Neural Networks

Überblick Künstliche Intelligenz

Marco Steinke

29.06.2021

The power of Neural Networks

Table of Contents

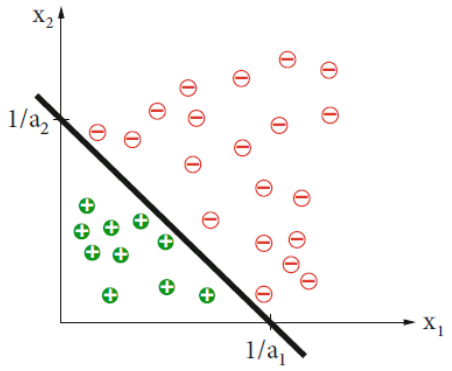
1. Construction and Structure

Heinrich Heine
Universität Düsseldorf

- 1.1 Perceptron
- 1.2 Deep Feedforward Networks
- 1.3 Example: Learning XOR
- 1.4 Computational Graph Language

1.1 Perceptron

Abb. 8.7 Eine linear separable zweidimensionale Datenmenge. Die Trenngerade hat die Gleichung $a_1 x_1 + a_2 x_2 = 1$



1.1 Perceptron

Heinrich Heine
Universität Düsseldorf

Beispiel:

- M+ = {(0, 1.8), (2, 0.6)}
  M- = {(-1.2, 1.4), (0.4, -1)}

  w = (1, 1)

1.1 Perceptron

- $\Delta w = x$, $\Delta w = -x$

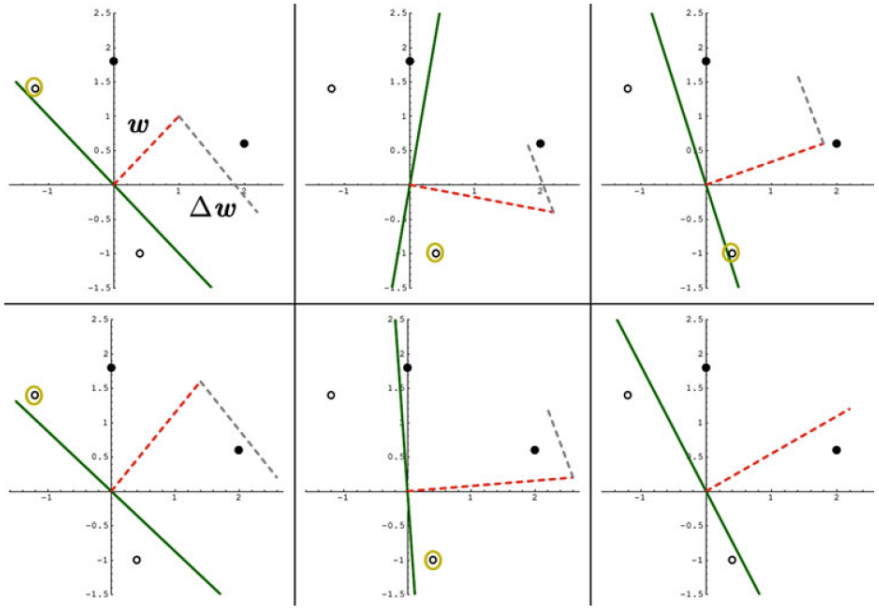


Abb. 8.10 Anwendung der Perzeptronlernregel auf zwei positive (●) und zwei negative (○) Daten-

1.1 Perceptron

Heinrich Heine
Universität Düsseldorf

Definition 8.3

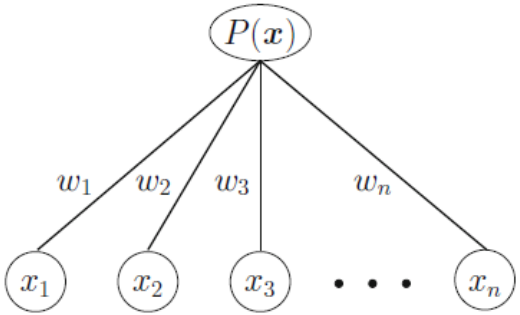
Sei $\boldsymbol{w} = (w_1, \ldots, w_n) \in \mathbb{R}^n$ ein Gewichtsvektor und $\boldsymbol{x} \in \mathbb{R}^n$ ein Eingabevektor. Ein **Perzeptron** stellt eine Funktion $P : \mathbb{R}^n \to \{0, 1\}$ dar, die folgender Regel entspricht:

$$P(\boldsymbol{x}) = \begin{cases} 1 & \text{falls } \boldsymbol{w}\,\boldsymbol{x} = \sum_{i=1}^{n} w_i x_i > 0 \\ 0 & \text{sonst} \end{cases}$$

1.1 Perceptron

Heinrich Heine
Universität Düsseldorf

Abb. 8.9 Graphische Darstellung eines Perzeptrons als zweilagiges neuronales Netz



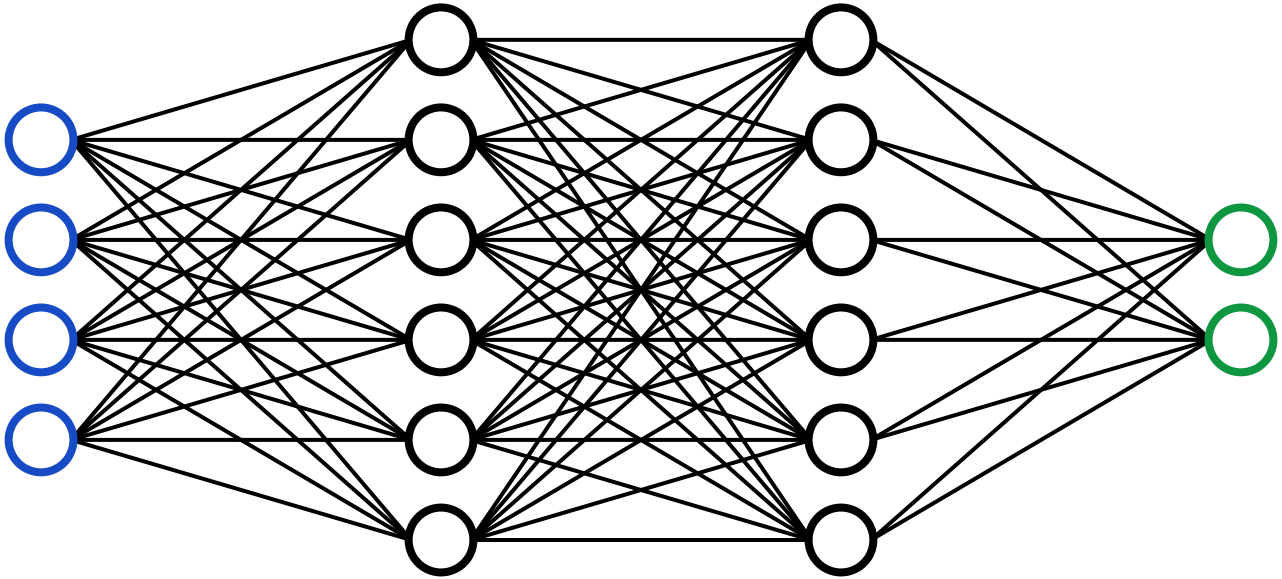
1.1 Perceptron

Heinrich Heine
Universität Düsseldorf

PerzeptronLernen(M_+, M_-)

$\boldsymbol{w}$ = beliebiger Vektor reeller Zahlen
Repeat
    For all $\boldsymbol{x} \in M_+$
        If $\boldsymbol{w}\,\boldsymbol{x} \leq 0$ **Then** $\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{x}$
    For all $\boldsymbol{x} \in M_-$
        If $\boldsymbol{w}\,\boldsymbol{x} > 0$ **Then** $\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{x}$
Until alle $\boldsymbol{x} \in M_+ \cup M_-$ werden korrekt klassifiziert

1.2 Deep Feedforward Networks

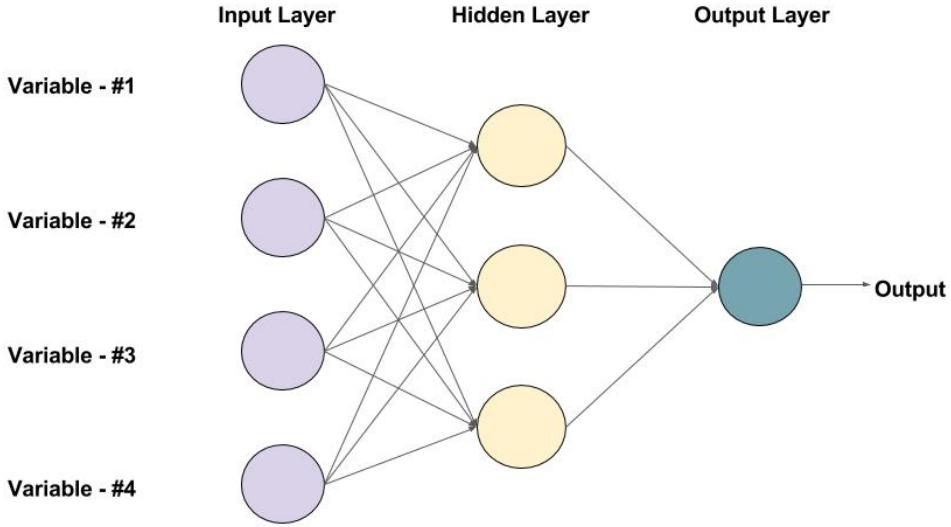


1.2 Deep Feedforward Networks

Heinrich Heine
Universität Düsseldorf

- Feedforward Neural Networks (FNN) / Multilayer Perceptrons (MLP)
- Approximation of function **f***.
- **y = f(x;Θ)**
- parameter‘s **Θ**

1.2 Deep Feedforward Networks



An example of a Feed-forward Neural Network with one hidden layer (with 3 neurons)

1.2 Deep Feedforward Networks

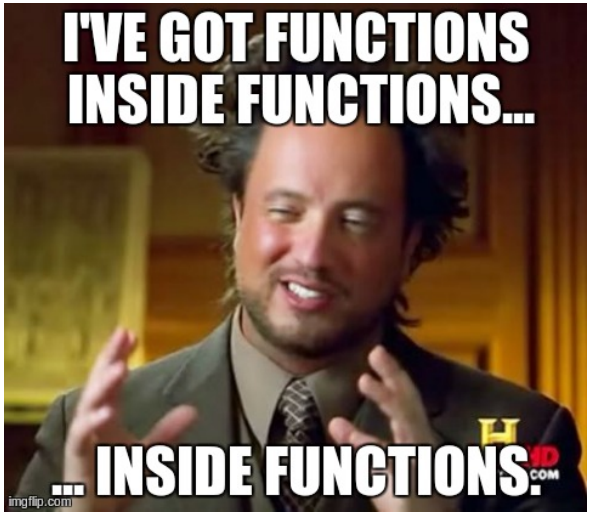
Heinrich Heine
Universität Düsseldorf

- Given $f^{(1)}, f^{(2)}, f^{(3)}$, define f as

$$f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$$

- Amount of **Layers** = depth

  → „Deep Learning“



1.2 Deep Feedforward Networks

Heinrich Heine
Universität Düsseldorf

- **f** gets closer to **f*** during training.
- **Training data** labeled with values of **f*** $y \approx f*(x)$

1.3 Example: Learning XOR

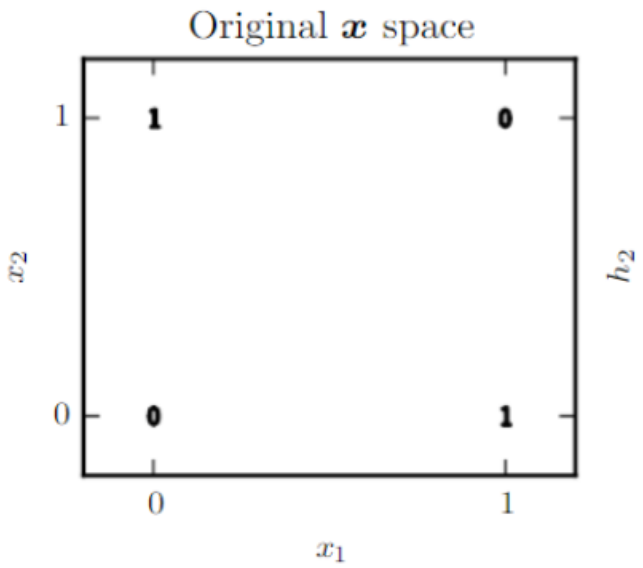
Heinrich Heine
Universität Düsseldorf

- Construct a FNN which learns XOR.
- Truth table = y = f*(x) .

“**exclusive OR**“ (XOR):

x	y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

1.3 Example: Learning XOR



"**exclusive OR**" (XOR):

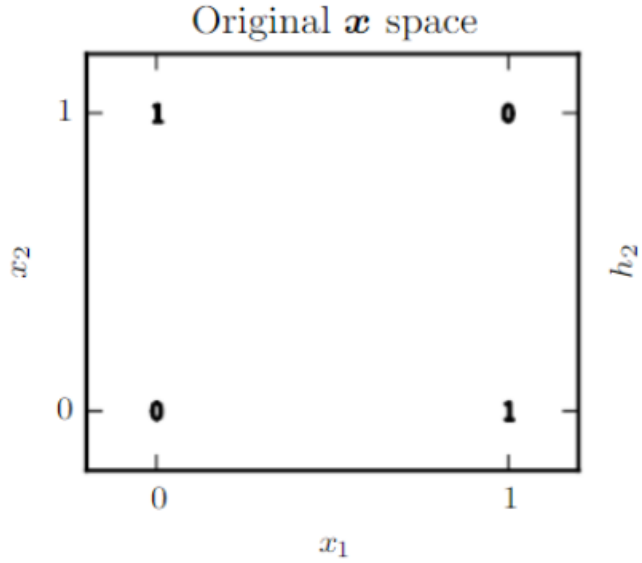
x	y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

1.3 Example: Learning XOR

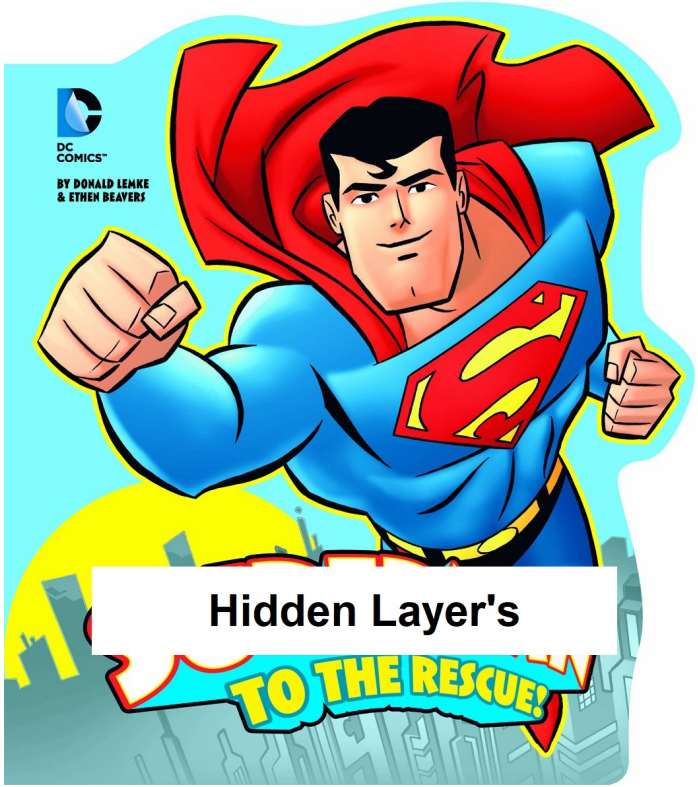
- **linear Regression**
- **Mean squared error loss function**

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

$$J(\Theta) = \frac{1}{4}\sum_{x \in X}\left(f*(x) - f(x;\Theta)\right)^2$$



1.3 Example: Learning XOR



https://www.amazon.de/Superman-Rescue-Comics-Donald-Lemke/dp/1479516880

1.3 Example: Learning XOR

Heinrich Heine
Universität Düsseldorf

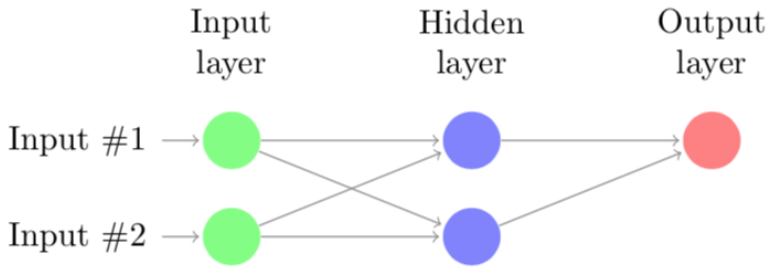
- **hidden layer:** $h = f^{(1)}(x; W, c)$
- Complete model: $y = f^{(2)}(h; w, b)$

$$f(X; W, c, w, b) = f^{(2)}(f^{(1)}(x))$$

- Output Layer still runs linear regression.
- Regression applied to **h**.

1.3 Example: Learning XOR

- What about $f^{(1)}(x)$?



1.3 Example: Learning XOR

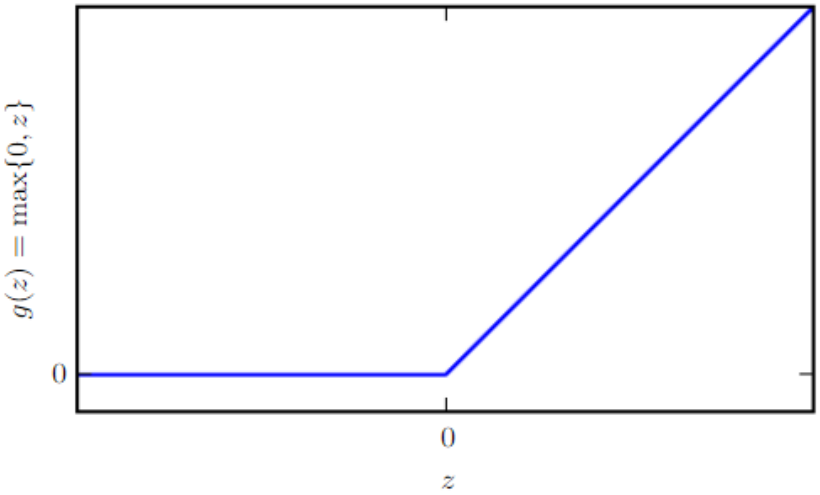
Heinrich Heine
Universität Düsseldorf

activation function:

rectified linear unit (ReLU)

1.3 Example: Learning XOR

- ReLU: **g(z) = max { 0, z }**
- Stückweise linear



1.3 Example: Learning XOR

Heinrich Heine
Universität Düsseldorf

Wende die **activation function** elementweise an:

$$h_i = g(W_i^T x + c_i) = max\{0, W_i^T x + c_i\}$$

1.3 Example: Learning XOR

Heinrich Heine
Universität Düsseldorf

$$f(x; W, c, w, b) = w^T max\{0, W^T x + c\} + b$$

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad c = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \quad w = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$$

1.3 Example: Learning XOR

Heinrich Heine
Universität Düsseldorf

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

1.3 Example: Learning XOR

Heinrich Heine
Universität Düsseldorf

$$XW = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$$

1.3 Example: Learning XOR

Heinrich Heine
Universität Düsseldorf

$$XW + c = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

1.3 Example: Learning XOR

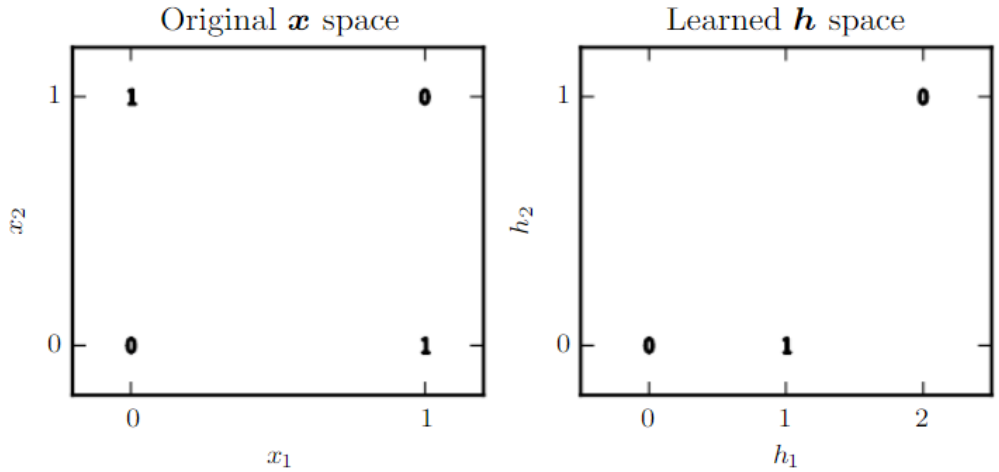
Heinrich Heine
Universität Düsseldorf

$$max\{0, W^T x + c\} = \left(\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix} \right)$$

1.3 Example: Learning XOR

- Mit **relu:**

$$max\{0, W^T x + c\} = \left(\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix} \right)$$

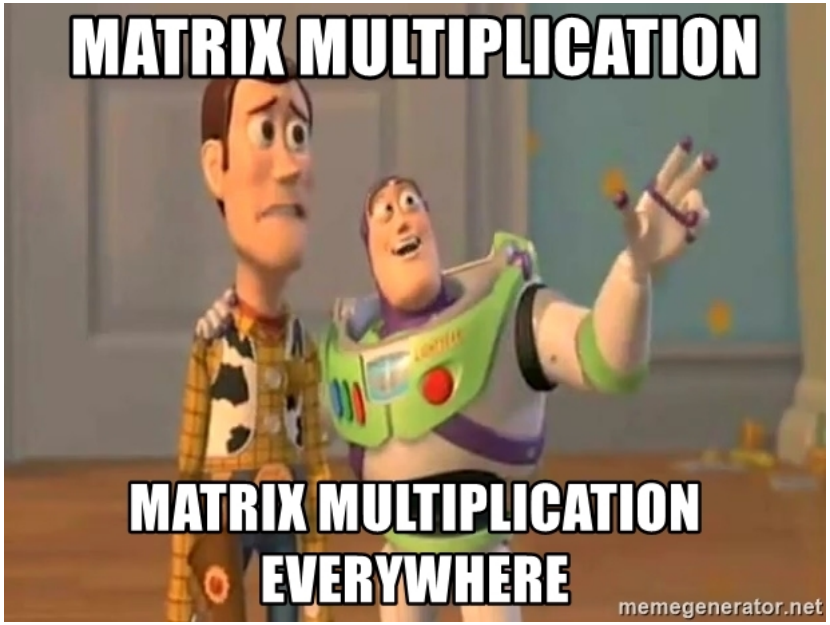


1.3 Example: Learning XOR

Heinrich Heine
Universität Düsseldorf

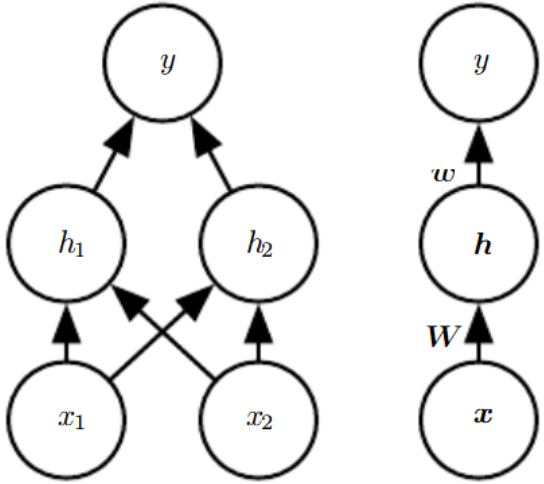
$$w^T max\{0, W^T x + c\} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix} * \begin{bmatrix} 1 & -2 \end{bmatrix}^T = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

1.3 Example: Learning XOR



https://memegenerator.net/instance/30188726/x-x-everywhere-matrix-multiplication-matrix-multiplication-everywhere

1.3 Example: Learning XOR



„Figure 6.2 Deep Learning, Ian Goodfellow“

1.4 Computational Graph Language

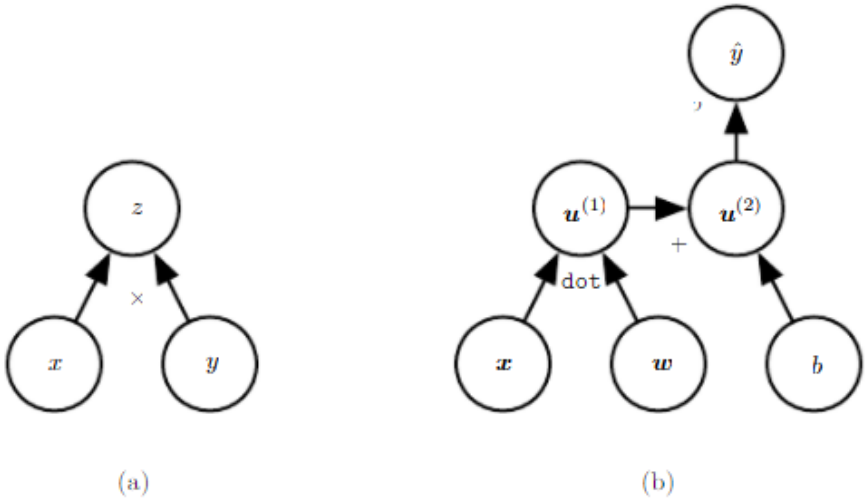
Heinrich Heine
Universität Düsseldorf

- At this point the structure of Neural networks needs some formalism.
- Thus the expressions **node** and **operation** need to be introduced.

1.4 Computational Graph Language

- Examples of operations:

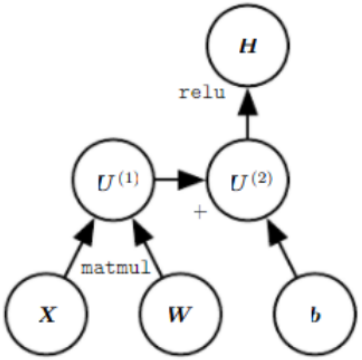
  (a) x: (x, y) => x*y (b) dot: Scalar product, +: sum



1.4 Computational Graph Language

- Examples of operations:

  (c) relu: z => max{ 0, z} (rectified linear unit)



1. Summary:

Heinrich Heine
Universität Düsseldorf

What we have learned:

Neural Networks run a composition of functions.

They are used to approximate functions.

Perceptron is not enough to solve XOR.

Hidden Layer‘s can perform transformations.

2. Training

- 2.1 Chain Rule
- 2.2 Backpropagation
- 2.3 Delta Rule

2.1 Chain Rule

Heinrich Heine
Universität Düsseldorf

- Necessary to understand Backpropagation.
- General chain rule: $$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}.$$
- Calculation of Gradients:

  → update weights
  → implement filters
  → ...

2.2 Backpropagation

Heinrich Heine
Universität Düsseldorf

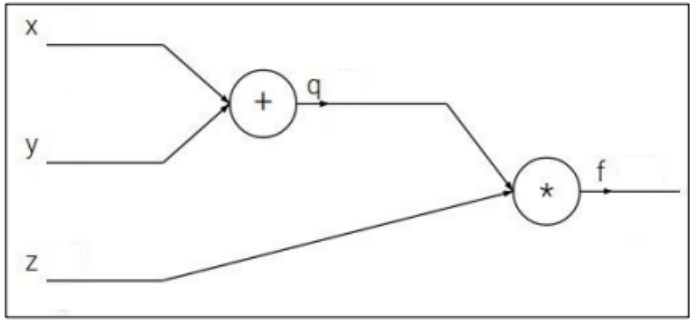
- Modify weights
- Calculation of Gradients

2.2 Backpropagation

Heinrich Heine
Universität Düsseldorf

Example:

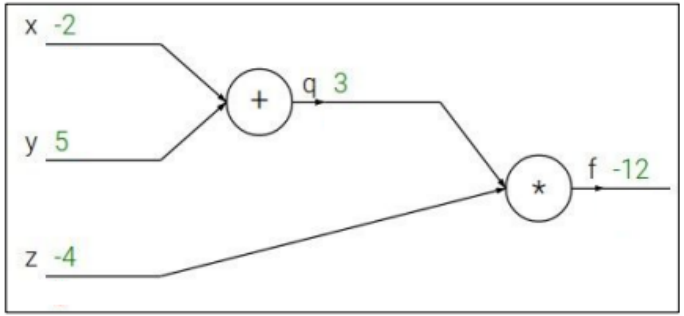
- f(x, y, z) = (x + y)z
- **Minibatch**



„Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 4 - 43 April 11, 2019“

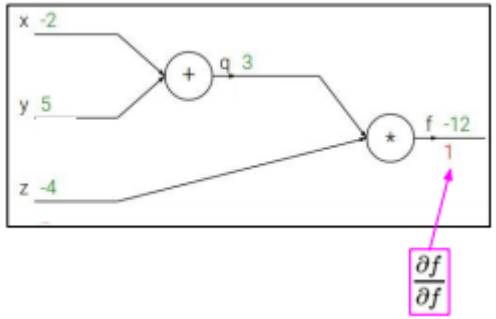
2.2 Backpropagation

Example:

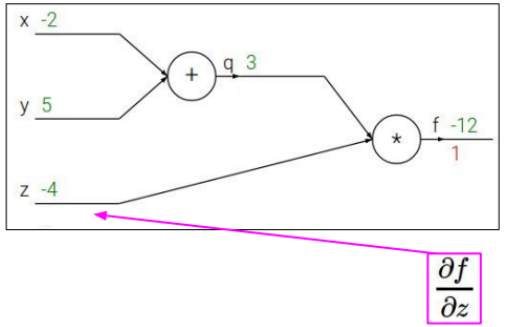


„Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 4 - 44 April 11, 2019“

2.2 Backpropagation

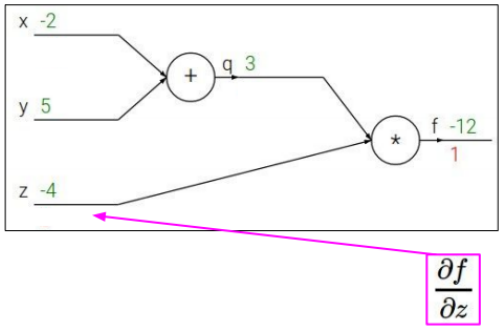


„Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 4 - 47 April 11, 2019“



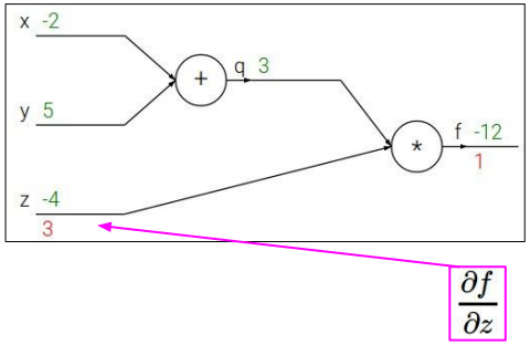
„Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 4 - 48 April 11, 2019“

2.2 Backpropagation



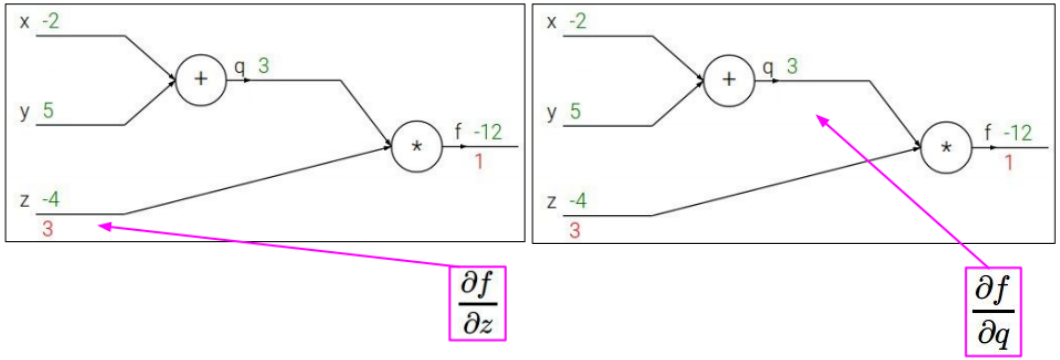
„Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 4 - 48 April 11, 2019“

2.2 Backpropagation



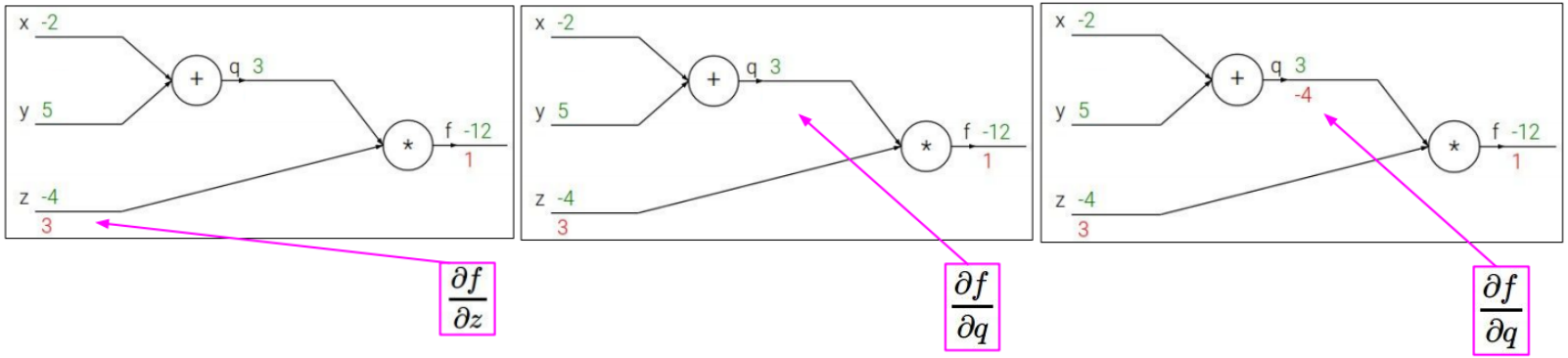
„Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 4 - 49ff April 11, 2019“

2.2 Backpropagation



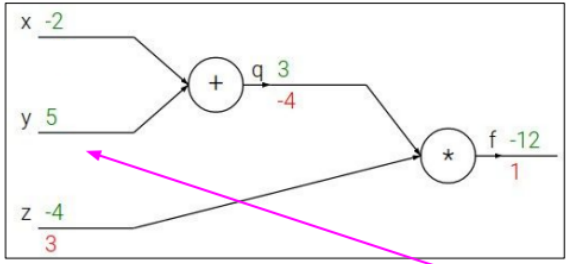
„Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 4 - 49ff April 11, 2019“

2.2 Backpropagation



„Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 4 - 49ff April 11, 2019“

2.2 Backpropagation

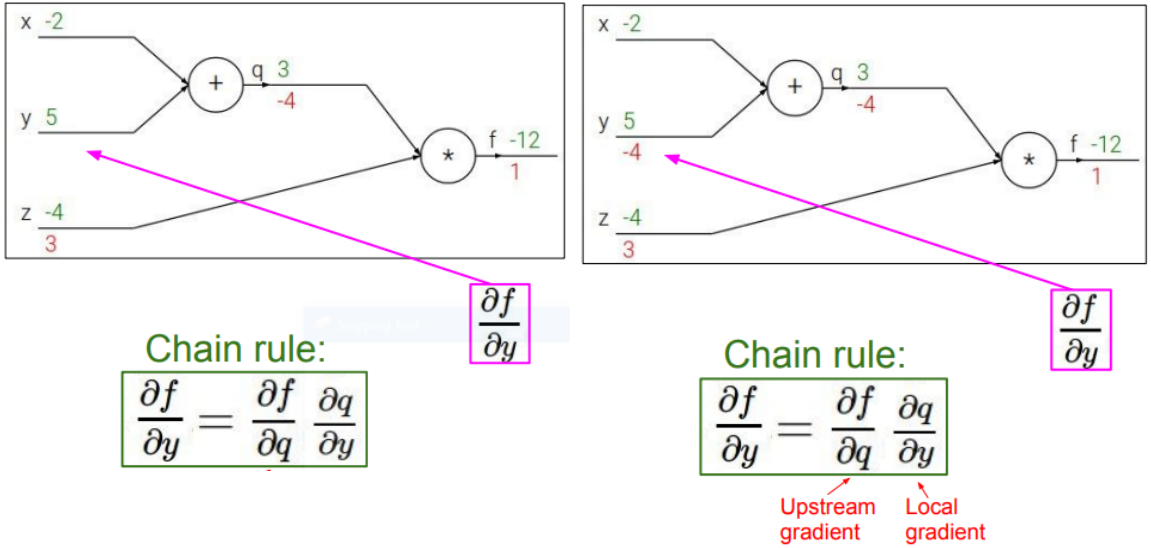


Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

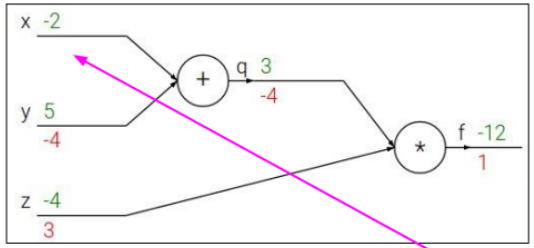
„Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 4 - 52ff April 11, 2019“

2.2 Backpropagation



„Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 4 - 52ff April 11, 2019“

2.2 Backpropagation



$$\frac{\partial f}{\partial x}$$

Chain rule:

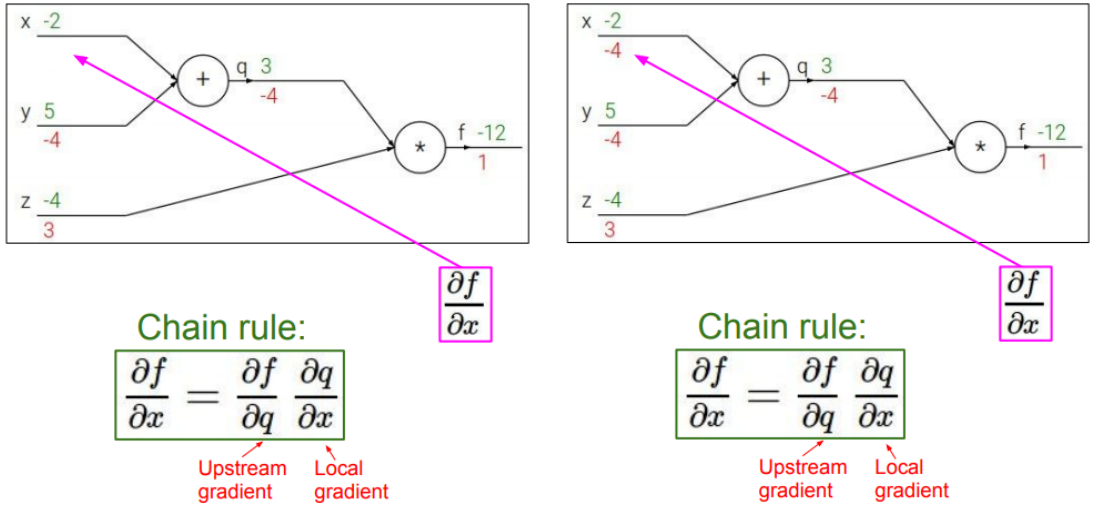
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

Upstream gradient

Local gradient

„Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 4 - 52ff April 11, 2019“

2.2 Backpropagation



„Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 4 - 52ff April 11, 2019“

2.3 Delta Rule

- Special case of Backpropagation
- For neuron j:

$$\Delta w_{ji} = \varepsilon (t_j - y_j) * x_i$$

$\varepsilon = Learning\ Rate$

$t_j = target\ output$

$y_j = actual\ output$

$x_i = ith\ input$

2. Summary:

Heinrich Heine
Universität Düsseldorf

What we have learned:

Backpropagation calculates gradients

Recursively update weights

Repeat until the gradients converges

3. Convolutional Neural Networks

Heinrich Heine
Universität Düsseldorf

- 3.1 Convolutional Neural Networks
- 3.2 Optimization

3.1 Convolutional Neural Networks

Heinrich Heine
Universität Düsseldorf

- Data with grid-based topology
- Images, measurements

0	1	2	3	4	5	6	7	8	9
0.1	0.3	0.7	0.6	0.2	0.1	0.5	0.6	0.4	0.2

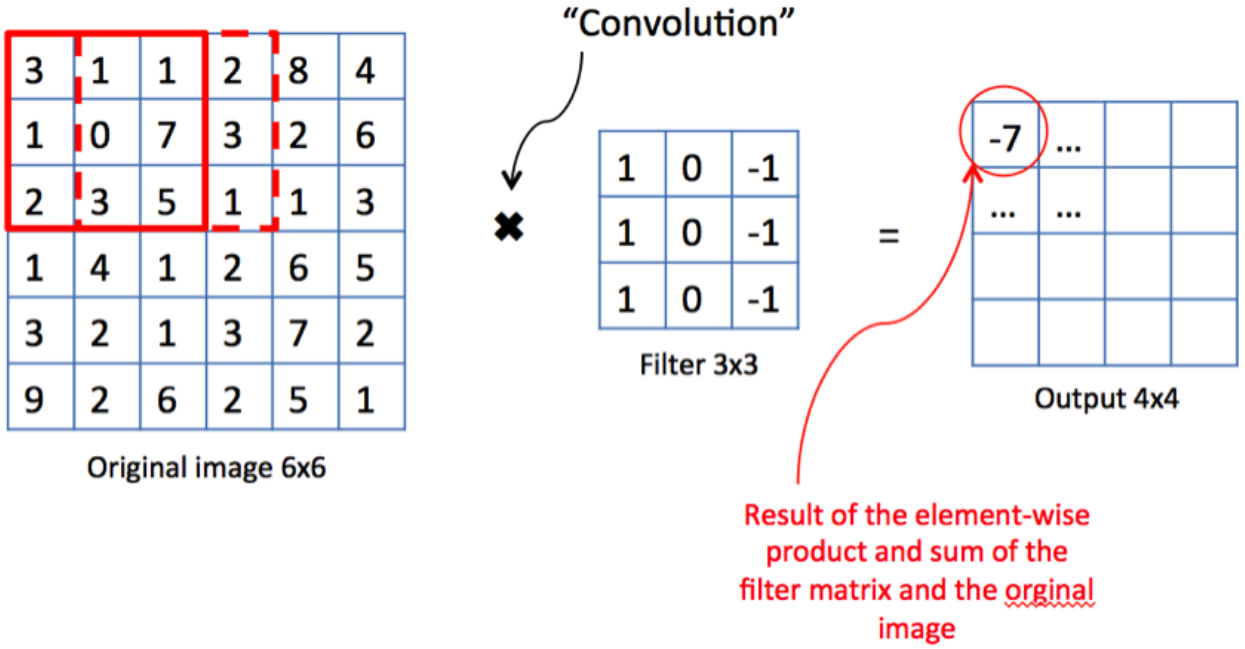
3.1 Convolutional Neural Networks

Heinrich Heine
Universität Düsseldorf

Recap:

- Input
- Kernel
- Feature Map

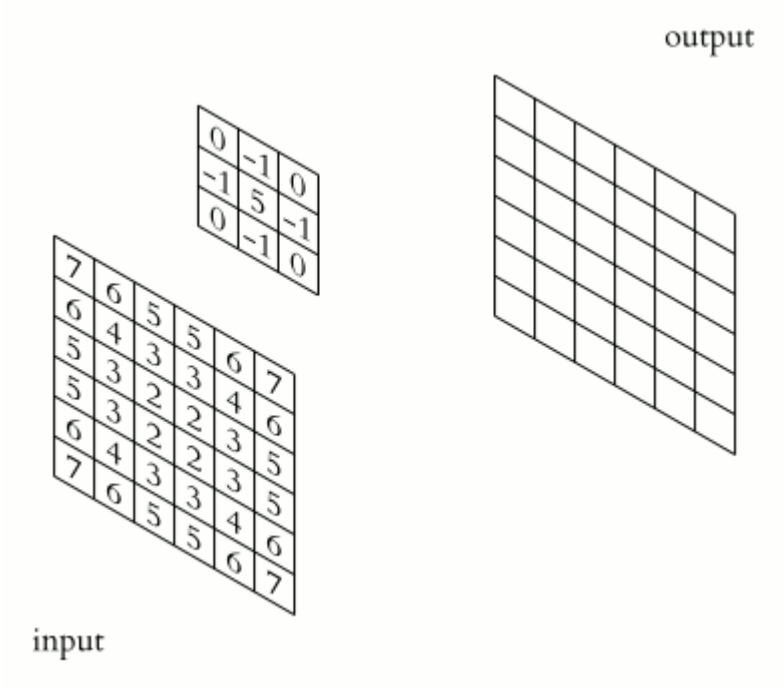
3.1 Convolutional Neural Networks



3.1 Convolutional Neural Networks

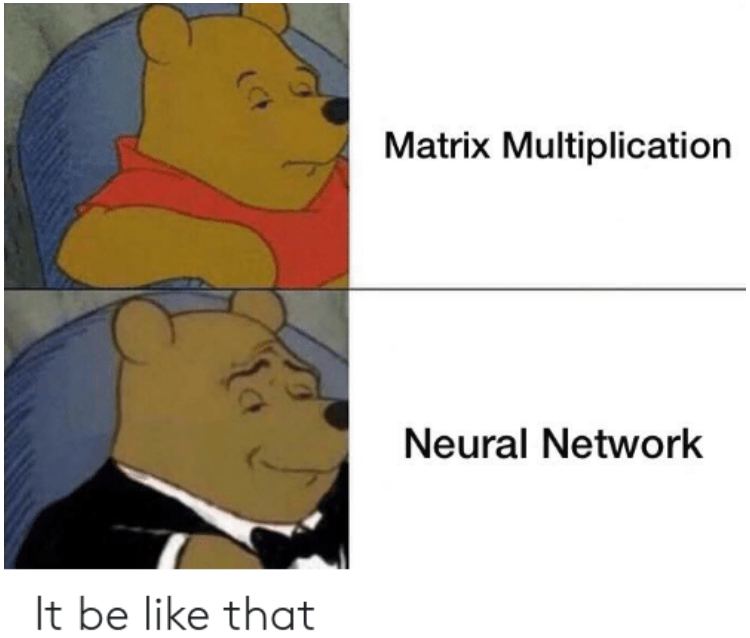
Heinrich Heine
Universität Düsseldorf

Padding



https://medium.com/machine-learning-bites/deeplearning-series-convolutional-neural-networks-a9c2f2ee1524

3.1 Convolutional Neural Networks



https://me.me/i/matrix-multiplication-neural-network-it-be-like-that-4a98811a05794edfbb78d5384c892268

3.1 Convolutional Neural Networks

Heinrich Heine
Universität Düsseldorf

Convolutional Neural Network:

A neural network which uses convolution in at least one of it layers.

3.1 Convolutional Neural Networks

Heinrich Heine
Universität Düsseldorf

- Convolution is never used alone!

- Reminder: Tesla‘s self drive AI

Reminder:


Lane Change: RIGHT
Ego Speed: 10.20 MPH
Time: 28654.559149000
CAL P 1.35 Y 0.10 R 0.00 deg
Vision fps: 16.93 Draw fps: 17.81 Display fps: 15.77
NL(0.00), E(0.99), F(0.00), TF(0.00), S(0.00)
NRW: FLP(0.00), FRP(0.00)
CutinExcited (Prb 0.59)
+0.0000 AUTO_HIGH_BEAM
+0.0000 BLINDED
+0.0000 RAINING
+0.0000 TIRE_SPRAY
+0.0001 WET_ROAD
0.4528 RESTRICTED
0.0002 CONTROLLED_ACCESS
AP
L:2 R:0 F:1 ON:1
W:9.3 AP:1.7 I:1
VS: 22.7 MPH St: 1
merge: 1.0 1 71.5 L
28
40
49
72
80
33
54
40
9m
MAIN

Reminder:

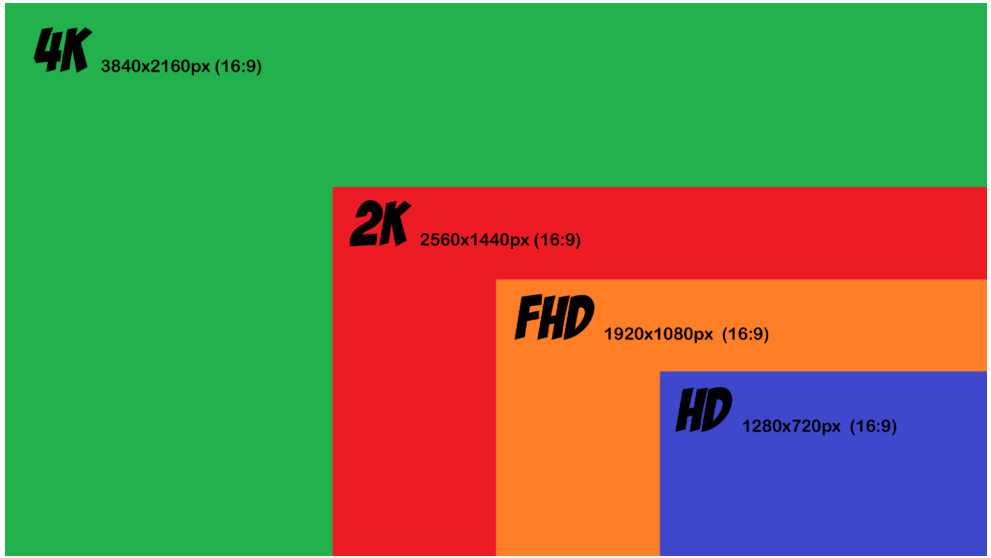
Heinrich Heine
Universität Düsseldorf

```
function matmult(A,B,l,m,n)
  C = zeroes(l,n)
  for i = 1 to l
    for k = 1 to n
      for j = 1 to m
        C(i,k) = C(i,k) + A(i,j) * B(j,k)
      end
    end
  end
  return C
```

Worst case: $O(n^3)$

3.2 Optimization

- Convolution can be very slow.



https://commons.wikimedia.org/wiki/File:Resolucions_4k.png

3.2 Optimization

https://img.devrant.com/devrant/rant/r_1688469_xaXLS.jpg

3.2 Optimization

Heinrich Heine
Universität Düsseldorf

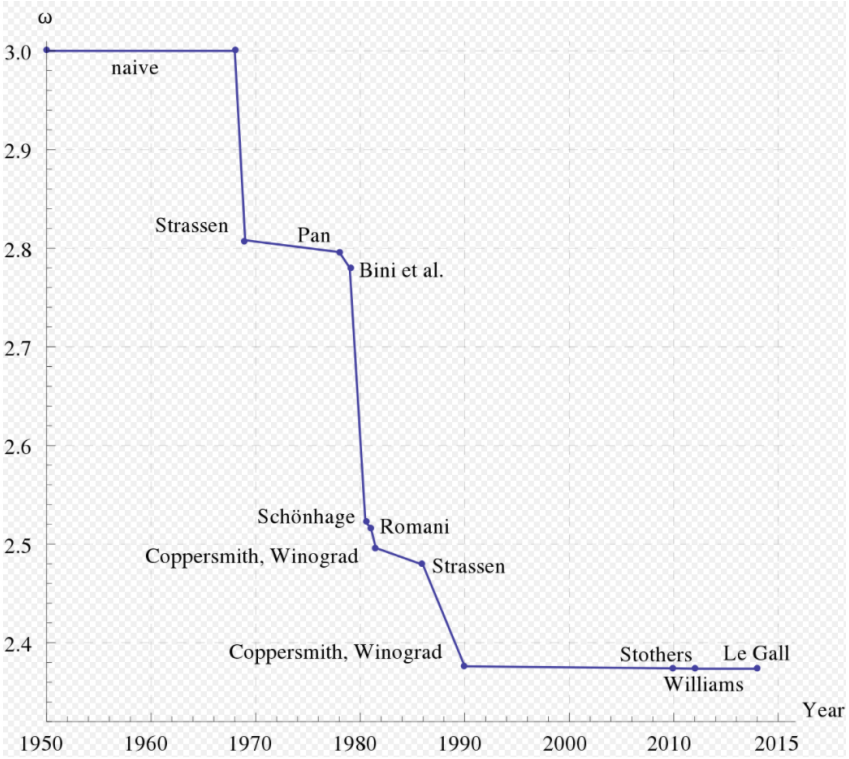
- **Algorithms**
- Pooling
- Dropout

3.2 Optimization

Heinrich Heine
Universität Düsseldorf

Algorithms:

$\omega = 2.5, O(n^{2.5})$



https://commons.wikimedia.org/wiki/File:Resolucions_4k.png

3.2 Optimization

Heinrich Heine
Universität Düsseldorf

- Algorithms
- **Pooling**
- Dropout

3.2 Optimization

- Activation function (e.g. ReLU)
- max-pooling

3.2 Optimization

Heinrich Heine
Universität Düsseldorf

- Algorithms
- Pooling
- **Dropout**

Any questions?