Извличане на знания от текст

Преслав Наков

5 март 2025 г.

Acknowledgments

Slides adapted from

Dan Jurafsky and Richard Socher

Word Similarity

Similarity Between Words

"**fast**" is similar to "**rapid**"

"**tall**" is similar to "**height**"

Question Answering:

- *Q: "How **tall** is Mt. Everest?"*
- *Candidate A: "The official **height** of Mount Everest is 29,029 feet"*

Application: Plagiarism Detection

MAINFRAMES

Mainframes are primarily referred to large computers with rapid, advanced processing capabilities that can execute and perform tasks equivalent to many Personal Computers (PCs) machines networked together. It is characterized with high quantity Random Access Memory (RAM), very large secondary storage devices, and high-speed processors to cater for the needs of the computers under its service.

Consisting of advanced components, mainframes have the capability of running multiple large applications required by many and most enterprises and organizations. This is one of its advantages. Mainframes are also suitable to cater for those applications (programs) or files that are of very high demand by its users (clients). Examples of such organizations and enterprises using mainframes are online shopping websites such as Ebay, Amazon, and computing-giant

MAINFRAMES

Mainframes usually are referred those computers with fast, advanced processing capabilities that could perform by itself tasks that may require a lot of Personal Computers (PC) Machines. Usually mainframes would have lots of RAMs, very large secondary storage devices, and very fast processors to cater for the needs of those computers under its service.

Due to the advanced components mainframes have, these computers have the capability of running multiple large applications required by most enterprises, which is one of its advantage. Mainframes are also suitable to cater for those applications or files that are of very large demand by its users (clients). Examples of these include the large online shopping websites -i.e. : Ebay, Amazon, Microsoft, etc.

Evaluating Similarity

- **Extrinsic (task-based, end-to-end) evaluation:**
  - Question answering
  - Spell checking
  - Essay grading

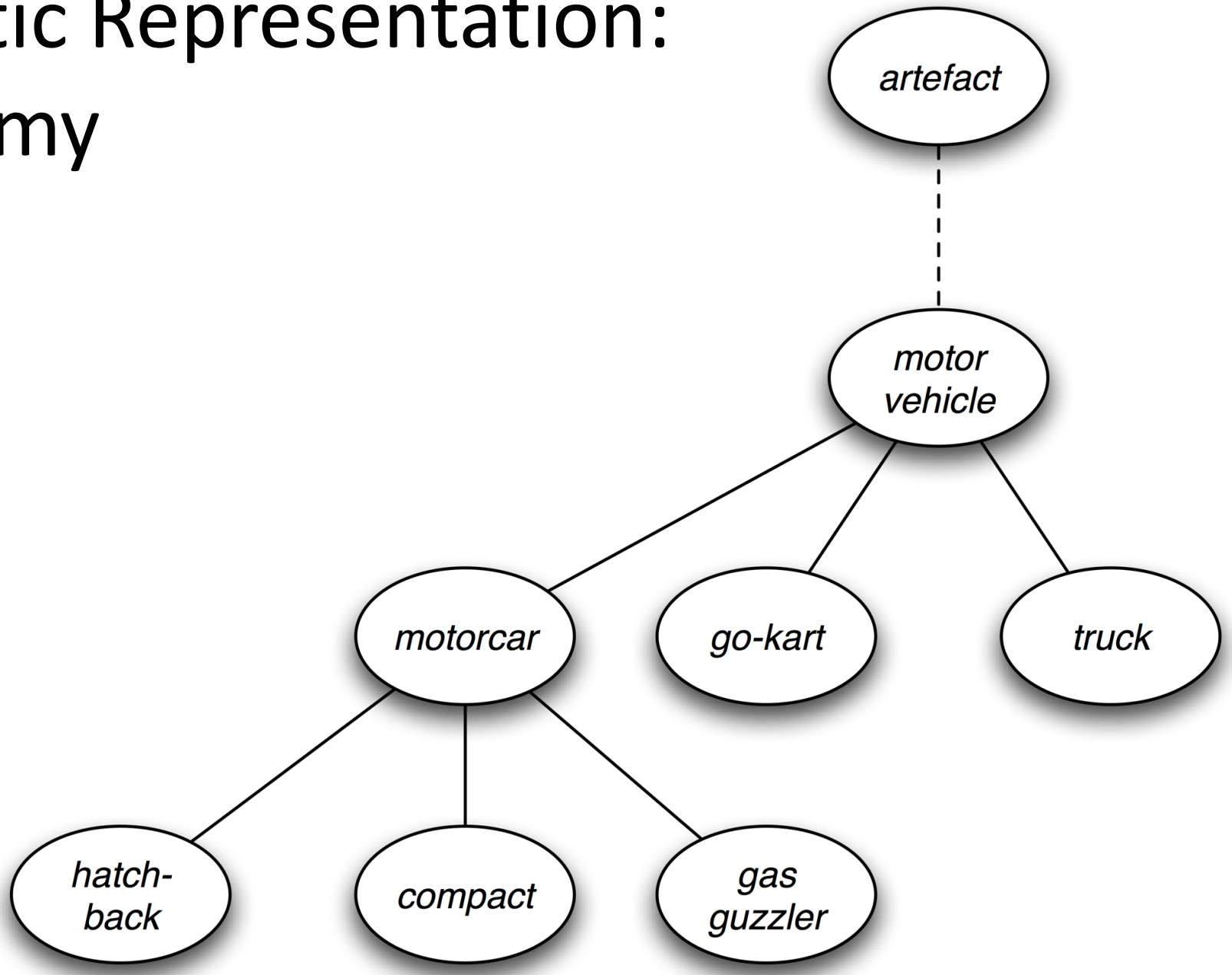
- **Intrinsic evaluation:**
  - Correlation between algorithm and human word similarity ratings
    - Wordsim353: 353 noun pairs rated 0-10. *sim(plane,car)=5.77*
  - Taking TOEFL multiple-choice vocabulary tests
    - Levied is closest in meaning to:

imposed, believed, requested, correlated

Word Semantics

Representation

Semantic Representation: Taxonomy



Semantic Representation: Taxonomy

Common answer: Use a taxonomy like WordNet that has hypernyms (is-a) relationships and synonym sets (good):

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

```
S: (adj) full, good
S: (adj) estimable, good, honorable, respectable
S: (adj) beneficial, good
S: (adj) good, just, upright
S: (adj) adept, expert, good, practiced,
proficient, skillful
S: (adj) dear, good, near
S: (adj) good, right, ripe
…
S: (adv) well, good
S: (adv) thoroughly, soundly, good
S: (n) good, goodness
S: (n) commodity, trade good, good
```

Using a Taxonomy for Semantics: Problems

- Great as a resource, but missing nuances, e.g., synonyms:
  - *adept, expert, good, practiced, proficient, skillful?*
- Missing new words (impossible to keep up to date):
  - *wicked, nifty, crack, wizard, genius, Brexit, COVID-19*
- Subjective
- Requires human labor to create and adapt
- Hard to compute accurate word similarity

Readily implemented similarities using WordNet by Ted Pedersen:
http://wn-similarity.sourceforge.net/

Using a Taxonomy for Semantics: Problems

The vast majority of rule-based **and** statistical NLP work regards words as atomic symbols: hotel, conference, walk

In vector space terms, this is a vector with one 1 and a lot of zeroes

[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

Dimensionality: 20K (speech) – 50K (PTB) – 500K (big vocab) – 13M (Google 1T)

We call this a "one-hot" representation. Its problem:

What is the problem with 1-hot representation?

Using a Taxonomy for Semantics: Problems

The vast majority of rule-based **and** statistical NLP work regards words as atomic symbols: hotel, conference, walk

In vector space terms, this is a vector with one 1 and a lot of zeroes

[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

Dimensionality: 20K (speech) – 50K (PTB) – 500K (big vocab) – 13M (Google 1T)

We call this a "one-hot" representation. Its problem:

motel [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0] AND
hotel [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0] = 0

Distributional Models of Meaning (= Vector-Space Models of Meaning) (= Vector Semantics)

Intuition: Zellig Harris (1954):

- "*oculist* and *eye-doctor* … occur in almost the same environments"
- "If *A* and *B* have almost identical environments, we say that they are synonyms."

Do you see a problem with this intuition?

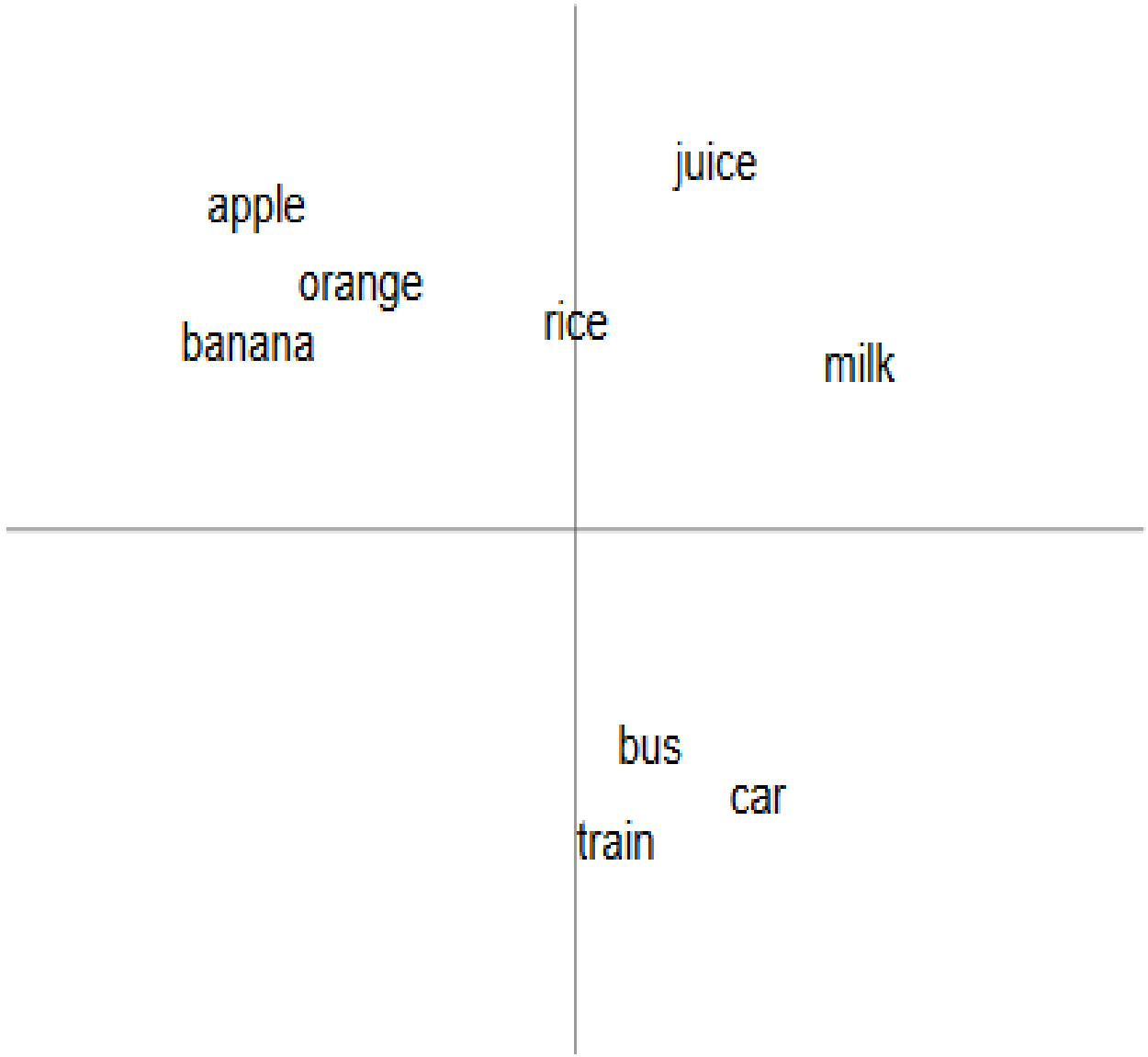
Distributional Models of Meaning (= Vector-Space Models of Meaning) (= Vector Semantics)

Intuition: Zellig Harris (1954):

- "*oculist* and *eye-doctor* … occur in almost the same environments"
- "If A and B have almost identical environments, we say that they are synonyms."

Problem: antonyms also occur in similar contexts!

You Shall Know a Word by the Company It Keeps



Distributed Representation for Semantics

You can get a lot of value by representing a word by means of its neighbors

"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

government debt problems turning into **banking** crises as has happened in

saying that Europe needs unified **banking** regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

Distributional Word Similarity

- Example: Suppose I asked you what is ***tesgüino****?*

A bottle of ***tesgüino*** is on the table
Everybody likes ***tesgüino***
Tesgüino makes you drunk
We make ***tesgüino*** out of corn.

What do you think it is?

Distributional Word Similarity

- Example: Suppose I asked you what is ***tesgüino****?*

> A bottle of ***tesgüino*** is on the table
> Everybody likes ***tesgüino***
> ***Tesgüino*** makes you drunk
> We make ***tesgüino*** out of corn.

- From context words humans can guess ***tesgüino*** means
  - an alcoholic beverage like beer
- Intuition for an algorithm:
  - **Two words are similar if they have similar word contexts.**

Types of Vectors Models

Sparse vector representations

1. Word co-occurrence matrices: one-hot

Dense vector representations:

2. Brown clusters
3. Singular value decomposition (e.g., LSA)
4. Derived from a neural network (e.g., word2vec)

Word Semantics

Word Co-occurrence Vectors

Co-occurrence Matrices

- How often a word occurs in a document
  - **Term-document matrix**

- Or how often a word co-occurs with another word
  - **Term-term matrix**
    (or **word-word co-occurrence matrix**
    or **word-context matrix**)

Documents in a Term-Document Matrix

- Each cell: count of word *w* in a document *d*:
  - Each document is a count vector in $\mathbb{N}^{|V|}$: a column below

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

Document Similarity in Term-Document Matrices

Two documents are similar if their vectors are similar

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

Words in a Term-Document Matrix

Each word is a count vector in $\mathbb{N}^D$: a row below

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

Word Similarity in a Term-Document Matrix

- Two **words** are similar if their vectors are similar

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

Words in a Term-Context Matrix

- Instead of entire documents, use smaller contexts
  - Paragraph
  - Window of $\pm$ 4 words
- A word is a vector over counts of context words
- Instead of each vector being of length D
- Each vector is now of length $|V|$
- The word-word matrix is $|V| \text{x} |V|$

Word-Word Matrix: Sample Contexts of 7 words

sugar, a sliced lemon, a tablespoonful of **apricot** preserve or jam, a pinch each of,
their enjoyment. Cautiously she sampled her first **pineapple** and another fruit whose taste she likened
well suited to programming on the digital **computer**. In finding the optimal R-stage policy from
for the purpose of gathering data and **information** necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar	…
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	
…	…						

Word-Word Matrix: Sample Contexts of 7 words

sugar, a sliced lemon, a tablespoonful of	**apricot**	preserve or jam, a pinch each of,
their enjoyment. Cautiously she sampled her first	**pineapple**	and another fruit whose taste she likened
well suited to programming on the digital	**computer**.	In finding the optimal R-stage policy from
for the purpose of gathering data and	**information**	necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar	...
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	
...	...						

Word-Word Matrix: Sample Contexts of 7 words

sugar, a sliced lemon, a tablespoonful of **apricot** preserve or jam, a pinch each of,
their enjoyment. Cautiously she sampled her first **pineapple** and another fruit whose taste she likened
well suited to programming on the digital **computer**. In finding the optimal R-stage policy from
for the purpose of gathering data and **information** necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar	...
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	
...	...						

What do you notice?

Word-Word Matrix: Sample Contexts of 7 words

sugar, a sliced lemon, a tablespoonful of **apricot** preserve or jam, a pinch each of,
their enjoyment. Cautiously she sampled her first **pineapple** and another fruit whose taste she likened
well suited to programming on the digital **computer**. In finding the optimal R-stage policy from
for the purpose of gathering data and **information** necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar	...
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	
...	...						

Word-Word Matrix: Sample Contexts of 7 words

sugar, a sliced lemon, a tablespoonful of **apricot** preserve or jam, a pinch each of,
their enjoyment. Cautiously she sampled her first **pineapple** and another fruit whose taste she likened
well suited to programming on the digital **computer**. In finding the optimal R-stage policy from
for the purpose of gathering data and **information** necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar	...
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	
...	...						

Word-Word Matrix: Sparseness

- We only showed 4x6, but the real matrix is 50,000 x 50,000
  - It is very **sparse**
    - i.e., most values are 0.
  - That is OK, since there are lots of efficient algorithms for sparse matrices.
- The size of the windows depends on your goals
  - The shorter the windows, the more **syntactic** the representation

    1-3 words: more syntax
  - The longer the windows, the more **semantic** the representation

    4-10 words: more semantics

Why?

Types of Word Co-occurrence

- First-order co-occurrence (**syntagmatic association**):
  - they are typically nearby each other
  - *wrote* is a first-order associate of *book* or *poem*

- Second-order co-occurrence (**paradigmatic association**):
  - they have similar neighbors
  - *wrote* is a second-order associate of words like *said* or *remarked*

Measuring Similarity: Dot Product

- Given two words *v* and *w*
- We will need a way to measure their similarity
- Most measures of vector similarity are based on the

  Dot product or **inner product** from linear algebra

$$\text{dot-product}(\vec{v},\vec{w}) = \vec{v}\cdot\vec{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

Do you see a problem with using dot product for measuring semantic similarity?

Measuring Similarity: Problem with Dot Product

$$\text{dot-product}(\vec{v},\vec{w}) = \vec{v}\cdot\vec{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

- **Dot product is longer if the vector is longer.** Vector length:

$$|\vec{v}| = \sqrt{\sum_{i=1}^{N} v_i^2}$$

- Vectors are longer if they have higher values in each dimension
- That means more frequent words will have higher dot products
- That is bad: a similarity measure should not be sensitive to word frequency

How can we fix this?

Solution: Cosine as a Similarity Measure

- Just divide the dot product by the length of the two vectors!

$$\frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|}$$

- This turns out to be the cosine of the angle between them!

$$\vec{a} \cdot \vec{b} = |\vec{a}||\vec{b}| \cos\theta$$

$$\frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|} = \cos\theta$$

Cosine for Computing Similarity

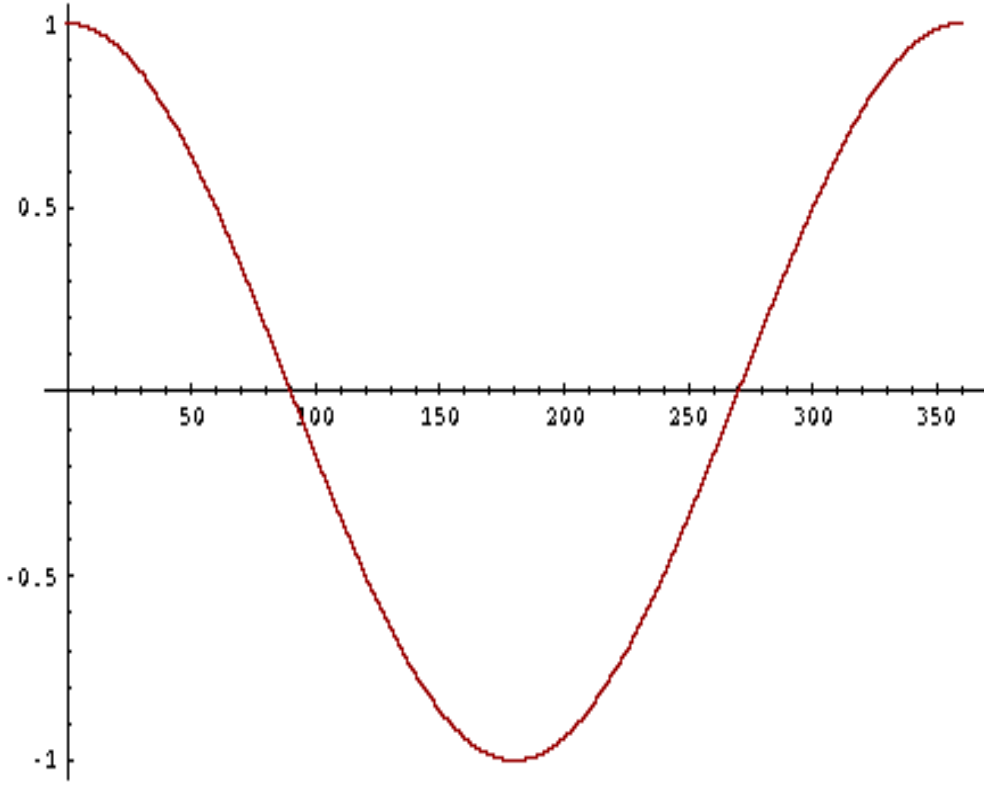
Dot product

Unit vectors

$$\cos(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\vec{v}}{|\vec{v}|}\cdot\frac{\vec{w}}{|\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

Cosine as a Similarity Measure

- -1: vectors point in opposite directions
- +1: vectors point in same directions
- 0: vectors are orthogonal



- Raw frequency or PPMI are non-negative, so cosine ranges in 0-1

$$\cos(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\vec{v}}{|\vec{v}|}\cdot\frac{\vec{w}}{|\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

	large	data	computer
apricot	2	0	0
digital	0	1	2
information	1	6	1

Which pair of words is more similar?

cosine(apricot,information) =

$$\frac{2+0+0}{\sqrt{4+0+0}\sqrt{1+36+1}} = \frac{2}{\sqrt{4}\sqrt{38}} = .16$$

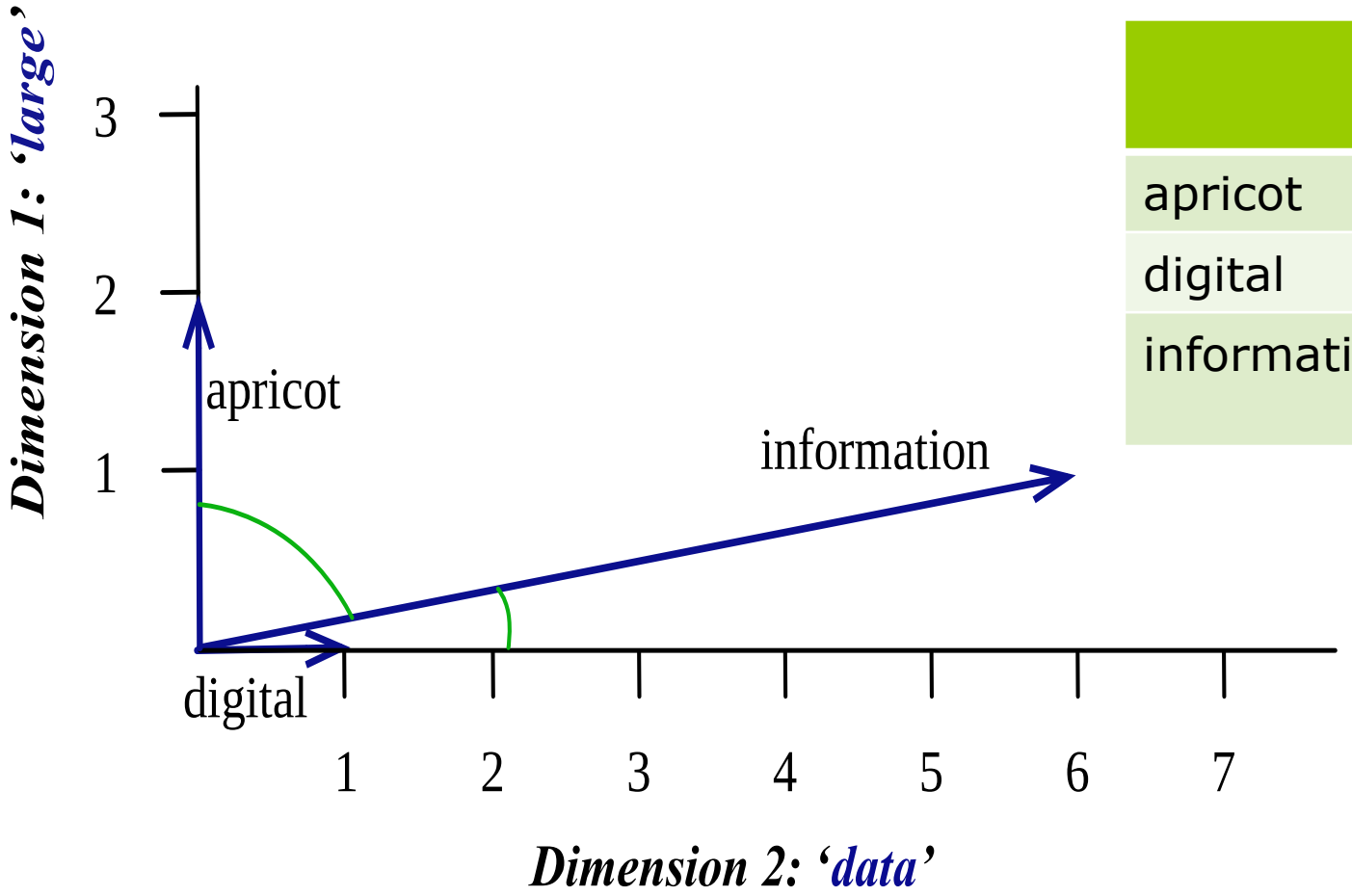
cosine(digital,information) =

cosine(apricot,digital) =

$$\frac{0+6+2}{\sqrt{0+1+4}\sqrt{1+36+1}} = \frac{8}{\sqrt{38}\sqrt{5}} = .58$$

$$\frac{0+0+0}{\sqrt{1+0+0}\sqrt{0+1+4}} = 0$$

Visualizing Vectors and Angles



	large	data
apricot	2	0
digital	0	1
information	1	6

Other Similarity Measures

$$\text{sim}_{\text{cosine}}(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i \times w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

$$\text{sim}_{\text{Jaccard}}(\vec{v},\vec{w}) = \frac{\sum_{i=1}^{N} \min(v_i,w_i)}{\sum_{i=1}^{N} \max(v_i,w_i)}$$

$$\text{sim}_{\text{Dice}}(\vec{v},\vec{w}) = \frac{2\times\sum_{i=1}^{N} \min(v_i,w_i)}{\sum_{i=1}^{N} (v_i+w_i)}$$

$$\text{sim}_{\text{JS}}(\vec{v}||\vec{w}) = D(\vec{v}|\frac{\vec{v}+\vec{w}}{2}) + D(\vec{w}|\frac{\vec{v}+\vec{w}}{2})$$

Word Semantics

Weighting

Term Weights: Term Frequency

- More frequent terms in a document are more important, i.e., more indicative of the topic.

  f_{ij} = frequency of term *i* in document *j*

- May want to normalize *term frequency* (*tf*) by dividing by the frequency of the most common term in the document:

  $tf_{ij} = f_{ij} / max_i\{f_{ij}\}$

Term Weights: Inverse Document Frequency

- Terms that appear in many *different* documents are *less* indicative of the overall topic.

  df_i = document frequency of term i

  = number of documents containing term i

  $\mathbf{idf_i}$ = inverse document frequency of term i,

  $\mathbf{= \log_2 (N / df_i)}$

  (N: total number of documents)

- An indication of a term's *discrimination* power.
- Log used to dampen the effect relative to *tf*.

TF.IDF Weighting

- A typical combined term importance indicator is *tf-idf weighting*:

$$w_{ij} = tf_{ij} \times idf_i = tf_{ij} \log_2 (N / df_i)$$

- A term occurring frequently in the document but rarely in the rest of the collection is given high weight.

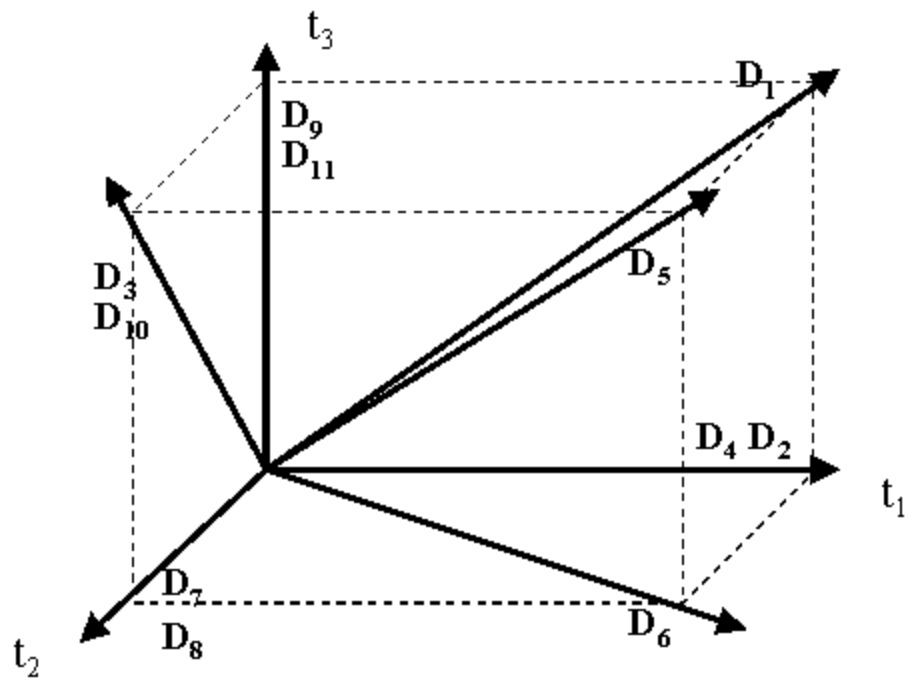
Vector Space Model

Sample Text

<BODY>Shr 34 cts vs 1.19 dlrs
    Net 807,000 vs 2,858,000
    Assets 510.2 mln vs 479.7 mln
    Deposits 472.3 mln vs 440.3 mln
    Loans 299.2 mln vs 327.2 mln
    Note: 4th qtr not available. Year includes 1985 extraordinary gain from tax carry forward of 132,000 dlrs, or five cts per shr.
 Reuter
</BODY></TEXT>

The Vector-Space Model

vs	$\vec{x} =$	5
mln		5
cts		3
;		3
&		3
000		4
loss		0
’		0
"		0
3		4
profit		0
dlrs		3
1		2
pct		0
is		0
s		0
that		0
net		3
lt		2
at		0



$$\cos\theta = \frac{\mathbf{X} \bullet \mathbf{Y}}{\|\mathbf{X}\| * \|\mathbf{Y}\|}$$

Word Semantics

Dense Vectors

Sparse vs. Dense Vectors

- Frequency vectors are
  - **long** (length $|V| = 20{,}000$ to $50{,}000$)
  - **sparse** (most elements are zero)

- Alternative: learn vectors which are
  - **short** (length 200-1000)
  - **dense** (most elements are non-zero)

Sparse vs. Dense Vectors

- **Why dense vectors?**
  - Short vectors may be easier to use as features in machine learning (less weights to tune)
  - Dense vectors may generalize better than storing explicit counts
  - They may do better at capturing synonymy:
    - *car* and *automobile* are synonyms
      - but are represented as distinct dimensions
      - this fails to capture the similarity between a word with *car* as a neighbor and a word with *automobile* as a neighbor

Getting Short Dense Vectors

- Brown clustering
- Singular Value Decomposition (SVD)
  - A special case of this is called LSA – Latent Semantic Analysis
- "Neural Language Model"-inspired predictive models
  - skip-grams and CBOW

Word Semantics

Dense Vectors: Brown Clustering

Brown Clustering

- An agglomerative clustering algorithm that clusters words based on which words precede or follow them

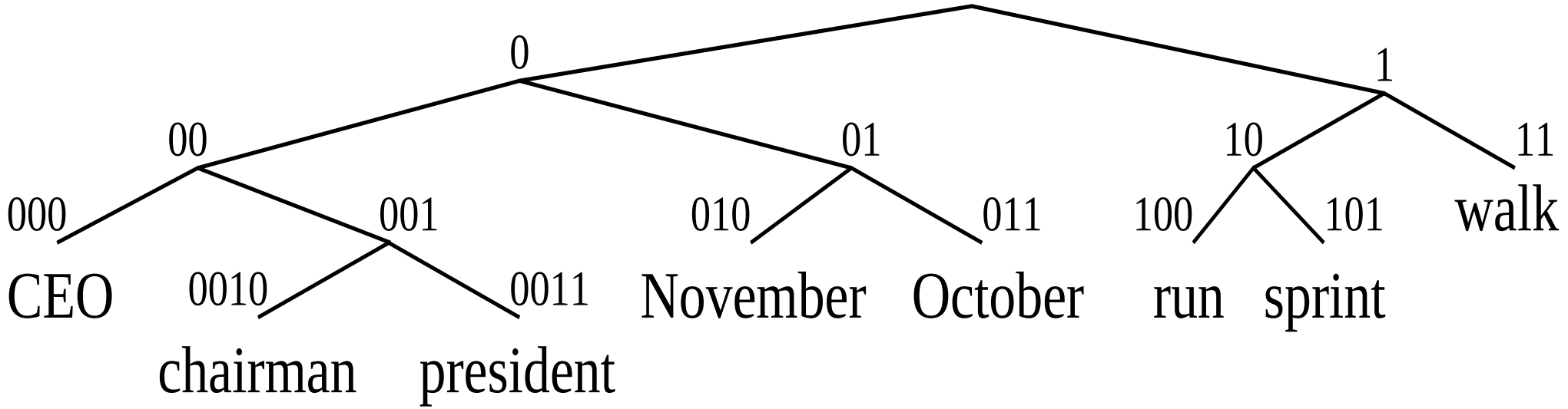
- These word clusters can be turned into a kind of vector

Brown Clustering Algorithm

- Each word is initially assigned to its own cluster
- We now consider merging each pair of clusters The highest quality merge is chosen
  - Quality = merges two words that have similar probabilities of preceding and following words
  - (More technically quality = smallest decrease in the likelihood of the corpus according to a class-based language model.)
- Clustering proceeds until all words are in one big cluster

Brown Clusters as Vectors

- By tracing the order in which clusters are merged, the model builds a binary tree from bottom to top.
- Each word represented by binary string = path from root to leaf
- Each intermediate node is a cluster
- Chairman is 0010, "months" = 01, and verbs = 1



Brown Cluster Examples

Friday Monday Thursday Wednesday Tuesday Saturday Sunday weekends Sundays Saturdays
June March July April January December October November September August
pressure temperature permeability density porosity stress velocity viscosity gravity tension
anyone someone anybody somebody
had hadn't hath would've could've should've must've might've
asking telling wondering instructing informing kidding reminding bothering thanking deposing
mother wife father son husband brother daughter sister boss uncle
great big vast sudden mere sheer gigantic lifelong scant colossal
down backwards ashore sideways southward northward overboard aloft downwards adrift

Class-Based Language Model

- Suppose each word was in some class c_i:

$$P(w_i|w_{i-1}) = P(c_i|c_{i-1})P(w_i|c_i)$$

$$P(\text{corpus}|C) = \prod_{i-1}^{n} P(c_i|c_{i-1})P(w_i|c_i)$$

Word Semantics

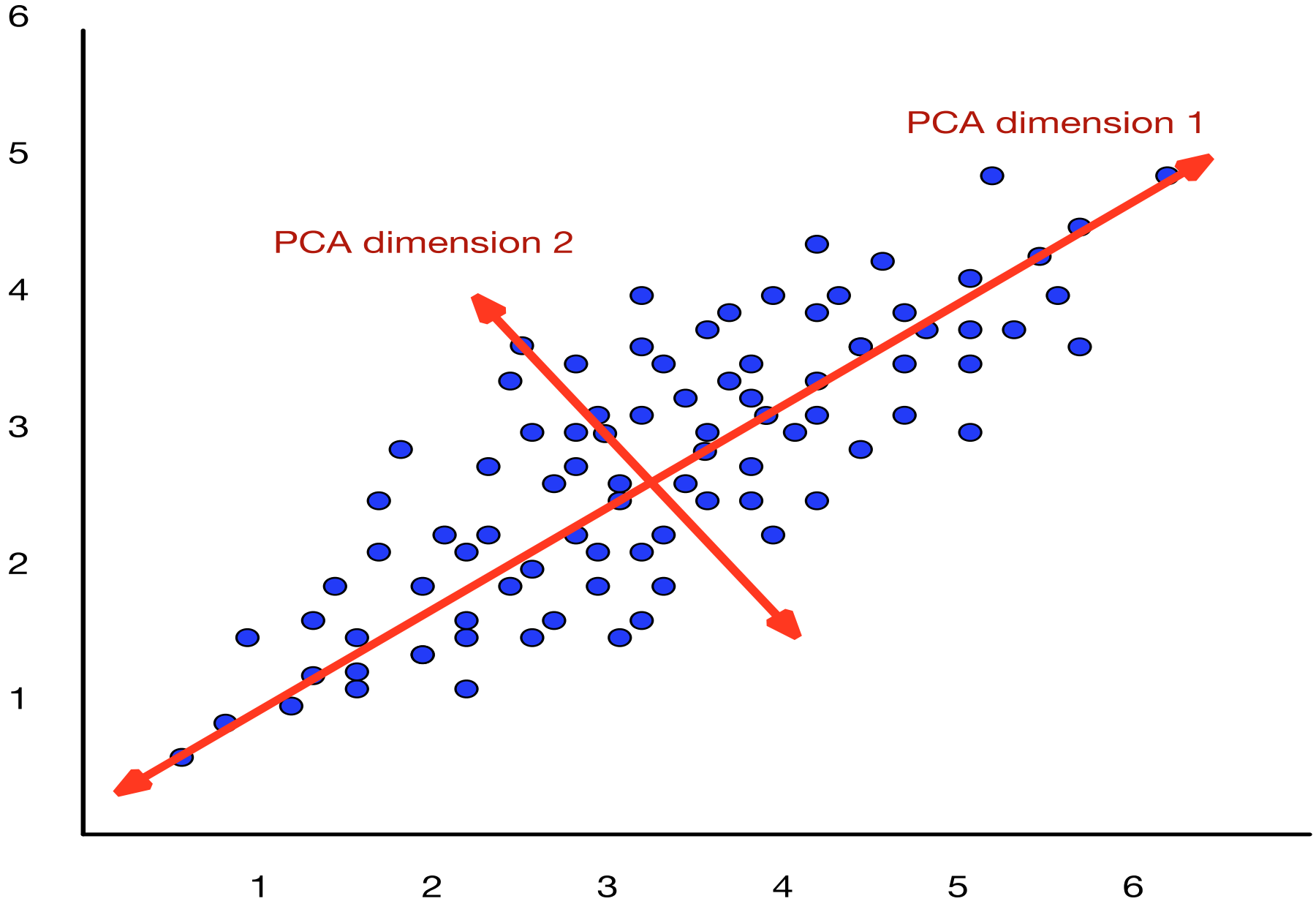
LSA: Embeddings from SVD

Intuition

- Approximate an *N*-dimensional dataset using fewer dimensions
  - by first rotating the axes into a new space
    - in which the highest order dimension captures the most variance in the original dataset
    - and the next dimension captures the next most variance, etc.

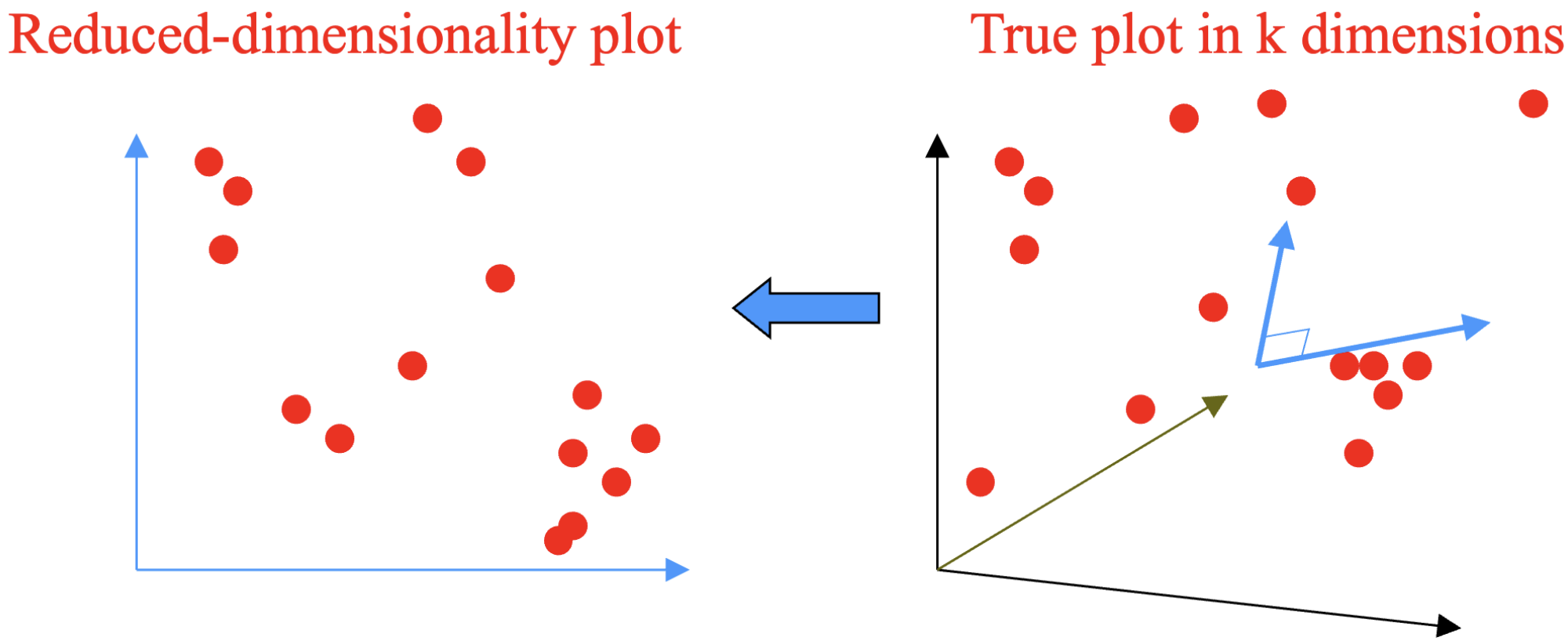
- Many such (related) methods:
  - PCA – principal components analysis
  - Factor Analysis
  - SVD – singular value decomposition

Dimensionality Reduction with PCA



SVD: Geometry

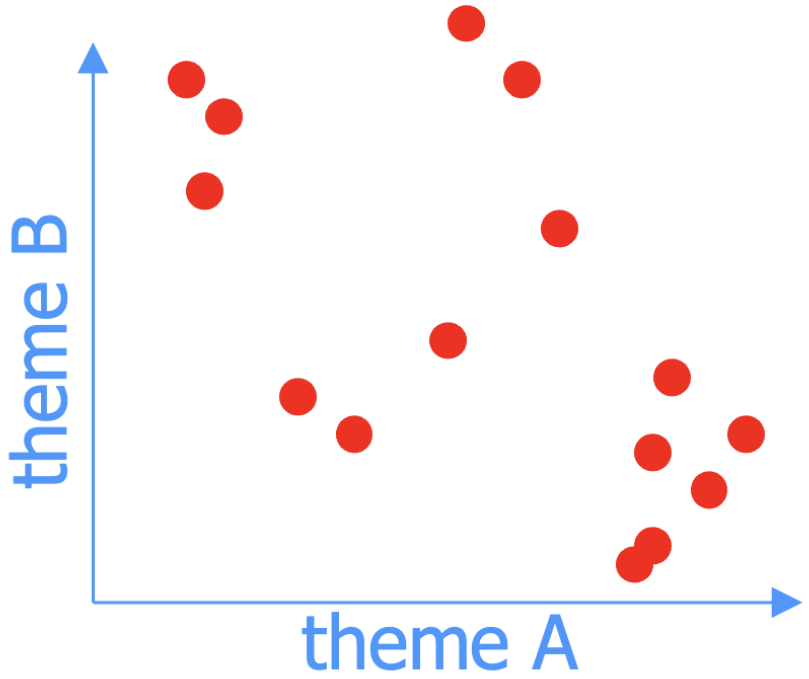
- Reduced plot is a perspective drawing of true plot
- It projects true plot onto a few axes
- ∃ a best choice of axes – shows most variation in the data.
  - Found by linear algebra: "Singular Value Decomposition" (SVD)



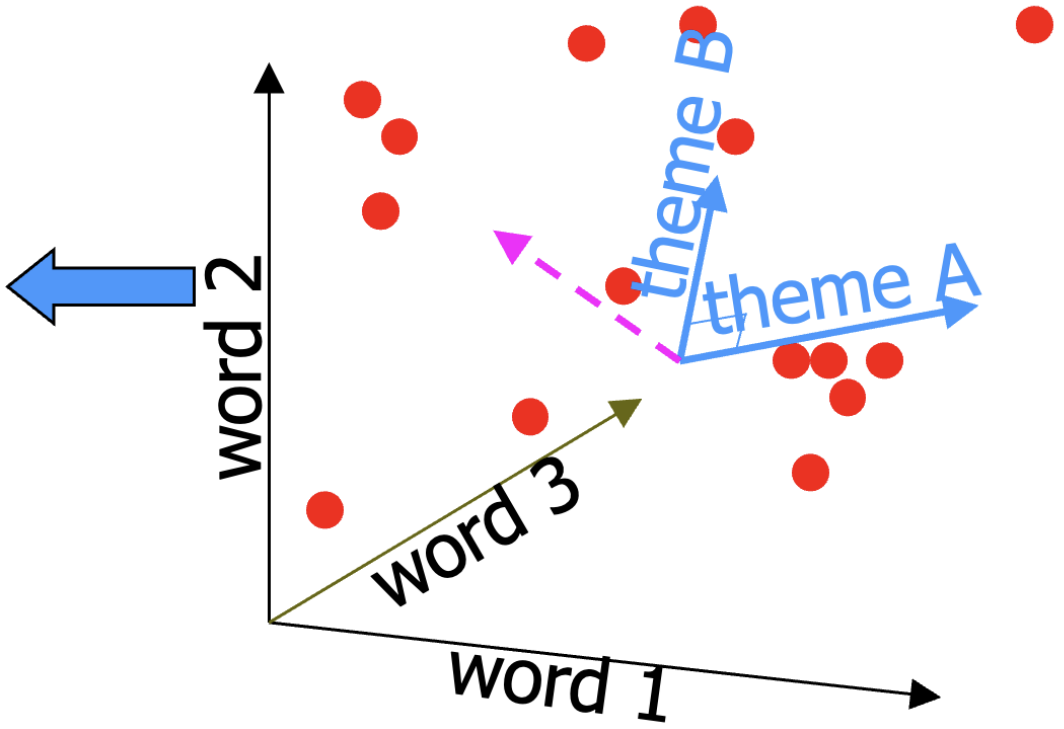
SVD: Geometry

- SVD plot allows best possible reconstruction of true plot
  (i.e., can recover 3-D coordinates with minimal distortion)
- Ignores variation in the axes that it did not pick out
- Hope that variation is just noise and we **want** to ignore it

Reduced-dimensionality plot



True plot in k dimensions



LSA: Intuition

- Implements the idea that the meaning of a passage is the sum of the meanings of its words:

  meaning of $word_1$ + meaning of $word_2$ + … + meaning of $word_n$ = meaning of passage

- This "bag of words" function shows that a passage is considered to be an unordered set of word tokens and the meanings are additive.

LSA: Intuition

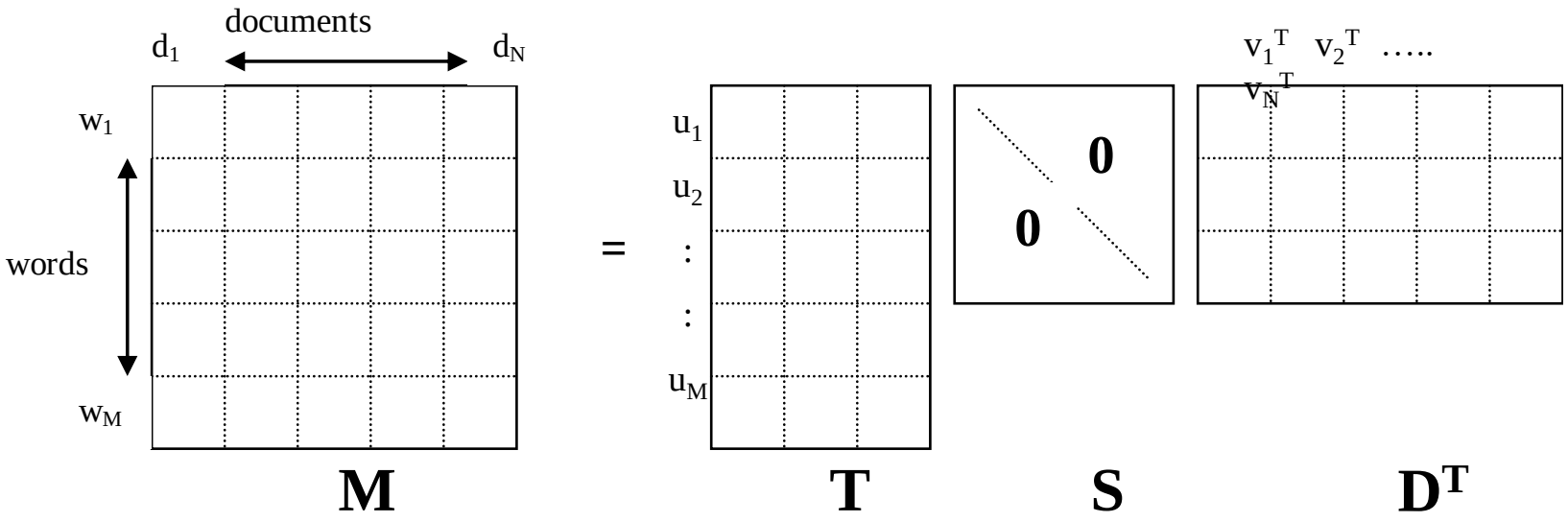
- Implements the idea that the meaning of a passage is the sum of the meanings of its words:

  meaning of $word_1$ + meaning of $word_2$ + … + meaning of $word_n$ = meaning of passage

- This "bag of words" function shows that a passage is considered to be an unordered set of word tokens and the meanings are additive.

Does this hold for the sparse vector-space model?

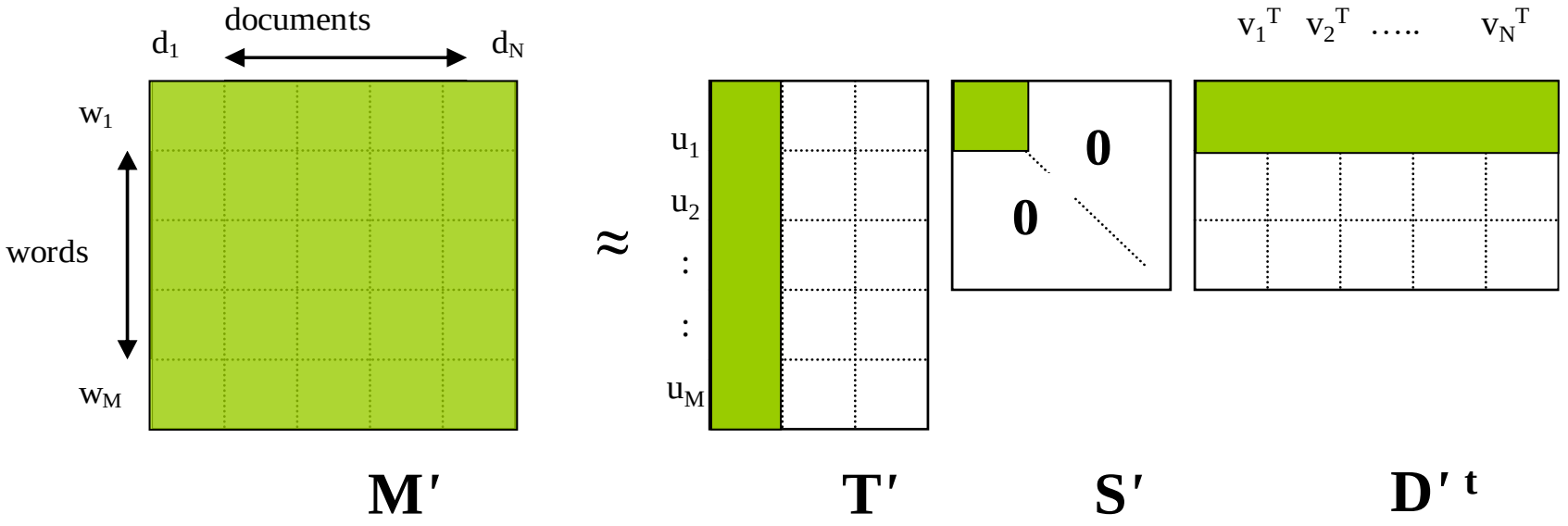
Singular Value Decomposition



T, D: orthogonal

S: diagonal

Matrix Truncation



M′ is the best least square approximation of **M** by a matrix of rank *k*

T′, D′: orthogonal

S′: diagonal

Comparing Two Documents

$$
\begin{aligned}
M^T M &= (TSD^T)^T (TSD^T) \\
&= DST^T TSD^T \\
&= DSSD^T \\
&= (DS)(DS)^T
\end{aligned}
$$

DS is a matrix of **document** vectors

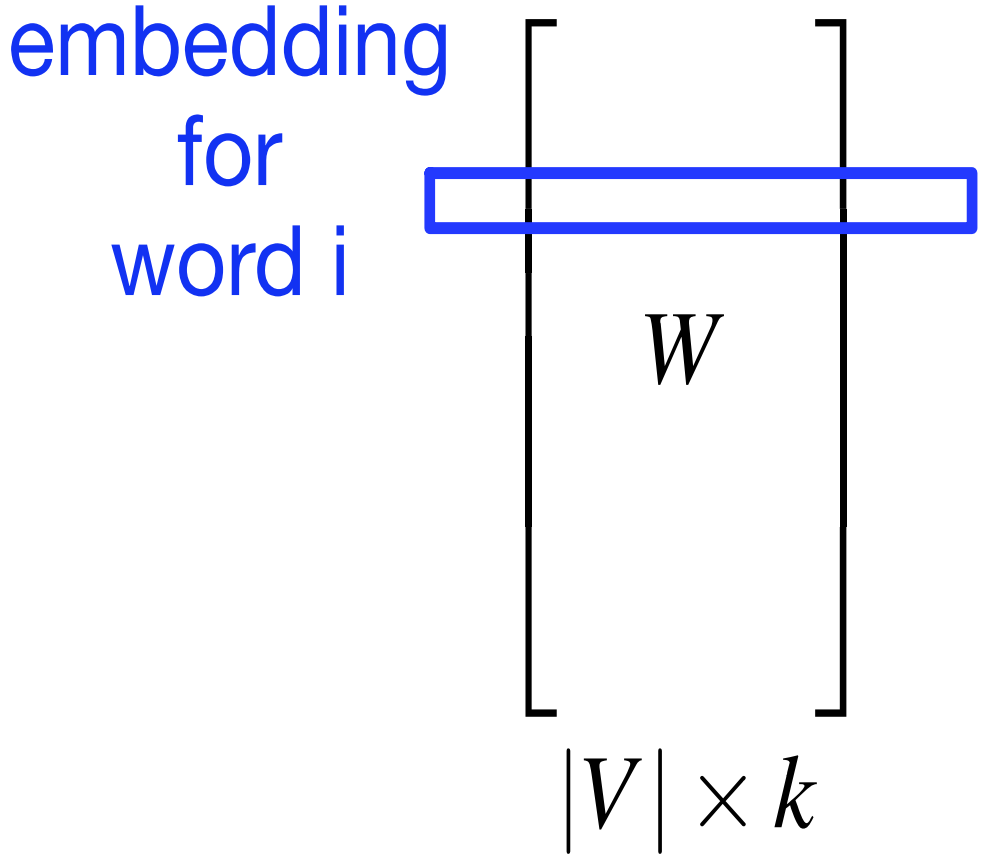
Comparing Two Terms

$$
\begin{aligned}
MM^T &= (TSD^T)(TSD^T)^T \\
&= TSD^T DST^T \\
&= TSST^T \\
&= (TS)(TS)^T
\end{aligned}
$$

TS is a matrix of **term** vectors

Truncated SVD Yields Embeddings

- Each row of the *W* matrix is a *k*-dimensional representation of each word *w*



Embeddings vs. Sparse Vectors

- Dense SVD embeddings sometimes work better than sparse matrices at tasks like word similarity

Why is this?

Embeddings vs. Sparse Vectors

- Dense SVD embeddings sometimes work better than sparse matrices at tasks like word similarity
  - Denoising: low-order dimensions may represent unimportant information
  - Truncation may help the models generalize better to unseen data.
  - Having a smaller number of dimensions may make it easier for classifiers to properly weigh the dimensions for the task.
  - Dense models may do better at capturing higher-order co-occurrences.

Representing Text as a Matrix

documents

words	d1	d2	d3	d4	d5	d6
cosmonaut	1	0	1	0	0	0
astronaut	0	1	0	0	0	0
moon	1	1	0	0	0	0
car	1	0	0	1	1	0
truck	0	0	0	1	0	1

M[i,j] = number of occurrence of a word **i** in document **j**

Matrix **T**

T- term matrix. Rows of **T** correspond to Rows of the original matrix **M**

dimensions

words

	dim1	dim2	dim3	dim4	dim5
cosmonaut	**-0.44**	**-0.30**	0.57	0.58	0.25
astronaut	**-0.13**	**-0.33**	-0.59	0.00	0.73
moon	**-0.48**	**-0.51**	-0.37	0.00	-0.61
car	**-0.70**	**0.35**	0.15	-0.58	0.16
truck	**-0.26**	**0.65**	-0.41	0.58	-0.09

Dim2 directly reflects the different co-occurrence patterns

Matrix $\mathbf{D}^t$

D – document matrix. Columns of $\mathbf{D}^t$ (rows of **D**) correspond to rows of the original matrix **M**

documents

dimensions	d1	d2	d3	d4	d5	d6
Dimension1	**-0.75**	**-0.28**	**-0.20**	**-0.45**	**-0.33**	**-0.12**
Dimension2	**-0.29**	**-0.53**	**-0.19**	**0.63**	**0.22**	**0.41**
Dimension3	0.28	-0.75	0.45	-0.20	0.12	-0.33
Dimension4	-0.00	0.00	0.58	0.00	-0.58	0.58
Dimension5	-0.53	0.29	0.63	0.19	0.41	-0.22

Dim2 directly reflects the different co-occurrence patterns

Reevaluating Document Similarities

- B = D′ x S′
- B is a dimensionality reduction of the original matrix **M**
- Compute **document correlation B'B**

	d1	d2	d3	d4	d5	d6
d1	1					
d2	0.78	1				
d3	0.40	0.88	1			
d4	0.47	-0.18	-0.62	1		
d5	0.74	0.16	-0.32	0.94	1	
d6	0.10	-0.54	-0.87	0.93	0.74	1

documents 5 & 6 are similar but documents 3 & 4 are not

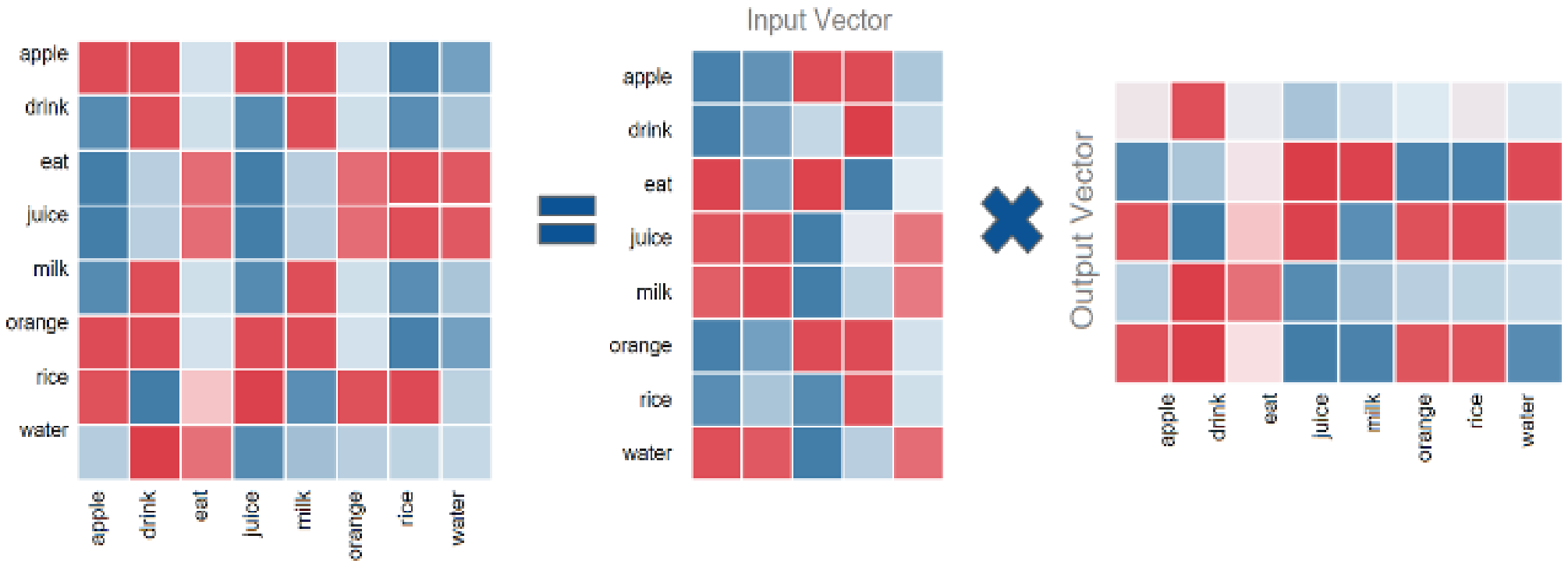
Types of Word Co-occurrences (Again)

- First-order co-occurrence (**syntagmatic association**):
  - typically nearby each other

- Second-order co-occurrence (**paradigmatic association**):
  - have similar neighbors

- Higher-order co-occurrence **(in LSA/embeddings)**
  - neighbors of neighbors, etc.

SVD for word-context matrices



The problem is that we may end up with matrices with huge number of rows and columns, which makes SVD computationally restrictive!

Word Semantics

Word2Vec

Word2Vec

- Popular embedding method
- Very fast to train
- Idea: **predict** rather than **count**

Word2Vec

- Instead of **counting** how often each word *w* occurs near "*apricot*"
- Train a classifier on a binary **prediction** task:
  - Is *w* likely to show up near "*apricot*"?
- We do not actually care about this task
  - but we will take the learned classifier weights as the word embeddings

Use Running Text as Implicitly Supervised Training Data

- A word *s* near *apricot*
  - Acts as a gold 'correct answer' to the question
  - "Is word *w* likely to show up near *apricot*?"

- No need for hand-labeled supervision

- The idea comes from **neural language modeling**
  - Bengio et al. (2003)
  - Collobert et al. (2011)

Word2Vec: Skip-Gram

1. Treat the target word and a neighboring context word as positive examples.

2. Randomly sample other words in the lexicon to get negative samples

3. Use logistic regression to train a classifier to distinguish those two cases

4. Use the weights as the embeddings

Word2Vec: Skip-Gram

Training sentence:

... lemon, a tablespoon of apricot jam a pinch ...

c1 c2 target c3 c4

Assume the context words are those in +/- 2 word window

Skip-Gram's Goal

Given a tuple (t,c) = target, context

- (*apricot*, *jam*)
- (*apricot*, *aardvark*)

Calculate the probability that *c* is a real context word:

P(+|t,c)

$P(-|t,c) = 1-P(+|t,c)$

Skip-Gram: How to Compute p(+|*t*,*c*)?

Intuition:

- Words are likely to appear near similar words
- Model the similarity with dot-product
- Similarity(t,c) $\propto t \cdot c$

Do you see a problem with this?

Skip-Gram: How to Compute p(+|*t*,*c*)?

Intuition:

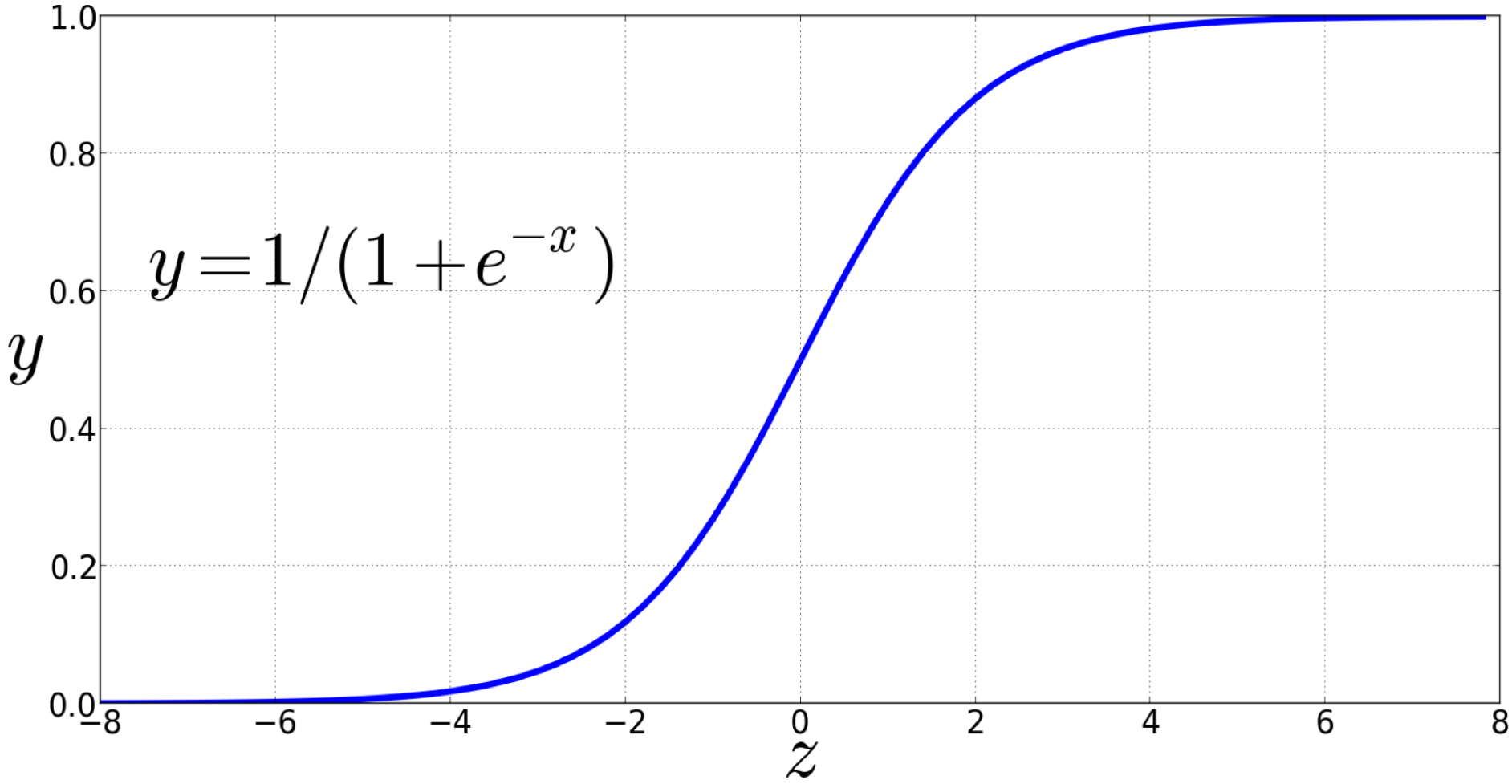
- Words are likely to appear near similar words
- Model the similarity with dot-product
- Similarity(t,c) $\propto t \cdot c$

Problem:

- *The dot product is not a probability!*
  - *(Neither is cosine)*

Turning a Dot Product into a Probability

The sigmoid lies between 0 and 1:



Turning a Dot Product into a Probability

$$P(+|t,c) = \frac{1}{1+e^{-t\cdot c}}$$

$$P(-|t,c) = 1 - P(+|t,c)$$

$$= \frac{e^{-t\cdot c}}{1+e^{-t\cdot c}}$$

Now, Consider All Context Words

Assuming that all context words are independent

$$P(+|t, c_{1:k}) = \prod_{i=1}^{\kappa} \frac{1}{1 + e^{-t \cdot c_i}}$$

$$\log P(+|t, c_{1:k}) = \sum_{i=1}^{k} \log \frac{1}{1 + e^{-t \cdot c_i}}$$

Skip-Gram Training Data

Training sentence:

... lemon, a tablespoon of apricot jam a pinch ...

c1 c2 t c3 c4

Training data: input/output pairs centering on *apricot*

Assume a +/- 2 word window

Skip-Gram Training Procedure

Training sentence:

... lemon, a tablespoon of apricot jam a pinch ...

c1 c2 t c3 c4

positive examples +

t	*c*
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

- For each positive example:
  - we create *k* negative examples
- Using *noise* words
  - any random word that is not *t*

Skip-Gram Training Procedure

Training sentence:

... lemon, a tablespoon of apricot jam a pinch ...

c1 c2 t c3 c4

k=2

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

negative examples -

t	c	t	c
apricot	aardvark	apricot	twelve
apricot	puddle	apricot	hello
apricot	where	apricot	dear
apricot	coaxial	apricot	forever

Skip-Gram Training: Choosing Noise Words

Could pick *w* according to their unigram frequency P(*w*)

More common to be chosen then according to $p_\alpha(w)$

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_w count(w)^\alpha}$$

α= ¾ works well because it gives rare noise words slightly higher probability

To show this, imagine two events p(a)=.99 and p(b) = .01:

$$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97$$

$$P_\alpha(b) = \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03$$

Skip-Gram Training Setup

- We represent words as vectors of some length (say 300), randomly initialized.

- Over the entire training set, we would like to adjust those word vectors such that we
  - **maximize** the similarity of the target word, context word pairs (*t*,*c*) drawn from the positive data
  - **minimize** the similarity of the (*t*,*c*) pairs drawn from the negative data.

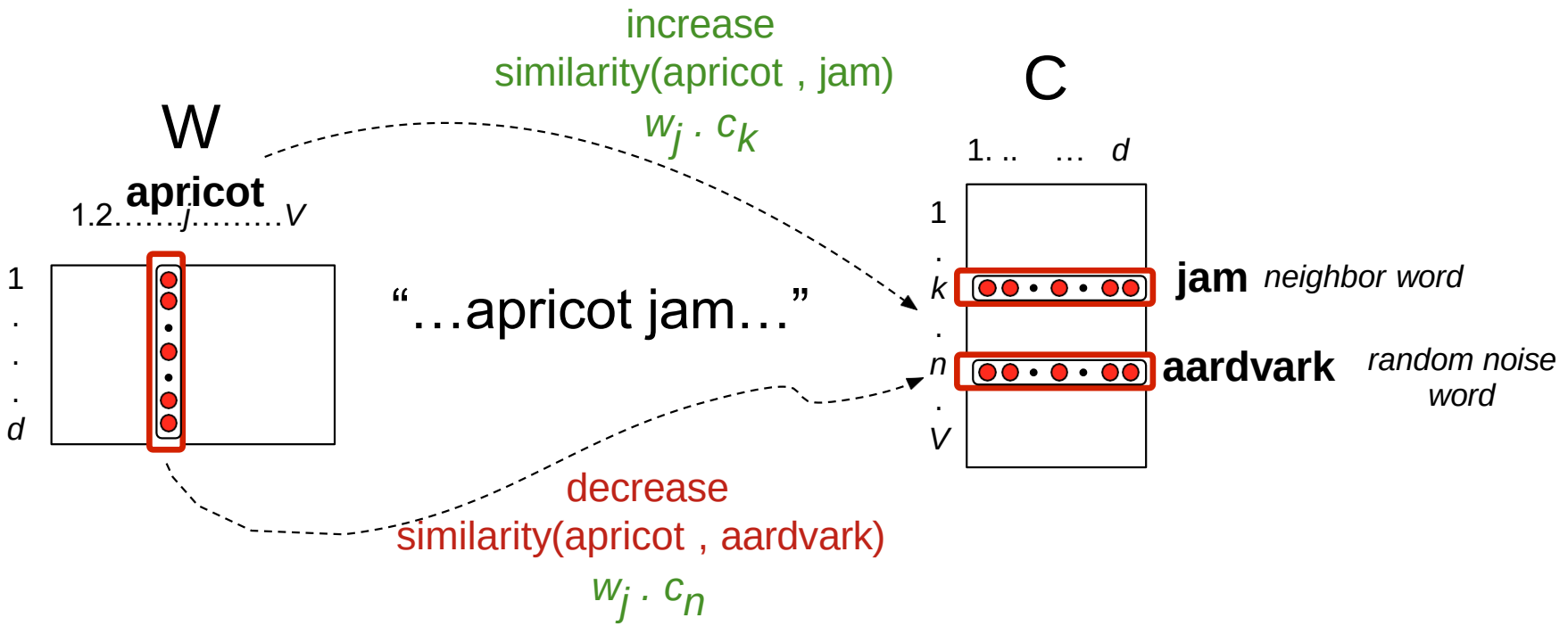
Skip-Gram: Training Procedure

Iterative process:

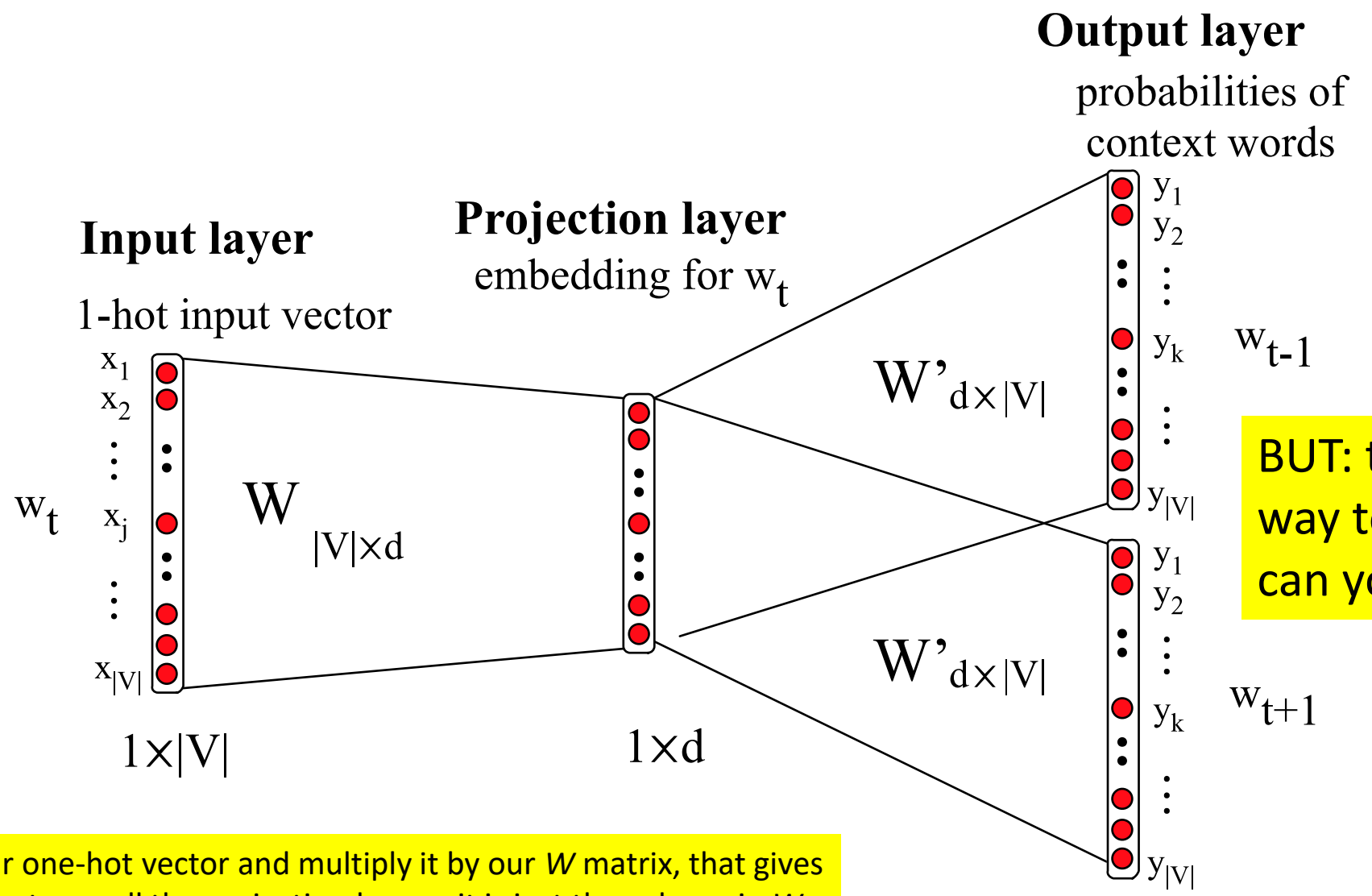
- Start with random vector weights
- Then repeatedly adjust the word weights:
  - make the positive pairs more likely
  - make the negative pairs less likely

$$\sum_{(t,c)\in+} logP(+|t,c) + \sum_{(t,c)\in-} logP(-|t,c)$$

Skip-Gram: Training Procedure



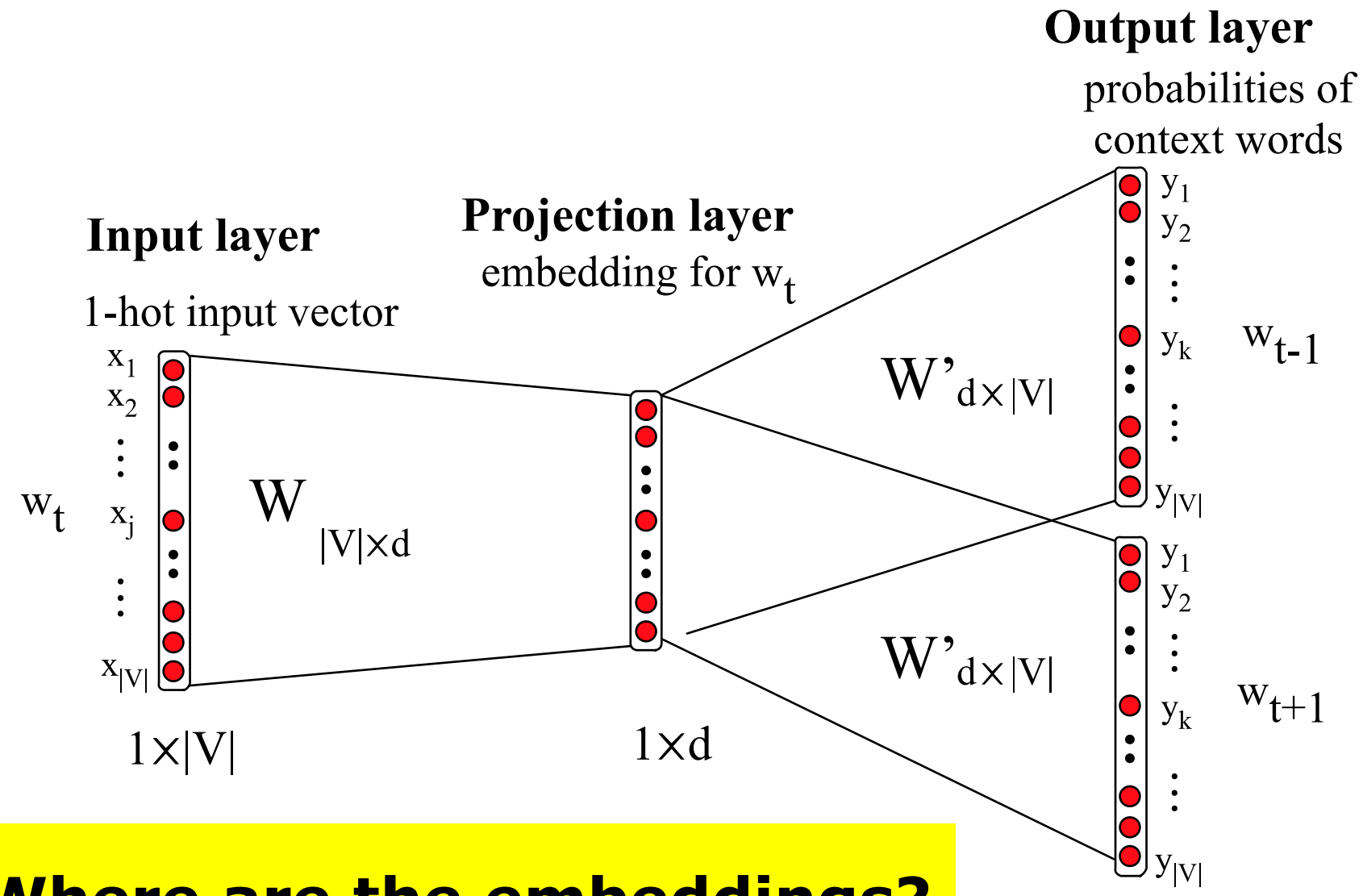
Skip-Gram: The Neural View



BUT: there is another way to get embeddings, can you see it?

If we take our one-hot vector and multiply it by our *W* matrix, that gives us a vector that we call the projection layer – it is just the column in *W* that corresponds to the word we are looking at.

Skip-Gram: The Neural View



Where are the embeddings?

Skip-Gram Learns Two Embeddings for Each Word *w*

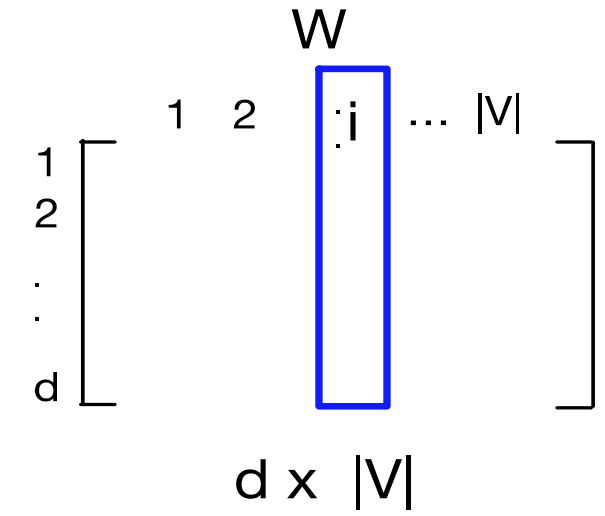
input embedding v_i in the input matrix W

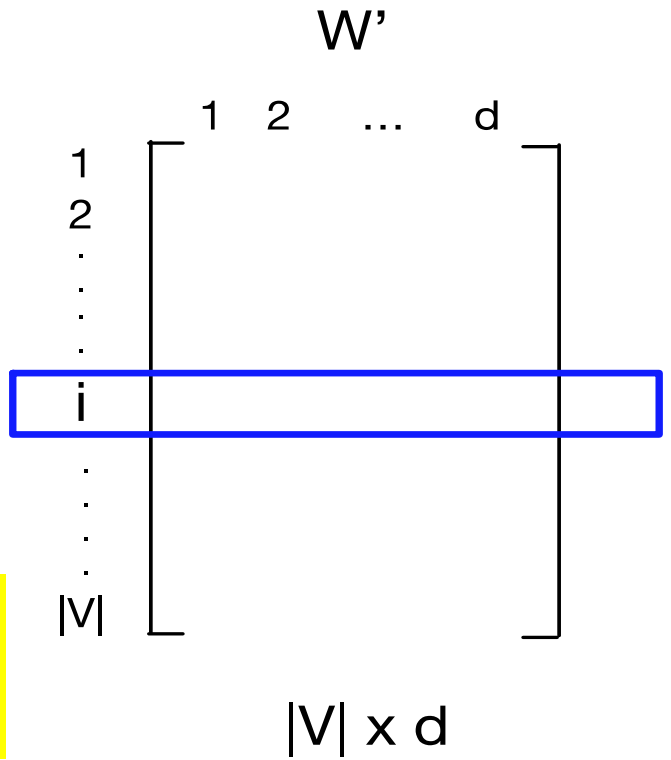
- Column i of the input matrix W is the $1 \times d$ embedding v_i for word i in the vocabulary.

output embedding v', in output matrix W'

- Row i of the output matrix W' is a $d \times 1$ vector embedding v'_i for word i in the vocabulary.

Which embedding should we use then?



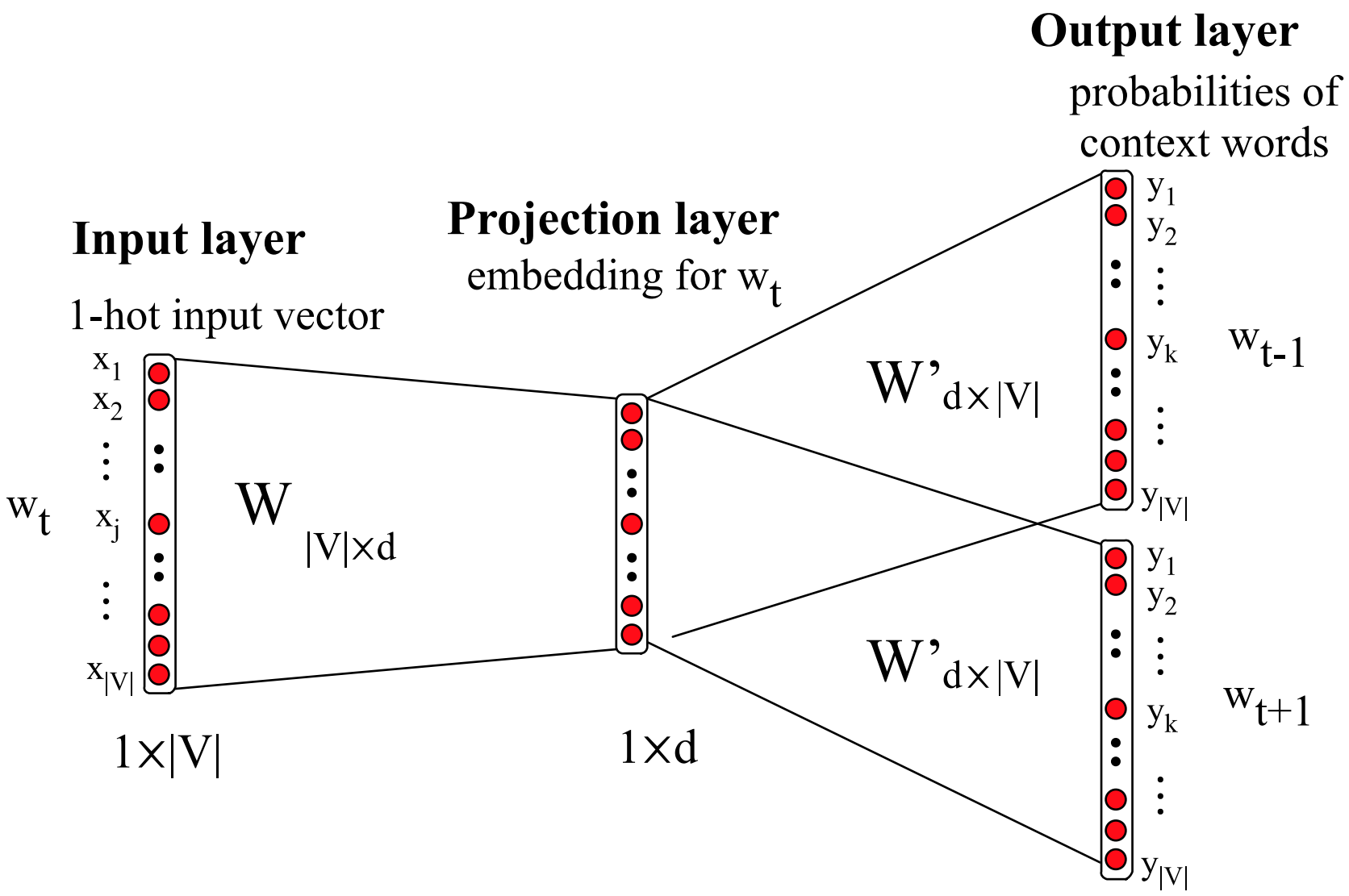


Embeddings from Skip-Gram and CBOW

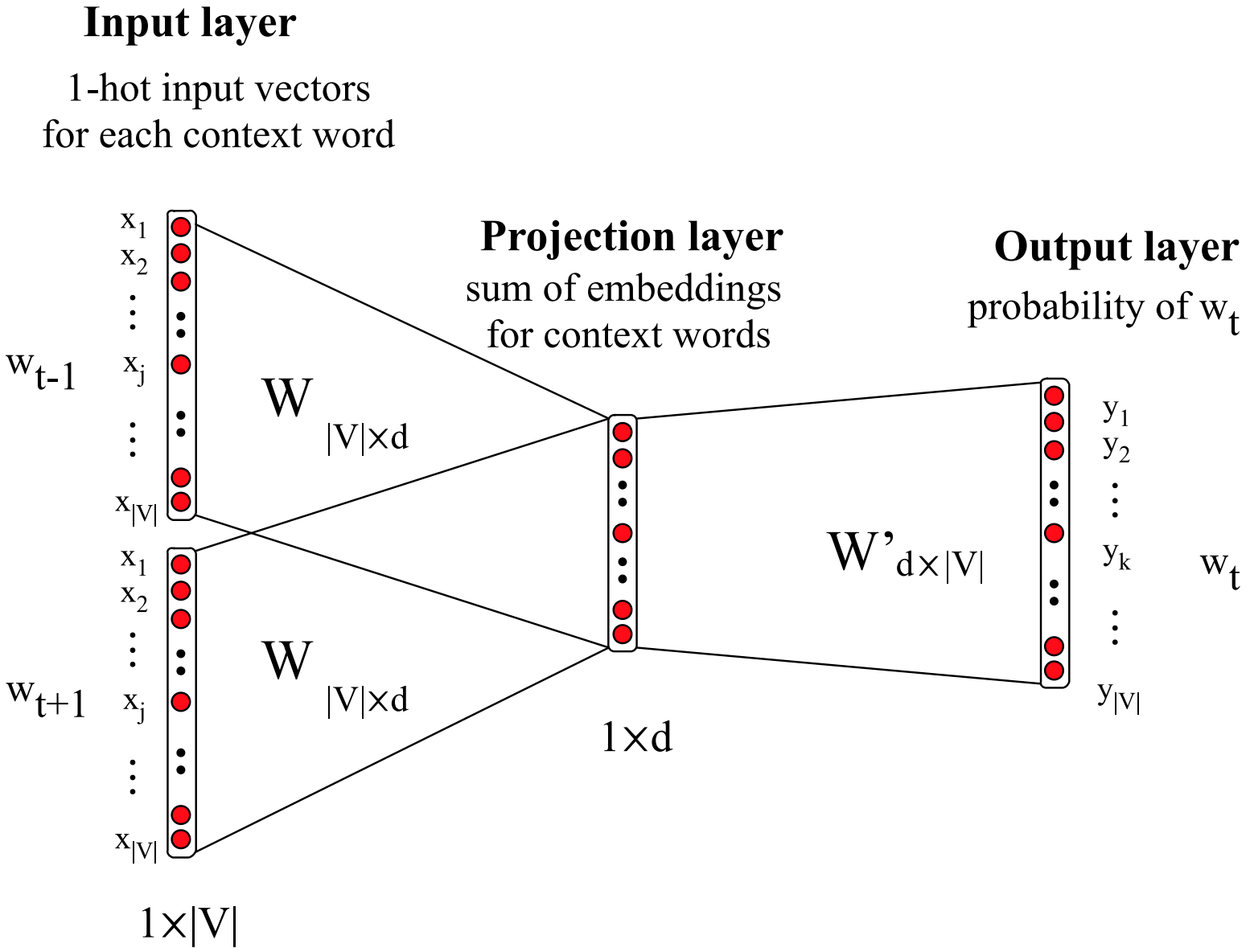
- Since we have two embeddings, v_j and v'_j for each word w_j

- We can
  - Just use v_j
  - Sum them
  - Concatenate them to make a double-length embedding

Skip-Gram (again)



CBOW: Continuous Bag of Words



Alternative (and More Precise?) Names

- **Skip-gram:** “predict context”
- **CBOW**: “predict word”

Evaluating Embeddings

Compare to human scores on word similarity tasks:

- WordSim-353 (Finkelstein et al., 2002)
- SimLex-999 (Hill et al., 2015)
- Stanford Contextual Word Similarity (SCWS) dataset (Huang et al., 2012)
- TOEFL dataset: *Levied is closest in meaning to: imposed, believed, requested, correlated*

Word2Vec

Properties

Skip-Gram: Vector Arithmetic

Inspired by analogy problems

Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

Word Analogies

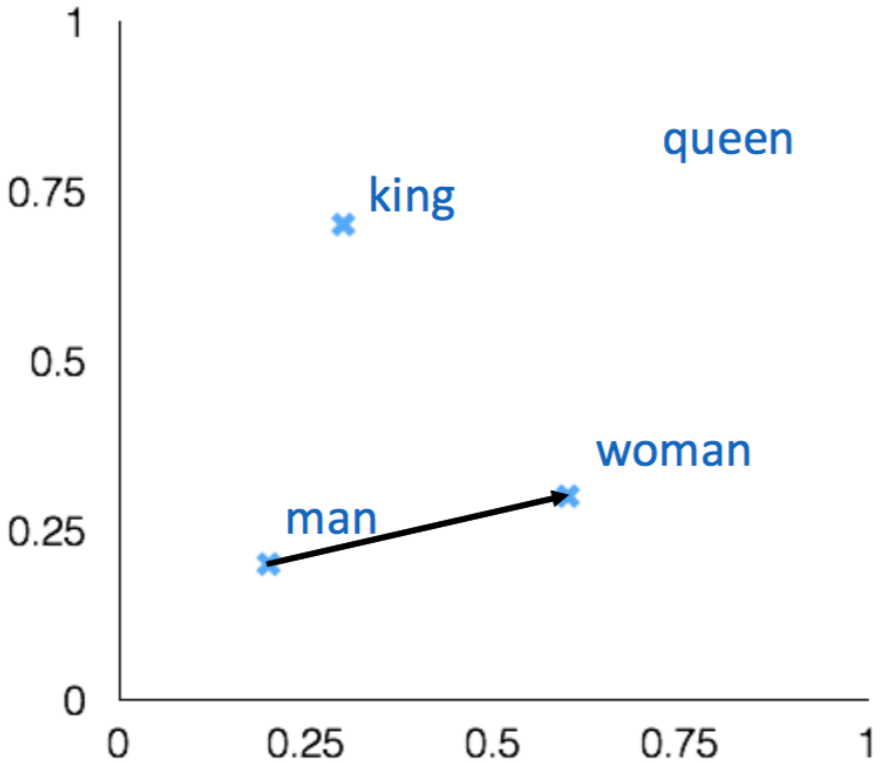
Test for linear relationships, examined by Mikolov et al. (2014)

a:b :: c:?

$$d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$$

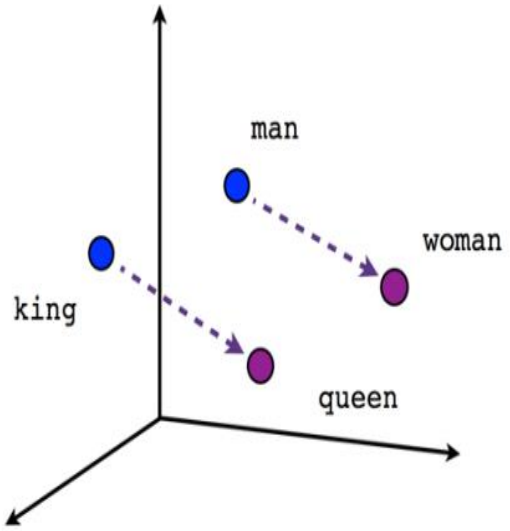
man:woman :: king:?

+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]

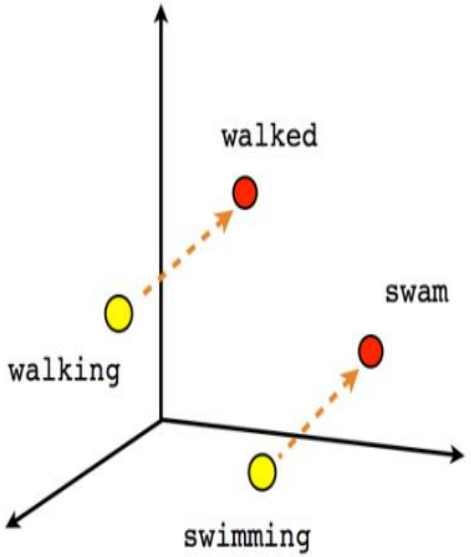


Embeddings Capture Relational Meaning

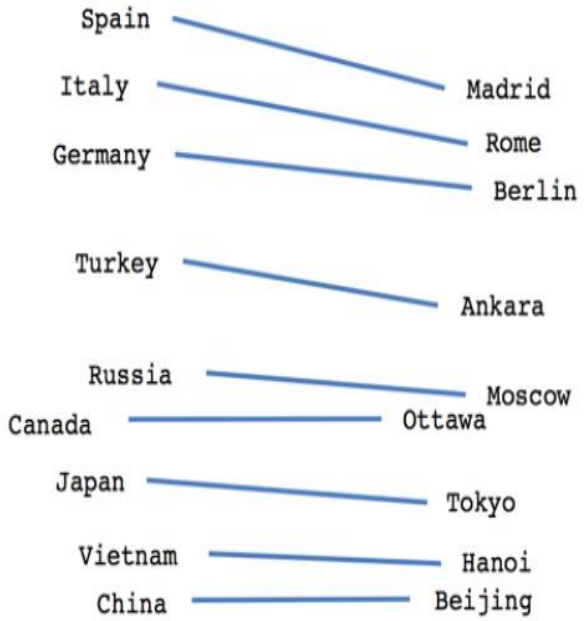
vector('*king*') - vector('*man*') + vector('*woman*') = vector('queen')
vector('*Paris*') - vector('*France*') + vector('*Italy*') = vector('Rome')



Male-Female

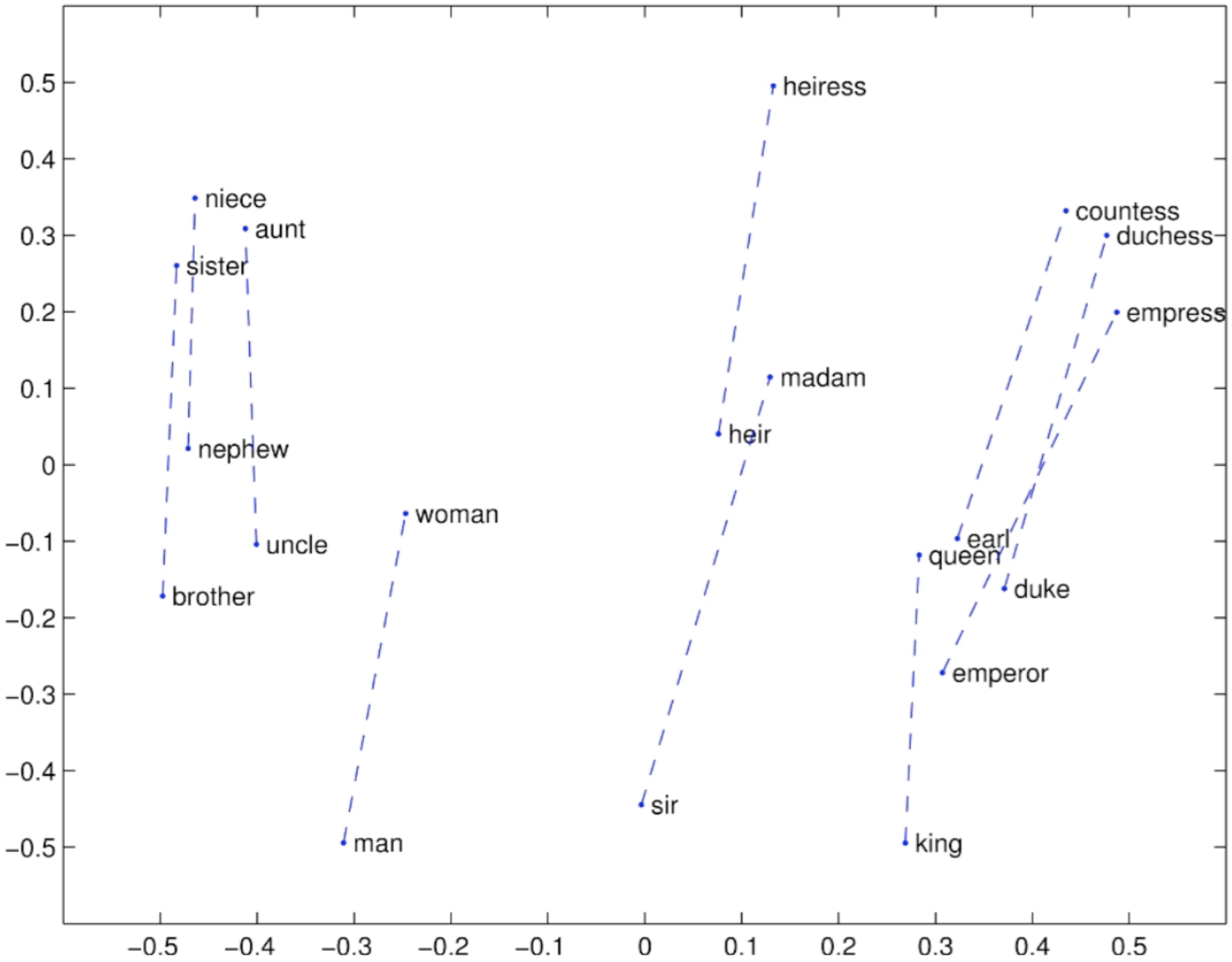


Verb tense

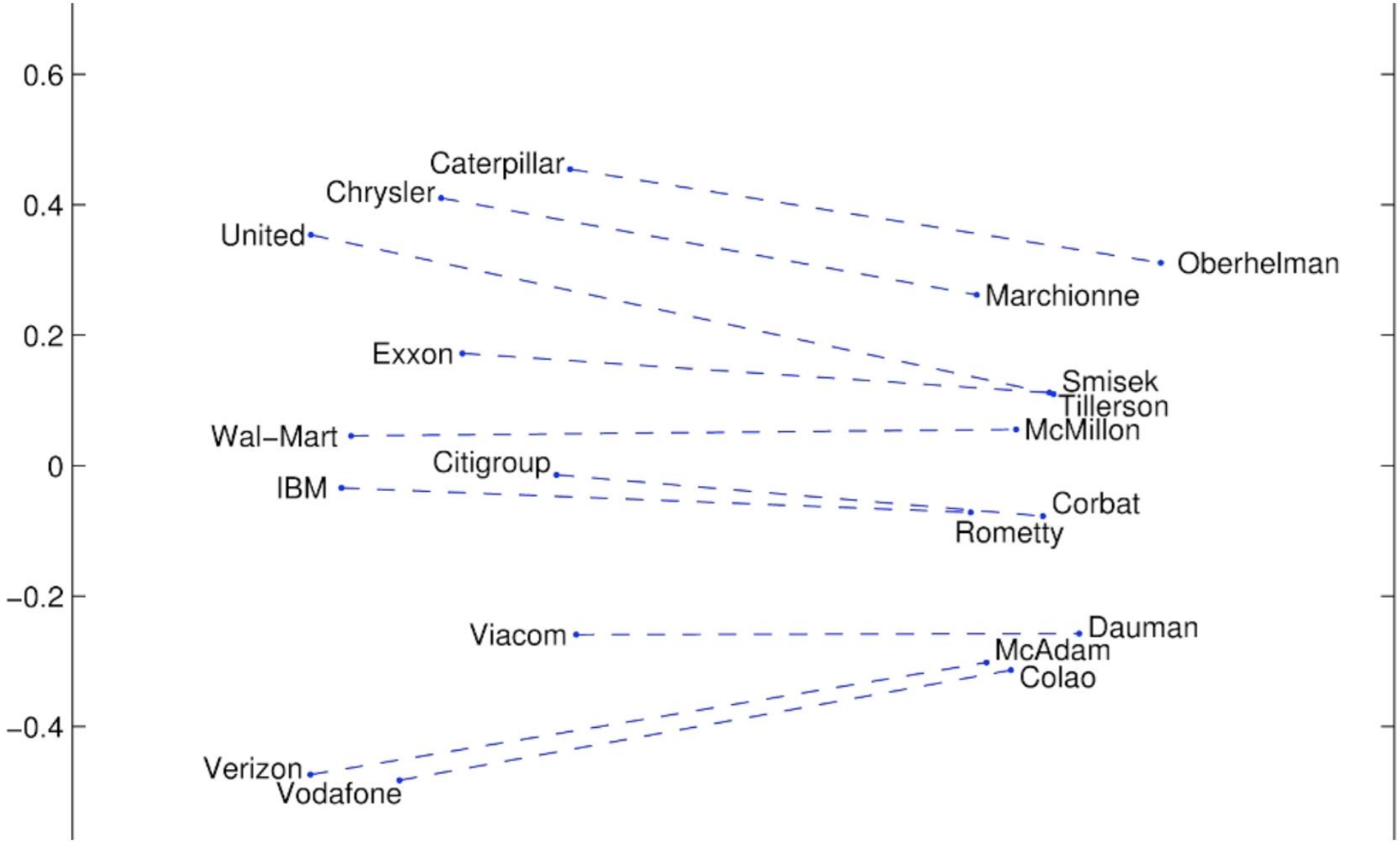


Country-Capital

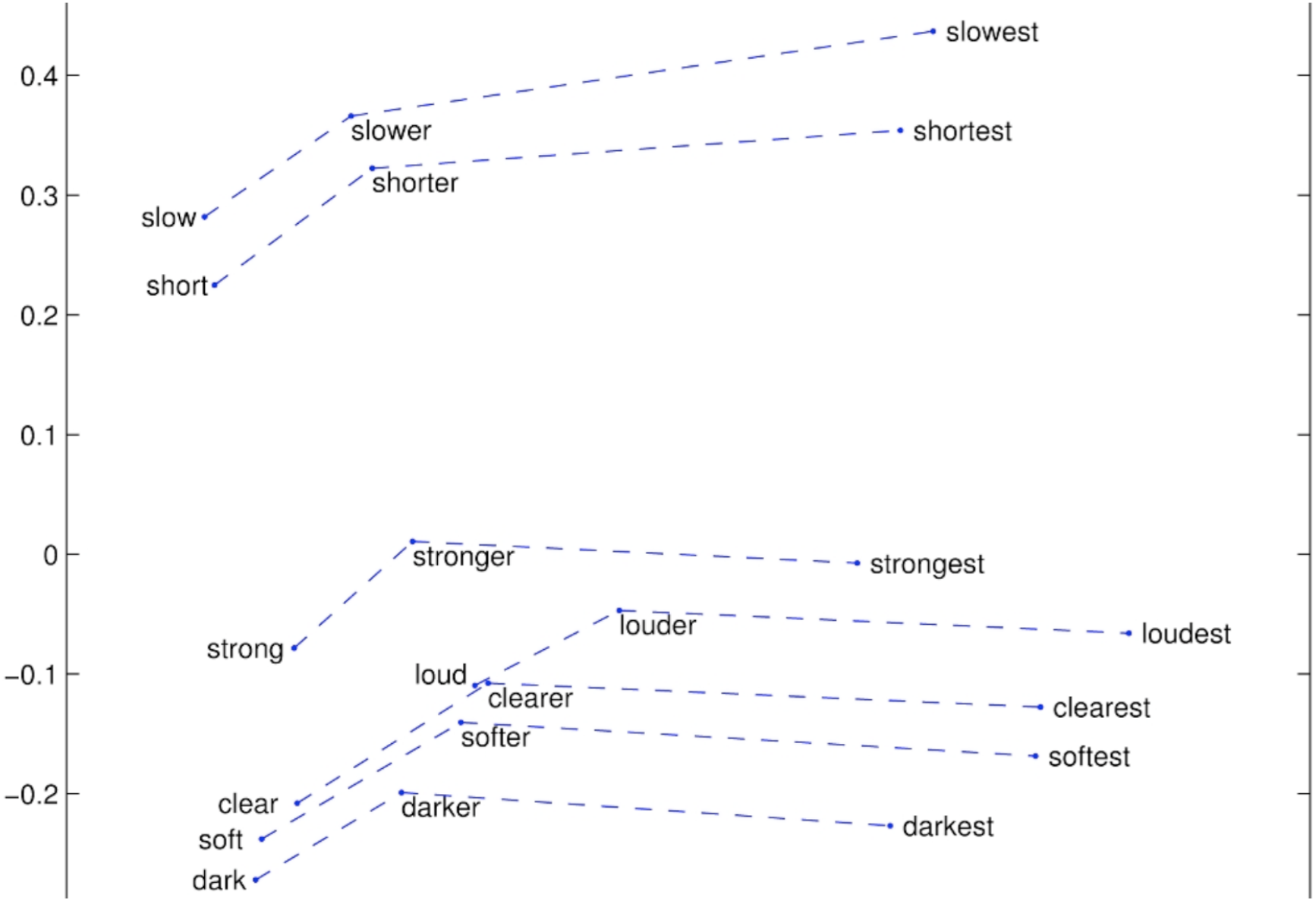
Embeddings Capture Relational Meaning



Embeddings Capture Relational Meaning



Embeddings Capture Relational Meaning

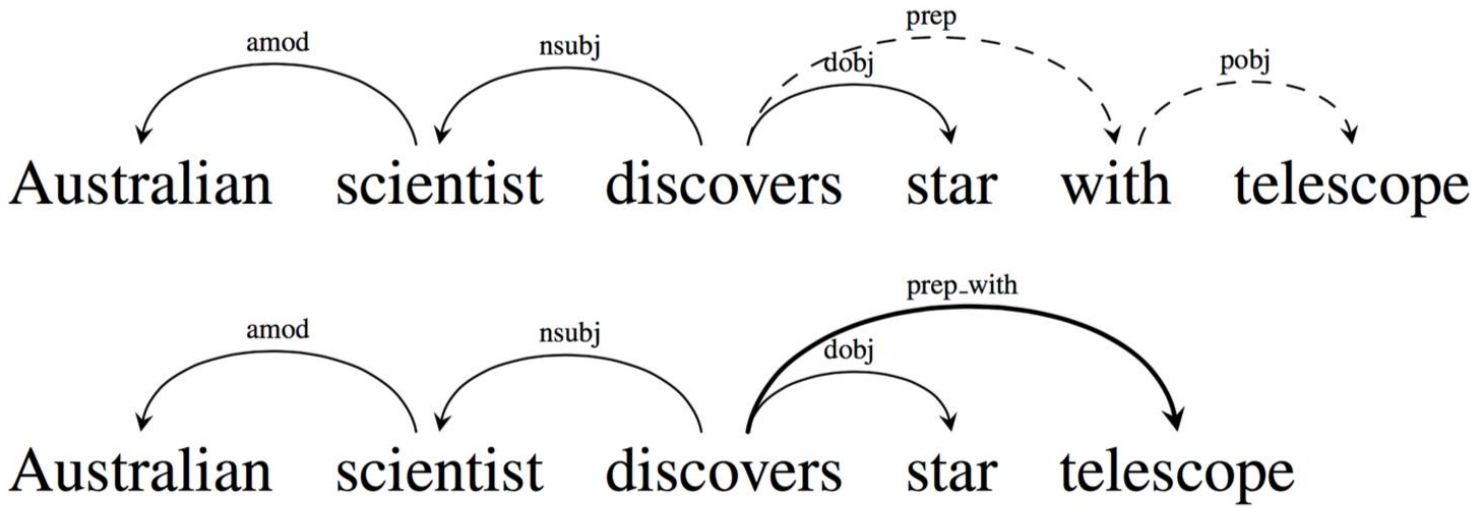


Word2Vec

Modeling the Context

Syntactic Embeddings

Dependency-based embeddings [Levy&Goldberg,2014a]



WORD	CONTEXTS
australian	scientist/amod^{-1}
scientist	australian/amod, discovers/nsubj^{-1}
discovers	scientist/nsubj, star/dobj, telescope/prep_with
star	discovers/dobj^{-1}
telescope	discovers/prep_with^{-1}

Dependency- vs. Word-Based Embeddings

[Levy&Goldberg,2014]

- **Words: topical**
- **Dependencies: functional**
  - also true for explicit representations [Lin,1998; Padó&Lapata,2007]

- Example: *Turing*
  - Words: *nondeterministic, non-deterministic, computability, deterministic, finite-state*
  - Dependencies: *Pauling, Hotelling, Heting, Lessing, Hamming*

Character N-gram Embeddings

[Bojanowski & al.,2016]

- '<' and '>' are added at the beginning and end of words to allow to distinguish prefixes and suffixes from other character sequences
- A word was represented by sum of the vector representation of its *n*-grams
- *n* : 3-6

where

<wh, whe, her, ere, re> + <where>

$$s(w, c) = \sum_{g \in \mathcal{G}_w} \mathbf{z}_g^\top \mathbf{v}_c$$

S : score of given word
w : given word
c : context word

g: *n*-gram size
Z_g: vector of *n*-gram g
V_C: vector of context word

Character N-gram Embeddings

[Bojanowski & al.,2016]

The best results were obtained when the n-gram vector was adapted in the skipgram model

Analogy question

A is to *B* as *C* is to *D*

D = *?*

			skipgram		null vector	summation of n-gram vector
			sg	cbow	sisg-	sisg
Arabic	AR	WS353	51	52	54	**55**
German	DE	GUR350	61	62	64	**70**
		GUR65	78	78	**81**	**81**
		ZG222	35	38	41	**44**
English	EN	RW	43	43	46	**47**
		WS353	72	**73**	71	71
Spanish	ES	WS353	57	58	58	**59**
French	FR	RG65	70	69	**75**	**75**
Romanian	RO	WS353	48	52	51	**54**
Russian	RU	HJ	59	60	60	**66**

Summary

Word Representations

Traditional Method - Bag of Words Model	Word Embeddings
• Uses one-hot encoding • Each word in the vocabulary is represented by one bit position in a HUGE vector. • For example, if we have a vocabulary of 10000 words, and "Hello" is the 4th word in the dictionary, it would be represented by: 0 0 0 1 0 0 . . . . . 0 0 0 0 • Context information is not utilized	• Stores each word in as a point in space, where it is represented by a vector of fixed number of dimensions (generally 300) • Unsupervised, built just by reading a huge corpus • For example, "Hello" might be represented as : [0.4, -0.11, 0.55, 0.3 . . . 0.1, 0.02] • Dimensions are projections along different axes, more of a mathematical concept.