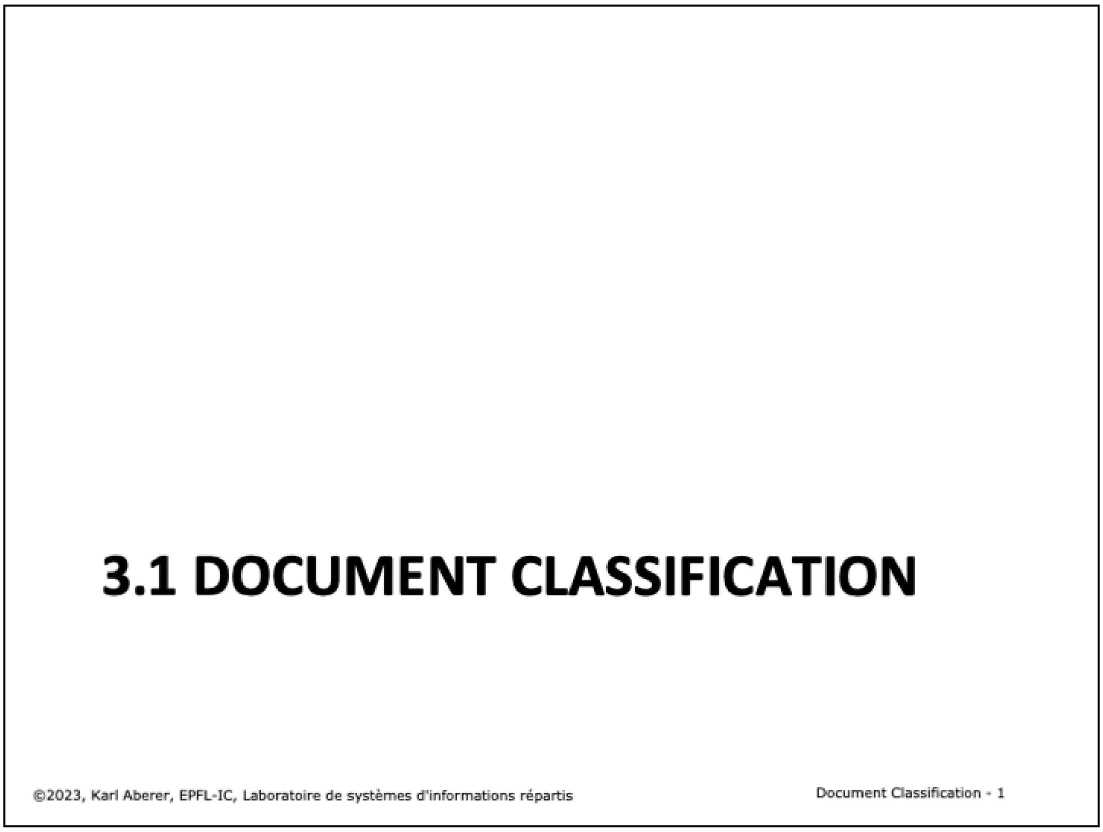
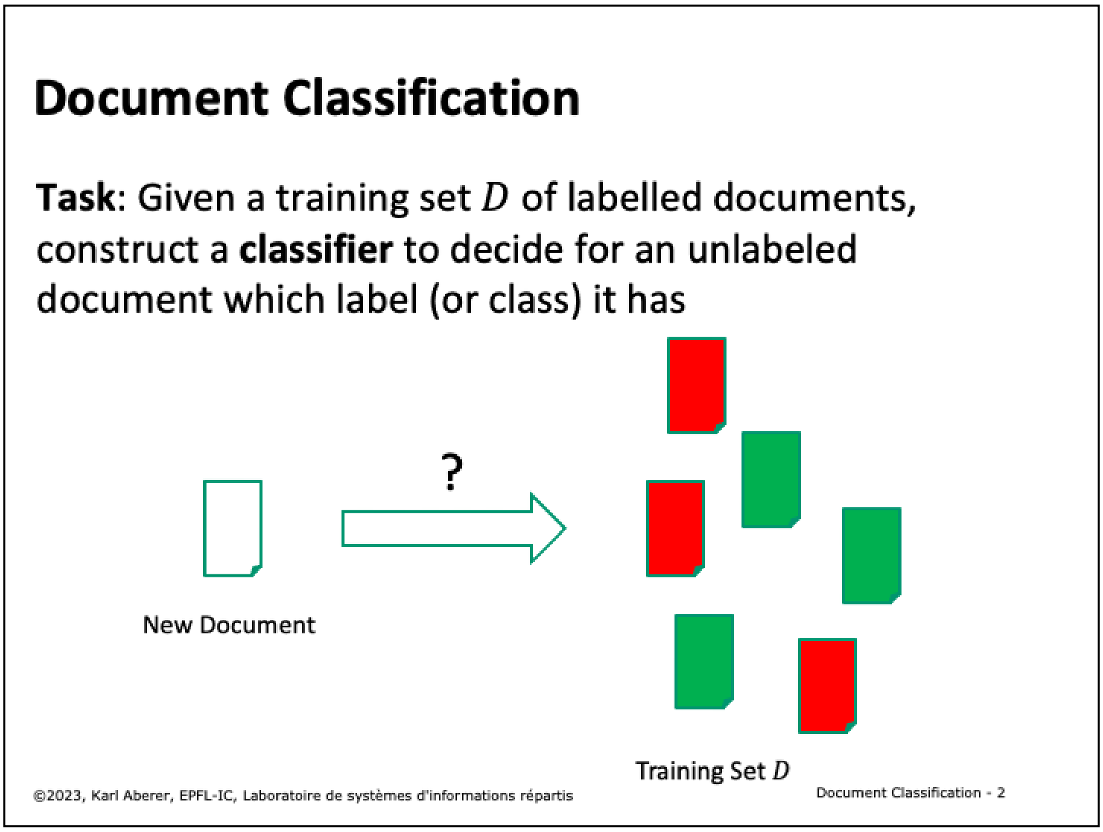


3.1 DOCUMENT CLASSIFICATION



Document Classification

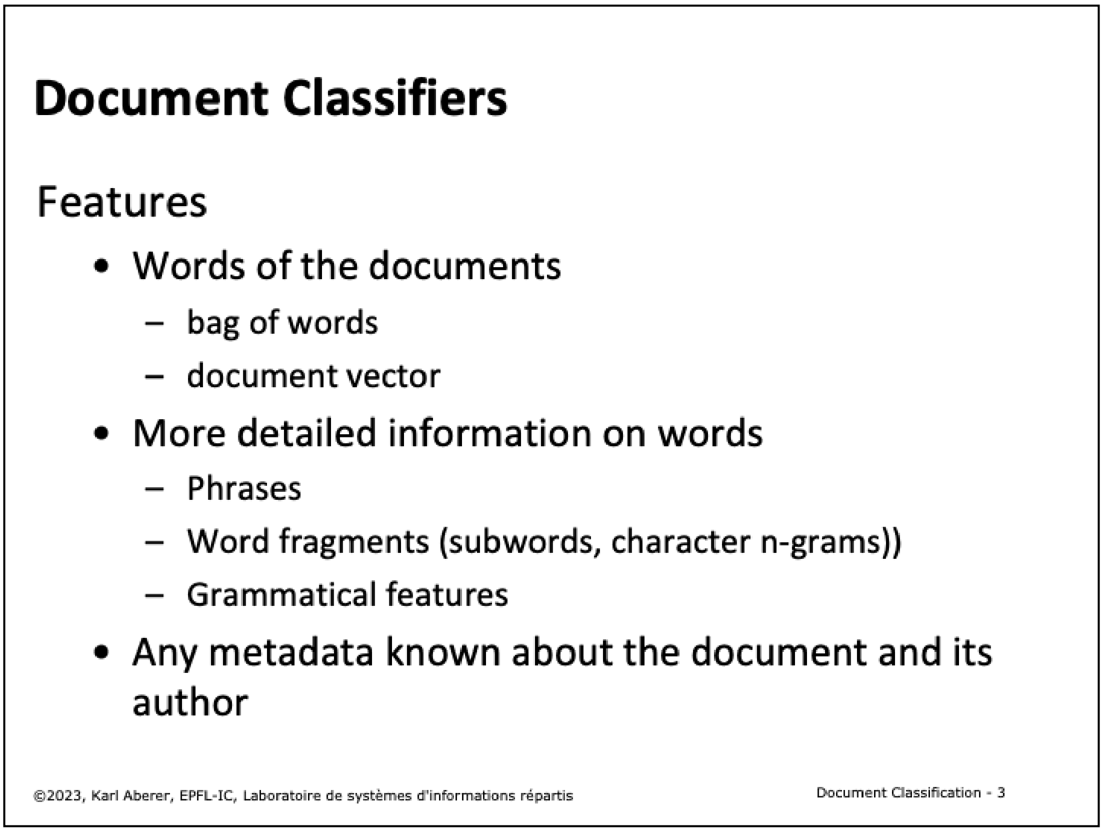
Task: Given a training set D of labelled documents, construct a **classifier** to decide for an unlabeled document which label (or class) it has

?

New Document

Training Set D

Document classification is the task to assign a class label to documents from an existing finite set of available labels. It is a task of huge practical importance in the management od documents. Information retrieval can be understood as one example of document classification, where the task is to distinguish relevant from non-relevant documents. But there are many other use cases, such as spam filtering, sentiment analysis, topic detection.



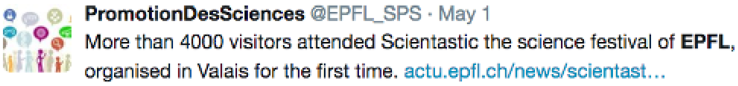
Document Classifiers

Features

- Words of the documents
  - bag of words
  - document vector
- More detailed information on words
  - Phrases
  - Word fragments (subwords, character n-grams))
  - Grammatical features
- Any metadata known about the document and its author

As for any classification problem, a central question is the choice of features used for classification. Like in information retrieval, the word features, which we called the full text, is an obvious choice. We can identify nevertheless many other features that can be exploited. Related to the text more detailed information about the words, including their grammatical features, can be used. Also, metadata on the documents, like the authorship or the history of usage during editing and reading can be considered.

Example



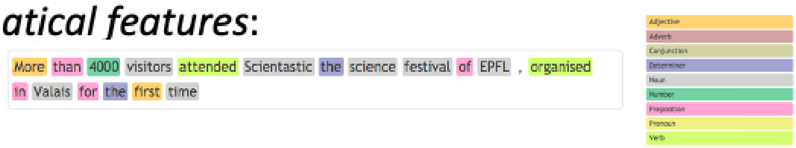
PromotionDesSciences @EPFL_SPS · May 1
More than 4000 visitors attended Scientastic the science festival of **EPFL**, organised in Valais for the first time. actu.epfl.ch/news/scientast…

Bag of words: {more, than, visitors, attend, …}

Phrases: {“science festival”, “first time”}

Word fragments: {mor, ore, tha, han, vis, isi, ..}

Grammatical features:

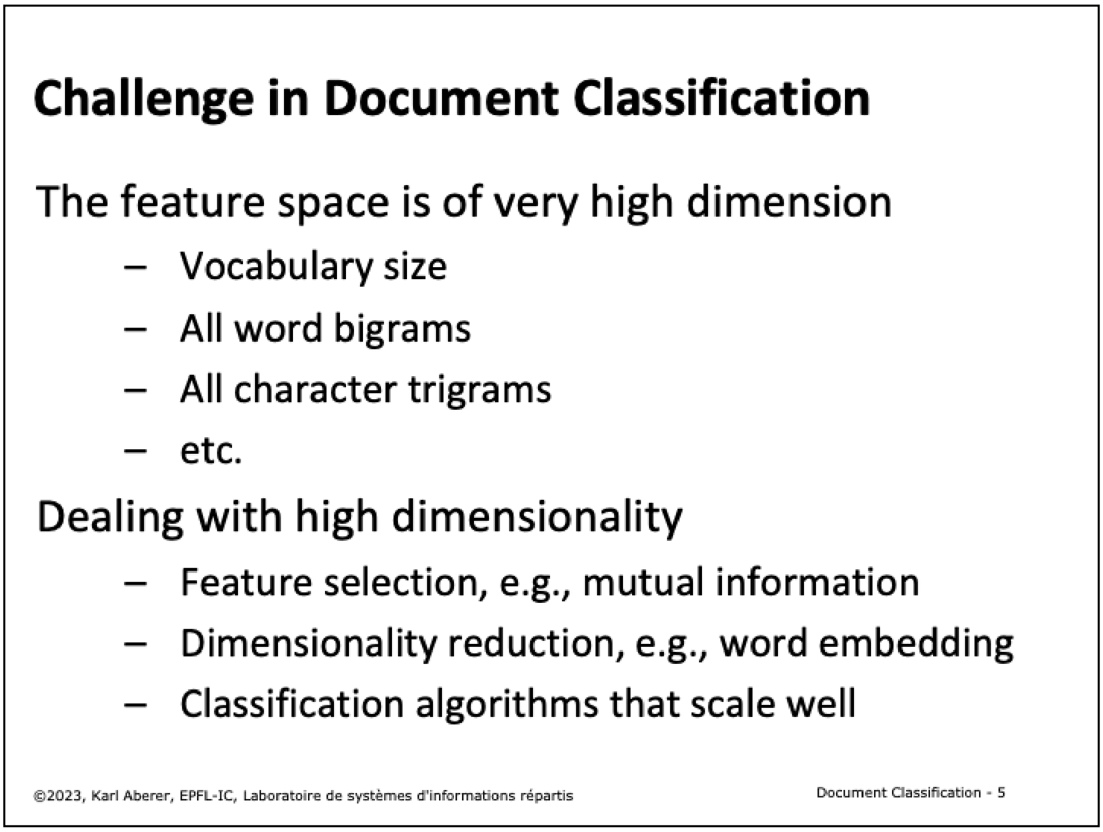


More than 4000 visitors attended Scientastic the science festival of EPFL , organised in Valais for the first time

http://parts-of-speech.info/

Author: @EPFL_SPS

Here we give a few examples of features that could be extracted from a simple tweet.



Challenge in Document Classification

The feature space is of very high dimension

- Vocabulary size
- All word bigrams
- All character trigrams
- etc.

Dealing with high dimensionality

- Feature selection, e.g., mutual information
- Dimensionality reduction, e.g., word embedding
- Classification algorithms that scale well

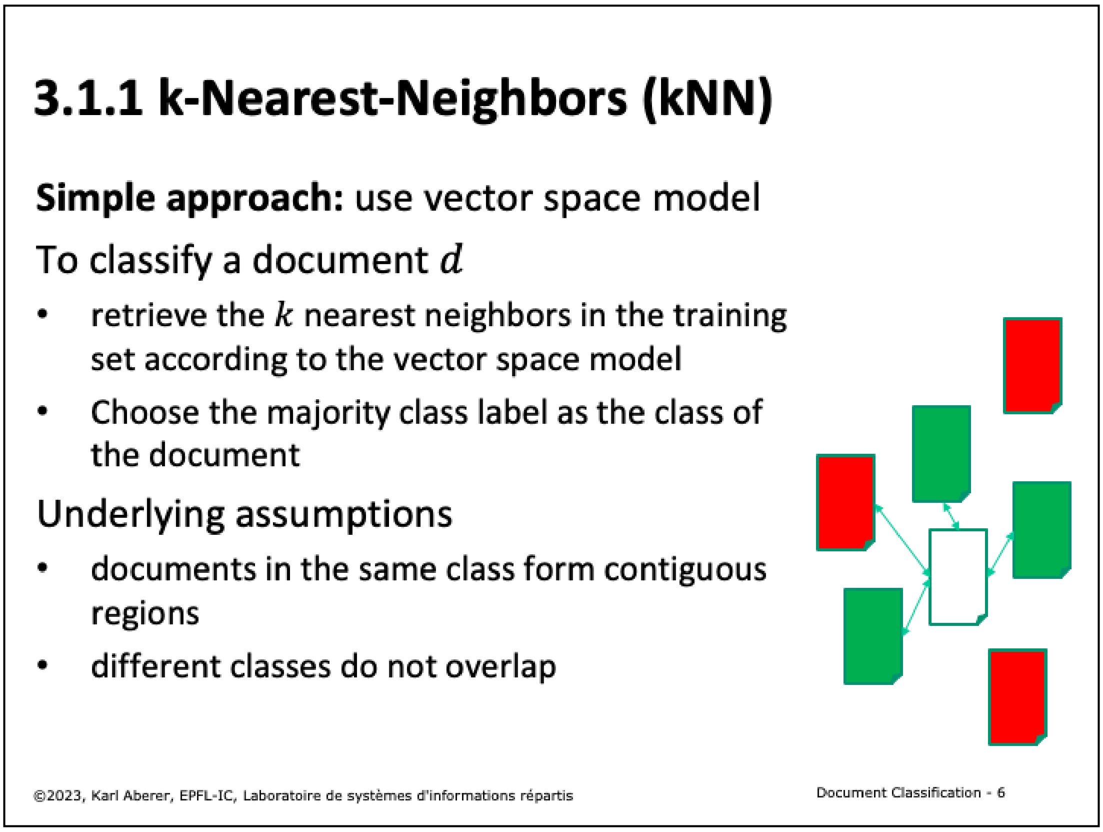
©2023, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Document Classification - 5

Document classification is a special case of classification, a problem widely studied in machine learning. What makes the problem specific, is the very large feature space that can be generated from documents, e.g., large vocabulary sizes.

In order to address the issue of large feature spaces, in machine learning different approaches are pursued.

- Feature selection is used to identify among the available features those that have the largest distinctive power on deciding the class label. Many approaches, such as mutual information, are available from standard machine learning.
- Dimensionality reduction is an approach that we have already introduced in the context of information retrieval, both with LSI and word embeddings.
- Finally, having large number of features, suggest to also prefer classification algorithms that scale well with large numbers of features.



3.1.1 k-Nearest-Neighbors (kNN)

Simple approach: use vector space model

To classify a document d

- retrieve the k nearest neighbors in the training set according to the vector space model
- Choose the majority class label as the class of the document

Underlying assumptions

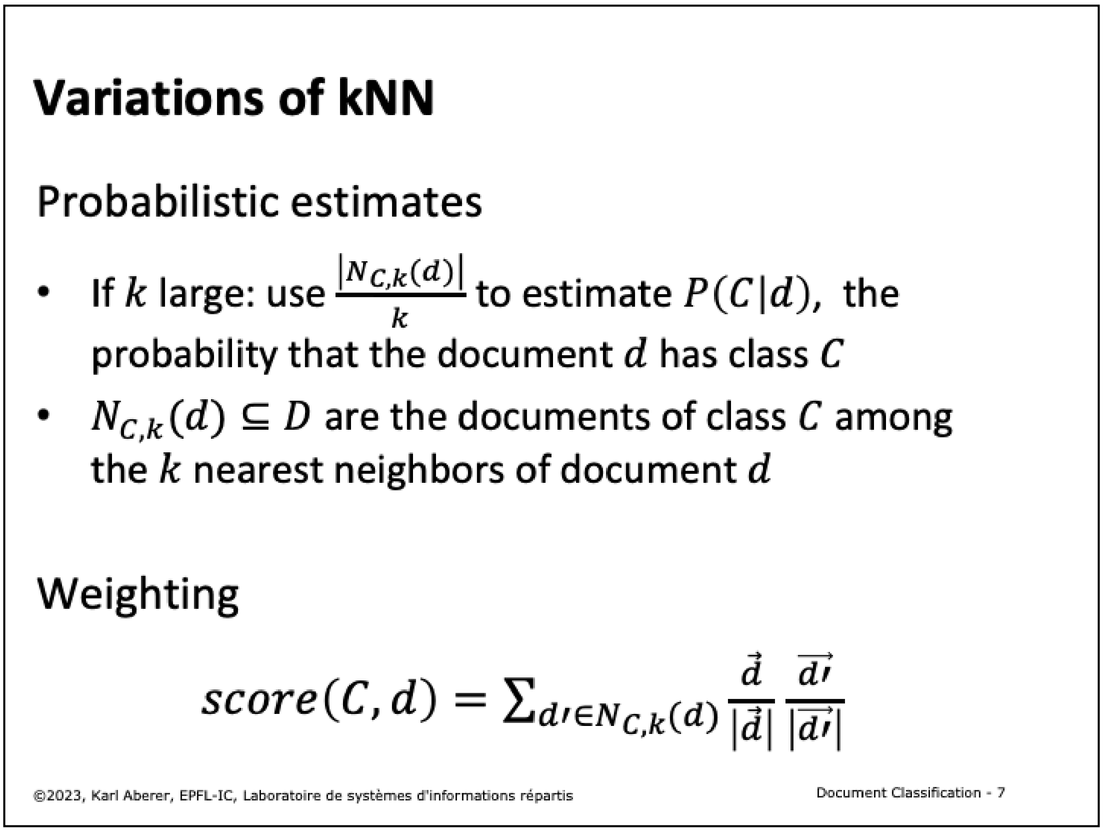
- documents in the same class form contiguous regions
- different classes do not overlap

A standard method for document classification is rooted in ideas we have already introduced for information retrieval. It is assumed that there exists a training set of labelled documents. In the kNN approach, the top k labelled documents that are most similar to the document to be classified are retrieved. The predicted class label is then chosen to be the one that has the majority among the retrieved documents. Ideally, k is chosen to be odd to break ties. The parameter k can also be determined using standard approaches of parameter tuning with a validation sets.

kNN relies on the implicit assumption that documents in the same class form contiguous regions, i.e., are close to each other, and that the regions of documents from different classes are not overlapping.

From the perspective of computational complexity, kNN exhibits the following properties:

- Training requires to compute vector representations of documents. In the case of using tf-idf weights this implies tokenization of documents and computing the idf statistics for the document collection.
- Inferring the class label requires the search for the closest k neighbors. Using simple scanning of the training set, the cost of inference grows linearly in the size of the training set, and therefore can be expensive.



Variations of kNN

Probabilistic estimates

- If k large: use $\frac{|N_{C,k}(d)|}{k}$ to estimate $P(C|d)$, the probability that the document d has class C
- $N_{C,k}(d) \subseteq D$ are the documents of class C among the k nearest neighbors of document d

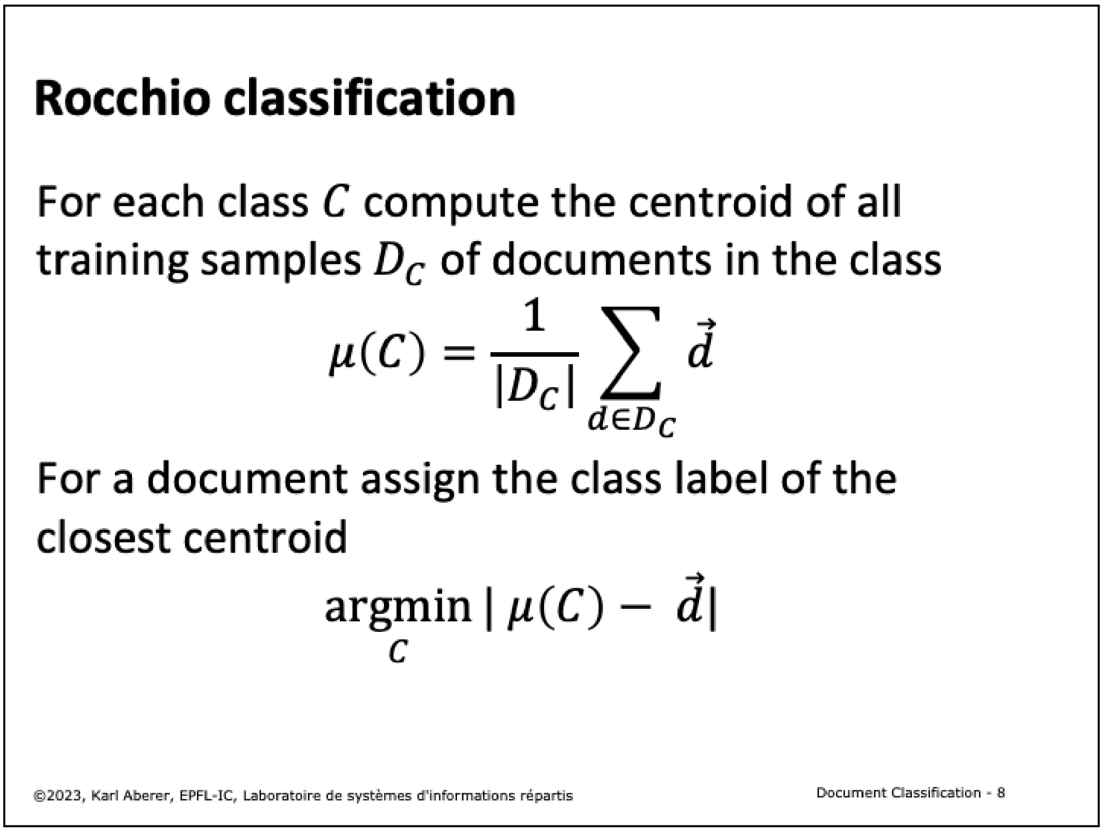
Weighting

$$score(C,d) = \sum_{d\prime \in N_{C,k}(d)} \frac{\vec{d}}{|\vec{d}|} \frac{\overrightarrow{d\prime}}{|\overrightarrow{d\prime}|}$$

©2023, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Document Classification - 7

Two variations of kNN can be used. With probabilistic estimates not a class label is returned, but a probability distribution of the class labels. This is meaningful as k grows larger. With weighting the neighboring documents of a class contribute according to their distance to the document to be labelled. Therefore, documents that are more remote have less influence on the decision of the label. In general, this method is more accurate than the basic method.



Rocchio classification

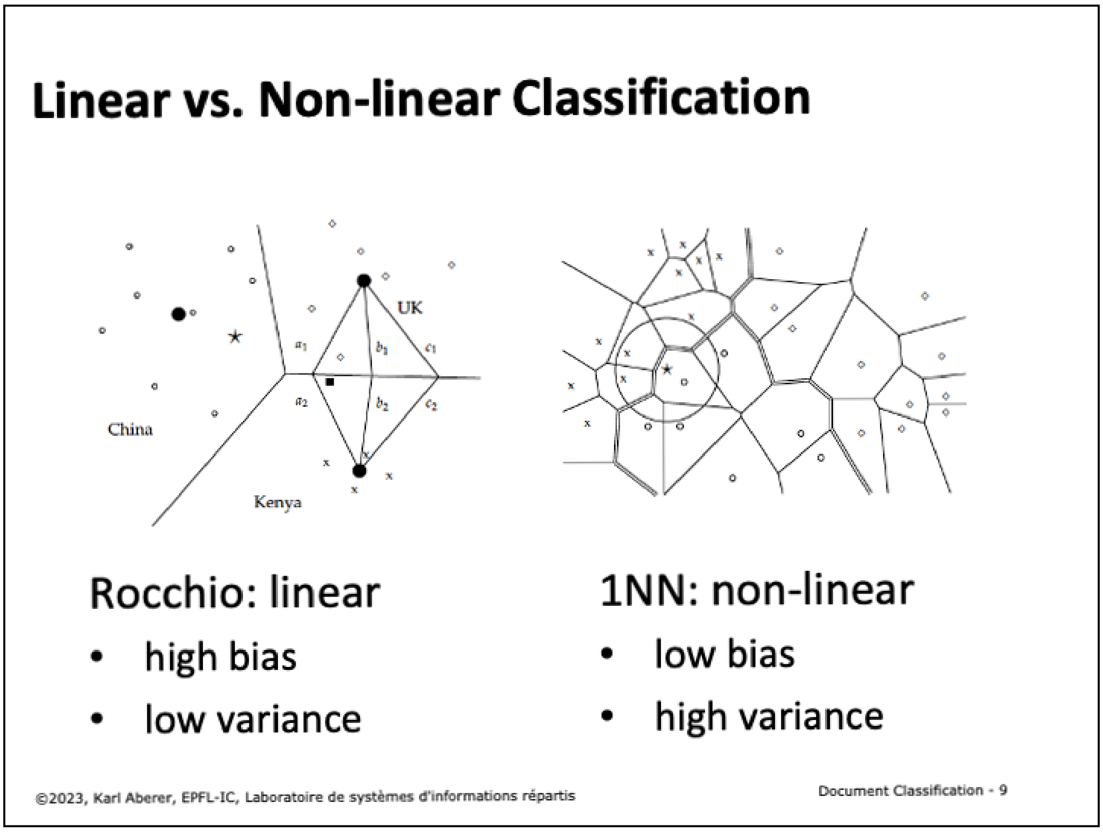
For each class C compute the centroid of all training samples D_C of documents in the class

$$\mu(C) = \frac{1}{|D_C|} \sum_{d \in D_C} \vec{d}$$

For a document assign the class label of the closest centroid

$$\underset{C}{\operatorname{argmin}} \, | \, \mu(C) - \vec{d} |$$

An alternative model using the vector representation of documents for classification is the so-called Rocchio classification. As the name suggests, it exploits the same ideas that are used for user relevance feedback with the Rocchio method. For each class label, the centroids of the document vectors o the documents in the training set are computed. These centroids are then used to decide the class label of document by selecting the closest centroid. This method assumes that classes are not only continguous in the vector space, but also convex. This assumption is many practical cases not valid.



Linear vs. Non-linear Classification

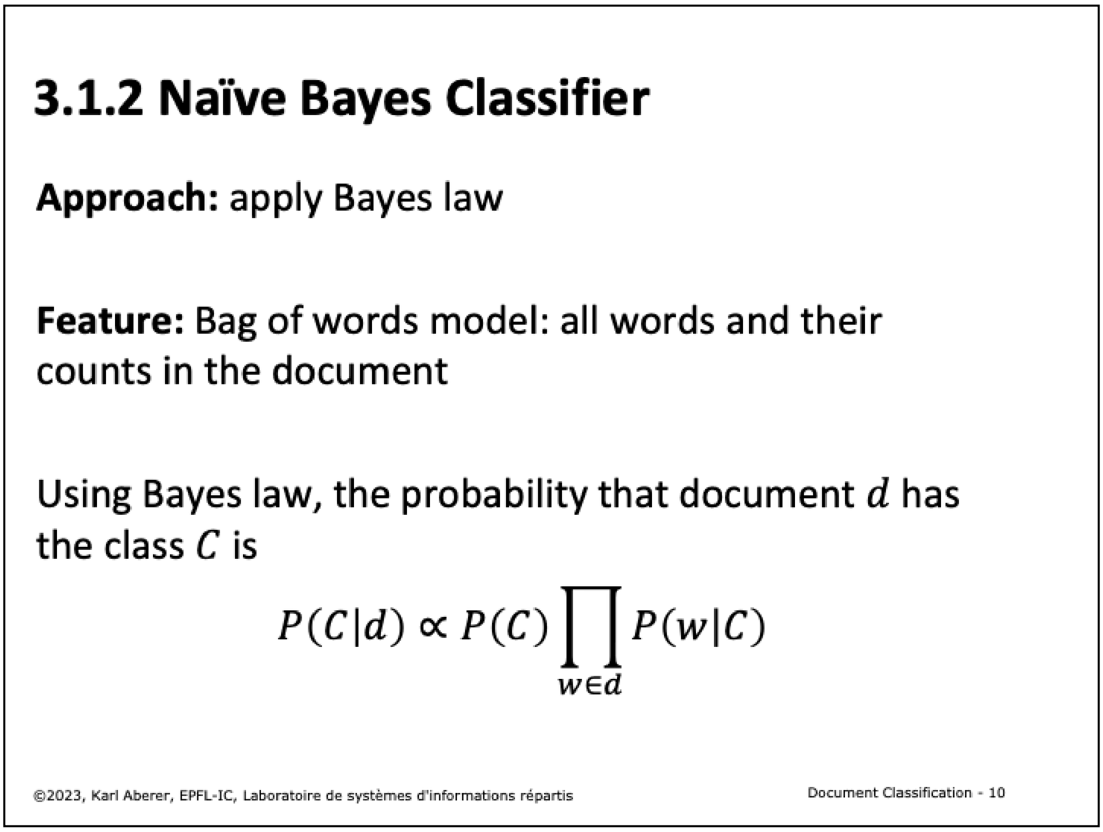
Rocchio: linear

- high bias
- low variance

1NN: non-linear

- low bias
- high variance

One of the most important questions in machine learning is the tradeoff among bias and variance. In short, bias essentially captures whether a model makes systematic mistakes independent of the training set used. High bias is in general associated with an incapacity of the model to capture essential aspects of the domain, which again results from too simple models. Variance captures the stability of the model under different training sets. High variance in general is associated with a too large capacity of the model to capture non-essential aspects of the domain, i.e. noise, a property that is called over-fitting. Rocchio and kNN provide a nice illustration of this trade-off. Rocchio is a linear model and as such not capable to capture non-linear decision boundaries among classes as illustrated on the left. Therefore for highly convoluted class boundaries the model will always have a large error. On the other hand, kNN is non-linear and can perfectly adapt to the nonlinear decision boundaries in the training data, when using k=1. However, this also means the model represents noise and outliers in the training data, and is prone to very instable predictions as the training data changes. kNN can be undrestood as a way to moderate the effect by "smoothing" the prediction boundaries over larger regions.



3.1.2 Naïve Bayes Classifier

Approach: apply Bayes law

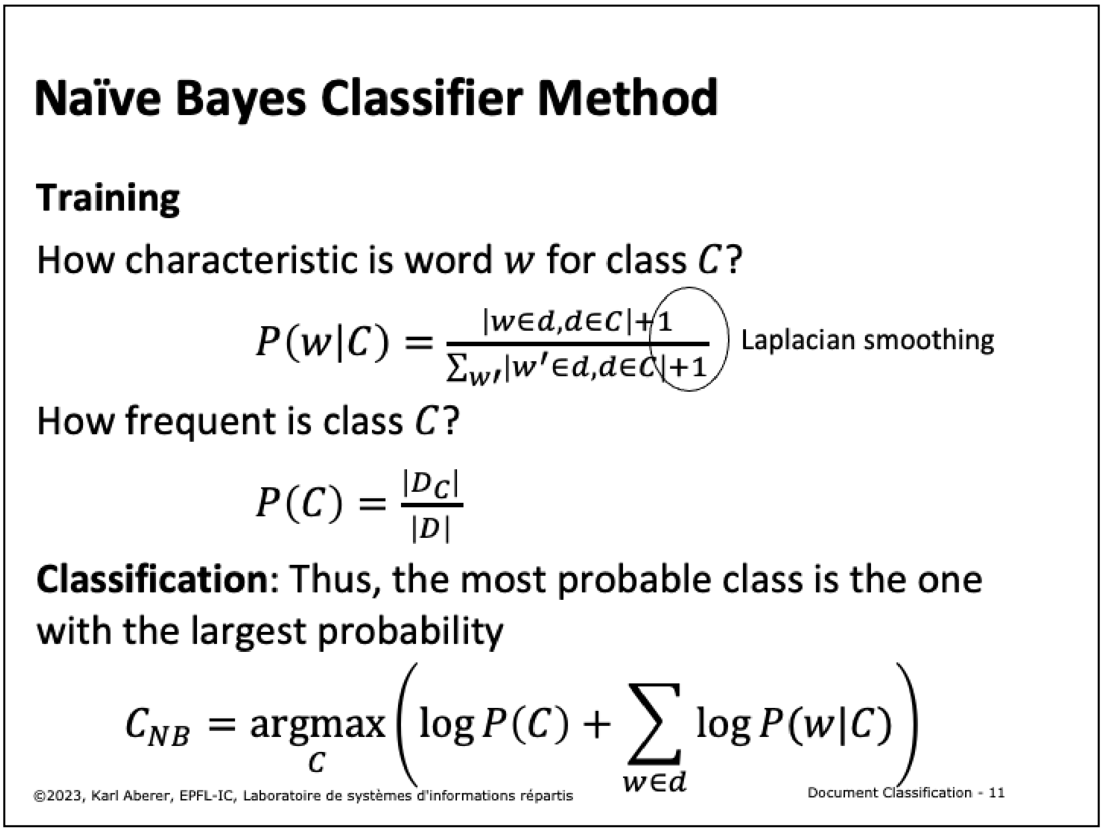
Feature: Bag of words model: all words and their counts in the document

Using Bayes law, the probability that document d has the class C is

$$P(C|d) \propto P(C) \prod_{w \in d} P(w|C)$$

Another frequently used method used in document classification is Naïve Bayes Classifier. Like kNN can be understood as the analogue of classification for vector space retrieval, Naïve Bayes can be understood as the analogue of classification for probabilistic retrieval. By applying Bayes law and making as in probabilistic retrieval an independence assumptions on the occurrence of words in documents, we can derive the probability of a document belonging to a class. The probabilities $P(C)$ and $P(w|C)$ are then derived from the available training data.

One problem in this formulation of the probability $P(C|D)$ is that for words that do not occur together with the class label in the training set, the probability of a document containing this word to have the class label becomes zero. This is an analogue problem we stated for the basic form of probabilistic retrieval, where documents not containing all query words would never be retrieved.



Naïve Bayes Classifier Method

Training

How characteristic is word w for class C?

$$P(w|C) = \frac{|w \in d, d \in C| + 1}{\sum_{w'} |w' \in d, d \in C| + 1}$$ Laplacian smoothing

How frequent is class C?

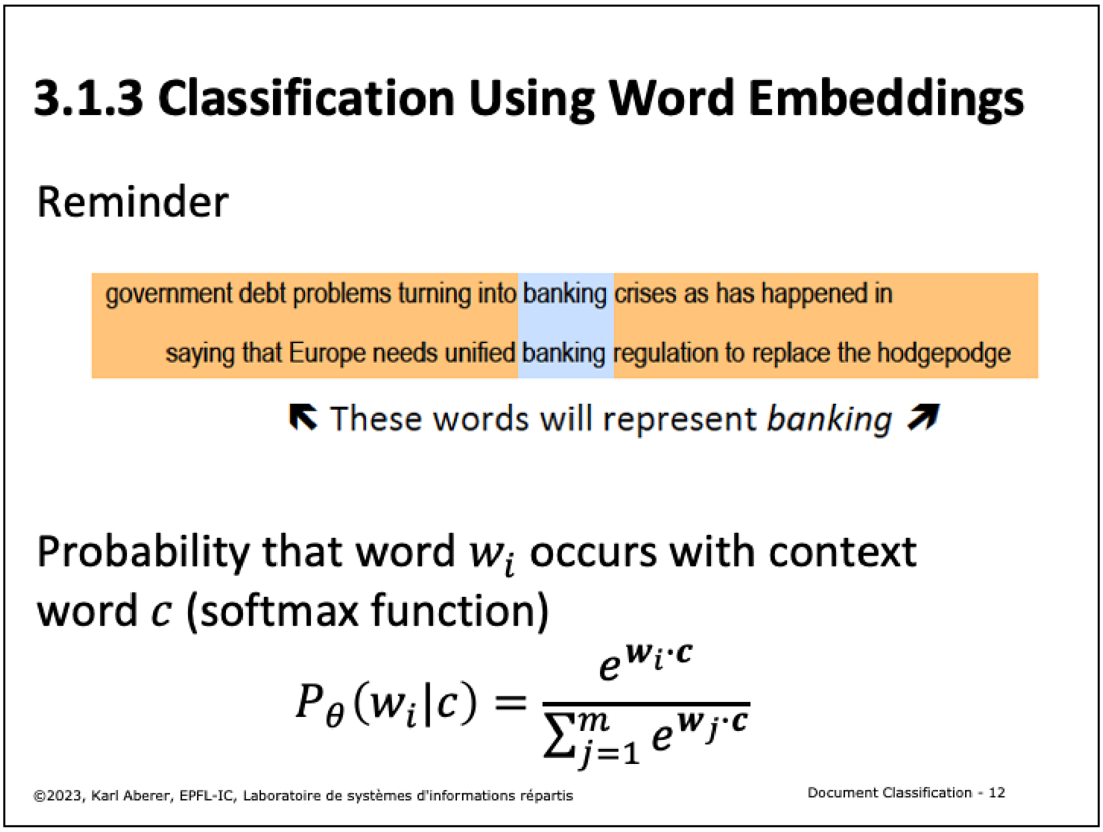
$$P(C) = \frac{|D_C|}{|D|}$$

Classification: Thus, the most probable class is the one with the largest probability

$$C_{NB} = \underset{C}{\operatorname{argmax}} \left(\log P(C) + \sum_{w \in d} \log P(w|C) \right)$$

For training a Naïve Bayes model we have to estimate the probabilities $P(w|C)$ and $P(C)$. $P(C)$ is straightforward to compute it is simply the probability of each class label to occur in the training set. For the probability $P(w|C)$ we apply Laplacian smoothing to avoid the problem of words not occurring for a class in the training set. Then the probability is estimated as the likelihood of word occurring in the documents of the class as compared to the overall number of occurrences of the word.

Classification is performed by selecting the class with the highest probability to occur. This is computed by applying the logarithm to the estimated probability, as the summation is numerically more stable than multiplication.



3.1.3 Classification Using Word Embeddings

Reminder

government debt problems turning into banking crises as has happened in

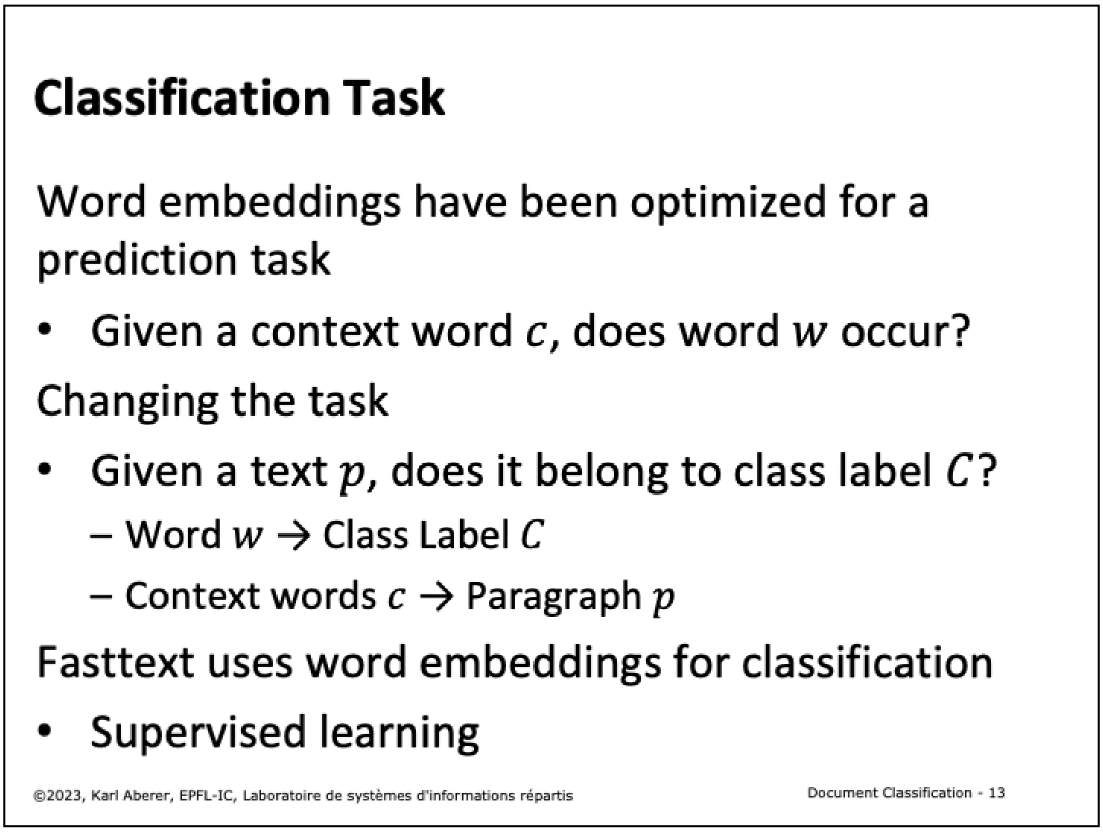
saying that Europe needs unified banking regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

Probability that word w_i occurs with context word c (softmax function)

$$P_\theta(w_i|c) = \frac{e^{\mathbf{w}_i \cdot \mathbf{c}}}{\sum_{j=1}^{m} e^{\mathbf{w}_j \cdot \mathbf{c}}}$$

Word embeddings provide a representation of documents that can not only be used for information retrieval, but also for document classification. They provide a low-dimensional feature vector that can be used for performing classification with supervised learning. We will now introduce one recent approach that has been proposed with the Fasttext approach. In fact, the initial application of FastText was document classification.



Classification Task

Word embeddings have been optimized for a prediction task

- Given a context word c, does word w occur?

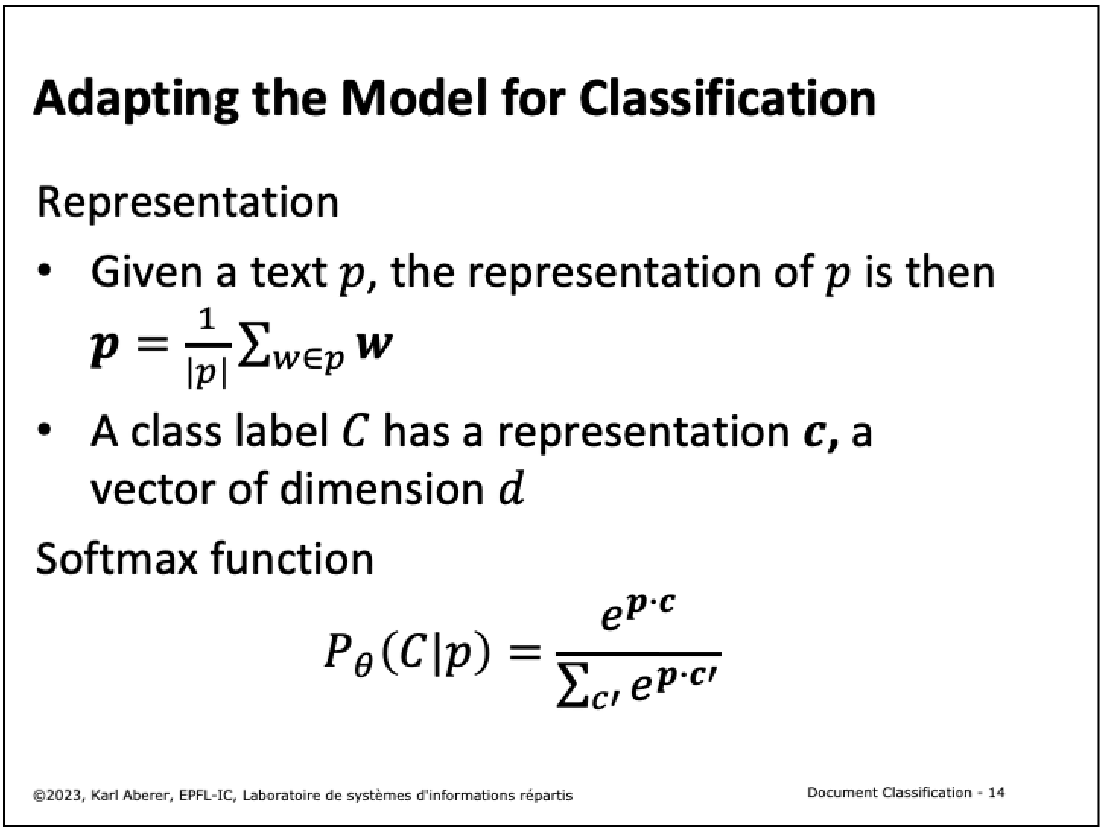
Changing the task

- Given a text p, does it belong to class label C?
  - Word w → Class Label C
  - Context words c → Paragraph p

Fasttext uses word embeddings for classification

- Supervised learning

Document classification is a task that is different from the task considered for constructing word embeddings in an unsupervised approach, as we have introduced it earlier. The task we had considered consisted of predicting the co-occurrence of words and context words. For document classification we have a training data set of labelled documents. Using that training data, we can modify the task as follows: instead of predicting for a given context word the corresponding word (or vice versa), we predict for a piece of text, a paragraph, the related class label. Therefore, the role of the context word will be taken over by the paragraph for which the label is predicted, and the prediction is on the class label and not on the center word.



Adapting the Model for Classification

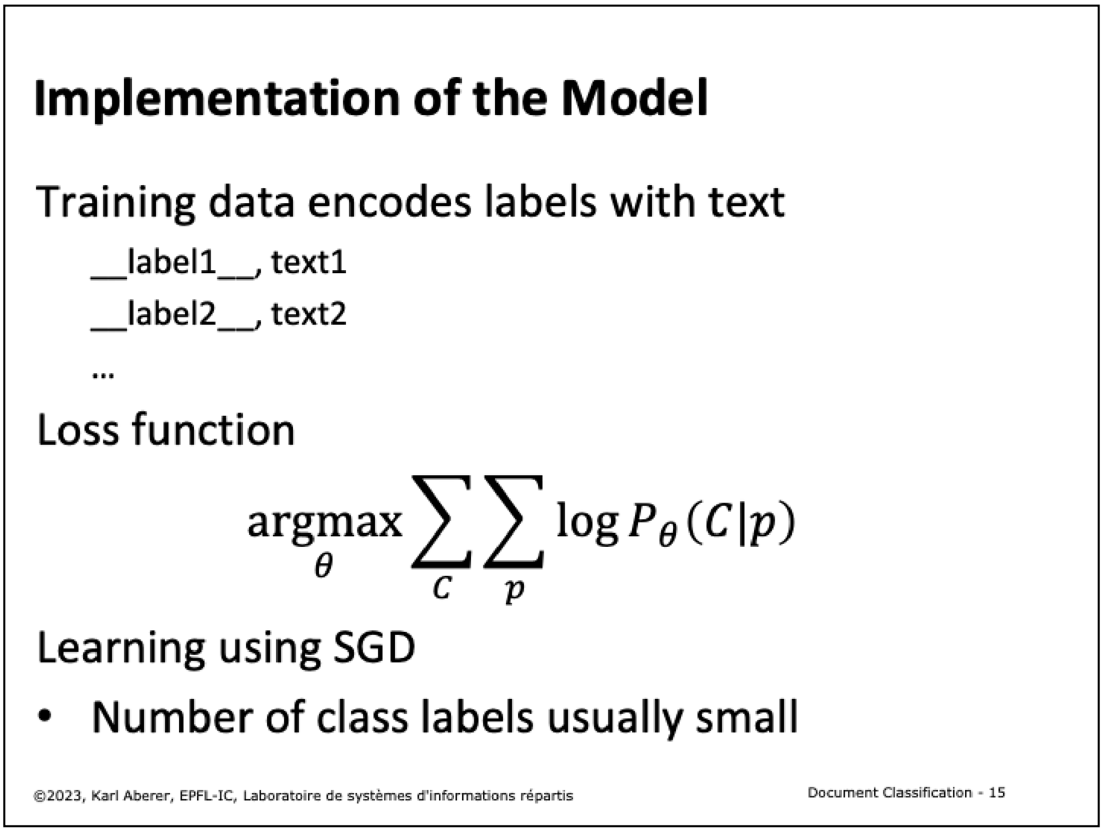
Representation

- Given a text p, the representation of p is then $\boldsymbol{p} = \frac{1}{|p|} \sum_{w \in p} \boldsymbol{w}$
- A class label C has a representation $\boldsymbol{c}$**,** a vector of dimension d

Softmax function

$$P_\theta(C|p) = \frac{e^{\boldsymbol{p} \cdot \boldsymbol{c}}}{\sum_{c'} e^{\boldsymbol{p} \cdot \boldsymbol{c}'}}$$

Since the prediction is performed not for single words, but for texts, the text from paragraphs is aggregated by computing the average word embedding vector of all words in the paragraph. For each class label also a d-dimensional vector is generated. The predicted probability of a class label occurring with a paragraph is then computed using the softmax function. Note that different to the use of the softmax function for the word prediction task, the number of class labels is usually small, so that the computation of the softmax function does not create signficant computational cost.



Implementation of the Model

Training data encodes labels with text

__label1__, text1

__label2__, text2

…

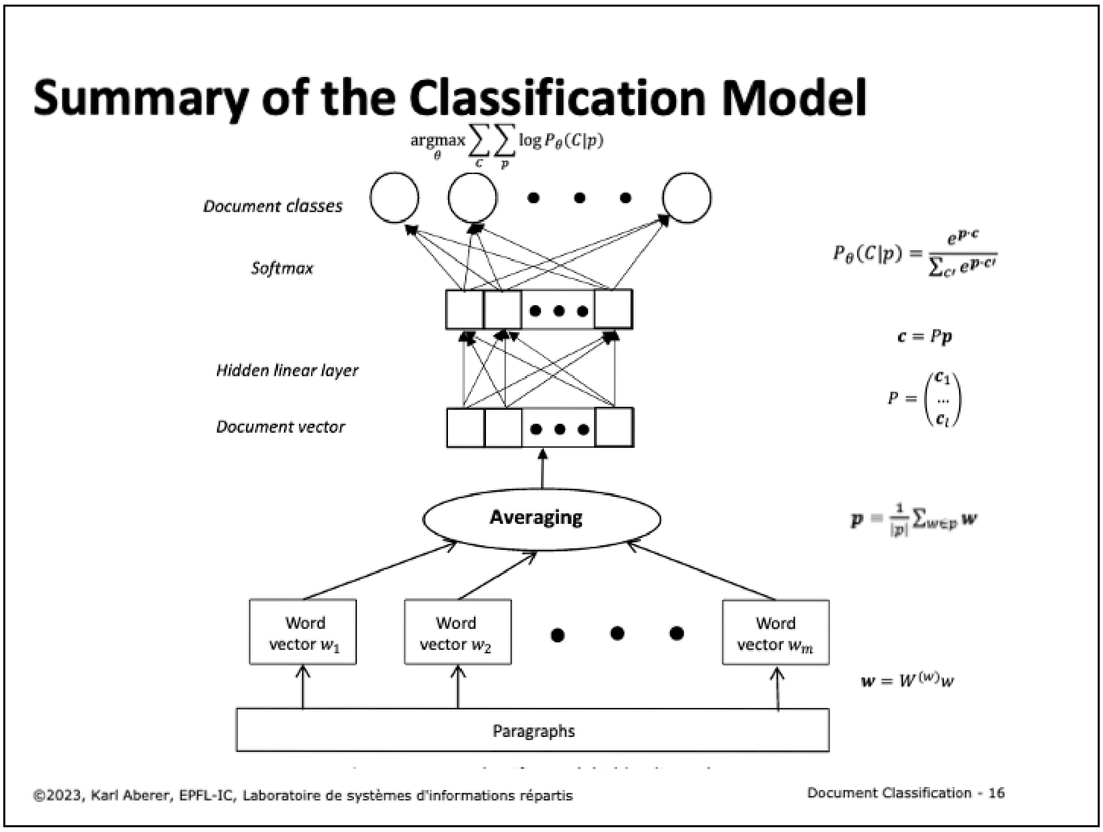
Loss function

$$\underset{\theta}{\operatorname{argmax}} \sum_{C} \sum_{p} \log P_{\theta}(C|p)$$

Learning using SGD

- Number of class labels usually small

For the implementation of the model the training data is presented as pairs of labels and text paragraphs. The simplest loss function is maximizing the logarithm of the joint probabilites of all class label-paragraph pairs. Learning can be performed using standard stochastic gradient descent. In practice, different variants of this model are implemented, e.g., using alternative loss functions.



Summary of the Classification Model

$$\underset{\theta}{\operatorname{argmax}} \sum_{C} \sum_{p} \log P_\theta(C|p)$$

$$P_\theta(C|p) = \frac{e^{\boldsymbol{p} \cdot \boldsymbol{c}}}{\sum_{c'} e^{\boldsymbol{p} \cdot \boldsymbol{c'}}}$$

$$\boldsymbol{c} = P\boldsymbol{p}$$

$$P = \begin{pmatrix} \boldsymbol{c}_1 \\ \dots \\ \boldsymbol{c}_l \end{pmatrix}$$

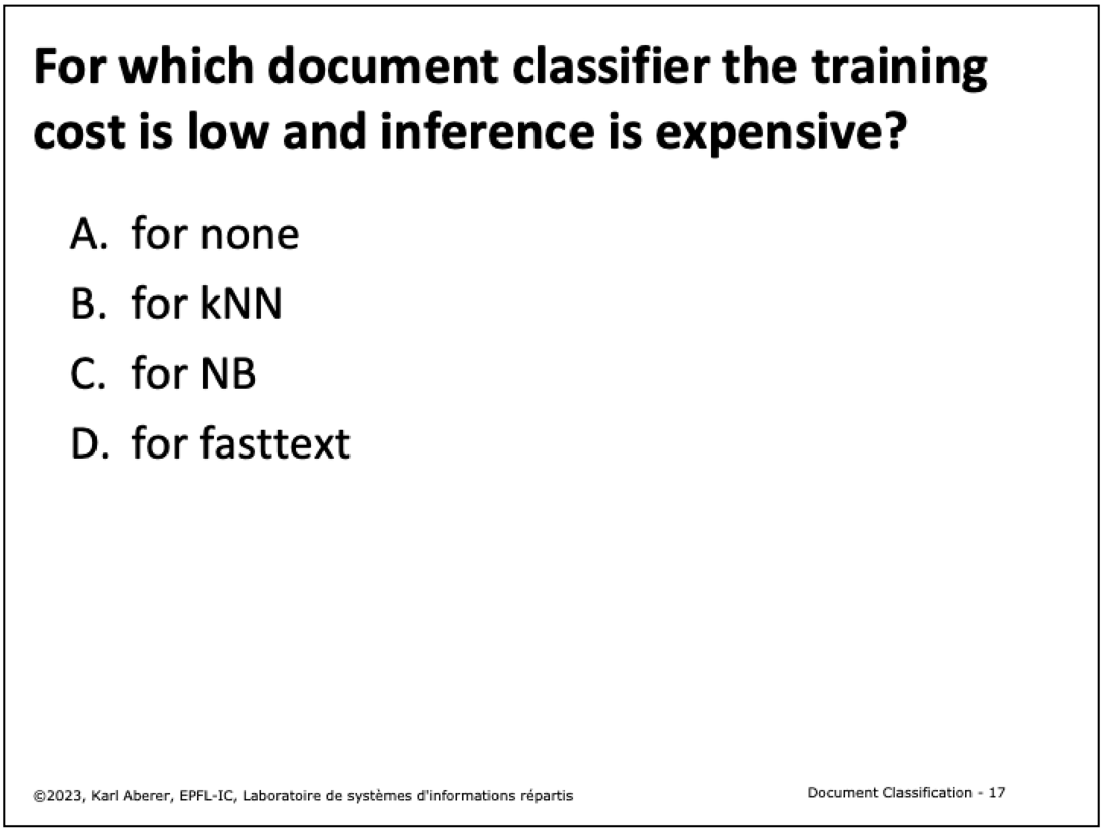
$$\boldsymbol{p} = \frac{1}{|p|} \sum_{w \in p} \boldsymbol{w}$$

$$\boldsymbol{w} = W^{(w)} w$$

Document Classification - 16

This figure summarizes the different steps of the fasttext classification model. Note that the encoding of the class labels into d-dimensional vectors can be representated by a matrix P, with the vectors for each of the l class labels as rows. The graphical representation used in this figure is a common way to describe a model used for learning. One can understand it to depict graphically the composition of a complex function.

Figure: Analysis and Optimization of fastText Linear Text Classifier, Vladimir Zolotov and David Kung, https://arxiv.org/abs/1702.05531



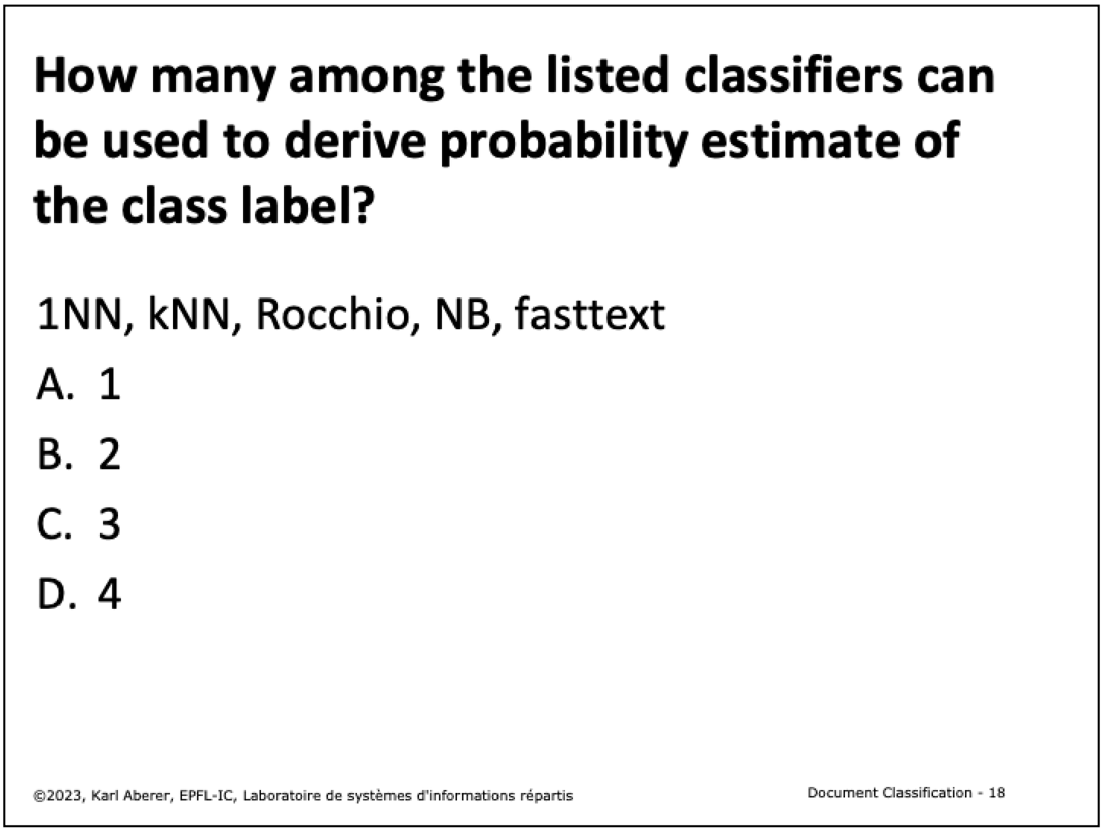
For which document classifier the training cost is low and inference is expensive?

A. for none

B. for kNN

C. for NB

D. for fasttext



How many among the listed classifiers can be used to derive probability estimate of the class label?

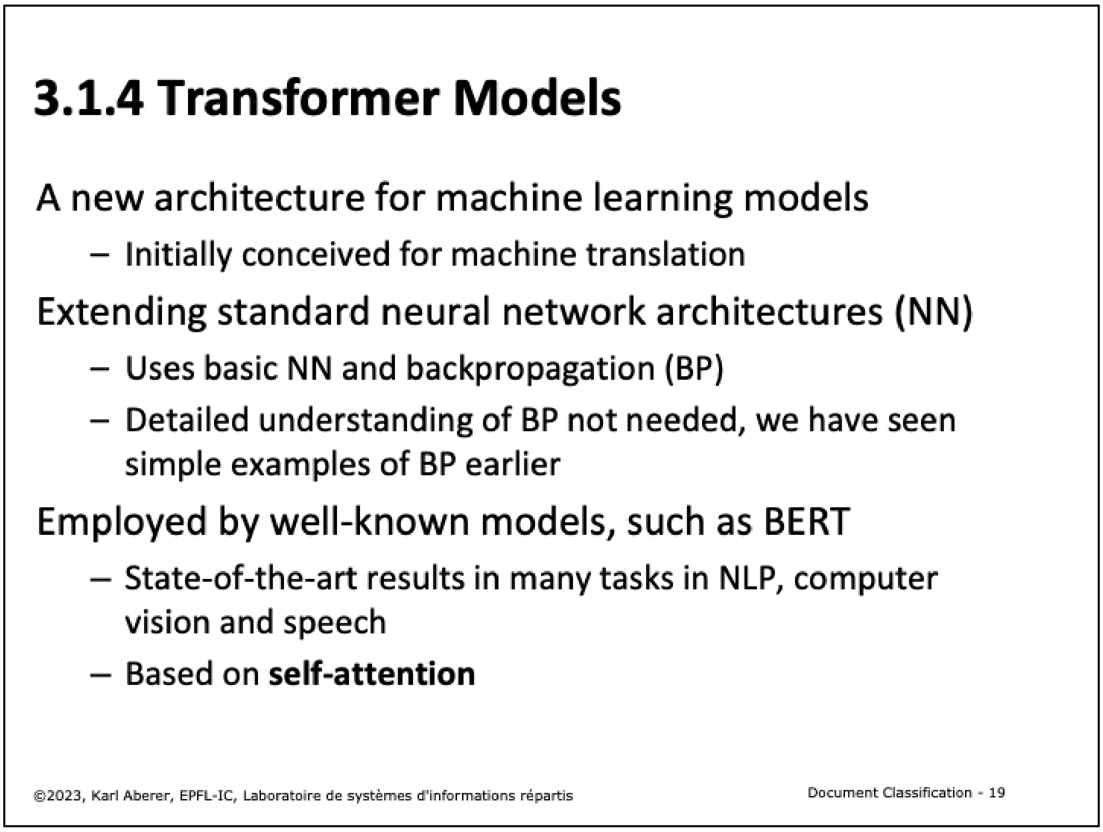
1NN, kNN, Rocchio, NB, fasttext

A. 1

B. 2

C. 3

D. 4



3.1.4 Transformer Models

A new architecture for machine learning models

- Initially conceived for machine translation

Extending standard neural network architectures (NN)

- Uses basic NN and backpropagation (BP)
- Detailed understanding of BP not needed, we have seen simple examples of BP earlier

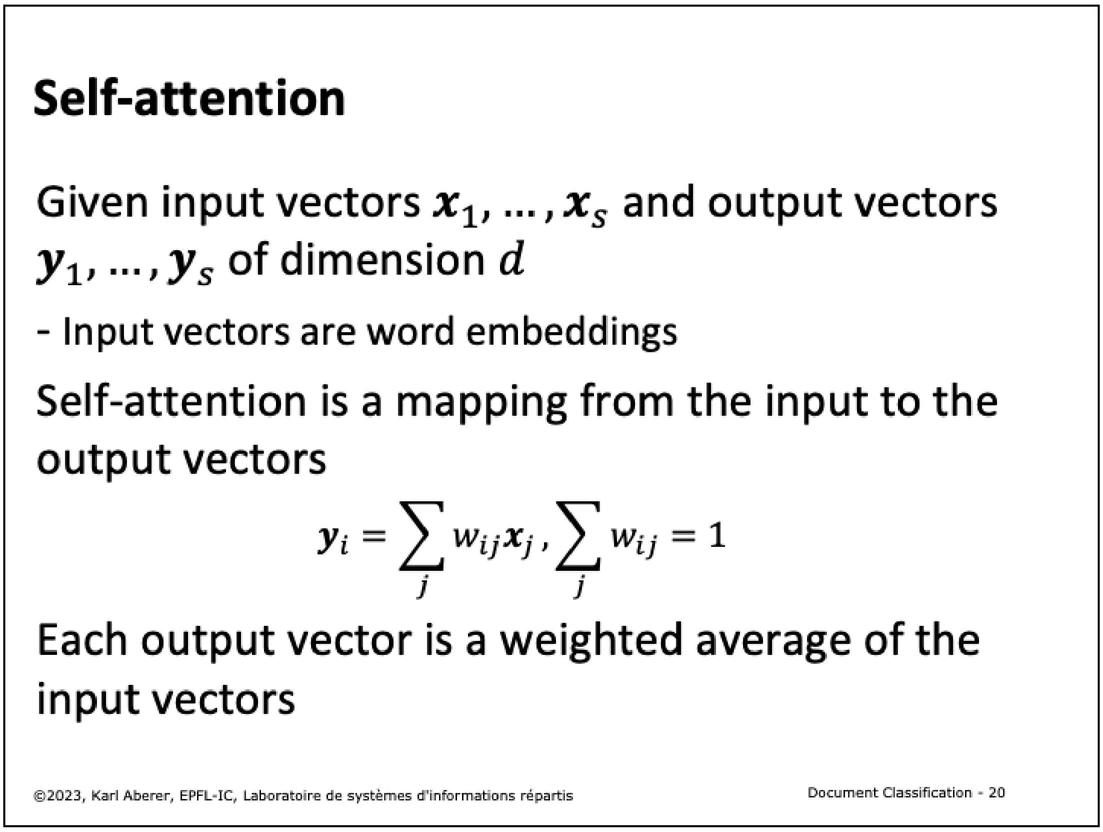
Employed by well-known models, such as BERT

- State-of-the-art results in many tasks in NLP, computer vision and speech
- Based on **self-attention**

©2023, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Document Classification - 19

Transformer models are a new type of models for creating text embedding, that have initially been developed for machine translation, and later been adapted for other tasks, such as document classification. They are based on an extension of neural networks, using a mechanism known as self-attention. Self-attention introduces into the model a way to capture contextual dependencies among words in a text, which has led to a significant improvement in the performance for many natural language processing tasks. One of the best-known models of this class is BERT.



Self-attention

Given input vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_s$ and output vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_s$ of dimension d

- Input vectors are word embeddings

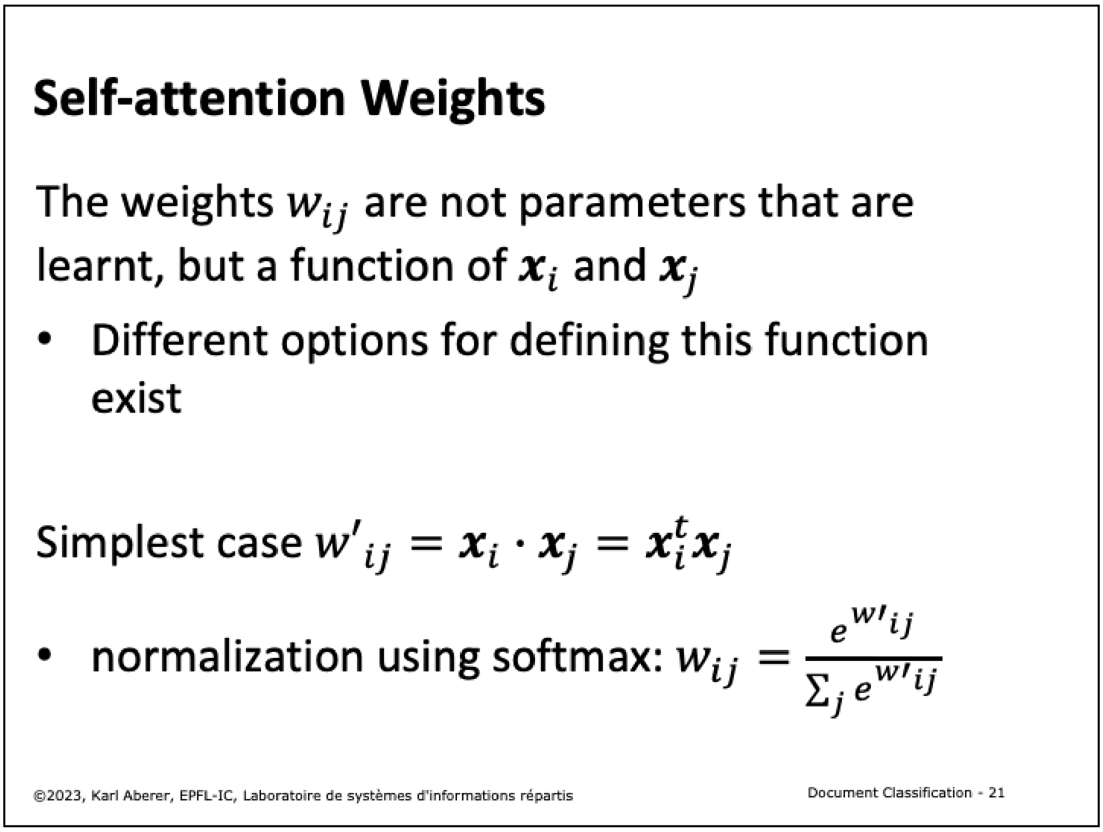
Self-attention is a mapping from the input to the output vectors

$$\boldsymbol{y}_i = \sum_j w_{ij}\boldsymbol{x}_j\,, \sum_j w_{ij} = 1$$

Each output vector is a weighted average of the input vectors

In its simplest form, the self-attention mechanism is very straightforward. It transforms a set of input vectors, e.g., representing the words of a sentence that are learnt like with word embeddings, to a set of output vectors of the same dimension. The self-attention mechanisms consists of a linear mapping from the input vectors to the output vectors. One can consider this mapping as computing weighted averages of the input vectors.

Note that in the following we will use a notation different from before, where we denote word vectors as $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_s$ and not as $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_s$. s one can interpret as the length of a sentence consisting of words $w_1, \ldots, w_s$. We will consistently denote vectors in bold.



Self-attention Weights

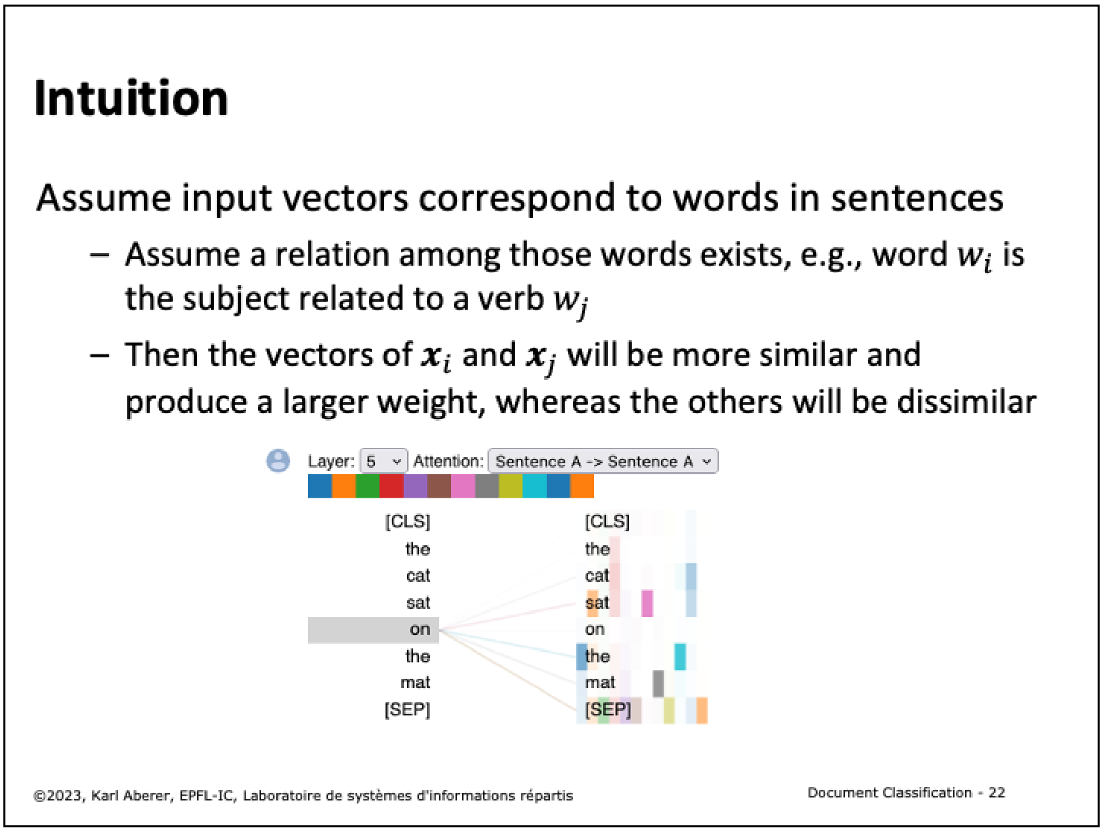
The weights w_{ij} are not parameters that are learnt, but a function of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$

- Different options for defining this function exist

Simplest case $w'_{ij} = \boldsymbol{x}_i \cdot \boldsymbol{x}_j = \boldsymbol{x}_i^t \boldsymbol{x}_j$

- normalization using softmax: $w_{ij} = \dfrac{e^{w'_{ij}}}{\sum_j e^{w'_{ij}}}$

However, the weights used in the linear mapping are not some arbitrary parameters that are learnt from training data but depend themselves on the input vectors. In the simplest case, this dependency would be given as the scalar product of two input vectors x_i and x_j producing the weight w_{ij}, followed by the application of a softmax function, such that the weights add up to 1 for one dimension. This implies that while learning the vector representation of words, a dependency among the different vectors in a sequence of words is introduced.



Intuition

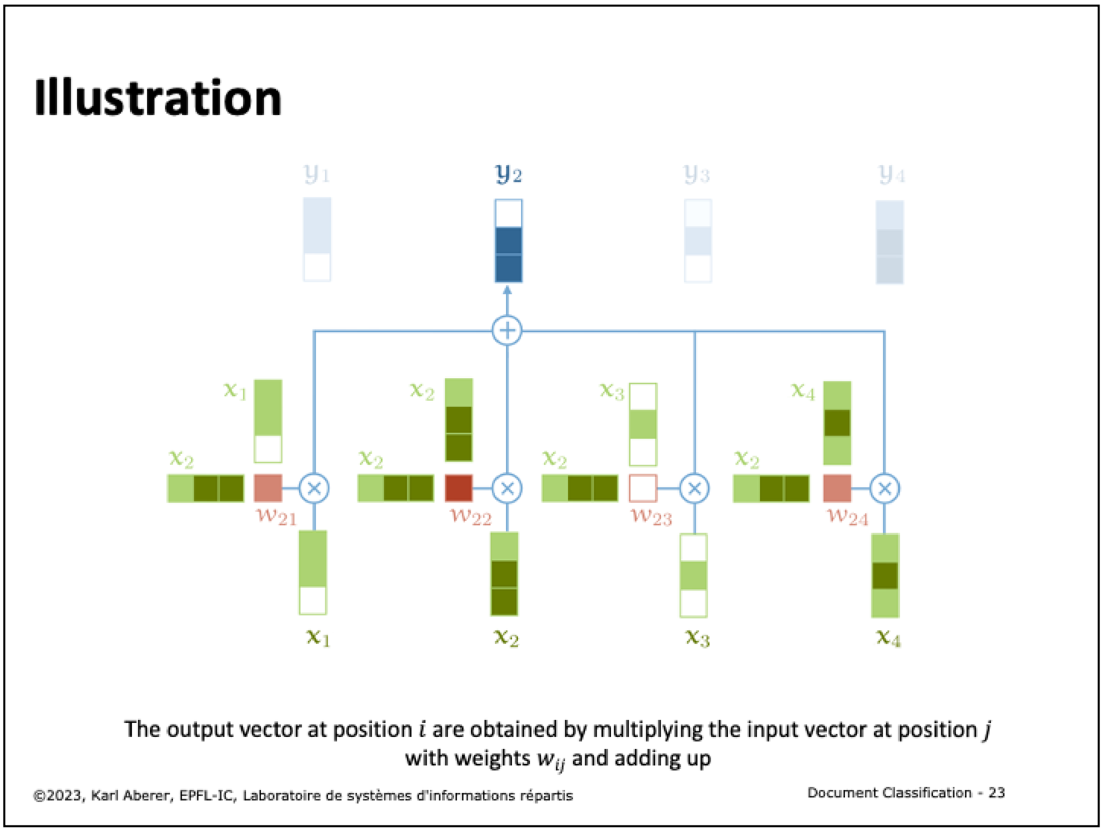
Assume input vectors correspond to words in sentences

- Assume a relation among those words exists, e.g., word w_i is the subject related to a verb w_j
- Then the vectors of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ will be more similar and produce a larger weight, whereas the others will be dissimilar

Layer: 5 Attention: Sentence A -> Sentence A

The self-attention mechanism is what enables to capture the contextual relationships among the words in a text. In the visualization the weights for one self-attention layer are shown. One can observe that the word “on” is most related to the words sat.

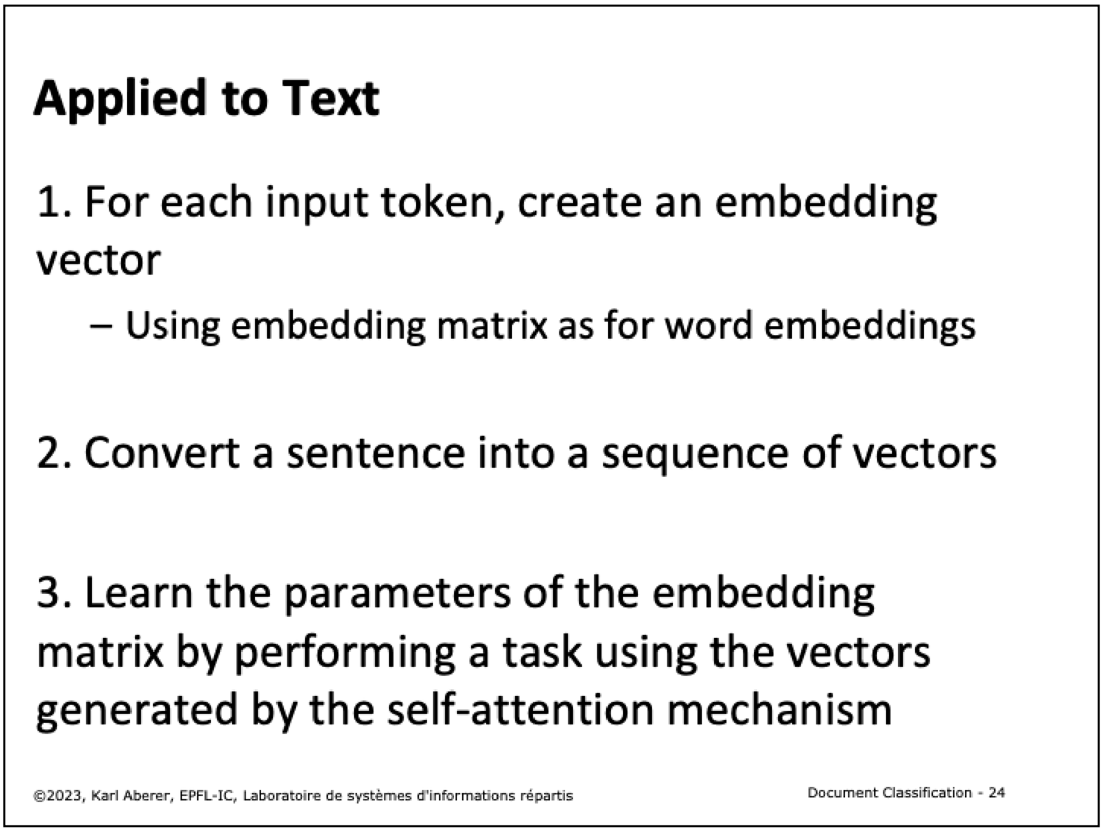
The visualization tool can be found on: https://towardsdatascience.com/deconstructing-bert-part-2-visualizing-the-inner-workings-of-attention-60a16d86b5c1



Illustration

The output vector at position i are obtained by multiplying the input vector at position j with weights w_{ij} and adding up

This figure graphically illustrates of how the output vectors are obtained from the input vectors. Note that the input vectors occur in three different ways, as input, as factor in the weight applied to itself, and as factor in the weight applied to another input vector.



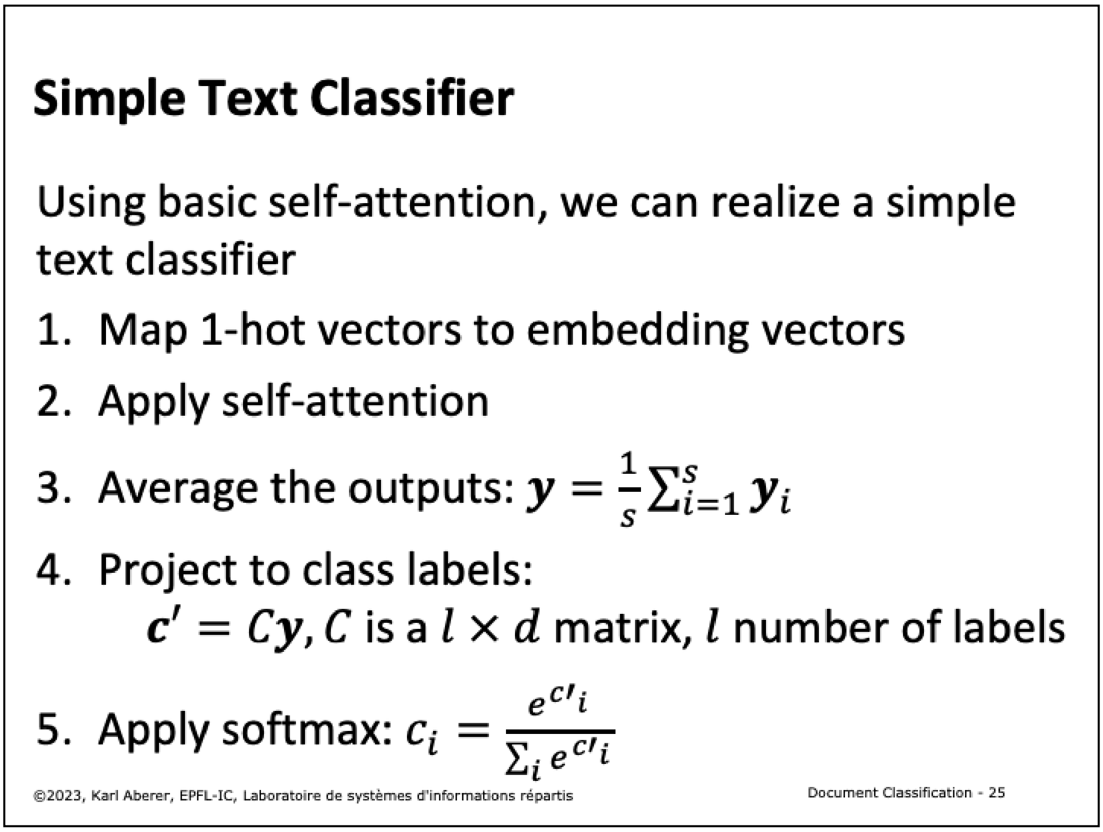
Applied to Text

1. For each input token, create an embedding vector
   - Using embedding matrix as for word embeddings

2. Convert a sentence into a sequence of vectors

3. Learn the parameters of the embedding matrix by performing a task using the vectors generated by the self-attention mechanism

When the self-attention mechanism is applied to text, the input vectors are word embeddings derived from the input token vectors (in 1-hot representation) using an embedding matrix, and a sentence is converted into a sequence of vectors. The sequence of vectors is then fed into the self-attention mechanism. The vectors resulting from the self-attention mechanism are then used to perform a task, i.e., are used to compute a loss function that is optimized by adapting the model parameters of the word embedding matrices using gradient descent.

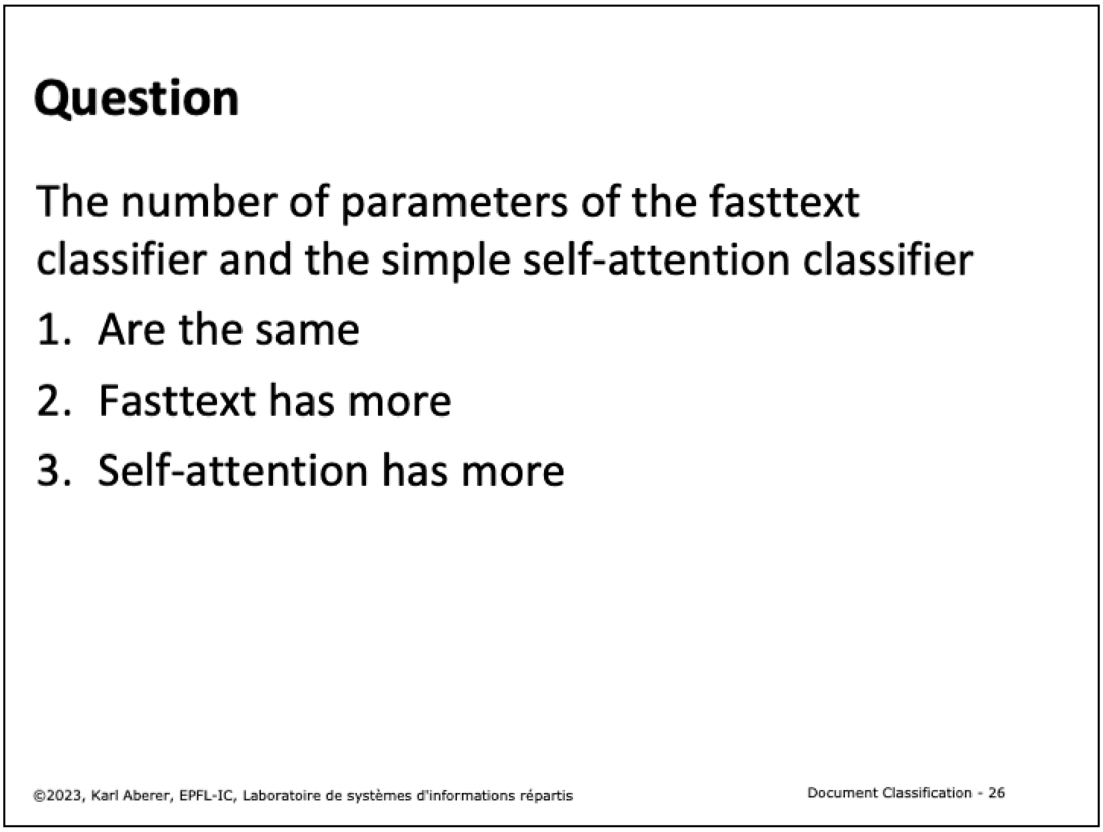


Simple Text Classifier

Using basic self-attention, we can realize a simple text classifier

1. Map 1-hot vectors to embedding vectors
2. Apply self-attention
3. Average the outputs: $\boldsymbol{y} = \frac{1}{s}\sum_{i=1}^{s} \boldsymbol{y}_i$
4. Project to class labels:
   $\boldsymbol{c}' = C\boldsymbol{y}, C$ is a $l \times d$ matrix, l number of labels
5. Apply softmax: $c_i = \frac{e^{c'_i}}{\sum_i e^{c'_i}}$

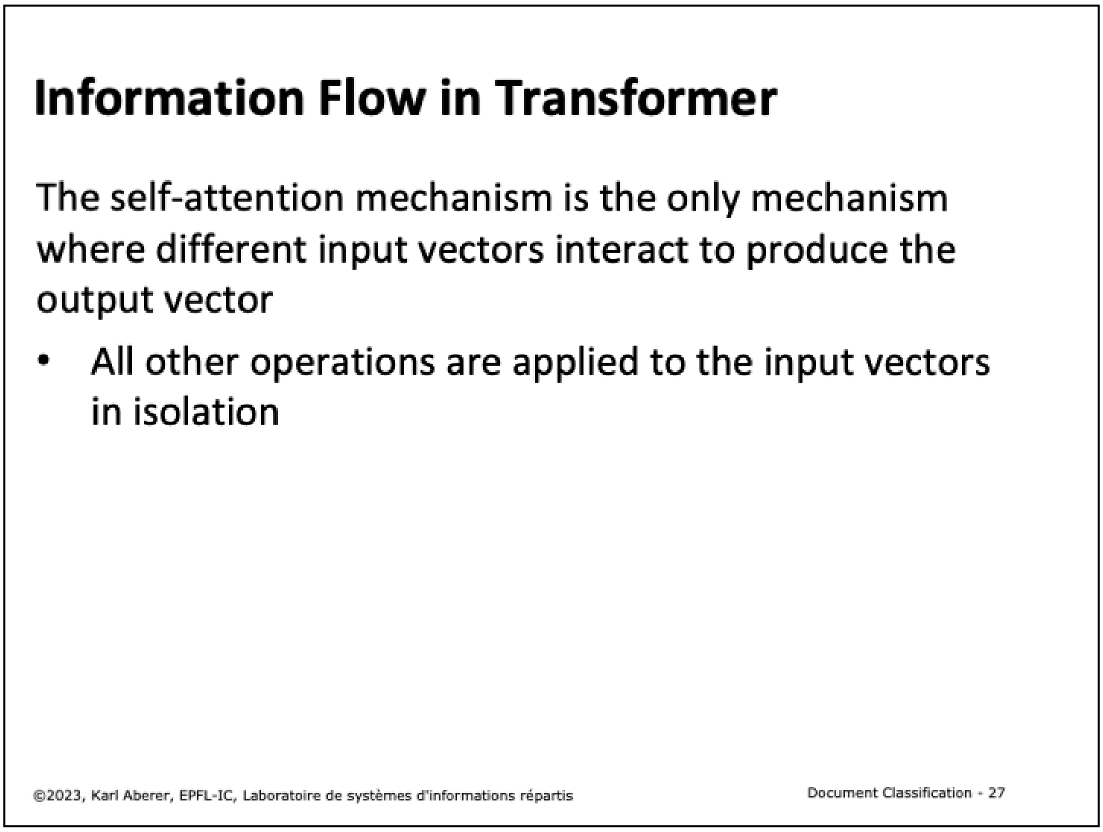
With the basic self-attention mechanism described, it would be already possible to construct a basic text classifier. For this, the output vectors generated from the self-attention mechanism are averaged, and then projected to a vector that has the dimension l of the label space (e.g. in case of binary classification to a vector of dimension 2). In order to interpret the outputs as probabilities, finally a softmax function is applied.



Question

The number of parameters of the fasttext classifier and the simple self-attention classifier

1. Are the same
2. Fasttext has more
3. Self-attention has more

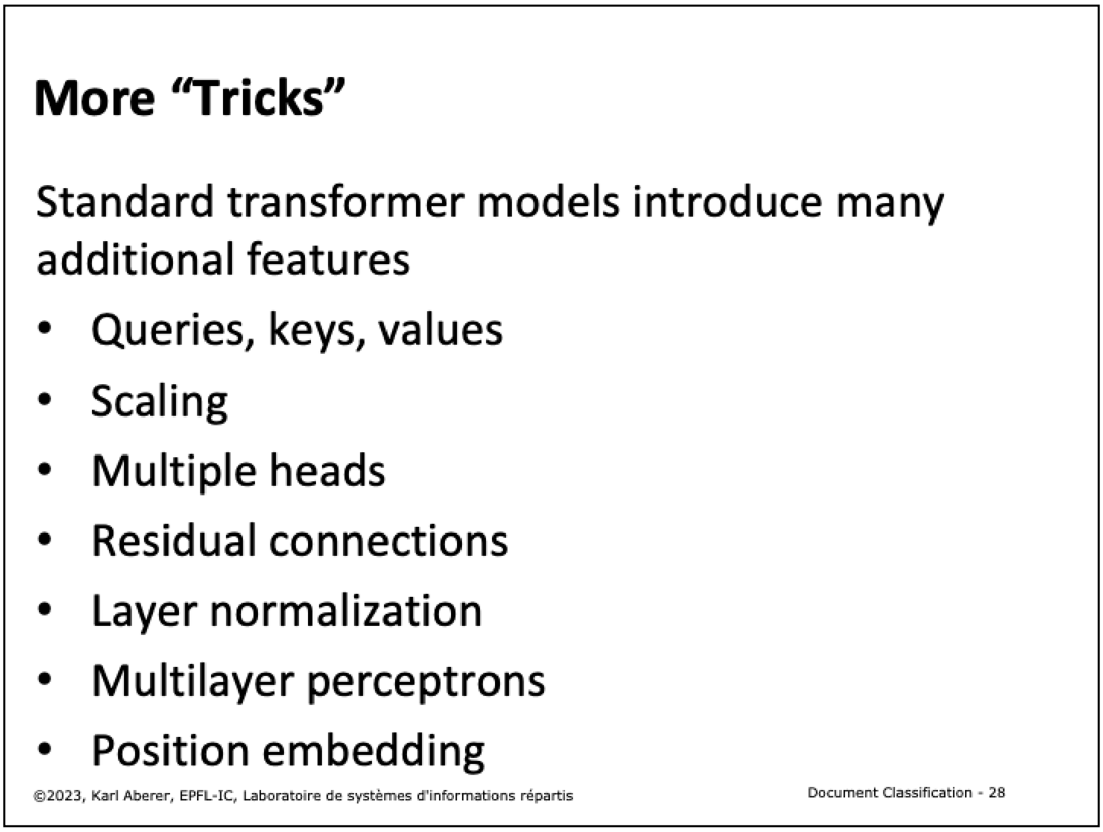


Information Flow in Transformer

The self-attention mechanism is the only mechanism where different input vectors interact to produce the output vector

- All other operations are applied to the input vectors in isolation

The input vectors interact among each other using the self-attention mechanism. In this way relations among different words in a sentence are captured. All other operations applied to vectors in a transformer network are performed independently.

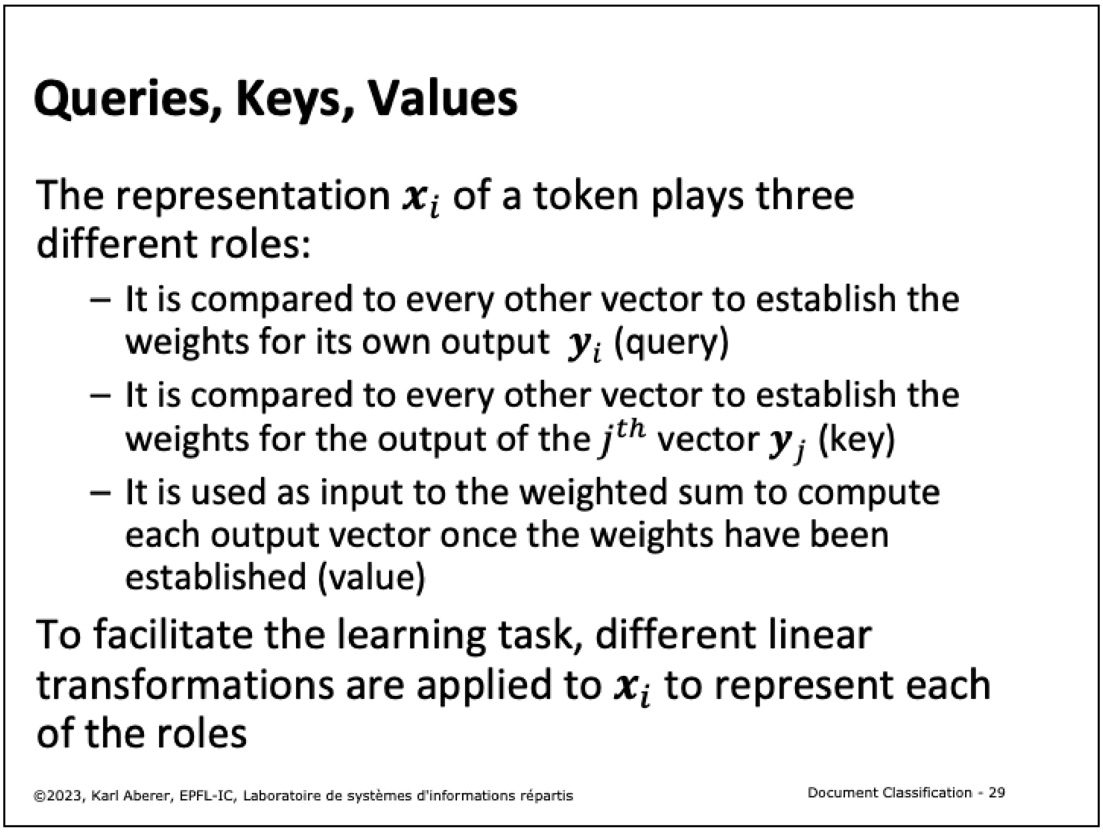


More “Tricks”

Standard transformer models introduce many additional features

- Queries, keys, values
- Scaling
- Multiple heads
- Residual connections
- Layer normalization
- Multilayer perceptrons
- Position embedding

We have shown how to construct a basic text classifier using a self-attention mechanism. In practice, many additional mechanisms are used to produce the transformer models used today in practice. The design of these mechanism is based on numerous theoretical and empirical insights gained in research. In the following we will introduce the main ideas, without being able to enter in all details.



Queries, Keys, Values

The representation $\boldsymbol{x}_i$ of a token plays three different roles:

- It is compared to every other vector to establish the weights for its own output $\boldsymbol{y}_i$ (query)
- It is compared to every other vector to establish the weights for the output of the j^{th} vector $\boldsymbol{y}_j$ (key)
- It is used as input to the weighted sum to compute each output vector once the weights have been established (value)

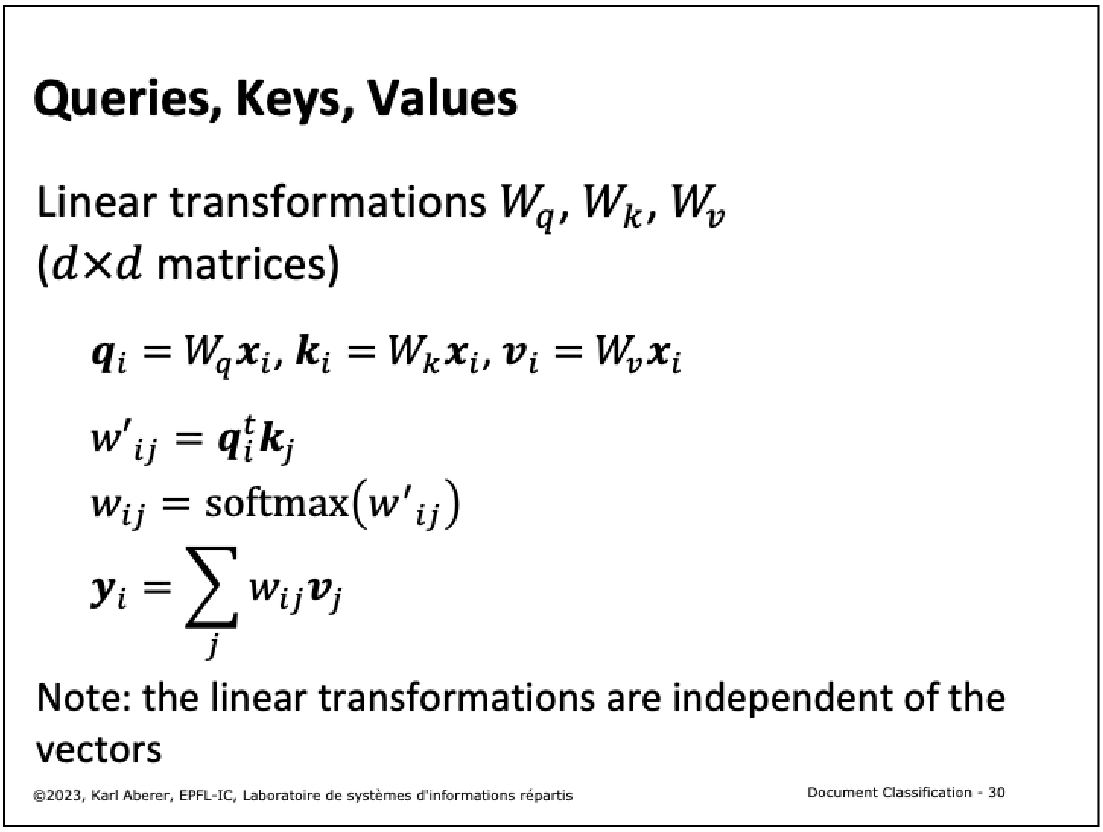
To facilitate the learning task, different linear transformations are applied to $\boldsymbol{x}_i$ to represent each of the roles

Document Classification - 29

As observed for the basic self-attention mechanism, the input vectors play 3 different rules in the self-attention mechanism.

The better accommodate for these three roles, linear transformation are introduced that generate 3 different vectors for them, from the input vectors.

These linear transformations introduce additional parameters that have to be learnt.



Queries, Keys, Values

Linear transformations W_q, W_k, W_v ($d \times d$ matrices)

$$\boldsymbol{q}_i = W_q \boldsymbol{x}_i, \boldsymbol{k}_i = W_k \boldsymbol{x}_i, \boldsymbol{v}_i = W_v \boldsymbol{x}_i$$

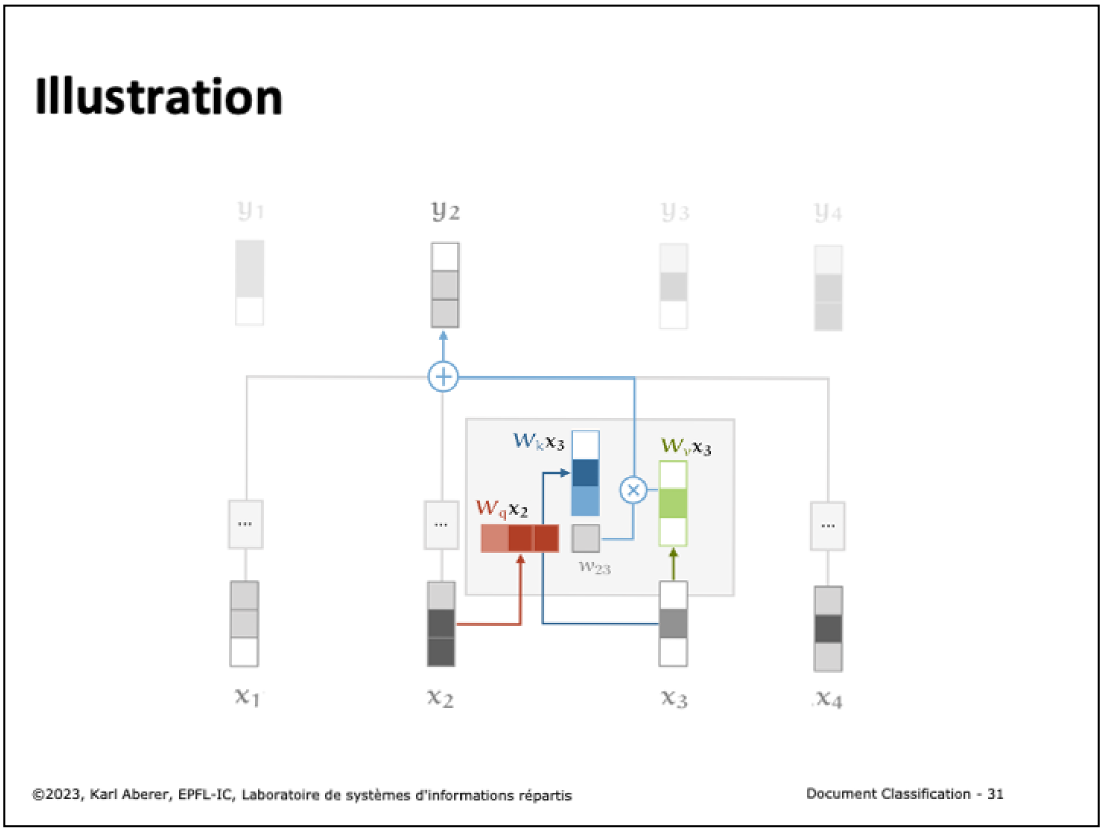
$$w'_{ij} = \boldsymbol{q}_i^t \boldsymbol{k}_j$$

$$w_{ij} = \mathrm{softmax}(w'_{ij})$$

$$\boldsymbol{y}_i = \sum_j w_{ij} \boldsymbol{v}_j$$

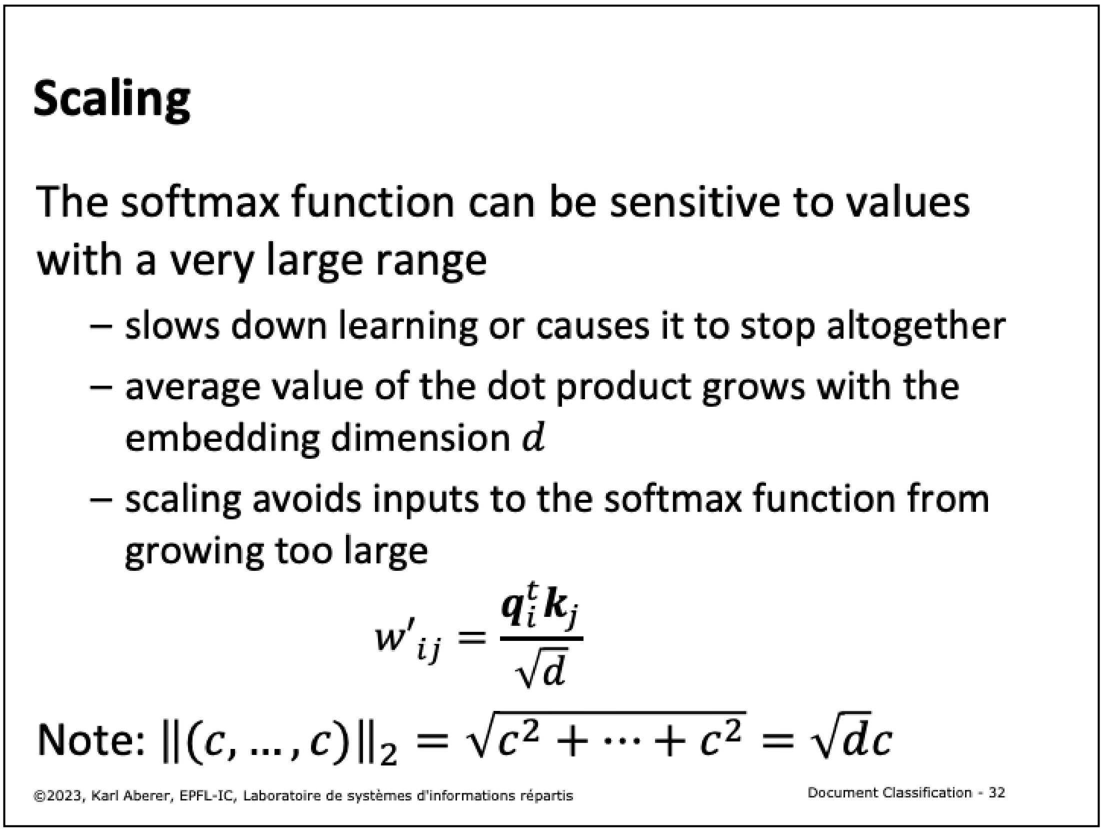
Note: the linear transformations are independent of the vectors

The linear transformations for query, key and value are part of the parameters that will be learnt, but are independent of the input vectors, i.e., only 3 constant linear mappings are learnt for a self-attention mechanism.



Illustration

The figure illustrates of how the 3 linear mappings are applied to the input vectors before the self-attention mechanisms computes a weighted average of the inputs.



Scaling

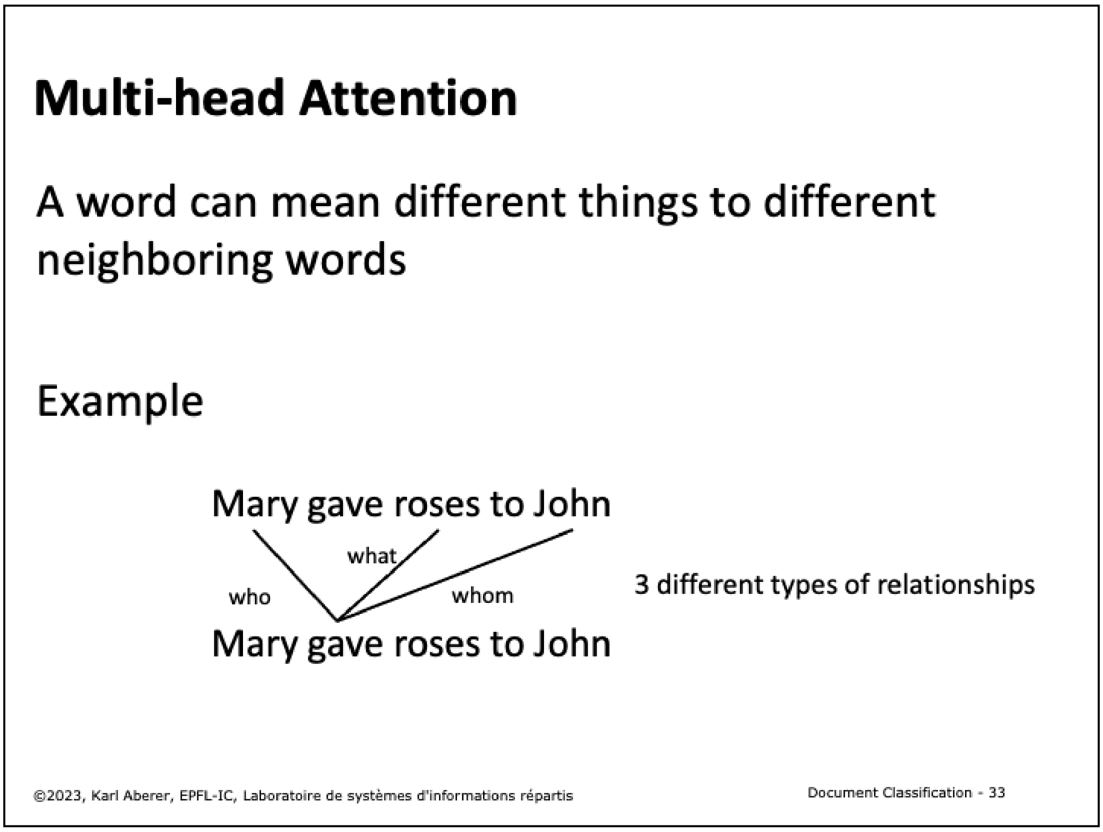
The softmax function can be sensitive to values with a very large range

- slows down learning or causes it to stop altogether
- average value of the dot product grows with the embedding dimension d
- scaling avoids inputs to the softmax function from growing too large

$$w'_{ij} = \frac{\boldsymbol{q}_i^t \boldsymbol{k}_j}{\sqrt{d}}$$

Note: $\|(c, \ldots, c)\|_2 = \sqrt{c^2 + \cdots + c^2} = \sqrt{d}c$

A second feature is scaling the vectors. This accounts for the fact that the softmax function becomes numerically unstable for values with a very large range. The problem is avoided by scaling the values, before feeding them into the softmax function. Scaling normalizes the output vectors to the same norm, independent of the dimension d chosen for the embedding vectors.



Multi-head Attention

A word can mean different things to different neighboring words

Example

Mary gave roses to John

who

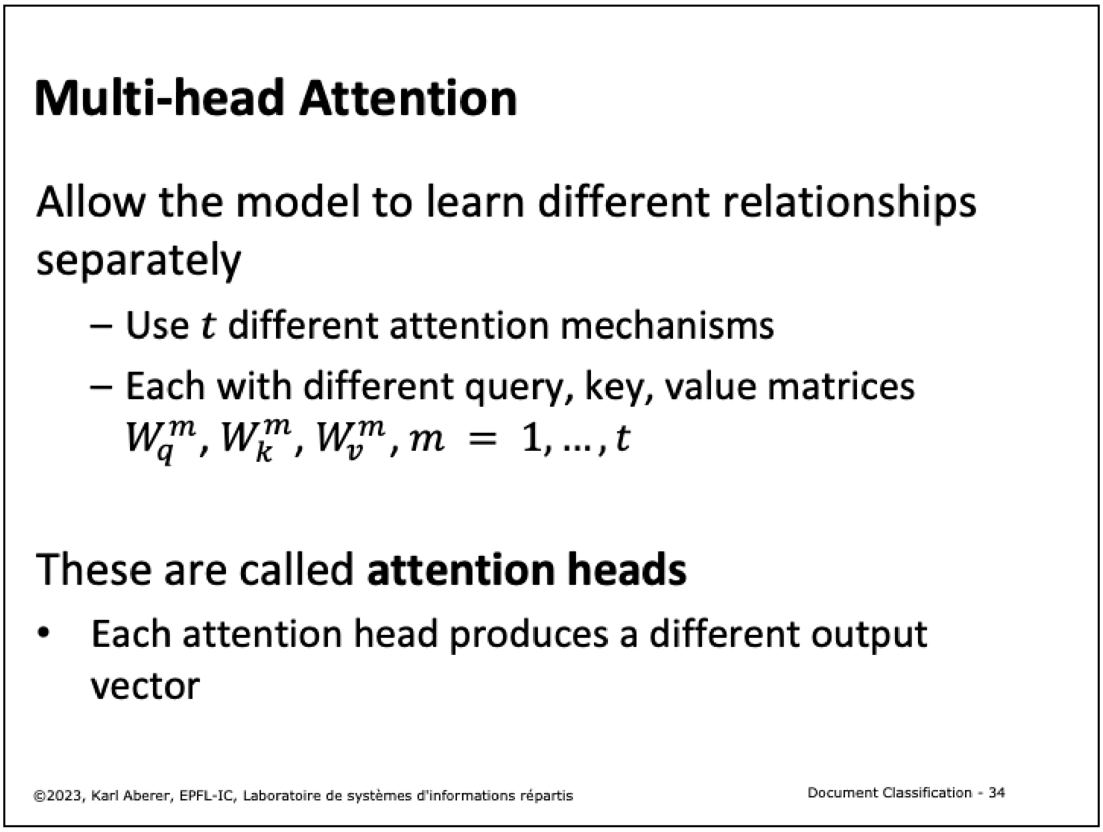
what

whom

Mary gave roses to John

3 different types of relationships

In text multiple, different, relationships exist between different words. Capturing all these relationship in a single representation of the input word is difficult. Therefore, transformer networks employ multiple self-attention mechanisms in parallel, each of which can learn a different kind of relationship. This is called multi-head attention.



Multi-head Attention

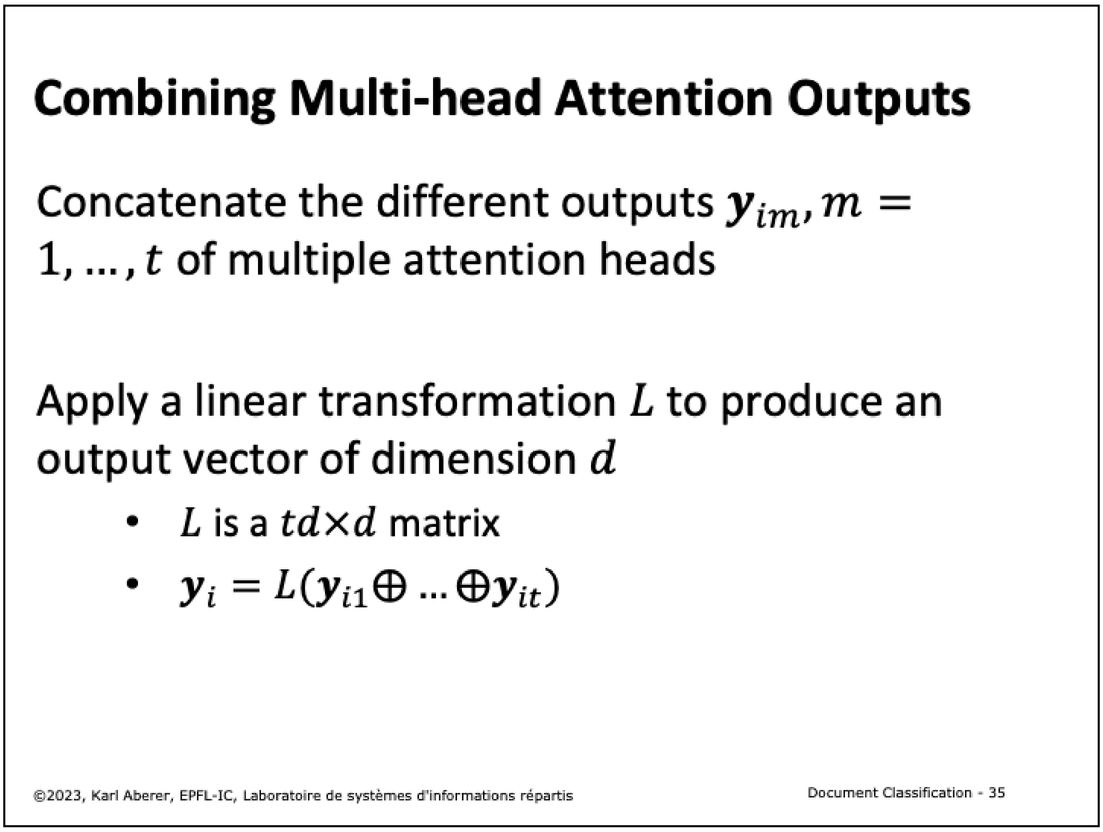
Allow the model to learn different relationships separately

- Use t different attention mechanisms
- Each with different query, key, value matrices $W_q^m, W_k^m, W_v^m, m = 1, \ldots, t$

These are called **attention heads**

- Each attention head produces a different output vector

When using multi-head attention, different query, key and value matrices are learnt for each attention head, allowing to represent different types of syntactic and semantic relationships.



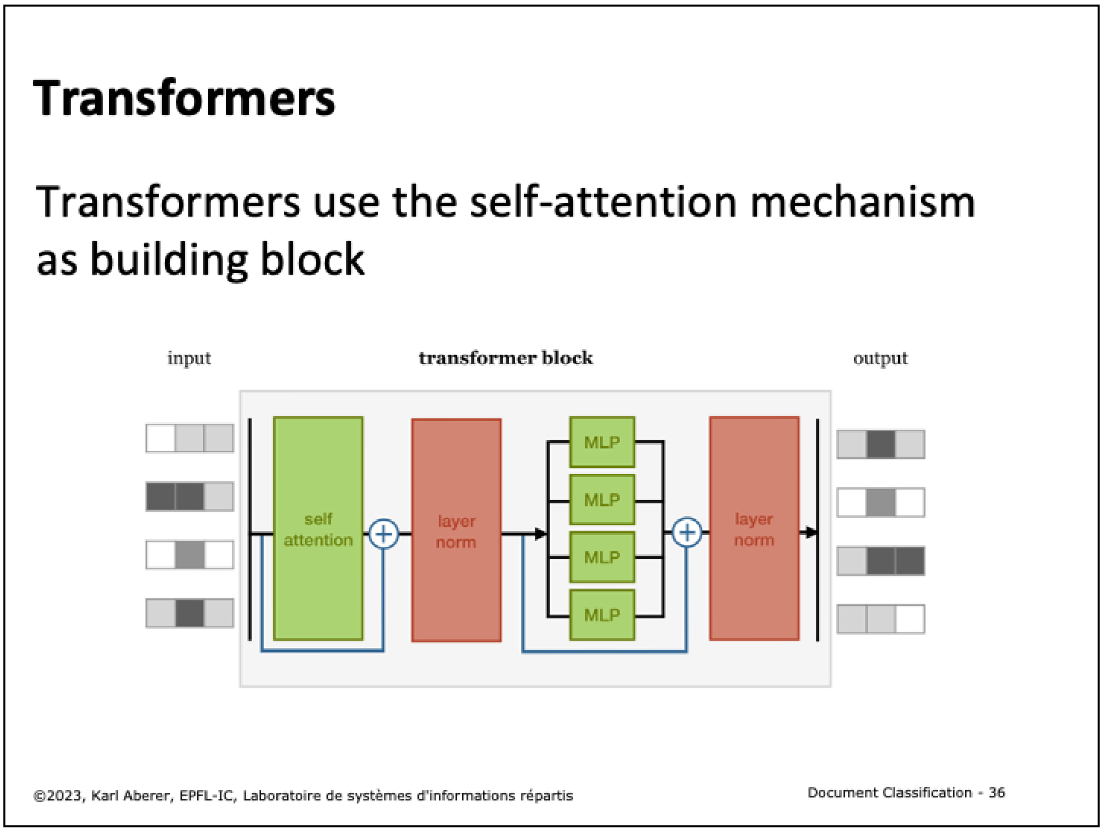
Combining Multi-head Attention Outputs

Concatenate the different outputs $\boldsymbol{y}_{im}, m = 1, \ldots, t$ of multiple attention heads

Apply a linear transformation L to produce an output vector of dimension d

- L is a $td \times d$ matrix
- $\boldsymbol{y}_i = L(\boldsymbol{y}_{i1} \oplus \ldots \oplus \boldsymbol{y}_{it})$

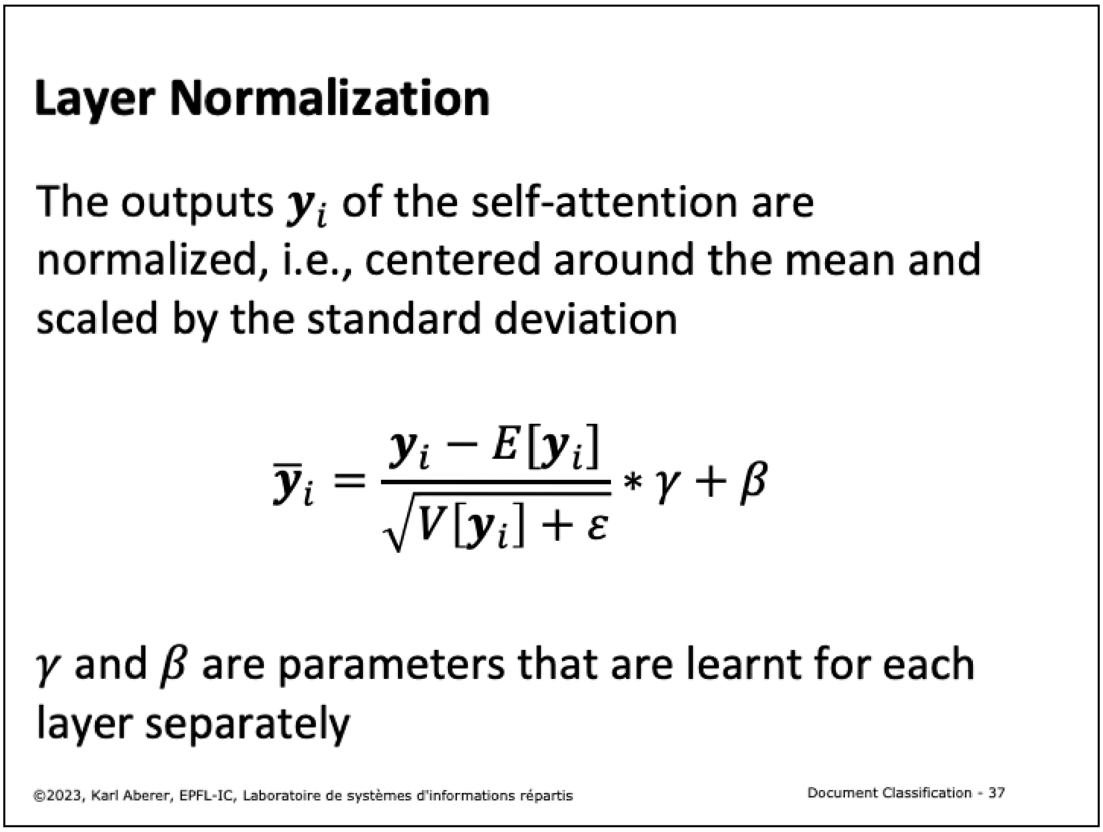
When using multi-head attention, multiple output vectors are produced. These are combined at the output, by applying a linear transformation that reduces the dimension of the concatenated output vectors, with is t*d, back to the standard embedding dimension d. The ⊕ symbol denotes concatenation of vectors.



Transformers

Transformers use the self-attention mechanism as building block

Using a multi-head attention mechanism allows now to establish what is called a transformer block. A transformer block takes the input vectors, applies the multi-head attention, and then performs further processing of the outputs. The additional processing steps are layer normalization and applying multi-layer perceptrons, a basic type of neural networks. We explain these steps next.



Layer Normalization

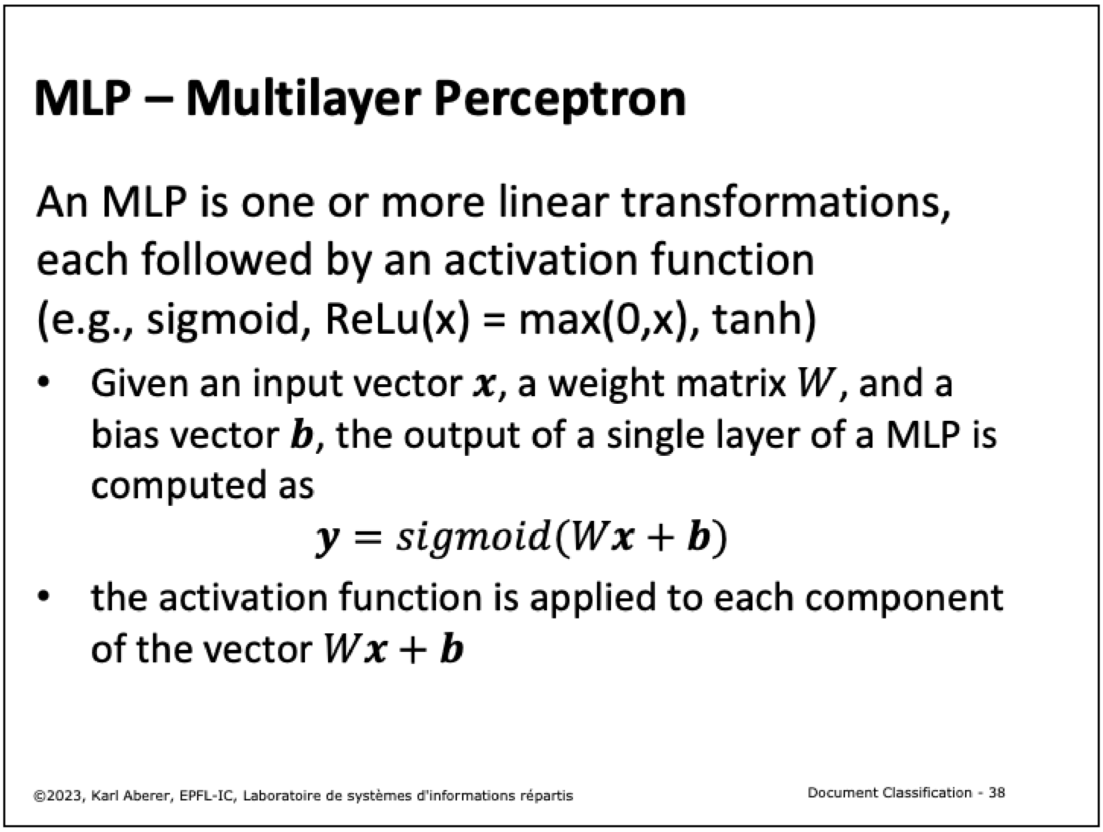
The outputs $\boldsymbol{y}_i$ of the self-attention are normalized, i.e., centered around the mean and scaled by the standard deviation

$$\overline{\boldsymbol{y}}_i = \frac{\boldsymbol{y}_i - E[\boldsymbol{y}_i]}{\sqrt{V[\boldsymbol{y}_i] + \varepsilon}} * \gamma + \beta$$

γ and β are parameters that are learnt for each layer separately

Normalization addresses the problem called internal covariate shift. Internal covariate shift refers to the change of distribution of values occurring during training of a neural network, going from one layer to the next. As the network learns and the weights are updated, the distribution of outputs of a specific layer in the network changes. This forces the higher layers to adapt to the drift occurring in lower layers, which slows down learning. Normalizing the inputs to the neural networks, eliminates this problem.

The gamma/beta values used for normalization are learnt for different layers separately but are the same for each layer.

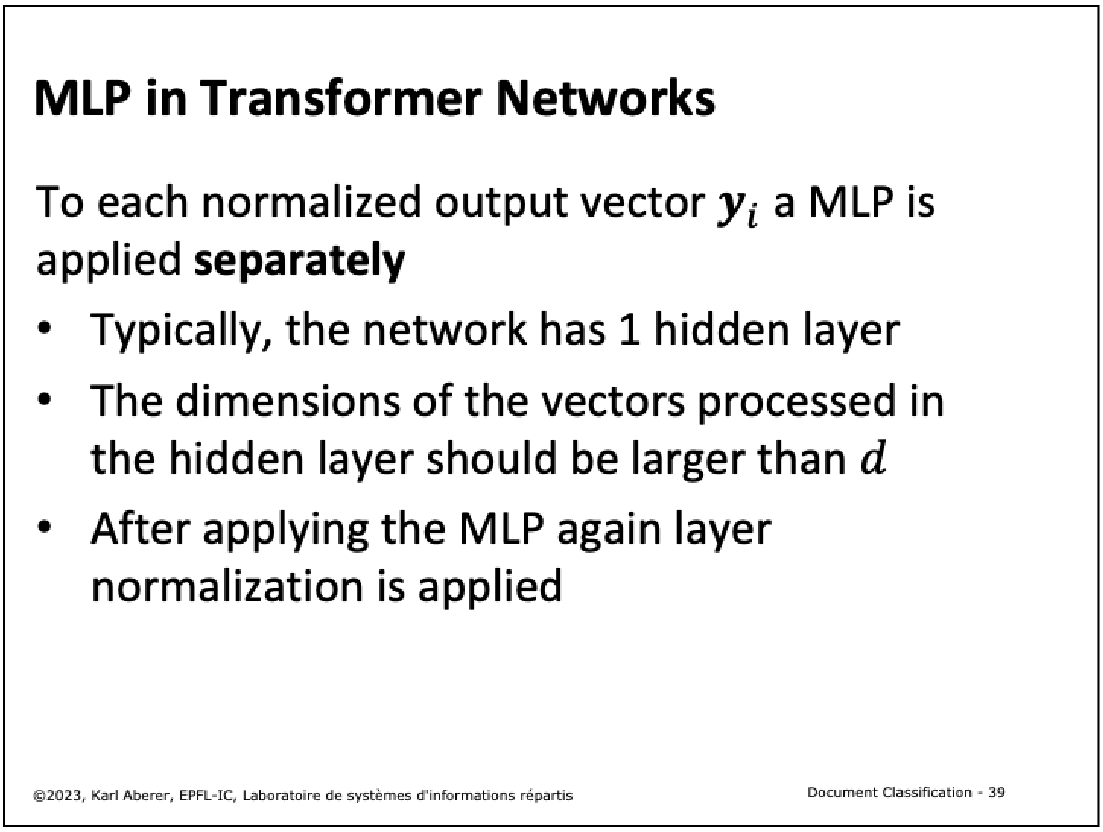


MLP – Multilayer Perceptron

An MLP is one or more linear transformations, each followed by an activation function (e.g., sigmoid, ReLu(x) = max(0,x), tanh)

- Given an input vector $\boldsymbol{x}$, a weight matrix W, and a bias vector $\boldsymbol{b}$, the output of a single layer of a MLP is computed as
$$\boldsymbol{y} = sigmoid(W\boldsymbol{x} + \boldsymbol{b})$$
- the activation function is applied to each component of the vector $W\boldsymbol{x} + \boldsymbol{b}$

MLPs are a simple form of neural networks. They apply first a linear transformation to the input vectors, using a weight matrix W and adding a bias vector b. To the components of the output vector an activation function is applied, to introduce non-linearity. In the original neural network models this function was a sigmoid or tanh function. In current models and standard transformer models it is the Relu function which has better numerical behavior. The Relu function is defined as relu(x) = max(0,x). The MLP is also often called Feed Forward Network (FFN).



MLP in Transformer Networks

To each normalized output vector $\boldsymbol{y}_i$ a MLP is applied **separately**

- Typically, the network has 1 hidden layer
- The dimensions of the vectors processed in the hidden layer should be larger than d
- After applying the MLP again layer normalization is applied

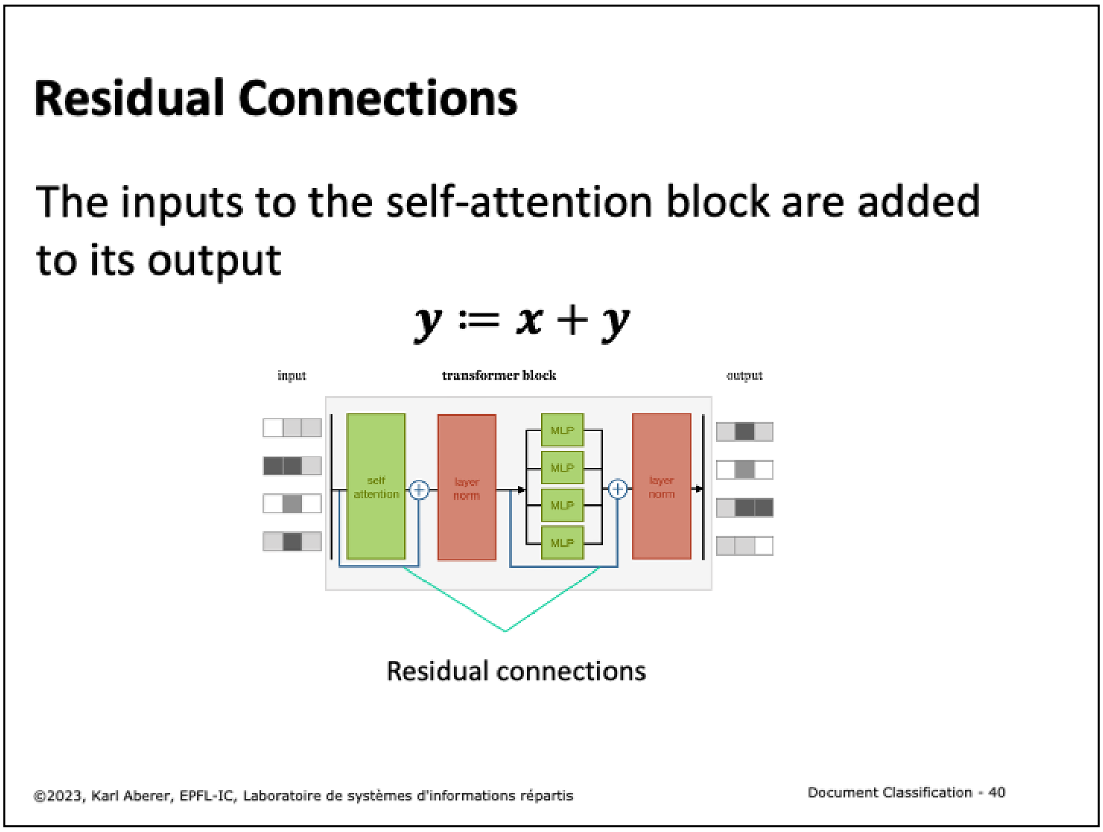
©2023, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Document Classification - 39

To each normalized output vector $\boldsymbol{y}_i$ an MLP is applied separately. The MLPs applied to each position have the same parameters. They consist of two linear transformations with an activation in between. Therefore, they have one hidden layer.

While the linear transformations are the same across different positions, they use different parameters from layer to layer. The dimensions of the vectors processed in the MLP, i.e., at the hidden layer, should be larger than d.

Multi-Head Attention (MHA) module and the Feed Forward Network (FFN) module play different roles in the Transformer architecture: The MHA allows the model to jointly attend to information from different subspaces, while the latter increases the non-linearity of the model. In original BERT, the ratio of the number of parameter between the MHA and FFN is always 1:2. To keep that ratio, the inner-layer in the MLP has dimensionality dx4.



Residual Connections

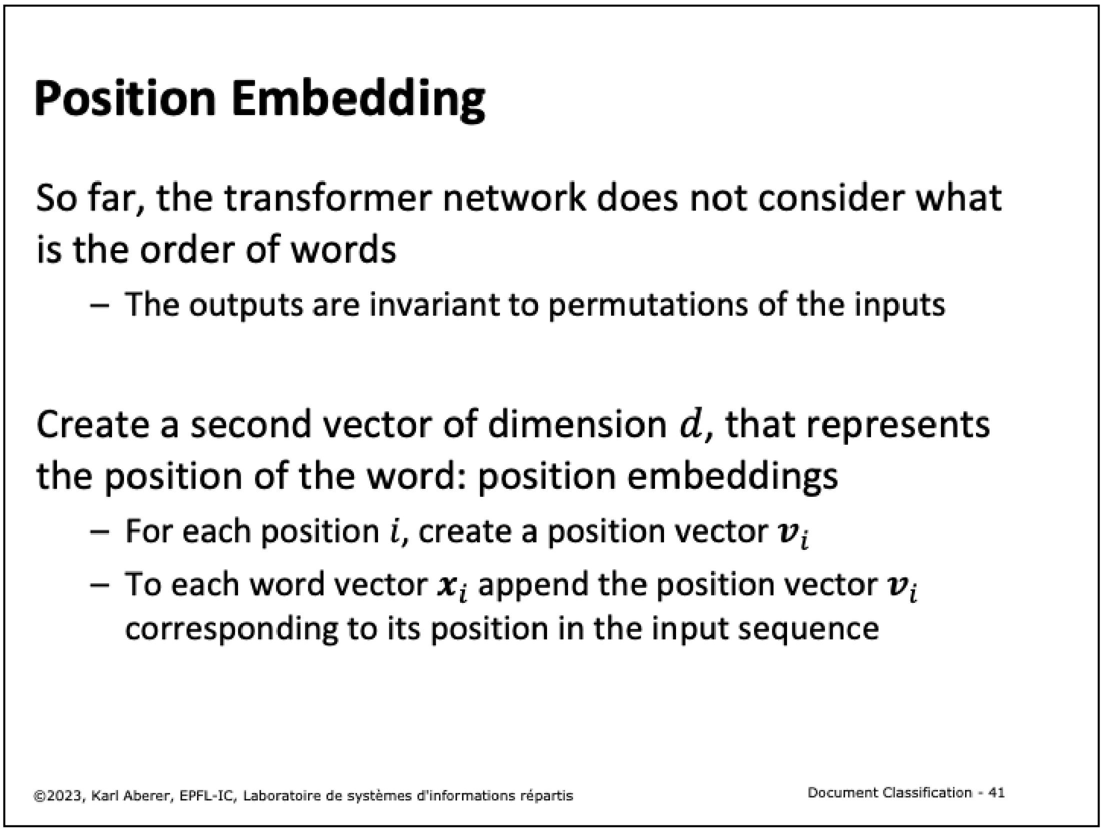
The inputs to the self-attention block are added to its output

$$y \coloneqq x + y$$

Residual connections

The inputs to the self-attention block are added to its output through so-called residual connections. Similarly, also for the MLPs.

Residual connections (also called skip connections) are used to allow gradients during backpropagation to flow through a network directly, without passing through non-linear activation functions. Non-linear activation functions, by nature of being non-linear, cause the gradients to explode or vanish (depending on the weights). This results in the problem of vanishing gradients which make it more difficult to learn from the feedback obtained through backpropagation.



Position Embedding

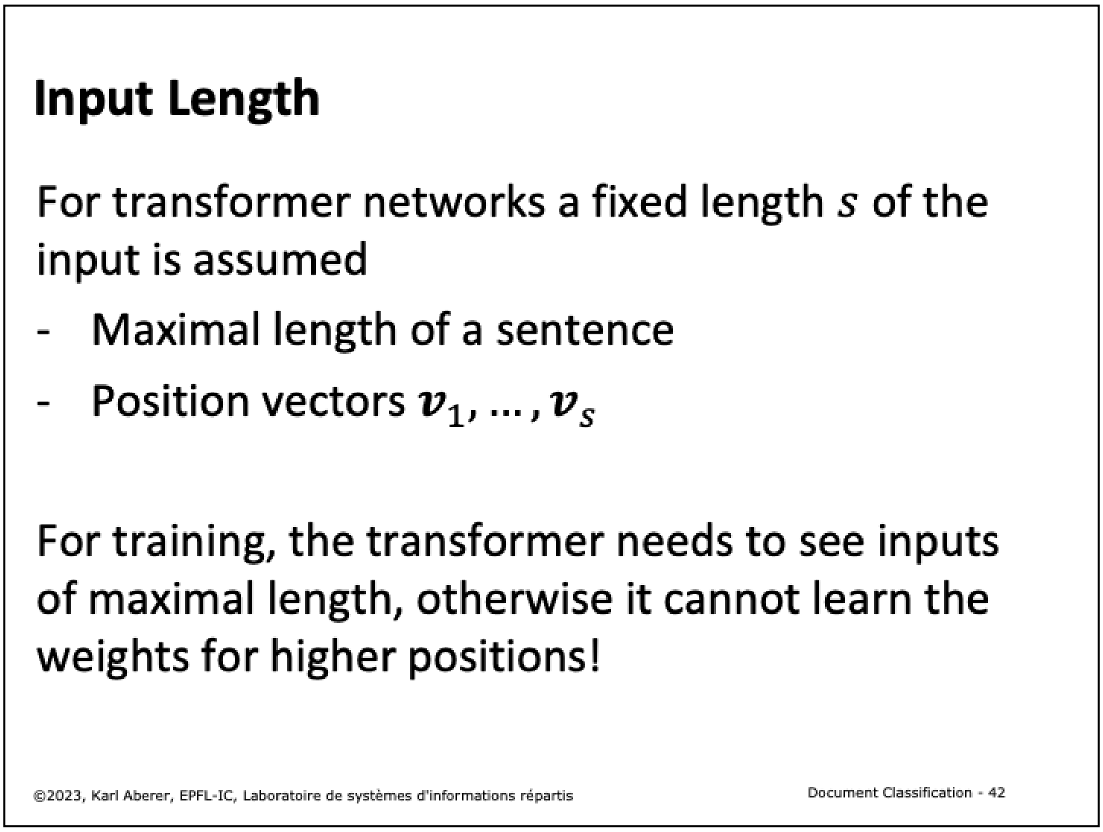
So far, the transformer network does not consider what is the order of words

- The outputs are invariant to permutations of the inputs

Create a second vector of dimension d, that represents the position of the word: position embeddings

- For each position i, create a position vector $\boldsymbol{v}_i$
- To each word vector $\boldsymbol{x}_i$ append the position vector $\boldsymbol{v}_i$ corresponding to its position in the input sequence

The way the transformer network is defined so far is insensitive to the order of the inputs. In other words, any permutation of the inputs would generate the same relations, which is naturally not how language works. Therefore, in addition the position of words in a sentence needs to be provided as input to the transformer network. The way this is achieved is by learning a position vector, which is the representation of an input position. Each word vector is concatenated with the position vector corresponding to the word's position within the sentence.



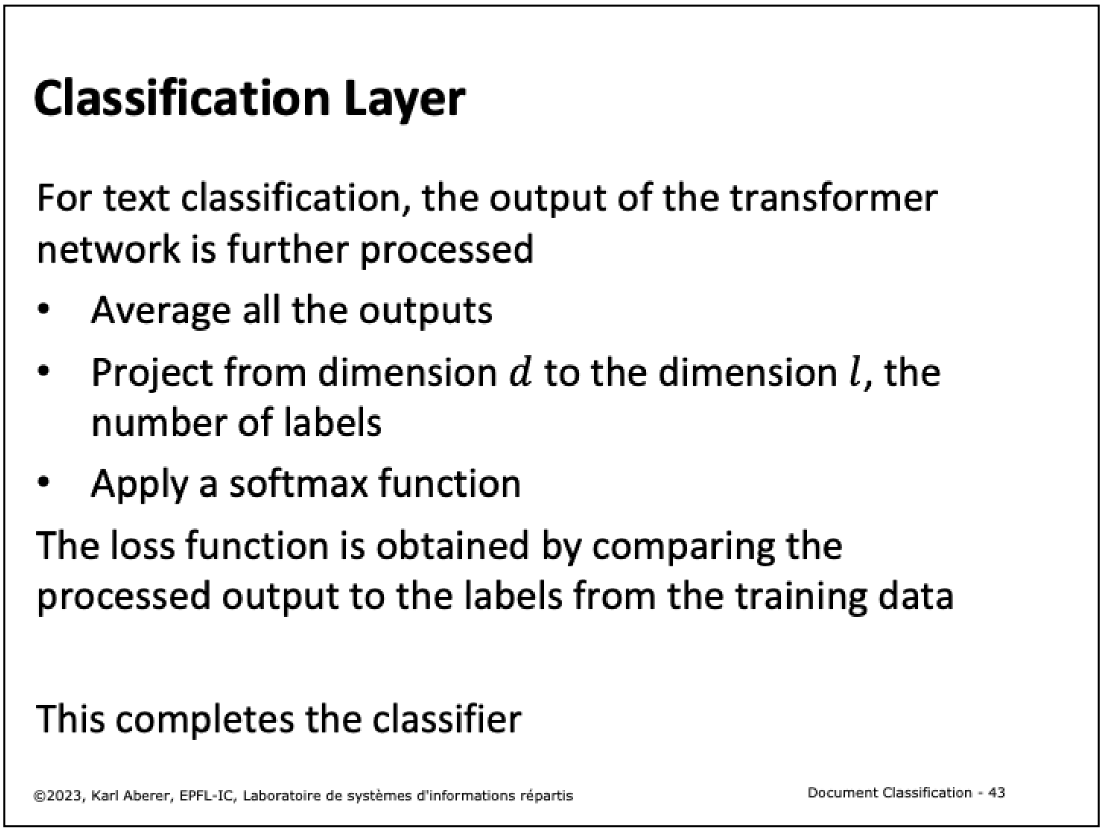
Input Length

For transformer networks a fixed length s of the input is assumed

- Maximal length of a sentence
- Position vectors $\boldsymbol{v}_1, \dots, \boldsymbol{v}_s$

For training, the transformer needs to see inputs of maximal length, otherwise it cannot learn the weights for higher positions!

In order to learn position vectors, the length of the sequences (sentences) needs to be fixed, or in other words a maximal sequence length needs to be specified. For learning the parameters this requires that the training data needs to have sequences that have this maximal length, otherwise the position vectors or the higher positions cannot be learnt.



Classification Layer

For text classification, the output of the transformer network is further processed

- Average all the outputs
- Project from dimension d to the dimension l, the number of labels
- Apply a softmax function

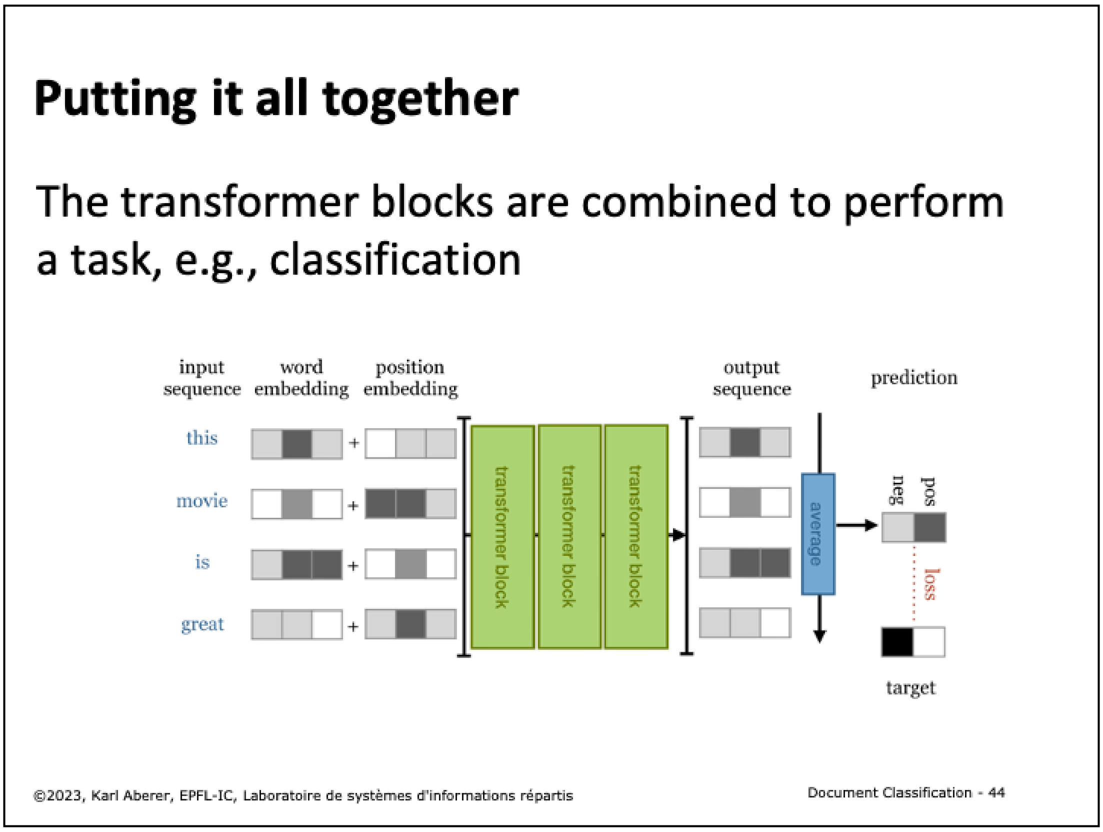
The loss function is obtained by comparing the processed output to the labels from the training data

This completes the classifier

When using a transformer network for text classification, the output vectors are used to produce a prediction for the classification label. In order to do this, a standard approach is to average the output vectors, project them using a linear mapping the the dimension of the label space and apply a softmax function to obtain values that can be interpreted as probabilities.

The predicted values are in the training phase compared to the labels from the training data, and a loss function is calculated from the difference. This is exactly same approach we have taken with our first simple text classifier using a self-attention mechanism.

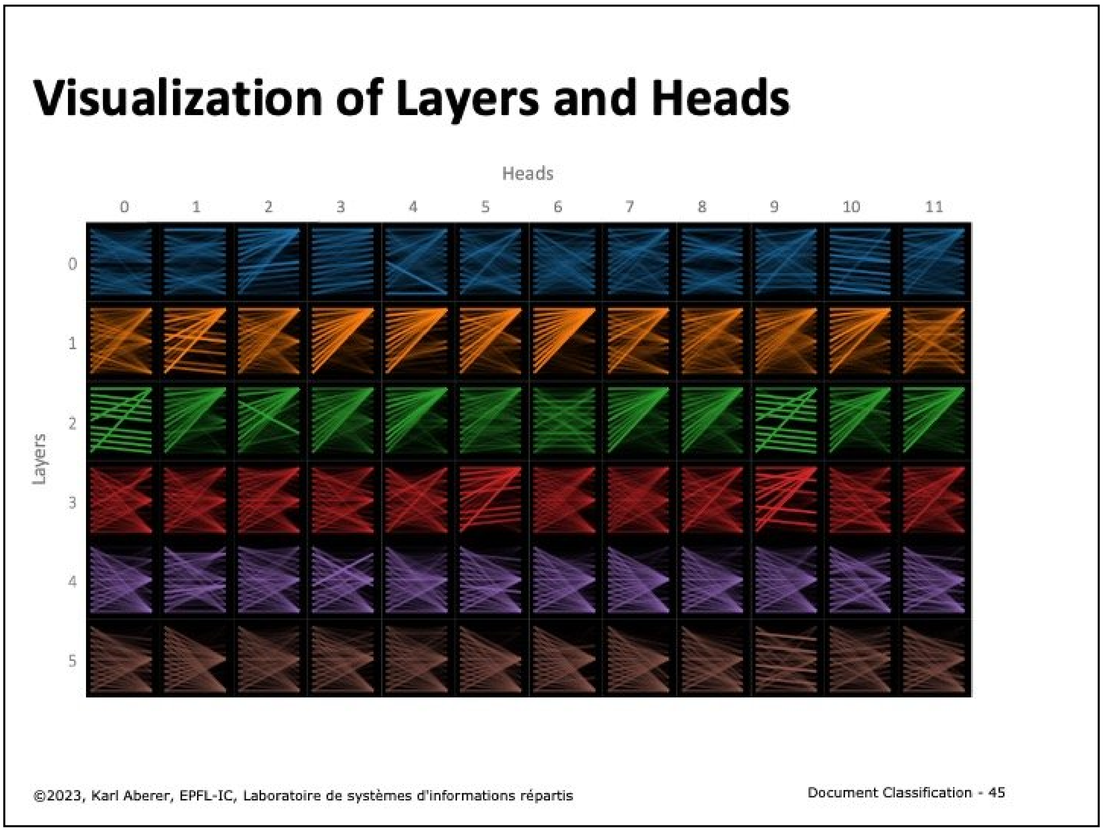
Using the loss function the parameters can then be learnt using backpropagation. The backpropagation is implemented in frameworks such as pytorch that allows to automatically derive the gradients for the complex function defined by the transformer network.



Putting it all together

The transformer blocks are combined to perform a task, e.g., classification

This diagram illustrates the overall architecture of a typical transformer network. We see that multiple transformer blocks are applied in sequence (each of which contains a multi-head self-attention mechanism).



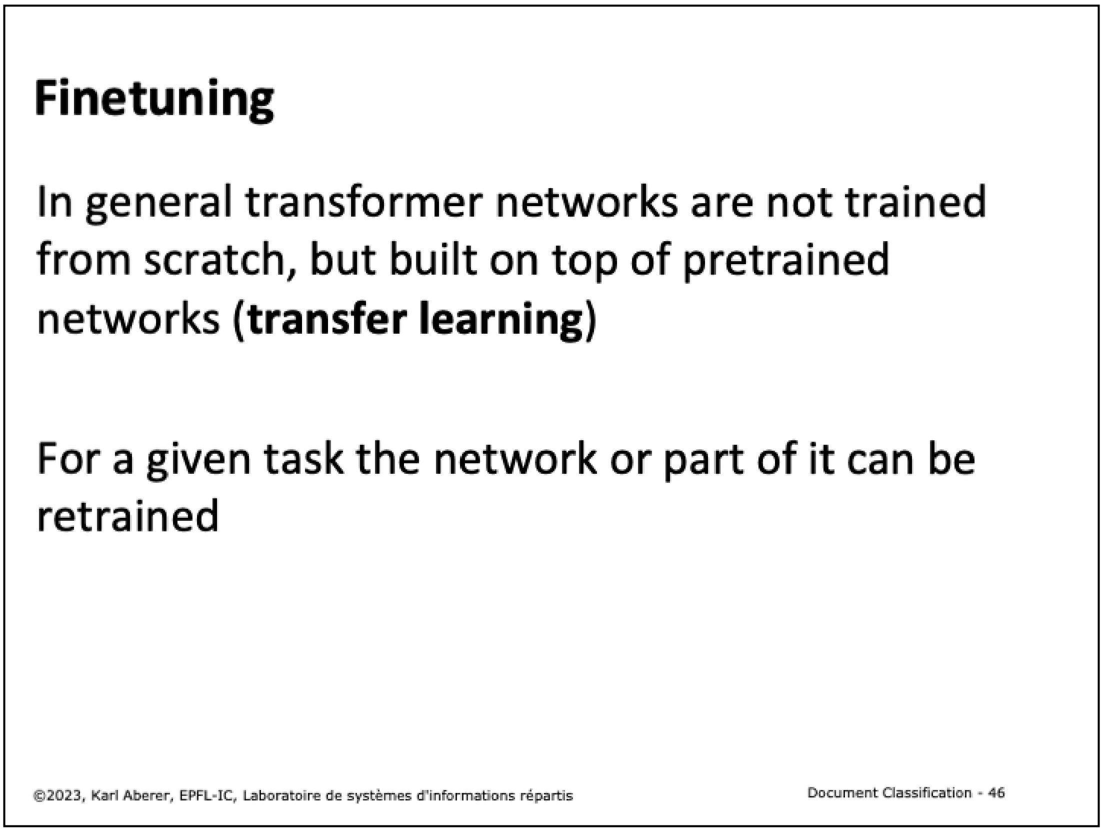
Visualization of Layers and Heads

©2023, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Document Classification - 45

From:

https://towardsdatascience.com/deconstructing-bert-part-2-visualizing-the-inner-workings-of-attention-60a16d86b5c1



Finetuning

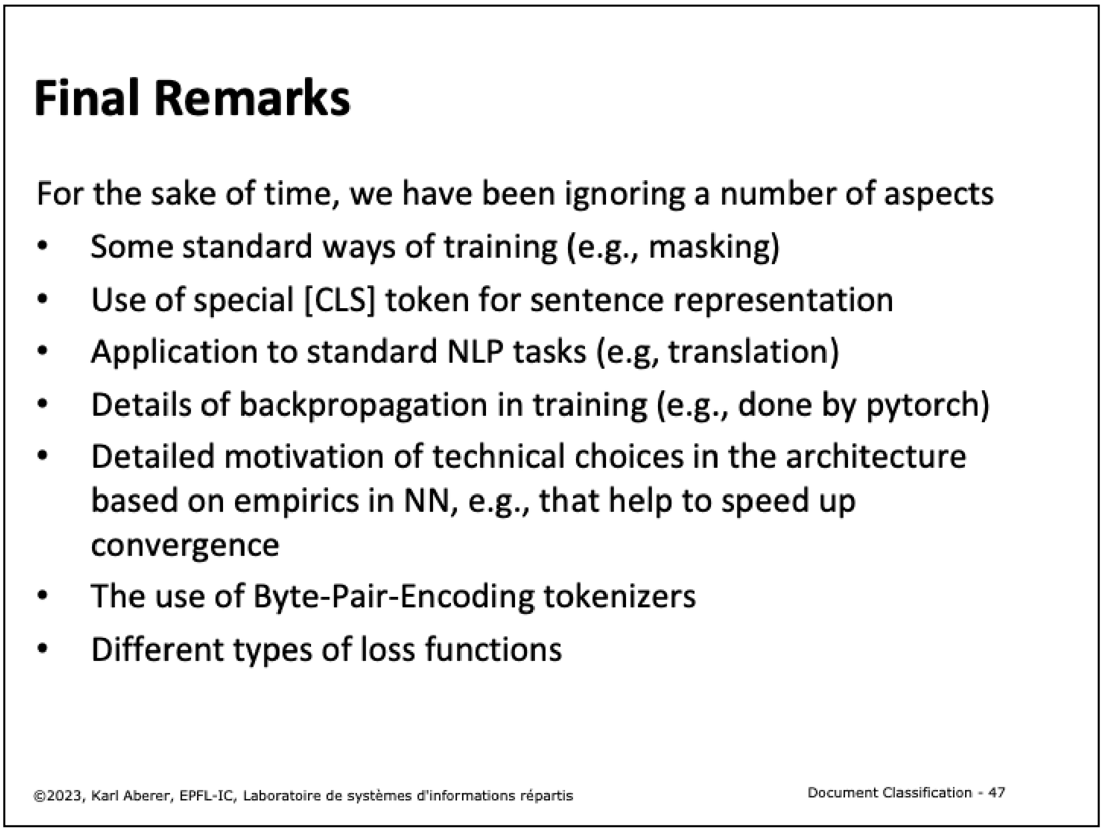
In general transformer networks are not trained from scratch, but built on top of pretrained networks (**transfer learning**)

For a given task the network or part of it can be retrained

In practice, given the very large number of parameters in transformer networks, they are usually not trained from scratch, in particular, when using them to build a document classifier. Instead, pretrained networks are used. For example, Huggingface is a platforms that provides a large number of pretrained models. The pretrained network is completed with a classification layer and retrained using the training data for classification. Retraining a network is called finetuning.

This approach has the advantage that general properties of natural language are already encoded in the network parameters and need not to be learnt from scratch every time. This training is in general hugely expensive.

There exist different variants of how fine-tuning is performed, depending whether all are only a part of the parameters of the pre-trained transformer network are modified during retraining.

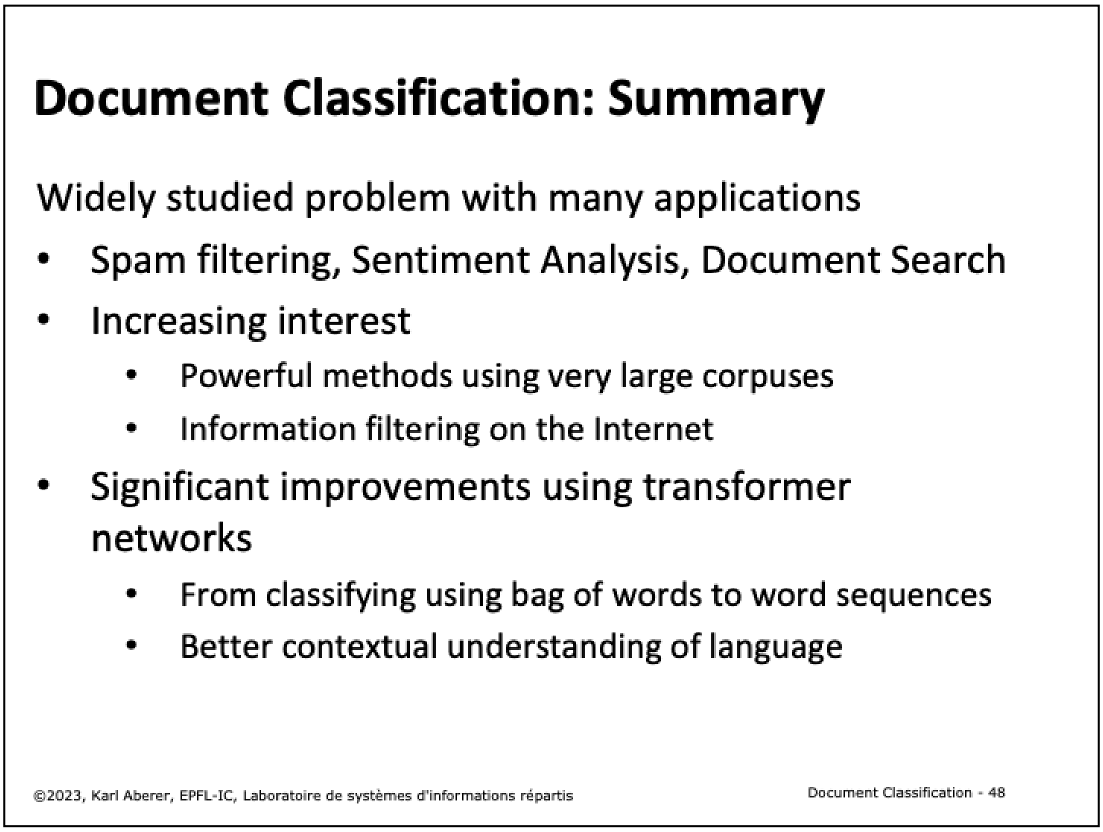


Final Remarks

For the sake of time, we have been ignoring a number of aspects

- Some standard ways of training (e.g., masking)
- Use of special [CLS] token for sentence representation
- Application to standard NLP tasks (e.g, translation)
- Details of backpropagation in training (e.g., done by pytorch)
- Detailed motivation of technical choices in the architecture based on empirics in NN, e.g., that help to speed up convergence
- The use of Byte-Pair-Encoding tokenizers
- Different types of loss functions

This introduction to transformer network aimed at highlighting the basic principles underlying their architecture. Many details related to their training, application, and the implementation of deep neural networks have not been discussed, as they are beyond the scope of this lecture.

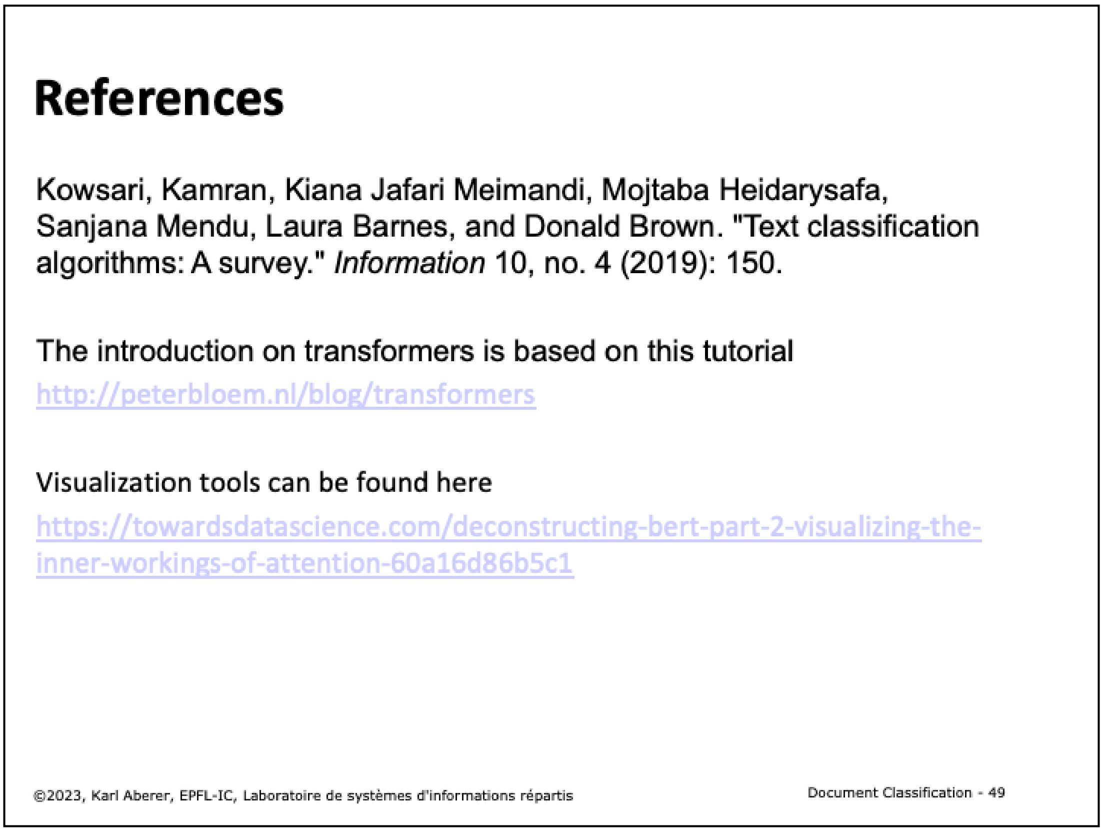


Document Classification: Summary

Widely studied problem with many applications

- Spam filtering, Sentiment Analysis, Document Search
- Increasing interest
  - Powerful methods using very large corpuses
  - Information filtering on the Internet
- Significant improvements using transformer networks
  - From classifying using bag of words to word sequences
  - Better contextual understanding of language

Document classification is an area of high interest, both because it is required in a growing number of application domains and because at the same time the classification methods are becoming increasingly powerful due to the availability of very large document corpuses, the growing computational power available and the recent developments around transformer networks.



References

Kowsari, Kamran, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. "Text classification algorithms: A survey." *Information* 10, no. 4 (2019): 150.

The introduction on transformers is based on this tutorial
http://peterbloem.nl/blog/transformers

Visualization tools can be found here
https://towardsdatascience.com/deconstructing-bert-part-2-visualizing-the-inner-workings-of-attention-60a16d86b5c1