WORD EMBEDDING

Vector semantics

Prof. Marco Viviani

marco.viviani@unimib.it

Word embedding – Definition

- The term **word embedding** indicates a set of techniques in *Natural Language Processing* (NLP) where **words or phrases** from the vocabulary are **mapped** to dense vectors of real numbers.

- Conceptually, it involves a **mathematical embedding** from a vector space with many dimensions per word to a vector space with a much lower dimension.

- **Models** to generate this mapping include:
  - Count-based models (Distributed semantic models)
  - Predictive models (Neural network models)

BACKGROUND

Text representation

Representing DOCUMENTS as vectors

- Each **document** is represented by a vector of words.
  - **Option 1**: Binary representation.

$|D| = N$

V

	d_1	d_2	d_3
bear	1	0	0
cat	0	1	0
frog	0	0	1

$d_1 = [1, \quad 0, \quad 0]$

$d_2 = [0, \quad 1, \quad 0]$

$d_3 = [0, \quad 0, \quad 1]$

Representing DOCUMENTS as vectors

- Each **document** is represented by a vector of words.
  - **Option 2**: Raw frequency representation.

	d_1	d_2	d_3
bear	85	0	0
cat	0	10	0
frog	0	0	44

$d_1 = [85, \quad 0, \quad 0]$ $\qquad$ $d_2 = [0, \quad 10, \quad 0]$

$d_3 = [0, \quad 0, \quad 44]$

Representing DOCUMENTS as vectors

- Each **document** is represented by a vector of words.
  - **Option 3**: Weighted representation.
    - Weighted term frequency (different possibilities)
    - **TF-IDF**

	d_1	d_2	d_3
bear	0.48	0	0
cat	0	0.48	0
frog	0	0	0.48

$d_1 = [0.48, \quad 0, \quad 0]$

$d_2 = [0, \quad 0.48, \quad 0]$

$d_3 = [0, \quad 0, \quad 0.48]$

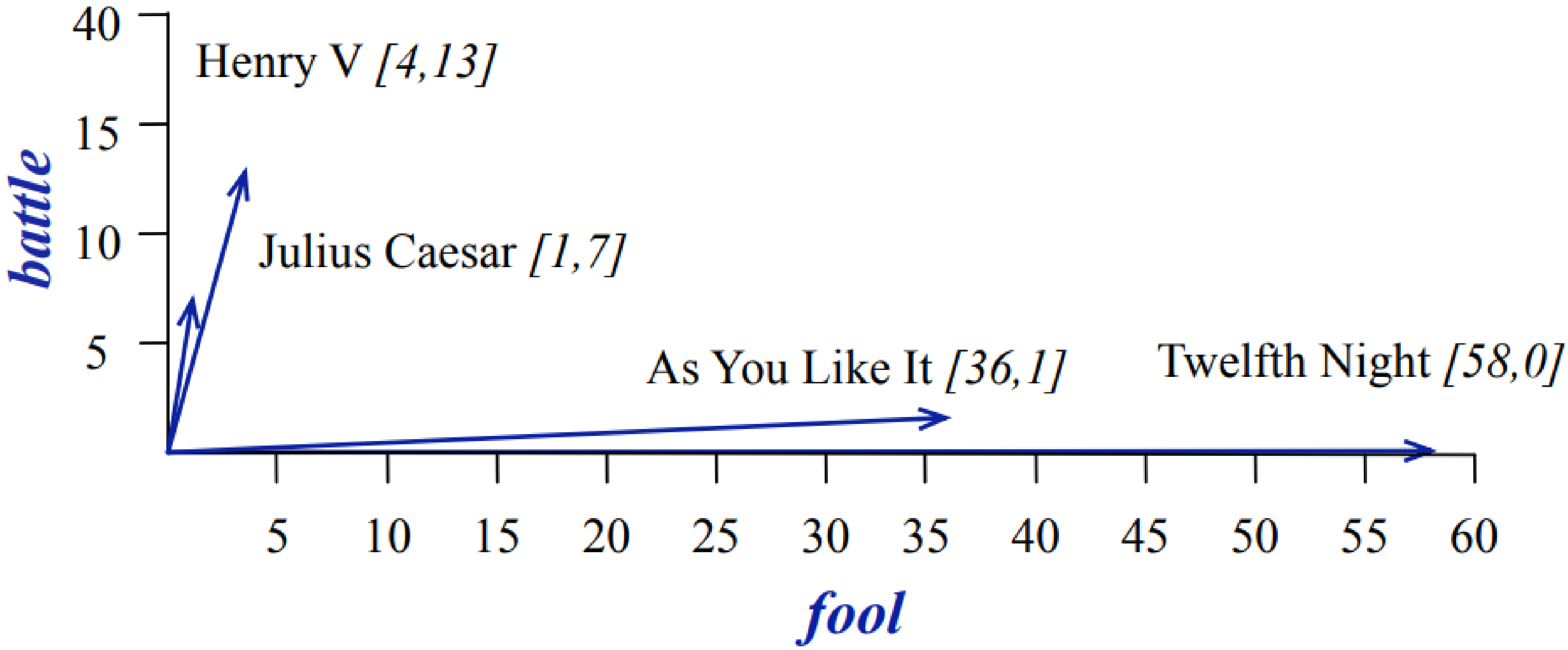
Similarity of DOCUMENTS

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

- Vectors of the two comedies are similar. They are different with respect to the history plays.
  - Comedies have more “fools” and “wits” and fewer “battles”.

- The **vector representation of documents** is at the basis of *Information Retrieval* → **Vector Space Model**

Visualizing similarity of DOCUMENTS

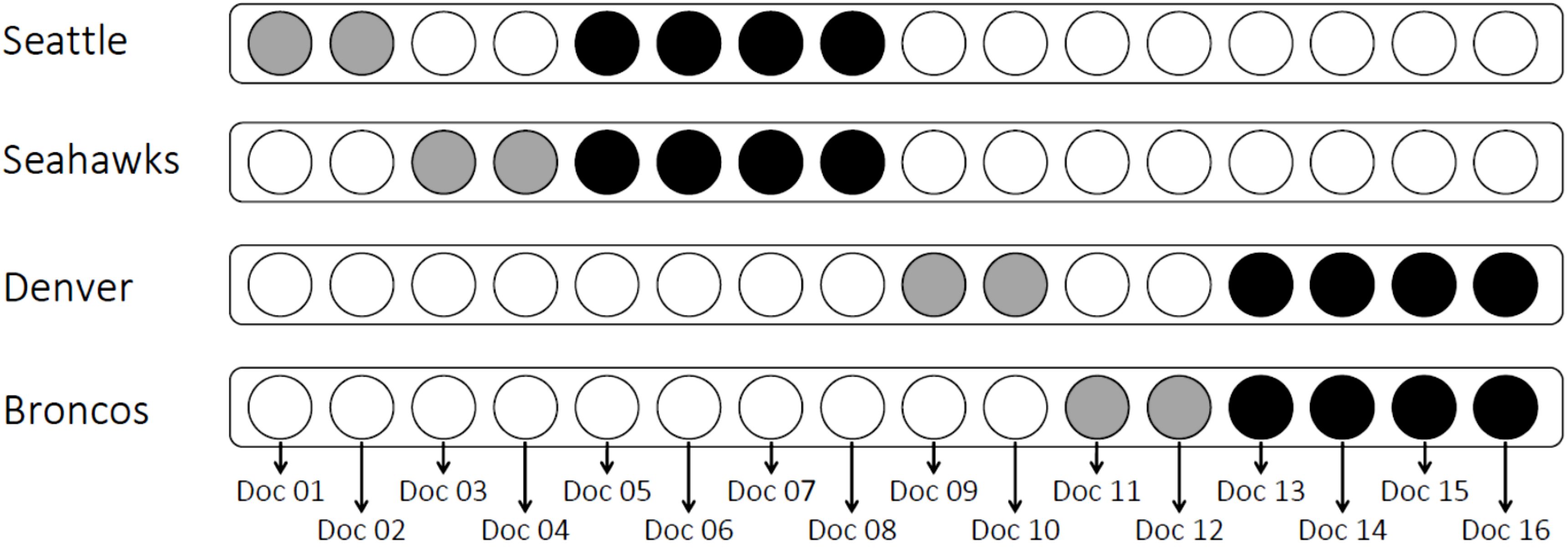


WORDS can be represented as vectors too

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

- In the **term-document matrix representation**, a possible interpretation could be:
  - *battle* is "the kind of word that occurs history plays, in Julius Caesar and Henry V especially".
  - *fool* is "the kind of word that occurs in comedies, especially Twelfth Night".

In-document features



(a) “In-documents” features

Similarity of WORDS

- **Usually**, the similarity of words is **NOT** computed by using the term-document representation.

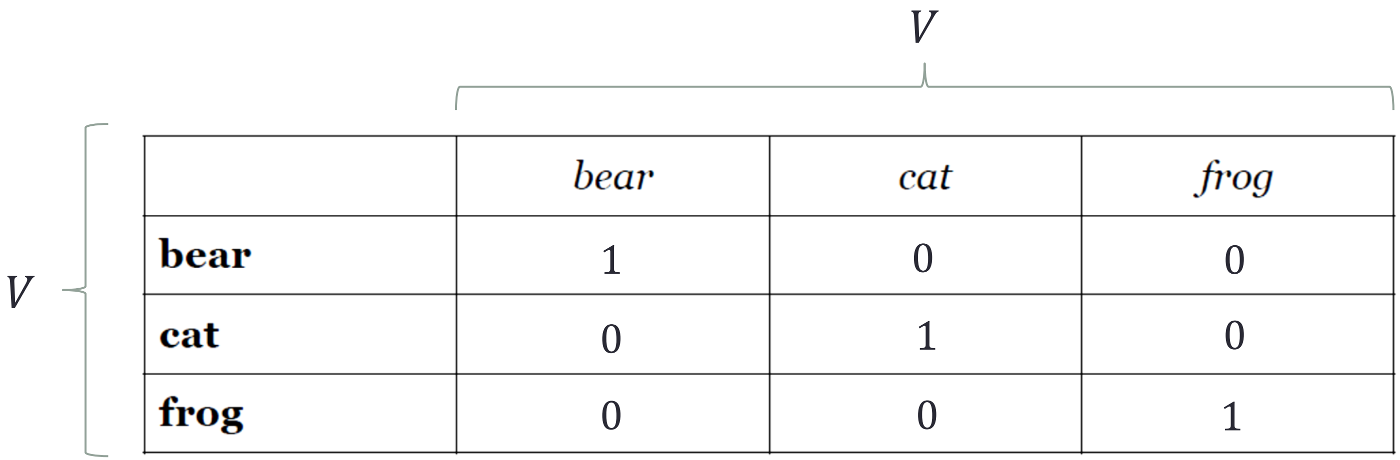
- Two words are similar if their «**context vectors**» are similar.
  - We are going to detail this concept in the next slides.

- The employed matrix representation, in this case, has **words** on both rows and columns.
  - Different representations and meanings.
  - Next slides.

Representing WORDS as vectors

1. Local representation

- Each **word** is represented by a vector of words.
  - **Option 1**: each element represents a different word.
    - Also known as “1-hot” or “1-of-V” or **local representation**.



V

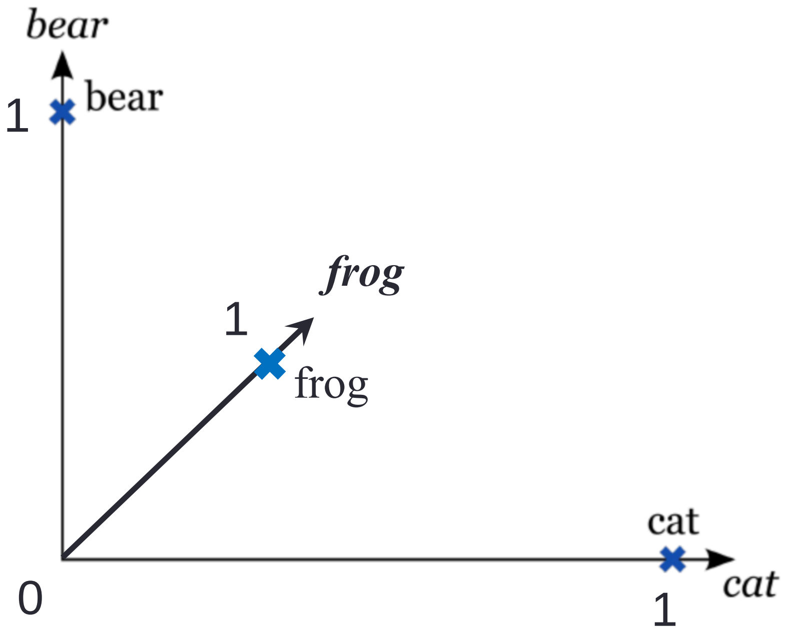
	bear	*cat*	*frog*
bear	1	0	0
cat	0	1	0
frog	0	0	1

V

bear $= [1,\ 0,\ 0]$ cat $= [0,\ 1,\ 0]$ frog $= [0,\ 0,\ 1]$

1-hot vectors

- 1-hot vectors tell us **very little**.
- We need a separate dimension for every word we want to represent (the base vectors in a vector space).



1-hot vectors

Few **problems** with the one-hot approach for encoding:

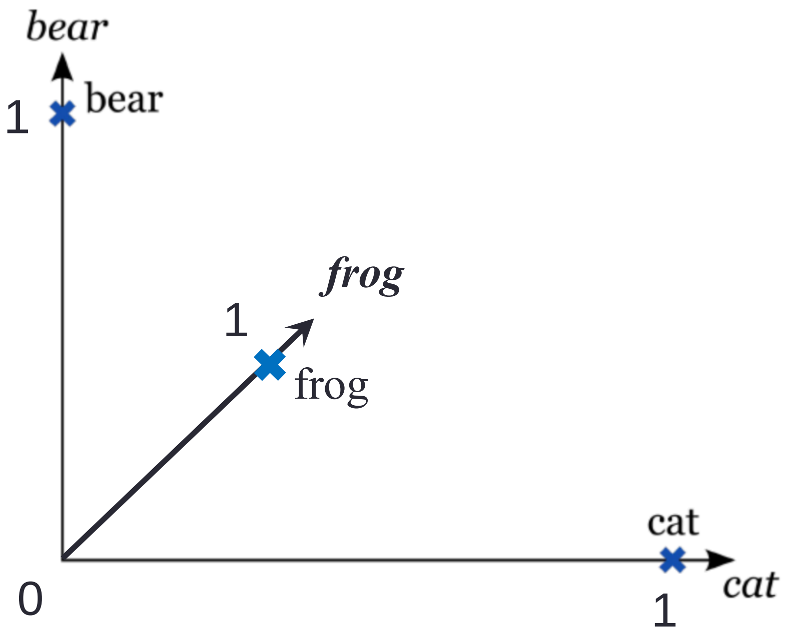
- The **number of dimensions** (the columns) **increases linearly** as we add words to the vocabulary.
  - For a vocabulary of 50,000 words, each word is represented with 49,999 zeros, and a single “one” value in the correct location. As such, memory use is prohibitively large.

- The matrix is **very sparse**, mainly made up of zeros.

- There is **no shared information between words** and **no commonalities between similar words**.

1-hot vectors

- There is **no shared information between words** and **no commonalities between similar words**.



$$bear = [1, \quad 0, \quad 0]$$

$$frog = [0, \quad 1, \quad 0]$$

$$cat = [0, \quad 0, \quad 1]$$

Representing WORDS as vectors

2. Distributed representation

- Each **word** is represented by a vector of words.
  - **Option 2**: **IDEA**: to each word of the vocabulary are associated k “context dimensions” that represent “properties” associated with the words of the vocabulary.
    - Also known as **distributed representation**.

AIM

C $|C| = k \ll |V|$

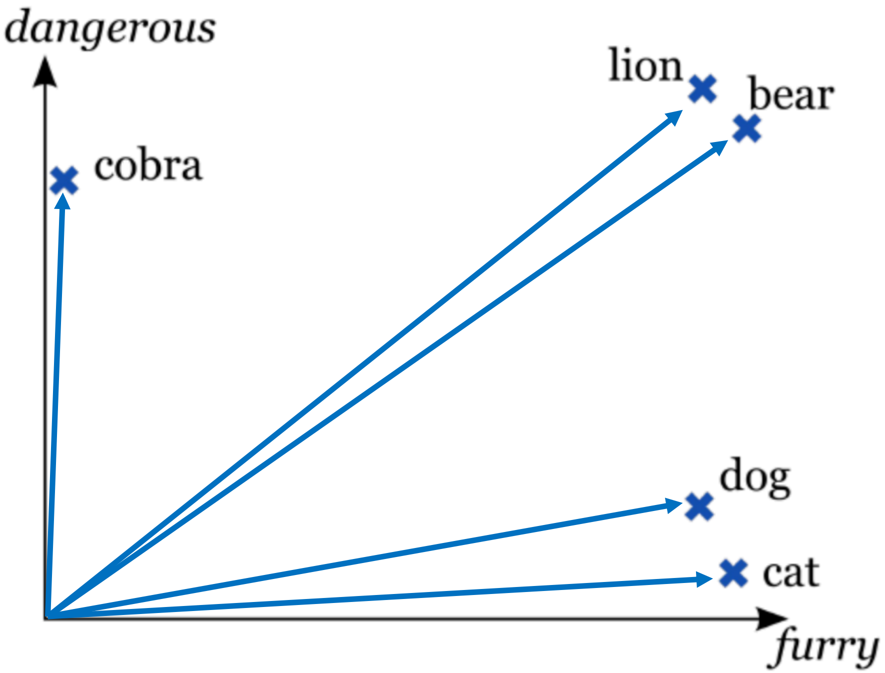
V

	furry	*dangerous*	*mammal*
bear	0.9	0.85	1
cat	0.85	0.15	1
frog	0	0.05	0

`bear = [0.9, 0.85, 1.0]` `cat = [0.85, 0.15, 1.0]`

Distributed representation

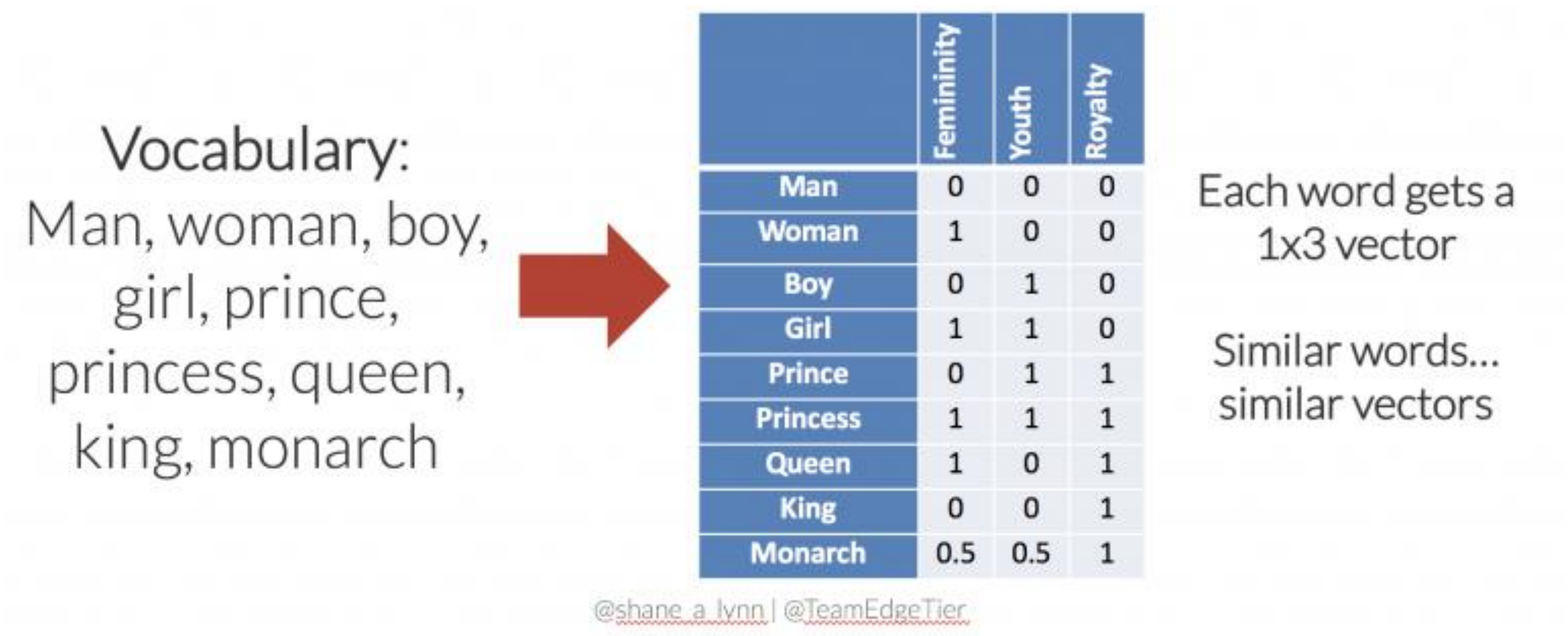
- “Distributed vectors” allow to group similar words/objects together, depending on the considered context.



	furry	*dangerous*
bear	0.9	0.85
cat	0.85	0.15
cobra	0.0	0.8
lion	0.85	0.9
dog	0.8	0.15

Distributed representation

- For simple scenarios, we can create a ***k*-dimensional mapping** for a simple example vocabulary by **manually choosing** contextual dimensions that make sense.



Vocabulary:
Man, woman, boy, girl, prince, princess, queen, king, monarch

	Femininity	Youth	Royalty
Man	0	0	0
Woman	1	0	0
Boy	0	1	0
Girl	1	1	0
Prince	0	1	1
Princess	1	1	1
Queen	1	0	1
King	0	0	1
Monarch	0.5	0.5	1

Each word gets a 1x3 vector

Similar words... similar vectors

@shane_a_lynn | @TeamEdgeTier

Relationships between words

- In a **well-defined distributed representation model**, calculations such as:

$$[king] - [man] + [woman] = [queen]$$

$$[Paris] - [France] + [Germany] = [Berlin]$$

(where $[x]$ denotes the vector for the word x) will actually work out!

$$[king] - [man] + [woman] = [queen]$$
$$[0, 0, 1] - [0,0,0] + [1,0,0] = [1,0,1]$$

Distributed representation: Advantages

Some well-known **advantages**:

- Each word is represented with a **$\boldsymbol{k}$-dimensional vector**
  - Optimal representations are those with $\boldsymbol{k} \ll |\boldsymbol{V}|$.
- **Similar words have similar vectors**
  - There’s a smaller distance between vector representation for “girl” and “princess”, than from “girl” to “prince”.

To be continued…

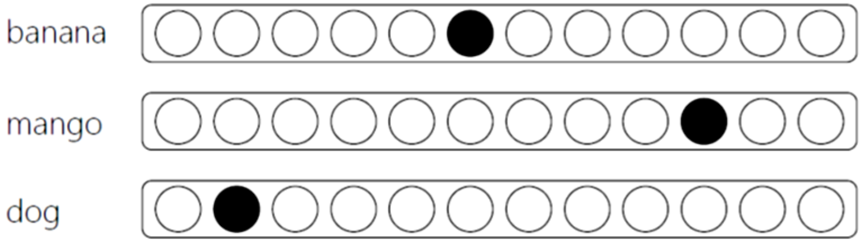
Distributed representation: Advantages

… cont’d

- **The resulting matrix is much less sparse** (less empty space), and we could potentially add further words to the vocabulary without increasing the dimensionality.
  - For instance, the word “child” might be represented with [0.5, 1, 0].

- **Relationships between words are captured and maintained**, e.g., the movement from king to queen, is the same as the movement from boy to girl, and could be represented by [+1, 0, 0].

Local VS Distributed representation



(a) Local representation

- **Local (or one-hot) representation**
  - Every term in vocabulary V is represented by a binary vector of length $|V|$, where one position in the vector is set to one and the rest to zero.
- **Distributed representation**
  - Every term in vocabulary V is represented by a real-valued vector of length k. The vector can be *sparse* or *dense*. The vector dimensions may be *observed* (e.g., hand-crafted features) or *latent* (e.g., embedding dimensions).

Extending to larger vocabularies

- **Forming k-dimensional vectors** that capture meaning in the same way that our simple example does, where similar words have similar vectors and relationships between words are maintained, **is not a simple task**.

- Manual assignment of vectors would be **impossibly complex**: individual dimensions cannot be directly interpretable.

- As such, **various algorithms** have been developed, some recently, that can take **large corpora of text** and create meaningful models.

Distributional hypothesis

- "Words which are similar in meaning occur in **similar contexts**".

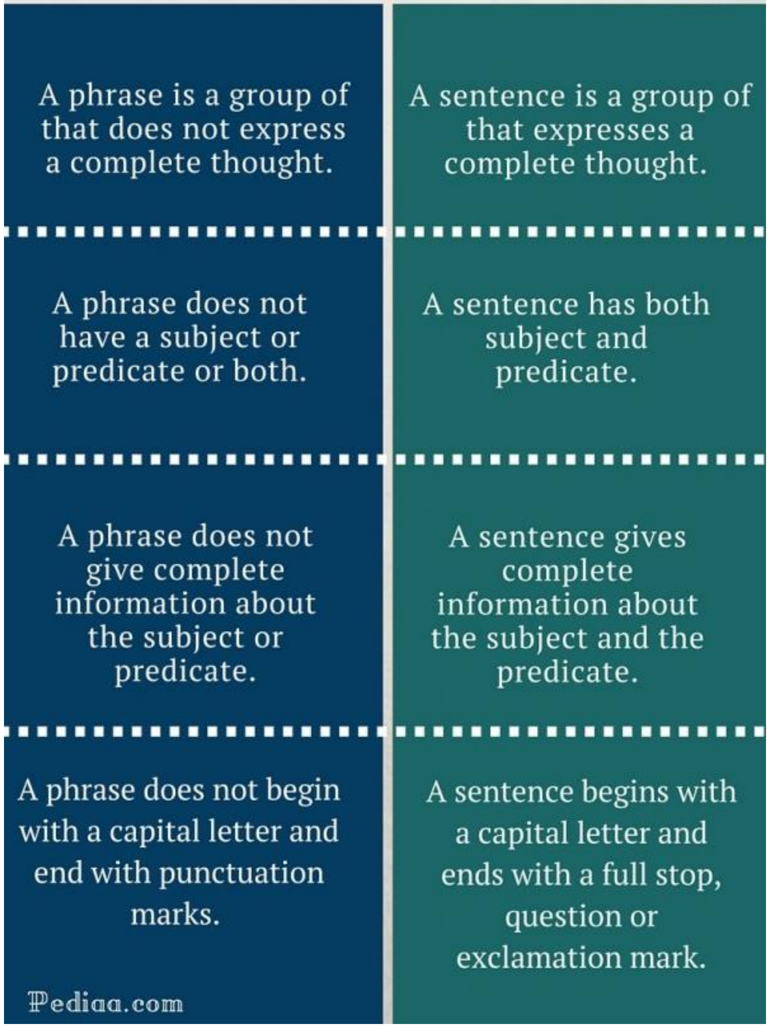
  (Harris, 1954)

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

  (Firth, 1957)

- **Central idea**: represent each word by some **context**:
  - E.g., words co-occurring with the considered word.
  - We can use different granularities of contexts: documents, sentences, phrases, *n*-grams.

Phrase VS sentence



A phrase is a group of that does not express a complete thought.	A sentence is a group of that expresses a complete thought.
A phrase does not have a subject or predicate or both.	A sentence has both subject and predicate.
A phrase does not give complete information about the subject or predicate.	A sentence gives complete information about the subject and the predicate.
A phrase does not begin with a capital letter and end with punctuation marks.	A sentence begins with a capital letter and ends with a full stop, question or exclamation mark.

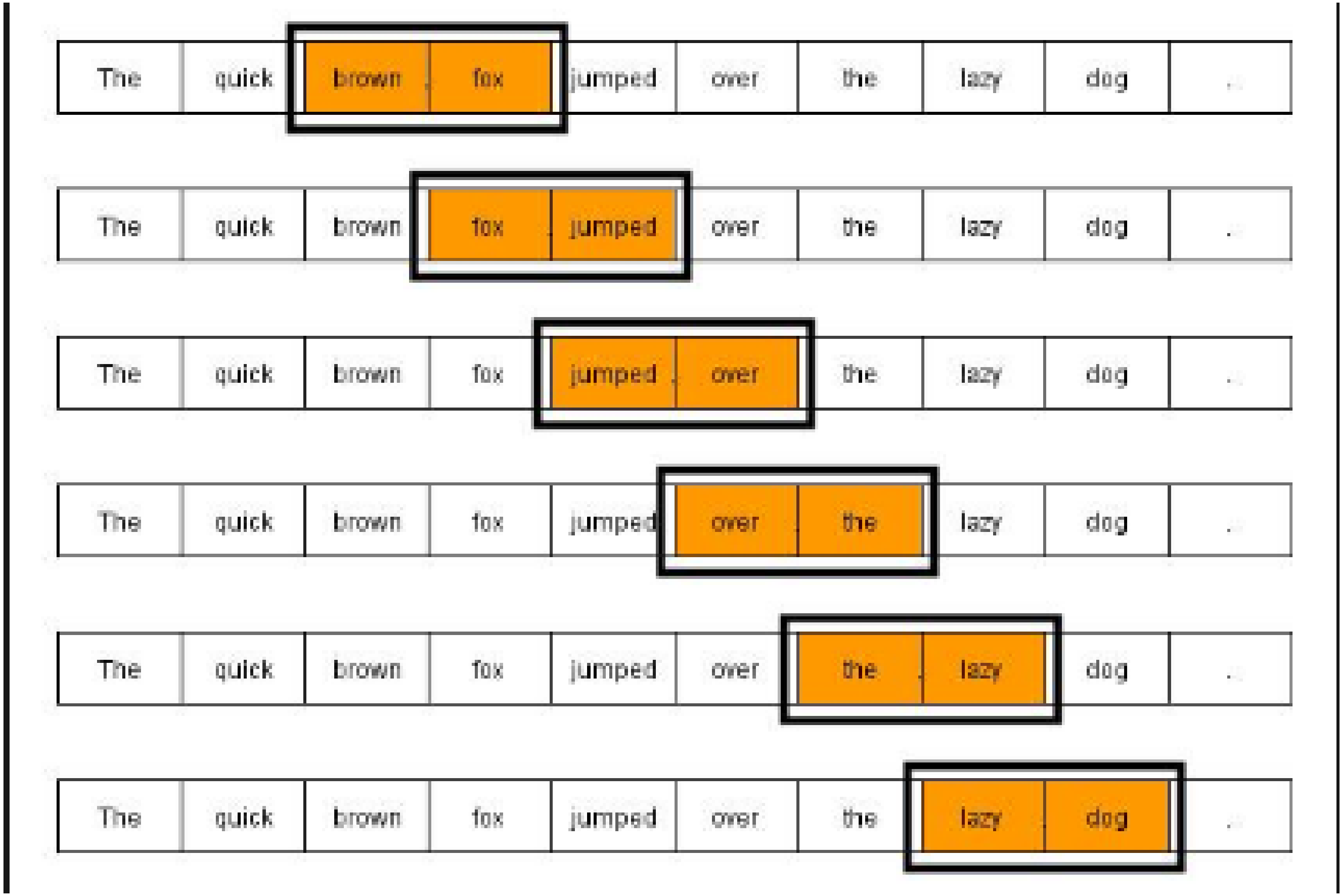
Pediaa.com

Phrase VS sentence: Example

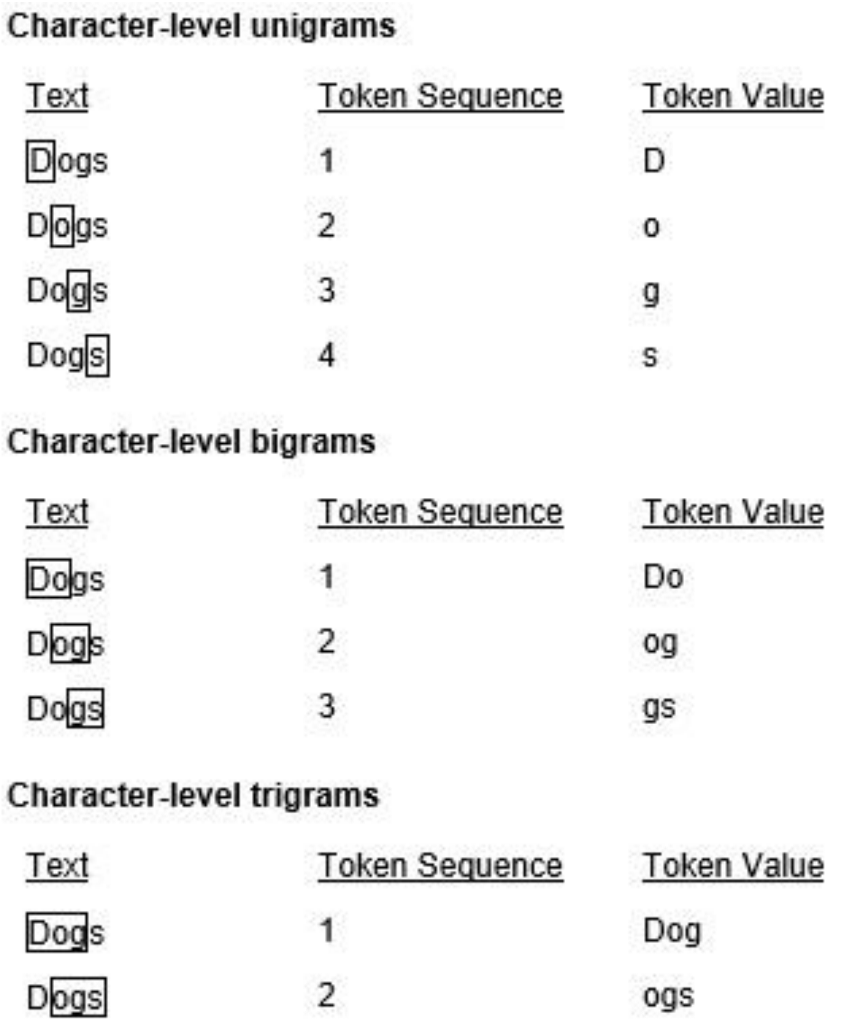
- **Phrase**: “Red apple”.
  - This is a phrase consisting of two words, “red” and “apple”;
  - It is not a complete thought on its own but conveys a simple description of an apple's color.

- **Sentence**: “The quick brown fox jumps over the lazy dog”.
  - This is a complete sentence;
  - It consists of multiple words and forms a grammatically correct and meaningful expression;
  - In this sentence, the subject is “the quick brown fox”, the verb is “jumps”, and the object is “over the lazy dog”;
  - The sentence conveys a clear action, where the fox is jumping over the dog.

Word-level n-grams



Character-level n-grams



Character-level unigrams

Text	Token Sequence	Token Value
Dogs	1	D
Dogs	2	o
Dogs	3	g
Dogs	4	s

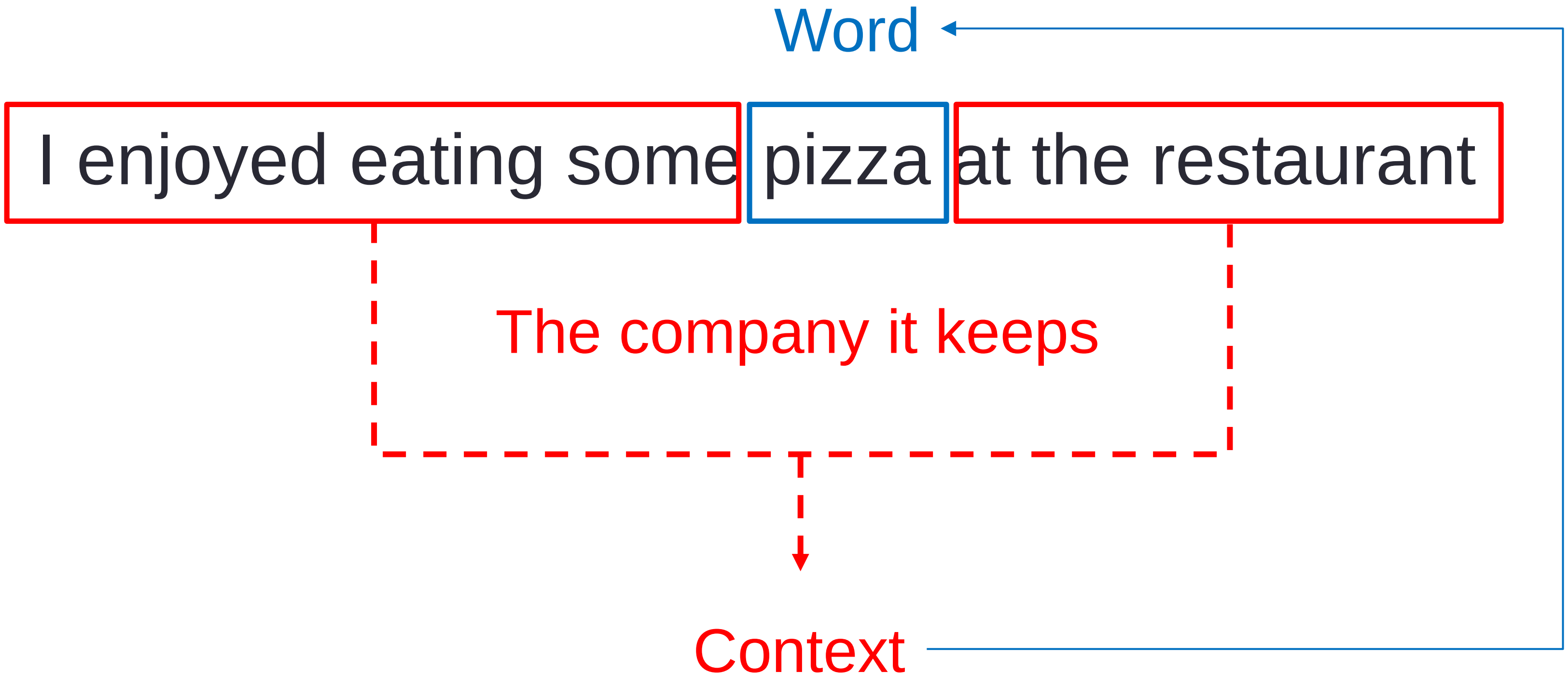
Character-level bigrams

Text	Token Sequence	Token Value
Dogs	1	Do
Dogs	2	og
Dogs	3	gs

Character-level trigrams

Text	Token Sequence	Token Value
Dogs	1	Dog
Dogs	2	ogs

A simple example (Neighbouring terms)



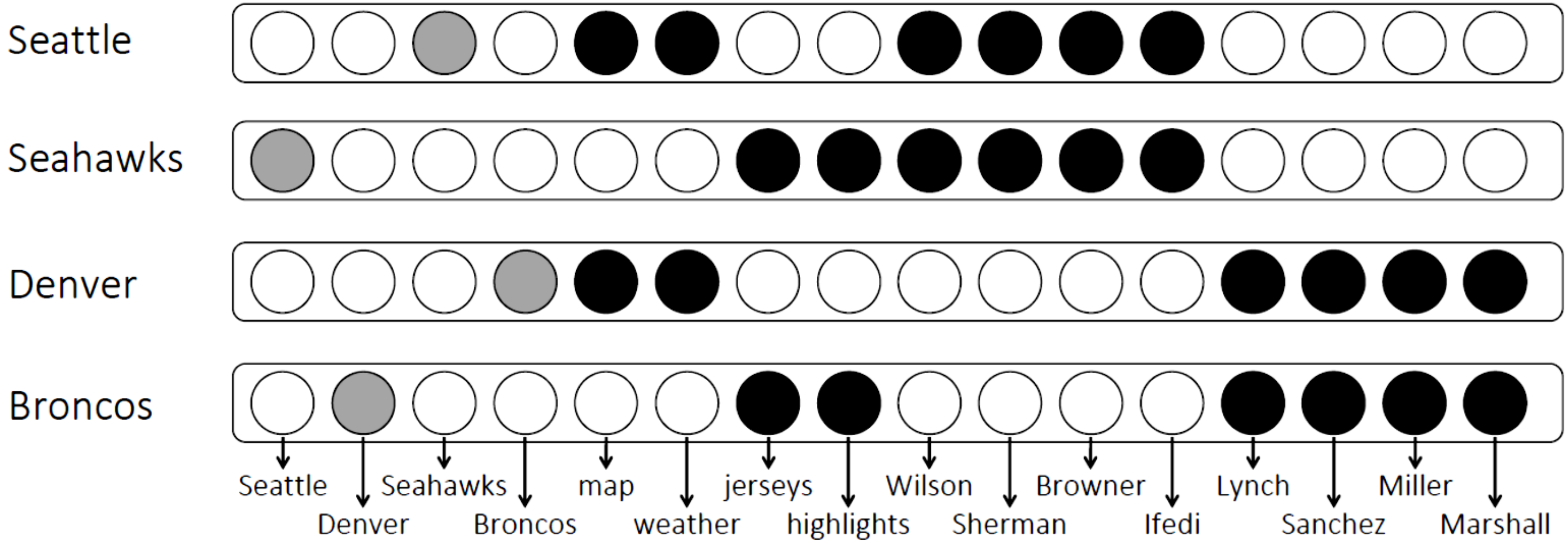
Word

I enjoyed eating some pizza at the restaurant

The company it keeps

Context

Neighbouring terms features



(b) "Neighbouring terms" features

COUNTING

CO-OCCURRING WORDS

Window-based Co-occurrence Matrix

- In this method, given a text corpus, we **count** the **number of times** each (context) word co-occurs:
  - inside a **window** of a particular size,
  - with the **word of interest** (i.e., **target word**).

- The resulting matrix is also known as (**window-based**)
  - **Word-word co-occurrence Matrix**
  - **Term-context Matrix**
  - **Count Matrix**

- Each word is represented by a so-called **Count Vector**.

A simple example

- One way of creating a vector for a word:
  - Let’s **count** how often a (context) word co-occurs together with specific other words.

- He is reading a magazine
- This magazine published my story
- She buys a magazine every month
- I was reading a newspaper
- The newspaper published an article
- He buys this newspaper every day

The considered text corpus

A simple example

- One way of creating a vector for a word:
  - Let’s **count** how often a (context) word co-occurs together with specific other words.

- He is reading a **magazine**
- This **magazine** published my story
- She buys a **magazine** every month

- I was reading a **newspaper**
- The **newspaper** published an article
- He buys this **newspaper** every day

The considered target words, i.e., **magazine** and **newspaper**

A simple example

- One way of creating a vector for a word:
  - Let's **count** how often a (context) word co-occurs together with specific other words.

- He is reading a **magazine**
- This **magazine** published my story
- She buys a **magazine** every month

- I was reading a **newspaper**
- The **newspaper** published an article
- He buys this **newspaper** every day

We select a window of **size 2**
with respect to the considered target words

A simple example

- One way of creating a vector for a word:
  - Let’s **count** how often a (context) word co-occurs together with specific other words.

- He is reading a **magazine**
- This **magazine** published my story
- She buys a **magazine** every month

- I was reading a **newspaper**
- The **newspaper** published an article
- He buys this **newspaper** every day

We build the window-based co-occurrence matrix

	reading	a	this	published	my	buys	the	an	every	month	day
magazine	1	2	1	1	1	1	0	0	1	1	0
newspaper	1	1	1	1	0	1	1	1	1	0	1

A simple example

- One way of creating a vector for a word:
  - Let’s **count** how often a (context) word co-occurs together with specific other words.

- He is reading a **magazine**
- This **magazine** published my story
- She buys a **magazine** every month

- I was reading a **newspaper**
- The **newspaper** published an article
- He buys this **newspaper** every day

context words

target words

	reading	a	this	published	my	buys	the	an	every	month	day
magazine	1	2	1	1	1	1	0	0	1	1	0
newspaper	1	1	1	1	0	1	1	1	1	0	1

How does this work in general?

- We calculate this count **not only** for specific target words, but **for all** the words in the text corpus.

- Let our **corpus** contain just three sentences and the **window size be 1**:
  1. I enjoy flying
  2. I like NLP
  3. I like deep learning

- The resulting co-occurrence matrix will then be?
  - **EXERCISE**

Exercise

I enjoy flying
I like NLP
I like deep learning

The text corpus

$$X = \begin{array}{c|ccccccc|} & I & like & enjoy & deep & learning & NLP & flying \\ \hline I & & & & & & & \\ like & & & & & & & \\ enjoy & & & & & & & \\ deep & & & & & & & \\ learning & & & & & & & \\ NLP & & & & & & & \\ flying & & & & & & & \end{array}$$

Solution

I enjoy flying
I like NLP
I like deep learning

$$X = \begin{array}{c|ccccccc} & \mathit{I} & \mathit{like} & \mathit{enjoy} & \mathit{deep} & \mathit{learning} & \mathit{NLP} & \mathit{flying} \\ \hline \mathit{I} & 0 & 2 & 1 & 0 & 0 & 0 & 0 \\ \mathit{like} & & & & & & & \\ \mathit{enjoy} & & & & & & & \\ \mathit{deep} & & & & & & & \\ \mathit{learning} & & & & & & & \\ \mathit{NLP} & & & & & & & \\ \mathit{flying} & & & & & & & \end{array}$$

Solution

I enjoy flying
I like NLP
I like deep learning

$$X = \begin{array}{c|ccccccc|} & I & like & enjoy & deep & learning & NLP & flying \\ \hline I & 0 & 2 & 1 & 0 & 0 & 0 & 0 \\ like & 2 & 0 & 0 & 1 & 0 & 1 & 0 \\ enjoy & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ deep & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ learning & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ NLP & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ flying & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}$$

To recap

Using a (Window-based) Word-word Co-occurrence Matrix representation for large text corpora:

- Generates a $|V| \times |V|$ co-occurrence matrix X.
- The distinction between a target word and a context word **is arbitrary** and that we are free to exchange the two roles.

Raw frequency is a bad representation

- **Frequency** is clearly useful; if *sugar* appears a lot near *apricot*, that's useful information.

- But overly frequent words like *the*, *it,* or *they* are not very informative about the context.

- **More frequent words dominate the vectors.**
  - Need a way that resolves this frequency paradox!
  - Can use a weighting scheme like:
    - TF-IDF (already seen in detail).
    - Pointwise Mutual Information (PMI).

Pointwise Mutual Information (PMI)

- **Pointwise Mutual Information**:
  - Do events x and y co-occur more than if they were independent?

$$\mathrm{PMI}(x, y) = \log_2 \left(\frac{P(x, y)}{P(x)P(y)} \right)$$

- **PMI between two words**: (Church & Hanks 1989)
  - Do words w_1 and w_2 co-occur more than if they were independent?

$$\mathrm{PMI}(w_1, w_2) = \log_2 \left(\frac{P(w_1, w_2)}{P(w_1)P(w_2)} \right)$$

Positive PMI (PPMI)

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

- **Negative values** are problematic:
  - Things are co-occurring **less than** we expect by chance.
  - Unreliable without enormous corpora.
    - Imagine w_1 and w_2 whose probability is each 10^{-6}.
    - Hard to be sure $P(w_1, w_2)$ is significantly different than 10^{-12}.

- We just replace negative PMI values by 0.
  - Positive PMI (PPMI) between w_1 and w_2:

$$\text{PPMI}(w_1, w_2) = \max\left(\log_2\left(\frac{P(w_1, w_2)}{P(w_1)P(w_2)}\right), 0\right)$$

Computing PPMI

- Let us consider the following **term-context matrix** X:

X	…	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	…
…	…	…	…	…	…	…	…

- Matrix X with W rows (words) and C columns (context words)
  - Please remember that W and C can be equal in real scenarios, in particular $W = C = |V|$.

Computing PPMI

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

- We need to compute:

$P(w_i, c_j) =$ (Count of co-occurrence of w_i and c_j in the context) / (Total word count in the context)

$P(w_i) =$ (Count of word w_i in the context) / (Total word count in the context)

$P(c_j) =$ (Count of word c_j w.r.t. target words) / (Total word count in the context)

Computing PPMI

- f_{ij} is the number of times the word w_i and c_j co-occur.

$$P(w_i, c_j) = \frac{f_{ij}}{\sum_{i=1}^{W} \sum_{j=1}^{C} f_{ij}}$$

$$P(w_i) = \frac{\sum_{j=1}^{C} f_{ij}}{\sum_{i=1}^{W} \sum_{j=1}^{C} f_{ij}}$$

$$P(c_j) = \frac{\sum_{i=1}^{W} f_{ij}}{\sum_{i=1}^{W} \sum_{j=1}^{C} f_{ij}}$$

Computing PPMI

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

- $P(w = \text{information}, c = \text{data}) = \frac{6}{19} = 0.32$
- $P(w = \text{information}) = \frac{11}{19} = 0.58 \quad P(c = \text{data}) = \frac{7}{19} = 0.37$

Computing PPMI

		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	

- $P(w = \text{information}, c = \text{data}) = \frac{6}{19} = \boxed{0.32}$
- $P(w = \text{information}) = \frac{11}{19} = \boxed{0.58}$ $P(c = \text{data}) = \frac{7}{19} = \boxed{0.37}$

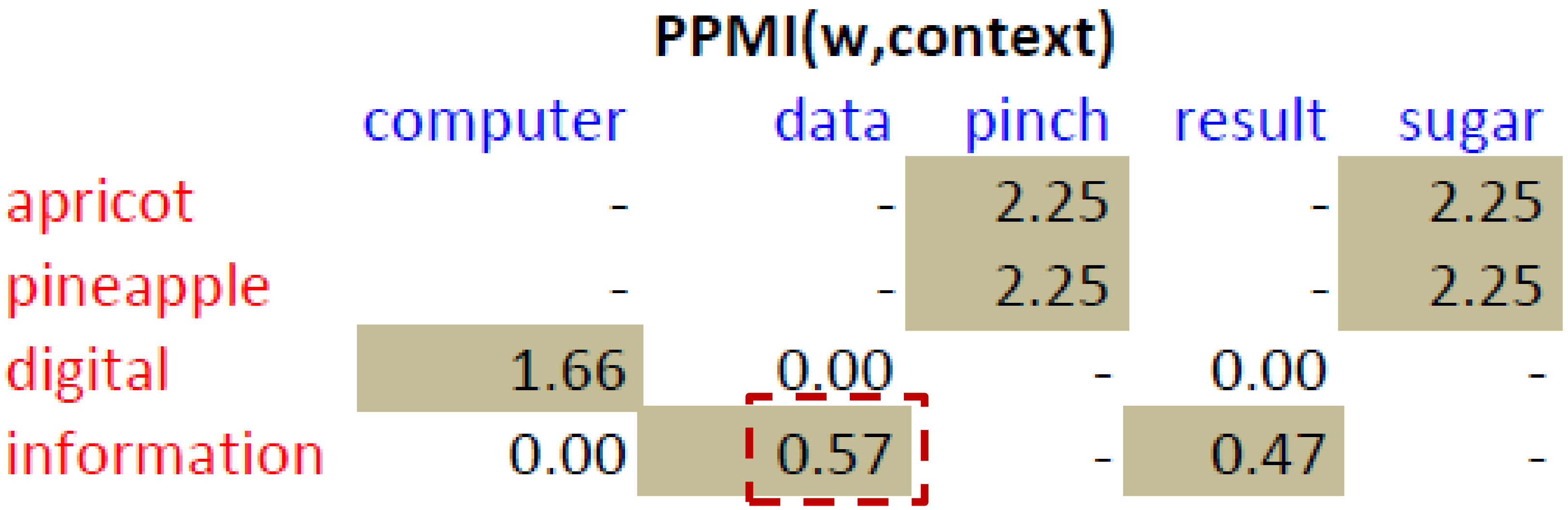
Computing PPMI

	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	

- $PPMI(\text{information, data}) = \max\left(\log_2\left(\frac{P(\text{information,data})}{P(\text{information})P(\text{data})}\right), 0\right)$

$$= \max\left(\log_2\left(\frac{0.32}{0.58*0.37}\right), 0\right) = \boxed{0.57}$$

Computing 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	-

Exercise

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

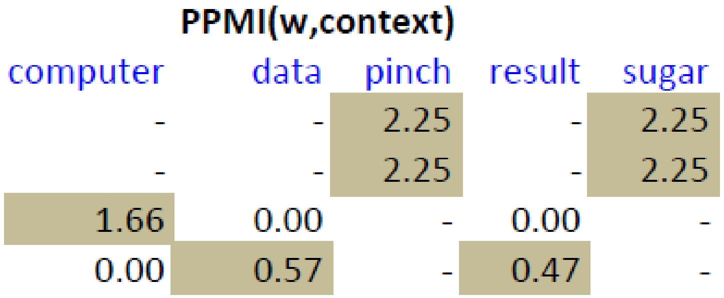
- $P(w = \text{information}, c = \text{result}) = -$
- $P(w = \text{information}) = -$ $P(c = \text{result}) = -$

Weighting (P)PMI

- (P)PMI is **biased toward infrequent events**.
  - Very rare words have very high PMI values.

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



PPMI(w,context)

computer	data	pinch	result	sugar
-	-	2.25	-	2.25
-	-	2.25	-	2.25
1.66	0.00	-	0.00	-
0.00	0.57	-	0.47	-

- Two solutions:
  1. Give rare context words slightly higher probabilities.
  2. Use add-k smoothing (which has a similar effect).
     - We add a value of k to every frequency in the term-context matrix.

Slightly higher probability to context words

- Raise the context probabilities to $\alpha = 0.75$ ($\alpha \in [0,1]$):

$$PPMI_{\alpha}(w,c) = \max\left(\log_2 \frac{P(w,c)}{P(w)P_{\alpha}(c)}, 0\right)$$

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

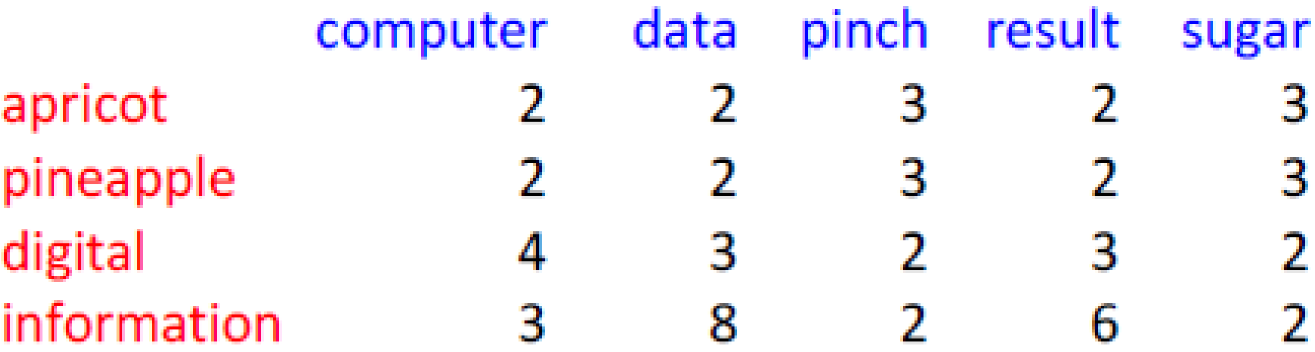
- This helps because $P_{\alpha}(c) > P(c)$ for rare c
  - Consider two context words, $P(a) = 0.99$ and $P(b) = 0.01$
  - $P_{\alpha}(a) = \frac{0.99^{0.75}}{0.99^{0.75}+0.01^{0.75}} = 0.97$ $\quad$ $P_{\alpha}(b) = \frac{0.01^{0.75}}{0.99^{0.75}+0.01^{0.75}} = 0.03$

Add-2 smoothing

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

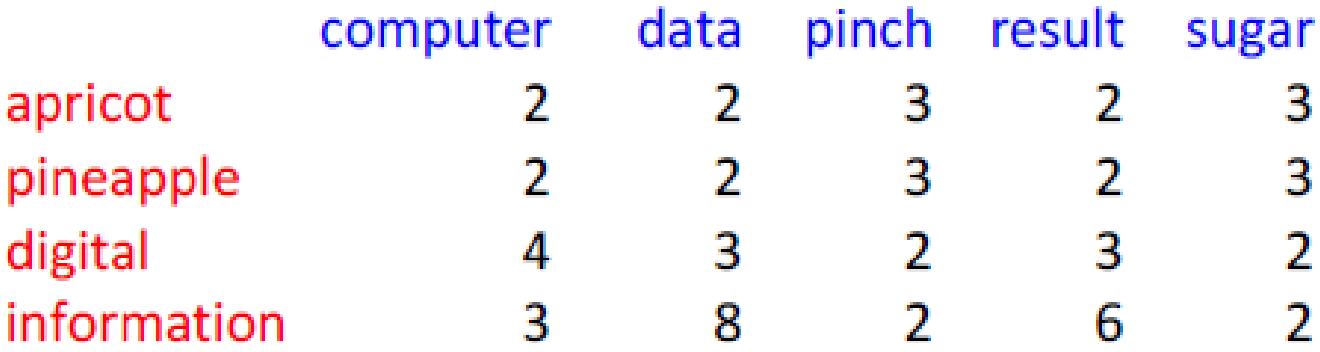
Add-2 Smoothed Count(w, context)



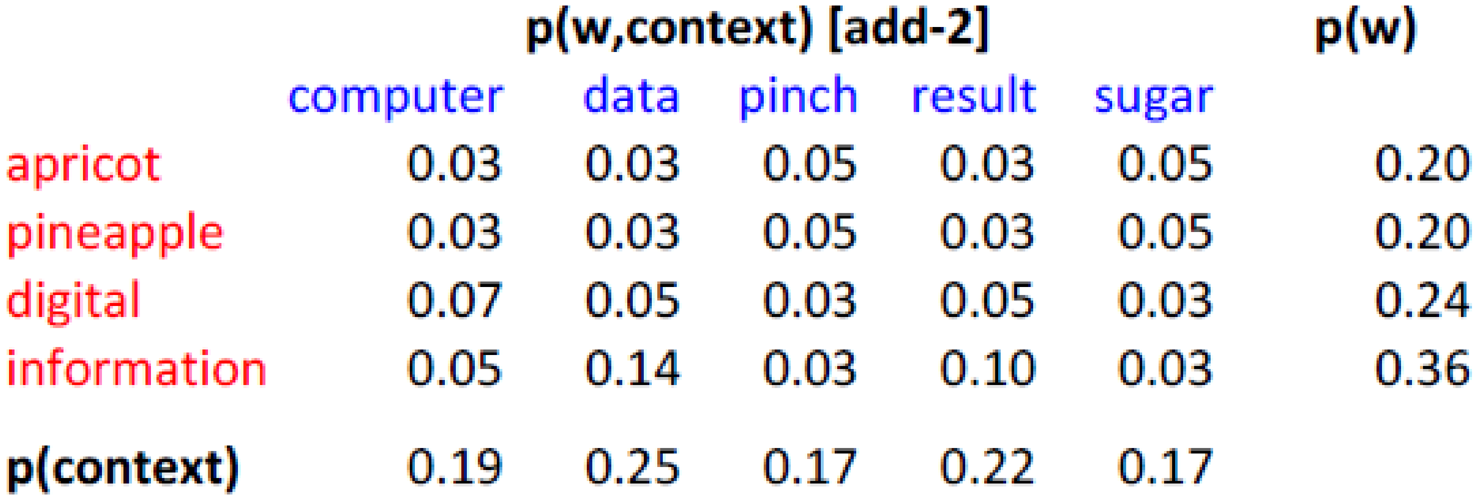
	computer	data	pinch	result	sugar
apricot	2	2	3	2	3
pineapple	2	2	3	2	3
digital	4	3	2	3	2
information	3	8	2	6	2

Add-2 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

	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	

	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	-

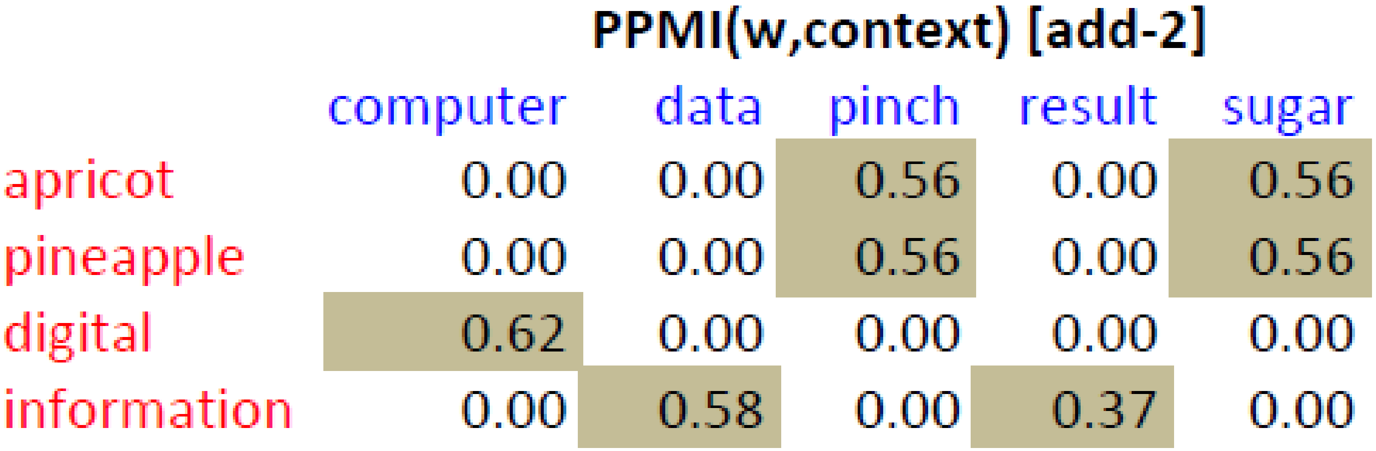
PPMI(w,context) [add-2]

	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

PPMI versus add-2 smoothed PPMI

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



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

From sparse to dense vectors

- **A Co-occurrence Matrix in reality is constituted by a very large number of words**
  - For each word, TF-IDF and PPMI vectors are:
    - **long** (length $|V|$ = 20,000 to 50,000);
    - **sparse** (most elements are equal to zero).

- There are techniques to learn **lower-dimensional vectors** for words, which are:
  - **short** (length = 50 to 1,000) (usually around 300);
  - **dense** (most elements are non-zero).

- These dense vectors are called **embeddings**.