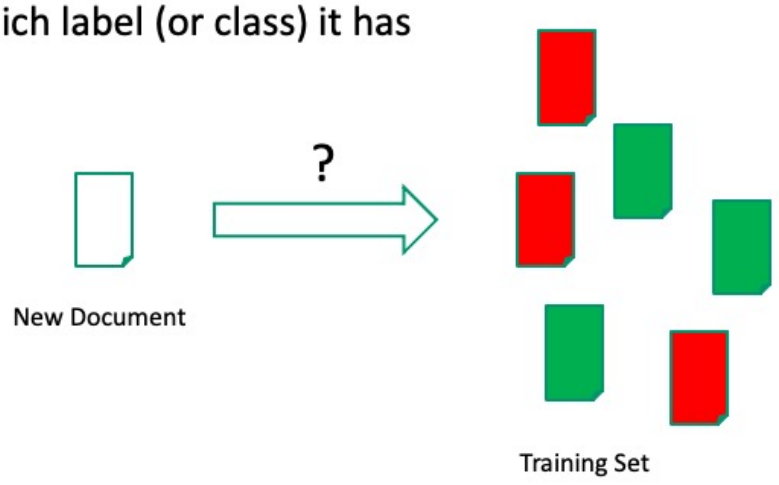
# 6. DOCUMENT CLASSIFICATION

## Document Classification

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







Document classification is a special case of classification where the objects to be classified are (unstructured documents). This task has huge importance in many application areas, such as spam filtering, sentiment analysis, document filtering etc. and has therefore been studied as a specific classification problem in very much detail.

## Document Classifiers

Features

- Words of the documents
  - bag of words
  - document vector
- More detailed information on words
  - Phrases
  - Word fragments
  - Grammatical features
- Any metadata known about the document and its author




An important question in document classification is the choice of features. A wide variety of possible features can be derived from text documents, many of which correspond to features used in document representation of information retrieval.

# 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…

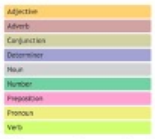
*Bag of words*: {more, than, visitors, attend, …}

*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

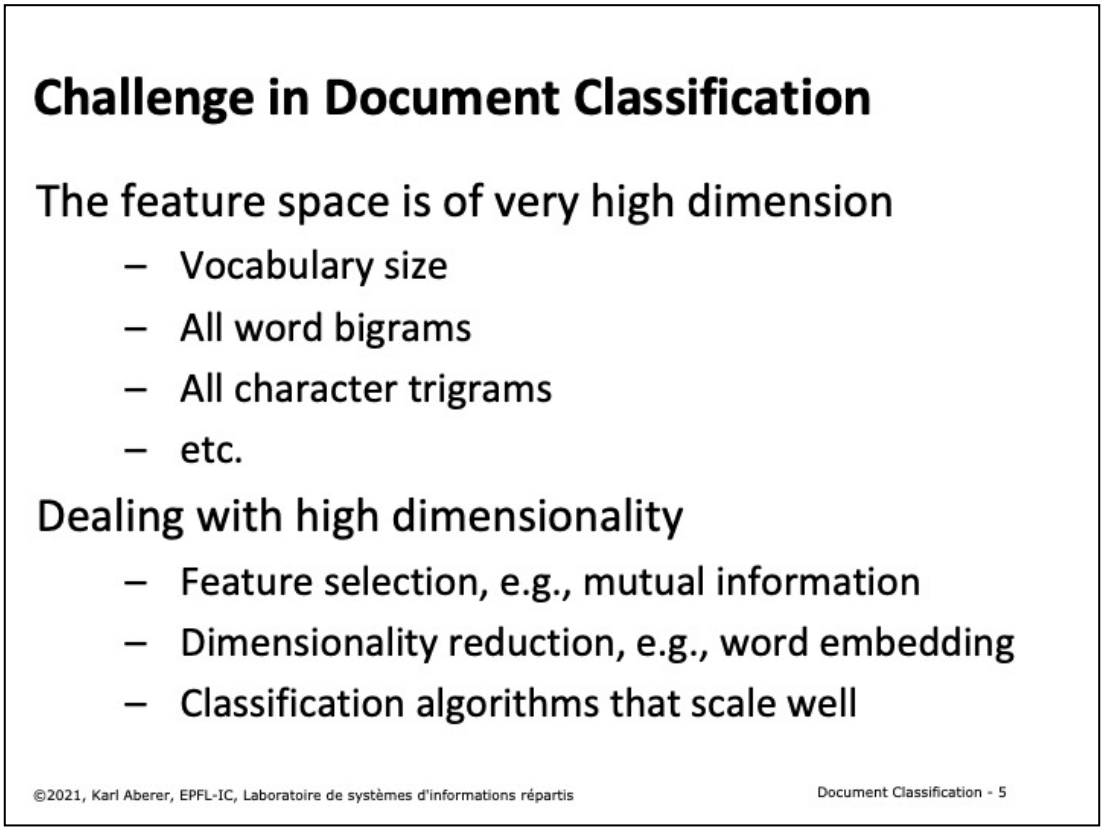



http://parts-of-speech.info/

*Author*: @EPFL_SPS




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




A problem that is characteristics for document classification is the fact that the feature space for documents can be huge. From information retrieval we know that the vocabulary size of a large document collection can grow to significant sizes, reaching millions of terms. When considering more features, such as word bigrams to capture phrases or n-grams in words, the size of the features space further explodes.

In order to deal with the high dimensionality of the feature space, different routes have been explored. A first is to perform feature selection, using e.g. mutual information, and retain only features that are specific for classification (with all the potential drawbacks discussed earlier). A second approach is to use dimensionality reduction methods, such as word embeddings or LSI, and represent then documents in the much lower dimensional concept space. Finally, one may also focus on using classification algorithms that scale well with dimensionality.

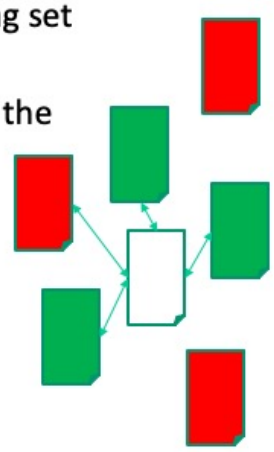
## Simple Method: k-Nearest-Neighbors (kNN)

**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

If $k$ large: $\frac{\#C}{k}$ estimates $P(C|D)$, the probability that the document has indeed class $C$







Actually, a very straightforward method for document classification, that scales well with dimensionality, is inspired by information retrieval. In the kNN method, the top k most similar documents according to the vector space model are retrieved, and then the majority label is used as the classification of the document. Strictly speaking with this method no classifier is learnt, but the classifier consists of the vector space representation of the whole document collection, so learning is trivial in that case. The method works in practice well. One critical choice is the parameter k, that determines how large the neighborhood is that is taken into account, and can be understood as the complexity of the model. In line with what we have seen on the dependency of bias and variance on model complexity. In the context of kNN a model is complex when k is small, since for every document very different sets of neighbors are chosen. Correspondingly it holds that for small k (complex model) we have low bias, but high variance and for large k we have high bias and low variance.

## 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 we can derive the probability of a document belonging to a class, from two estimators (probabilities) that have to be learnt from the training data. For this method to work, words not occurring in the training set, but in the test set, need to have non-zero probabilities.

## 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 $C$?

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

**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 the model we have to compute for each class how likely a word is to appear in that class and we have to compute the relative frequencies of the classes, i.e. the probability of a class label to occur. Note that in the estimation for P(w|C) we add +1 to the word counts. This is called Laplacian smoothing and is necessary to avoid to assign zero probability to terms that do not occur in the training set, to avoid that subsequently a document that does contain such a term is considered as impossible when using the classifier. The reasoning is the same why smoothing has been used in probabilistic information retrieval.

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

## 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 $c$ (skipgram model)

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





For document classification word embeddings can be used in different ways. One obvious way would be to use word embedding vectors as word representation, and to represent then documents as an aggregate of those vectors to obtain a low-dimensional feature space. However, another possibility is to use the word embedding approach in a more direct way, which has resulted in a classification algorithm that has been recently developed at Facebook and is known as Fasttext.

## 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 mode




In this approach to document classification the idea of creating word embeddings is slightly modified. Instead of trying to predict the a word from its context, for document classification one tries to predict the class label from a text fragment, e.g. a paragraph. Technically speaking, the two problems are equivalent, as in both cases we try to predict some word from a set of words. Thus the same method for learning the word embedding can be applied, using stochastic gradient descent. Once the model is learnt, as with Naïve Bayes the class label with the highest probability to occur is chosen as prediction for the document class. If there is not enough training data, instead of learning the word vectors $v_w$ also pre-trained vectors from other document collections can be used.

## Adapting the Model for Classification

Representation

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

Loss function

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

## Implementation of the Model

Same approach as for standard word embedding

- Direct SGD, hierarchical softmax, approximation with negative sampling
- Both using skipgram and CBOW model

Training data encodes labels with text

__label1__, text1

__label2__, text2

…

Prediction of class label using softmax ($P_\theta$)

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

A. for none
B. for kNN
C. for NB
D. for fasttext

## Transformer Models

A new architecture for machine learning models

- Initially conceived for machine translation

Extending standard neural network architectures

- Uses basic NN and backpropagation
- 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**




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 a number of natural language processing tasks. One of the best known models of this class is BERT.

## Self-attention

Given input vectors $x_1, \ldots, x_s$ and output vectors $y_1, \ldots, y_s$ of dimension $k$

- Input vectors are learnt from data

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

$$y_i = \sum_j w_{ij} x_j \,, \sum_j w_{ij} = 1$$

Each output vector is a weighted average of the input vectors




The basic self-attention mechanism looks very straightforward. It transforms a set of input vectors, e.g. representing the words of a sentence, to a set of output vectors of the same length. The self-attention mechanisms realizes a linear mapping from the input vectors to the output vectors. One can consider this mapping as computing weighted averages of the input vectors.

## Self-attention Weights

The weights $w_{ij}$ are not simply parameters (that are learnt), but a function of $x_i$ and $x_j$

- Different options for defining this function exist

Simplest case $w'_{ij} = x_i \cdot x_j = x_i^t x_j$

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

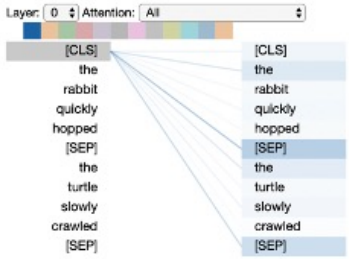



However, the weights used in the linear mapping are not arbitrary values (that would be learnt from training data), but depend themselves on the input vectors. In the simplest case, this dependency would be given as the scalar product of two input vectors $x_i$ and $x_j$ producing the weight $w_{ij}$, after applying a softmax functions, such that the weights add up to 1 for one dimension i. This means the input vectors will be learnt in a way that they encode the relationship between two different items in the input.

## Intuition

Assume input vectors correspond to words in sentences

- Assume the input vectors model some relation among those words, e.g., word $x_i$ is the subject related to a verb $x_j$
- Then the vectors of $x_i$ and $x_j$ will be similar to produce a large weight, whereas the others will be dissimilar

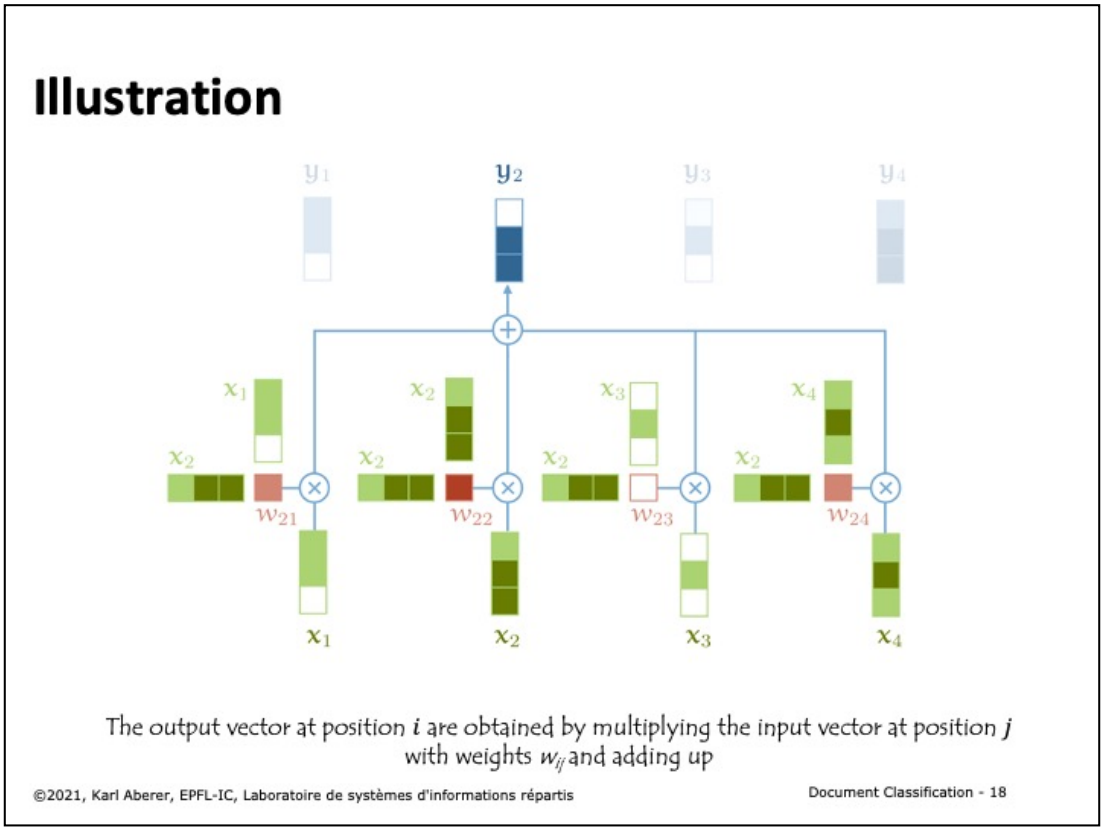






The visualization can be found on:

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




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 derived from the input vectors using an embedding matrix (that will be learnt), and a sentence is converted to a sequence of vectors.

## 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: $y = \frac{1}{s}\sum_{i=1}^{s} y_i$
4. Project to class labels:
   $c' = Cy, C$ is a $k\ x\ l$ matrix
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 would be averaged, and then projected to a vector that has the dimension 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.

## 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.

## More “Tricks”

Standard transformer models introduce a number of additional features

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




The basic mechanism that we described so far is in practice extended by a number of additional features that are the result of significant (empirical and theoretical) investigation on the design of transformer models.

## Queries, Keys, Values

The representation $x_i$ of a token plays three different roles:

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

To facilitate the learning task we apply different linear transformations to $x_i$ for each of the roles




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

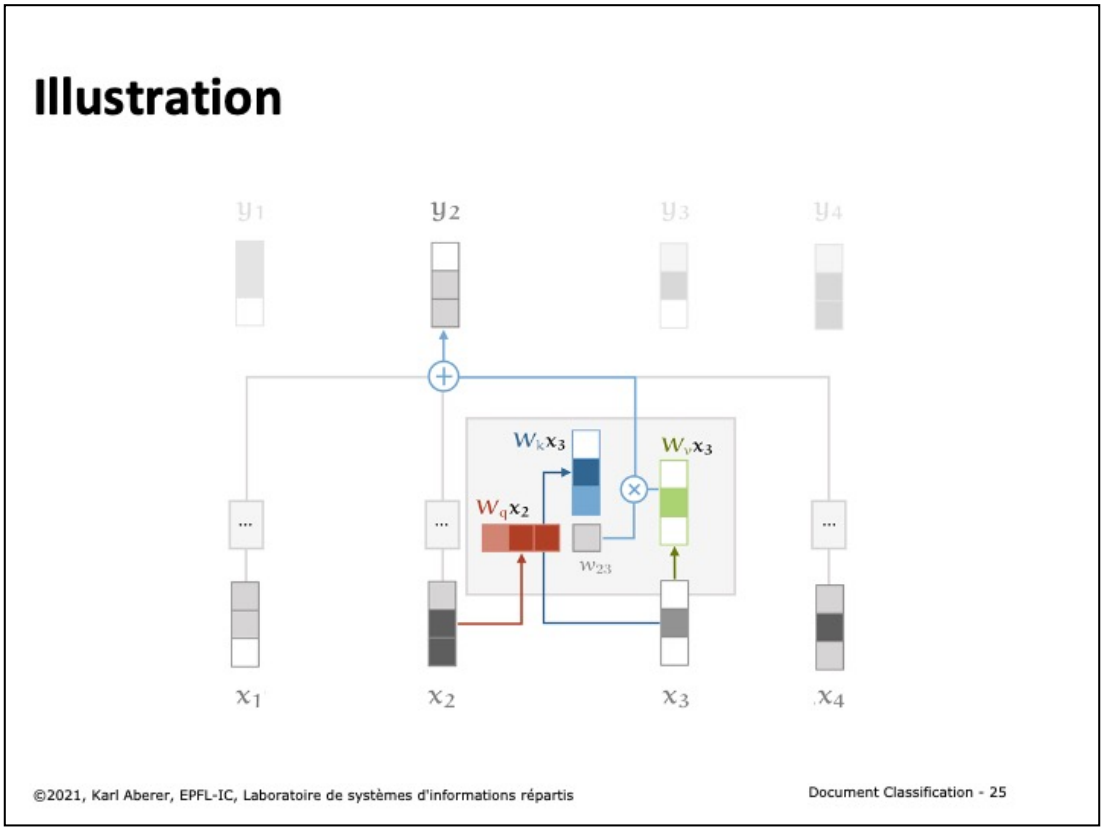
Linear transformations $W_q, W_k, W_v$
($k \times k$ matrices)

$$q_i = W_q x_i,\ k_i = W_k x_i,\ v_i = W_v x_i$$
$$w'_{ij} = q_i^t k_j$$
$$w_{ij} = \text{softmax}(w'_{ij})$$
$$y_i = \sum_j w_{ij} x_j$$

Note: the linear transformations are independent of the vectors




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




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 very large input values

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

$$w'_{ij} = \frac{q_i^t k_j}{\sqrt{k}}$$

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

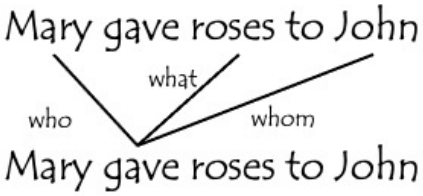



A second feature is scaling the vectors. This accounts for the fact that the softmax function does not work well with large input values. The scaling essentially normalizes the output vectors to the same norm, independent of the dimension k chosen for the embedding vectors. The problem is avoided by scaling the values, before feeding them into the softmax function.

## Multihead Attention

A word can mean different things to different neighboring words

Example




3 different types of relationships




When analyzing text, multiple, different, relationships exist between different words. Capturing all these relationship in a single representation of the input word is not feasible. Therefore transformer networks employ multiple self-attention mechanisms in parallel, each of which can learn a different kind of relationship. This is called multihead attention.

## Multihead 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




Using multihead attention, different query, key and value matrices are learnt for each attention head, allowing to represent different types of relationships.

## Combining Multi-head Attention Outputs

Concatenate the different outputs $y_{im}$ of multiple attention heads

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

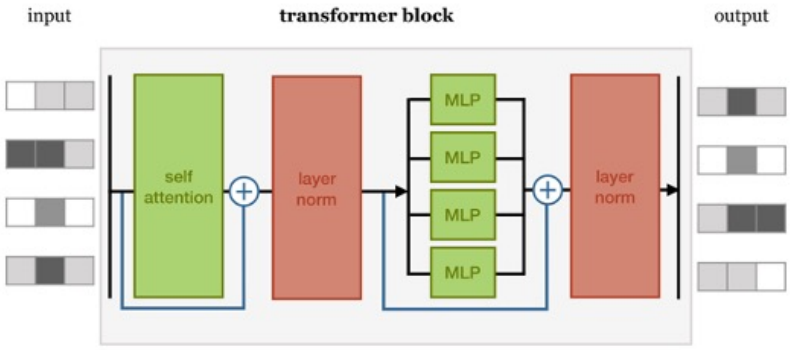
- $L$ is a $tk \times k$ matrix
- $y_i = L(y_{i1} \oplus \ldots \oplus y_{it})$




When using multihead attention, different 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*k, back to the standard embedding dimension k. The $\oplus$ symbol denotes concatenation of vectors.

## Transformers

### Transformers use the self-attention mechanism as building block





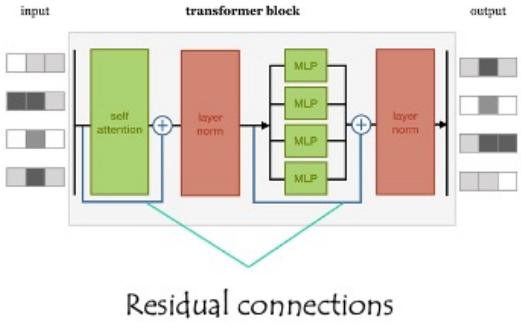


Using a multi-head attention mechanism allows now to establish what is called a transformer block. A transformer block takes the input vectors, applies the multi-head attention, and then performs further processing of the outputs. The additional processing steps are layer normalization and applying multi-layer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

## Residual Connections

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

$$x + y$$







The inputs to the self-attention block are added to its output through so-called residual connections.

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). Residual connections form conceptually a 'bus' which flows right the way through the network, and in reverse, the gradients can flow backwards along it too.

## Layer Normalization

The outputs $y_i$ of the self-attention are normalized, i.e., centered around the mean and scaled by the standard deviation

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

$\gamma$ and $\beta$ are parameters that are learnt for each layer separately




The outputs of the self-attention are normalized, i.e. centered around the mean on scaled by the standard deviation.

Normalization helps with the problem called internal covariate shift. Internal covariate shift refers to changing distribution of values during training occurring within a neural network, i.e. going from one layer to the next. This happens because, 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 that drift, which slows down learning. After normalizing the input in the neural network, we don't have to worry about the scale of input features being extremely different.

The gamma/beta values are learnt for each layer separately.

## MLP – Multilayer Perceptron

To each normalized output vector $y_i$ a MLP is applied **separately**

- A MLP is a simple neural network
- The dimensions of the vectors processed in the MLP should be larger than $k$
- After applying the MLP again a residual connection and layer normalization is applied




To each normalized output vector $y_i$ a MLP is applied separately. The MLP is also called Feed Forward Network (FFN).

The same MLP is applied to each position separately and identically. It consists of two linear transformations with a ReLU activation in between. 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 should be larger than k-

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 kx4.

## Explaining MLP

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

- Given an input vector $x$, a weight matrix $W$, and a bias vector $b$, the output of a single layer of a MLP is computed as

$$y = sigmoid(Wx + b)$$

- the activation function is applied to each component of the vector




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 compenents 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 simply defined as relu(x) = max(0,x)

## Position Embedding

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

- The outputs are invariant to permutations of the inputs

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

- For each position $i$, create a position vector $v_i$
- To each word vector $x_i$ append the position vector $v_i$ corresponding to its position in the input sequence




The way the transformer network is defined so far is not sensitive to the order of the inputs. In other words, any permutation of the inputs would generate the same relations. Therefore in addition the position of words in a sentence needs to be taken into account. The way this is achieved is by learning another vector, which is the representation of an input position. Each word vector is concatenated with the position vector corresponding to the words position within the sentence.

## Input Length

For transformers a fixed length s of the input is assumed

- Maximal length of a sentence
- Position vectors $v_1, \ldots, v_s$

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




In order to introduce position embeddings the length of the sequences (sentences) needs to be fixed, or in other words a maximal sequence length needs to be specified. In terms of 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

If the task is text classification, the output of the transformer network is further processed

- Average all the outputs
- Project from dimension $k$ 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

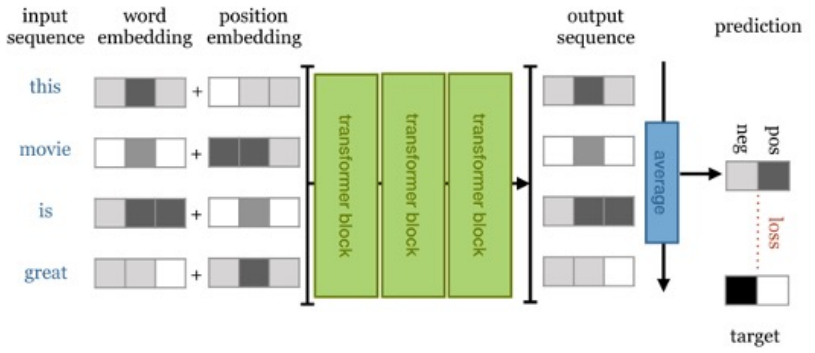



When using a transformer network for text classification, the output vectors are used to produce a predication for the classification label. In order to do this a standard approach is to average the output vectors, project them using a linear mapping the the dimension of the label space and apply a softmax 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. 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

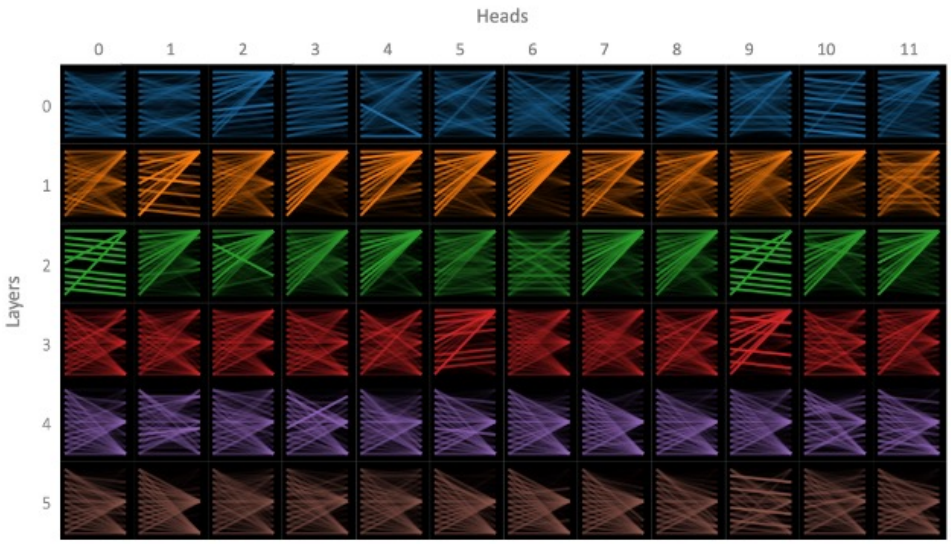






This diagram illustrates the overall architecture of a typical transformer network. What we can see is that multiple transformer blocks are applied in sequence (each of which contains a multi-head self-attention mechanism).

# Visualization of Layers and Heads








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 being used (that are distributed in particular through the huggingface platform) and the preexisting network is completed with a classification layer and retrained using the training data. This 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. Retraining such a network is called finetuning.

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 for some 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




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