Chapter 4

“The new n-grams” — Convolutional Neural Networks

Content of this Chapter

1. Modelling Text Flow – Sequences of Words and Characters
2. Convolutional Neural Networks (in Computer Vision)
    1. Components of CNNs
    2. Backpropagation in CNNs
3. Convolutional Neural Networks in NLP?

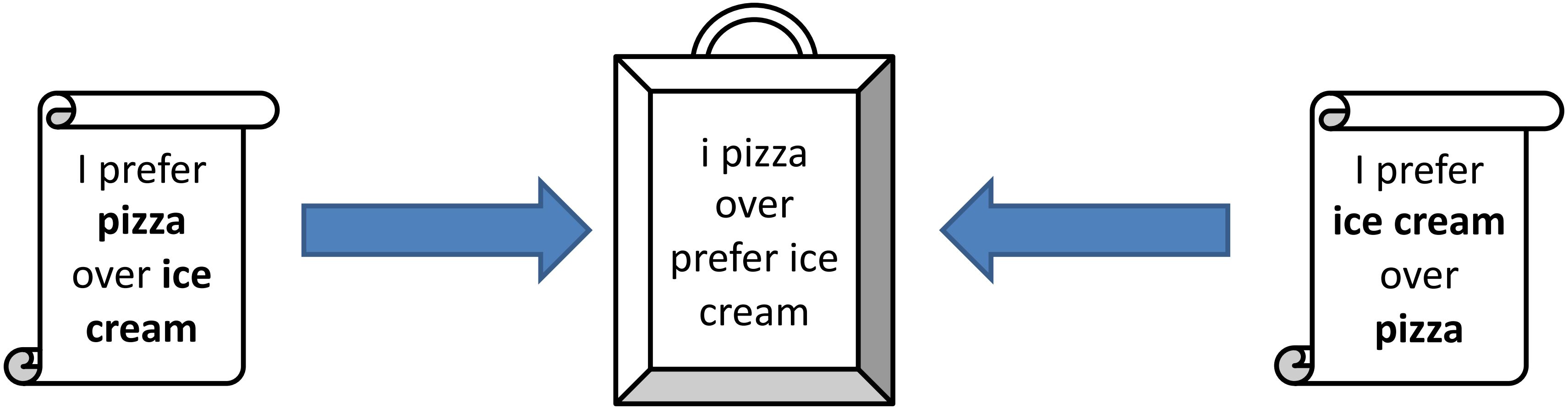
4.1 Modelling the Text Flow – Sequences of Words and Characters

- Text is more than just words!
- Models for larger „chunks“ needed

Recall from first lecture

Modelling Documents – Bag of Words

- Works rather well...
- ...but loses a lot of information!



Recall from first lecture

Modelling Documents – Sequence of Words

- We need to consider word order!

→ A document d is an ordered sequence of words

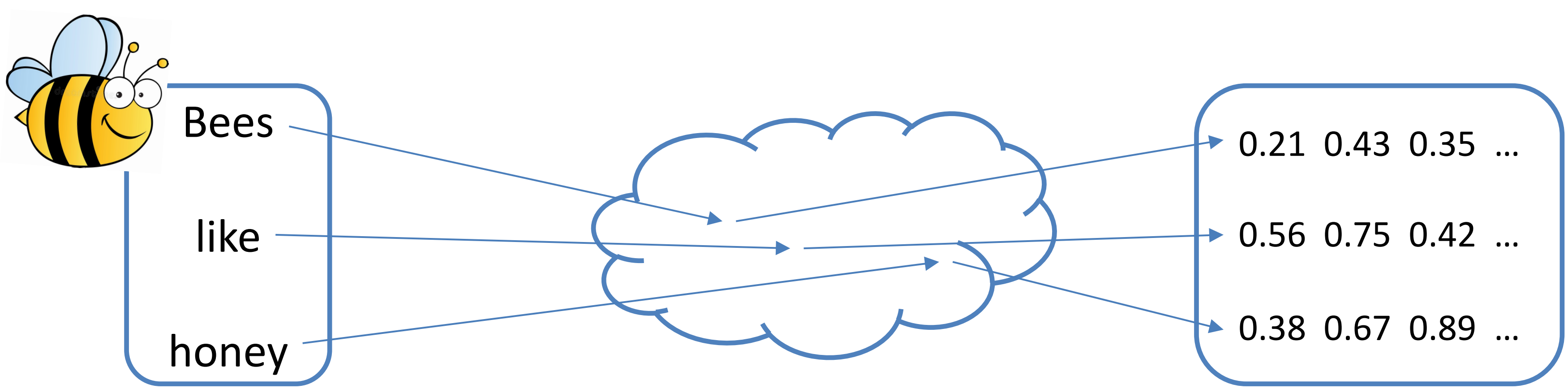
- $d = (d_1, d_2, \ldots, d_n)$

- Now we need to find a good representation for words

Modelling Documents – Sequence of Words

- $d = (d_1, d_2, \ldots, d_n)$
- We need to find a good representation for words
  → Done. Use word embeddings!

- Using a word embedding E of size 300, a document of length l is now an $\mathrm{l} \times 300$ matrix d



Modelling Documents – Sequence of Words

$$d = \begin{pmatrix} \text{Bees} \\ \text{like} \\ \text{honey} \end{pmatrix} = \begin{pmatrix} 0.21 & 0.43 & 0.35 & \ldots \\ 0.56 & 0.75 & 0.42 & \ldots \\ 0.38 & 0.67 & 0.89 & \ldots \end{pmatrix}$$

- What to do with this?
  Goal: Some kind of classification,
  e.g. $c(d) =$ „positive“ for sentiment classification

- How to achieve this?
  Some ideas on the following slides

Classifying Documents – SVM

$$d = \begin{bmatrix} \text{Bees} \\ \text{like} \\ \text{honey} \end{bmatrix} = \begin{bmatrix} 0.21 & 0.43 & 0.35 & \dots \\ 0.56 & 0.75 & 0.42 & \dots \\ 0.38 & 0.67 & 0.89 & \dots \end{bmatrix}$$

- Support Vector Machine = Classic text classifier
- Usually used with bag-of-words representations
- How to:
  - SVMs take as input a vector → Concatenate all embeddings
  - Loses information about „local contexts“ from the matrix

$$\begin{bmatrix} 0.21 & 0.43 & 0.35 & \dots \\ 0.56 & 0.75 & 0.42 & \dots \\ 0.38 & 0.67 & 0.89 & \dots \end{bmatrix} \Rightarrow \begin{bmatrix} 0.21 & 0.43 & 0.35 & \dots & 0.56 & 0.75 & 0.42 & \dots & 0.38 & 0.67 & 0.89 & \dots \end{bmatrix}$$

- Tends not to work well in this setting

Classifying Documents – Fully Connected Network

$d =$ Bees like honey = 0.21 0.43 0.35 … 0.56 0.75 0.42 … 0.38 0.67 0.89 …

- Let's use neural networks!
- We know fully connected networks…
- … which still take as input a vector

→ Same problem as SVMs

→ Additionally: Pretty large input (see next slide for an example)!

Classifying Documents – Fully Connected Network

- Document length: $\mathrm{l} = 300$ words
- Embedding size: $s = 300$
- Size of the first hidden layer: $h = 5000$

→ Shape of first weight matrix:

$$\underbrace{300 \cdot 300}_{\text{Input size}} \times \underbrace{5000}_{\text{First hidden layer size}} \Rightarrow 450{,}000{,}000$$

→ Float32: 32 bit per number

→ $450{,}000{,}000 \cdot 32$ bit = 1.8 GB for the weights

→ And $450{,}000{,}000$ weights to optimise in a single layer!

Classifying Documents – Fully Connected Network

- Document length: $\mathrm{l} = 300$ words
- Embedding size: $s = 300$
- Size of the first hidden layer: $h = 5000$
- Batch size: 1024

→ Shape of input data:

$$\underbrace{1024}_{\text{Batch size}} \times 300 \underbrace{\cdot}_{\text{Input size}} 300 = 92{,}160{,}000$$

→ Float32: 32 bit per number

→ $92{,}160{,}000 \cdot 32$ bit = 0.4 GB for the input

→ 1.8 GB + 0.4 GB = 2.2 GB only for the first layer! This is a lot!

Classifying Documents – Smarter Ideas

- Using SVMs or fully connected networks does not work well!

→ Use some other network architecture

Convolutional Neural Networks

- Focus on the local neighbourhood (similar to n-grams)

Today!

Recurrent Neural Networks

- Model the sentence as a temporal sequence of words

Next chapter!

Transformers

- Model words of a text as relationships between them

Later in the semester

4.2 Convolutional Neural Networks (in Computer Vision)

- What is the idea behind CNNs?
- Why are they used in computer vision?
- How to do backpropagation on a CNN?

Convolutional Neural Networks

- … are a type of „partially connected“ feedforward networks (details later)
- … originate from computer vision
- … are often used for image classification
- … outperform classical image recognition by a large margin
- … model features over „areas“ of growing size in images:
  - Early layers model local neighbourhoods (detect edges, …)
  - Later layers combine the features to detect larger objects

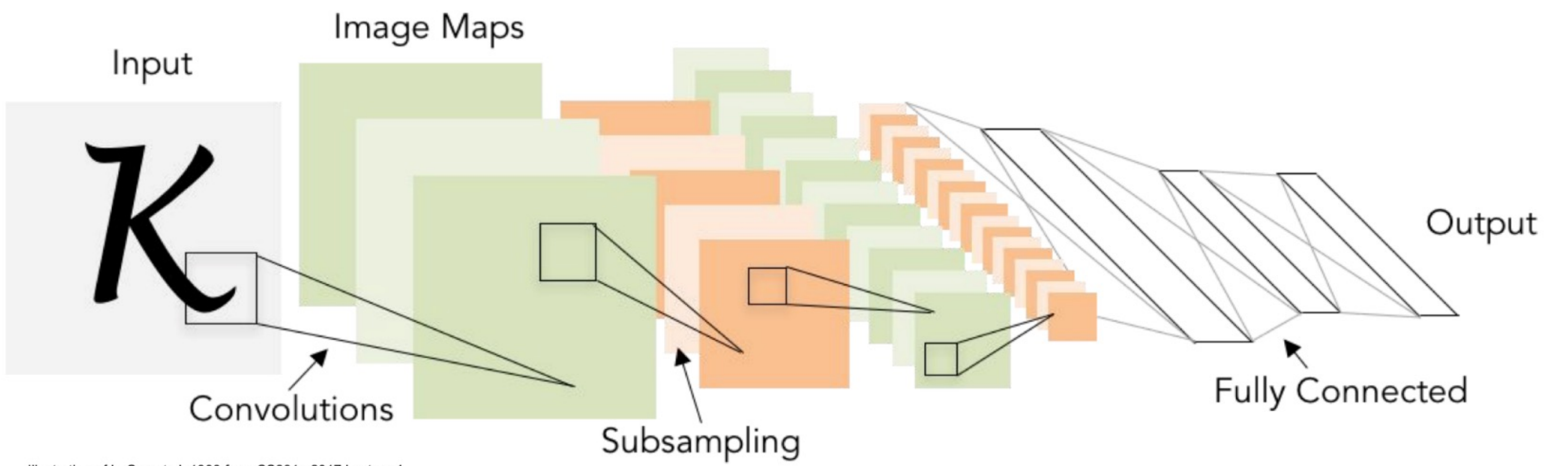


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

CNNs in Computer Vision

- Typical structure of a CNN:
  1. Some convolutional layers
  2. Pooling layer
  3. Repeat 1 and 2 as many times as your GPU allows
  4. Fully connected layers
  5. Softmax

Coming soon

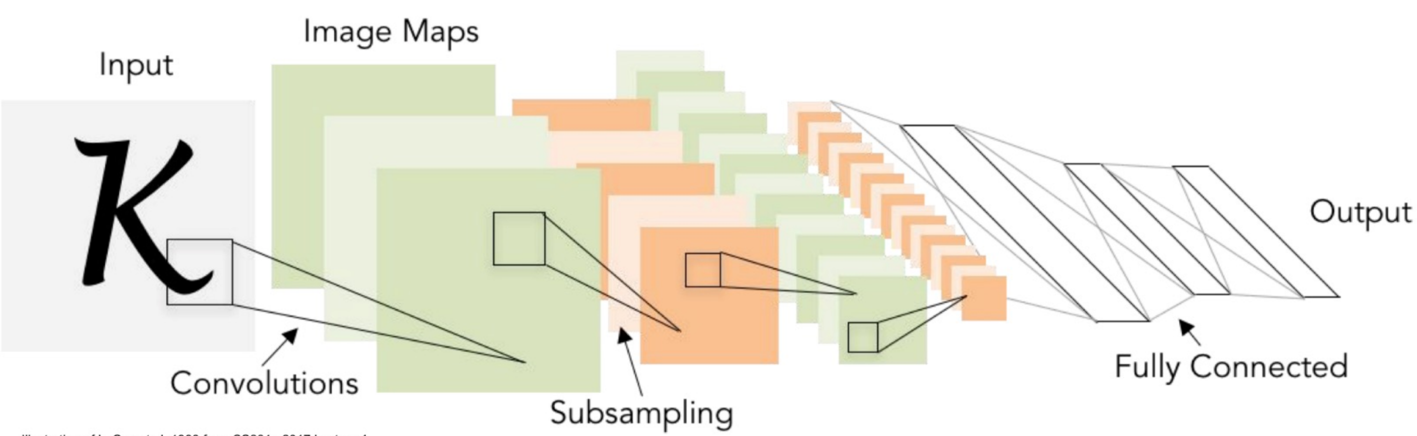


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

- Intuition:
  – Early layers extract very local features (edges, ...)
  – Later layers combine these to detect larger objects

A Bit of CNN History

- The following slides on the historical development of CNNs have been copied from the Stanford course CS231n by Fei-Fei Li

A bit of history:

Hubel & Wiesel,

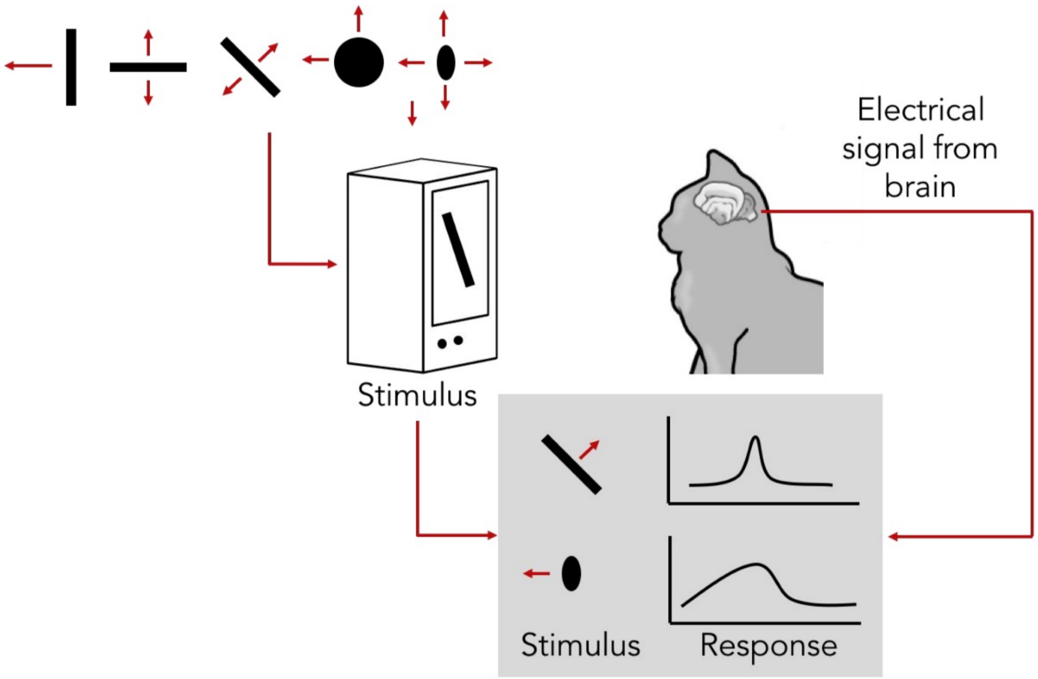
1959

RECEPTIVE FIELDS OF SINGLE NEURONES IN
THE CAT'S STRIATE CORTEX

1962

RECEPTIVE FIELDS, BINOCULAR INTERACTION
AND FUNCTIONAL ARCHITECTURE IN THE CAT'S VISUAL CORTEX

1968...



Hierarchical organization

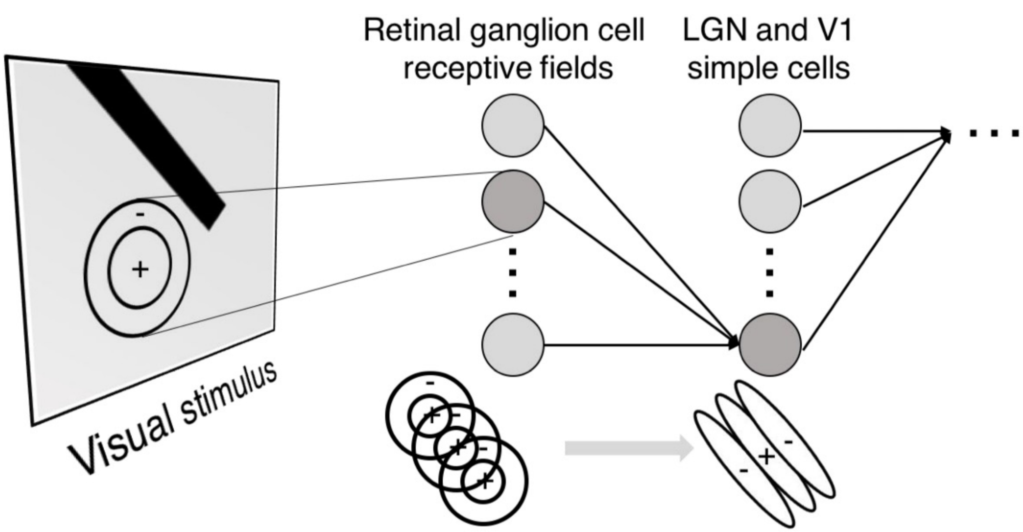


Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

Simple cells:
Response to light orientation

Complex cells:
Response to light orientation and movement

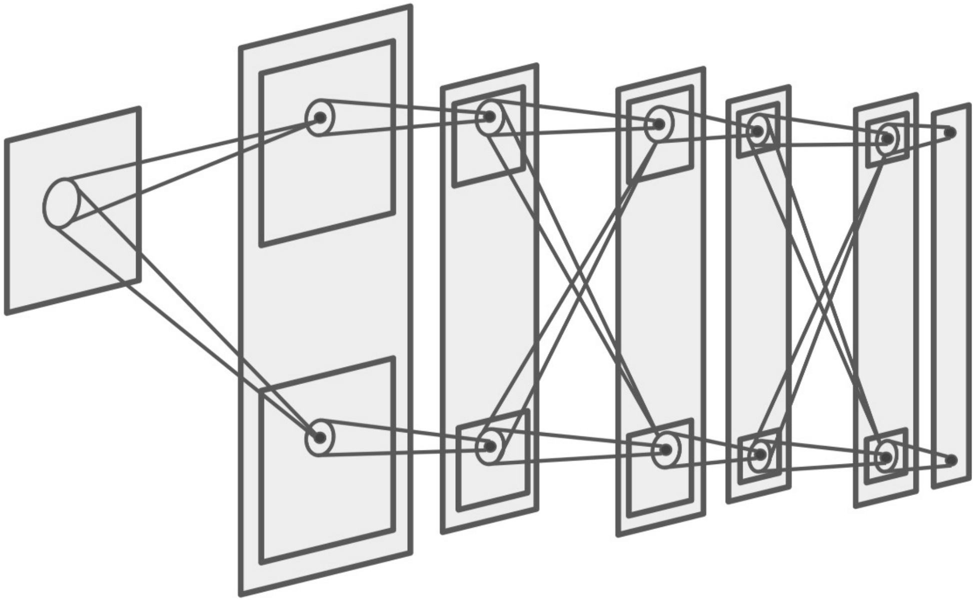
Hypercomplex cells:
response to movement with an end point

No response

Response (end point)

A bit of history:

Neocognitron
[Fukushima 1980]

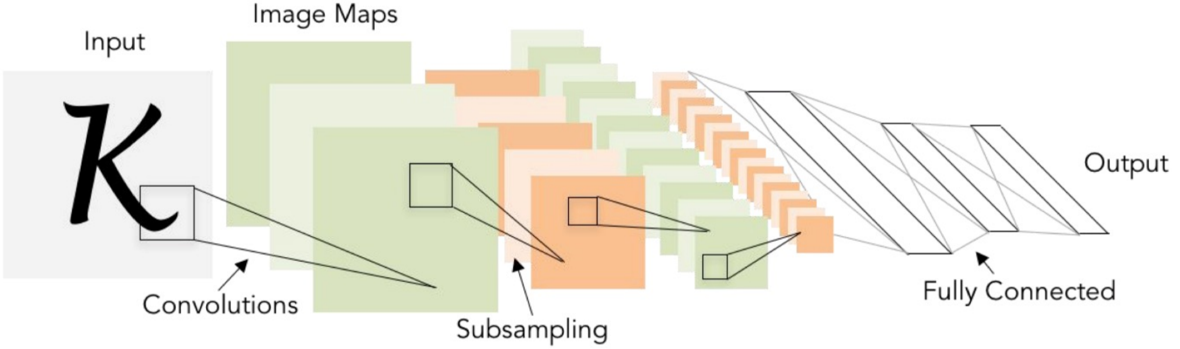


“sandwich” architecture (SCSCSC…)
simple cells: modifiable parameters
complex cells: perform pooling

A bit of history:

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]

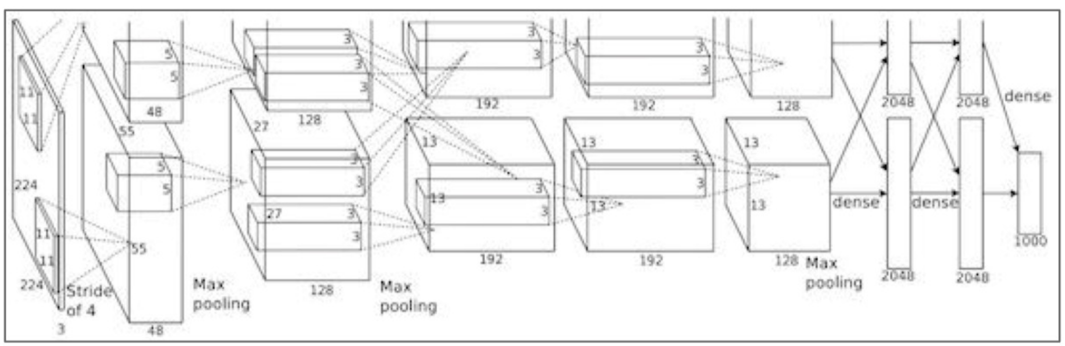


LeNet-5

A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

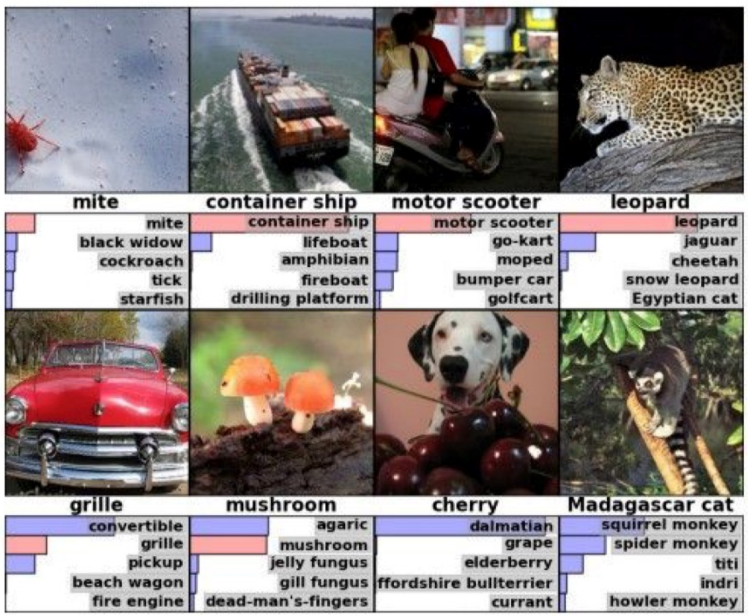
[Krizhevsky, Sutskever, Hinton, 2012]



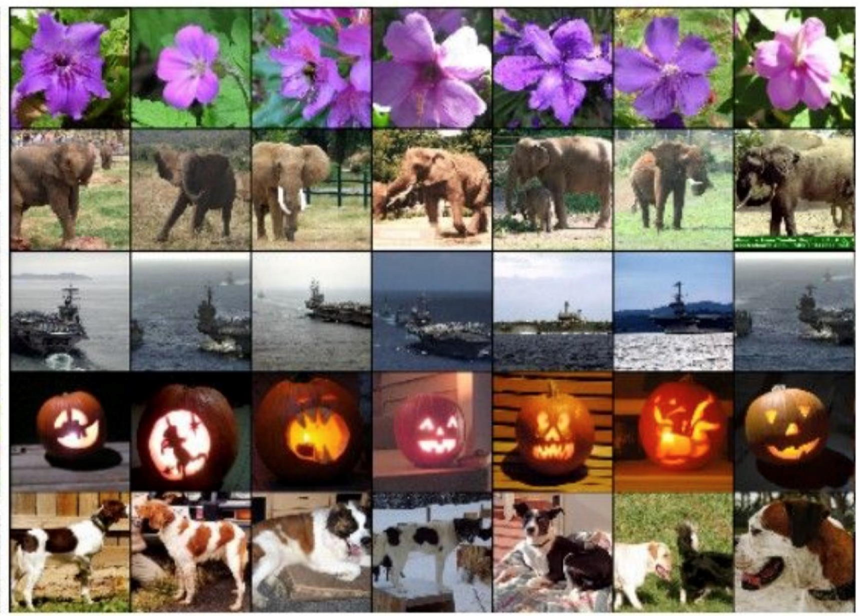
“AlexNet”

Fast-forward to today: ConvNets are everywhere

Classification



Retrieval



Convolutions

Convolutions

- To understand CNNs, take a look at their basic building blocks: **Convolutions**

- Convolution = classic tool in image processing (even without machine learning!)
- Intuition: Slide a function over an image, create a modified image
- Can model operations like edge detection, blurring, …

Convolutions

- In image processing:
  - A convolution $\mathrm{M} * C$ „applies“ a kernel matrix C to a (usually) larger matrix M by sliding C over M, constructing a new matrix M'
  - C is rotated by 180°
  - Values of M' constructed by summing up neighbouring values, weighted by C:

35	40	41	45	50
40	40	42	46	52
42	46	50	55	55
48	52	56	58	60
56	60	65	70	75

M

$*$

0	1	0
0	0	0
0	0	0

C

$=$

		56		

M' (partial)

$$M'_{a,b} = \sum_{i,j \in \{-1,0,1\}} M_{a-i,b-j} \cdot C_{i,j}$$

Convolution vs. Cross Correlation

- **Convolution ($\mathbf{A} * \boldsymbol{B}$)**: Rotate B by 180° and slide it over A
- **Cross Correlation ($\mathbf{A} \otimes \boldsymbol{B}$)**: Slide matrix B over A

- Similar, sometimes the same. When?

→ If $B = rot_{180°}\{B\}$

Convolution vs. Cross Correlation

- Convolution advantage: associative operation

$$(A * B) * C = A * (B * C)$$

→ Not important for CNNs: We always have non-linear activation function between convolutions

→ **We use Cross Correlations (but call them Convolutions)**

  – more intuitive
  – simpler to work with

Convolutions

$$\underbrace{\begin{array}{|c|c|c|c|c|}\hline 35 & 40 & 41 & 45 & 50 \\ \hline 40 & 40 & 42 & 46 & 52 \\ \hline 42 & 46 & 50 & 55 & 55 \\ \hline 48 & 52 & 56 & 58 & 60 \\ \hline 56 & 60 & 65 & 70 & 75 \\ \hline \end{array}}_{M} \ast \underbrace{\begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array}}_{C} = \underbrace{\begin{array}{|c|c|c|c|c|}\hline & & & & \\ \hline & & & & \\ \hline & & 56 & & \\ \hline & & & & \\ \hline & & & & \\ \hline \end{array}}_{M' \text{ (partial)}}$$

$$M'_{a,b} = \sum_{i,j \in \{-1,0,1\}} M_{a-i,b-j} \cdot C_{i,j}$$

Convolutions (Cross Correlation)

35	40	41	45	50
40	40	42	46	52
42	46	50	55	55
48	52	56	58	60
56	60	65	70	75

M

$\otimes$

0	1	0
0	0	0
0	0	0

C

=

		42		

M' (partial)

$$M'_{a,b} = \sum_{i,j \in \{-1,0,1\}} M_{a+i,b+j} \cdot C_{i,j}$$

Convolutions

$$\underbrace{\begin{bmatrix} 35 & 40 & 41 & 45 & 50 \\ 40 & 40 & 42 & 46 & 52 \\ 42 & 46 & 50 & 55 & 55 \\ 48 & 52 & 56 & 58 & 60 \\ 56 & 60 & 65 & 70 & 75 \end{bmatrix}}_{M} * \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{C} = \underbrace{\begin{bmatrix} & & & & \\ & & & & \\ & & 42 & & \\ & & & & \\ & & & & \end{bmatrix}}_{M' \text{ (partial)}}$$

$$M'_{a,b} = \sum_{i,j \in \{-1,0,1\}} M_{a+i,b+j} \cdot C_{i,j}$$

- Note: Problems on the edges of the matrix:

35	40	41	45	50
40	40	42	46	52
42	46	50	55	55
48	52	56	58	60
56	60	65	70	75

→ What to do here?

Skip the edges

- Smaller output ($M' < M$)
- Can quickly become a problem after multiple convolutions

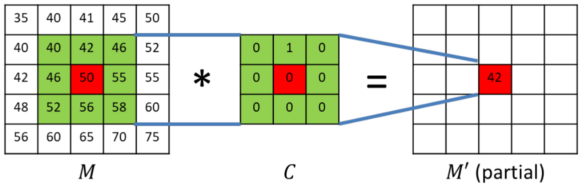
Padding

- Add „something“ around M

→ M' is of the same size as M

See next slide

Convolutions — Padding



$$M'_{a,b} = \sum_{i,j\in\{-1,0,1\}} M_{a+i,b+j} \cdot C_{i,j}$$

- Add „something“ around M to keep the same size for M'
- What to add?

→ Common Choice: Zeros

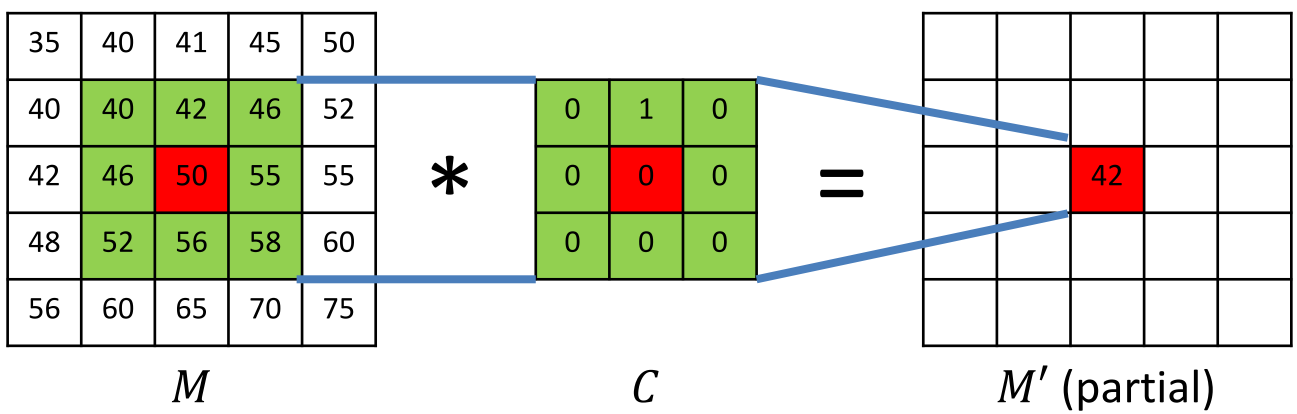
→ No influence on resulting value (summing up zeros)

35	40	41	45	50
40	40	42	46	52
42	46	50	55	55
48	52	56	58	60
56	60	65	70	75

0	0	0	0	0	0	0
0	35	40	41	45	50	0
0	40	40	42	46	52	0
0	42	46	50	55	55	0
0	48	52	56	58	60	0
0	56	60	65	70	75	0
0	0	0	0	0	0	0

Effect of Convolutions

- Look at our convolution from before again:



- Can you guess its effect on the image?

→ Applying C on M results in an image that is shifted one pixel towards the bottom!

Effect of Convolutions

- Some more convolutions:

Original Image
Source: http://xkcd.com

0	1	0
1	-4	1
0	1	0

Edge detection

$$\frac{1}{9} \cdot$$

1	1	1
1	1	1
1	1	1

Blurring

0	-1	0
-1	5	-1
0	-1	0

Sharpening

Convolutional Layers

Convolutional Layers in Neural Networks

- Main layer type in CNNs
- CNN learns one (or many) convolutions from the data
- Necessary parameters:
  - Number of filters = How many different filters (convolutions) to learn in the layer
  - Filter size = size of the convolution matrix
  - Stride = How many pixels to move right before applying the convolution again
  - Padding (no padding, zero padding, …)
  - Dilation = Spacing between the filter points

1	0	1	0	1
0	0	0	0	0
1	0	1	0	1
0	0	0	0	0
1	0	1	0	1

- Example: 1 filter of size 3×3 with a stride of 2 (no padding)

35	40	41	45	50
40	40	42	46	52
42	46	50	55	55
48	52	56	58	60
56	60	65	70	75

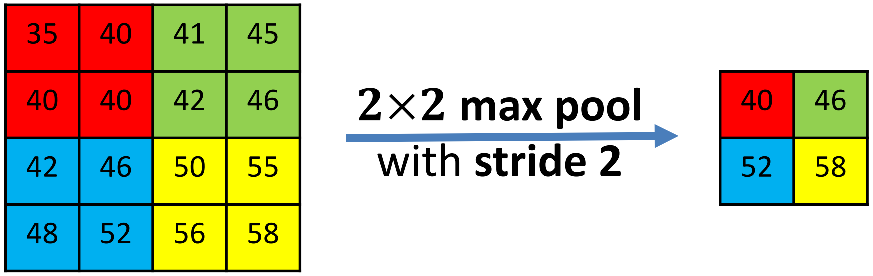
$*$

0	1	0
1	-4	1
0	1	0

Pooling Layers

Pooling Layers in Neural Networks

- Second component of CNNs: **Pooling Layers**
- Reduce the size of the input image in a predefined manner
- **No learned weights!** Purely static operation
- Common types:
  - **Max Pooling**
    - Extract the maximum of $n{\times}m$ (**pool size**) entries in the input
    - Move k (**stride**) entries ahead in the input
  - Average Pooling
    - Similar, but extract average instead of max



Pooling Layers

- Why use pooling layers?

→ Reduce the number of parameters in following layers

→ Not all filters activated on all pixels

  → Only use pixels in a close neighbourhood that provide the strongest signal

→ **What** is in the picture, not **where**!

Properties of CNNs

- Some notable properties of CNNs:
  - **Location Invariance:**
    Same filters applied at all positions of the image → Location of an edge/feature does not matter
  - **Compositionality:**
    Pooling layers shrink input → Convolutions of the same size in later stages of the network cover larger areas, they compose the information from earlier layers

CNNs vs. Fully Connected Networks

- Why use CNNs at all?

→ Parameter sharing (as with RNNs)

→ Massively fewer parameters than fully connected networks!

  - Example:
    - 300×300 pixel input, map to hidden layer of same size
    - Fully connected: $(300 \cdot 300) \cdot (300 \cdot 300) = 8{,}100{,}000{,}000$ parameters!
    - Convolutional with 100 filters of size 3×3: $100 \cdot 3 \cdot 3 = 900$ parameters ☺

→ Less RAM needed, less prone to overfitting

→ Convolutions heavily used in image processing
  → Even faster on GPUs than normal matrix multiplications!

Demo Time

- Andrej Karpathy: Implementation of CNNs in JavaScript

- Online-Demo with feature visualisation
- Training a CNN on the MNIST dataset (handwritten digits)

https://cs.stanford.edu/%7Ekarpathy/convnetjs/demo/mnist.html

Backpropagation in CNNs

Backpropagation in CNNs

- Weights in CNNs need to be tuned during training

→ How to do that?

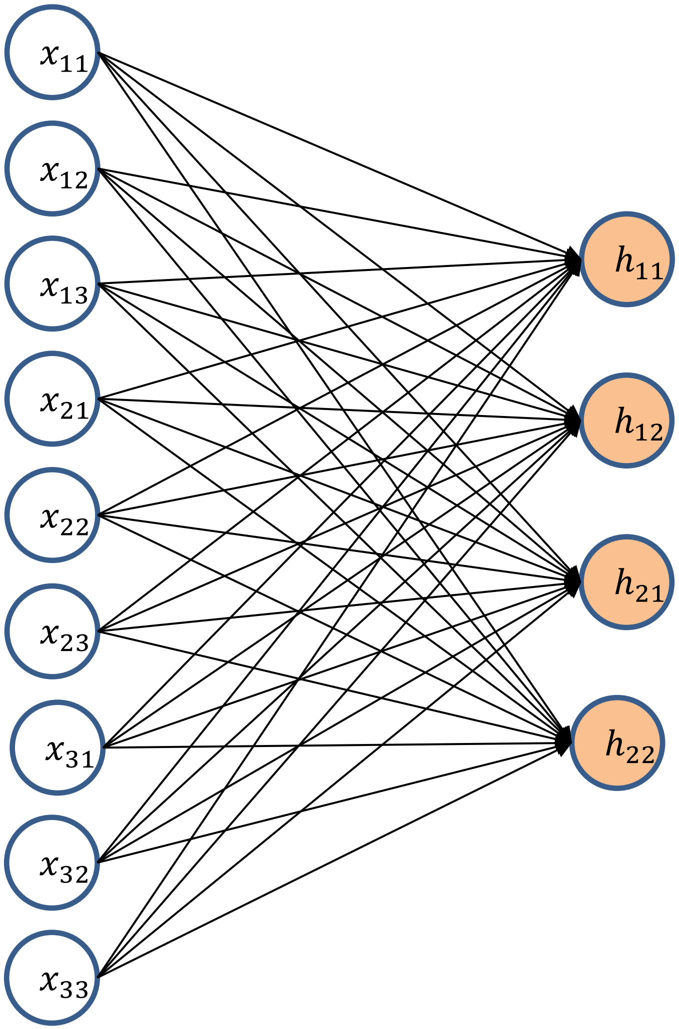
- Strategy:
  Reduce the problem to one that you already know!

x_{11}	x_{12}	x_{13}
x_{21}	x_{22}	x_{23}
x_{31}	x_{32}	x_{33}

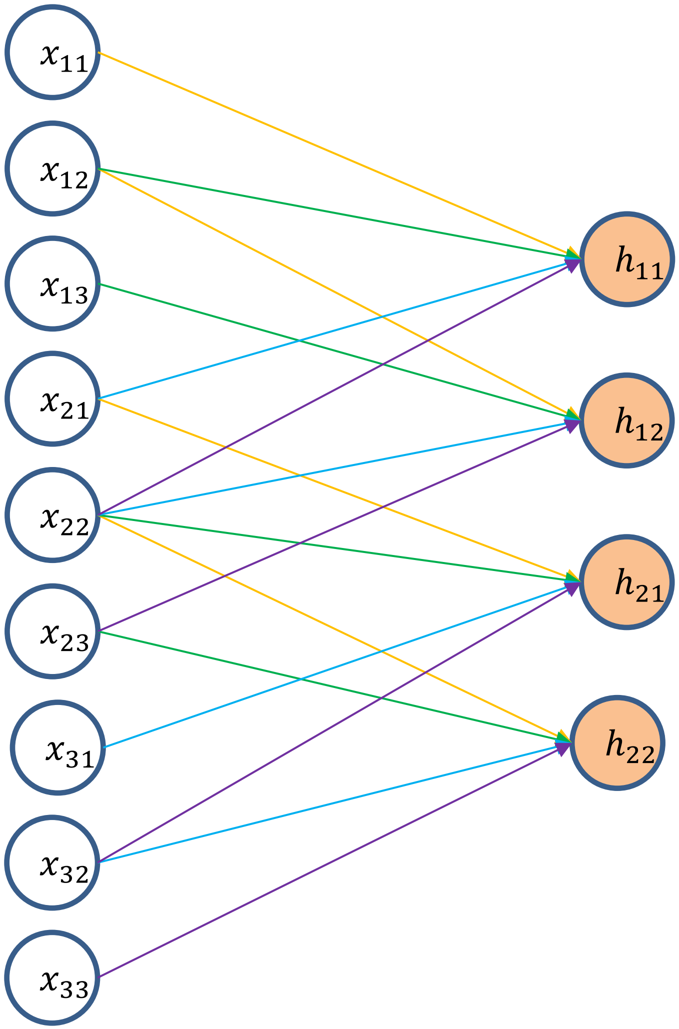
- We know Backpropagation for Fully Connected Networks
- A CNN is basically a „partially connected“ network
  (see following slides)

From FCNs to CNNs

Fully Connected Network



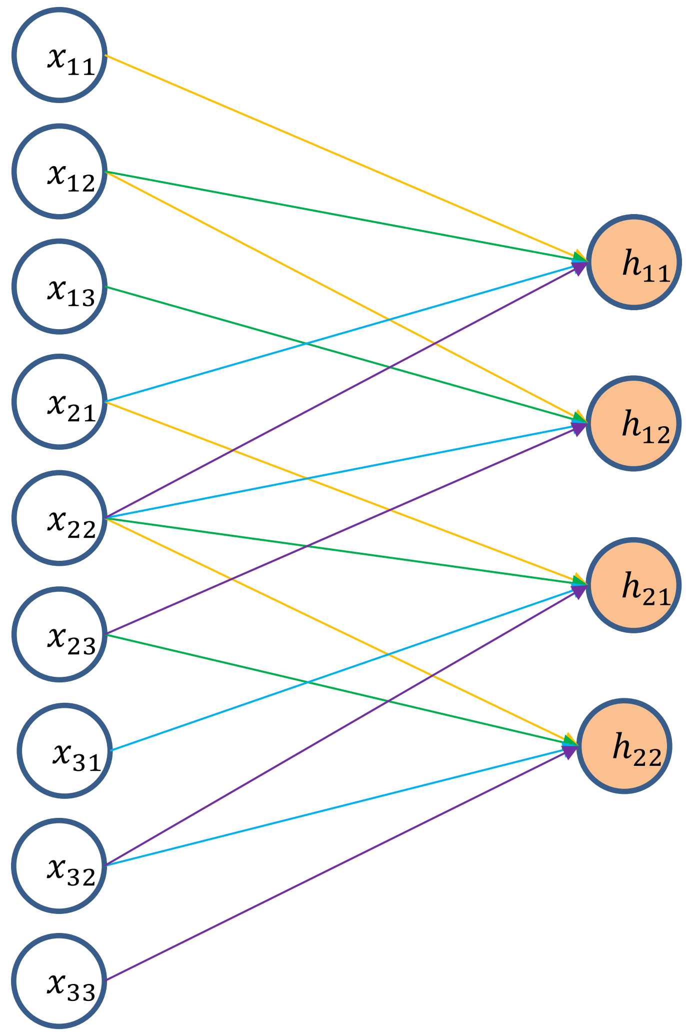
Convolutional Network



CNNs as Partially Connected Networks

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} * \begin{bmatrix} w_{11} & w_{12} \\ w_{12} & w_{22} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} \\ h_{12} & h_{22} \end{bmatrix}$$

$$h_{11} = w_{11}x_{11} + w_{12}\ x_{12} + w_{21}x_{21} + w_{22}x_{22}$$
$$h_{12} = w_{11}x_{12} + w_{12}\ x_{13} + w_{21}x_{22} + w_{22}x_{23}$$
$$h_{21} = w_{11}x_{21} + w_{12}\ x_{22} + w_{21}x_{31} + w_{22}x_{32}$$
$$h_{22} = w_{11}x_{22} + w_{12}\ x_{23} + w_{21}x_{32} + w_{22}x_{33}$$



Backpropagation — Update steps

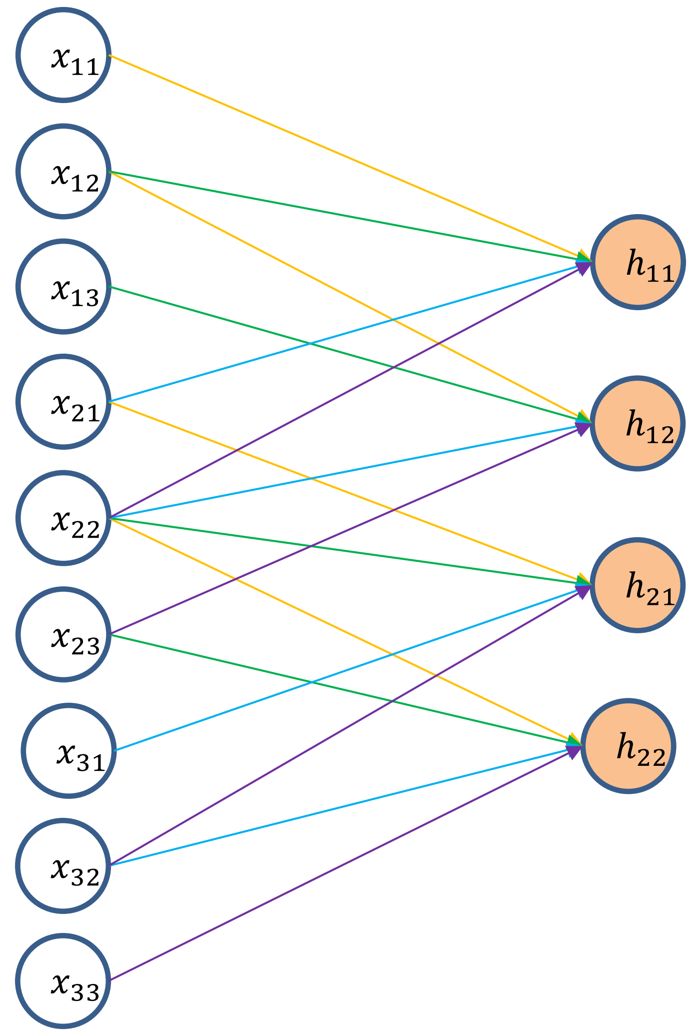
- Notation:
  To save space, define $\delta_v := \frac{\partial L}{\partial v}$ for the following slides
  (placeholder v will be replaced by x or some other variable)

- Remember:
  For each layer, we need to compute two things with Backpropagation:
  1. The gradient that flows to the previous layer **(Propagation)**
  2. The update for the parameters in this layer (**Weight Update**)

Remember the Network Structure:

$$\begin{array}{|c|c|c|}\hline x_{11} & x_{12} & x_{13} \\ \hline x_{21} & x_{22} & x_{23} \\ \hline x_{31} & x_{32} & x_{33} \\ \hline \end{array} * \begin{array}{|c|c|}\hline w_{11} & w_{12} \\ \hline w_{12} & w_{22} \\ \hline \end{array} = \begin{array}{|c|c|}\hline h_{11} & h_{12} \\ \hline h_{12} & h_{22} \\ \hline \end{array}$$

$$h_{11} = w_{11}x_{11} + w_{12}\ x_{12} + w_{21}x_{21} + w_{22}x_{22}$$
$$h_{12} = w_{11}x_{12} + w_{12}\ x_{13} + w_{21}x_{22} + w_{22}x_{23}$$
$$h_{21} = w_{11}x_{21} + w_{12}\ x_{22} + w_{21}x_{31} + w_{22}x_{32}$$
$$h_{22} = w_{11}x_{22} + w_{12}\ x_{23} + w_{21}x_{32} + w_{22}x_{33}$$



Update Step 1: Error Propagation in Partially Conn. Net

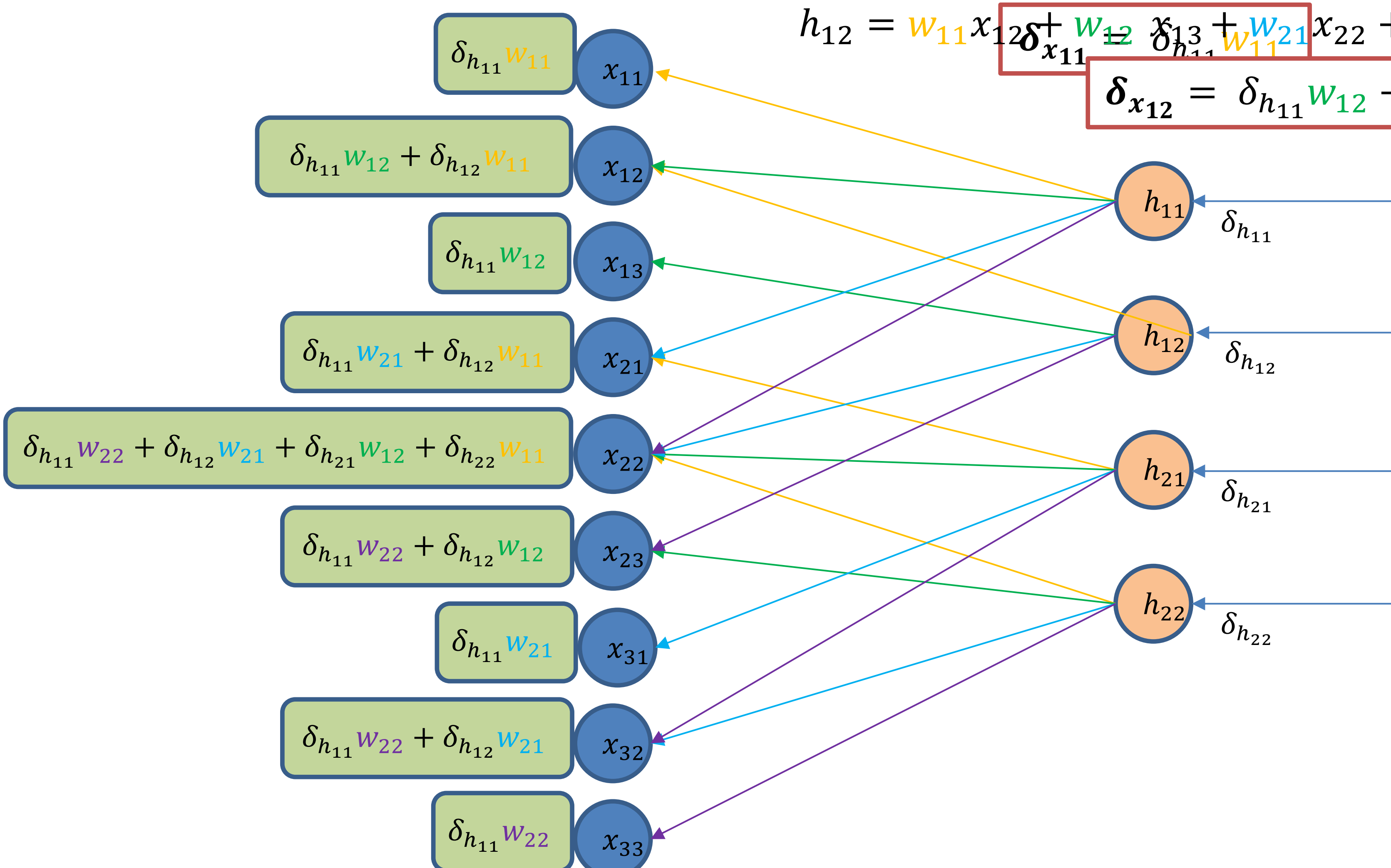
Find $\delta_{x_{mn}}$: Look at all h_{ij} depending on x_{mn}!

$$h_{11} = w_{11}x_{11} + w_{12}\ x_{12} + w_{21}x_{21} + w_{22}x_{22}$$

$$h_{12} = w_{11}x_{12} + w_{12}\ x_{13} + w_{21}x_{22} + w_{22}x_{23}$$

$$\boxed{\delta_{x_{11}} = \delta_{h_{11}} w_{11}}$$

$$\boxed{\delta_{x_{12}} = \delta_{h_{11}} w_{12} + \delta_{h_{12}} w_{11}}$$



Update Step 1 — Error Propagation

- We can do backpropagation just like in a fully connected network!

- But this is element-wise, and thus slow

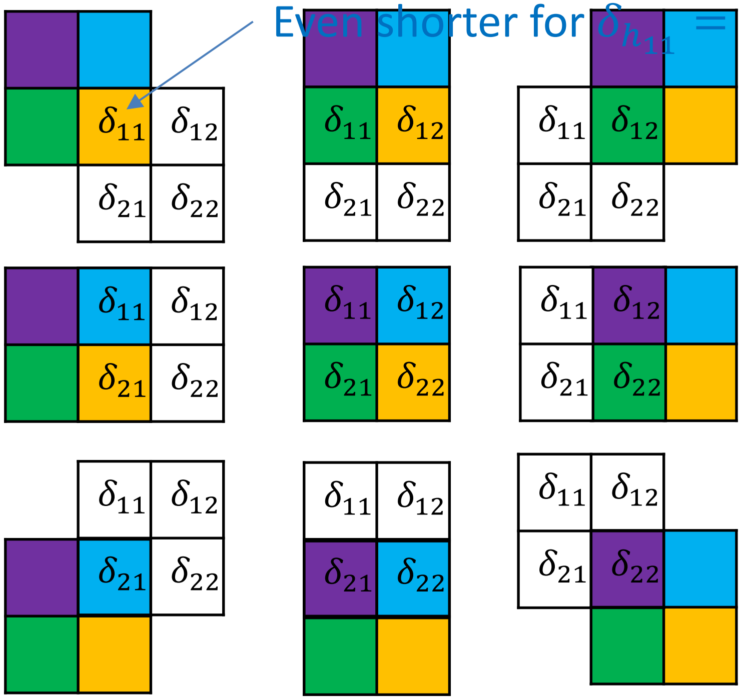
→ How to do it in a smarter (i.e., faster) way?

→ Let‘s find some matrix operation that does the trick!

Update Step 1: Error Propagation as Convolution

$$\begin{bmatrix} \delta x_{11} & \delta x_{12} & \delta x_{13} \\ \delta x_{21} & \delta x_{22} & \delta x_{23} \\ \delta x_{31} & \delta x_{32} & \delta x_{33} \end{bmatrix} = \begin{bmatrix} \delta_{11} & \delta_{12} \\ \delta_{21} & \delta_{22} \end{bmatrix} * \begin{bmatrix} w_{22} & w_{21} \\ w_{12} & w_{11} \end{bmatrix}$$

???



Even shorter for $\delta_{h_{11}} = \frac{\partial L}{\partial h_{11}}$

Same as w, just rotated by 180°!

$$\delta x_{11} = \delta_{11} w_{11}$$

$$\delta x_{12} = \delta_{11} w_{12} + \delta_{12} w_{11}$$

$$\delta x_{13} = \delta_{12} w_{12}$$

$$\delta x_{21} = \delta_{11} w_{21} + \delta_{21} w_{11}$$

$$\delta x_{22} = \delta_{11} w_{22} + \delta_{12} w_{21} + \delta_{21} w_{12} + \delta_{22} w_{11}$$

$$\delta x_{23} = \delta_{12} w_{22} + \delta_{22} w_{12}$$

$$\delta x_{31} = \delta_{12} w_{21}$$

$$\delta x_{32} = \delta_{11} w_{22} + \delta_{12} w_{21}$$

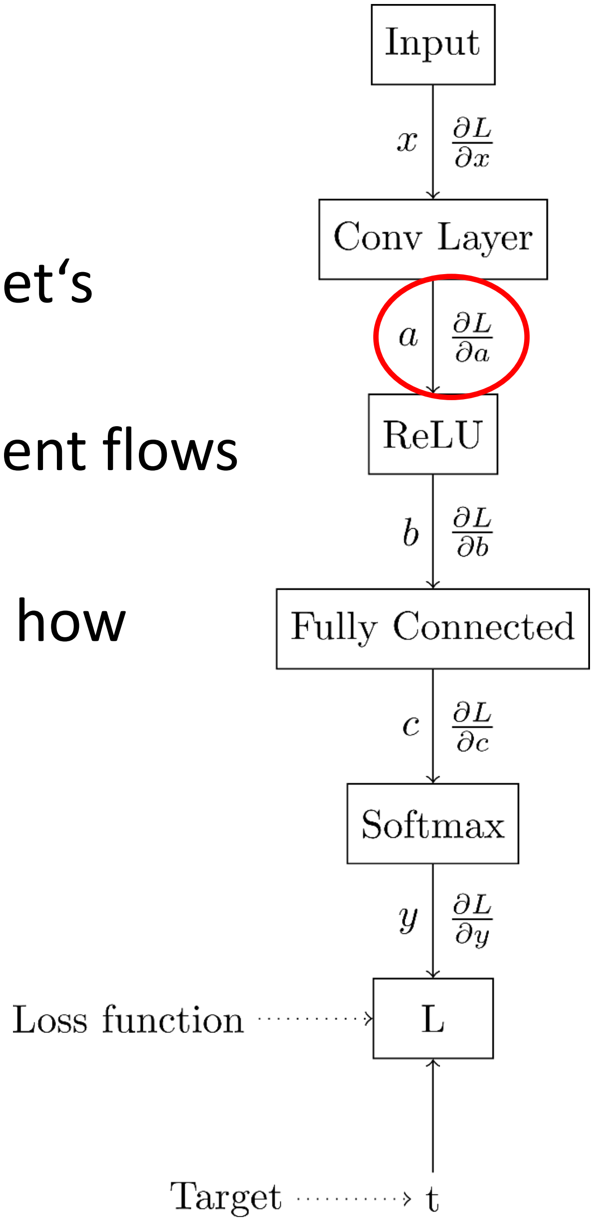
$$\delta x_{33} = \delta_{11} w_{22}$$

This looks suspiciously like a convolution!

Update Step 1: Error Propagation

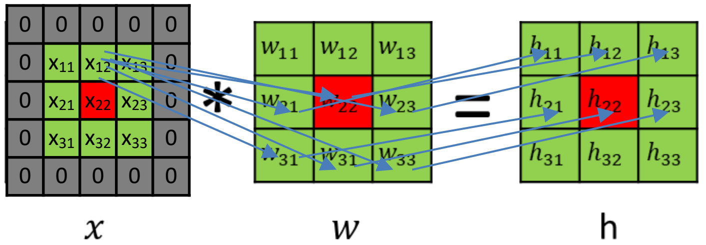
- Now that we know the intuition behind this, let‘s derive the general rule mathematically!
- Given a CNN, we want to know how the gradient flows through a convolutional layer
- In the example on the right, we already know how to get $\frac{\partial L}{\partial a}$ using backpropagation

→ Derive a formula for $\frac{\partial L}{\partial x}$

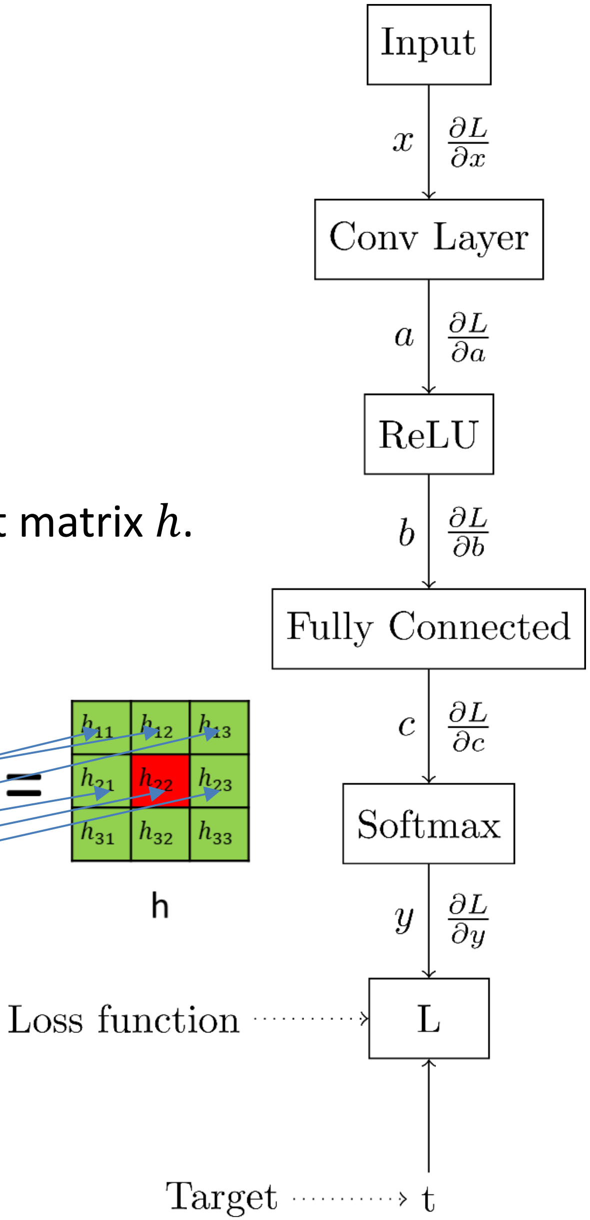


Update Step 1: Error Propagation

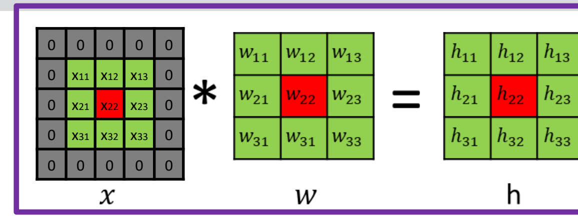
- The input to a CNN is usually a matrix x
- Thus:
  - Get a gradient for each entry in x
  - Each value $x_{a,b}$ influences its „neighbours" in the output matrix h.
  - For a kernel of size 3×3, $x_{a,b}$ will have influence on
    $h_{a-1,b-1}, h_{a-1,b}, h_{a-1,b+1},$
    $h_{a,b-1}, h_{a,b}, h_{a,b+1},$
    $h_{a+1,b-1}, h_{a+1,b}, h_{a+1,b+1}$



- Next slide: derivation of a formula for $\frac{\partial L}{\partial x}$



Update Step 1: Error Propagation



0.5 height of the filter matrix, $\mathrm{H}'_{\mathrm{w}} = \left\lfloor \frac{height}{2} \right\rfloor$

0.5 width of the filter matrix

Derivative from the next layer („Nachdifferenzieren“)

$$\frac{\partial \mathrm{L}}{\partial x_{a,b}} = \sum_{m=-H'_{\mathrm{w}}}^{\mathrm{H}'_{\mathrm{w}}} \sum_{n=-W'_{\mathrm{w}}}^{W'_{\mathrm{w}}} \delta_{a+m,b+n} \frac{\partial h_{a+m,b+n}}{\partial x_{a,b}}$$

Derivative of the convolution‘s output w.r.t its input.
$h_{a+m,b+n}$ are all outputs depending on $x_{a,b}$.

Update Step 1: Error Propagation

$$\underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & x_{11} & x_{12} & x_{13} & 0 \\ 0 & x_{21} & x_{22} & x_{23} & 0 \\ 0 & x_{31} & x_{32} & x_{33} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{x} * \underbrace{\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{31} & w_{33} \end{bmatrix}}_{w} = \underbrace{\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}}_{\mathrm{h}}$$

0.5 height of the filter matrix, $\mathrm{H}'_{\mathrm{w}} = \left\lfloor \frac{height}{2} \right\rfloor$

0.5 width of the filter matrix

Derivative from the next layer („Nachdifferenzieren“)

Derivative of the convolution‘s output w.r.t its input. $h_{a+m,b+n}$ are all outputs depending on $x_{a,b}$.

$$\frac{\partial \mathrm{L}}{\partial x_{a,b}} = \sum_{m=-H'_{\mathrm{w}}}^{\mathrm{H}'_{\mathrm{w}}} \sum_{n=-W'_{\mathrm{w}}}^{W'_{\mathrm{w}}} \delta_{a+m,b+n} \frac{\partial h_{a+m,b+n}}{\partial x_{a,b}}$$

Simplify the red expression:

$$\frac{\partial h_{a+m,b+n}}{\partial x_{a,b}} = \frac{\partial}{\partial x_{a,b}} \underbrace{\sum_{m'=-H'_{\mathrm{w}}}^{\mathrm{H}'_{\mathrm{w}}} \sum_{n'=-W'_{\mathrm{w}}}^{W'_{\mathrm{w}}} w_{m'+H'_{w}+1,n'+W'_{w}+1} \cdot x_{a+m+m',b+n+n'}}_{h_{a+m,b+n}}$$

Update Step 1: Error Propagation

$$\underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & x_{11} & x_{12} & x_{13} & 0 \\ 0 & x_{21} & x_{22} & x_{23} & 0 \\ 0 & x_{31} & x_{32} & x_{33} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{x} * \underbrace{\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{31} & w_{33} \end{bmatrix}}_{w} = \underbrace{\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}}_{\mathrm{h}}$$

0.5 height of the filter matrix, $\mathrm{H}'_{\mathrm{w}} = \left\lfloor \frac{height}{2} \right\rfloor$

0.5 width of the filter matrix

Derivative from the next layer („Nachdifferenzieren“)

Derivative of the convolution‘s output w.r.t its input. $h_{a+m,b+n}$ are all outputs depending on $x_{a,b}$.

$$\frac{\partial \mathrm{L}}{\partial x_{a,b}} = \sum_{m=-H'_{\mathrm{w}}}^{\mathrm{H}'_{\mathrm{w}}} \sum_{n=-W'_{\mathrm{w}}}^{W'_{\mathrm{w}}} \delta_{a+m,b+n} \frac{\partial h_{a+m,b+n}}{\partial x_{a,b}}$$

Simplify the red expression:

$$\frac{\partial h_{a+m,b+n}}{\partial x_{a,b}} = \frac{\partial}{\partial x_{a,b}} \underbrace{\sum_{m'=-H'_{\mathrm{w}}}^{\mathrm{H}'_{\mathrm{w}}} \sum_{n'=-W'_{\mathrm{w}}}^{W'_{\mathrm{w}}} w_{m'+H'_{w}+1,n'+W'_{w}+1} \cdot x_{a+m+m',b+n+n'}}_{h_{a+m,b+n}}$$

$$= w_{-m+H'_{w}+1,-n+W'_{w}+1}$$

All other $w_{\dots}$ are not multiplied with $x_{a,b}$ and thus disappear!

Update Step 1: Error Propagation

$$\underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & x_{11} & x_{12} & x_{13} & 0 \\ 0 & x_{21} & x_{22} & x_{23} & 0 \\ 0 & x_{31} & x_{32} & x_{33} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{x} * \underbrace{\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{31} & w_{33} \end{bmatrix}}_{w} = \underbrace{\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}}_{\mathrm{h}}$$

0.5 height of the filter matrix, $\mathrm{H}'_\mathrm{w} = \left\lfloor \frac{height}{2} \right\rfloor$

0.5 width of the filter matrix

Derivative from the next layer („Nachdifferenzieren“)

$$\frac{\partial \mathrm{L}}{\partial x_{a,b}} = \sum_{m=-\mathrm{H}'_\mathrm{w}}^{\mathrm{H}'_\mathrm{w}} \sum_{n=-W'_\mathrm{w}}^{W'_\mathrm{w}} \delta_{a+m,b+n} \frac{\partial h_{a+m,b+n}}{\partial x_{a,b}}$$

Derivative of the convolution‘s output w.r.t its input. $h_{a+m,b+n}$ are all outputs depending on $x_{a,b}$.

Simplify the red expression:

$$\frac{\partial h_{a+m,b+n}}{\partial x_{a,b}} = \frac{\partial}{\partial x_{a,b}} \underbrace{\sum_{m'=-H'_\mathrm{w}}^{\mathrm{H}'_\mathrm{w}} \sum_{n'=-W'_\mathrm{w}}^{W'_\mathrm{w}} w_{m'+H'_w+1,n'+W'_w+1} \cdot x_{a+m+m',b+n+n'}}_{h_{a+m,b+n}}$$

All other $w_{...}$ are not multiplied with $x_{a,b}$ and thus disappear!

$$= w_{-m+H'_w+1,-n\,+W'_w+1}$$

This is the „rotated“ *convolution* from before!

$$\frac{\partial L}{\partial x_{a,b}} = \sum_{m=-H'_\mathrm{w}}^{H'_\mathrm{w}} \sum_{n=-W'_\mathrm{w}}^{W'_\mathrm{w}} \delta_{a+m,b+n} w_{-m+H'_w+1,-n\,+W'_w+1}$$

$$= \delta_{a-H'_w:a+H'_w,b-W'_w:b+W'_w} * rot_{180°}\{w\}$$

Recall

Backpropagation — Update steps

- Notation:
  To save space, define $\delta_v := \frac{\partial L}{\partial v}$ for the following slides

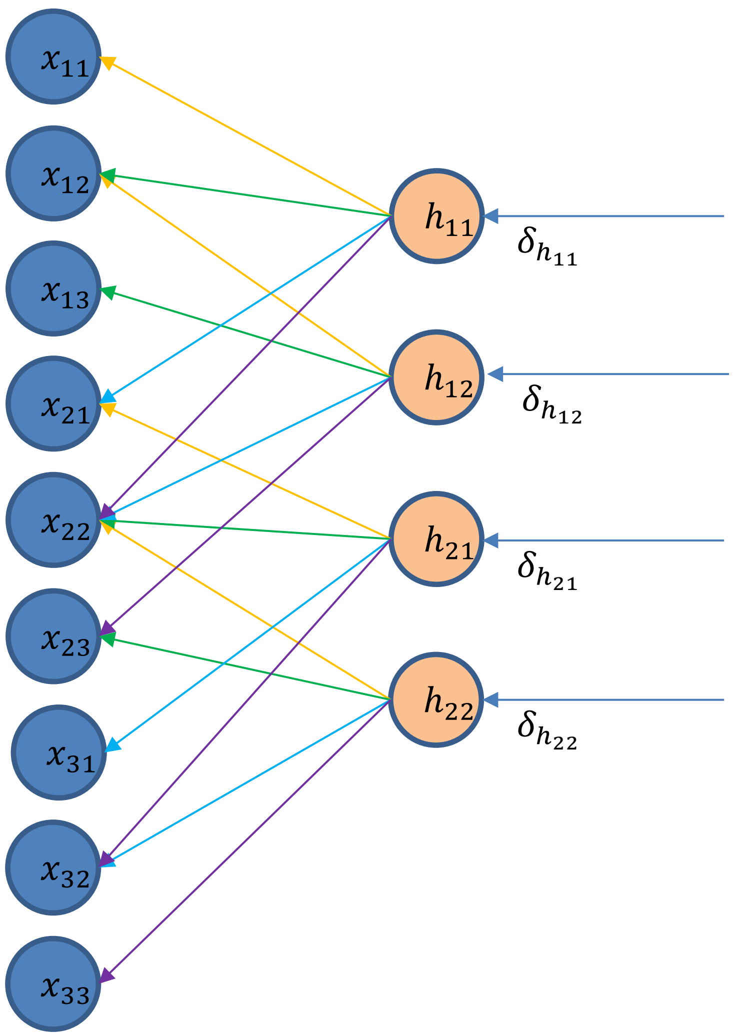
- Remember:
  For each layer, we need to compute two things with Backpropagation:
  1. The gradient that flows to the previous layer (**Propagation**) ✅
  2. **The update for the parameters in this layer (Weight Update)**

Update Step 2: Weights

- Up next: Updating the weights

→ Same procedure as for the error propagation:
  - Derive the update as an element-wise operation
  - Generalise to a matrix operation

Update Step 2: Weight Update in Partially Conn. Net



$$h_{11} = w_{11}x_{11} + w_{12}\,x_{12} + w_{21}x_{21} + w_{22}x_{22}$$
$$h_{12} = w_{11}x_{12} + w_{12}\,x_{13} + w_{21}x_{22} + w_{22}x_{23}$$
$$h_{21} = w_{11}x_{21} + w_{12}\,x_{22} + w_{21}x_{31} + w_{22}x_{32}$$
$$h_{22} = w_{11}x_{22} + w_{12}\,x_{23} + w_{21}x_{32} + w_{22}x_{33}$$

$$\delta_{w_{11}} = x_{11}\delta_{h_{11}} + x_{12}\delta_{h_{12}} + x_{21}\delta_{h_{21}} + x_{22}\delta_{h_{22}}$$
$$\delta_{w_{12}} = x_{12}\delta_{h_{11}} + x_{13}\delta_{h_{12}} + x_{22}\delta_{h_{21}} + x_{23}\delta_{h_{22}}$$
$$\delta_{w_{21}} = x_{21}\delta_{h_{11}} + x_{22}\delta_{h_{12}} + x_{31}\delta_{h_{21}} + x_{32}\delta_{h_{22}}$$
$$\delta_{w_{22}} = x_{22}\delta_{h_{11}} + x_{23}\delta_{h_{12}} + x_{32}\delta_{h_{21}} + x_{33}\delta_{h_{22}}$$

Update Step 2: Weight Update as Convolution

$$\begin{bmatrix} \delta_{w_{11}} & \delta_{w_{12}} \\ \delta_{w_{21}} & \delta_{w_{22}} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} * \begin{bmatrix} \delta_{h_{11}} & \delta_{h_{12}} \\ \delta_{h_{21}} & \delta_{h_{22}} \end{bmatrix}$$

$$\delta_{w_{11}} = x_{11}\delta_{h_{11}} + x_{12}\delta_{h_{12}} + x_{21}\delta_{h_{21}} + x_{22}\delta_{h_{22}}$$

$$\delta_{w_{12}} = x_{12}\delta_{h_{11}} + x_{13}\delta_{h_{12}} + x_{22}\delta_{h_{21}} + x_{23}\delta_{h_{22}}$$

$$\delta_{w_{21}} = x_{21}\delta_{h_{11}} + x_{22}\delta_{h_{12}} + x_{31}\delta_{h_{21}} + x_{32}\delta_{h_{22}}$$

$$\delta_{w_{22}} = x_{22}\delta_{h_{11}} + x_{23}\delta_{h_{12}} + x_{32}\delta_{h_{21}} + x_{33}\delta_{h_{22}}$$

Update Step 2: Weight Update as Convolution

Width of the output

Height of the output

Derivative from the next layer („Nachdifferenzieren“)

Half the height and width of the output

$$\frac{\partial L}{\partial w_{a,b}} = \sum_{i=1}^{H_h} \sum_{j=1}^{W_h} \delta_{i,j} \frac{\partial h_{i,j}}{\partial w_{a,b}}$$

Derivative of the convolution‘s output w.r.t its weights. $h_{i,j}$ are all outputs depending on $w_{a,b}$.

$$\frac{\partial h_{i,j}}{\partial w_{a,b}} = \frac{\partial}{\partial w_{a,b}} \sum_{m'=-H_h'}^{H_h'} \sum_{n'=-W_h'}^{W_h'} x_{i+m',j+n'} \cdot w_{m'+H_h'+1,n'+W_h'+1} = x_{i+a-H_h'-1,j+b-W_h'-1}$$

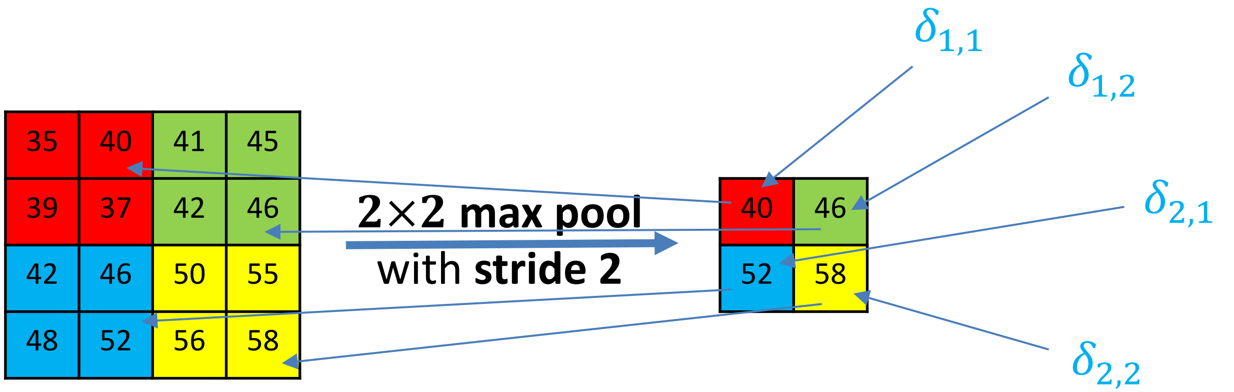
$$\frac{\partial L}{\partial w_{a,b}} = \sum_{i=1}^{H_h} \sum_{j=1}^{W_h} \delta_{i,j} x_{i+a-H_h'-1,j+b-W_h'-1}$$

$$= \delta * x_{a-H_h':a+H_h',b-W_h':b+W_h'}$$

This also looks like a *convolution*!

Pooling Layer

- Pooling layers do not have any weights, no calculation needed
- Error is passed on to previous layer:
  - For max pooling, the largest input receives the gradient δ backpropagated from the next layer, all other inputs have gradient 0
  - For average pooling with pooling size $n \times m$, all inputs receive gradient $\frac{\delta}{m \cdot n}$



4.3 Convolutional Neural Networks in NLP

- Are CNNs useful in NLP?
- If yes, how are they used?

CNNs in NLP

- So far: Only described use in Computer Vision…
- … but this is an NLP lecture!

- More recently, CNNs became popular in NLP, too:
  - Kalchbrenner et al., 2014: A Convolutional Neural Network for Modelling Sentences
  - **Kim, 2014: Convolutional Neural Networks for Sentence Classification**
  - Nguyen and Grishman, 2015: Relation Extraction: Perspective from Convolutional Neural Networks
  - …

CNNs for Sentence Classification

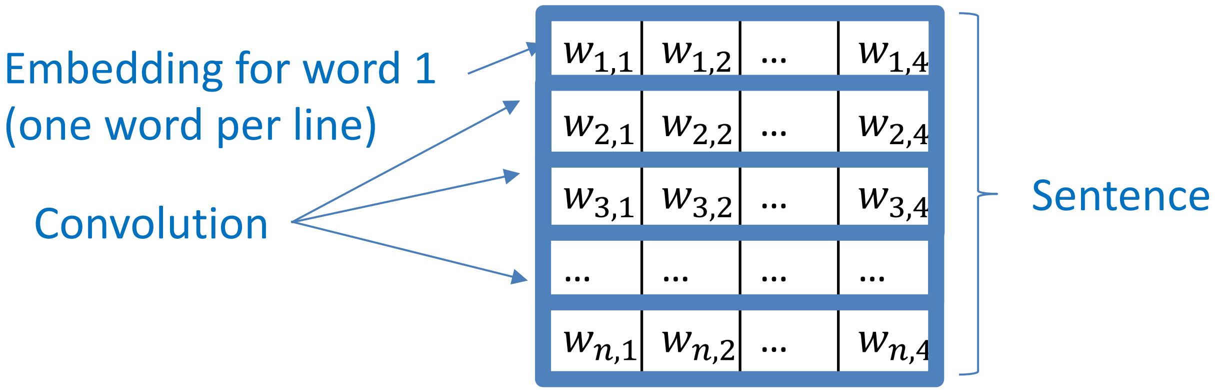
- Kim, 2014 (EMNLP)
- Very simple CNN model…
- … with **very** good results!
- Beats previous state-of-the-art in several tasks
  - Sentiment Analysis
  - Subjectivity Detection*
  - Question Answering

* didn‘t beat state-of-the-art, but came very close

Kim, Yoon: Convolutional Neural Networks for Sentence Classification. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014,* S. 1746--1751

CNNs for Sentence Classification

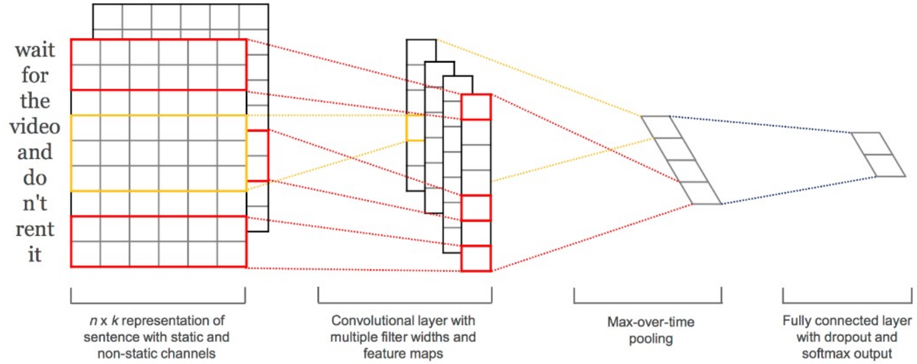
- Input representation: Concatenated word embeddings



- Convolutions over full embeddings!

→ Filter size $n \times m$ (n = variable, m = length of the embeddings)

CNNs for Sentence Classification



- Network architecture:
  - No stacked convolutional layers!
  - One layer of convolutions
    - Different filter sizes: $n \in \{3,4,5\}$
    - Each with ~100-600 filters
    - Stride 1 → Do not skip any words/n-grams
  - Concatenate the output of all filters along the depth axis
  - „Max-over-time-pooling“:
    Max Pooling over the full filter output → Select the highest output for each filter
  - Fully connected layer
  - Dropout
  - Softmax

CNN for Sentence Classification — Word Embeddings

- Word embeddings used to encode the input
- Evaluates multiple variants:
  - Randomly initialise embeddings, train with model (**cnn-rand**)
  - Initialise word embeddings with Word2Vec, keep fixed (**cnn-static**)
  - Initialise word embeddings with Word2Vec, train with model (**cnn-nonstatic**)

- Finding:
  Using pre-trained embeddings and further optimising them for the task at hand works best!

CNN for Sentence Classification — Word Embeddings

- Trick: Multi-channel word embeddings (**cnn-multichannel**)
  - Represent input as 3-dimensional matrix
  - 3rd dimension: different, independent word embedding vectors
  - Convolutions computed slightly differently (see papers for details), general ideas still apply

  - During training:
    - keep one dimension fixed
    - train the other

  - Effect:
    Keep information from original embeddings, but also include „extra“ for the task/dataset

CNN for Sentence Classification — Regularisation

- Multiple regularisation methods used:
  - Dropout
    - Apply Dropout after the fully connected layer
  - L2-maxnorm constraint
    - Set a hard limit s on l2-norm for weights w of the fully connected layer
    - If, after the update step, $||w||_2 > s$, rescale w to $||w||_2 = s$
    - Prevents single weights from getting too large

CNN for Sentence Classification — Evaluation

- Evaluation conducted on multiple standard datasets for different tasks
- We take a closer look at the following:
  - Question Classification: TREC (Li and Roth, 2002)
  - Sentiment Analysis: Stanford Sentiment Treebank (SST; Socher et al., 2013)

CNN for Sentence Classification — Evaluation

- Question Classification: TREC (Li and Roth, 2002)
  - Given a question, classify it into one of 6 types (questions about abbreviations, entites, descriptions, humans, locations or numerical values)
  - Training set: 5952 questions
  - Test set: 500 questions

Question	Label
What films featured the character Popeye Doyle?	Entity
How many Community Chest cards are there in Monopoly?	Numerical value
Where do the adventures of `` The Swiss Family Robinson '' take place?	Location
How can I register my website in Yahoo for free ?	Description

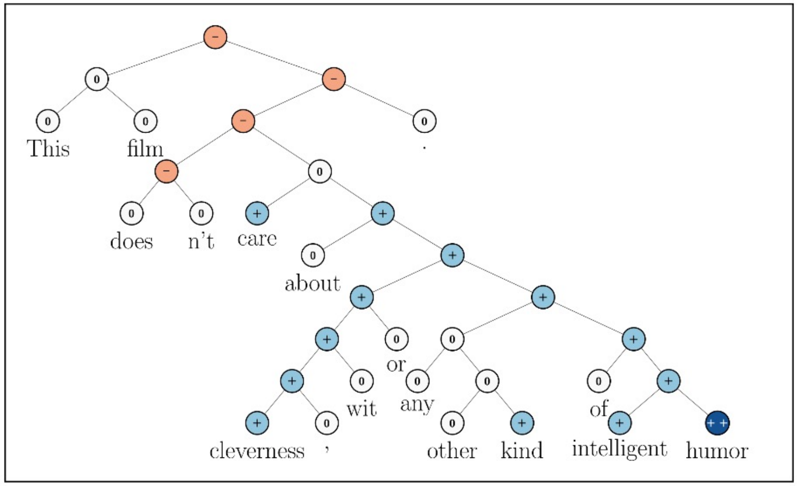
CNNs for Sentence Classification — Evaluation

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	–	–	–	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	–	–	–	–
RNTN (Socher et al., 2013)	–	45.7	85.4	–	–	–	–
DCNN (Kalchbrenner et al., 2014)	–	48.5	86.8	–	93.0	–	–
Paragraph-Vec (Le and Mikolov, 2014)	–	**48.7**	87.8	–	–	–	–
CCAE (Hermann and Blunsom, 2013)	77.8	–	–	–	–	–	87.2
Sent-Parser (Dong et al., 2014)	79.5	–	–	–	–	–	86.3
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	–	–	**93.6**	–	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	–	–	93.4	–	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	–	–	**93.6**	–	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	–	–	–	–	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	–	–	–	–	–	82.7	–
SVM_S (Silva et al., 2011)	–	–	–	–	**95.0**	–	–

Table 2: Results of our CNN models against other methods. **RAE**: Recursive Autoencoders with pre-trained word vectors from Wikipedia (Socher et al., 2011). **MV-RNN**: Matrix-Vector Recursive Neural Network with parse trees (Socher et al., 2012). **RNTN**: Recursive Neural Tensor Network with tensor-based feature function and parse trees (Socher et al., 2013). **DCNN**: Dynamic Convolutional Neural Network with k-max pooling (Kalchbrenner et al., 2014). **Paragraph-Vec**: Logistic regression on top of paragraph vectors (Le and Mikolov, 2014). **CCAE**: Combinatorial Category Autoencoders with combinatorial category grammar operators (Hermann and Blunsom, 2013). **Sent-Parser**: Sentiment analysis-specific parser (Dong et al., 2014). **NBSVM, MNB**: Naive Bayes SVM and Multinomial Naive Bayes with uni-bigrams from Wang and Manning (2012). **G-Dropout, F-Dropout**: Gaussian Dropout and Fast Dropout from Wang and Manning (2013). **Tree-CRF**: Dependency tree with Conditional Random Fields (Nakagawa et al., 2010). **CRF-PR**: Conditional Random Fields with Posterior Regularization (Yang and Cardie, 2014). **SVM_S**: SVM with uni-bi-trigrams, wh word, head word, POS, parser, hypernyms, and 60 hand-coded rules as features from Silva et al. (2011).

CNNs for Sentence Classification — Evaluation

- Sentiment Analysis: Stanford Sentiment Treebank (Socher et al., 2013)
  - Dataset of movie reviews annotated for polarity
  - Specialty: Provides annotations for all subtrees of the parse trees!
    → Very large number of samples
  - Introduced along with a new type of network: **Recursive Neural Tensor Networks (RNTN)**
    - Network with a structure to model compositionality over parse trees



Socher, R., Perelygin, A., Wu, J. Y., Chuang, J., Manning, C. D., Ng, A. Y. & Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. *Proceedings of the conference on empirical methods in natural language processing (EMNLP)* (p./pp. 1642), .

CNNs for Sentence Classification — Evaluation

- Sentiment Analysis: Stanford Sentiment Treebank (Socher et al., 2013)
  - Variant SST-1:
    - Given a sentence from a movie review, classify it into one of 5 classes (very positive, positive, neutral, negative, very negative)

    - Training set: 11855 sentences or phrases
      (phrases are also used in training to have more training data)
    - Test set: 2210 sentences
      (phrases are **not** used for testing to get a more accurate estimate of the performance)

CNNs for Sentence Classification — Evaluation

- Sentiment Analysis: Stanford Sentiment Treebank (Socher et al., 2013)
  - Variant SST-2:
    - Given a sentence from a movie review, classify it into one of 2 classes (positive, negative)
    - Dropped neutral samples, merged positive and negative classes
    - Training set: 9613 sentences or phrases
    - Test set: 1821 sentences

CNNs for Sentence Classification — Evaluation

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	–	–	–	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	–	–	–	–
RNTN (Socher et al., 2013)	–	45.7	85.4	–	–	–	–
DCNN (Kalchbrenner et al., 2014)	–	48.5	86.8	–	93.0	–	–
Paragraph-Vec (Le and Mikolov, 2014)	–	**48.7**	87.8	–	–	–	–
CCAE (Hermann and Blunsom, 2013)	77.8	–	–	–	–	–	87.2
Sent-Parser (Dong et al., 2014)	79.5	–	–	–	–	–	86.3
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	–	–	**93.6**	–	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	–	–	93.4	–	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	–	–	**93.6**	–	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	–	–	–	–	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	–	–	–	–	–	82.7	–
SVM_S (Silva et al., 2011)	–	–	–	–	**95.0**	–	–

Table 2: Results of our CNN models against other methods. **RAE**: Recursive Autoencoders with pre-trained word vectors from Wikipedia (Socher et al., 2011). **MV-RNN**: Matrix-Vector Recursive Neural Network with parse trees (Socher et al., 2012). **RNTN**: Recursive Neural Tensor Network with tensor-based feature function and parse trees (Socher et al., 2013). **DCNN**: Dynamic Convolutional Neural Network with k-max pooling (Kalchbrenner et al., 2014). **Paragraph-Vec**: Logistic regression on top of paragraph vectors (Le and Mikolov, 2014). **CCAE**: Combinatorial Category Autoencoders with combinatorial category grammar operators (Hermann and Blunsom, 2013). **Sent-Parser**: Sentiment analysis-specific parser (Dong et al., 2014). **NBSVM, MNB**: Naive Bayes SVM and Multinomial Naive Bayes with uni-bigrams from Wang and Manning (2012). **G-Dropout, F-Dropout**: Gaussian Dropout and Fast Dropout from Wang and Manning (2013). **Tree-CRF**: Dependency tree with Conditional Random Fields (Nakagawa et al., 2010). **CRF-PR**: Conditional Random Fields with Posterior Regularization (Yang and Cardie, 2014). **SVM_S**: SVM with uni-bi-trigrams, wh word, head word, POS, parser, hypernyms, and 60 hand-coded rules as features from Silva et al. (2011).

CNNs for Sentence Classification — Evaluation

- General findings:
  - Model performs very well!
  - Using pre-trained word embeddings is better than randomly initialising
  - No clear tendency for using cnn-static, cnn-nonstatic or cnn-multichannel

CNNs for Sentence Classification — Parameter Study

- Kim performed some hyper-parameter search
- Follow-up paper by Zhang and Wallace (2015):
  - Very extensive parameter studies!
  - Further improve results on some datasets
  - Main findings:
    - Pay attention to statistical variation! Training the same model on the same data can lead to results differing up to 1.5 percentage points!
    - Different word embeddings (Word2Vec/GloVe) perform differently for different tasks
    - Filter sizes have large influence on the results
    - Max-over-time-pooling outperforms other strategies, no need to tune
    - Regularisation has little effect on this model

Zhang, Y. & Wallace, B. (2015). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification (cite arxiv:1510.03820)

CNNs for Sentence Classification — Parameter Advise

- Empirical advise from Zhang and Wallace (2015):
  - Use a single filter size at first. Tune by line search, then explore the neighbourhood (e.g., if size 7 works well, add filters of size 6 and 8)
  - A number of filters per size in the range of 100-600 usually works well
  - Try Dropout rates in the range 0 - 0.5. In case of strong overfitting, use higher rates
  - Try different activation functions. Tanh and ReLU usually work best
  - Use max-over-time-pooling
  - Again: Pay attention to statistical variance!

Zhang, Y. & Wallace, B. (2015). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification (cite arxiv:1510.03820)

Our research with TextCNN: Emote-Controlled

- **Unsupervised Sentiment Analysis** on Twitch.tv comments
- Use **emotes** as sentiment indicators
- Lexicon-based approach creates *weak labels* for **TextCNN**

Other goodies:

- We computed emote embeddings and were able to calculate intensifications
- We were able to show that the Diablo Immortal announcement was not well-received by the audience

Konstantin Kobs, Albin Zehe, Armin Bernstetter, Julian Chibane, Jan Pfister, Julian Tritscher, and Andreas Hotho. 2020. Emote-Controlled: Obtaining Implicit Viewer Feedback Through Emote-Based Sentiment Analysis on Comments of Popular Twitch.tv Channels. Trans. Soc. Comput. 3, 2, Article 7 (April 2020), 34 pages. DOI:https://doi.org/10.1145/3365523

Our research with TextCNN: Emote-Controlled

Intensification of emotes

LUL **relates to** OMEGALUL **as X to Y**		
X	**Y**	**Explanation**
FeelsGoodMan	FeelsAmazingMan	Approval/satisfaction intensifies to amazement
FeelsBadMan	PepeHands	Sadness is intensified by crying
EZ	POGGERS	Extraordinary moves and moments in the (game) stream
cmonBruh	HYPERBRUH	An emote that is mostly used if the streamer's commentary can be interpreted as racist intensifies to an emote that is used in situations of clear racism.
WutFace	(puke)	Puking often follows disgust
4Head	4House	Intensifications in the emote text representations
4House	4Mansion	

Konstantin Kobs, Albin Zehe, Armin Bernstetter, Julian Chibane, Jan Pfister, Julian Tritscher, and Andreas Hotho. 2020. Emote-Controlled: Obtaining Implicit Viewer Feedback Through Emote-Based Sentiment Analysis on Comments of Popular Twitch.tv Channels. Trans. Soc. Comput. 3, 2, Article 7 (April 2020), 34 pages. DOI:https://doi.org/10.1145/3365523

Our research with TextCNN: Where to Submit?

- Use *title*, *abstract*, and *keywords* of a **scientific publication** to guess the **conference or journal**
- Make the output **interpretable** by highlighting which words were important for the classification *(gradients w.r.t. to input)*

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

We introduce a new language representation model called BERT, which stands for Bidirectional Encoder Representations from Transformers. . . .

→ NAACL

Web demo: https://wheretosubmit.ml

Kobs, K., Koopmann, T., Zehe, A., Fernes, D., Krop, P., & Hotho, A. (2020, November). Where to Submit? Helping Researchers to Choose the Right Venue. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings* (pp. 878-883).