

# Chap 2: Text Representation

By Waad ALMASRI

# Introduction

# Definition

Text representation: the conversion of raw text to a suitable numerical form

*"A good text representation must:*

- *Break the sentence into lexical units such as lexemes, words and phrases*
- *Derive the meaning for each of the lexical units*
- *Understand the syntactic (grammatical) structure of the sentence*
- *Understand the context in which the sentence appears"*

# Text representation categories

Text representation approaches are classified into 4 categories:

- Basic vectorization approaches
- Distributed representations
- Universal language representation
- Handcrafted features

# Vector space models VSM

Vector Space Model (VSM) is a mathematical model that represents text units as vectors.

A common similarity measure between 2 text units represented by vectors A & B is the cosine similarity:

$$\cos(\theta) = \frac{A \cdot B}{\|A\|_2 \|B\|_2} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

# Basic vectorization approaches

# A corpus example

Document 1 (D1)	The dog plays with the boy.
Document 2 (D2)	The dog eats meat.
Document 3 (D3)	The boy eats food.
Document 4 (D4)	The boy plays with the dog.

This corpus is composed of 4 documents.

The vocabulary  $V$  of this corpus cleaned and pre-processed (lowercasing, ignoring punctuations, stop words removal) consists of 6 words: [dog, plays, boy, eats, meat, food]; i.e.  $|V| = 6$

In other terms, every document can be represented with a vector of size 6.

# One hot encoding

- Every word  $w$  in the vocabulary corpus  $V$  has a unique ID between 1 and  $|V|$ .
- Every word is thus represented by a binary vector of size  $|V|$  where 0/1 is equivalent to the absence/presence of the word

## Example

Word	ID	One Hot Encoding
dog	1	[1 0 0 0 0 0]
plays	2	[0 1 0 0 0 0]
boy	3	[0 0 1 0 0 0]
eats	4	[0 0 0 1 0 0]
meat	5	[0 0 0 0 1 0]
food	6	[0 0 0 0 0 1]

Document	Document represented as a one hot encoding
dog plays boy	[[1 0 0 0 0 0], [0 1 0 0 0 0], [0 0 1 0 0 0]]
dog eats meat	[[1 0 0 0 0 0], [0 0 0 1 0 0], [0 0 0 0 1 0]]
boy eats food	[[0 0 1 0 0 0], [0 0 0 1 0 0], [0 0 0 0 0 1]]
boy plays dog	[[0 0 1 0 0 0], [0 1 0 0 0 0], [1 0 0 0 0 0]]



# One hot encoding (2)

## Pros

- Intuitive to understand
- Easy to implement

## Cons

- The size of the vector is proportional to  $|V|$ , which in reality can be very large  $\Rightarrow$  sparse representations (mostly zeros) that are computationally inefficient to
  - Store
  - Compute with
  - Learn from (sparsity leads to overfitting)
- Sentences have different length and ML models need the feature vectors to be the same length
- It does not capture the semantics in the sentences (meaning of words, similarities, etc.)
- Words that are not included in the training corpus cannot be represented by the model  $\Rightarrow$  OOV Out Of Vocabulary problem

# Bag of words (BoW)

- Intuition behind BoW: every class in the dataset is characterized by a set of words. Two sentences having nearly the same words belong to the same class (bag).
- Every word  $w$  in the vocabulary corpus  $V$  has a unique ID between 1 and  $|V|$ .
- Every word is thus represented by a vector of size  $|V|$  where 0/f is equivalent to the absence/frequency of the word in the document

Example

Word	ID
dog	1
plays	2
boy	3
eats	4
meat	5
food	6

Document	Document represented via BoW
dog plays boy	[111000]
dog eats meat	[100110]
boy eats food	[001101]
boy plays dog	[111000]
dog and dog play	[210000]

# Bag of words (BoW) (2)

## Pros

- Intuitive to understand
- Easy to implement
- It captures the semantic similarities in the sentences (documents with the same words have the same vectors)
- Sentences have a fixed-length encoding no matter what their lengths (frequency came in hand)

## Cons

- The size of the vector is proportional to  $|V| \Rightarrow$  sparsity; one solution is to limit the  $V$  to  $n$  number of the most frequent words
- It does not capture the semantics in the sentences (meaning of words, similarities in synonyms, etc.)
- Words that are not included in the training corpus cannot be represented by the model  $\Rightarrow$  OOV Out of Vocabulary problem
- Word order is lost, thus the naming “bag”

# Bag of N-grams (BoN)

- Text is broken into chunks of N contiguous words; each chunk is called an N-gram
- The vocabulary corpus V becomes a collection of N-grams, each having a unique ID between 1 and |V|.
- Every document is thus represented by a vector of size |N-grams| or |V|+|N-grams| where 0/f is equivalent to the absence/frequency of the N-gram in the document

Example: here we choose the bigrams => our vocabulary consists of an ensemble of 2 words => |V| becomes 8 not 6.

bigram	ID
dog plays	1
plays boy	2
dog eats	3
eats meat	4

bigram	ID
boy eats	5
eats food	6
boy plays	7
plays dog	8

Document	Document represented via BoN
dog plays boy	[11000000]
dog eats meat	[00110000]
boy eats food	[00001100]
boy plays dog	[00000011]
dog and dog play	[00000000]

# Bag of N-grams (BoN) (2)

## Pros

- Intuitive to understand
- Easy to implement
- It captures some contextual and **word-order** information in the form of n-gram
- It captures the semantic similarities in the sentences (documents with the same n-grams have the same vectors)
- Sentences have a fixed-length encoding no matter what their lengths

## Cons

- As N increases, the vector size increases rapidly => sparsity
- Words that are not included in the training corpus cannot be represented by the model => OOV Out of Vocabulary problem

# Term Frequency - Inverse Document Frequency (TF-IDF)

Intuition behind TF-IDF: if a word  $w$  occurs frequently in document  $d_i$  but not in document  $d_j$ , then its importance should increase proportionally to its frequency in  $d_i$  and inversely proportionally to its frequency in  $d_j$ .

TF-IDF score = TF \* IDF s.t. TF and IDF defined as:

$$TF(t, d) = \frac{\text{Nbr of occurrences of term } t \text{ in doc } d}{\text{Total Nbr of terms in the doc } d}$$

$$IDF(t, d) = \log_e \left( \frac{\text{Total Nbr of docs in the corpus}}{\text{Nbr of docs with the term } t \text{ in them}} \right)$$

# Term Frequency - Inverse Document Frequency (TF-IDF) (2)

Word	TF score	IDF score	TF-IDF score
dog	3/12	$\log_2(4/3)$	0.1037
plays	2/12	$\log_2(4/2)$	0.167
boy	3/12	$\log_2(4/3)$	0.1037
eats	2/12	$\log_2(4/2)$	0.167
meat	1/12	$\log_2(4/1)$	0.167
food	1/12	$\log_2(4/1)$	0.167

Document	TF-IDF vector [dog, plays, boy, eats, meat, food]
dog plays boy.	[0.1037, 0.167, 0.1037, 0, 0, 0]
dog eats meat.	[0.1037, 0, 0, 0.167, 0.167, 0]
boy eats food.	[0, 0, 0.1037, 0.167, 0, 0.167]
boy plays dog.	[0.1037, 0.167, 0.1037, 0, 0, 0]



# Term Frequency - Inverse Document Frequency (TF-IDF) (3)

## Pros

- It captures some contextual
- It captures the semantic similarities in the sentences (computing a similarity measure like euclidean or cosine similarity between TF-IDF vectors is feasible)
- Sentences have a fixed-length encoding no matter what their lengths

## Cons

- Curse of high dimensionality and sparsity => it holds back the learning capability and is computationally inefficient
- They are discrete representations, relationship between words is ignored (order does not matter)
- Words that are not included in the training corpus cannot be represented by the model => OOV Out of Vocabulary problem



# Distributed Representations

# Key terms

- **Distributional similarity:** a.k.a. Connotation, the meaning of the word is defined by the context in which it appears.
- **Denotation:** the literal meaning of the word
- **Distributional hypothesis:** words that occur in similar contexts have similar meanings. In the VSM, two words occurring in similar contexts must have close representation vectors.
- **Distributional representation:** representation schemes obtained based on distributional hypotheses. They are  $|V|$ -dimensional vectors extracted from a co-occurrence matrix. Examples: One hot encoding, BoW, BoN, TF-IDF.
- **Distributed representation:** a more compact and dense version of the distributional representation, i.e. lower dimension and less sparsity (hardly any zeros)
- **Embedding:** it is the task of mapping the vector space of the distributional representation to the vector space of the distributed representation (from high-dimensionality high-sparsity to low-dimensionality low-sparsity)
- **Vector semantics:** a set of NLP methods to learn the word distributional representations from a large corpus

# Word embeddings

## Embeddings

- Learn rich semantic relationships
- Are low-dimensional => computationally more efficient
- And dense (no zero values) => robust learning with ML

One of the most famous and the first embeddings is Word2vec, a two-layer neural network for word embeddings by Mikolov. In his work, Mikolov proved that Word2vec captures word analogy relationships; i.e. King-Man + Woman = Queen

# Embeddings

Given a text corpus, we need to assign to every word a vector that best captures its meaning.

To infer the meaning of every word, Word2vec uses distributional similarity and distributional hypothesis; two different words in the same context must be related and thus should have vectors closer in the embeddings space

A randomly initial vector  $v_w$  is assigned to every word  $w$ . Then, the Word2vec model updates the values  $v_w$  by predicting this  $v_w$  given the vectors of the words appearing in the context of  $w$ .

# Pre-trained word embeddings

They are embeddings trained by a third party on a large corpus like wikipedia, articles, ... the entire web and shared online.

They are a large dictionary with keys as the words and values as the vectors of these words.

Top most common pre\*trained embeddings are:

- word2vec : Google
- GloVe: Stanford
- Fasttext: Meta (Facebook)

They are available in various dimensions  $d=25,50,100,200,300,600$

# Pre-trained word embeddings

```
from gensim.models import Word2Vec, KeyedVectors

pretrained_path = "NLPBookTut/GoogleNews-vectors-negative300.bin"

print("loading Word2vec...")

w2v_model = KeyedVectors.load_word2vec_format(pretrained_path, binary=True )

print("Nbr of words in word2vec vocabulary:", len(w2v_model .vocab))

print("Beautiful is similar to:", w2v_model.most_similar["beautiful"])
```

Now let us search for a word not in word2vec model like practicalnlp

# Training our own embeddings

The original word2vec approach proposes two variants:

- Continuous Bag Of Words (CBOW)
- SkipGram

# CBOW

CBOW's objective is to build a language model that predicts a center word given the context words.

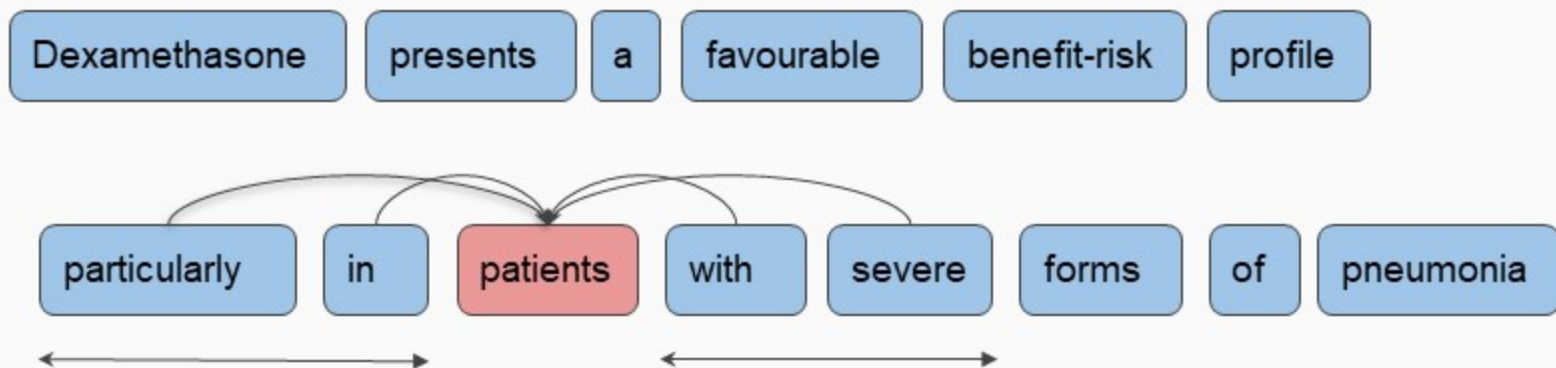
A language model = a statistical model that learns the probability distribution over sequence of words, i.e. given a sentence of  $m$  words, it assigns the probability  $\Pr(w_1, w_2, \dots, w_m)$  to the whole sentence. It gives higher probabilities to semantically and syntactically correct sentences and lower ones to semantically and/or syntactically incorrect ones.

Example:  $s_1$  = I live in Paris since 2017.  $s_2$  = I since live 2017 in Paris.  
 $\Pr(s_1) = 1.0$  and  $\Pr(s_2)=0.0$



## CBOW (2)

Toy corpus: Dexamethasone presents a favourable benefit-risk profile, particularly in patients with severe forms of pneumonia.

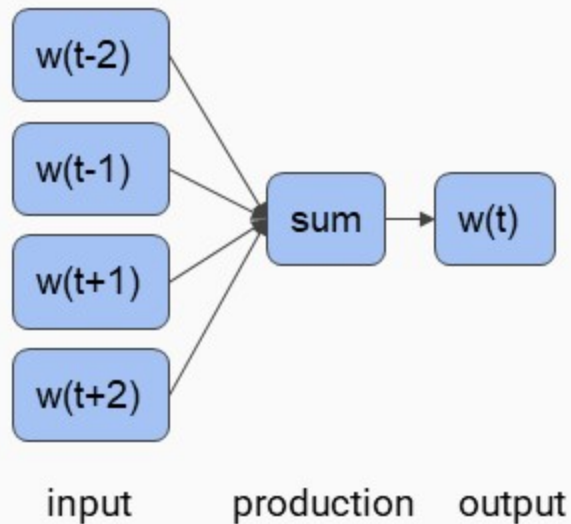


# CBOW (3)

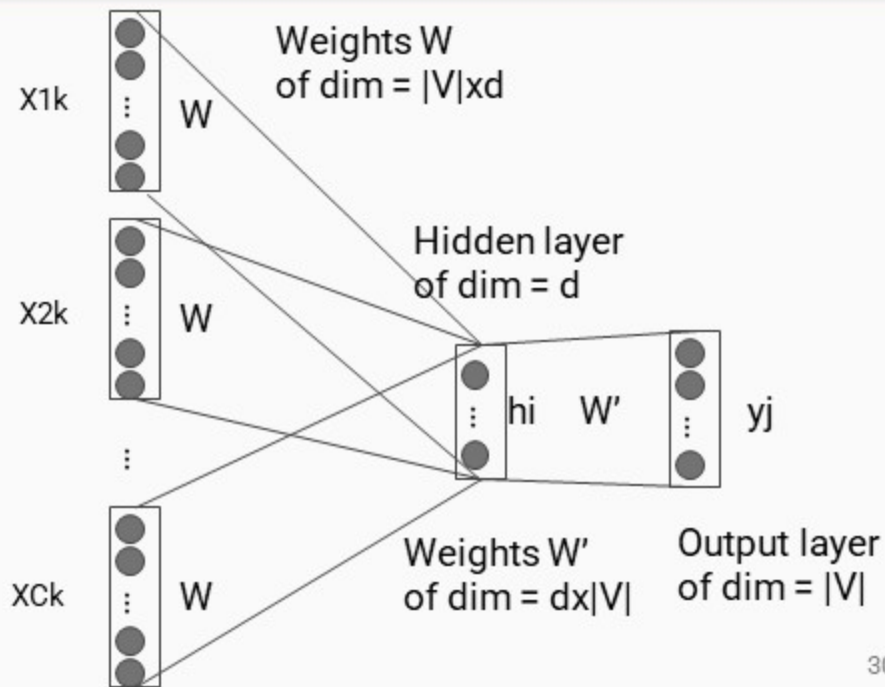
Sliding window =  $2k + 1$  ; here  $k=2 \Rightarrow$  1 center word and 2 context words from each side

Source Text	Training samples (context, target)
Dexamethasone presents a favourable benefit-risk profile, particularly in patients with severe forms of pneumonia.	((presents, a), Dexamethasone )
Dexamethasone presents a favourable benefit-risk profile, particularly in patients with severe forms of pneumonia.	((Dexamethasone, a, favourable ), presents )
Dexamethasone presents a favourable benefit-risk profile, particularly in patients with severe forms of pneumonia.	((Dexamethasone, presents, favourable, benefit-risk), a)
Dexamethasone presents a favourable benefit-risk profile, particularly in patients with severe forms of pneumonia.	(( presents, a, benefit-risk, profile), favourable)
...	

# CBOW Model



Input layer  
of dim =  $C \times |V|$



# SkipGram

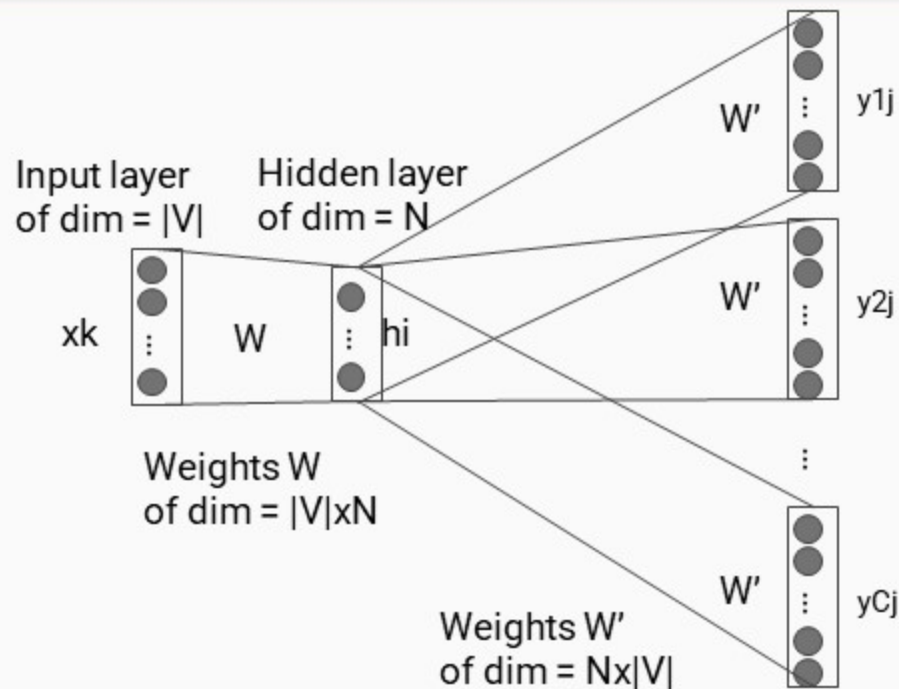
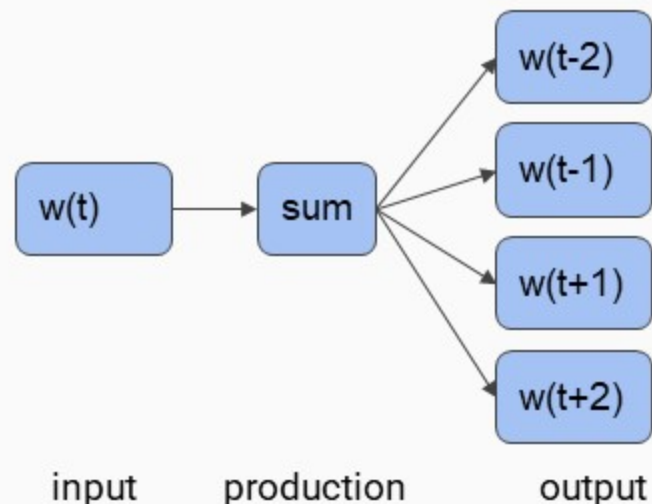
Sliding window =  $2k + 1$  ; here  
 $k=2 \Rightarrow$  1 center word and 2  
context words from each side

SkipGram's objective is to build a language model that predicts the context words given the center word.

Source Text	Training samples (context, target)
Dexamethasone presents a favourable benefit-risk profile, particularly in patients with severe forms of pneumonia.	(Dexamethasone, presents) (Dexamethasone, a)
Dexamethasone presents a favourable benefit-risk profile, particularly in patients with severe forms of pneumonia.	(presents, Dexamethasone) - (presents, a) - (presents, favourable)
Dexamethasone presents a favourable benefit-risk profile, particularly in patients with severe forms of pneumonia.	(a, Dexamethasone) - (a, presents) - (a, favourable) - (a, benefit-risk)
...	

# Continuous SkipGram Model

Output layer  
of dim =  $C \times |V|$



# Training Word2Vec - Hyperparameters

- There are several implementations for CBOW & SkipGram. The top common used implementation is gensim.
- Training a word embedding is not only about the model but also its hyperparameter: window size, dimension of the embedding vector, learning rate, number of epochs, optimizer, etc.
  - Usually, the word vector's dimension is chosen between 50 and 500.
  - For the context window, it represents the length of the context we are looking for to learn the vector's representation.



# The OOV problem

- When the training corpus is large enough that OOV words are rarely existant
- When existing, one simple approach is to simply exclude them
  - However, when OOV words are more than 20% of the training words, the performance of the NLP model can be easily deteriorated.
- Another approach consists of creating vectors that are initialized randomly and assign them to OOV words
- Advanced approach: modify the training process by integrating characters and subword level linguistic components (prefixes, suffixes, word endings, etc.)

## The OOV problem (2)

Advanced approach: modify the training set by using subword information like morphological properties (prefixes, suffixes, word endings), by using character-based representations.

FastText (from FAIR: Facebook AI Research) is a character-based word embeddings model.

- A word is represented by its character n-grams
- fastText learns the embedding of words' n-grams
- A word embedding = aggregation of the character n-grams embeddings
- Thus, when an OOV appears, its embedding is computed using the character n-grams forming this word



# Text embeddings

In NLP applications, we don't deal with atomic words but sentences, paragraphs, etc. Thus, how do we represent these large texts ?

- Simple approach: aggregate the embeddings of the words forming these texts (sum, average, etc.)
  - Disadvantage: the text as a whole is not captured, context nor ordering is considered
  - Advantage: they practically work well and this is demonstrated in CBOW (summing the embeddings of context words is used to predict the center word.)

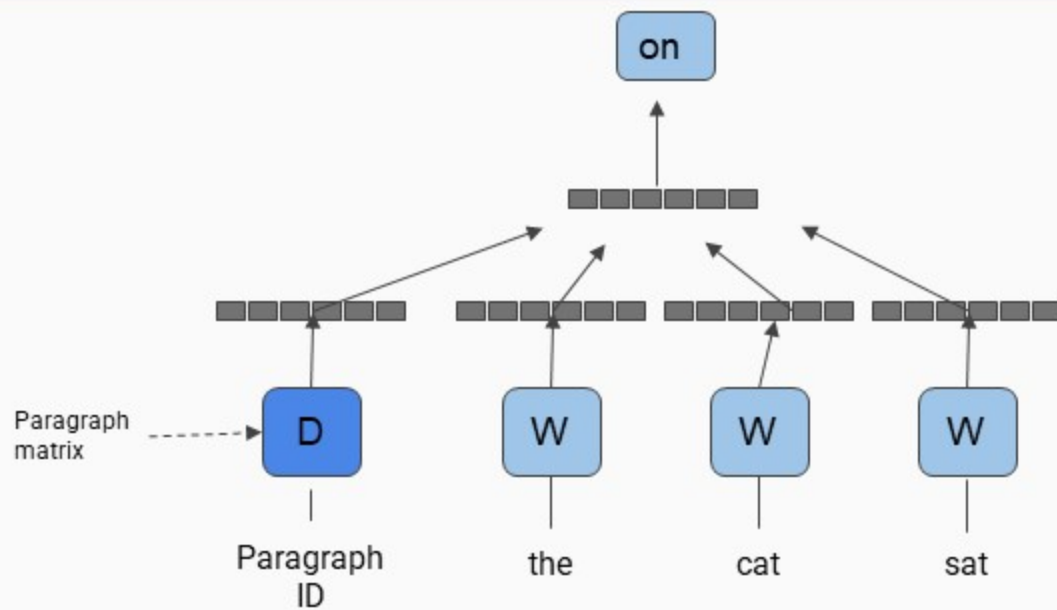
# Text embeddings

In NLP applications, we don't deal with atomic words but sentences, paragraphs, etc. Thus, how do we represent these large texts ?

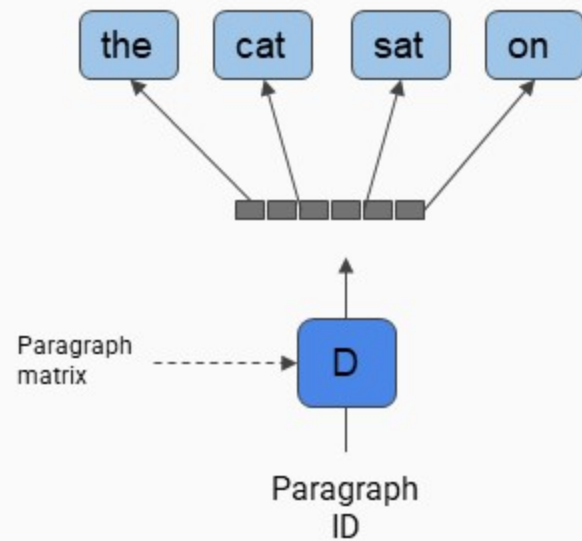
- Simple approach: aggregate the embeddings of the words forming these texts (sum, average, etc.)
- Doc2Vec: an arbitrary-length text representation that learns taking the context words into account

# Text embeddings

Doc2vec is based on the paragraph vectors framework.



DM: Distributed Memory



DBOW: Distributed Bag Of Words

# Universal text representation

Should every word get one fixed representation?

# Universal text representation

- Words can mean different things in different contexts. With the representations explained previously, this contextual meaning is not captured.
- In 2018, the idea of “contextual word representations” came to address this problem. “The language modeling” was introduced; its aim is to predict the next likely word in a sequence of words.
  - N-gram frequencies to estimate the probability of the next word given a history of words
  - neural language models: recurrent neural networks (RNN), transformers (ELMo, BERT)



# Tips & Tricks

- All text representations are inherently biased based on training data. Such biases can have serious implications on the performance of NLP models and systems that rely on these representations.
  - Understanding biases that may be present in learned embeddings and developing methods for addressing them is very important [1].
- Unlike vectorization approaches, pre-trained embeddings are generally large-sized files (gigabytes) => problems in deployment scenarios
  - Use in-memory databases like Redis with a cache on top to address scaling and latency issues
- Encoding specific aspect of texts and relationships between sentences and other domain or application-specific needs that may not be captured by the embeddings is essential
  - Sarcasm detection
- Do not be carried away by new big NLP models, choose what fits best the application in hand and consider practical business metrics like ROI, infrastructure constraints, etc. before deploying in production

# Visualizing embeddings

# Dimensionality Reduction

Dimensionality reduction can be achieved in the following ways:

- Feature elimination: dropping features → information loss 
- Feature selection: ranking features via statistical tests → the importance scores vary between tests 
- Feature extraction: creating new independent features by linearly or non-linearly combining the features in the data → dimensionality reduction techniques



# Principal Component Analysis (PCA)

- It is a linear feature extraction technique
- It performs a linear mapping of the data to a lower-dimensional space to maximize the variance of the data in the low-dimensional representation
- It combines input features in a specific way that least important features are dropped.
- The new features (components) are all independent of one another

# t-distributed Stochastic Neighbor Embedding (t-SNE)

t-distributed Stochastic Neighbor Embedding or t-SNE:

It is a non-linear technique used for visualizing high-dimensional data like embeddings by reducing them to two or three-dimensional data, all while maintaining the same data distributions in original high-dimensional input space and low-dimensional output space.

In simpler terms, t-Distributed stochastic neighbor embedding (t-SNE) minimizes the divergence between two distributions: a distribution that measures pairwise similarities of the input objects and a distribution that measures pairwise similarities of the corresponding low-dimensional points in the embedding.

# t-SNE



Here, we show feature vectors obtained from MNIST digits dataset.

The digit-images are passed through a CNN, which outputs vectors.

The latter are plot via t-SNE.

t-SNE shows that these vectors are useful because vectors of the same class tend to cluster together

# References

1. Tolga et al. Man is to computer programmer as woman is to homemaker? Debiasing word embeddings.
2. <https://www.datacamp.com/community/tutorials/introduction-t-sne>
3. Vajjala, Sowmya, et al. Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems. O'Reilly Media, 2020.