

# Vector Semantics and Embeddings

Berlin Chen

Department of Computer Science & Information Engineering  
National Taiwan Normal University

## References:

1. D. Jurafsky & J. H. Martin, *Speech and Language Processing* (3rd), Chapter 6 & Teaching Material

## Prologue (1/3)

荃者所以在魚，得魚而忘荃 Nets are for fish;  
Once you get the fish, you can forget the net.  
言者所以在意，得意而忘言 Words are for meaning;  
Once you get the meaning, you can forget the words

《莊子》第26章 外物

- **Distributional Hypothesis**: Words that occur in similar contexts tend to have similar meanings
    - This link between similarity in how words are distributed and similarity in what they mean is called the distributional hypothesis
  - The hypothesis was first formulated in the 1950s by linguists like Joos (1950), Harris (1954), and Firth (1957)
    - Notice that words which are synonyms (like **oculist** and **eye-doctor**) tended to occur in the same environment (e.g., near words like eye or examined)
    - The amount of meaning difference between two words “corresponding roughly to the amount of difference in their environments”
- 近朱者赤、近墨者黑？
- In this lecture, we discuss **vector semantics**, which instantiates this linguistic (distributional) hypothesis by learning **representations of the meaning of words**, called **embeddings**, directly from their distributions in texts

## Prologue (2/3)

- **Representation Learning**: automatically learn useful representations of the input text
  - Finding such **self-supervised ways** to learn representations of the input, instead of creating representations by hand via **feature engineering**, is an important focus of recent NLP research (Bengio et al., 2013)
  - It is not satisfactory that a word is represented as a string of letters, or an index in a vocabulary list
- **Vector Semantics: Categorization of Embeddings**

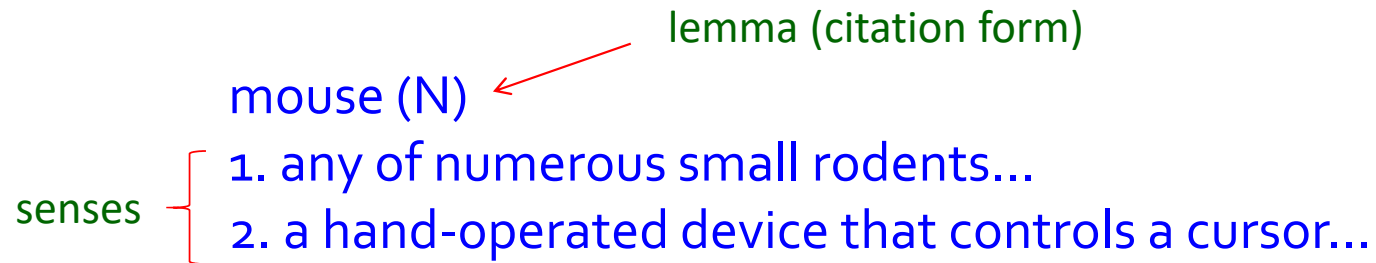
	Sparse	Dense
Static	tf-idf, PPMI, etc	Word2Vec, GloVe, fasttext, LSA, NMF, PLSA, LDA, etc.
Dynamic	---	Transformer, BERT, GPT, etc.

## Prologue (3/3)

- It is desirable that word embeddings can capture the fact, for example, that the meanings of **buy**, **sell**, and **pay** offer differing perspectives on the same underlying **purchasing event**
  - If I buy something from you, you've probably sold it to me, and I likely paid you
- More generally, **a model of word meaning** should allow us to draw inferences to address meaning-related downstream tasks like question-answering or dialogue

# Lemmas and Senses

- Let's look at how one word (we'll choose mouse) might be defined in a dictionary (simplified from the online dictionary WordNet)



The form *mouse* would also be the lemma for the word *mice*. Similarly, *sing* is the lemma for *sing*, *sang*, *sung*.

- A **sense** or “**concept**” is the meaning component of a word
- Lemmas can be **polysemous** (have multiple senses), e.g., **bass**
- **Word Sense Disambiguation (WSD)**: the task of determining which sense of a word is being used in a particular context

# Relations between Senses: Synonymy

- **Synonymy (同義)**: when one word has a sense whose meaning is identical to a sense of another word, or nearly identical, we say the two senses of those two words are **synonyms (同義詞)**

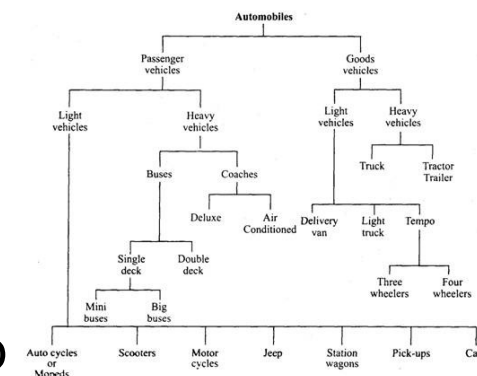
- A more formal definition of synonymy (between words rather than senses): two words are synonymous **if they are substitutable for one another in any sentence without changing the truth conditions of the sentence**, the situations in which the sentence would be true

*car / automobile    water / H<sub>2</sub>O    big / large    couch / sofa*



- Probably no two words are absolutely identical in meaning (**principle of contrast**)
- The word **synonym** is therefore used to describe a relationship of approximate or rough **synonymy**

*H<sub>2</sub>O* in a hiking guide?  
*my **big** sister ≠ my **large** sister*



# Word Similarity

*coffee vs. tea*  
*cat vs. dog*

- While words do not have many synonyms, most words do have lots of similar words
  - **cat** is not a synonym of **dog**, but cats and dogs are certainly similar words (both are pets)
- Knowing how similar two words are can help in computing how similar the meaning of two phrases or sentences are (**useful for many NLP tasks**)

Word 1	Word 2	Similarity
vanish	disappear	9.8
behave	obey	7.3
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

One way of getting values for word similarity is to ask humans to judge how similar one word is to another. For example the SimLex-999 dataset (Hillet al., 2015) gives values on a scale from 0 to 10.

## Word Relatedness (1/2)

- Also called “Word Association” in psychology
- The meaning of two words can be related in ways other than similarity

*coffee and cup*

*scalpel and surgeon*

- One common kind of relatedness between words is if they belong to the same **semantic field** or **semantic frame/role**
- **Semantic Field**: a set of words which cover a particular semantic domain and bear structured relations with each other

### **Hospitals:**

*surgeon, scalpel, nurse, anaesthetic, hospital*

### **Restaurants:**

*waiter, menu, plate, food, menu, chef*

### **Houses:**

*door, roof, kitchen, family, bed*

- Semantic fields are also related to **topic models**, both of which are very useful tools for discovering topical structure in documents



## Word Relatedness (2/2)

- A **semantic frame/role** is a set of words that denote perspectives or participants in a particular type of event
  - E.g., a **commercial transaction event** can be encoded lexically by using verbs like **buy** (the event from the perspective of the buyer), **sell** (from the perspective of the seller), **pay** (focusing on the monetary aspect), or nouns like **buyer**
  - Frames have semantic roles (like **buyer**, **seller**, **goods**, **money**), and words in a sentence can take on these roles

## Sentiment (1/2)

- Words have **affective meanings** or **connotations** 言外之意
  - Positive connotations (*happy*)
  - Negative connotations (*sad*)
- The word connotation has different meanings in different fields, but here we use it to mean the aspects of a word's meaning that are related to a writer or reader's **emotions**, **sentiment**, **opinions**, or **evaluations**

Connotations can be subtle:

- Positive connotation: *copy, replica, reproduction*
- Negative connotation: *fake, knockoff, forgery*
- Some words describe positive evaluation (*great, love*) and others negative evaluation (*terrible, hate*)
- Positive or negative **evaluation language** is called **sentiment**
  - Word sentiment plays a role in important tasks like sentiment analysis, stance (立場) detection, and applications of NLP to the language of politics and consumer reviews

## Sentiment (2/2)

- Early work on affective meaning (Osgood et al., 1957) found that words varied along three important dimensions of affective meaning
  - **valence** (效價): the pleasantness of the stimulus
  - **arousal** (喚起): the intensity of emotion provoked by the stimulus
  - **dominance** (支配): the degree of control exerted by the stimulus

	Word	Score		Word	Score
Valence	love	1.000		Toxic (令人不快的、惡毒的)	0.008
	happy	1.000		nightmare	0.005
Arousal	elated (興高采烈的)	0.960		mellow (令人愉快的)	0.069
	frenzy (極度的激動)	0.965		napping (昏昏欲睡的)	0.046
Dominance	powerful	0.991		weak	0.045
	leadership	0.983		empty	0.081

Values from NRC VAD Lexicon  
(Mohammad 2018)

## Vector Semantics (1/2)

- **Vector semantics** is the standard way to represent word meaning in NLP, helping us model many of the aspects of word meaning we discussed above
  - Vectors for representing words are called **embeddings**
    - Notice that the meaning of **embedding in mathematics** is a mapping from one space or structure to another, although this meaning has shifted
- The idea of vector semantics is to represent a word as a point in a multidimensional semantic space
  - Which is derived from the distributions of word neighbors
  - Similar words are “nearby in semantic space”
  - Each word is a vector, not just a string like “apple” or an index like “ $w_{60}$ ”

## Vector Semantics (2/2)

- A visualization of embeddings learned for **sentiment analysis**, showing the location of selected words projected down from 60-dimensional space into a two dimensional space
  - Notice the distinct regions containing positive words, negative words, and neutral function words



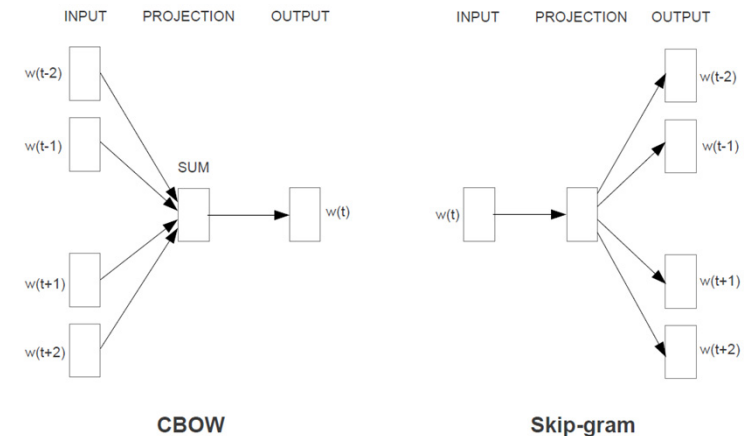
# Two Kinds of Word Embeddings

- **tf-idf**

- A common baseline model and frequently used in information retrieval (IR)
- (static) **sparse** vectors
- The meaning of words are defined by a simple function of the counts of nearby words

- **Word2vec**

- (static) **dense** vectors
- Word representation is created by training a classifier to **predict** whether a word is likely to appear nearby
- A further extensions of word2vec called **contextual embeddings** (like **BERT**)



## Term-Document Matrix (1/2)

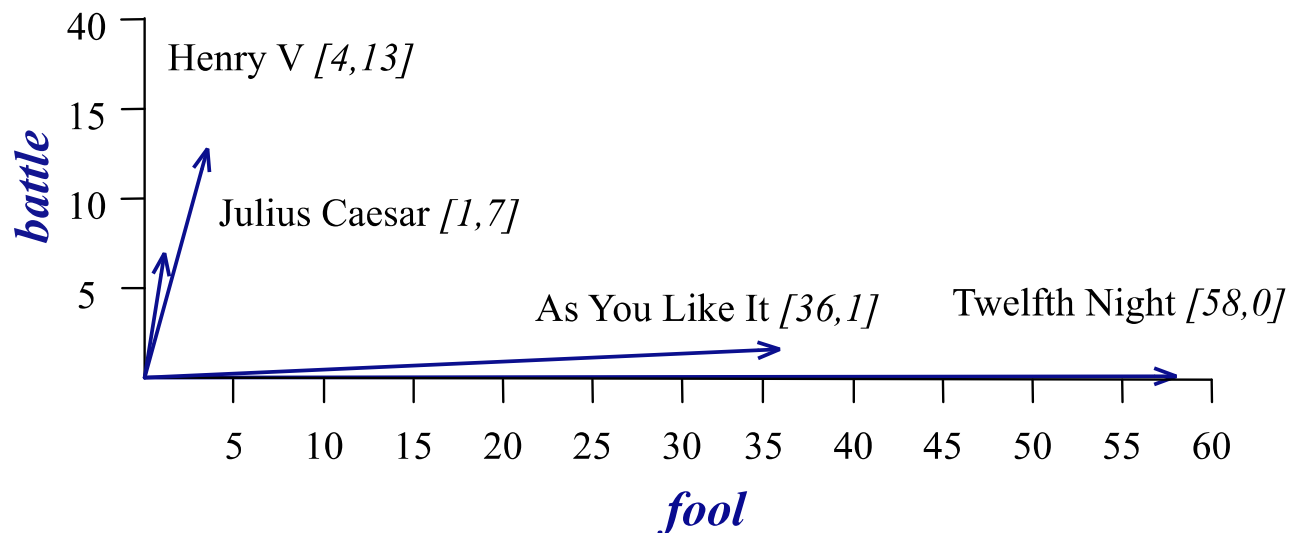
- Each document is represented by a vector of words
  - Imagine we have a collection of documents, such as all the works of Shakespeare
  - We can represent documents in such a collection by **a term-document matrix**
  - In this matrix each row represents a word in the vocabulary and each column represents a document from the collection

	As You Like It	Twelfth Night	Julius Caesar	Henry V
<b>battle</b>	1	0	7	13
<b>good</b>	114	80	62	89
<b>fool</b>	36	58	1	4
<b>wit</b>	20	15	2	3

A simplified term-document matrix for four plays by Shakespeare.

## Term-Document Matrix (2/2)

- A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*



- The comedies (*As You Like It* and *Twelfth Night*) have high values for the *fool* dimension and low values for the *battle* dimension



## Vectors are the Basis of Information Retrieval

- Vectors are similar for the two comedies

	<b>As You Like It</b>	<b>Twelfth Night</b>	<b>Julius Caesar</b>	<b>Henry V</b>
<b>battle</b>	1	0	7	13
<b>good</b>	114	80	62	89
<b>fool</b>	36	58	1	4
<b>wit</b>	20	15	2	3

- Meanwhile, comedies are different than the other two
  - Comedies have more fools and wits and fewer battles

## Words as Vectors: Document Dimensions

- The term-document matrix also allows us to represent the meaning of a word by the documents it tends to occur in
  - Associating each word with a **word vector** (a **row vector** rather than a column vector)

	As You Like It	Twelfth Night	Julius Caesar	Henry V
<b>battle</b>	1	0	7	13
<b>good</b>	114	80	62	89
<b>fool</b>	36	58	1	4
<b>wit</b>	20	15	2	3

- *battle* is the kind of word that occurs in Julius Caesar and Henry V
- *fool* is the kind of word that occurs in comedies, especially Twelfth Night

## Words as Vectors: Word Dimensions (1/2)

- As an alternative, we can use the **term-term matrix**, also called the **word-word matrix** or the **term-context matrix** to derive word embeddings

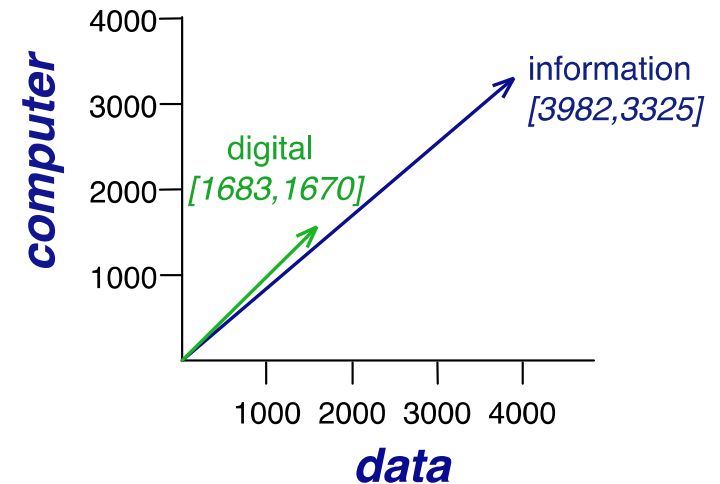
	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

- This matrix is thus of dimensionality  $|V| \times |V|$  and each cell records the number of times the row (target) word and the column (context) word co-occur in some context in some training corpus
- Two words are similar in meaning if their context vectors are similar (e.g., *digital* and *information* are more similar to each other)

is traditionally followed by **cherry** pie, a traditional dessert  
often mixed, such as **strawberry** rhubarb pie. Apple pie  
computer peripherals and personal **digital** assistants. These devices usually  
a computer. This includes **information** available on the internet

## Words as Vectors: Word Dimensions (2/2)

- A spatial visualization of word vectors for *digital* and *information*, showing just two of the dimensions, corresponding to the words *data* and *computer*



- **Rule of Thumb:** In real life, the length of the vector  $|V|$ , is generally the size of the vocabulary, often between 10,000 and 50,000 words
  - Namely, using the most frequent words in the training corpus
  - Keeping words after about the most frequent 50,000 or so is generally not helpful
- Most of those numbers in the term-term matrix are zero these are **sparse vector representations**
  - There are efficient algorithms for storing and computing with sparse matrices

## Dot Product for Measuring Similarity (1/2)

- The **dot product** between two vectors is a scalar:

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

- The dot product tends to be high when the two vectors have large values in the same dimensions
- Dot product can thus be a useful similarity metric between vectors
- Alternatively, vectors that have zeros in different dimensions—**orthogonal vectors**—will have a dot product of 0, representing their strong dissimilarity

## Dot Product for Measuring Similarity (2/2)

- Problem with raw dot-product
  - Dot product favors long vectors; dot product is higher if a vector is longer (namely, with higher values in many dimension)

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

Vector Length

- Frequent words (of, the, you) have long vectors, since they occur many times with other words
- Therefore, dot product overly favors frequent words when using it as a measure for word similarity
- It is desirable to have a similarity metric that tells us how similar two words are regardless of their frequency

## Cosine for Measuring Similarity (1/2)

- **Cosine Metric**: a modification of dot product which normalizes for the vector length by dividing the dot product by the lengths of each of the two vectors
- The cosine similarity metric between two vectors  $\mathbf{v}$  and  $\mathbf{w}$

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

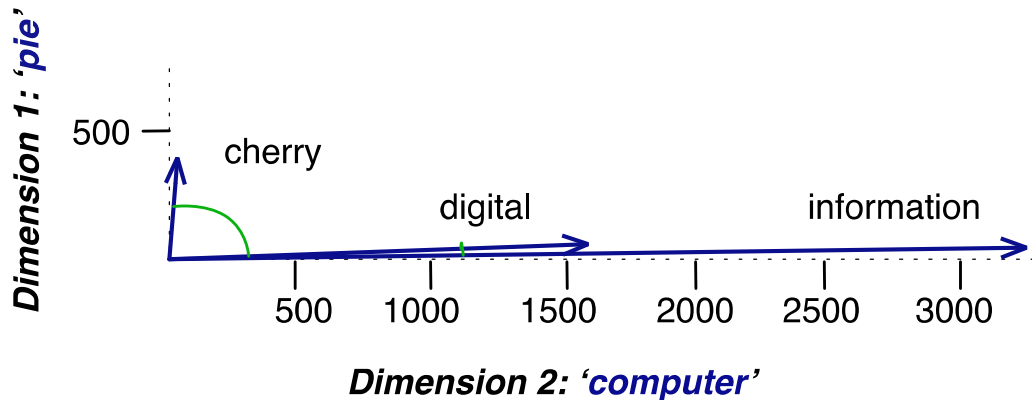
- $-1$  : vectors point in opposite directions
  - $+1$  : vectors point in same directions
  - $0$  : vectors are orthogonal
- Since raw frequency values are non-negative, the cosine for term-term matrix vectors ranges from 0–1

## Cosine for Measuring Similarity (2/2)

	pie	data	computer
cherry	442	8	2
digital	5	1,683	1,670
information	5	3,982	3,325

$$\cos(\text{cherry, information}) = \frac{442 * 5 + 8 * 3982 + 2 * 3325}{\sqrt{442^2 + 8^2 + 2^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .017$$

$$\cos(\text{digital, information}) = \frac{5 * 5 + 1683 * 3982 + 1670 * 3325}{\sqrt{5^2 + 1683^2 + 1670^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .996$$



A (rough) graphical demonstration of cosine similarity, showing vectors for the words **cherry**, **digital**, and **information** in the two dimensional space defined by counts of the words **computer** and **pie** nearby. Note that the angle between **digital** and **information** is smaller than the angle between **cherry** and **information**.



## Paradox of Raw Frequency based Representations

- The co-occurrence matrices we have seen represent each cell by word frequencies
  - Frequency is clearly useful; if *sugar* appears a lot near *apricot* (杏仁), that is useful information
  - But overly frequent words like *the*, *it*, or *they* are not very informative about the context
  - How can we balance these two conflicting constraints?

## Two Common Solutions for Word Weighting

- **tf-idf**: tf-idf value for word  $t$  in document  $d$

$$\text{tf-idf}(t, d) = \text{tf}_{t,d} \times \text{idf}_t$$

Words like “the” or “it” have very low idf.

- **PMI**: pointwise mutual information

$$\text{PMI}(w_i, w_j) = \log \frac{P(w_i, w_j)}{P(w_i)P(w_j)}$$

To see if words like "good" appear more often with "great" than we would expect by chance.

## Term frequency (tf)

- Term frequency ([Luhn, 1957](#)) is the raw frequency of the word  $t$  in the document  $d$

$$\text{tf}_{t,d} = \text{count}(t, d)$$

- Instead of using raw count, we squash it a bit

$$\text{tf}_{t,d} = \log_{10}(\text{count}(t, d) + 1) \quad \text{or}$$

$$\text{tf}_{t,d} = \begin{cases} 1 + \log_{10}(\text{count}(t, d)) & \text{if } \text{count}(t, d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

- The intuition is that a word appearing 100 times in a document does not make that word 100 times more likely to be relevant to the meaning of the document
- Because we cannot take the log of 0, we normally add 1 to the count

## Document Frequency (df)

- The second factor in tf-idf is used to give a higher weight to words that occur only in a few documents
  - Terms that are limited to a few documents are useful for discriminating those documents from the rest of the collection
  - Terms that occur frequently across the entire collection are not as helpful
- The **document frequency**  $df_t$  of a term  $t$  is the number of documents it occurs in
  - Document frequency is not the same as the **collection frequency** of a term, which is the total number of times the word appears in the whole collection in any document

	<b>Collection Frequency</b>	<b>Document Frequency</b>
Romeo	113	1
action	113	31

Statistics are accumulated from the collection of Shakespeare's 37 plays.

# Inverse Document Frequency (idf)

- Accordingly, we can emphasize discriminative words like Romeo via the inverse document frequency (idf) term weight (Sparck Jones, 1972)

$$itf_t = \log_{10} \left( \frac{N}{df_t} \right) \quad \text{or} \quad itf_t = \log_{10} \left( 1 + \frac{N}{df_t} \right)$$

- $N$  is the total number of documents in the collection

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

Raw Counts

	As You Like It	Twelfth Night	Julius Caesar	Henry V
<b>battle</b>	1	0	7	13
<b>good</b>	114	80	62	89
<b>fool</b>	36	58	1	4
<b>wit</b>	20	15	2	3

tf-idf

	As You Like It	Twelfth Night	Julius Caesar	Henry V
<b>battle</b>	0.074	0	0.22	0.28
<b>good</b>	0	0	0	0
<b>fool</b>	0.019	0.021	0.0036	0.0083
<b>wit</b>	0.049	0.044	0.018	0.022

## Pointwise Mutual Information (PMI)

- Pointwise Mutual Information (Fano, 1961) is one of the most important concepts in NLP
- PMI can be used as a measure of how often two word  $w_i$  and  $w_j$  occur, compared with what we would expect if they were independent

$$\text{PMI}(w_i, w_j) = \log_2 \frac{P(w_i, w_j)}{P(w_i)P(w_j)} = \log_2 \frac{P(w_i|w_j)}{P(w_i)} \begin{matrix} > \\ = \\ < \end{matrix} 0 (?)$$

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

San Francisco  $\log_2 \frac{P(\text{Francisco}|\text{San})}{P(\text{Francisco})}$

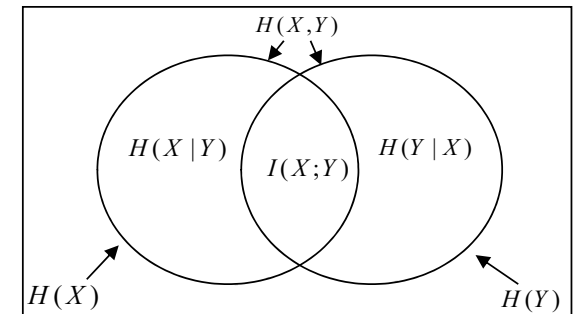
Santa Barbara

## Recall: Mutual Information (MI)

- MI is the **information reduction in uncertainty of one random variable due to knowing about another**, or in other words, the amount of information one random variable contains about another

$$I(X; Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

$$\begin{aligned} I(X; Y) &= \sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \\ &= \mathbf{E} \left[ \log \frac{p(x,y)}{p(x)p(y)} \right] \end{aligned}$$



- MI is a **symmetric, non-negative** measure of the common information in the two variables
  - MI is 0 only when two variables are independent

## Positive Pointwise Mutual Information (PPMI)

- The negative values of PMI are problematic
  - Things are co-occurring less than we expect by chance
  - Unreliable without enormous corpora
  - Imagine  $w_i$  and  $w_j$  whose probability is each  $10^{-6}$ 
    - Hard to be sure  $P(w_i, w_j)$  is significantly different than  $10^{-12}$
    - Further, it is not clear people are good at “unrelatedness”
- **Positive PMI (PPMI)**: we hence just replace negative PMI values by 0

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



## Computing PPMI on a Term-Context Matrix (1/2)

- Given a matrix  $F$  with  $|V|$  rows (words) and  $|C|$  columns (contexts)
  - $f_{ij}$  is the of times  $f_i$  occurs in context  $c_j$

$$P_{ij} = \frac{f_{ij}}{\sum_i |V| \sum_j |C| f_{ij}}$$

$$P_{i*} = P(w_i) = \frac{\sum_j |C| f_{ij}}{\sum_i |V| \sum_j |C| f_{ij}}$$

$$P_{*j} = P(c_j) = \frac{\sum_i |V| f_{ij}}{\sum_i |V| \sum_j |C| f_{ij}}$$

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

target word

context word

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

Co-occurrence counts for four words in 5 contexts in the Wikipedia corpus, together with the marginals, pretending for the purpose of this calculation that no other words/contexts matter.

	p(w,context)					p(w)
	computer	data	result	pie	sugar	p(w)
cherry	0.0002	0.0007	0.0008	0.0377	0.0021	0.0415
strawberry	0.0000	0.0000	0.0001	0.0051	0.0016	0.0068
digital	0.1425	0.1436	0.0073	0.0004	0.0003	0.2942
information	0.2838	0.3399	0.0323	0.0004	0.0011	0.6575
p(context)	0.4265	0.4842	0.0404	0.0437	0.0052	

## Computing PPMI on a Term-Context Matrix (2/2)

- Resulting PPMI matrix (negatives are replaced by 0)

	computer	data	result	pie	sugar
cherry	0	0	0	4.38	3.30
strawberry	0	0	0	4.10	5.51
digital	0.18	0.01	0	0	0
information	0.02	0.09	0.28	0	0

$$\begin{aligned}
 & \text{PPMI}(w_i = \text{information}, c_j = \text{data}) \\
 &= \max\left(\log_2 \frac{.3399}{.6575 \times 0.4842}, 0\right) \\
 &= .0944
 \end{aligned}$$

	p(w,context)					p(w)
	computer	data	result	pie	sugar	p(w)
cherry	0.0002	0.0007	0.0008	0.0377	0.0021	0.0415
strawberry	0.0000	0.0000	0.0001	0.0051	0.0016	0.0068
digital	0.1425	0.1436	0.0073	0.0004	0.0003	0.2942
information	0.2838	0.3399	0.0323	0.0004	0.0011	0.6575
p(context)	0.4265	0.4842	0.0404	0.0437	0.0052	

## Weighted PPMI

- PMI and PPMI have the problem of being biased toward infrequent events
  - Very rare words tend to have very high PMI values
- Two solutions:
  - Give rare words slightly higher probabilities
  - Use add-one smoothing (which has a similar effect)

$$\text{PPMI}_\alpha(w_i, c_j) = \max\left(\log_2 \frac{P(w_i, c_j)}{P(w_i)P_\alpha(c_j)}, 0\right)$$

$$P_\alpha(c_j) = \frac{\text{count}(c_j)^\alpha}{\sum_j^{|C|} \text{count}(c_j)^\alpha} = \frac{\sum_j^{|V|} f_{ij}^\alpha}{\sum_i^{|V|} \sum_j^{|C|} f_{ij}^\alpha}$$

- For example,  $\alpha = 0.75$

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

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

## Sparse Vectors vs. Dense Vectors (1/2)

- tf-idf (or PMI/PPMI) vectors are
  - **long** (with number of dimensions  $|V|= 20,000$  to  $50,000$ )
  - **sparse** (most elements are zero)
- Alternatively, we may learn vectors which are
  - **short** (with number of dimensions 50-1000)
  - **dense** (most elements are non-zero, real-valued numbers that can be negative)

## Sparse Vectors vs. Dense Vectors (2/2)

- Why dense vectors?
  - Short vectors may be easier to use as **features** in machine learning (far fewer weights of an ML model to tune)
  - Dense vectors may **generalize** better than explicit counts
  - Dense vectors may do better at capturing synonymy:
    - **car** and **automobile** are synonyms, whereas they are indexed by distinct dimensions of sparse vectors (e.g., tf-idf vectors)
      - a word with **car** as a neighbor and a word with **automobile** as a neighbor should be similar, but are not for sparse vector representations

It turns out that dense vectors work better in NLP tasks than sparse vectors.

## Common Methods for Getting Short Dense Vectors

- Singular Value Decomposition (SVD)
  - A well-studied instantiation of this is Latent Semantic Analysis (LSA)
- “Neural Language Model”-inspired models
  - [Word2vec \(skip-gram, CBOW\)](#) (Mikolov et al. 2013), [GloVe](#) (Pennington et al., 2014).
- Alternative to these “static embeddings” are more recent methods for learning dynamic contextual embeddings
  - Contextual Embeddings (ELMo, BERT)
  - Compute distinct embeddings for a word in its context
  - Separate embeddings for each token of a word

For all these methods, each dimension of a word embedding does not have a clear interpretation.

## Publicly-available Tools for Simple Static Embeddings

- Word2vec (Mikolov et al.), Google  
<https://code.google.com/archive/p/word2vec/>
- GloVe (Pennington et al.), Stanford  
<http://nlp.stanford.edu/projects/glove/>

Notice that: The use of dense vectors to model word meaning, and indeed the term embedding, grew out of the latent semantic indexing (LSI) model (Deerwester et al., 1988) recast as LSA (latent semantic analysis) (Deerwester et al., 1990).

## Basic Notion of Word2vec [Google](#)

- Word2vec embeddings are static static beddings, meaning that the method learns one fixed embedding for each word in the vocabulary
- The intuition of word2vec is that instead of counting how often each **word**  $w$  occurs near, say, **apricot** (杏仁), we'll instead train a classifier on a binary prediction task: “Is **word**  $w$  likely to show up near **apricot**?”
  - Such a learning paradigm avoids the need for any sort of hand-labeled supervision signals, often called **self-supervision**
  - This notion was first proposed in the task of **neural language modeling (NLM)** (Bengio et al., 2003; Collobert et al. 2011), showing that NLM could just use the next word in running text as its supervision signal
  - NLM can could be used to learn an embedding representation for each word as part (viz. **byproduct**) of doing this prediction task



## Word2Vec: Learning skip-gram Embeddings (1/8)

- The intuition of skip-gram

1. Treat the target **word**  $w$  and a neighboring context **word**  $c$  as **positive examples**
2. Randomly sample other words in the lexicon to get **negative examples**; namely *skip-gram* with negative sampling (SGNS)
3. Use **logistic regression** to train a classifier to distinguish those two cases
4. Use the learned weights as the embeddings

Assume a +/- 2 word window, given a training sentence:

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

$c_1$        $c_2$        $w$        $c_3$        $c_4$

## Word2Vec: Learning skip-gram Embeddings (2/8)

- Goal: train a classifier that is given a candidate (**word**, **context**) pair
  - For example, (apricot杏仁, jam 果醬) and (apricot, aardvark 土豚), while it is expected to return the probability where  $c$  is a real context word or not (true for jam, false for aardvark)
  - Assign each positive (or negative) pair a probability

$$P(+|w, c)$$

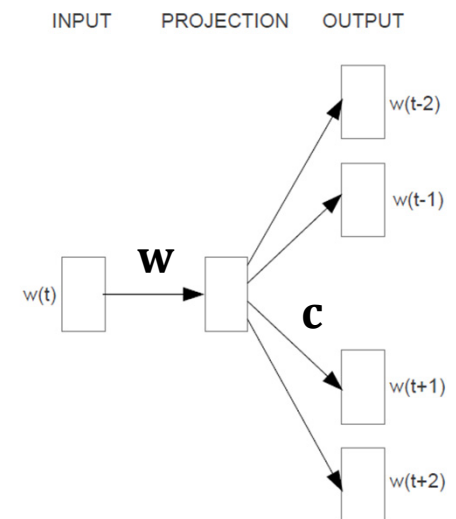
Bernoulli distribution

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

- These probability terms can be computed based on a suitable similarity metric, such as dot product

$$\text{Similarity}(w, c) = \mathbf{c} \cdot \mathbf{w}$$

- Which has a value ranging from  $-\infty$  to  $\infty$



# Word2Vec: Learning skip-gram Embeddings (3/8)

- To turn the dot product into a probability, we use the logistic or sigmoid function  $\sigma(\cdot)$  the fundamental core of logistic regression

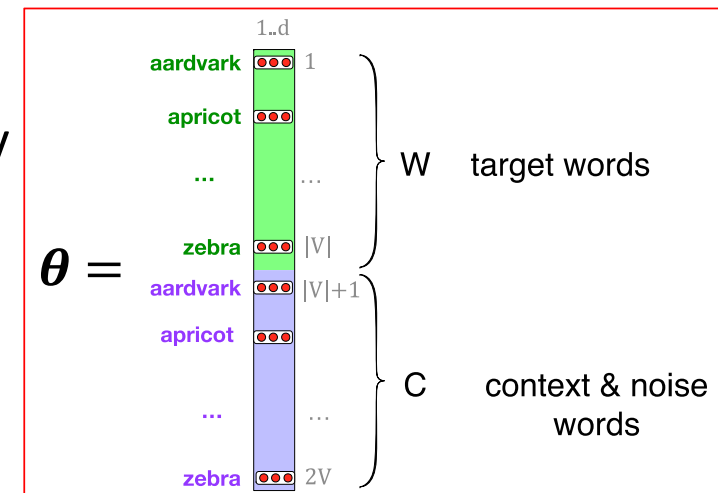
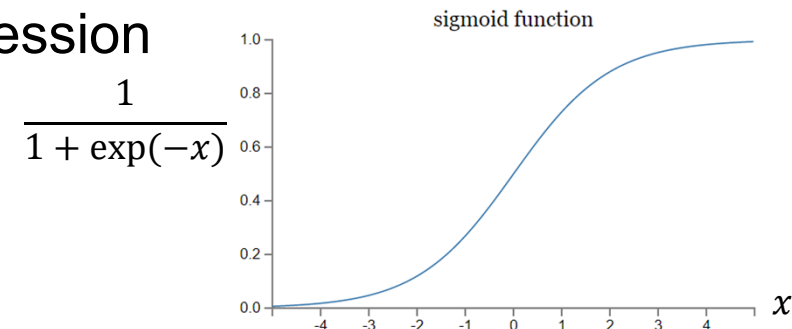
$$P(+|w, c) = \sigma(\mathbf{c} \cdot \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{c} \cdot \mathbf{w})}$$

$$P(-|w, c) = 1 - P(+|w, c) \stackrel{?}{=} \sigma(-\mathbf{c} \cdot \mathbf{w}) = \frac{1}{1 + \exp(\mathbf{c} \cdot \mathbf{w})}$$

- This is for one context word, but we have lots of context words. We then assume independence and just multiply them

$$P(+|w, c_{1:L}) = \prod_{i=1}^L P(+|w, c_i) = \prod_{i=1}^L \sigma(\mathbf{c}_i \cdot \mathbf{w})$$

$$\log P(+|w, c_i) = \sum_{i=1}^L \log \sigma(\mathbf{c}_i \cdot \mathbf{w})$$



## Brief Explanation

$$\begin{aligned}P(-|w, c) &= 1 - P(+|w, c) \\&= 1 - \sigma(\mathbf{c} \cdot \mathbf{w}) \\&= 1 - \frac{1}{1 + \exp(-\mathbf{c} \cdot \mathbf{w})} \\&= \frac{\exp(-\mathbf{c} \cdot \mathbf{w})}{1 + \exp(-\mathbf{c} \cdot \mathbf{w})} \\&= \frac{1}{1 + \frac{1}{\exp(-\mathbf{c} \cdot \mathbf{w})}} \\&= \frac{1}{1 + \exp(\mathbf{c} \cdot \mathbf{w})}\end{aligned}$$

## Word2Vec: Learning skip-gram Embeddings (4/8)

- skip-gram with negative sampling (SGNS)

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

$c_1$        $c_2$        $w$        $c_3$        $c_4$

### positive examples +

$w$	$c_{\text{pos}}$
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

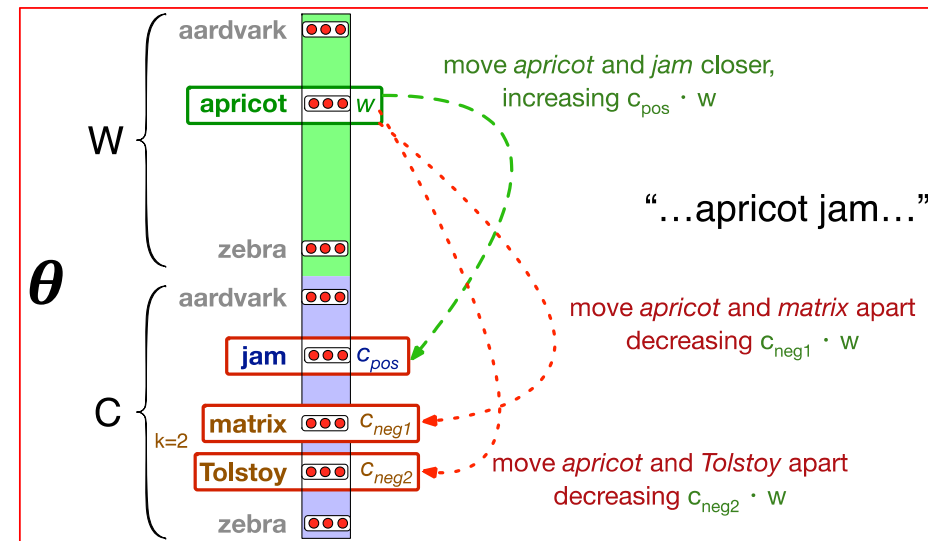
### negative examples -

$w$	$c_{\text{neg}}$	$w$	$c_{\text{neg}}$
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

## Word2Vec: Learning skip-gram Embeddings (5/8)

- skip-gram with negative sampling (SGNS)
  - Maximize the similarity of the target with the actual context words, and minimize the similarity of the target with the  $k$  negative sampled non-neighbor words

$$\begin{aligned}
 L_{CE} &= -\log \left[ P(+|w, c_{pos}) \prod_{i=1}^k P(-|w, c_{neg_i}) \right] \\
 &= -\left[ \log P(+|w, c_{pos}) + \sum_{i=1}^k \log P(-|w, c_{neg_i}) \right] \\
 &= -\left[ \log P(+|w, c_{pos}) + \sum_{i=1}^k \log(1 - P(+|w, c_{neg_i})) \right] \\
 &= -\left[ \log \sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) + \sum_{i=1}^k \log \sigma(-\mathbf{c}_{neg_i} \cdot \mathbf{w}) \right]
 \end{aligned}$$



# Word2Vec: Learning skip-gram Embeddings (6/8)

- The derivatives of the loss function

$$L_{CE} = - \left[ \log \sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) + \sum_{i=1}^k \log \sigma(-\mathbf{c}_{neg_i} \cdot \mathbf{w}) \right]$$

$$\frac{\partial L_{CE}}{\partial \mathbf{c}_{pos}} = \underbrace{[\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1]}_{?} \mathbf{w}$$

Start with randomly initialized C and W matrices, then incrementally do updates.

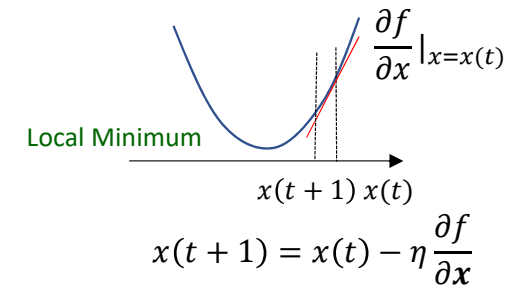
$$\frac{\partial L_{CE}}{\partial \mathbf{c}_{neg_i}} = \underbrace{\sigma(\mathbf{c}_{neg_i} \cdot \mathbf{w})}_{?} \mathbf{w}$$

$$\frac{\partial L_{CE}}{\partial \mathbf{w}} = [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1] \mathbf{c}_{pos} + \sum_{i=1}^k \sigma(\mathbf{c}_{neg_i} \cdot \mathbf{w}) \mathbf{c}_{neg_i}$$

$$\mathbf{c}_{pos}^{t+1} = \mathbf{c}_{pos}^t - \eta [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1] \mathbf{w}^t$$

$$\mathbf{c}_{neg_i}^{t+1} = \mathbf{c}_{neg_i}^t - \eta \sigma(\mathbf{c}_{neg_i} \cdot \mathbf{w}) \mathbf{w}^t$$

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \left[ [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1] \mathbf{c}_{pos} + \sum_{i=1}^k \sigma(\mathbf{c}_{neg_i} \cdot \mathbf{w}) \mathbf{c}_{neg_i} \right]$$

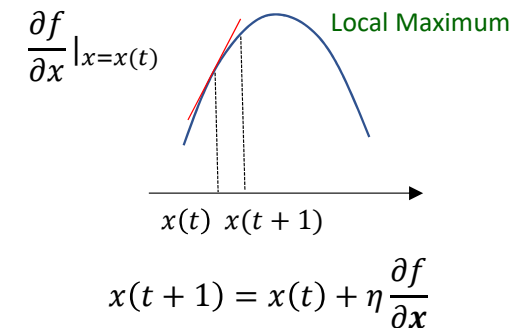


Gradient descent is based on the observation that if the **multi-variable function**  $F(\mathbf{x})$  is **defined** and **differentiable** in a neighborhood of a point  $\mathbf{a}$ , then  $F(\mathbf{x})$  decreases *fastest* if one goes from  $\mathbf{a}$  in the direction of the negative **gradient** of  $F$  at  $\mathbf{a}$ ,  $-\nabla F(\mathbf{a})$ . It follows that, if

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$$

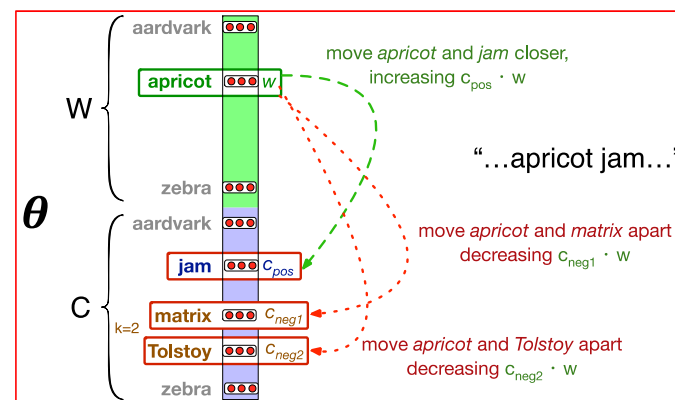
for a small enough step size or **learning rate**  $\gamma \in \mathbb{R}_+$ .

[https://en.wikipedia.org/wiki/Gradient\\_descent](https://en.wikipedia.org/wiki/Gradient_descent)



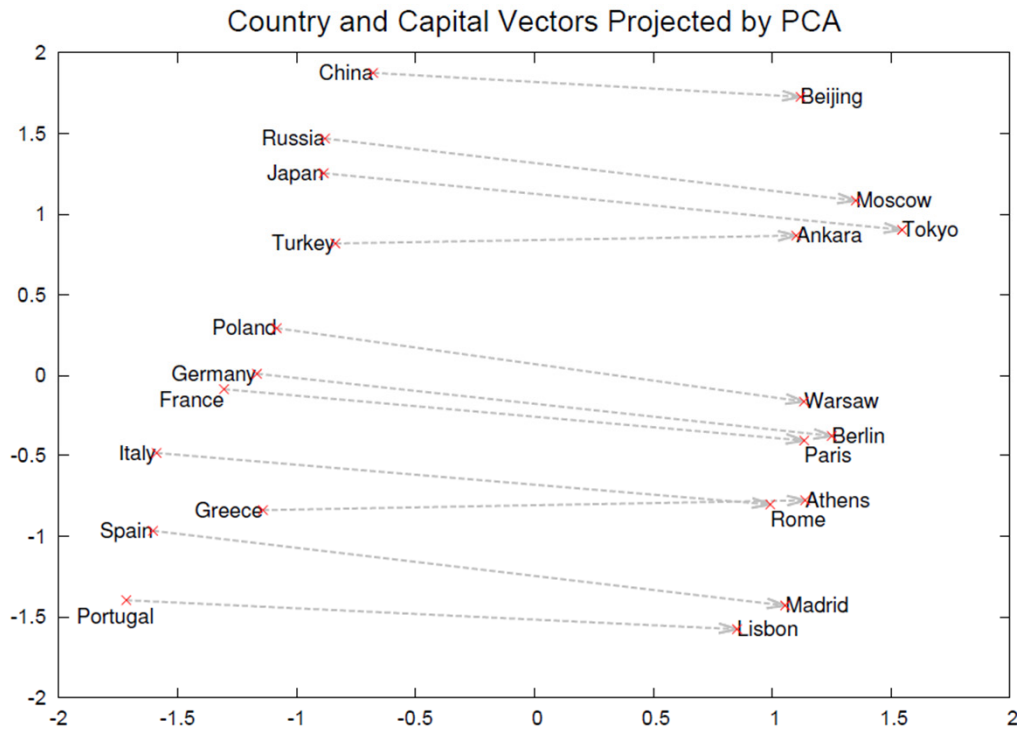
## Word2Vec: Learning skip-gram Embeddings (7/8)

- Recall that the skip-gram model learns two separate embeddings for each word  $w_i$ 
  - The target embedding  $w_i$  and the context embedding  $c_i$ , stored in two matrices, the target matrix  $W$  and the context matrix  $C$
- It is common to just add them together, representing word  $w_i$  with the vector  $w_i + c_i$
- Alternatively we can throw away the context matrix  $C$  and just represent each word  $w_i$  by the vector  $w_i$





# Word2Vec: Learning skip-gram Embeddings (8/8)



$$\mathbf{b}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \operatorname{distance}(\mathbf{x}, \mathbf{b} - \mathbf{a} + \mathbf{a}^*)$$

Rome      Paris      France      Italy

Rumelhart and Abrahamson (1973) proposed the parallelogram model for solving simple analogy problems of the form **b** is to **a** as what is to **a\***?

Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

# Word2Vec: skip-gram vs. CBOW

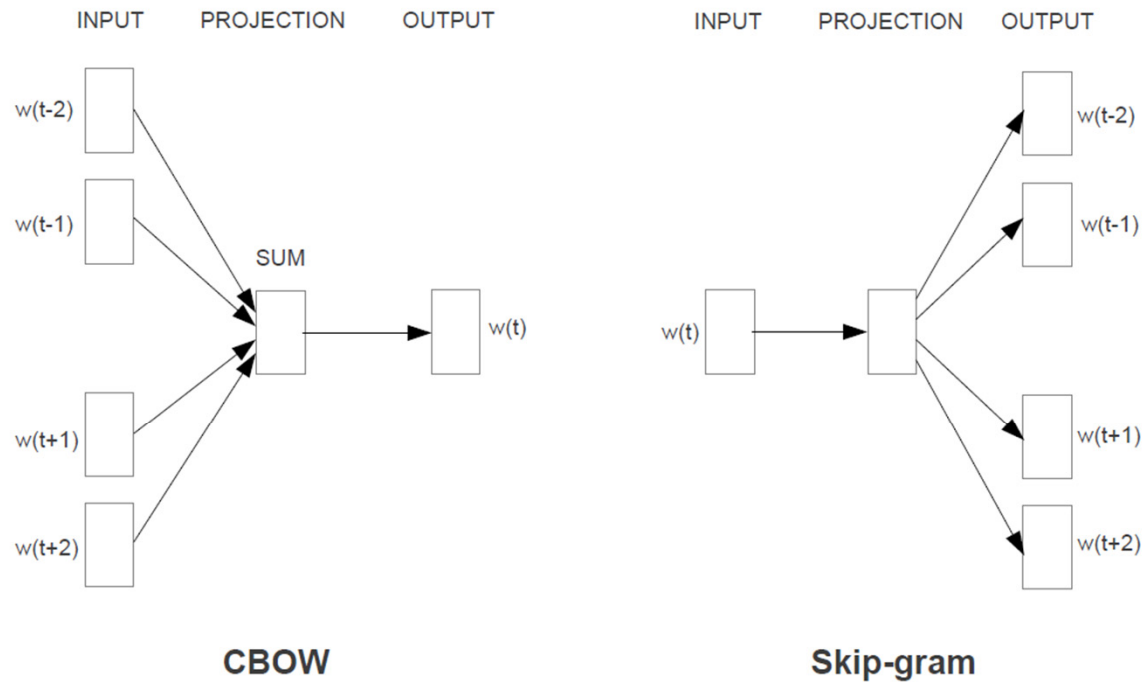


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

## Semantic Properties of Embeddings

- One parameter of vector semantic models (like word2vec vectors) is the size of the context window used to collect counts
- This is generally between 1 and 10 words on each side of the target word (for a total context of 2-20 words)
- The choice depends on the goals of the representation:
  - Shorter context windows tend to lead to representations that are a bit more syntactic, since the information is coming from immediately nearby words, namely similar words with the same parts of speech
    - “wrote” is a first-order (**syntagmatic**; 詞語的) associate of “book” or “poem”
  - When vectors are computed from long context windows, the highest cosine words to a target word  $w$  tend to be words that are topically related but not similar
    - “wrote” is a second-order (**paradigmatic**; 詞形變化的) associate of words like “said” or “remarked”

## Bias and Embeddings

- In addition to their ability to learn word meaning from text, word embeddings also reproduce the implicit **biases** and **stereotypes** (刻板印象) that were latent in the text
  - Bolukbasi et al. (2016) find that the closest occupation to ‘computer programmer’ - ‘man’ + ‘woman’ in word2vec embeddings trained on news text is ‘homemaker’
  - Also the embeddings similarly suggest the analogy ‘father’ is to ‘doctor’ as ‘mother’ is to ‘nurse’
- As pointed out by (Crawford 2017, Blodgett et al. 2020)
  - **Representation harm**: a harm caused by a system demeaning or even ignoring some social groups
  - **Allocation harm**: a system allocates resources (jobs or credit) unfairly to different groups
- Debiasing and harm reduction remain an open problem !

# Evaluating Vector Models (1/2)

- Extrinsic evaluation:
  - To see whether the adoption of vector models can improve the performance various downstream NLP tasks such as [summarization](#) and [information retrieval \(IR\)](#), etc.
- Intrinsic evaluation:
  - To see their performance on [word similarity](#) tasks, computing the correlation between an algorithm's word similarity scores and word similarity ratings assigned by humans
  - Or, to see their performance on the [word analogy](#) tasks

$$\mathbf{b}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \operatorname{distance}(\mathbf{x}, \mathbf{b} - \mathbf{a} + \mathbf{a}^*)$$

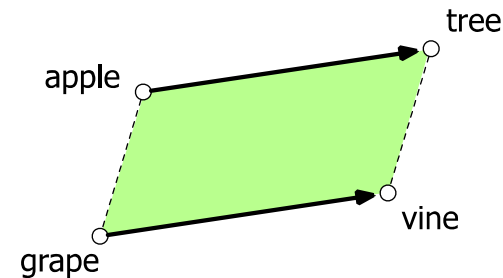
↑  
vine

children  
spout

tree    apple    grape

cities    city    child

leg    table    teapot



## Evaluating Vector Models (2/2)

- All embedding algorithms suffer from inherent variability:
  - For example, because of **randomness in the initialization** and **the random negative sampling**, algorithms like word2vec may produce different results even from the same dataset, and individual documents in a collection may strongly impact the resulting embeddings
- Possible mitigation strategy
  - When embeddings are used to study word associations in particular corpora, it is best practice to train multiple embeddings with bootstrap sampling over documents and average the results

# Vector Models for Extractive Summarization

- To leverage word/sentence/document embeddings for extractive document summarization (sentence ranking)

- **Vector Space Model** (Cosine Similarity Measure)

$$\text{SIM}(S, D) = \frac{\mathbf{v}_S \cdot \mathbf{v}_D}{|\mathbf{v}_S| |\mathbf{v}_D|} \quad \text{where} \quad \mathbf{v}_D = \frac{\sum_{w \in D} \mathbf{v}_w}{|D|}$$

- **Graph-based Model** (Centrality Measure)

$$\text{WS}(v_i) = (1 - \alpha) + \alpha \times \sum_{v_j \in \text{In}(v_i)} \frac{w_{ji}}{\sum_{v_k \in \text{Out}(v_j)} w_{jk}} \cdot \text{WS}(v_j) \quad \text{where} \quad w_{ij} = w_{ji} = \frac{\mathbf{v}_i \cdot \mathbf{v}_j}{|\mathbf{v}_i| |\mathbf{v}_j|}$$

- **Language Model** (Document Likelihood Measure)

$$P(D|S) = \prod_{w_j \in D} \left[ \lambda \cdot \sum_{w_i \in S} P_{\text{MLE}}(w_i|S) \cdot P_{\text{WE}}(w_j|w_i) + (1 - \lambda) \cdot P_{\text{MLE}}(w_j|C) \right]^{c(w_j, D)}$$

$$\text{where} \quad P_{\text{WE}}(w_j|w_i) = \frac{\exp(\mathbf{v}_i \cdot \mathbf{v}_j)}{\sum_{w_k \in V} \exp(\mathbf{v}_i \cdot \mathbf{v}_k)}$$

# fasttext (1/2) Facebook

- The objective of the skip-gram model is defined as maximizing the following log-likelihood

$$LL = \sum_{t=1}^T \sum_{c \in C_t} \log P(c|w_t) \quad \text{where} \quad P(c|w_t) = \frac{\exp(\text{SIM}(\mathbf{w}_t, \mathbf{c}))}{\sum_{c' \in C_t} \exp(\text{SIM}(\mathbf{w}_t, \mathbf{c}'))}$$

scoring function  
 $\text{SIM}(\mathbf{w}_t, \mathbf{c}) = \mathbf{w}_t^T \mathbf{c}$

- This goal can be alternatively framed as a set of binary classification tasks
  - To independently best predict the presence (or absence) of context words (while minimizing the occurrence of non-context words) with a binary logistic loss

$$\log(1 + \exp(-\text{SIM}(\mathbf{w}_t, \mathbf{c}))) + \sum_{w \in NC_t} \log(1 + \exp(\text{SIM}(\mathbf{w}_t, \mathbf{w})))$$

$l(\mathbf{w}_t, \mathbf{c})$                        $l(\mathbf{w}_t, \mathbf{w})$   
Logistic loss function      negatives randomly sampled from the vocabulary

logistic regression  
 $\frac{1}{1 + \exp(-x)}$

$$NLL = \sum_{t=1}^T \left[ \sum_{c \in C_t} l(\mathbf{w}_t, \mathbf{c}) + \sum_{w \in NC_t} l(\mathbf{w}_t, \mathbf{w}) \right]$$



## fasttext (2/2)

- Subword modeling

- Each word  $w$  is represented as a bag of character  $n$ -gram. We add special boundary symbols  $<$  and  $>$  at the beginning and end of words, allowing to distinguish prefixes and suffixes from other character sequences

*where*  $\Rightarrow$   $< where > < wh, ehe, her, ere, re >$  ( $n = 3$  here)

- Suppose that you are given a dictionary of  $n$ -grams of size  $V_G$ . Given a word  $w$ , let us denote by  $g_w$  ( $g_w \subset G = \{g_1, g_2, \dots, g_{V_G}\}$ ) the set of  $n$ -grams appearing in  $w$ , the scoring function can be thus defined by

$$\text{SIM}(\mathbf{w}_t, \mathbf{c}) = \sum_{g \in g_w} \mathbf{g}^T \mathbf{c}$$

# GloVe: Global Vectors for Word Representation Stanford

- A global log bilinear regression model that combines the advantages of global matrix factorization and local context window methods
  - GloVe assumes there exists a **log bilinear function**  $F$  that can model the triplet relations among words  $(w_i, w_j, w_k)$  with their associated embeddings  $(\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k)$

$$F \left( (\mathbf{v}_i - \mathbf{v}_j)^T \tilde{\mathbf{v}}_k \right) = \frac{F(\mathbf{v}_i^T \tilde{\mathbf{v}}_k)}{F(\mathbf{v}_j^T \tilde{\mathbf{v}}_k)} = \frac{P(w_k|w_i)}{P(w_k|w_j)} = \frac{(N_{i,k}/N_i)}{(N_{j,k}/N_j)}$$

To be estimated

GloVe builds on top the ratios of conditional probabilities from the word-word co-occurrence matrix

- The function can be instantiated by  $F = \exp$ , namely

$$\mathbf{v}_i^T \tilde{\mathbf{v}}_k \approx \log P(w_k|w_i) = \log N_{i,k} - \log N_i$$

co-occurrence count of  $w_i$  and  $w_k$ 
occurrence count of  $w_i$

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

- To enforce exchange symmetry to the above equation

$$\mathbf{v}_i^T \tilde{\mathbf{v}}_k + \mathbf{b}_i + \tilde{\mathbf{b}}_k \approx \log N_{i,k}$$


- The training objective for GloVe

$$J = \sum_{i,k} f(N_{i,k}) (\mathbf{v}_i^T \tilde{\mathbf{v}}_k + \mathbf{b}_i + \tilde{\mathbf{b}}_k - \log N_{i,k})$$

## $n$ -gram Language Models

- The  $n$ -gram language model that determines the probability of an upcoming word given the previous  $n-1$  word history is the most prominently used

$$\begin{aligned} P(\mathbf{W} = w_1, w_2, \dots, w_m) \\ &= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\dots P(w_m|w_1, w_2, \dots, w_{m-1}) \\ &= P(w_1) \prod_{i=2}^m P(w_i|w_1, w_2, \dots, w_{i-1}) \end{aligned}$$

  
Chain Rule

Multiplication of Conditional Probabilities

- $n$ -gram assumption

$$P(w_i|w_1, w_2, \dots, w_{i-1}) \approx P(w_i|\underbrace{w_{i-n+1}, w_{i-n+2}, \dots, w_{i-1}}_{\text{History of length } n-1})$$

$$P(w_i|w_1, w_2, \dots, w_{i-1}) \approx P(w_i|w_{i-2}, w_{i-1}) \quad \text{Trigram}$$

$$P(w_i|w_1, w_2, \dots, w_{i-1}) \approx P(w_i|w_{i-1}) \quad \text{Bigram}$$

$$P(w_i|w_1, w_2, \dots, w_{i-1}) \approx P(w_i) \quad \text{Unigram}$$

## Known Weakness of $n$ -gram Language Models

- $n$ -gram language models are Sensitive to changes in the style or topic of the text on which they are trained
  - Assume the probability of next word in a sentence depends only on the identity of last  $n-1$  words
    - Capture only **local contextual information** or **lexical regularity (word ordering relationships)** of a language
- Ironically,  $n$ -gram language models take no advantage of the fact that what is being modeled is language
  - Frederick Jelinek said “***put language back into language modeling***” (1995)

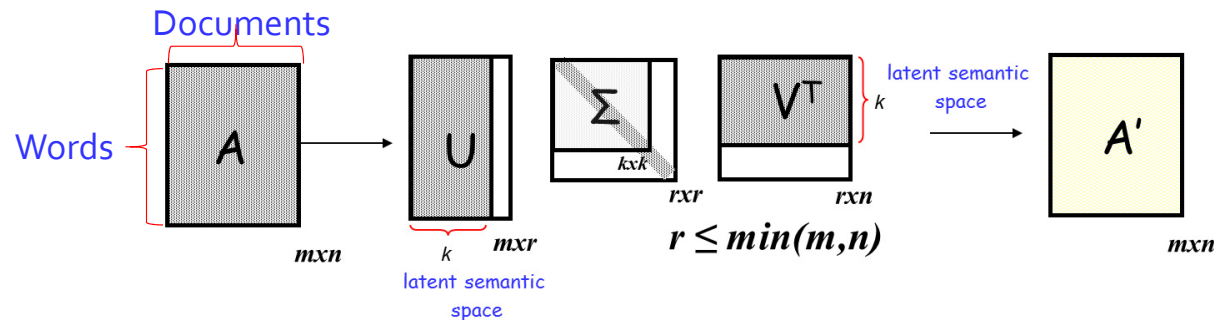
$$P(w_i | w_1, w_2, \dots, w_{i-1}) \approx P_{\text{Tirgram}}(w_i | w_{i-2}, w_{i-1})$$

# Topic Modeling

- Topic language models have been introduced and investigated to complement the  $n$ -gram language models
  - A commonality among them is that a set of latent topic variables is introduced to describe the “**word-document**” co-occurrence characteristics
- Models developed generally follow two lines of thought
  - Algebraic
    - Latent Semantic Analysis (LSA) (Deerwester et al., 1990), nonnegative matrix factorization (NMF) (Lee and Seung, 1999), etc.
  - Probabilistic
    - Probabilistic latent semantic analysis (PLSA) (Hofmann, 2001), latent Dirichlet allocation (LDA) (Blei et al., 2003), Word Topic Model (Chen, 2009) etc.

# Latent Semantic Analysis (LSA) (1/3)

- Start with a matrix describing the intra- and Inter-document statistics between all terms and all documents
- Singular value decomposition (SVD) is then performed on the matrix to project all term and document vectors onto a reduced latent topical space



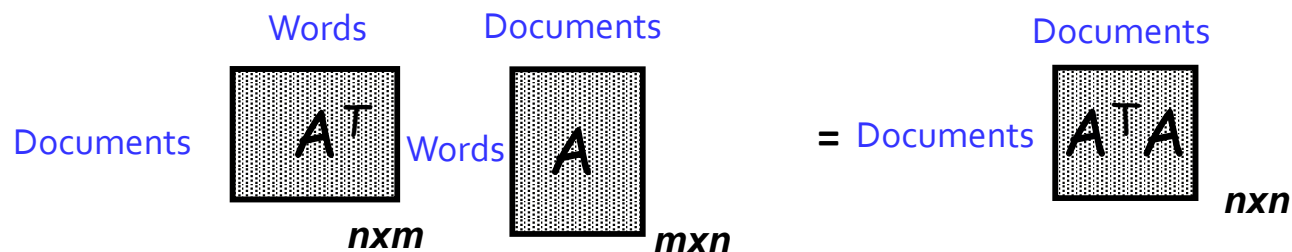
$$\|A\|_F^2 = \sum_{i=1}^m \sum_{j=1}^n a_{ij}^2 \Rightarrow \|A\|_F^2 = \sigma_1^2 + \sigma_2^2 + \dots + \sigma_r^2 \quad ?$$

- In the context of IR, matching between queries and documents can be carried out in this topical space

1. G. W. Furnaset et al., "Information Retrieval using a Singular Value Decomposition Model of Latent Semantic Structure," *SIGIR1988*
2. T. K. Landauer et al. (eds.), *Handbook of Latent Semantic Analysis*, Lawrence Erlbaum, 2007

## Latent Semantic Analysis (2/3)

- The latent space of LSA is derived on top of eigen-decomposition of the matrix  $A^T A$ 
  - Each entry of  $A^T A$  represents the correlation (inner product; closeness relationship) between any document (vector) pairs
- The column vectors  $v_j$  in  $V$  actually are eigenvectors of  $A^T A$ 
  - $A^T A$  is symmetric and all its diagonal entities are positive  $(A^T A)v_i = \lambda_i v_i$ 
    - All eigenvalues  $\lambda_j$  are nonnegative real numbers
    - All eigenvectors  $v_j$  are orthonormal
    - Singular values  $\sigma_j$  in  $\Sigma$  are the square roots of  $\lambda_j$   $\left(\sigma_j = \sqrt{\lambda_j}\right)$



LSA bears similarity to PCA (Principal Component Analysis), and has the aim of finding a subspace determined by the eigenvectors of  $A^T A$  that preserves most of the relationships (a kind of simple structure information) between documents (compositions).

## Latent Semantic Analysis (3/3)

- Pro

- A clean formal framework and a clearly defined optimization criterion (least-squares)
  - Conceptual simplicity and clarity
- Handle synonymy problems (“heterogeneous vocabulary”)
  - Replace individual terms as the descriptors of documents by independent “**artificial concepts**” that can be specified by any one of several terms (or documents) or combinations

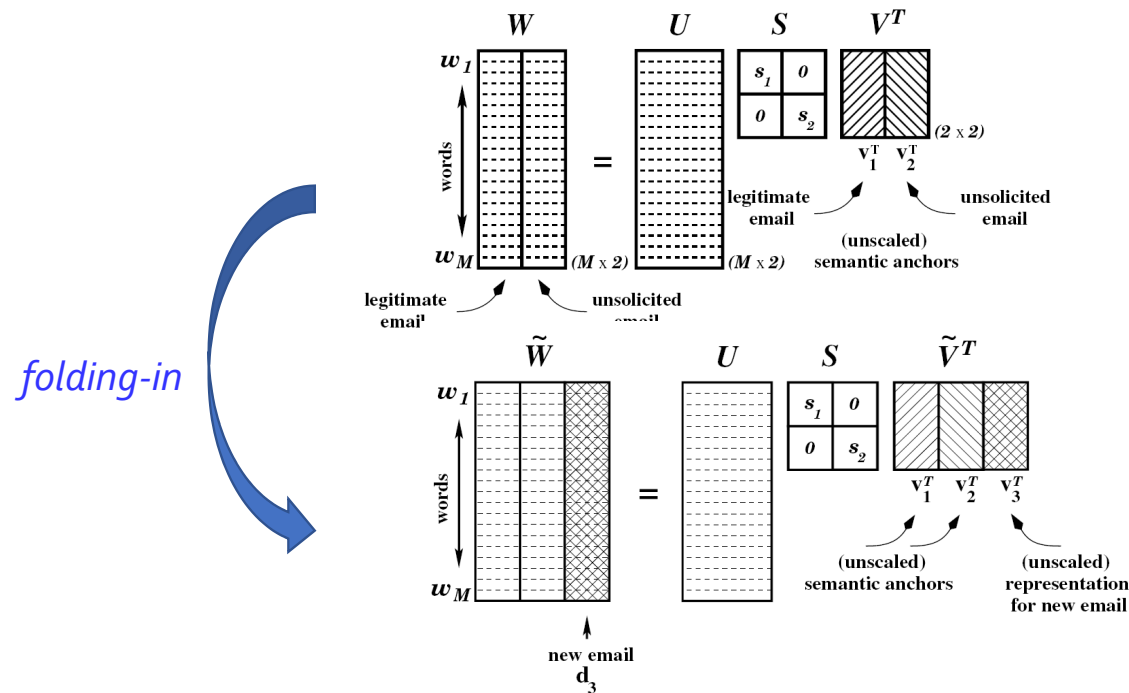
- Con

- Contextual or positional information for words in documents is discarded (the so-called “**bag-of-words**” assumption)
- High computational complexity (e.g., SVD decomposition)
- Word and document representations have negative values
- Exhaustive search is needed when comparing among documents or between a query (word) and a document (cannot make use of inverted files ?)



# LSA: Application to Junk E-mail Filtering

- One vector represents the centroid of all e-mails that are of interest to the user, while the other the centroid of all e-mails that are not of interest



# LSA: Application to Cross-lingual Language Modeling

- Assume that a document-aligned (instead of sentence-aligned) Chinese-English bilingual corpus is provided

$$\begin{array}{c}
 \begin{array}{|c|c|c|c|}
 \hline
 & \mathbf{w} & & \\
 \hline
 d_1^E & d_2^E & \dots & d_N^E \\
 \hline
 d_1^C & d_2^C & \dots & d_N^C \\
 \hline
 \end{array} \\
 M \times N
 \end{array}
 =
 \begin{array}{c}
 \begin{array}{|c|}
 \hline
 \mathbf{U} \\
 \hline
 \end{array}
 \times
 \begin{array}{|c|}
 \hline
 \mathbf{S} \\
 \hline
 \end{array}
 \times
 \begin{array}{|c|}
 \hline
 \mathbf{V}^T \\
 \hline
 \end{array} \\
 M \times R \quad R \times R \quad R \times N
 \end{array}$$

SVD of a word-document matrix for CL-LSA.

$$\begin{array}{c}
 \begin{array}{|c|c|c|c|}
 \hline
 & \bar{\mathbf{w}} & & \\
 \hline
 \bar{d}_1^E & \bar{d}_2^E & \dots & \bar{d}_P^E \\
 \hline
 0 & 0 & \dots & 0 \\
 \hline
 \end{array} \\
 M \times P
 \end{array}
 =
 \begin{array}{c}
 \begin{array}{|c|}
 \hline
 \mathbf{U} \\
 \hline
 \end{array}
 \times
 \begin{array}{|c|}
 \hline
 \mathbf{S} \\
 \hline
 \end{array}
 \times
 \begin{array}{|c|}
 \hline
 \bar{\mathbf{V}}^T \\
 \hline
 \end{array} \\
 M \times R \quad R \times R \quad R \times P
 \end{array}$$

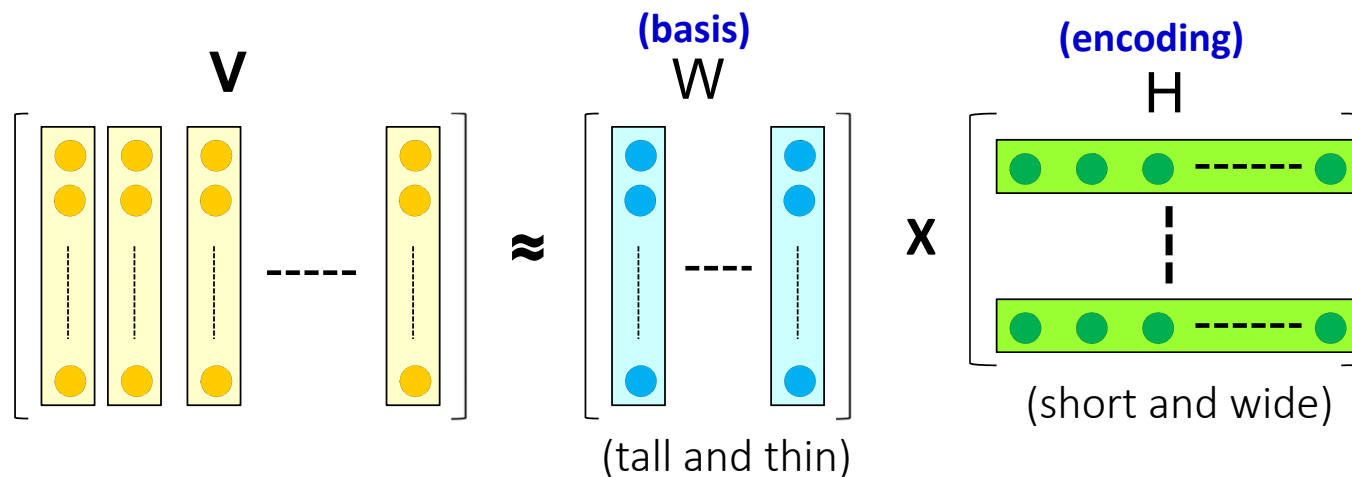
Folding-in a monolingual corpus into LSA.

$$P_{\text{CL-LSA-Unigram}}(c|d_i^E) = \sum_e P_T(c|e)P(e|d_i^E)$$

$$P_T(c|e) \approx \frac{\text{sim}(\bar{c}, \bar{e})^\gamma}{\sum_{c'} \text{sim}(\bar{c}', \bar{e})^\gamma} \quad (\gamma \gg 1)$$

# Nonnegative Matrix Factorization (NMF)

- NMF approximates data with an **additive and linear combination** of nonnegative components (or basis vectors)
  - Given a **nonnegative data matrix**  $V \in \mathbb{R}^{L \times M}$ , NMF computes another two **nonnegative matrices**  $W \in \mathbb{R}^{L \times R}$  and  $H \in \mathbb{R}^{R \times M}$  such that  $V \approx WH$ 
    - $R \ll L$  and  $R \ll M$  to ensure *efficient encoding*

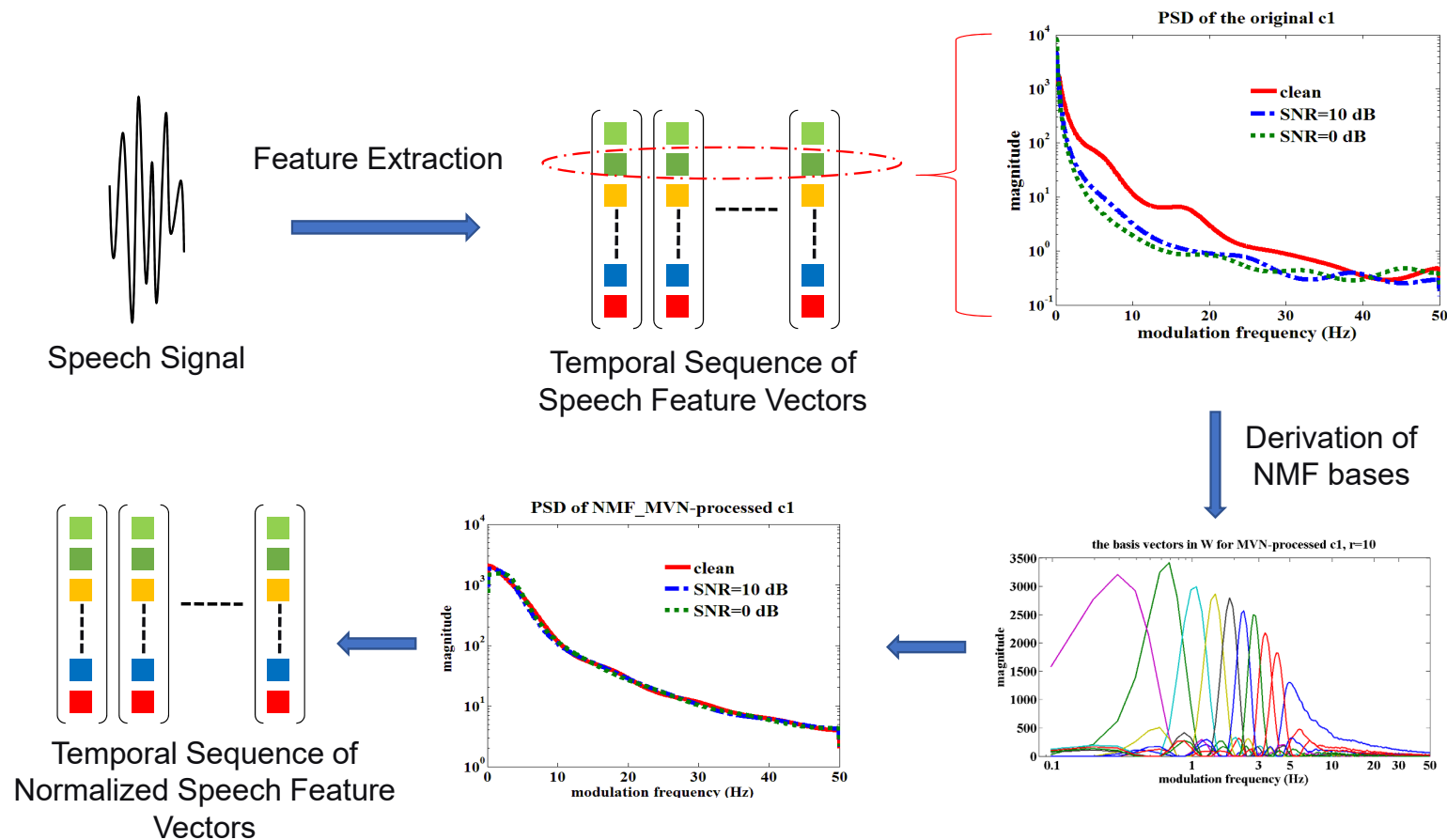


$$\mathbf{v} \approx \mathbf{W}\mathbf{h} = \sum_{r=1}^R h_r \mathbf{w}_r = h_1 \mathbf{w}_1 + \dots + h_R \mathbf{w}_R$$

D. D. Lee and H. S. Seung,  
 "Learning the parts of  
 objects by non-negative  
 matrix factorization,"  
*Nature*, 1999.

# NMF: Application

- Modulation Spectrum Factorization for Automatic Speech Recognition (ASR)



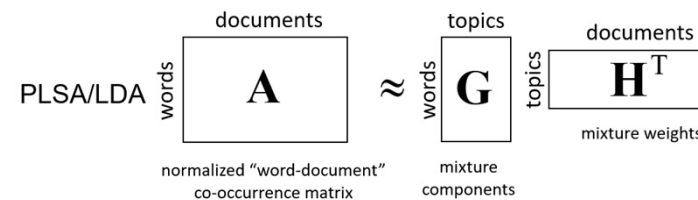
W.-Y. Chu, et al., "Modulation spectrum factorization for robust speech recognition," *APSIPA ASC*, 2011.

# Probabilistic Latent Semantic Analysis (PLSA)

- Each document as a whole consists of a set of shared latent topics with different weights -- a **document topic modeling (DTM)** approach
  - Each topic in turn offers a unigram (multinomial) distribution for observing a given word

$$P_{\text{PLSA}}(w|D) = \sum_{k=1}^K P(w_i|T_k)P(T_k|D)$$

- LDA (latent Dirichlet allocation) differs from PLSA mainly in the inference of model parameters:
  - PLSA assumes the model parameters are fixed and unknown
  - LDA places additional a priori constraints on the model parameters, i.e., thinking of them as random variables that follow some Dirichlet distributions



1. T. Hoffmann, "Unsupervised learning by probabilistic latent semantic analysis," *Machine Learning*, 2001.
2. D. M. Blei et al., "Latent Dirichlet allocation," *Journal of Machine Learning Research*, 2003.

# PLAS: Empirical Evaluation

Semantic Fields (see. p.p. 8 in this handout)

aviation	space missions	family love	Hollywood love
Aspect 1	Aspect 2	Aspect 3	Aspect 4
plane	space	home	film
airport	shuttle	family	movie
crash	mission	like	music
flight	astronauts	love	new
safety	launch	kids	best
aircraft	station	mother	hollywood
air	crew	life	love
passenger	nasa	happy	actor
board	satellite	friends	entertainment
airline	earth	cnn	star

The 2 aspects to most likely generate the word 'flight' (left) and 'love' (right), derived from a  $K = 128$  aspect model of the TDT1 document collection. The displayed terms are the most probable words in the class-conditional distribution  $P(w_j | z_k)$ , from top to bottom in descending order.

# PLSA vs. LDA

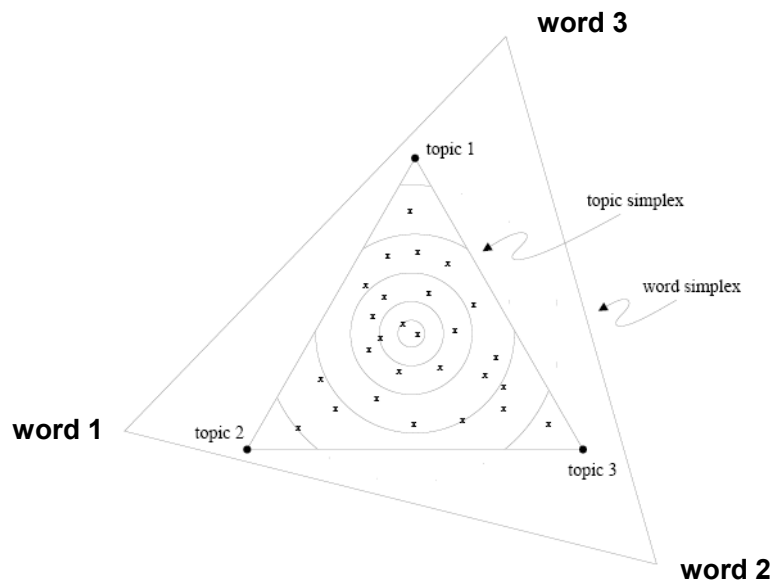
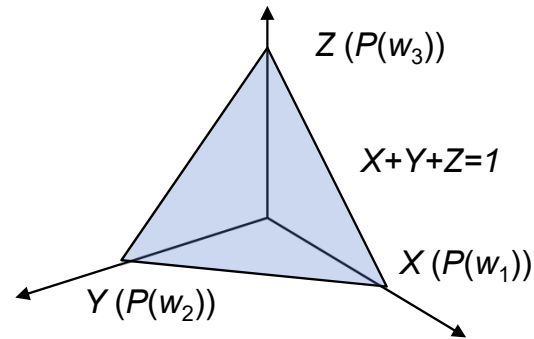


Figure 4: The topic simplex for three topics embedded in the word simplex for three words. The corners of the word simplex correspond to the three distributions where each word (respectively) has probability one. The three points of the topic simplex correspond to three different distributions over words. The mixture of unigrams places each document at one of the corners of the topic simplex. The pLSI model induces an empirical distribution on the topic simplex denoted by x. LDA places a smooth distribution on the topic simplex denoted by the contour lines.



## Formulation of LDA

$$P(d|\alpha, \beta) = \int f(\theta_d|\alpha) \left( \prod_{n=1}^{N_d} \sum_{z_{dn}} p(z_{dn}|\theta_d) p(w_n|z_{dn}, \beta) \right) d\theta_d$$

Document Likelihood

$$P(D|\alpha, \beta) = \prod_{d=1}^M \int f(\theta_d|\alpha) \left( \prod_{n=1}^{N_d} \sum_{z_{dn}} p(z_{dn}|\theta_d) p(w_n|z_{dn}, \beta) \right) d\theta_d$$

Collection Likelihood