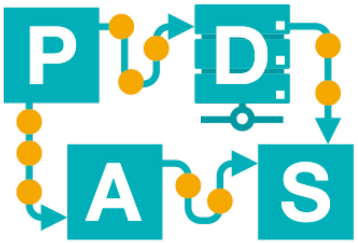
Text Mining (2/2)

Lecture 17

IDS-L17



Chair of Process and Data Science

RWTH AACHEN UNIVERSITY

Outline of Today's Lecture

- N-gram model to exploit the order of words
- Autoencoders to reduce the number of dimensions and to handle sparseness.
- word2vec to capture context.

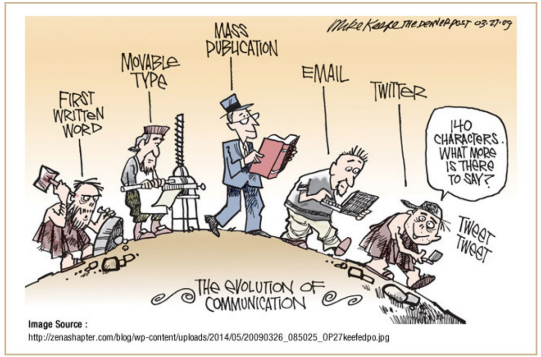


Image Source :
http://zenashapter.com/blog/wp-content/uploads/2014/05/20090326_085025_0P27keefedpo.jpg

Chair of Process and Data Science

Material (only as background information or if you want to dive deeper)

- Jurafsky, Martin, *"Speech and Language Processing"*, chapters 4 through 8: https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf
- Manning, Ragavan, Schütze, "Introduction to Information Retrieval", chapters 6, and 13 through 18: https://nlp.stanford.edu/IR-book/pdf/irbookprint.pdf

Chair of Process and Data Science

Definitions - Recap

- **Corpus**: collection of pieces of text with a consistent nature (articles, forum posts, tweets, etc)
- **Documents**: the fragments of text in a corpus
- **Annotated corpus**: corpus in which (a) the documents or (b) fractions of documents have been enriched with metadata.

Preprocessing - Recap

Necessary to preprocess text to get structured data.

1. **Dividing the text in discrete units (tokenization)**
2. **Removing irrelevant tokens (stopwords)**
   (remove ‘the’, ‘as’, ‘in’, ‘me’, ‘you’, ‘which’, ‘on’, etc.)
3. **Normalize the tokens so that the same concept is always represented by the same token**
   1. **Represent concepts with stems**
   2. **Represent concepts with lemmas**

Word	Stem	Lemma
bakery	baker	bake
bakeries	baker	bake
police	polic	police
policy	polic	policy
numerical	numer	numerical

Bag of Words - Recap

- You can use the Bag of Words (BoW) model to represent documents in a corpus.
- It is simply the bag or multiset of the tokens contained in a document of the corpus.
- Advantages:
  - Simple
  - Effective in some applications
- Disadvantages:
  - The order of the words in a document is lost
  - The representation is very sparse (one feature per word in the dictionary)

Tfidf - Recap

Tfidf weighting is a simple way to represent the relevance of a word in a document (frequent in current document, infrequent in whole corpus).

$$tf(w,d) = \#of\ occurrences\ of\ word\ w\ in\ document\ d$$

$$idf(w) = log_2\left(\frac{N}{\#of\ documents\ that\ contain\ w\ at\ least\ once}\right)$$

$$tfidf(w,d) = tf(w,d) * idf(w)$$

N-gram model

Completion Prediction

In order to better frame the importance of the order of words, let's look at another Text Mining application: completion prediction

Given a sequence of words, predict the next!

Apple gets most of the revenues selling cell _____

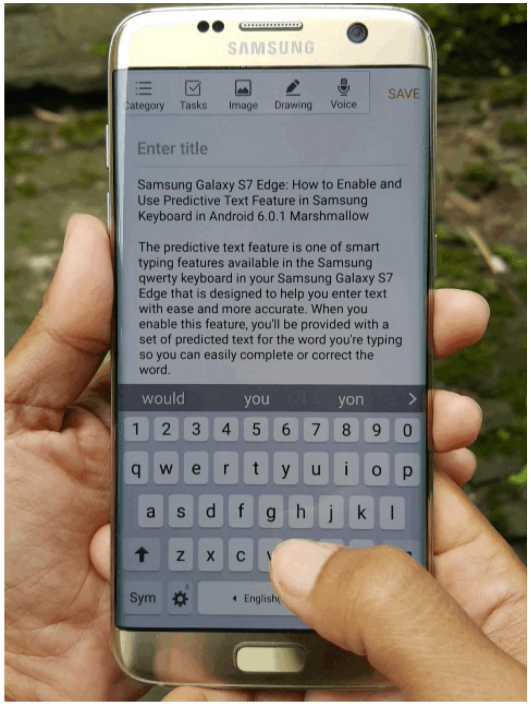
Your code crashed, it has a _____

Completion Prediction

Heavily used in smartphones!

Apple gets most of the revenues selling cell _____

Your code crashed, it has a _____



Chair of Process and Data Science

Completion Prediction

Clearly, the BoW model is not very useful here.

We need a model capable of retaining some information about the order of words.

Introducing the n-gram model

N-gram model

The idea behind n-gram models is to have sequences of consecutive tokens, instead of individual tokens.

The n in n-gram indicates the length of the sequence.

Some examples:

N-gram model

“Apples are good for you.”

Unigram model: [‘apples’, ‘are’, ‘good’, ‘for’, ‘you’]

Bigram model: [(‘apples’, ‘are’), (‘are’, ‘good’), (‘good’, ‘for’), (‘for’, ‘you’)]

Trigram model:

[(‘apples’, ‘are’, ‘good’), (‘are’, ‘good’, ‘for’), (‘good’, ‘for’, ‘you’)]

N-gram model (N=1)

“Apples are good for you.”

Unigram model: [‘apples’, ‘are’, ‘good’, ‘for’, ‘you’]

Notice that the unigram model is equivalent to BoW!

N-gram model (N=2)

Unigram model: [‘apples’, ‘are’, ‘good’, ‘for’, ‘you’]

Bigram model: [(‘apples’, ‘are’), (‘are’, ‘good’), (‘good’, ‘for’), (‘for’, ‘you’)]

N-gram model (N=3)

Unigram model: [‘apples’, ‘are’, ‘good’, ‘for’, ‘you’]

Bigram model: [(‘apples’, ‘are’), (‘are’, ‘good’), (‘good’, ‘for’), (‘for’, ‘you’)]

Trigram model:

[(‘apples’, ‘are’, ‘good’), (‘are’, ‘good’, ‘for’), (‘good’, ‘for’, ‘you’)]

Recall from Lecture 8: Basic statistics

- $P(X)$ = probability that X holds.
- $P(X, Y)$ = probability that both X and Y hold.
- $P(X|Y)$ = probability that X holds given Y **(conditional probability).**
- **Product rule (always holds):** $P(X, Y) = P(X|Y) \times P(Y)$.
- **Chain rule (always holds):** $P(A, B, \dots, Y, Z) = P(A, B, \dots, Y|Z)P(Z) = P(A|B, \dots, Z)P(B|C, \dots, Z) \dots P(Y|Z)P(Z)$.

Recall from Lecture 8: Basic statistics

- $P(\neg X) = 1 - P(X)$
- $P(\neg X|Y) = 1 - P(X|Y)$
- $P(Y) = P(Y|X)P(X) + P(Y|\neg X)P(\neg X)$
- **Assuming independence:**
  $P(A, B, \dots, Y, Z) = P(A)P(B) \dots P(Z)$.
  Warning: Only if $A, B, \dots, Y, Z$ are really independent!

Preprocessing

As mentioned in the first lecture, preprocessing steps are context- and application-dependent.

For example, while stopword removal is useful for document classification, it could be skipped for completion prediction (since it is desirable to predict stopwords).

For sake of clarity, the examples in this lecture do not use any preprocessing steps (i.e., no stopword removal and no stemming/lemmatization).

N-gram model: motivation

The motivation behind the n-grams is estimating the probability of a word given prior context:

P(phones | Apple gets most of the revenues selling cell)

We want to find an estimate of P.

N-gram model: motivation

An n-gram uses n-1 tokens for context:

- **Unigram**:
  - P(phone)
- **Bigram**:
  - P(phone | cell)
- **Trigram**:
  - P(phone | your cell)

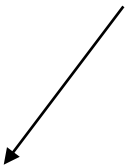
Probability that a random word in the corpus is “phone”

N-gram model: motivation

An n-gram uses n-1 tokens for context:

- **Unigram:**
  - P(phone)
- **Bigram:**
  - P(phone | cell)
- **Trigram:**
  - P(phone | your cell)

Probability that a random word in the corpus is “phone” given that the previous is “cell”



N-gram model: motivation

An n-gram uses n-1 tokens for context:

- **Unigram:**
  - P(phone)
- **Bigram:**
  - P(phone | cell)
- **Trigram:**
  - P(phone | your cell)

Probability that a random word in the corpus is “phone” given that the two previous words are “your” and “cell”

Markov assumption

When working with n-grams we typically assume some Markov assumption to be valid (i.e., limited memory).

Markov assumption: the future behavior of a system depends only on its recent (fixed length) history.

Example

2-gram
If you wait too long for the perfect moment, the perfect moment will pass you by.

3-gram
If you wait too long for the perfect moment, the perfect moment will pass you by.

4-gram
If you wait too long for the perfect moment, the perfect moment will pass you by.

5-gram
If you wait too long for the perfect moment, the perfect moment will pass you by.

6-gram
If you wait too long for the perfect moment, the perfect moment will pass you by.

Markov assumption

Markov assumption in completion prediction:

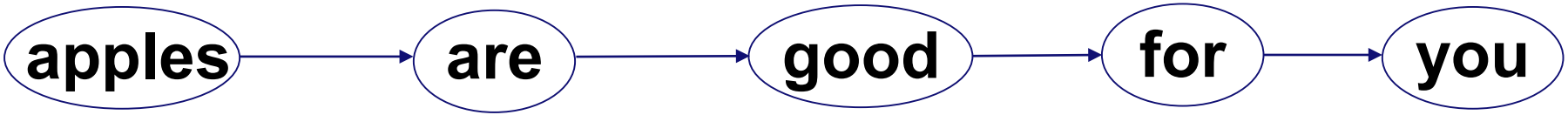
a word depends only on its (fixed length) sequence of predecessors.

A k-th order Markov Model is a model where the next state depends from the previous k.

N-grams are (N-1)th order Markov models.

Markov assumption: bigrams

"Apples are good for you."



Arrows from w_1 to w_2 mean "w_2 depends on w_1".

N-gram model

Our goal is to quantify the strength of the relationship represented by the arrows (with numbers).

Given n-grams from a corpus, we have to learn a function that quantifies the probability of a certain word w given the prior context.

N-gram model: notation

Some text-specific notation:

- **Word sequences:** $w_1^n = w_1 \ldots w_n$

- **Count function:** $C(w_1^n)$

(returns the occurrences of w_1^n in the corpus)

Notice that we are not referring to a specific position in the document or corpus, just to the length

N-gram model

- **Chain rule:**

$$P(w_1^n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2)...P(w_n \mid w_1^{n-1}) = \prod_{k=1}^{n} P(w_k \mid w_1^{k-1})$$

$w_1^n = w_1 \ldots w_n$ $\quad$ $w_1^{k-1} = w_1 \ldots w_{k-1}$ $\quad$ $w_i^j = w_i \ldots w_j$

N-gram model

Markov assumption: a certain word depends only on limited context

- **Assumption on bigrams:** $P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})$

  Last $2-1=1$ words matter

- **Assumption on N-grams** $P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1})$

  Last $N-1$ words matter

need to handle the case $n-N+1 < 1$: use default values

N-gram model

Directly from the Markov assumption:

- **Bigram approximation:** $P(w_1^n) \approx \prod_{k=1}^{n} P(w_k \mid w_{k-1})$
- **N-gram approximation:** $P(w_1^n) \approx \prod_{k=1}^{n} P(w_k \mid w_{k-N+1}^{k-1})$

Maximum Likelihood Estimation (MLE)

How to estimate these probabilities?

Maximum Likelihood Estimation: given a context, the word best explained (in the corpus) by that context will be the prediction.

In our case, we will use counts of sequences and relative frequencies to find the MLE estimate for words.

MLE in N-gram models

We can now base our MLE estimations on the relative frequencies of sequences in the corpus:

- **Bigram MLE estimation:**
  (take the most likely w_n given the last word w_{n-1})

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1} w_n)}{C(w_{n-1})}$$

- **N-gram MLE estimation:**
  (take the most likely w_n given the last $N-1$ words)

$$P(w_n \mid w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1} w_n)}{C(w_{n-N+1}^{n-1})}$$

N-grams: example 1

Let's see some bigram probabilities in a corpus of 3 sentences.

```
<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>
```

In order to have a context for bigrams, we augment every sentence with special symbols for start (<s>) and end (</s>).

N-grams: example 1

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$P(\text{I}|\text{<s>}) = \frac{2}{3} = .67$ $P(\text{Sam}|\text{<s>}) = \frac{1}{3} = .33$ $P(\text{am}|\text{I}) = \frac{2}{3} = .67$

$P(\text{</s>}|\text{Sam}) = \frac{1}{2} = 0.5$ $P(\text{Sam}|\text{am}) = \frac{1}{2} = .5$ $P(\text{do}|\text{I}) = \frac{1}{3} = .33$

N-grams: example 1

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$P(\text{I}|\text{<s>}) = \frac{2}{3} = .67$ $P(\text{Sam}|\text{<s>}) = \frac{1}{3} = .33$ $P(\text{am}|\text{I}) = \frac{2}{3} = .67$

$P(\text{</s>}|\text{Sam}) = \frac{1}{2} = 0.5$ $P(\text{Sam}|\text{am}) = \frac{1}{2} = .5$ $P(\text{do}|\text{I}) = \frac{1}{3} = .33$

N-grams: example 1

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$P(\text{I}|\text{<s>}) = \frac{2}{3} = .67$ $P(\text{Sam}|\text{<s>}) = \frac{1}{3} = .33$ $P(\text{am}|\text{I}) = \frac{2}{3} = .67$

$P(\text{</s>}|\text{Sam}) = \frac{1}{2} = 0.5$ $P(\text{Sam}|\text{am}) = \frac{1}{2} = .5$ $P(\text{do}|\text{I}) = \frac{1}{3} = .33$

N-grams: example 1

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$P(\text{I}|\text{<s>}) = \frac{2}{3} = .67$ $P(\text{Sam}|\text{<s>}) = \frac{1}{3} = .33$ $P(\text{am}|\text{I}) = \frac{2}{3} = .67$

$P(\text{</s>}|\text{Sam}) = \frac{1}{2} = 0.5$ $P(\text{Sam}|\text{am}) = \frac{1}{2} = .5$ $P(\text{do}|\text{I}) = \frac{1}{3} = .33$

N-grams: example 1

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$$P(\text{I}|\text{<s>}) = \frac{2}{3} = .67 \qquad P(\text{Sam}|\text{<s>}) = \frac{1}{3} = .33 \qquad P(\text{am}|\text{I}) = \frac{2}{3} = .67$$

$$P(\text{</s>}|\text{Sam}) = \frac{1}{2} = 0.5 \qquad P(\text{Sam}|\text{am}) = \frac{1}{2} = .5 \qquad P(\text{do}|\text{I}) = \frac{1}{3} = .33$$

N-grams: example 1

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

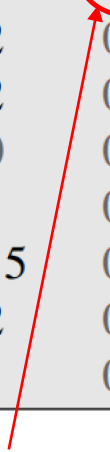
$P(\text{I}|\text{<s>}) = \frac{2}{3} = .67$ $P(\text{Sam}|\text{<s>}) = \frac{1}{3} = .33$ $P(\text{am}|\text{I}) = \frac{2}{3} = .67$

$P(\text{</s>}|\text{Sam}) = \frac{1}{2} = 0.5$ $P(\text{Sam}|\text{am}) = \frac{1}{2} = .5$ $P(\text{do}|\text{I}) = \frac{1}{3} = .33$

N-grams: example 2

Consider the following counts for bigrams (this is a small selection from a larger corpus):

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0



This indicates the number of times the word “want” follows the word “i”

N-grams: example 2

We can apply what we saw before to get a matrix of MLE probability estimations for bigrams:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

In the full matrix the rows sum to 1

each word has a probability distribution based on the vocabulary

N-grams: example 2

We can apply what we saw before to get a matrix of MLE probability estimations for bigrams:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

N-grams: example 2

How to estimate the probability of whole sentences?

Markov assumption and Chain Rule: we can just multiply together bigram probabilities

$$
\begin{aligned}
&P(\texttt{<s> i want english food </s>}) \\
&\quad = P(\texttt{i}\,|\,\texttt{<s>})P(\texttt{want}\,|\,\texttt{i})P(\texttt{english}\,|\,\texttt{want}) \\
&\qquad\qquad P(\texttt{food}\,|\,\texttt{english})P(\texttt{</s>}\,|\,\texttt{food}) \\
&\quad = .25 \times .33 \times .0011 \times 0.5 \times 0.68 \\
&\quad = = .000031
\end{aligned}
$$

$$P(w_1^n) = \prod_{k=1}^{n} P(w_k \mid w_{k-N+1}^{k-1})$$

N-grams: effectiveness

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; –It cannot be but so.

Sentences generated from different n-gram models trained on Shakespeare's bibliography (punctuation treated as words).

N-grams: effectiveness

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; –It cannot be but so.

This is quite convincing!

Sentences generated from different n-gram models trained on Shakespeare's bibliography (punctuation treated as words).

N-grams: effectiveness

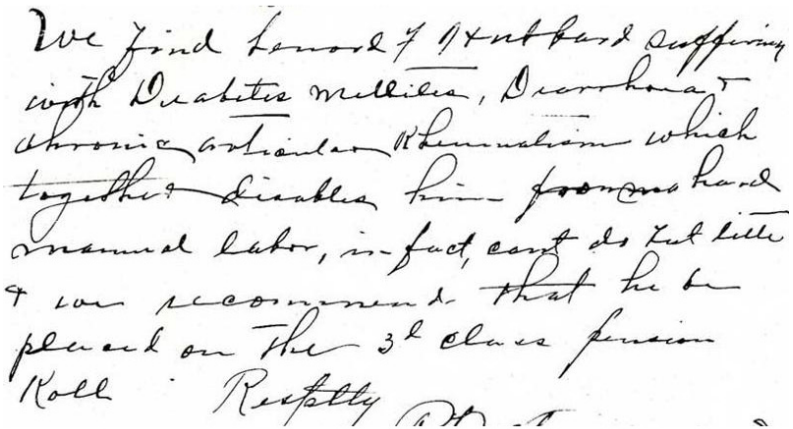
Initial fragment (bold face) and intended, missing part	Prediction
Please complete your address.	your address.
Kindly excuse the incomplete shipment.	excuse the
Our supplier notified us that the pants are undeliverable.	notified us that the
The mentioned order is not in our system.	not in our system.
We recommend that you write down your login name and password.	that you write down your login name and password.
The value will be accounted for in your invoice.	be accounted for in your invoice.
Please excuse the delay.	delay.
Please excuse our mistake.	the delay. ⬅
If this is not the case give us a short notice.	us your address and customer id. ⬅

Another example of completion prediction: n-gram based predictions of answers from a customer service office. The length of the predicted text depends on the confidence of the prediction (that is, the probability value for the sequence of words). We can see some errors in the last two sentences.

N-grams

As we have seen, n-grams are very useful in predicting the next word.

You can also use them to estimate probabilities for missing/damaged text



If an optical character recognition software fails to recognize a word, you can use n-grams to estimate the most likely match!

N-grams limitations: sparseness

Our old enemy: sparseness!

In the Shakespeare example, the (very small) corpus has 884,647 sentences using a vocabulary of 29,066 words.

- **Bigrams: $(29{,}066)^2$ = 884 million**
- **Trigrams: $(29{,}066)^3$ = 24 trillion**
- **4-grams: $(29{,}066)^4$ = 713 million billion!**

N-grams limitations: sparseness

- **Bigrams: $(29{,}066)^2$ = 884 million**
- **Trigrams: $(29{,}066)^3$ = 24 trillion**
- **4-grams: $(29{,}066)^4$ = 713 million billion!**

If you were to build a vector model based directly on n-grams, that's the number of dimensions you will have, and the number of columns that your dataset would have!

The vast majority of your dataset would be just zeroes. This is the reason why they are non-applicable (as-is) to classification problems!

N-grams limitations: sparseness

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

The sparseness is already visible in this bigram table (although we aimed to give a matrix that is as dense as possible)

N-grams on characters

For this reason, the basic version of the n-gram model is often successfully applied on characters rather than words.

The vocabulary is just the alphabet!

	4-grams
Once_upon_a_time:	Once
Once_upon_a_time:	nce_
Once_upon_a_time:	ce_u
Once_upon_a_time:	e_up

N-grams limitations: zeroes

Imagine now that we split a corpus into training and test set:

- we train an n-gram model from the training corpus
- we test it on sequences of length n-1 from the test corpus

What happens if in the test corpus there is a sequence of length n unseen in the training data?

These are called zeroes. Our model will assign probability 0 to these events!

N-grams limitations: zeroes

Recalling concepts from the previous lectures: the n-gram model “as is” tends to overfit the training data.

This is because it is very hard to generalize to unseen combinations of words.

N-grams: smoothing

Solution: smoothing

We “chip off” some probability from highly-likely sequences and we spread it among unseen sequences, assigning a small but non-zero probability.

N-grams: smoothing

A very simple smoothing: Laplace smoothing

Add +1 to all cells of the N-gram matrix before calculating probabilities!

Laplace smoothing for bigrams:

$$P^*_{\text{Laplace}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

In order to have probabilities that sum to 1, we need to normalize the denominator. V is the size of the vocabulary

N-grams: smoothing

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

The same bigram matrix as before, all the counts are increased by 1.

N-grams: smoothing

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Probabilities for bigrams after Laplace smoothing. A bigram model built on these probabilities can generalize to unseen sequences.

N-grams: Unknowns <UNK>

Another limitation: what if the test corpus contains words that did not occur at all in the training corpus?

These are called Out Of Vocabulary words (OOV), or unknowns.
Obviously the model cannot predict them!

We can model such unknown words using a specific pseudo-word: <UNK>

N-grams: Unknowns <UNK>

First possible (partial) solution: If we can build a vocabulary in advance, I can perform my analysis based on that.

- **Choose/build a vocabulary V**
- **For each word w in the training corpus**
  - **If w ∉ V, replace w by <UNK>**
- **Estimate the probabilities as usual, treating <UNK> as a word**

N-grams: resources

Since n-grams are not tied to copyright law, there are many n-gram databases online.

Many let you download a corpus (often a very large sample) in order to run experiments!

COCA corpus Corpus of Contemporary American English

COCA corpus: n-grams from up to 430 million words (US English)

N-grams data

introduction samples (COCA) other datasets related sites get data

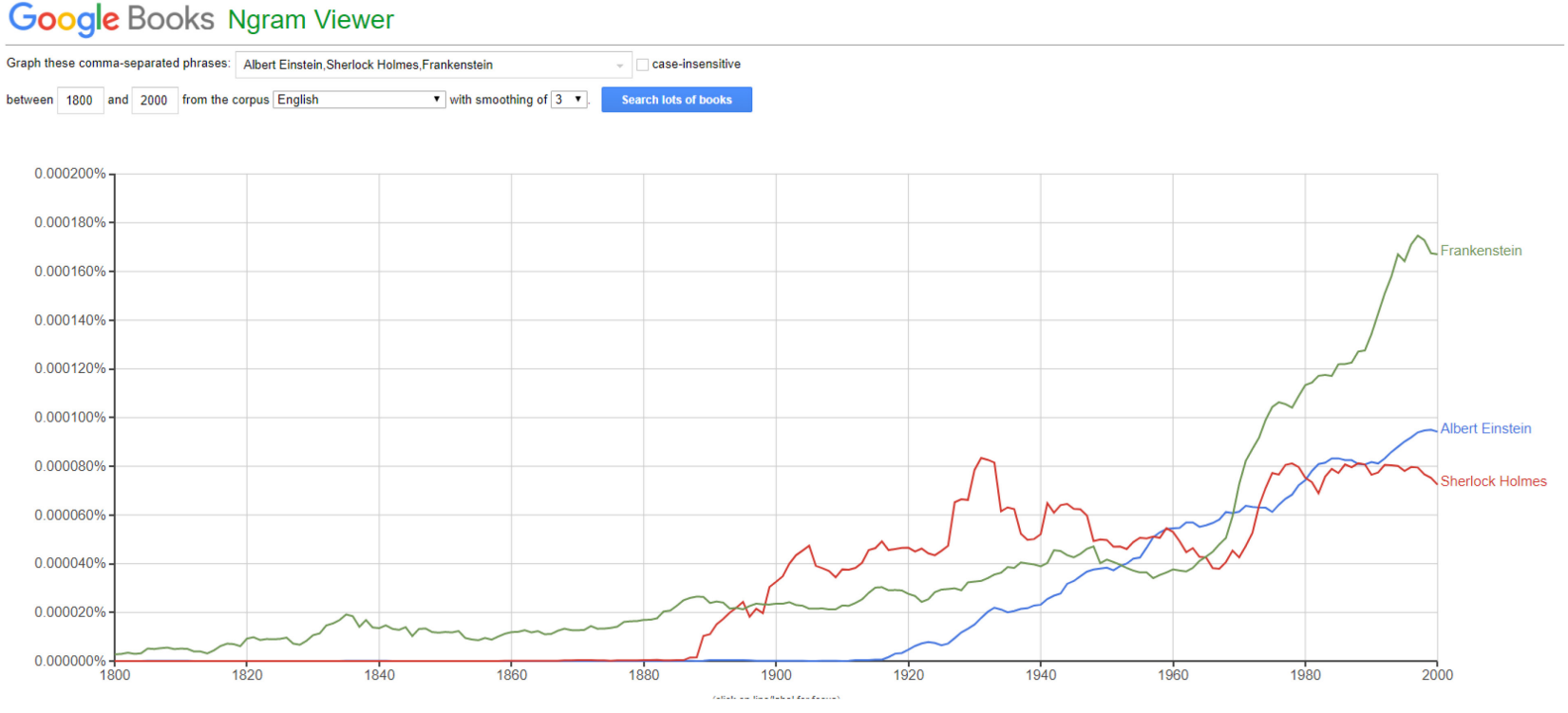
SAMPLES: LEVEL 2

You can purchase n-grams sets that contain all 1, 2, 3, and 4-grams that occur at least three times in the Corpus of Contemporary American English (when it was about 430 million words in size; it continues to grow each year). Although you probably only need a subset of these files, all 21 files are included in the purchase price. The samples files that are available from this page include all entries for words beginning with the letter [u]. (See note 1)

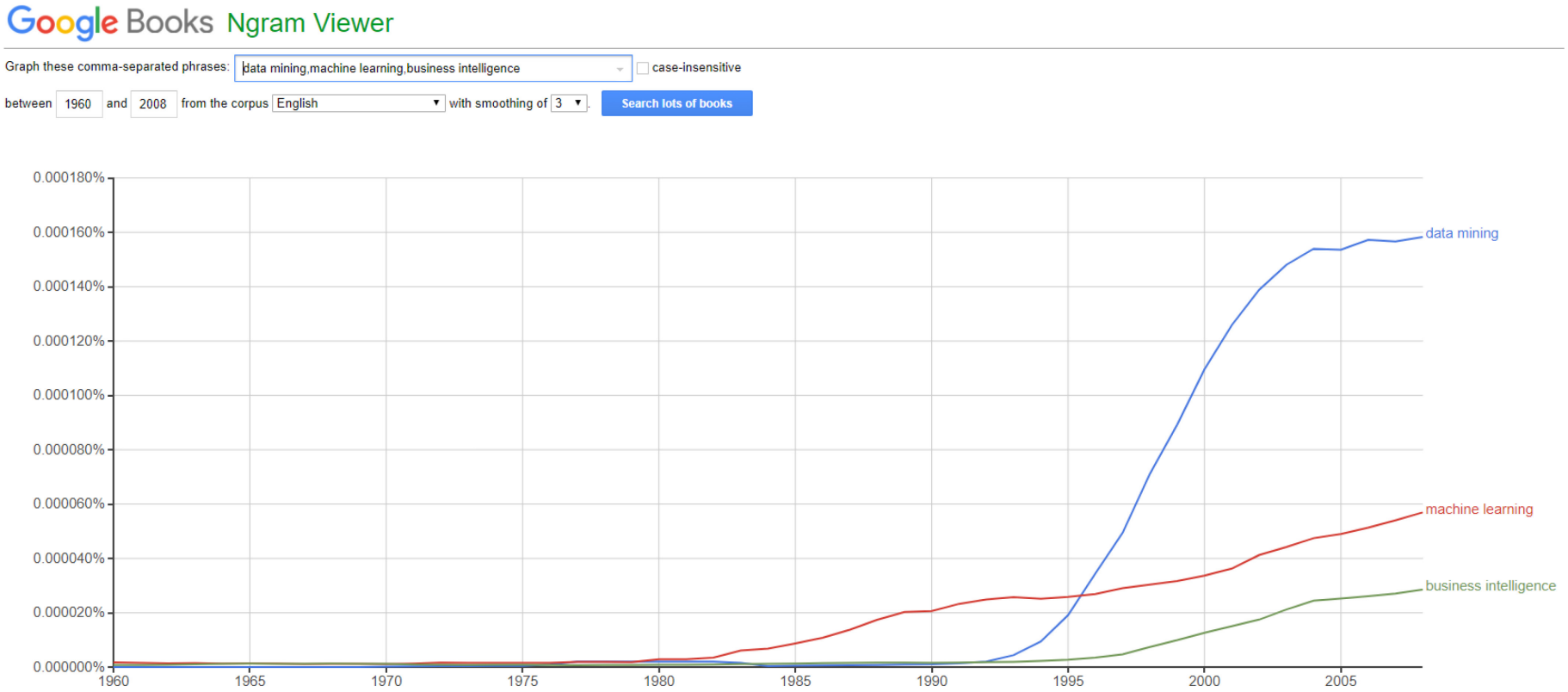
For the 2-grams and 3-grams, you have a choice of n-grams with or without part of speech (i.e. 2 options), and either case sensitive or case insensitive (i.e. 2 options), as well as n-grams that are all words, or n-grams containing at least one punctuation or number (i.e. 2 options). In other words, there are eight options total for both the 2-grams and 3-grams. For the 4-grams, there is just one file available (case sensitive, with part of speech, and no punctuation) and for the 1-grams there are no entries for punctuation.

Google Books n-gram Viewer

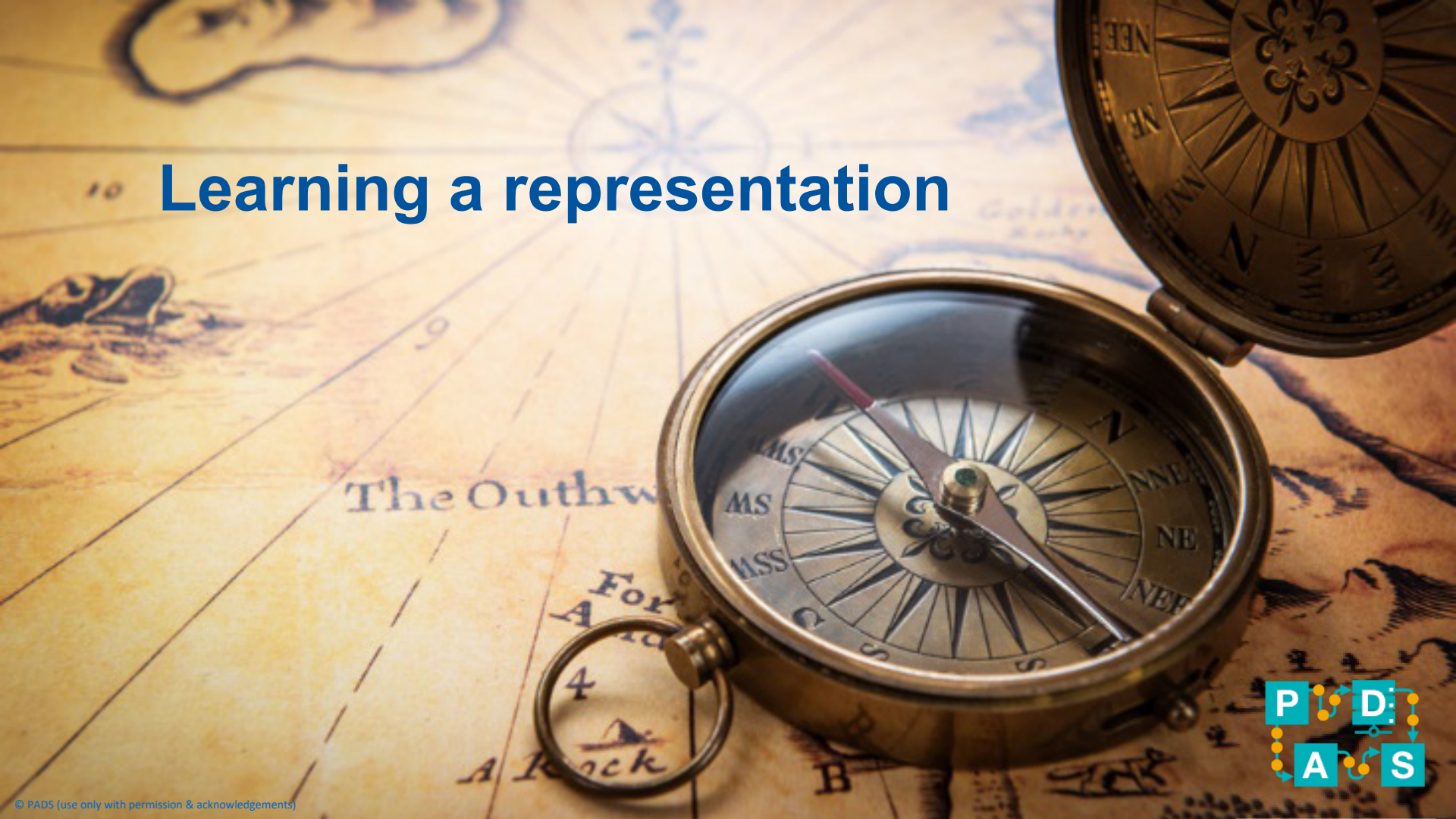
Google lets you access to datasets of n-grams from the Google Book project. There's also an interactive online viewer:



Google Books n-gram Viewer



Learning a representation



Sparseness

The problem of sparse data remains.

A static representation of words will always be some function in terms of the length of the dictionary

Sparseness

This is because words behave like categorical data.

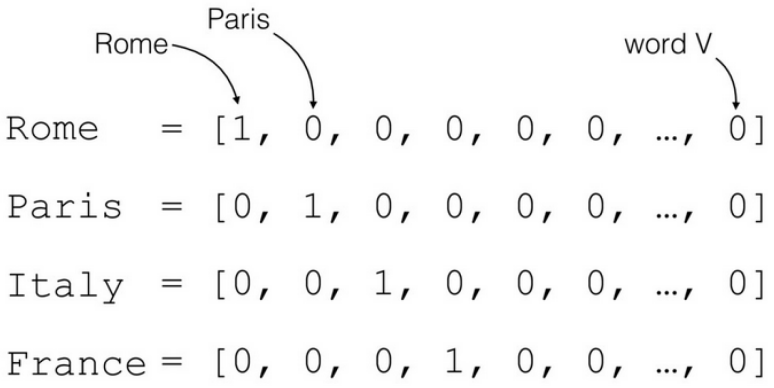
Categorical data cannot be represented by integers: it needs a one-hot representation:

- **A vector of length equal to the number of possible values**
- **One cell in the vector will be equal to 1, the others equal to 0**

Sparseness

Usually this is not a problem, because the number of possible values is limited.

In the case of Text Mining, the number of possible values is the size of the vocabulary.



Learning a representation

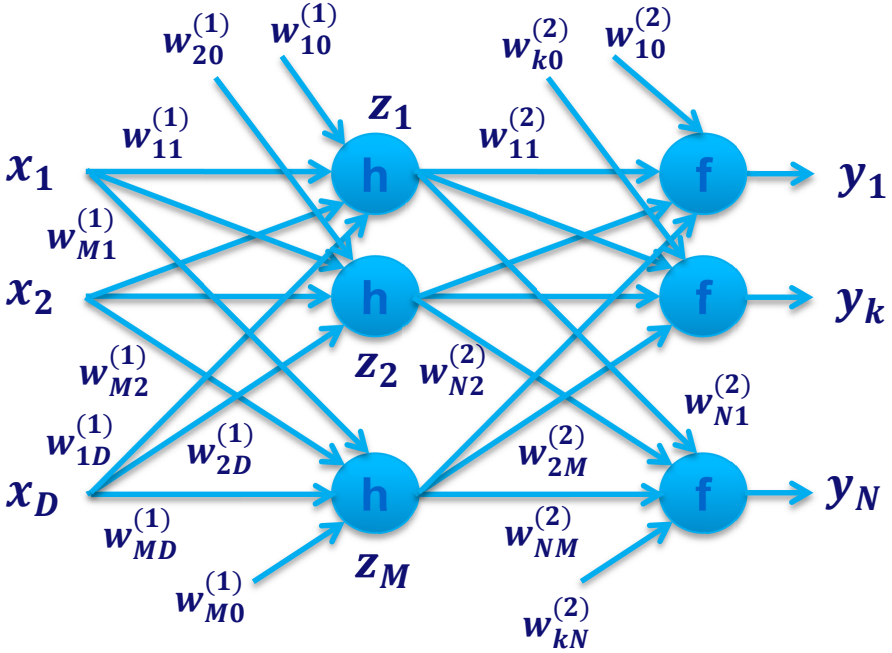
Idea: using a Machine Learning algorithm to automatically learn a representation of data

We will see an example using Neural Networks

Sample Neural Network Model

Recall from Lecture 7: Two layer neural network

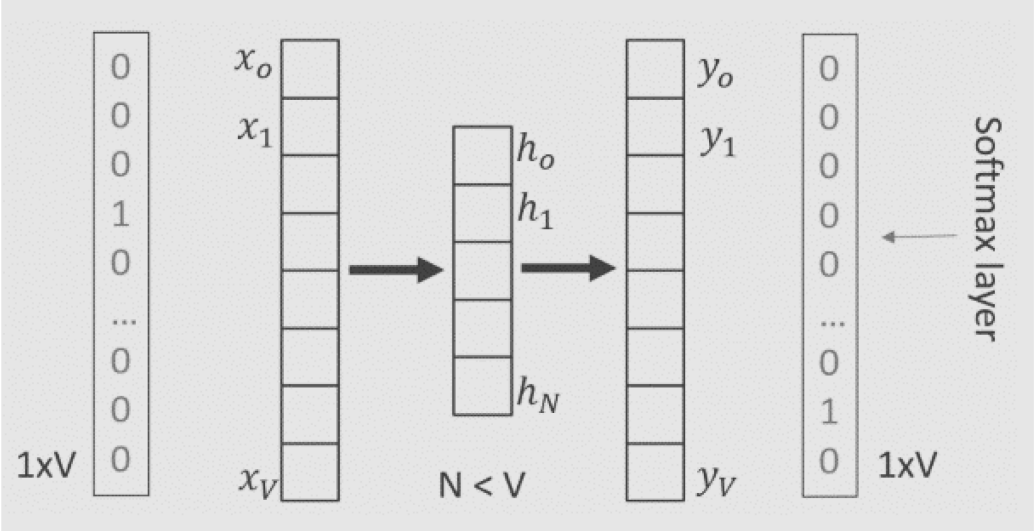
$$y_k(\boldsymbol{x},\boldsymbol{w}) = f\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$



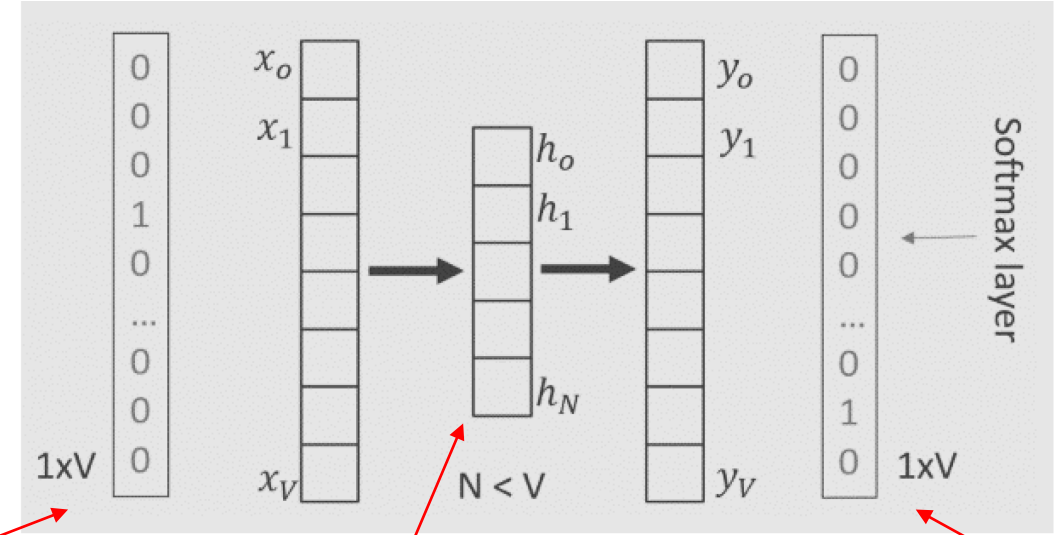
Autoencoders

Imagine a Neural Network built like this:

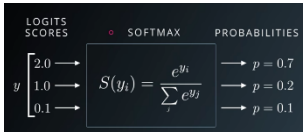
- **Input and output layers of dimension V**
- **Hidden layer of dimension N, with N < V**



Autoencoders



Softmax: probabilistic interpretation of the output



Input vector: size of the vocabulary V

Hidden layer: size N < V

Output layer: size of the vocabulary V

Autoencoders

We will trick this network into becoming a compression algorithm for text.

Let's train the network with the same data in input and output!

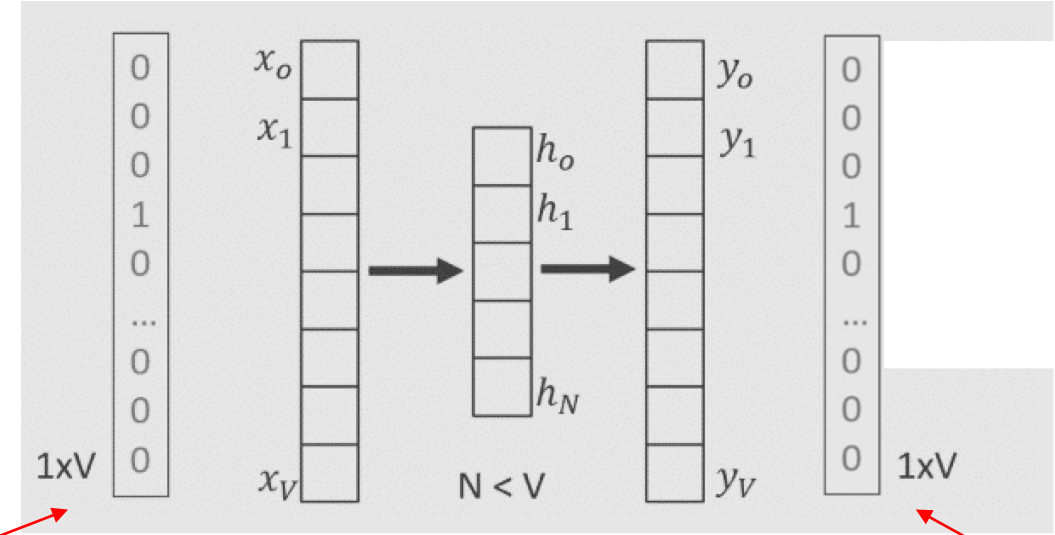
- **Convert my corpora in one-hot encoded vectors**
- **Feed one by one these vectors into the V inputs of my NN**
- **Perform backpropagation comparing the output with the input**

Autoencoders

At convergence, we will have a network that outputs (almost) the same one hot encoded vector given as input.

What happens to the hidden layer?

Autoencoders

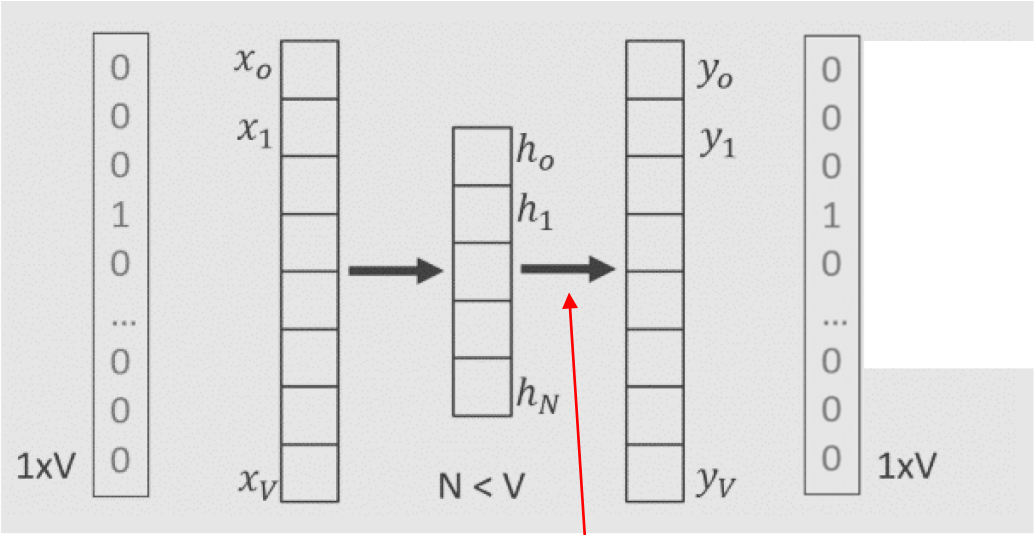


Input vector: a word w

Input layer: V neurons
(one-hot encoding of the vocabulary)

The same word w is used as target

Autoencoders



Output of the hidden layer: compressed (N-dimensional) representation of w

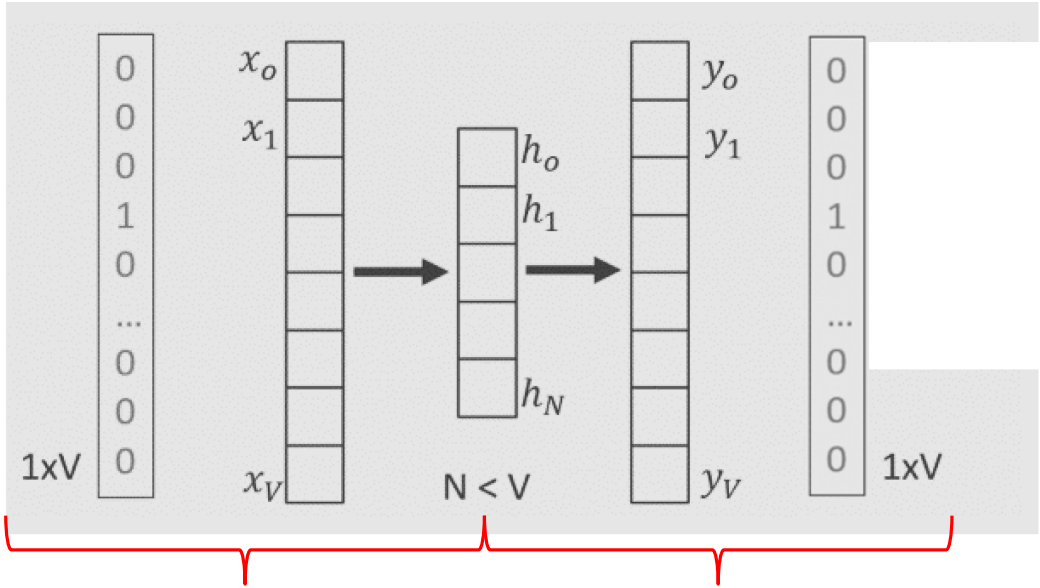
Autoencoders

The output of the hidden layer gives us a compressed representation of a certain word. The compression ratio is V/N.

These neural networks are called autoencoders, and are an example of how to automatically learn a representation of data.

It is possible to split an autoencoder in the encoding and decoding part after the training:

Autoencoders



Encoder:
from V dimensions to N

Decoder:
from N dimensions to V

Learning a representation

Autoencoders are an example of how it is possible to learn new data representations automatically. Learning a (smaller) representation of data is often called **embedding**.

This a very advanced topic, and also would deserve a whole course!

When applied to text, it is referred to as **word embeddings**.

word2vec

Word embeddings

Compression is in general very useful, but let's take a step forward and also incorporate context.

N-grams allow us to consider the order of words as well as context.

Can we obtain a word embedding that contain order and context?

For that, we are going to need an n-gram generalization:

skip-grams.

Skip-grams (k-skip n-grams)

A Skip-gram is an n-gram where it is allowed to "skip" some words. k-skip n-grams are sequences of n words that are at a maximum distance of k positions.

"Hi Jen did you eat the cake?"

1-skip 3-grams: [‘Hi Jen did’, ‘Hi Jen you’, ‘Jen did you’, ‘Jen did eat’, ‘Hi did you’, ‘did you eat’, ‘did you the’, you eat cake’, …]

2-skip 3-grams: [‘Hi Jen eat’, ‘Hi you cake’, ‘Jen you the’, …]

(regular n-grams can be considered 0-skip n-grams)

Skip-grams

Skip-grams are useful because they can associate a more general notion of context than n-grams.

Part of the surrounding context is left out (skipped): this means less overfitting when used in learning.

Let's use skip-grams in another word embedding technique:

word2vec

word2vec

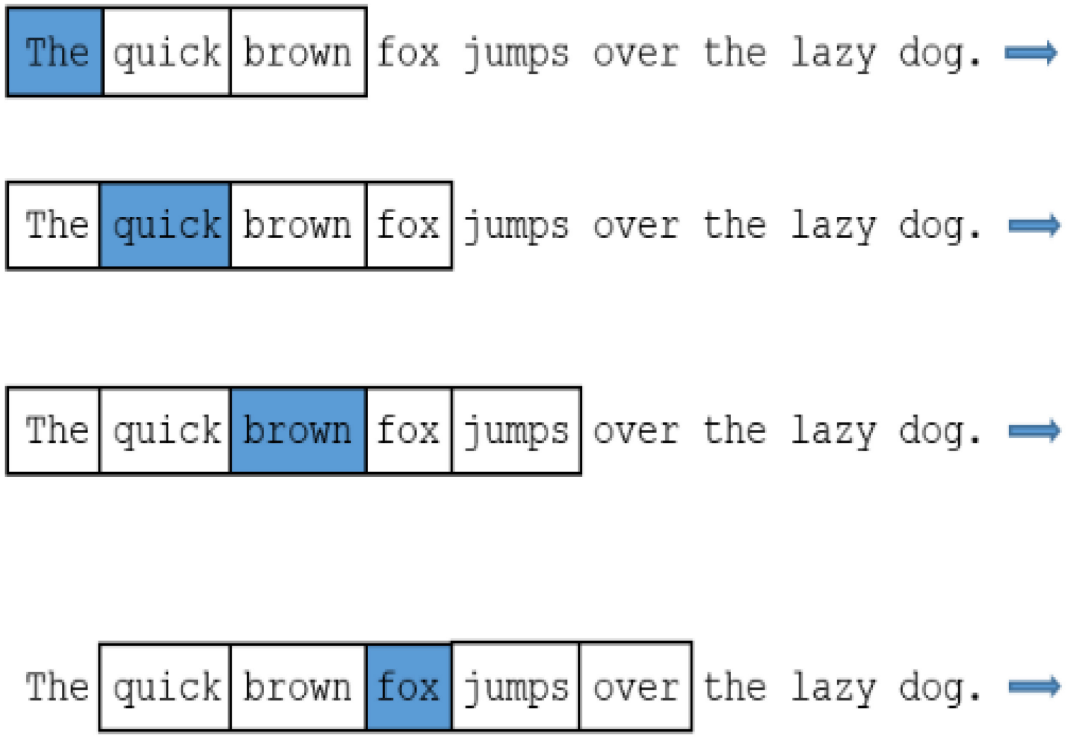
The idea behind word2vec can be seen as an extension of the autoencoder tactic: instead of learning a compressed representation of a word from itself, we will learn it from its context.

In the first step, we build a training set extracting skip-grams from text:

word2vec

For each word, we consider the context in a sliding window around it. We create 2-tuples with the word and every skip-gram in the window that does not contain it.

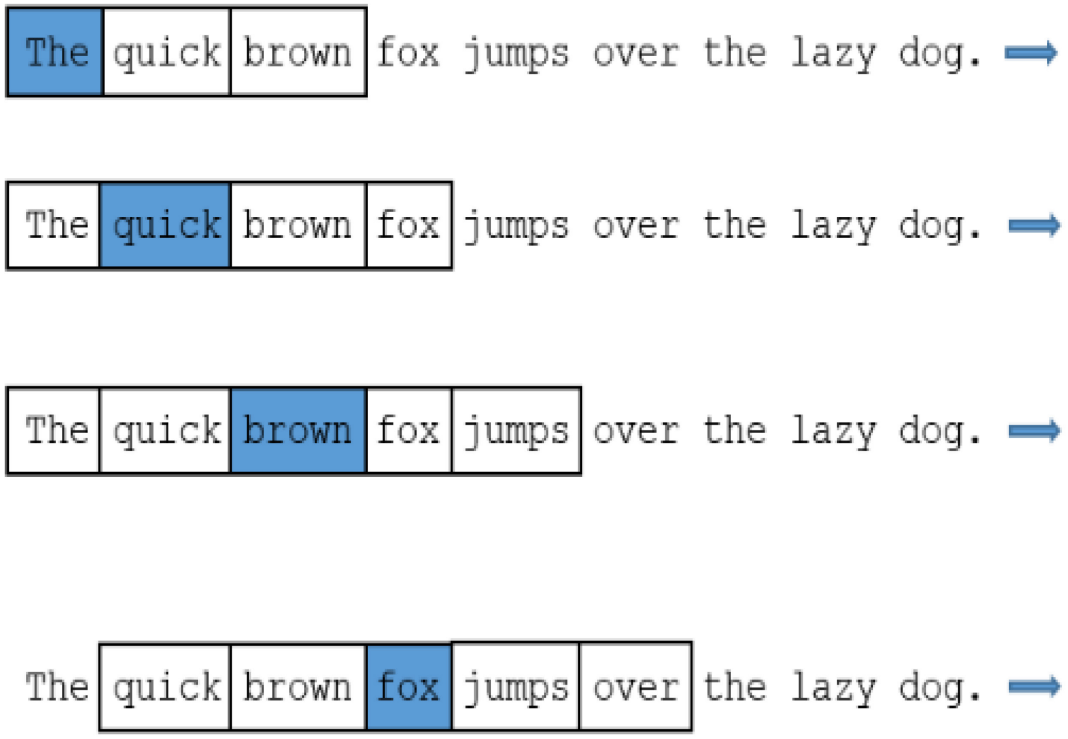
Example with 2-skip 3-grams and a window of 2 one each side.



word2vec

For each word, we consider the context in a sliding window around it. We create 2-tuples with the word and every skip-gram in the window that does not contain it.

Example with 2-skip 3-grams and a window of 2 one each side.



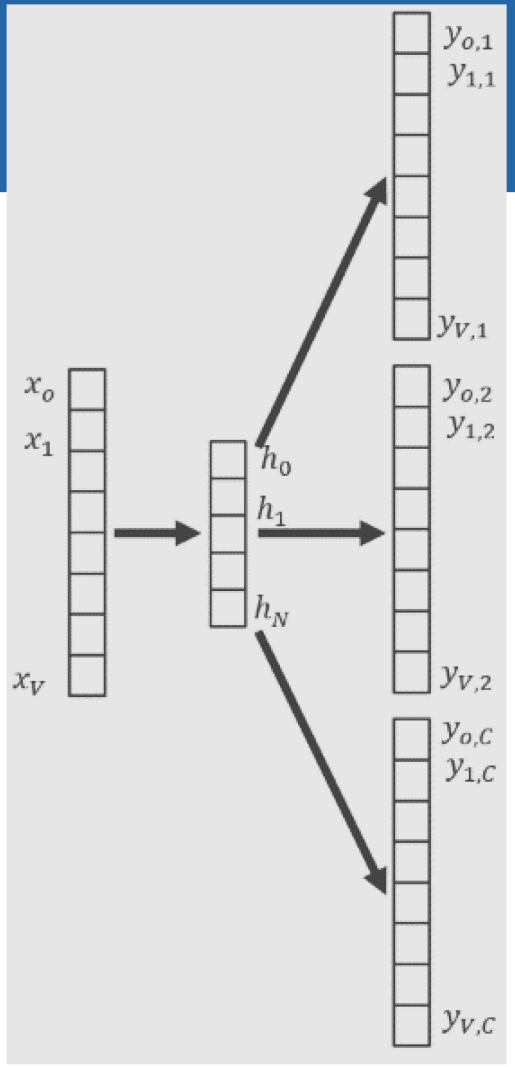
Notice that the order matters!

word2vec

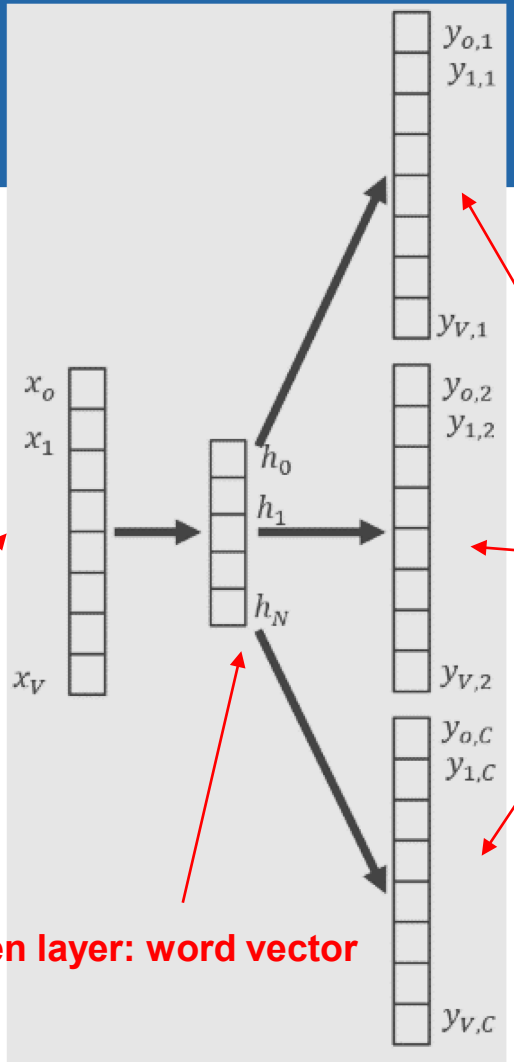
Then, we train a neural network with a certain word as input and the context as output, using the tuples we built as training set.

This is what the network looks like for our training set

(couples word-sequence of length 3):



word2vec

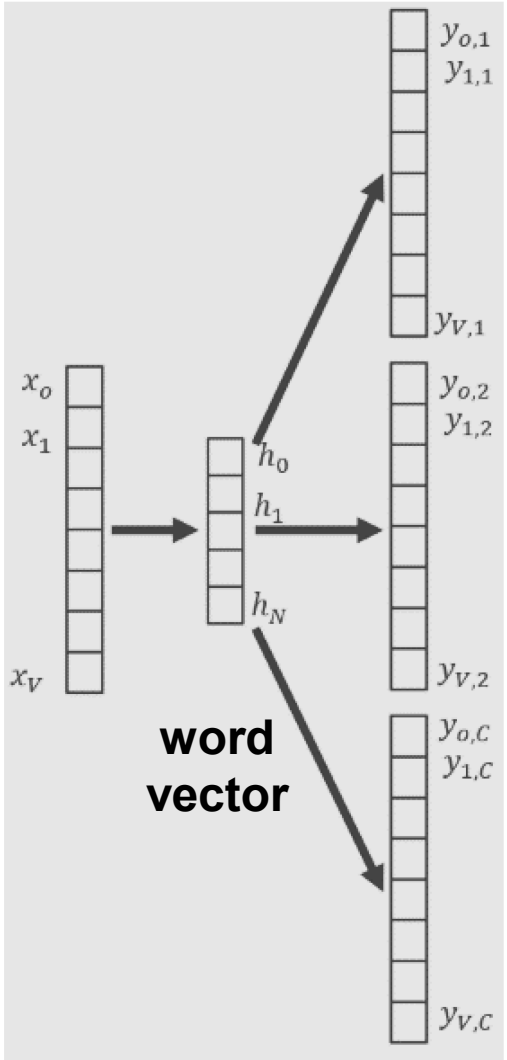


Target: context of the word w (skip-gram)

Input word w (one-hot encoded) as training data

Hidden layer: word vector

(brown, [the, quick, fox])
(brown, [the, quick, jumps])
(brown, [the, fox, jumps])
(brown, [quick, fox, jumps])



brown
brown
brown
brown

the
the
the
quick
quick
quick
fox
fox
fox
jumps
jumps
jumps

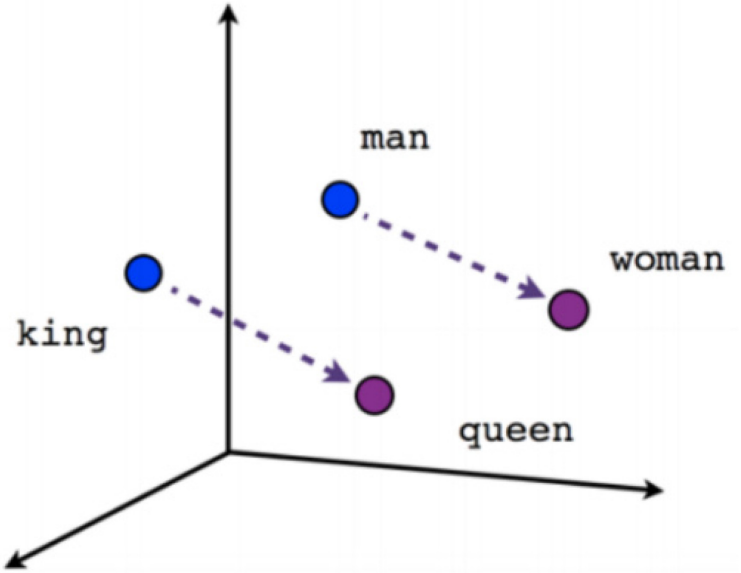
fake training task (goal is to find low-dimensional word vectors)

word2vec

This way, not only the representation of a word is compressed, but it self-contains part of the context.

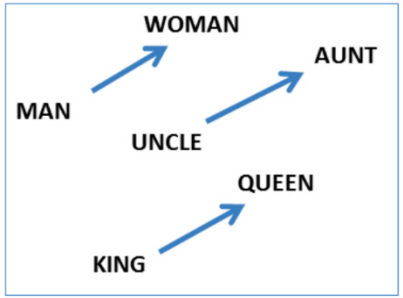
In terms of the final result, this gives us a powerful and more meaningful representation.

- **Vectors for similar words are close to eachother.**
- **It can be possible to add and remove context with vector operations!**

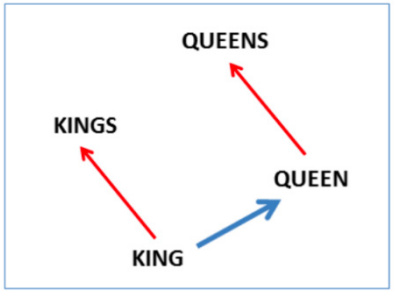


(king – man + woman) = queen

Semantics hidden in word vectors



gender



plural

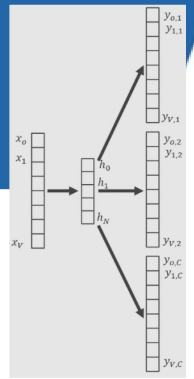


Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

What about documents?

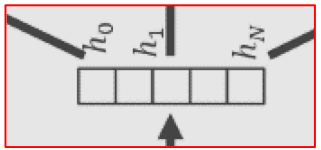
The word2vec algorithm finds a suitable representation for words, but what about documents?

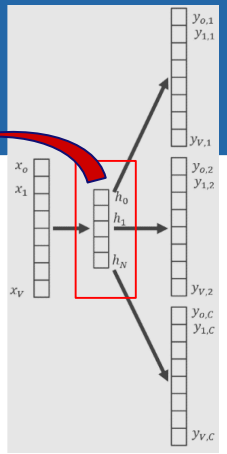
A simple method, good enough for some applications, is to represent a vector as concatenation of element-wise average, minimum and maximum of the word vectors. This carries enough information about the words to perform classification tasks.

What about documents?

Word vectors

The	0	3	1	6
brown	2	2	5	1
fox	4	2	0	0
...	...	...	...	...





Document vector

"The brown fox"	2	2.33333	2	2.33333	0	2	0	0	4	3	5	6

Element-wise average

Element-wise minimum

Element-wise maximum

doc2vec

We can do better!

Why not learning a representation for documents?

This is called doc2vec. It relies on the same principles of word2vec, but this time, we will one-hot encoded documents, and create a training set of 2-tuples (one-hot encoded document, skip-gram from that document).

doc2vec (with 2-skip 3-grams)

0001 ‘Cats are the only pet of the felines family, while dogs are canids.’

(0001, [<s>, cats, are])
(0001, [cats, are, the])
(0001, [cats, are, only])
...

0010 ‘Cats are the third-most popular pet in the US.’

(0010, [<s>, cats, are])
(0010, [cats, are, the])
(0010, [cats, are, third])
...

0100 ‘Dogs have been selected for millennia as pet animals.’

(0100, [<s>, dogs, have])
(0100, [dogs, have, been])
(0100, [dogs, have, selected])
...

1000 ‘Normally, dogs are not aggressive towards other dogs outside their territory.’

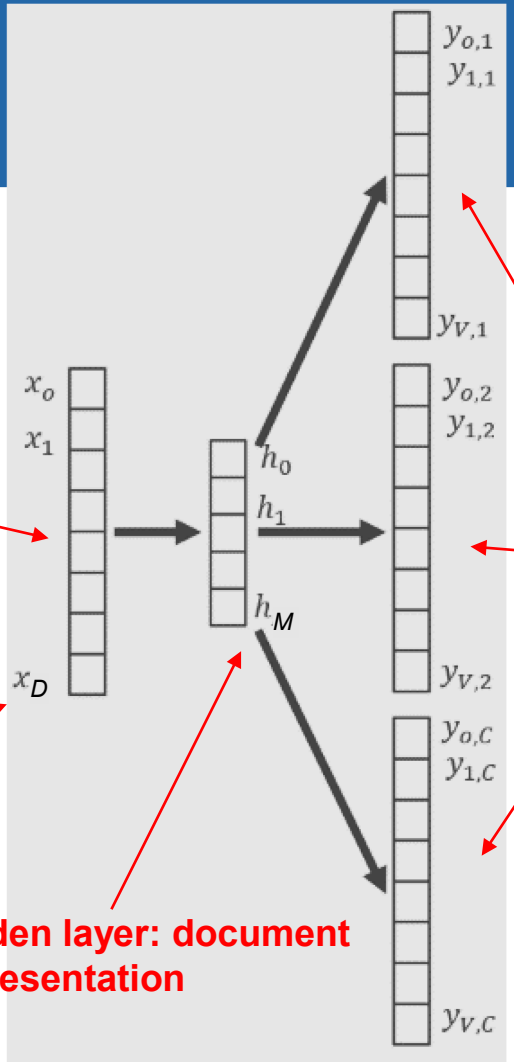
(1000, [<s>, normally, dogs])
(1000, [normally, dogs, are])
(1000, [normally, dogs, not])
...

doc2vec

Then, we can simply learn a representation for the document by training a neural network with the same architecture seen for word2vec.

This time, the input will be the one-hot encoding of documents, the output is a skip-gram contained in that document.

doc2vec



Input document d (one-hot encoded) as training data

Target: skip-gram contained in document d (one-hot encoded)

D: size of corpus

Hidden layer: document representation

doc2vec

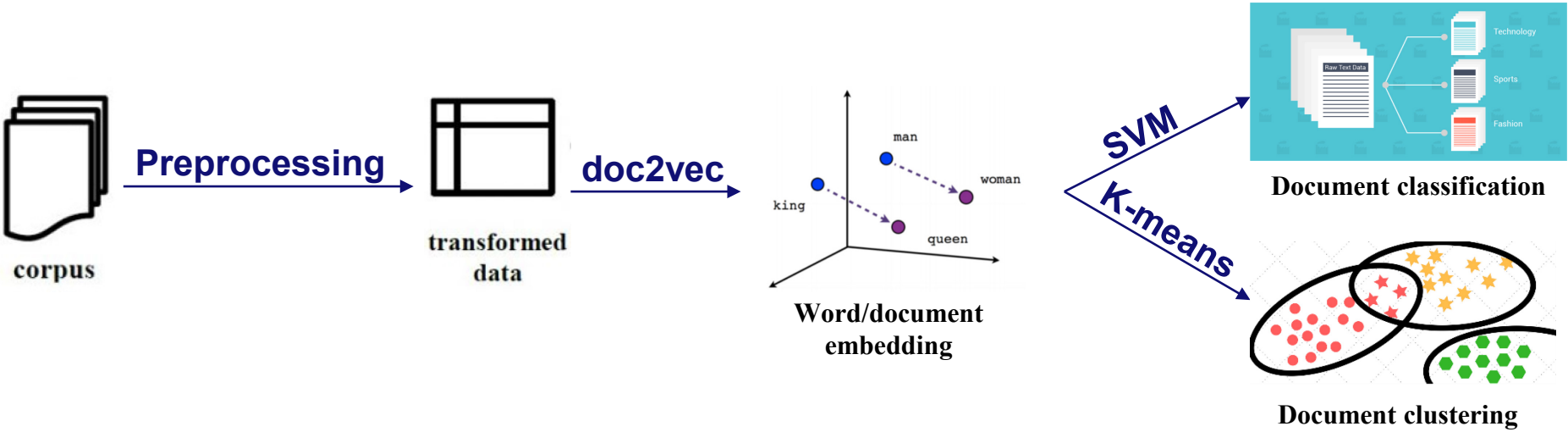
Doc2vec allows you to find a low-dimensional, expressive, powerful and context-aware vector representation for documents in the corpus.

This representation solves the problems we have seen in these two lectures:

- Keeping track of the order of words.
- Avoid the problem of the high sparseness of text data.
- Gives a fixed length representation of documents suitable as input for other Data Science applications.

Text Mining Pipeline with doc2vec

Applying doc2vec, an example of pipeline for text is now this:



Conclusion

Short summary of lecture

- Text can be statistically modeled using n-grams.
- It is possible to use neural networks to learn a new low dimensional representation for the high-dimensional data.
- The word/document embeddings provided by word2vec and doc2vec allow to translate text into semantically rich fixed-length vectors (taking into account context).

#	Lecture	date	day
Lecture 1	Introduction	10/10/2018	Wednesday
Lecture 2	Crash Course in Python	11/10/2018	Thursday
Instruction 1	*Python*	*12/10/2018*	*Friday*
Lecture 3	Basic data visualisation/exploration	17/10/2018	Wednesday
Lecture 4	Decision trees	18/10/2018	Thursday
Instruction 2	*Decision trees and data visualization/exploration*	*19/10/2018*	*Friday*
Lecture 5	Regression	24/10/2018	Wednesday
Lecture			
Instruction 3			
Lecture			
Instruction 4			
Lecture			
Lecture			
Instruction 5			
Lecture 1			
Lecture 1			
Lecture 1			
Lecture 13	Sequence mining	22/11/2018	Thursday
Instruction 6	*Clustering, frequent items sets, association rules*	*23/11/2018*	*Friday*
Lecture 14	Process mining (unsupervised)	28/11/2018	Wednesday
Lecture 15	Process mining (supervised)	29/11/2018	Thursday
Instruction 7	*Process mining and sequence mining*	*30/11/2018*	*Friday*
Lecture 16	Text mining (1/2)	05/12/2018	Wednesday
Instruction 8	*Text mining and process mining*	*06/12/2018*	*Thursday !!*
Lecture 17	Text mining (2/2)	12/12/2018	Wednesday
Lecture 18	Data preprocessing, data quality, binning, etc.	13/12/2018	Thursday
Lecture 19	Visual analytics & information visualization	19/12/2018	Wednesday
backup		20/12/2018	Thursday
Instruction 9	*Text mining, preprocessing and visualization*	*21/12/2018*	*Friday*
Lecture 20	Responsible data science (1/2)	09/01/2019	Wednesday
Lecture 21	Responsible data science (2/2)	10/01/2019	Thursday
Instruction 10	*Responsible data science*	*11/01/2019*	*Friday*
Lecture 22	Big data (1/2)	16/01/2019	Wednesday
Lecture 23	Big data (2/2)	17/01/2019	Thursday
Instruction 11	*Big data*	*18/01/2019*	*Friday*
Lecture 24	Closing	23/01/2019	Wednesday
backup		24/01/2019	Thursday
Instruction 12	*Example exam questions*	*25/01/2018*	*Friday*
backup		30/01/2019	Wednesday
backup		31/01/2019	Thursday
extra	*Question hour*	*01/02/2019*	*Friday*

Lecture 17	Text mining (2/2)	12/12/2018	Wednesday
Lecture 18	Data preprocessing, data quality, binning, etc.	13/12/2018	Thursday
Lecture 19	Visual analytics & information visualization	19/12/2018	Wednesday
backup		20/12/2018	Thursday
Instruction 9	*Text mining, preprocessing and visualization*	*21/12/2018*	*Friday*