Natural Language Processing

NLP Theory fundamentals

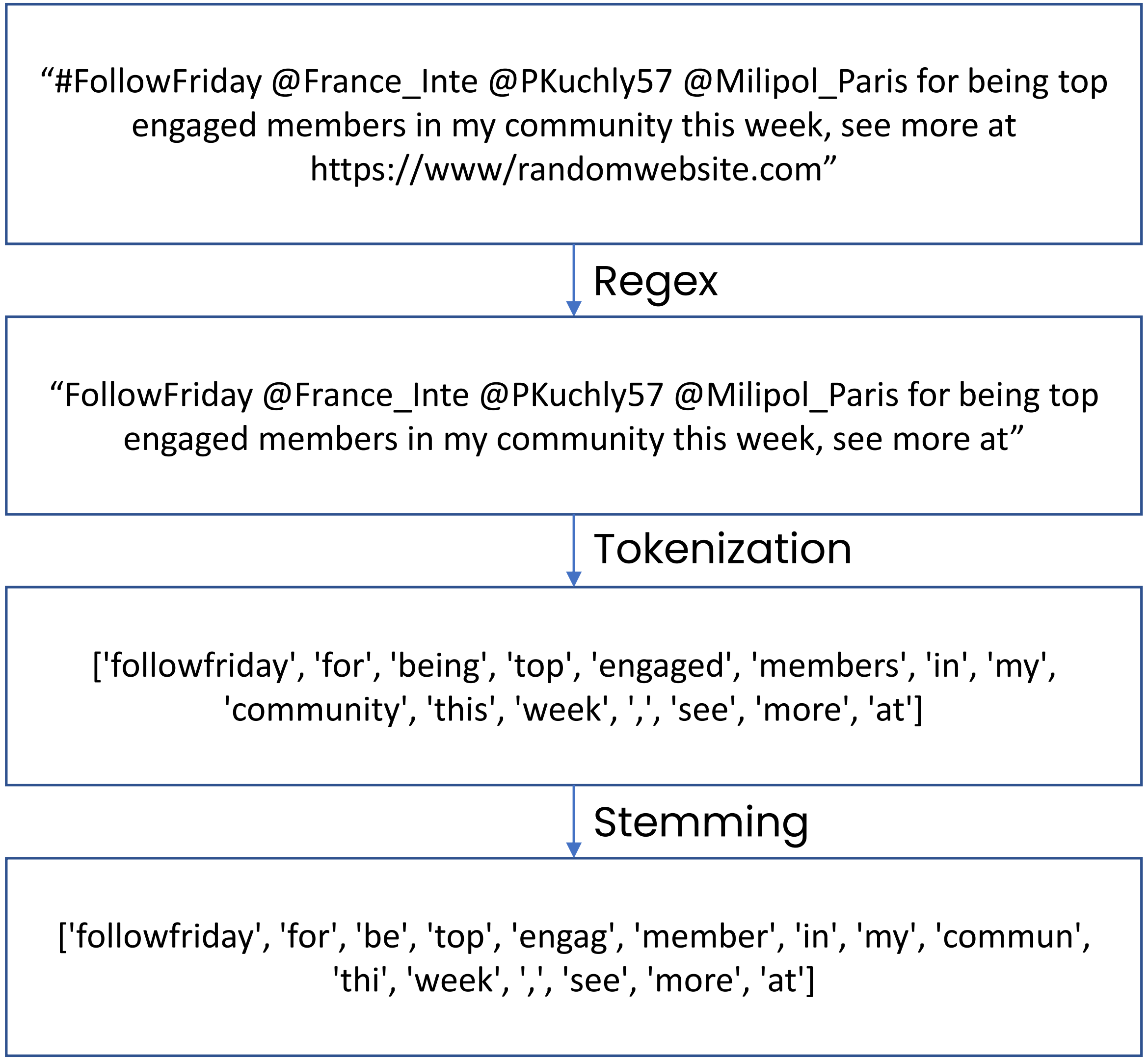
W2 Agenda

- Text preparation and tokenization
- Word embeddings and similarity
- Sequence models
- Attention & Transformers

Text preparation and tokenization

How to clean text before NLP tasks – text preprocessing

- Use REGEX first to remove any unwanted patterns
- Tokenize text into words
- Simplify words with stemming



“#FollowFriday @France_Inte @PKuchly57 @Milipol_Paris for being top engaged members in my community this week, see more at https://www/randomwebsite.com”

Regex

“FollowFriday @France_Inte @PKuchly57 @Milipol_Paris for being top engaged members in my community this week, see more at”

Tokenization

['followfriday', 'for', 'being', 'top', 'engaged', 'members', 'in', 'my', 'community', 'this', 'week', ',', 'see', 'more', 'at']

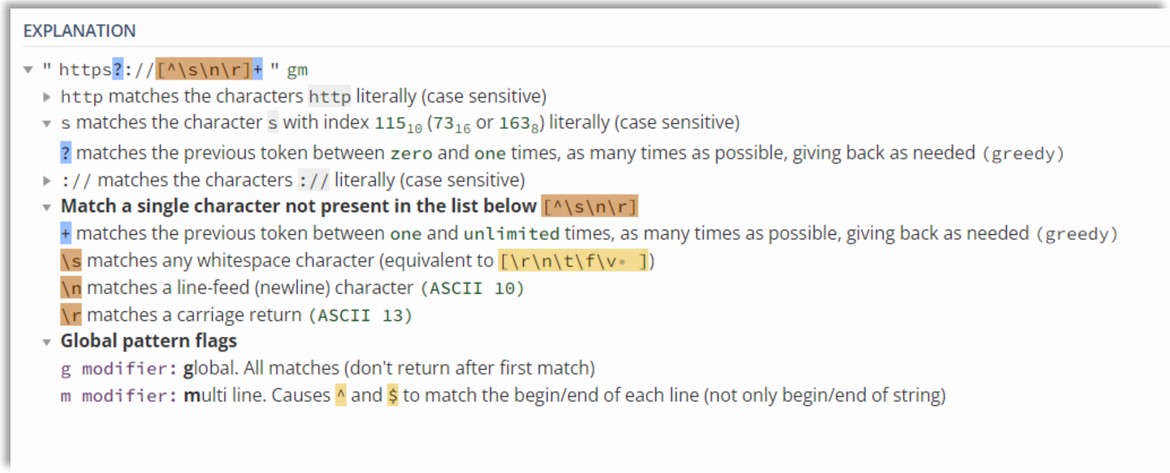
Stemming

['followfriday', 'for', 'be', 'top', 'engag', 'member', 'in', 'my', 'commun', 'thi', 'week', ',', 'see', 'more', 'at']

REGEX – „simple” but indispensable

- Each REGEX should start with… checking stack-overflow
- How to remove all hyperlinks from text with REGEX?
- re.sub(r'https?://[^\s\n\r]+', ‘’, text)

REGEX 101 helps with testing and interpretability



EXPLANATION

" https?://[^\s\n\r]+ " gm

http matches the characters http literally (case sensitive)

s matches the character s with index 115_{10} (73_{16} or 163_{8}) literally (case sensitive)

? matches the previous token between zero and one times, as many times as possible, giving back as needed (greedy)

:// matches the characters :// literally (case sensitive)

Match a single character not present in the list below [^\s\n\r]

+ matches the previous token between one and unlimited times, as many times as possible, giving back as needed (greedy)

\s matches any whitespace character (equivalent to [\r\n\t\f\v])

\n matches a line-feed (newline) character (ASCII 10)

\r matches a carriage return (ASCII 13)

Global pattern flags

g modifier: global. All matches (don't return after first match)

m modifier: multi line. Causes ^ and $ to match the begin/end of each line (not only begin/end of string)

Tokenization depends on generalization and some real life case text examples is not tokenizable unless small changes are applied

- If you have a mix of characters and numbers some tokenizers will fail e.g. *„Price for this appartment is 500.0zł”*
- Same applies to use of special characters like ["/",":", "-", "\\","+","<", ">"] when used without spaces

Tokenization – BERT example

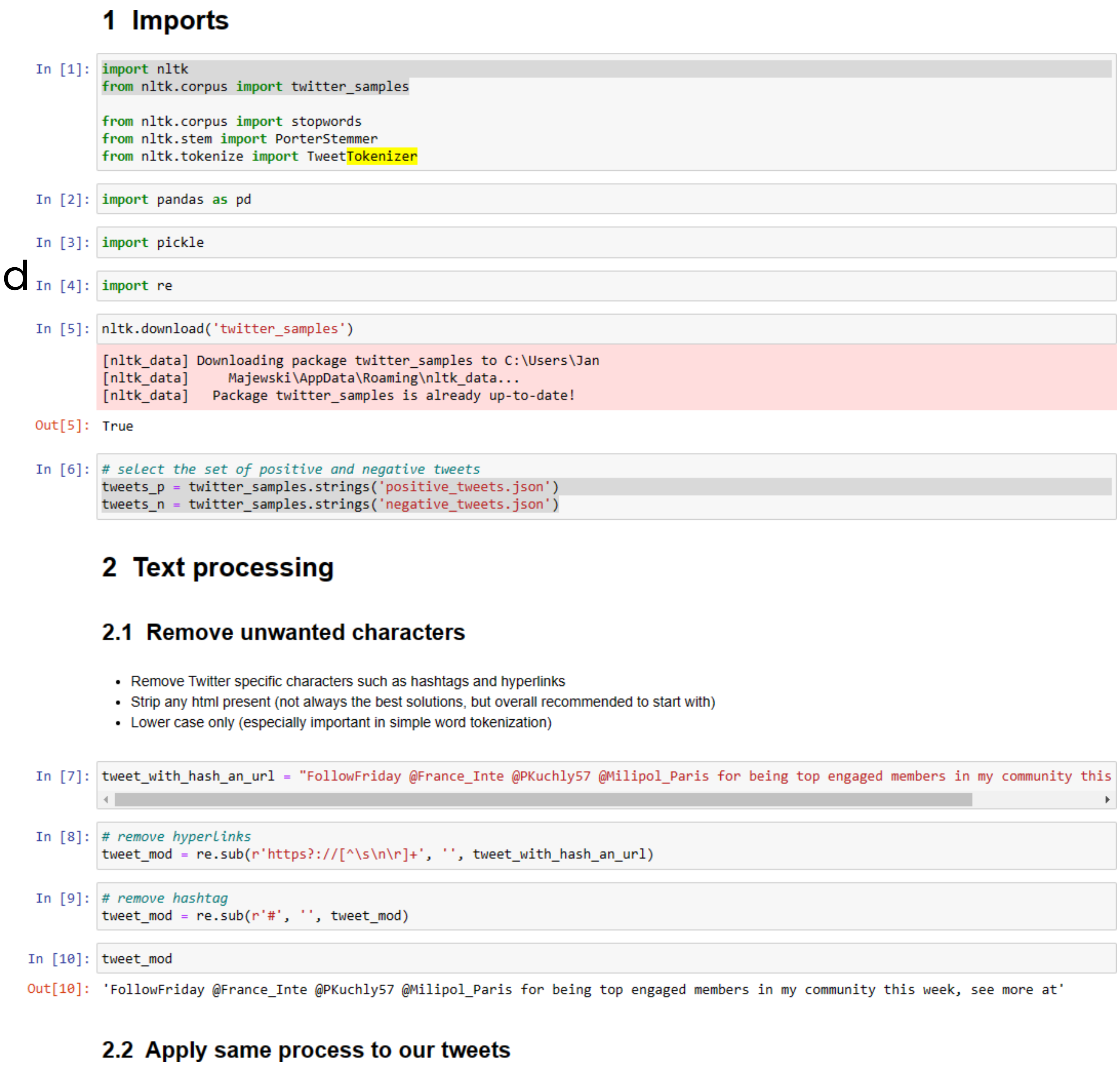
- SoTA Tokenizers are more complex in handling words
- They will have some most common words in their vocabulary
- But for less frequent ones are represented by multiple tokens
- Some words can not be present in tokenizer vocab, they would et an OOV token
- in BERT tokenizer case they would probably still be tokenized as chars

BERT Tokenizer example

- Spell has its own token – 6297
- Mispelled will be tokenized as [3335, 11880, 3709], which corresponds to ['miss', '##pel', '##led']

Jupyter notebook part 1:

Proceed to notebook Text-preprocessing-and-vectorization and follow steps 1-5



1 Imports

```
In [1]: import nltk
        from nltk.corpus import twitter_samples

        from nltk.corpus import stopwords
        from nltk.stem import PorterStemmer
        from nltk.tokenize import TweetTokenizer
```

```
In [2]: import pandas as pd
```

```
In [3]: import pickle
```

```
In [4]: import re
```

```
In [5]: nltk.download('twitter_samples')

        [nltk_data] Downloading package twitter_samples to C:\Users\Jan
        [nltk_data]     Majewski\AppData\Roaming\nltk_data...
        [nltk_data]   Package twitter_samples is already up-to-date!

Out[5]: True
```

```
In [6]: # select the set of positive and negative tweets
        tweets_p = twitter_samples.strings('positive_tweets.json')
        tweets_n = twitter_samples.strings('negative_tweets.json')
```

2 Text processing

2.1 Remove unwanted characters

- Remove Twitter specific characters such as hashtags and hyperlinks
- Strip any html present (not always the best solutions, but overall recommended to start with)
- Lower case only (especially important in simple word tokenization)

```
In [7]: tweet_with_hash_an_url = "FollowFriday @France_Inte @PKuchly57 @Milipol_Paris for being top engaged members in my community this
```

```
In [8]: # remove hyperlinks
        tweet_mod = re.sub(r'https?://[^\s\n\r]+', '', tweet_with_hash_an_url)
```

```
In [9]: # remove hashtag
        tweet_mod = re.sub(r'#', '', tweet_mod)
```

```
In [10]: tweet_mod

Out[10]: 'FollowFriday @France_Inte @PKuchly57 @Milipol_Paris for being top engaged members in my community this week, see more at'
```

2.2 Apply same process to our tweets

Word vectors and similarity

How to vectorize words

- Translating words into vectors might sound abstrack at first but with a text corpus large enought we can vectorize words based on the context they appear in
- We can approach word vectors by word-by-word approach if we want to classify their meaning
- Or any other context e.g. word-by-doc if we want to Focus on task-specific vectors, which will allow us to use text vectorization to classify text by its domain

Data Science is **popular** these days

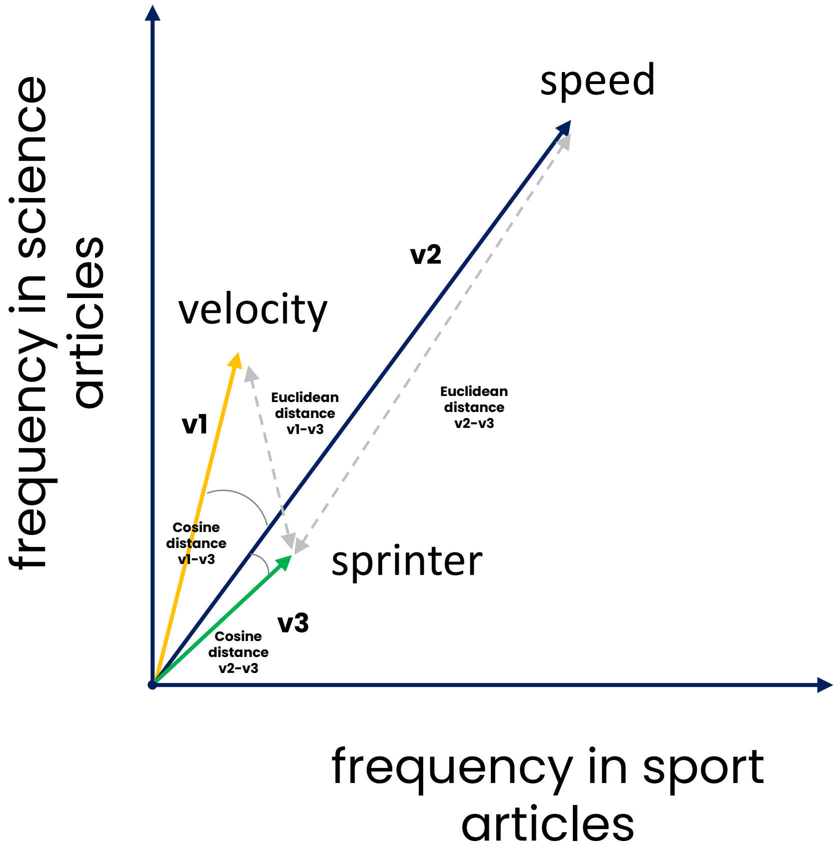
Quality of **data** is **popular** issue among companies

Co-occurrence matrix allows gives us a meaningful word vector based on words

	popular	science	quality
data	2	1	1

How to measure similarity

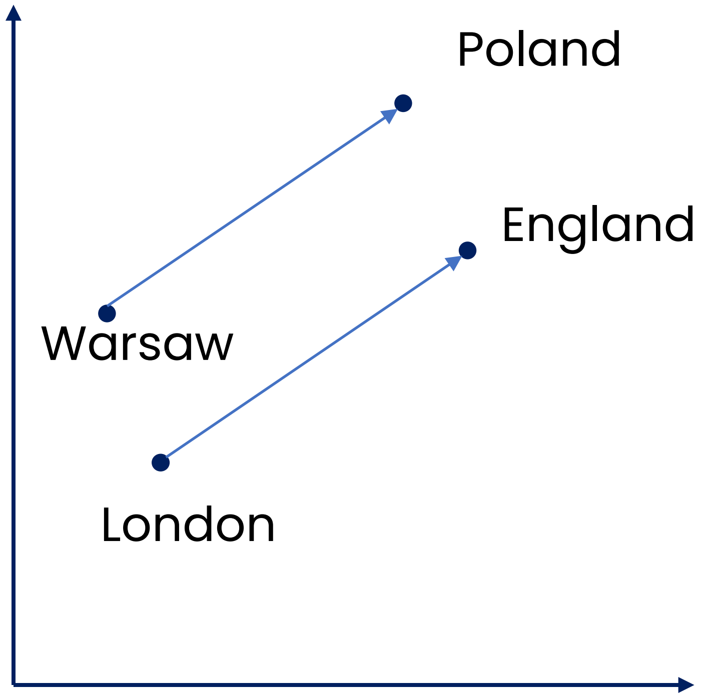
- Measuring vector similarity is challenging, especially in high dimensionality
- While Euclidean distance is easily applicable for 2D data it performs badly with higher dimensions
- Cosine distance is most popular in measuring similarity
- It is based on vectors direction and ranges from -1 (opposing vectors) to 1 (proportional vectors), while orthogonal vectors have a similarity of 0.



Word vectors allow us to actually extract relations between words

- If we take two word pairs – (Poland, Warsaw) and (England, London) both these pairs will create a similar vector
- This is the product of similar context they are present in
- If we have a large corpus of news, and articles there will be multiple coocurences of phrases like „{Warsaw/London} is the capital city of {Poland, England}”

Word vector space



Word vectors are not just some abstract conversion of words, to numbers. As they are based on words context and coocurence vectors keep some of this information

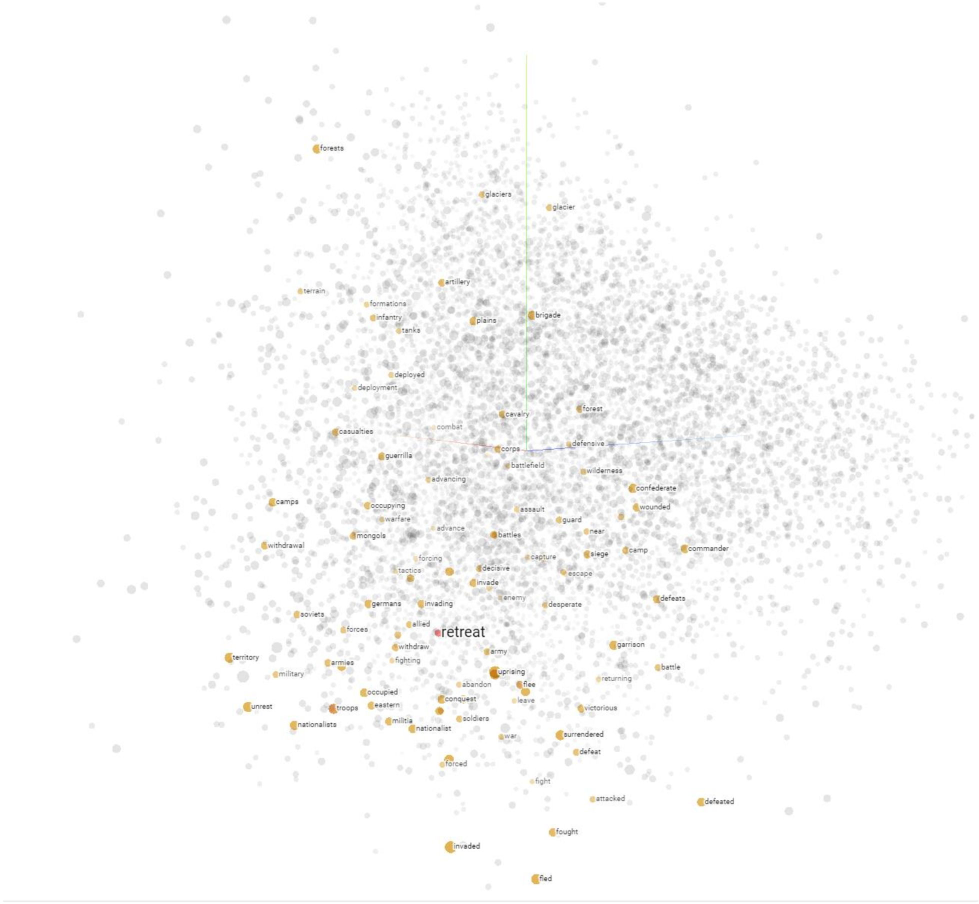
Classical word embedding method examples

- **word2vec** (Google, 2013)
- ***Continuous bag-of-words (CBOW):*** the model learns to predict the center word given some context words.
- ***Continuous skip-gram / Skip-gram with negative sampling (SGNS)***: the model learns to predict the words surrounding a given input word.
- ***Global Vectors (GloVe) (Stanford, 2014)*****:** factorizes the logarithm of the corpus's word co-occurrence matrix, similar to the count matrix you've used before.
- ***fastText (Facebook, 2016)****:* based on the skip-gram model and takes into account the structure of words by representing words as an n-gram of characters. It supports out-of-vocabulary (OOV) words.

Classical (non Deep learning) word embeddings were an important step in NLP development and are great for understanding the basics, but they are not used that much in real life use cases anymore

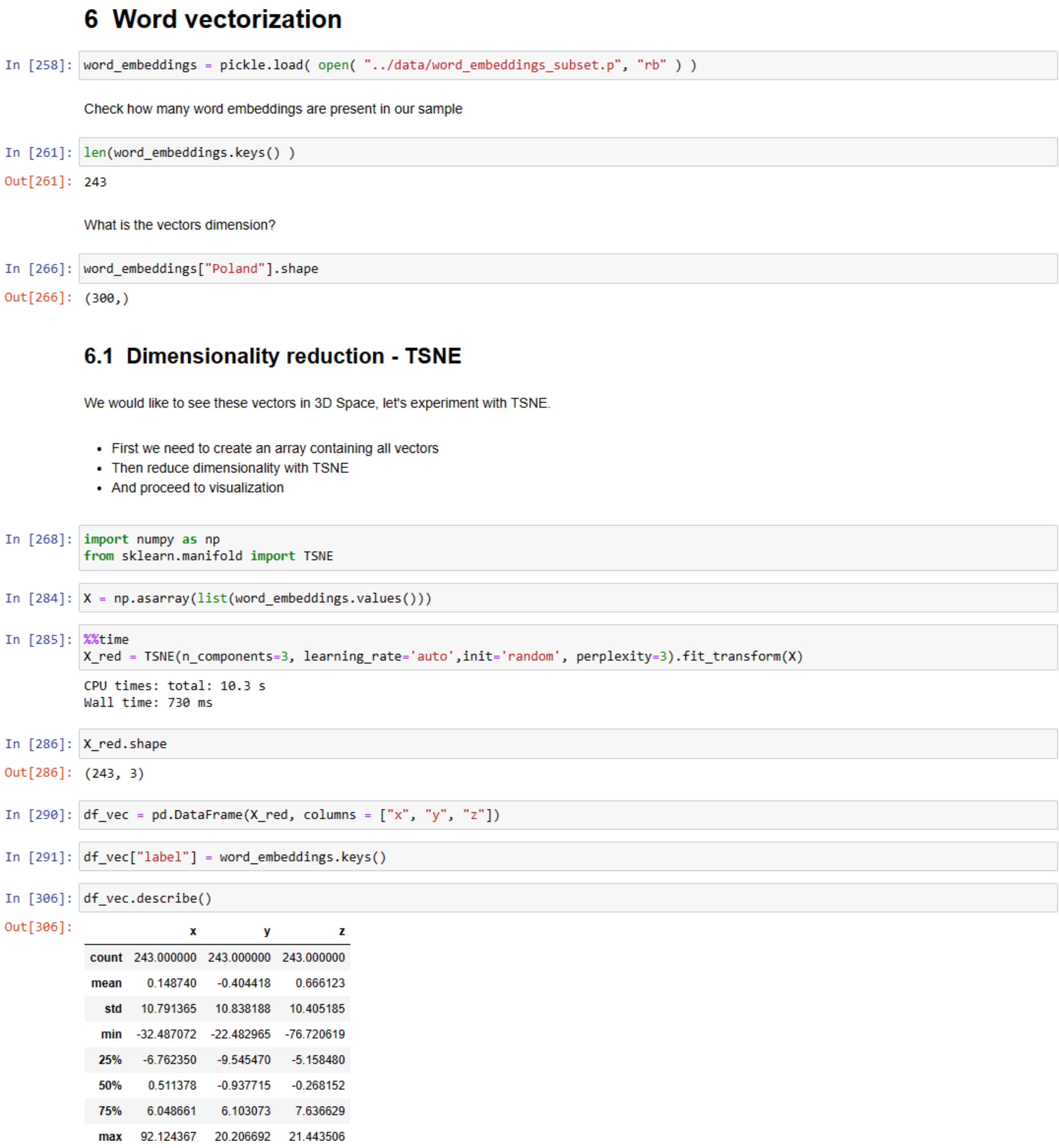
Visualizing vectors – dimensionality reduction

- Multi-dimensional vectors are hard to visualize as we can see only 3D
- With word vectors they tend to have hundreds of dimensions
- Depending on the task, majority of the information might be hidden within ~5% of dimensions only
- We use Dimensionality Reduction algorithms to visualize similarities in high-dimensional space without loosing too much information
- Most commonly used algorithms include T-SNE or PCA
- Explained variance represents loss of information due to dimensionality reduction
- Experiment yourself on https://projector.tensorflow.org/



Jupyter notebook part 2:

Proceed to notebook Text-preprocessing-and-vectorization and follow steps 6-8



6 Word vectorization

```
In [258]: word_embeddings = pickle.load( open( "../data/word_embeddings_subset.p", "rb" ) )
```

Check how many word embeddings are present in our sample

```
In [261]: len(word_embeddings.keys() )

Out[261]: 243
```

What is the vectors dimension?

```
In [266]: word_embeddings["Poland"].shape

Out[266]: (300,)
```

6.1 Dimensionality reduction - TSNE

We would like to see these vectors in 3D Space, let's experiment with TSNE.

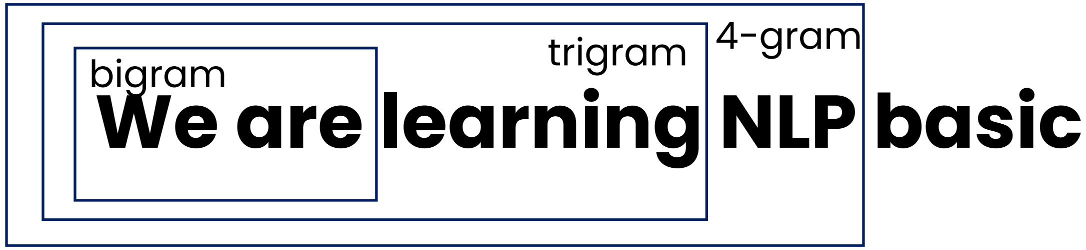
- First we need to create an array containing all vectors
- Then reduce dimensionality with TSNE
- And proceed to visualization

```
In [268]: import numpy as np
          from sklearn.manifold import TSNE

In [284]: X = np.asarray(list(word_embeddings.values()))

In [285]: %%time
          X_red = TSNE(n_components=3, learning_rate='auto',init='random', perplexity=3).fit_transform(X)

          CPU times: total: 10.3 s
          Wall time: 730 ms

In [286]: X_red.shape

Out[286]: (243, 3)

In [290]: df_vec = pd.DataFrame(X_red, columns = ["x", "y", "z"])

In [291]: df_vec["label"] = word_embeddings.keys()

In [306]: df_vec.describe()

Out[306]:
```

	x	y	z
count	243.000000	243.000000	243.000000
mean	0.148740	-0.404418	0.666123
std	10.791365	10.838188	10.405185
min	-32.487072	-22.482965	-76.720619
25%	-6.762350	-9.545470	-5.158480
50%	0.511378	-0.937715	-0.268152
75%	6.048661	6.103073	7.636629
max	92.124367	20.206692	21.443506

What are N-grams?

- N-grams help determining probability of words happening in specific sequence
- One of their common use cases is autocorrect
- Their complexity explodes once we go into higher n-values
- N-grams are not used very often as sequential models and transformers perform better in the majority of cases
- Their are however important fundation for NLP development



Leveraging Neural Neutworks in embeddings

- There are multiple ways of word embeddings, but leveraging Deep Learning for this task is becoming more and more common

- This is an example of a self-supervised task. It is unsupervised itself, as we have no labeled data for the correct embeddings. However the corpus we use for training provides necessary contex, which has some similarities to a supervised learning problem

- Embeddings are often created as by-product of a supervised NLP task, this allows us to guide our embeddings to match our specific objective

Examples of more popular DL embeddings

- BERT (Google, 2018):
- ELMo (Allen Institute for AI, 2018)
- GPT-2 (OpenAI, 2018)

These pre-trained embeddings for the following model can be downloaded and leveraged in other tasks

Leveraging Neural Neutworks in embeddings

- There are multiple ways of word embeddings, but leveraging Deep Learning for this task is becoming more and more common

- This is an example of a self-supervised task. It is unsupervised itself, as we have no labeled data for the correct embeddings. However the corpus we use for training provides necessary contex, which has some similarities to a supervised learning problem

- Embeddings are often created as by-product of a supervised NLP task, this allows us to guide our embeddings to match our specific objective

Examples of more popular DL embeddings

- BERT (Google, 2018):
- ELMo (Allen Institute for AI, 2018)
- GPT-2 (OpenAI, 2018)

These pre-trained embeddings for the following model can be downloaded and leveraged in other tasks

Evaluating word embeddings

Intristic

- Sanity checking embeddings to see if they reflect logical/linguistic relation
- Our Country->Capital example was an example of such evaluation
- Clustering and Visualization can be further leveraged to see if word embeddings are correctly grouped
- Not very scalable if we do not have at least some labeled data

Extrinsic

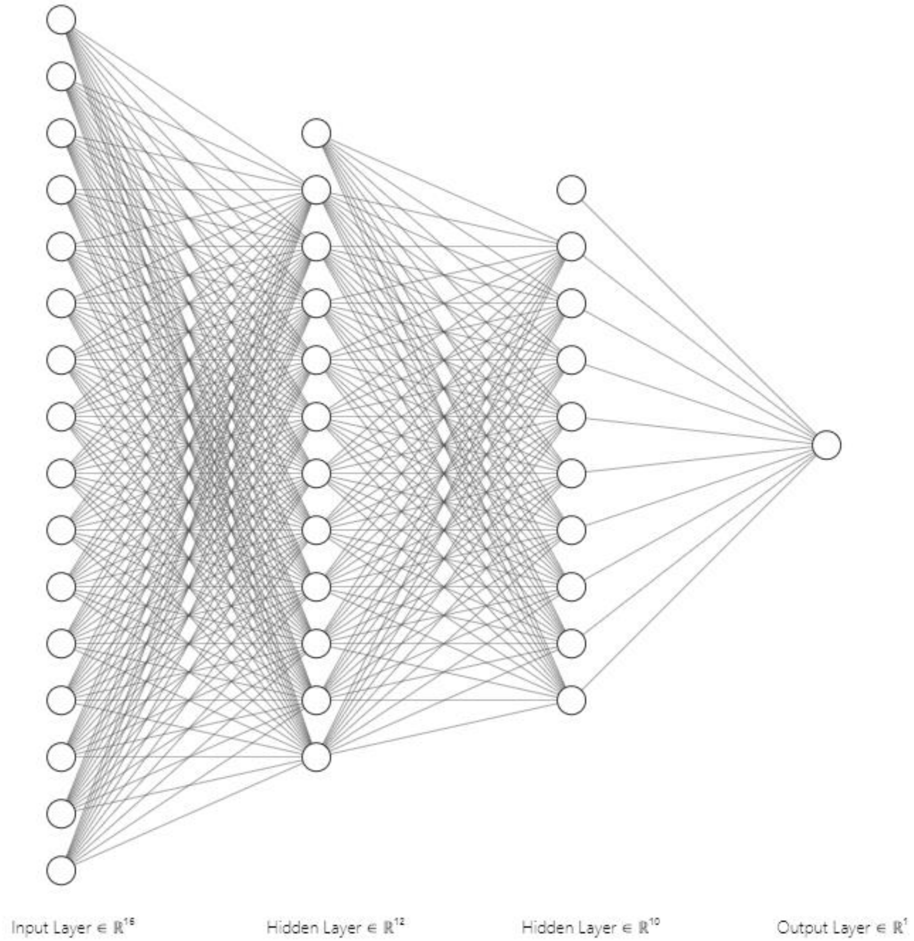
- „Trial by combat" – we test embeddings on a task like Named Entity Recognition or Part of speech tagging and compare against a baseline
- Time consuming than a quick sanity check
- Easier to measure – we only have better embeddings if they beat the baseline
- Another degree of freedom, where the model performing the task itself effects the results

Sequence models

Intro to Deep Neural Networks

- Deep Neural Networks can learn complex patterns and handle language data well
- Another advantage of DNNs is the fact, that their training process creates langiage embeddings as a side-product
- We can extract embeddings (with different dimensionality) by disecting one of hidden layers

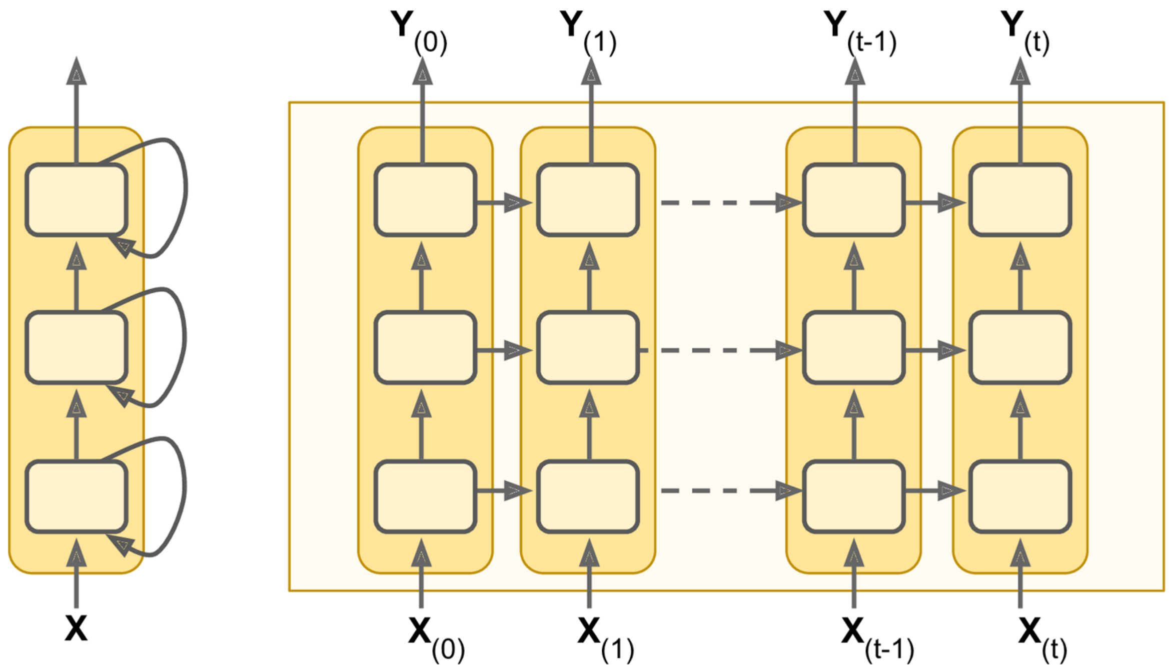
Deep Neural Network diagram


Input Layer $\in \mathbb{R}^{16}$
Hidden Layer $\in \mathbb{R}^{12}$
Hidden Layer $\in \mathbb{R}^{10}$
Output Layer $\in \mathbb{R}^{1}$

Reccurent Neural Networks

- RNNs is a DNN architecture created specially for sequential data such as timeseries or language
- It has the ability to „remember” outputs from previous steps
- More advanced cells like LSTMs and GRUs make the „memory” aspect of Neural Networks even more robust

Recurrent Neural Networks

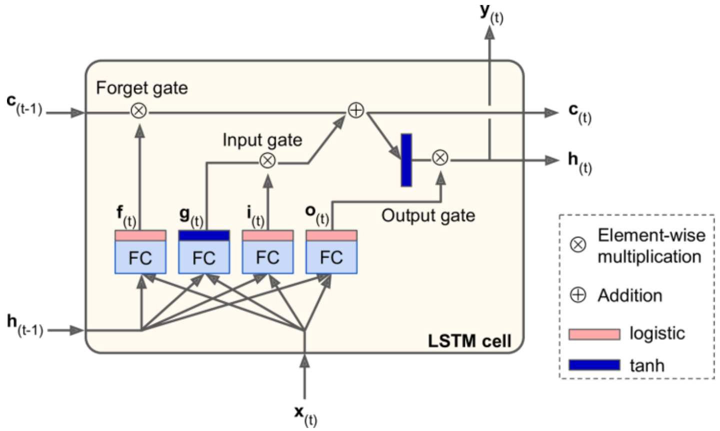


Source: https://www.oreilly.com/

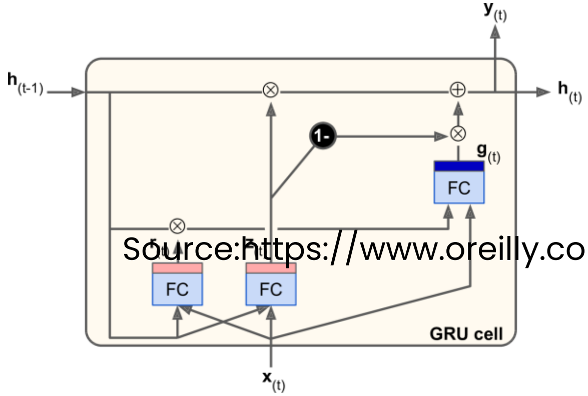
How memory aspect makes RNNs so useful

- N-gram models can become extremely heavy as the possible numer of word combinations quickly explodes
- RNNs are much better at grasping the context of words, which are not directly together
- Furthermore LSTM and GRU cells, which are more complex than simple neuron cells have specific gates, which memorize important information
- Despite their complicated appearance we can use them in place of classic RNN layers without significant changes

Long-short-term-memory cell



Gated Recurrent Unit



Source:https://www.oreilly.com/

Source: https://www.oreilly.com/

Attention & Transformers

Sequence models shortcomings

- Reccurent models rely strongly on sequentiality, where output of previous state is essential for calculating next state
- This makes parallelization impossible and creates issues when working with larger sequence lenghts
- This sequential approach also looses information with the distance. Majority of the information at the begging of the text will be lost towards the end

Key RNN challenges

No paralel computing

Loss of information

Vanishing gradients

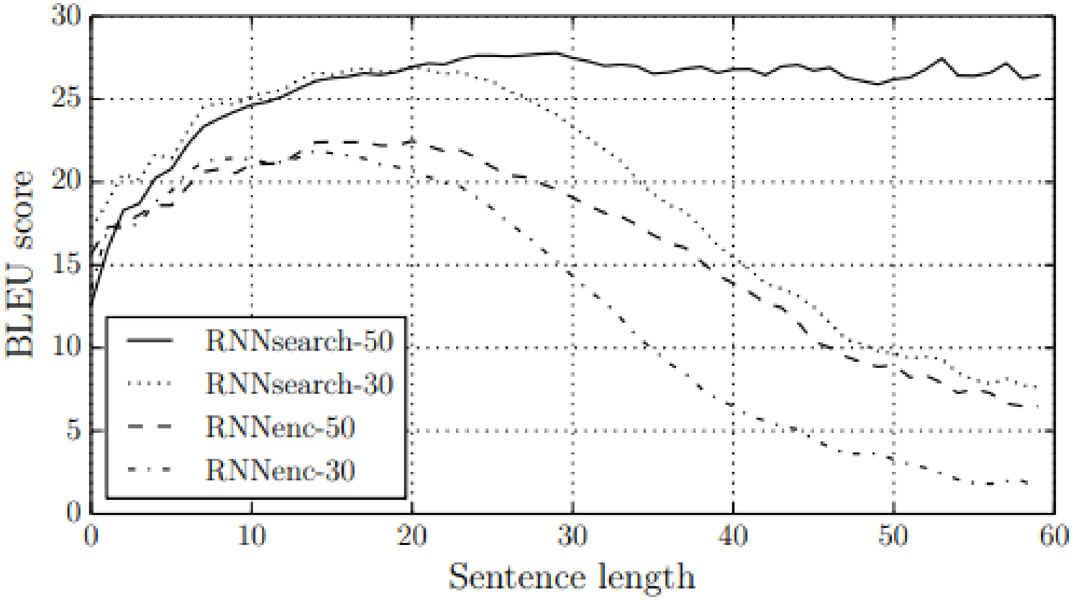
How to implement attention to sequential models

- One way to fix RNNs issue with longer
- sequences is to apply attention mechanism

- Traditionally all previous hidden states are combined into one vector

- Attention mechanism allows to summarize all previous states in a more meaningfull way creating a context vector

Performance on BLEU translation benchmark by sentence lenghth



Source: „Neural Machine Translation by Jointly Learning to Align and Translate" Dzmitry Bahdanau

Transformers

- Transformer models are a type of Neural Network architecture designed to process sequential data first introduced in "Attention Is All You Need" by Vaswani et al. in 2017.
- Despite their initial use in NLP, they are very versatile and widely adapter in other areas of ML
- They are able to process input sequences in paralel
- Their attention mechanism allows to avoid loss of key information, even if is high distance apart within the sequence

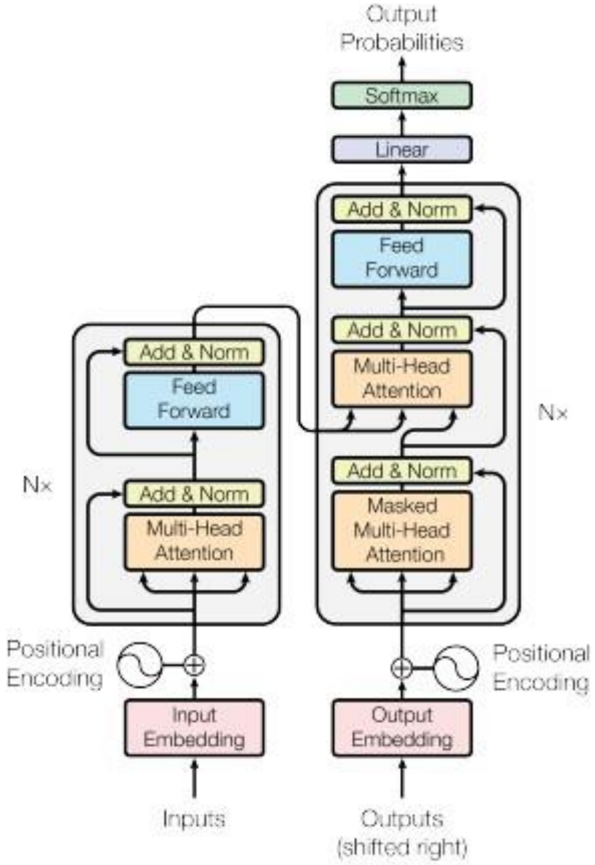


Figure 1: The Transformer - model architecture.

Source „Attention Is All You Need" by Vaswani et al. in 2017.

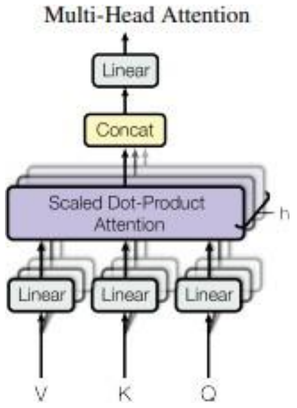
Transformers – high level overview

Encoder:

Processes the input data (e.g. a sentence) and converts it into a set of attention-based representations. These representations capture the context and relationships between different elements in the input.

Each Encoder layer consist of 2 sublayers:

1. Multi-head self-attention mechanism
2. Fully connected feed-forward Network



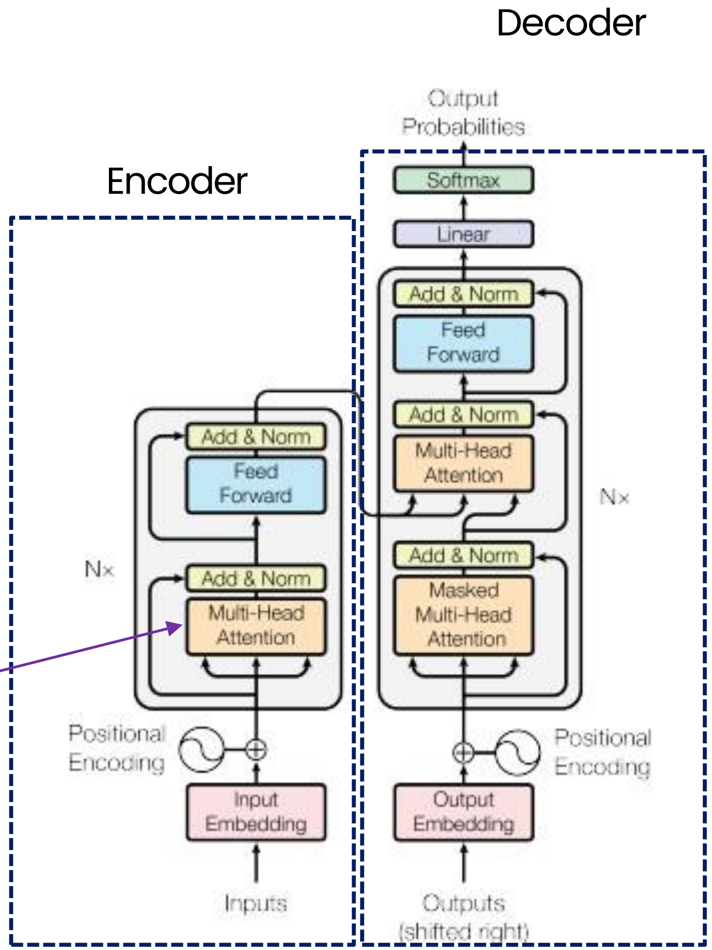


Figure 1: The Transformer - model architecture.

Decoder:

Generates the output data (e.g., the translated sentence in another language) step by step. It uses the representations from the encoder and the previously generated outputs to predict the next element in the sequence.

Each Encoder layer consist of 2 sublayers already present in Decoder + an additional sublayer:

3. Masked Multi-Head attention to prevent positions from attending to subsequent positions

The masking together with output embeddings offset guarantees that the model does not cheat by peeking at future tokens

Source „Attention Is All You Need" by Vaswani et al. in 2017.

Transformers – Self Attention

This mechanism allows the model to weigh the importance of different parts of the input when processing a particular element. For example, in a sentence, the importance or relevance of other words when considering a specific word.

- **Attention Scores**: The model computes scores to determine how much focus to put on other parts of the input for each word in the sequence.

- **Attention Weights**: These scores are then normalized to form a distribution (using functions like softmax), so that they add up to one.

-**Contextual Representation**: Each word's representation is then updated by summing up the representations of all words, weighted by these attention weights. This process ensures that each word’s new representation is a blend of its own and others' based on their relevance.

Transformers – high level overview

Order of introduction

ELMo

- Embeddings from Language Model by University of Washington in 2018
- Uses bi-directional LSTM, which use inputs from boths left and right of the sequence
- Suffered issues with long-term dependencies as it was not a tranformer model

GPT

- Generative Pre-training Transformer by OpenAi in 2018
- Uni-directional transformer model

BERT

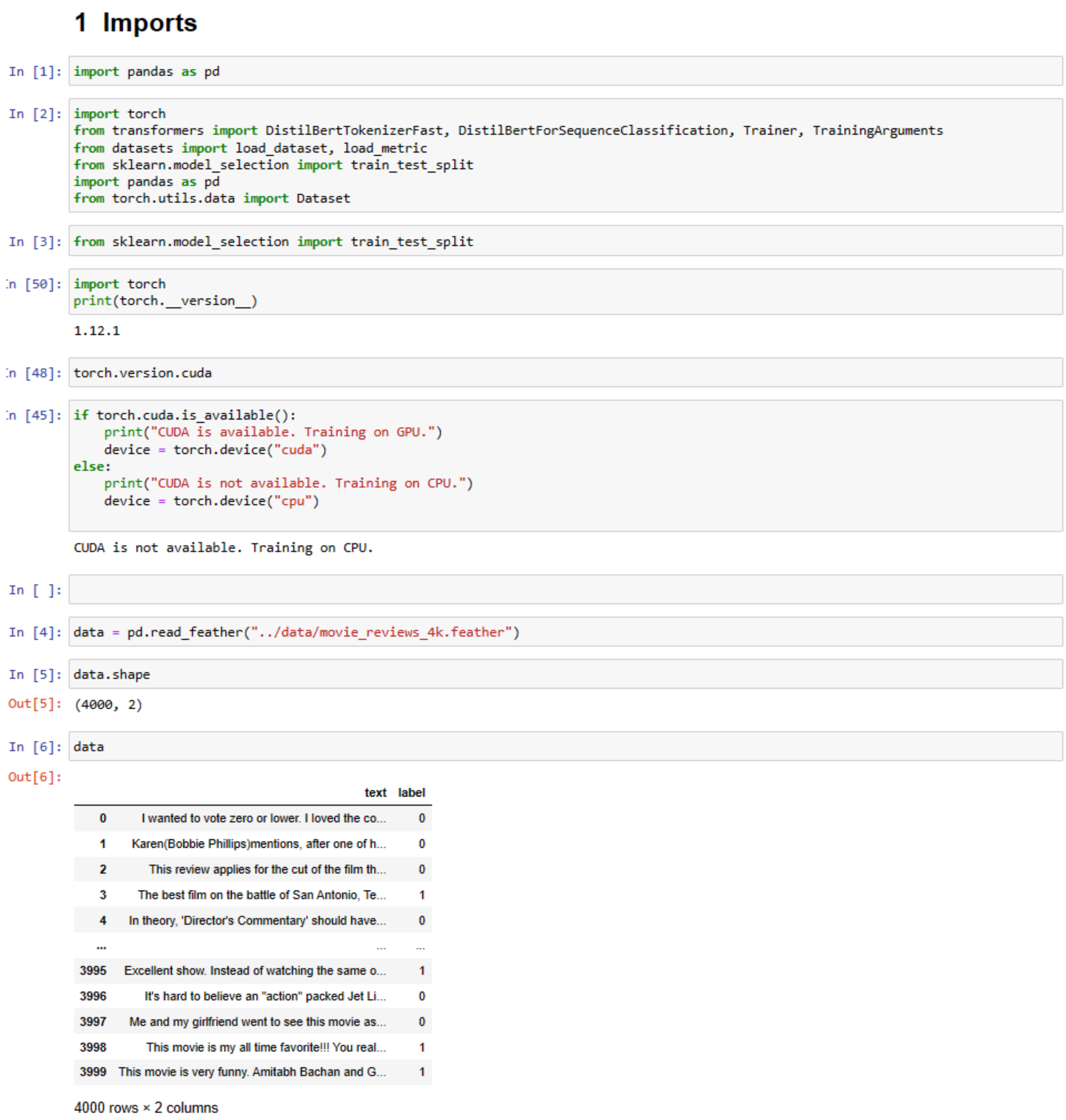
- Bidirectional Encoder Representations from Transformers by Google in 2018
- Bidirectionality implemented into transformers

T5

- Text-To-Text Transfer Transformer developed by Google in 2020
- Leverages transfer-learning and transformer architecture
- Designed to perform wide range of task by fine-tuning on specific datasets

Transformer fine-tuning exercise

- Go to Sentiment_classification_with_BERT.ipynb where we will leverage Transfer Learning with BERT model in our own tweet sentiment classification



1 Imports

```
In [1]: import pandas as pd

In [2]: import torch
        from transformers import DistilBertTokenizerFast, DistilBertForSequenceClassification, Trainer, TrainingArguments
        from datasets import load_dataset, load_metric
        from sklearn.model_selection import train_test_split
        import pandas as pd
        from torch.utils.data import Dataset

In [3]: from sklearn.model_selection import train_test_split

In [50]: import torch
         print(torch.__version__)

         1.12.1

In [48]: torch.version.cuda

In [45]: if torch.cuda.is_available():
             print("CUDA is available. Training on GPU.")
             device = torch.device("cuda")
         else:
             print("CUDA is not available. Training on CPU.")
             device = torch.device("cpu")

         CUDA is not available. Training on CPU.

In [ ]:

In [4]: data = pd.read_feather("../data/movie_reviews_4k.feather")

In [5]: data.shape

Out[5]: (4000, 2)

In [6]: data

Out[6]:
```

	text	label
0	I wanted to vote zero or lower. I loved the co...	0
1	Karen(Bobbie Phillips)mentions, after one of h...	0
2	This review applies for the cut of the film th...	0
3	The best film on the battle of San Antonio, Te...	1
4	In theory, 'Director's Commentary' should have...	0
...	...	...
3995	Excellent show. Instead of watching the same o...	1
3996	It's hard to believe an "action" packed Jet Li...	0
3997	Me and my girlfriend went to see this movie as...	0
3998	This movie is my all time favorite!!! You real...	1
3999	This movie is very funny. Amitabh Bachan and G...	1

4000 rows × 2 columns