Convolutional Neural Networks (CNN)

Manfred Klenner

Department of Computerlinguistik
University of Zurich

December 7, 2020

Introduction

- so far: fully connected layers, e.g. feedforward network
- vision: a picture of a dog and cat, learning to identify the cat (or the dog)
- a pixel of the cat needs not to be connected with any of the dog
- thus we can cut down the number of weights (by reducing the connections through pooling)

Introduction

- so far: fully connected layers, e.g. feedforward network
- vision: a picture of a dog and cat, learning to identify the cat (or the dog)
- a pixel of the cat needs not to be connected with any of the dog
- thus we can cut down the number of weights (by reducing the connections through pooling)

Introduction

- so far: fully connected layers, e.g. feedforward network
- vision: a picture of a dog and cat, learning to identify the cat (or the dog)
- a pixel of the cat needs not to be connected with any of the dog
- thus we can cut down the number of weights (by reducing the connections through pooling)

Introduction

- so far: fully connected layers, e.g. feedforward network
- vision: a picture of a dog and cat, learning to identify the cat (or the dog)
- a pixel of the cat needs not to be connected with any of the dog
- thus we can cut down the number of weights (by reducing the connections through pooling)

Used material

- for a very good overview see: https://arxiv.org/abs/1603.07285
- Vincent Dumoulin, Francesco Visin (2018):'A guide to convolution arithmetic for deep learning'
- great visualization: github.com/vdumoulin/conv_arithmetic/blob/master/README.md

Used material

- for a very good overview see: https://arxiv.org/abs/1603.07285
- Vincent Dumoulin, Francesco Visin (2018):'A guide to convolution arithmetic for deep learning'

- great visualization: github.com/vdumoulin/conv_arithmetic/blob/master/README.md

Used material

- for a very good overview see: https://arxiv.org/abs/1603.07285
- Vincent Dumoulin, Francesco Visin (2018):'A guide to convolution arithmetic for deep learning'

- great visualization: github.com/vdumoulin/conv_arithmetic/blob/master/README.md

Used material

- for a very good overview see: https://arxiv.org/abs/1603.07285
- Vincent Dumoulin, Francesco Visin (2018):'A guide to convolution arithmetic for deep learning'

- great visualization: github.com/vdumoulin/conv_arithmetic/blob/master/README.md

CNN

A architecture where regions of the input features are mapped locally to some output region

- one or more input feature maps
- kernels (filter): slides across the input feature maps with a stride (steps taken)
- kernel size normally is smaller then input size
  - takes the Hadamard product (cell-wise multiplication)
  - takes the sum of the product (which gives the single output cell)
- padding 0 (since filters on the edges are less well defined)

  so to speak: the zeros act as indicators of edge regions
- pooling
- fully connected (end) layers

CNN

A architecture where regions of the input features are mapped locally to some output region

- one or more input feature maps
- kernels (filter): slides across the input feature maps with a stride (steps taken)
- kernel size normally is smaller then input size
  - takes the Hadamard product (cell-wise multiplication)
  - takes the sum of the product (which gives the single output cell)
- padding 0 (since filters on the edges are less well defined)

  so to speak: the zeros act as indicators of edge regions
- pooling
- fully connected (end) layers

CNN

A architecture where regions of the input features are mapped locally to some output region

- one or more input feature maps
- kernels (filter): slides across the input feature maps with a stride (steps taken)
- kernel size normally is smaller then input size
  - takes the Hadamard product (cell-wise multiplication)
  - takes the sum of the product (which gives the single output cell)
- padding 0 (since filters on the edges are less well defined)

  so to speak: the zeros act as indicators of edge regions
- pooling
- fully connected (end) layers

CNN

A architecture where regions of the input features are mapped locally to some output region

- one or more input feature maps
- kernels (filter): slides across the input feature maps with a stride (steps taken)
- kernel size normally is smaller then input size
  - takes the Hadamard product (cell-wise multiplication)
  - takes the sum of the product (which gives the single output cell)
- padding 0 (since filters on the edges are less well defined)

  so to speak: the zeros act as indicators of edge regions
- pooling
- fully connected (end) layers

CNN

A architecture where regions of the input features are mapped locally to some output region

- one or more input feature maps
- kernels (filter): slides across the input feature maps with a stride (steps taken)
- kernel size normally is smaller then input size
  - takes the Hadamard product (cell-wise multiplication)
  - takes the sum of the product (which gives the single output cell)
- padding 0 (since filters on the edges are less well defined)

  so to speak: the zeros act as indicators of edge regions
- pooling
- fully connected (end) layers

CNN

A architecture where regions of the input features are mapped locally to some output region

- one or more input feature maps
- kernels (filter): slides across the input feature maps with a stride (steps taken)
- kernel size normally is smaller then input size
  - takes the Hadamard product (cell-wise multiplication)
  - takes the sum of the product (which gives the single output cell)
- padding 0 (since filters on the edges are less well defined)

  so to speak: the zeros act as indicators of edge regions
- pooling
- fully connected (end) layers

CNN

A architecture where regions of the input features are mapped locally to some output region

- one or more input feature maps
- kernels (filter): slides across the input feature maps with a stride (steps taken)
- kernel size normally is smaller then input size
  - takes the Hadamard product (cell-wise multiplication)
  - takes the sum of the product (which gives the single output cell)
- padding 0 (since filters on the edges are less well defined)

  so to speak: the zeros act as indicators of edge regions
- pooling
- fully connected (end) layers

CNN

A architecture where regions of the input features are mapped locally to some output region

- one or more input feature maps
- kernels (filter): slides across the input feature maps with a stride (steps taken)
- kernel size normally is smaller then input size
  - takes the Hadamard product (cell-wise multiplication)
  - takes the sum of the product (which gives the single output cell)
- padding 0 (since filters on the edges are less well defined)

  so to speak: the zeros act as indicators of edge regions
- pooling
- fully connected (end) layers

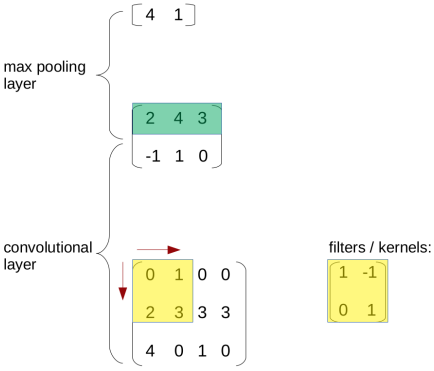
CNN

A architecture where regions of the input features are mapped locally to some output region

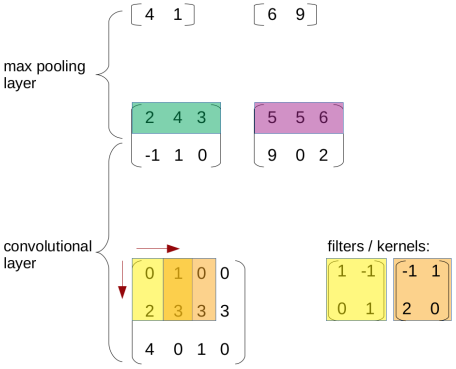
- one or more input feature maps
- kernels (filter): slides across the input feature maps with a stride (steps taken)
- kernel size normally is smaller then input size
  - takes the Hadamard product (cell-wise multiplication)
  - takes the sum of the product (which gives the single output cell)
- padding 0 (since filters on the edges are less well defined)

  so to speak: the zeros act as indicators of edge regions
- pooling
- fully connected (end) layers

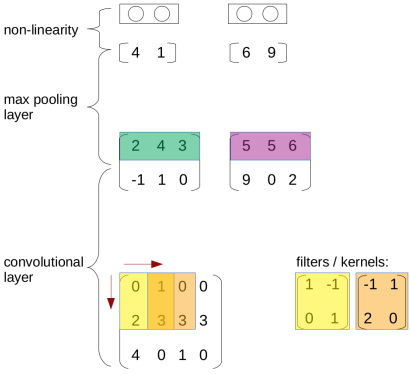
Example


4 1
max pooling layer
2 4 3
-1 1 0
convolutional layer
0 1 0 0
2 3 3 3
4 0 1 0
filters / kernels:
1 -1
0 1

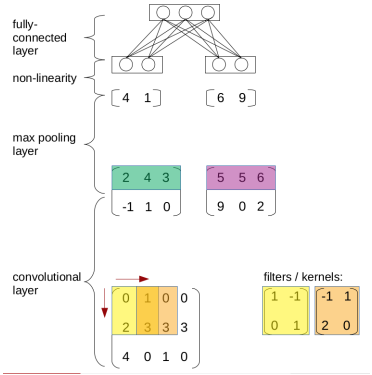
Example


4 1
6 9
max pooling layer
2 4 3
-1 1 0
5 5 6
9 0 2
convolutional layer
0 1 0 0
2 3 3 3
4 0 1 0
filters / kernels:
1 -1
0 1
-1 1
2 0

Example


non-linearity
4 1
6 9
max pooling layer
2 4 3
-1 1 0
5 5 6
9 0 2
convolutional layer
0 1 0 0
2 3 3 3
4 0 1 0
filters / kernels:
1 -1
0 1
-1 1
2 0

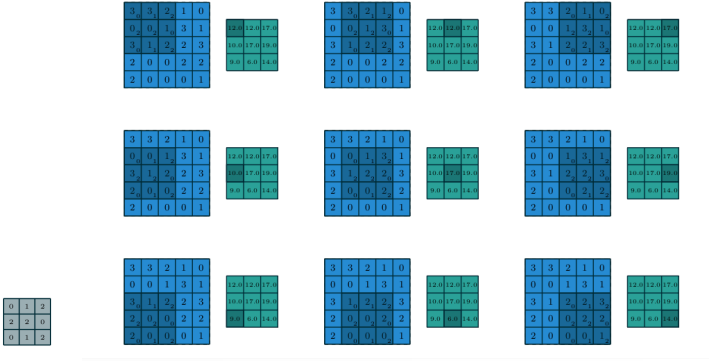
Example


fully-connected layer
non-linearity
max pooling layer
convolutional layer
4 1
6 9
2 4 3
-1 1 0
5 5 6
9 0 2
0 1 0 0
2 3 3 3
4 0 1 0
filters / kernels:
1 -1
0 1
-1 1
2 0

CNN Parameters

- i_j: input size along axis j
- k_j: kernel size along axis j
- s_j: stride (distance between two consecutive positions of the kernel) along axis j
- p_j: zero padding (number of zeros added at the beginning and at the end of an axis) along axis j
- $i_1 = i_2 = 5, k_1 = k_2 = 3, ...$

kernel:



CNN: Illustration Filtering

- 3 × 3 kernel applied to a 5 × 5 input
- padded with a 1 × 1 border of zeros
- using 2 × 2 strides
- strides constitute a form of subsampling

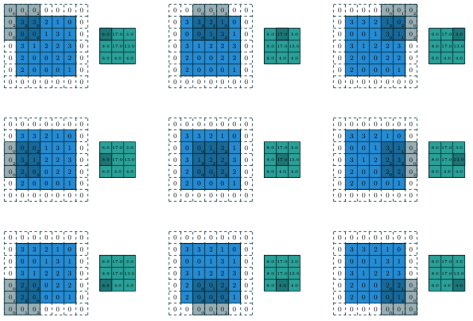


Figure 1.2: Computing the output values of a discrete convolution for $N = 2$, $i_1 = i_2 = 5$, $k_1 = k_2 = 3$, $s_1 = s_2 = 2$, and $p_1 = p_2 = 1$.

CNN: More Filters, More Input Maps

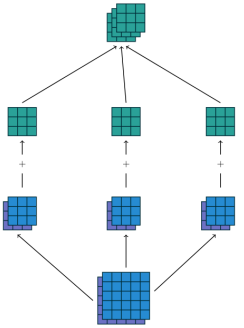


Figure 1.3: A convolution mapping from two input feature maps to three output feature maps using a $3 \times 2 \times 3 \times 3$ collection of kernels $\mathbf{w}$. In the left pathway, input feature map 1 is convolved with kernel $\mathbf{w}_{1,1}$ and input feature map 2 is convolved with kernel $\mathbf{w}_{1,2}$, and the results are summed together elementwise to form the first output feature map. The same is repeated for the middle and right pathways to form the second and third feature maps, and all three output feature maps are grouped together to form the output.

CNN: Pooling

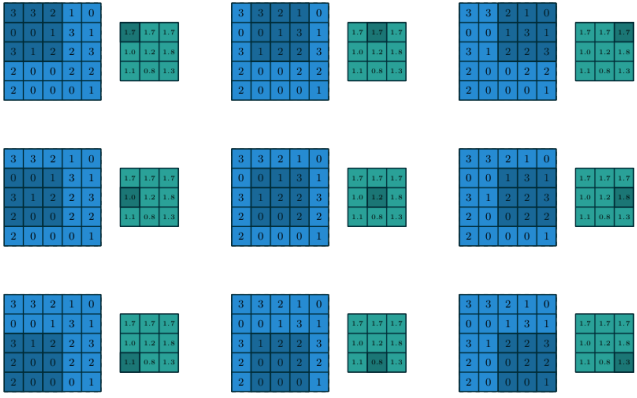


Figure 1.5: Computing the output values of a 3×3 average pooling operation on a 5×5 input using 1×1 strides.

CNN: Pooling

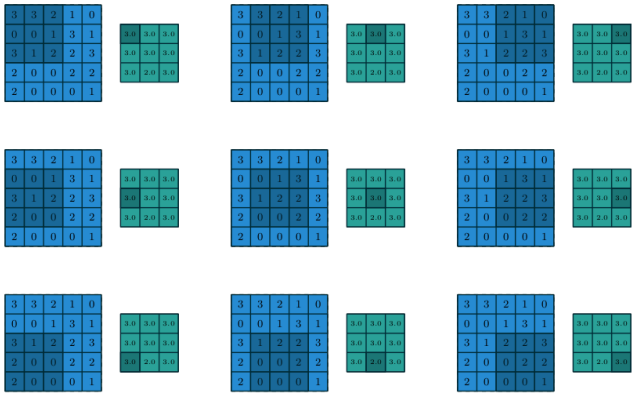


Figure 1.6: Computing the output values of a 3×3 max pooling operation on a 5×5 input using 1×1 strides.

CNN: further illustrations

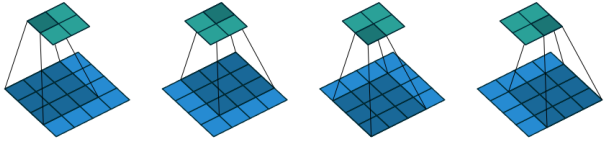


Figure 2.1: (No padding, unit strides) Convolving a 3×3 kernel over a 4×4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$).

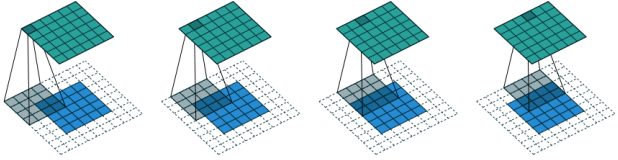


Figure 2.2: (Arbitrary padding, unit strides) Convolving a 4×4 kernel over a 5×5 input padded with a 2×2 border of zeros using unit strides (i.e., $i = 5$, $k = 4$, $s = 1$ and $p = 2$).

CNN: Half Padding

half padding: keeps the size of the input, full padding: increases the size of the input

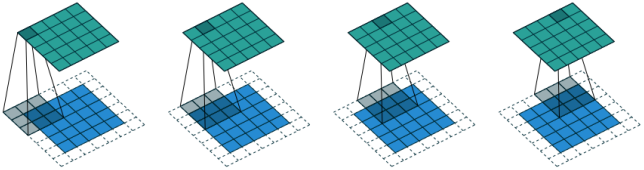


Figure 2.3: (Half padding, unit strides) Convolving a 3×3 kernel over a 5×5 input using half padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 1$).

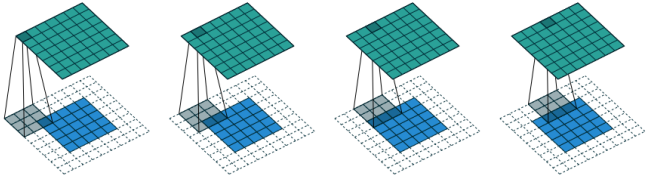


Figure 2.4: (Full padding, unit strides) Convolving a 3×3 kernel over a 5×5 input using full padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 2$).

CNN: Dilated Convolution

Dilated convolutions "inflate" the kernel by inserting spaces between the kernel elements. Parameter d (d-1 is inserted, so d=2 means 1 space)

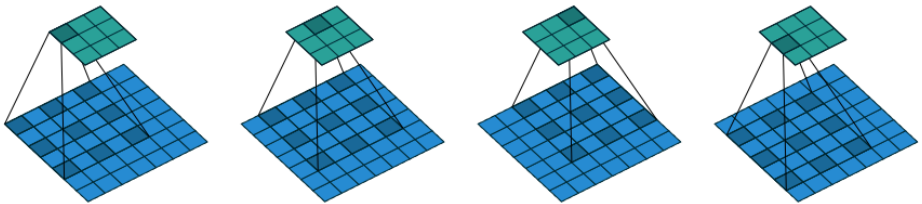


Figure 5.1: (Dilated convolution) Convolving a 3×3 kernel over a 7×7 input with a dilation factor of 2 (i.e., $i = 7$, $k = 3$, $d = 2$, $s = 1$ and $p = 0$).

Hyperparameter: CNN

- Convolutional layer:
  - Number of filters
  - Filter size (height and width)
  - Stride
- Max pooling layer:
  - Aggregation function (max, average, k-max, ...)
  - Pooling window size (height and width)
  - Stride

Used in Sentiment analysis, Relation classification, Spoken language understanding, ...

Hyperparameter: CNN

- Convolutional layer:
  - Number of filters
  - Filter size (height and width)
  - Stride
- Max pooling layer:
  - Aggregation function (max, average, k-max, ...)
  - Pooling window size (height and width)
  - Stride

Used in Sentiment analysis, Relation classification, Spoken language understanding, ...

Hyperparameter: CNN

- Convolutional layer:
  - Number of filters
  - Filter size (height and width)
  - Stride
- Max pooling layer:
  - Aggregation function (max, average, k-max, ...)
  - Pooling window size (height and width)
  - Stride

Used in Sentiment analysis, Relation classification, Spoken language understanding, ...

Hyperparameter: CNN

- Convolutional layer:
  - Number of filters
  - Filter size (height and width)
  - Stride
- Max pooling layer:
  - Aggregation function (max, average, k-max, ...)
  - Pooling window size (height and width)
  - Stride

Used in Sentiment analysis, Relation classification, Spoken language understanding, ...

Hyperparameter: CNN

- Convolutional layer:
  - Number of filters
  - Filter size (height and width)
  - Stride
- Max pooling layer:
  - Aggregation function (max, average, k-max, ...)
  - Pooling window size (height and width)
  - Stride

Used in Sentiment analysis, Relation classification, Spoken language understanding, ...

Hyperparameter: CNN

- Convolutional layer:
  - Number of filters
  - Filter size (height and width)
  - Stride
- Max pooling layer:
  - Aggregation function (max, average, k-max, ...)
  - Pooling window size (height and width)
  - Stride

Used in Sentiment analysis, Relation classification, Spoken language understanding, ...

Hyperparameter: CNN

- Convolutional layer:
  - Number of filters
  - Filter size (height and width)
  - Stride
- Max pooling layer:
  - Aggregation function (max, average, k-max, ...)
  - Pooling window size (height and width)
  - Stride

Used in Sentiment analysis, Relation classification, Spoken language understanding, ...

Hyperparameter: CNN

- Convolutional layer:
  - Number of filters
  - Filter size (height and width)
  - Stride
- Max pooling layer:
  - Aggregation function (max, average, k-max, ...)
  - Pooling window size (height and width)
  - Stride

Used in Sentiment analysis, Relation classification, Spoken language understanding, ...

Autoencoder

Quoting Wikipedia 'An autoencoder is a type of artificial neural network used to learn efficient data codings in an unsupervised manner. The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction.'
Quote: Convolutional Autoencoder is a variant of Convolutional Neural Networks that are used as the tools for unsupervised learning of convolution filters.

Movie Review: Sentiment

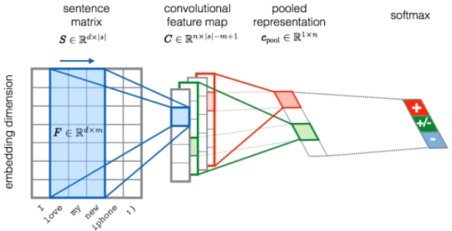


Image retrieved from Stack Exchange

- we train our own embeddings, words representend by a vector of size 100
- 4 different filters on the word vectors to create convolutional feature maps (but not in the implementation under the web address below)
- maximum pooling
- softmax to transform a vector of size 1x4 to a vector of size 1x3 for classification

see: towardsdatascience.com/convolutional-neural-network-in-natural-language-processing-96d67f91275c

Movie Review: Sentiment

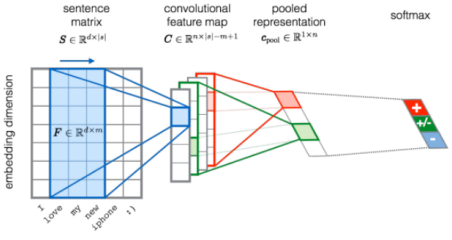


Image retrieved from Stack Exchange

- we train our own embeddings, words representend by a vector of size 100
- 4 different filters on the word vectors to create convolutional feature maps (but not in the implementation under the web address below)
- maximum pooling
- softmax to transform a vector of size 1x4 to a vector of size 1x3 for classification

see: towardsdatascience.com/convolutional-neural-network-in-natural-language-processing-96d67f91275c

Movie Review: Sentiment

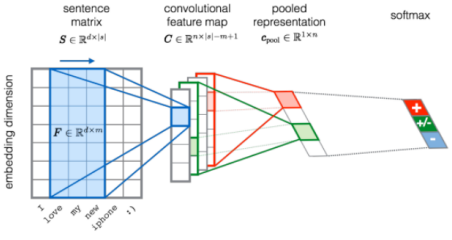


Image retrieved from Stack Exchange

- we train our own embeddings, words representend by a vector of size 100
- 4 different filters on the word vectors to create convolutional feature maps (but not in the implementation under the web address below)
- maximum pooling
- softmax to transform a vector of size 1x4 to a vector of size 1x3 for classification

see: towardsdatascience.com/convolutional-neural-network-in-natural-language-processing-96d67f91275c

Movie Review: Sentiment

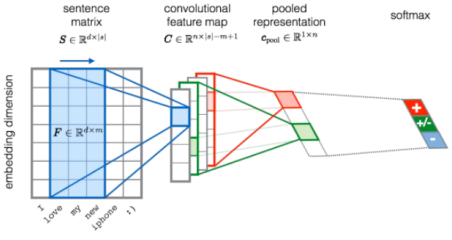


Image retrieved from Stack Exchange

- we train our own embeddings, words representend by a vector of size 100
- 4 different filters on the word vectors to create convolutional feature maps (but not in the implementation under the web address below)
- maximum pooling
- softmax to transform a vector of size 1x4 to a vector of size 1x3 for classification

see: towardsdatascience.com/convolutional-neural-network-in-natural-language-processing-96d67f91275c

Keras used for loading the data

```
tf.keras.datasets.imdb.load_data(
    path='imdb.npz', num_words=None, skip_top=0, maxlen=None, seed=113,
    start_char=1, oov_char=2, index_from=3, **kwargs
    )
```

This is a dataset of 25,000 movies reviews from IMDB, labeled by sentiment (positive/negative). Reviews have been preprocessed, and each review is encoded as a list of word indexes (integers). For convenience, words are indexed by overall frequency in the dataset, so that for instance the integer "3" encodes the 3rd most frequent word in the data. This allows for quick filtering operations such as: "only consider the top 10,000 most common words, but eliminate the top 20 most common words".

Frameworks: overview

Pytorch: nn.Conv2d

nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2))

- in channel 16
- out channel 33
- kernel size: either integer, then quadratic, or tuple: (height, width) (3,5)
- stride
- padding

Wrap-up: how machine learning works

- try to find a separating hyperplane (in the simplest case of perceptron)
- this means to minimize a loss function
- we no longer look for a single hyperplane in ffn, rnn, ..
- but we still minimize loss: if the loss quantifies the error, this must be successful then
- based on the gradient of the loss function
- however complex a function might be (and nested)
- with chain rule we can fix any parameter (weights in our case)
- clearly, there are problems: vanishing gradients, overfitting, exploding gradients
- and there is much room for variations
- different loss and activation functions
- non-linearity as a very important building block, XOR problem
- are our data always non-linear: no, but most of time (if very complex)
- we looked at the mathematical details - perceptron, feed forward this is basic stuff
- after we discussed the very idea and how we use compational graph in order to backpropagate, there was no need to dive into every math. details

Wrap-up

- we use always backprop, but clearly there differences, e.g. with rnn, cnn
- but the principle are the same
- pytorch already provides the building blocks and eases the creation of NN
- different architectures:
  - rnn, lstm, gru for sequence modelling
  - rnn: a hidden state that was updated constantly
  - lstm: how it works - various gates: forget, update (softmax: multiply, add)
  - cell state als long term memory
  - we didn't talk about decoder - encoder, b-directional rnn
  - here we take the last hidden state as input for the decoder RNN
  - there we run the rnn from left to right and the other way round as well and concatenate
- self attention, transformer: residual (bypass), normalization (scale), drop out (randomly distract)
  - we looked at BERT (Google) and how to use it (pretrained) and how to train it (transfer learning)
  - we applied it to real NLP problems - sentiment analysis
  - there are lot of examples on the web: e.g. NER, text generation, question answering, reading comprehension

Wrap-up: acquired skills, ...

- know how to use out of the box sklearn given some data set
- know how to verify statistical progress - t-test
- be able to adapt existing or writing from scratch NN

- DL: lots of data needed, otherwise traditional ML migth be better
- note: with companies often no annotated data are available (at the beginning)
- annotation is needed - but companies are slow (in decision processes)
- even rule-based ML might be needed

- you're well prepared for the Spring NMT after this course
- but: practise - this is crucial
- take data (there is a lot available) and apply DL
- carry out experiments - with other loss (write your own?), different achitectures

Wrap-up: linguistics and DL

- linguistics - where is it?
- take cnn example: bagof words, a word is represented by its frequency ranking
- no pos, no lexicon, no morphology, parsing ....
- wordform are represented by their freq as indices of a randomly initialized embedding
- thus, computer scientist are active in our field as well
- we have to catch up with their technical skill, otherwise they are the better NLP engineers
- linguistic features are not totaly pointless, though
- some researchers still use it and find improvements