



# Natural Language Processing

AIS

Romain Benassi

Spring 2022

# Romain Benassi



**Data Scientist (PhD)** with specialty in Deep Learning, **NLP**, Energy and Bayesian methods

- Since 2020: Consultant at ***Publicis Sapient France***, mainly in **NLP** field

publicis  
sapient

- 2013-2019 : Data Scientist in the field of energy (in several start-ups)

- 2013 : PhD on Bayesian statistics (***Centrale-Supélec***)



- 2009 : Engineer's degree (***IMT Atlantique***) with specialization in signal processing



# Romain Benassi

## Experience in **Natural Language Processing (NLP)**

Since 2019, three projects on this field

- **Vidal**: work on deep learning algorithms for automatic indexation of medical documents
- **Enedis**: Automatic classification, and topic extraction, from strategic text documents
- **Tamalou** project: development of a supervised clustering algorithm and automatic reply association for medical texts from internet forums



Co-speaker in Devoxx France 2022 : *l'IA pour le bon usage du médicament* (Vidal & Publicis Sapient)

<https://www.youtube.com/watch?v=1qeiou8GGj8>



publicis  
sapient

One blog article (in French)

<https://blog.engineering.publicissapient.fr/2021/03/17/nlp-concepts-cles-et-etat-de-lart/>

Online formation certifications (as student)

- Natural Language Processing Specialization (4-course specialization)
- Introduction to TensorFlow for Artificial Intelligence, Machine Learning, and Deep Learning
- Natural Language Processing in TensorFlow



# Course Schedule

- **Course 1:** NLP introduction
- **Course 2:** Word embedding
- **Course 3:** LSTM (Long Short-Term Memory) principle
- **Course 4:** “Attention” Mechanism and Transformer Architectures
- **Course 5:** Chatbot

# Evaluation

A graded exam will be used as evaluation and will be done at the beginning of the last course (**Course 5**).

This exercise will contain

- multiple-choice-questions (MCQ)
- theoretical questions
- Coding questions

# Course Schedule

- **Course 1:** NLP introduction
- **Course 2:** Word embedding
- **Course 3:** LSTM (Long Short-Term Memory) principle
- **Course 4:** “Attention” mechanism and Transformer architectures
- **Course 5:** Chatbot

# Course Schedule

- **Course 1:** NLP introduction
  - NLP use cases
  - Preprocessing (tokenization, stemming, lemmatization...)
  - NLP libraries (spaCy and NLTK)
  - Text Mining (Bag of words, TF-IDF)
  - Topic Modeling (NMF, LDA)
- **Course 2:** Word embeddings
- **Course 3:** Long Short-Term Memory (LSTM) architecture
- **Course 4:** “Attention” mechanism and Transformer architectures
- **Course 5:** Chatbot

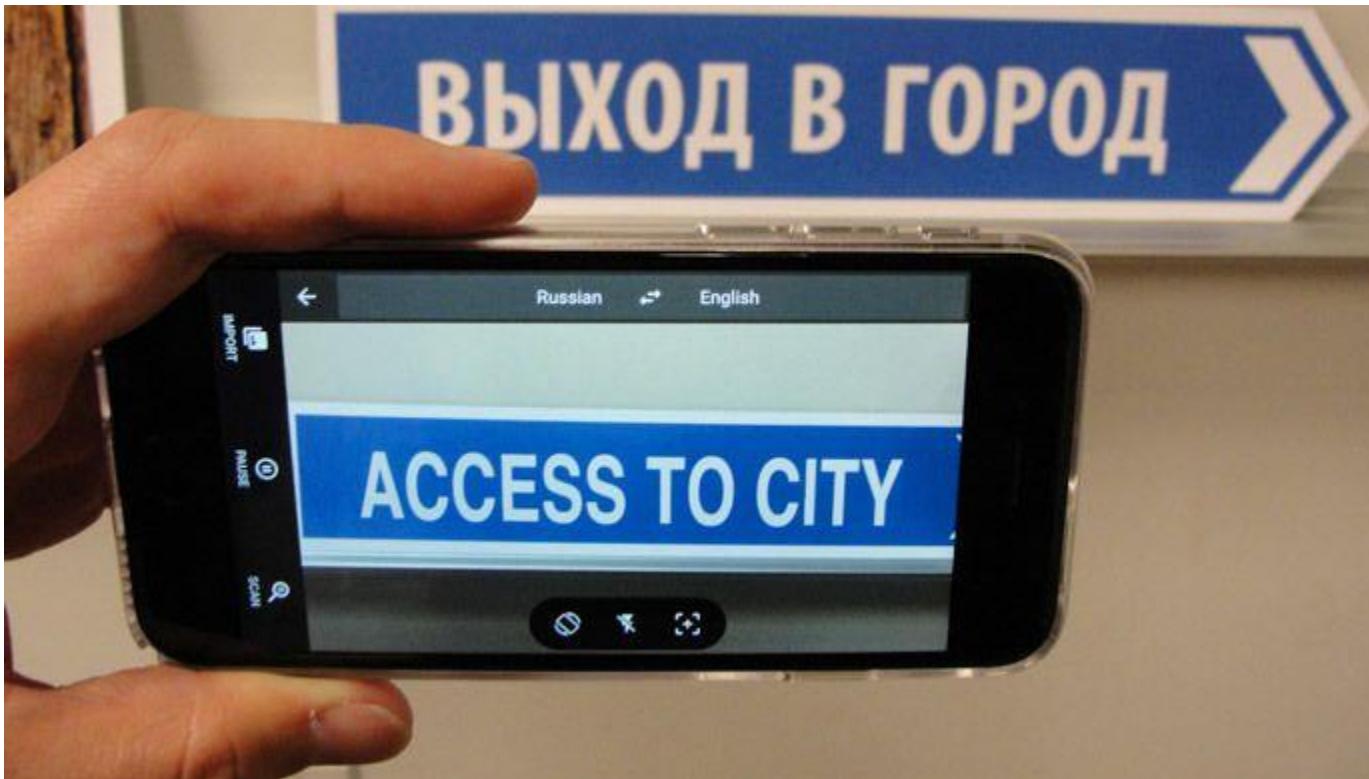


# Course 1: NLP Introduction

# NLP use cases

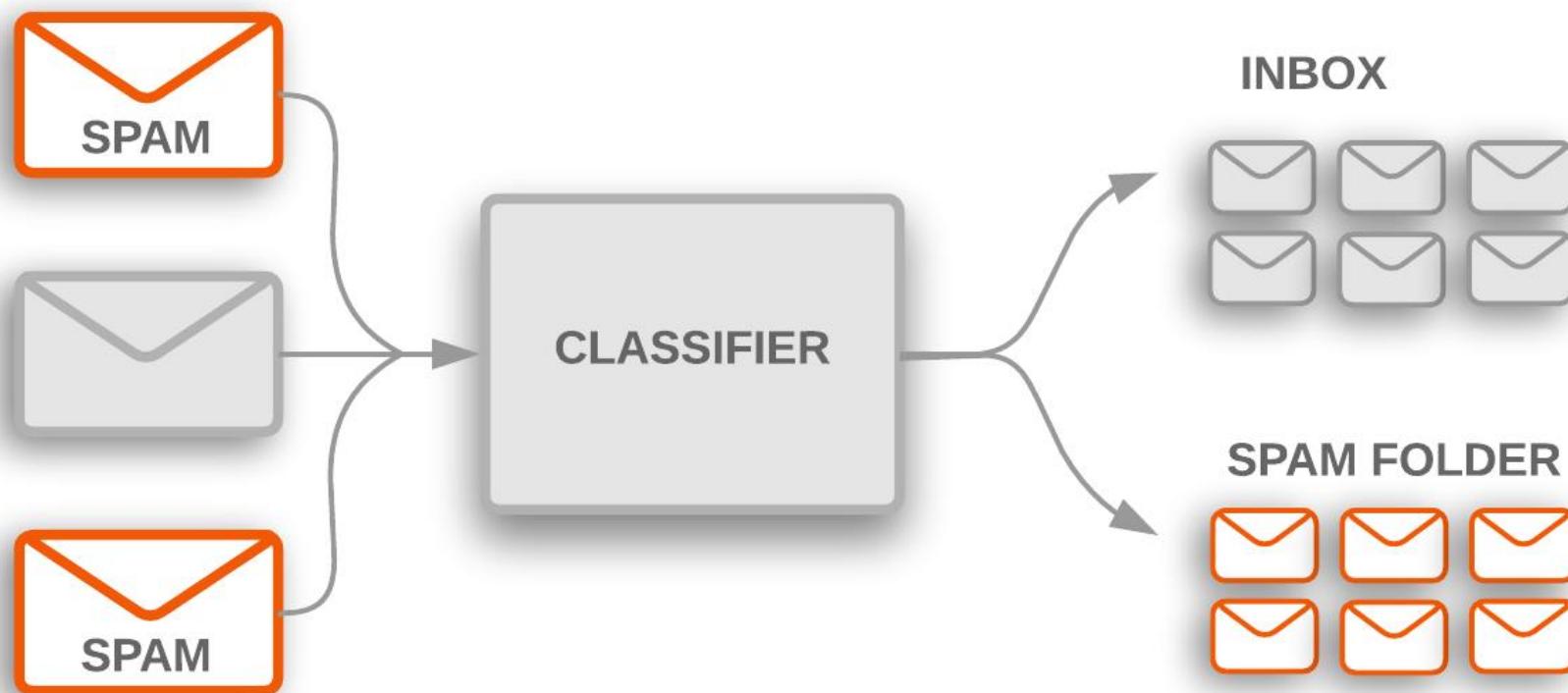
# NLP use cases

## Translation



# NLP use cases

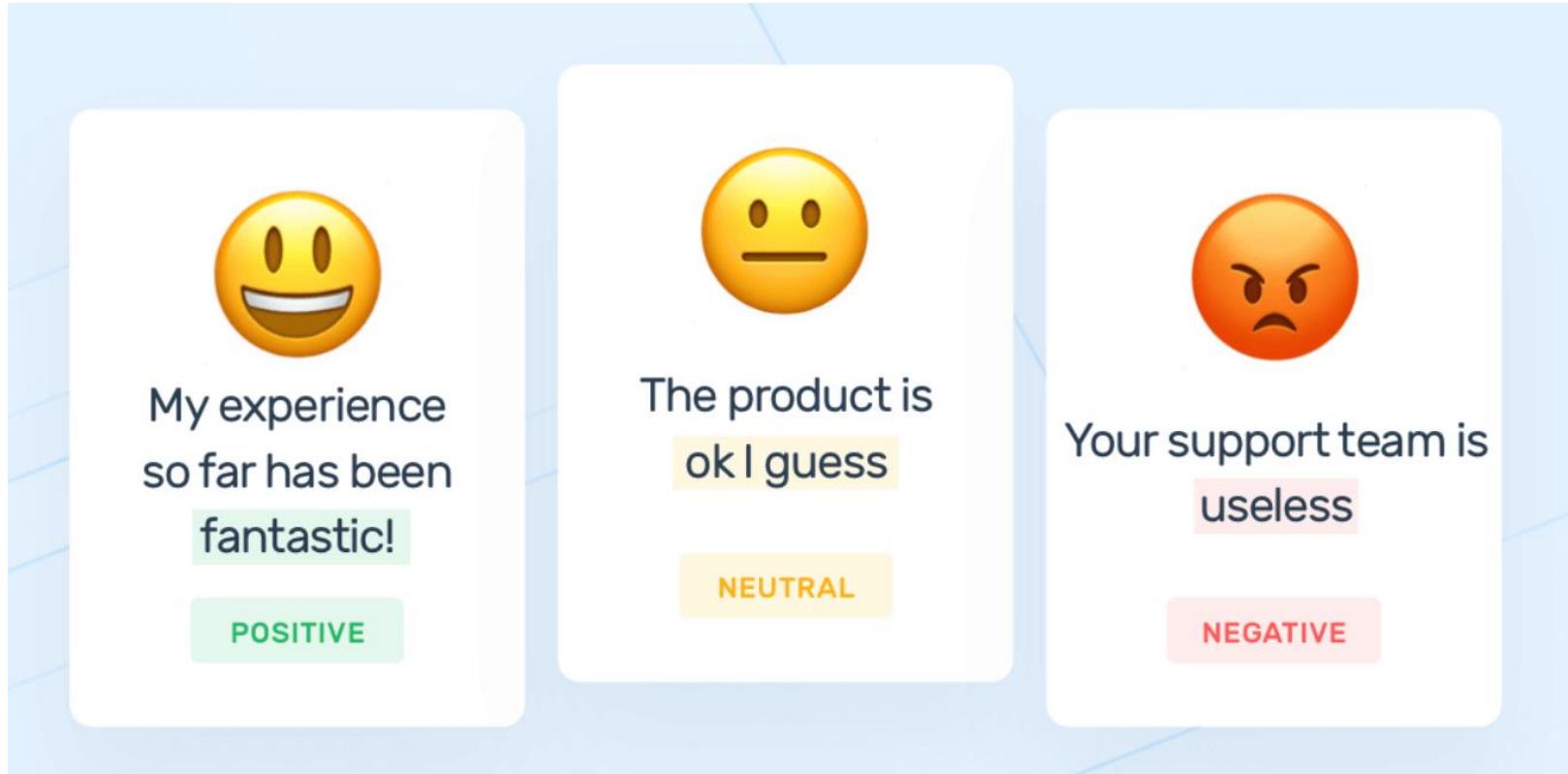
## Spam detection



<https://medium.com/@naveen.kumar.k/naive-bayes-spam-detection-7d087cc96d9d>

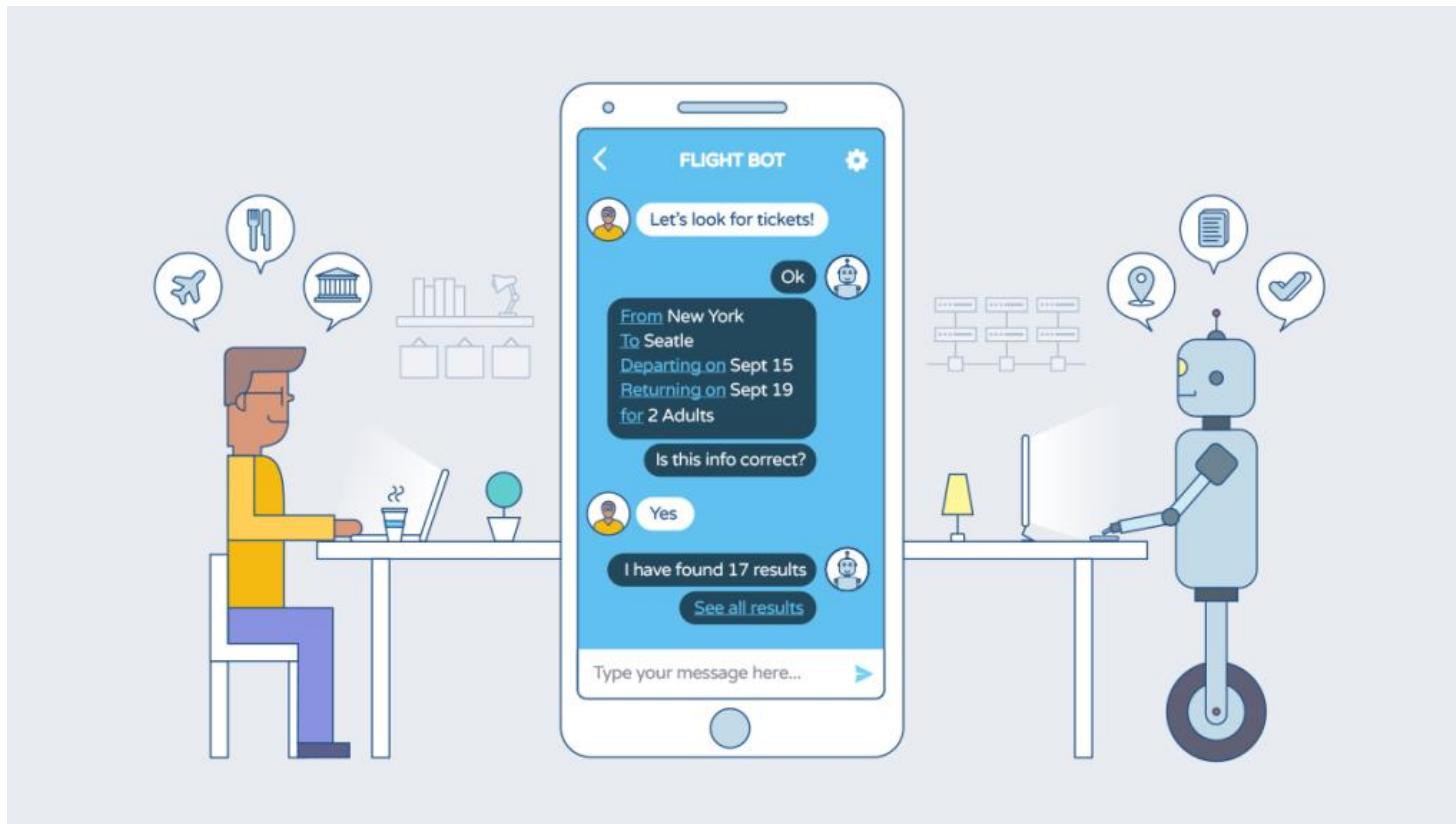
# NLP use cases

## Sentiment analysis



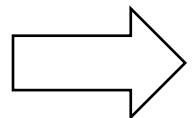
# NLP use cases

## Chatbot



# NLP main issue

Computers are known to be good at handling numerical data



**What can we do with text data?!**

# Preprocessing

# Common NLP tasks

- Tokenization
- Stemming
- Lemmatization
- Removal of stop words

# Tokenization

Tokenization is the breaking up of raw text into smaller relevant parts (tokens). These parts can be words or groups of successive letters.

Example of word tokenization:

**Raw text :** *My son's friend, however, plays a high-risk game.*

**Tokenized :** [My] [son] ['s] [friend] [,] [however] [,] [plays] [a] [high] [-] [risk] [game] [.]

# Stemming

Stemming is the process of reducing a word to its root form (word stem).

Most of the time, it consists to drop the last letters of the word until the stem is reached.

Example of stemming:

fishing, fished or fisher => fish

# Stemming

- Porter's algorithm (1980) is one of the most well-known stemming procedure used for English language
- Five reduction phases used in order to get the stem of the word
- Different variants of the original Porter's stemming procedure exists like Snowball or the English Stemmer (Porter2 stemmer)

# Stemming

- Porter's algorithm (1980) is one of the most well-known stemming procedure used for English language
- Five reduction phases used in order to get the stem of the word
- Different variants of the original Porter's stemming procedure exists like Snowball or the English Stemmer (Porter2 stemmer)

## First phase

<b>S1</b>	<b>S2</b>	<b>word</b>	<b>stem</b>
SSES	→ SS	caresses	caress
IES	→ I	ponies	poni
		ties	ti
SS	→ SS	caress	caress
S	→	cats	cat

## More sophisticated rules

<b>S1</b>	<b>S2</b>	<b>word</b>	<b>stem</b>
(m>0) ATIONAL	→ ATE	relational	relate
		national	national
(m>0) EED	→ EE	agreed	agree
		feed	feed

# Stemming

## Porter Stemmer (NLTK) example

```
run---->run
runner---->runner
ran---->ran
runs---->run
easily---->easili
fairly---->fairli
fairness---->fair
```

# Stemming

Snowball Stemmer (NLTK) was developed as well by Porter and offers a slight improvement both in logic and speed.

```
run      -----> run
runner   -----> runner
ran      -----> ran
runs     -----> run
easily   -----> easili
fairly   -----> fair
fairness -----> fair
```

# Lemmatization

Lemmatization is the algorithmic process of determining the canonical form (lemma) of a word.

Unlike stemming, it is not only a word reduction but depends on the meaning of the word in a sentence and needs to consider a language's full vocabulary.

## Example of lemmatization:

was -> be

meeting -> meet or meeting (depending on the context)

walking -> walk (identical to stemming)

# Lemmatization

- Lemmatization gives a more informative result than stemming
- It looks at surrounding text to correctly identify the part of speech and meaning of the word
- For these reasons, a lemmatization procedure exists directly in spaCy whereas this is not the case for stemming, considered as a less efficient preprocessing step
- Stemming procedures (including both Porter and Snowball) can be found in NLTK if needed

# Removal of stop-words

- A stop-word is a word so frequent that it makes it not relevant to take it into account during an analysis
- Most of the time, they are filtered out before any NLP processing
- Obviously, each language has its own specific set of stop-words
- Examples
  - In English: the, is, at, which, on...
  - In French: le, la, du, ce...
- spaCy has a list of 305 English stop-words

# NLP Libraries

# spaCy and NLTK

- spaCy (2015) and NLTK (2001) are both popular NLP libraries
- spaCy implements generally only one algorithm for each task, but the most efficient one currently available
- NLTK provides generally several algorithms for each tasks, but with less efficient implementations

spaCy gives more efficient implementations but offers less possibilities than NLTK

The choice between both should be made depending on the applications

# spaCy and NLTK

SYSTEM	ABSOLUTE (MS PER DOC)			RELATIVE (TO SPACY)		
	TOKENIZE	TAG	PARSE	TOKENIZE	TAG	PARSE
spaCy	0.2ms	1ms	19ms	1x	1x	1x
CoreNLP	0.18ms	10ms	49ms	0.9x	10x	2.6x
ZPar	1ms	8ms	850ms	5x	8x	44.7x
NLTK	4ms	443ms	n/a	20x	443x	n/a

# spaCy and NLTK

	SPACY	SYNTAXNET	NLTK	CORENLP
Programming language	Python	C++	Python	Java
Neural network models	✓	✓	✗	✓
Integrated word vectors	✓	✗	✗	✗
Multi-language support	✓	✓	✓	✓
Tokenization	✓	✓	✓	✓
Part-of-speech tagging	✓	✓	✓	✓
Sentence segmentation	✓	✓	✓	✓
Dependency parsing	✓	✓	✗	✓
Entity recognition	✓	✗	✓	✓
Coreference resolution	✗	✗	✗	✓

# NLP Basics: Tutorial

*course1\_basics\_NLP.ipynb*

**Goal:** Get used to classic NLP preprocessing operations (lemmatization, stemming, entity detection...) using both spaCy and NLTK library

# Text Mining

# Text Mining

- Examining of document collection (*corpus*) to discover new information
- This can be used as a first step for lots of different applications (topic modeling, spam detection, sentiment analysis, security...)
- We will focus on two of the most famous text mining approaches
  - Bag-of-words (BoW) model
  - TF-IDF

# Bag-of-words (BoW) model

In this model, a document  $d$  is represented as a set of tuples

$$\{w: \text{nb of occurrences of } w \text{ in } d \mid \text{for all word } w \text{ in } d\}$$

## Example

For the (short) document:

*“Xavier likes to play football. Eric likes football too.”*

The bag-of-word representation is:

$$\{Xavier: 1, likes: 2, to: 1, play: 1, football: 2, Eric: 1, too: 1\}$$

# Bag-of-words (BoW) model

In this model, a document  $d$  is represented as a set of tuples

$$\{w: \text{nb of occurrences of } w \text{ in } d \mid \text{for all word } w \text{ in } d\}$$

## Example

For the (short) document:

“Xavier *likes* to play *football*. Eric *likes* *football* too.”

The bag-of-word representation is:

$$\{Xavier: 1, likes: 2, to: 1, play: 1, football: 2, Eric: 1, too: 1\}$$

It is equivalent to a word-frequency histogram representation

# Bag-of-words (BoW) model

If we have several texts:

*Document1: “Xavier likes to play football. Eric likes football too.”*

*Document2: “Eric prefers tennis to football.”*

We have the representation below:

	Xavier	likes	to	play	football	Eric	too	prefers	tennis
Document1	1	2	1	1	2	1	1	0	0
Document2	0	0	1	0	1	1	0	1	1

This representation can be used to characterize texts (e.g in spam filtering)

# Bag-of-words (BoW) model

If we have several texts:

*Document1: “Xavier likes to play football. Eric likes football too.”*

*Document2: “Eric prefers tennis to football.”*

The documents are  
transformed into numerical  
representation

We have the representation below:

	Xavier	likes	to	play	football	Eric	too	prefers	tennis
Document1	1	2	1	1	2	1	1	0	0
Document2	0	0	1	0	1	1	0	1	1

This representation can be used to characterize texts (e.g. in spam filtering)

# Bag-of-words (BoW) model

This simple model have some drawbacks

- Term frequencies are not necessarily the best representation of a text
- Each time of new word appears, the length of the vector increases as well
- Most of the time, text representations are sparse (many zeros)
- No information about grammar or about the original word ordering is kept

# TF-IDF

- TF-IDF means term frequency-inverse document frequency
- This is a ponderation method used in information retrieval.
- This statistical measure gives **an evaluation of how important is a word to a document**, depending on the corpus considered
- The weight increase proportionally to the occurrences of a word in a text
- This weight is **also** offset by the number of documents in the corpus containing the word

# TF-IDF

**TF-IDF** is calculated as :  $\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D)$

## TF-IDF: $tf(t,d)$

TF-IDF is calculated as :  $tfidf(t, d, D) = \boxed{tf(t, d)} \cdot idf(t, D)$

### Term frequency $tf(t,d)$

$$tf(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}},$$

With  $f_{t,d}$  the number of times term  $t$  appears in document  $d$

# TF-IDF: $tf(t,d)$

Term frequency  $tf(t,d)$  variants :

Variants of term frequency (tf) weight

weighting scheme	tf weight
binary	0, 1
raw count	$f_{t,d}$
term frequency	$f_{t,d} / \sum_{t' \in d} f_{t',d}$
log normalization	$\log(1 + f_{t,d})$
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$
double normalization K	$K + (1 - K) \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

## TF-IDF: $\text{idf}(t, d)$

TF-IDF is calculated as :  $\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \boxed{\text{idf}(t, D)}$

### Inverse document frequency $\text{idf}(t, D)$

$$\text{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

With  $N$  the number of documents in corpus  $D$

# TF-IDF: $\text{idf}(t,d)$

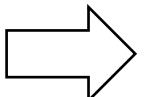
Inverse document frequency  $\text{idf}(t,D)$  variants :

Variants of inverse document frequency (idf) weight

weighting scheme	idf weight ( $n_t =  \{d \in D : t \in d\} $ )
unary	1
inverse document frequency	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
inverse document frequency smooth	$\log \left( \frac{N}{1 + n_t} \right) + 1$
inverse document frequency max	$\log \left( \frac{\max_{\{t' \in d\}} n_{t'}}{1 + n_t} \right)$
probabilistic inverse document frequency	$\log \frac{N - n_t}{n_t}$

# TF-IDF

**TF-IDF** is calculated as :  $\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D)$



$$\frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \cdot \log \frac{N}{|\{d \in D : t \in d\}|}$$

# TF-IDF

*Document1:* “Xavier likes to play football. Eric likes football too.”

*Document2:* “Eric prefers tennis to football.”

	Xavier	likes	to	play	football	Eric	too	prefers	tennis
Document1	1	2	1	1	2	1	1	0	0
Document2	0	0	1	0	1	1	0	1	1
idf	Log(2)	Log(2)	0	Log(2)	0	0	Log(2)	Log(2)	Log(2)
tf_doc1	1/9	2/9	1/9	1/9	2/9	1/9	1/9	0	0
tf_doc2	0	0	1/5	0	1/5	1/5	0	1/5	1/5
tf-idf_doc1	0,033	0,067	0	0,033	0	0	0,033	0	0
tf-idf_doc2	0	0	0	0	0	0	0	0,06	0,06

# TF-IDF

*Document1:* “Xavier likes to play football. Eric likes football too.”

*Document2:* “Eric prefers tennis to football.”

	Xavier	likes	to	play	football	Eric	too	prefers	tennis
Document1	1	2	1	1	2	1	1	0	0
Document2	0	0	1	0	1	1	0	1	1
idf	Log(2)	Log(2)	0	Log(2)	0	0	Log(2)	Log(2)	Log(2)
tf_doc1	1/9	2/9	1/9	1/9	2/9	1/9	1/9	0	0
tf_doc2	0	0	1/5	0	1/5	1/5	0	1/5	1/5
tf-idf_doc1	0,033	0,067	0	0,033	0	0	0,033	0	0
tf-idf_doc2	0	0	0	0	0	0	0	0,06	0,06

Again, the documents  
are transformed  
into  
numerical  
representation

# TF-IDF: Exercise

*course1\_tfidf\_ex.ipynb*

**Goal:** Illustration of TF-IDF on a very simple corpus

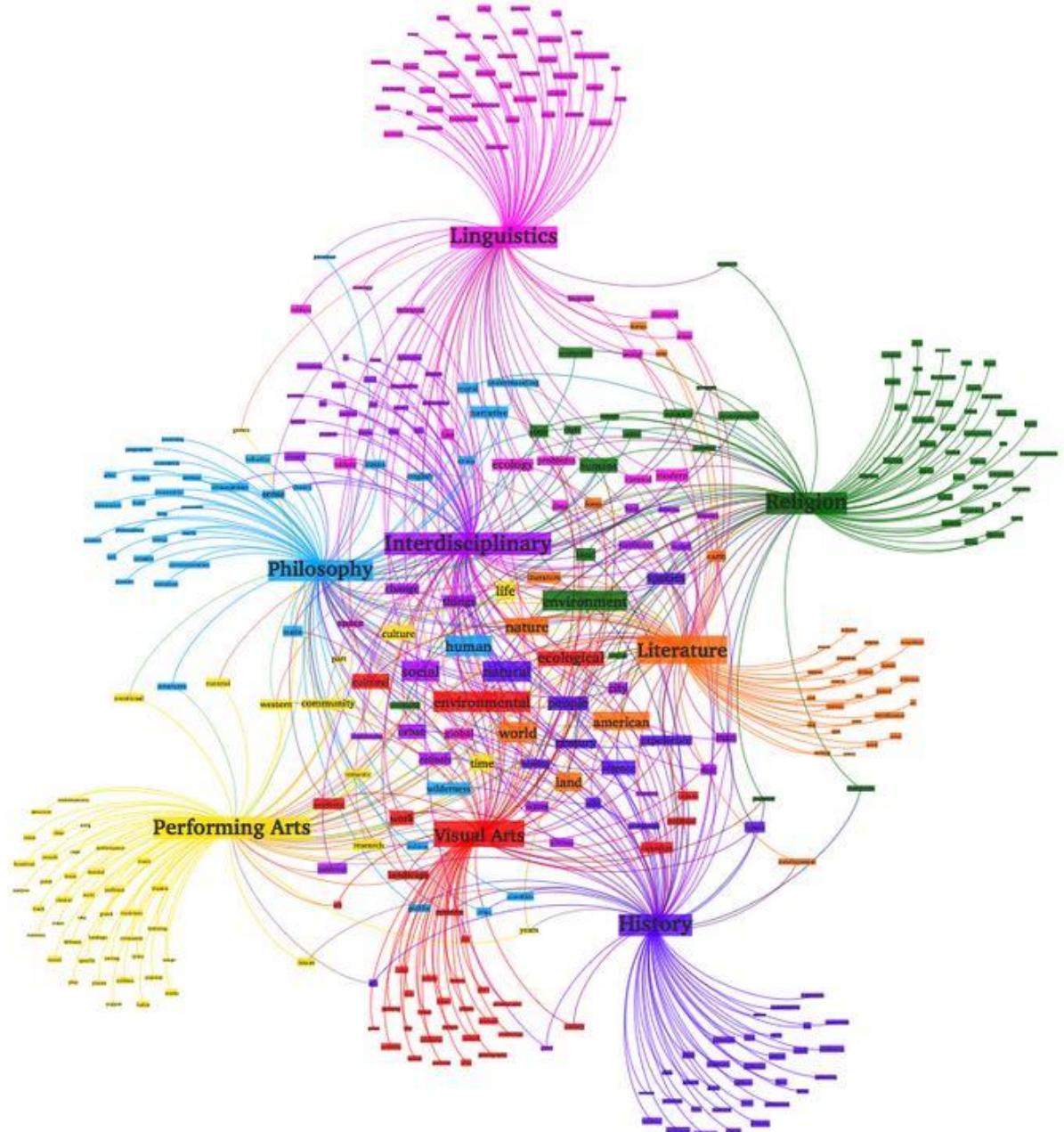
## Remarks:

- We compute the TF-IDF twice on the same corpus (directly from the formula and with sklearn library)
- The results may be different depending on the settings used with the sklearn function

# Topic Modeling

# Topic modeling

Clustering document into topics



# Topic modeling

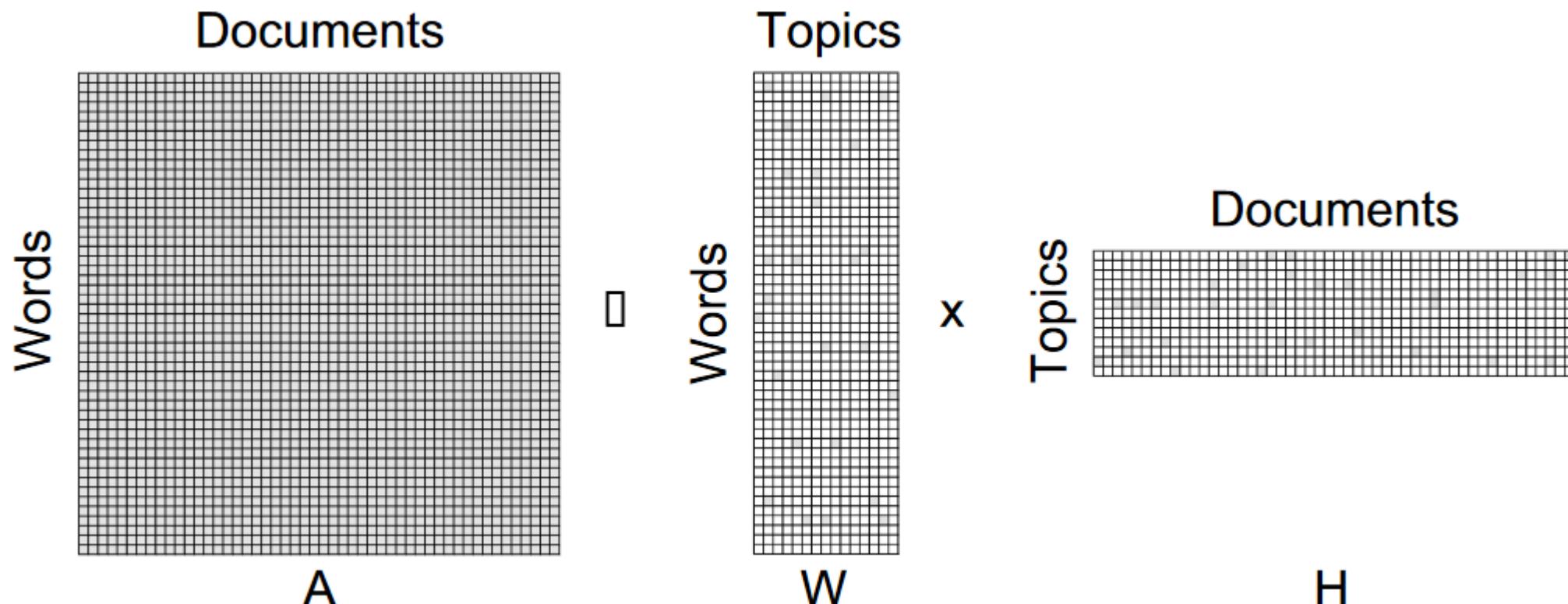
- Unsupervised algorithms
- Use to analyze text by clustering document into topics
- The idea is for the documents in the same cluster to correspond to a common topic
- Can be used to analyze large volumes of text

# Non-negative Matrix factorization (NMF)

- Non-negative Matrix Factorization is an unsupervised topic modeling algorithm
- The number of clusters must be defined by the user
- It allows both clustering and dimensionality reduction
- Generally, it is used from a **TF-IDF procedure** performed on all the documents of the corpus

# Non-negative Matrix factorization (NMF)

The procedure involves the decomposition of matrix A in the product of two other matrices W and H. All three matrices are non-negative



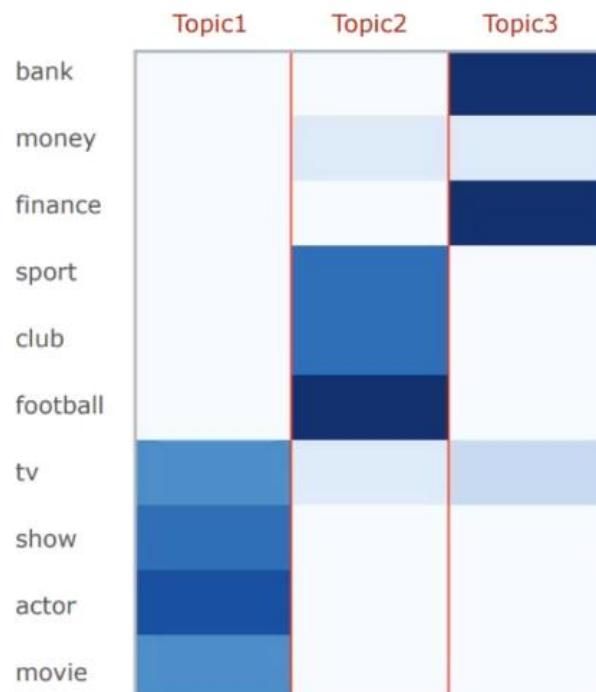
# Non-negative Matrix factorization (NMF)

The A' matrix comes from a TF-IDF vectorization

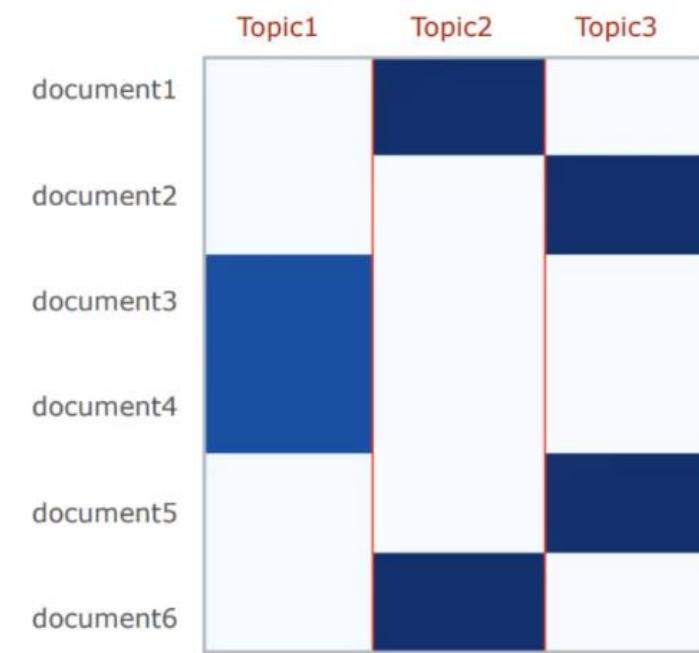


# Non-negative Matrix factorization (NMF)

*Basis vectors  $\mathbf{W}$ : topics  
(clusters)*



*Coefficients  $\mathbf{H}^T$ : memberships  
for documents*



# Non-negative Matrix factorization (NMF)

## Procedure

- **Inputs:** Matrix A, a number  $k$ , matrices W and H randomly generated
- **Objective function:** the reconstruction error between A and WH

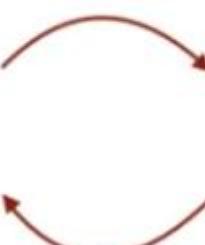
$$\frac{1}{2} \|\mathbf{A} - \mathbf{WH}\|_F^2 = \sum_{i=1}^n \sum_{j=1}^m (A_{ij} - (WH)_{ij})^2$$

- The computation of W and H are performed sequentially with an **Expected-Maximization (EM)** procedure until convergence

1. Update **H**

$$H_{cj} \leftarrow H_{cj} \frac{(W\mathbf{A})_{cj}}{(W\mathbf{WH})_{cj}}$$

2. Update **W**

$$W_{ic} \leftarrow W_{ic} \frac{(\mathbf{AH})_{ic}}{(\mathbf{WHH})_{ic}}$$


# Latent Dirichlet Allocation (LDA)

- Dirichlet was a German mathematician (1805-1859)
- The Dirichlet Distribution was named after him
- LDA is a statistical model based on this specific distribution
- It was first presented in 2002 (Blei, Ng and Jordan) for document topic detection
- Since then, this model has been used in many applications (data mining, NLP..)

# Latent Dirichlet Allocation (LDA)

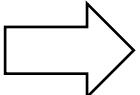
## LDA principles

- Documents with similar topics use similar groups of words
- Latent topics correspond to groups of word frequently occurring together in the corpus documents

# Latent Dirichlet Allocation (LDA)

## LDA principles

- Documents with similar topics use similar groups of words
- Latent topics correspond to groups of word frequently occurring together in the corpus documents

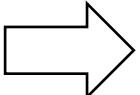


- Each document is a mixture of a small number of topics
- Each word's presence is attributable to a topic (considered here as a probability distribution over words)

# Latent Dirichlet Allocation (LDA)

## LDA principles

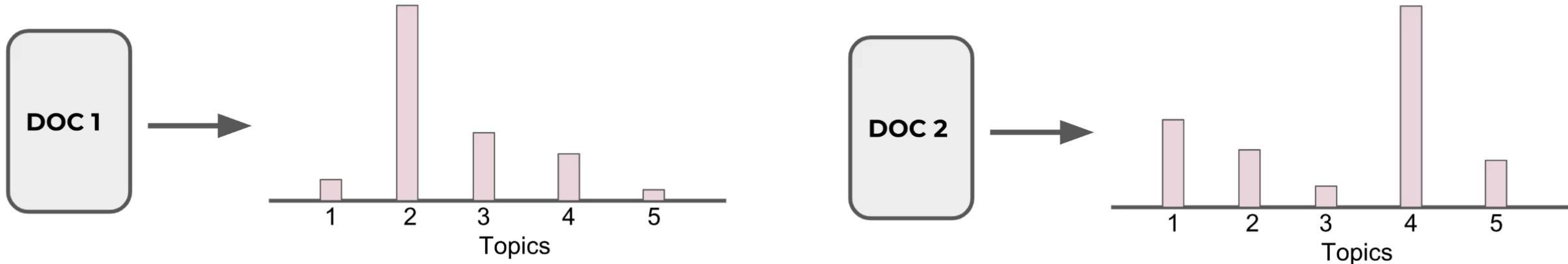
- Documents with similar topics use similar groups of words
- Latent topics correspond to groups of word frequently occurring together in the corpus documents



- **Each document is a mixture of a small number of topics**
- Each word's presence is attributable to a topic (considered here as a probability distribution over words)

# Latent Dirichlet Allocation (LDA)

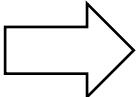
Each document is a mixture of a small number of topics



# Latent Dirichlet Allocation (LDA)

## LDA principles

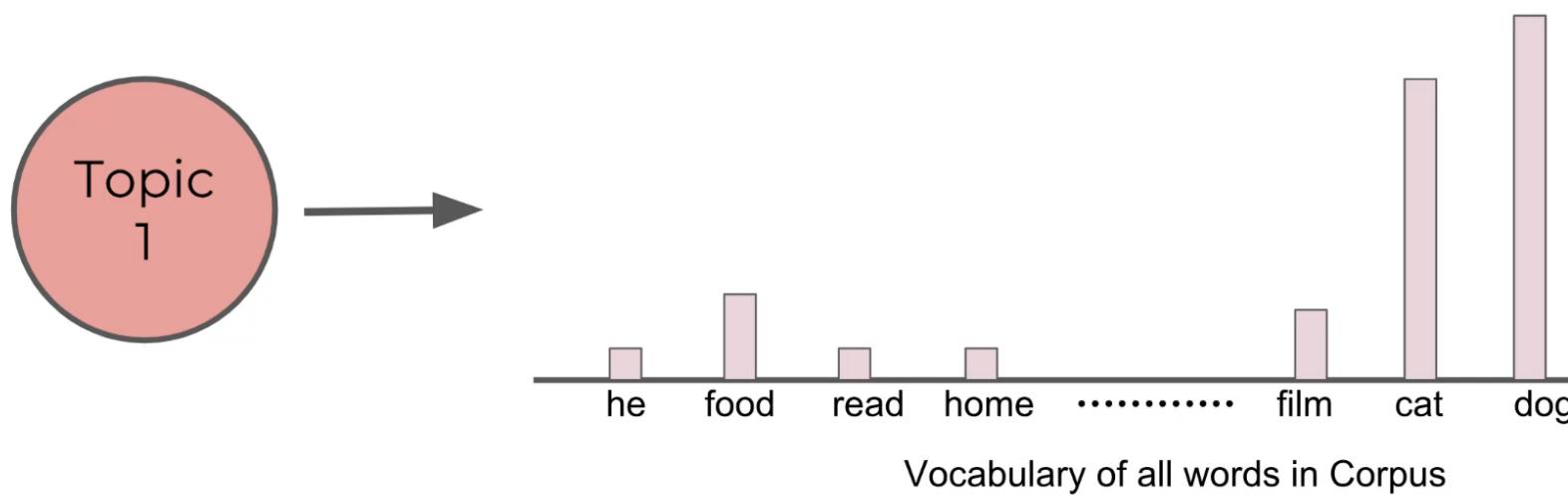
- Documents with similar topics use similar groups of words
- Latent topics correspond to groups of word frequently occurring together in the corpus documents



- Each document is a mixture of a small number of topics
- **Each word's presence is attributable to a topic (considered here as a probability distribution over words)**

# Latent Dirichlet Allocation (LDA)

Each word's presence is attributable to a topic (considered here as a probability distribution over words)



# Latent Dirichlet Allocation (LDA)

A document can be seen as

- A chosen N number of words
- A specific mixture of topic (selected according to a Dirichlet distribution over the set of topics)-e.g.: 50% cinema, 30% sport, 20% animals

Each word in the document are generated by the following procedure

- Choose a topic (according to the topics multinomial distribution) and generate from it a word

This is the generative model used to create a set of documents

# Latent Dirichlet Allocation (LDA)

Let's go back to the topic modelling objective

- We have a set of documents
- We chose a number  $k$  of topics to discover
- We use LDA for topic modelling, and to determine the topic representation of each document and the words associated to each topic

# Latent Dirichlet Allocation (LDA)

## Procedure

1. Randomly assign, according to a Dirichlet distribution, each word of each document to one of the  $k$  topics
2. You have a first (but meaningless) representation of the word and document distribution of the topics
3. We iterate over every word and every document in the following way. For each topic  $t$ , document  $d$  and word  $w$  we calculate:  
 $P(t/d)$  = proportion of words in  $d$  currently assigned to  $t$   
 $P(w/t)$  = proportion of assignments to  $t$  over all documents containing  $w$
4. Reassign each word  $w$  a new topic  $t$ , according the probability  $p(w/t) \cdot p(t/d)$
5. Repeat the previous steps enough times to converge to acceptable topic assignments

# Latent Dirichlet Allocation (LDA)

- In the end of the procedure, we can assign document to topics
- However, we do not have relevant labels to associate topics with
- It is up to the user, in a next step, to find a way to identify theses topics found

# LDA & NLM: Exercise

*course1\_LDA\_NMF\_ex.ipynb*

**Goal:** Learn how to use **LDA** and **NMF** to address **Topic Modeling** on a **real dataset**

## Remarks:

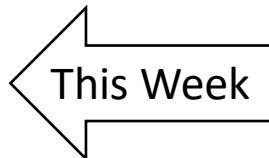
- We use *scikit-learn* implementation for **LDA** and **NMF**
- The dataset is composed of **real** questions taken from the internet forum **Quora**

# Take-away from Course 1

- One of the main issues of NLP is finding a **relevant numerical representation for texts**
- Before converting texts into numerical vectors, some **preprocessing steps** (tokenization, lemmatization, stemming, stop word removal...) exist and can prove **very useful**
- Bag-of-words (BoW) and TF-IDF can transform a corpus of texts into an **array of numerical vectors**
- These numerical vectors **can be used as input data** for classical ML procedures (e.g., topic modeling with LDA and NMF algorithms which rely respectively on BoW and TF-IDF)

# Course Schedule

- **Course 1:** NLP introduction
- **Course 2: Word embedding**
  - Word embedding introduction
  - Cosine similarity distance
  - Text embedding
  - Sentiment analysis
- **Course 3:** Long Short-Term Memory (LSTM) architecture
- **Course 4:** “Attention” mechanism and Transformer architectures
- **Course 5:** Chatbot implementation



# Course 2: Word embedding

# Word embedding introduction

# Tokenization and one-hot encoding steps

## Principle

*To be or not to be*

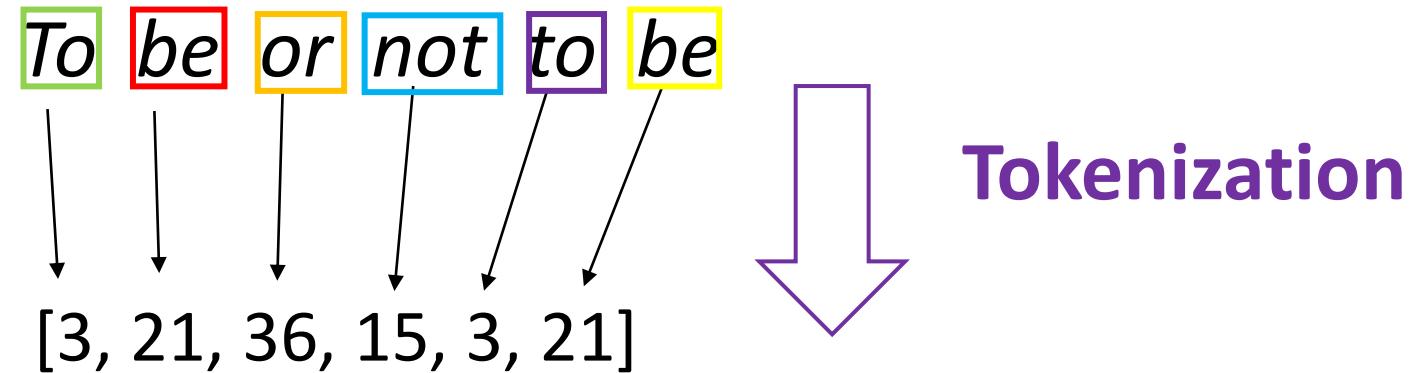
# Tokenization and one-hot encoding steps

## Principle

*To be or not to be*      Tokenization

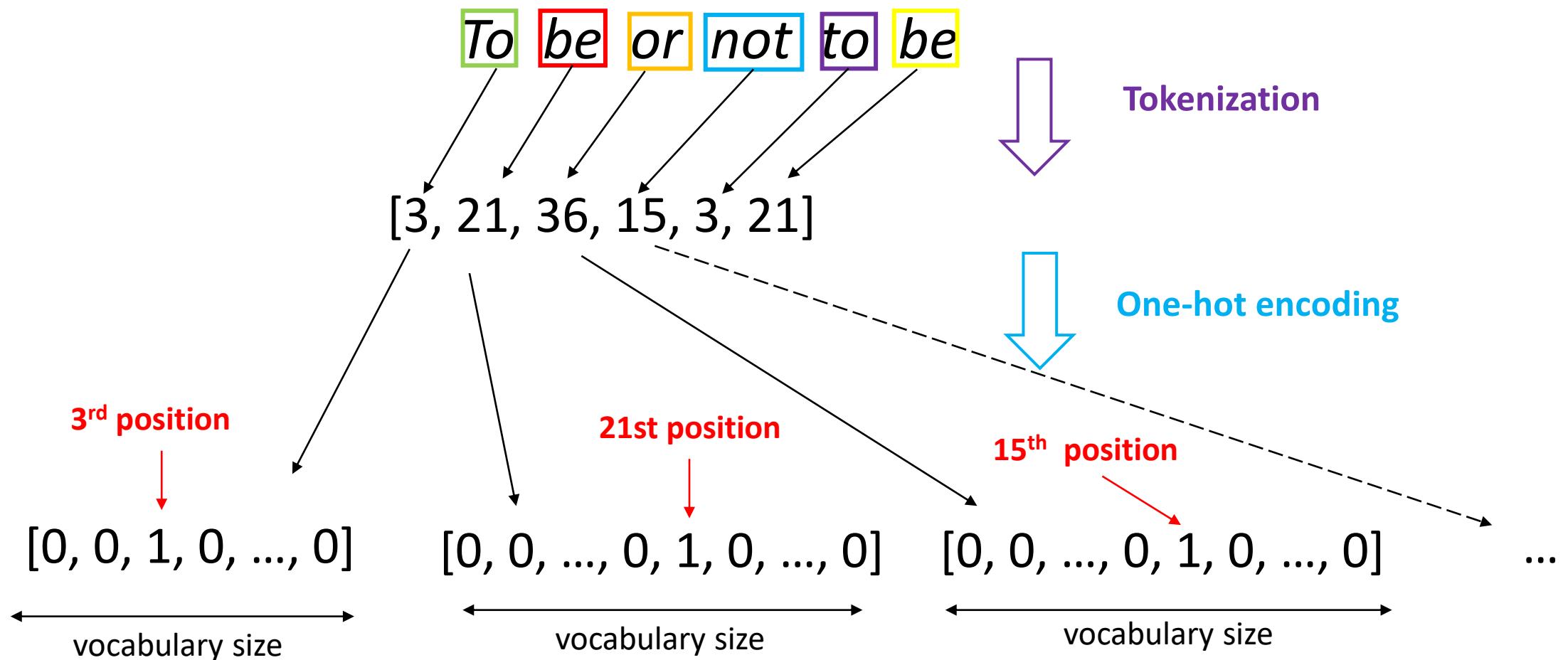
# Tokenization and one-hot encoding steps

## Principle



Each token is associated to its **number** in the **vocabulary** considered

# Tokenization and one-hot encoding steps

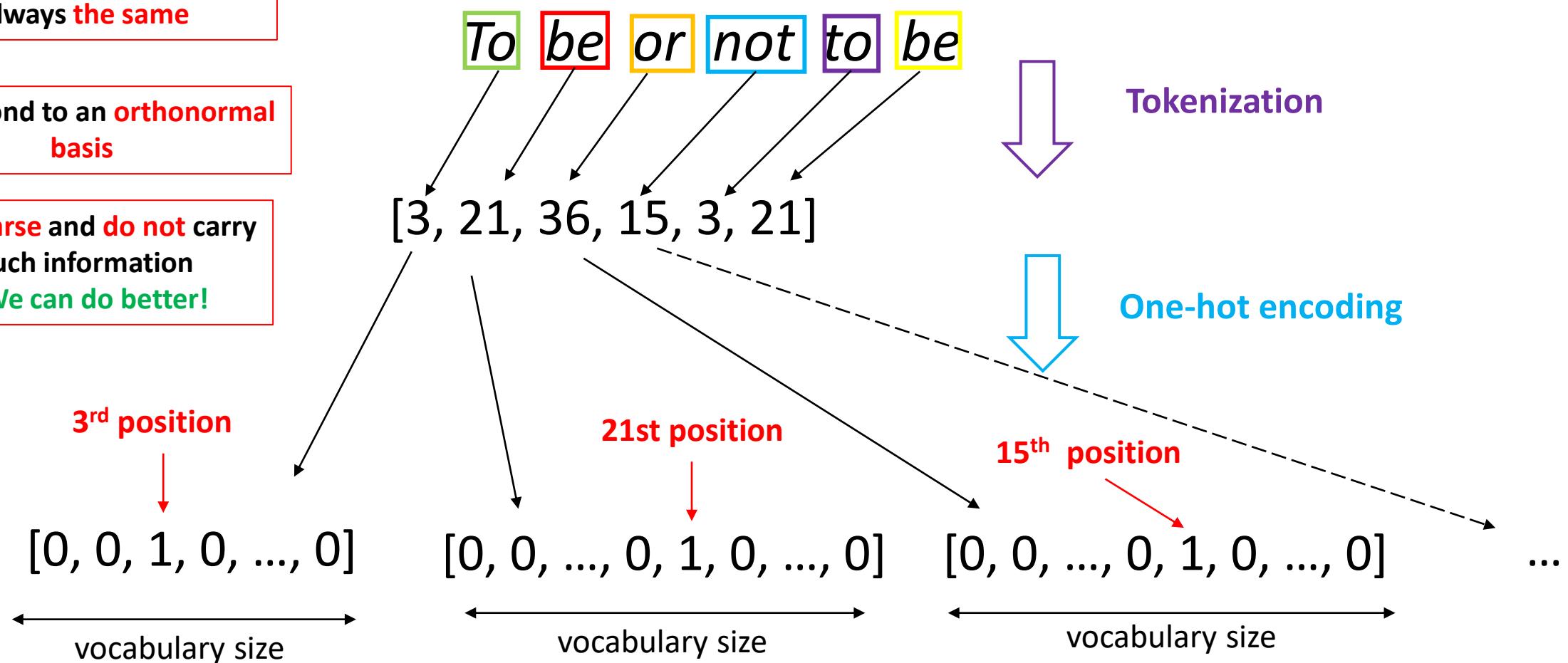


# Tokenization and one-hot encoding steps

The **distance** between two different one-hot vectors is always **the same**

Correspond to an **orthonormal basis**

Very **sparse** and do not carry much information  
=> **We can do better!**



# Word embeddings

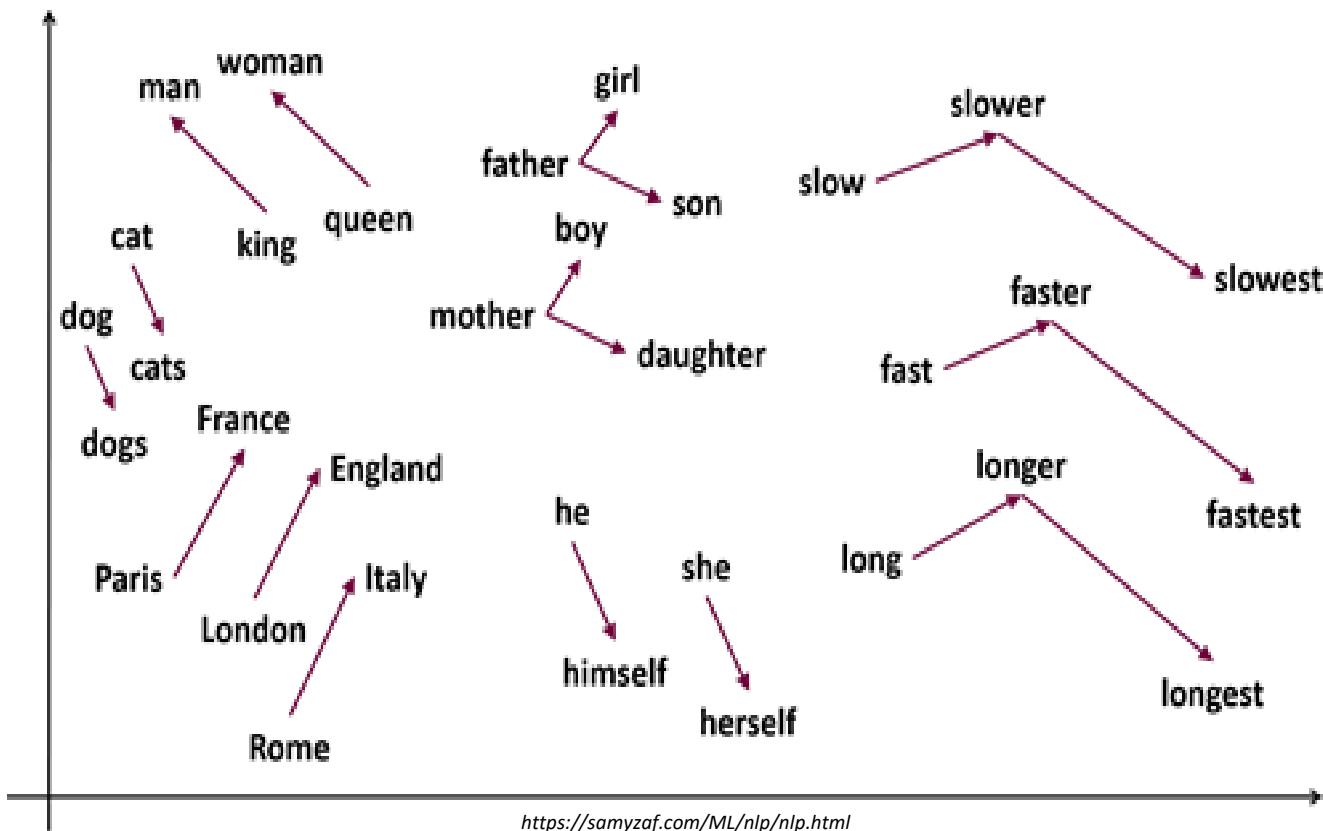
## Principles

- To get numerical vectors as representation of words
- **Goal** : two words with closed meaning should be represented by closed vectors as well
- This is a different approach from the classical one *hot encoding*, no need to consider vectors with a size equals to the number of words in the vocabulary

Theoretically, a word embedding should consider a vector space in which that kind of relationship between vectors should be verified

$$\mathbf{king} - \mathbf{man} + \mathbf{woman} \approx \mathbf{queen}$$

# Word embedding: Illustration



king – man + woman ≈ queen

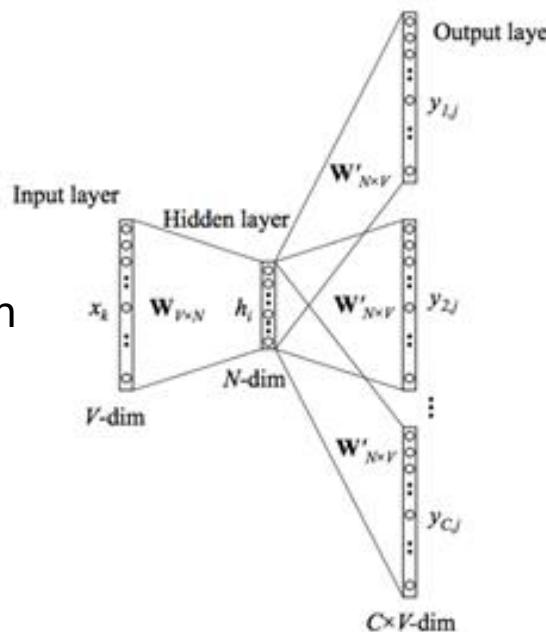
France – Paris + London ≈ England

# Word embeddings: Principle

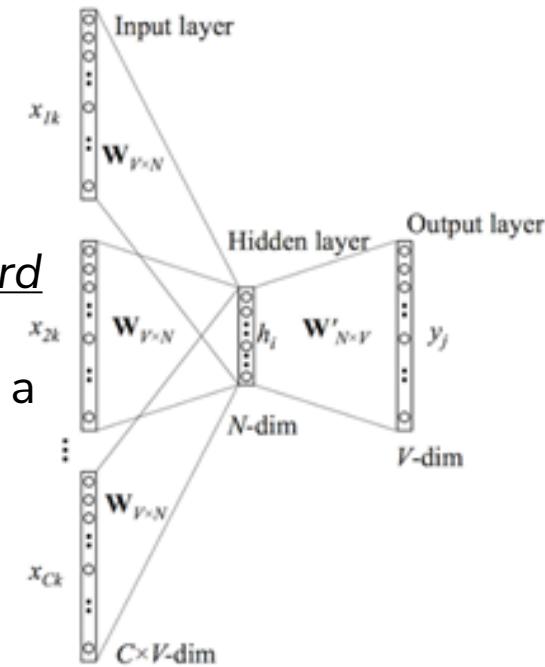
How are generated word embeddings?

- By training a neural network on **huge** corpus
- **Two main approaches** are generally used for that

Skip-gram model:  
Context prediction from  
a word



Continuous Bag of Word (CBOW) model:  
Word prediction from a context



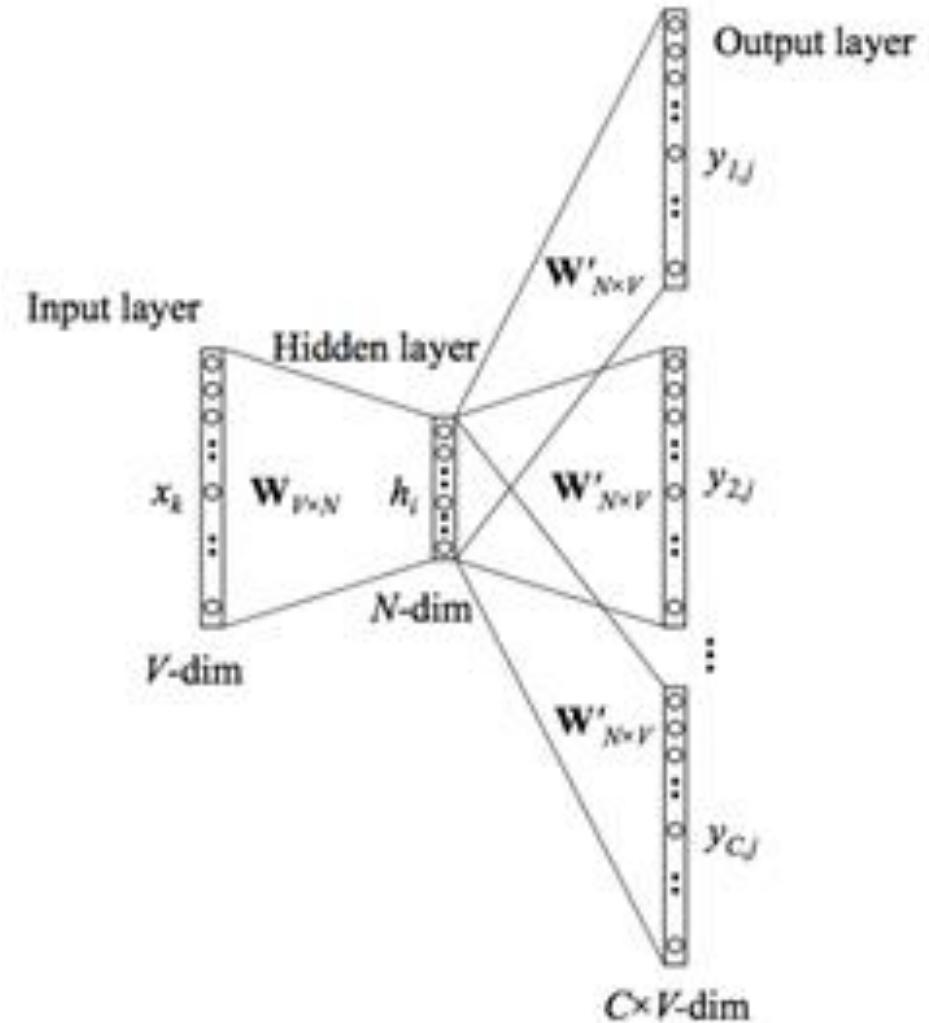
# Word embedding: Skip-gram model

Train to predict the **context** from a given **word**

## Example:

*The cat **sat** on the mat*

The word **sat** is given as an input and we try to predict **cat** and **mat** at position -1 and 3 (stop words are generally not predicted)



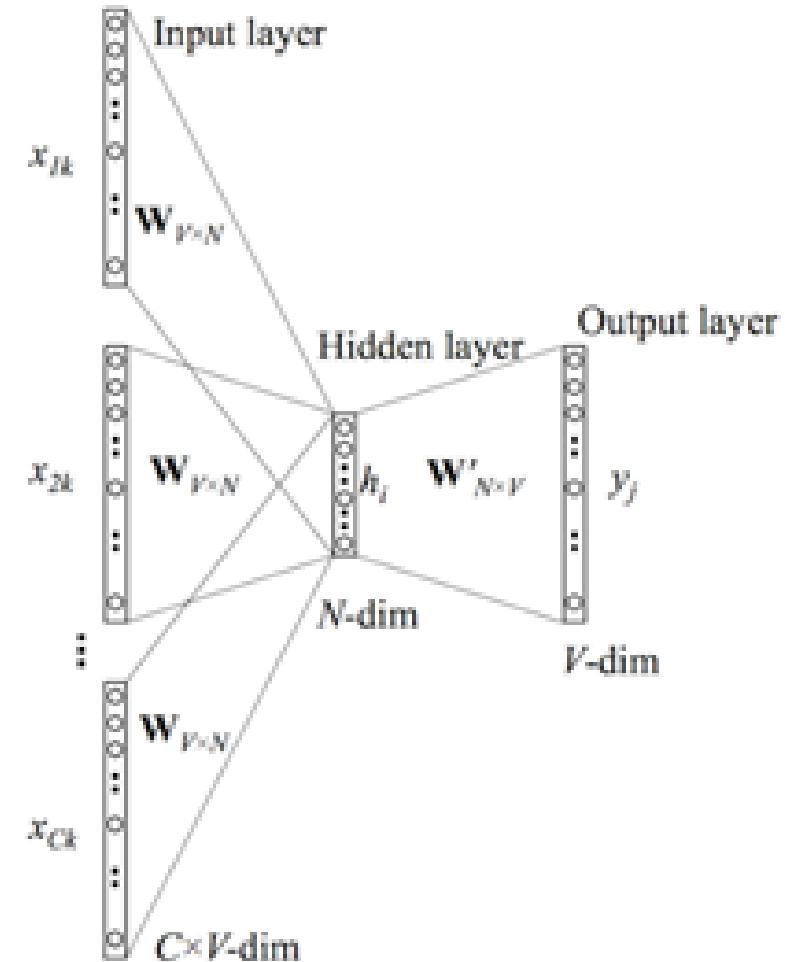
# Word embedding: CBOW model

Train to predict a **word** from  
a given **context**

## Example:

*The cat sat on the mat*

The words [*The*] [*cat*] [*on*] [*the*] [*mat*]  
are given as inputs and we try  
to predict the word *sat*



# Embedding: word2vec

- **word2vec** is a tool providing an **efficient** implementation for word embedding generation
- It allows you to choose between the **two main** algorithms
  - The continuous bag-of-words (CBOW) model
  - The skip-gram model
- It needs a (**huge**) text corpus as input in order to produce an efficient word embedding model as output
- The learning step can be **avoided** in loading pre-trained models
  - **Google News** model has been trained on about 100 billion (!) words and gives a 300-dimension embedding

# Embedding: spaCy

- **spaCy** allows to load models with **pre-trained embeddings**
- **For example**, both '*en\_core\_web\_md*' or '*en\_core\_web\_lg*' models (respectively *medium* and *large*) contain **300-dimension vector** for each **token** in the vocabulary
- E.g.,

```
nlp = spacy.load('en_core_web_md')
nlp.vocab['king'].vector
```

gives the **300-dimension vector** representation of the token “king”

# Embedding: fastText

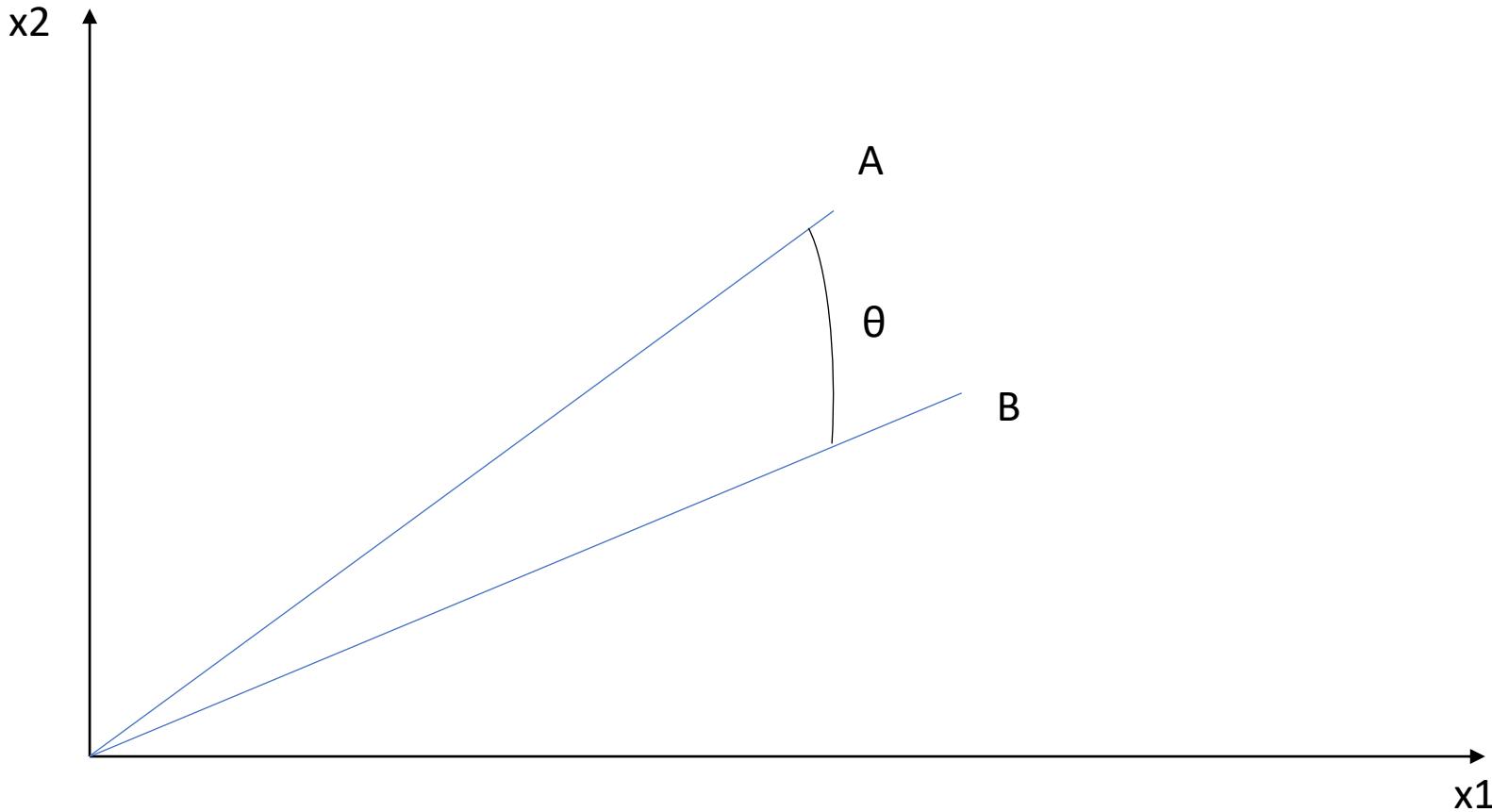
- **fastText** is an open-source library allowing to use **pre-trained word embeddings** (and text classifiers) for almost 300 languages
- It has been developed by Facebook's AI Research (FAIR) lab
- Unlike most of other embeddings, dependent of a vocabulary, fastText treats each word as composed of **N-grams** (subsequences of characters)
  - ➡ **fastText can generate vectors for word not even in the training corpus**
- This gives some **advantages** of *fastText* over more classical embedding models
- However, a drawback of this model is a **high memory requirement** to load and use it

# Cosine similarity distance

# Cosine similarity distance

- The **cosine similarity** is a measure of similarity between two vectors which relies on the **cosine of their angle**
- This is the distance generally used to compare **two documents**
- Most of the time, the text embeddings have very **high dimensions**, so all texts are far from each others with a Euclidean distance
- To consider a distance **based only on the angle is more relevant** to determine similarity between vectors in that kind of space

# Cosine similarity distance



# Cosine similarity distance

For two vectors A and B, the cosine similarity distance is the value

$$D(A,B) = 1 - SC(A,B)$$

with **SC(A,B)** the **cosine similarity**

$$\frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

The **smallest** the angle between A and B is, the **closest** they are.

# Embedding: Exercise

*Course2\_embedding\_illustration\_ex.ipynb*

**Goal:** Check the validity of the relationship:  $king - man + woman \approx queen$   
from three libraries with pre-trained embedding models:

- *spaCy*
- *Glove*
- *fastText*

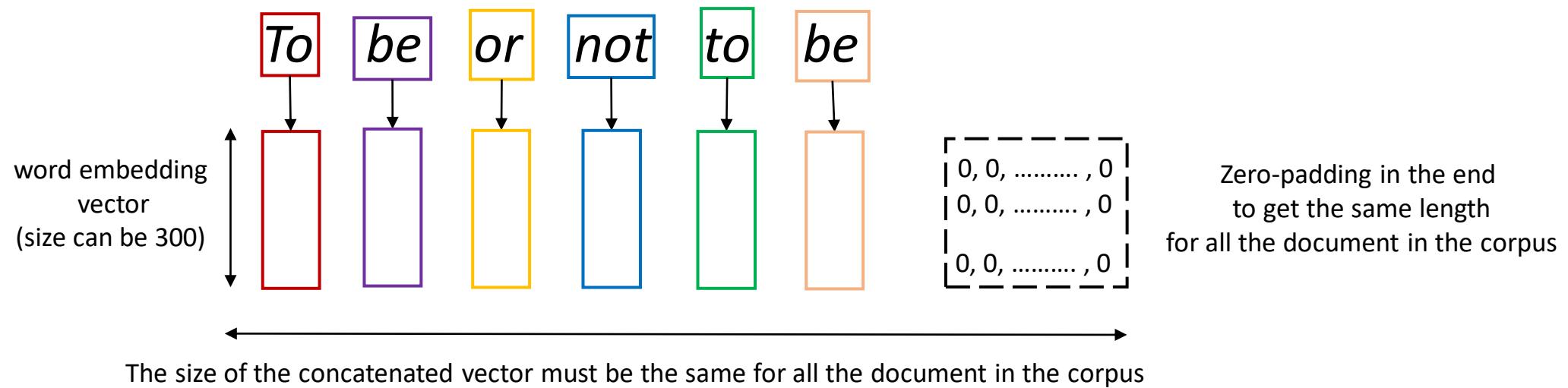
# Text embedding

# Embedding: Text embedding

- Word embedding allows to transform **tokens** into numerical representation
- For some applications, it may be useful to get numerical representation for **entire texts**
- **Different solutions** can be used in order to get that result

# Embedding: Sequence embedding

**One solution is to concatenate the embedding vectors of all the tokens of the text**

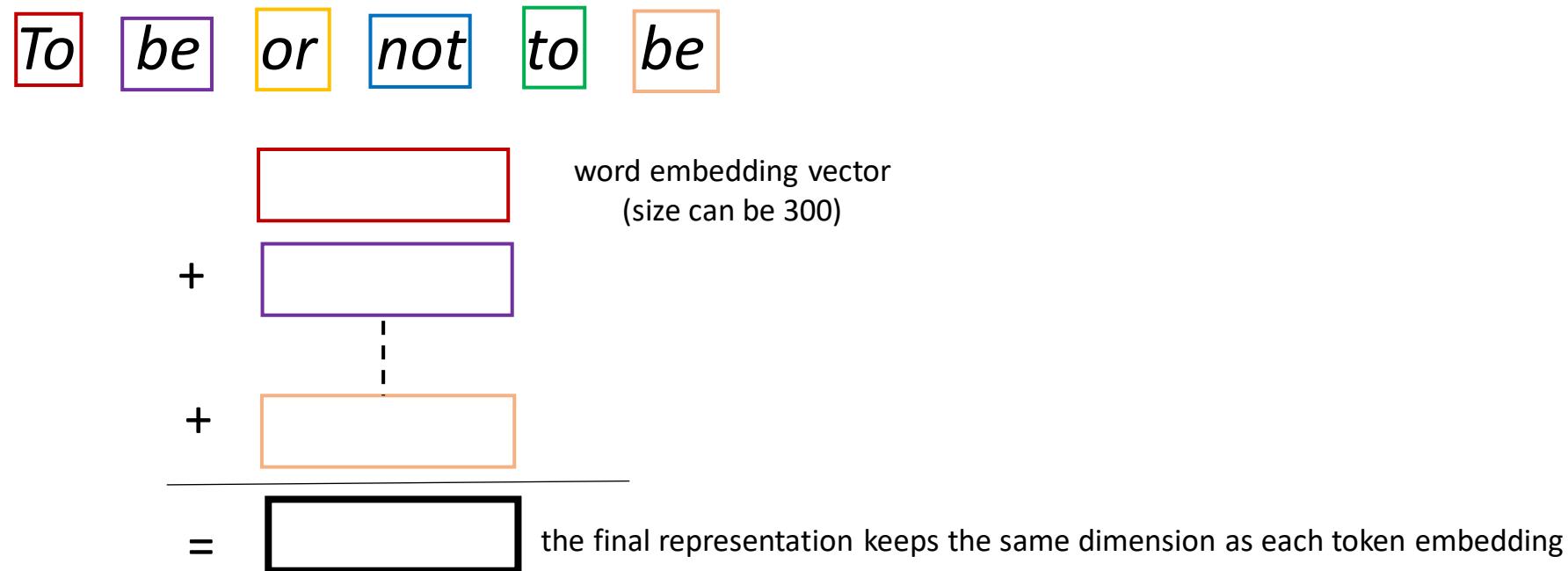


## Remarks

- to keep **the same dimensions** for each text of a corpus, there is a need to **truncate** the number of tokens or to **add padding** (depending on the number of tokens)
- The dimension of each text representation can be very **high** (e.g., several thousands)

# Embedding: Text embedding

**Another** solution is to average the embedding vectors of all the tokens in the text



## Remarks

- There is **no need** to use **padding** or **truncation** to keep the same dimension
- The dimension of the text embedding remains quite **low**
- There is a significant **loss of information** in comparison with the previous method

# Embedding: Text embedding

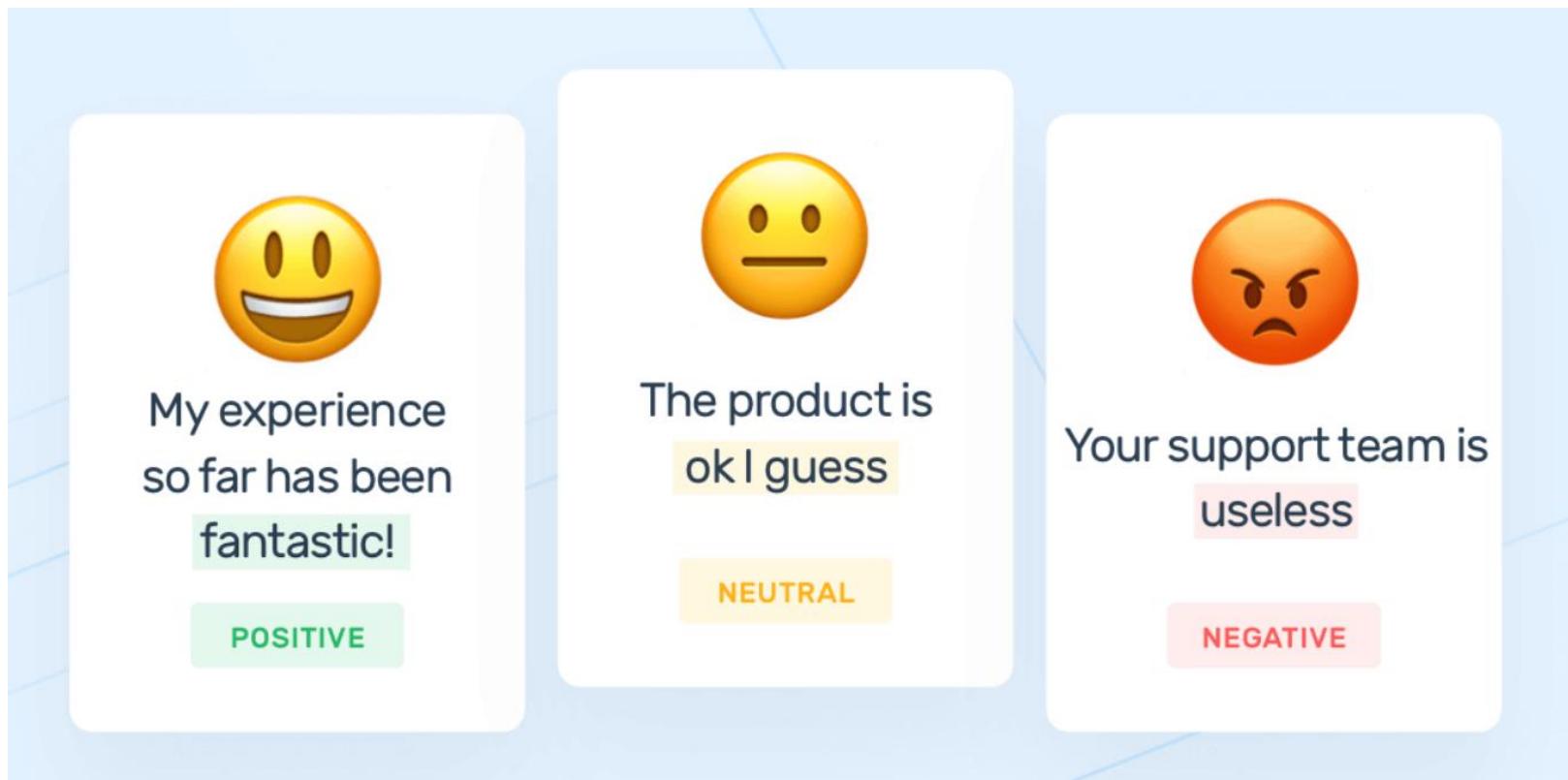
Some libraries offers to deal directly with **text** embeddings

- **spaCy** can compute directly text embedding (with averaging method)
- The model ***doc2vec***, based on the ***word2vec*** logic, allows to build text embedding
- Some models directly available from ***tensorFlow Hub***

# Sentiment Analysis

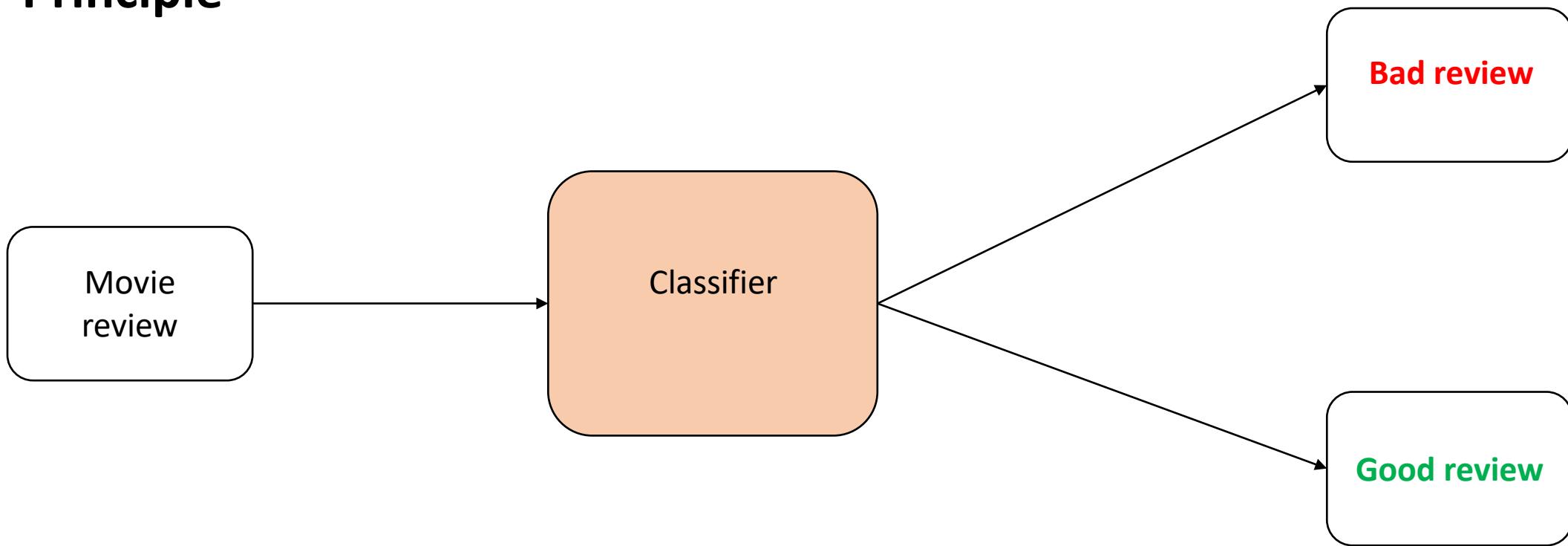
# Sentiment Analysis

What is sentiment analysis ?



# Sentiment Analysis

## Principle



The **IMDB dataset**, from the famous internet site of the same name and containing numerous **real movie reviews**, is a very useful and famous dataset to train **sentiment analysis** classifier

# Sentiment Analysis: Principle

- Sentiment analysis is a specific task consisting in characterizing the **presence of a sentiment** inside a text
- The difficulty may depend on the sentiment searched for
- The most famous example of sentiment analysis consists in categorizing a text into **positive** (“love”) or **negative** (“hate”) feeling
- This kind of positive-negative application can make sense for reviews (movies...), reactions to a specific event (Twitter...)

# Sentiment Analysis: Embeddings

To train a sentiment analysis classifier, **numerical representation** of texts are **needed** as inputs

**Text embeddings** are generally **good** options as such inputs

- All **supervised classification** algorithms can be used
- **Deep learning** and **neural network** can be good options as well

# Embedding: Exercise

*Course2\_spacy\_svm\_ex.ipynb*

**Goal:** Use of spaCy **text embedding** model to train a Support Vector Machine (**SVM**) for a **sentiment analysis** application on **IMDB dataset**

## Remarks:

- The computation of text embedding can be **time consuming**
- To avoid this issue, the notebook allows you to load text embeddings **already computed**

# Sentiment Analysis: Vader model

- Some **pre-trained** sentiment analysis **models** are available
- One of them is **Vader** (Valence Aware Dictionary for sEntiment Reasoning) and can be found in the **NLTK** package
- Vader is a model allowing positive/negative sentiment text **classification** and characterize also the **intensity** of the emotion
- It is specifically train to analyze **social media text** (e.g., it can analyze smileys)

```
# Create a SentimentIntensityAnalyzer object.  
sid_obj = SentimentIntensityAnalyzer()  
  
sid_obj.polarity_scores(":")  
{'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound': 0.4588}  
  
sid_obj.polarity_scores(":(")  
{'neg': 1.0, 'neu': 0.0, 'pos': 0.0, 'compound': -0.4404}
```

# Embedding: Exercise

*Course2\_sentiment\_analysis\_vader\_ex.ipynb*

**Goal:** Use of Vader model for **sentiment analysis** application on **IMDB dataset**

## Remarks:

- The Vader model is pre-trained and be directly used without training
- This model was pre-trained specifically for sentiment analysis

# Sentiment Analysis: Neural Network (NN)

## Implementing a neural network with **Keras**

- Keras provides functions to load some common datasets such as **IMDB dataset**
- A NN can be created using Keras Sequential API
- NN can be created by adding **layers** to a model
- **Deep Learning** corresponds to chaining together several layers in order to build complex structures

# Sentiment Analysis: Neural Network (NN)

## Data Preprocessing

```
import tensorflow_datasets as tfds

datasets, info = tfds.load("imdb_reviews", as_supervised=True, with_info=True)

train_size = info.splits["train"].num_examples
batch_size = 32

train_set = datasets["train"].shuffle(10000).repeat().batch(batch_size).prefetch(tf.data.AUTOTUNE)

test_size = info.splits["test"].num_examples
test_set = datasets["test"].repeat().batch(batch_size).prefetch(tf.data.AUTOTUNE)
```

# Sentiment Analysis: Neural Network (NN)

## Data Preprocessing

The tensorflow\_datasets API allows to load easily common datasets  
(here *imdb\_reviews*)

```
import tensorflow_datasets as tfds

datasets, info = tfds.load("imdb_reviews", as_supervised=True, with_info=True)

train_size = info.splits["train"].num_examples
batch_size = 32

train_set = datasets["train"].shuffle(10000).repeat().batch(batch_size).prefetch(tf.data.AUTOTUNE)

test_size = info.splits["test"].num_examples
test_set = datasets["test"].repeat().batch(batch_size).prefetch(tf.data.AUTOTUNE)
```

# Sentiment Analysis: Neural Network (NN)

## Data Preprocessing

```
import tensorflow_datasets as tfds

datasets, info = tfds.load("imdb_reviews", as_supervised=True, with_info=True)

train_size = info.splits["train"].num_examples
batch_size = 32

train_set = datasets["train"].shuffle(10000).repeat().batch(batch_size).prefetch(tf.data.AUTOTUNE)
test_size = info.splits["test"].num_examples
test_set = datasets["test"].repeat().batch(batch_size).prefetch(tf.data.AUTOTUNE)
```

A split between “train” and “test” dataset is already done

# Sentiment Analysis: Neural Network (NN)

## Data Preprocessing

```
import tensorflow_datasets as tfds

datasets, info = tfds.load("imdb_reviews", as_supervised=True, with_info=True)

train_size = info.splits["train"].num_examples
batch_size = 32

train_set = datasets["train"].shuffle(10000).repeat().batch(batch_size).prefetch(tf.data.AUTOTUNE)

test_size = info.splits["test"].num_examples
test_set = datasets["test"].repeat().batch(batch_size).prefetch(tf.data.AUTOTUNE)
```

**Ensure the training data are well distributed**

# Sentiment Analysis: Neural Network (NN)

## Data Preprocessing

```
import tensorflow_datasets as tfds

datasets, info = tfds.load("imdb_reviews", as_supervised=True, with_info=True)

train_size = info.splits["train"].num_examples
batch_size = 32

train_set = datasets["train"].shuffle(10000).repeat().batch(batch_size).prefetch(tf.data.AUTOTUNE)

test_size = info.splits["test"].num_examples
test_set = datasets["test"].repeat().batch(batch_size).prefetch(tf.data.AUTOTUNE)
```

Used to repeat the initial dataset, possibly forever

# Sentiment Analysis: Neural Network (NN)

## Data Preprocessing

```
import tensorflow_datasets as tfds

datasets, info = tfds.load("imdb_reviews", as_supervised=True, with_info=True)

train_size = info.splits["train"].num_examples
batch_size = 32

train_set = datasets["train"].shuffle(10000).repeat().batch(batch_size).prefetch(tf.data.AUTOTUNE)

test_size = info.splits["test"].num_examples
test_set = datasets["test"].repeat().batch(batch_size).prefetch(tf.data.AUTOTUNE)
```

**Return a dataset consisting of groups of items**



# Sentiment Analysis: Neural Network (NN)

## Data Preprocessing

```
import tensorflow_datasets as tfds

datasets, info = tfds.load("imdb_reviews", as_supervised=True, with_info=True)

train_size = info.splits["train"].num_examples
batch_size = 32

train_set = datasets["train"].shuffle(10000).repeat().batch(batch_size).prefetch(tf.data.AUTOTUNE) Used to improve the performance by optimizing some data reading  
(setting to tf.data.AUTOTUNE allows an automatic dynamic choice)

test_size = info.splits["test"].num_examples
test_set = datasets["test"].repeat().batch(batch_size).prefetch(tf.data.AUTOTUNE)
```

# Embedding: Exercise

*Course2\_sentiment\_analysis\_nn\_training\_ex.ipynb*

*and/or*

*Course2\_sentiment\_analysis\_nn\_training\_spacy\_ex.ipynb*

**Goal:** train a neural network for **sentiment analysis** application on **IMDB dataset**

## Remarks:

- The first notebook **load IMDB dataset** and a **text embedding** model **directly from tensorflow API**
- The second notebook **train the network from spaCy text embedding computer before** (you can compare the results with those got from the SVM model from a previous exercise)
- If you have time, try to train both, but you can start with the **first one**

# Sentiment Analysis: Neural Network (NN)

- The use of tf.keras, instead of just Keras, allows for a better integration with other TensorFlow components
- We can create a model using the **Sequential API** of Keras by **adding layers to the stack**
- We must define the **number of layers**, the **kind of layers**, the **dimensions** and the **activation function**

```
model = tf.keras.models.Sequential([
    hub.KerasLayer(embed,
                  dtype=tf.string, input_shape=[], output_shape=[50]),
    Dense(128, activation="relu"),
    Dense(300, activation="relu"),
    Dense(1, activation="sigmoid")
])
```

# Sentiment Analysis: Neural Network (NN)

The summary method displays all model's information

- The ones defined previously
- The total number of trainable and non-trainable parameters

```
model.summary()

Model: "sequential"

Layer (type)          Output Shape         Param #
=====
keras_layer (KerasLayer)    (None, 50)        48190600
dense (Dense)           (None, 128)       6528
dense_1 (Dense)          (None, 300)       38700
dense_2 (Dense)          (None, 1)        301
=====
Total params: 48,236,129
Trainable params: 45,529
Non-trainable params: 48,190,600
```

# Sentiment Analysis: Neural Network (NN)

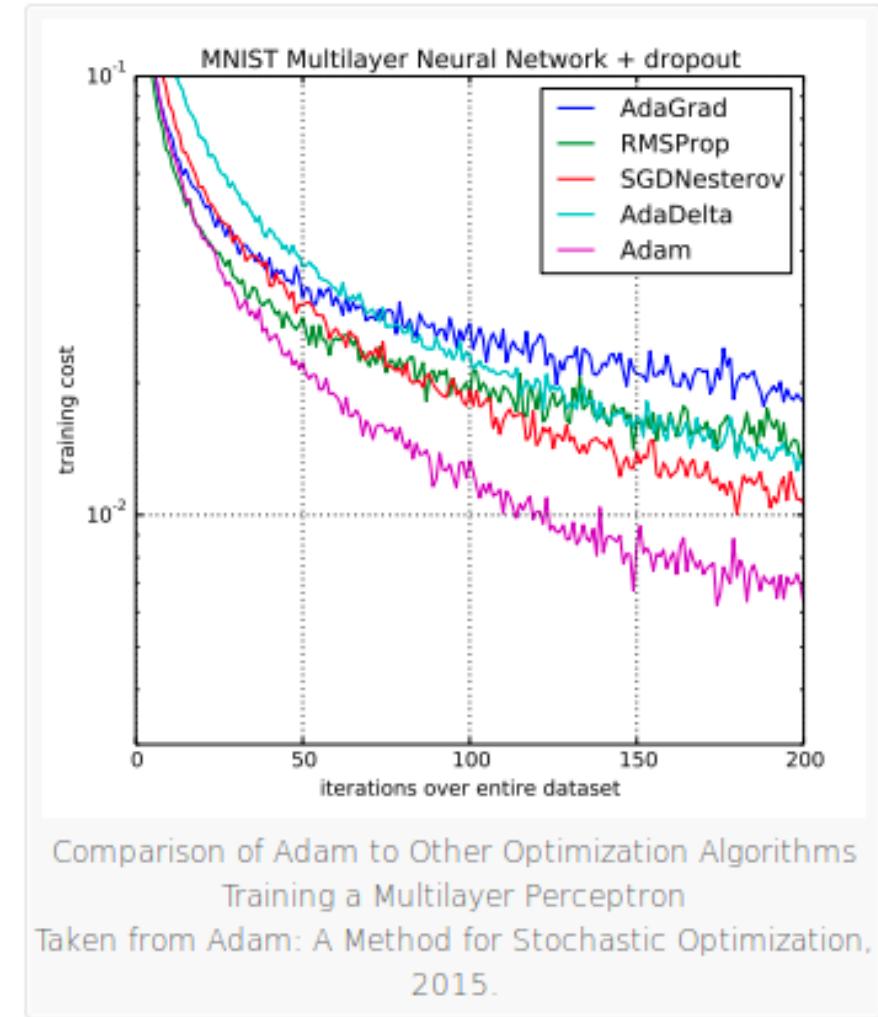
## Optimizers Variants

- Most optimizers are **implemented** in Tensorflow

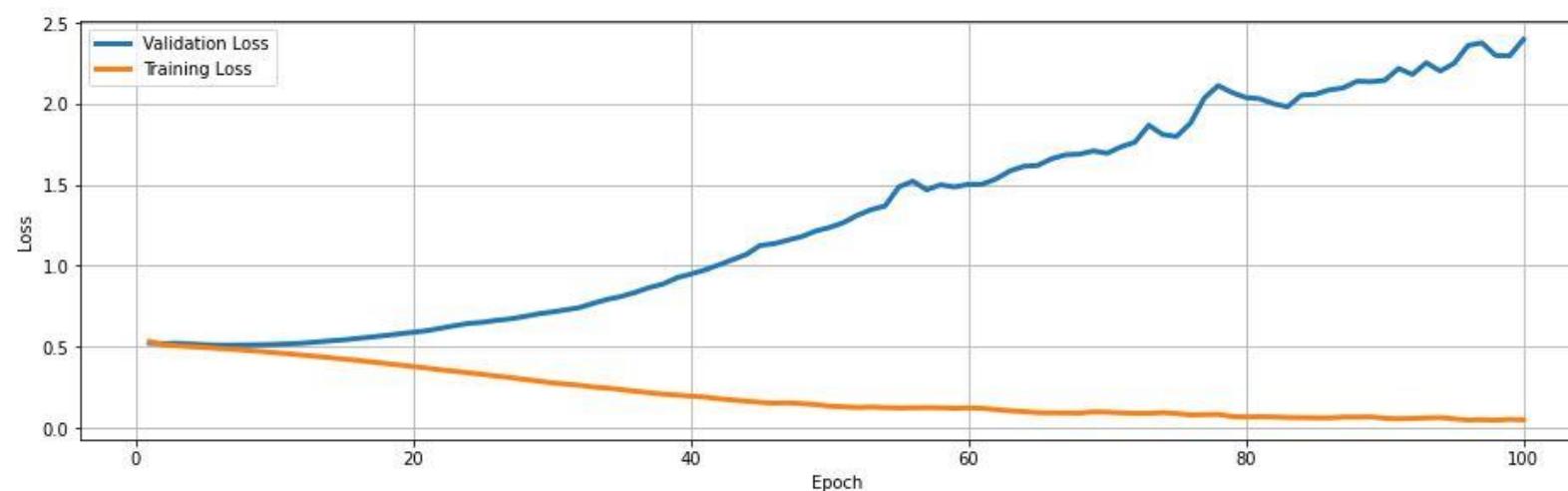
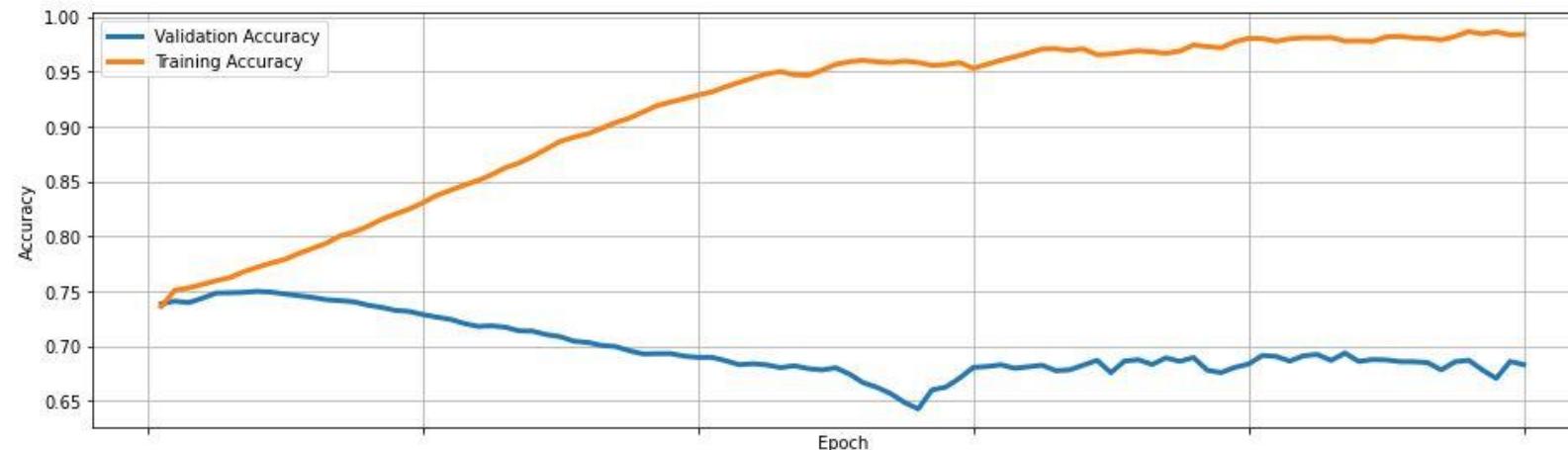
```
optimizer = tf.keras.optimizers.SGD(lr=0.001, momentum=0.9)
optimizer = keras.optimizers.SGD(lr=0.001, momentum=0.9, nesterov=True)
optimizer = tf.keras.optimizers.Adagrad(lr=0.001)
optimizer = tf.keras.optimizers.RMSprop(lr=0.001, rho=0.9)
optimizer = tf.keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999)
optimizer = tf.keras.optimizers.Nadam(lr=0.001, beta_1=0.9, beta_2=0.999)
```

- Both optimizer and loss function must be entered during the **compilation** step

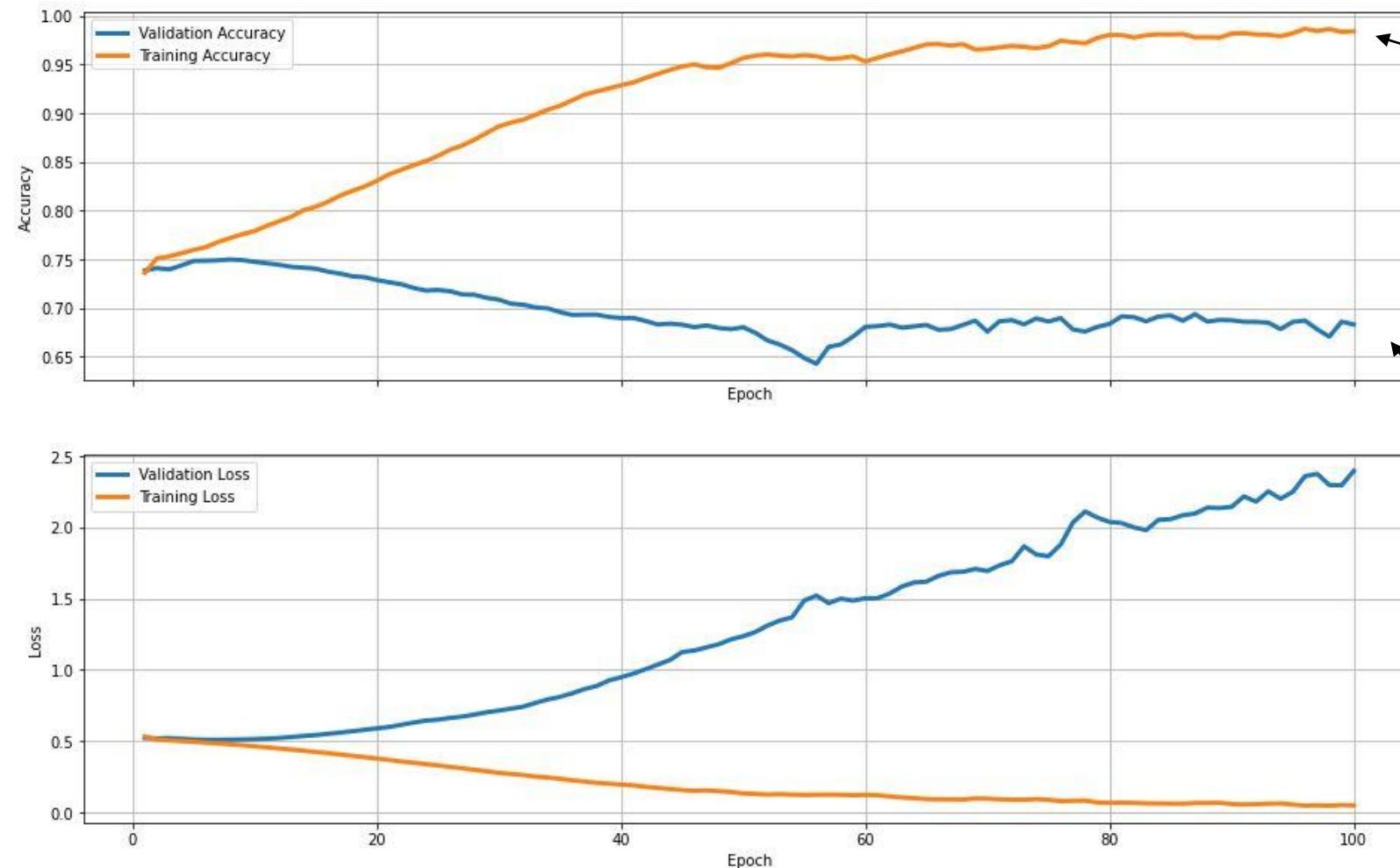
```
model.compile(loss="binary_crossentropy", optimizer=optimizer)
```



# Sentiment Analysis: Neural Network (NN)



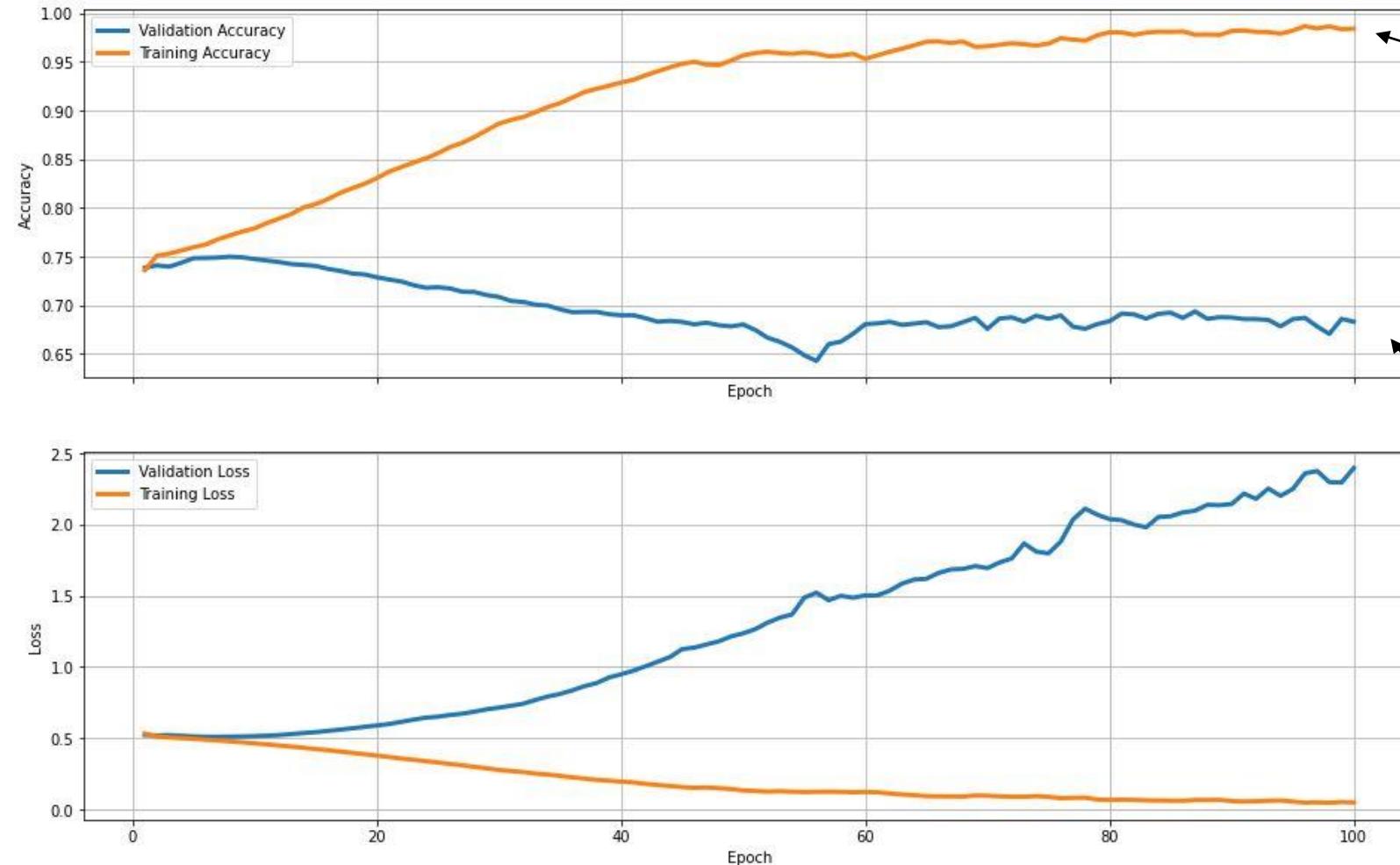
# Sentiment Analysis: Neural Network (NN)



The results seem good on the train set...

... but there are significantly less good on the test set

# Sentiment Analysis: Neural Network (NN)



The results seem good on the train set...

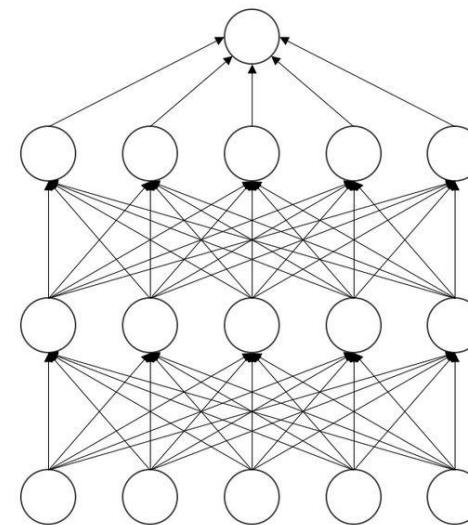
... but there are significantly less good on the test set

**Overfitting !!!**

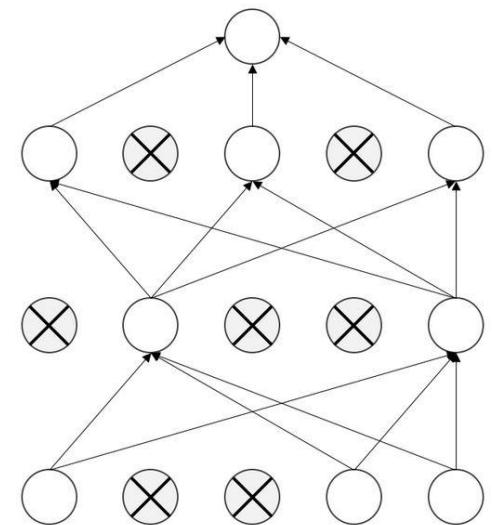
# Sentiment Analysis: Neural Network (NN)

- To solve **overfitting** issues in a Neural Network, a powerful tool is to add **Dropout** layers
- **Dropout** : randomly dropping out (setting to 0) some of output features of a given layer
- It adds noise inside the network in order to prevent the neurons from being too sensitive to variations

```
model = tf.keras.models.Sequential([
    hub.KerasLayer(embed,
                  dtype=tf.string, input_shape=[], output_shape=[50]),
    Dense(128, activation="relu"),
    Dropout(rate=0.8),
    Dense(300, activation="relu"),
    Dropout(rate=0.8),
    Dense(1, activation="sigmoid")
])
```



Standard Neural Net



After applying dropout

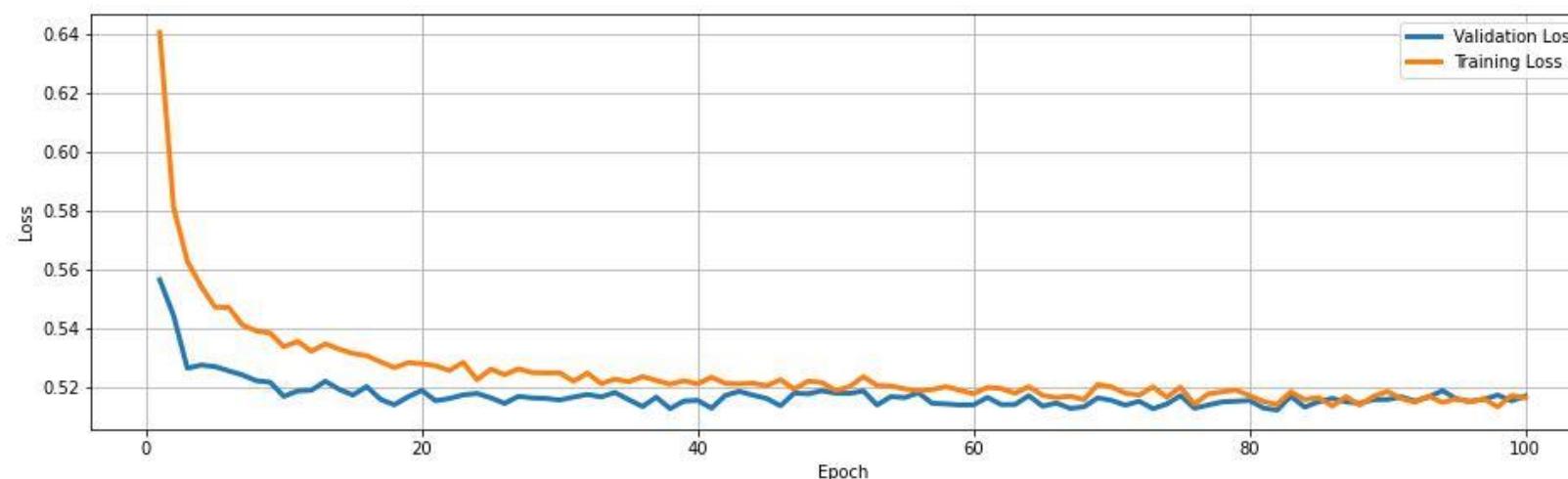
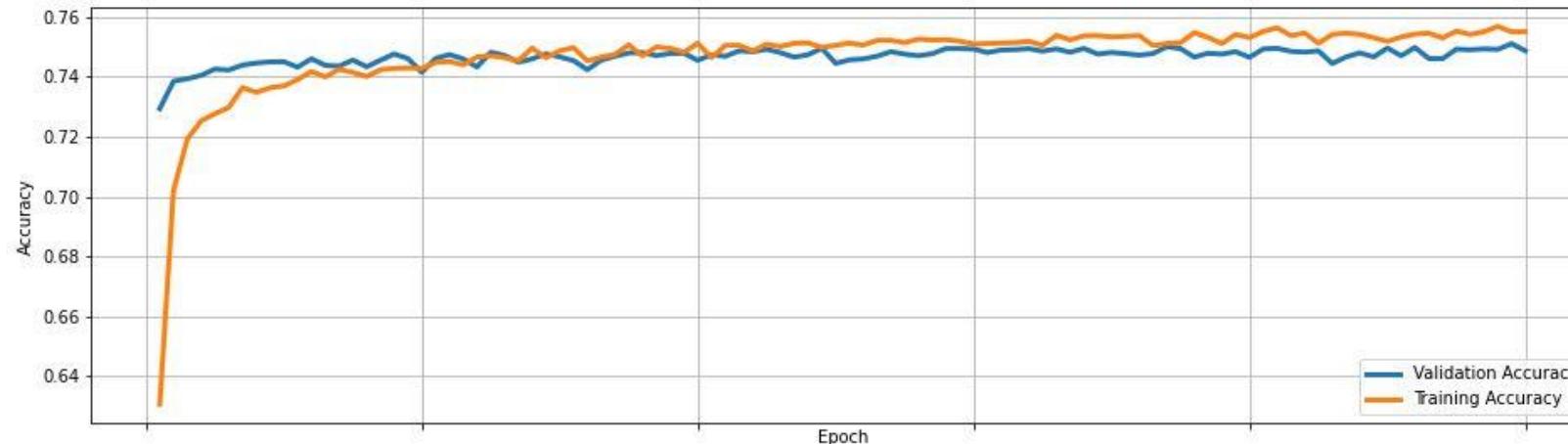
# Sentiment Analysis: Neural Network (NN)

```
▶ model.summary()
```

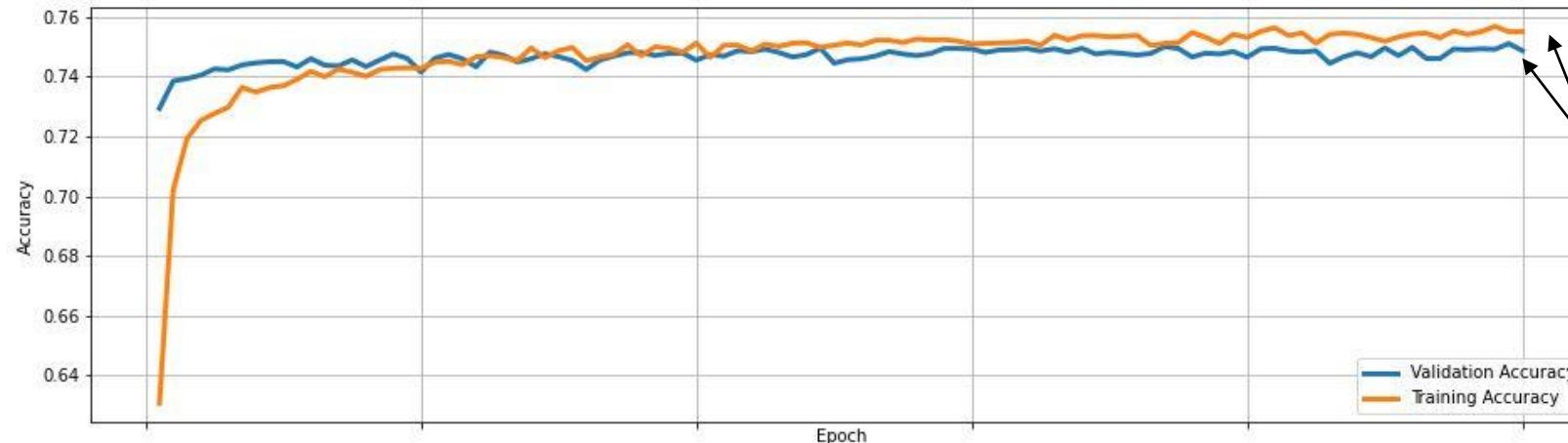
```
□ Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
=====		
keras_layer_2 (KerasLayer)	(None, 50)	48190600
dense_6 (Dense)	(None, 128)	6528
dropout_2 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 300)	38700
dropout_3 (Dropout)	(None, 300)	0
dense_8 (Dense)	(None, 1)	301
=====		
Total params: 48,236,129		
Trainable params: 45,529		
Non-trainable params: 48,190,600		

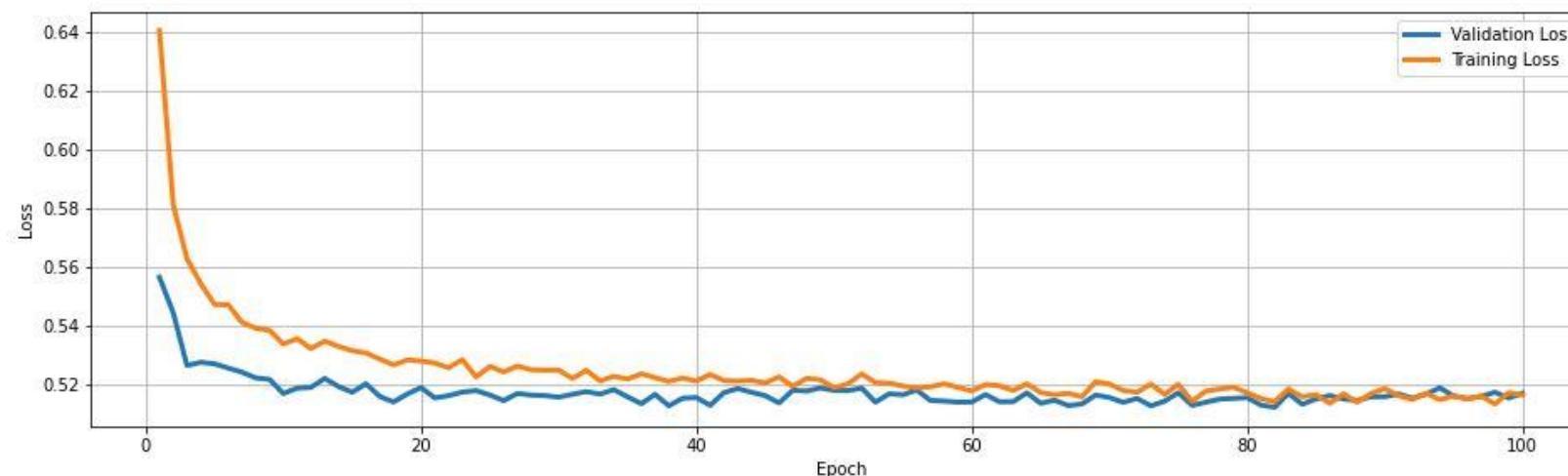
# Sentiment Analysis: Neural Network (NN)



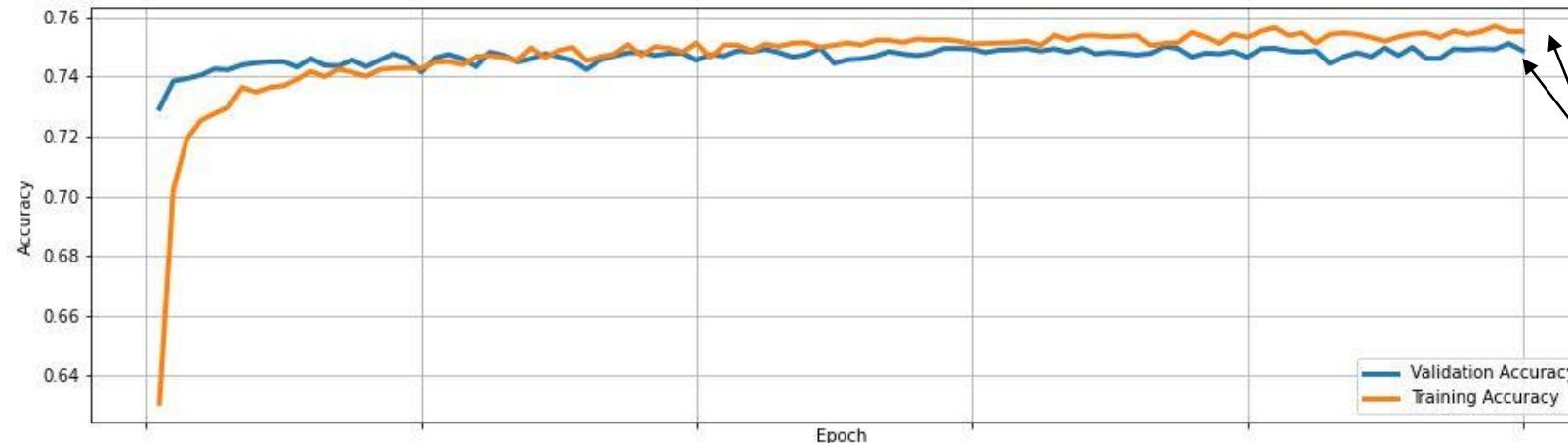
# Sentiment Analysis: Neural Network (NN)



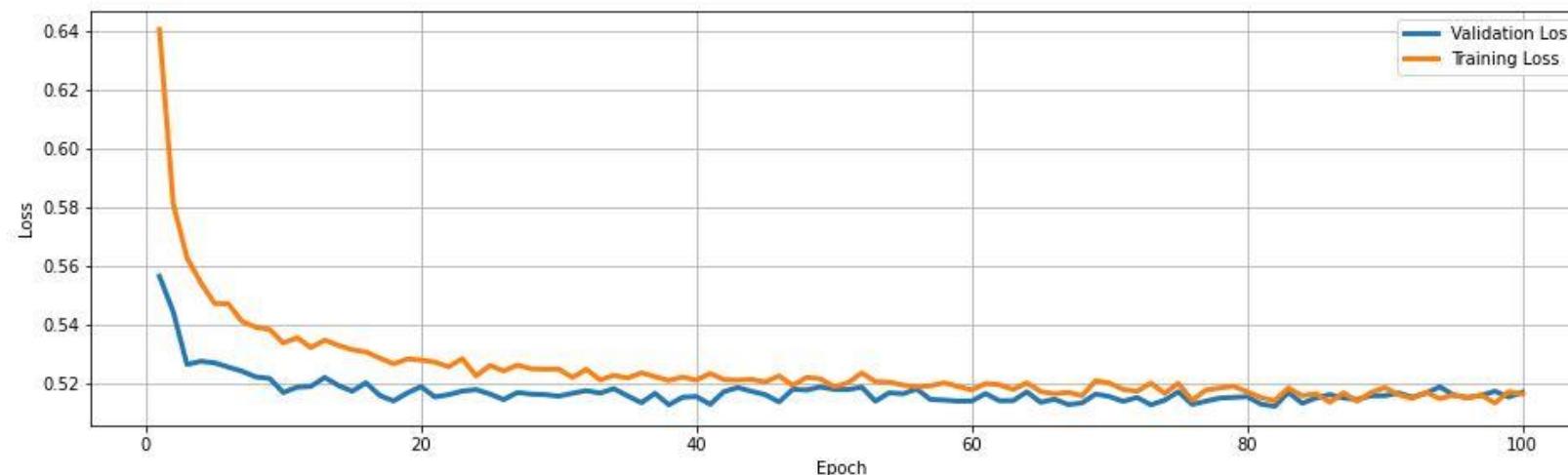
The results are similar for both train and test set



# Sentiment Analysis: Neural Network (NN)



The results are similar for both train and test set



Overfitting prevented !

# Take-away from Course 2

- Word embedding is a powerful way to convert text into numerical value vector
- Solves the problems of sparsity and very high dimension met with basics methods like Bag of words and TF\_IDF (see Course 1)
- Allows as well a better representativity of the language : words are close in the vector space if their meaning is similar
- Word embeddings come from neural network training on HUGE datasets: need to use pre-trained libraries for general use cases
- To compare the distance between two word-vectors, the cosine similarity distance is generally the best choice
- Word embeddings can be used as inputs for NLP use cases (such as sentiment analysis), and are often used as neural network first layer inputs

# Course Schedule

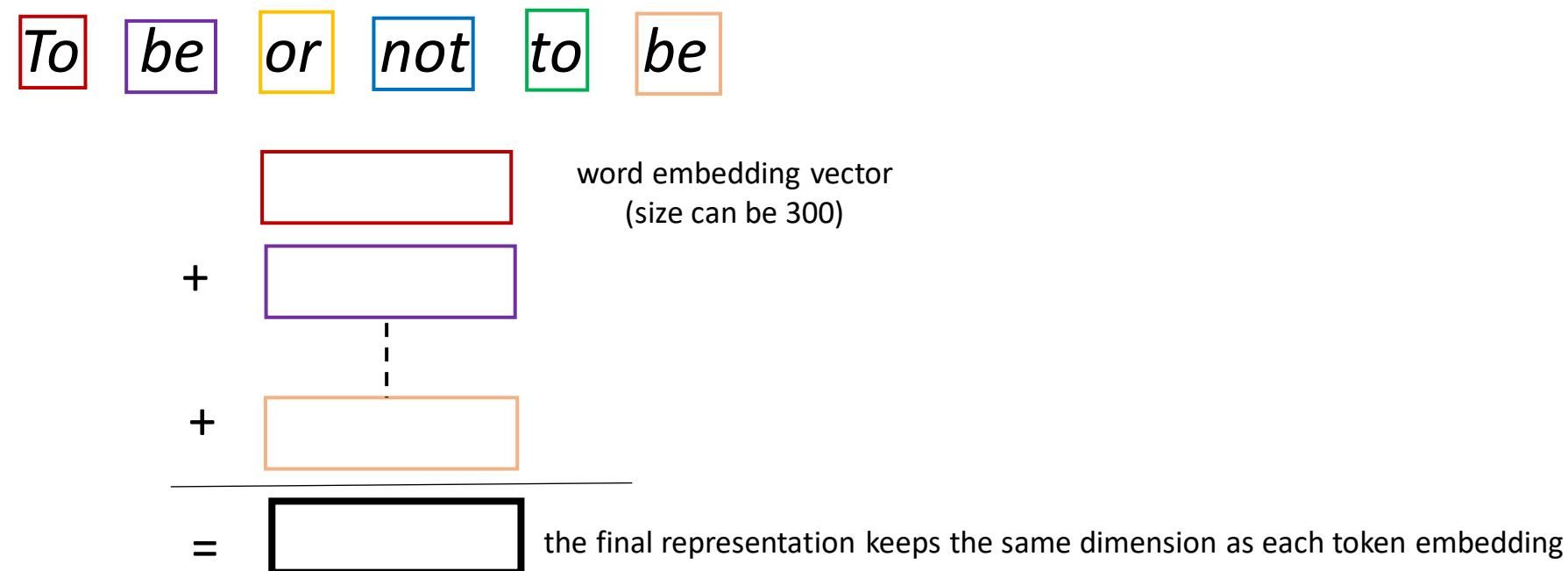
- **Course 1:** NLP introduction
- **Course 2:** Word embedding
- **Course 3: Long Short Term Memory (LSTM) architecture**
  - Recurrent Neural Network (RNN)
  - LSTM principle and architecture
  - Text preprocessing (to feed an LSTM)
  - Use case studies
    - Sentiment analysis
    - Translation
    - Text generation
- **Course 4:** “Attention” mechanism and Transformer architectures
- **Course 5:** Chatbot implementation



# Previously in Course 2

# Embedding: Text embedding

One can get a **text embedding** in **averaging** the embedding vectors of all the tokens in the text

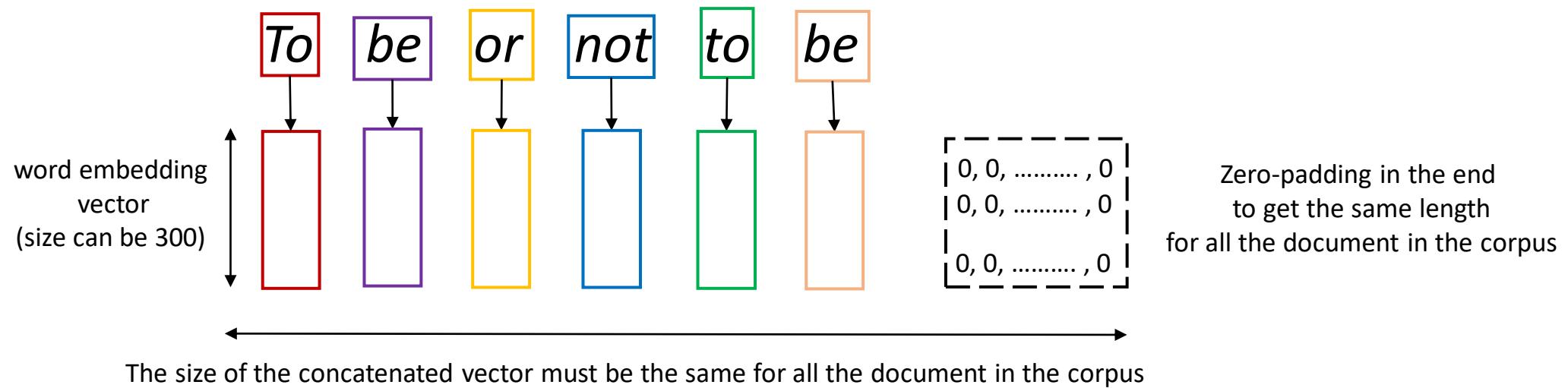


## Remarks

- The dimension of the text embedding remains quite **low**
- There is a significant **loss of information** in comparison with the previous method

# Embedding: Sequence embedding

**One solution is to concatenate the embedding vectors of all the tokens of the text**



## Remarks

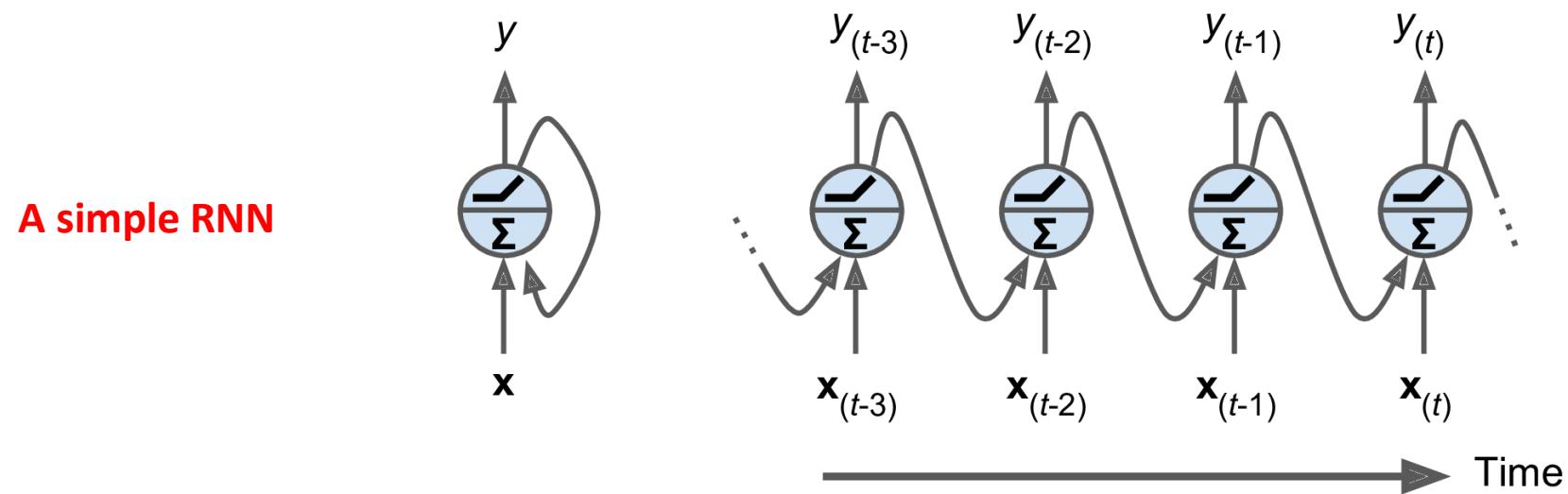
- to keep **the same dimensions** for each text of a corpus, there is a need to **truncate** the number of tokens or to **add padding** (depending on the number of tokens)
- The dimension of each text representation can be very **high** (e.g., several thousands)

# Course 3: Long Short Term Memory (LSTM)

# Recurrent Neural Network (RNN)

# Recurrent Neural Network (RNN): Definition

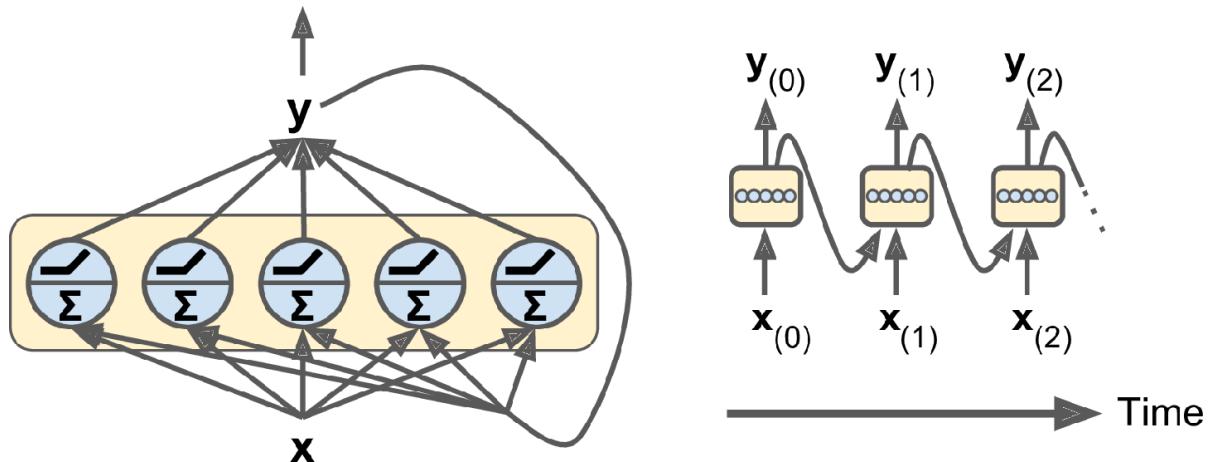
- A **RNN** is a kind of neural network dealing with **recurrent** connection
- Allows to deal with **temporal sequences**
- E.g., on the figure below, a sequence **X** is given as **input**, and we get a sequence **Y** as **output**



# Recurrent Neural Network (RNN): Definition

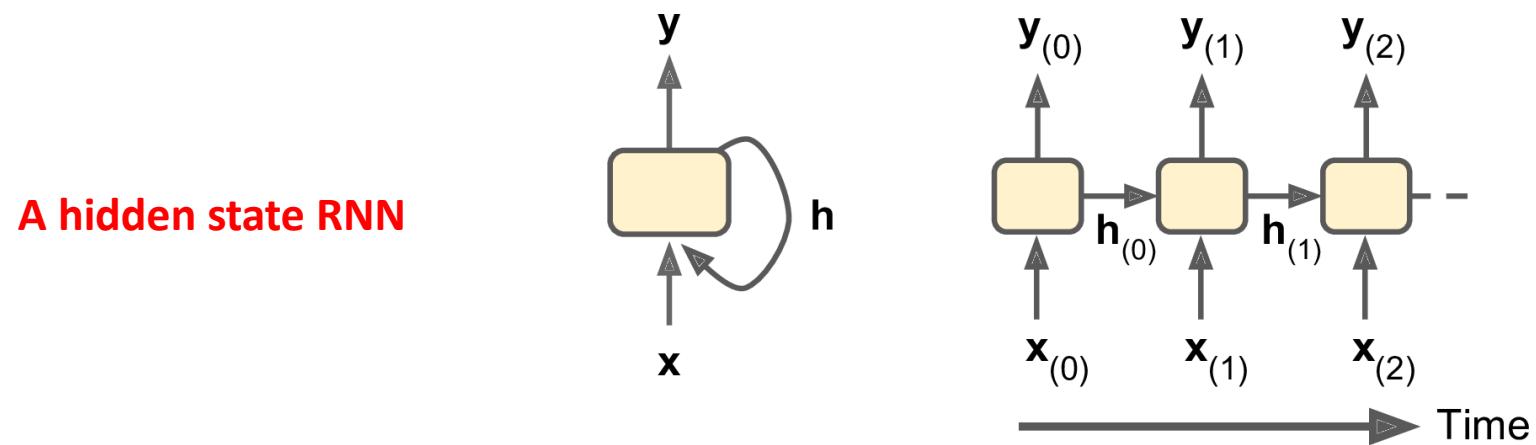
- A **RNN** is a kind of neural network dealing with **recurrent** connection
- Allows to deal with **temporal sequences**
- E.g., on the figure below, a sequence **X** is given as **input**, and we get a sequence **Y** as **output**

A multi-neuron RNN

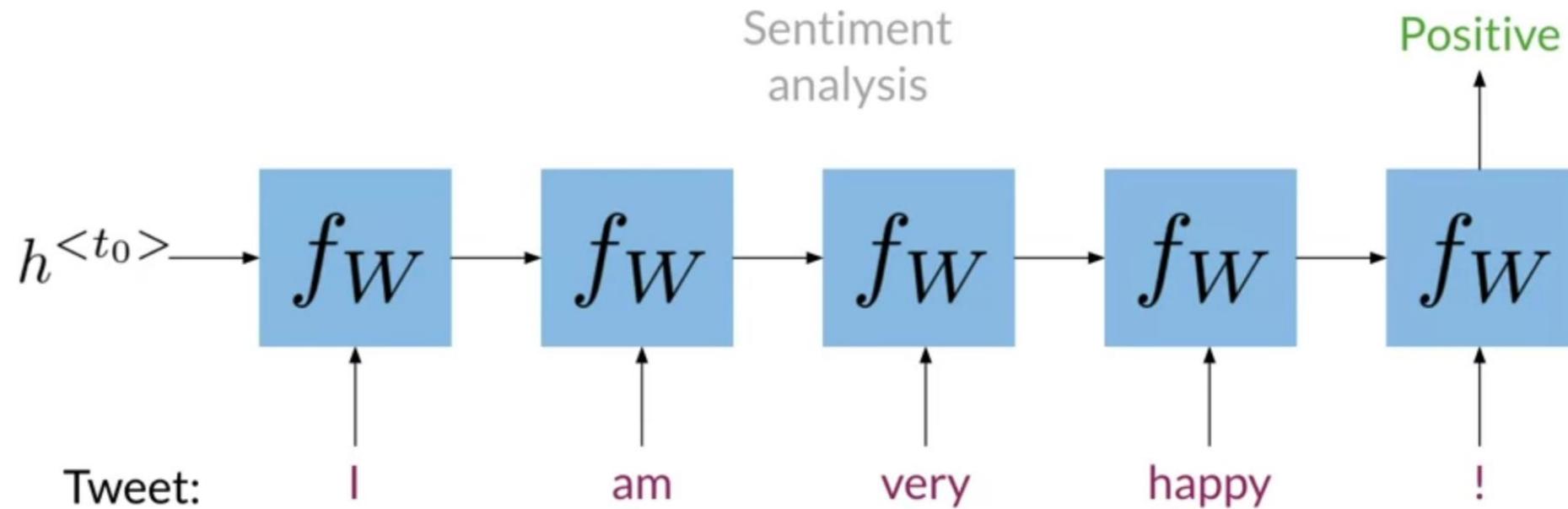


# Recurrent Neural Network (RNN): Definition

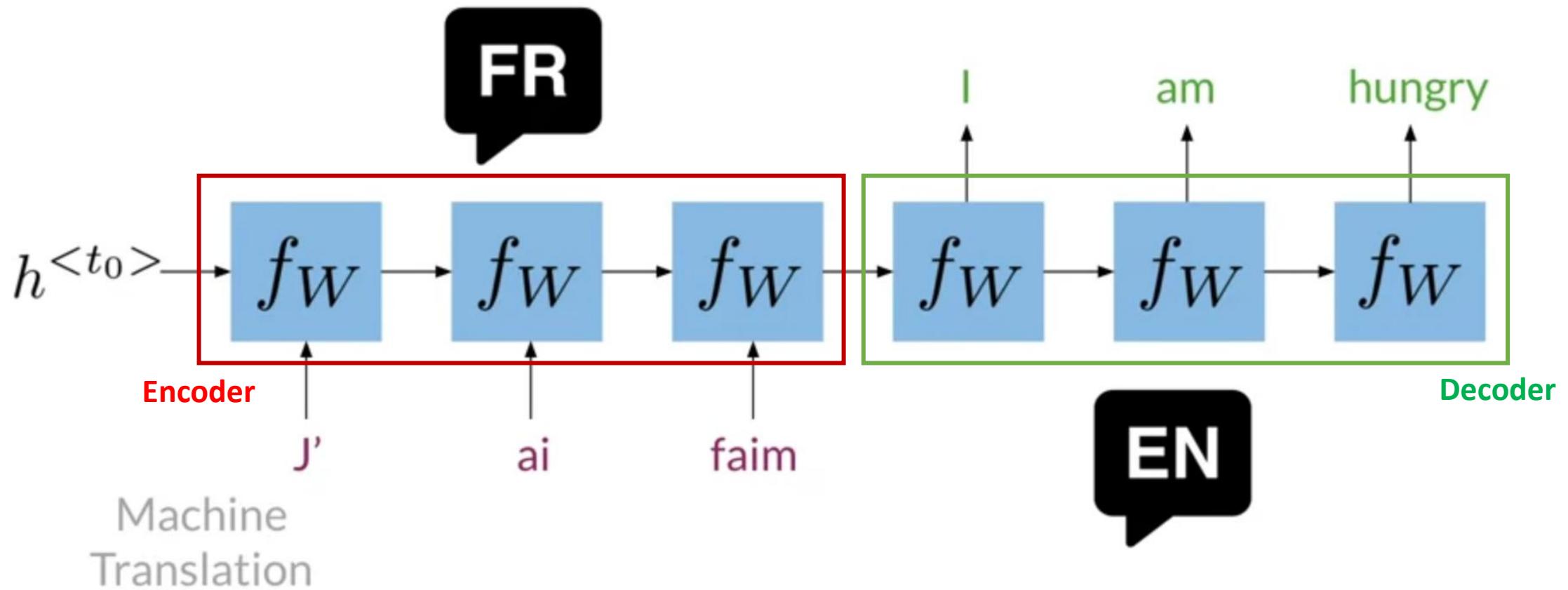
- A **RNN** is a kind of neural network dealing with **recurrent** connection
- Allows to deal with **temporal sequences**
- E.g., on the figure below, a sequence **X** is given as **input**, and we get a sequence **Y** as **output**
- **A cell hidden state  $h$  and the output  $y$  may be different**



# RNN Applications : Many to one

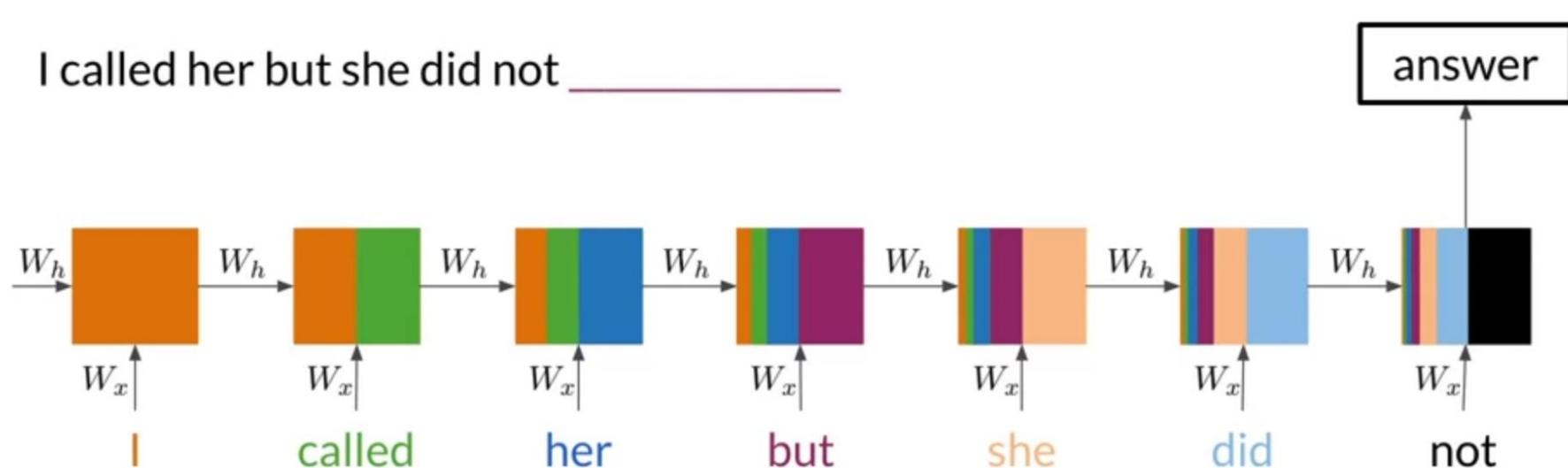


# RNN Applications : Many to many



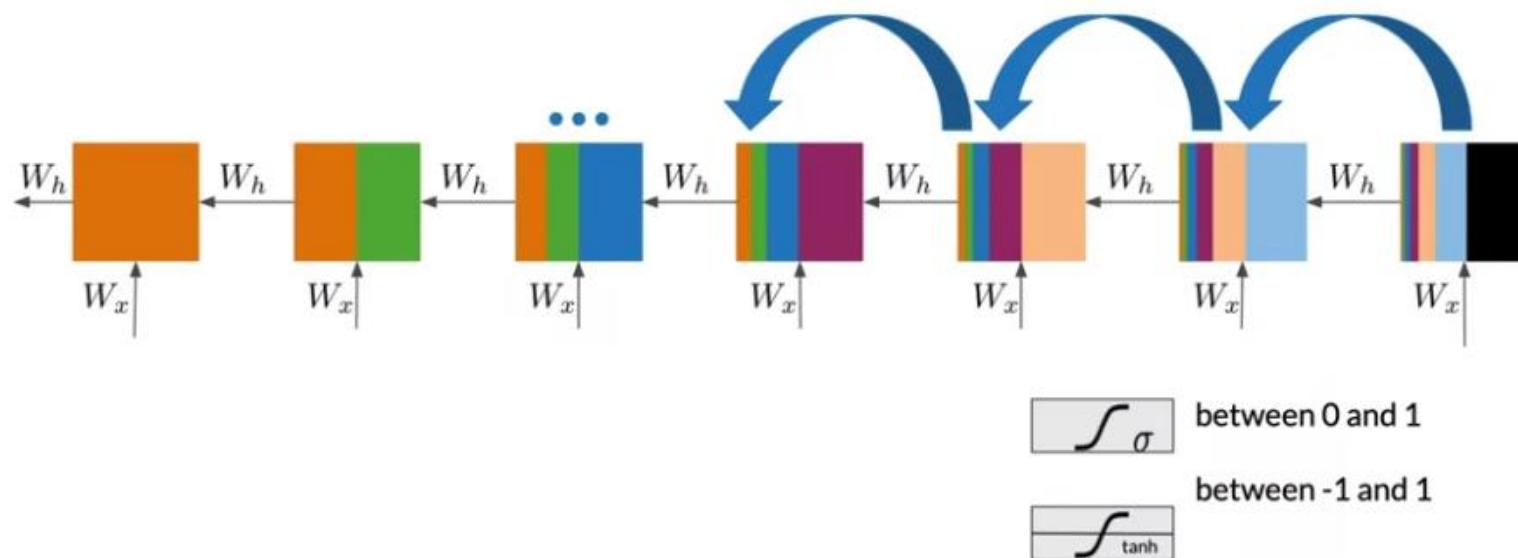
# RNN : Long sequence issue

- As shown in the illustration, RNN can be used for **text generation**
- However, we can see that, after a **few** cells only, the **impact** of each word almost **disappear**



# RNN : Vanishing gradient

- **Vanishing gradient** is another common problem with RNN especially for very long sequences => **makes training non efficient**
- Having a close look at **gradient backpropagation equations** shows that the gradient issues are due to the **derivative of the *tanh* activation function (<1)**



# LSTM principle and architecture

# LSTM (Long Short Term Memory)

## Principle

- The classical deep neural network (e.g., **RNN**) cannot avoid both **vanishing gradient problem** and **loss of early information**
- **LSTM networks** were invented by **Hochreiter** and **Schmidhuber** in 1997
  1. To deal with the **vanishing gradient problem**
  2. To proceed **entire sequences** of data **without forgetting** the meaning of the **early information** proceeded

# LSTM (Long Short Term Memory)

## Example

*I used to live in France, I speak French fluently*

The word **French** can describe both a language or a nationality. Here the **context** help us to know it corresponds to the language.

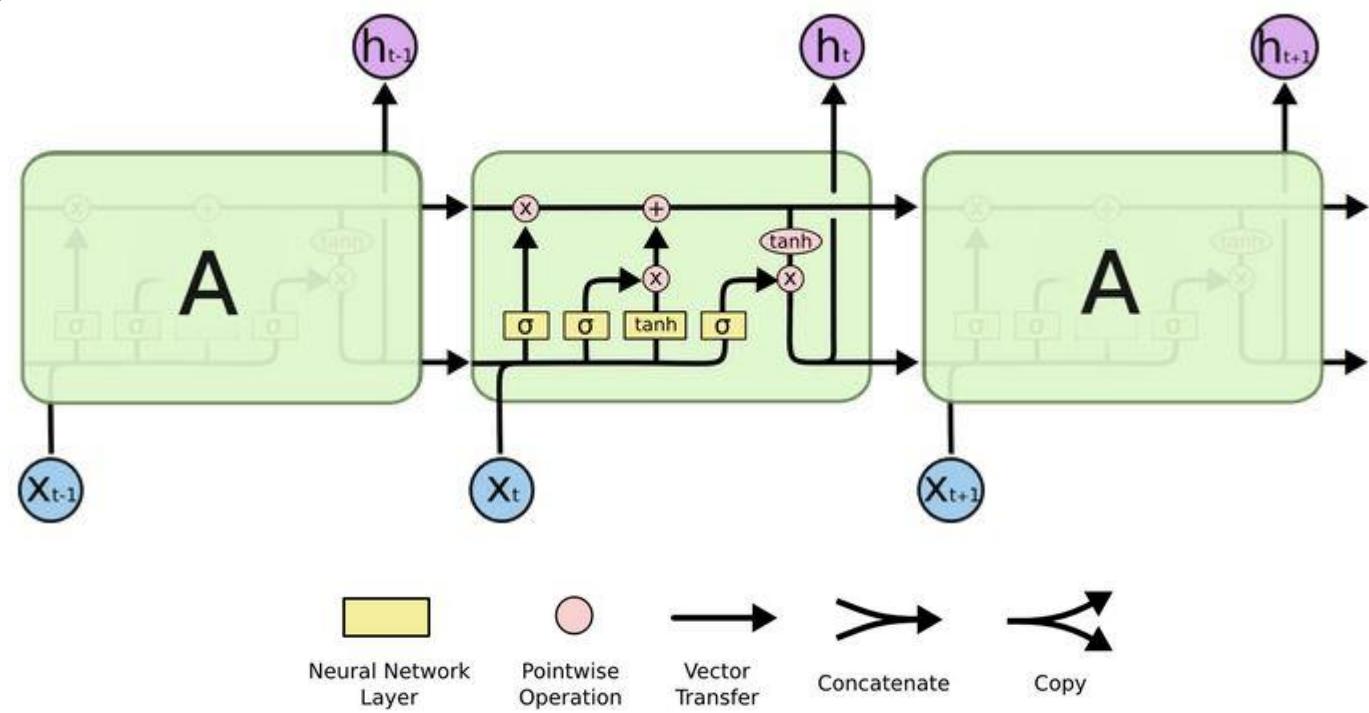
**LSTM aims to make good use of the context to determine the meaning of the analyzed words**

# LSTM (Long Short Term Memory)

A **LSTM architecture** is composed of a set of cells, each of those containing **three gates**

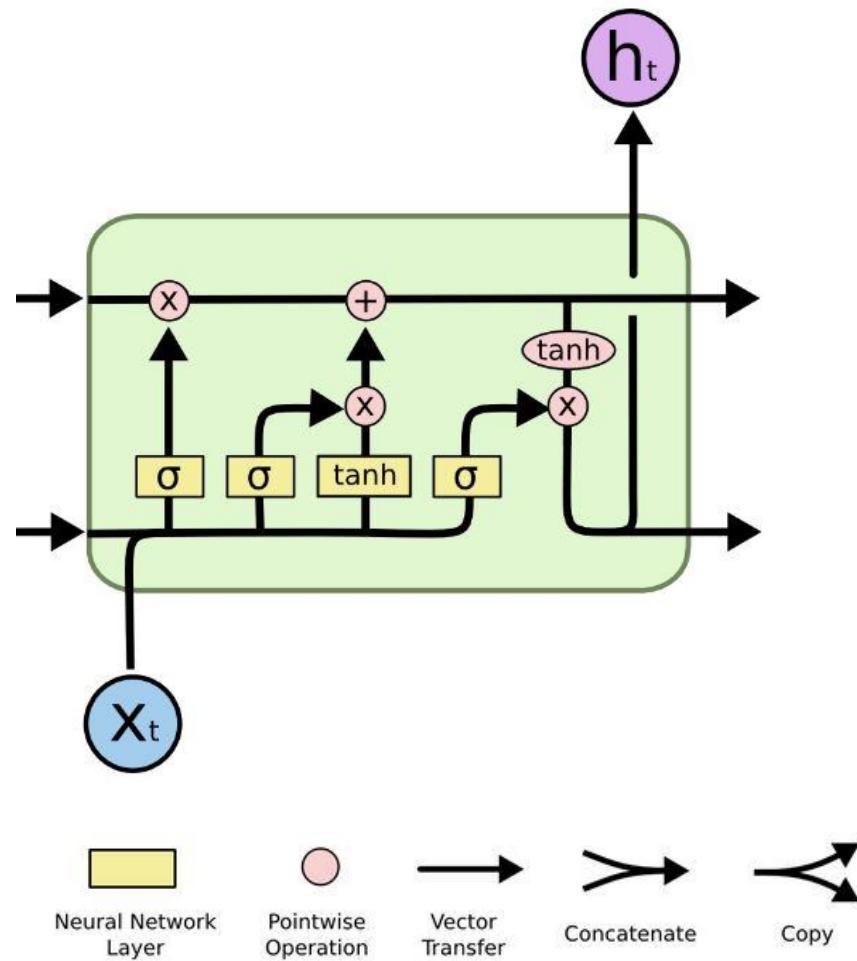
- **An input gate**: decides what to add
- **An output gate**: decides what the next hidden state will be
- **A forget gate**: decides what to keep

Repeated LSTM modules  
(each of them containing four layers)



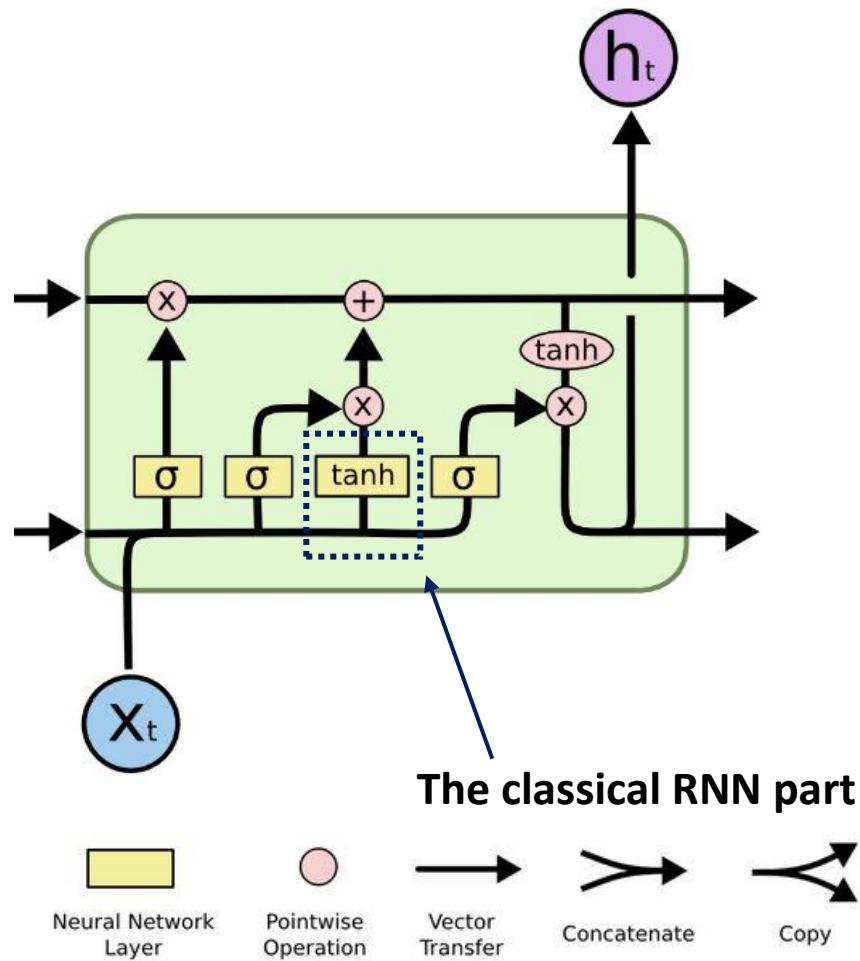
# LSTM (Long Short Term Memory)

## LSTM module



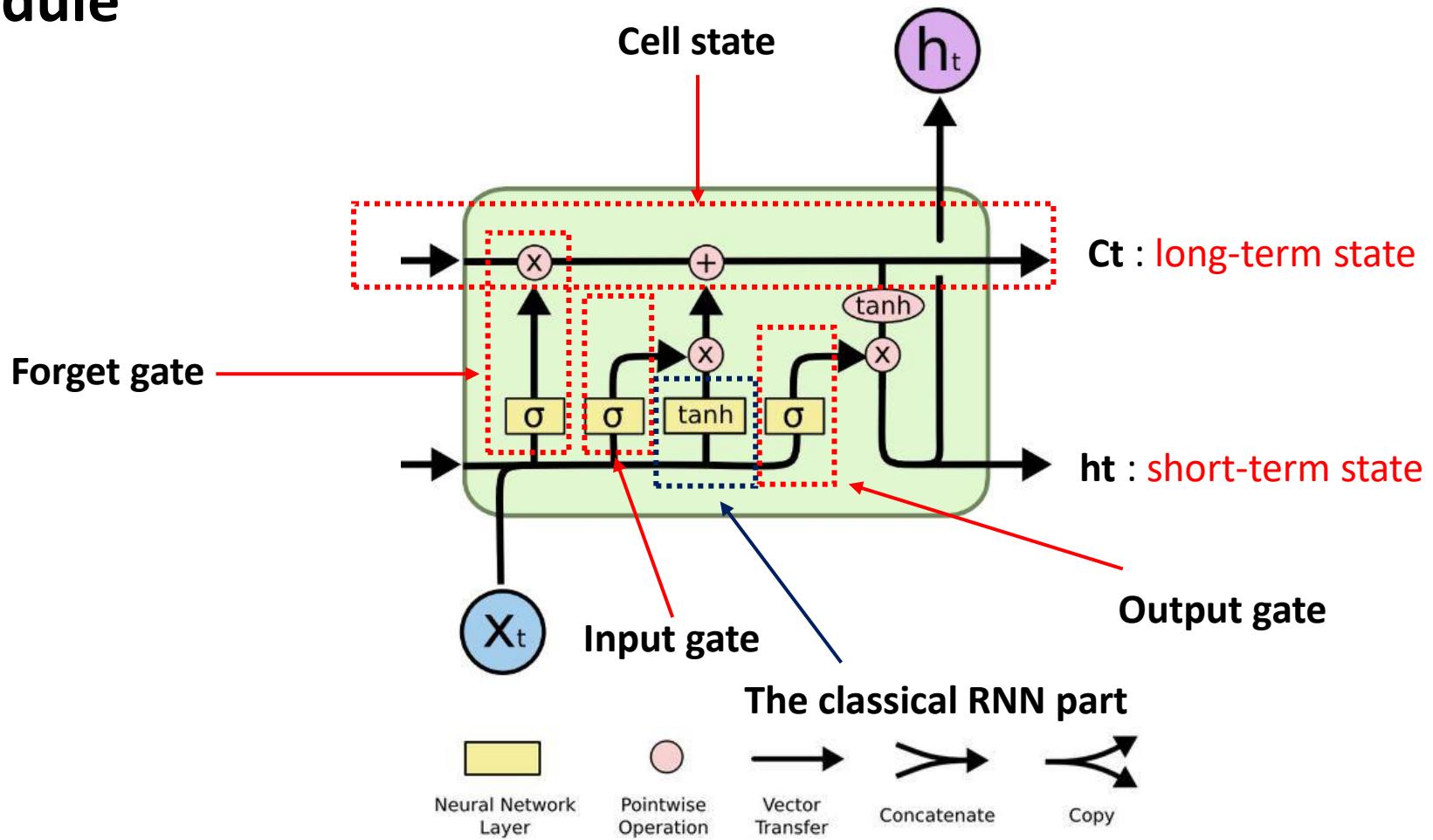
# LSTM (Long Short Term Memory)

## LSTM module



# LSTM (Long Short Term Memory)

## LSTM module



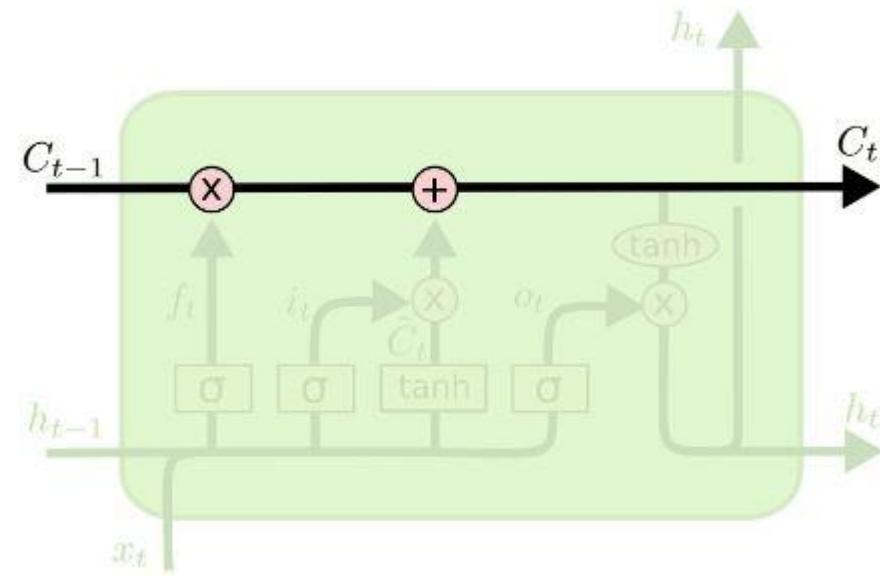
# LSTM (Long Short Term Memory)

**Cell state:**  $C_t$

The **cell state** runs down the **entire chain**  
with some **linear interactions**

through each LSTM module met down the way

Gates allow LSTM to optionally  
remove or add information to the **cell state**



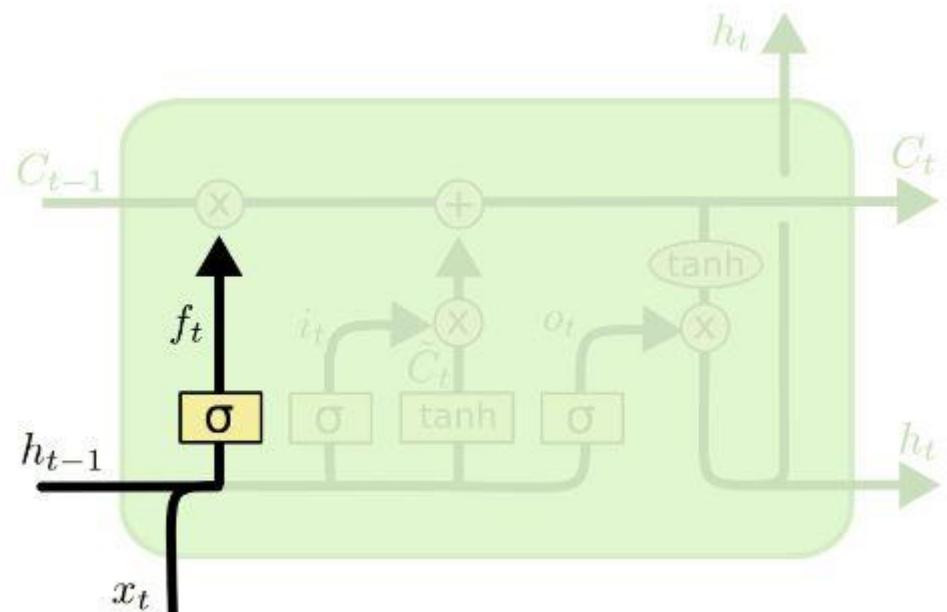
# LSTM (Long Short Term Memory)

**Forget gate:** decide what information we drop from the **cell state**

The **sigmoid** output a number between **0 and 1** for each number in the **cell state**  $C_{t-1}$

**Example:** the cell state might include the **gender** of a **present** subject.

If a **new** subject is seen => **forget** the previous one



$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

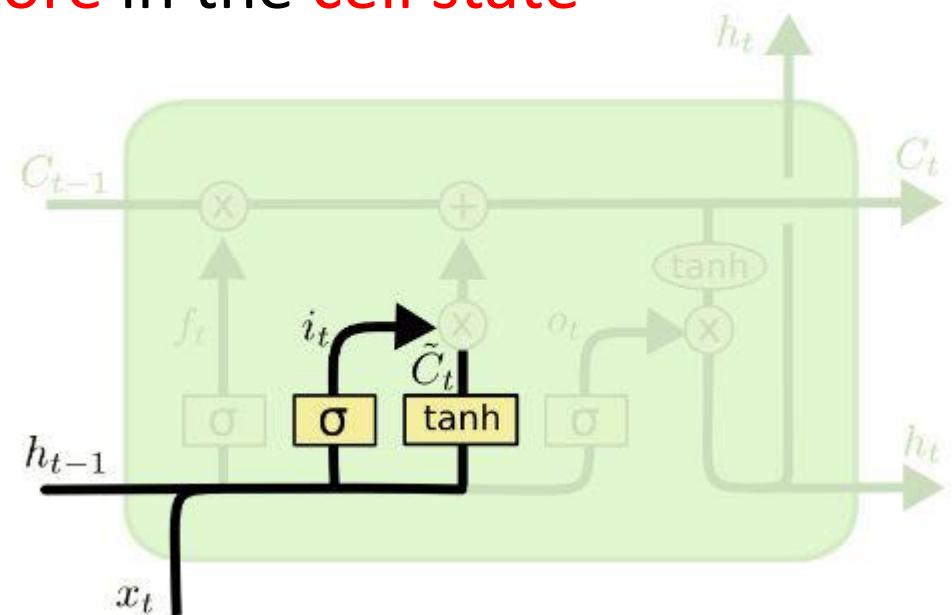
# LSTM (Long Short Term Memory)

**Input gate:** decide what new information we **store** in the **cell state**

**Two parts:**

- A sigmoid layer to decide which values we **update**
- A tanh layer to create a vector of new **candidate values** for  $C_t$

**Example:** we **add** the gender of the **new subject** to replace the **previous** one which has been **forgotten**



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

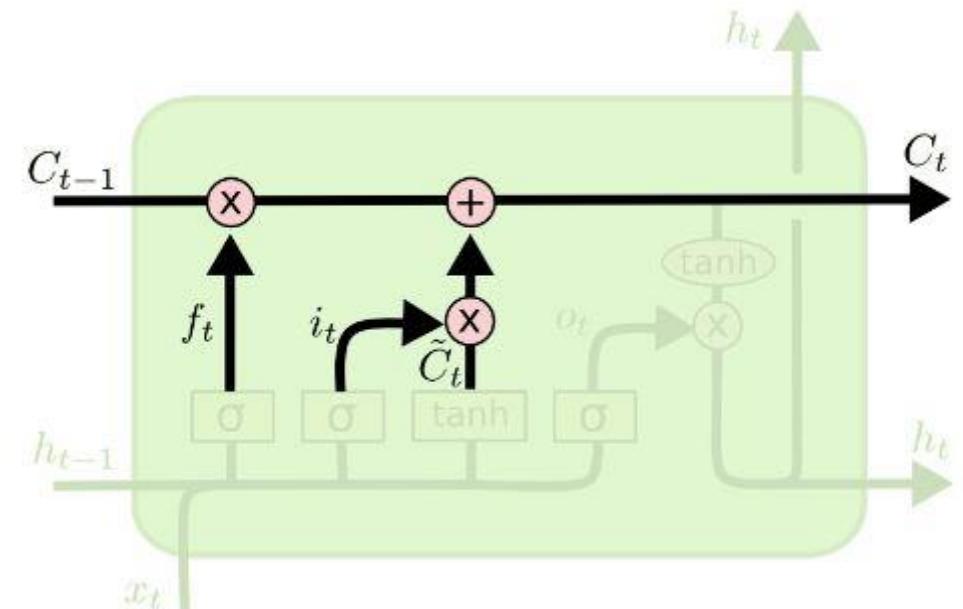
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

# LSTM (Long Short Term Memory)

**Cell state update:**  $C_{t-1} \Rightarrow C_t$

- The previous state is multiplied by  $f_t$ , to forget the things we decided earlier
- We add the new candidate values, scaled by how much we decided to update earlier

**Example:** we drop the information about the old gender and add the information related to the new one



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

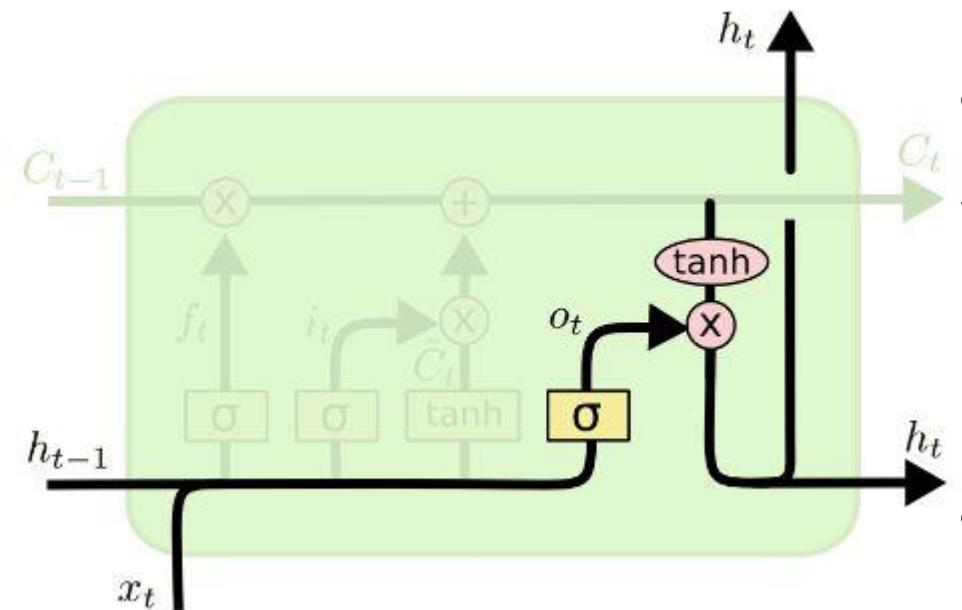
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

# LSTM (Long Short Term Memory)

**Output gate:** decide what information to **output**

The **output** is a filtered version of the cell state

- First, we run a **sigmoid layer** to decide what part of the cell state **to output**
- Then, we **rescale** the cell state (using a tanh layer) and **multiply** it by the result of the **output gate**



**Example:** it might **output** whether the **subject** is **singular** or **plural**, in order to know how to **conjugate** the following verb

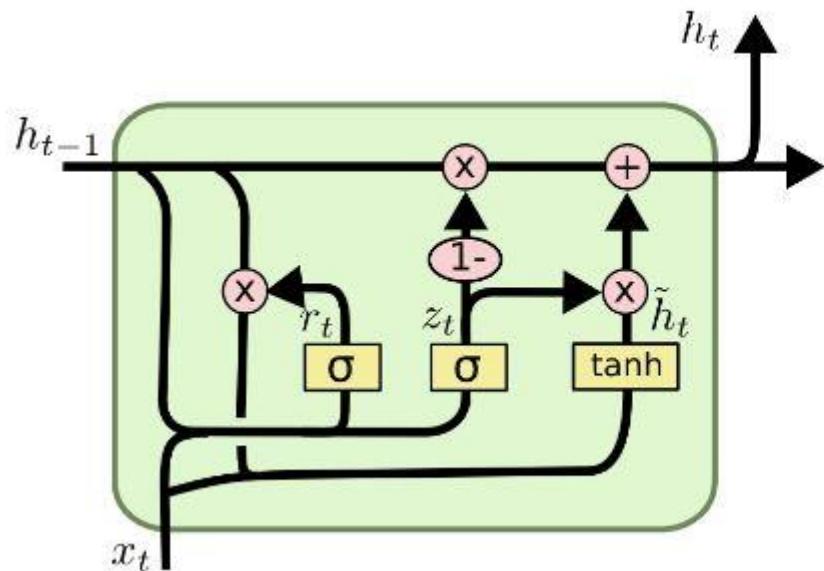
$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

# Gated Recurrent Unit (GRU)

Introduced in 2014, it corresponds to a simplified LSTM variant with only two gates

- The **forget** and **input** gates are combined into a single **update** gate which defines how much information to keep
- A **reset** gate is defined to determine how much information to forget
- Both LSTM and GRU can be useful depending on the specific application



$$z_t = \sigma (W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma (W_r \cdot [h_{t-1}, x_t])$$

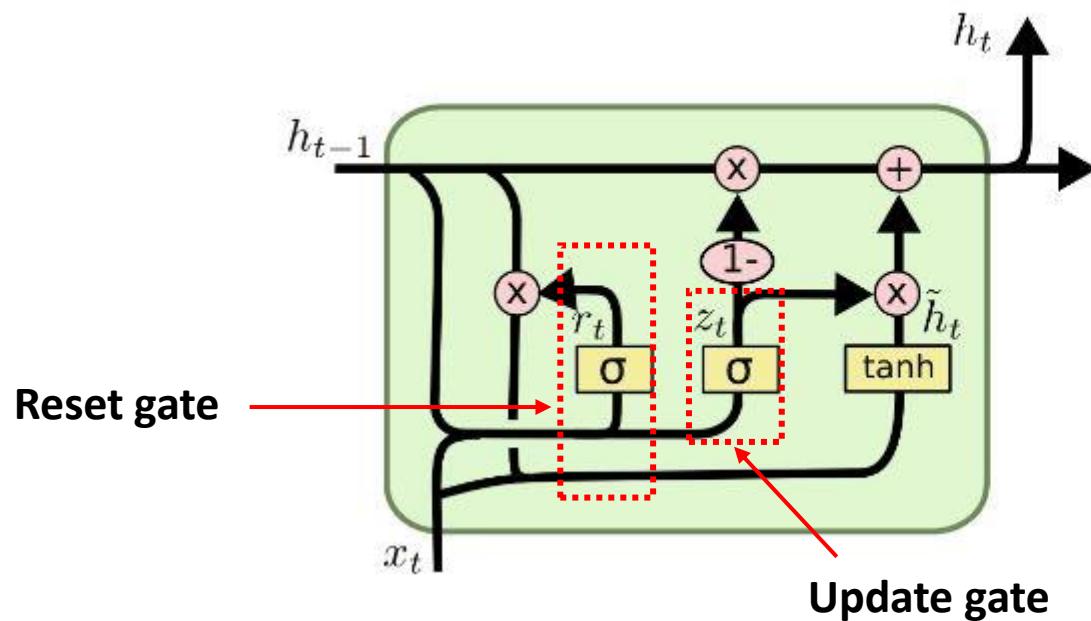
$$\tilde{h}_t = \tanh (W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

# Gated Recurrent Unit (GRU)

Introduced in 2014, it corresponds to a simplified LSTM variant with only two gates

- The **forget** and **input** gates are combined into a single **update** gate which defines how much information to keep
- A **reset** gate is defined to determine how much information to forget
- Both LSTM and GRU can be useful depending on the specific application



$$z_t = \sigma (W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma (W_r \cdot [h_{t-1}, x_t])$$

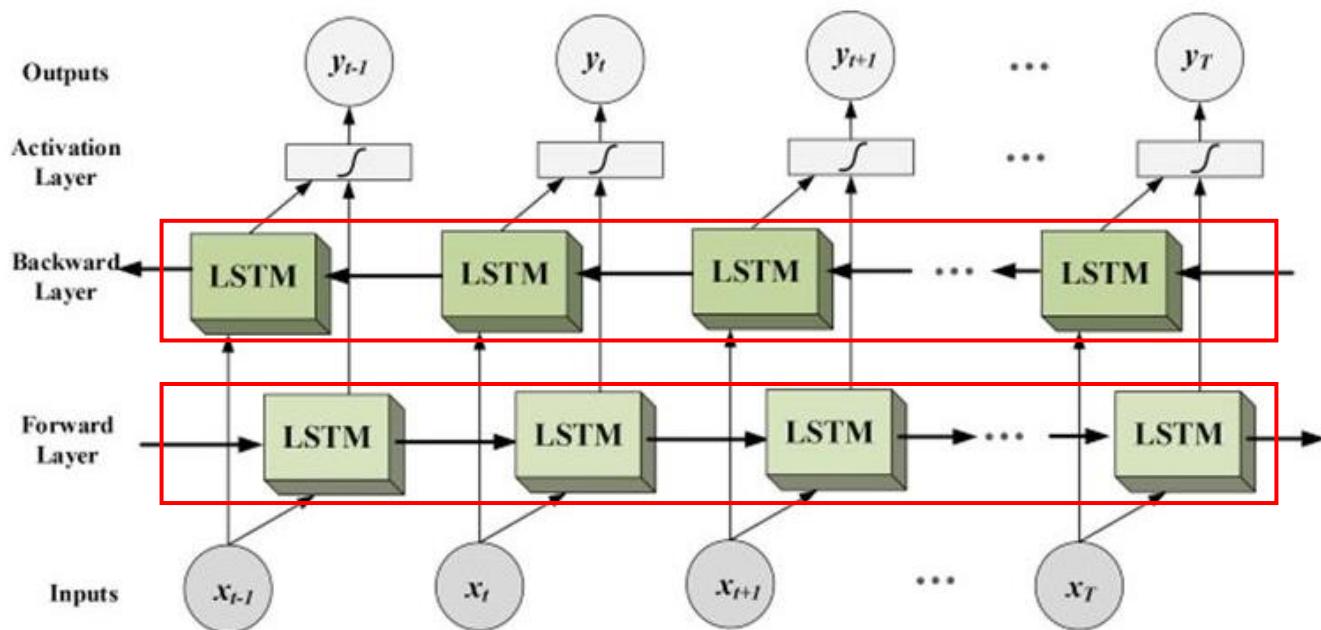
$$\tilde{h}_t = \tanh (W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

# Bidirectional LSTM

In a **classic LSTM** architecture, the sentence/sequence is processed only in **one direction** (generally from the past to the present or future)

The **bidirectional** structure process the data in **both directions** => can be useful for some applications



The Queen of the United Kingdom

The queen of hearts

The queen of bees

Here, the word queen will be encode differently if read in reverse order

# LSTM (Long Short Term Memory)

## Applications

- **Text analysis**
- **Text translation**
- **Next-word prediction**
- Chatbots
- Music composition
- Image captioning
- Speech recognition

# Text preprocessing (to feed an LSTM)

# Text preprocessing

The **LSTM architecture** was conceived to process **long sequences** without any **long-term dependency issues**

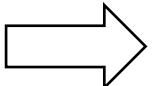
We train an **LSTM Network with chunks of text**:

1. Each chunk of text must be split into sequences => **Tokenization**
2. Each sequence must have the same length => **Padding**
3. Each element in a sequence should be converted into numerical values => **Embedding**

# Text preprocessing

The **LSTM architecture** was conceived to process **long sequences** without any **long-term dependency issues**

We train an **LSTM Network** with **chunks of text**:

- 
1. Each chunk of text must be split into sequences => **Tokenization**
  2. Each sequence must have the same length => **Padding**
  3. Each element in a sequence should be converted into numerical values => **Embedding**

# Tokenization

## Principle

*To be or not to be*

# Tokenization

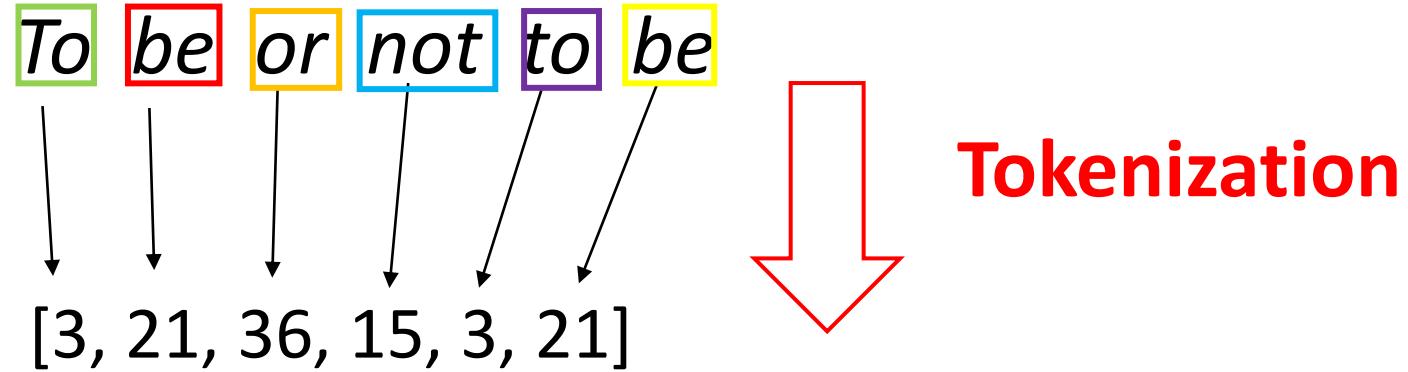
## Principle

*To be or not to be*

Tokenization

# Tokenization

## Principle

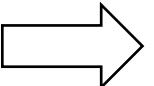


Each token is associated to its **number** in the **vocabulary** considered

# Text preprocessing

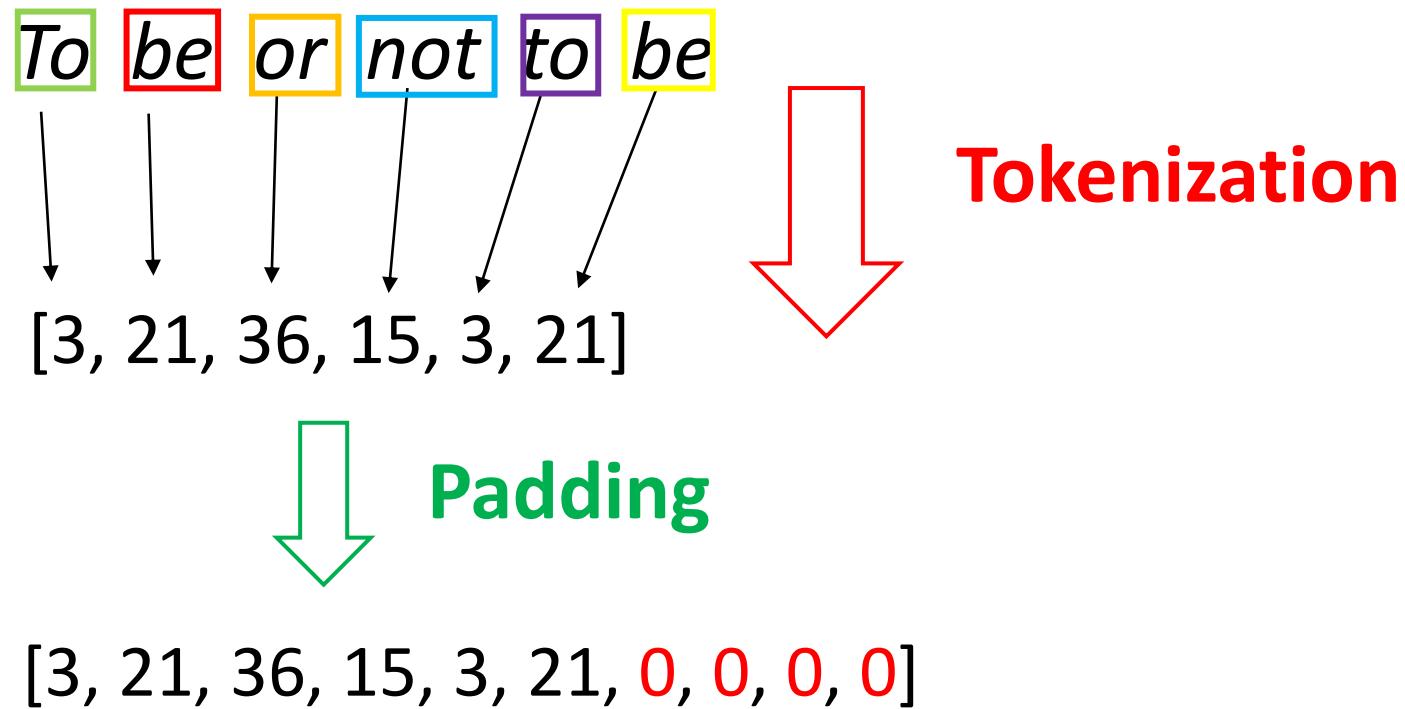
The **LSTM architecture** was conceived to process **long sequences** without any **long-term dependency issues**

We train an **LSTM Network** with **chunks of text**:

- 
1. Each chunk of text must be split into sequences => **Tokenization**
  2. **Each sequence must have the same length => Padding**
  3. Each element in a sequence should be converted into numerical values => **Embedding**

# Padding

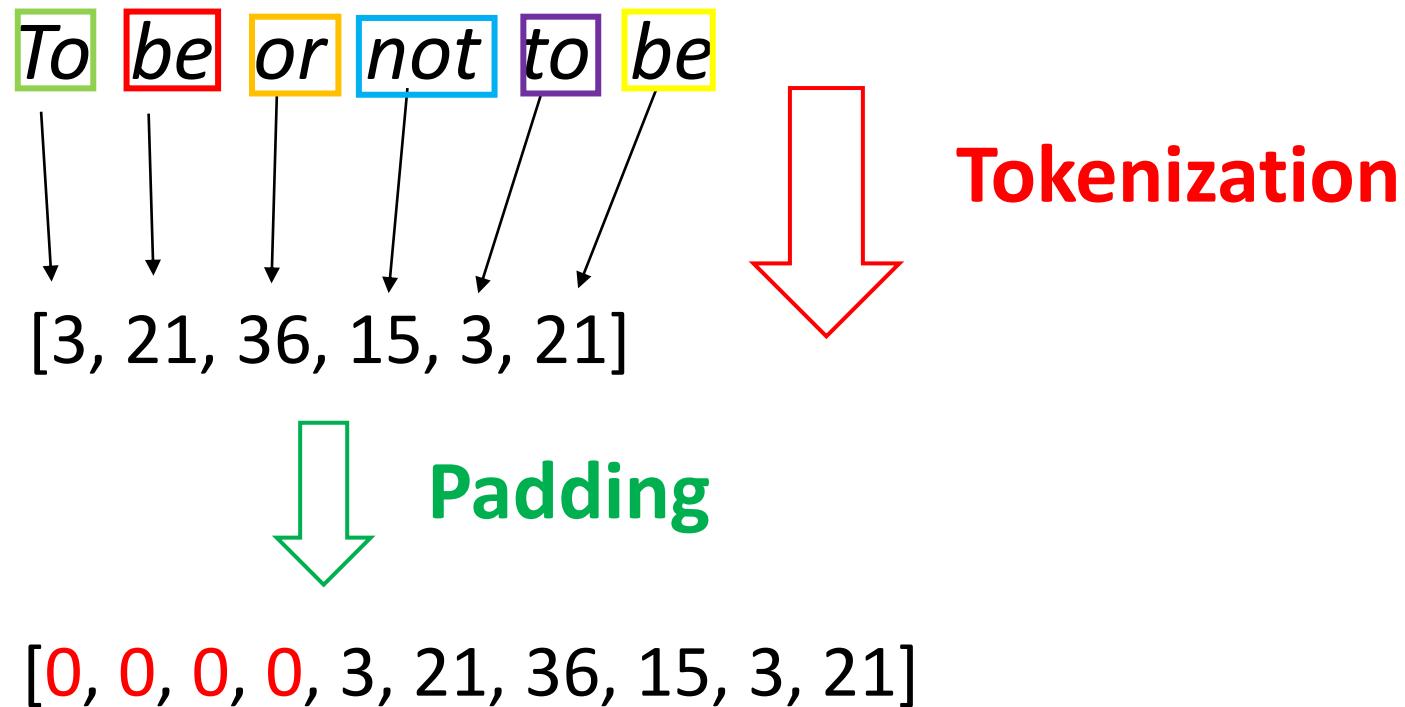
## Principle



We add **additional zeros** in the **end** in order to consider a **fixed length** for every sequence

# Padding (Variant)

## Principle



We can as well add the **additional zeros** in the **beginning** instead

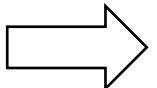
The choice may depend on the application

# Text preprocessing

The **LSTM architecture** was conceived to process **long sequences** without any **long-term dependency issues**

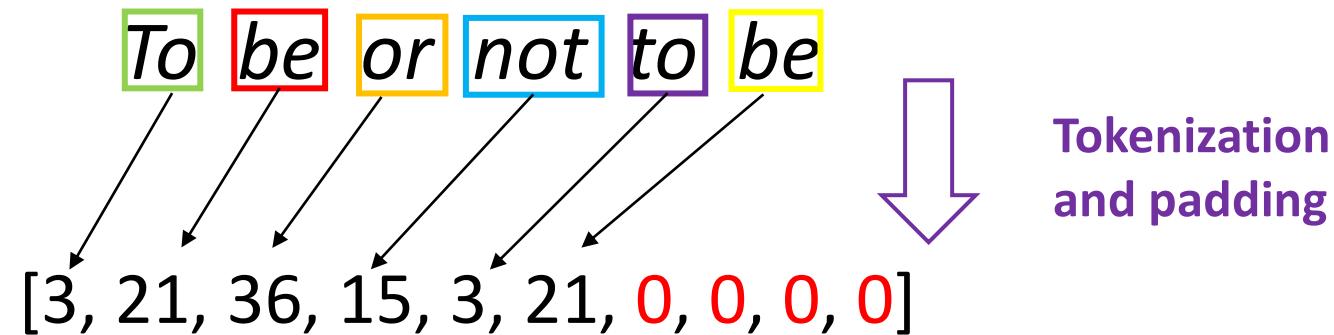
We train an **LSTM Network** with **chunks of text**:

1. Each chunk of text must be split into sequences => **Tokenization**
2. Each sequence must have the same length => **Padding**
3. Each element in a sequence should be converted into numerical values => **Embedding**



# Embedding

Once we get the padded sequence



two possibilities concerning the next step

1. Use of a pre-trained embedding (see **Course 2**)
2. Learn a new embedding from scratch (use of an Embedding layer in Keras)

# Embedding

As we saw during last **Course**:

*Word embeddings come from neural network training on **HUGE** datasets: need to use **pre-trained libraries** for general use cases*

**BUT** in some cases, we are not interested by a model with a good representation of a language **in general** but only for a **specific** application.

In that case, and if the training data set is **big enough**, we can learn from scratch a new embedding for **the specific task** we are interested in.

(Another possibility is to **fine-tune** a pre-trained model – let's see that later in **Course 4!**)

# Embedding: One-hot encoding step

To feed an embedding training network, we must use a relevant format for the input

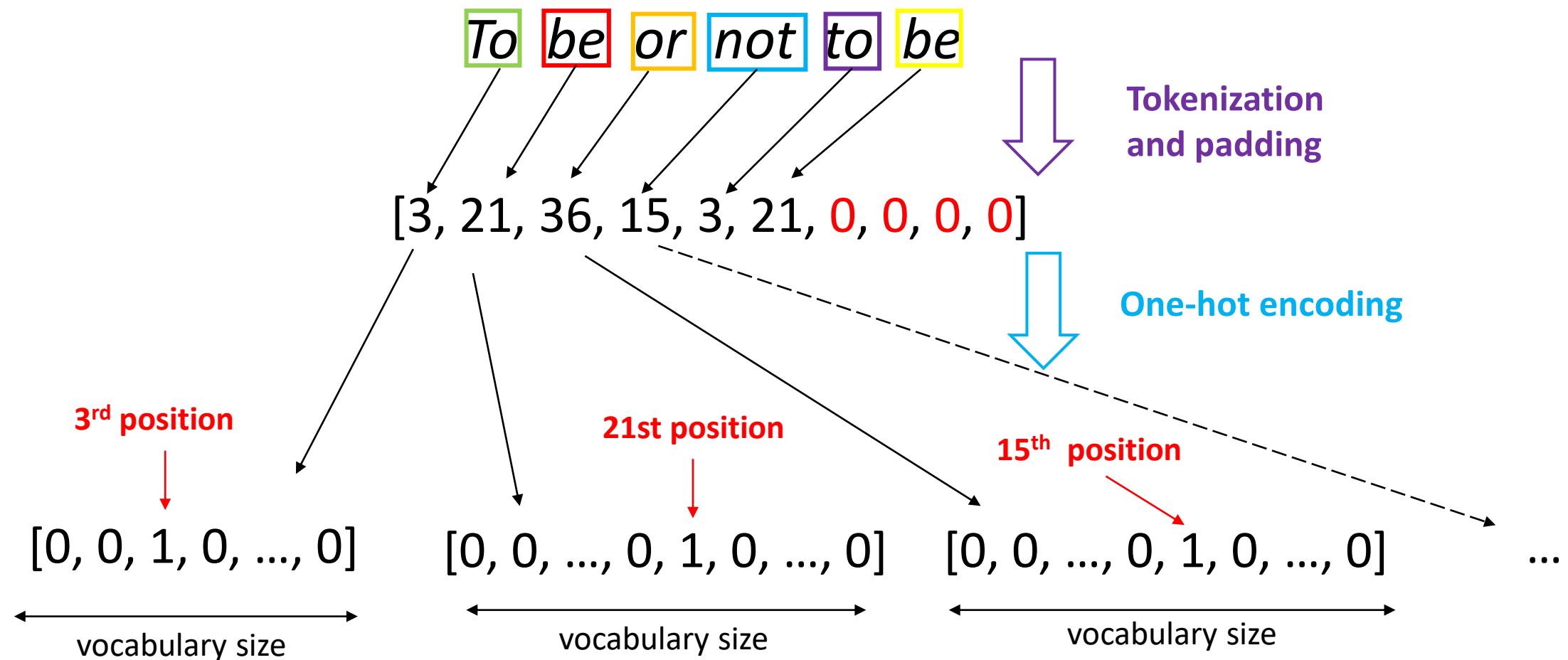
Normally, we cannot use the token sequence directly as inputs of the neural network because the numerical values are misleading

[3, 21, 36, 15, 3, 21, **0, 0, 0, 0**]

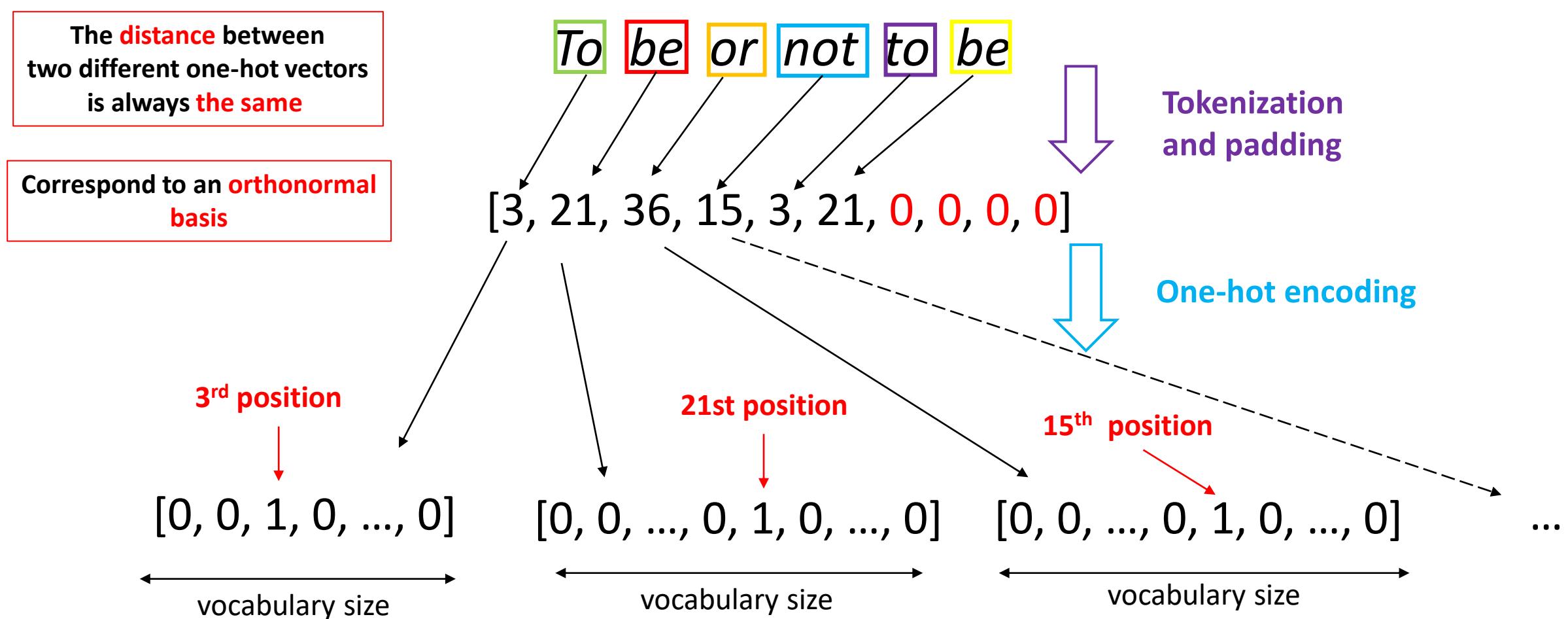
Indeed, all tokens are **independent**, the network should **NOT** consider that tokens “2” and “3” are closer than, let’s say, tokens “2” and “3000”

Because of that, a **more neutral** numerical representation is **needed** => **One-hot encoding**

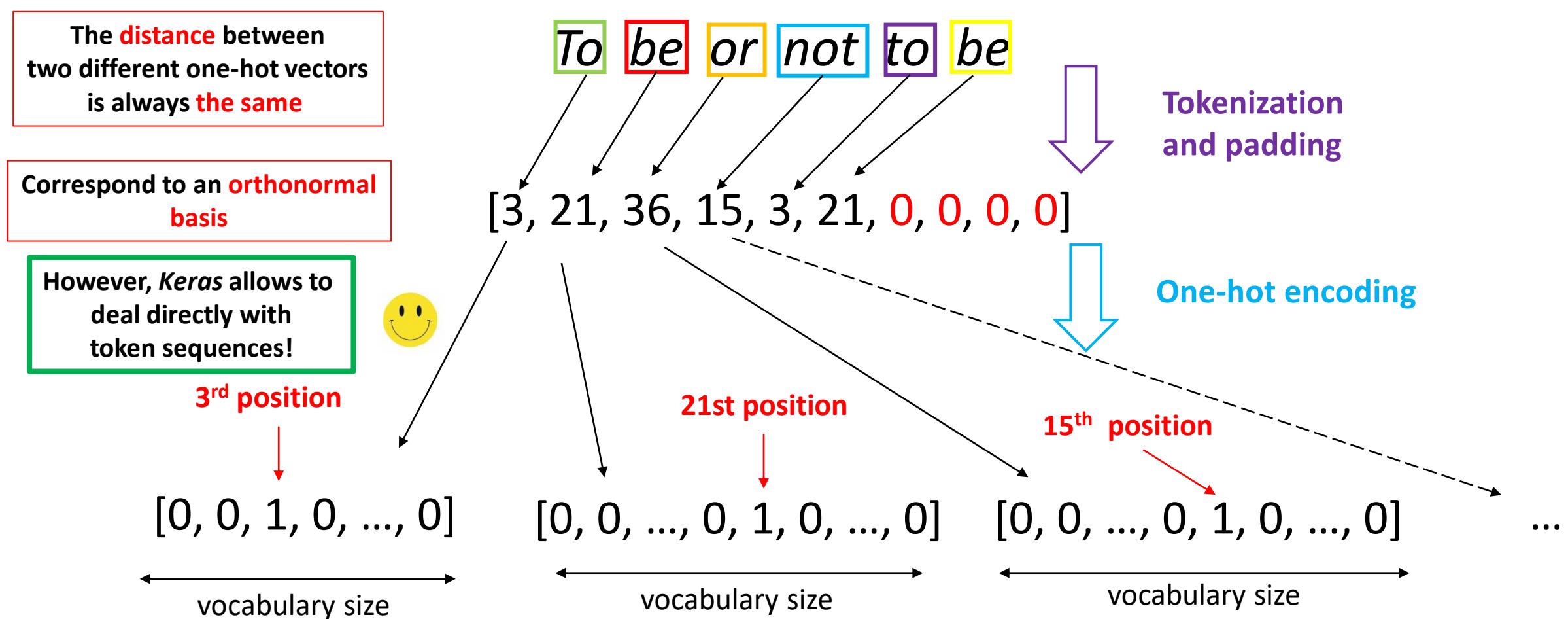
# Embedding: One-hot encoding step



# Embedding: One-hot encoding step



# Embedding: One-hot encoding step



# Embedding: *Embedding* Keras layer

In order to **train an embedding from scratch** (using **directly** the **token** sequences), one can use the ***Embedding* Keras layer**

```
# Model Definition with LSTM
model = tf.keras.Sequential([
    Embedding(vocab_size, embedding_dim, input_length=max_length),
    Bidirectional(LSTM(32)),
    Dense(6, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

# Embedding: *Embedding* Keras layer

In order to **train an embedding from scratch** (using **directly** the **token** sequences), one can use the ***Embedding* Keras layer**

```
# Model Definition with LSTM
model = tf.keras.Sequential([
    Embedding(vocab_size, embedding_dim, input_length=max_length),
    Bidirectional(LSTM(32)),
    Dense(6, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

The diagram shows the code for a model definition using an Embedding layer. Three annotations are present:

- An annotation for the parameter `vocab_size` points to the first argument of the `Embedding` layer. It is highlighted with a red box and labeled "Number of tokens in the vocabulary".
- An annotation for the parameter `embedding_dim` points to the second argument of the `Embedding` layer. It is highlighted with a red box and labeled "Size of the wanted embedding  
=> The choice here is up to the user".
- An annotation for the parameter `input_length=max_length` points to the third argument of the `Embedding` layer. It is highlighted with a red box and labeled "Fixed length of the padded sequences".

# Text sequence preprocessing: Exercise

*course3\_text\_sequence\_preprocessing\_ex.ipynb*

**Goal:** Get used to the basics of text sequence preprocessing before feeding an RNN/LSTM network **IMDB dataset**

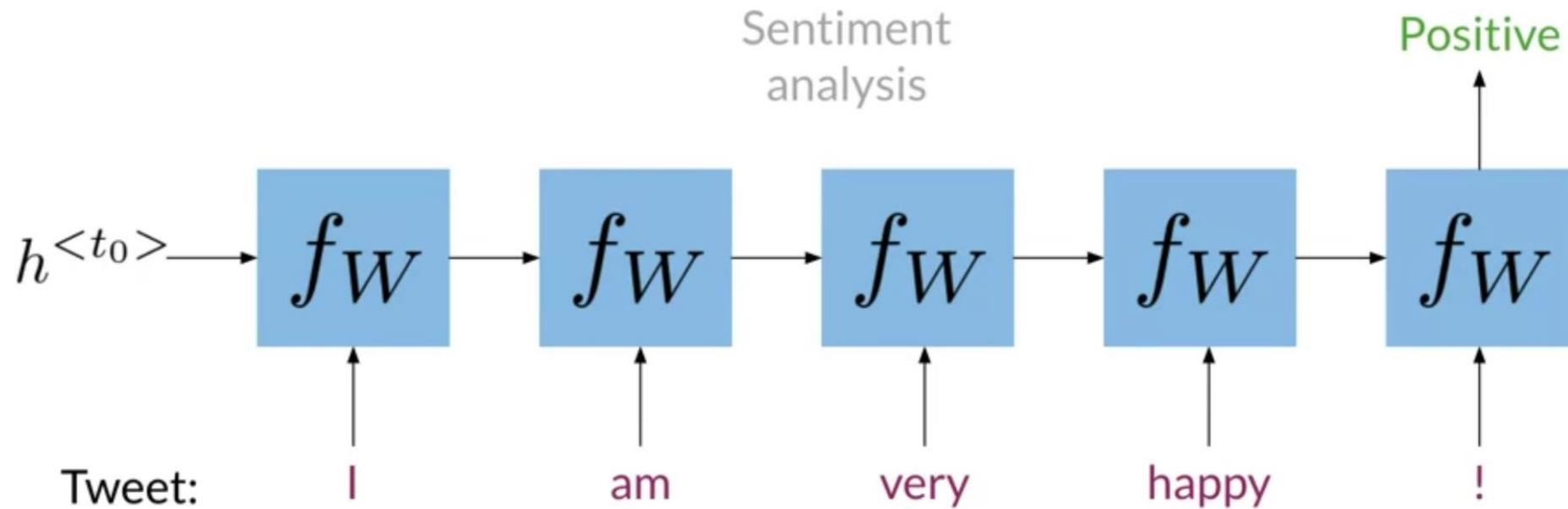
## Remarks:

- We will see how to compute **tokenization, padding and one-hot encoding** from the **Keras API**
- We use (again) **IMDB reviews dataset** for this exercise

# Use cases

# Use cases: Sentiment analysis

# Sentiment analysis: Illustration



# Sentiment analysis: Exercise

*course3\_sentiment\_analysis\_LSTM\_ex.ipynb*

**Goal:** Use of **text sequences** to feed neural networks such as **LSTM** for a **sentiment analysis** use case

## Remarks:

- The **main difference** with last **Course** sentiment analysis exercises is that we do not use an **average text embedding** as input any longer, but the **concatenated token embedding vector sequences**
- Another difference is that we do not use a pre-trained embedding model but **learn it from the dataset itself**

# Sentiment analysis : LSTM model

Use of a Bidirectional LSTM model

```
# Model Definition with LSTM
model = tf.keras.Sequential([
    Embedding(vocab_size, embedding_dim, input_length=max_length),
    Bidirectional(LSTM(32)),
    Dense(6, activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```



# Sentiment analysis : GRU model

## Use of a Bidirectional GRU model

```
# Model definition with GRU
model = tf.keras.Sequential([
    Embedding(vocab_size, embedding_dim, input_length=max_length),
    Bidirectional(GRU(32)),
    Dense(6, activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```



# Sentiment analysis : LSTM vs GRU

## LSTM

```
# Model Definition with LSTM
model = tf.keras.Sequential([
    Embedding(vocab_size, embedding_dim, input_length=max_length),
    Bidirectional(LSTM(32)),
    Dense(6, activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
<hr/>		
embedding_1 (Embedding)	(None, 120, 16)	160000
bidirectional_1 (Bidirectional)	(None, 64)	12544
dense_2 (Dense)	(None, 6)	390
dense_3 (Dense)	(None, 1)	7
<hr/>		
Total params: 172,941		
Trainable params: 172,941		
Non-trainable params: 0		

**172, 941**

## GRU

```
# Model definition with GRU
model = tf.keras.Sequential([
    Embedding(vocab_size, embedding_dim, input_length=max_length),
    Bidirectional(GRU(32)),
    Dense(6, activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
embedding (Embedding)	(None, 120, 16)	160000
bidirectional (Bidirectional)	(None, 64)	9600
dense (Dense)	(None, 6)	390
dense_1 (Dense)	(None, 1)	7
<hr/>		
Total params: 169,997		
Trainable params: 169,997		
Non-trainable params: 0		

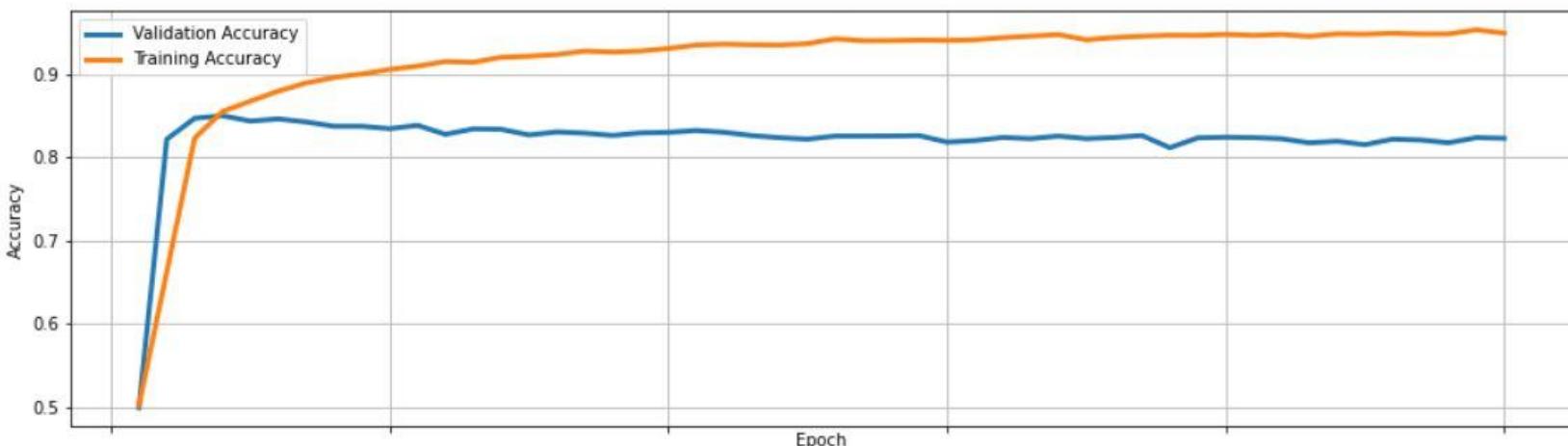
**169, 997**

A (little) less trainable parameters for the GRU model

# Sentiment analysis : Results

- Using a GRU model (with Dropout)
- 50 epochs

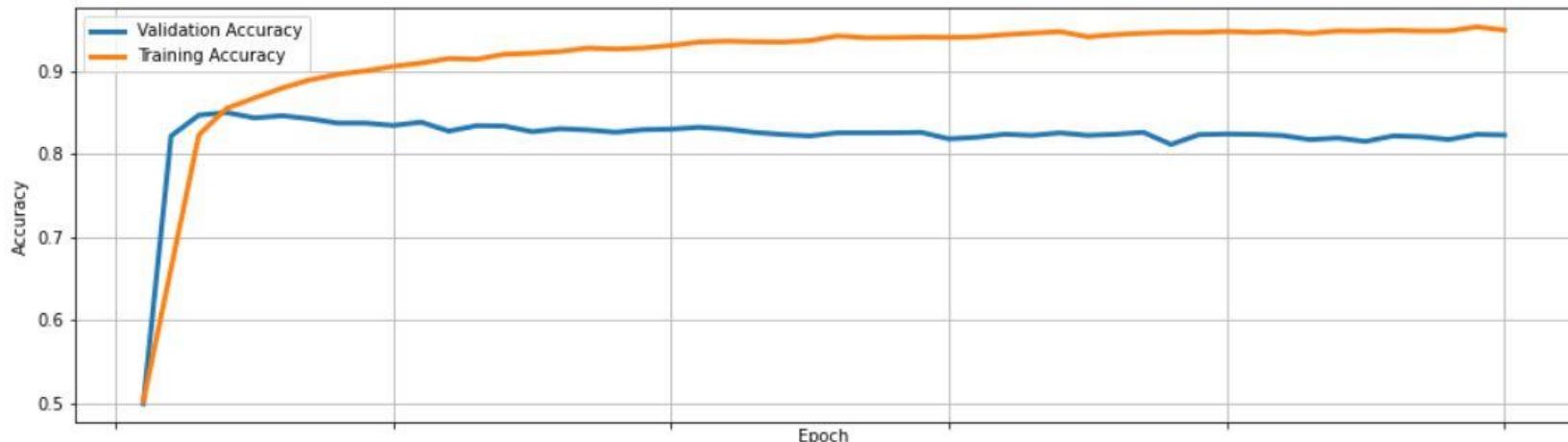
```
# Model definition with GRU
model = tf.keras.Sequential([
    Embedding(vocab_size, embedding_dim, input_length=max_length),
    Dropout(0.5),
    Bidirectional(GRU(32)),
    Dense(6, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```



# Sentiment analysis : Results

- Using a GRU model (with Dropout)
- 50 epochs

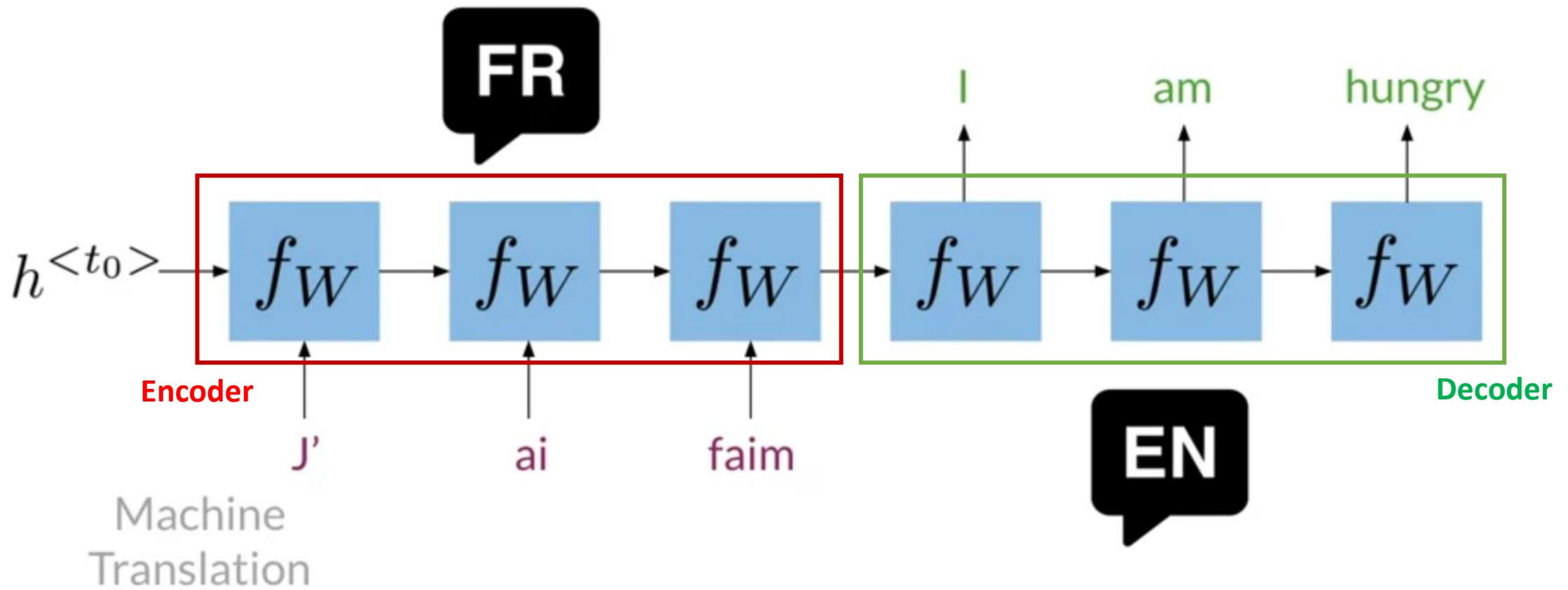
```
# Model definition with GRU
model = tf.keras.Sequential([
    Embedding(vocab_size, embedding_dim, input_length=max_length),
    Dropout(0.5),
    Bidirectional(GRU(32)),
    Dense(6, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```



You can definitely achieve better results  
by tweaking the model a little!

# Use cases: Translation

# Translation: Illustration



# Translation: Exercise

*course3\_translation\_LSTM.ipynb*

**Goal:** Have a look at how an English-French translation LSTM model can be trained

## Remarks:

- The model is an **adaptation** of some **articles** found on the internet (the references are given inside the notebook)
- The students **do not have to complete anything**, the model would be **too long to train** to work on it during the class
- The results obtain in the current notebook comes from a model which has **not converged yet**, so the results are currently quite **low**

# Translation : A closer look at the model

```
model = Sequential()
model.add(Embedding(eng_vocab_size, 512, input_length=eng_length, mask_zero=True))
model.add(LSTM(512))
#----- Choose either the RepeatVector or the Bidirectional
model.add(RepeatVector(fren_length))
#model.add(Bidirectional(LSTM(fren_length), return_sequences = True))
#-----
model.add(LSTM(512, return_sequences=True))
model.add(Dense(fren_vocab_size, activation='softmax'))
```

# Translation : A closer look at the model

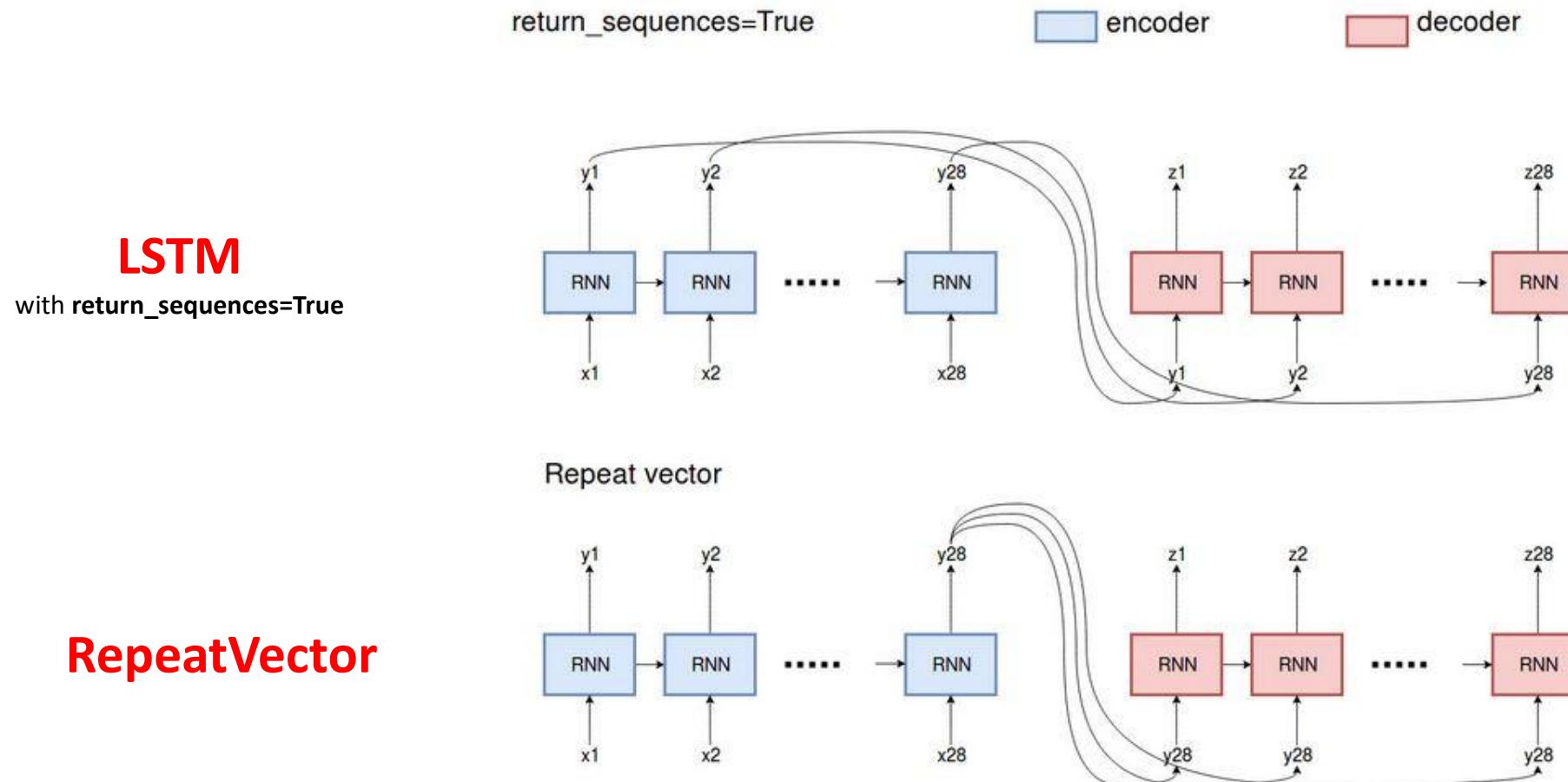
```
model = Sequential()
model.add(Embedding(eng_vocab_size, 512, input_length=eng_length, mask_zero=True))
model.add(LSTM(512))
#----- Choose either the RepeatVector or the Bidirectional
model.add(RepeatVector(fren_length))
#model.add(Bidirectional(LSTM(fren_length), return_sequences = True))
#-----
model.add(LSTM(512, return_sequences=True))
model.add(Dense(fren_vocab_size, activation='softmax'))
```

Be careful, this option is needed to feed the LSTM layer coming just after

Two possibilities on this **specific layer** between **RepeatVector** and **Bidirectional(LSTM)**

The goal is to feed the **LSTM layer** coming next after

# Translation: ReturnSequence vs RepeatVector

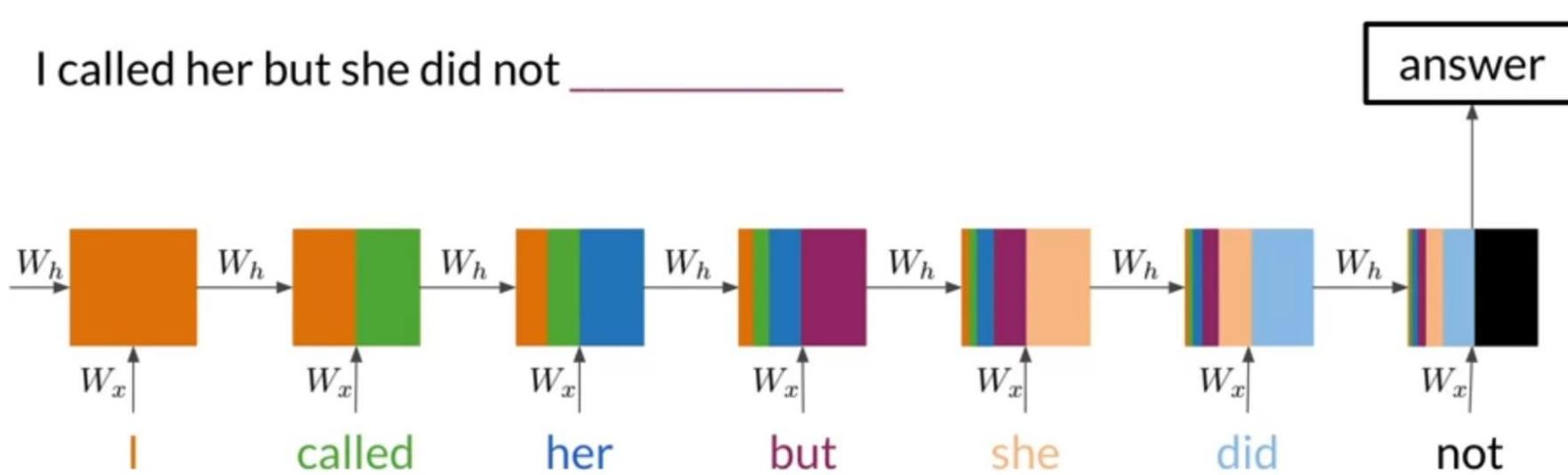


<https://stackoverflow.com/questions/51749404/how-to-connect-lstm-layers-in-keras-repeatvector-or-return-sequence-true>

**RepeatVector** will only return the **last value** of the output sequence to the next layer

# Use cases: Text generation

# Text generation: Illustration



# Text generation: Shakespeare sonnets

- We want to train a model able to generate text with the **same writing style as Shakespeare**
- For that, we use as database a compilation of his sonnets
- To train the model, we use sequences from the sonnet corpus
- As input, we give the **beginning of a sentence** and, as label, the **word coming just after**

# Text generation: Shakespeare sonnets

Example: the sentence ***To be or not to be*** gives **5 training sequences** to feed the model:

1. *To be*
2. *To be or*
3. *To be or not*
4. *To be or not to*
5. *To be or not to be*

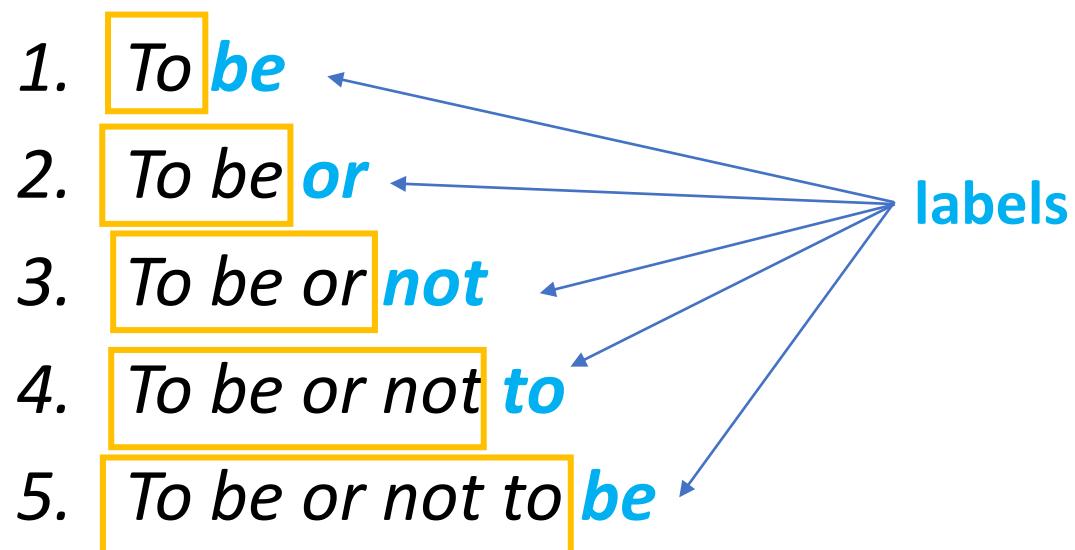
# Text generation: Shakespeare sonnets

Example: the sentence ***To be or not to be*** gives **5 training sequences** to feed the model:

1. *To be*
  2. *To be or*
  3. *To be or not*
  4. *To be or not to*
  5. *To be or not to be*
- labels**
- 
- The diagram illustrates the creation of five training sequences from the sentence 'To be or not to be'. A central point labeled 'labels' has five blue arrows pointing to each of the five numbered sequences on the left. The sequences are: 1. To be, 2. To be or, 3. To be or not, 4. To be or not to, and 5. To be or not to be.

# Text generation: Shakespeare sonnets

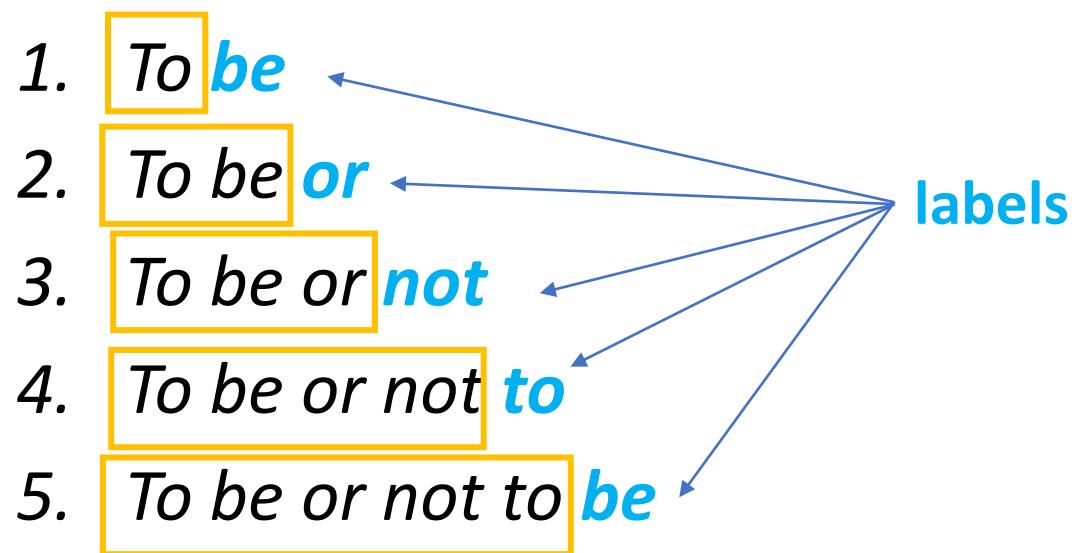
Example: the sentence ***To be or not to be*** gives **5 training sequences** to feed the model:



Input sequences for training the model

# Text generation: Shakespeare sonnets

Example: the sentence ***To be or not to be*** gives **5 training sequences** to feed the model:



Input sequences for training the model

A **n**-token sequence in the corpus gives **n-1** input sequences for **training**

# Text generation: Exercise

*course3\_text\_generation\_LSTM\_ex.ipynb*

**Goal:** To solve a complex NLP problem from the text processing to model tuning. In the end, we should get a model able of generating text with Shakespeare characteristic style!

## **Remark:**

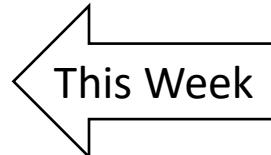
- We are not looking for especially good predictions, we only want the model being able to generate text looking like Shakespeare
- If the generated text is not too repetitive and looks like Shakespeare, even if it does not make much sense, you can consider it being good enough

# Take-away from Course 3

- For some **complex NLP** applications (**translation, text generation...**), it is needed to use **text** as temporal **sequences**
- The **Recurrent Neural Network** (**RNN**) are designed to process such data
- However, **RNN** have some **limitations** such as **loss of information for long sequences** as well as **vanishing gradient issues**
- A **good alternative** are the **LSTM** and their variants (**GRU**)
- Some **specific preprocessing steps** must be performed in order to use texts as sequences able to **train** that kind of model
- If the **train dataset** is **big enough** and the **task specific enough**, Keras allows to easily **train from scratch** **embeddings** (see *Embedding layer*)

# Course Schedule

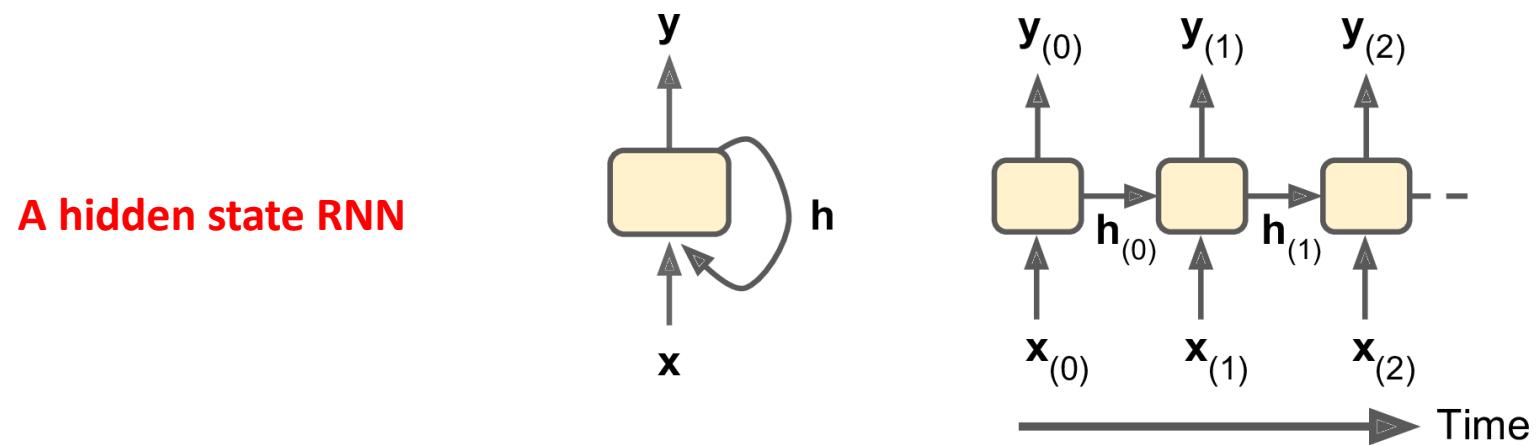
- **Course 1:** NLP introduction
- **Course 2:** Word embedding
- **Course 3:** Long Short Term Memory (LSTM) architecture
- **Course 4: “Attention” mechanism and Transformer architectures**
  - Encoder-decoder
  - Attention Mechanism
  - Transformer architectures
  - BERT model
  - Current state-of-the-art
  - Fine tuning use case on GPT-2
- **Course 5:** Chatbot introduction



# Previously in Course 3

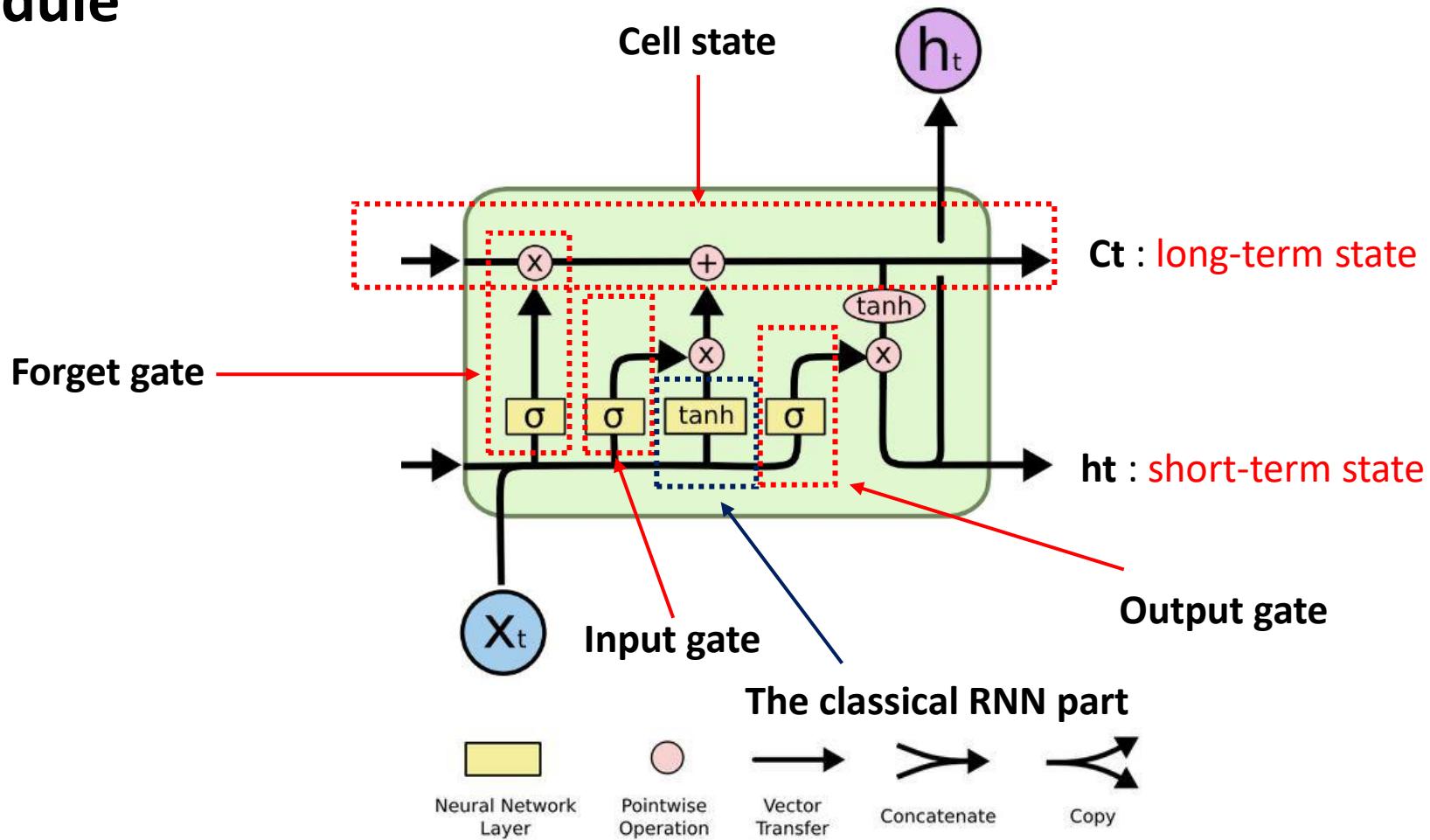
# Recurrent Neural Network (RNN): Definition

- A **RNN** is a kind of neural network dealing with **recurrent** connection
- Allows to deal with **temporal sequences**
- E.g., on the figure below, a sequence **X** is given as **input**, and we get a sequence **Y** as **output**
- **A cell hidden state  $h$  and the output  $y$  may be different**



# LSTM (Long Short Term Memory)

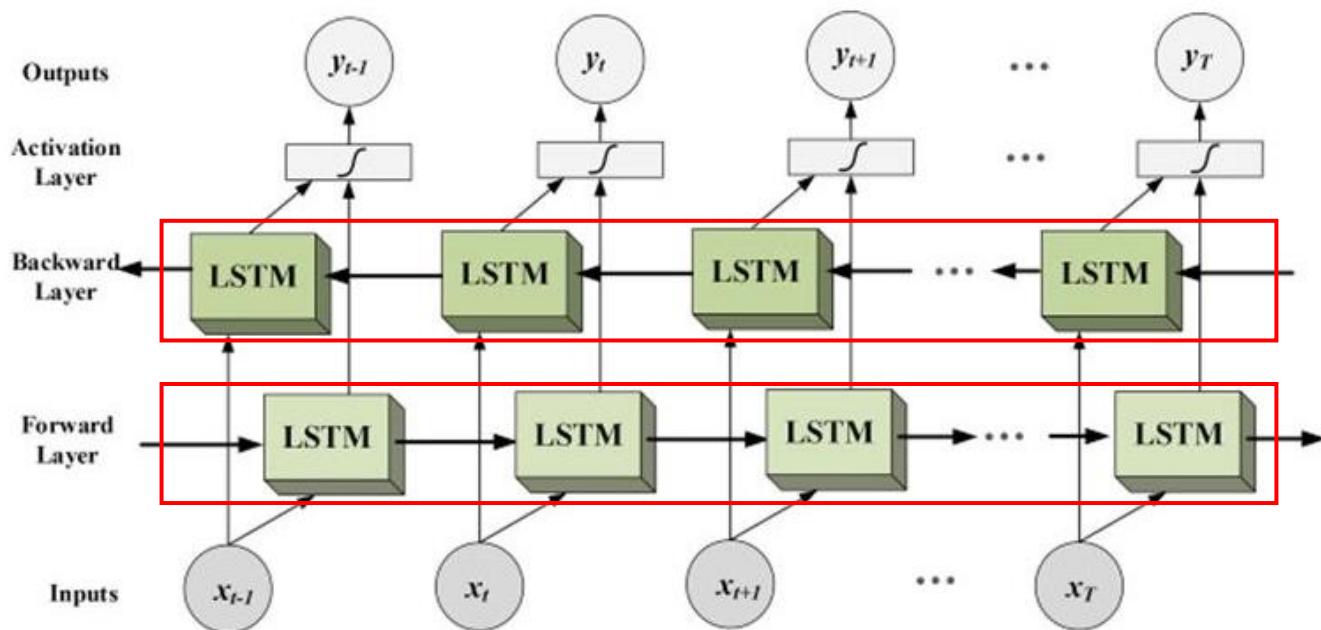
## LSTM module



# Bidirectional LSTM

In a **classic LSTM** architecture, the sentence/sequence is processed only in **one direction** (generally from the past to the present or future)

The **bidirectional** structure process the data in **both directions** => can be useful for some applications



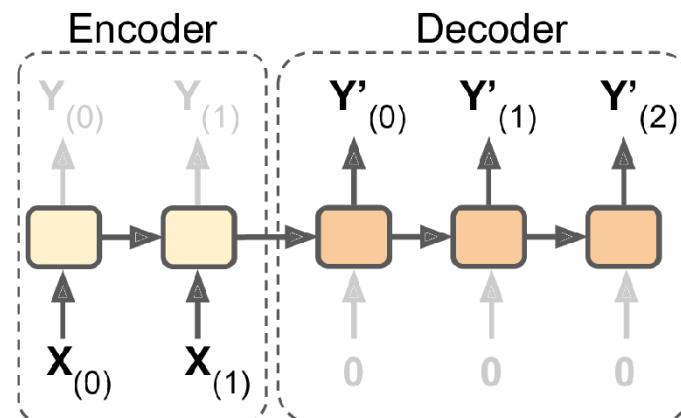
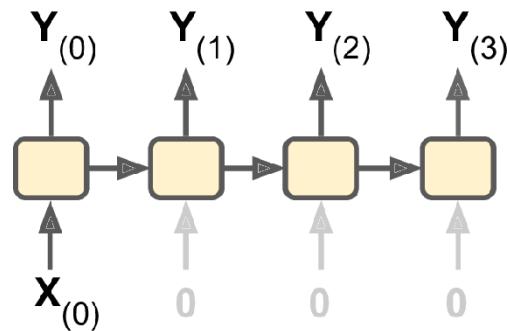
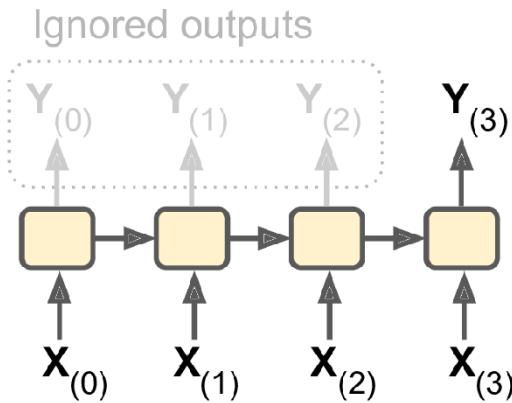
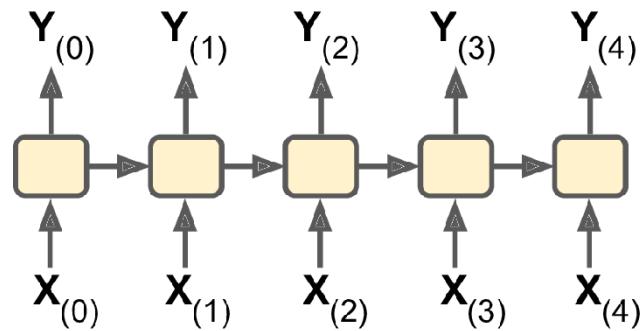
The Queen of the United Kingdom

The queen of hearts

The queen of bees

Here, the word queen will be encode differently if read in reverse order

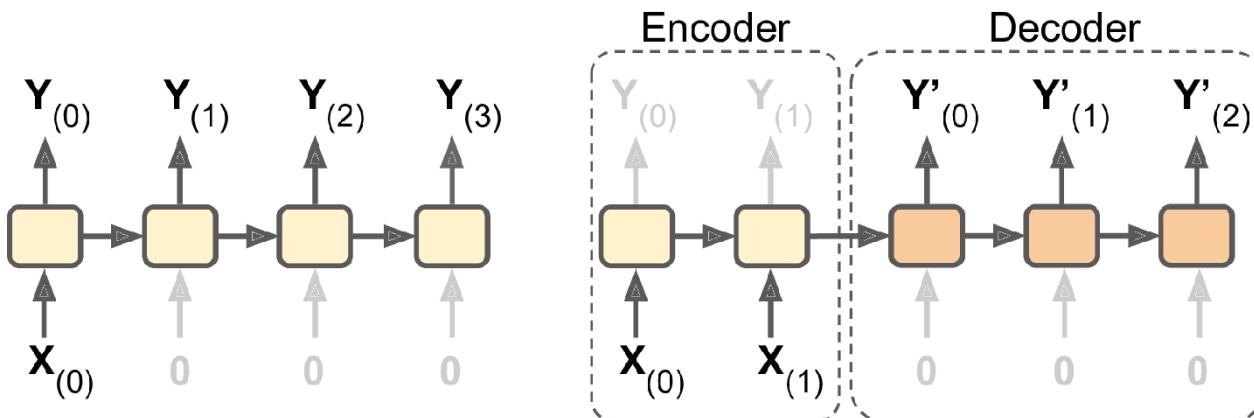
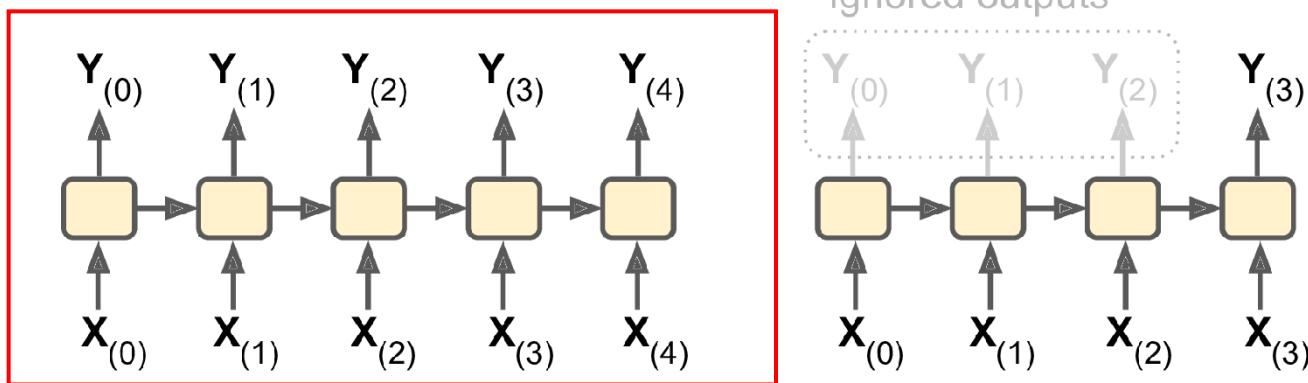
# Recurrent Neural Network (RNN): Use cases



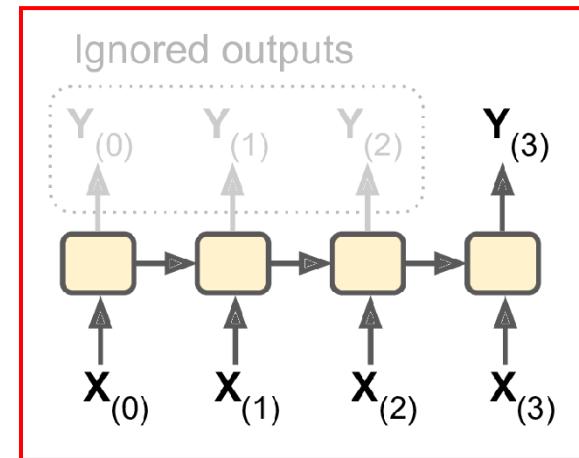
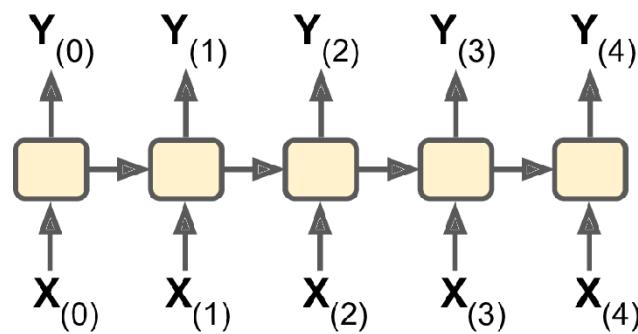
# Recurrent Neural Network (RNN): Use cases

**sequence-to-sequence**

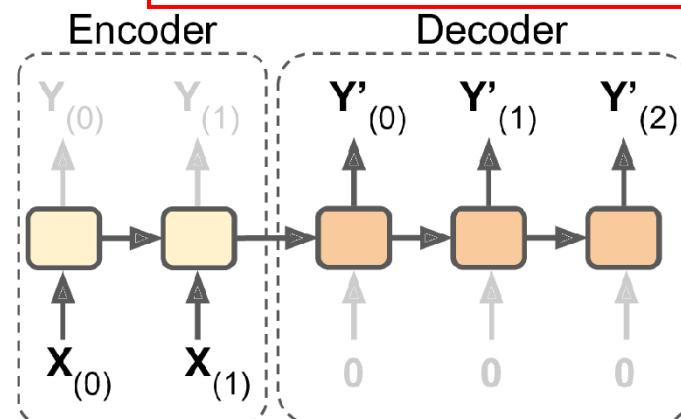
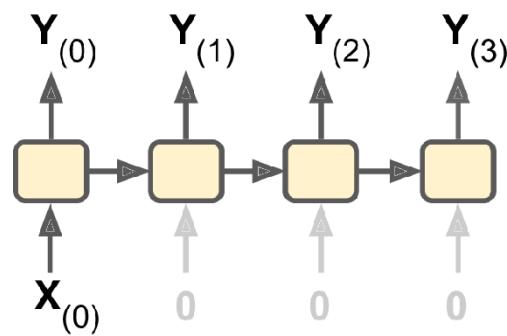
e.g. predicting time series



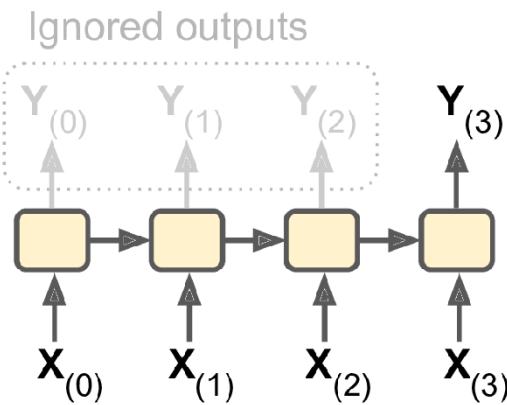
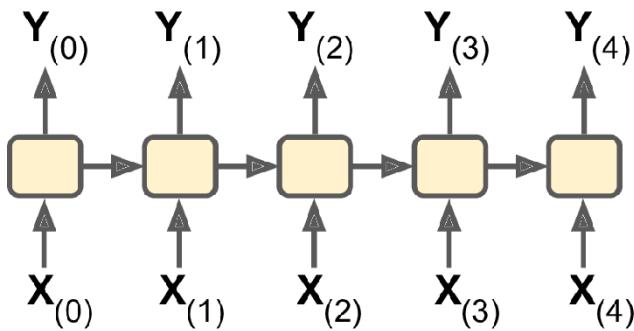
# Recurrent Neural Network (RNN): Use cases



**sequence-to-vector**  
e.g. sentiment analysis

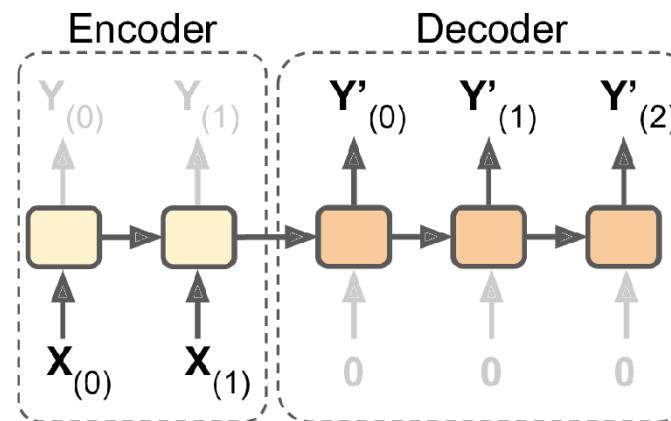
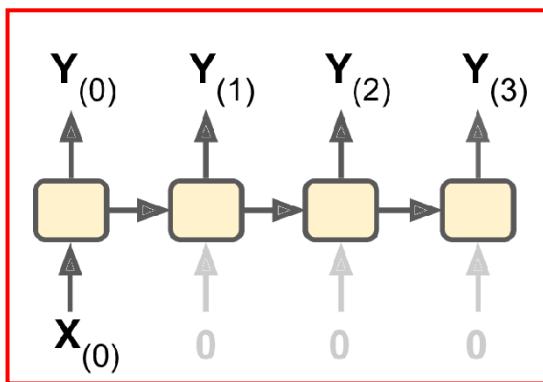


# Recurrent Neural Network (RNN): Use cases

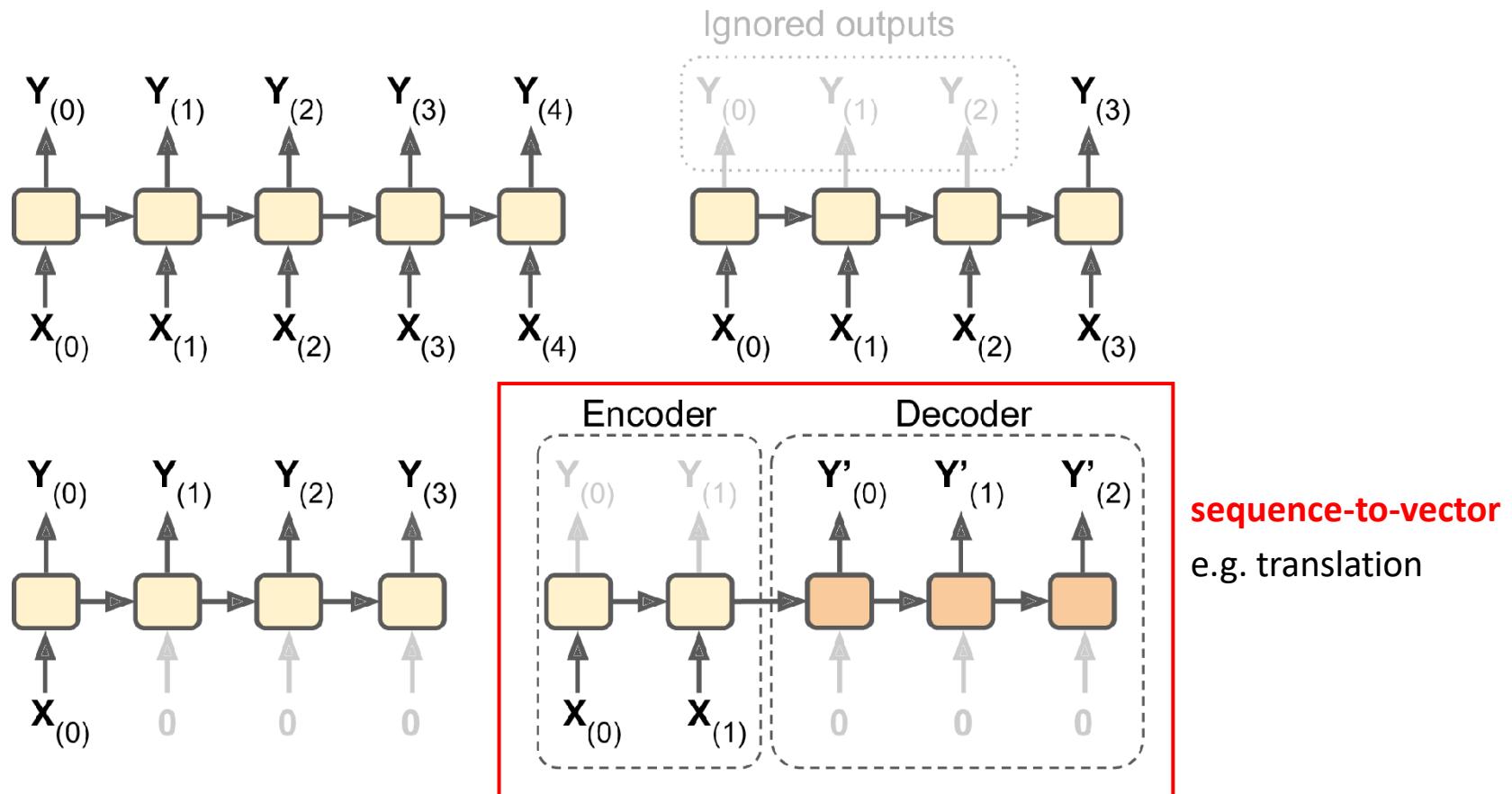


**vector-to-sequence**

e.g. image caption generation



# Recurrent Neural Network (RNN): Use cases



# Course 4: Attention Mechanism and Transformer Architectures

# Encoder-decoder

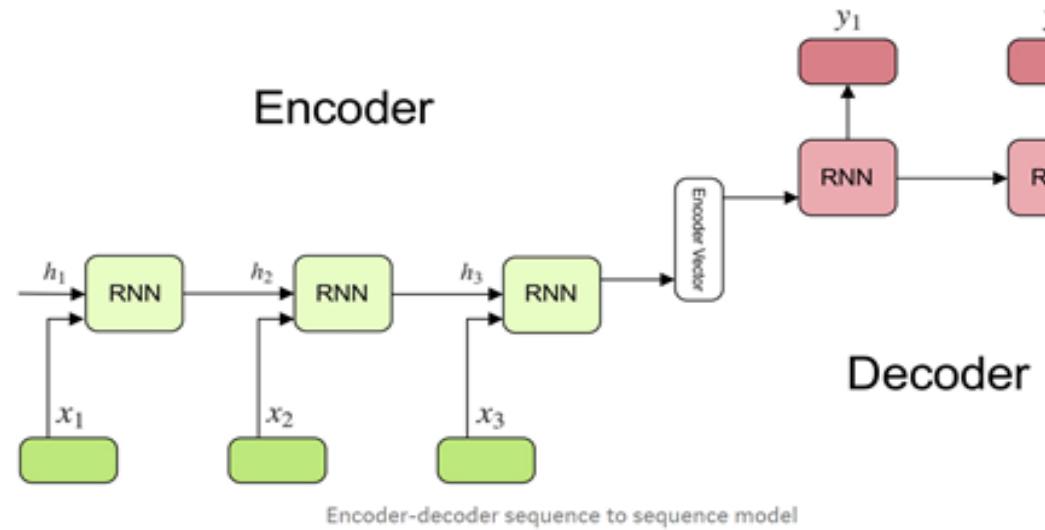
# Encoder-decoder: A core concept

## Principle

The goal is to project the input values into a smaller vector space before returning into the larger vector space expected

## Architecture

- An encoder
- An encoded vector
- A decoder



<https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346>

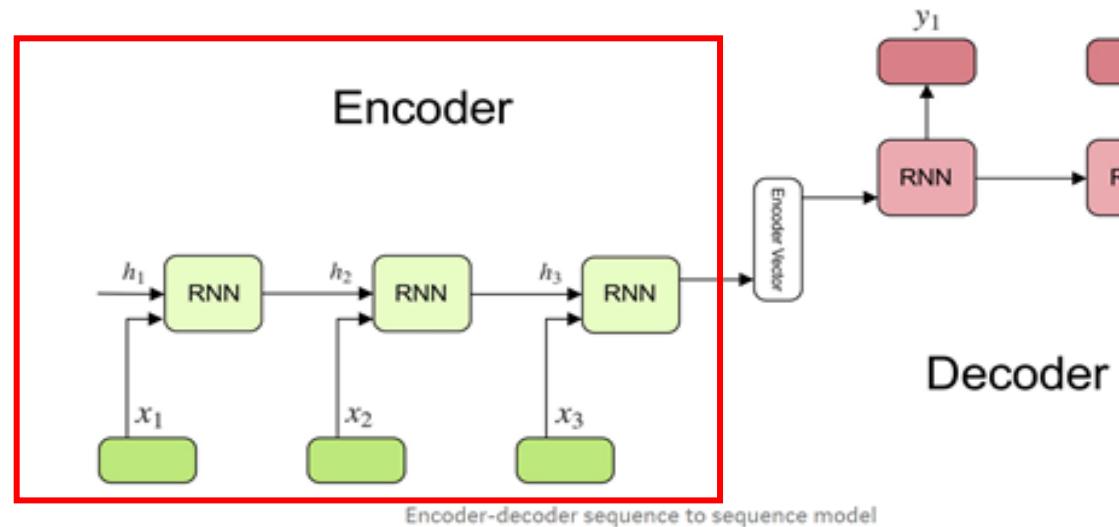
# Encoder-decoder: A core concept

## Principle

The goal is to project the input values into a smaller vector space before returning into the larger vector space expected

## Architecture

- **An encoder**
- An encoded vector
- A decoder



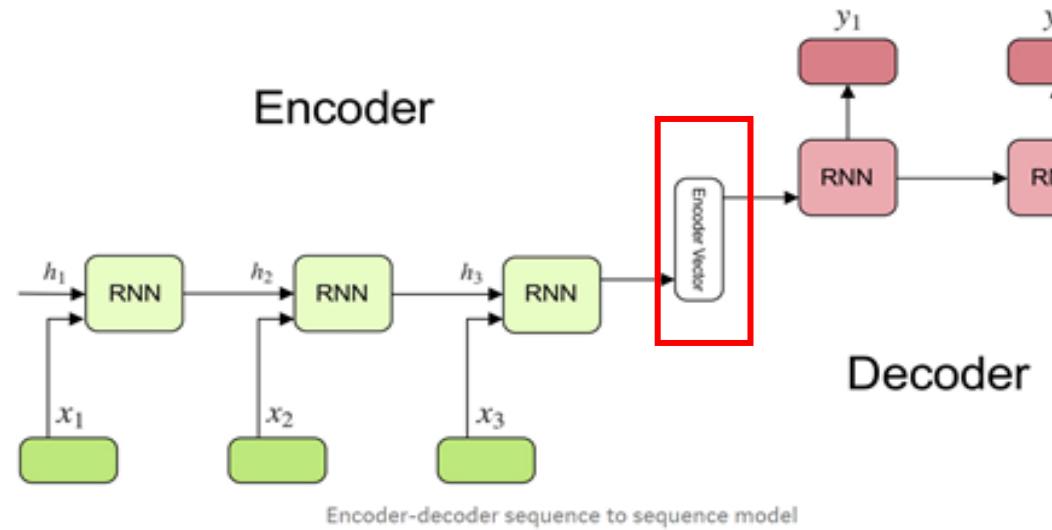
# Encoder-decoder: A core concept

## Principle

The goal is to project the input values into a smaller vector space before returning into the larger vector space expected

## Architecture

- An encoder
- **An encoded vector**
- A decoder



<https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346>

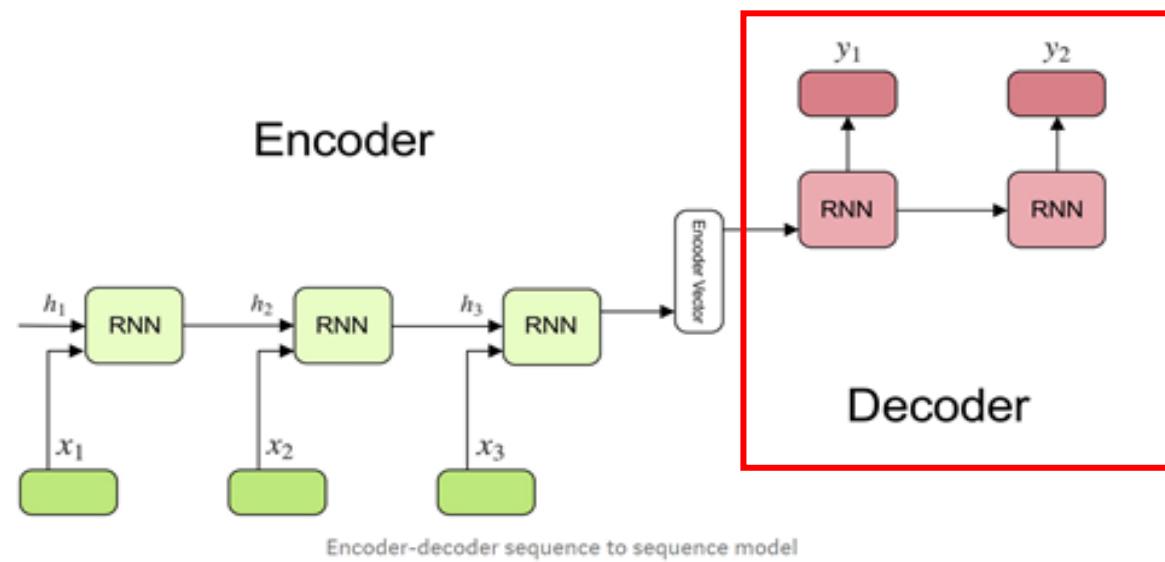
# Encoder-decoder: A core concept

## Principle

The goal is to project the input values into a smaller vector space before returning into the larger vector space expected

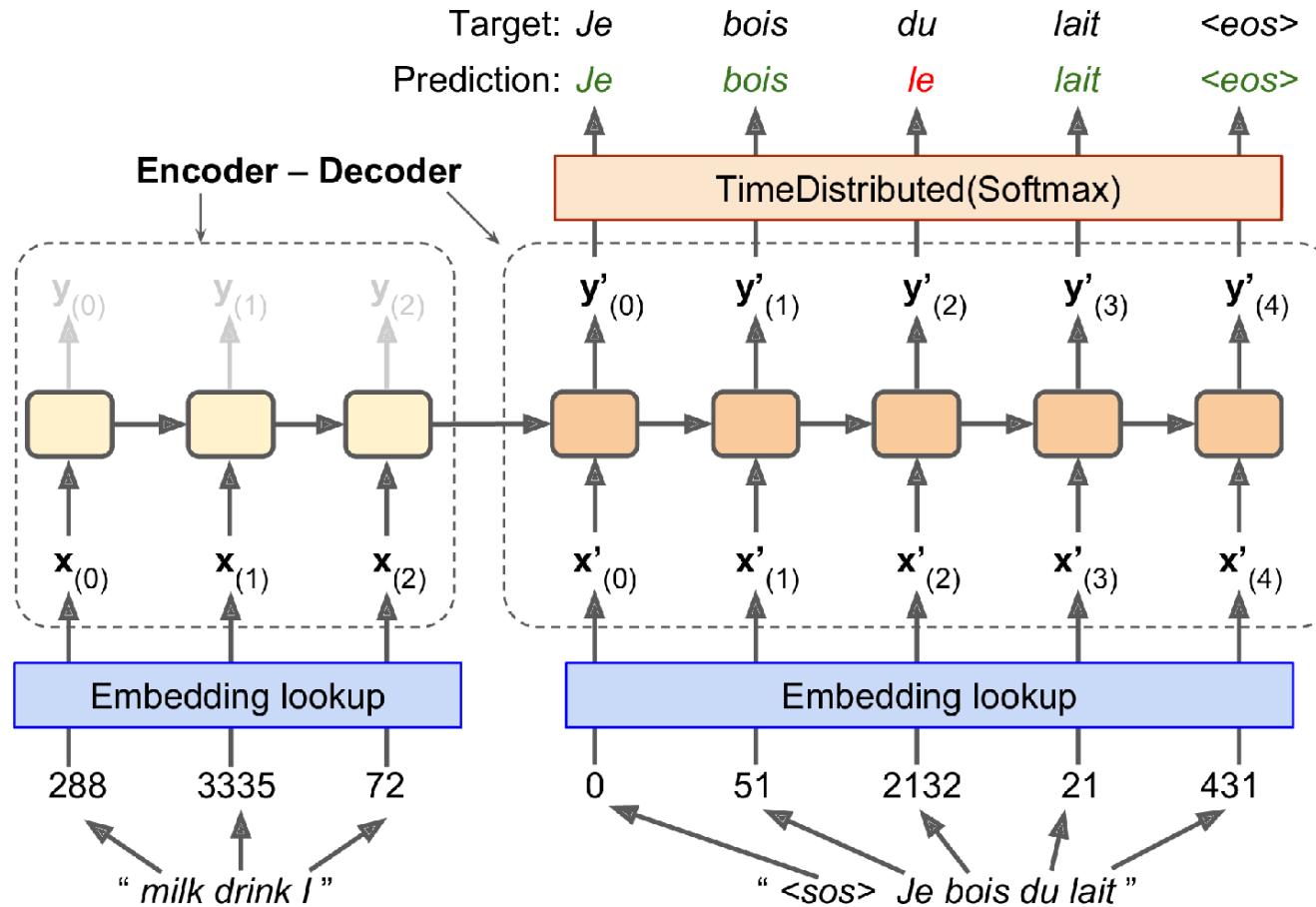
## Architecture

- An encoder
- An encoded vector
- **A decoder**



<https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346>

# Encoder-decoder: Translation example



# Encoder-decoder: Keras

```
import tensorflow_addons as tfa

encoder_inputs = keras.layers.Input(shape=[None], dtype=np.int32)
decoder_inputs = keras.layers.Input(shape=[None], dtype=np.int32)
sequence_lengths = keras.layers.Input(shape=[], dtype=np.int32)

embeddings = keras.layers.Embedding(vocab_size, embed_size)
encoder_embeddings = embeddings(encoder_inputs)
decoder_embeddings = embeddings(decoder_inputs)

encoder = keras.layers.LSTM(512, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_embeddings)
encoder_state = [state_h, state_c]

sampler = tfa.seq2seq.sampler.TrainingSampler()

decoder_cell = keras.layers.LSTMCell(512)
output_layer = keras.layers.Dense(vocab_size)
decoder = tfa.seq2seq.basic_decoder.BasicDecoder(decoder_cell, sampler,
                                                output_layer=output_layer)
final_outputs, final_state, final_sequence_lengths = decoder(
    decoder_embeddings, initial_state=encoder_state,
    sequence_length=sequence_lengths)
Y_proba = tf.nn.softmax(final_outputs.rnn_output)

model = keras.Model(inputs=[encoder_inputs, decoder_inputs, sequence_lengths],
                    outputs=[Y_proba])
```

# Encoder-decoder: Keras

```
import tensorflow_addons as tfa

encoder_inputs = keras.layers.Input(shape=[None], dtype=np.int32)
decoder_inputs = keras.layers.Input(shape=[None], dtype=np.int32)
sequence_lengths = keras.layers.Input(shape=[], dtype=np.int32)

embeddings = keras.layers.Embedding(vocab_size, embed_size)
encoder_embeddings = embeddings(encoder_inputs)
decoder_embeddings = embeddings(decoder_inputs)

encoder = keras.layers.LSTM(512, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_embeddings)
encoder_state = [state_h, state_c]

sampler = tfa.seq2seq.sampler.TrainingSampler()

decoder_cell = keras.layers.LSTMCell(512)
output_layer = keras.layers.Dense(vocab_size)
decoder = tfa.seq2seq.basic_decoder.BasicDecoder(decoder_cell, sampler,
                                                output_layer=output_layer)
final_outputs, final_state, final_sequence_lengths = decoder(
    decoder_embeddings, initial_state=encoder_state,
    sequence_length=sequence_lengths)
Y_proba = tf.nn.softmax(final_outputs.rnn_output)

model = keras.Model(inputs=[encoder_inputs, decoder_inputs, sequence_lengths],
                    outputs=[Y_proba])
```

encoder

decoder

model construction

allow to get both ht and Ct

to tell the decoder at each step what was the previous output

# Attention Mechanism

# Attention Mechanism: Introduction

## Problem to solve

- Even with a LSTM architecture, it can be difficult to characterize long sentences in an efficient way

## Attention mechanism principle

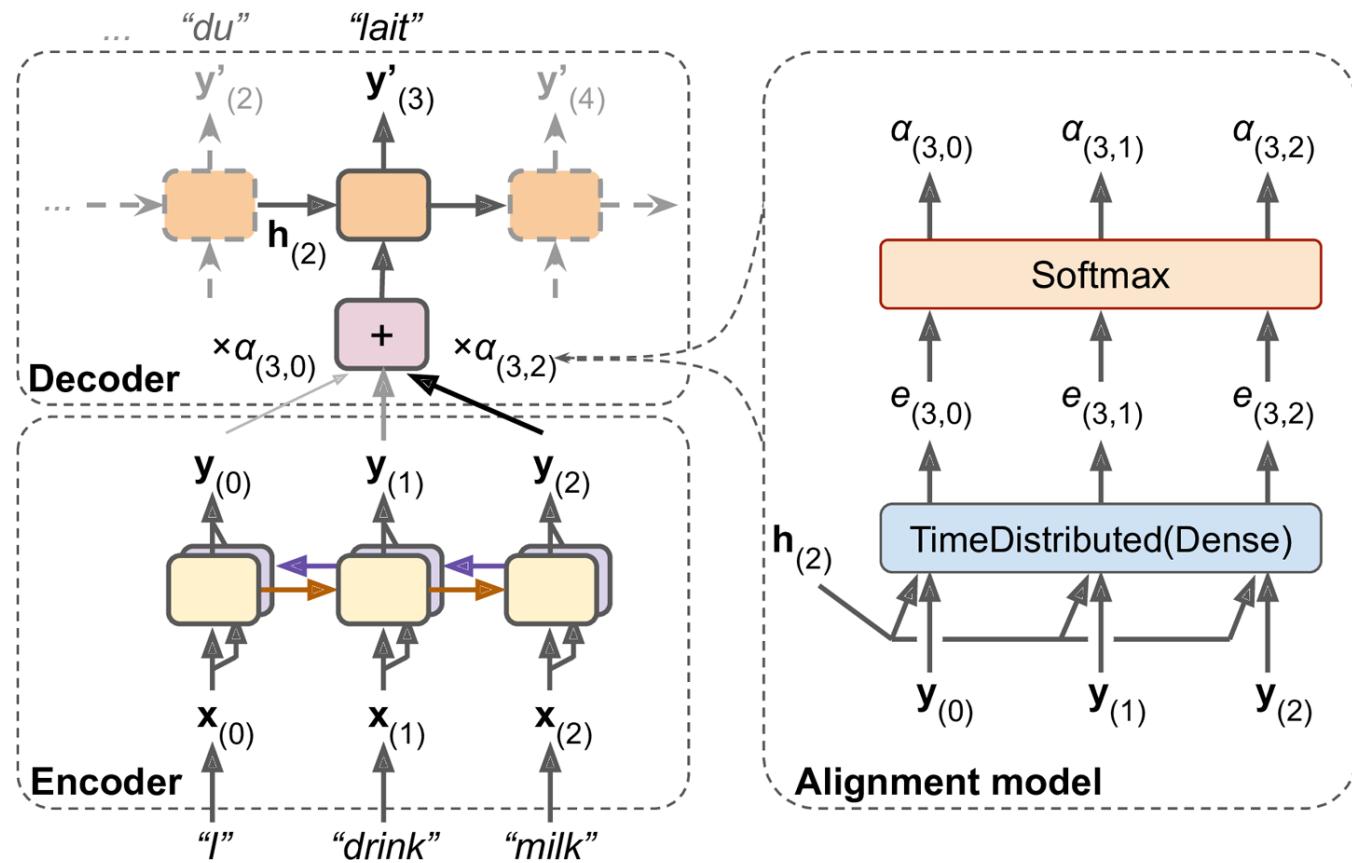
- To put more weight on specific word positions of the sentence depending **both** on the **context** and the **relative positions of words**

## Example

*Despite being from Uttar Pradesh, as he was **brought up** in Bengal, he is more comfortable in **Bengali**.*

To predict the word **Bengali**, the expressions **brought up** and **Bengal** must be associated to higher weights.

# Attention Mechanism: Translation illustration

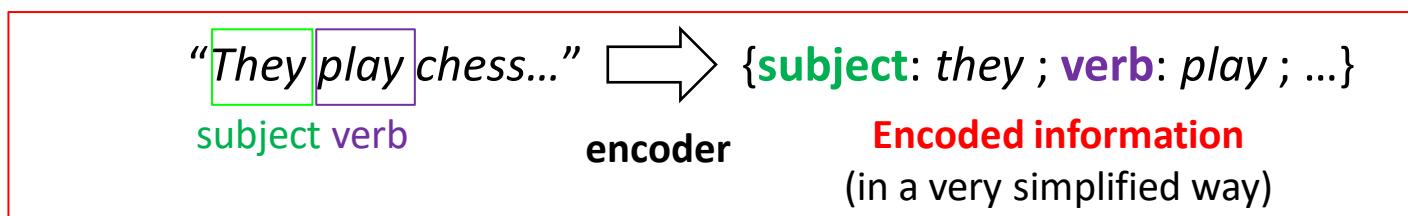


# Attention Mechanism: Get the intuition

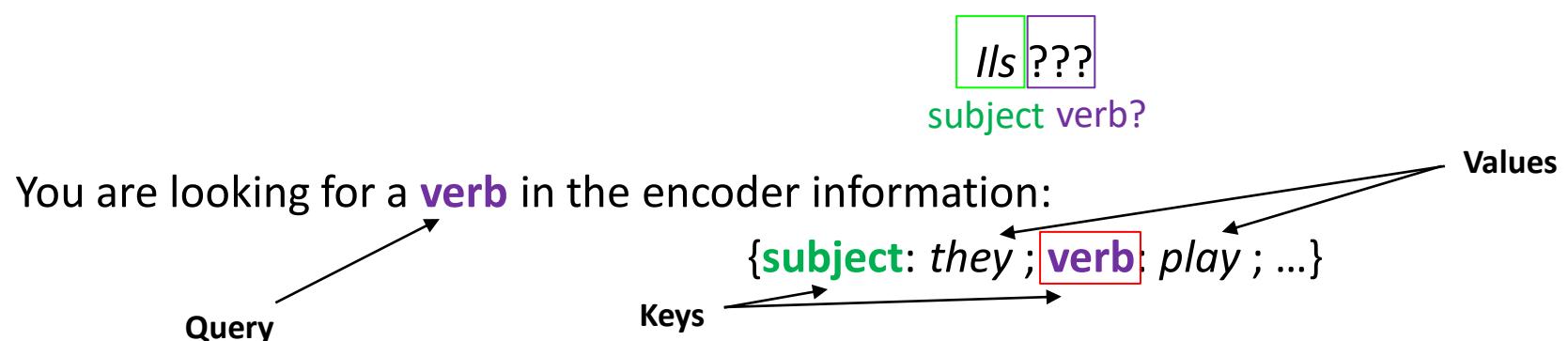
Imagine you want to translate the sentence using an **encoder-decoder**:

*“They play chess...”*

Actually, you expect the encoder to deal with it that way:



Just imagine you already translated the **first word** and are looking for the **next one**:



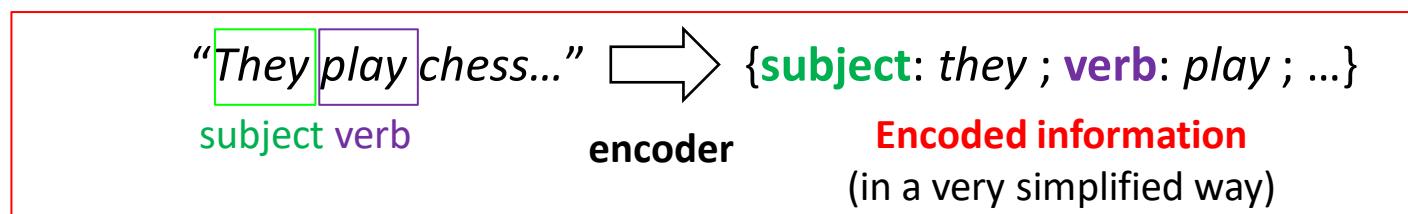
You are looking for a **verb** in the encoder information:

# Attention Mechanism: Get the intuition

Imagine you want to translate the sentence using an **encoder-decoder**:

*“They play chess...”*

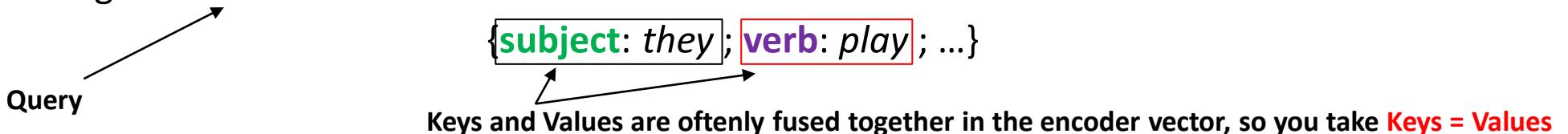
Actually, you expect the encoder to deal with it that way:



Just imagine you already translated the **first word** and are looking for the **next one**:

*Ils ???*  
subject verb?

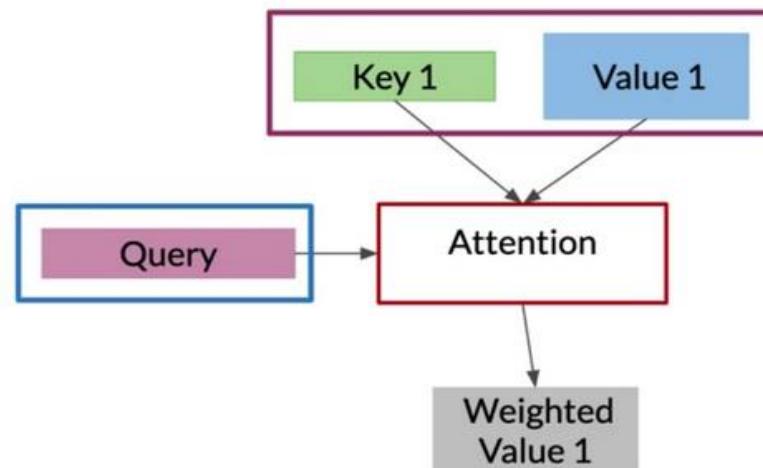
You are looking for a **verb** in the encoder information:



# Attention Mechanism: Principle

## Formalism

- To associate a **query** and a set of (**key-value**) tuples with an **output**
- The **query**, the (**key-value**) set and the **output** are all vectors
- The output corresponds to the **weighted sum** of values, with weights determined from a **compatibility function** between the query and the corresponding key



# Attention Mechanism: Formula

## Formula

The mathematical expression generally used to compute the attention function is

$$\text{softmax}(QK^T)V$$

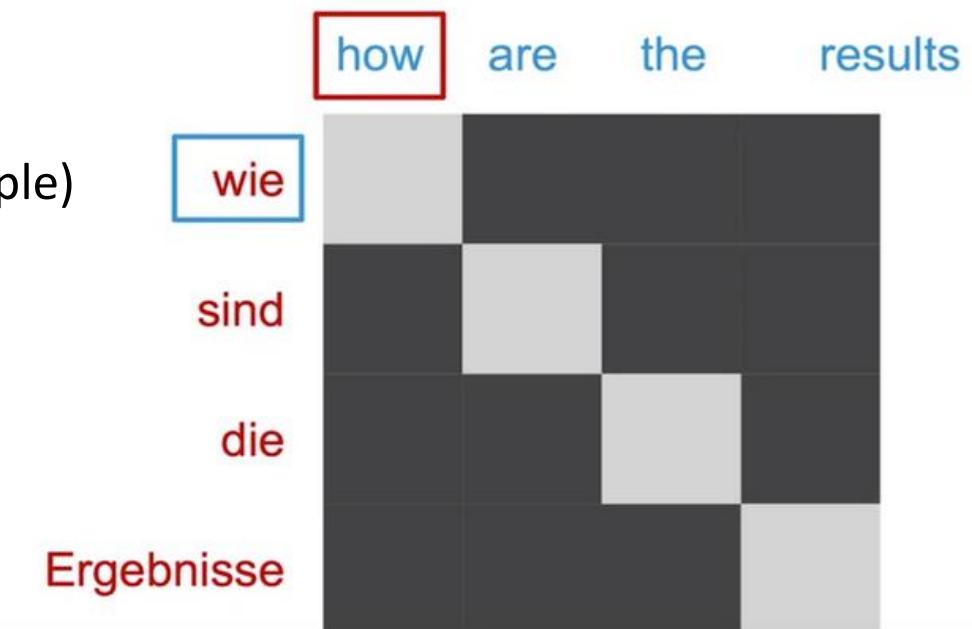
With  $Q$ ,  $K$  and  $V$  respectively the *query*, *key* and *value*, matrices.

The **softmax** function gives normalized values between 0 and 1

# Attention Mechanism: Illustration 1

Example for the case of English-German translation

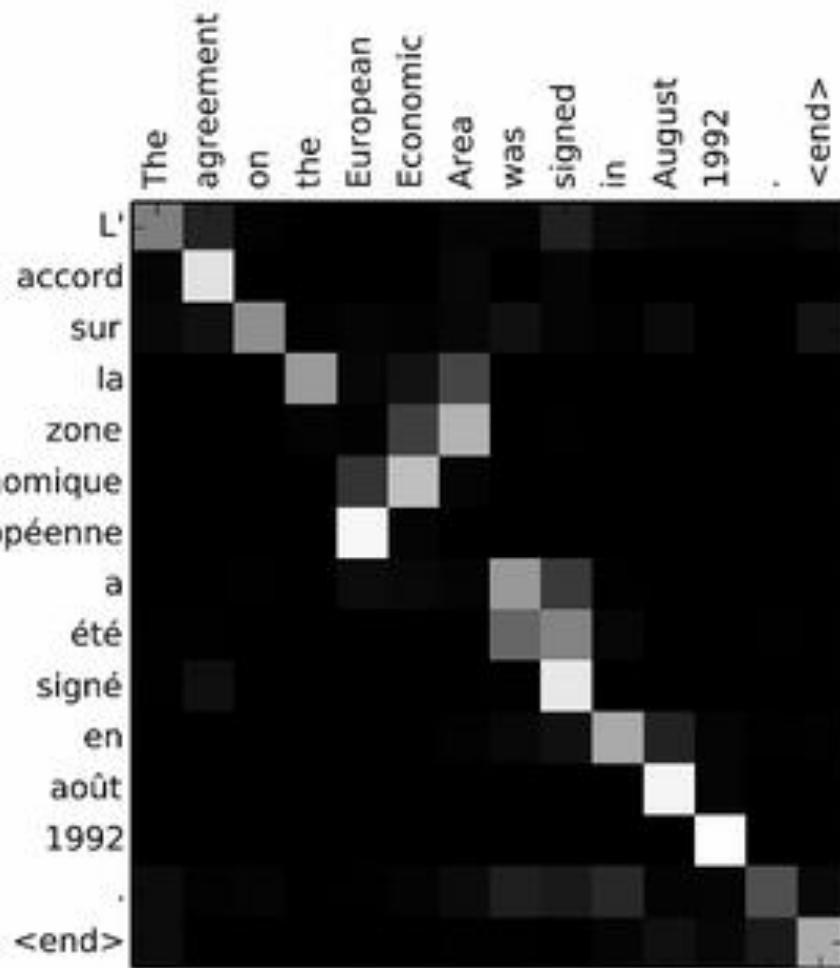
- The bright squares show where the algorithm « **look** » to translate a word
- Here, words in English and German **share the same order**
- So, the match is only on the **diagonal**
- No need to look other word positions (on this specific example)



# Attention Mechanism: Illustration 2

## Example for the case of English-French translation

- Here, the match is **not on a specific word only**
- The figure shows the contribution of each word
- **Several words** can contribute to the translation of another



# Attention Mechanism: Examples

## Intuition behind **query** and (**key**, **value**) concepts

Depends on the application

In NLP, usually, the **key** and **value** correspond to the **same vectors**:

- **Translation:** the **query** can be the encoded sentence vector in **one language** and both **key** and **value** can be the **encoded** sentence vector in the **other language**
- **Text similarity:** the **query** can be the sequence embedding of the **first piece of text** and both **key** and **value** can be the sequence embedding of the **second piece of text**

**However**, in some cases, the concepts key and value **can be different**:

- if you are looking for a video in *Youtube*, the **query** can be derived from the **text you entered**, the **keys** can correspond to **the video descriptions** and the **values** can be **the videos themselves**

# Attention Mechanism: Keras Layer

# An Attention Layer is available in Keras

Here we can see an illustration for translation

# Attention Mechanism: Keras Layer

An Attention Layer is available in Keras

Here we can see an illustration for translation

```
# Attention layer
attention_layer = AttentionLayer(name='attention_layer')
attention_out, attention_states = attention_layer({
    "values": encoded_en,
    "query": encoded_fr})
```

As usual in that kind of context,  
the **keys** are implicitly the same  
as the **values** (the encoded  
English sentences)

Here, the **values** correspond to  
the encoded **English** sentences

Here, the **queries** correspond to  
the encoded **French** sentences

# Visual Attention: Illustration

**Goal:** Generate automatically the caption of a picture with an attention mechanism



*“A woman is throwing a frisbee in a park”*

# Visual Attention: Illustration

**Goal:** Generate automatically the caption of a picture with an attention mechanism



*A woman is throwing a **frisbee** in a park*

# Transformer architecture

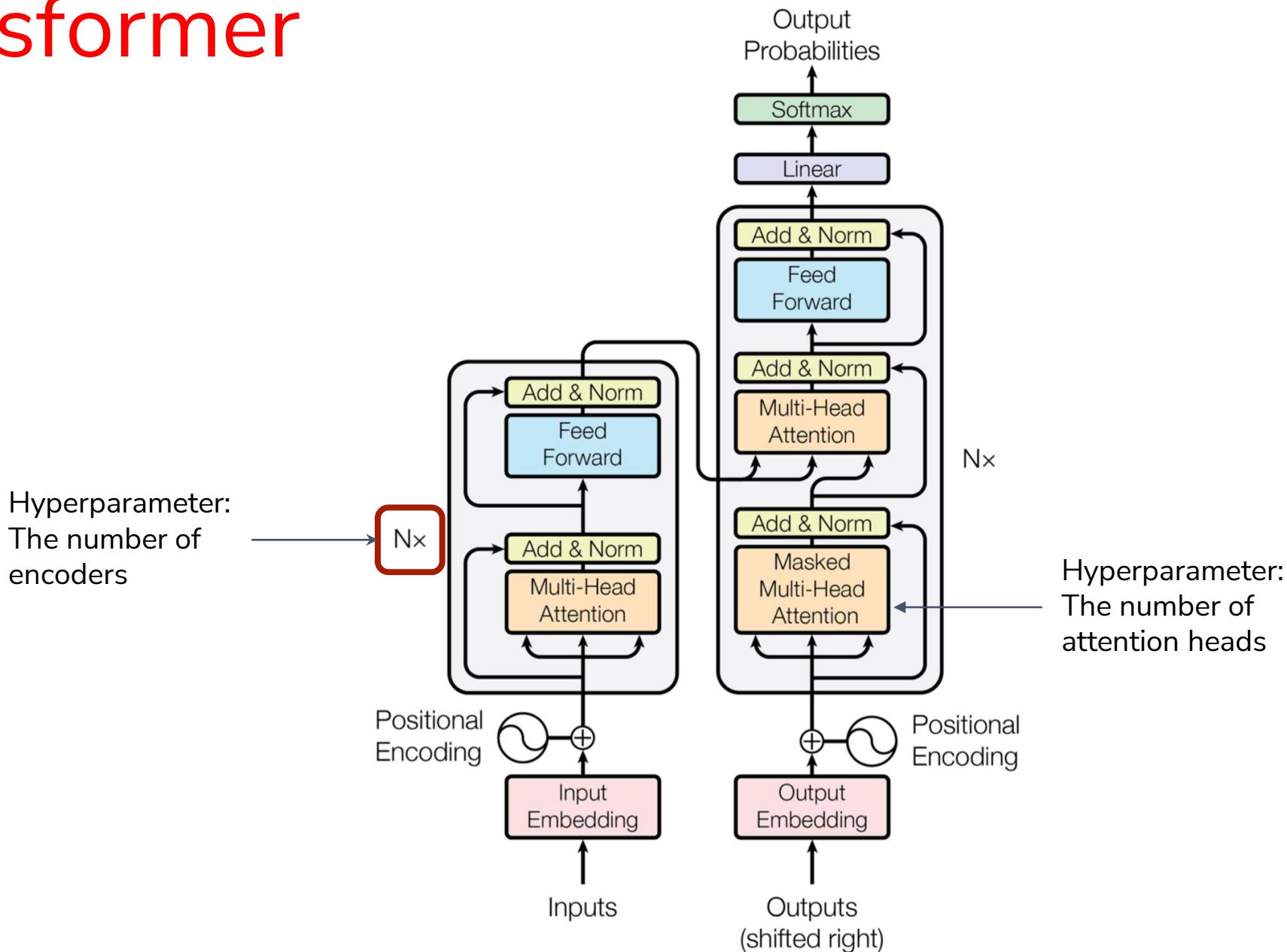
# Transformer: Introduction

## Principle

- A quite new deep learning model (2017), introduced in the **game changing** article *Attention is all you need*
- Relies heavily on the **Attention mechanism**
- **Does not need to process a sequence in a specific order**
- Solves the issue of keeping into memory the information related to distant words (and that even **without LSTM**)
- Makes possible a significant use of **parallelization** computing
- At every moment, the algorithm can access the complete set of the successive states visited during the procedure

The main idea is that the **Attention mechanism alone**, without any recurrent sequential procedure, **is powerful enough to reach the state of the art**

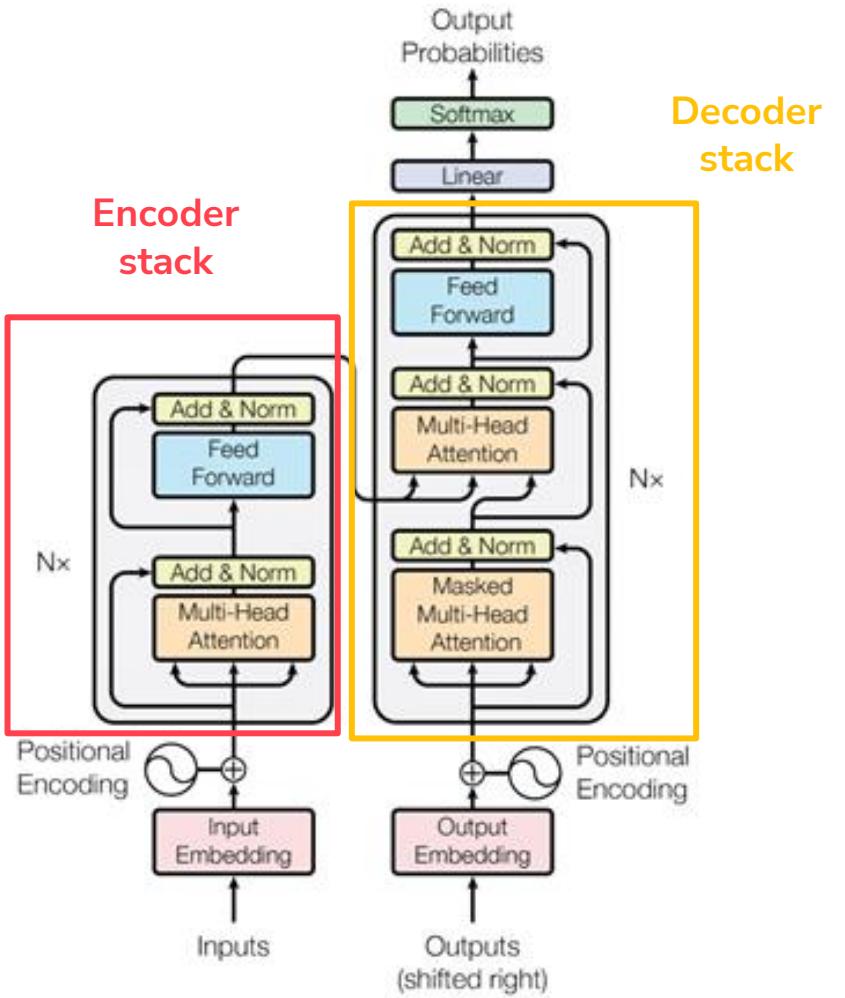
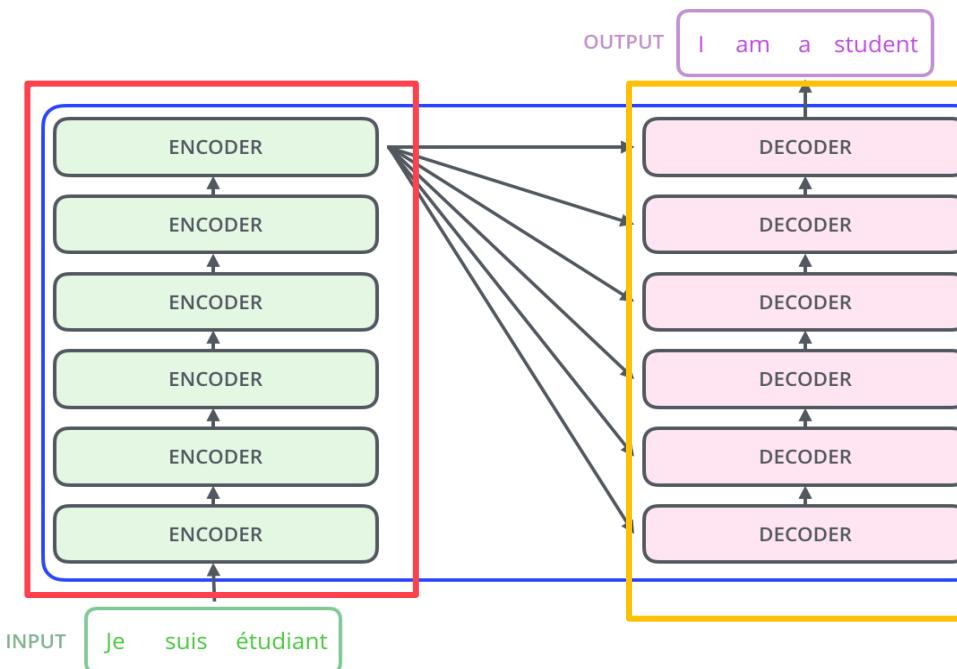
# Transformer



# Transformer: Architecture

The Transformer is made of two main components:

- A stack of **identical encoders** (independents form each other)
- Followed by a stack of **identical decoders** (independents from each other)

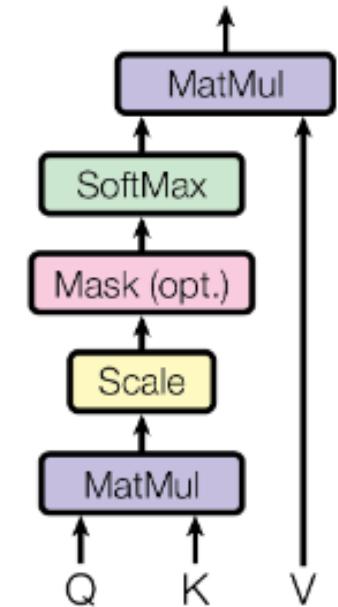


# Transformer: Scaled Dot-Product Attention

## Formula

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$d_k$  corresponds to the query and key dimension



- The **normalization** prevents the result of the dot-product to reach high values and, doing so, **prevents the gradient to vanish** into small values
- An **additive** variant of the attention function exists, but the dot-product version is preferred here for the efficiency of the matrix calculations it allows

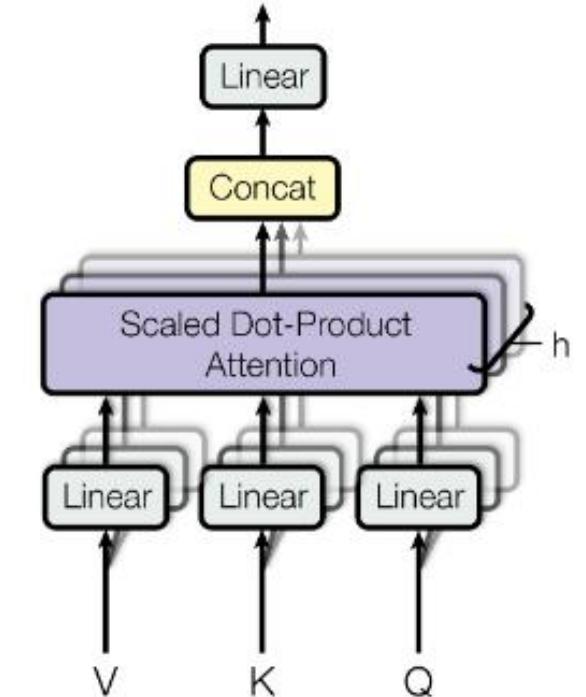
# Transformer: Multi-Head Attention

## Principle

- **Several Attention** layers computed in parallel
  - the queries, keys and values are projected  $h$  times
  - the  $h$  results are concatenated
  - then projected one last time
- Enable the use of different sub-space information
- The mathematical expression is the following

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where  $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

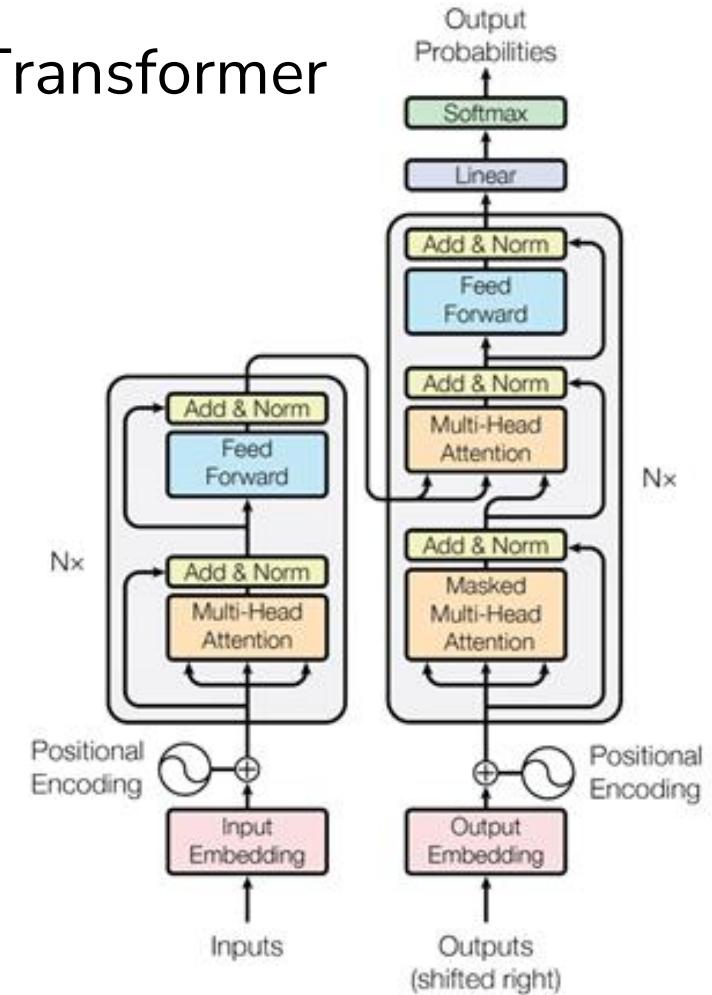


Where the  $W_i^*$  corresponds to the respective projection matrices

# Transformer: Multi-Head Attention

The **Multi-Head Attention** is used three times in the Transformer

- Inside the encoder
- Inside the decoder
- At the encoder-decoder interface

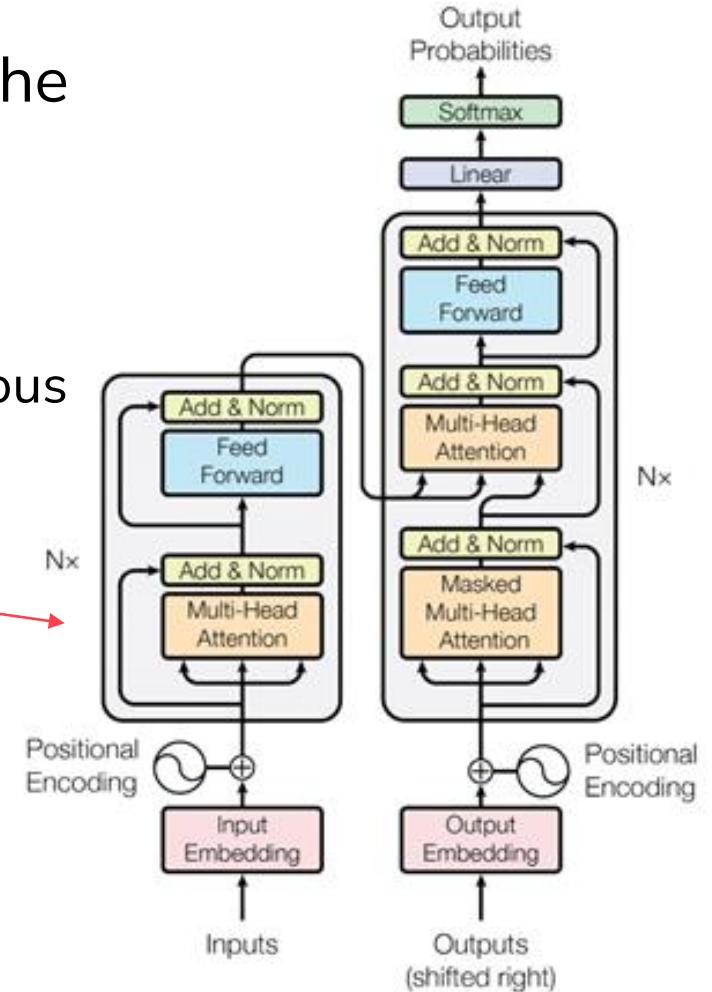


# Transformer: Multi-Head Attention

The **Multi-Head Attention** is used three times in the Transformer

- Inside the encoder
  - the queries, keys and values all come from the previous encoder output
- Inside the decoder
- At the encoder-decoder interface

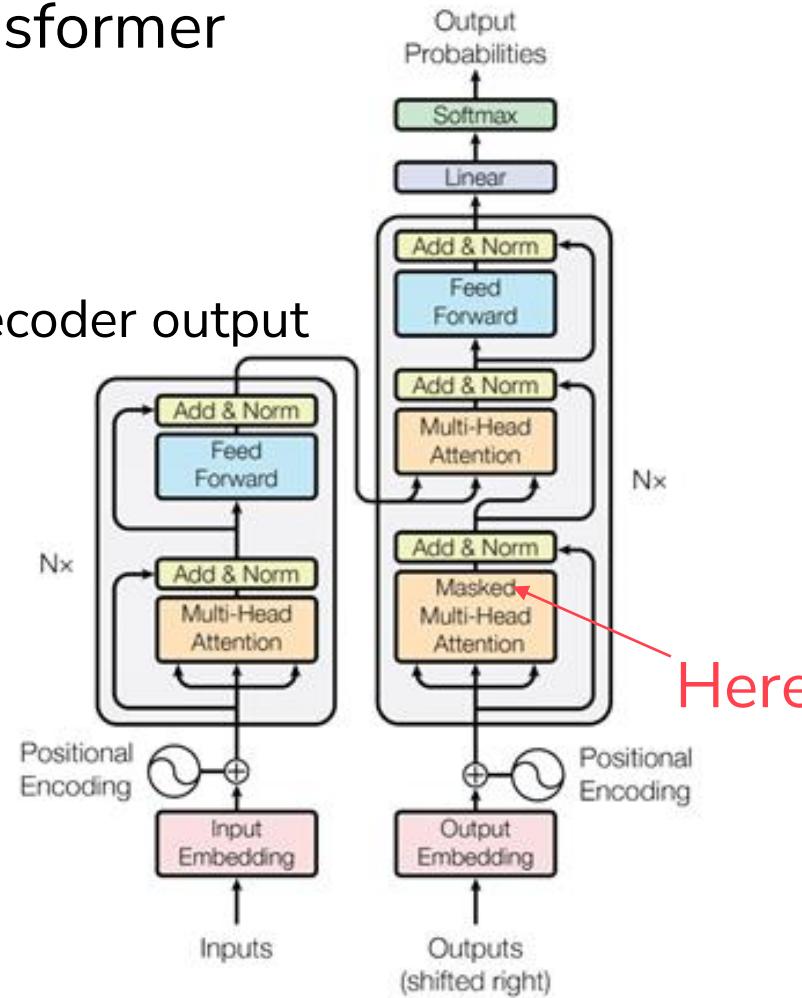
Here



# Transformer: Multi-Head Attention

The **Multi-Head Attention** is used three times in the Transformer

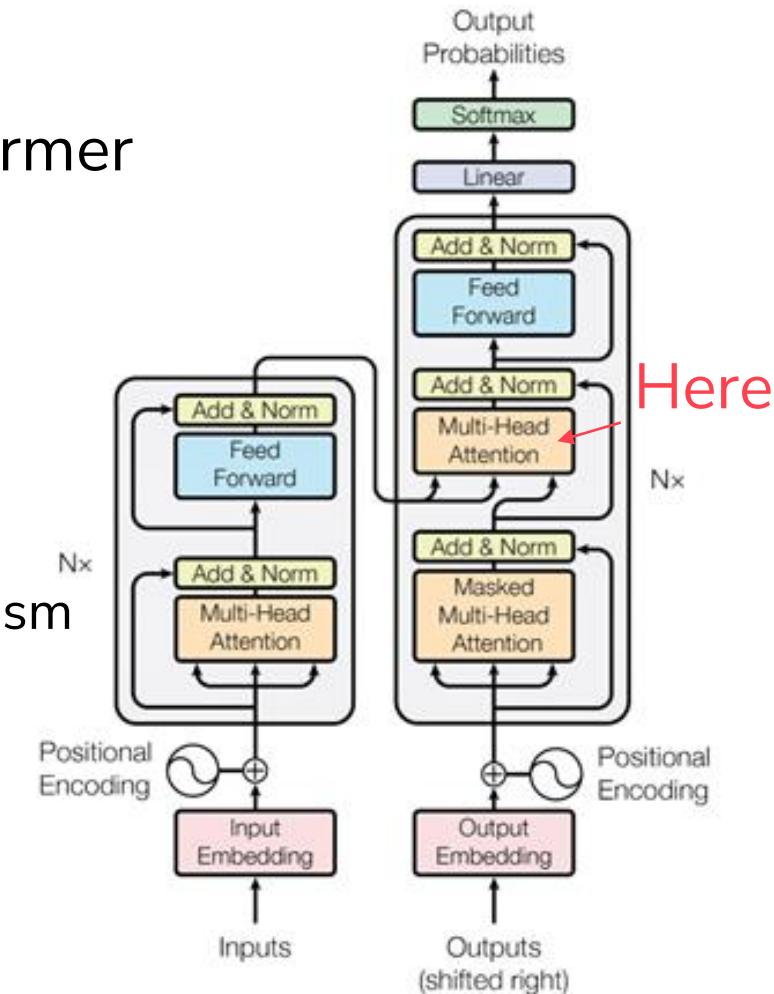
- Inside the encoder
- **Inside the decoder**
  - the queries, keys and values all come from the previous decoder output
  - use of a « mask » to decode only from known elements
- At the encoder-decoder interface



# Transformer: Multi-Head Attention

The **Multi-Head Attention** is used three times in the Transformer

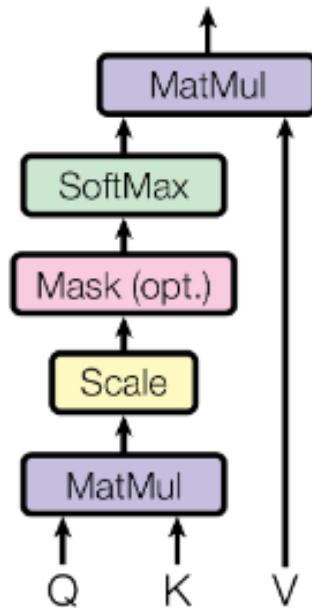
- Inside the encoder
- Inside the decoder
- **At the encoder-decoder interface**
  - the queries come from the previous decoder layer
  - the keys and values come from the encoder outputs
  - identical to the classical “encoder-decoder” Attention mechanism



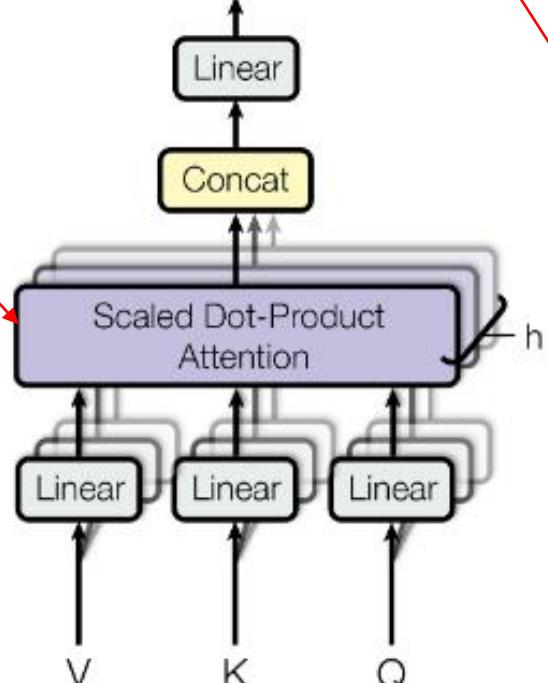
# Transformer: Attention

Transformer

Attention



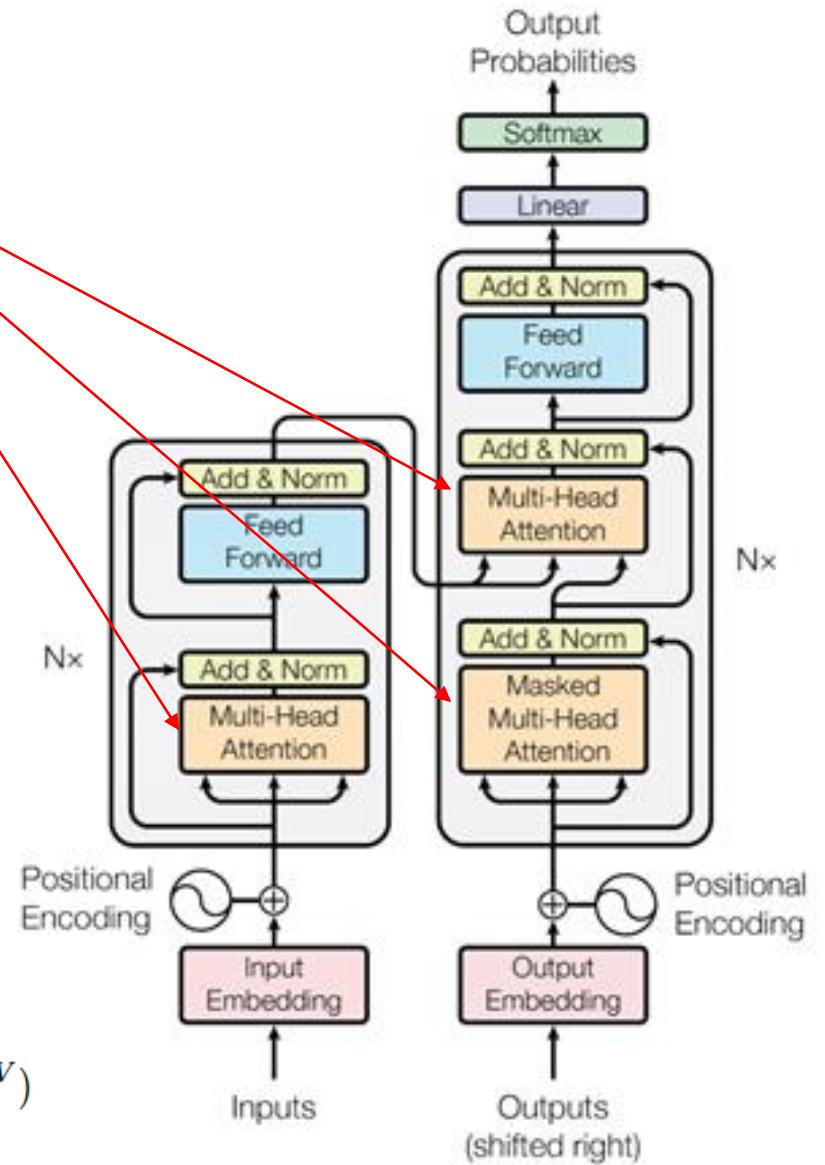
multi-head  
Attention



$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

$$\text{where } \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



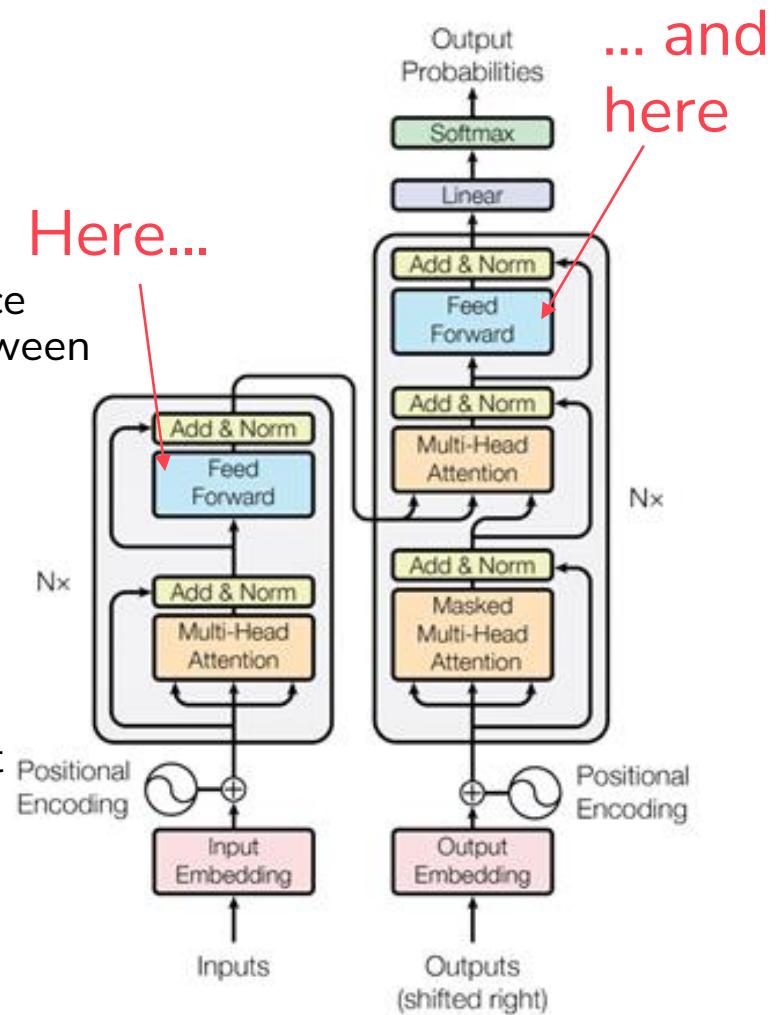
# Transformer: Feed-Forward Networks

## Principle

- **Feed-Forward network** in each **encoder** and each **decoder** layer
- Each of these Feed-Forward network is completely connected
- Applied in an independent and identical way to each element in the sequence
- Consist in two linear transformations, with the use of a ReLu fonction in between

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

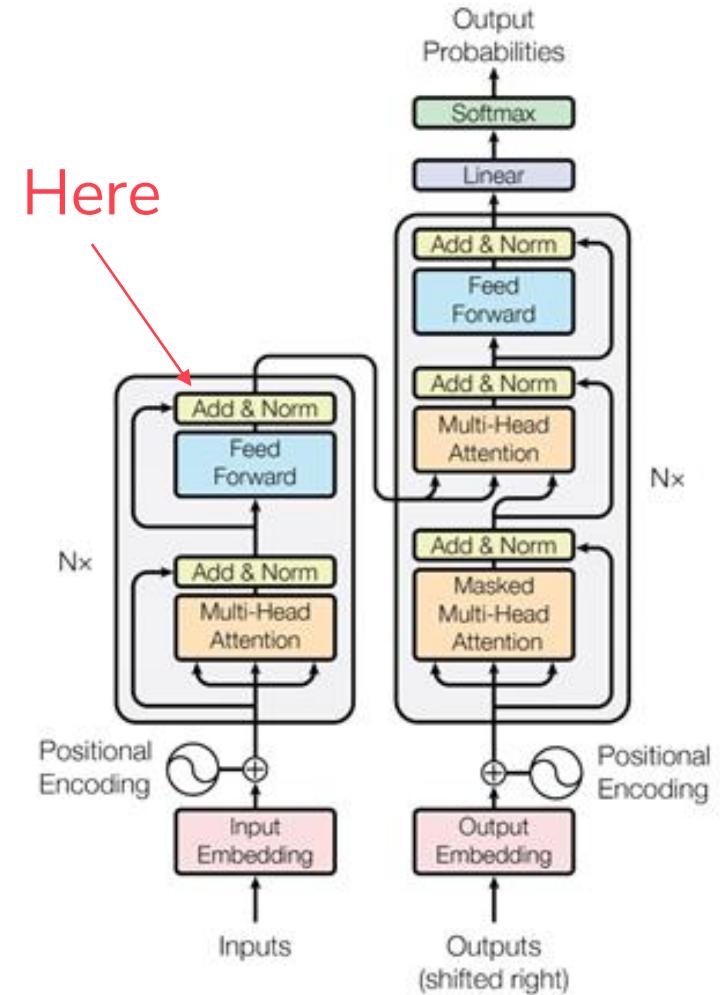
- The linear transformations are identical for each element in the sequence...
- ... but are different for each layer
- Its role is to process the output from one attention layer to better fit the next



# Transformer: The residual connection

## Principle

- **Add:** To strengthens the Transformer memory.
- **Normalisation:** To reduce the needed steps to optimize the network.



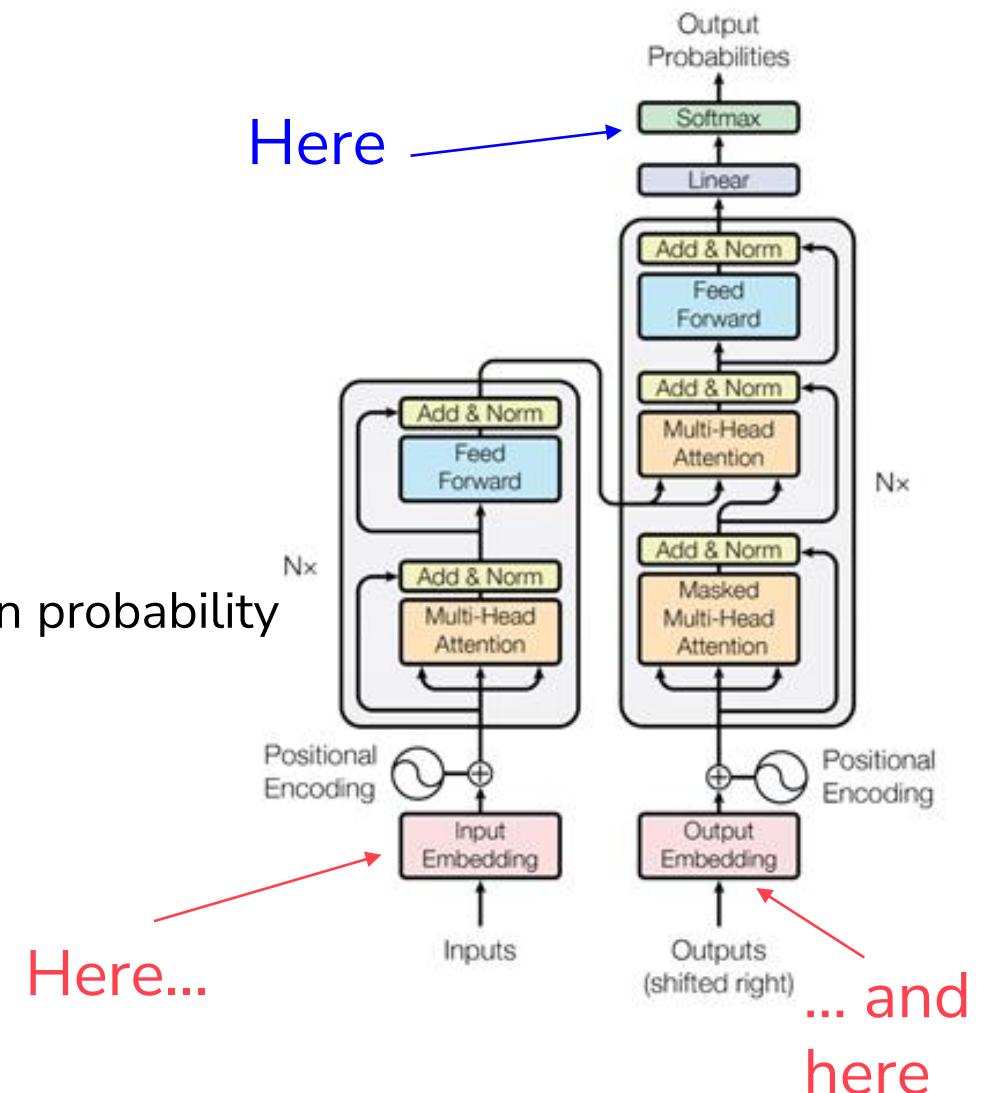
# Transformer: Embeddings et Softmax

Embedding learning in order to convert

- The input *tokens*
- The output *tokens*

Use of a **Softmax** layer

- Convert the *decoder* output into the next token prediction probability



# Transformer: Positional Encoding

## Principle

- Keep in memory the *token positions* in the sequence
- Add of “*positional encodings*” to the input embeddings
- Used just before the encoder-decoder stacks

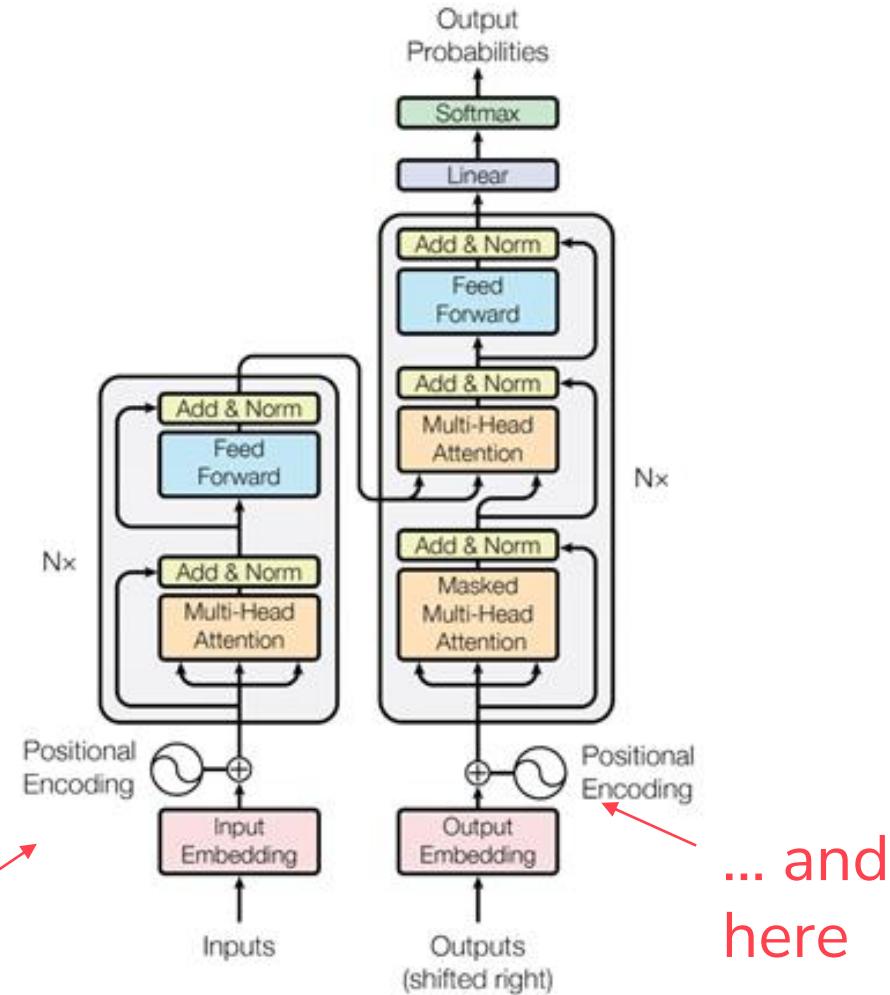
## Formulas

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

- with  $pos$  the position and  $i$  the dimension
- Each dimension corresponds to a sinusoid
- For each  $k$ ,  $PE_{pos+k}$  is a linear function of  $PE_{pos}$

Here...



# Transformer: Results on the Benchmark BLEU

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet	23.75			
Deep-Att + PosUnk		39.2		$1.0 \cdot 10^{20}$
GNMT + RL	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble	26.36	<b>41.29</b>	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1		$3.3 \cdot 10^{18}$
Transformer (big)	<b>28.4</b>	<b>41.8</b>		$2.3 \cdot 10^{19}$

# Transformer: Tutorial notebook

*course4\_transformer\_tutorial.ipynb*

**Goal:** Illustration of how a Transformer can be build step by step in Python

## Remarks:

- This notebook comes from a **tutorial notebook freely accessible** (the original reference is given at the very beginning, in the first cell)

# BERT model

# BERT: Introduction

BERT (2018) is for *Bidirectional Encoder Representations from Transformers*

- A Transformer variant with a **Bidirectional** specificity
- Thanks to it, there is no need for the end sequence token masking constraint anymore (into the decoders Attention mechanism)
- Build an efficient and generic language representation

The two main BERT steps

- A **pre-training** step on non-labelled data
- A **specialization** step (“**fine tuning**”) for a specialized task, from pre-trained parameters
  - There is only need to train the « **last** » **layers** of the model to get such a specialization

**French versions** of BERT exists, the most famous one being probably **CamemBERT**

# BERT: Architecture

The initial **BERT** architecture (2018) is the following:

- Length of the processed sequences: **512** (Limitation to contain a quadratic explosion)
- Stack of **Nx = 12 blocks** of encoders and decoders (a 24-blocks variant exists)
- **h = 12** attention heads
- It means there are  $h \times Nx = 12 \times 12 = 144$  distinct attention mechanisms
- Each head gives an output vector of size: **64**
- A hidden layer of  $h \times 64 = 768$  dimensions
- Roughly **110 millions** of parameters for the basic version (12 encoders) and **340 millions** for the large version (24 encoders)

Reminder, the initial Transformer architecture (2017)

- Length of the processed sequences: **512** as well
- Stack of **Nx = 6 blocks** of encoders and decoders
- **h = 8** attention heads

# BERT: Pre-training

**BERT is pre-trained from two unsupervised tasks:**

Masked Language Modeling (MLM)

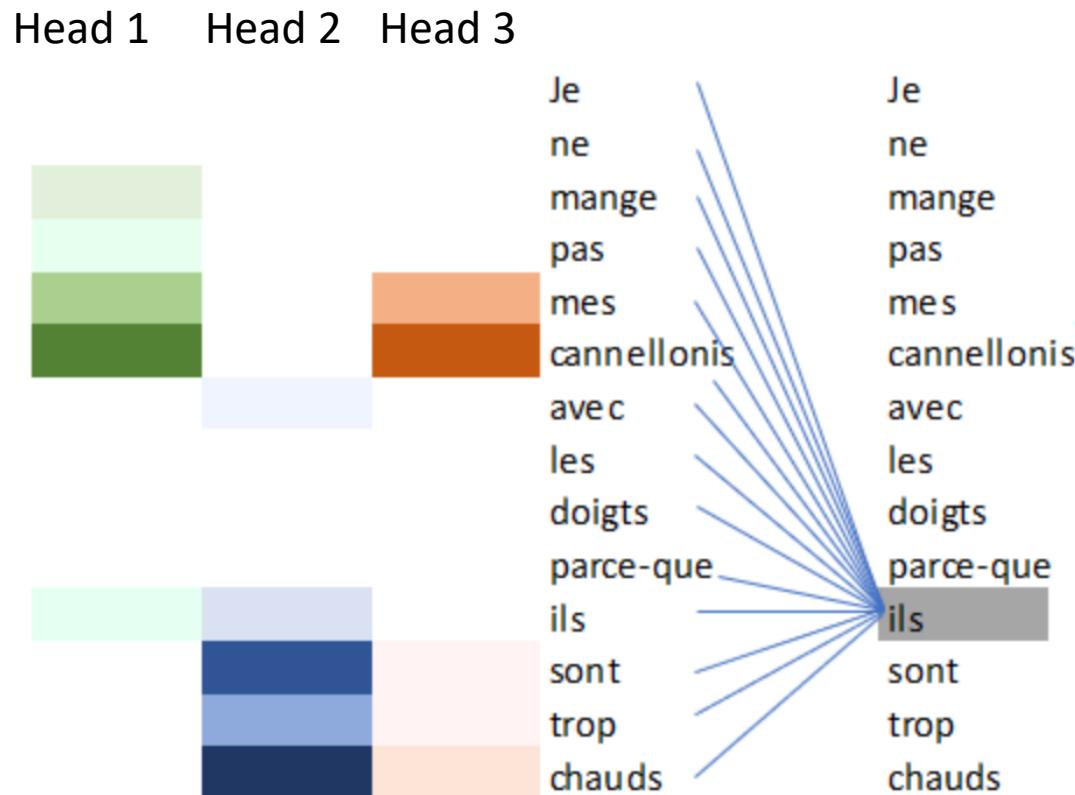
- The model is pre-trained from sequences decomposed into *tokens*
- The goal is to predict some of these tokens (15%) randomly masked in the text

Next Sentence Prediction (NSP)

- The goal is to understand the relationship between two successive sentences
- To do that, the algorithm is trained in order to predict the sentence following a given sentence
- The training set is composed of sentence tuples (A,B)
  - In 50% of the cases, B is the **sentence following** the sentence A (the tuple is then labelled *IsNext*)
  - In 50% of the cases, B is a sentence **randomly** chosen in the corpus (*NotNext* label)

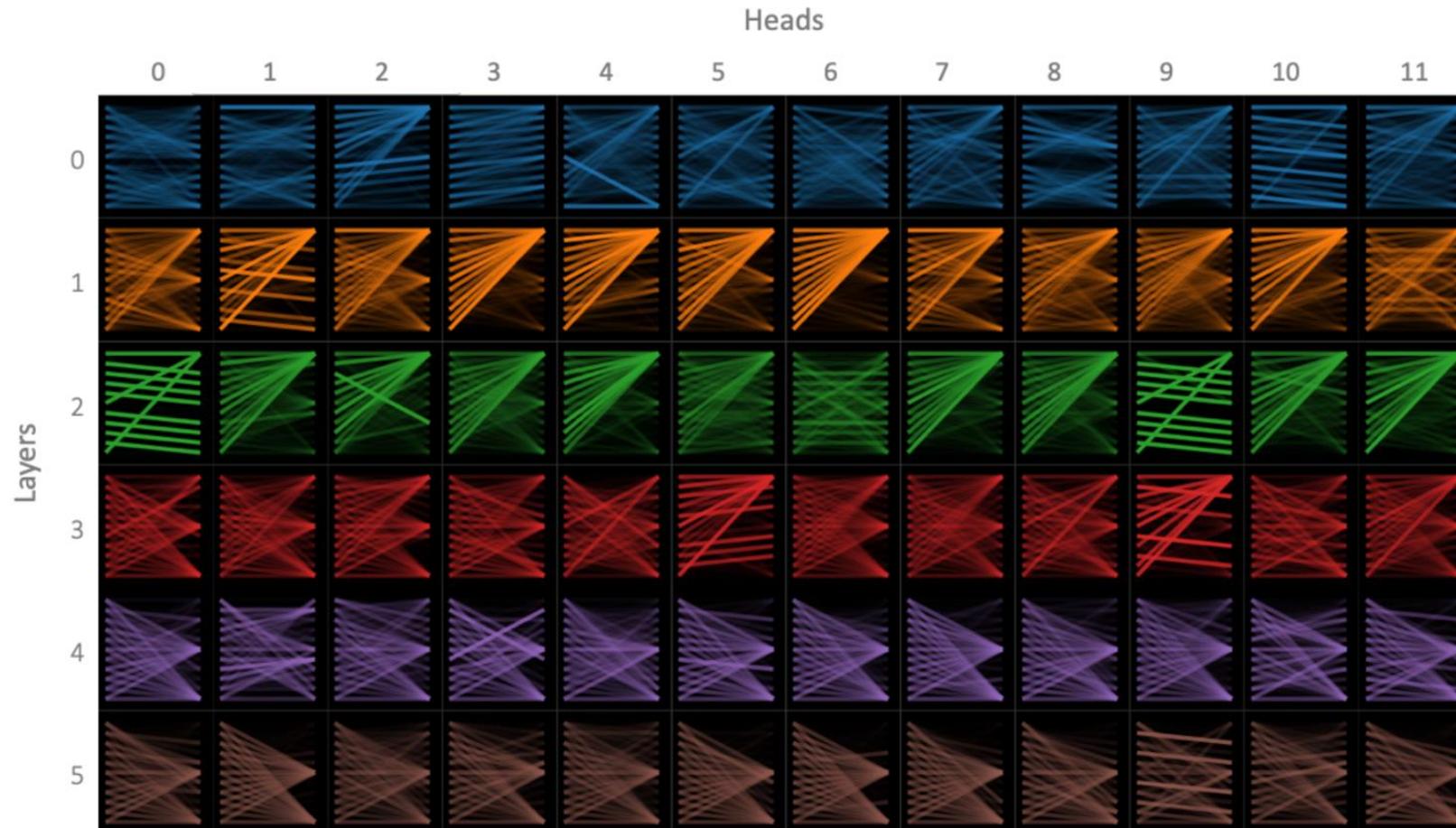
# CamemBERT and Attention mechanism

Attention level, for each head, relatively to the french word “ils” (« they »). Each head “see” **different** things from others (**complementarity**)



# CamemBERT and Attention mechanism

On the first 6 layers (out of 12), attention mechanism visualization example over the 12 heads



# BERT: Fine-tuning

The initial pre-training step needs:

- A **huge** corpus of **generic** texts
- Very **high** computational resources

The fine-tuning step relies on the already pre-trained algorithm, so it only needs :

- A **reduce** and **specialized** corpus
- Significantly **reduces** resources

The aim is obviously to move from a **generic language** modelization to a **task specific one**

# Hugging Face



- Hugging Face is an open-source provider of NLP technologies for Python
- A Hugging Face **Transformer** package is available and quite popular
- It gives access to pretrained models such as BERT but also to modules to perform fine-tuning
- It previously supported only **Pytorch**, but now **Tensorflow** is more and more supported as well

# BERT implementation: Tutorial notebook

*course4\_transformer\_model\_libraries.ipynb*

**Goal:** Illustration of how pre-trained Transformer can be loaded and used from open-source libraries such as Hugging Face

## Remarks:

- This notebook comes from a **tutorial notebook freely accessible** (the original reference is given at the very beginning, in the first cell)

# Current state-of-the-art

# GPT



- **OpenAI** is an AI research laboratory founded in 2015 by Elon Musk, Sam Altman and others
- The **GPT** model (2018) was designed by OpenAI researchers (Altman et al.)
- Its architecture is composed of a stack of 12 transformers modules (using only Masked Multi-Head Attention layers) and was train on a large dataset
- It is fine-tuned on tasks such as text classification, similarity and question answering...

# GPT-2



- **GPT-2** (2019) was developed few months later, with a similar architecture but larger (1.5 billion parameters!)
- It can be good on many tasks even without fine-tuning
- A **smaller version** of the GPT-2 model is available (“only” 117 million parameters)
- We will see a GPT-2 fine-tuning application later in this **Course**

# GPT-3



- GPT-3 is a language model developed by the Californian start-up OpenAI, it succeeds GPT and GPT-2 in **July 2020**
- GPT-3 has much more parameters than GPT-2 (175 billions VS 1.5 billion!)
- GPT-3 has been trained on a larger dataset than GPT-2 (50 times bigger)
- Unlike BERT, GPT-3 does not need to be fine-tuned for a specific task, it achieves high performance from the pre-trained model only (**zero-shot learning**)

# GPT-3



## Applications

- Text generation
- Chatbot
- Sum-up information
- Code generation
- Translation...

All classical NLP tasks can be virtually addressed by GPT-3

# GPT-3



## Limits

- Cannot be trained on a personal computer...
- ... and would need close to **5 million dollars** for training only
- The results can be “**unsafe**” (different kind of bias)
- Microsoft announced in September 2020 that it had **licensed exclusive** use of GPT-3
- **Not open-source**, can be used with an API only (after application)
- Some organizations are working on their own version of GPT-style architecture in order to offer an open-source alternative

# Volume comparison

Model	Corpus (Giga words)	Parameters (Giga)	Cost (Tera flops)
BERTLARGE	250	0.36	$5.33 \times 10^8$
CamemBERT	138	0.11	
RoBERTa Large	2 000	0.36	$4.26 \times 10^9$
GPT-3	300	174.00	$3.14 \times 10^{11}$

Reference: LE MAGAZINE DE L'INTELLIGENCE ARTIFICIELLE

# DALL-E (January 2021)



DALL-E is an AI (using GPT-3) to generate images from text inputs

## Text prompt

an illustration of a baby radish in a tutu walking a dog



an armchair in the shape of an avocado

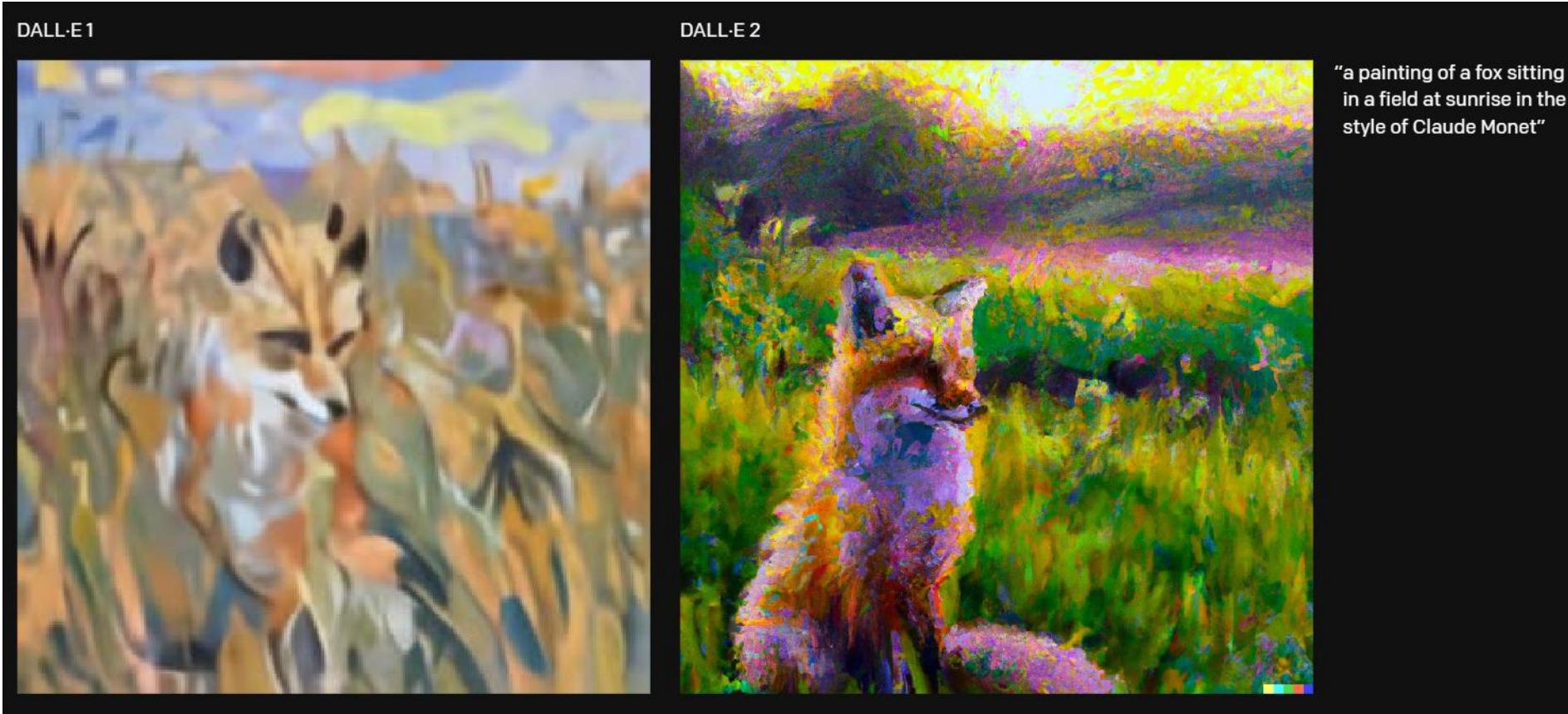


OpenAI has refused to allow access to code source or API → possibility to get on a waitlist

# DALL-E 2 (April 2022)



A new DALL-E version, more powerful has been released recently



Do not hesitate to have a look on DALL-E 2 impressive features → <https://openai.com/dall-e-2/>

# DALL-E mini



A simpler DALL-E model (**DALL-E mini**) is available on Hugging Face

*A dog and cat sleeping together*



It is not as powerful as the classical model but still quite interesting to test  
<https://huggingface.co/spaces/dalle-mini/dalle-mini>

# Use case: Text generation

<https://transformer.huggingface.co/>



## Write With Transformer

Get a modern neural network to  
auto-complete your thoughts.

This web app, built by the Hugging Face team, is the official demo of the  
`🤗/transformers` repository's text generation capabilities.

### Checkpoints



DistilGPT-2

The student of the now ubiquitous GPT-2 does not come short of its teacher's expectations. Obtained by distillation, DistilGPT-2 weighs 37% less, and is twice as fast as its OpenAI counterpart, while keeping the same generative power. Runs smoothly on an iPhone 7. The dawn of lightweight generative transformers?

[Start writing](#) [More info](#)



Arxiv-NLP

Built on the OpenAI GPT-2 model, the Hugging Face team has fine-tuned the small version on a tiny dataset (60MB of text) of Arxiv papers. The targeted subject is Natural Language Processing, resulting in a very Linguistics/Deep Learning oriented generation.

[Start writing](#) [More info](#)

### Models



GPT-2

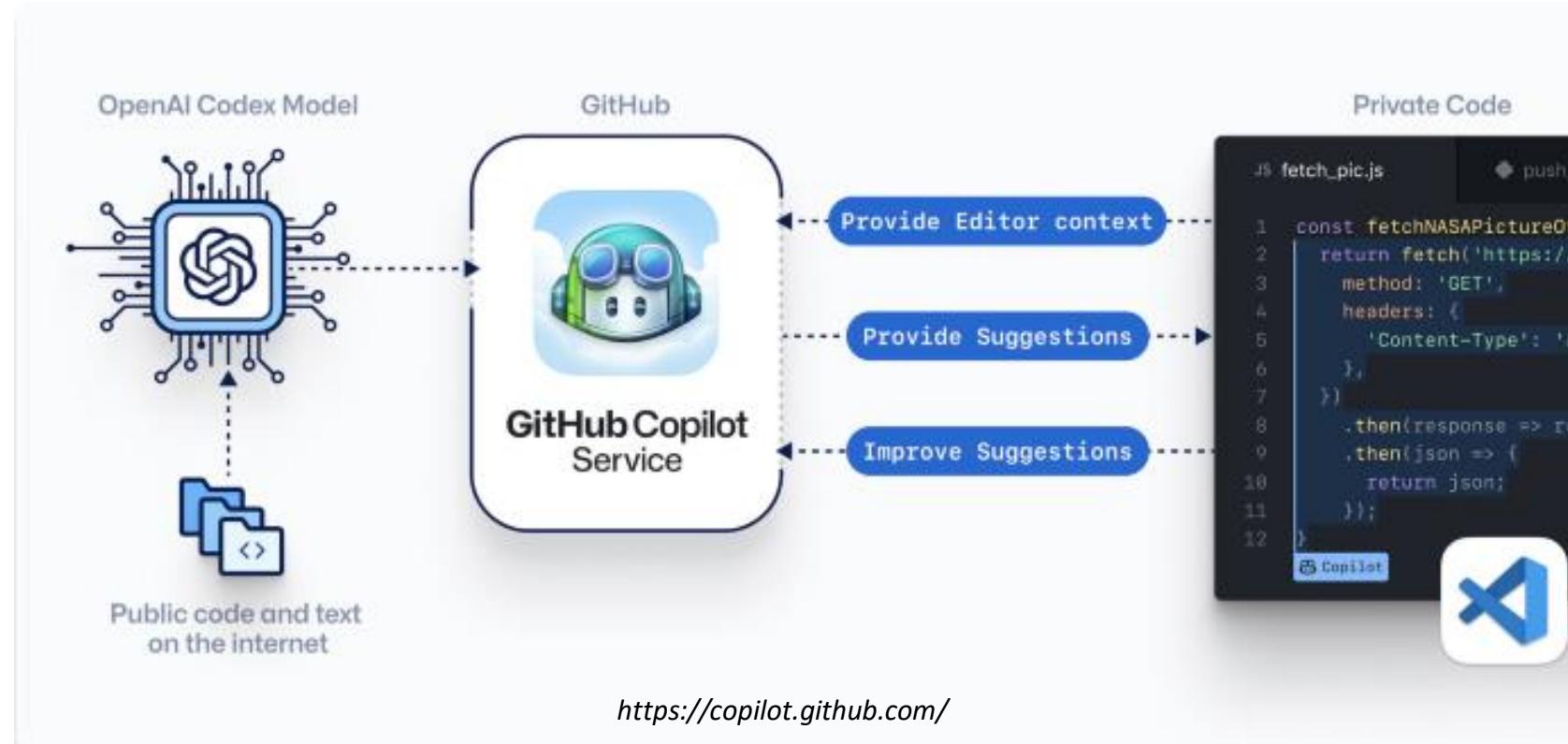
The almighty king of text generation, GPT-2 comes in four available sizes, only three of which have been publicly made available. Feared for its fake news generation capabilities, it currently stands as the most syntactically coherent model. A direct successor to the original GPT, it reinforces the already established pre-training/fine-tuning killer duo. From the paper: Language Models are Unsupervised Multitask Learners by Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei and Ilya Sutskever.

[Start writing](#)

# GitHub Copilot



- An AI pair programmer that provides autocomplete-style suggestion for various languages (including Java and Python)
- It has been trained on billions lines of code from GitHub



# GitHub Copilot

A screenshot of a code editor interface. On the left, there is a file named "main.py" containing the following Python code:

```
1
2 def calculateDaysBetweenDates(begin, end):
3
4
5
6
7
```

The word "calculateDaysBetweenDates" is underlined with a red squiggly line, indicating it's a suggestion from GitHub Copilot. On the right, a floating window titled "GitHub Copilot" shows the suggested completion:

```
Accept solution
def calculateDaysBetweenDates(begin, end):
    # begin and end are datetime objects
    # returns the number of days between begin and end
    return (end - begin).days
```

A button labeled "Accept solution" is visible at the top of this window.

# GitHub Copilot



The screenshot shows a code editor interface with two panes. On the left, the file `main.py` is open, containing the following code:

```
10
11
12     # generate the highest common denominator of two numbers
13
14 def gcd(a, b):
```

On the right, the GitHub Copilot pane displays three suggested implementations of the `gcd` function, each with an "Accept solution" button:

- Solution 1:

```
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a
```
- Solution 2:

```
def gcd(a, b):
    if a == 0:
        return b
    return gcd(b % a, a)
```
- Solution 3:

```
def gcd(a, b):
    if b == 0:
        return a
    return gcd(b, a % b)
```

# GitHub Copilot



The screenshot shows a code editor interface with two panes. On the left, the file `main.py` is open, containing the following code:

```
51
52
53     # check if a number is a prime number
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
```

A cursor is positioned at line 54, where the text `# check if a number is a prime number` is highlighted with a yellow underline. A small orange lightbulb icon is placed next to the start of this line. The right pane is titled "GitHub Copilot" and shows two identical code snippets:

```
# check if a number is a prime number
def isPrime(n):
    if n == 1:
        return False
    for x in range(2, n):
        if n % x == 0:
            return False
    return True
```

Below each snippet is a button labeled "Accept solution".

# GitHub Copilot



A screenshot of a code editor interface showing two panes. The left pane is titled 'main.py' and contains the following Python code:

```
38
39 # Get average runtime of successful runs in seconds
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
```

The right pane is titled 'GitHub Copilot' and shows two identical code snippets for calculating the average runtime of successful runs. Both snippets include an 'Accept solution' button above them. The first snippet uses a total runtime variable and a for loop with an if condition. The second snippet uses a total variable and a for loop with an if condition.

```
# Get average runtime of successful runs in seconds
def getAverageRuntime(runs):
    # runs is a list of Run objects
    # returns the average runtime of successful runs in seconds
    totalRuntime = 0
    for run in runs:
        if run.success:
            totalRuntime += run.runtime
    return totalRuntime / len(runs)

# Get average runtime of successful runs in seconds
def getAverageRuntime(runs):
    # runs is a list of Run objects
    # returns the average runtime of successful runs in seconds
    total = 0
    for run in runs:
        if run.success:
            total += run.runtime
    return total / len(runs)
```

# Fine-tuning use case on GPT-2

# GPT-2: Description

- OpenAI released in February 2019 a paper describing GPT-2
- GPT-2 is an **AI text generation model** based on the **Transformer architecture**
- As we saw previously in the Course, GPT-3 succeeded GPT-2 in July 2020
- However, GPT-2 being a less complex model it can be more accessible to use for testing purposes

## GPT-2: Fine-tuning

- GPT-2 is used mainly for automatic text generation
- It can be fine-tuned to generate specific kind of text
- For fine-tuning, you need to use the pre-trained GPT-2 version and a corpus containing the texts you want as references
- The goal is to move the weights of the model in order to specialize it

# GPT-2: Fine-tuning application

- There exists a notebook **freely accessible** that gives the possibility of **very easy** GPT-2 **fine tuning** (created by **Max Woolf**, Data Scientist at BuzzFeed)
- By default, the fine-tuning is done to perform **Shakespeare style text** generation => just like in the **Course 3** graded exercise
- However, a fine-tuned GPT-2 should give **way better results** than what you would achieve with a **model trained from scratch**
- We can try to check this point

# GPT-2: Fine-tuning application

- There exists a notebook **freely accessible** that gives the possibility of **very easy** GPT-2 **fine tuning** (created by **Max Woolf**, Data Scientist at BuzzFeed)
- By default, the fine-tuning is done to perform **Shakespeare style text** generation => just like in the **Course 3** graded exercise
- However, a fine-tuned GPT-2 should give **way better results** than what you would achieve with a **model trained from scratch**
- We can try to check this point => **but may we find something funnier than Shakespeare this time?** ☺

# GPT-2: Fine-tuning application

- With a relevant corpus, you can generate any kind of text from a fine-tuned GPT-2
- You can even generate fictional stories
- For example, a “new” episode of your favorite TV show or the next book of your favorite saga
- **How can we do that?**

# GPT-2: Fine-tuning application

- Being personally a **comic book reader**, I wanted to try it to generate the plot of a new *Amazing Spider-Man* comic book issue
- Since 1963, there were more than **800 issues**



Amazing Spider-Man co...  
ew.com



Amazing Spider-Man co...  
ew.com



Amazing Spider-Man co...  
ew.com



Amazing Spider-Man co...  
ew.com



Amazing Spider-Man co...  
ew.com



Amazing Spider-Man co...  
ew.com



Amazing Spider-Man co...  
ew.com



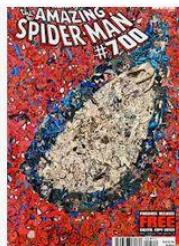
Submarine Channel | Top 5: Be...  
submarinechannel.com



Amazing Spider-Man co...  
ew.com



The Amazing Spider-Ma...  
sanctuary.fr



Submarine Channel | To...  
submarinechannel.com



Amazing Spider-Man co...  
ew.com



Submarine Channel | Top ...  
submarinechannel.com



Marvel superhero comic...  
pinterest.com



Amazing Spider-Man co...  
ew.com



Amazing Spider-Man #55 P...  
legacycomics.com



The Amazing Spider-Ma...  
marvel.com



Submarine Channel | Top ...  
submarinechannel.com

# GPT-2: Fine-tuning application

- The detailed synopsis for every issue are available in some internet sites
- For example, in *marvel.fandom.com*
- I developed a code to **build a corpus** by **scrapping** automatically all the synopsis
- I used this corpus to **fine-tune GPT-2** and generate a **whole new issue**

The screenshot shows a web page from the Marvel Database. At the top, there's a red header bar with the text 'MARVEL' and 'Marvel Database [database]'. Below the header, the URL 'marvel.fandom.com' is visible. The main content area is titled 'Amazing Spider-Man Vol 1 100'. It features a comic book cover for 'The Amazing Spider-Man' #100, which is part of the 'Six-Armed Saga' story arc. The cover art shows Spider-Man in his iconic suit. Below the cover, there's a quote from Peter Parker: 'I drank the potion— even though it was *united*— it was *united*— because I wanted it to change me but— not like this! *NOT LIKE THIS!*' followed by a sad face emoji. The page lists various characters associated with the issue, including 'Featured Characters' (Spider-Man), 'Supporting Characters' (Aunt May, Gwen Stacy, George Stacy, Daily Bugle Staff, J. Jonah Jameson, Ned Leeds, Betty Brant), 'Antagonists' (Vulture, Lizard, Green Goblin, Doctor Octopus, Kingpin), 'Other Characters' (NYPD), 'Locations' (New York City), 'Items' (Spider-Man's Web-Shooters, Vulture's Wings, Doctor Octopus' Tentacles), and 'Vehicles' (Goblin Glider). On the right side of the page, there's a sidebar for 'DISCORD' showing '631 Members Online' and a list of recent activity. At the bottom, a red-bordered box highlights the 'Synopsis for "The Spider or the Man?"' section, which describes the plot involving Spider-Man's struggle with his alter ego Peter Parker.

# GPT-2: Fine-tuning application

- I fine-tuned GPT-2 with my corpus.
- I started a text with the sentence  
*Spider-Man discovered the Green Goblin secret identity...*  
and **I generated the rest**
- **Just for fun**, I looked for illustration pictures for every sentence generated (just to make it looks like a real comic book) **and** I also generated fake illustrations from **DALL-E mini**

# GPT-2: Fine-tuning application



*Spider-Man discovered the Green Goblin secret identity...*

# GPT-2: Fine-tuning application

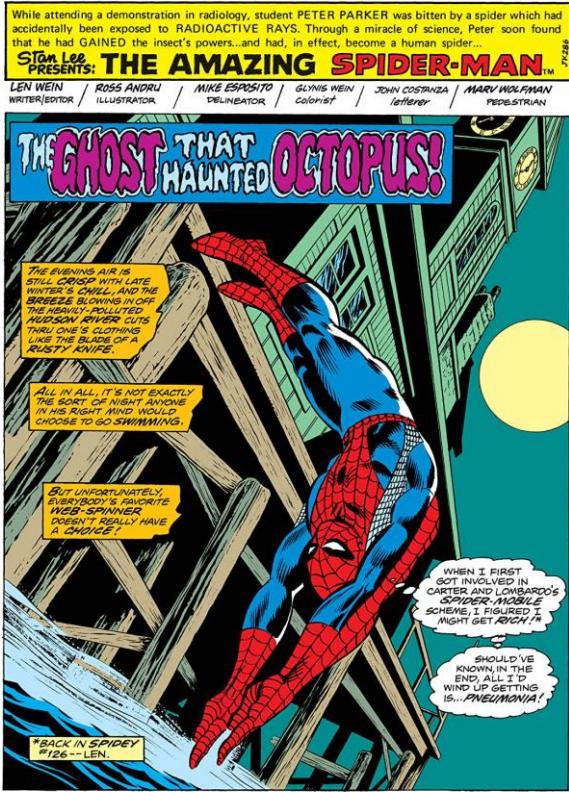
Picture generated with DALL-E mini



*Spider-Man discovered the Green Goblin secret identity...*

Real illustration taken from the original comic book

# GPT-2: Fine-tuning application



© 2015 Marvel Characters, Inc. All rights reserved. All characters featured in this issue and the distinctive names and likenesses thereof, and all related indicia are trademarks of Marvel Characters, Inc. No similarity between any of the names, characters, persons, and/or institutions in this magazine with those of any living or dead person or institution is intended, and any such similarity which may exist is purely coincidental. WWW.MARVEL.COM

...and saved the boy from drowning.

# GPT-2: Fine-tuning application

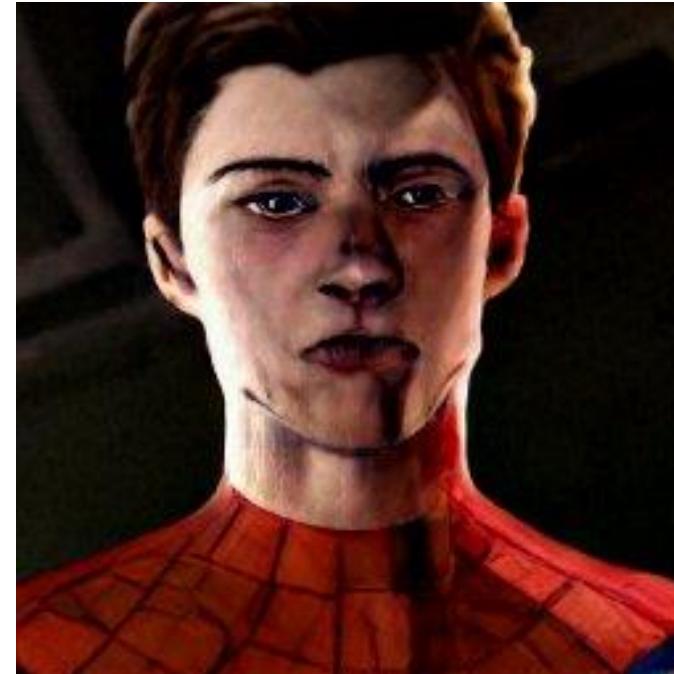


He then helped Spider-Man locate the Goblin's **hidden hideout** and escape into the Hudson River.

# GPT-2: Fine-tuning application



134



Spider-Man then resumed his **civilian identity**...

# GPT-2: Fine-tuning application



...and returned to the city where **he** and his wife **Mary Jane** were reunited.

# GPT-2: Fine-tuning application



Mary Jane is indeed Spider-Man's wife

...and returned to the city where he and his wife **Mary Jane** were reunited.

# GPT-2: Fine-tuning application



He soon found himself in a **life-altering battle with the Green Goblin...**

# GPT-2: Fine-tuning application



...with the Goblin having once more claimed he was responsible for the **death of Harry Osborn.**

# GPT-2: Fine-tuning application



...with the Goblin having once more claimed he was responsible for the  
**death of Harry Osborn.**

→ Harry is actually the Goblin's dead son, and his death is indeed an important point in the classic Spider-Man mythology

# GPT-2: Fine-tuning application



Spider-Man is then contacted by the people who were once his friends,  
the **Zabat-zombies**...

# GPT-2: Fine-tuning application



Spider-Man is then contacted by the people who were once his friends,  
the **Zabat-zombies**... ← ?!

# GPT-2: Fine-tuning application



...and tells them that he is on a mission, to get **Harry** and **Liz** off the Island of New York at this very moment.

# GPT-2: Fine-tuning application



...and tells them that he is on a mission, to get **Harry and Liz** off the Island of New York at this very moment.

→ Harry and Liz are actually a couple

# GPT-2: Fine-tuning application



...and tells them that he is on a mission, to get **Harry** and **Liz** off the Island of New York at this very moment.

But wait... was he not declared dead just before!?

# GPT-2: Fine-tuning application



Spider-Man tells the zombie horde that if they kill them now, **the whole world will be wiped out.**

# GPT-2: Fine-tuning application



Spider-Man tells the zombie horde that if they kill them now, **the whole world will be wiped out.** END of the story

# GPT-2: Fine-tuning application

- The story stays **consistent** with the **premise**, especially concerning the **characters** involved
- The plot is **completely new...**
- ... but **does not really make much sense**
- Especially in the long run, some plot **inconsistencies** start to appear
- But this is promising enough for just few minutes of fine-tuning

# GPT-2 fine-tuning: Exercise

*course4\_GPT-2 Text-Generating Model\_fine\_tuning\_ex.ipynb*

**Goal:** Fine-tune GPT-2 for specific text generation

**Remarks:**

- This notebook is **highly based** on the Max Woolf notebook mentioned before
- You can use the **corpus** already **given** but do not hesitate to build your **own corpus**
- Try to change the settings of the different parameters to understand the impact of each of them

# Take-away from Course 4

- LSTM architectures are **historically important** but there are **not** the state-of-the-art **any more**
- The “**Attention**” mechanism allowed a **significant performance improvement** than using LSTM only
- It has been demonstrated in the seminal paper ***Attention is all you need (2017)***, thanks to the **Transformer architecture** introduced in it, that “Attention” **alone** is enough to reach the state-of-the-art (no LSTM needed and no need to process a sequence in a specific order)
- **BERT** is an example of such **pre-trained Transformer models**, and it can be **fined-tuned** for specific tasks
- **Hugging Face** offers quite convenient tools **for using and fine-tuning BERT**
- The best models available (like **GPT-3**) can achieve very impressive results on a lot of tasks, but **are not open source**
- However, some **fine-tuning can be done** and sometimes quite **easily** (see **GPT-2**) for interesting results

# Course Schedule

- **Course 1:** NLP introduction
- **Course 2:** Word embedding
- **Course 3:** Long Short Term Memory (LSTM) architecture
- **Course 4:** “Attention” mechanism and Transformer architectures
- **Course 5: Chatbot introduction**
  - Chatbot core concepts
  - Rasa, a chatbot framework



# Chatbot core concepts

# Chatbot core concepts: Scope of applications

- Customer interactions handling
    - Amazon, Booking, Orange
  - Online business
  - Human ressources
  - Bank
  - Services and travels
    - booking or recommendations (hostel, restaurant, train/plane ticket... (Google, Trip Advisor, Transavia, SNCF...))
    - Luggage tracking (Air France)
  - News, cooking, dating sites, games, teaching, weather...
- + 100 000 chatbots launched on Messenger since 2016**

# Chatbot core concepts: Levels

Usually, it is considered that there are several levels concerning the conversational ability of a chatbot

- **FAQ:** understands simple and direct questions, in order to give the user a reply. Generally based on predefined rules
- **Contextual chatbot:** considers the context and the changes which can happen during the discussion (rules + machine learning)
- **Personalized chatbot:** knows well the user context and is aware of the relevant information to address during the talk
- **Autonomous Chatbot:** almost like a human being

# Chatbot core concepts: Levels

Usually, it is considered that there are several levels concerning the conversational ability of a chatbot

- **FAQ:** understands simple and direct questions, in order to give the user a reply. Generally based on predefined rules
- **Contextual chatbot:** considers the context and the changes which can happen during the discussion (rules + machine learning)
- **Personalized chatbot:** knows well the user context and is aware of the relevant information to address during the talk
- **Autonomous Chatbot:** almost like a human being

# Chatbot core concepts: Description

The core concepts can be slightly different from a solution to another. Below, those generally found:

- **Domain:** defines the perimeter in which the chatbot acts
- **Connector:** allows to connect the chatbot to one or several communication channels (example: slack)
- **Intention:** understands what the user wants, and categorizes his query to determine the relevant action to fulfill it

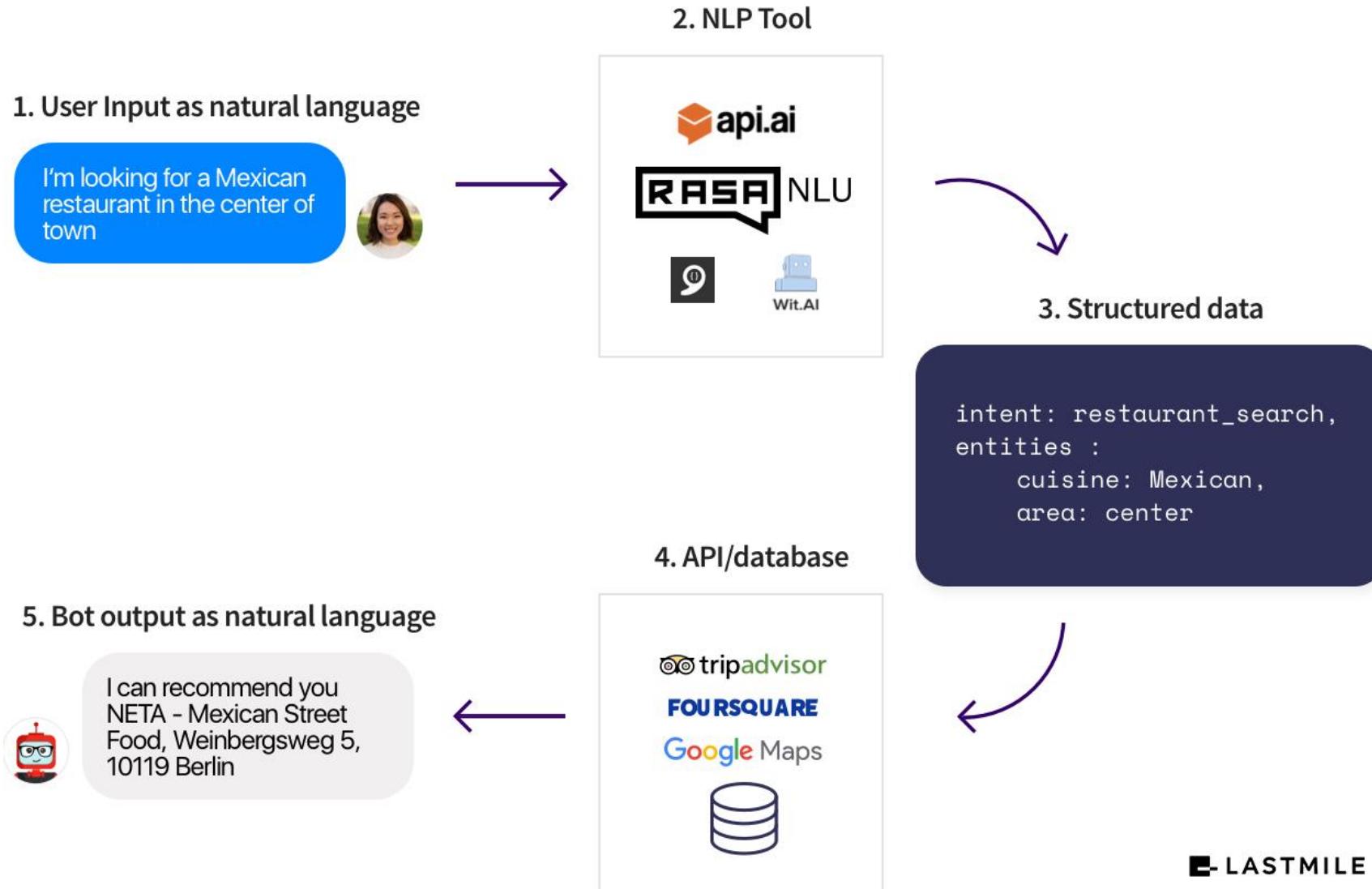
# Chatbot core concepts: Description

- **Entities:** corresponds to words used by the chatbot to understand the intention
- **Stories:** example conversations that train an assistant to respond correctly depending on what the user said previously. It is a central element to define the internal logic of a chatbot
- Other concepts can exist in some frameworks, such as action, rules, forms...

# Chatbot core concepts: Components

- **Dialogue management:** dialogue building tools and decision making => How the user interacts with the chatbot
- **Natural Language Understanding (NLU) engine** => How the chatbot understands what the user tries to say (intention).
- **Connectors** => How the chatbot interfaces itself to a third-party application

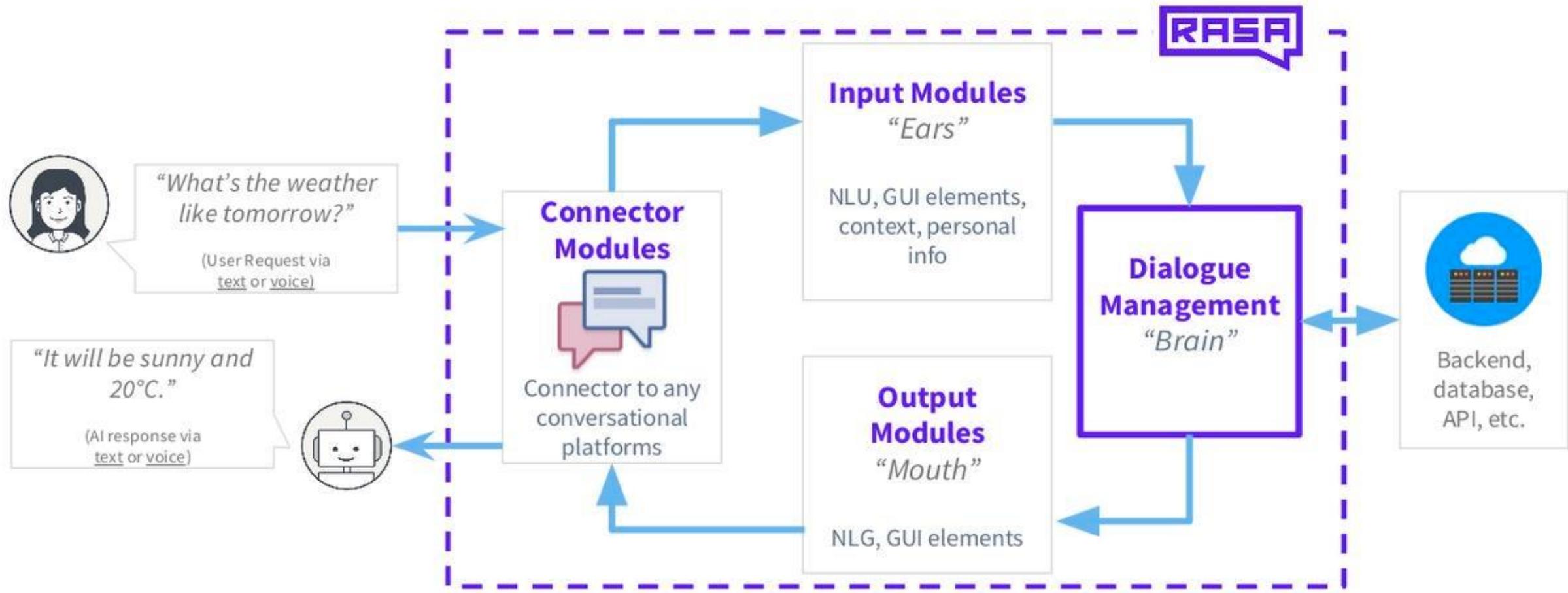
# Chatbot core concepts : Illustration



# Rasa, a chatbot framework

# RASA: Principle

Rasa the OSS to build conversational software with ML



# RASA: Dialogue example

I'm looking for a place to eat

I know lots of restaurants!  
Anything specific?

I love chatbots

Everybody does!

So, which cuisine?

A cheap one

Ok, and what city?

Actually, something fancy

Ok, expensive it is.

So which city?

Can you show me some restaurants yet?

I need some more info first.  
Where would you like to eat?

Why?

I need to know which city to search for restaurants.

**Chitchat:** Making small-talk or asking questions unrelated to the task

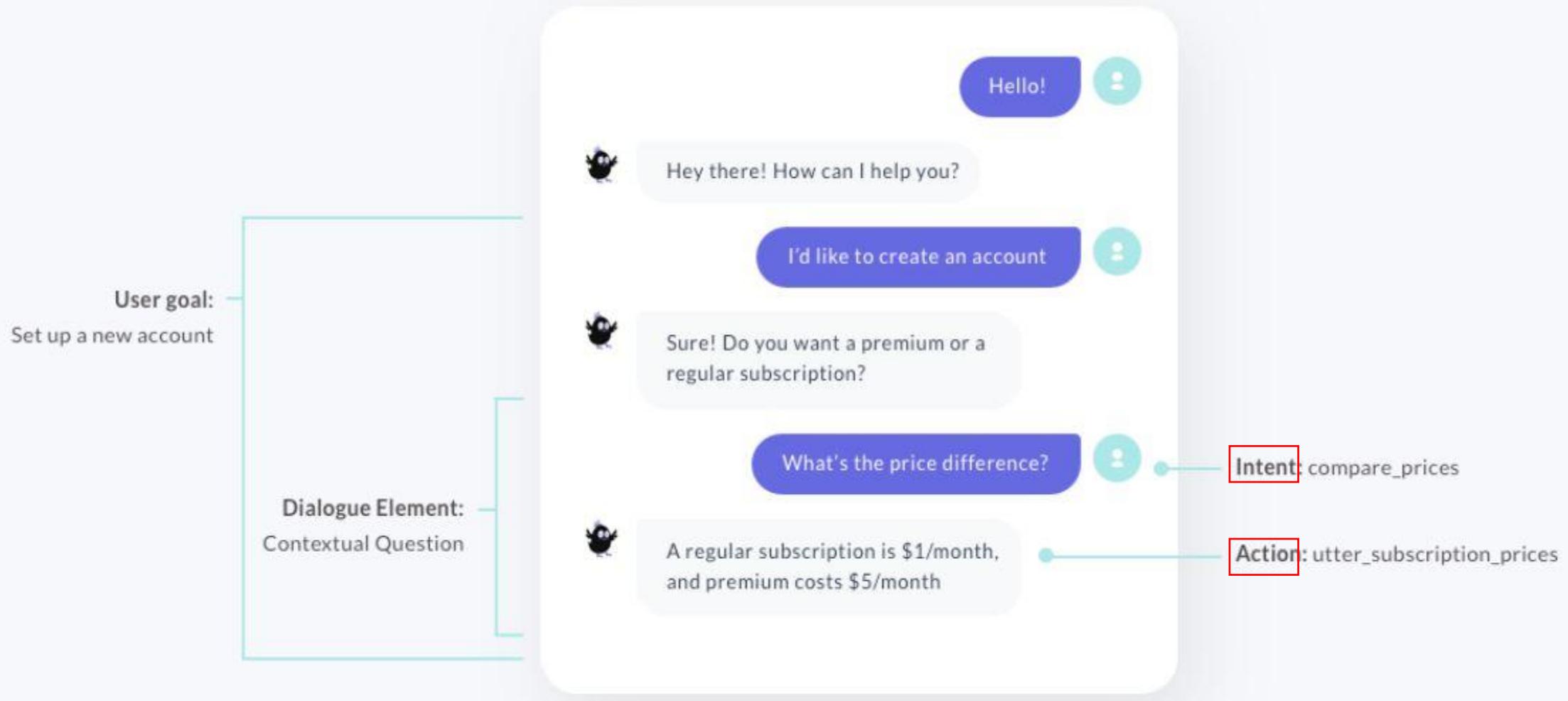
**Correction:** User has changed their mind about something

**Broad context:** Questions referring to the broader context of conversation, such as the status of the current task

**Narrow context:** Questions referring to immediate context, e.g. the most recent system utterance



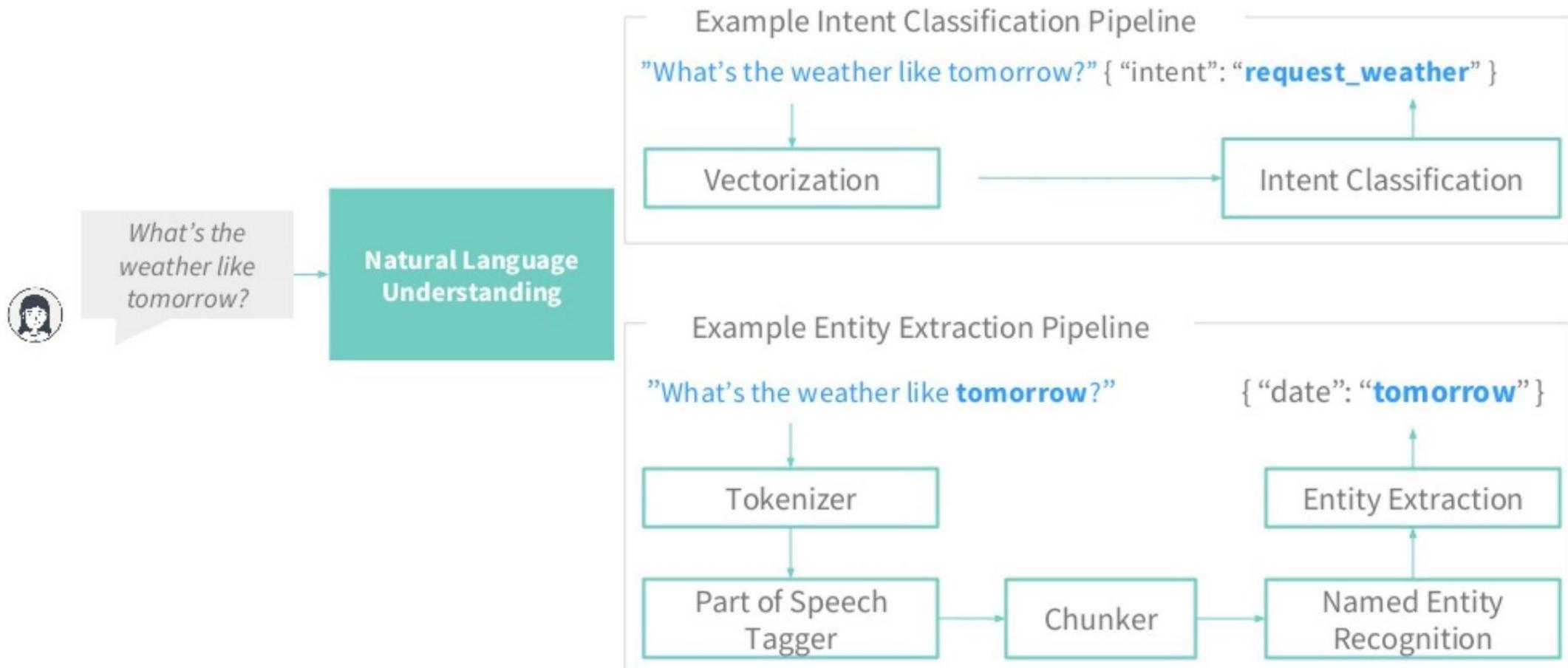
# RASA: Dialogue mechanisms



# Chatbot implementation with Rasa

# Implementation: NLU

## Rasa NLU: Natural Language Understanding



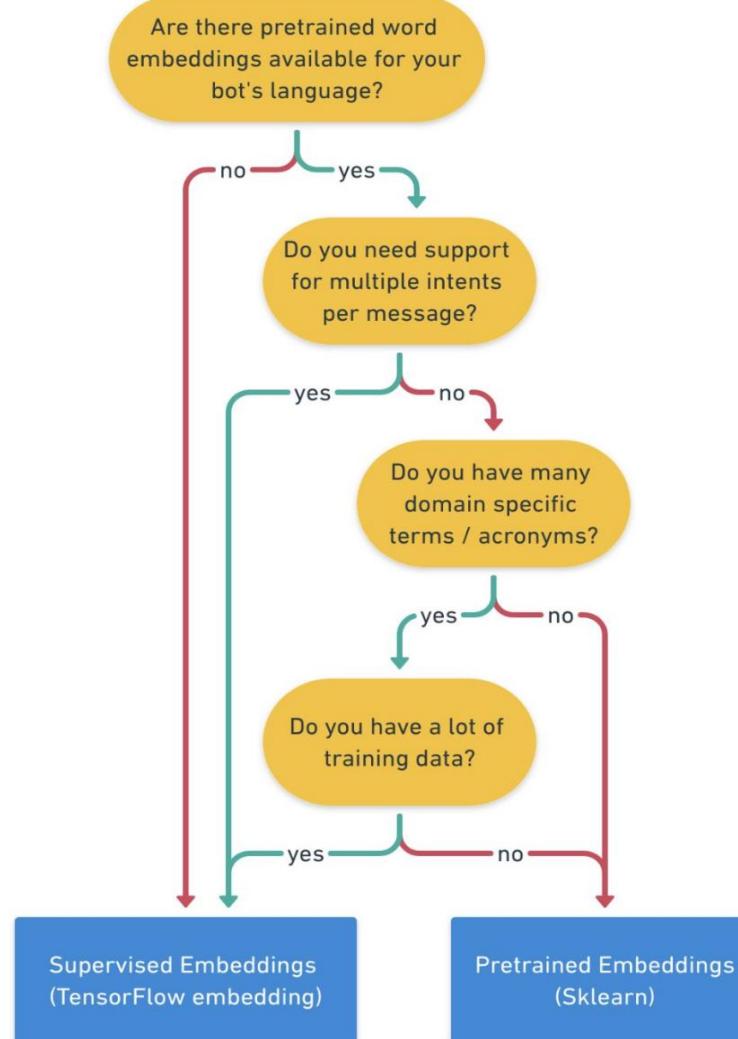
# Implementation: Intentions => learn how to listen

```
nlu:  
  - intent: greet  
    examples: |  
      - Hi  
      - Hey!  
      - Hallo  
      - Good day  
      - Good morning  
  
  - intent: subscribe  
    examples: |  
      - I want to get the newsletter  
      - Can you send me the newsletter?  
      - Can you sign me up for the newsletter?  
  
  - intent: inform  
    examples: |  
      - My email is example@example.com  
      - random@example.com  
      - Please send it to anything@example.com  
      - Email is something@example.com
```

# Implementation: Stories => learn how to speak

```
stories:  
  - story: greet and subscribe  
    steps:  
      - intent: greet  
      - action: utter_greet  
      - intent: subscribe  
      - action: newsletter_form  
      - active_loop: newsletter_form
```

# Implementation: Pre-configured pipeline



# Implementation: Pre-configured pipeline

Rasa comes with two pre-configured pipelines

- **Pre-trained embeddings**
  - use the spaCy library to load pre-trained models
- **Supervised embeddings**
  - train an embedding model from scratch

Both pipeline performs intent classification and entity extraction

# Implementation: Pre-configured pipeline

supervised\_embeddings

```
language: "en"

pipeline:
- name: "WhitespaceTokenizer"
- name: "RegexFeaturizer"
- name: "CRFEntityExtractor"
- name: "EntitySynonymMapper"
- name: "CountVectorsFeaturizer"
- name: "CountVectorsFeaturizer"
  analyzer: "char_wb"
  min_ngram: 1
  max_ngram: 4
- name: "EmbeddingIntentClassifier"
```

pretrained\_embeddings\_spacy

```
language: "en"

pipeline:
- name: "SpacyNLP"
- name: "SpacyTokenizer"
- name: "SpacyFeaturizer"
- name: "RegexFeaturizer"
- name: "CRFEntityExtractor"
- name: "EntitySynonymMapper"
- name: "SklearnIntentClassifier"
```

# Implementation: Pipeline components

- **Tokenizer**
  - WhitespaceTokenizer
  - SpacyTokenizer
- **Named Entity Recognition (NER)**
  - CRFEntityExtractor
  - SpacyEntityExtractor
- **Featurizers**
  - CountVectorsFeaturizer
  - SpacyFeaturizer
- **Intent Classifiers**
  - EmbeddingIntentClassifier
  - SklearnIntentClassifier



## Intent Classification

Intent Classification		
Pipeline	Featurizer	Intent Classifier
Supervised embeddings	CountVectorsFeaturizer	EmbeddingIntentClassifier
Pre-trained embeddings	SpacyFeaturizer	SklearnIntent Classifier

# Implementation: Building dialogue management

- **Domain**
  - Define the environment in which the chatbot operates
- **Custom actions**
  - Response actions with custom codes (Python...)
- **Slots**
  - Used to save some important information for the chatbot

# Implementation: Domain

- intents
- entities
- slots
- responses
- actions
- forms

```
1  session_config:
2    session_expiration_time: 60
3    carry_over_slots_to_new_session: true
4  intents:
5    - greet
6    - goodbye
7    - affirm
8    - slot
9      entities:
10     - drug
11     - sideeffect
12     - molecule
13   slots:
14     product:
15       type: text
16       influence_conversation: true
17     drug:
18       type: text
19       influence_conversation: true
20
21   responses:
22     utter_greet:
23       - text: Bonjour, comment puis-je vous aider ? Je suis capable de répondre à vos
24         questions sur les effets indésirables.
25     utter_cheer_up:
26       - image: https://i.imgur.com/nGF1K8f.jpg
27       - text: 'Here is something to cheer you up:'
28     utter_did_that_help:
29       - text: Est-ce que j'ai répondu à votre demande ?
30     utter_happy:
31       - text: Parfait, continuez !
32     utter_ask_product:
33       - text: '*** Veuillez choisir un produit ***'
34   actions:
35     - list_all_products
36     - list_all_product_sideeffects
37     - list_all_products_molecule
38     - list_all_products_molecule_suite
39   forms:
40     product_form:
41       product:
42         - type: from_text
43
```

# Implementation: Custom actions

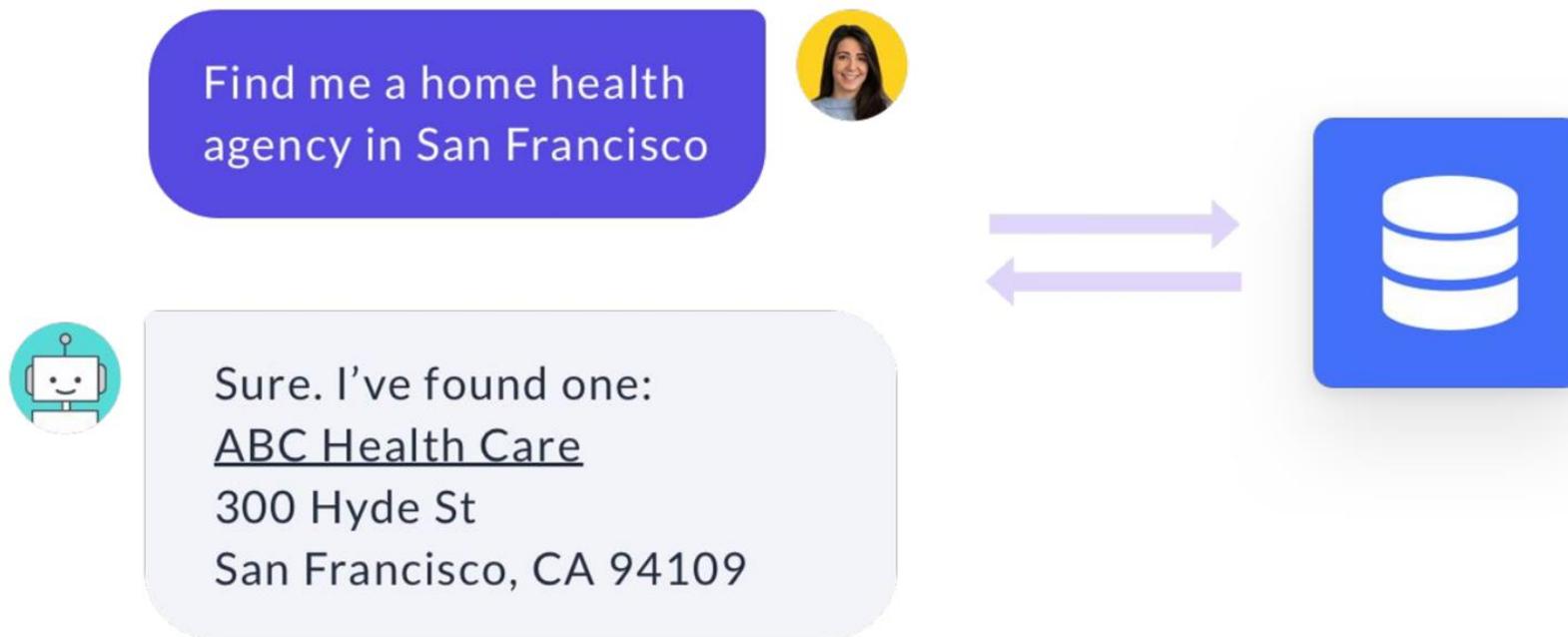
- It is possible to add response template directly into the domain file...

```
actions:  
  - list_all_products  
  - list_all_product_sideeffects  
  - list_all_products_molecule  
    - list_all_products_molecule_suite
```

- ... however, it allows only limited and predefined kind of replies
- There is an **alternative** to define much more complex kind of replies, in using custom codes
- These custom actions can **define anything** from **text response**, to **database connection** or **API call**...
- Python code can obviously be used for specific applications

# Implementation: Custom actions

You can interact with a database to help enrich the dialogue

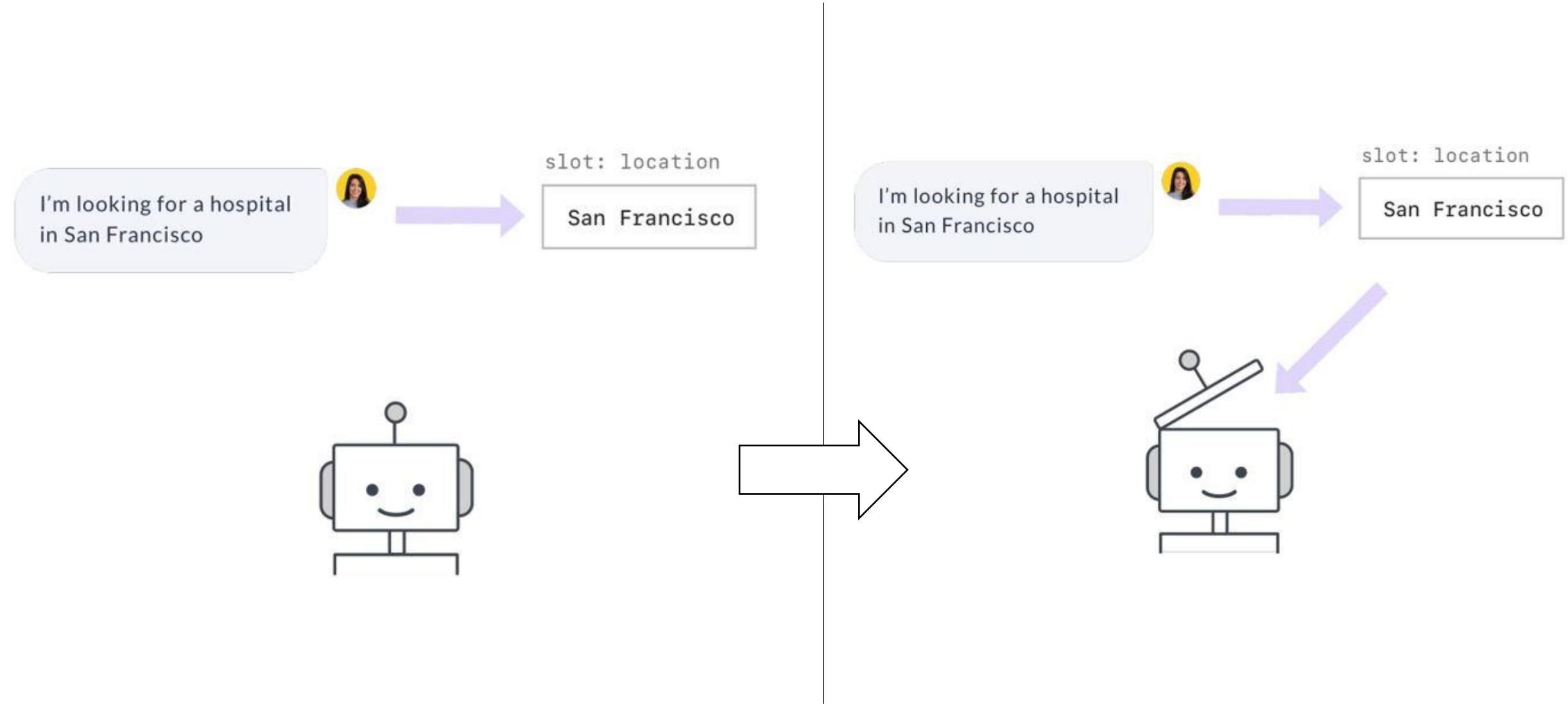


# Implementation: Slots

- **Slots** are key-value paired used to store important information
  - gathered from the user
  - gathered from the context
- It works as a **memory** for the chatbot

```
slots:  
  product:  
    type: text  
    influence_conversation: true  
  products:  
    type: list  
    influence_conversation: false  
  molecule:  
    type: text  
    influence_conversation: true  
  drug:  
    type: text  
    influence_conversation: true  
  sideeffect:  
    type: text  
    influence_conversation: true
```

# Implementation: Slots



# Use case implementation

# Use case implementation: Exercise

See *rasa\_chatbot\_implementation\_exercise.pdf* for the instructions

**Goal:** To implement a chatbot with Rasa framework for simple use cases (like asking for restaurant location)

## **Remark:**

- Do not hesitate to have a look at *rasa\_ebook.pdf* for documentation on Rasa

# Take-away from Course 5

- Chatbots are **increasingly present** in a lot of fields
- Chatbots are **still a developing area** heavily related to **AI** and **machine learning**
- Different **levels** of chatbots exist from the most **basic** to the most **sophisticated**
- Some useful **frameworks** however allow to develop rather quickly some interesting chatbot
- **Rasa** is one of those framework (**Tock**, not mentioned previously in the course, is another one)

# References Course 1

## Online formations

- <https://www.udemy.com/course/nlp-natural-language-processing-with-python>
- <https://www.coursera.org/specializations/natural-language-processing>

## Book

Koehn, Statistical Machine Translation, Cambridge University Press (2009)

# References Course 2

## Online formations

- <https://www.udemy.com/course/nlp-natural-language-processing-with-python>
- <https://www.coursera.org/specializations/natural-language-processing>
- <https://www.coursera.org/learn/natural-language-processing-tensorflow>

## Internet site

- <https://towardsdatascience.com/skip-gram-nlp-context-words-prediction-algorithm-5bbf34f84e0c>

## Book

Koehn, Statistical Machine Translation, Cambridge University Press (2009)

## Formation (for the *Optimizers* part)

Deep Learning with Tensorflow, *Publicis Sapient France*

# References Course 3

## Book

A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems* (2019)

## Online formations

- <https://www.udemy.com/course/nlp-natural-language-processing-with-python>
- <https://www.coursera.org/specializations/natural-language-processing>
- <https://www.coursera.org/learn/natural-language-processing-tensorflow>

## Internet sites

- <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- <https://blog.engineering.publicissapient.fr/2020/09/23/long-short-term-memory-lstm-networks-for-time-series-forecasting/>
- <https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577>

# References Course 4

## Book

A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems* (2019)

## Online formations

- <https://www.udemy.com/course/nlp-natural-language-processing-with-python>
- <https://www.coursera.org/specializations/natural-language-processing>
- <https://www.coursera.org/learn/natural-language-processing-tensorflow>

## Internet sites

- <https://arxiv.org/abs/1706.03762> (*Attention is all you need*)
- [https://www.youtube.com/watch?reload=9&v=OyFJWRnt\\_AY](https://www.youtube.com/watch?reload=9&v=OyFJWRnt_AY)
- <http://jalammar.github.io/illustrated-transformer/>
- <https://medium.com/dissecting-bert/dissecting-bert-part-1-d3c3d495cdb3>
- <https://camembert-model.fr/>
- <https://huggingface.co/>
- <https://lesdieuxducode.com/blog/2019/4/bert--le-transformer-model-qui-sentraine-et-qui-represente>
- [https://www.youtube.com/watch?v=S0zQoTkX\\_YQ](https://www.youtube.com/watch?v=S0zQoTkX_YQ) (github copilot)

## Online notebooks

- <https://colab.research.google.com/github/tensorflow/text/blob/master/docs/tutorials/transformer.ipynb>
- [https://colab.research.google.com/github/sarthakmalik/GPT2.Training.Google.Colaboratory/blob/master/Train\\_a\\_GPT\\_2\\_Text\\_Generating\\_Model\\_w\\_GPU.ipynb](https://colab.research.google.com/github/sarthakmalik/GPT2.Training.Google.Colaboratory/blob/master/Train_a_GPT_2_Text_Generating_Model_w_GPU.ipynb)
- [https://colab.research.google.com/github/ziadloo/attention\\_keras/blob/master/examples/colab/LSTM.ipynb](https://colab.research.google.com/github/ziadloo/attention_keras/blob/master/examples/colab/LSTM.ipynb)

# References Course 5

## Online formations

- <https://www.udemy.com/course/nlp-natural-language-processing-with-python>
- <https://www.coursera.org/specializations/natural-language-processing>

## Internet sites

- <https://rasa.com/docs/rasa/playground/>
- <https://blog.rasa.com/>
- [https://www.slideshare.net/PyData/deprecating-the-state-machine-building-conversational-ai-with-the-rasa-stack-justina-petraityt?from\\_action=save](https://www.slideshare.net/PyData/deprecating-the-state-machine-building-conversational-ai-with-the-rasa-stack-justina-petraityt?from_action=save)
- <https://info.rasa.com/masterclass-ebook>