Natural Language Processing

Class 09:Lexical Semantics

- Vector representations / word embeddings
- Context features
- Weighting Frequencies
- Similarity Measures
- Evaluation

3 May 2018

Katrien Beuls

(slides based on Jurafsky & Martin, 3rd edition, Ch. 15-16)

VUB VRIJE UNIVERSITEIT BRUSSEL

Why vector models of meaning? computing the 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 29029 feet"*

Word similarity for 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

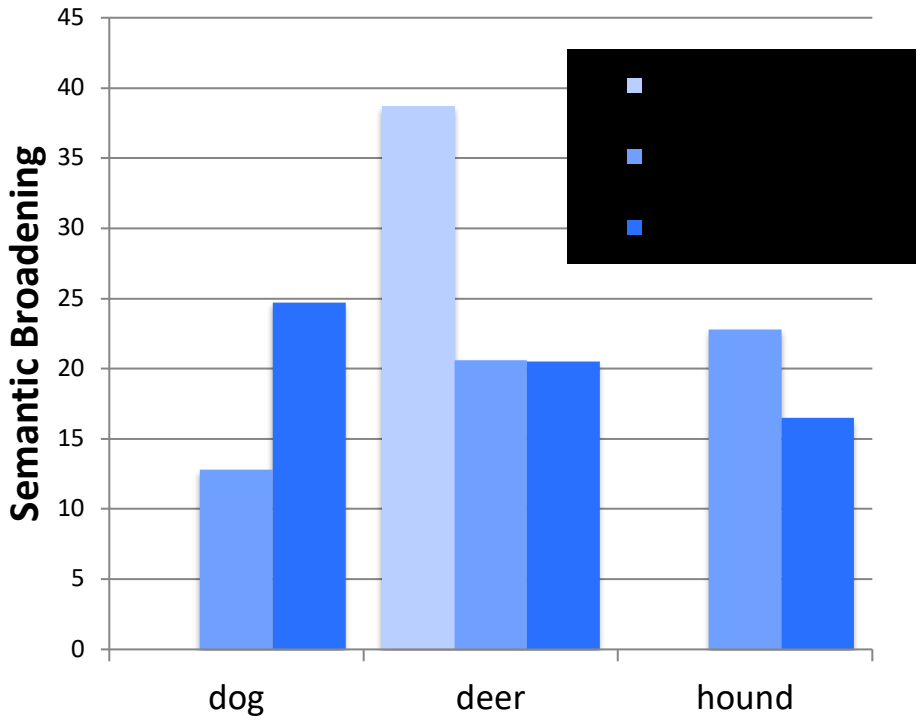
MAINFRAMES

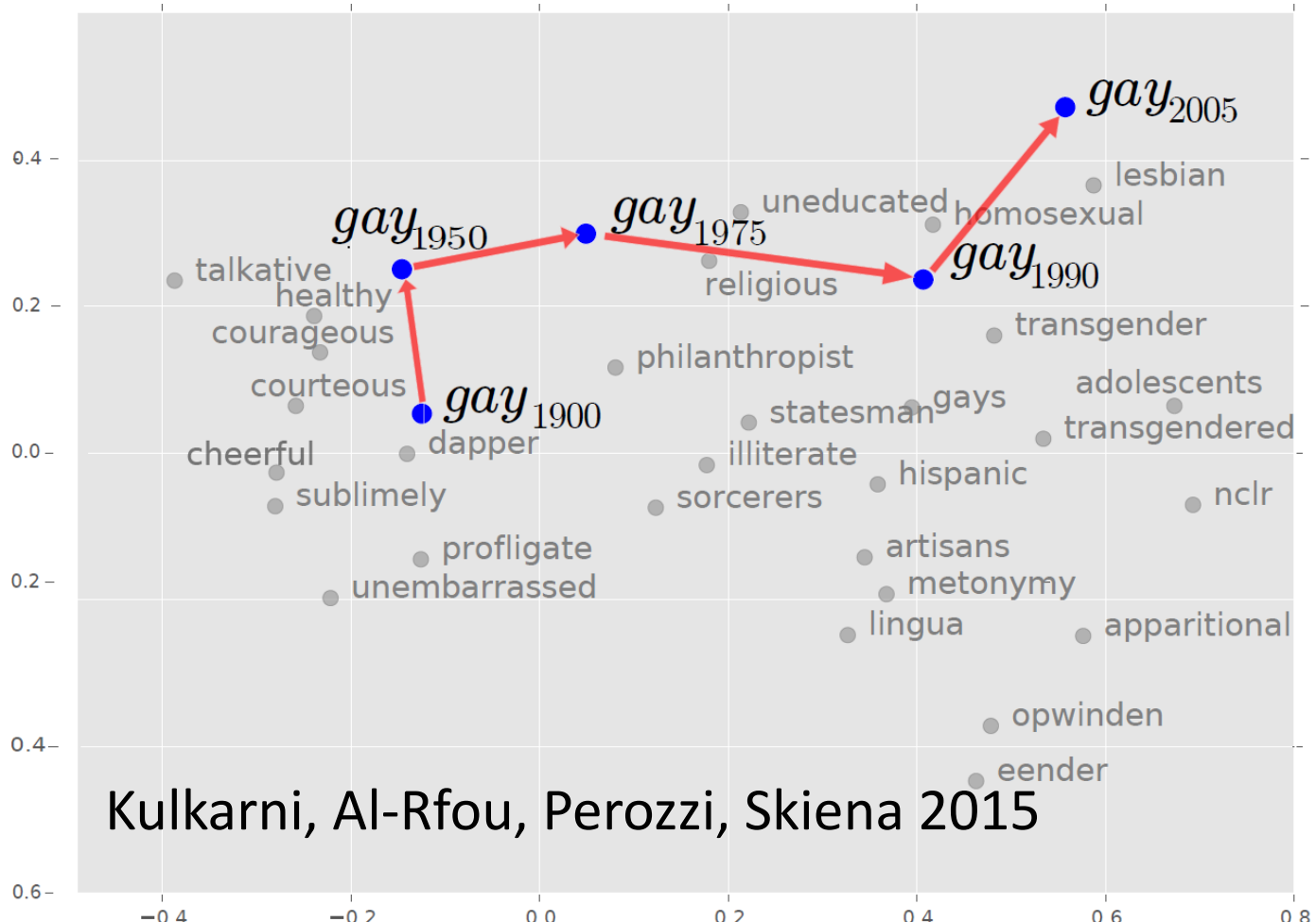
Mainframes usually are referred those computers with fast, advanced processing capabilities that could perform by itself tasks that may require a lot of Personal Computers (PC) Machines. Usually mainframes would have lots of RAMs, very large secondary storage devices, and very fast processors to cater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

Word similarity for historical linguistics: semantic change over time

Sagi, Kaufmann Clark 2013





Kulkarni, Al-Rfou, Perozzi, Skiena 2015

Distributional models of meaning = vector-space models of meaning = vector semantics

Intuitions: 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."

Firth (1957):

- "You shall know a word by the company it keeps!"

Intuition of distributional word similarity

Nida example:

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 algorithm:

- Two words are similar if they have similar word contexts.

Four kinds of vector models

Sparse vector representations

1. Mutual-information weighted word co-occurrence matrices

Dense vector representations:

2. Singular value decomposition (Latent Semantic Analysis)
3. Neural-network-inspired models (skip-grams, CBOW)
4. Brown clusters (IBM clustering)

Shared intuition

Model the meaning of a word by “embedding” in a vector space.

The meaning of a word is a vector of numbers

- Vector models are also called “**embeddings**”.

Contrast: word meaning is represented in many computational linguistic applications by a vocabulary index (“word number 545”)

Old philosophy joke:

Q: What’s the meaning of life?

A: LIFE’

Term-document matrix

Each cell: count of term t in a document d: $tf_{t,d}$:

- 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

Term-document matrix

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

The 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

The words 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

The word-word or word-context matrix

Instead of entire documents, use smaller contexts

- Paragraph
- Window of $\pm$ 4 words

A word is now defined by 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|x|V|

Word-Word matrix
Sample contexts ± 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

We showed only 4x6, but the real matrix is 50,000 x 50,000

- So it's very **sparse**
  - Most values are 0.
- That's OK, since there are lots of efficient algorithms for sparse matrices.

The size of windows depends on your goals

- The shorter the windows , the more **syntactic** the representation
  - ± 1-3 very syntactic
- The longer the windows, the more **semantic** the representation
  - ± 4-10 more semantic

Two kinds of co-occurrence between two words

(Schütze and Pedersen, 1993)

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*.

Weighting raw frequencies

Problem with raw counts

Raw word frequency is not a great measure of association between words

- It's very skewed
  - "the" and "of" are very frequent, but maybe not the most discriminative

We'd rather have a measure that asks whether a context word is **particularly informative** about the target word.

Positive Pointwise Mutual Information (PPMI)

Pointwise Mutual Information

Pointwise mutual information:

Do events x and y co-occur more than if they were independent?

$$\text{PMI}(X,Y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$

PMI between two words: (Church & Hanks 1989)

Do words x and y co-occur more than if they were independent?

$$\text{PMI}(word_1, word_2) = \log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}$$

Positive Pointwise Mutual Information

PMI ranges from $-\infty$ to $+\infty$

But the negative values are problematic

- Things are co-occurring **less than** we expect by chance
- Unreliable without enormous corpora
  - Imagine w1 and w2 whose probability is each 10^{-6}
  - Hard to be sure p(w1,w2) is significantly different than 10^{-12}
- Plus it's not clear people are good at "unrelatedness"

So we just replace negative PMI values by 0

Positive PMI (PPMI) between word1 and word2:

$$\text{PPMI}(word_1, word_2) = \max\left(\log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}, 0\right)$$

Computing PPMI on a term-context matrix

Matrix F with W rows (words) and C columns (contexts)

f_{ij} is # of times w_i occurs in context c_j

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \qquad p_{i*} = \frac{\sum_{j=1}^{C} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \qquad p_{*j} = \frac{\sum_{i=1}^{W} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

	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

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}} \qquad ppmi_{ij} = \begin{cases} pmi_{ij} & \text{if } pmi_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

Count(w,context)

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

p(w=information,c=data) = 6/19 = .32

p(w=information) = 11/19 = .58

p(c=data) = 7/19 = .37

$$p(w_i) = \frac{\sum_{j=1}^{C} f_{ij}}{N} \qquad p(c_j) = \frac{\sum_{i=1}^{W} f_{ij}}{N}$$

p(w,context) **p(w)**

	computer	data	pinch	result	sugar	
apricot	0.00	0.00	0.05	0.00	0.05	0.11
pineapple	0.00	0.00	0.05	0.00	0.05	0.11
digital	0.11	0.05	0.00	0.05	0.00	0.21
information	0.05	0.32	0.00	0.21	0.00	0.58
p(context)	0.16	0.37	0.11	0.26	0.11	

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}}$$

	p(w,context)					p(w)
	computer	data	pinch	result	sugar	
apricot	0.00	0.00	0.05	0.00	0.05	0.11
pineapple	0.00	0.00	0.05	0.00	0.05	0.11
digital	0.11	0.05	0.00	0.05	0.00	0.21
information	0.05	0.32	0.00	0.21	0.00	0.58
p(context)	0.16	0.37	0.11	0.26	0.11	

pmi(information,data) = $\log_2$ (.32 / (.37*.58)) = .58

(.57 using full precision)

PPMI(w,context)

	computer	data	pinch	result	sugar
apricot	-	-	2.25	-	2.25
pineapple	-	-	2.25	-	2.25
digital	1.66	0.00	-	0.00	-
information	0.00	0.57	-	0.47	-

Weighting PMI

PMI is biased toward infrequent events

- Very rare words have very high PMI values

Two solutions:

- Give rare words slightly higher probabilities
- Use add-one smoothing (which has a similar effect)

Weighting PMI: Giving rare context words slightly higher probability

Raise the context probabilities to $\alpha = 0.75$:

$$\text{PPMI}_\alpha(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P_\alpha(c)}, 0)$$

$$P_\alpha(c) = \frac{count(c)^\alpha}{\sum_c count(c)^\alpha}$$

This helps because $P_\alpha(c) > P(c)$ for rare c

Consider two events, P(a) = .99 and P(b)=.01

$$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75}+.01^{.75}} = .97 \quad P_\alpha(b) = \frac{.01^{.75}}{.01^{.75}+.01^{.75}} = .03$$

Laplace smoothing

Add-2 Smoothed Count(w,context)

	computer	data	pinch	result	sugar
apricot	2	2	3	2	3
pineapple	2	2	3	2	3
digital	4	3	2	3	2
information	3	8	2	6	2

p(w,context) [add-2] **p(w)**

	computer	data	pinch	result	sugar	
apricot	0.03	0.03	0.05	0.03	0.05	0.20
pineapple	0.03	0.03	0.05	0.03	0.05	0.20
digital	0.07	0.05	0.03	0.05	0.03	0.24
information	0.05	0.14	0.03	0.10	0.03	0.36
p(context)	0.19	0.25	0.17	0.22	0.17	

PPMI versus add-2 smoothed PPMI

PPMI(w,context)

	computer	data	pinch	result	sugar
apricot	-	-	2.25	-	2.25
pineapple	-	-	2.25	-	2.25
digital	1.66	0.00	-	0.00	-
information	0.00	0.57	-	0.47	-

PPMI(w,context) [add-2]

	computer	data	pinch	result	sugar
apricot	0.00	0.00	0.56	0.00	0.56
pineapple	0.00	0.00	0.56	0.00	0.56
digital	0.62	0.00	0.00	0.00	0.00
information	0.00	0.58	0.00	0.37	0.00

Alternative to PPMI for measuring association

tf-idf (that's a hyphen not a minus sign)

The combination of two factors

- **Term frequency** (Luhn 1957): frequency of the word (can be logged)
- **Inverse document frequency** (IDF) (Sparck Jones 1972)
  - N is the total number of documents
  - df_i = "document frequency of word *i*"
  - = # of documents with word *I*
- w_{ij} = *word* i *in document* j

$$\text{idf}_i = \log\left(\frac{N}{df_i}\right)$$

$$w_{ij} = tf_{ij}\, idf_i$$

tf-idf not generally used for word-word similarity

But is by far the most common weighting when we are considering the relationship of words to documents

Measuring vector similarity

Measuring similarity

Given 2 target words *v* and *w*

We'll need a way to measure their similarity.

Most measure of vectors similarity are based on the:

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

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

- High when two vectors have large values in same dimensions.
- Low (in fact 0) for **orthogonal vectors** with zeros in complementary distribution

Problem with dot product

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

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

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

Vectors are longer if they have higher values in each dimension

That means more frequent words will have higher dot products

That's bad: we don't want a similarity metric to be sensitive to word frequency

Solution: cosine

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

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

v_i is the PPMI value for word v in context i

w_i is the PPMI value for word w in context i.

Cos($\vec{v},\vec{w}$) is the cosine similarity of $\vec{v}$ and $\vec{w}$

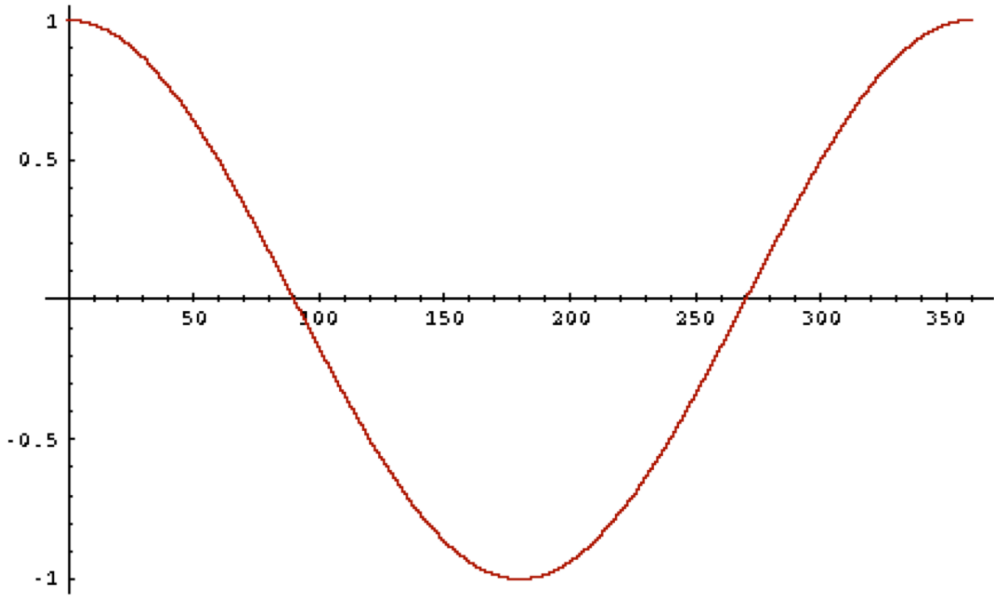
Cosine as a similarity metric

-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 range 0-1



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

	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) =

$$\frac{2+0+0}{\sqrt{2+0+0}\ \sqrt{1+36+1}} = \frac{2}{\sqrt{2}\sqrt{38}} = .23$$

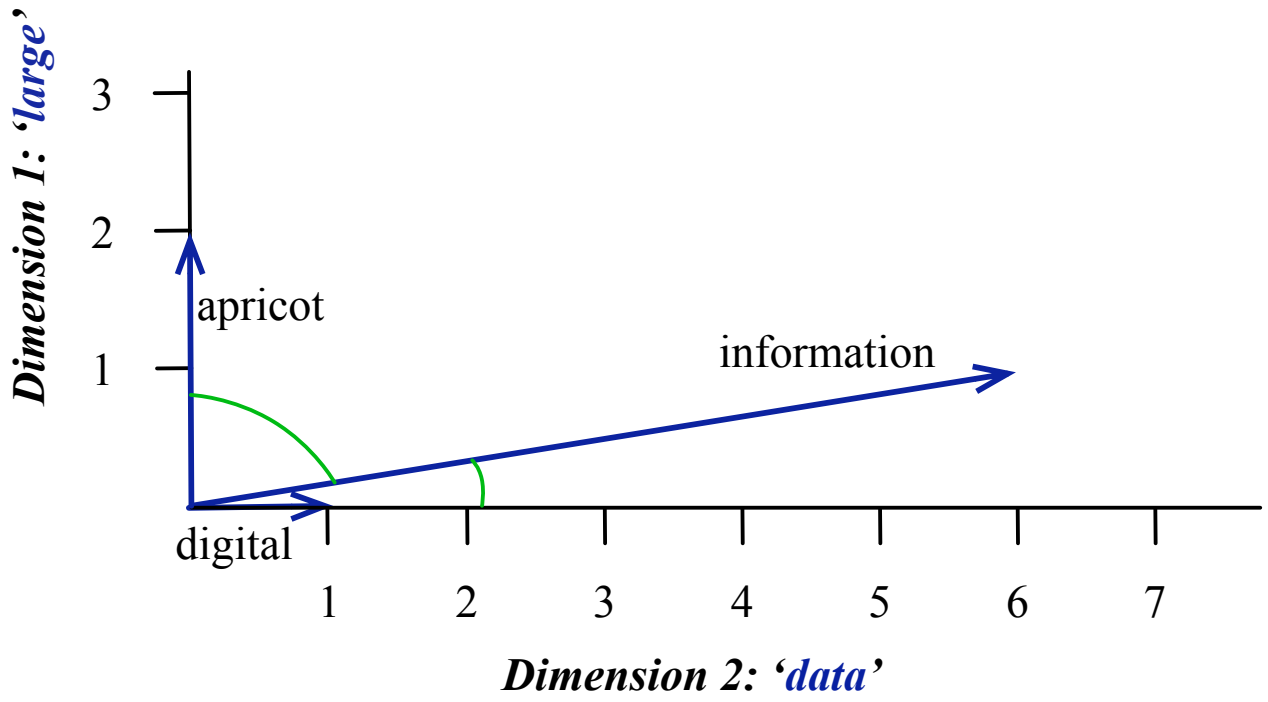
cosine(digital,information) =

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

cosine(apricot,digital) =

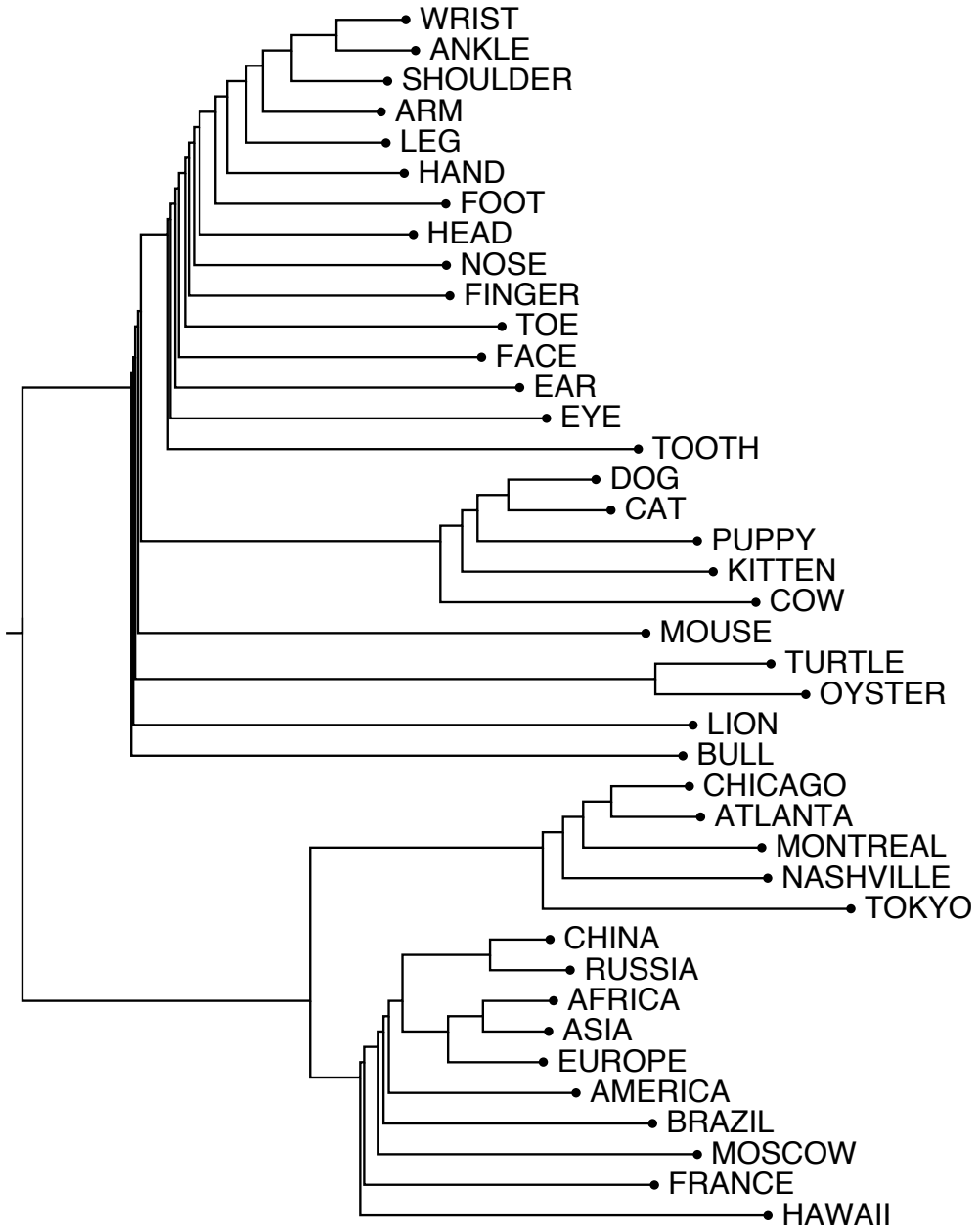
$$\frac{0+0+0}{\sqrt{1+0+0}\ \sqrt{0+1+4}} = 0$$

Visualizing vectors and angles



	large	data
apricot	2	0
digital	0	1
information	1	6

Clustering vectors to visualize similarity in co-occurrence matrices



Rohde et al. (2006)

Other possible similarity measures

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

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

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

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

Syntactic context features

Using syntax to define a word's context

Zellig Harris (1968)

> "The meaning of entities, and the meaning of grammatical relations among them, is related to the restriction of combinations of these entities relative to other entities"

Two words are similar if they have similar syntactic contexts

Duty and **responsibility** have similar syntactic distribution:

Modified by adjectives	additional, administrative, assumed, collective, congressional, constitutional …
Objects of verbs	assert, assign, assume, attend to, avoid, become, breach..

Co-occurrence vectors based on syntactic dependencies

Dekang Lin, 1998 "Automatic Retrieval and Clustering of Similar Words"

Each dimension: a context word in one of R dependency relations

- Subject-of- "absorb"

Instead of a vector of $|V|$ features, a vector of $R|V|$

Example: counts for the word *cell* :

	subj-of, absorb	subj-of, adapt	subj-of, behave	...	pobj-of, inside	pobj-of, into	...	nmod-of, abnormality	nmod-of, anemia	nmod-of, architecture	...	obj-of, attack	obj-of, call	obj-of, come from	obj-of, decorate	...	nmod, bacteria	nmod, body	nmod, bone marrow
cell	1	1	1		16	30		3	8	1		6	11	3	2		3	2	2

Syntactic dependencies for dimensions

Alternative (Padó and Lapata 2007):

- Instead of having a |V| x R|V| matrix
- Have a |V| x |V| matrix
- But the co-occurrence counts aren't just counts of words in a window
- But counts of words that occur in one of R dependencies (subject, object, etc).
- So M("cell","absorb") = count(subj(cell,absorb)) + count(obj(cell,absorb)) + count(pobj(cell,absorb)), etc.

PMI applied to dependency relations

"Drink it" more common than "drink wine"

But "wine" is a better "drinkable" thing than "it"

Hindle, Don. 1990. Noun Classification from Predicate-Argument Structure. ACL

Object of "drink"	Count	PMI
tea	2	11.8
liquid	2	10.5
wine	2	9.3
anything	3	5.2
it	3	1.3

Evaluating semantic similarity

Evaluating similarity

Extrinsic (task-based, end-to-end) Evaluation:

- Question Answering | Spell Checking | Essay grading | Plagiarism detection

Intrinsic Evaluation:

- Correlation between algorithm and human word similarity ratings
  - Wordsim353: 353 noun pairs rated 0-10. *sim(plane,car)=5.77*
- Correlation with distances in lexical ontology
  - WordNet: hierarchical taxonomy of synonyms
- Taking TOEFL multiple-choice vocabulary tests
  - `Levied is closest in meaning to:`

    `imposed, believed, requested, correlated`

Dense Vectors

Sparse versus dense vectors

PPMI 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 versus 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 similarity between a word with *car* as a neighbor and a word with *automobile* as a neighbor

Three methods for getting short dense vectors

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

1. Dense Vectors via SVD

Intuition

Approximate an N-dimensional dataset using fewer dimensions

By first rotating the axes into a new space

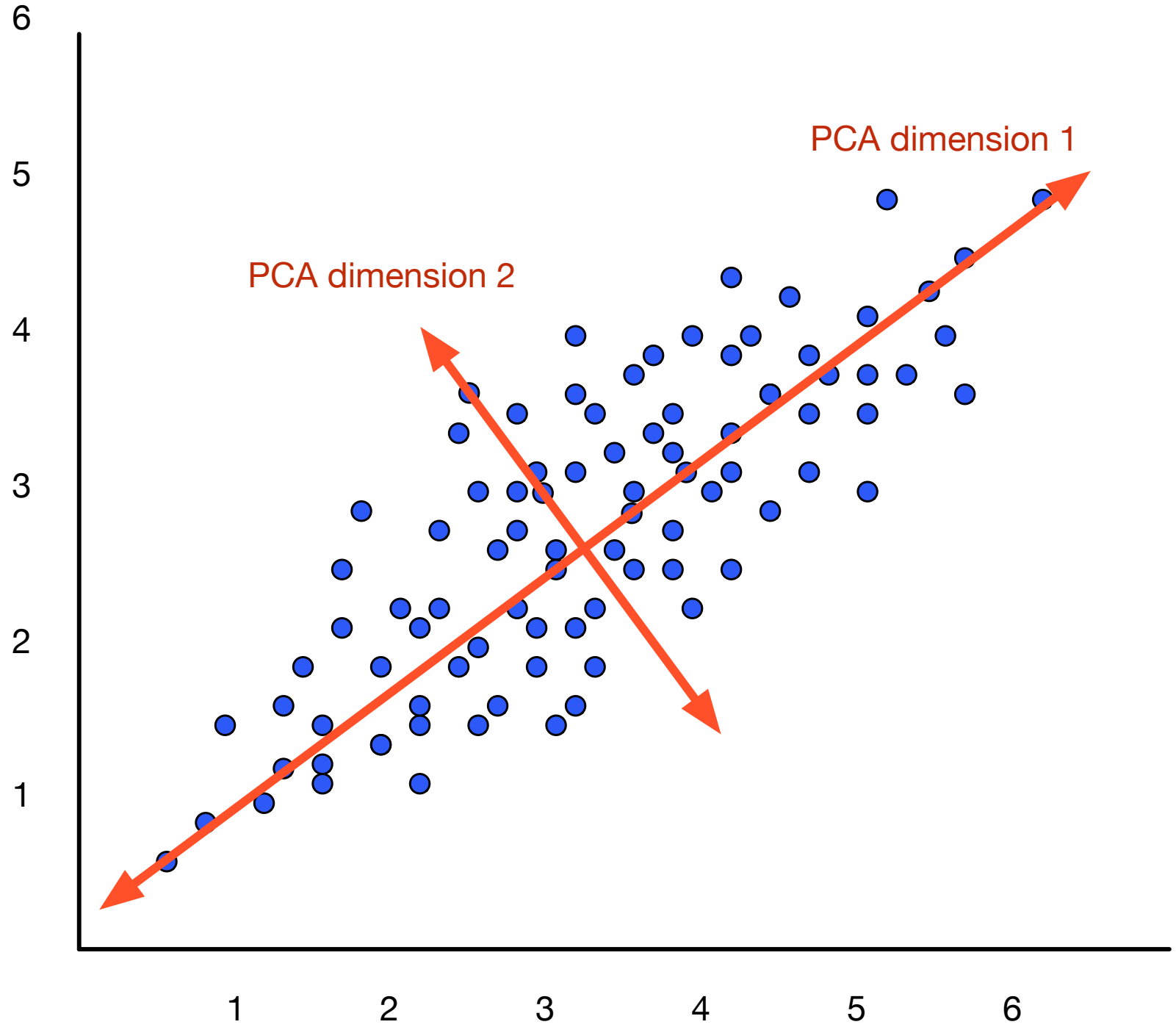
In which the highest order dimension captures the most variance in the original dataset

And the next dimension captures the next most variance, etc.

Many such (related) methods:

- PCA – principle components analysis
- Factor Analysis
- SVD

Dimensionality reduction



Singular Value Decomposition

Any rectangular w x c matrix X equals the product of 3 matrices:

W: rows corresponding to original but *m* columns represents a dimension in a new latent space, such that

- M column vectors are orthogonal to each other
- Columns are ordered by the amount of variance in the dataset each new dimension accounts for

S: diagonal *m* x *m* matrix of **singular values** expressing the importance of each dimension.

C: columns corresponding to original but *m* rows corresponding to singular values

SVD applied to term-document matrix: Latent Semantic Analysis

Deerwester et al (1988)

If instead of keeping all *m* dimensions, we just keep the top *k* singular values. Let's say 300.

The result is a least-squares approximation to the original X

But instead of multiplying, we'll just make use of W.

Each row of W:

- A k-dimensional vector
- Representing word *w*

SVD applied to term-term matrix

$$\underset{|V|\times|V|}{\begin{bmatrix} X \end{bmatrix}} = \underset{|V|\times|V|}{\begin{bmatrix} W \end{bmatrix}} \underset{|V|\times|V|}{\begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_V \end{bmatrix}} \underset{|V|\times|V|}{\begin{bmatrix} C \end{bmatrix}}$$

(I'm simplifying here by assuming the matrix has rank |V|)

Truncated SVD on term-term matrix

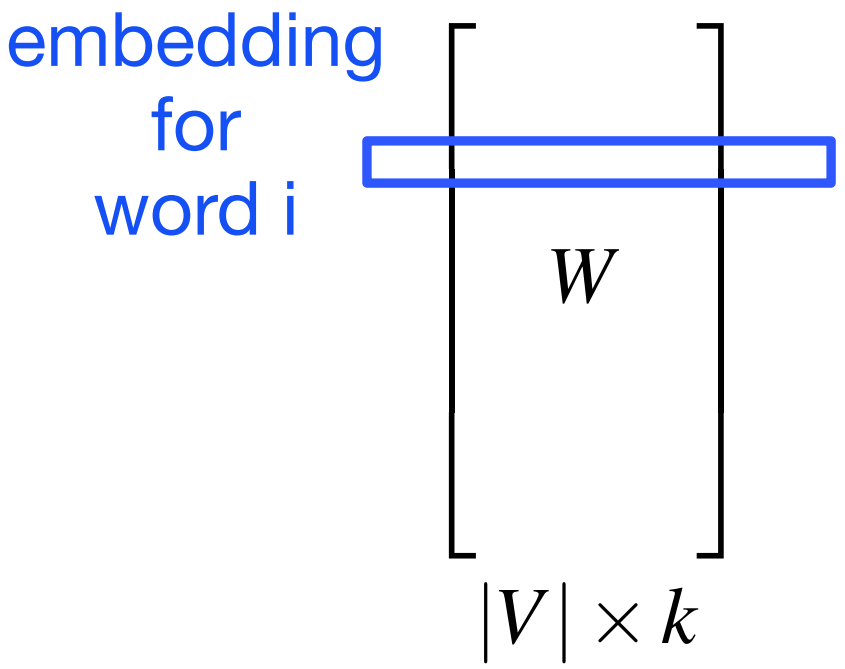
$$\underset{|V|\times|V|}{\begin{bmatrix} X \end{bmatrix}} = \underset{|V|\times k}{\begin{bmatrix} W \end{bmatrix}} \underset{k\times k}{\begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_k \end{bmatrix}} \underset{k\times|V|}{\begin{bmatrix} C \end{bmatrix}}$$

Truncated SVD produces embeddings

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

K might range from 50 to 1000

Generally we keep the top k dimensions, but some experiments suggest that getting rid of the top 1 dimension or even the top 50 dimensions is helpful (Lapesa and Evert 2014).



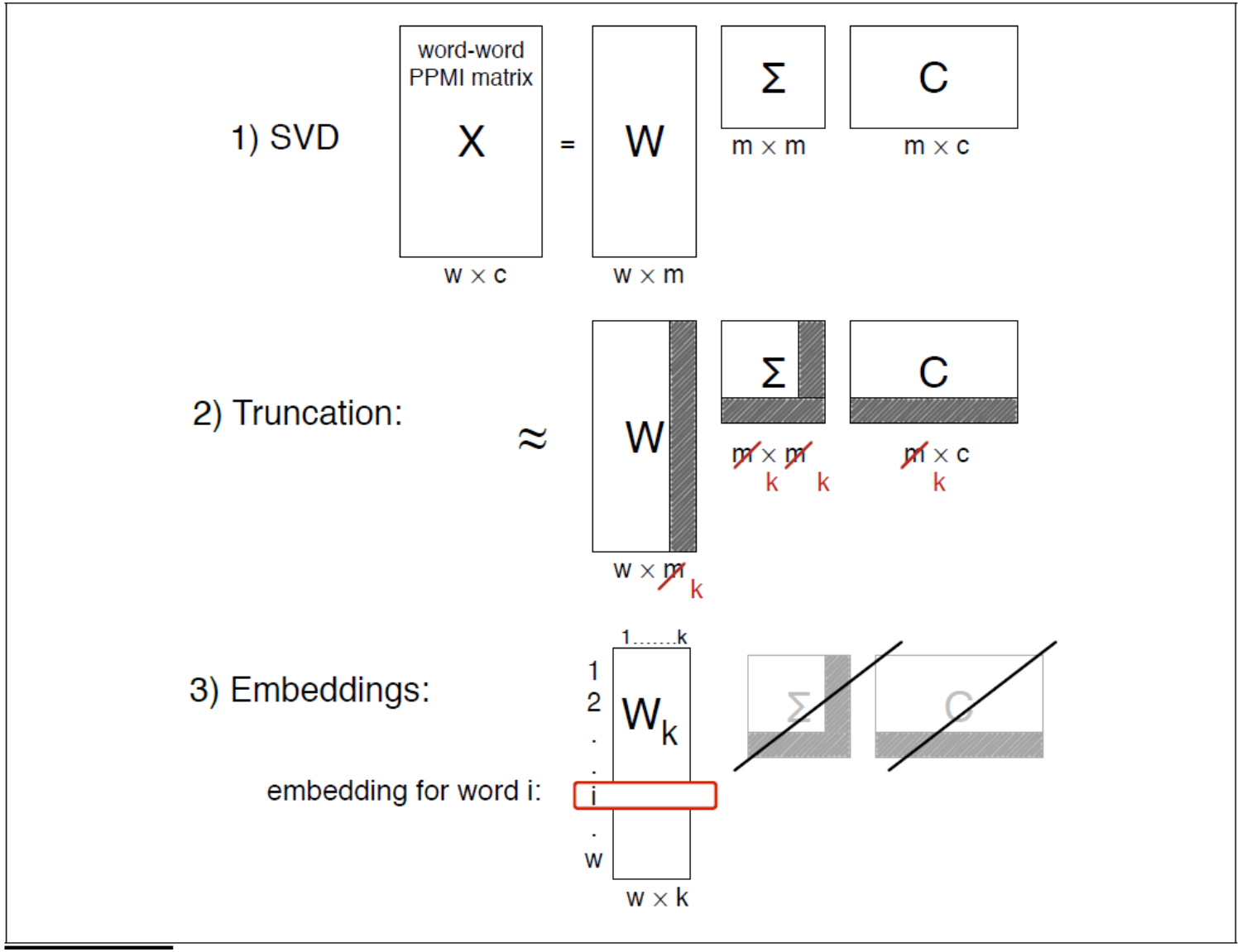
LSA more details

300 dimensions are commonly used

The cells are commonly weighted by a product of two weights

- Local weight: Log term frequency
- Global weight: either idf or an entropy measure

Let's return to PPMI word-word matrices



Embeddings versus sparse vectors

Dense SVD embeddings sometimes work better than sparse PPMI matrices at tasks such as 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 weight the dimensions for the task.
- Dense models may do better at capturing higher order co-occurrence.

2. Embeddings inspired by neural language models: skip-grams and CBOW

VRIJE
UNIVERSITEIT
BRUSSEL

Prediction-based models: An alternative way to get dense vectors

Skip-gram (Mikolov et al. 2013a) **CBOW** (Mikolov et al. 2013b)

Learn embeddings as part of the process of word prediction.

Train a neural network to predict neighboring words

- Inspired by **neural net language models**.
- In so doing, learn dense embeddings for the words in the training corpus.

Advantages:

- Fast, easy to train (much faster than SVD)
- Available online in the `word2vec` package
- Including sets of pretrained embeddings!

Skip-grams

Predict each neighboring word

- in a context window of 2C words
- from the current word

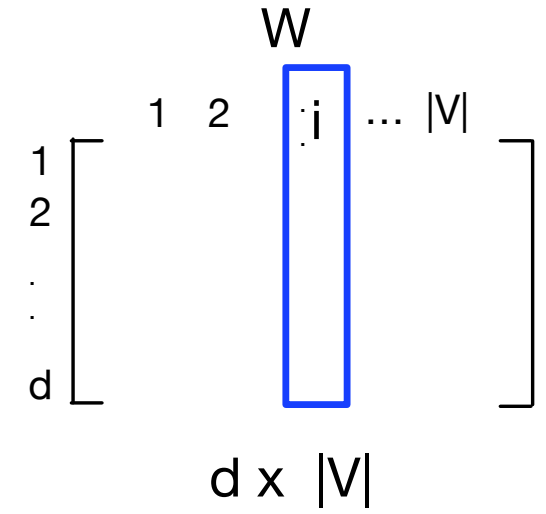
So for C=2, we are given word w_t and predicting these 4 words:

$$[w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}]$$

Skip-grams learn 2 embeddings for each w

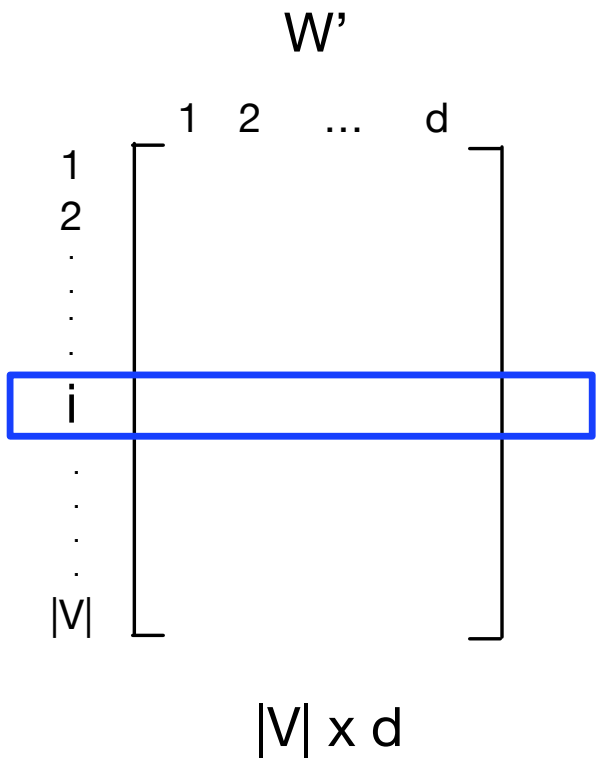
input embedding *v,* in the input matrix *W*

Column *i* of the input matrix *W* is the 1×*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* × 1 vector embedding v'_i for word *i* in the vocabulary.

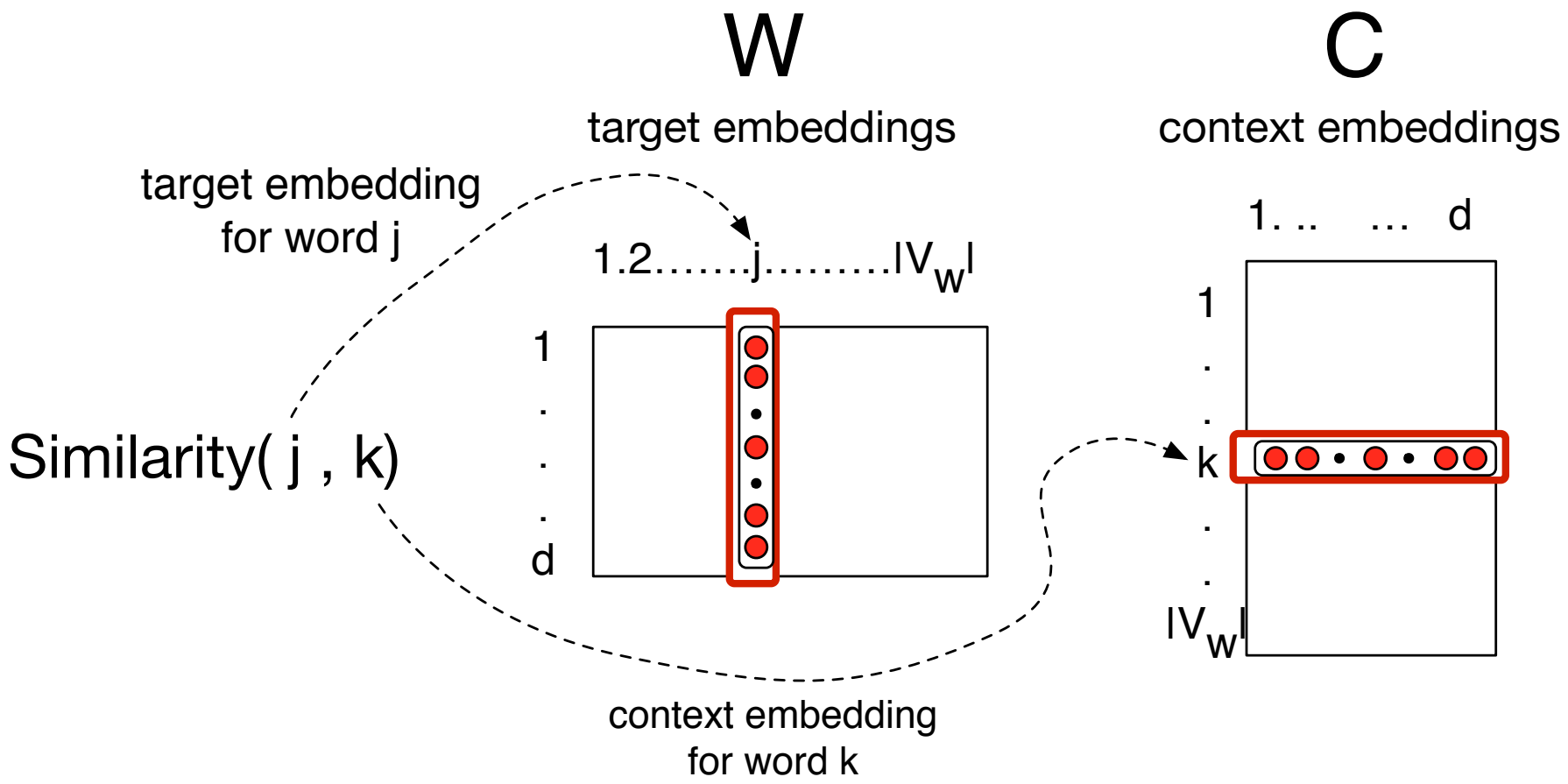


Setup

Walking through corpus pointing at word $w(t)$, whose index in the vocabulary is j, so we'll call it w_j ($1 < j < |V|$).

Let's predict $w(t+1)$, whose index in the vocabulary is k ($1 < k < |V|$). Hence our task is to compute $P(w_k|w_j)$.

Intuition: similarity as dot-product between a target vector and context vector



Similarity is computed from dot product

Remember: two vectors are similar if they have a high dot product

- Cosine is just a normalized dot product

So:

- Similarity(j,k) $\propto c_k \cdot v_j$

We'll need to normalize to get a probability

Turning dot products into probabilities

Similarity(j,k) = $c_k \cdot v_j$

We use softmax to turn into probabilities

$$p(w_k|w_j) = \frac{exp(c_k \cdot v_j)}{\sum_{i \in |V|} exp(c_i \cdot v_j)}$$

Embeddings from W and W’

Since we have two embeddings, v_j and c_j for each word w_j

We can either:

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

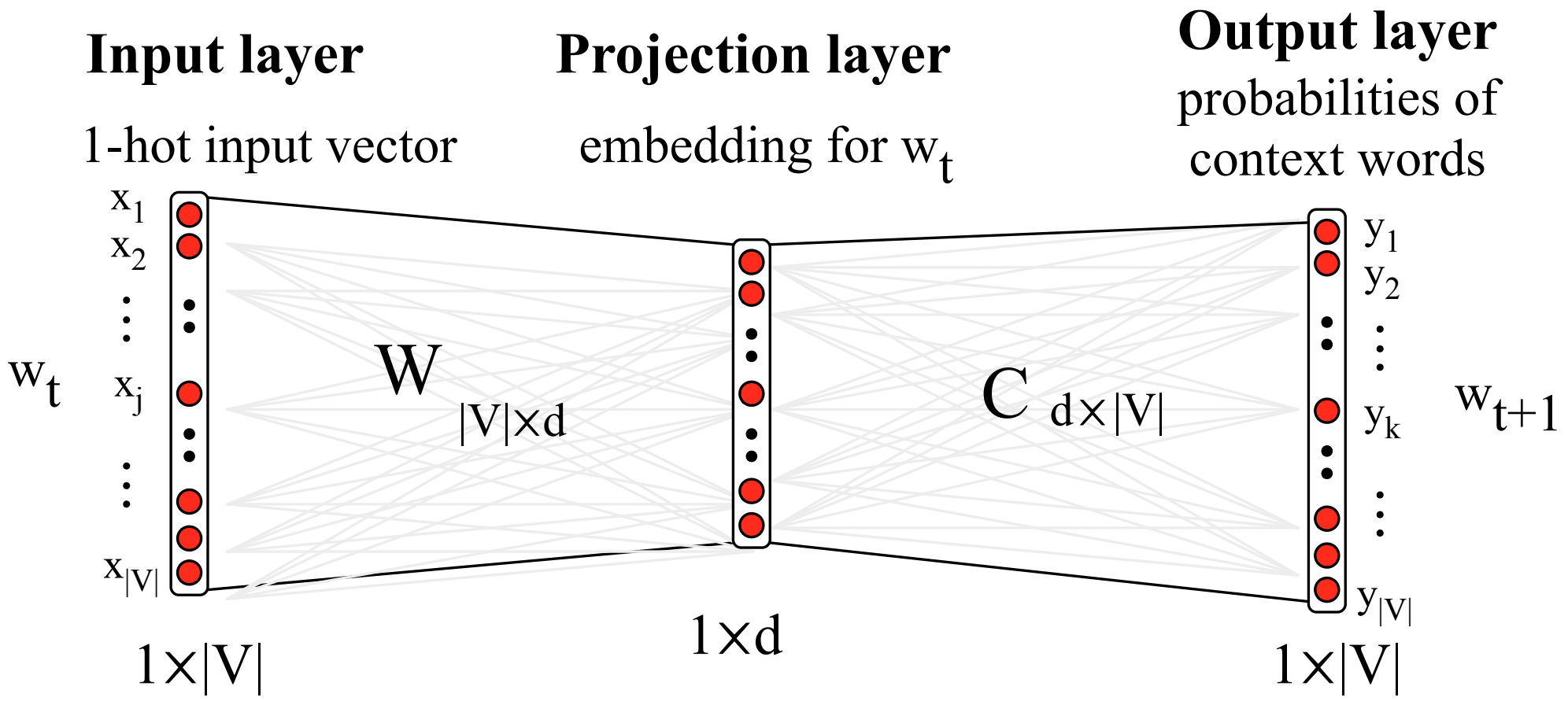
Learning

Start with some initial embeddings (e.g., random)

iteratively make the embeddings for a word

- more like the embeddings of its neighbors
- less like the embeddings of other words.

Visualizing W and C as a network for doing error backprop



One-hot vectors

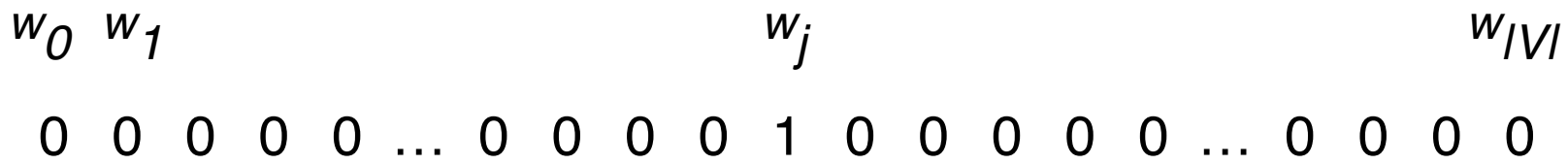
A vector of length |V|

1 for the target word and 0 for other words

So if “popsicle” is vocabulary word 5

The **one-hot vector** is

[0,0,0,0,1,0,0,0,0……0]



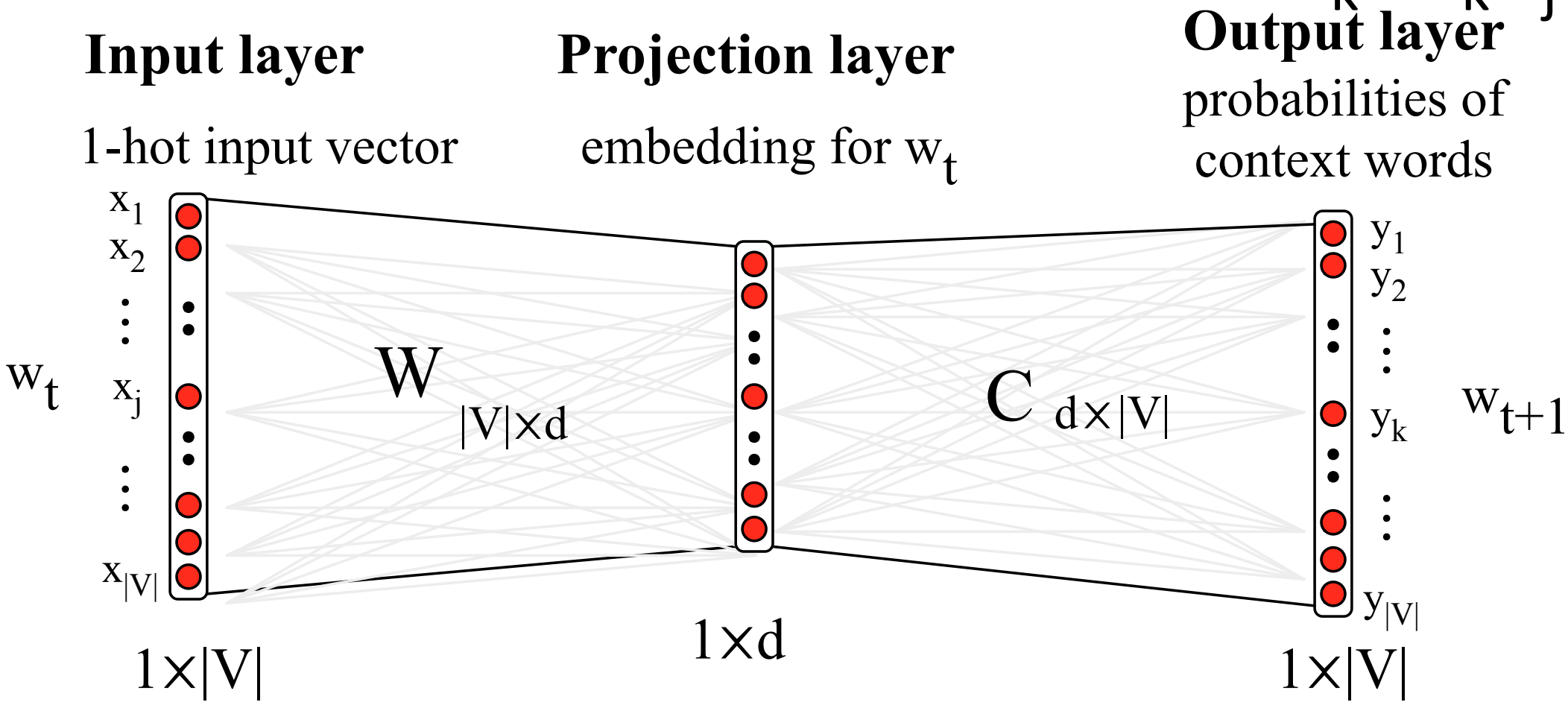
Skip-gram

$o = Ch$

$h = v_j$

$o_k = c_k h$

$o_k = c_k \cdot v_j$



Problem with the softmax

The denominator: have to compute over every word in vocab

$$p(w_k|w_j) = \frac{exp(c_k \cdot v_j)}{\sum_{i \in |V|} exp(c_i \cdot v_j)}$$

Instead: just sample a few of those negative words

Goal in learning

Make the word like the context words

lemon, a [tablespoon of apricot preserves or] jam
c1 c2 w c3 c4

$$\sigma(x) = \frac{1}{1+e^{x}}$$

We want this to be high:

$$\sigma(c1 \cdot w) + \sigma(c2 \cdot w) + \sigma(c3 \cdot w) + \sigma(c4 \cdot w)$$

And not like *k* randomly selected "noise words"

[cement metaphysical dear coaxial apricot attendant whence forever puddle]
n1 n2 n3 n4 n5 n6 n7 n8

We want this to be low:

$$\sigma(n1 \cdot w) + \sigma(n2 \cdot w) + \ldots + \sigma(n8 \cdot w)$$

Skipgram with negative sampling: Loss function

$$\log \sigma(c \cdot w) + \sum_{i=1}^{\kappa} \mathbb{E}_{w_i \sim p(w)} \left[\log \sigma(-w_i \cdot w) \right]$$

Relation between skipgrams and PMI!

If we multiply WW'^{T}

We get a |V|x|V| matrix M , each entry m_{ij} corresponding to some association between input word i and output word j

Levy and Goldberg (2014b) show that skip-gram reaches its optimum just when this matrix is a shifted version of PMI:

$$WW'^{T} = M^{\text{PMI}} - \log k$$

So skip-gram is implicitly factoring a shifted version of the PMI matrix into the two embedding matrices.

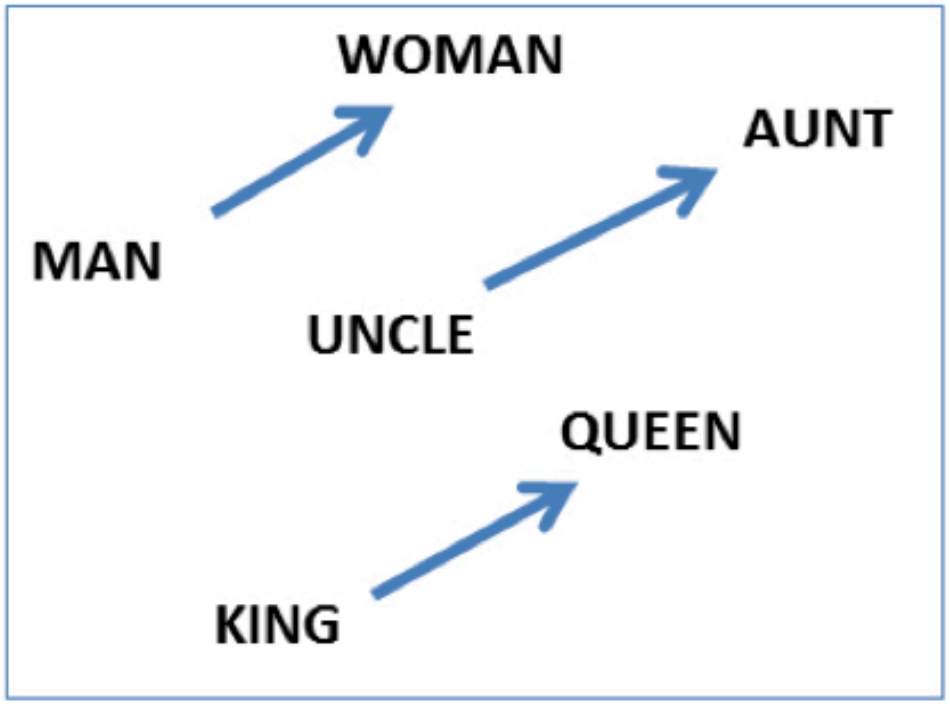
Properties of embeddings

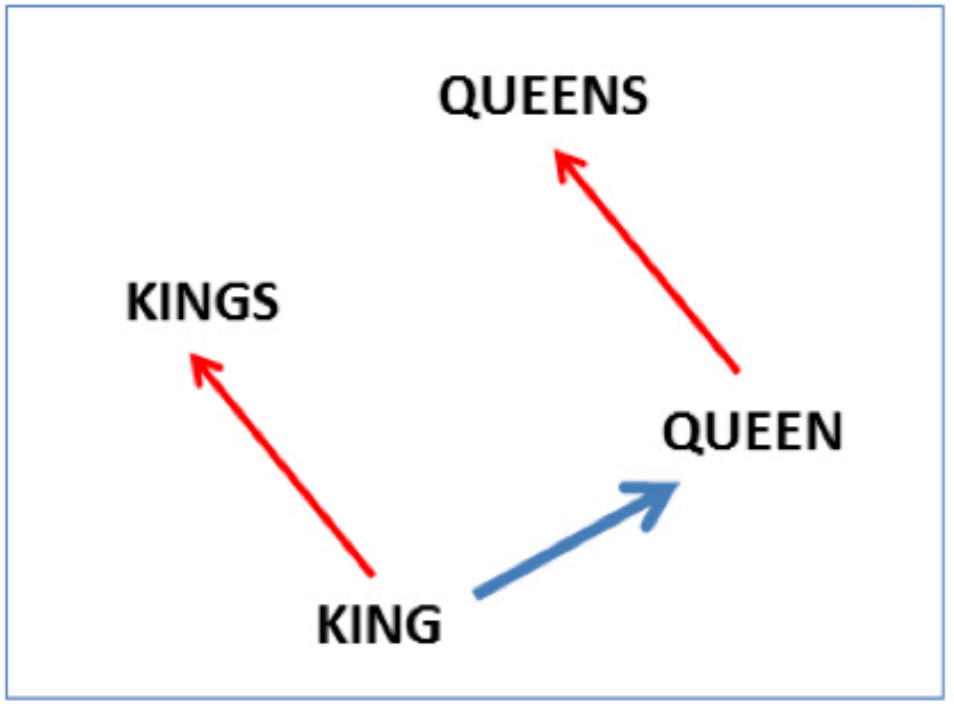
Nearest words to some embeddings (Mikolov et al. 20131)

target:	Redmond	Havel	ninjutsu	graffiti	capitulate
	Redmond Wash.	Vaclav Havel	ninja	spray paint	capitulation
	Redmond Washington	president Vaclav Havel	martial arts	grafitti	capitulated
	Microsoft	Velvet Revolution	swordsmanship	taggers	capitulating

Embeddings capture relational meaning!

vector(‘*king*’) - vector(‘*man*’) + vector(‘*woman*’) ≈ vector(‘queen’)
vector(‘*Paris*’) - vector(‘*France*’) + vector(‘*Italy*’) ≈ vector(‘Rome’)





Vector Semantics

Brown clustering (ex cursus)

VRIJE UNIVERSITEIT BRUSSEL

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

We'll give a very brief sketch here.

Brown clustering algorithm

Each word is initially assigned to its own cluster.

We now consider consider merging each pair of clusters. 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.

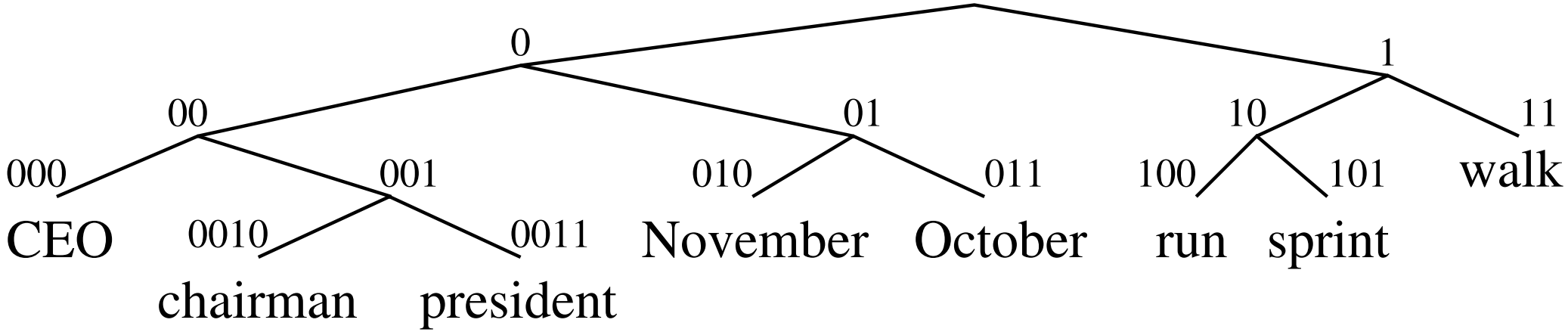
Brown Clusters as vectors

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

Homework:

- Installation of Babel2 (FCG)
- Try out the word2vec word embeddings in NLTK using gensim:
- https://radimrehurek.com/gensim/tutorial.html
- http://streamhacker.com/2014/12/29/word2vec-nltk/

Next week (Paul van Eecke):

- FCG tutorial (bring your computer)
- Explanation of third assignment