UNIVERSITÀ DI PARMA

DIPARTIMENTO DI INGEGNERIA E ARCHITETTURA

Natural Language Processing (NLP)

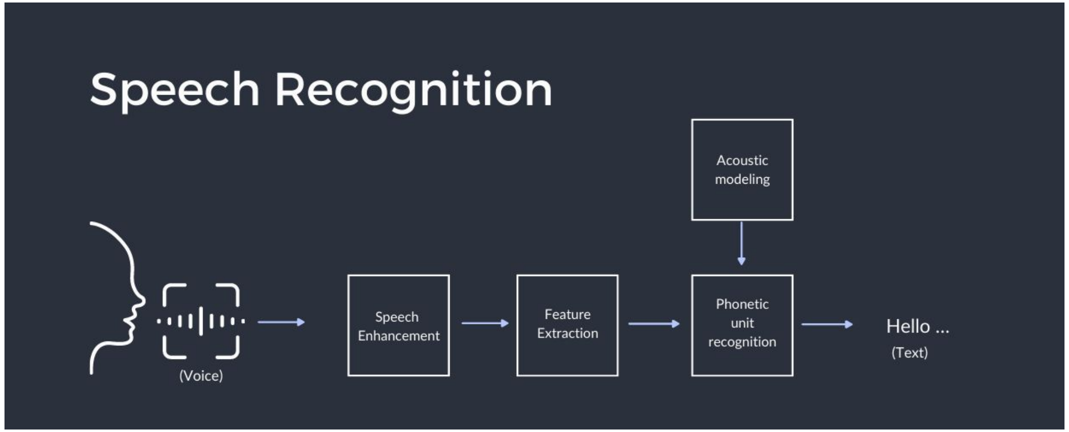
Gianfranco Lombardo, Ph.D
gianfranco.lombardo@unipr.it

What’s NLP ?

- Natural Language Processing represents the intersection among Linguistics and Computer Science (and in particular the subfield of Artificial Intelligence)
    - This term refers to every technique that enables to understand and manipulate large amount of documents, texts and words by a computer software
    - In general the main goal is to improve and develop the human-machine communication
        - However NLP is also used to automatically analyze large amount of documents for Information retrieval especially nowadays with the advent of many internet services and platforms

NLP common tasks

- **Speech Recognition**: Given a sound clip of someone speaking, determine the textual representation of the speech.
  - It requires skills on signal processing (speech), acoustic modeling, Machine Learning for sequence modeling (Like Recurrent Neural Networks), and then NLP skills



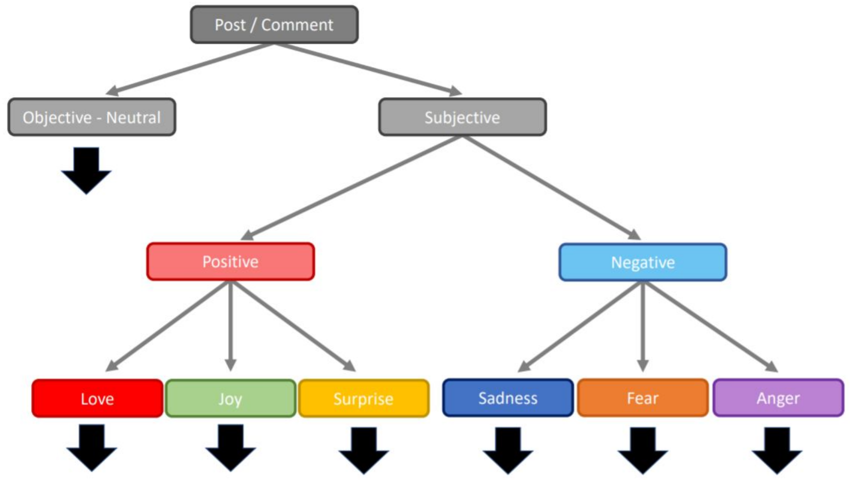
NLP common tasks

- **Sentiment analysis**: Extract subjective information usually from a set of documents, often using online reviews to determine "polarity" about specific objects
  - It is usually a classification task where we assign one label to each sentence: Usually labels are {Positive,Negative} with sometimes an additional label for the Neutral sentiment

- A variant is called Emotion Detection where usually psychological models are used to define this task as a multi-label task with several positive emotions and several negative emotions

Example: Hierarchical classifier for sentiment analysis on Facebook

- One way to model Sentiment analysis is using a Hierarchical classifier that recognize emotions using smaller classifier
  - Usually a Hierarchical classifier is better than a multi-label flat classifier
- Our goal was to train an emotion detection system for the Italian language to study a patients group on Facebook



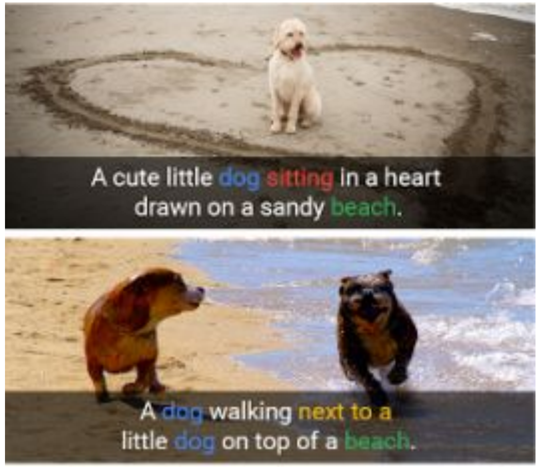
NLP common tasks

- **Topic segmentation and recognition**: Given a chunk of text, separate it into segments each of which is devoted to a topic, and identify the topic of the segment or assign a topic to an entire document

  - Usually it is modeled as a classification task
    - Using the entire text to build a “sample” for the ML model
    - Topic modeling: Extracting the most important words (features) and assign the topic using this words (Information retrieval skills)

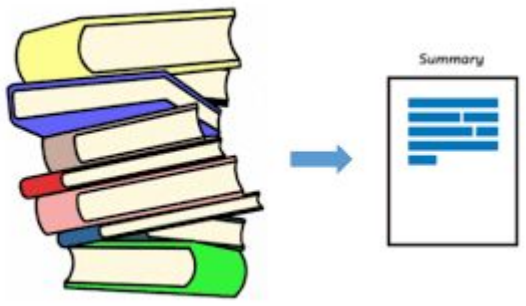
NLP common tasks

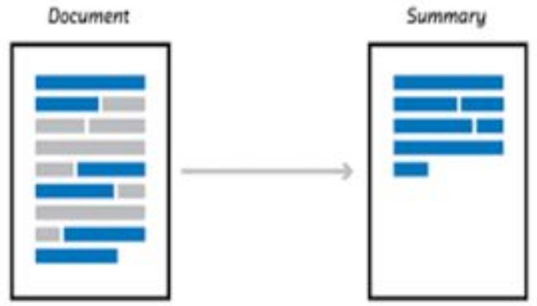
- **Image captioning**: Given an image we want to generate a text that describes the image content
  - Generative model: We do not aim to "predict" a label or a value but we want to generate something new
  - It requires skills on both Computer Vision and NLP


A cute little dog sitting in a heart drawn on a sandy beach.
A dog walking next to a little dog on top of a beach.

NLP common tasks

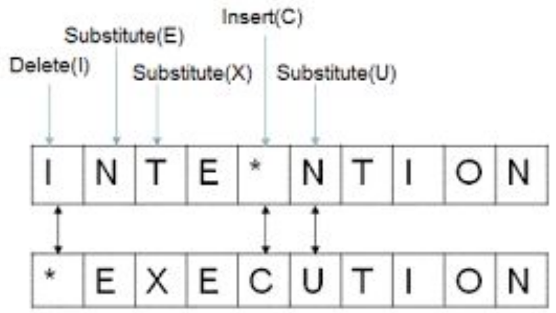
- **Text/Document summarization:** Given a document we aim to generate a shorter version of that document minimizing the information loss and maximizing the similarity between the original document and the new one.
  - It can be modeled as a subset of most meaningful sentences extraction or with a generative approach

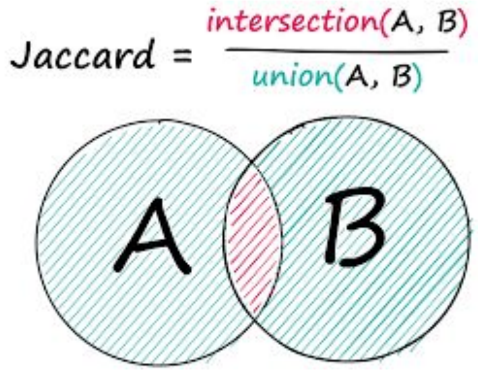




NLP common tasks

- **Document Similarity:** We want to recognize how similar are two or more documents
    - Information retrieval techniques are used to encode documents in a form that enables a similarity measure
        - **Jaccard Similarity**: Ratio of common words among the two documents and all the union of the two documents' words
        - **Edit distance**: How similar two strings are based on the number of edits (Insertions,deletions, substitutions) it takes to change one string into the other

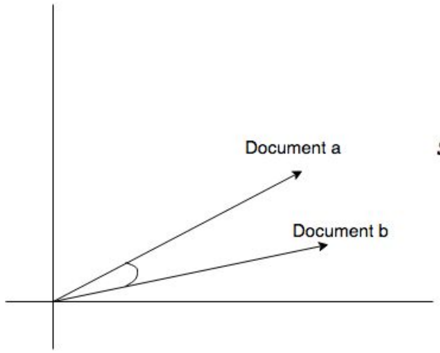




NLP common tasks

- **Document Similarity:** We want to recognize how similar are two or more documents
    - Neural networks can be used to perform ***Representation Learning*** of each document and their words to get ***word features vectors*** or document vectors: It is possible to compute the ***Cosine Similarity***.
    - Uncommon but possible: Train a model to perform a regression on a similarity measure

Cosine Similarity



$$sim(a,b) = cos\theta = \frac{\vec{a}.\vec{b}}{\| \vec{a} \| \| \vec{b} \|}$$

NLP Tasks: Part-of-Speech Tagging (PoS Tagging)

- PoS Tagging is a general task that aims to classify (or tag) each word in a text as corresponding to a particular part of speech. For Indo-European languages we distinguish 8 different elements:

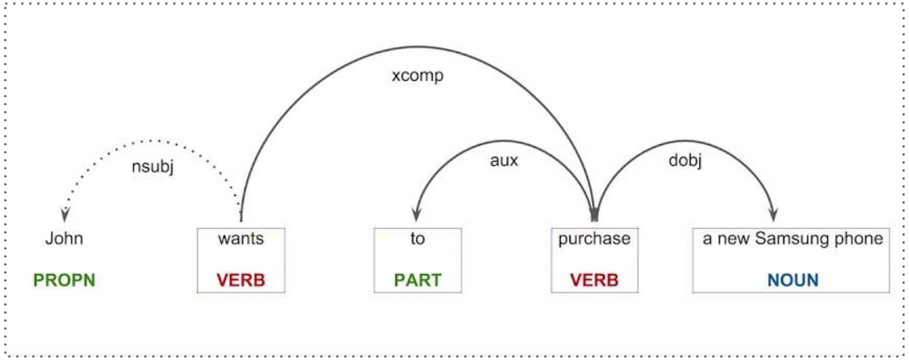
N -> Noun (sostantivo)	e.g.,	chair, home, dog
V -> Verb	e.g.,	study,debate,drive
ADJ -> Adjective	e.g.,	purple,tall,ridiculous
ADV -> Adverb		unfortunately, slowly
P -> Preposition		of, by, to
Pro -> Pronoun		I, me, mine
DET -> Determiner		The, a, that, those
CONJ -> Conjuction		and, or

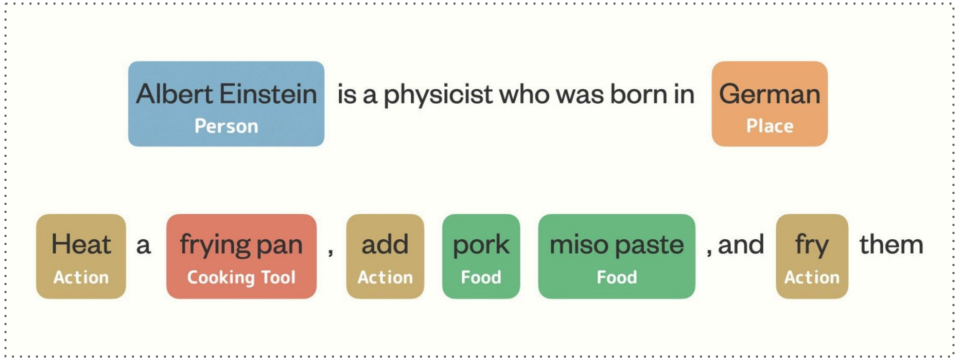
NLP Tasks: Named Entity Recognition - NER

- Named entity recognition is a linguistic task that involves the identification of proper names in text and classification into a set of predefined categories of interest (e.g., person, location, organisation, date-time, measures)

- It classifies also set of words

- NER can be also domain-specific (name of drugs, medical conditions, bibliographic references…etc)





Sequence Data

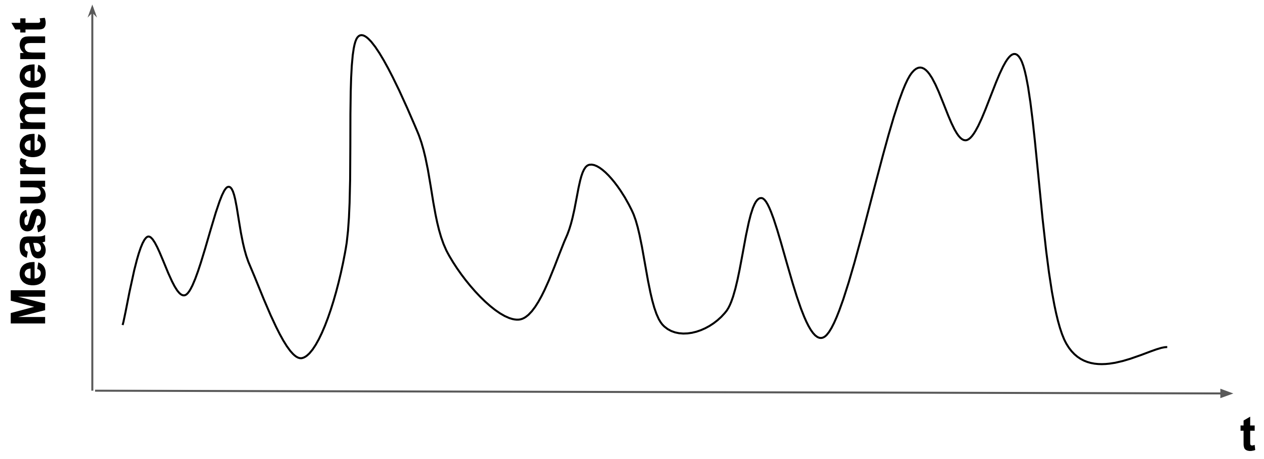
- Until now the features of each sample were some characteristics that help to describe and distinguish each sample
    - They were usually also independent from each other (If two features are highly correlated we can avoid to use one of them for example)

- Sometimes our data can present another important characteristic that we ignored until now: **sequentiality** (or time)

- However for some domains this aspect is crucial:
    - **Time series**: A series of values of a quantity obtained at successive times, often with equal intervals between them

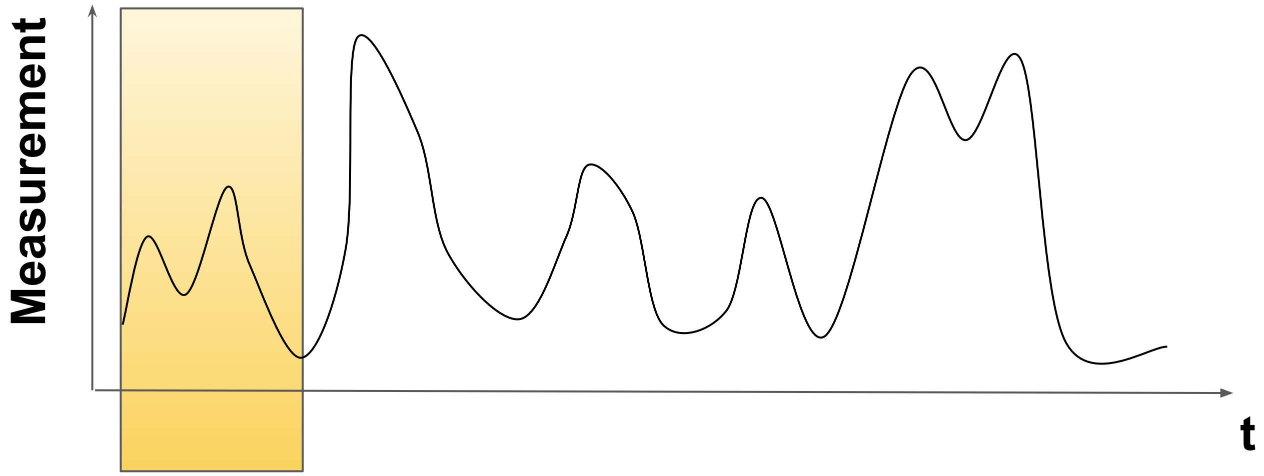
    - **Natural Language Processing**: The sequence of words in a sentence has a meaning and its important

Sliding window to analyze sequence data



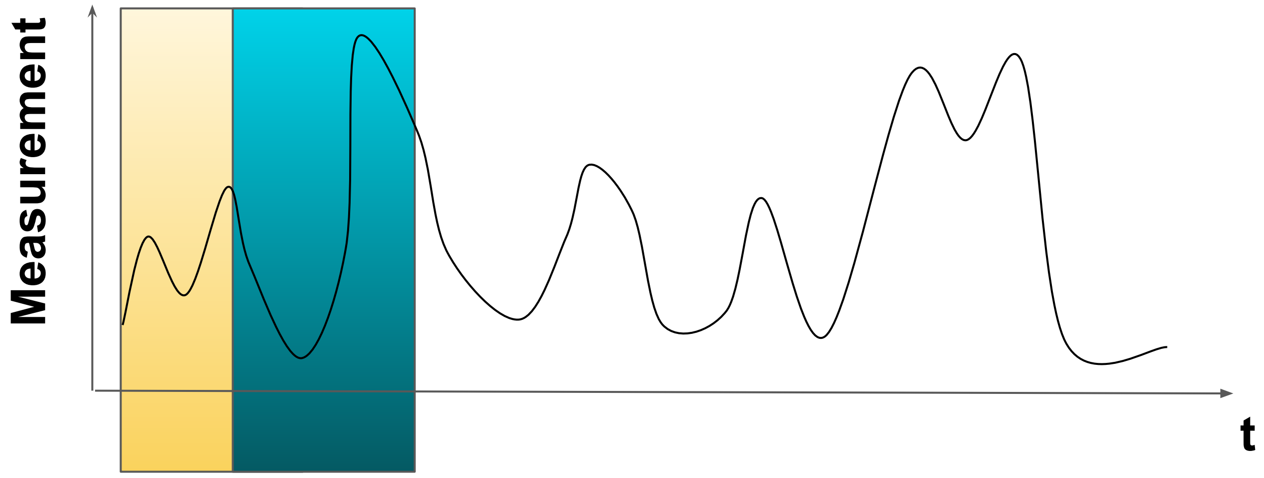
How can we extract features from a sequence ?

Sliding window to analyze sequence data



We define a sliding window and we extract features from each window

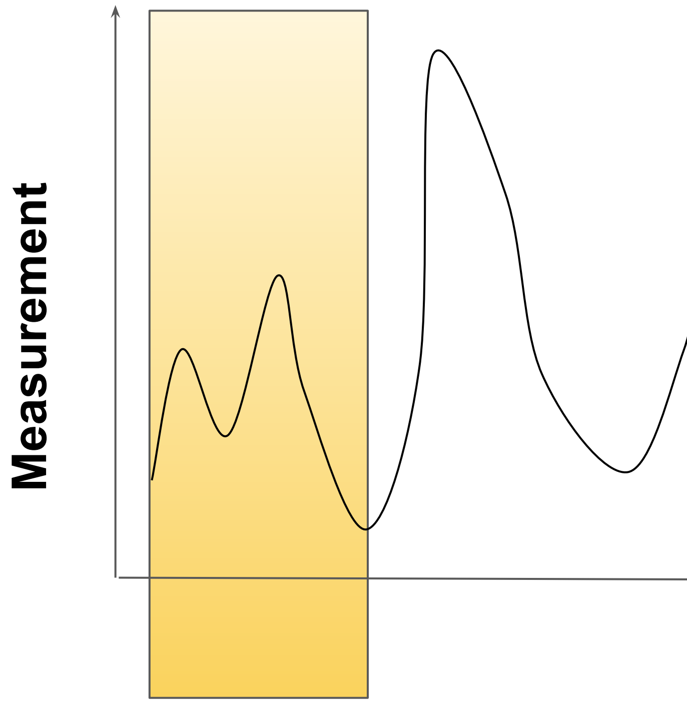
Sliding window to analyze sequence data



- We define a sliding window and we extract features from each window
    - Often with an overlapping of 50%

Sliding window to analyze sequence data

- We can compute statistics or apply signal processing techniques to manual extract features
  - Average, Standard Deviation, Fourier Transform
  - In this way, we extract the traditional feature x and from each window we get some features
  - We can apply all the algorithms already seen

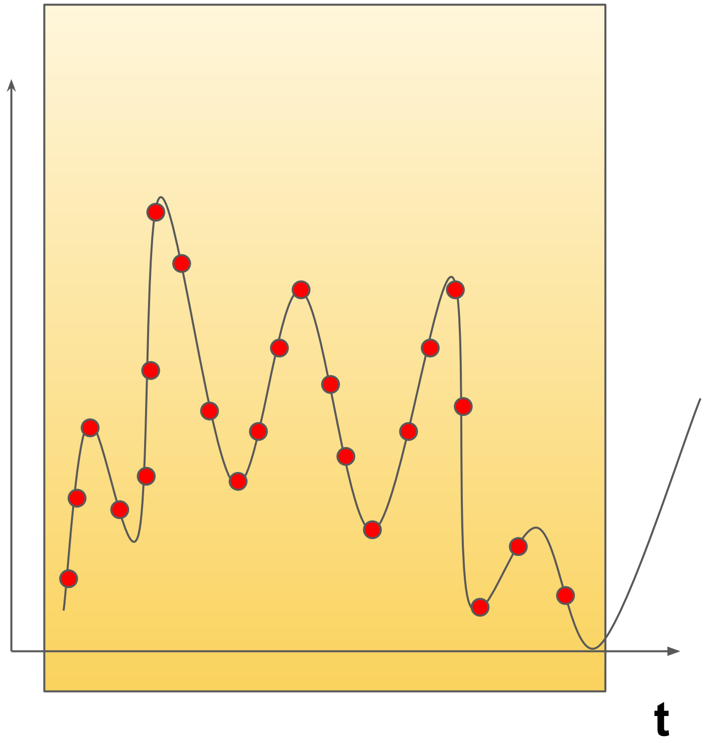


Sliding window to analyze sequence data

- Another techniques to maintain sequentiality is to extract features from the window by sampling the signal inside.

- Each sample x(0), x(1), x(2).....x(n) will be a different feature of our example

- ...Are the previous models able to learn from a structure like this ?



Example: Window of 2300 ms with a sampling frequency of 100ms

Text as a sequence

- Text can be seen as a sequence of words

- The window can be a sentence in this case and we often refers to it as the ***context***

- We do not need to apply sampling as in the signal processing because our signal is already discrete (a list of words).

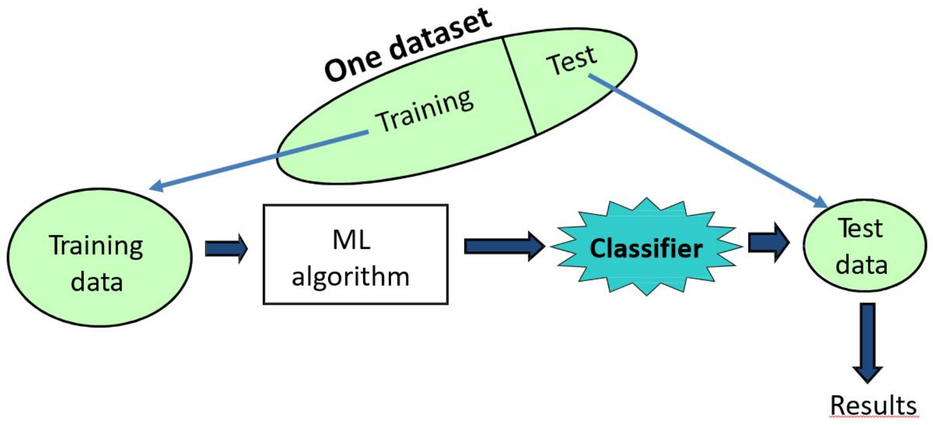
- In general, words are the features of each sample in NLP…but do we have to take into account each word ?

- Moreover, words are not numerical values so we need to encode words in some way

Text mining

Training Set	
Attribute 1	**Class**
• This movie is awsome	POSITIVE
• I didn't like that movie so much	NEGATIVE

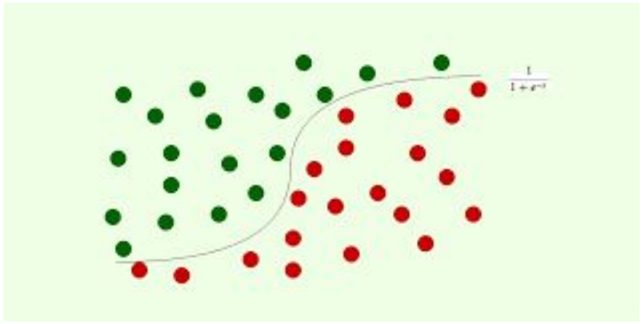
Test Set	
Attribute 1	**Class**
• I really enjoyed that movie	POSITIVE



Problem: What are the features ?

Classical Dataset for Classification

X_1	X_2	Class
4.2	1.3	Class A
4.1	1.6	Class B
..	..	..



Sentence Classification

Attribute 1	Class
• This movie is awsome	POSITIVE
• I didn’t like that movie so much	NEGATIVE

?

The Bag-Of-Word model (BoW)

- We build a vocabulary with all of the words and encode each sentence with binary vectors
- Vectors’ size is the same of the vocabulary. Each vectors’ component is a feature
- We can group words to get more sophisticated features (N-gram)

(1) John likes to watch movies.
(2) John also likes to watch football games.

Collect the Bag of words

John	likes	to	watch	movies	also	football	games	Mary	too

Use the Bag of Words to represent the sentences with numbers

BOW with Count Value

	John	likes	to	watch	movies	also	football	games	Mary	too
(1)	1	1	1	1	1	0	0	0	0	0
(2)	1	1	1	1	0	1	1	1	0	0

BoW with Python

```
from sklearn.feature_extraction.text
import CountVectorizer

corpus = ['This is the first document.' ,
'This document is the second document.' ,
'And this is the third one.' ,
'Is this the first document?' ]

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)

print(vectorizer.get_feature_names())

print(X.toarray())
```

Output

```
vectorizer.get_feature_names()

['and', 'document', 'first',
'is', 'one', 'second', 'the',
'third', 'this']

----------------------------

X.toarray()

[[0 1 1 1 0 0 1 0 1]
[0 2 0 1 0 1 1 0 1]
[1 0 0 1 1 0 1 1 1]
[0 1 1 1 0 0 1 0 1]]
```

Text Normalization (or pre-processing)

- Normalizing text means converting it to a more convenient, standard form

- First we want to separate out or tokenize words from running text, the task of **tokenization**
    - Words are often separated from each other by whitespace, but whitespace is not always sufficient
        - New York or Rock ‘n’ roll

- We may want to consider groups of words together, the so-called N-grams

BoW with Python - Ngrams

```
...

corpus = ['This is the first document.' ,
'This document is the second document.' ,
'And this is the third one.' ,
'Is this the first document?' ]

...

CountVectorizer (ngram_range=(1,2))

...
```

Output

```
vectorizer.get_feature_names ()

['and', 'and this', 'document',
'document is', 'first', 'first
document', 'is', 'is the', 'is
this', 'one', 'second', 'second
document', 'the', 'the first',
'the second', 'the third',
'third', 'third one', 'this',
'this document', 'this is',
'this the']
```

First problems of BoW

- The dimension of the encoding vector for each word or for each document depends mainly on the vocabulary size
  - This is not positive since in general a Language has 500k different words considering the Derivational morphology (morphology that creates new lexemes)
  - If we create a dictionary using only our dataset we may have the same problem is our dataset is big and rich
    - Computational issues and Curse of dimensionality

- While preprocessing text we should also think to a strategy to reduce the number of words that we take into account.

BoW with Python - Most frequented words

- We can select a maximum number of desidered features
- Only most frequent words in the corpus are encoded

```
...

corpus = ['This is the first document.' ,
'This document is the second document.' ,
'And this is the third one.' ,
'Is this the first document?' ]

...

CountVectorizer (max_features=3)

#it takes the most frequent

...
```

Output

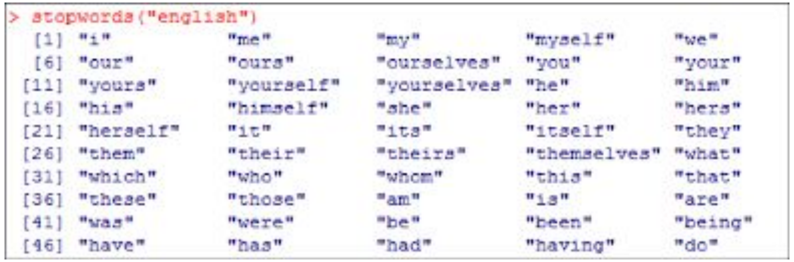
```
vectorizer.get_feature_names (
)
['document', 'is', 'the']

X.toarray()

[[1  1  1]
[2  1  1]
[0  1  1]
[1  1  1]]
```

Stopwords

- If we keep only most frequent words we probably will keep the most common words in a language that would not be useful as features to learn how to distinguish for example a positive or negative sentence

- We usually remove the so-called **Stopwords (Most common words in a language)** as first pre-processing task

- If stopwords have been removed we can select a max feature parameters in order to keep the most frequent words in our dataset (stopwords excluded)



```
> stopwords("english")
 [1] "i"          "me"         "my"         "myself"     "we"
 [6] "our"        "ours"       "ourselves"  "you"        "your"
[11] "yours"      "yourself"   "yourselves" "he"         "him"
[16] "his"        "himself"    "she"        "her"        "hers"
[21] "herself"    "it"         "its"        "itself"     "they"
[26] "them"       "their"      "theirs"     "themselves" "what"
[31] "which"      "who"        "whom"       "this"       "that"
[36] "these"      "those"      "am"         "is"         "are"
[41] "was"        "were"       "be"         "been"       "being"
[46] "have"       "has"        "had"        "having"     "do"
```

Lemmatization and Stemming

- In addition we need to perform another task that is **Lemmatization**: Determining that two words have the same root, despite their surface differences:
  - Sang, sung and sings are all forms of the verb sing

  - It is a complex task that requires human knowledge and often it is performed with an associative Map.

- **Stemming** refers to a simpler version of lemmatization in which we mainly just strip suffixes from the end of the word:
  - Beautiful, beauty —> become beaut

N-Grams can be expensive

- To take into account also nouns composed by multiple words we can compute all the possible n-grams. However it is a trade-off because we want to represent these nouns but at the same time we do not want that the number of features literally growths exponentially

- Using n-grams above 3 is not suggested for computational limits

- Sometimes multiple words should be encoded together to have a better representation especially for some tasks related with knowledge extraction

- An idea could be considering the structure of each sentence to distinguish which “role” every word or set of words have

  - In this case Named Entity Recognition **(NER)** may help to detect this set of words

BoW limits

- BoW represents words as discrete symbols and it is defined as a **localist representation**
  - Indeed words are represented independently from the context, order and frequency with the **one-hot encoding**
    - And documents are represented in the same way too
    - One-hot vectors are **orthogonal** and thus similarity measures based on vectors (like Cosine similarity) cannot be used
    - One-hot vectors are not so representative if we use these vectors as features for ML

- Example: In web search, if user searches for “Seattle motel”, we would like to match documents containing “Seattle hotel” but

  - motel = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **1** 0 0 0 0]
  - hotel = [0 0 0 0 0 0 0 **1** 0 0 0 0 0 0 0 0 0 0 0 0]

TF-IDF

- The tf–idf or TFIDF, short for ***Term Frequency–Inverse Document Frequency***, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

  - Document **A** has words**,** *«hello»* appears **5** times in the document **A**:

$$\textbf{Tf}_{\textbf{hello,A}} = \frac{5}{100} = 0.05$$

  - If we have **1000** documents in the collection, and *«hello»* appears in **10** documents:

$$\textbf{Idf}_{\textbf{hello}} = \log\frac{1000}{10} = 2$$

  - So:

$$\textbf{Tf-Idf}_{\textbf{hello,A}} = 0{,}05 \ * 2 = 0.1$$

BoW with TF-IDF

```
from sklearn.feature_extraction.text
import TfidfVectorizer

corpus = ['This is the first document.' ,
'This document is the second document.' ,
'And this is the third one.' ,
'Is this the first document?' ]

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(corpus)

print(vectorizer.get_feature_names())
print(X.toarray())
```

Output

```
vectorizer.get_feature_names()

['and', 'document', 'first', 'is',
'one', 'second', 'the', 'third',
'this']

-----------------------------

X.toarray()

[[0.  0.46 0.58 0.38 0.  0.  0.38 0.
0.38]
 [0.  0.68 0.  0.28 0.  0.53 0.28 0.
0.28]
 [...]
]
```

BREAK

Toward Distributional Semantics

- BoW is a localist representation and does not take into account several important factors depending on the task:
  - Words’ order in a document
  - Synonyms
  - The context
  - Orthogonality
  - Low informative word vectors with one-hot-encoding
    - Dimension related with the dictionary size

- Using TF-IDF based BoW we solved only a part of these problems
  - Vector dimension is still problematic! No orthogonality but Curse of dimension still present
  - No words’ order
  - We take into account the general context of all the documents and the vectors are more “informative” as features rather than using only the one-hot BoW

Distributional Semantics

- **Distributional semantics:** *A word's meaning is given by the words that frequently appear close-by*
  - *"You shall know a word by the company it keeps" (J.R. Firth 1957)*

- When a word ***p*** appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window)

- **Distributed Representation**: Use the many context of ***w*** to build up (learn) a representation of **w**

…government debt problems turning into banking crises as happened in 2009….

….saying that Europe needs unified banking regulation to replace the….

…Indica has just given its banking system a shot in the arm…..

Distributional Semantics

- What we want to build:
  - Dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts (For example: King and Queen should be similar since they co-occur in the same contexts)
  - Arbitrary-dimension word vectors: We want to choose dimension as a design parameter
  - Since these vectors should be based on the different contexts words belong to in our corpus (dataset) we want to learn these vectors with a data-driven algorithm and by solving an optimization problem
- Word vectors are called **word embeddings**

Word2Vec - Mikolov et al 2013 (Google)

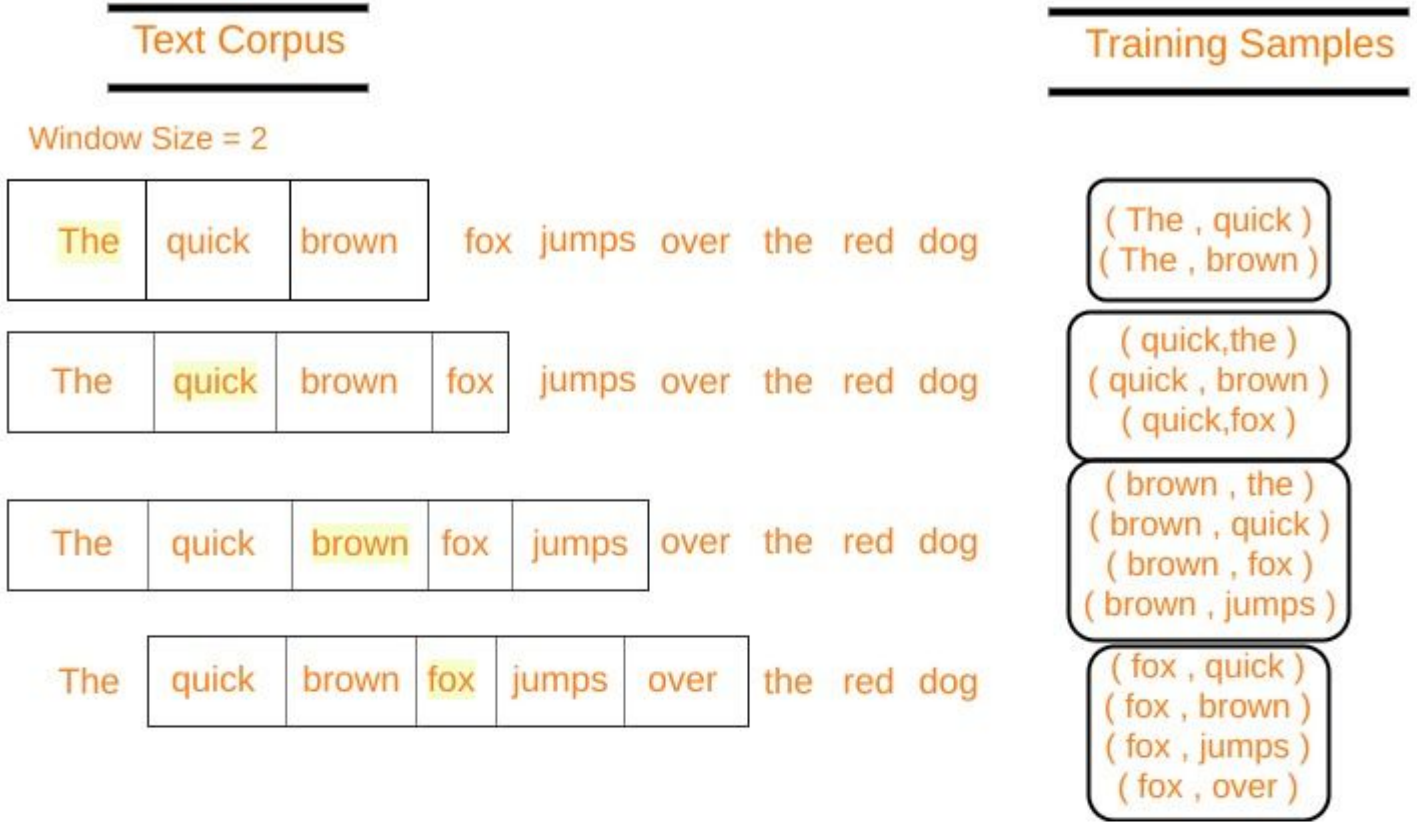
- Word2Vec is an algorithm for learning word vectors that exploits the Representation Learning ability of Neural Networks (NNs) (the so-called embedding)

- To learn these vector representation of each word **w** we want to take into account every context of surrounding words of that word (Window)

- The goal is to train a Neural Net <u>to predict the context given a word in input</u> (it is called the **Skip-gram model**)

- To achieve this result we should train a particular (but simple) NN that have the same number of input units on the output layer and with a compression layer in the middle

How training samples are extracted for Word2Vec



Encoder- Decoder Architecture

- Suppose you have these sentences
  - *The **King** is sleeping in the castle*
  - *The Queen is walking in the castle*

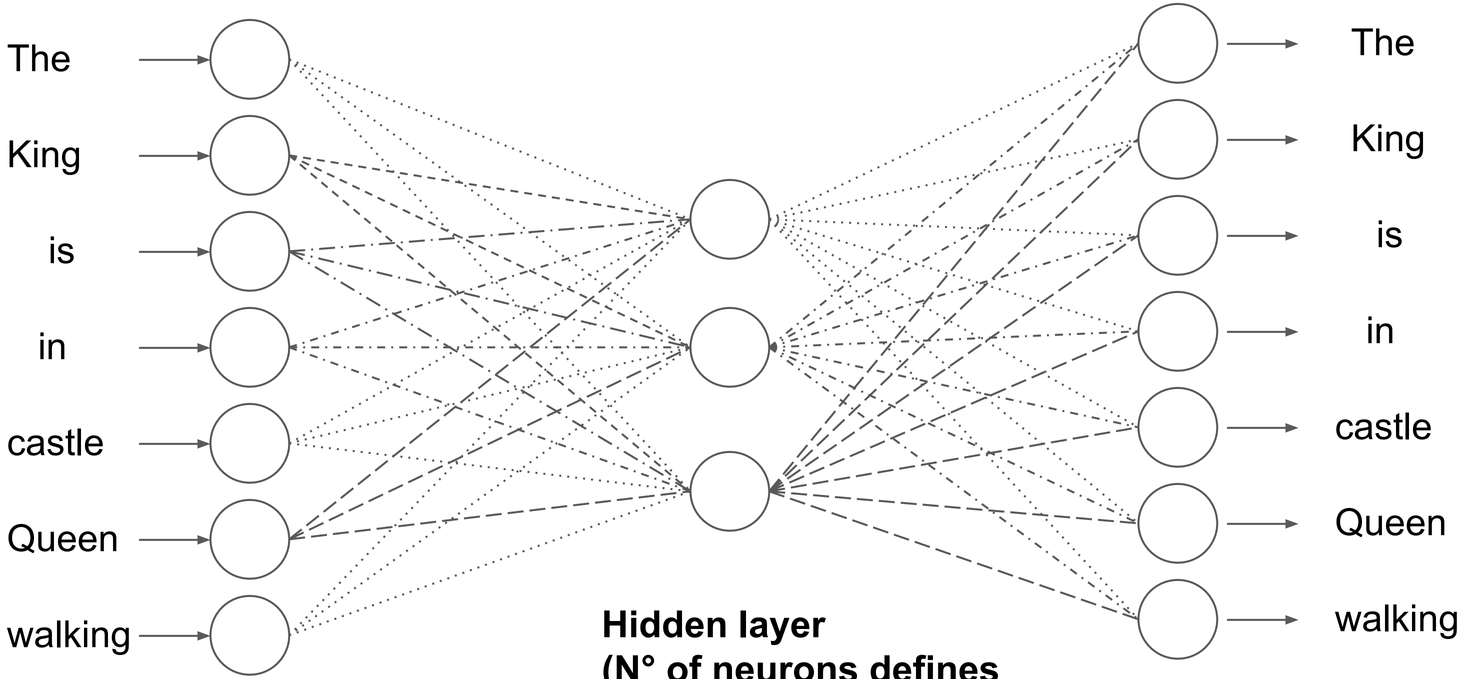
- Suppose a window of context equal to 5
  - It means that for each word we consider the 5 words on the left and the 5 words on the right

- So if the input is **King**, the Neural net should learn to predict **{The, is, sleeping, in, castle}**

- Since the input and the output can be every word in the dictionary, input layer and output layer have the same dimension

- Words are encoded with one-hot encoding

Encoder- Decoder Architecture

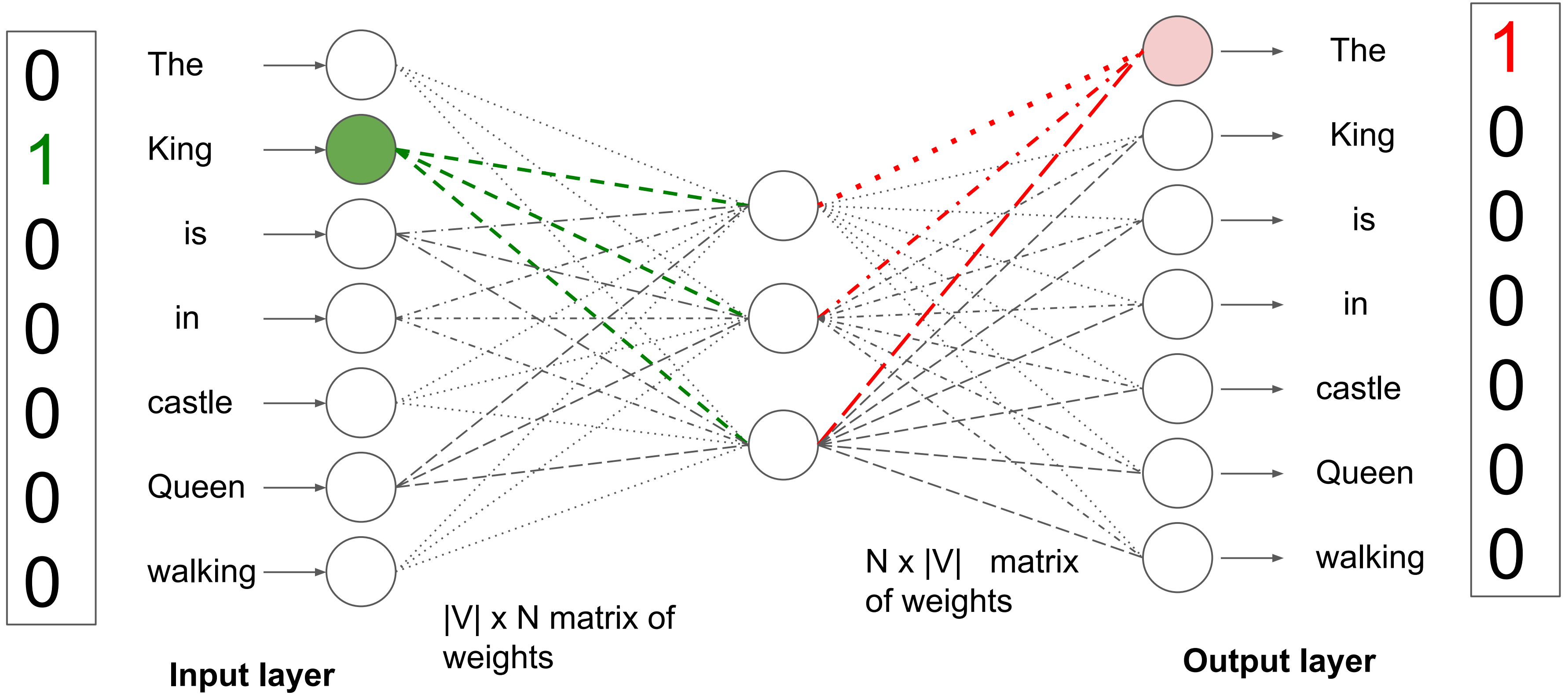


Input layer (Dimension of vocabulary V)

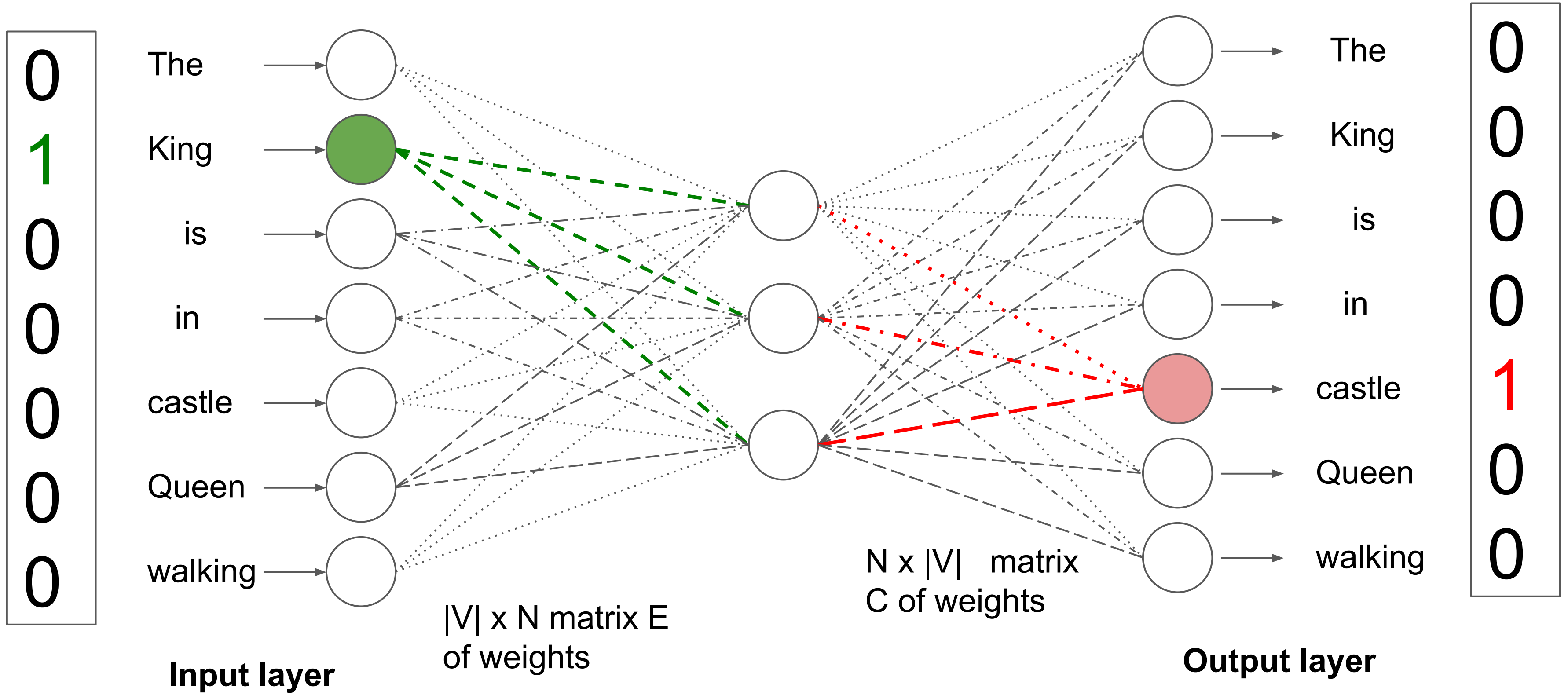
Hidden layer (N° of neurons defines the dimension of word embeddings) In this case N=3

Output layer (Dimension of vocabulary V)

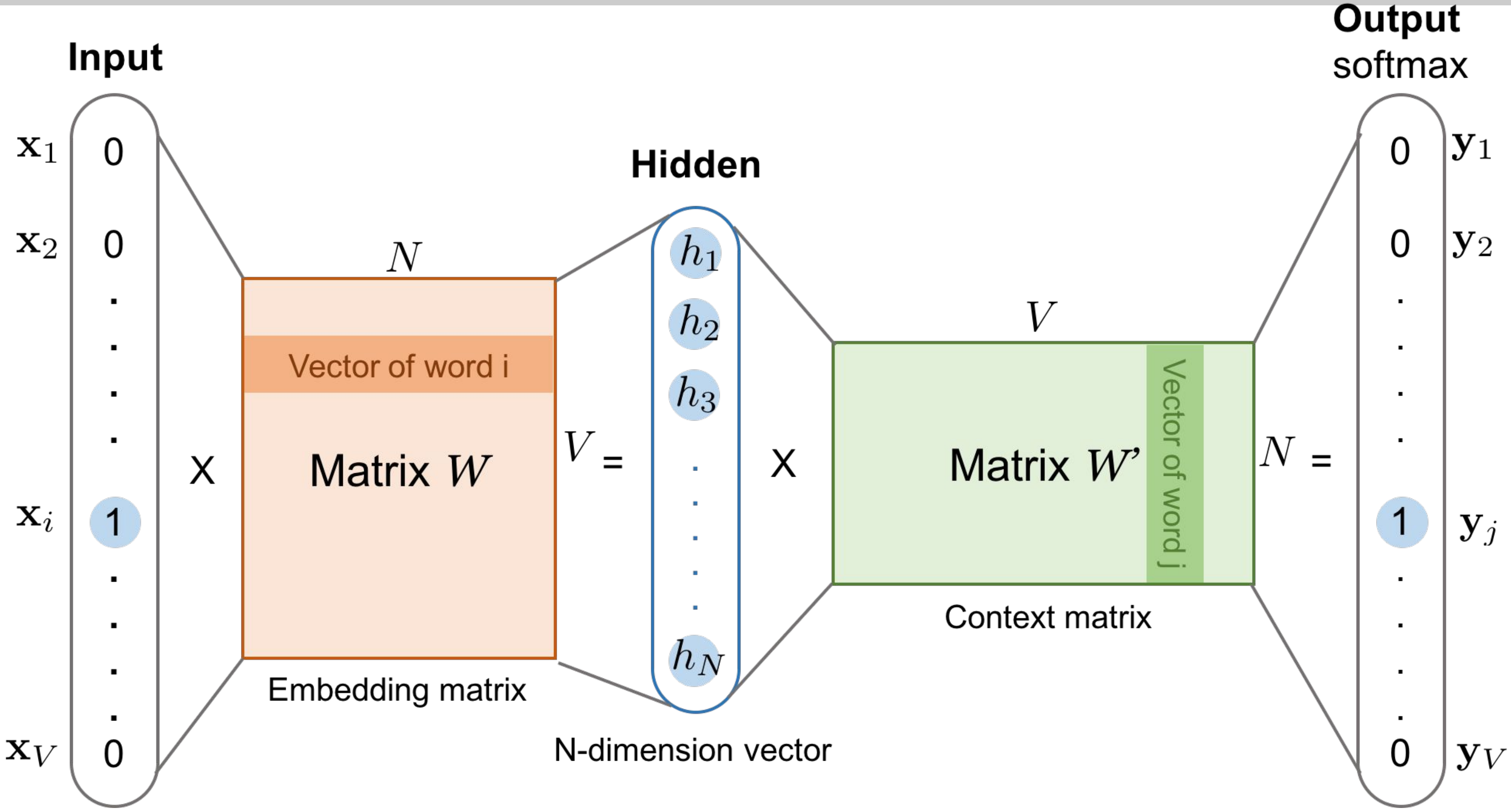
Skip-gram model {King, The}



Skip-gram model {King, castle}



Skip-gram model - In general



Skip-gram model - In general

$$\begin{array}{l|c|}
\text{The} & 0 \\
\text{King} & 1 \\
\text{is} & 0 \\
\text{in} & 0 \\
\text{castle} & 0 \\
\text{Queen} & 0 \\
\text{walking} & 0
\end{array}
\qquad
\begin{bmatrix}
0.3 & 0.5 & 0.1 \\
\underline{\mathbf{0.01}} & \underline{\mathbf{0.1}} & \underline{\mathbf{0.8}} \\
0.1 & 0.25 & 0.4 \\
0.8 & 0.5 & 0.5 \\
0.9 & 0.5 & 0.1 \\
0.2 & 0.5 & 0.1 \\
0.11 & 0.2 & 0.3
\end{bmatrix}$$

Matrix E and in green the embedding for the word King

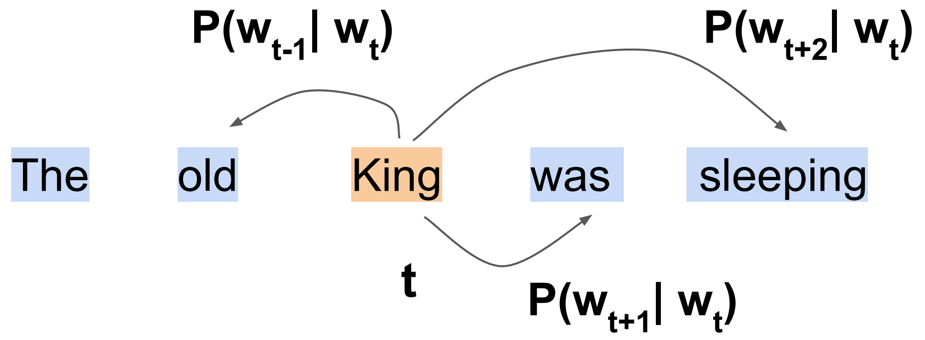
$$\begin{bmatrix}
0.1 & 0.3 & 0.2 & 0.6 & 0.9 & 0.1 & 0.2 \\
0.2 & 0.1 & 0.4 & 0.1 & 0.8 & 0.3 & 0.5 \\
0.1 & 0.5 & 0.2 & 0.1 & 0.9 & 0.4 & 0.1
\end{bmatrix}$$

Matrix C that determines the output

$$\begin{array}{l|c|}
\text{The} & 0 \\
\text{King} & 0 \\
\text{is} & 0 \\
\text{in} & 0 \\
\text{castle} & 1 \\
\text{Queen} & 0 \\
\text{walking} & 0
\end{array}$$

Word2Vec formally

- Every word in a fixed vocabulary has to be represented by a vector with an arbitrary dimension
- For each position ***t*** in the text (that represents the center of the context) consider to learn to predict the surrounding words ***o*** in the context
- Formally we are computing conditional probabilities that a word o occurs given the word at position **t**
- Keep adjusting the word vectors to maximize this probability (learning weights during training)



Word2Vec Objective (cost) function

- For each position t=1…..T, predict context words within a window of fixed size m, given center word w_j

Likelihood = L(θ) = $$\prod_{t=1}^{T} \prod_{-m \leq j \leq m} P(w_{t-1} | w_t; \Theta)$$

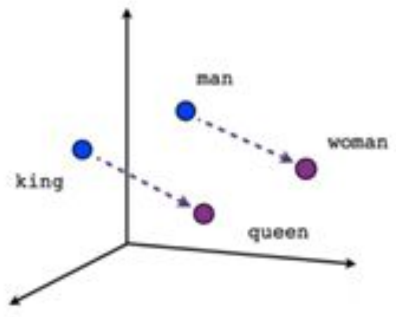
- The objective function J(θ) is the average negative log likelihood

$$J(\Theta) = -\frac{1}{T} log L(\Theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m} P(w_{t-1} | w_t; \Theta)$$

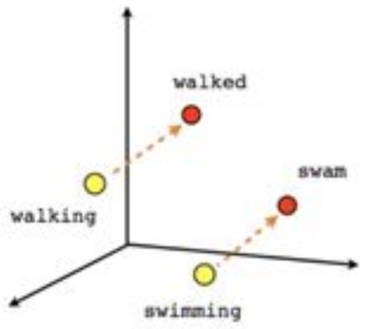
- **Minimizing this objective functions** $\Longleftrightarrow$ **Maximizing predictive accuracy**

Semantic Algebra

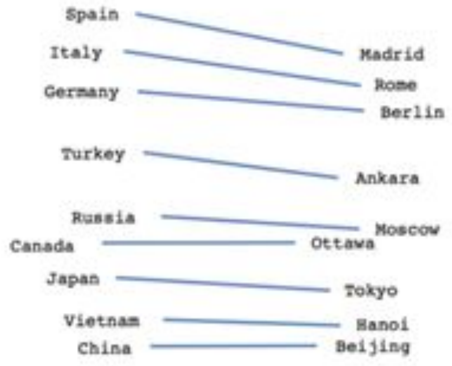
- Word embeddings learnt with Doc2Vec seem to enable to a semantic algebra among words using their relative distances



Male-Female



Verb tense

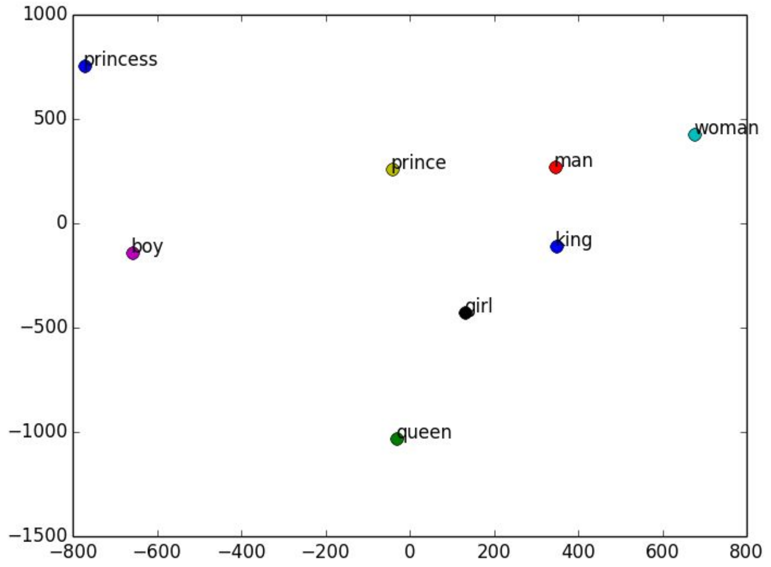


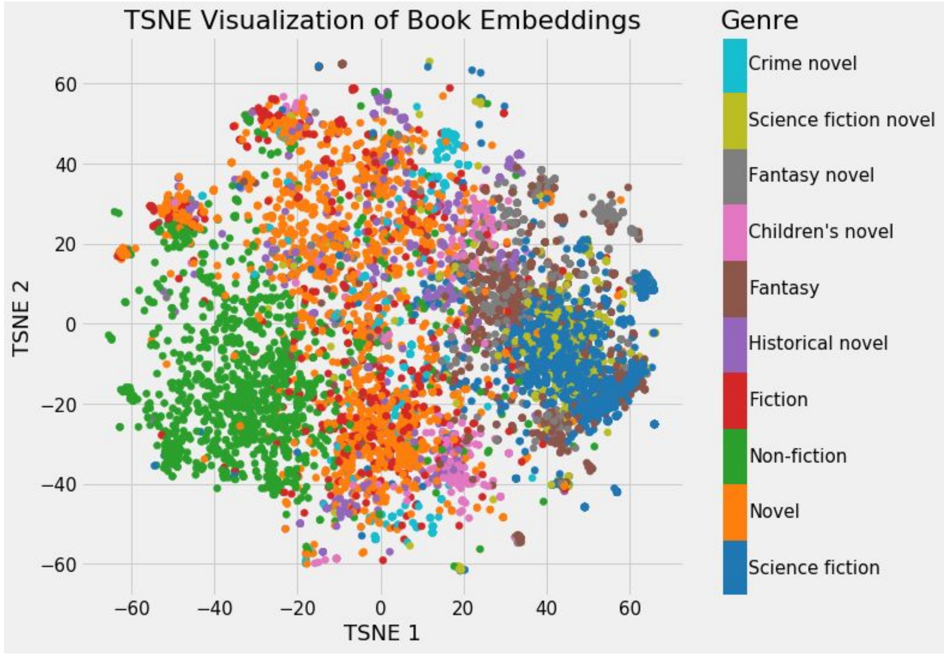
Country-Capital

- It was shown for example that vector(”King”) - vector(”Man”) + vector(”Woman”) results in a vector that is closest to the vector representation of the word Queen

Word embedding visualization

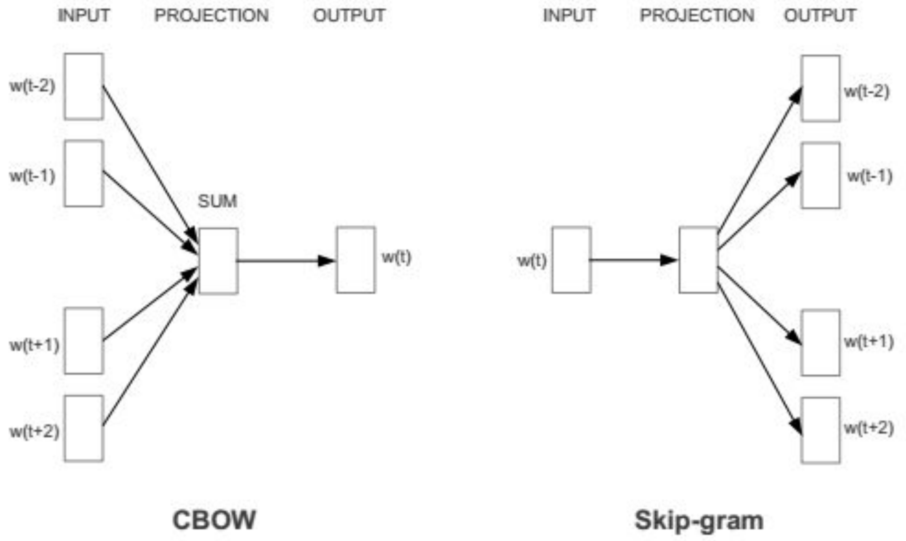
Usually Word embedding visualization is performed using the t-SNE dimensionality reduction technique because it preserves the relative distances in the original multivariate space





Continuous Bag-of-Words

An alternative to the Skip-gram in Word2Vec is the CBoW (Continuous Bag-of-words) where the Neural network is trained to predict the missing word given the context



What about Documents?

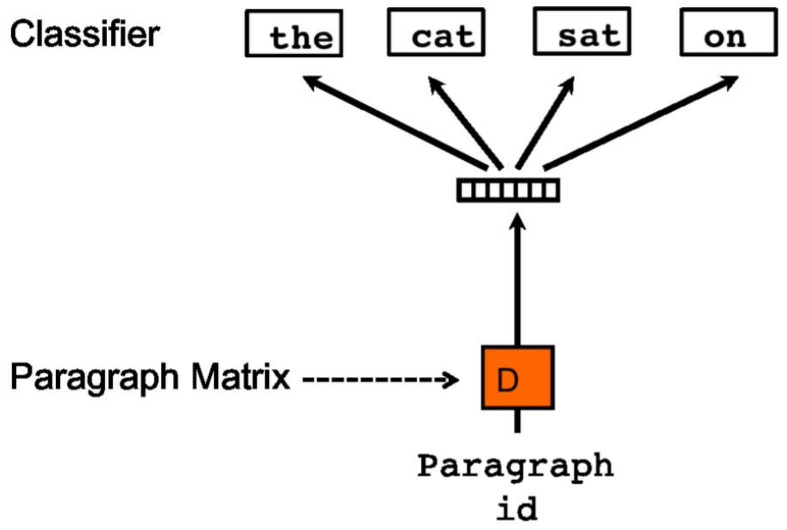
- We saw how to learn word feature vectors but what we have to do to encode an entire document ?
    - We can learn word embeddings and then take the average or the sum of them to encode each document
    - We can learn a Document embedding with **Doc2Vec**

Doc2Vec - Mikolov et al 2014

- We add a special word in the document with a specific id (called Paragraph ID)

- Then we say that the context of this special word is the entire document (all the words in the document)

- Learning word-embeddings we will learn also the embedding for the Paragraph ID that will be the embedding for the entire document


Classifier
the
cat
sat
on
Paragraph Matrix
D
Paragraph id