Identifying, representing and correcting words

Benoît Sagot
Inria (ALMAnaCH), Paris (France)

MVA — Speech and Language Processing — Class #4 — 12th February, 2018

Credit and disclaimer: some of the following slides are taken from, illustrated or inspired by presentations and article figures by Jurafsky, Goldberg, Melamud and others

Processing textual data

- A text is a **sequence of characters**
  - letters, ideograms, syllabograms…
  - punctuation marks
  - whitespaces (not in all writing systems)
- Most Natural Language Processing (NLP) systems rely on a **double structuring** of such a sequence
  - Macroscopic units: “**sentences**” (utterrances, speech turns)
  - Microscopic units: “**words**”
- Typically, sentences are processed individually as sequences of words

Processing textual data

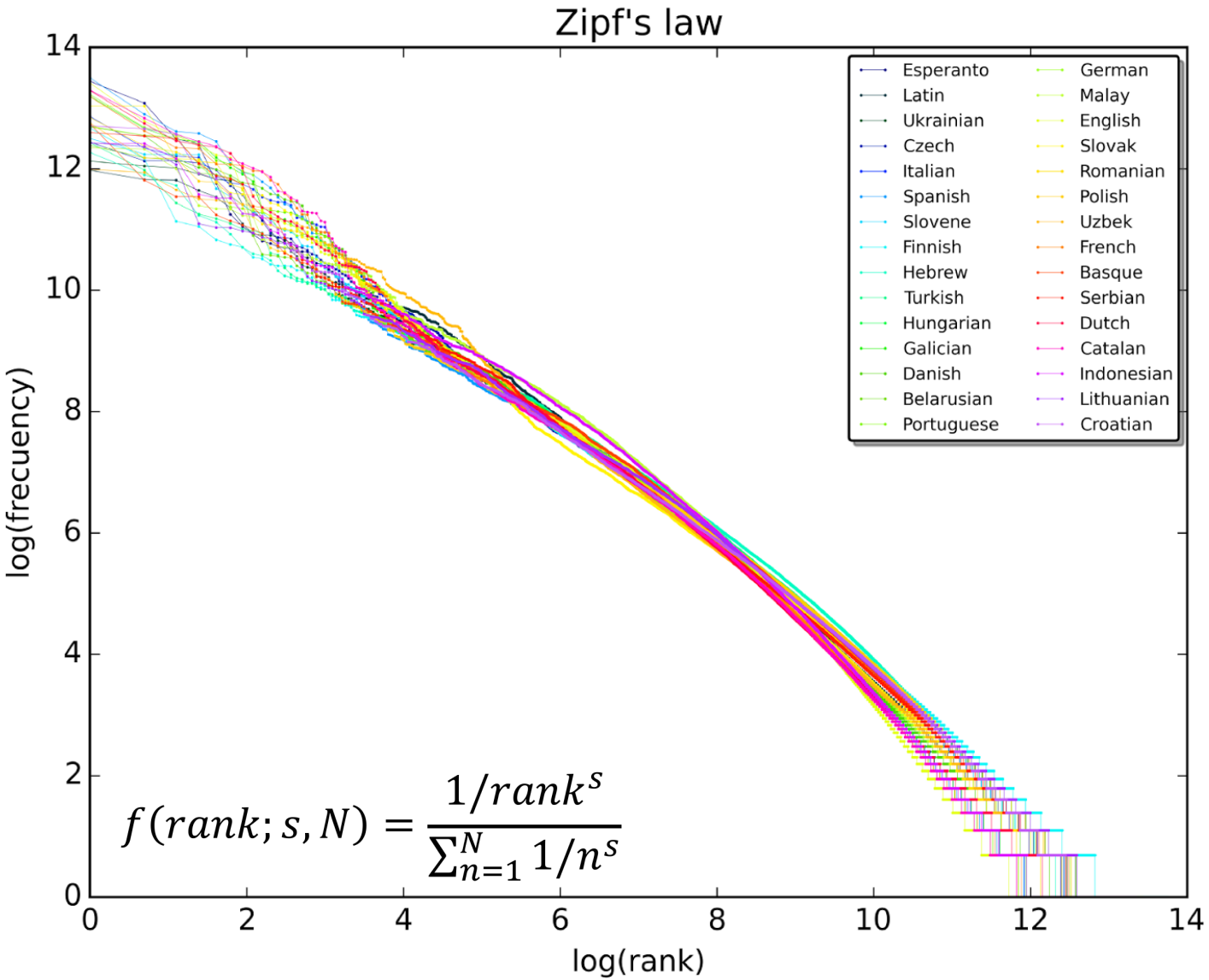
- A text is a **sequence of characters**
  - letters, ideograms, syllabograms…
  - punctuation marks
  - whitespaces (not in all writing systems)
- Most Natural Language Processing (NLP) systems rely on a **double structuring** of such a sequence
  - Macroscopic units: "**sentences**" (utterrances, speech turns)
  - Microscopic units: "**words**"
- Typically, sentences are processed individually as sequences of words
- This raises **two challenges**:
  - How do we represent words?
  - How do we identify words and sentences in raw texts?

Word representations

Lexical sparsity

A plot of the rank versus frequency for the first 10 million words in 30 Wikipedias

(source: Wikipedia; data: dumps from Oct 2015)



Lexicons and thesauruses

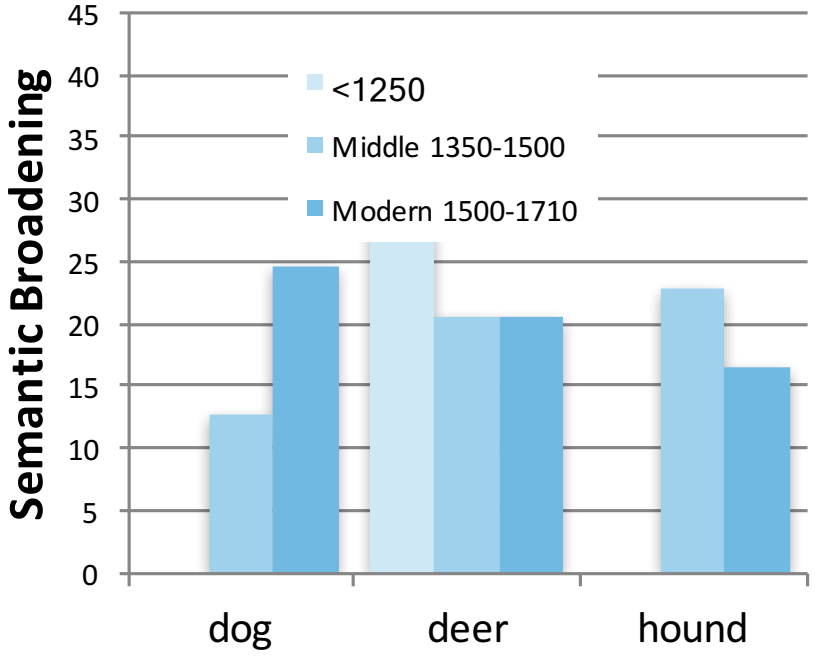
- **Advantages**
  - Possibility to encode rich linguistic information and to cover rare cases not seen in corpora
  - It is another source of linguistic information, next to annotated corpora
- **Drawbacks**
  - Costly to develop, do not exist for all languages
  - Static, fixed meanings and words
  - Limited coverage
  - Structural organisation not always relevant (it is difficult to create a hierarchy of meanings for adjectives and verbs)

Vector representations: what for?

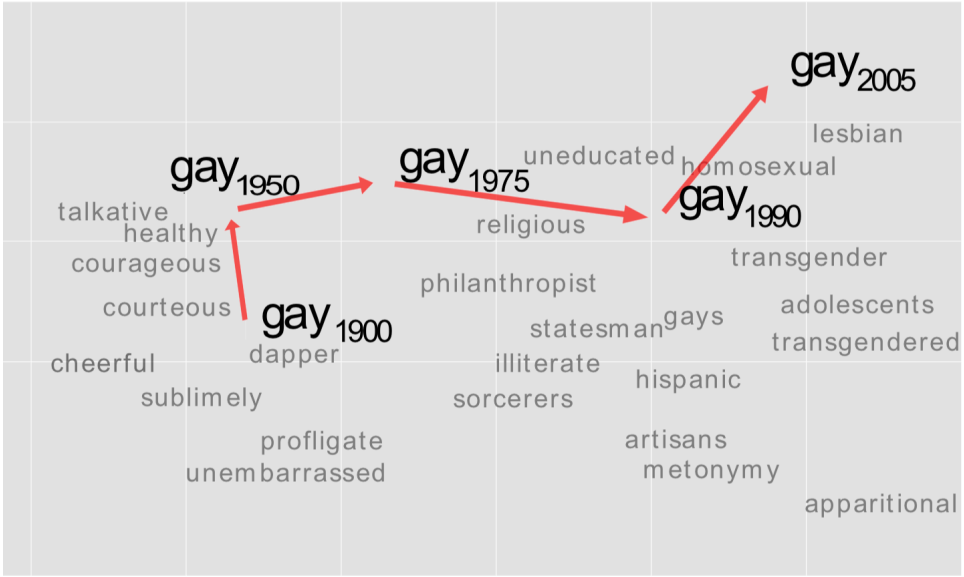
- Question answering:
  - Question: *How **tall** is Mt. Everest?*
    Candidate answer: *The official **height** of Mount Everest is 8848m*
- Plagiarism detection:
  - *Mainframes **are primarily** referred to as **large** computers with **rapid**, advanced processing capabilities that **can execute and** perform tasks **equivalent to many** Personal Computers (PCs)*
  - *Mainframes **usually are** referred to as large computers with **fast**, advanced processing capabilities that **could** execute and perform **by itself** tasks **that may require a lot of** Personal Computers (PCs)*
- We need to be able to measure **word similarity**

Word similarity for historical linguistics: semantic change over time

Sagi *et al.* 2013



Kulkarni *et al.* 2015



Distributional models of meaning

- 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!”
- **Distributional hypothesis**

Distributional models of meaning

- Nida example: Suppose I asked you what is tesgüino?

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

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

Different kinds of vector models

- **Sparse vector representations**
  1. Mutual-information weighted word co-occurrence matrices
- **Dense vector representations**
  2. Brown clusters
  3. Singular value decomposition (and Latent Semantic Analysis)
  4. Neural-network-inspired models (skip-grams, CBOW)
- **Shared intuition**
  - The meaning of a word is modelled by “embedding” it in a vector space
  - A meaning is represented as a vector
  - The meaning of a word is called a “**word embedding**”

Word and co-occurrence vectors

Cooccurrence matrices

- **Cooccurrence** = appearing in the same environment
  - document
  - immediate context
- **Coocurrence matrix** = frequency counts of *(word, environment)* pairs
- Two main categories of cooccurrence matrices
  - **Term-document matrix**: how often word *i* occurs in document *j*
  - **Term-term (word-word, word-context) matrix**: how often word *i* occurs in the immediate vicinity of word *j*

Term-document matrix

- **Each cell: count of word *w* ∈ *V* in a document *d* ∈ *D***

	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

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

	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

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

	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

Similarity in term-document matrices

- Two documents are similar if their vectors are similar

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

Similarity in term-document matrices

- Two documents are similar if their vectors are similar
- 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

Similarity in word-word matrices

- Instead of entire documents, use **smaller contexts**
  - Paragraph
  - Window of ± 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 $|V|^2$

Word-word matrix

- Example with 7-word contexts

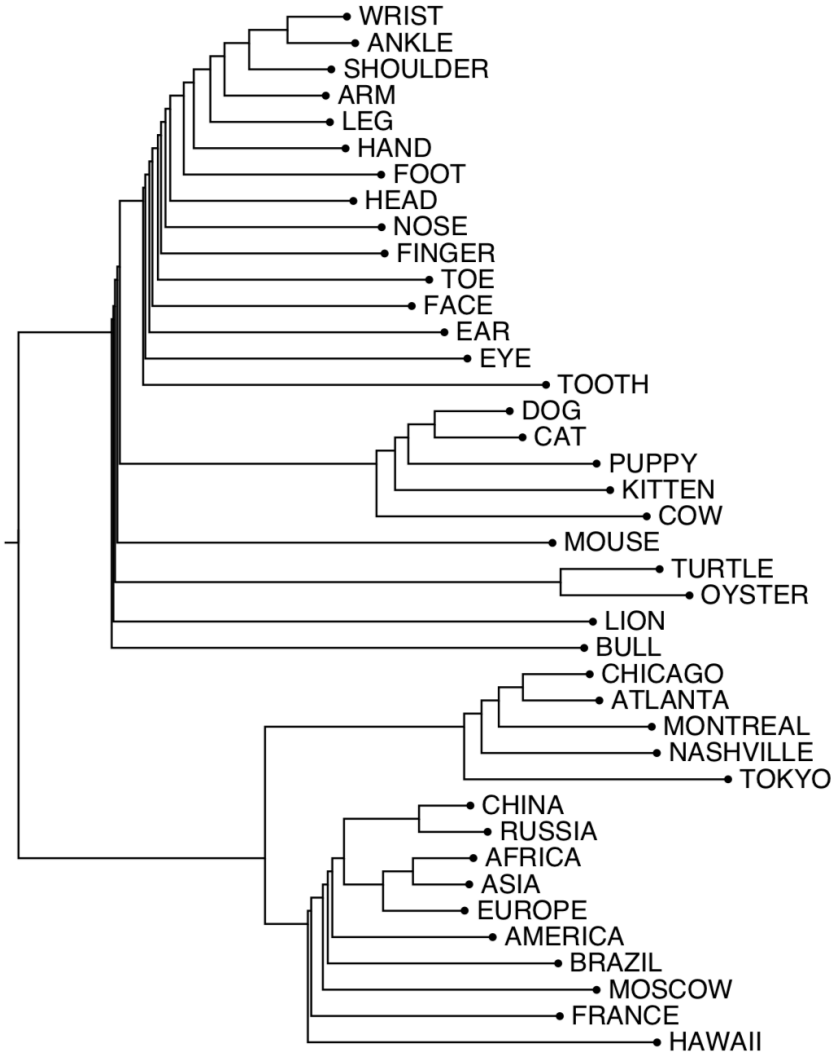
sugar, a sliced lemon, a tablespoonful of **apricot** preserve or jam, a pinch each of,
their enjoyment. Cautiously she sampled her first **pineapple** and another fruit whose taste she likened
well suited to programming on the digital **computer**. In finding the optimal R-stage policy from
for the 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 500,000 x 500,000
  - **Very sparse** (most values are 0)
  - That's acceptable, since there are lots of efficient algorithms for sparse matrices.
- Similarity is measured using the **cosine** between two vectors (i.e. the **dot product** between normalised vectors)
- The **size of context windows** depends on your goals
  - The **shorter** the windows, the more **syntactic** the representation
    1-3 ~ syntactic similarity
  - The **longer** the windows, the more **semantic** the representation
    4-10 ~ semantic/topical similarity

Similarity-based hierarchical clustering



First-order vs. second-order similarity

- 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 neighbours.
  - *wrote* is a second-order associate of words like *said* or *remarked*.

Positive Pointwise Mutual Information

Problems with raw counts

- Raw word frequency is not a great measure of association between words
  - It is very skewed
    - For ex.: “the” and “of” are very frequent, but maybe not the most discriminative
- We would be more interested in a measure that would give more importance to context words that are **more informative** about the target word
  - Positive Pointwise Mutual Information (PPMI)

Pointwise Mutual information (PMI)

- General definition: do events *x* and *y* cooccur more than if they were independent?

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

- PMI between two words: do words *x* and *y* cooccur more than if they were independent?

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

- PMIs range from -∞ to +∞

Positive Pointwise Mutual information (PPMI)

- **Negative PMI values are problematic**
  - They correspond to words co-occurring **less** than we expect by chance
  - Unreliable without a really huge corpora
    - Imagine two rare words w_1 and w_2 (say, frequency = 10^{-6}): it is hard to compare P(w_1,w_2) with 10^{-12} and know whether the difference is statistically significative…
  - It is unclear whether we have intuitions about unrelatedness
    - We are interested in similarity (relatedness)
- **We replace negative PMI values by 0 => PPMI**

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

PPMI on an example

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
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 PPMI

- **PPMI is biased toward infrequent events**
  - Very rare words have very high PMI values
- Two solutions
  - Transform counts using an exponential (exponent <1) to give rare context words slightly higher counts

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

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

  - Use "**Laplace smoothing**", i.e. **add 1 to all counts** (it has a similar effect)
    - Or **add 2**, etc.

Example with add-2 smoothing

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

Beyond cosine similarity

- Another way to improve over cosine similarity on count matrices is to use **more sophisticated similarity measures**
  - A few examples:

$$\text{sim}_{\text{cosine}}(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}| \cdot |\vec{w}|}$$

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

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

Beyond cosine similarity

- When working on word-document matrices, the most frequently used transformation is not PPMI but **tf-idf**

$$(\text{tf-idf}_D)_{ij} = (\text{tf}_D)_{ij} \cdot (\text{idf}_D)_{ij}$$

$(\text{tf}_D)_{ij}$ = "term frequency" = #occurrences of the word w_i in document d_j (can be normalised by the document length, binarised, log…)

$(\text{idf}_D)_{ij}$ = "inverse document frequency" = $\log\left(|D| / \text{df}_i\right)$, where df*i* is the number of documents containing w_i

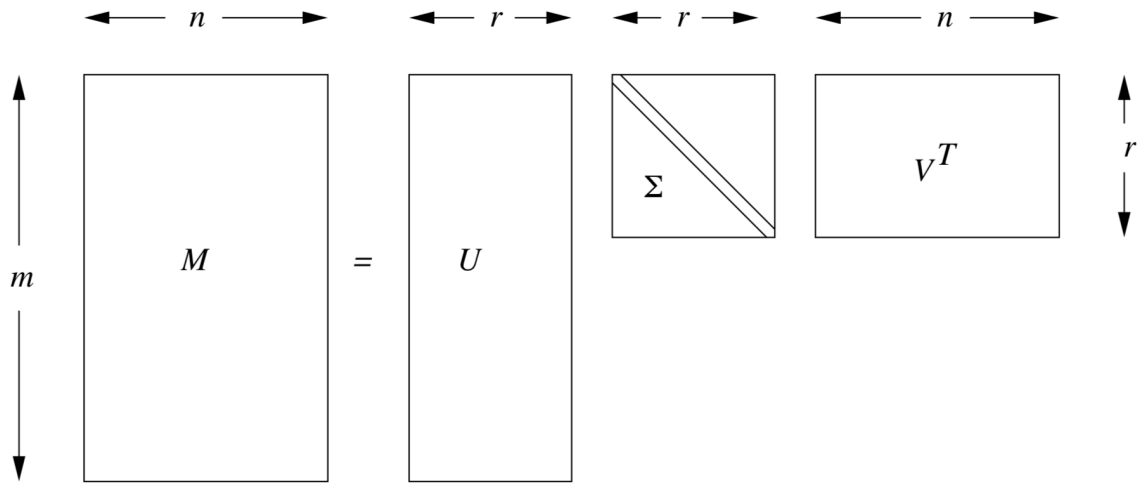
Building dense vectors using SVD

Sparse vs. dense vectors

- **PPMI vectors are sparse**
  - Up to 500,000 dimensions, if not more (depends on the language)
  - Most values are 0
  - Cosine distance not optimal on such vectors, no information is shared between similar words
  - Many parameters to use, many weights to train in machine learning algorithms using such vectors
- **How can we get dense, low-dimensionality vectors?**
  - Many techniques have been proposed. 2 examples:
    - Classical: singular value decomposition (SVD)
    - More recently: side-effect of predictive neural models

Singular Value Decomposition

- **Theorem**
  for any rectangular *m*×*n* real matrix M of rank *r*, there exist
  - an *m*×*r* orthogonal matrix U,
  - an *r*×*r* diagonal matrix Σ whith diagonal values ≥ 0 ("**singular values**"),
  - and an *n*×*r* orthogonal matrix V
  - such that **M = U Σ V^T**
- If singular values are sorted in decreasing order (e.g. amount of variance captured by the corresponding dimension), then Σ is unique



Singular Value Decomposition: an example

	Matrix	Alien	Star Wars	Casablanca	Titanic
John	1	1	1	0	0
Jack	3	3	3	0	0
Jill	4	4	4	0	0
Jenny	5	5	5	0	0
Jane	0	0	0	4	4
Joe	0	0	0	5	5
Jim	0	0	0	2	2

Singular Value Decomposition: an example

	Matrix	Alien	Star Wars	Casablanca	Titanic
John	1	1	1	0	0
Jack	3	3	3	0	0
Jill	4	4	4	0	0
Jenny	5	5	5	0	0
Jane	0	0	0	4	4
Joe	0	0	0	5	5
Jim	0	0	0	2	2

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 0 & 0 & 2 & 2 \end{bmatrix} = \begin{bmatrix} .14 & 0 \\ .42 & 0 \\ .56 & 0 \\ .70 & 0 \\ 0 & .60 \\ 0 & .75 \\ 0 & .30 \end{bmatrix} \begin{bmatrix} 12.4 & 0 \\ 0 & 9.5 \end{bmatrix} \begin{bmatrix} .58 & .58 & .58 & 0 & 0 \\ 0 & 0 & 0 & .71 & .71 \end{bmatrix}$$

M U Σ V^{T}

SVD applied to term-document matrices: Latent Semantic Analysis

- **We discard all latent dimensions apart from the first *k* ones** (e.g. *k*=300)
  - U is replaced by an $m \times k$ matrix U_k,
    Σ is replaced by a $k \times k$ diagonal matrix Σ_k with only the *k* first singular values,
    U is replaced by an $n \times k$ matrix V_k
    and $\mathbf{M \approx U_k \Sigma_k V_k^T}$
  - We get an optimal approximation of M (least-square)
- We get a *k*-dimensional vector for each word (an "**embedding**") by reading U_k's rows

SVD applied to word-word matrices

- **Same technique**
  - Only difference: $m = n$
- Again, we only keep the top k dimensions
  - In fact, it might help to discard the first dimension(s) and keep the k following ones
- Again, we get a k-dimensional vector for each word (an "**embedding**") by reading U_k's rows

Does it work better than sparse vectors?

- **In short, it does**
  - Lower dimensions represent information that is not important, not necessarily significant: **denoising**
  - Removing lower dimensions results in **generalisations**, including capturing **higher-order cooccurrence**
  - Fewer dimensions = models easier to learn (**fewer weights**)

Building dense vectors using a neural network

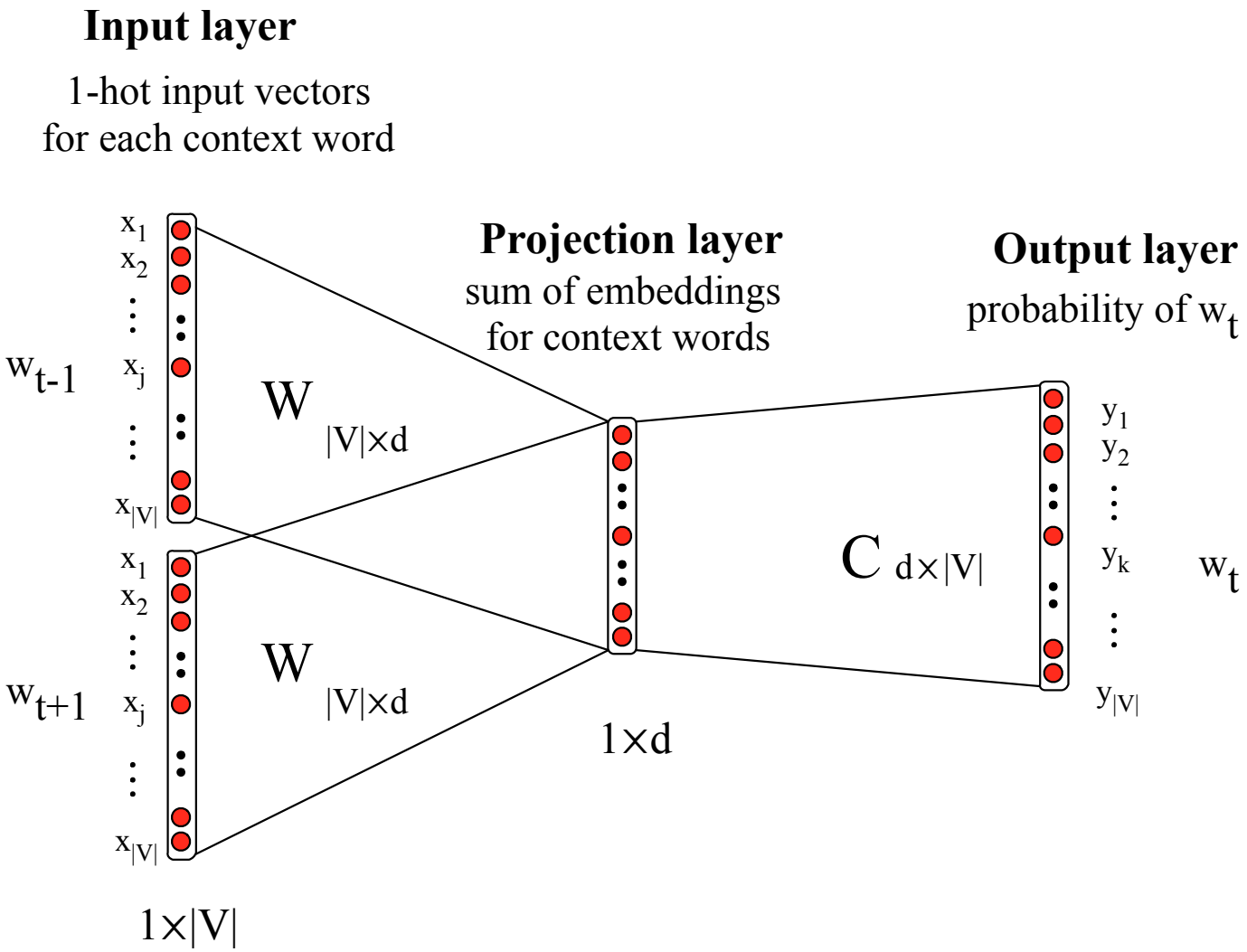
Prediction-based embeddings

- Different approach for implementing the same intuition
  - Underlying principle is still the distributional hypothesis
  - Instead of starting with **word counts to quantify the notion of distribution**, we will **learn to predict words based on distributional properties of their contexts**
  - We will do so **using a neural approach**
- Underlying idea: we will train a neural network to perform a given word-based, **auxilliary task** and extract **intermediate representations** as word representations (word embeddings)
- Approach popularised by the **word2vec** package (Mikolov et al. 2013a,b)
  - Easy and fast to train
  - Freely available, pre-trained embeddings

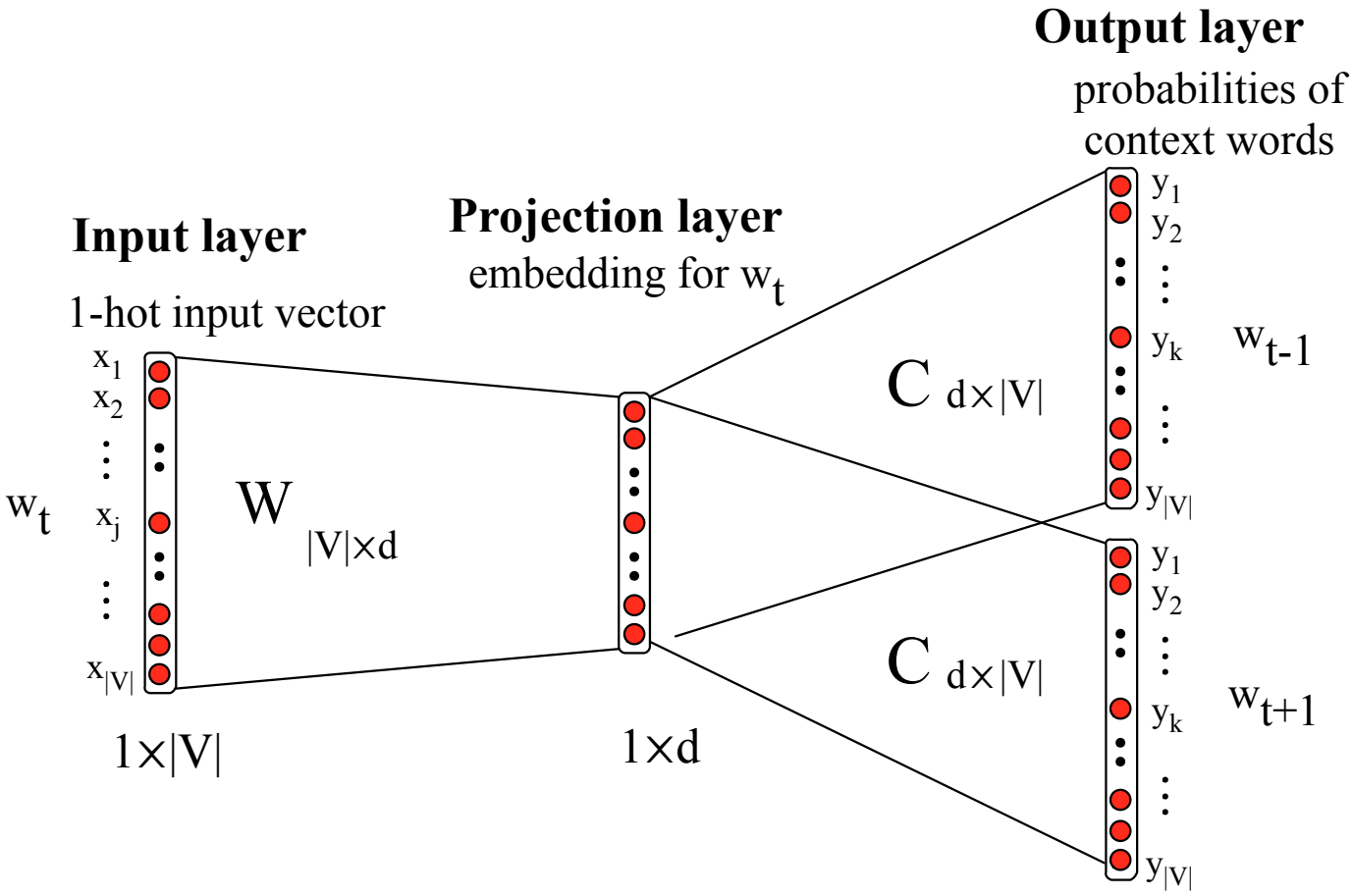
Prediction-based embeddings

- Neural network with one hidden layer
  - This hidden layer will provide the embeddings
- Input and output layers use **1-hot vector representations**
  - Word $w_i \in V$ is represented by a $|V|$-dimensional vector whose values are all 0 except for the i-th one which is 1
  - Contexts involving several words have 0 values except for those dimensions corresponding to these words
  - In the output layer, each value is a probability (notwithstanding the application of the SoftMax operation)
- Two types of predictions in word2vec:
  - Given a word, predict its neighbours: **skip-gram**
  - Given a context surrounding a position, predict the word that should fill this position: **CBoW** (continuous bag of words)

CBoW


Input layer
1-hot input vectors
for each context word
Projection layer
sum of embeddings
for context words
Output layer
probability of w_t
w_{t-1}
w_{t+1}
x_1 x_2 x_j $x_{|V|}$
$W_{|V|\times d}$
$C_{d\times|V|}$
y_1 y_2 y_k $y_{|V|}$
w_t
$1\times d$
$1\times|V|$

Skip-grams

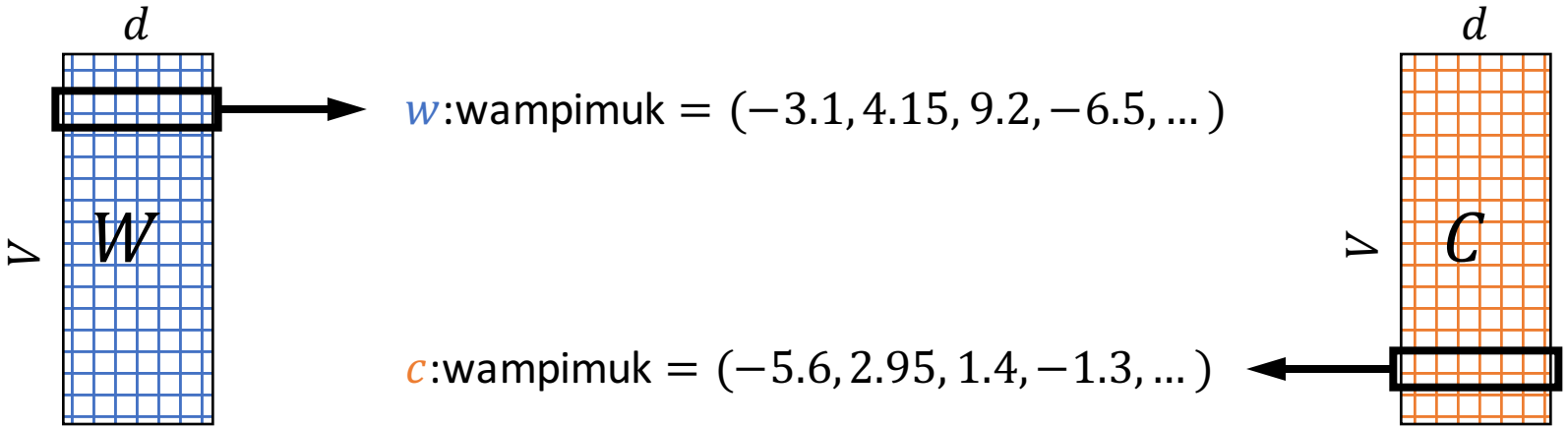

Output layer
probabilities of
context words
Input layer
1-hot input vector
Projection layer
embedding for w_t
w_t
x_1
x_2
x_j
$x_{|V|}$
$W_{|V|\times d}$
$C_{d\times|V|}$
$C_{d\times|V|}$
y_1
y_2
y_k
$y_{|V|}$
w_{t-1}
w_{t+1}
$1\times|V|$
$1\times d$

Focus on skip-grams

- The skip-gram approach creates a vector for each word $w \in V$
  - Each such vector has d latent dimensions (e.g. d=100)
- **It learns a matrix W** whose rows represent each word $w \in V$
- It also learns **a similar auxiliary matrix C** of context vectors
  - In fact, each word has two embeddings

Example (Goldberg apud Baroni):

Marco saw a furry little wampimuk hiding in the tree.



Negative sampling

Marco saw a furry little wampimuk hiding in the tree.

- Positive examples
  (wampimuk, furry), (wampimuk, little), (wampimuk, hiding), etc.

- No negative examples can be directly extracted from the data
  - **Negative sampling** is used to artificially create negative examples
  - How? By replacing the context word in such pairs with randomly selected words
    - "Randomly selected" in the sense of the unigram distribution (changing this does change the results)
    - For each positive example, *k* negative examples are built

Negative sampling

Goldberg *et al.* 2014

- **Maximize:** $\sigma(\vec{w} \cdot \vec{c})$
  - c was **observed** with w

words	**contexts**
wampimuk	furry
wampimuk	little
wampimuk	hiding
wampimuk	in

- **Minimize:** $\sigma(\vec{w} \cdot \vec{c}')$
  - c' was **hallucinated** with w

words	**contexts**
wampimuk	Australia
wampimuk	cyber
wampimuk	the
wampimuk	1985

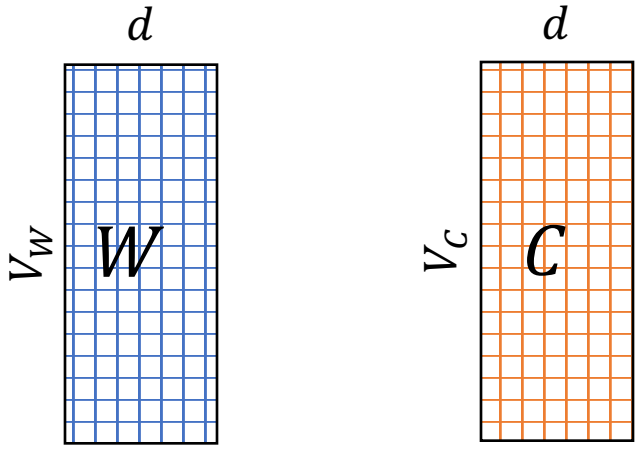
Skip-Gram with Negative Sampling (SGNS)

- Word embeddings created by the skip-gram model using negative sampling is the standard state-of-the-art

- word2vec's SGNS outperforms count-based (PPMI) vectors when used in virtually all NLP tasks
  - or does it?
  - Goldberg and colleagues have investigated what makes word2vec's SGNS so much better…

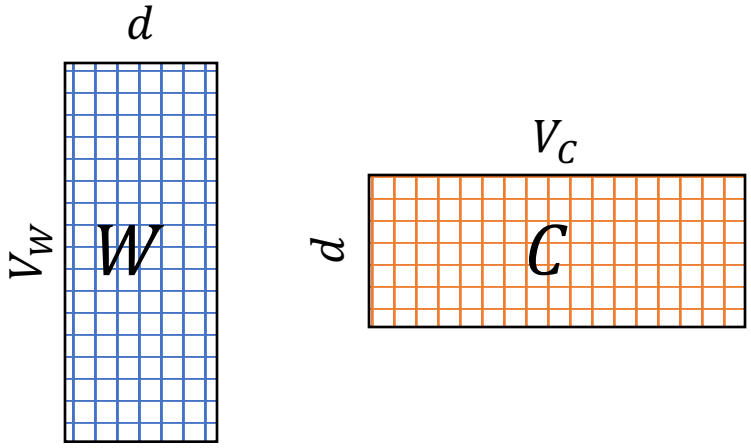
What is SGNS learning?

- Take both SGNS embedding matrices,



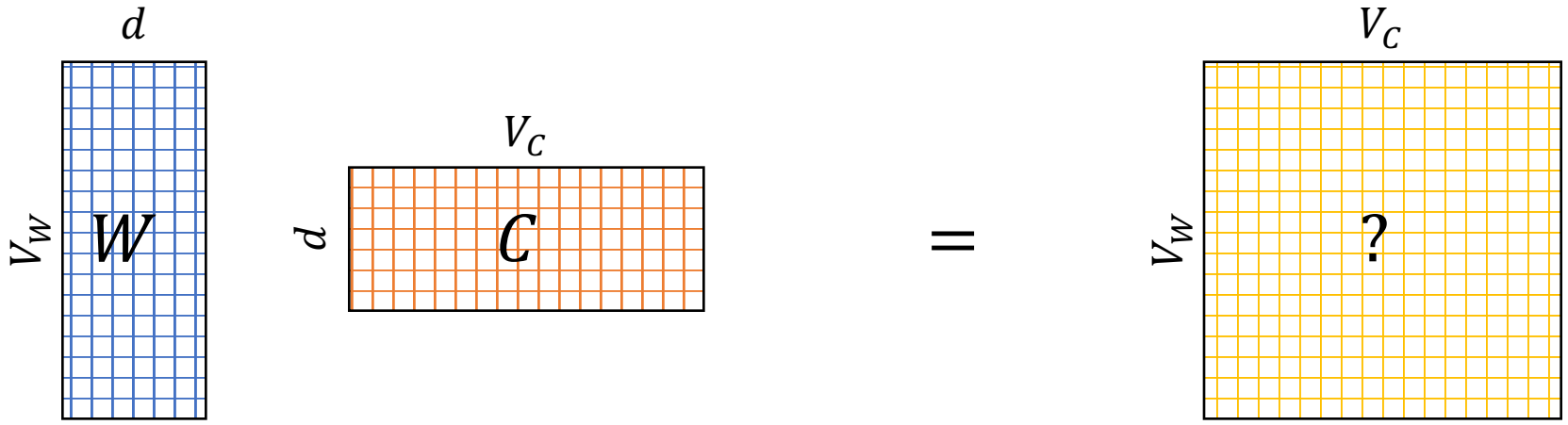
What is SGNS learning?

- Take both SGNS embedding matrices, multiply them



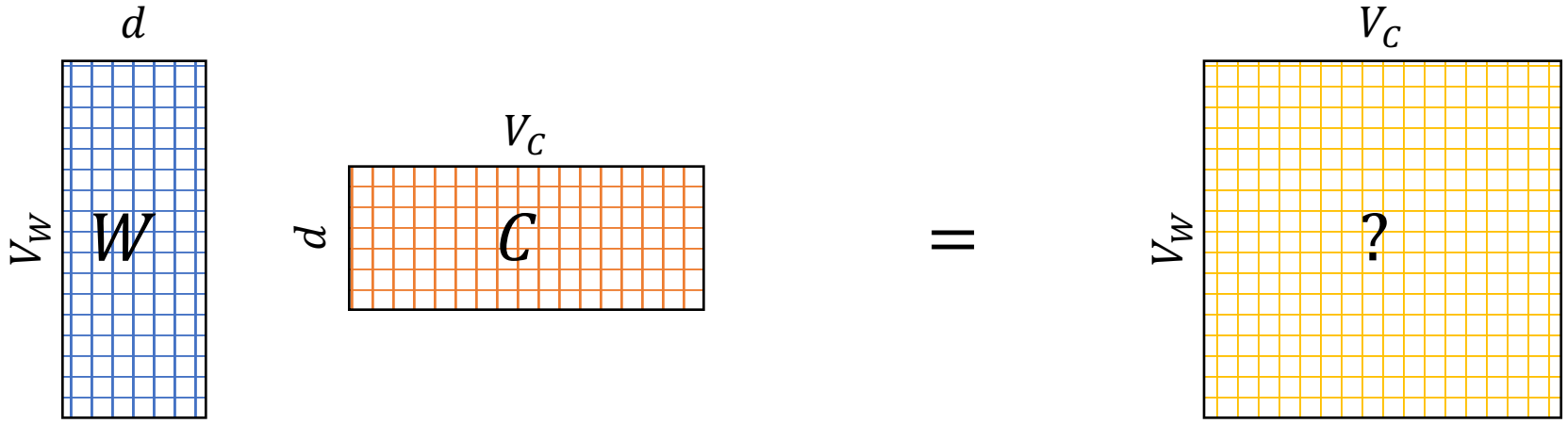
What is SGNS learning?

- Take both SGNS embedding matrices, multiply them,
- You get a $|V|^2$ matrix in which each cell describes the relation between a target word and a context word



What is SGNS learning?

- Take both SGNS embedding matrices, multiply them,
- You get a $|V|^2$ matrix in which each cell describes the relation between a target word and a context word
- It turns out that when *d* is large and after enough training iterations, **you get the PMI matrix**…



What is SGNS learning?

- Take both SGNS embedding matrices, multiply them,
- You get a $|V|^2$ matrix in which each cell describes the relation between a target word and a context word
- It turns out that when *d* is large and after enough training iterations, **you get the PMI matrix**… except for a constant:

$$W \cdot C^T \longrightarrow M_{PMI} - \log k$$

(Levy & Goldberg 2014)

What is SGNS learning?

- Take both SGNS embedding matrices, multiply them,
- You get a $|V|^2$ matrix in which each cell describes the relation between a target word and a context word
- It turns out that when *d* is large and after enough training iterations, **you get the PMI matrix**… except for a constant:

$$W \cdot C^T \longrightarrow M_{PMI} - \log k$$

(Levy & Goldberg 2014)

- So **SGNS is factorising the word-word PMI matrix**
  - **SVD does this too!**
  - Mathematically, nothing really new then…

Online question 1

What is the **main** reason that explains that, for a few years now, word2vec's SGNS embeddings have been acknowledged as the best word vectors?

1. Although the underlying maths are the same when *d* is large and the corpus size is large, SGNS performs better on real-life *d*s and corpus sizes
2. The word2vec implementation of SGNS makes use of additional, technical tricks (that could be back-ported to SVD-like approaches); these technical tricks explain the difference
3. Neural networks have become so fashionable, especially when rebranded as "Artificial Intelligence", that switching to a neural-based approach was inevitable, even if it is not really better *per se*

https://api.socrative.com/rc/KBcrE4

On the importance of little details

SGNS or SVD?

- Plenty of evidence that (neural) embeddings outperform traditional (e.g., SVD) methods
- Two famous papers:
  - “Don’t Count, Predict!” (Baroni et al., ACL 2014)
  - GloVe (Pennington et al., EMNLP 2014)

- How is it possible, since we just shown that SGNS performs a similar thing to the traditional, (P)PMI-based approach?

- The answer is: *the devil is in the detail*

Hyperparameters

- word2vec does not only introduce **2 new algorithms**
- **It also includes and tunes several hyperparameters**
- For word2vec's SGNS, here are a few of these hyperparameters:
  - **Preprocessing**
    - Dynamic Context Windows
    - Subsampling
    - Deleting Rare Words
  - **Association Metric**
    - Shifted PMI
    - Context Distribution Smoothing

Hyperparameters

- word2vec does not only introduce **2 new algorithms**
- **It also includes and tunes several hyperparameters**
- For word2vec's SGNS, here are a few of these hyperparameters:
  - **Preprocessing**
    - Dynamic Context Windows
    - Subsampling
    - Deleting Rare Words
  - **Association Metric**
    - Shifted PMI
    - Context Distribution Smoothing

Dynamic context windows

Marco saw a furry little wampimuk hiding in the tree.

Dynamic context windows

Marco saw a furry little wampimuk hiding in the tree.

word2vec: $\frac{1}{4}$ $\frac{2}{4}$ $\frac{3}{4}$ $\frac{4}{4}$ $\frac{4}{4}$ $\frac{3}{4}$ $\frac{2}{4}$ $\frac{1}{4}$

GloVe: $\frac{1}{4}$ $\frac{1}{3}$ $\frac{1}{2}$ $\frac{1}{1}$ $\frac{1}{1}$ $\frac{1}{2}$ $\frac{1}{3}$ $\frac{1}{4}$

Aggressive: $\frac{1}{8}$ $\frac{1}{4}$ $\frac{1}{2}$ $\frac{1}{1}$ $\frac{1}{1}$ $\frac{1}{2}$ $\frac{1}{4}$ $\frac{1}{8}$

Context distribution smoothing

- We said negative sampling was performed based on the **unigram** distribution: a context word *c* is chosen with probability:

$$P(c) = \frac{\#c}{\sum_{c' \in V_C} \#c'}$$

- In practice, the same trick used earlier to defined PPMI_α is applied **in order to give rare context words a slightly higher probability**:

$$P^{0.75}(c) = \frac{(\#c)^{0.75}}{\sum_{c' \in V_C} (\#c')^{0.75}}$$

- This significantly improves the results!

Comparing algorithms

- Once hyperparameters are identified and separated from algorithms
  - We can now improve all algorithms with all hyperparameters (unless impossible)
  - Which finally gives a way to compare algorithms
- Goldberg and colleagues did that (5,600+ experiments)

Settings

Classic Vanilla Setting

(commonly used for distributional baselines)

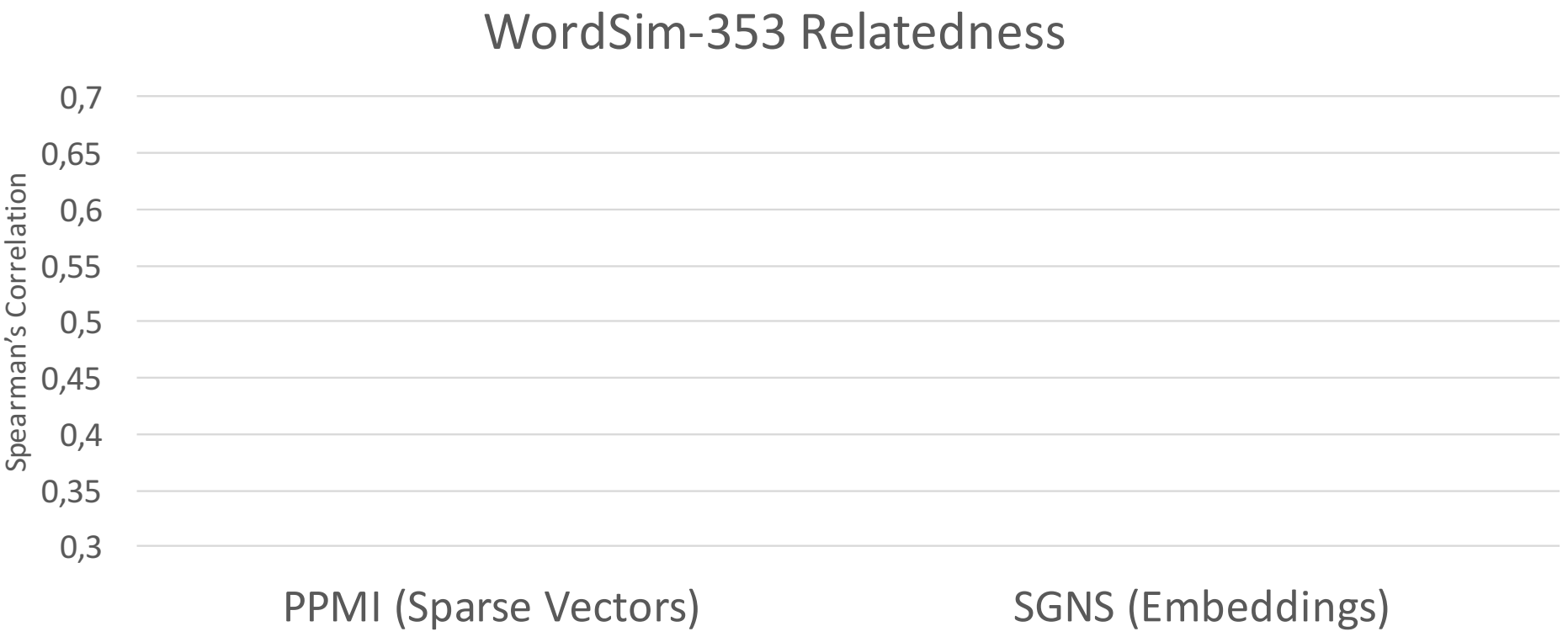
- Preprocessing
  - <None>
- Postprocessing
  - <None>
- Association Metric
  - Vanilla PMI/PPMI

Recommended word2vec Setting

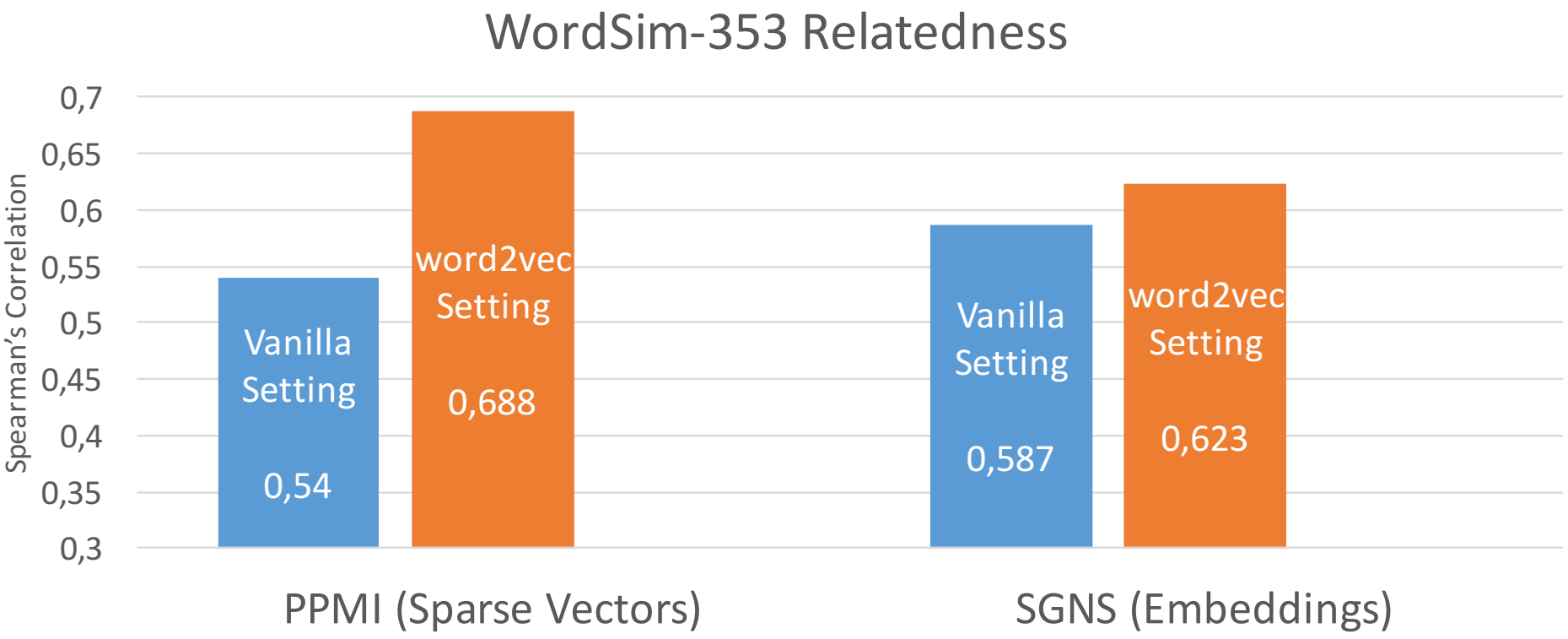
(tuned for SGNS)

- Preprocessing
  - Dynamic Context Window
  - Subsampling
- Postprocessing
  - <None>
- Association Metric
  - Shifted PMI/PPMI
  - Context Distribution Smoothing

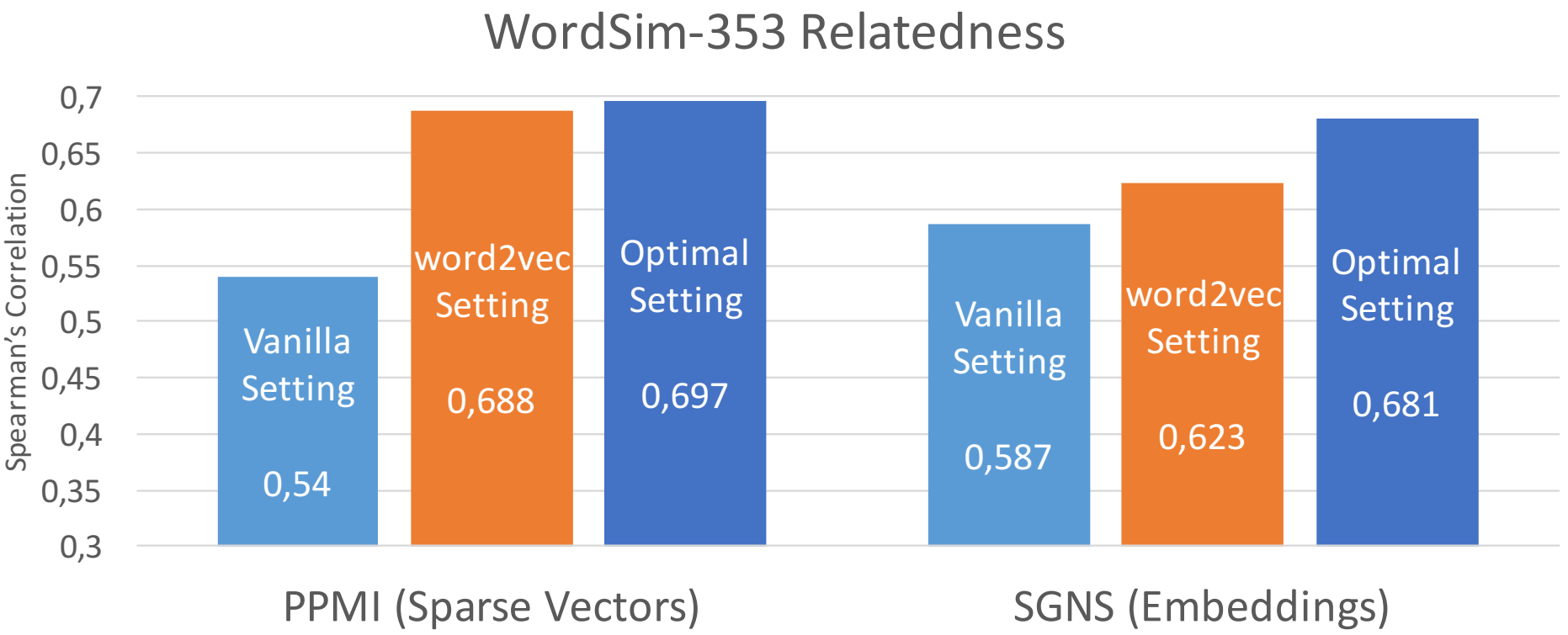
Results


WordSim-353 Relatedness
Spearman's Correlation
0,7
0,65
0,6
0,55
0,5
0,45
0,4
0,35
0,3
PPMI (Sparse Vectors)
SGNS (Embeddings)

Results


WordSim-353 Relatedness
Spearman's Correlation
0,7
0,65
0,6
0,55
0,5
0,45
0,4
0,35
0,3
Vanilla Setting
0,54
word2vec Setting
0,688
Vanilla Setting
0,587
word2vec Setting
0,623
PPMI (Sparse Vectors)
SGNS (Embeddings)

Results


WordSim-353 Relatedness
Spearman's Correlation
0,7
0,65
0,6
0,55
0,5
0,45
0,4
0,35
0,3
Vanilla Setting
0,54
word2vec Setting
0,688
Optimal Setting
0,697
Vanilla Setting
0,587
word2vec Setting
0,623
Optimal Setting
0,681
PPMI (Sparse Vectors)
SGNS (Embeddings)

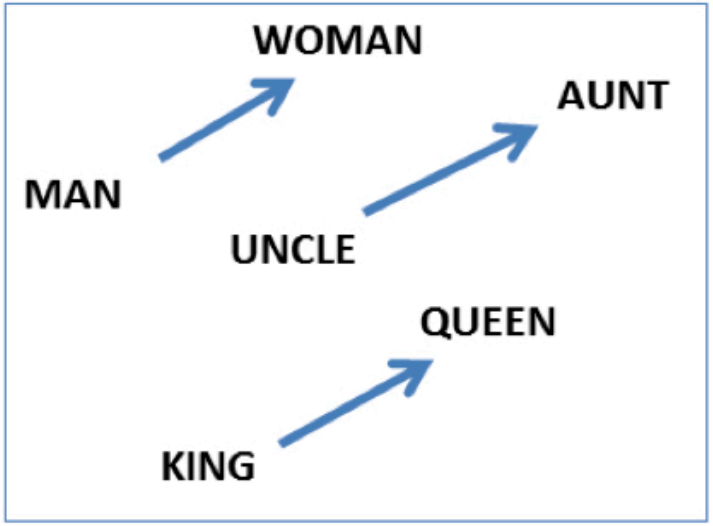
Results

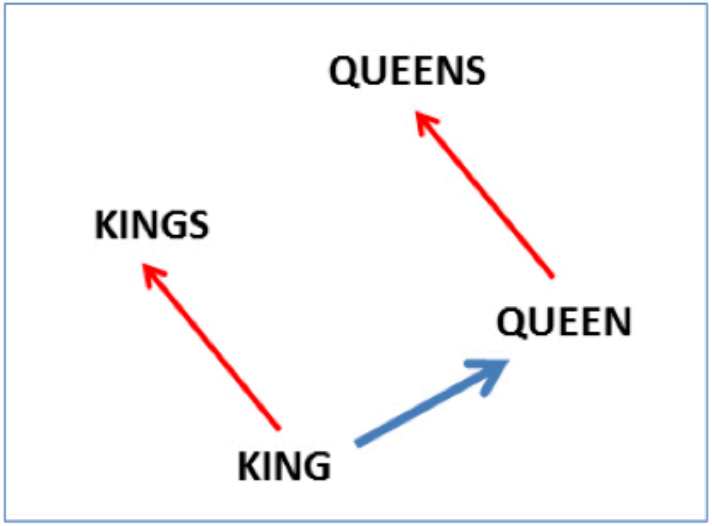
- The impact of hyperparameters is higher than that of algorithms
- Better hyperparameters often has stronger effects than more data
- Neural-based approaches do not always outperform traditional approaches
  - contra Baroni et al; 2014 (“Don’t count, predict!”): predicting is not better, but word2vec hyperparameters make it better
  - They do, sometimes. It seems that SGNS performs better than traditional methods on syntactic analogies (not on semantic analogies)

Analogy: the structure of word similarity

vector(*'king'*) – vector(*'man'*) + vector(*'woman'*) ≈ vector('queen')

vector(*'Paris'*) – vector(*'France'*) + vector(*'Italy'*) ≈ vector('Rome')





Improving context representation

Context in word2vec

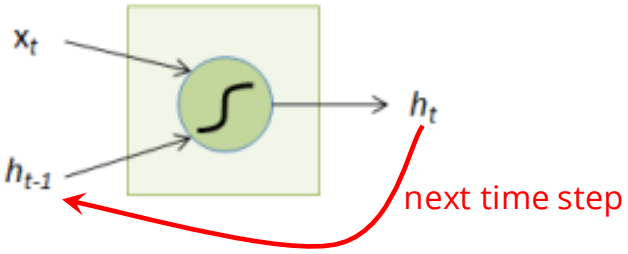
- Context in word2vec is a (weighted) bag of words
  - Context words are taken independently of one another
  - …and independently of their position with respect to the target word
- However, such information is crucial for correctly modelling the context
  - *I am not happy to see you*
    vs. *I am happy to see you*
  - *The cat eats the mouse*
    vs. *The mouse eats the cat*

Refining context representation

- What we want to do
  - Build a representation of the **whole context**, not of the individual context words
  - We need to **selectively accumulate information** from the (left and right) context
  - Obvious answer: use an **RNN**
    - Typically, use an **LSTM** (Long Short-Term Memory) layer

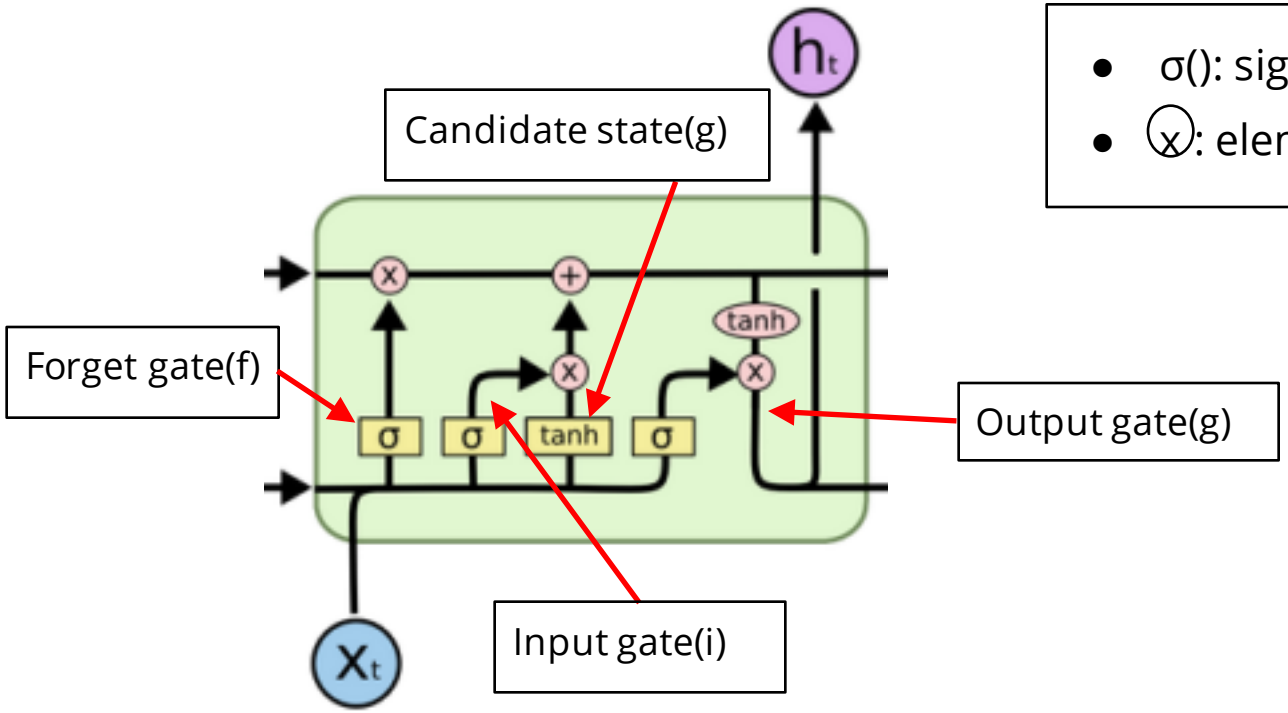
Reminder: the RNN cell

- x_t: Input at time t
- h_{t-1}: State at time t-1



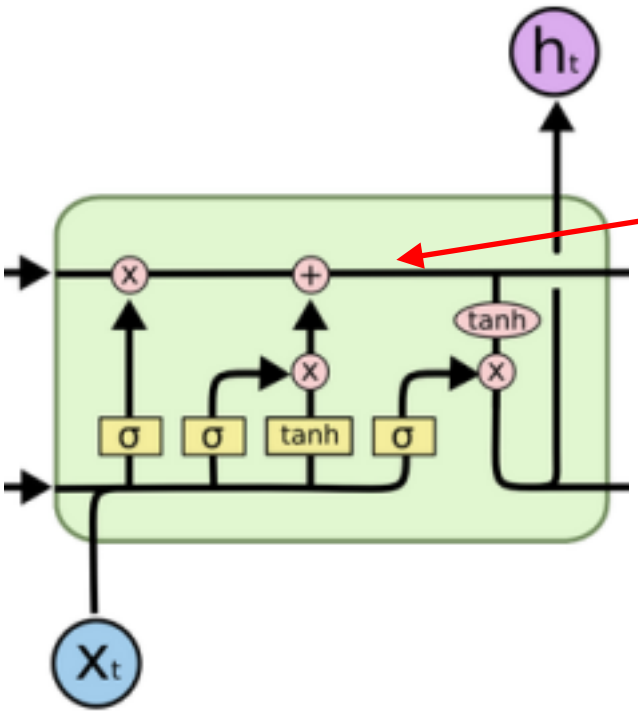
$$h_t = f(W_h h_{t-1} + W_x x_t)$$

The LSTM cell



- σ(): sigmoid non-linearity
- ⊗: element-wise multiplication

The LSTM cell



$$\begin{pmatrix} \tilde{\mathbf{f}}_t \\ \tilde{\mathbf{i}}_t \\ \tilde{\mathbf{o}}_t \\ \tilde{\mathbf{g}}_t \end{pmatrix} = \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b}$$

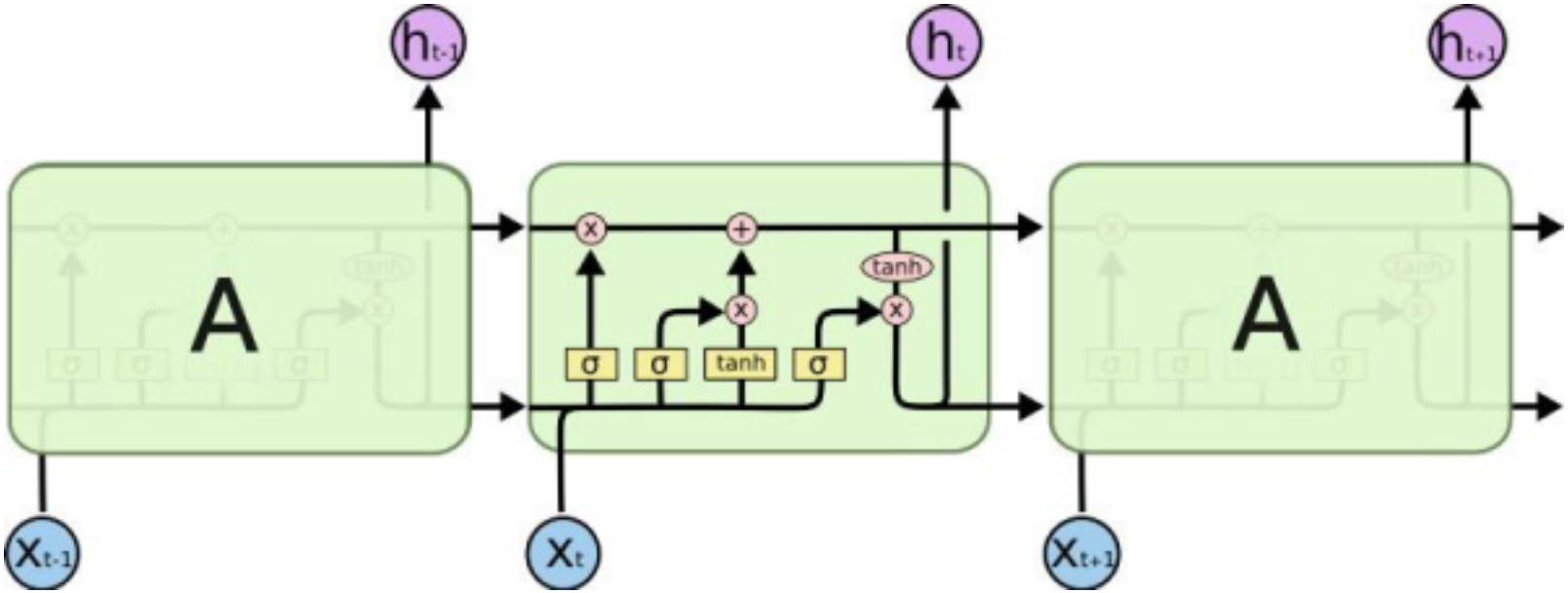
$$\mathbf{c}_t = \sigma(\tilde{\mathbf{f}}_t) \odot \mathbf{c}_{t-1} + \sigma(\tilde{\mathbf{i}}_t) \odot \tanh(\tilde{\mathbf{g}}_t)$$

$$\mathbf{h}_t = \sigma(\tilde{\mathbf{o}}_t) \odot \tanh(\mathbf{c}_t),$$

Forget old state

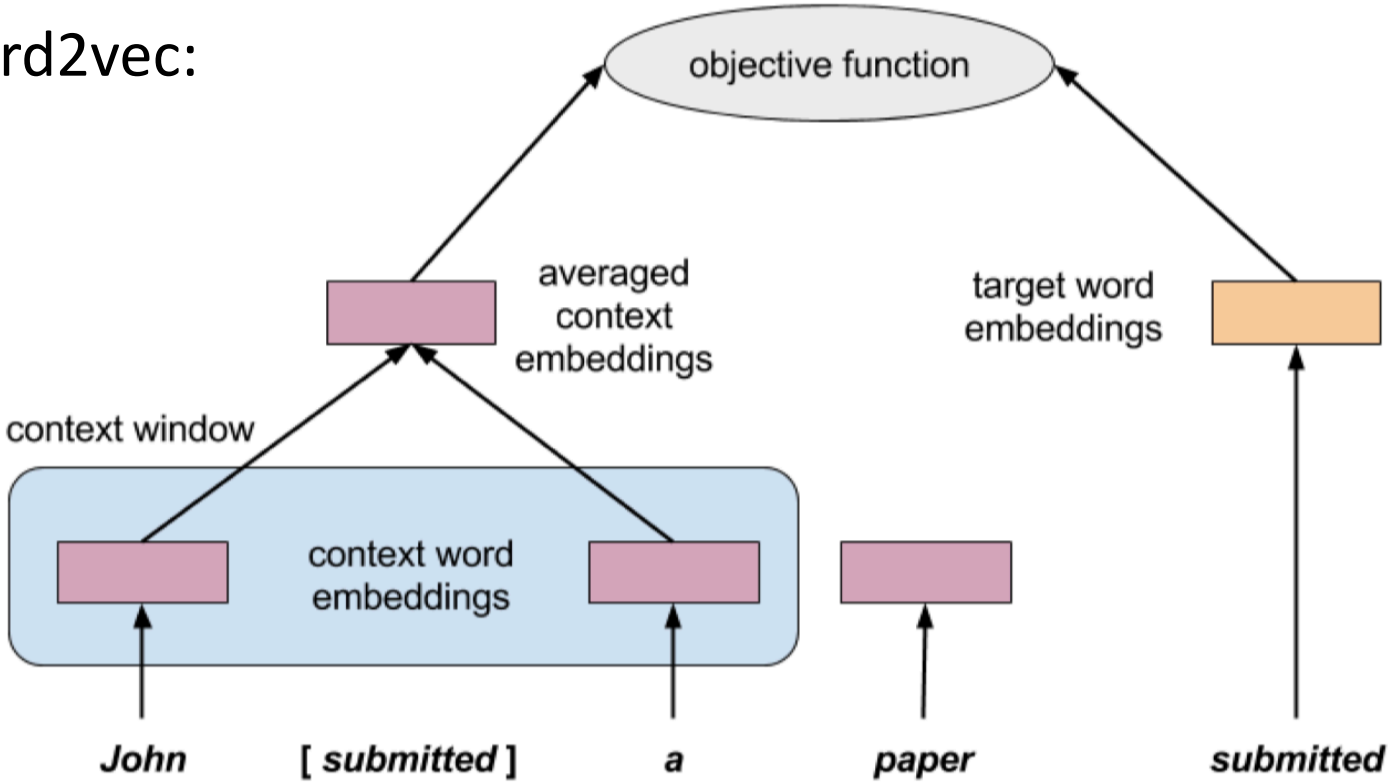
Remember new state

The LSTM network



context2vec

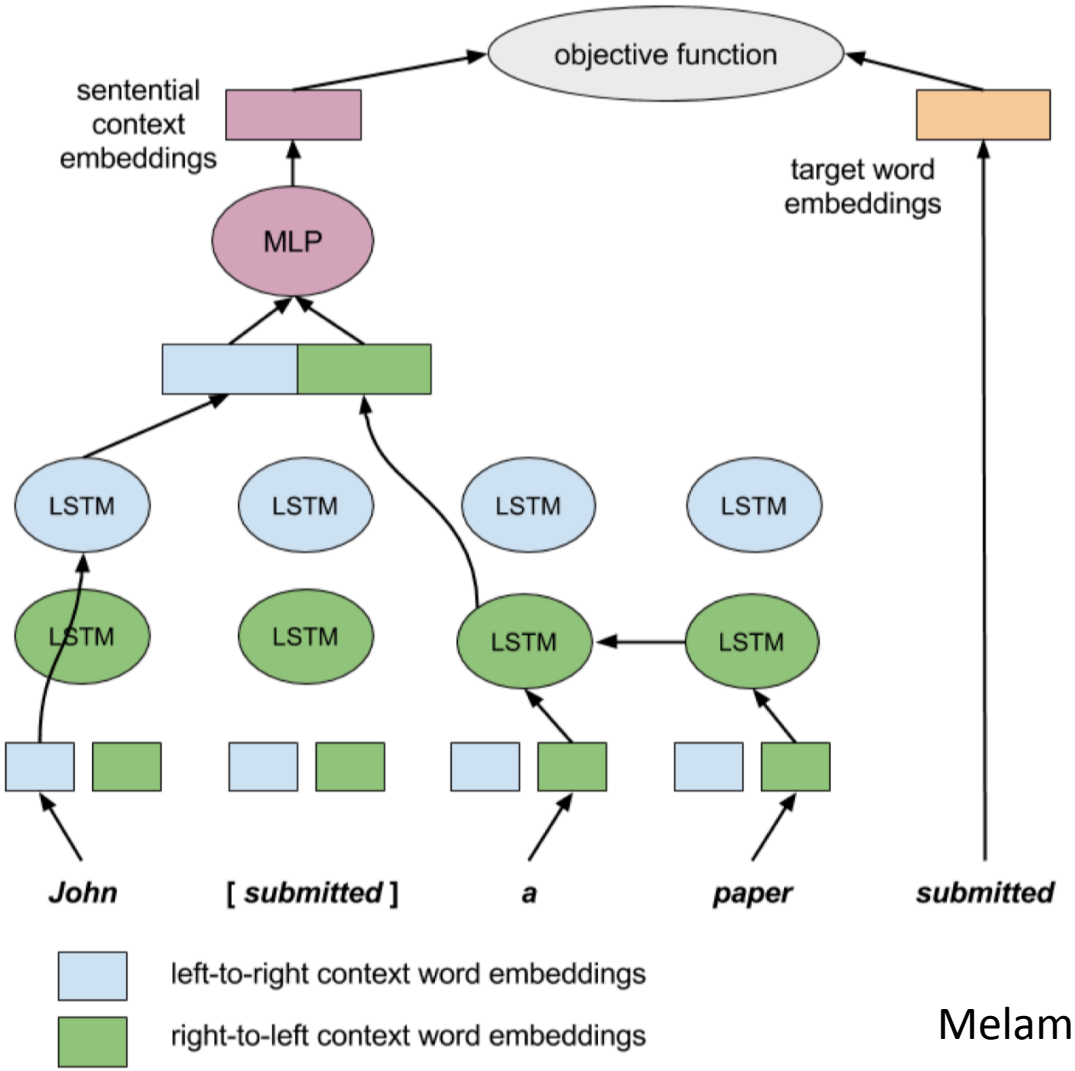
- word2vec:



Melamud et al. 2016

context2vec

- context2vec:



Melamud et al. 2016

Identifying
words and sentences

What is a sentence?

- Output of a **macroscopic segmentation**
  - **Self-contained syntactic structure**
  - Semantically related to other sentences via specific phenomena (the "**discursive**" level)
    - anaphora
    - discourse relations
  - (dubious) Prosodically marked (pauses, intonation)
  - Typographically marked
    - In the Latin script, the full stop (or period) is marked with an ambiguous, sometimes overloaded symbol; the same holds for the uppercase first letter of a "sentence"

What is a sentence?

- This is an **idealised view**
  - Self-contained syntactic structures are not that frequent in speech
  - …and do not always match the “intuitive” notion of sentence
    “Malheur à toi si tu refuses”, le menaça-t-il.
  - Typography is sometimes misleading
    Best. Movie. Ever.
    Dès maintenant, la mobilisation est de mise. Pour l’amour des mots.
    The grocery sells cucumbers, lettuce, radishes, etc.
  - Nested structures, e.g. with quotes:
    "It is basically the perfect sort of tool to find objects like 'Oumuamua. We expect to find 100s of them with the LSST," Dr Fraser says.

What is a word?

- No linguist would dare define a unique notion “word”
- At least four notions must be distinguished:
  - The prosodic word
  - The typographic word, or **token**
  - The morphosyntactic word, or **wordform** (or **form**)
  - The **semantic word**
- Let us review the last three concepts
  - And mismatches between these three notions

Tokens

- **A token is a purely typographic unit**
  - **Conventionally**, deterministically defined
- Starting point:
  - Many writing systems have “punctuation marks”
  - Some of them have a typographic separator (e.g. whitespace)
- In writing systems with a typographic separator, a token can be defined as:
  - A sequence of characters containing no separators and no punctuation marks,
  - or a punctuation mark
  - Ex.: *All of a sudden, he started playing table tennis with John Doe.*

Tokens

- **A token is a purely typographic unit**
  - **Conventionally**, deterministically defined
- Starting point:
  - Many writing systems have “punctuation marks”
  - Some of them have a typographic separator (e.g. whitespace)
- In writing systems with a typographic separator, a token can be defined as:
  - A sequence of characters containing no separators and no punctuation marks,
  - or a punctuation mark
  - Ex.: *All of a sudden , he started playing table tennis with John Doe .*

Tokens

- In writing systems without a typographic separator, the simplest way to define a token is to consider each individual character as a token
  - Remember splitting a sentence into tokens must be **deterministic** by definition
  - Ex.: 我的漢語說得不太好。

    ฉันฟังไม่เข้าใจ

Tokens

- In writing systems without a typographic separator, the simplest way to define a token is to consider each individual character as a token
  - Remember splitting a sentence into tokens must be **deterministic** by definition
  - Ex.: 我 的 漢 語 說 得 不 太 好 。

    ฉ ั น ฟ ั ง ไ ม่ เ ข ้ า ใ จ

Online question 2

Let us consider the following (real!) French tweet:
#CamelCase. TiensEncoreUnTermeQueJeNeConnaissaitPas.
How many tokens does it contain?

1. 2 tokens
2. 3 tokens
3. 4 tokens
4. 5 tokens
5. 6 tokens
6. 12 tokens
7. 14 tokens
8. 15 tokens

https://api.socrative.com/rc/KBcrE4

Wordforms (or forms)

- A **wordform** is a syntactically atomic unit
  - It can receive annotations such as a part-of-speech (noun, adjective, etc…), morphological features (plural, dative…)
  - It corresponds to the leaves in syntactic structures
  - Not easy to identify! Ambiguities…
  - Sometimes, the token modifies the form (uppercase, errors…)
- **Mismatches** between tokens and forms are numerous:
  - 1 token corresponding to multiple forms = **amalgam**
    Ex.: aux (= à les), du (= de les OR du) won't (= will not)
    Sp. dámelo (= da me lo)
  - multiple tokens corresponding to 1 form = **compound word**
    Ex.: au fur et à mesure all of a sudden
  - multiple tokens corresponding to multiple forms
    Ex.: au fur et à mesure du (= au_fur_et_à_mesure_de le)

Named entities

- A **named entity** is a real-world object that can be denoted individually
  - Standard named entities: people, locations, organisations
  - Extended named entities: dates, addresses, URLs, e-mail addresses, numbers…
- A named entity **mention** is an utterance denoting a named entity
  - Examples: Emmanuel Macron, Los Angeles, Apple Inc.
  - This denotation can be ambiguous…
- Named entity mentions have specific properties, apart from their specific, individual denotation:
  - Specific internal structure (**local grammar**), often culture-dependent more than language-dependent
    > **They can be viewed as atomic for the grammar of the language**, and therefore as **special forms**
- Ex.: *All of a sudden , he started playing table tennis with John Doe .*

Named entities

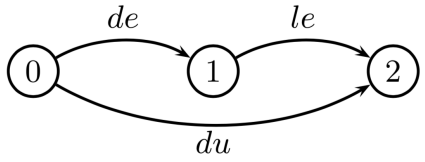
- A **named entity** is a real-world object that can be denoted individually
  - Standard named entities: people, locations, organisations
  - Extended named entities: dates, addresses, URLs, e-mail addresses, numbers…
- A named entity **mention** is an utterance denoting a named entity
  - Examples: Emmanuel Macron, Los Angeles, Apple Inc.
  - This denotation can be ambiguous…
- Named entity mentions have specific properties, apart from their specific, individual denotation:
  - Specific internal structure (**local grammar**), often culture-dependent more than language-dependent
    > **They can be viewed as atomic for the grammar of the language**, and therefore as **special forms**
- Ex.: *all_of_a_sudden , he started playing table tennis with John_Doe .*

Semantic words

- A **semantic word** is a (sequence of) form(s) with non-compositional meaning
  - I.e. its meaning cannot be inferred from the meaning of its constitutive forms
    Ex.: *pomme de terre*, *red herring*
  - Often ambiguous!
    Ex.: *Il a sorti la pomme de terre / Il a modelé une pomme de terre cuite*
    W. C. wrote how he used red herring to lay a false trail, while training hunting dogs
  - Close but distinct form the notion of **term** (a conventional unit)
    Ex.: *machine à laver* (not **appareil à nettoyer* !)
- Ex.: *all_of_a_sudden , he started playing table_tennis with John_Doe .*

Which words do we need?

- The aim of word embeddings was to represent “words” in a way that captures similarities between them
- Small context window => syntactic-ish similarity
  - Wordforms?
- Larger context window => semantic-ish similarity
  - Semantic words?
- In fact…
  - **Many people use tokens**, because they are easy to get
  - Identifying forms is non-trivial because **non-deterministic**
    We can represent this ambiguity using automata



  - Identifying semantic words is even more difficult; in fact, it is an **ill-defined task**

Noisy tokens?

- The situation is in fact even worse with noisy inputs and language variation
- Example: web texts
- It would be helpful to correct/normalise this before (or jointly with) further processing (including wordform identification)

Phenomenon	Attested example	Std. counterpart	Gloss
	Ergographic phenomena		
Diacritic removal	*demain c'est l'***ete**	*demain c'est l'été*	'tomorrow is summer'
Phonetization	*je suis* **oqp**	*je suis occupé*	'I'm busy'
Simplification	*je* **sé**	*je sais*	'I know'
Spelling errors	*tous mes* **examen**	*tous mes examens*	'All my examinations
	son *normaux*	*sont normaux*	are normal'
	Transverse phenomena		
Contraction	**nimp**	*n'importe quoi*	'rubbish'
	qil	*qu'il*	'that he'
Typographic diaeresis	**c a dire**	*c'est-à-dire*	'namely'
	c t	*c'était*	'it was'
	Marks of expressiveness		
Punctuation transgression	*Joli !!!!!!*	*Joli !*	'nice!'
Graphemic stretching	*superrrrrrrrr*	*super*	'great'
Emoticons/smileys	*:-), <3*	–	–

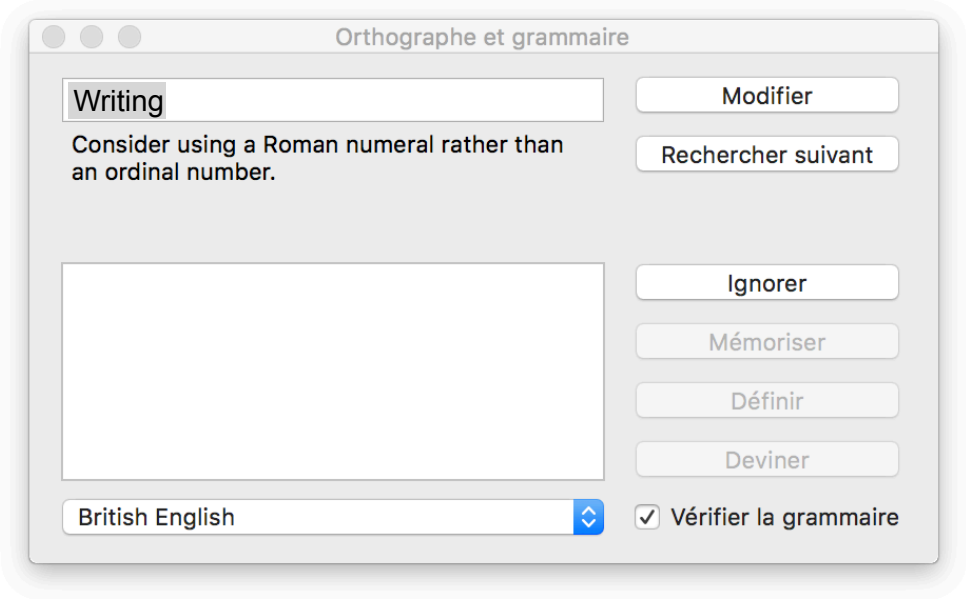
Spelling correction and edit distance

Spelling correction

- Two main application types:
  - Writign assistance

Spelling correction

- Two main application types:
  - Writign assistance



Spelling correction

- Two main application types:
  - Writing assistance
  - Correction of texts before NLP processing
- Two subtasks:
  - **Spelling error detection**
    - Maybe the hardest subtask, because many erroneous spellings produce existing words!
  - **Spelling error correction**
    - Autocorrect: *hte -> the*
    - Suggested correction
    - Suggested list of possible corrections (sorted)

Types and rates

- **Types**:
  - Non-word errors (easy to detect with a large lexicon)
    vs. real-word errors (harder to detect, e.g. “noisy channel” approach)
  - Typographical errors vs. “Cognitive” errors
  - Lexical errors vs. Grammatical errors
- **Rates**:
  - 26%: Web queries (Wang et al. 2003)
  - 13%: Retyping, no backspace (Whitelaw et al. on English & German)
  - 7%: Words corrected retyping on phone-sized “organiser”
  - 2%: Words uncorrected on “organiser” (Soukoreff and MacKenzie 2003)
  - 1-2%: Retyping (Kane and Wobbrock 2007, Gruden et al. 1983)

A non-word example

acress

Edit distance

- Creating candidate corrections
- Damereau-Levenshtein distance = minimal distance between two strings, based on a closed inventory of possible operations:
  - insertion of a character
  - deletion of a character
  - substitution of a character with another one
  - swap (transposition) between two adjacent characters
- Computed using dynamic programming
  - We fill a matrix whose (*i,j*) element stores the minimum number of operations needed to produce the *j*-prefix $w'_0 \ldots w'_j$ of the target word from the *i*-prefix $w_0 \ldots w_i$ of the source word

Levenshtein distance

```
LevenshteinDistance(s1, s2)
 1  for i ← 0 to |s1|
 2  do m[i, 0] = i
 3  for j ← 0 to |s2|
 4  do m[0, j] = j
 5  for i ← 1 to |s1|
 6  do for j ← 1 to |s2|
 7       do if s1[i] = s2[j]
 8            then m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]}
 9            else  m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1}
10  return m[|s1|, |s2|]
```

Operations: insert (cost 1), delete (cost 1), replace (cost 1)

On our non-word example

- Words at a distance 1 from `acress`

Error	Candidate Correction	Correct Letter	Error Letter	Type
acress	actress	t	–	deletion
acress	cress	–	a	insertion
acress	caress	ca	ac	transposition
acress	access	c	r	substitution
acress	across	o	e	substitution
acress	acres	–	s	insertion
acress	acres	–	s	insertion

Edit distance and spelling errors

- 80% of errors are within edit distance 1
  - Almost all errors within edit distance 2
- We must also allow insertion of space or hyphen
  - *thisidea* -> *this idea*
  - *inlaw* -> *in-law*

anagement maagement maanagement
maangement magagement magement
mamagement mamangement manaagement manaement
managaement manageement manageemnt managegment
managemaent managemant managememt managemen managemenet
managementt managemet managemetn managemnent managemnet
managemnt managemrnt managemt managenent managenment managent
managerment managhement managmeent managmement managment managnment
manament manamgement mananement manangment manasgement
manegement manegment mangaement mangagement mangagment
mangament mangement manggement mangment
mangmt menagement mgmt mgnt
mnagement mngmnt mngmt

Candidate generation

- Several options:
  1. Run through dictionary, check edit distance with each word
  2. Generate all words within edit distance ≤ *k* (e.g. *k* = 1 or 2) and then intersect them with dictionary
  3. Use a character *k*-gram index and find dictionary words that share "most" *k*-grams with word
  4. Compute them fast with a Levenshtein finite state transducer
  5. Have a precomputed hash of words to possible corrections

Refinement: weighted operations

sub[X, Y] = Substitution of X (incorrect) for Y (correct)

X	Y (correct)																									
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	0	7	1	342	0	0	2	118	0	1	0	0	3	76	0	0	1	35	9	9	0	1	0	5	0
b	0	0	9	9	2	2	3	1	0	0	0	5	11	5	0	10	0	0	2	1	0	0	8	0	0	0
c	6	5	0	16	0	9	5	0	0	0	1	0	7	9	1	10	2	5	39	40	1	3	7	1	1	0
d	1	10	13	0	12	0	5	5	0	0	2	3	7	3	0	1	0	43	30	22	0	0	4	0	2	0
e	388	0	3	11	0	2	2	0	89	0	0	3	0	5	93	0	0	14	12	6	15	0	1	0	18	0
f	0	15	0	3	1	0	5	2	0	0	0	3	4	1	0	0	0	6	4	12	0	0	2	0	0	0
g	4	1	11	11	9	2	0	0	0	1	1	3	0	0	2	1	3	5	13	21	0	0	1	0	3	0
h	1	8	0	3	0	0	0	0	0	0	2	0	12	14	2	3	0	3	1	11	0	0	2	0	0	0
i	103	0	0	0	146	0	1	0	0	0	0	6	0	0	49	0	0	0	2	1	47	0	2	1	15	0
j	0	1	1	9	0	0	1	0	0	0	0	2	1	0	0	0	0	0	5	0	0	0	0	0	0	0
k	1	2	8	4	1	1	2	5	0	0	0	0	5	0	2	0	0	0	6	0	0	0	4	0	0	3
l	2	10	1	4	0	4	5	6	13	0	1	0	0	14	2	5	0	11	10	2	0	0	0	0	0	0
m	1	3	7	8	0	2	0	6	0	0	4	4	0	180	0	6	0	0	9	15	13	3	2	2	3	0
n	2	7	6	5	3	0	1	19	1	0	4	35	78	0	0	7	0	28	5	7	0	0	1	2	0	2
o	91	1	1	3	116	0	0	0	25	0	2	0	0	0	0	14	0	2	4	14	39	0	0	0	18	0
p	0	11	1	2	0	6	5	0	2	9	0	2	7	6	15	0	0	1	3	6	0	4	1	0	0	0
q	0	0	1	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	0	14	0	30	12	2	2	8	2	0	5	8	4	20	1	14	0	0	12	22	4	0	0	1	0	0
s	11	8	27	33	35	4	0	1	0	1	0	27	0	6	1	7	0	14	0	15	0	0	5	3	20	1
t	3	4	9	42	7	5	19	5	0	1	0	14	9	5	5	6	0	11	37	0	0	2	19	0	7	6
u	20	0	0	0	44	0	0	0	64	0	0	0	0	2	43	0	0	4	0	0	0	0	2	0	8	0
v	0	0	7	0	0	3	0	0	0	0	0	1	0	0	1	0	0	0	8	3	0	0	0	0	0	0
w	2	2	1	0	1	0	0	2	0	0	1	0	0	0	0	7	0	6	3	3	1	0	0	0	0	0
x	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0
y	0	0	2	0	15	0	1	7	15	0	0	0	2	0	6	1	0	7	36	8	5	0	0	1	0	0
z	0	0	0	7	0	0	0	0	0	0	0	7	5	0	0	0	0	2	21	3	0	0	0	0	3	0

(Kernighan, Church, Gale 1990)

Refinement: weighted operations

- Such data makes it possible to compute substitution probabilities
- Can be generalised to all 4 types of operations
  - Use add-1 smoothing to allow for unseen operations

> **Channel model**

Candidate Correction	Correct Letter	Error Letter	x|w	P(x|word)
actress	t	–	c|ct	.000117
cress	–	a	a|#	.00000144
caress	ca	ac	ac|ca	.00000164
access	c	r	r|c	.000000209
across	o	e	e|o	.0000093
acres	–	s	es|e	.0000321
acres	–	s	ss|s	.0000342

Language model

- Noisy channel approach: multiply channel model by language model
- We want to prefer more frequent corrections
  - Simplest idea: unigram probability

Candidate Correction	Correct Letter	Error Letter	x|w	P(x|word)	P(word)	10^9 *P(x|w)P(w)
actress	t	-	c|ct	.000117	.0000231	2.7
cress	-	a	a|#	.00000144	.000000544	.00078
caress	ca	ac	ac|ca	.00000164	.00000170	.0028
access	c	r	r|c	.000000209	.0000916	19
across	o	e	e|o	.0000093	.000299	2.8
acres	-	s	es|e	.0000321	.0000318	1.0
acres	-	s	ss|s	.0000342	.0000318	1.0

Language model

- Noisy channel approach: multiply channel model by language model
- We want to prefer more frequent corrections
  - Simplest idea: unigram probability

Candidate Correction	Correct Letter	Error Letter	x|w	P(x|word)	P(word)	10^9 *P(x|w)P(w)
actress	t	-	c|ct	.000117	.0000231	2.7
cress	-	a	a|#	.00000144	.000000544	.00078
caress	ca	ac	ac|ca	.00000164	.00000170	.0028
access	c	r	r|c	.000000209	.0000916	19
across	**o**	**e**	**e|o**	**.0000093**	**.000299**	**2.8**
acres	-	s	es|e	.0000321	.0000318	1.0
acres	-	s	ss|s	.0000342	.0000318	1.0

Language model

- Better: **incorporating context**
  - Simplest way: bigram language model
  - Better: with linear interpolation with unigram model to avoid unseen bigrams

    $P_{li}(w_k|w_{k-1}) = \lambda.P_{uni}(w_k) + (1-\lambda).P_{mle}(w_k|w_{k-1})$

    (where $P_{mle}(w_k|w_{k-1}) = \#(w_k|w_{k-1})/\#w_{k-1}$ is called the maximum likelihood estimate)
  - Even better: the unigram model can be smoothed (e.g. add-1)
- Imagine the following context for our example:

  a stellar and versatile acress whose combination of sass and glamour…
- Counts from the Corpus of Contemporary American English with add-1 smoothing
  - P(actress|versatile)=.000021 P(whose|actress) = 0.0010
    P(across|versatile) =.000021 P(whose|across) = 0.000006
    P("versatile actress whose") = .000021 x .0010 = 210.10^{-10}
    P("versatile across whose") = .000021 x .000006 = 1.10^{-10}

Language model

- Better: **incorporating context**
  - Simplest way: bigram language model
  - Better: with linear interpolation with unigram model to avoid unseen bigrams

    $P_{li}(w_k|w_{k-1}) = \lambda.P_{uni}(w_k) + (1-\lambda).P_{mle}(w_k|w_{k-1})$

    (where $P_{mle}(w_k|w_{k-1}) = \#(w_k|w_{k-1})/\#w_{k-1}$ is called the maximum likelihood estimate)
  - Even better: the unigram model can be smoothed (e.g. add-1)
- Imagine the following context for our example:

  a stellar and versatile acress whose combination of sass and glamour…
- Counts from the Corpus of Contemporary American English with add-1 smoothing
  - P(actress|versatile)=.000021 P(whose|actress) = 0.0010
    P(across|versatile) =.000021 P(whose|across) = 0.000006
    P("versatile actress whose") = .000021 x .0010 = 210.10^{-10}
    P("versatile across whose") = .000021 x .000006 = 1.10^{-10}
  - **The best correction is "actress"!**

Real-word errors

- Much harder to detect
- Are they frequent?
  - 25-40% of spelling errors are real words (Kukich 1992)
  - More real-word errors in content produced on smartphones since the generalisation of semi-automatic correction
- General idea:
  - Guess which words *could* be errors
  - Produce candidates but **include the original word amongst them**
    - **Homophones** are important (phonetisation)
  - Chose best candidate with noisy channel model

Practical assignment 3

Goal

- Bigram models are simple… and today we have studied more sophisticated ways to represent the context

- The goal of this assignment is to develop a **normalisation system for English tweets**
  - that can change raw English tweets into (partially) normalised tweets, suitable for further NLP processing
  - using an **edit-distance-based error model** and **context2vec as a language model**

- No virtual machine, no Docker: you are on your own

Error model (formal similarity)

- You will use the notion of edit distance as computed by the Levenshtein algorithm
  - or any improvement thereof that you prefer

- You are requested to code the algorithm yourself using dynamic programming
  - you are **not** allowed to use an existing package to compute the edit distance

Language model (contextual information)

- You will use **context2vec**

- Possible, **non-mandatory extensions**
  - You can train context2vec models or use existing ones
  - You can develop a training corpus and learn whatever you want on this corpus (e.g. the way to combine contextual and formal similarity information), or you can use a fully unsupervised approach.
  - You can also develop complementary resources
    - a normalisation lexicon
    - word embeddings such as those provided by GloVe (tweet-based embeddings available), etc.
  - You can design ways to deal with one-to-many (e.g. ttyl -> talk to you later) and even many-to-many mappings, or chose to only deal with one-to-one mappings.

Assignment deliverable

- An archive containing your code and a short report
- Please respect the instructions given in the assignment description on the GitHub
  - Failure to do so will be penalised

https://github.com/edupoux/MVA_2018_SL/tree/master/TD_%233


That’s all for today!