



# Natural Language Processing

AIS

Romain Benassi

Course 2

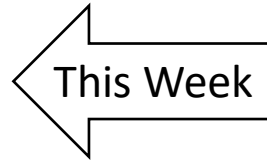
Spring 2022

# 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
- **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

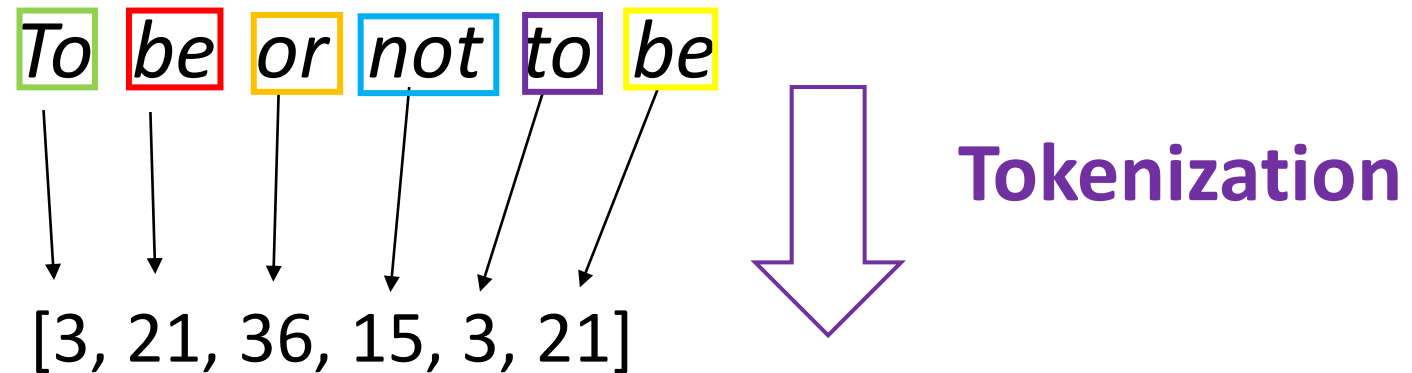
## Principle

*To be or not to be*

**Tokenization**

# Tokenization and one-hot encoding steps

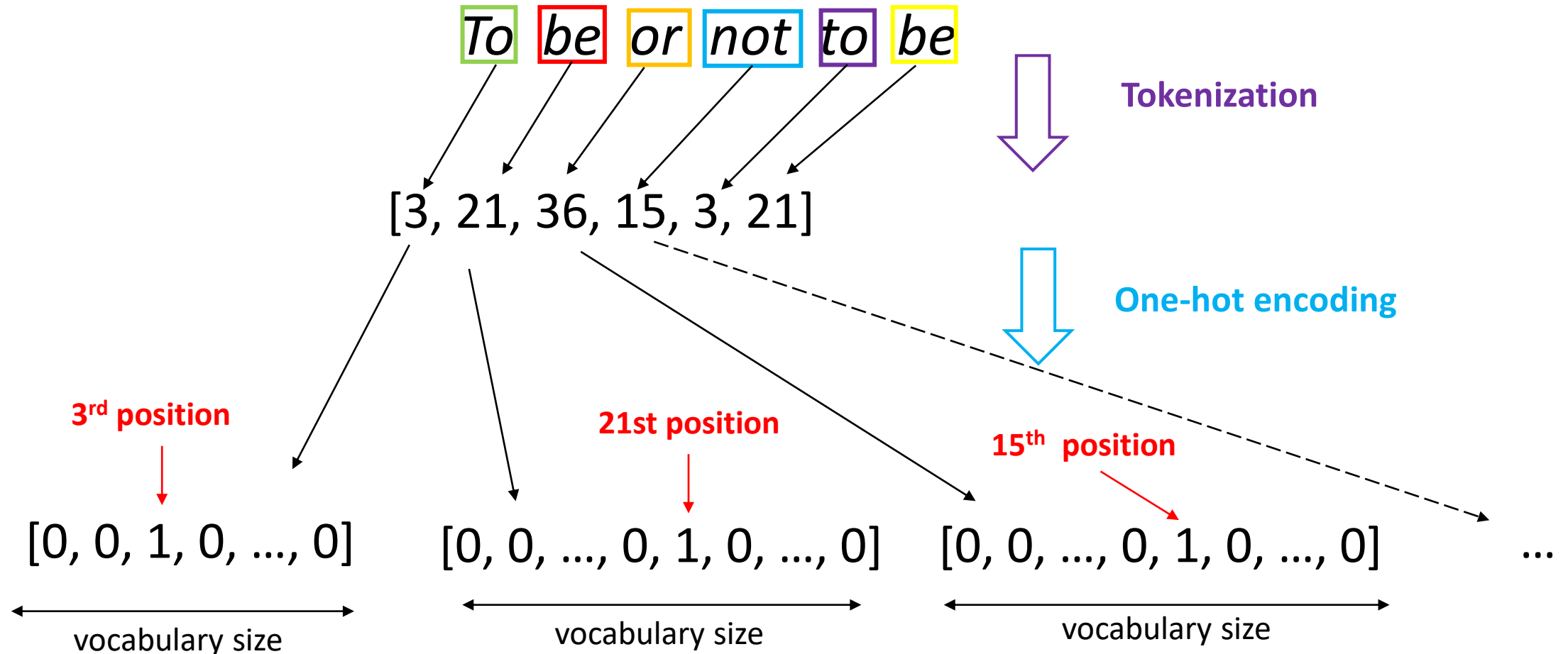
## Principle



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



# Tokenization and one-hot encoding steps

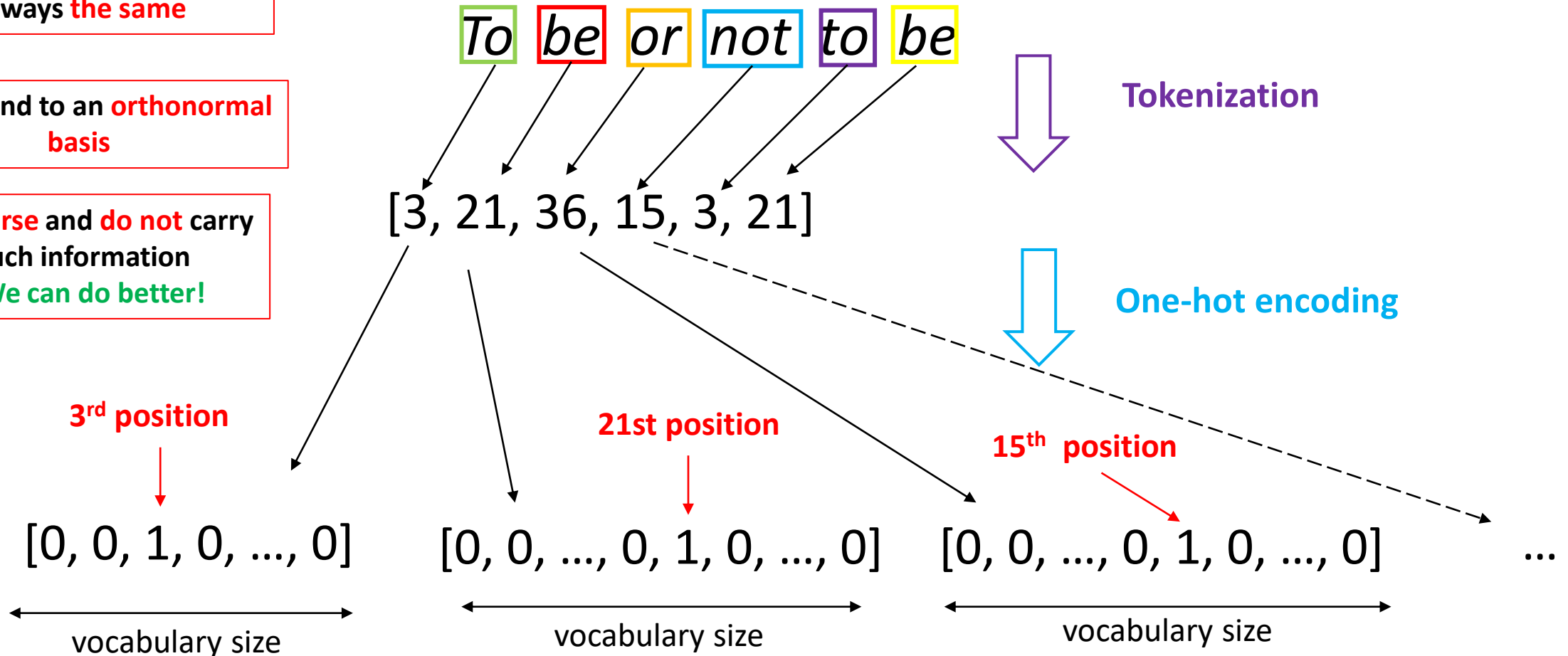


# Tokenization and one-hot encoding steps

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

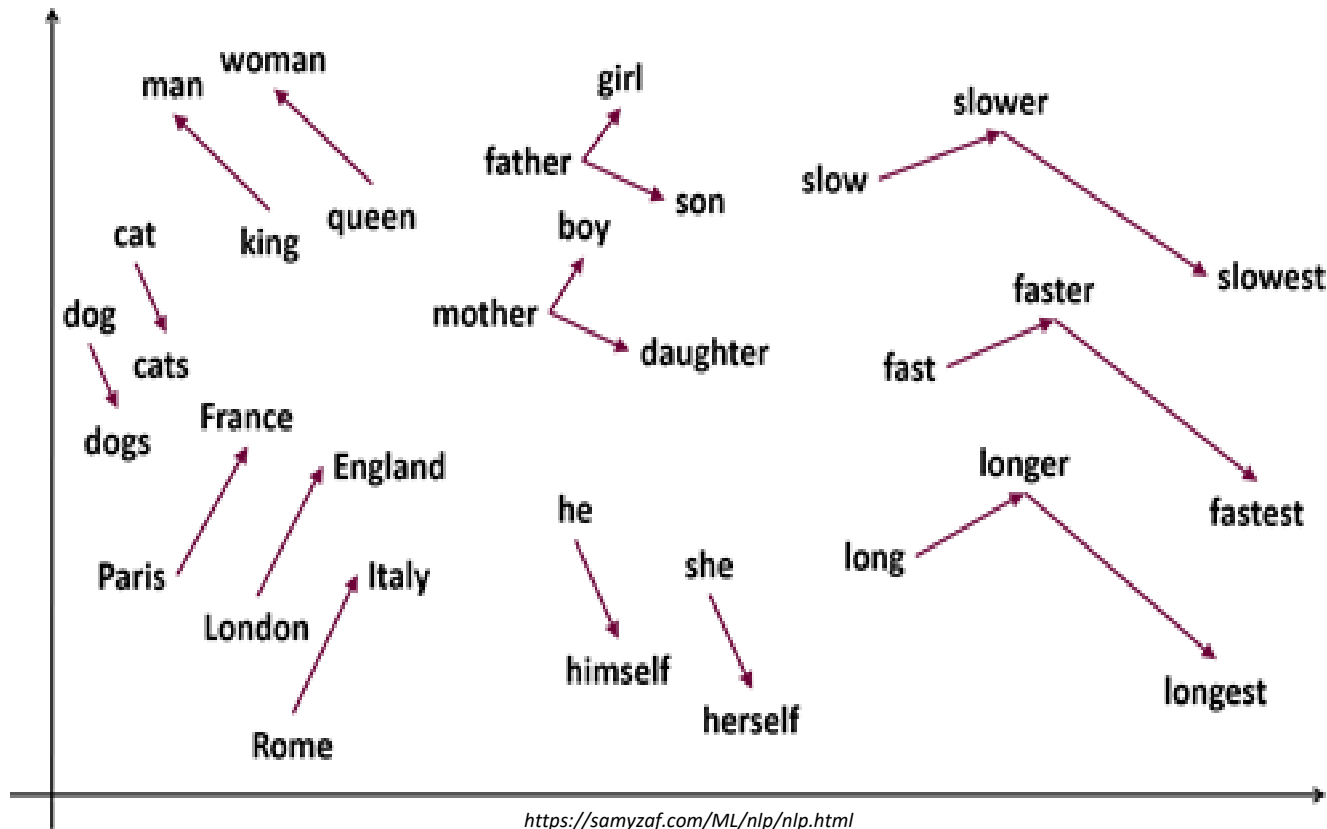
## Principles

- 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

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

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

# Word embedding: Illustration



king – man + woman  $\approx$  queen

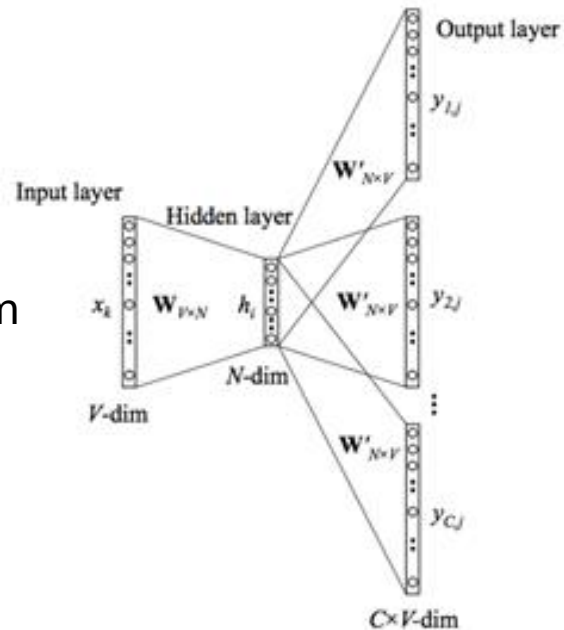
France – Paris + London  $\approx$  England

# Word embeddings: Principle

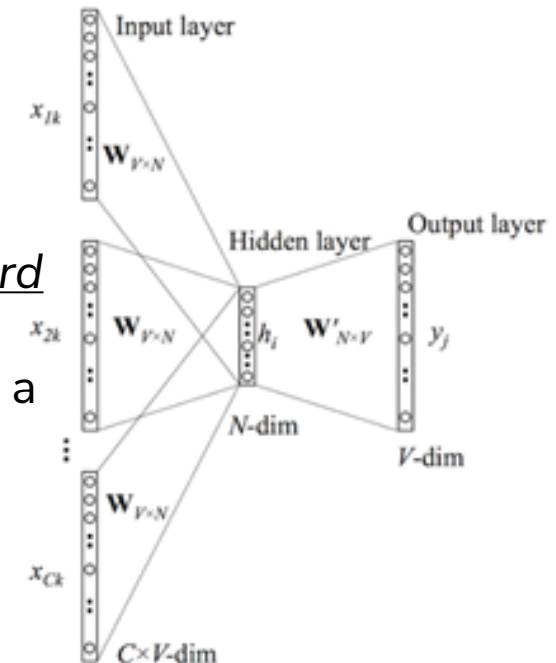
How are generated word embeddings?

- 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



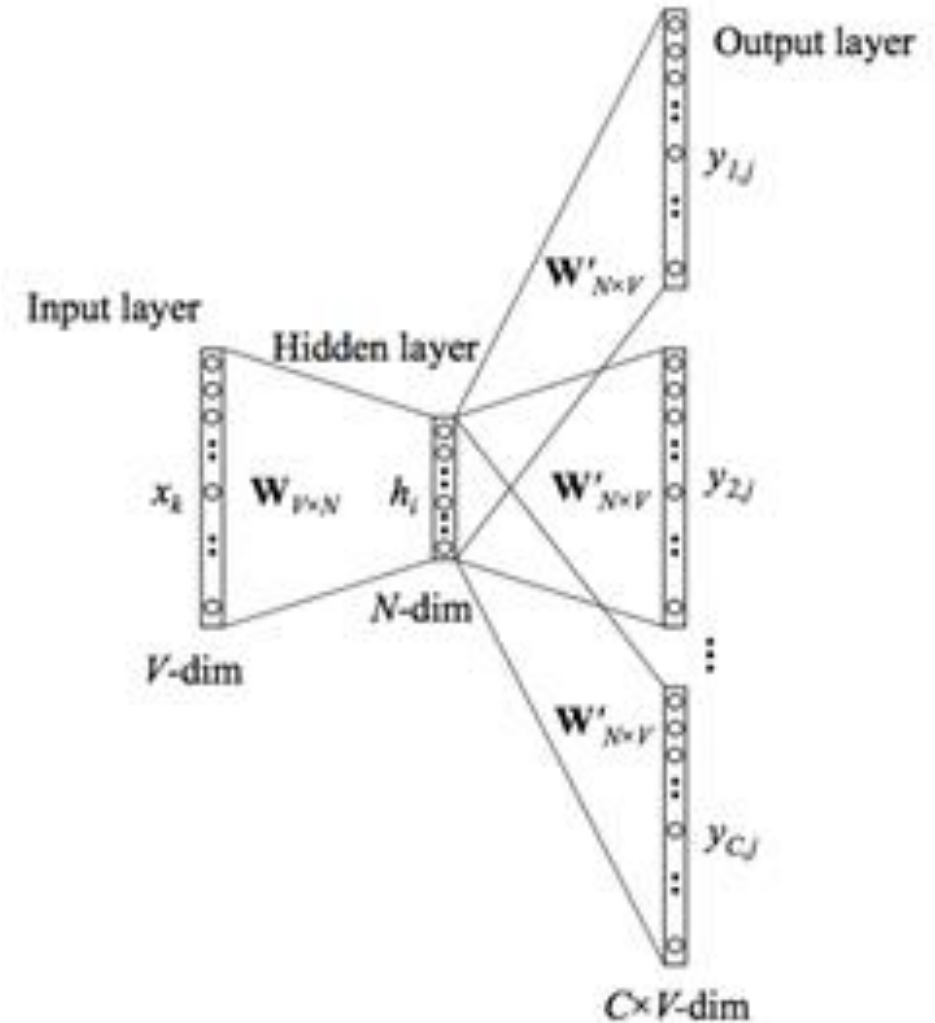
# Word embedding: Skip-gram model

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)



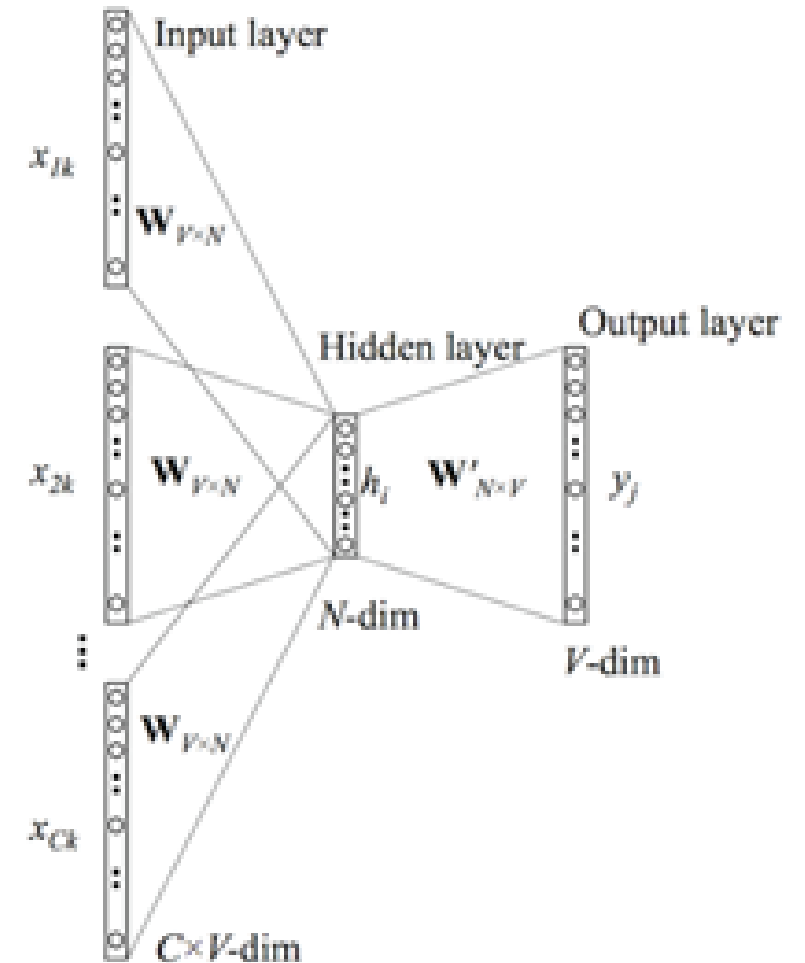
# Word embedding: CBOW model

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

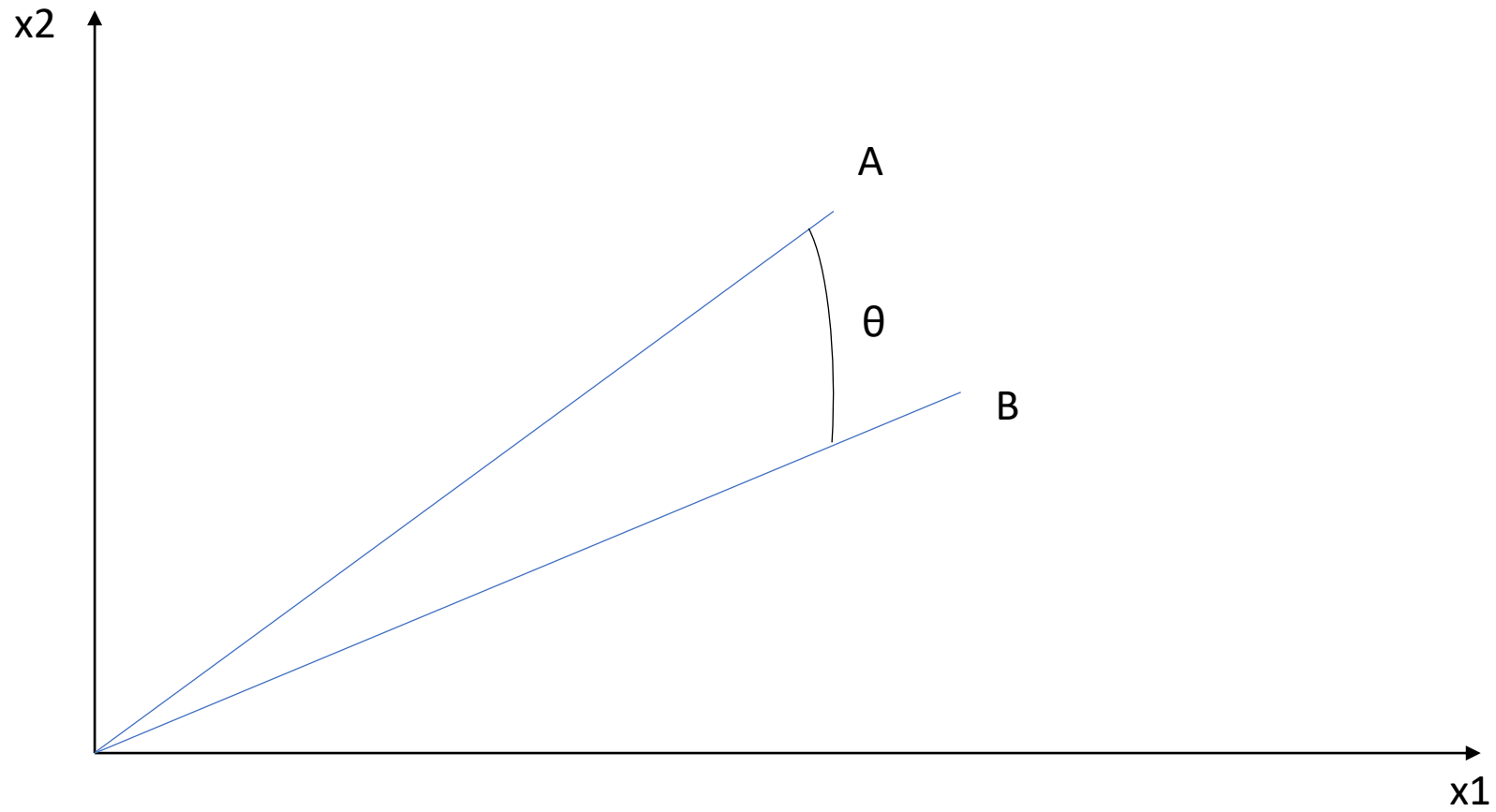
- *fastText* is an open-source library allowing to use **pre-trained word embeddings** (and text classifiers) for almost 300 languages
- It has been developed by Facebook's AI Research (FAIR) lab
- Unlike most of other embeddings, dependent of a vocabulary, fastText treats each word as composed of **N-grams** (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

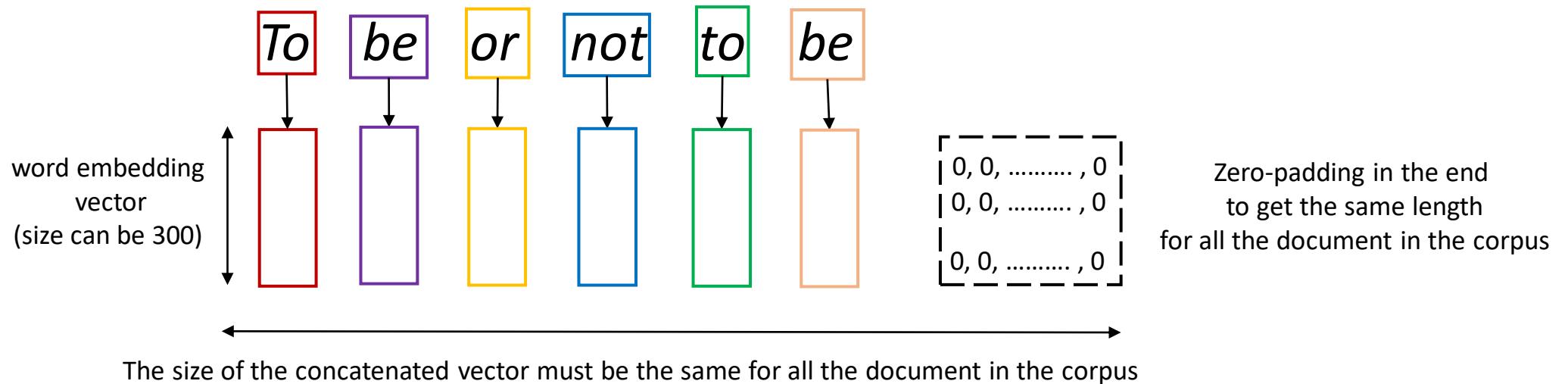


# Embedding: Text embedding

- 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



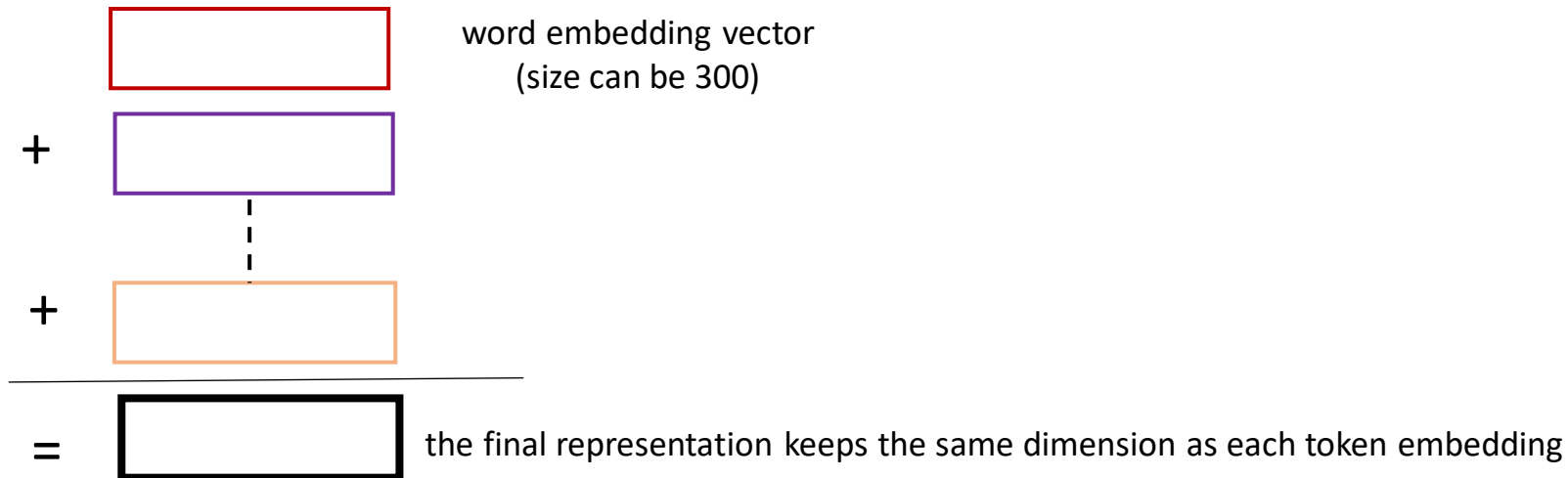
## Remarks

- 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

*To* *be* *or* *not* *to* *be*



## 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

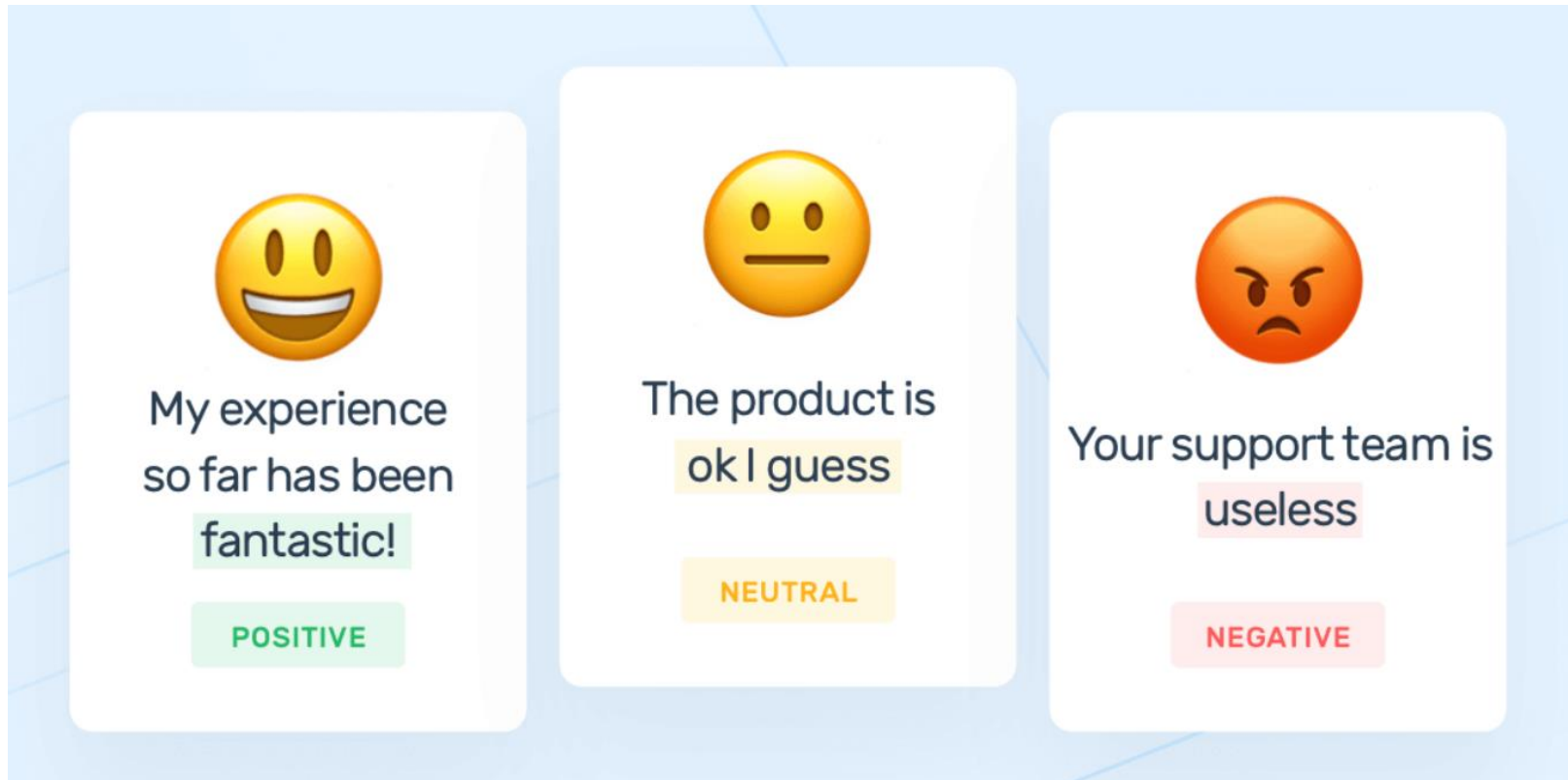
Some libraries offers to deal directly with **text** embeddings

- **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

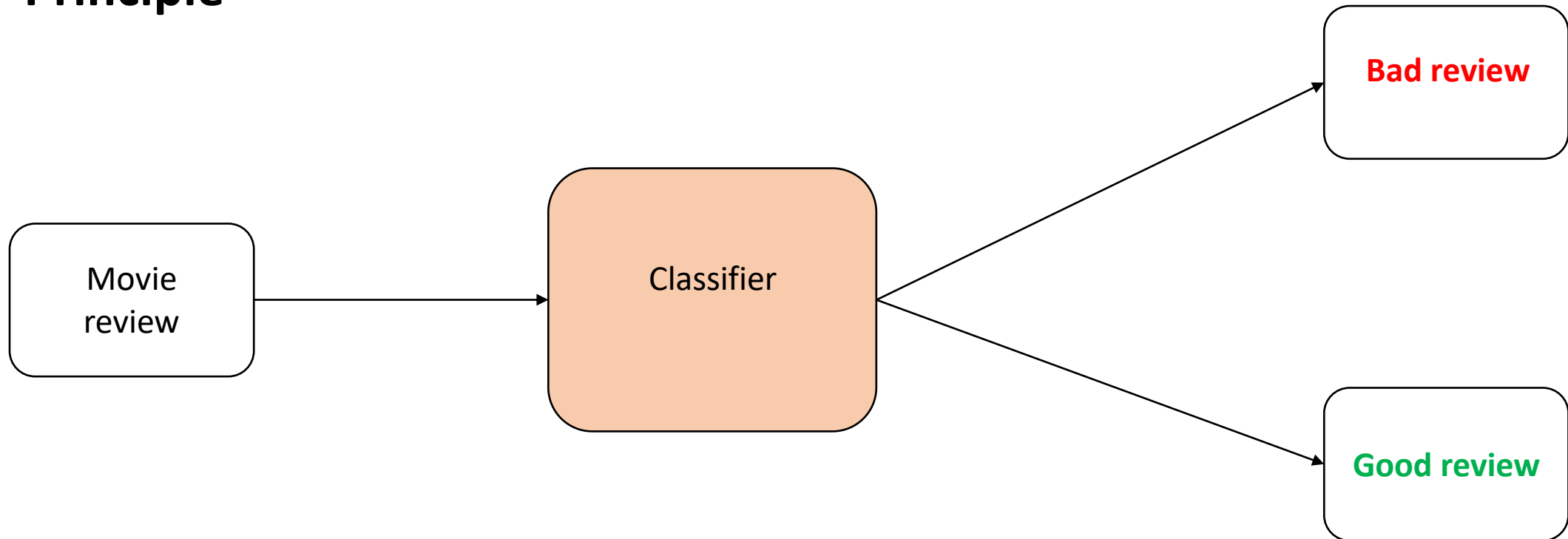
# Sentiment Analysis

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 **A**ware **D**ictionary for sEntiment **R**easoning) 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)

test_size = info.splits["test"].num_examples
test_set = datasets["test"].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)



# 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)

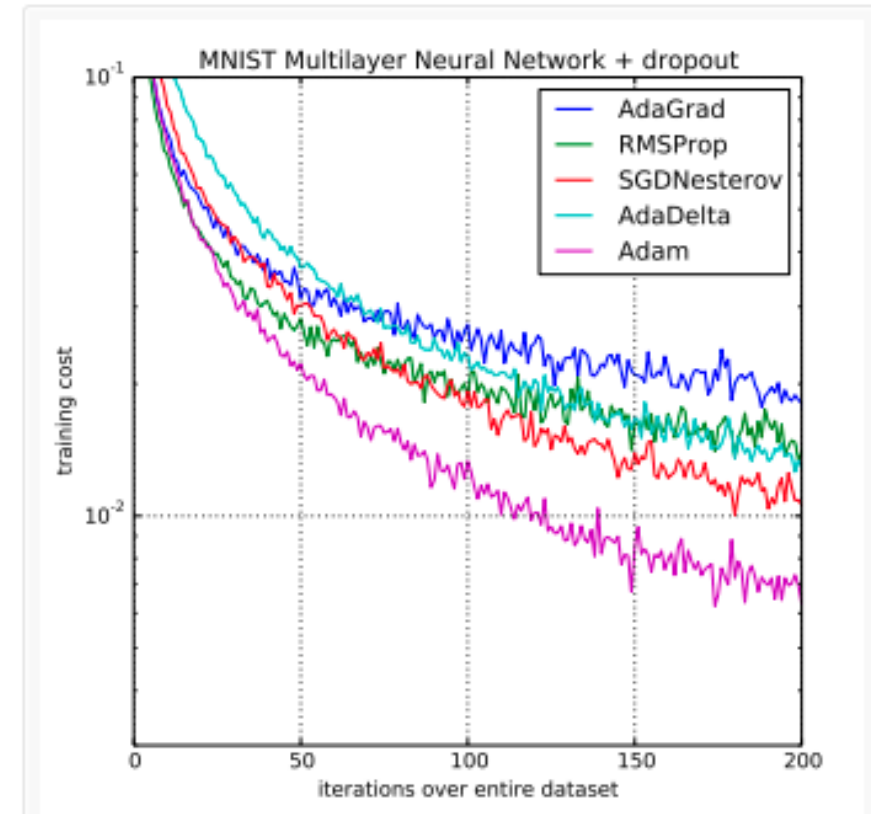
## Optimizers Variants

- 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

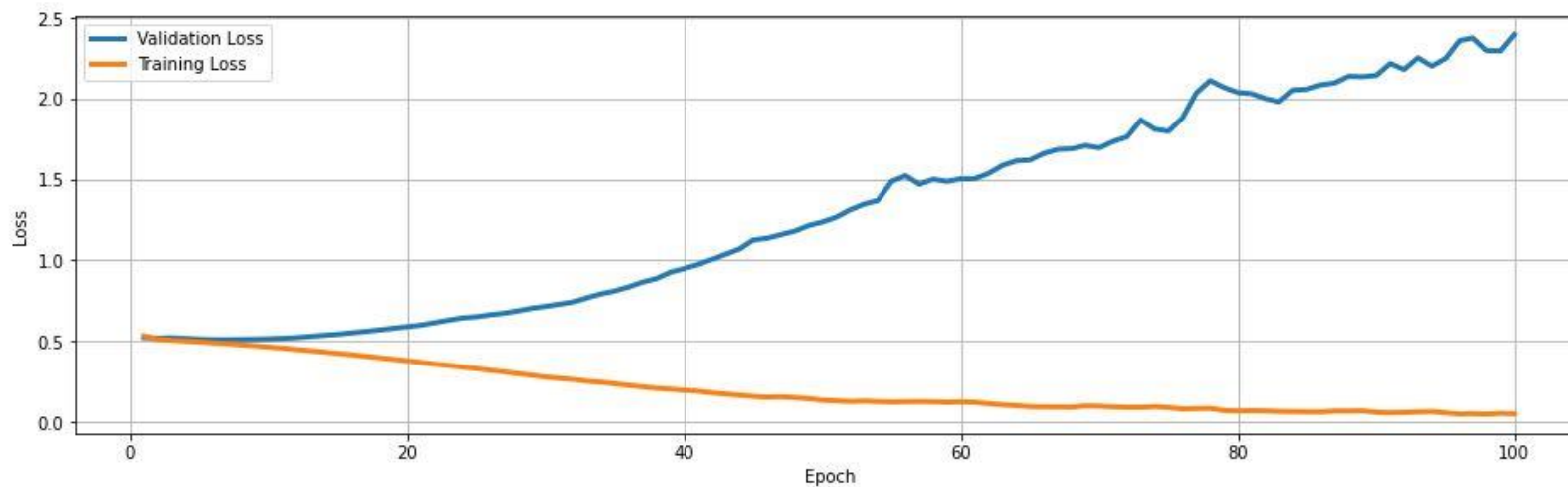
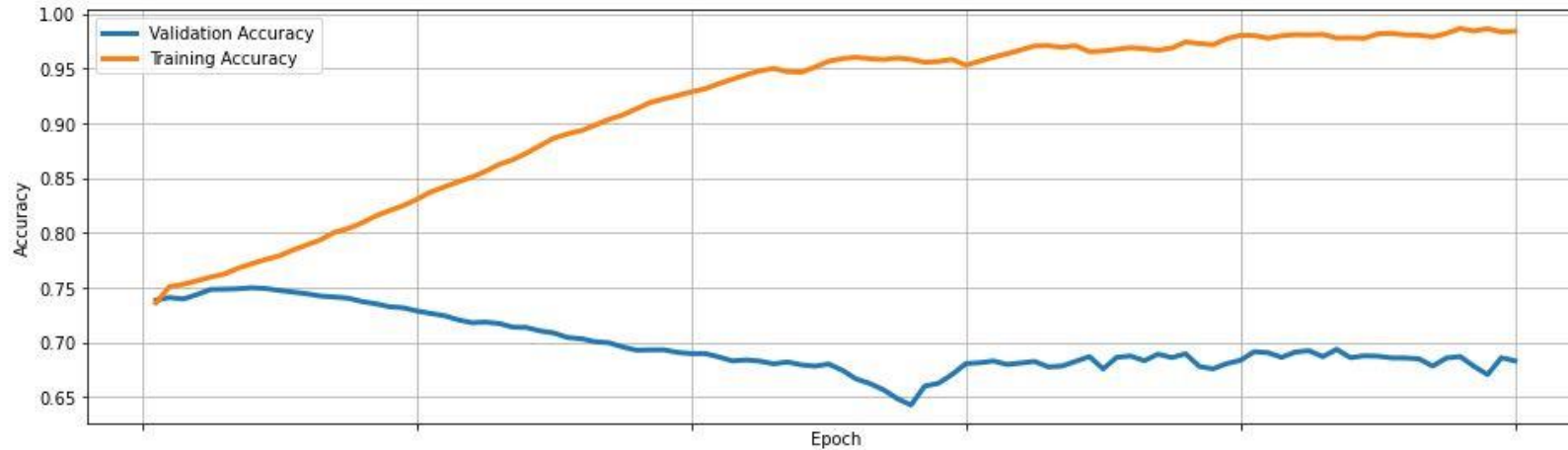
```
model.compile(loss="binary_crossentropy", optimizer=optimizer)
```



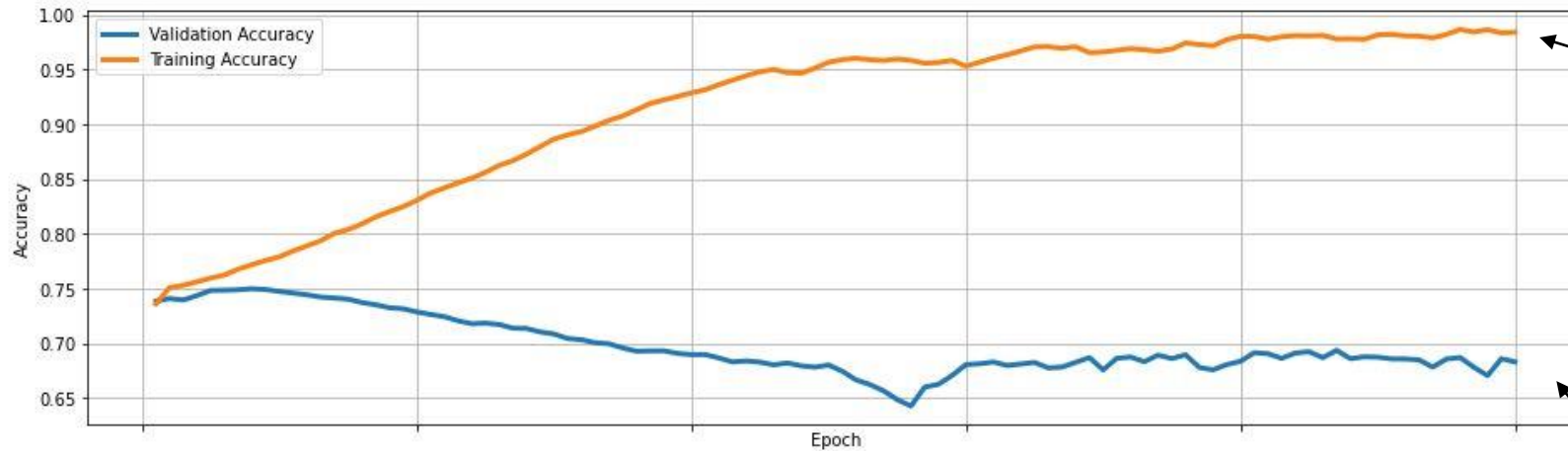
Comparison of Adam to Other Optimization Algorithms  
Training a Multilayer Perceptron  
Taken from Adam: A Method for Stochastic Optimization,  
2015.



# Sentiment Analysis: Neural Network (NN)

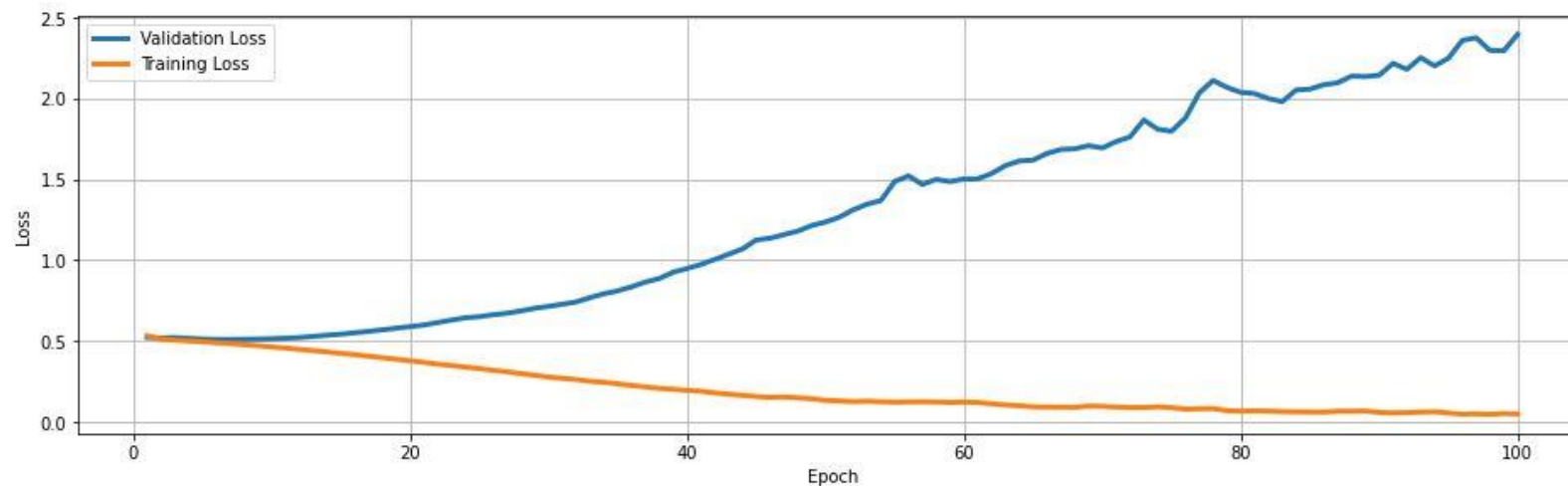


# Sentiment Analysis: Neural Network (NN)

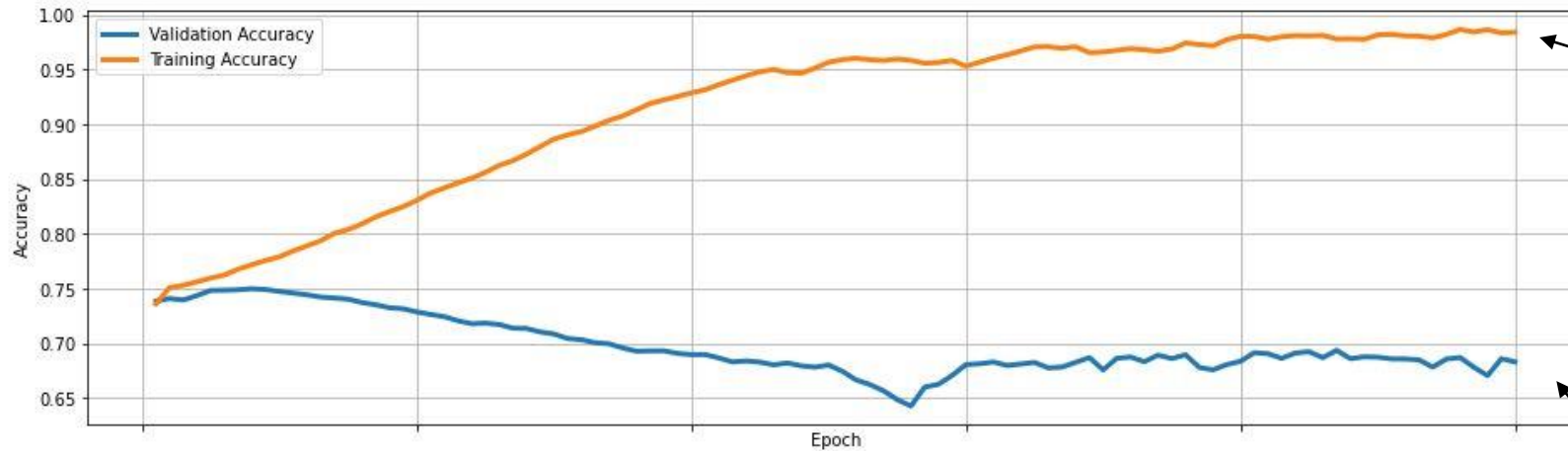


The results seem good on the train set...

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

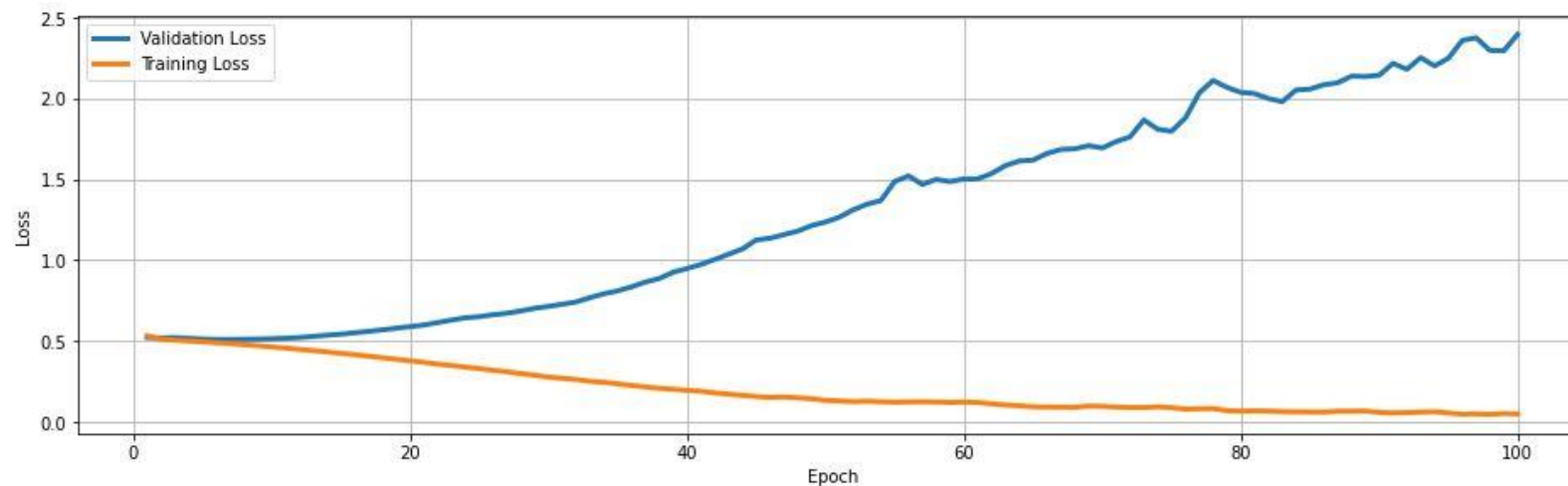


# Sentiment Analysis: Neural Network (NN)



The results seem good on the train set...

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

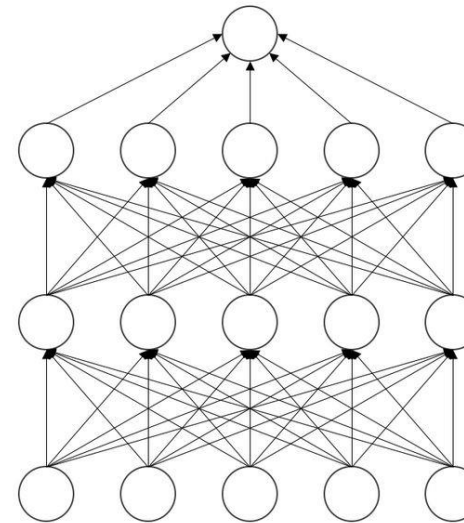


**Overfitting !!!**

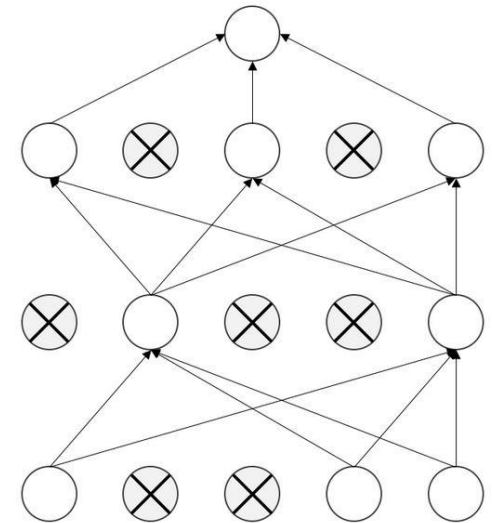
# Sentiment Analysis: Neural Network (NN)

- 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

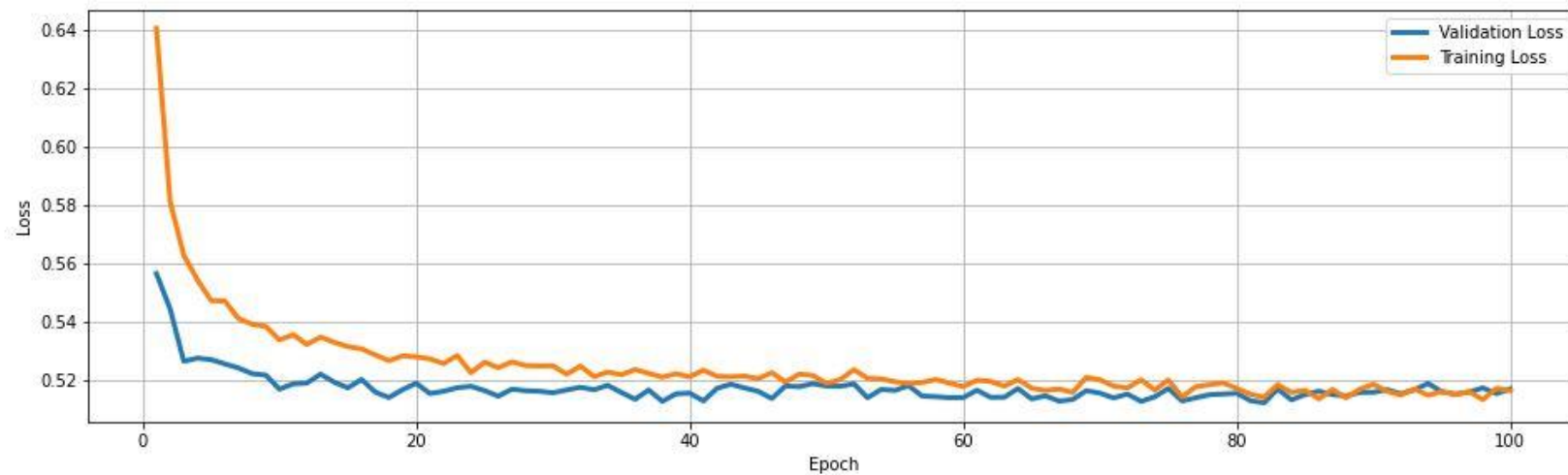
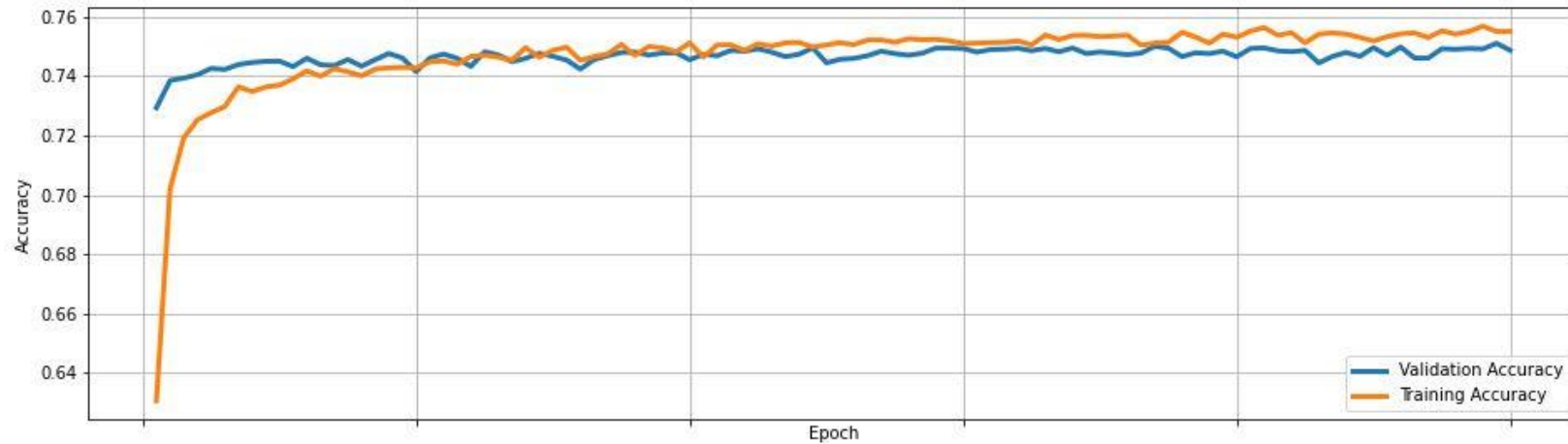
# Sentiment Analysis: Neural Network (NN)

▶ `model.summary()`

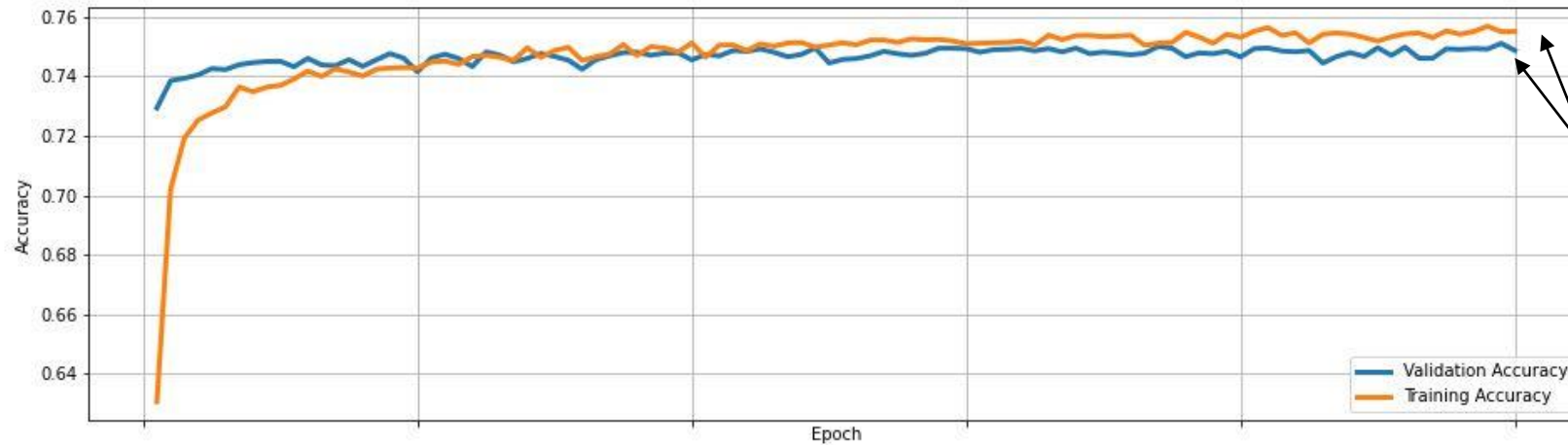
↳ Model: "sequential\_2"

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		

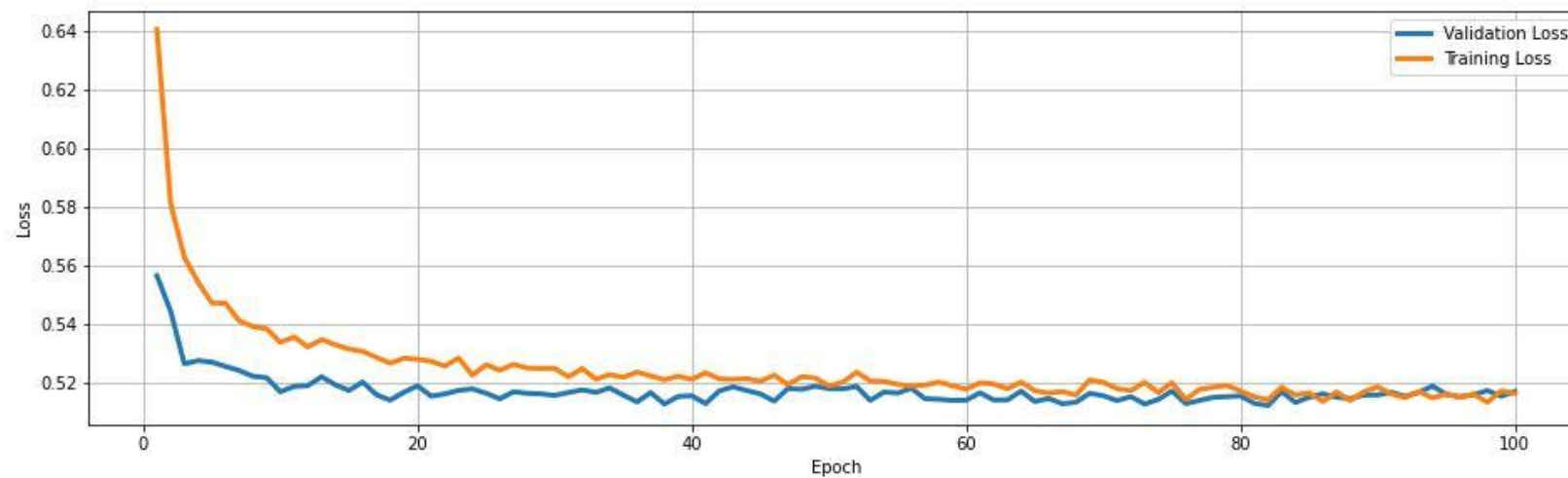
# Sentiment Analysis: Neural Network (NN)



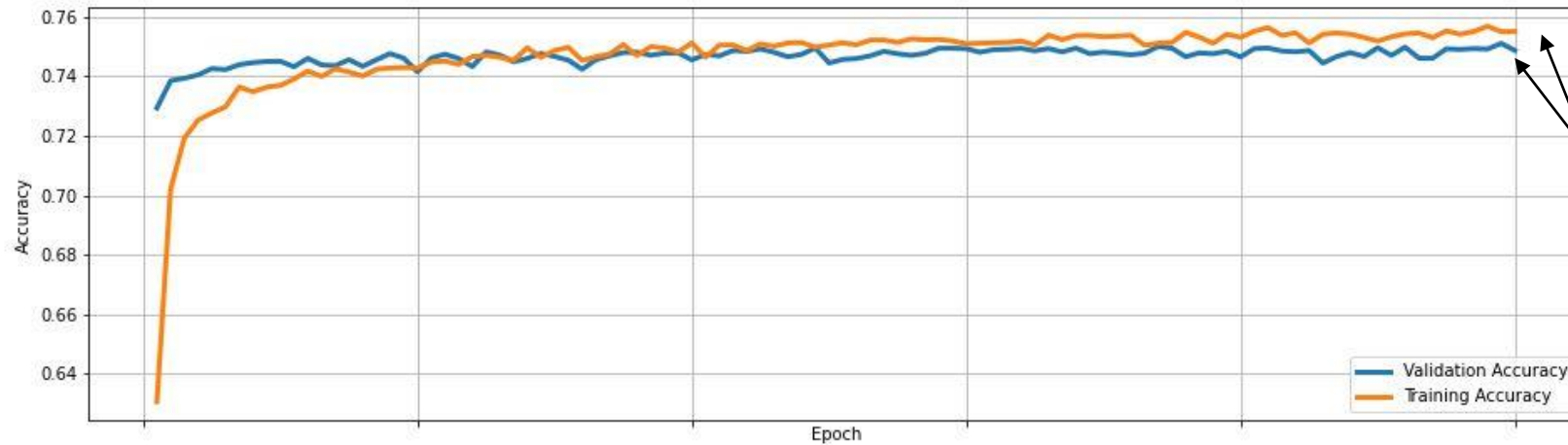
# Sentiment Analysis: Neural Network (NN)



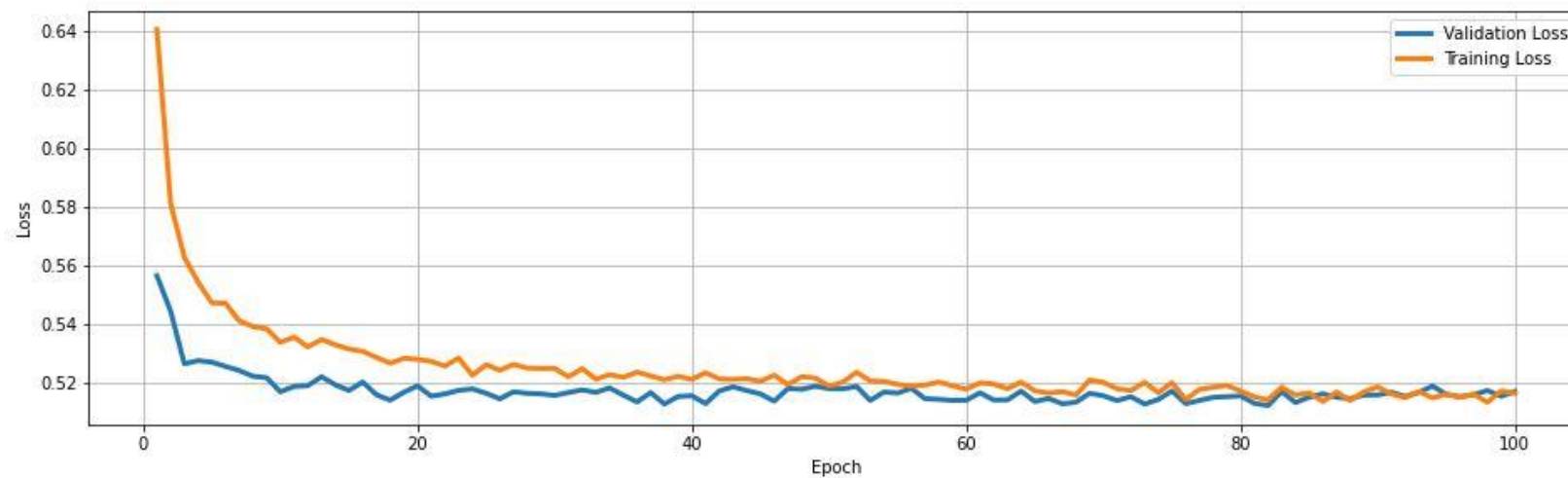
The results are similar for both train and test set



# 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**

# References

## 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*