 보이고, 학습이 상대적으로 빠름

Word2vec : overview

Exponentiation makes anything positive

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

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

Normalize over entire vocabulary to give probability distribution

- This is an example of the **softmax function** $\mathbb{R}^n \to \mathbb{R}^n$

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

- 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

To train the model: Compute all vector gradients

- Recall: θ represents **all** model parameters, in one long vector
- In our case with *d*-dimensional vectors and *V*-many words:

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

- Remember: every word has two vectors
- We optimize these parameters by walking down the gradient

Word2vec

Why two vectors? → Easier optimization. Average both at the end.

Two model variants:

1. Skip-grams (SG)

   Predict context (”outside”) words (position independent) given center word

2. Continuous Bag of Words (CBOW)

   Predict center word from (bag of) context words

This lecture so far: **Skip-gram model**

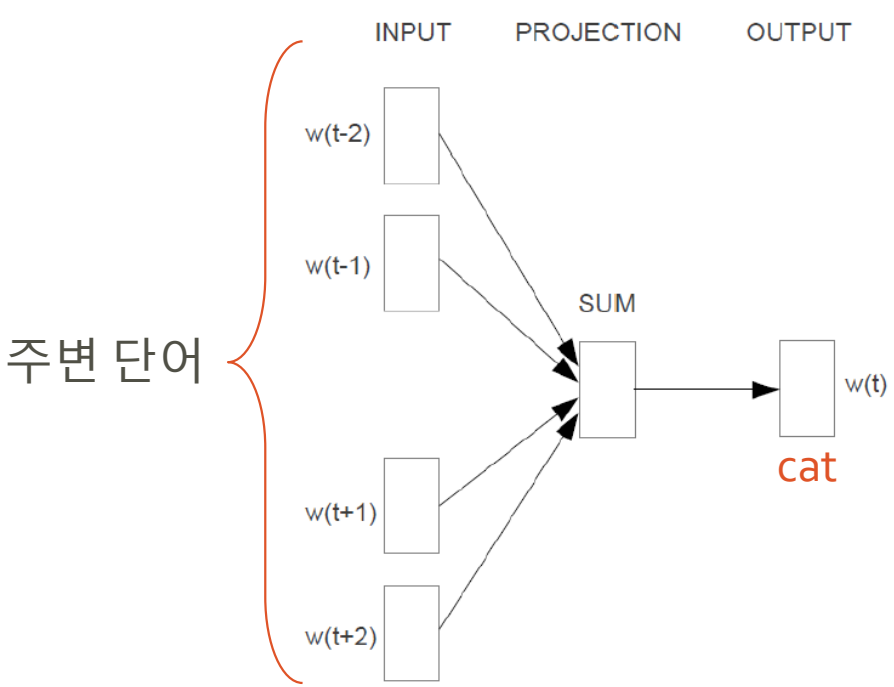
Additional efficiency in training:

1. Negative sampling

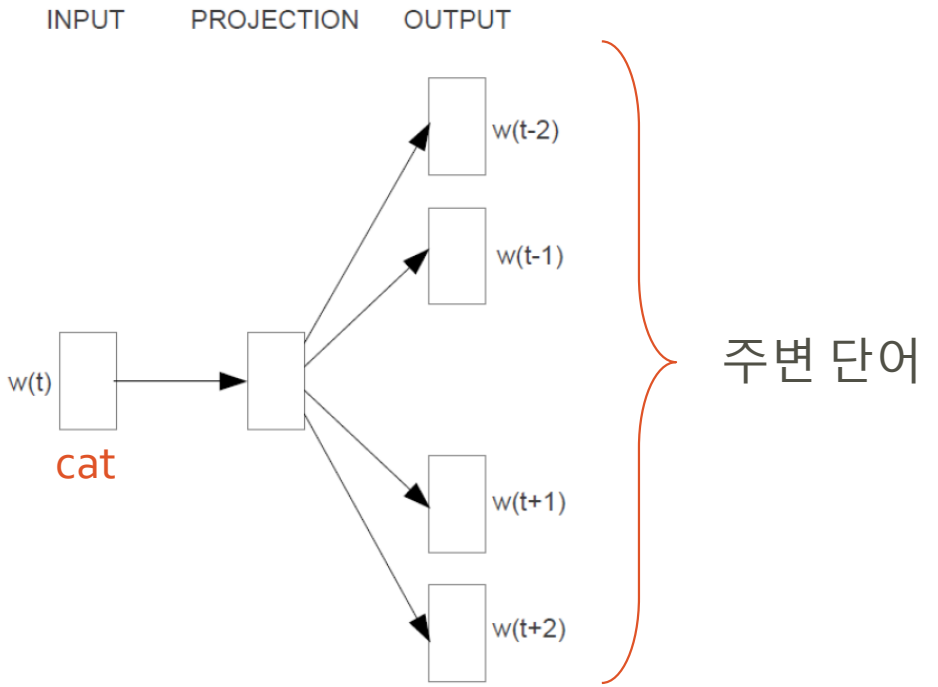
So far: Focus on **naïve softmax** (simpler training method)

Two Models of Word2Vec

Continuous Bag-Of-Words (CBOW)



Skip-gram

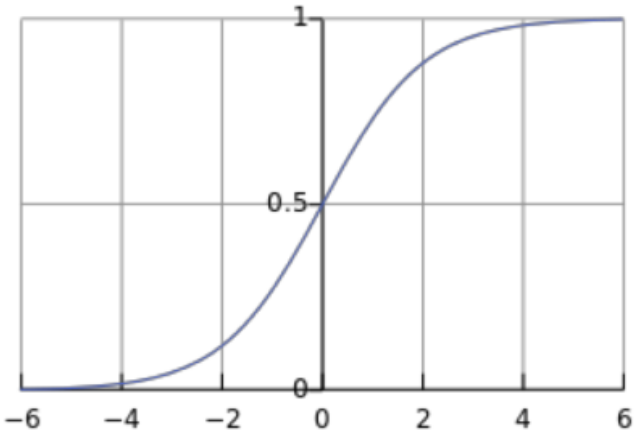


The skip-gram model with negative sampling

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



The skip-gram model with negative sampling

- Notation more similar to class and HW2:

$$J_{neg-sample}(\boldsymbol{o}, \boldsymbol{v}_c, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))$$

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

- $P(w)=U(w)^{3/4}/Z$, 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

기존의 방법으로 Word 간 유사도 구하려면?

- 전 슬라이드의 방법을 사용하여 term-document matrix 구축
- e.g., Document 1 = "John likes movies. Mary likes too."
  Document 2 = "John also likes football."
- 두 간어가 얼마나 자주 같은 문서에서 등장했나로 유사도를 측정

Vocabulary	Doc 1	Doc 2
John	1	1
likes	2	1
movies	1	0
also	0	1
football	0	1
Mary	1	0
too	1	0

Co-occurrence similarity

- John & likes : 2 (documents)
- movies & also: 0 (documents)

Word의 기존의 다른 Vector Representation 방법들

- Matrix factorization 기반 기법들
  - **Singular vector representation** (혹은 latent semantic indexing)
  - Nonnegative matrix factorization
- Probabilistic topic modeling 기반 기법들
  - Probabilistic latent semantic indexing
  - Latent Dirichlet allocation

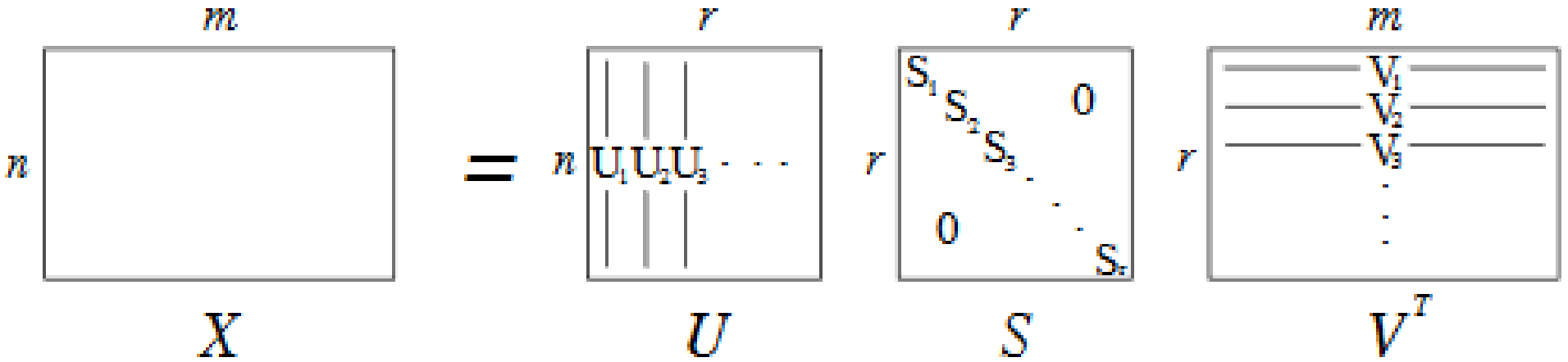
Singular Value Decomposition

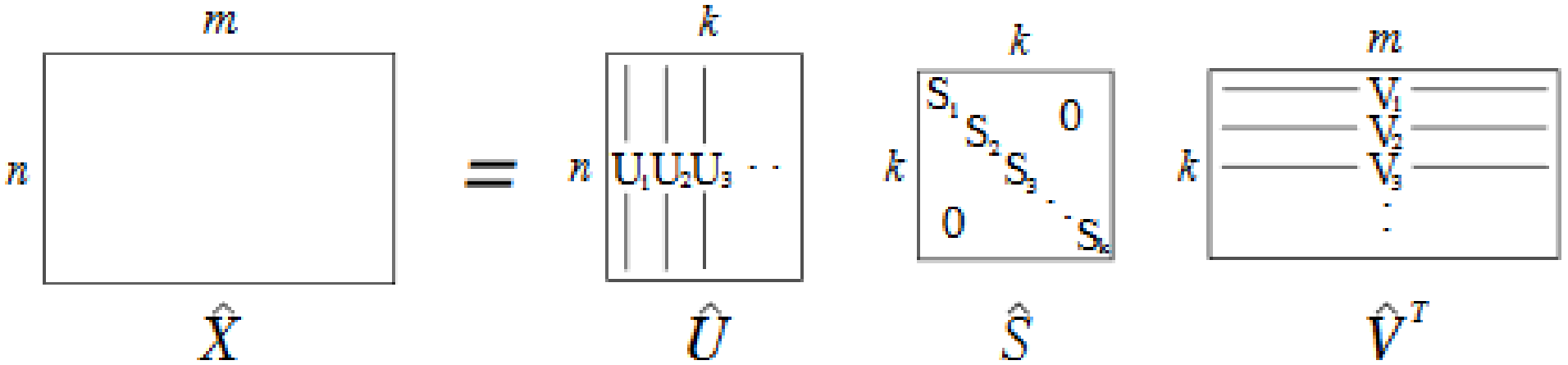
Corpus = {"I like deep learning"
"I like NLP"
"I enjoy flying"}

Context = previous word and next word

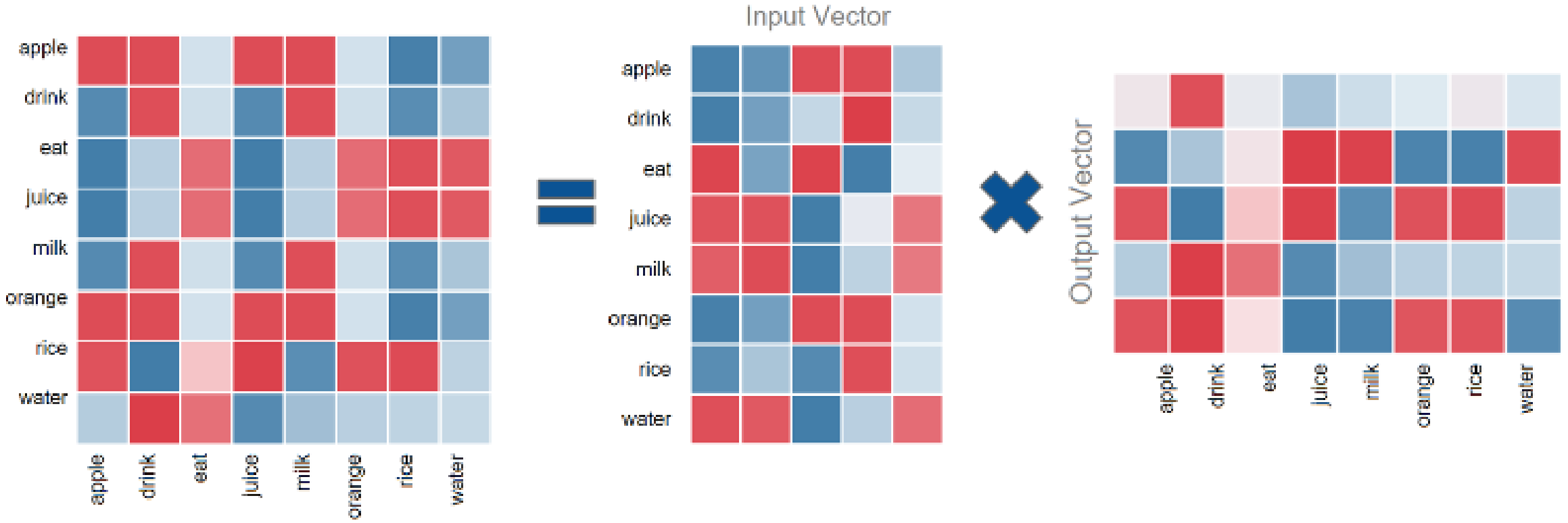
counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Dimension Reduction using Singular Value Decomposition





Singular Value Decomposition Example

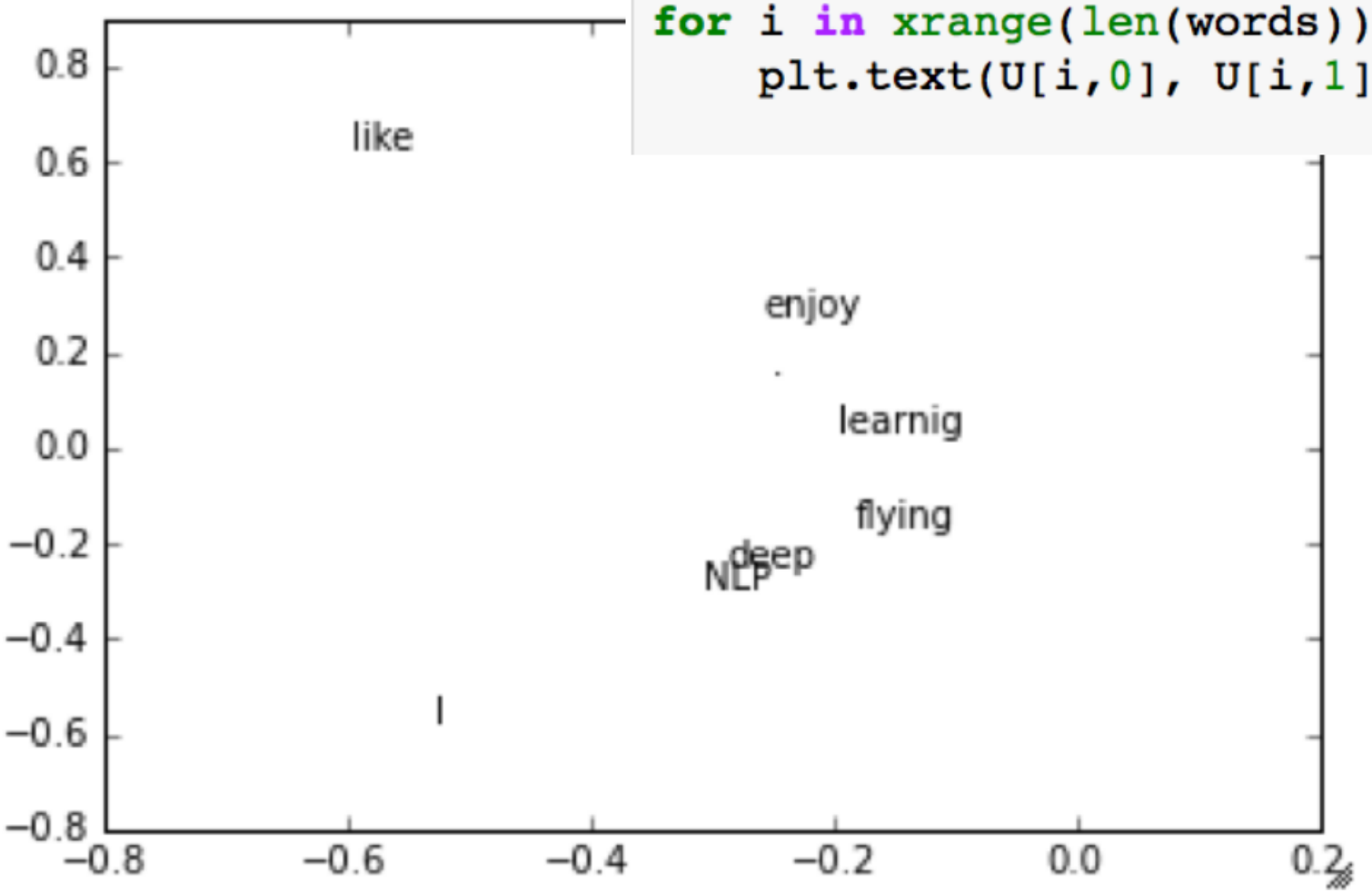


The problem with this method, is that we may end up with matrices having billions of ro ws and columns, which makes SVD computationally restrictive.

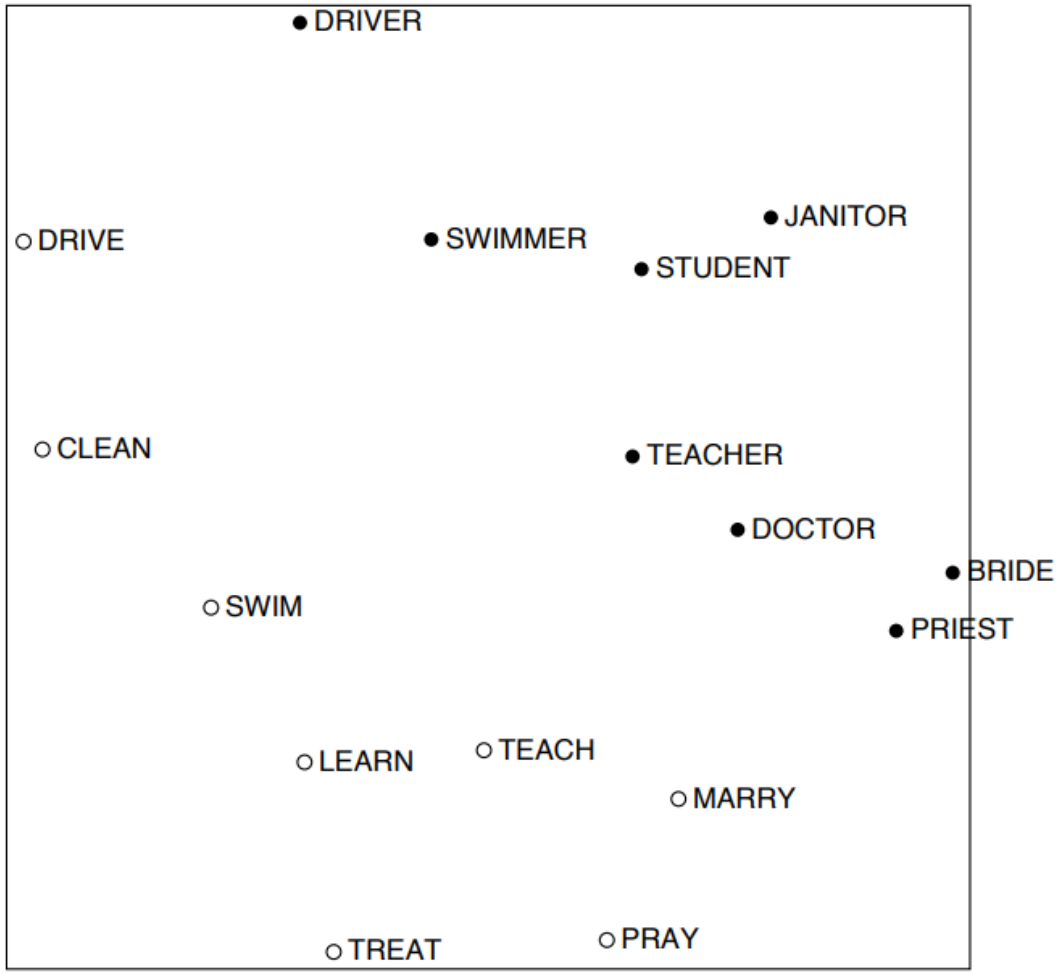
Singular Value Decomposition Example

Corpus: I like deep learning. I like NLP. I enjoy flying.
Printing first two columns of U corresponding to the 2 biggest singular values

```
for i in xrange(len(words)):
    plt.text(U[i,0], U[i,1], words[i])
```

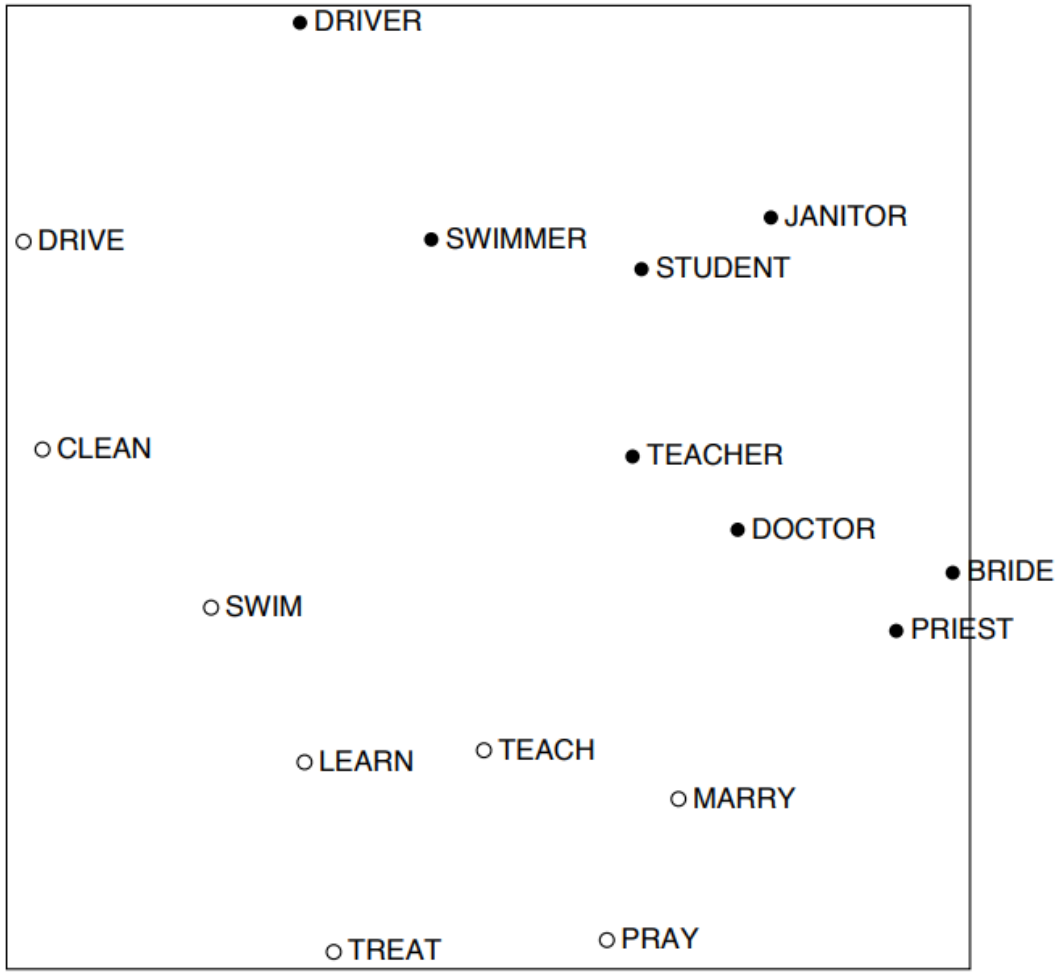


Interesting semantic patterns emerge in the vectors



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

Interesting semantic patterns emerge in the vectors



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

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

Scaling the counts in the cells can help ***a lot***

- Problem: function words (*the, he, has*) are too frequent → syntax has too much impact. Some fixes:
  - min(X,t), with t ≈ 100
  - Ignore them all
- Ramped windows that count closer words more
- Use Pearson correlations instead of counts, then set negative values to 0
- Etc.

Count based vs. direct prediciton

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

- Fast training
- Efficient usage of statistics
- Primarily used to capture word similarity
- Disproportionate importance given to large counts

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

- Scales with corpus size
- Inefficient usage of statistics
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity

Encoding meaning in vector differences

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

	x = solid	x = gas	x = water	x = 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 in vector differences

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

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

Encoding meaning in vector differences

Q: How can we capture ratios of co-occurrence probabilities as linear meaning components in a word vector space?

A: Log-bilinear model: $w_i \cdot w_j = \log P(i|j)$

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

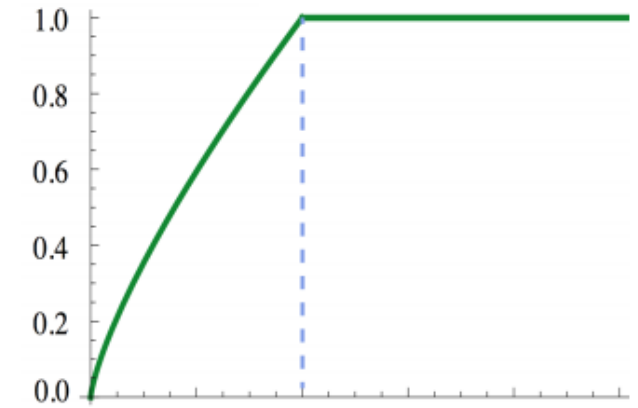
Combining the best of both worlds GloVe

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

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

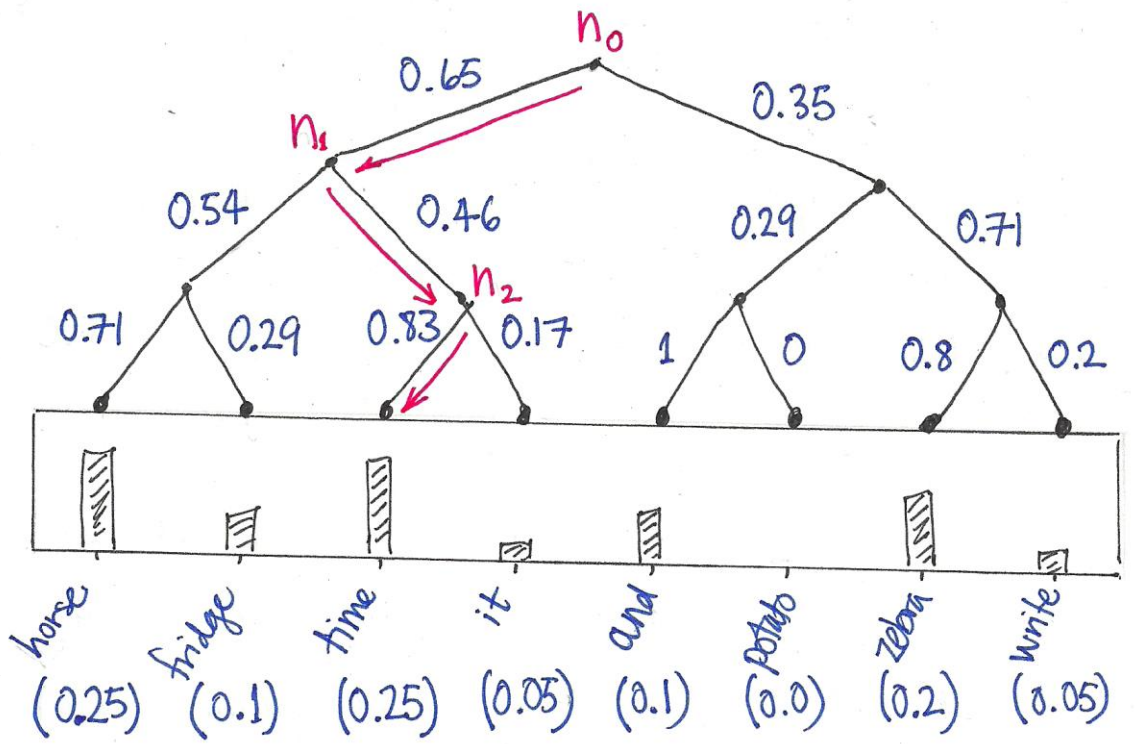
- Fast training
- Scalable to huge corpora
- Good performance even with small corpus and small vectors

$f \sim$



Efficient Solution 1: Hierarchical Softmax

- 각 단어들을 leaves로 가지는 binary tree를 생성함.
- 해당하는 단어의 확률을 계산할 때 root에서부터 해당 leaf로 가는 path에 따라서 확률을 곱해나가는 식으로 해당 단어가 나올 최종적인 확률을 계산함.
- 즉, 각각의 internal node를 왼쪽, 오른쪽 노드들로 분기하는 Bernoulli 확률 변수라 생각하여, 각 단어의 확률을 계산함



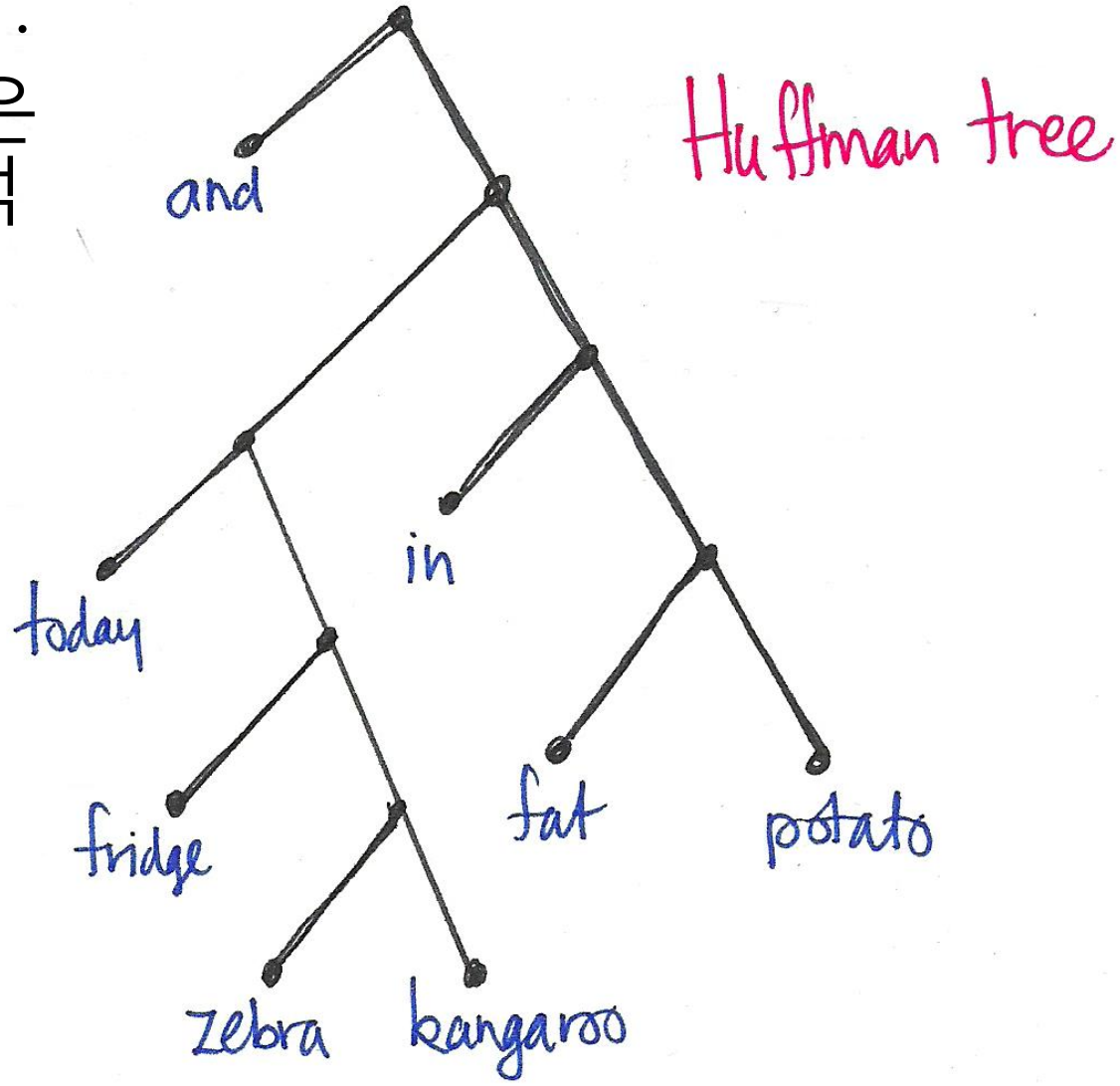
$$P(\texttt{time}|\mathcal{C}) = P_{n_0}(\text{left}|\mathcal{C})P_{n_1}(\text{right}|\mathcal{C})P_{n_2}(\text{left}|\mathcal{C})$$

- 이를 통해 V번의 계산을 log V번의 계산으로 줄일 수 있음.

Efficient Solution 1: Hierarchical Softmax

- Binary tree는 Huffman tree를 사용함.
- 즉, 자주 등장하는 단어들은 보다 짧은 path로 도달할 수 있기 때문에 전체적인 계산량이 더 낮아지는 효과를 냄.

word	count
fat	3
fridge	2
zebra	1
potato	3
and	14
in	7
today	4
kangaroo	2



Word2Vec의 적용 분야

자연어 처리 거의 모든 분야에서 성능 향상을 가져왔음.

- Word similarity
- Machine translation
- Part-of-speech tagging and named entity recognition
- Sentiment analysis
- Clustering
- Semantic lexicon building

Word2Vec의 적용 분야– Machine Translation

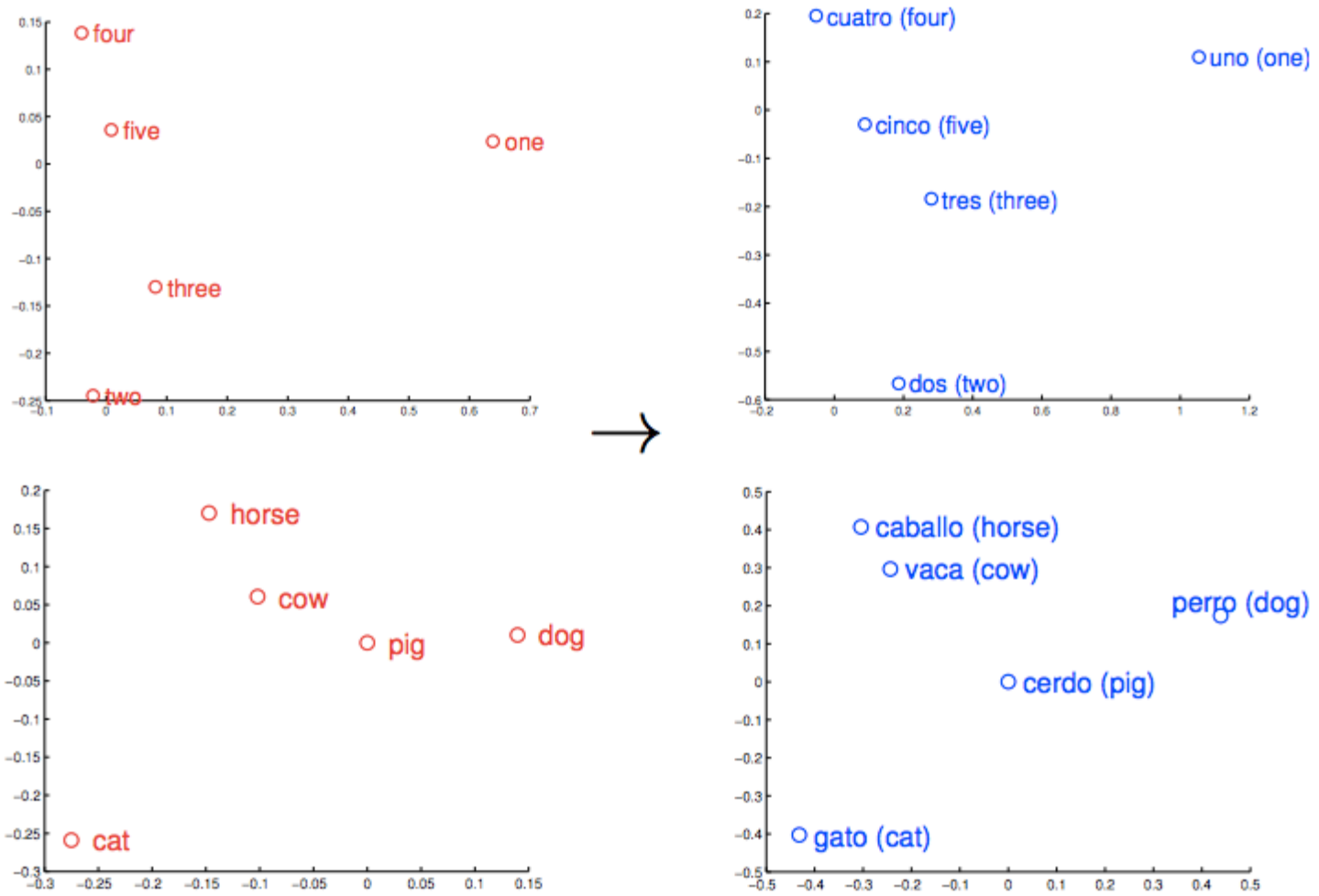
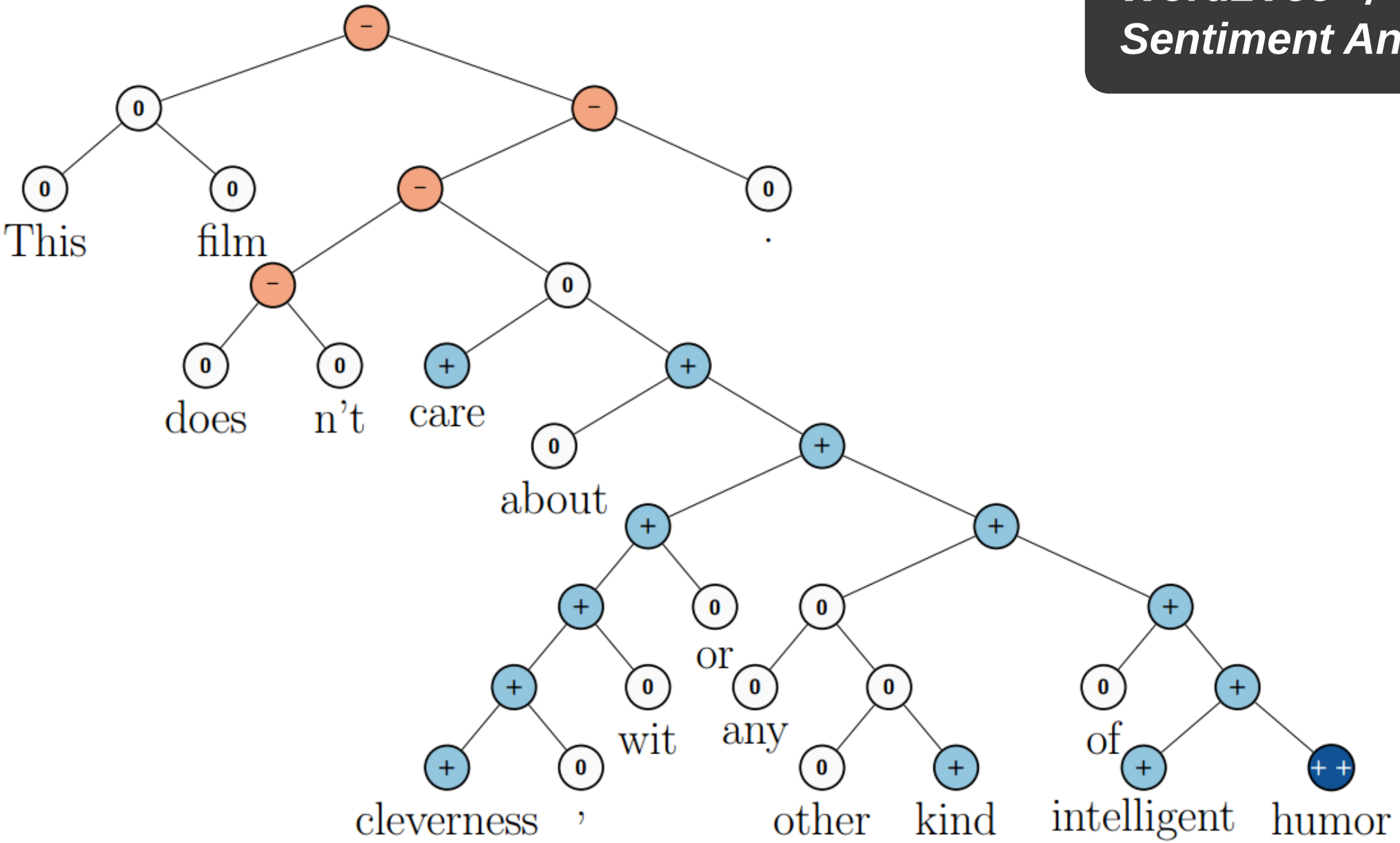


Figure 1: Distributed word vector representations of numbers and animals in English (left) and Spanish (right). The five vectors in each language were projected down to two dimensions using PCA, and then manually rotated to accentuate their similarity. It can be seen that these concepts have similar geometric arrangements in both spaces, suggesting that it is possible to learn an accurate linear mapping from one space to another. This is the key idea behind our method of translation.

Word2Vec의 적용 분야– Sentiment Analysis



Word2Vec의 적용 분야 – Image Captioning

Figure 5. A selection of evaluation results, grouped by human rating.

References

코드

- Python: https://radimrehurek.com/gensim/models/word2vec.html
- C++: https://code.google.com/archive/p/word2vec/
- FastText: https://github.com/facebookresearch/fastText

Useful resources

- https://shuuki4.wordpress.com/2016/01/27/word2vec-%EA%B4%80%EB%A0%A8-%EC%9D%B4%EB%A1%A0-%EC%A0%95%EB%A6%AC/
- https://code.facebook.com/posts/1438652669495149/fair-open-sources-fasttext/
- http://cs224d.stanford.edu/
- https://ronxin.github.io/wevi/
- https://www.lucypark.kr/slides/2015-pyconkr/
- http://cs224n.stanford.edu/