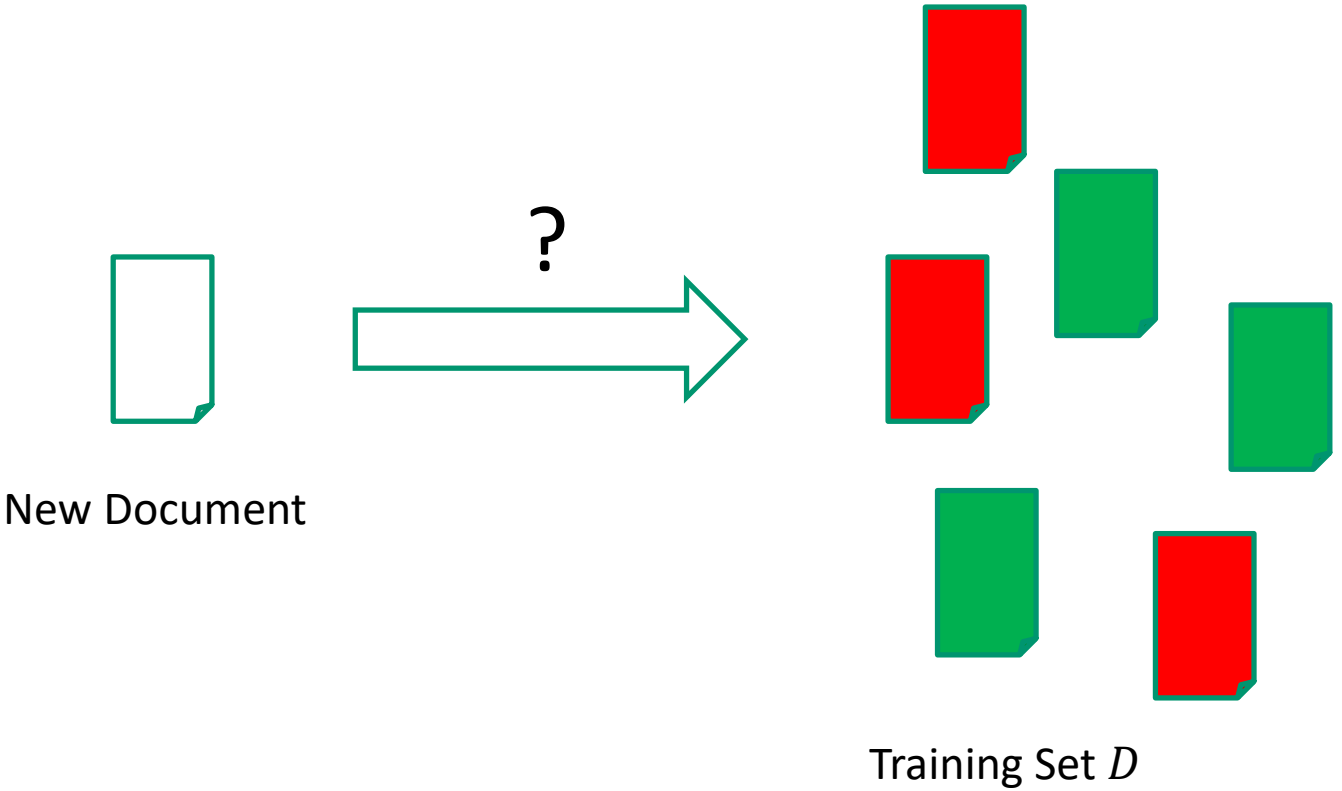
2.3 DOCUMENT CLASSIFICATION

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



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

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…

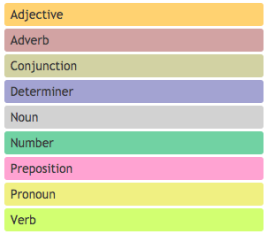
Bag of words: {more, than, visitors, attend, …}

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

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

2.3.1 k-Nearest-Neighbors (kNN)

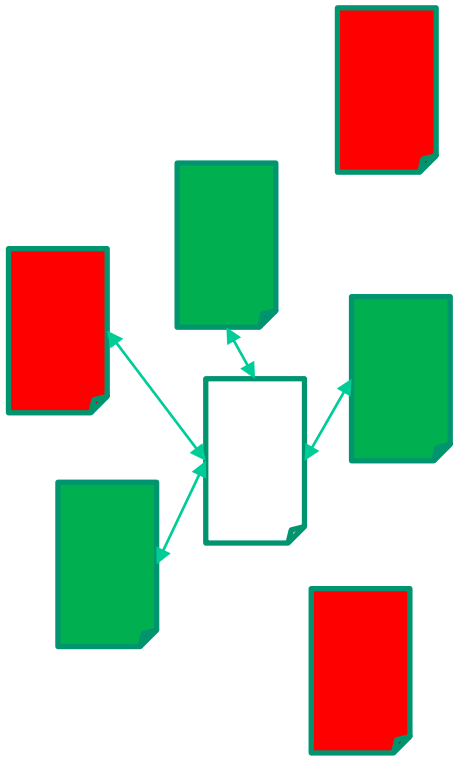
Simple approach: use vector space model

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



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

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

Rocchio classification

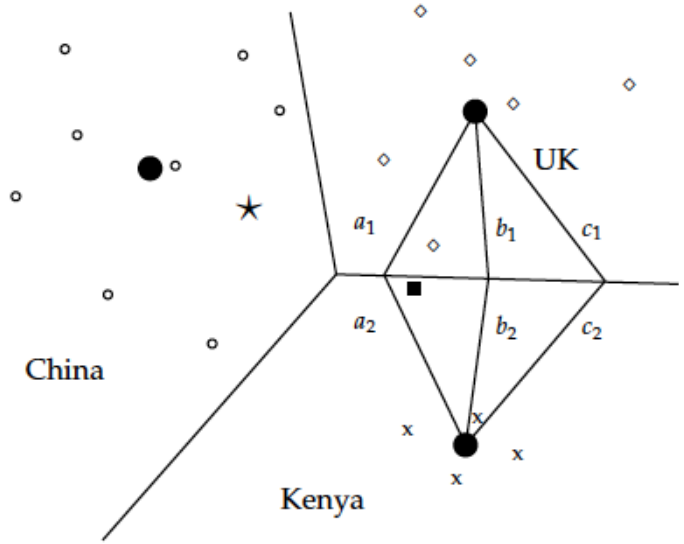
For each class C compute the centroid of all training samples D_C of documents in the class

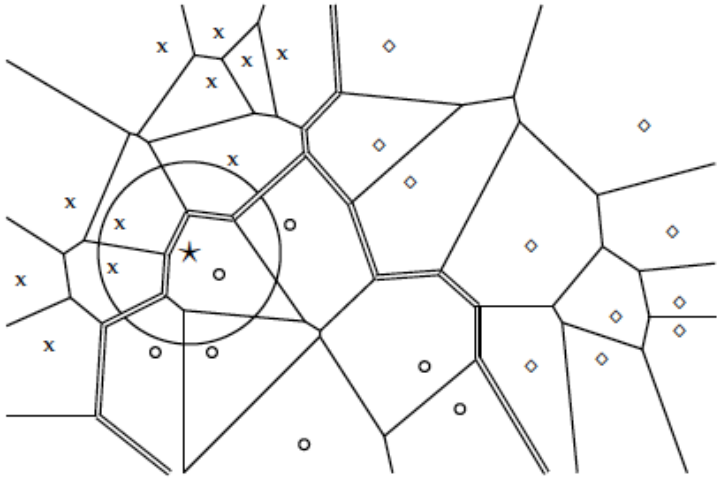
$$\mu(C) = \frac{1}{|D_C|} \sum_{d \in D_C} \vec{d}$$

For a document assign the class label of the closest centroid

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

Linear vs. Non-linear Classification





Rocchio: linear

- high bias
- low variance

1NN: non-linear

- low bias
- high variance

2.3.2 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

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

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?

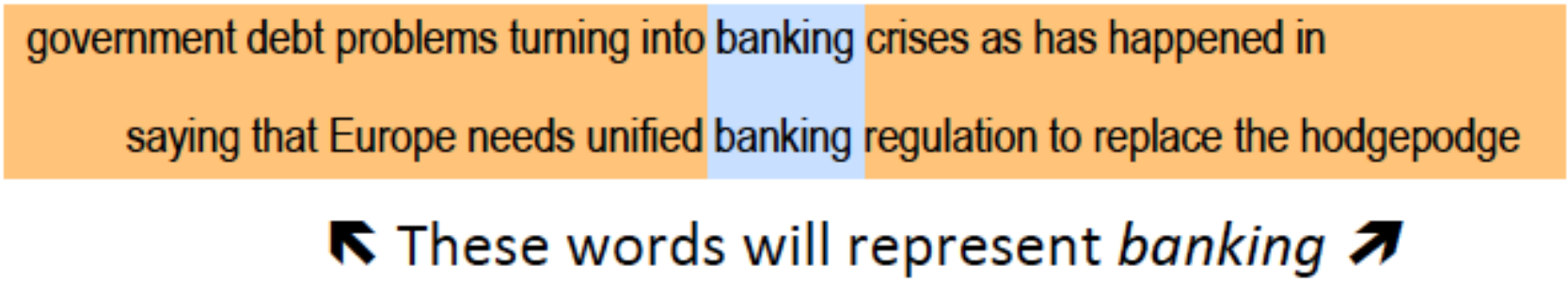
$$P(C) = \frac{|D_C|}{|D|}$$

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

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

2.3.3 Classification Using Word Embeddings

Reminder



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

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

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

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

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

Implementation of the Model

Training data encodes labels with text

__label1__, text1

__label2__, text2

…

Loss function

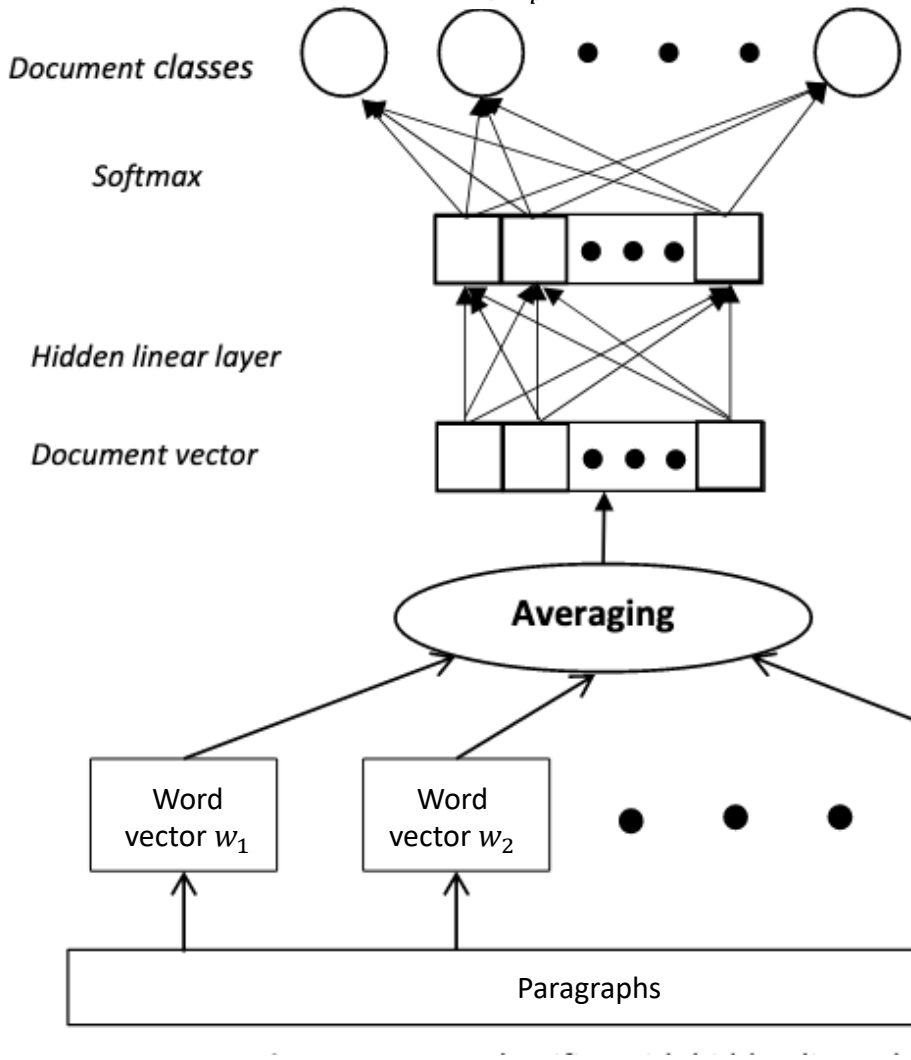
$$\underset{\theta}{\operatorname{argmax}} \sum_{C} \sum_{p} \log P_\theta(C|p)$$

Learning using SGD

- Number of class labels usually small

Summary of the Classification Model

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



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

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

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

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

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

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

2.3.4 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**

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

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

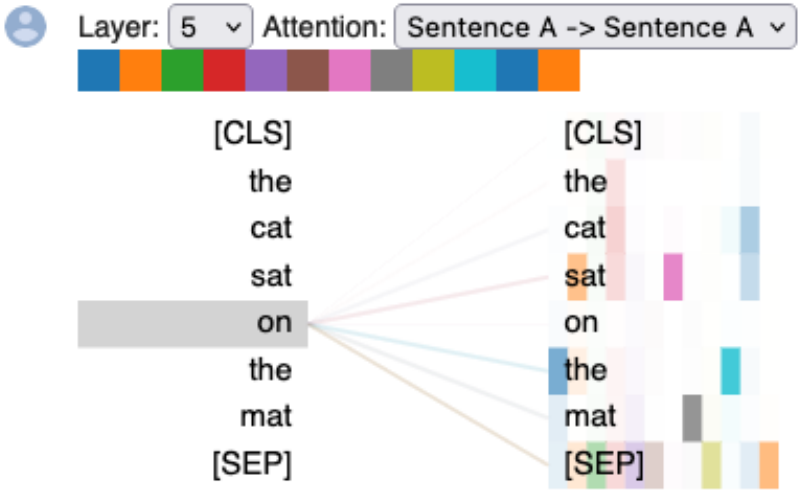
Simplest case $w'_{ij} = \boldsymbol{x}_i \cdot \boldsymbol{x}_j = \boldsymbol{x}_i^t \boldsymbol{x}_j$

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

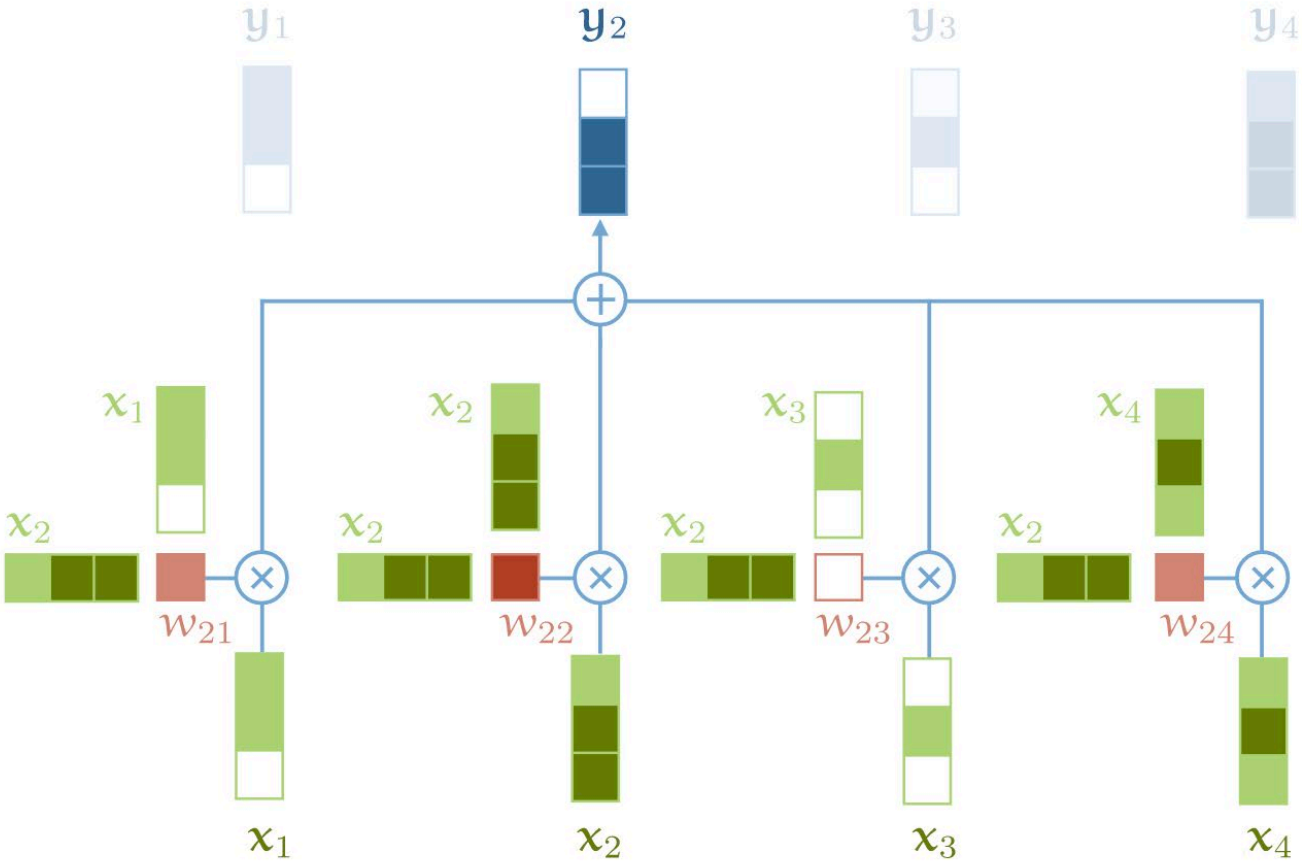
Intuition

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



Illustration



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

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

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 $d \times l$ matrix, l number of labels
5. Apply softmax: $c_i = \frac{e^{c'_i}}{\sum_i e^{c'_i}}$

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

More “Tricks”

Standard transformer models introduce many additional features

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

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

Queries, Keys, Values

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

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

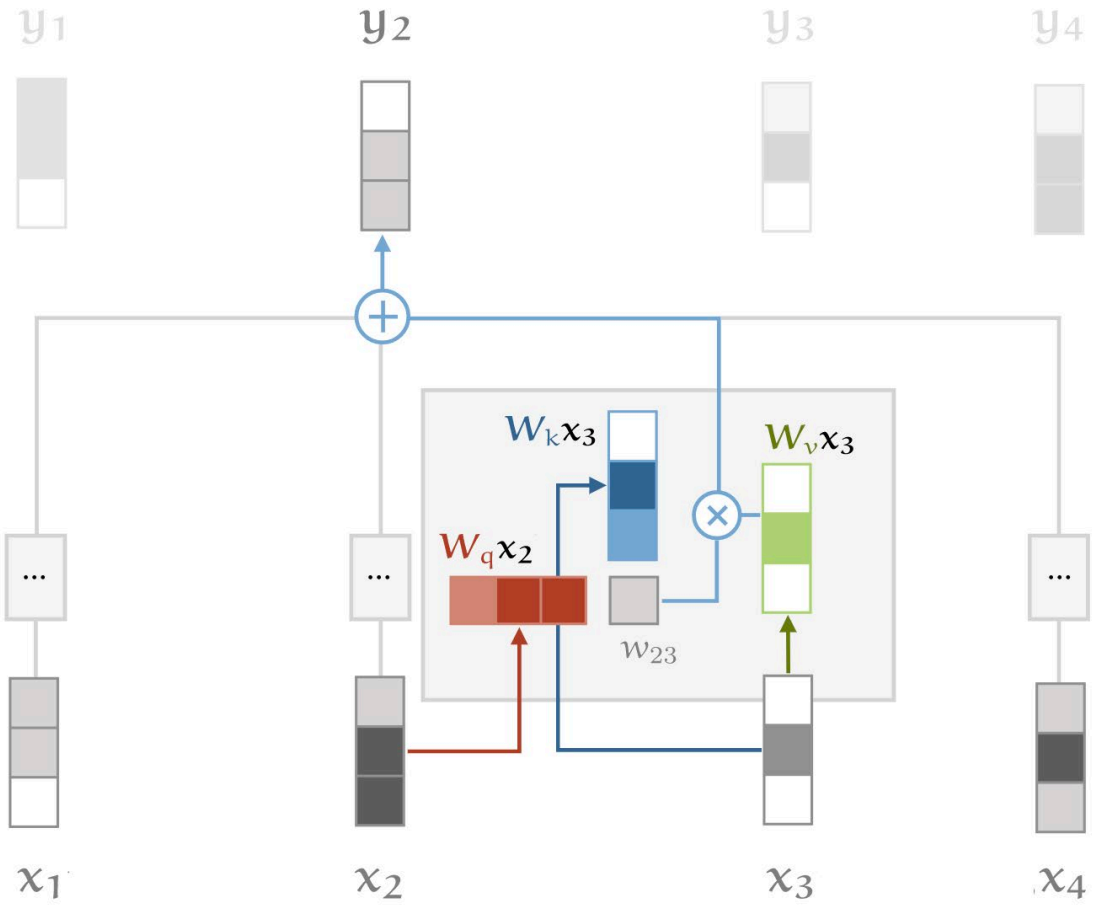
$$w'_{ij} = \boldsymbol{q}_i^t \boldsymbol{k}_j$$

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

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

Note: the linear transformations are independent of the vectors

Illustration



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

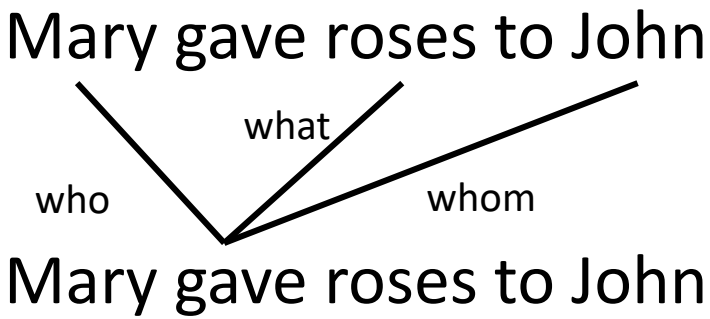
$$w'_{ij} = \frac{\boldsymbol{q}_i^t \boldsymbol{k}_j}{\sqrt{d}}$$

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

Multi-head Attention

A word can mean different things to different neighboring words

Example



3 different types of relationships

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

Combining Multi-head Attention Outputs

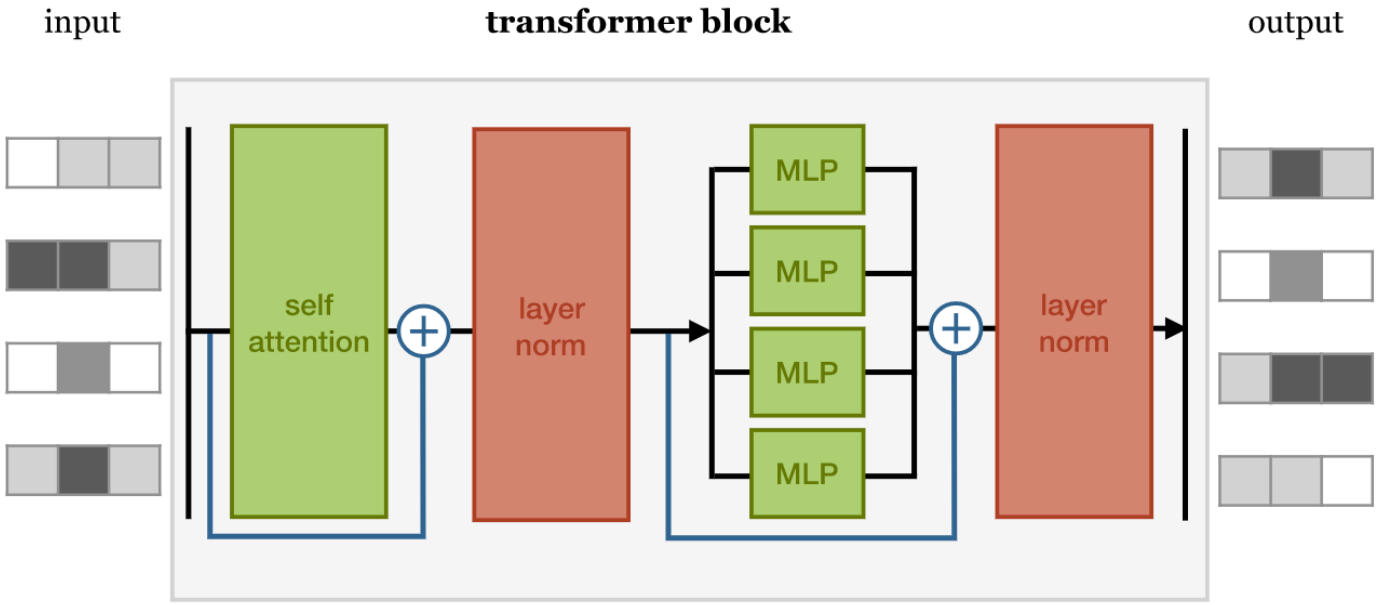
Concatenate the different outputs $\boldsymbol{y}_{im}, m = 1, \ldots, t$ of multiple attention heads

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

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

Transformers

Transformers use the self-attention mechanism as building block



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

MLP – Multilayer Perceptron

An MLP is one or more linear transformations, each followed by an activation function (e.g., sigmoid, ReLu, 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 $W\boldsymbol{x} + \boldsymbol{b}$

MLP in Transformer Networks

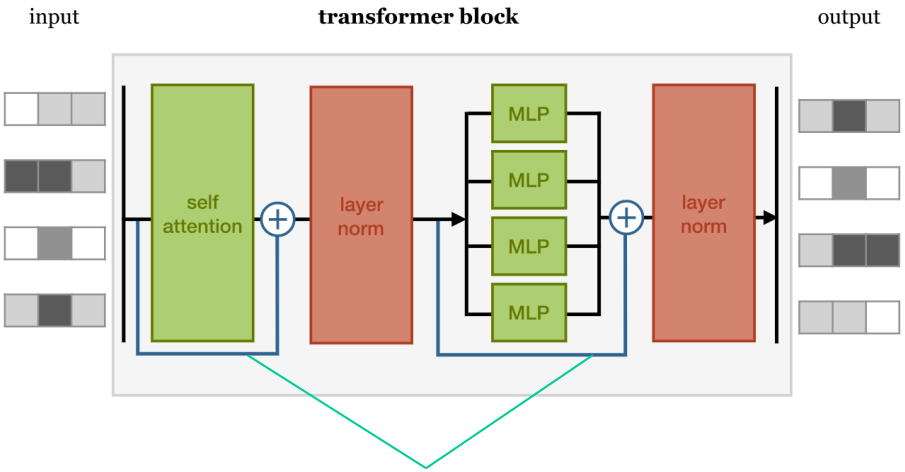
To each normalized output vector $\boldsymbol{y}_i$ a MLP is applied **separately**

- 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

Residual Connections

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

$$y := x + y$$



Residual connections

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

Input Length

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

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

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

Classification Layer

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

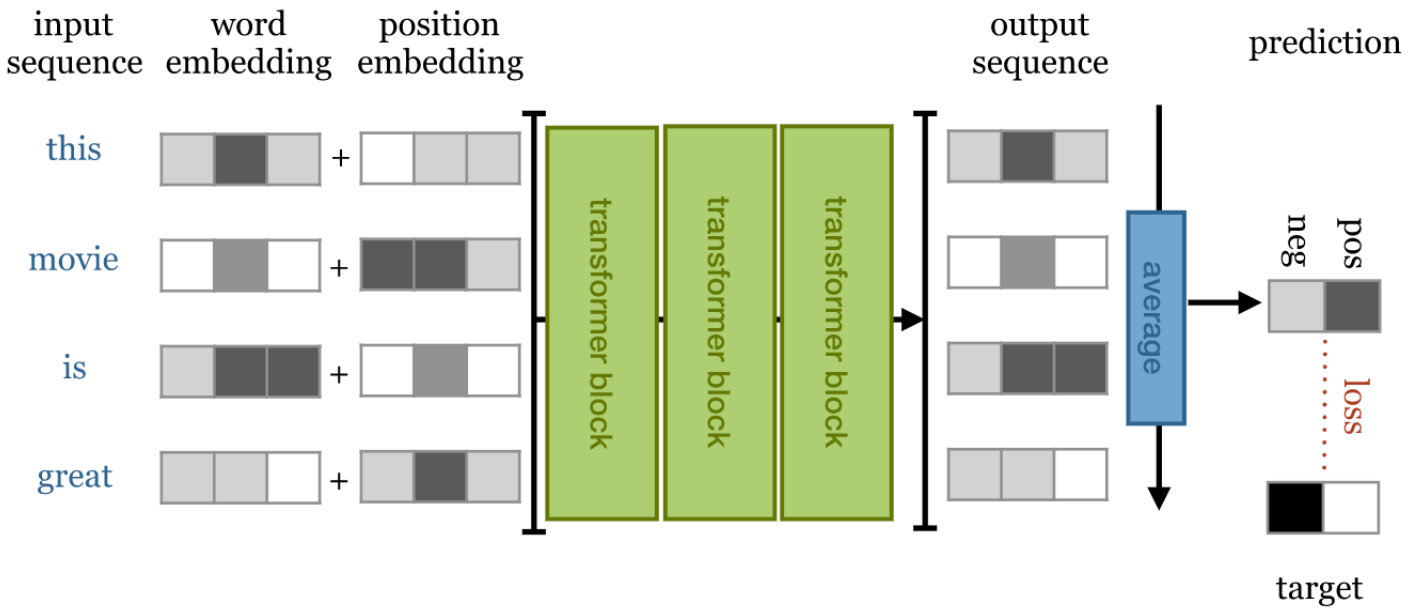
- Average all the outputs
- Project from dimension d to the dimension l, the number of labels
- Apply a softmax function

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

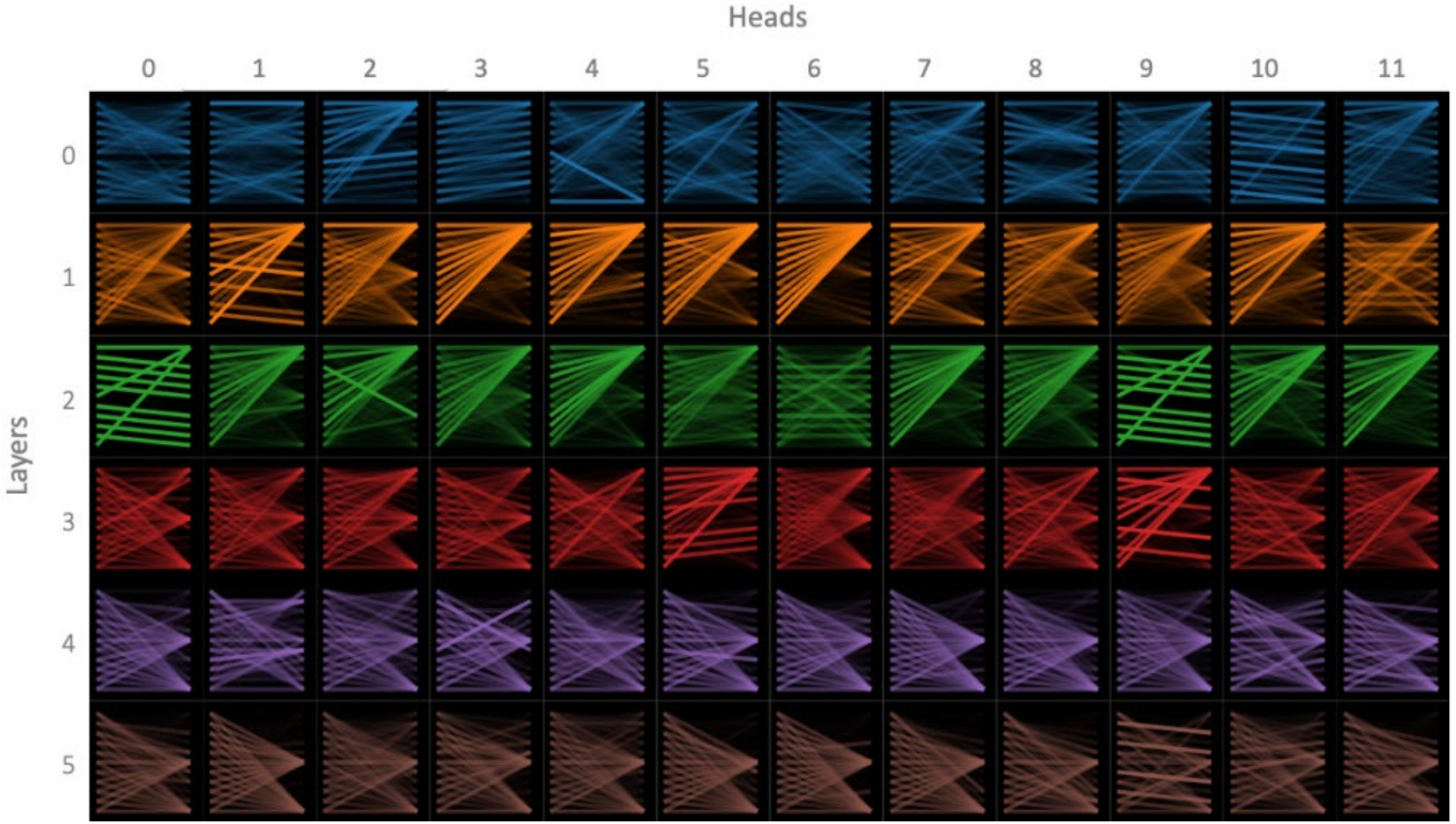
This completes the classifier

Putting it all together

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



Visualization of Layers and Heads



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

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

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

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