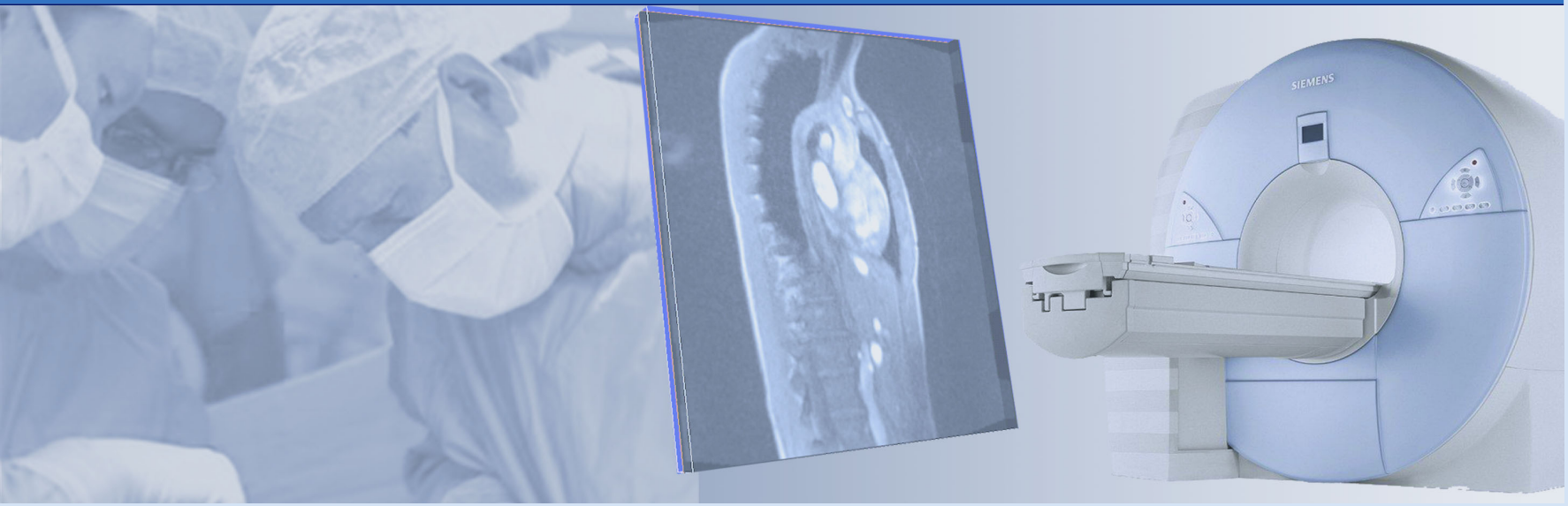
Computer- and robot-assisted Surgery



NCT

NATIONALES CENTRUM
FÜR TUMORERKRANKUNGEN
PARTNERSTANDORT DRESDEN
UNIVERSITÄTS KREBSCENTRUM UCC

getragen von:
Deutsches Krebsforschungszentrum
Universitätsklinikum Carl Gustav Carus Dresden
Medizinische Fakultät Carl Gustav Carus, TU Dresden
Helmholtz-Zentrum Dresden-Rossendorf

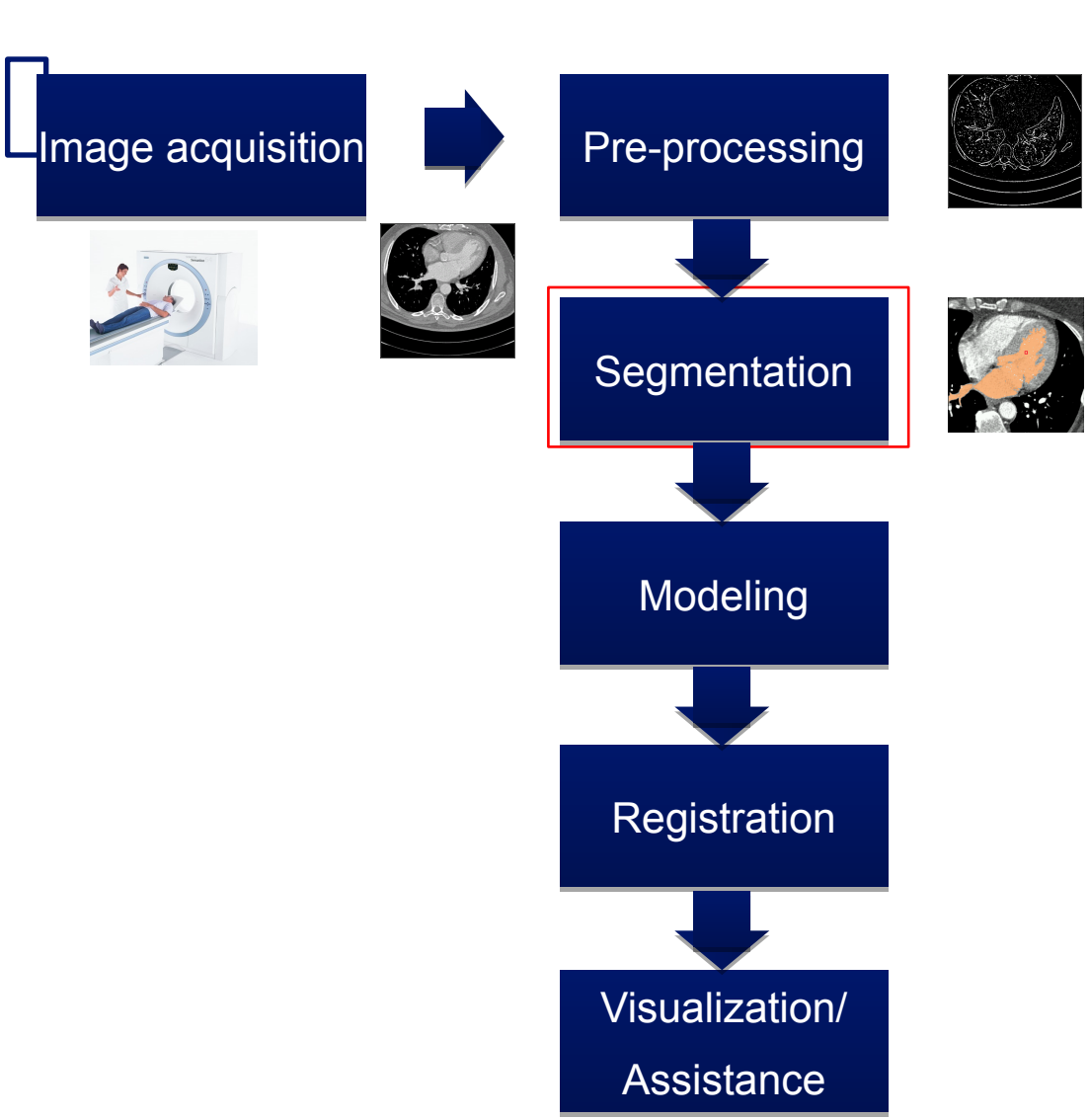
Sebastian Bodenstedt

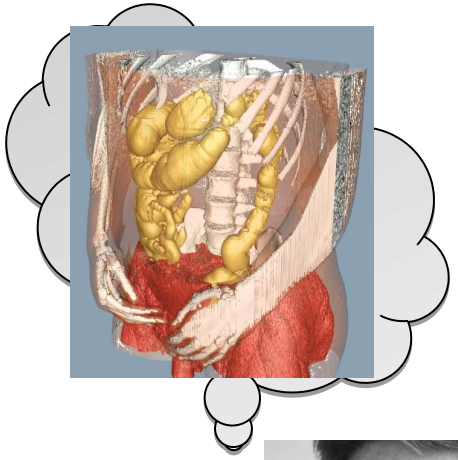
Lecture 10 – Segmentation III/ML Basics II

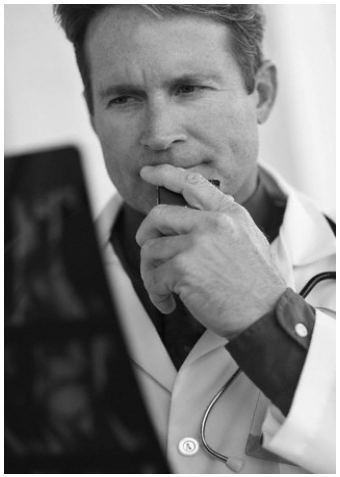
Oral Exam

If you want to participate in the oral exam, please contact
Ricarda Abdel-Bary (ricarda.abdel-bary@nct-dresden.de)

Process chain







Contents

Neural Networks: How can images be classified and segmented?

- What are artificial neural networks?
  - How can I train a neural network?
- Convolutional Neural Networks
  - Image classification (what can be seen in an image)
  - Image segmentation (where can something be seen in an image)

NCT

Image classification: Which objects are in an image?

Dog or cat?

Image classification: Which objects are in an image?

Classifier

Dog or cat?

Training/ Determine parameters

Image segmentation: Where in the image are objects?

Segmentation

Image segmentation: Where in the image are objects?

Segmentation

Training/
Determine
parameters

Basics Neural Networks

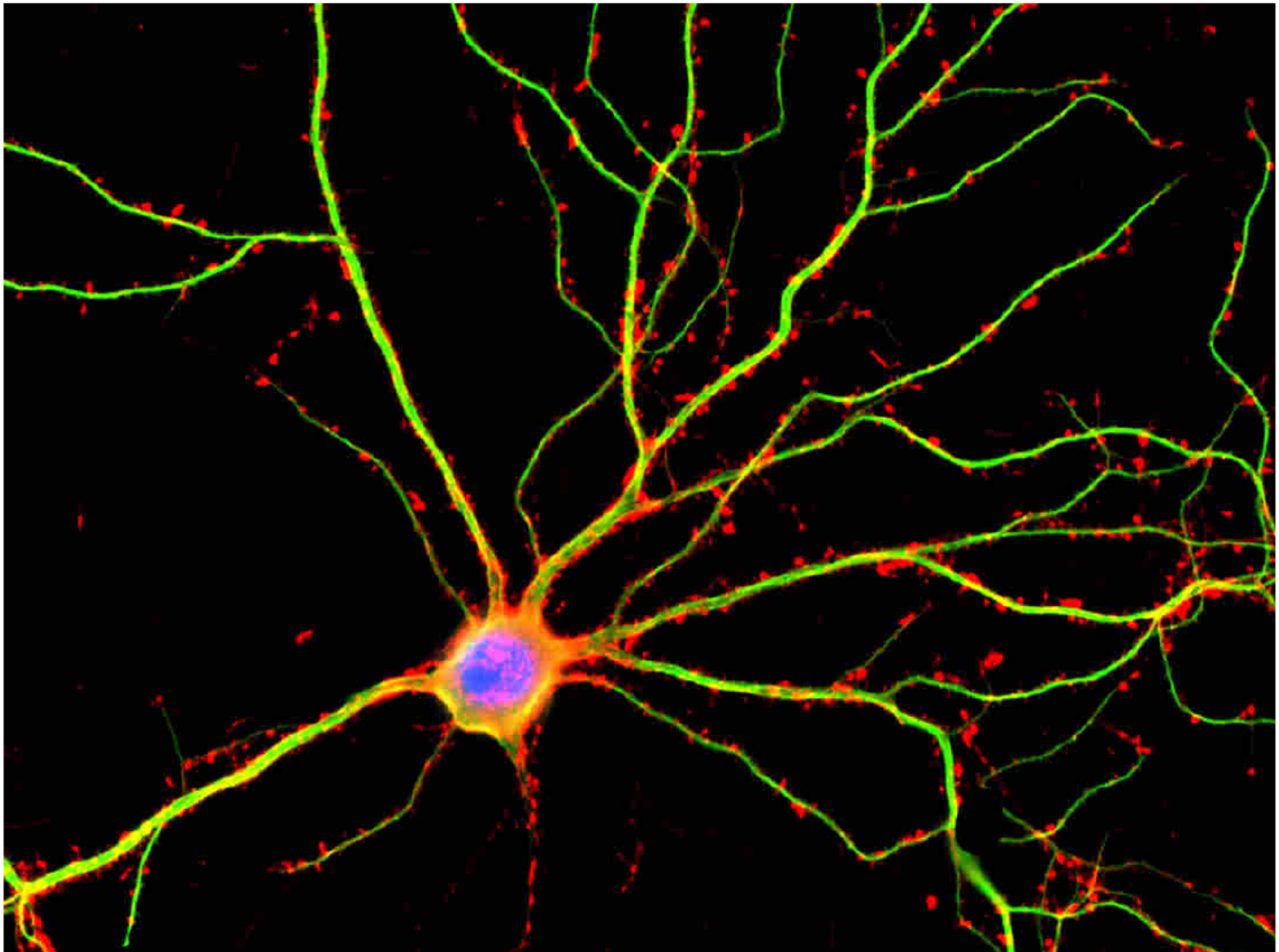
NCT

Basics Neuronal Networks

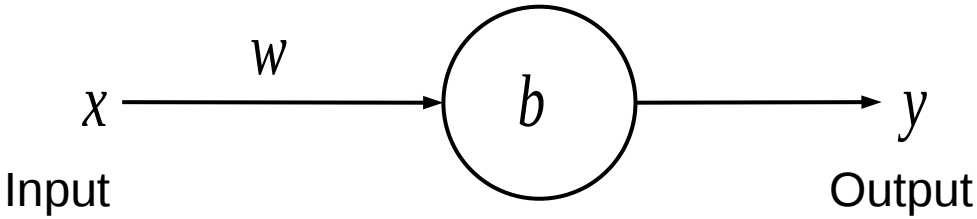
- What are artificial neural networks?
  - Biological motivated machine learning method
    - Consists out of artificial neurons (Perceptron)
- Classification
  - Supervised learning
    - Required training examples x_t and labels y_t
    - Learns a function f: y = f(x)
  - Symbolic and sub-symbolic ($y \in \mathbb{R}^N$)

NCT

Neuron

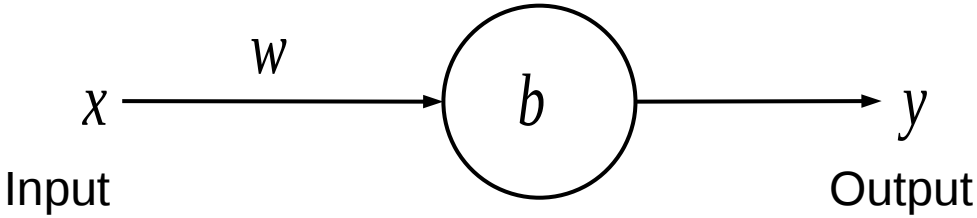


Perceptron (Neuron)

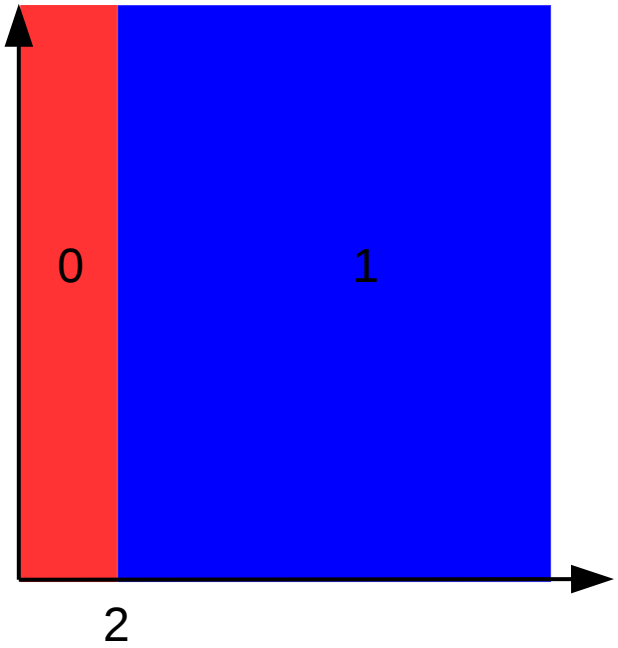


$$y = \begin{cases} 0 \, if \; w \cdot x \leqslant b \\ 1 \, if \; w \cdot x > b \end{cases}$$

Perceptron (Neuron) - Example

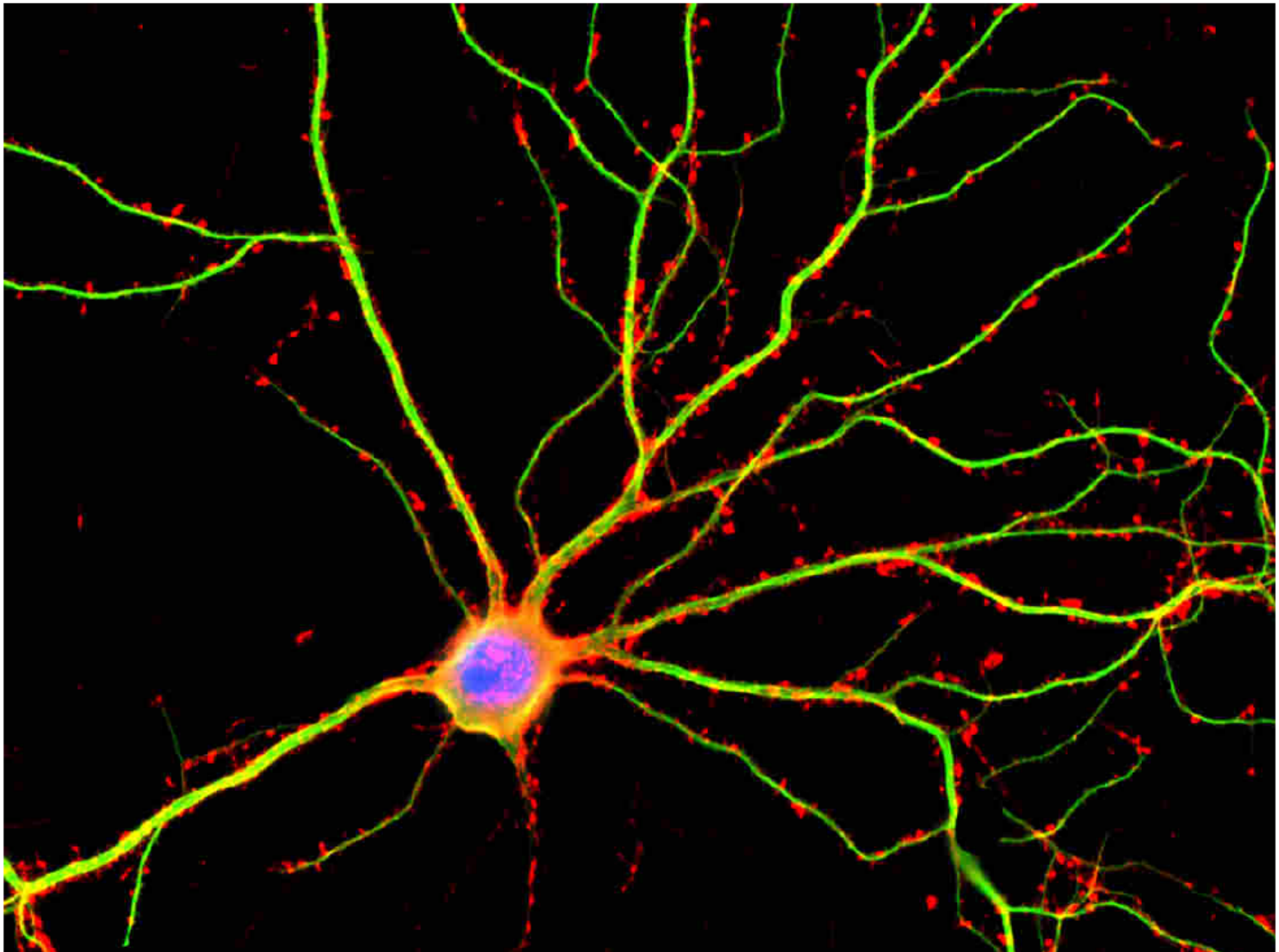


$$y = \begin{cases} 0 \; if \; w \cdot x \leqslant b \\ 1 \; if \; w \cdot x > b \end{cases}$$

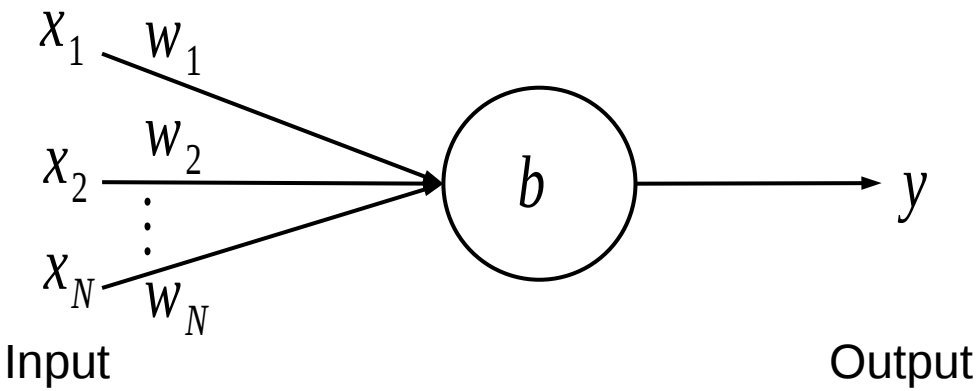


W = 1, b = 2

Neuron

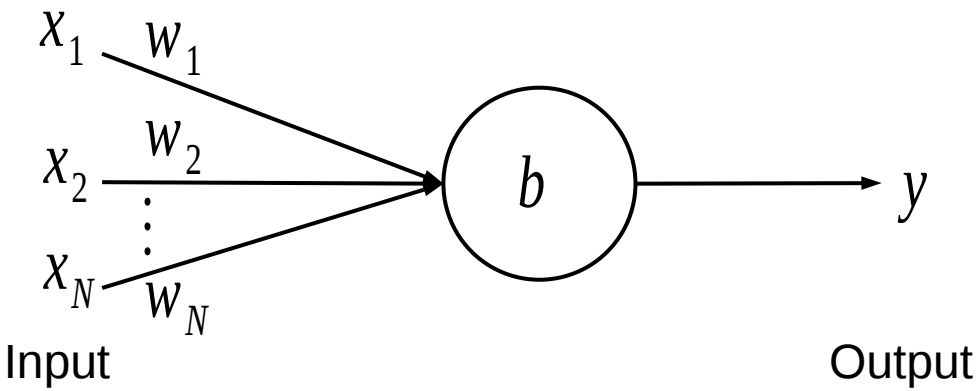


Perceptron (Neuron)



$$y = \begin{cases} 0 \; if \sum_i w_i \cdot x_i \leqslant b \\ 1 \; if \sum_i w_i \cdot x_i > b \end{cases}$$

Perceptron (Neuron)

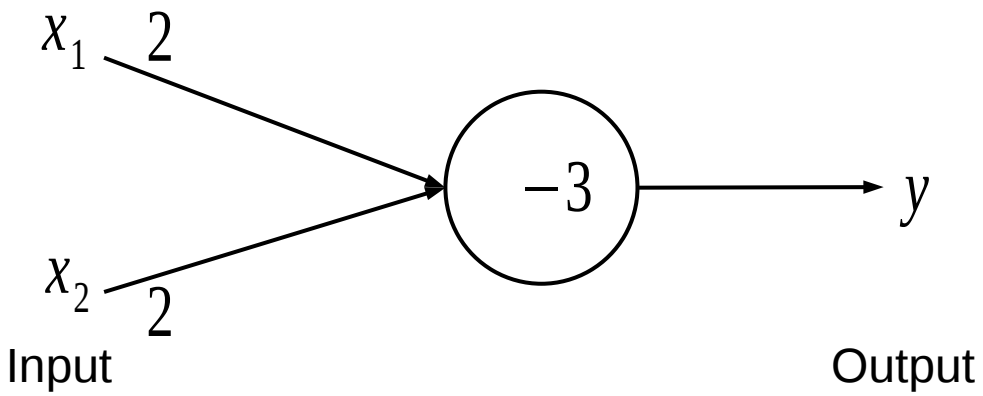


$$w \cdot x \equiv \sum_i w_i \cdot x_i$$

$$b \equiv -b$$

$$y = \begin{cases} 0 \; if \; w \cdot x + b \leqslant 0 \\ 1 \; if \; w \cdot x + b > 0 \end{cases}$$

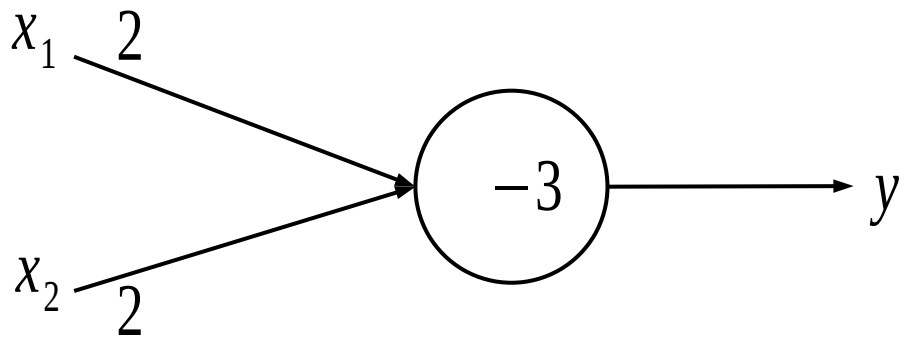
Perceptron (Neuron)



Assume: $x_1, x_2 \in \{0, 1\}$

- Which function is being computed?

Perceptron (Neuron)



	$X_1 = 0$	$X_1 = 1$
$X_2 = 0$	$w \cdot x + b = -3$ $y = 0$	$w \cdot x = -1$ $y = 0$
$X_2 = 1$	$w \cdot x + b = -1$ $y = 0$	$w \cdot x = 1$ $y = 1$

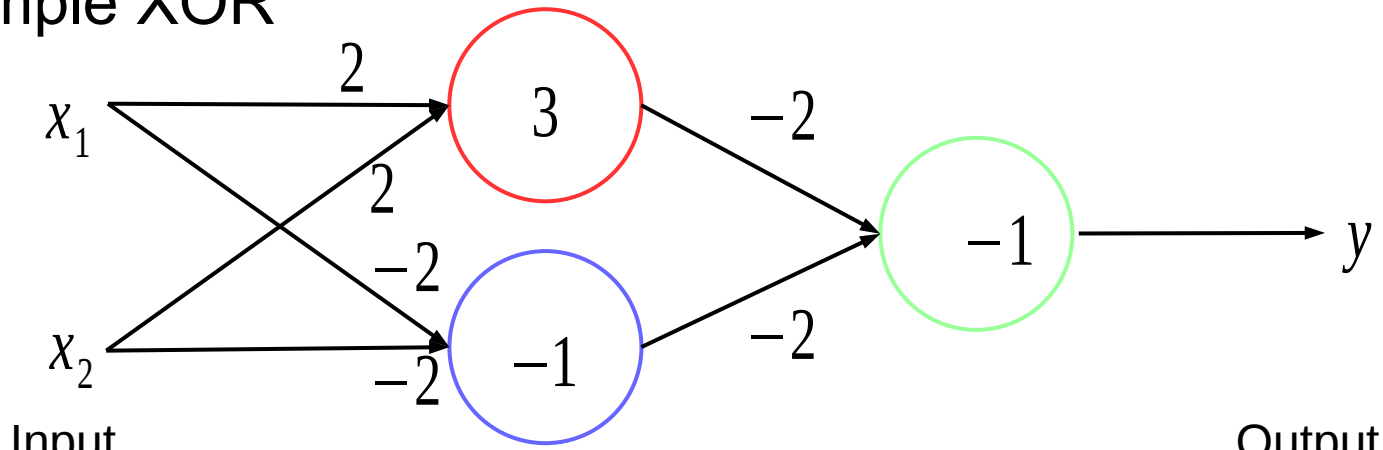
=> Logical AND

NCT

Multi-layered Network

- Perceptrons are linear discriminators
  - Usable for simple logical operations

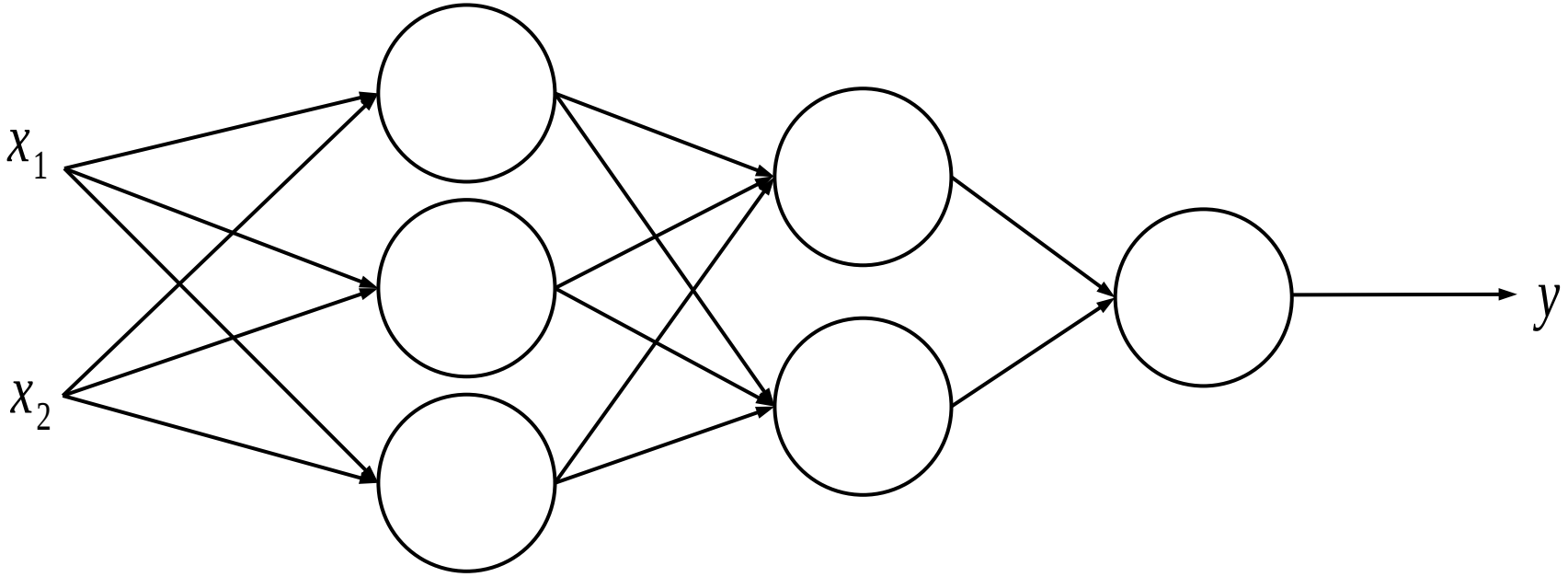
    (AND, OR, NAND, …)
  - Complex operations required more layers
- Example XOR



00	0	1	0
01	0	0	1
10	0	0	1
11	1	0	0

Multi-layered Network

- Multi-layered networks can represent arbitrary functions
  - Turing complete (assuming infinite depth)



Input Hidden layer h_1 Hidden layer h_2 Output

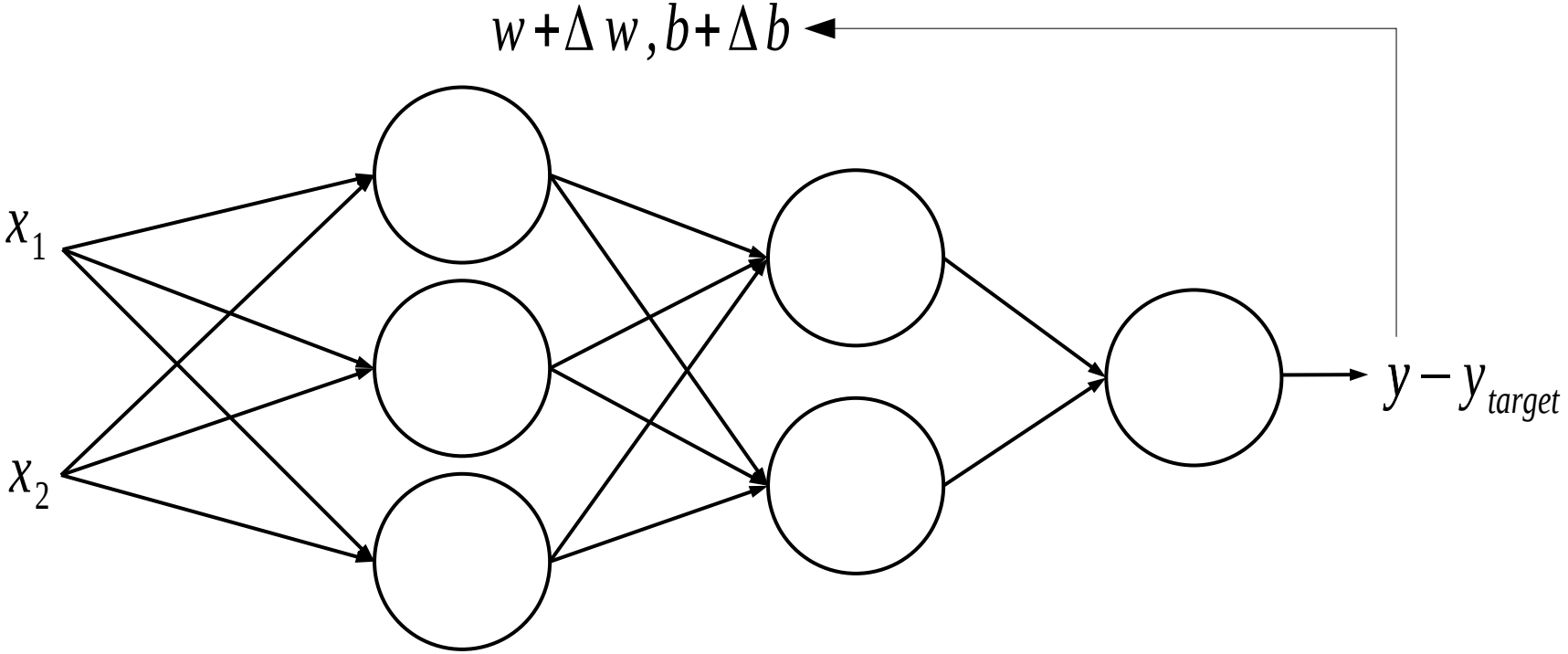
$$h_1 = W_1 \cdot x + b_1$$
$$h_2 = W_2 \cdot h_1 + b_2$$
$$y = W_3 \cdot h_2 + b_3$$

$$W_1 = [w_{1,1} | w_{1,2} | w_{1,3}] \in \mathbb{R}^{2\times 3}, b_1 = [b_{1,1} | b_{1,2} | b_{1,3}]^T \in \mathbb{R}^3$$
$$W_2 = [w_{2,1} | w_{2,2}] \in \mathbb{R}^{3\times 2}, b_2 = [b_{2,1} | b_{2,2}]^T \in \mathbb{R}^2$$
$$W_3 = [w_{3,1}] \in \mathbb{R}^{2\times 1}, b_3 = [b_{3,1}]^T \in \mathbb{R}^1$$

NCT

Parameter estimation - Training

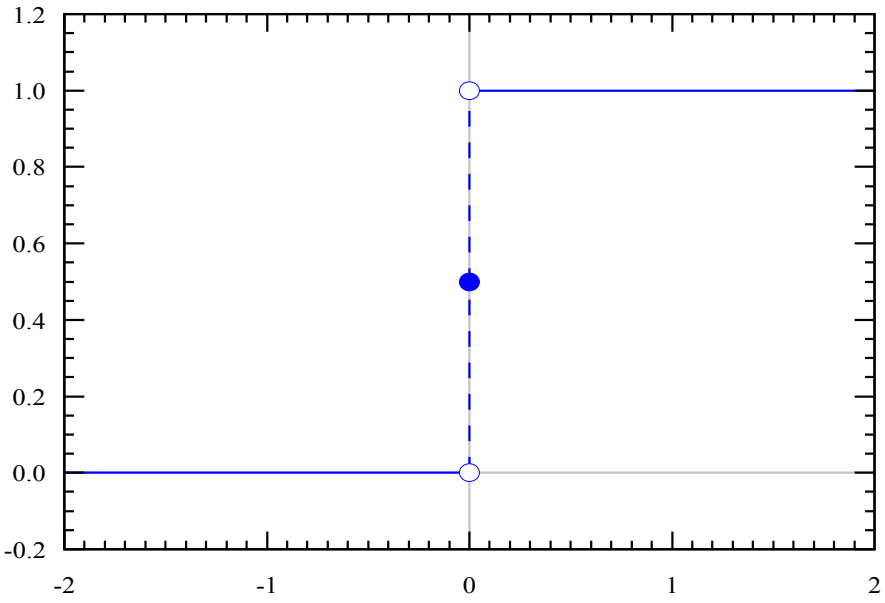
- Multi-layered networks can represent arbitrary functions
  - Given x and y
  - How to determine network parameters?



- Through small changes of the weights, the output of the network approaches the solution: The network is learning

Training - Problem

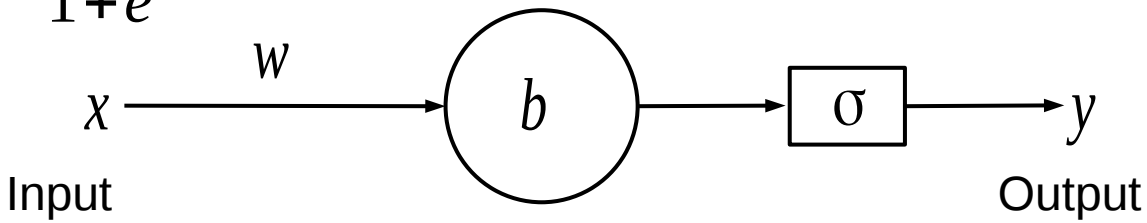
- But the network consists out of perceptrons
  - Small changes in weights can have large impact on the behavior of the entire network



Perceptron

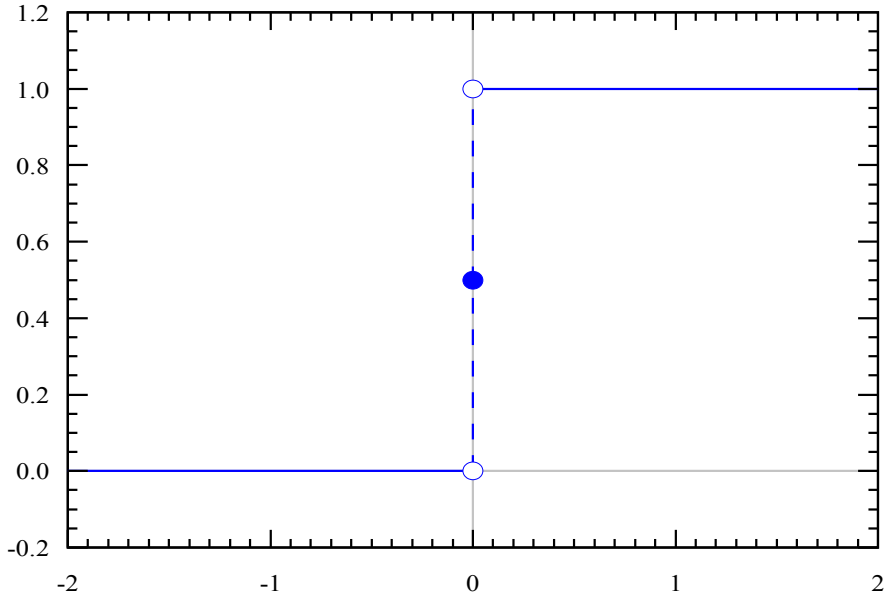
Training - Problem

- But the network consists out of perceptrons
  - Small changes in weights can have large impact on the behavior of the entire network
- Solution: Sigmoid non-linearity
  - $\sigma(z) = \frac{1}{1+e^{-z}}$

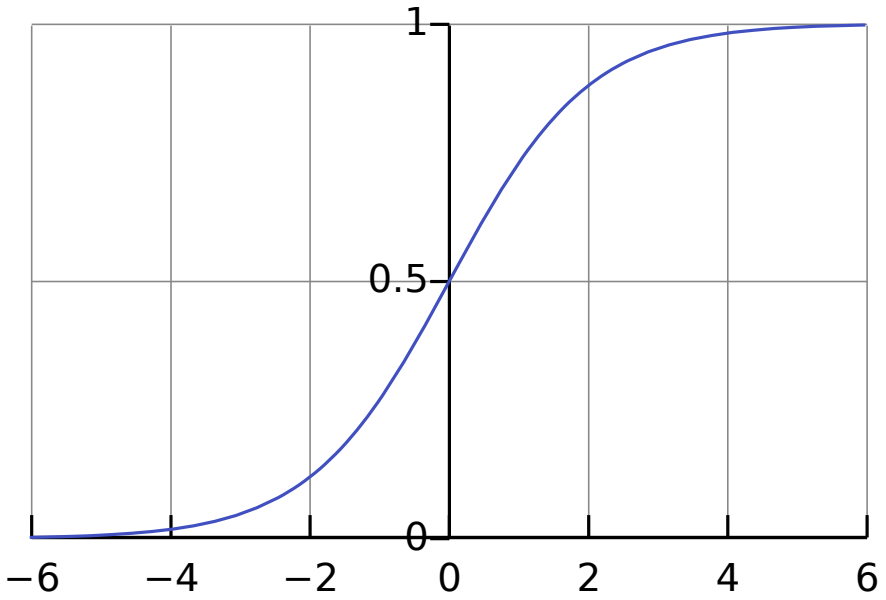


$$y = \sigma(w \cdot x + b)$$

Training - Sigmoid



Perceptron



Neuron + Sigmoid

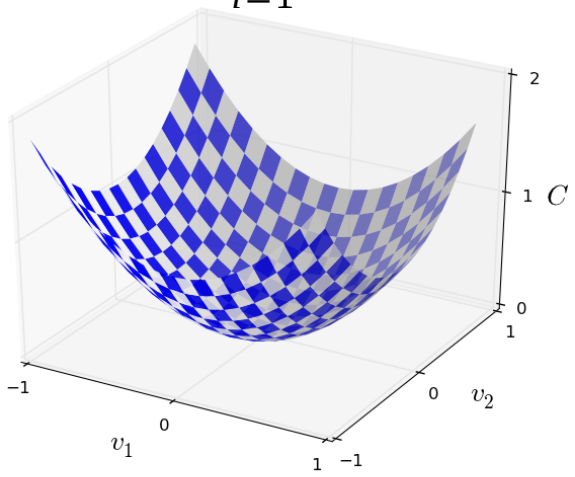
- Due to the smoothness, it is guaranteed that small changes in the weights only cause small changes in the output

NCT

Training – Gradient descent

- Optimization through minimization of cost function
  - Example: Euclidean norm

$$C(W,b)=\frac{1}{2n}\sum_{i=1}^{n}\|y(x_i)-y_{target}\|^2$$



- Gradient descent

$$W_k=W_k-\eta\frac{\delta C}{\delta W_k}$$

$$b_k=b_k-\eta\frac{\delta C}{\delta b_k}$$

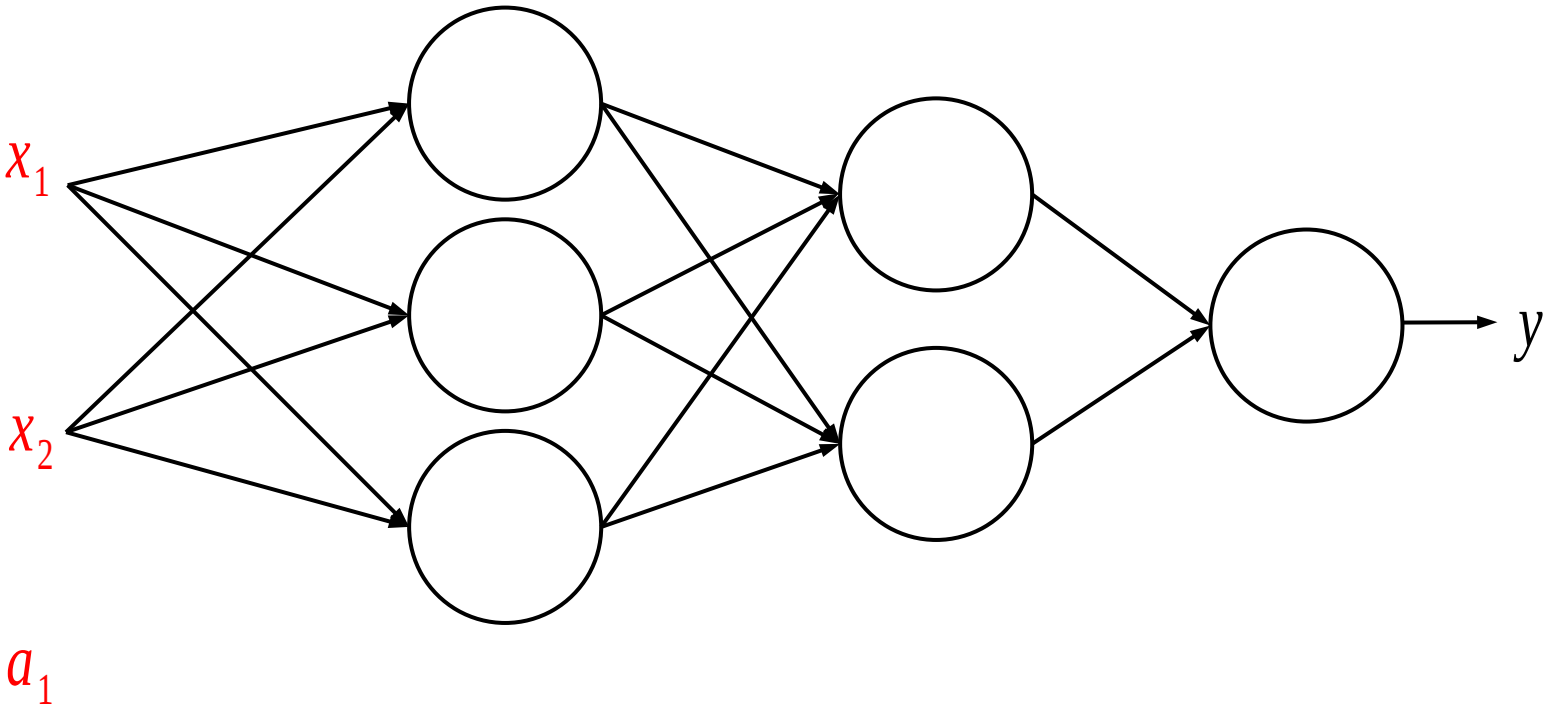
η: Learning rate, determines step size

- η too large: Minimums are „skipped”
- η too small: Local Minimums cannot be escaped

NCT

Training - Backpropagation

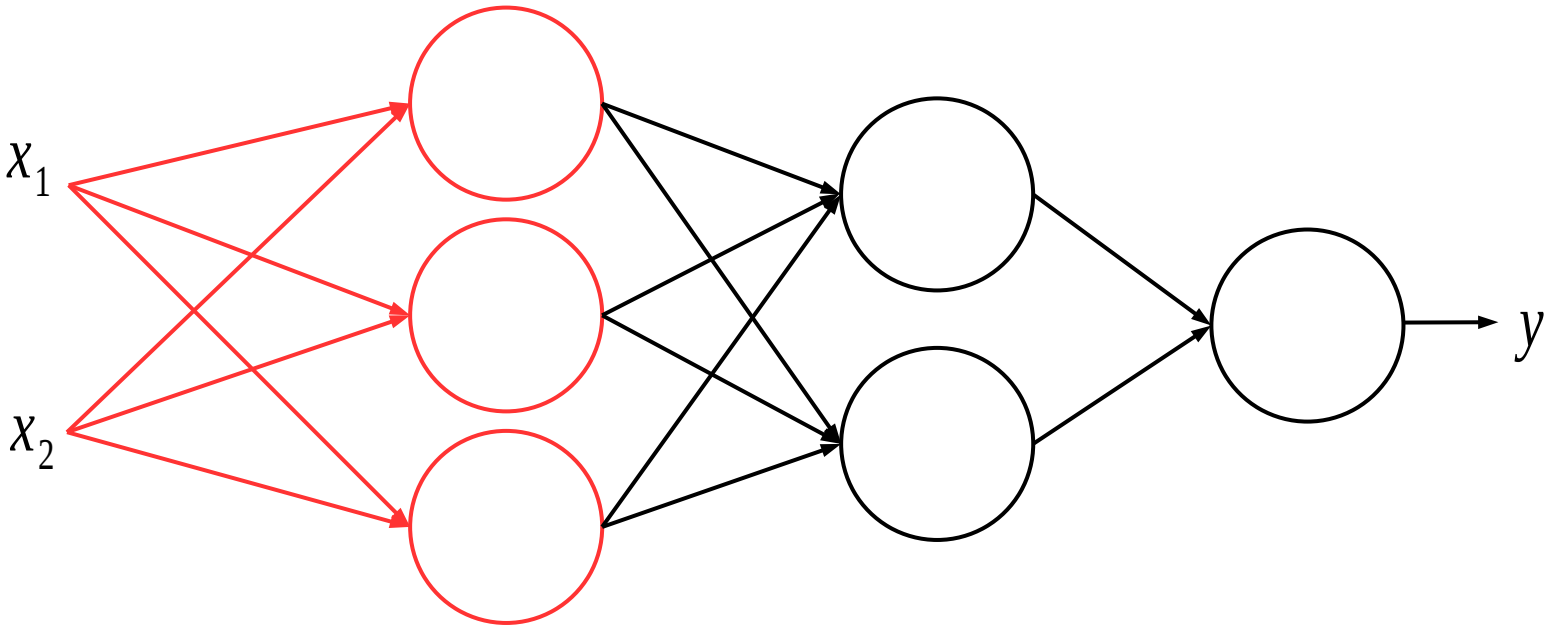
- Determine the partial derivatives through Backpropagation
- 1st step: Activation of the input layer $a_1 = x$



Training - Backpropagation

- 2nd step: For the remaining layers (2...L) compute:

$$z_l = W_l \cdot a_{l-1} + b_l \qquad l = 2, \ldots, L$$
$$a_l = \sigma(z_l)$$

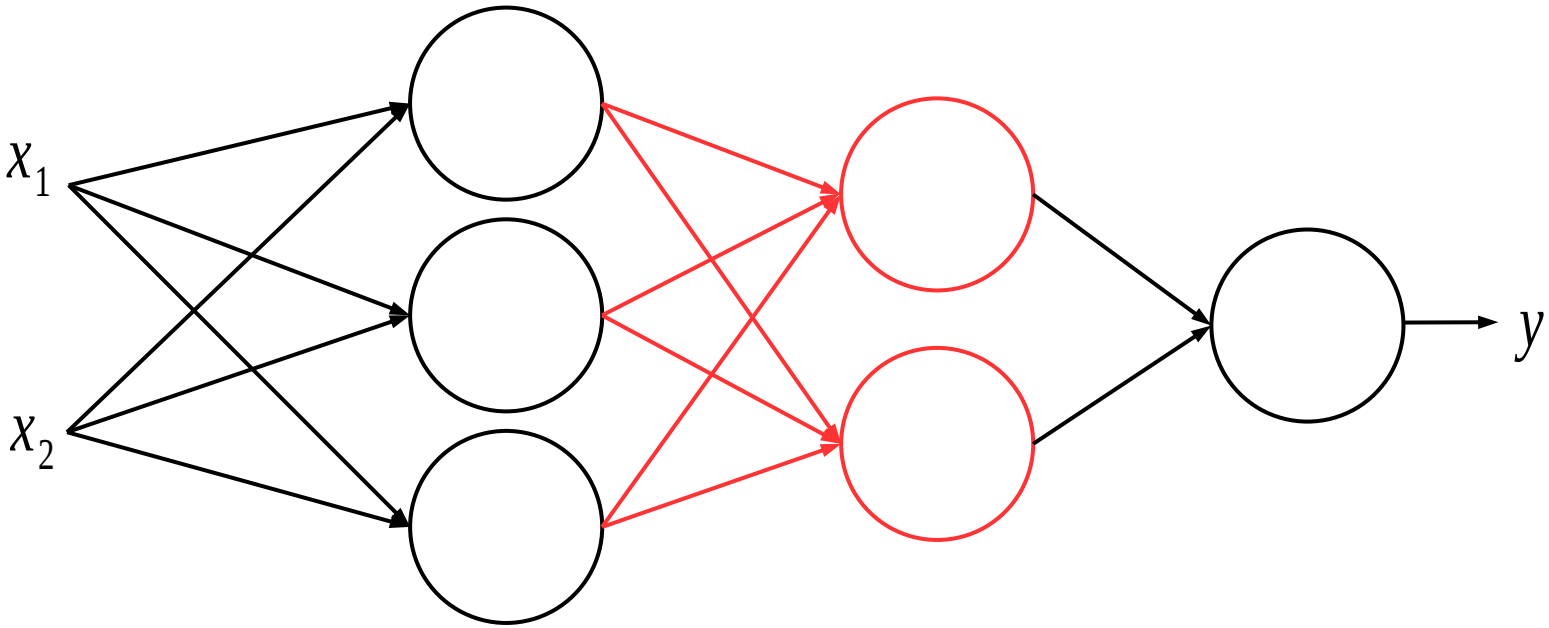


NCT

Training - Backpropagation

- 2nd step: For the remaining layers (2...L) compute:

$$z_l = W_l \cdot a_{l-1} + b_l \qquad l = 2, \ldots, L$$
$$a_l = \sigma(z_l)$$



a_1

$z_2 = W_2 \cdot a_1 + b_2$
$a_2 = \sigma(z_2)$

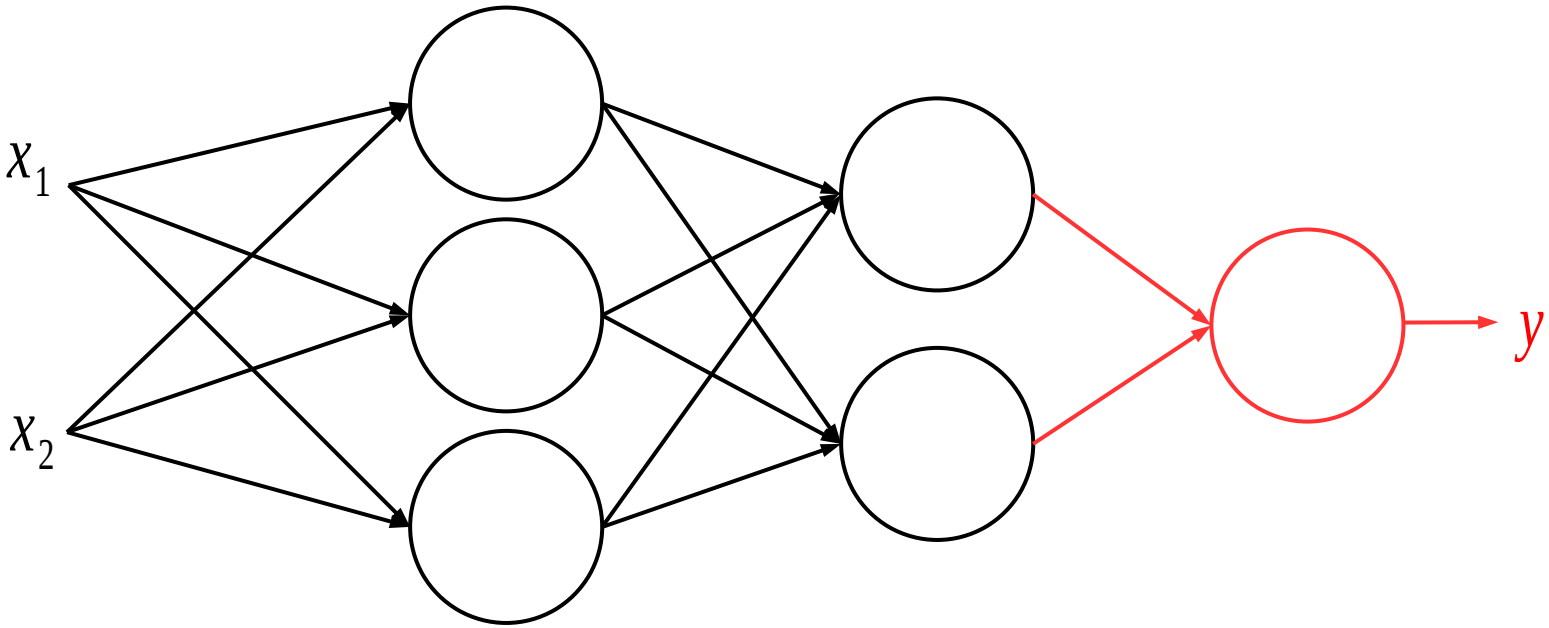
$z_3 = W_3 \cdot a_2 + b_3$
$a_3 = \sigma(z_3)$

NCT

Training - Backpropagation

- 2nd step: For the remaining layers (2...L) compute:

$$z_l = W_l \cdot a_{l-1} + b_l \qquad l = 2, \ldots, L$$
$$a_l = \sigma(z_l)$$



a_1

$z_2 = W_2 \cdot a_1 + b_2$
$a_2 = \sigma(z_2)$

$z_3 = W_3 \cdot a_2 + b_3$
$a_3 = \sigma(z_3)$

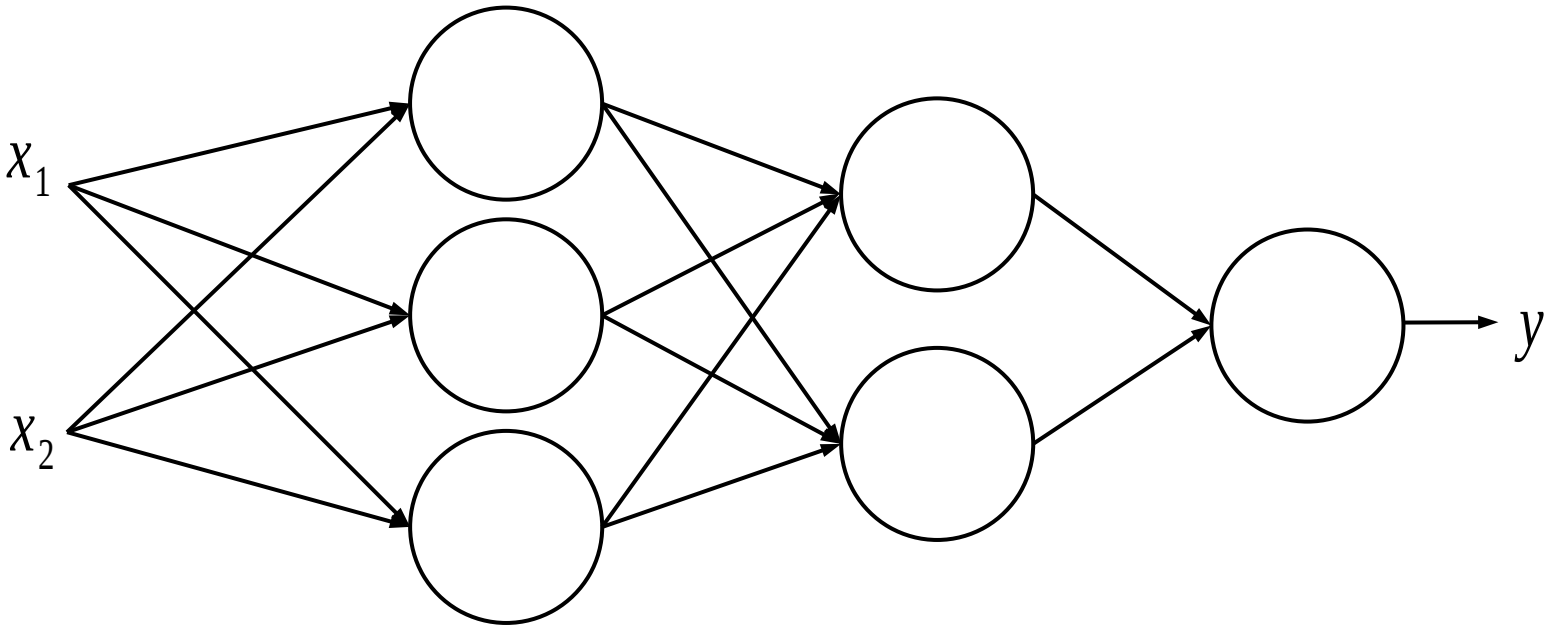
$z_4 = W_4 \cdot a_3 + b_4$
$y = a_4 = \sigma(z_4)$

NCT

Training - Backpropagation

- 3rd step: Compute output error

$$C(W,b)=\frac{1}{2n}\sum_{i=1}^{n}\|a_L-y_{target}\|^2$$



NCT

Training - Backpropagation

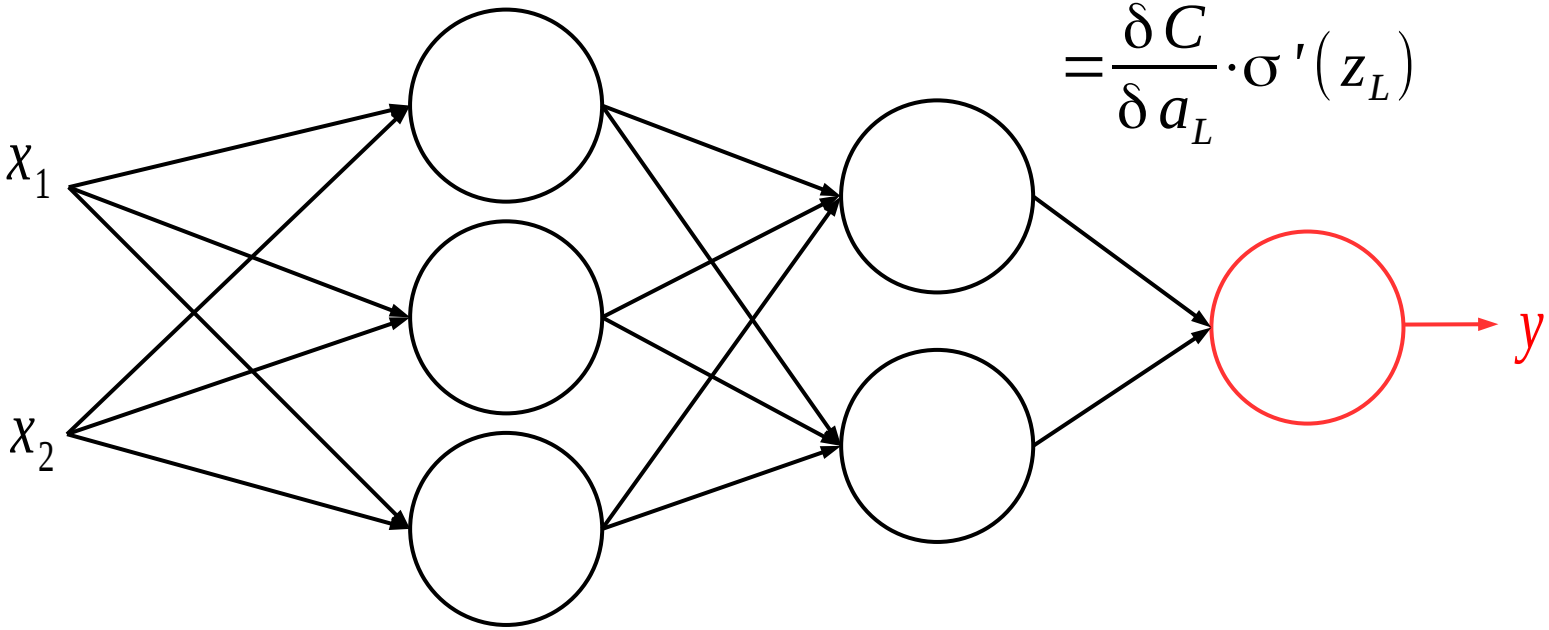
- 3rd step: Compute output error

$$C(W,b)=\frac{1}{2n}\sum_{i=1}^{n}\|a_L-y_{target}\|^2$$

$$\delta_L=\frac{\delta C}{\delta z_L}$$

$$=\frac{\delta C}{\delta a_L}\cdot\frac{\delta a_L}{\delta z_L}$$

$$=\frac{\delta C}{\delta a_L}\cdot\sigma'(z_L)$$



$$a_1$$

$$z_2=W_2\cdot a_1+b_2 \qquad a_2=\sigma(z_2)$$

$$z_3=W_3\cdot a_2+b_3 \qquad a_3=\sigma(z_3)$$

$$z_4=W_4\cdot a_3+b_4 \qquad y=a_4=\sigma(z_4)$$

NCT

Training - Backpropagation

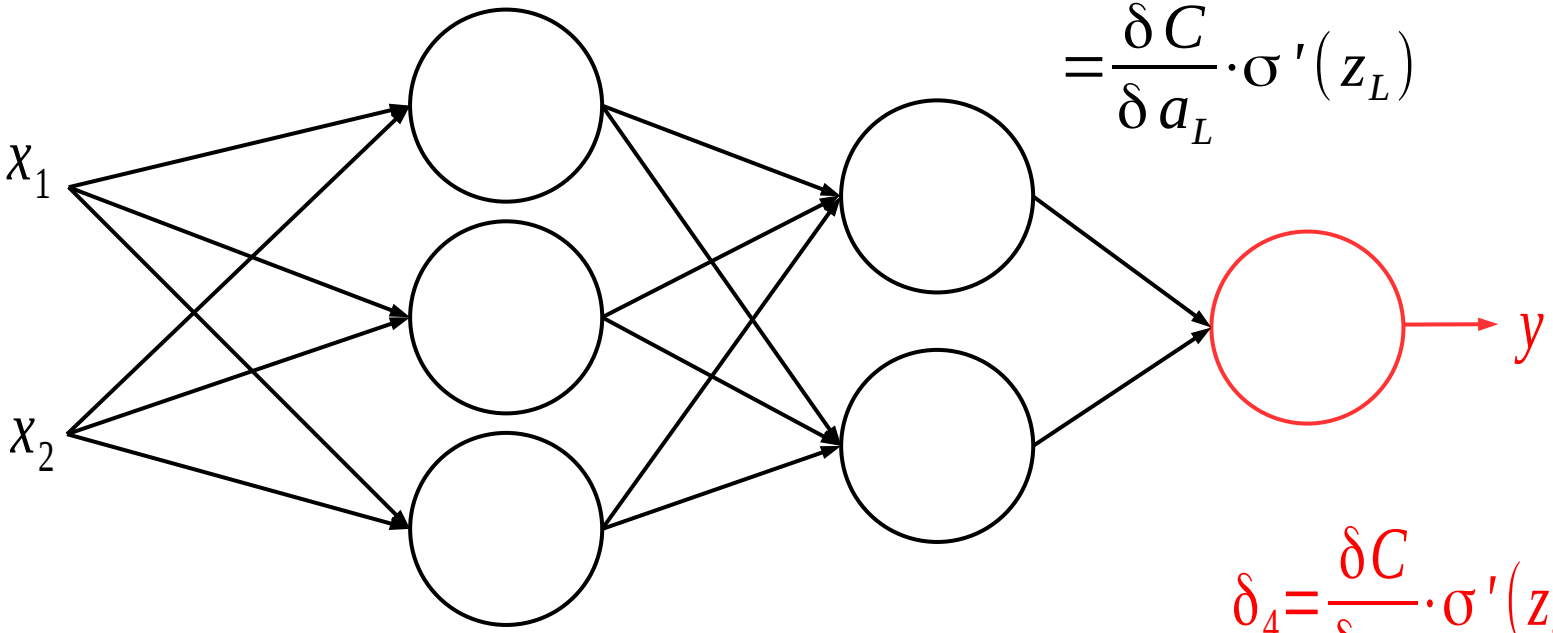
- 3rd step: Compute output error

$$C(W,b)=\frac{1}{2n}\sum_{i=1}^{n}\|a_L-y_{target}\|^2$$

$$\delta_L=\frac{\delta C}{\delta z_L}$$

$$=\frac{\delta C}{\delta a_L}\cdot\frac{\delta a_L}{\delta z_L}$$

$$=\frac{\delta C}{\delta a_L}\cdot\sigma'(z_L)$$



$$\delta_4=\frac{\delta C}{\delta a_4}\cdot\sigma'(z_4)$$

a_1

$z_2=W_2\cdot a_1+b_2$

$a_2=\sigma(z_2)$

$z_3=W_3\cdot a_2+b_3$

$a_3=\sigma(z_3)$

$z_4=W_4\cdot a_3+b_4$

$y=a_4=\sigma(z_4)$

NCT

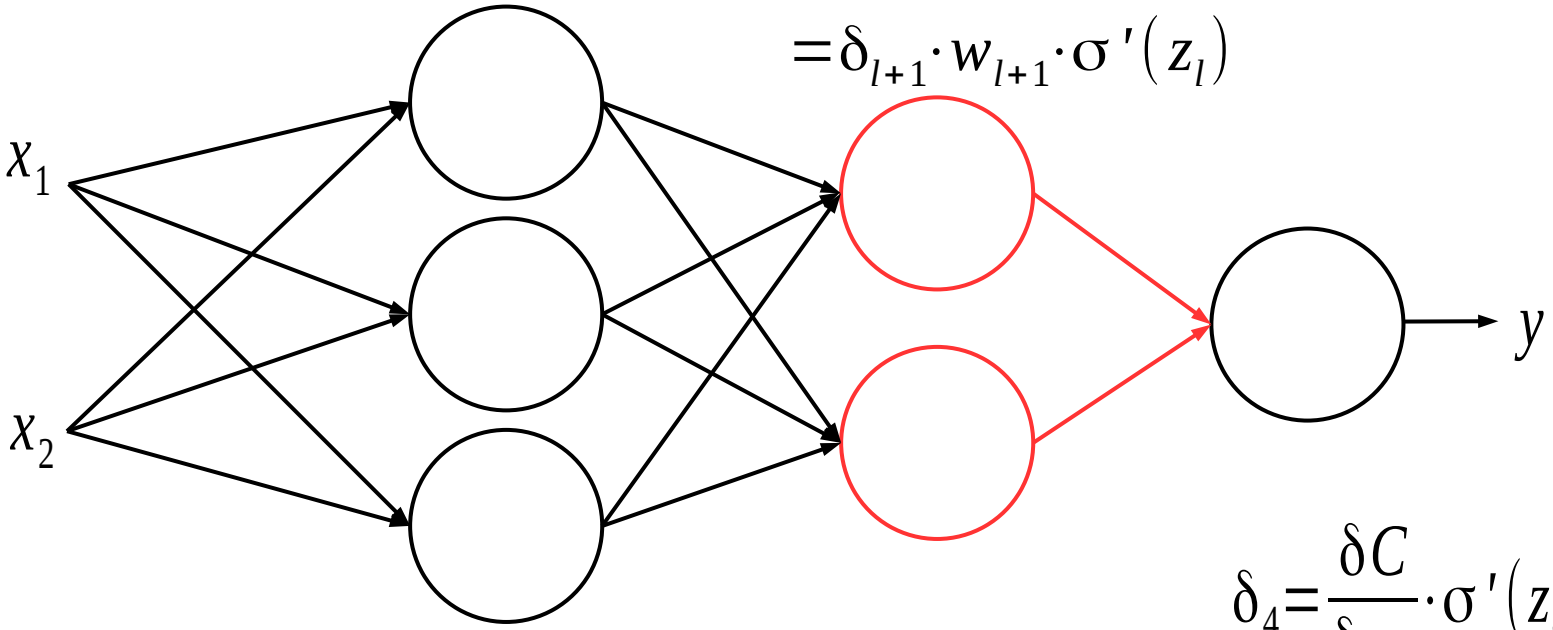
Training - Backpropagation

- 4th step: Propage output error back to the remaining layers

$$\delta_l = \frac{\delta C}{\delta z_l} = \frac{\delta C}{\delta z_{l+1}} \cdot \frac{\delta z_{l+1}}{\delta z_l}$$

$$= \delta_{l+1} \cdot \frac{\delta z_{l+1}}{\delta a_l} \cdot \frac{\delta a_l}{\delta z_l} \qquad l = L-1, \ldots, 2$$

$$= \delta_{l+1} \cdot w_{l+1} \cdot \sigma'(z_l)$$



$$\delta_4 = \frac{\delta C}{\delta a_4} \cdot \sigma'(z_4)$$

a_1

$z_2 = W_2 \cdot a_1 + b_2$
$a_2 = \sigma(z_2)$

$z_3 = W_3 \cdot a_2 + b_3$
$a_3 = \sigma(z_3)$

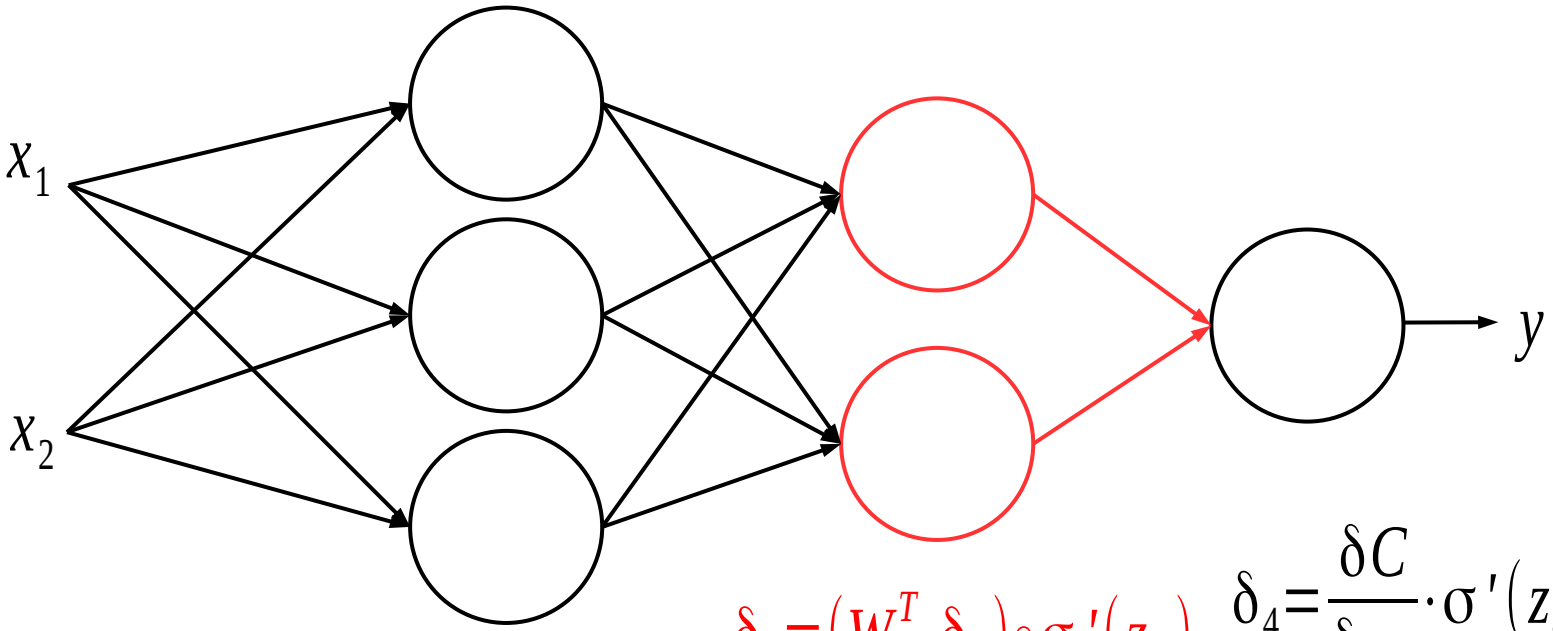
$z_4 = W_4 \cdot a_3 + b_4$
$y = a_4 = \sigma(z_4)$

NCT

Training - Backpropagation

- 4th step: Propage output error back to the remaining layers

$$\delta_l = (W_{l+1}^T \cdot \delta_{l+1}) \circ \sigma'(z_l) \qquad l = L-1, \ldots, 2$$



$$\delta_3 = (W_4^T \cdot \delta_4) \circ \sigma'(z_4) \qquad \delta_4 = \frac{\delta C}{\delta a_4} \cdot \sigma'(z_4)$$

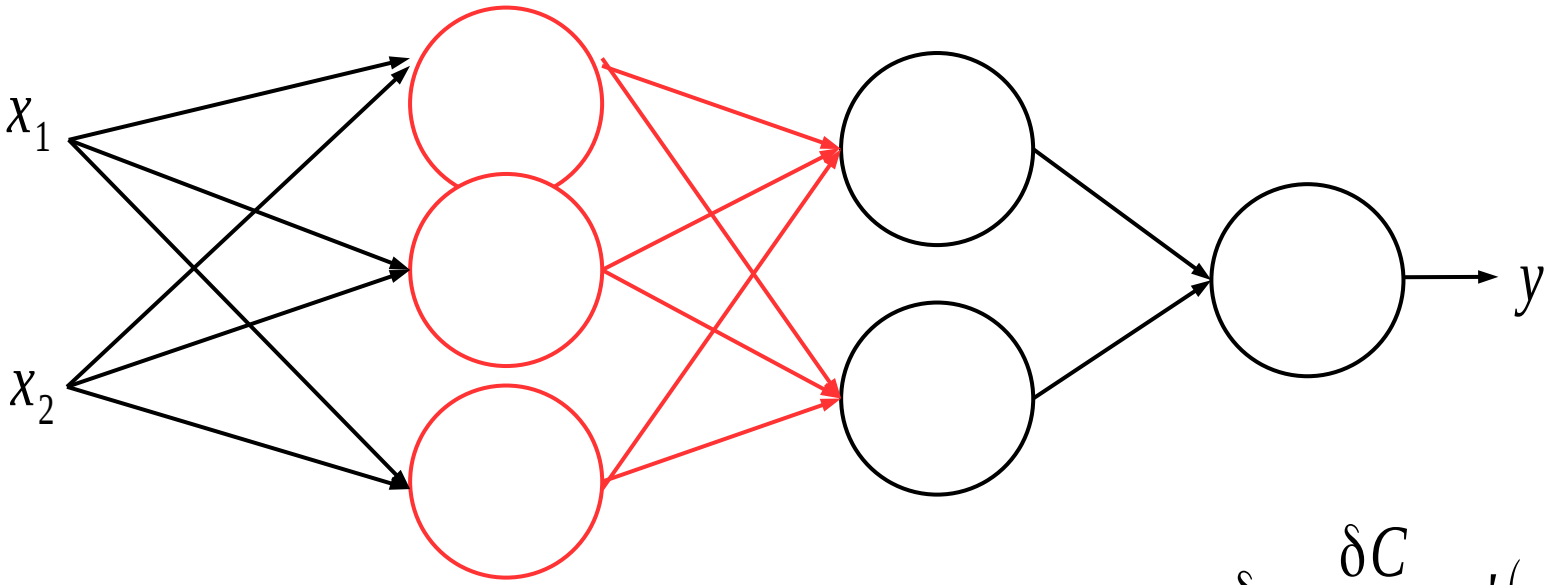
$$a_1 \qquad z_2 = W_2 \cdot a_1 + b_2 \qquad z_3 = W_3 \cdot a_2 + b_3 \qquad z_4 = W_4 \cdot a_3 + b_4$$
$$a_2 = \sigma(z_2) \qquad a_3 = \sigma(z_3) \qquad y = a_4 = \sigma(z_4)$$

NCT

Training - Backpropagation

- 4th step: Propage output error back to the remaining layers

$$\delta_l = (W_{l+1}^T \cdot \delta_{l+1}) \circ \sigma'(z_l) \qquad l = L-1, \ldots, 2$$



$$\delta_2 = (W_3^T \cdot \delta_3) \circ \sigma'(z_2) \quad \delta_3 = (W_4^T \cdot \delta_4) \circ \sigma'(z_3) \quad \delta_4 = \frac{\delta C}{\delta a_4} \cdot \sigma'(z_4)$$

$$a_1 \qquad z_2 = W_2 \cdot a_1 + b_2 \qquad z_3 = W_3 \cdot a_2 + b_3 \qquad z_4 = W_4 \cdot a_3 + b_4$$
$$a_2 = \sigma(z_2) \qquad a_3 = \sigma(z_3) \qquad y = a_4 = \sigma(z_4)$$

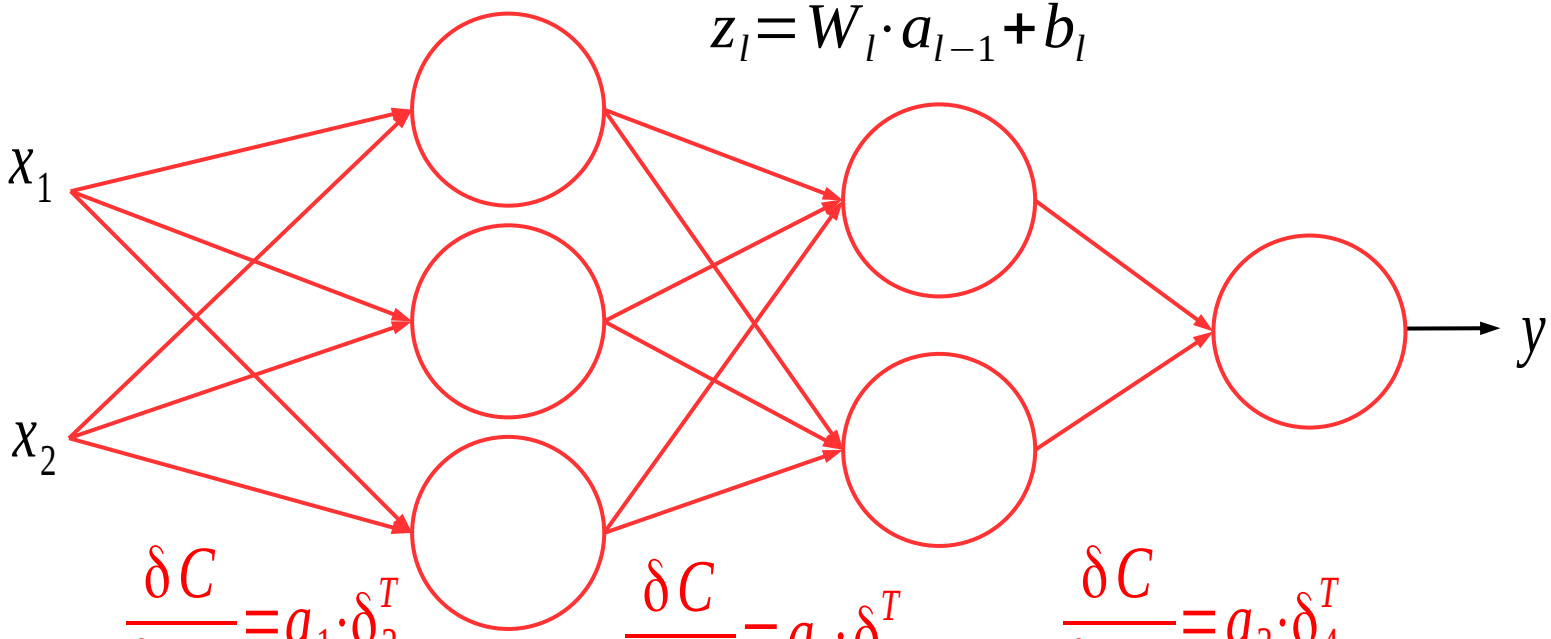
NCT

Training - Backpropagation

- 5th step: Compute partial derivatives

$$\frac{\delta C}{\delta W_l} = \frac{\delta C}{\delta z_l} \cdot \frac{\delta z_l}{\delta W_l} = \frac{\delta z_l}{\delta W_l} \cdot \delta_l^T = a_{l-1} \cdot \delta_l^T \qquad \frac{\delta C}{\delta b_l} = \frac{\delta C}{\delta z_l} \cdot \frac{\delta z_l}{\delta b_l} = \delta_l$$

$$z_l = W_l \cdot a_{l-1} + b_l$$



$$\frac{\delta C}{\delta W_2} = a_1 \cdot \delta_2^T \qquad \frac{\delta C}{\delta W_3} = a_2 \cdot \delta_3^T \qquad \frac{\delta C}{\delta W_4} = a_3 \cdot \delta_4^T$$

$$\frac{\delta C}{\delta b_2} = \delta_2 \qquad \frac{\delta C}{\delta b_3} = \delta_3 \qquad \frac{\delta C}{\delta b_4} = \delta_4$$

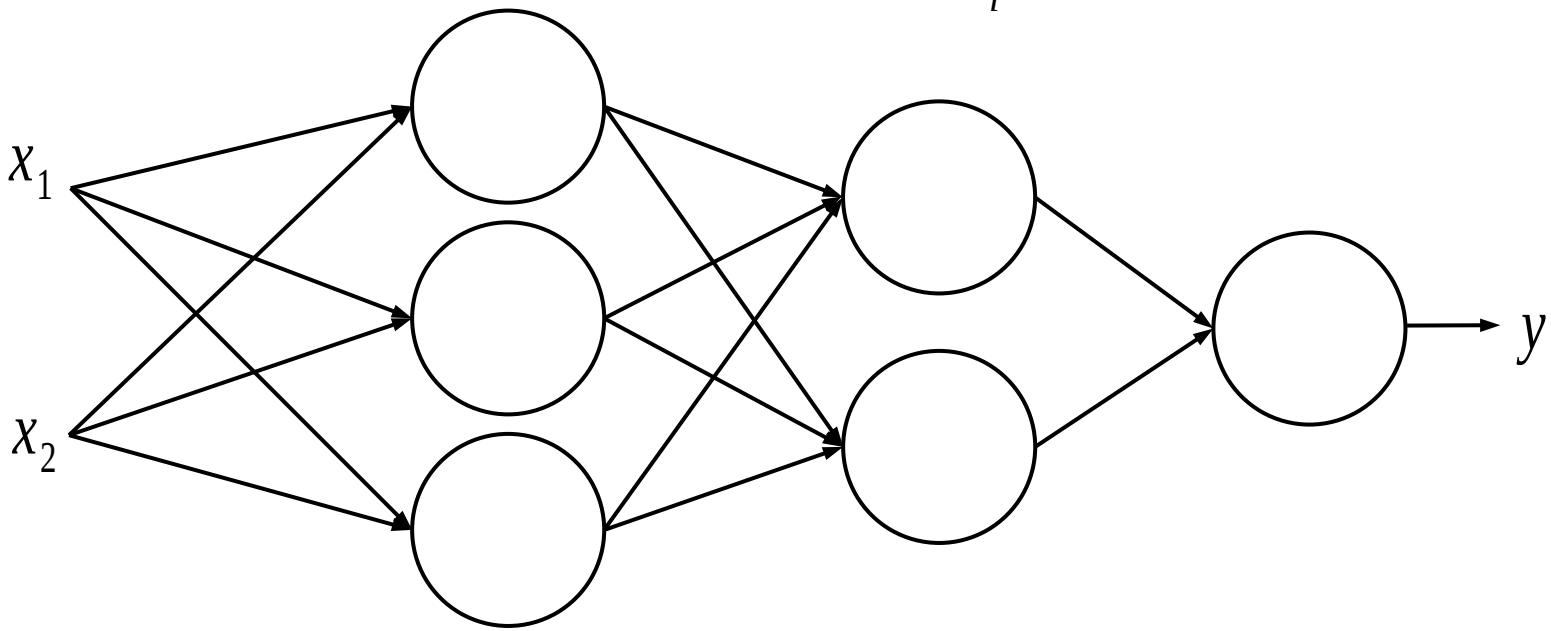
NCT

Training - Backpropagation

- 6th step: Gradient descent

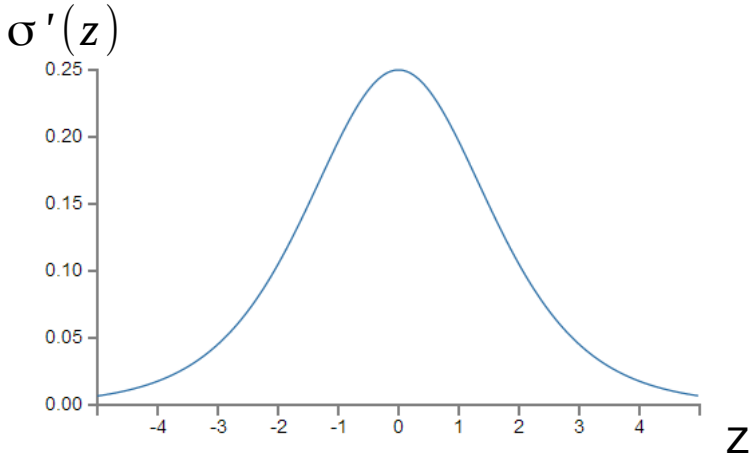
$$W_l = W_l - \eta \frac{\delta C}{\delta W_l}$$

$$b_l = b_l - \eta \frac{\delta C}{\delta b_l}$$



Deep networks: Deep Learning

- Arbitrary deep networks are powerful (Turing-complete)
  - Problem when training with Backpropagation
  - $\delta_L = \frac{\delta C}{\delta a_L} \cdot \sigma'(z_L)$
  - $\delta_l = (w_{l+1}^T \cdot \delta_{l+1}) \circ \sigma'(z_l) \qquad l = L-1, \ldots, 2$



  - Through the influence of the derivative of the sigmoid, the gradient is shrinking

    => Vanishing Gradients

NCT

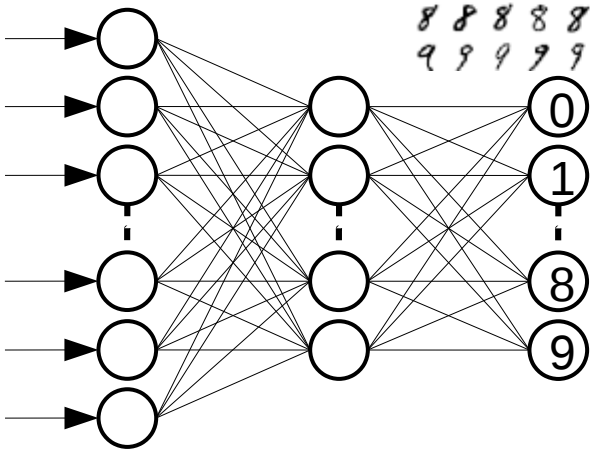
Deep networks: Deep Learning

- Solution: different non-linearities
  - Tanh $\sigma(z) = \tanh(z)$
  - ReLu $\sigma(z) = max(0, z)$
  - SoftMax $\sigma(z)_j = \frac{e^z_j}{\sum_k e^z_k}$
    - Sum of the output neurons is normed (add up to 1)
    - Useful for classification (for N classes, N output neurons)
    - Output can be interpreted as class probabilities
- Derivatives have larger value range
  - Vanishing Gradient still not impossible
  - Also opposite (exploding Gradient) possible

NCT

Classification with neural networks

- Example problem: Recognizing handwritten digits
  - MNIST dataset of handwritten digits
  - Images 28x28p
  - Centered on digit



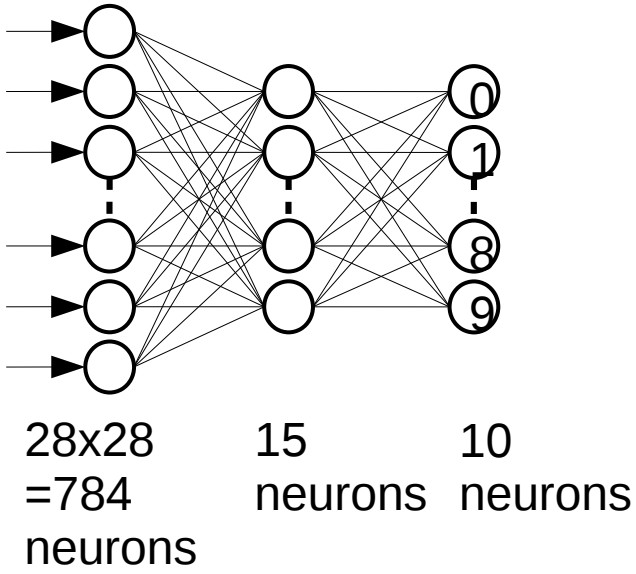
28x28 =784 neurons

15 neurons

10 neurons

Classification with neural networks

- Recognizing handwritten digits

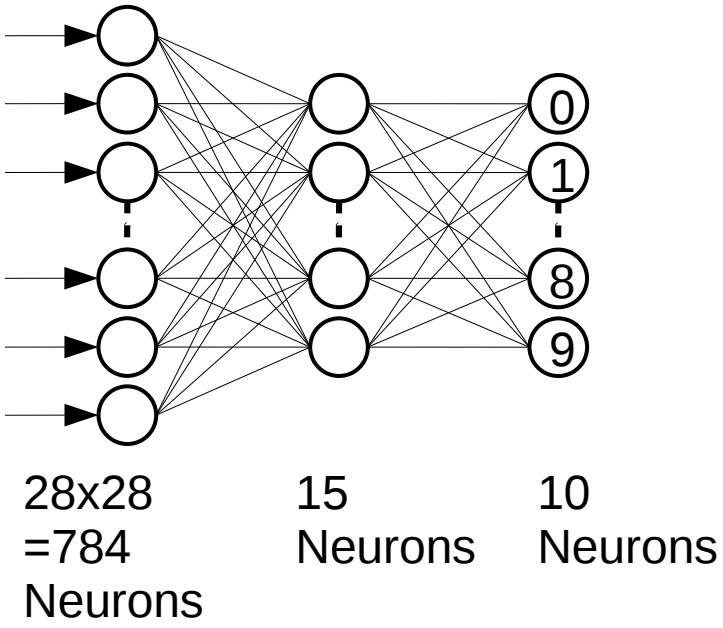


- One output neuron per class: one-hot encoding
- Non-linearity for output neurons: Softmax
  - Since classification problem
- As cost function: Categorical Cross-entropy
  - $C = -\sum_{j}^{C} t_j \cdot \log y_j$

t_j: Groundtruth for class *i*
y_j: Prediction for class *i*

Classification with neural networks

- Recognizing handwritten digits



- Each layer extracts feature from the previous layer
  - E.g. Layer 1 could learn to recognize lines, layer 2 could use the lines to recognize shapes (Triangles, …)
- What happens when a digit that is correctly classified is moved a few pixels?

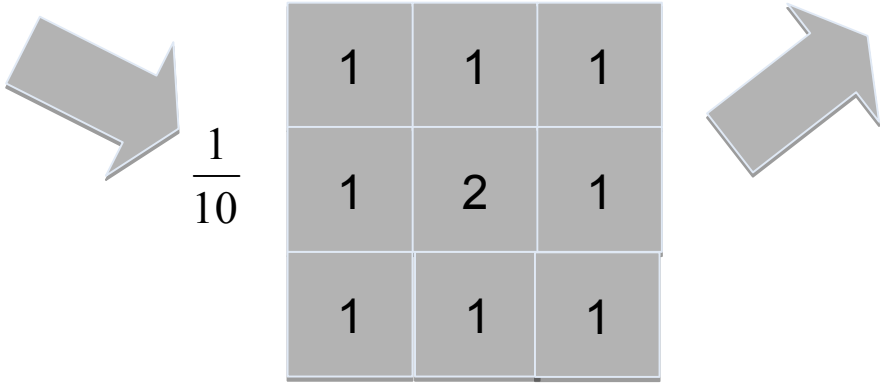
NCT

Example

Input

0	2	2	2	0
1	10	2	10	1
2	2	20	2	2
1	10	5	10	1
0	2	10	2	1

Output



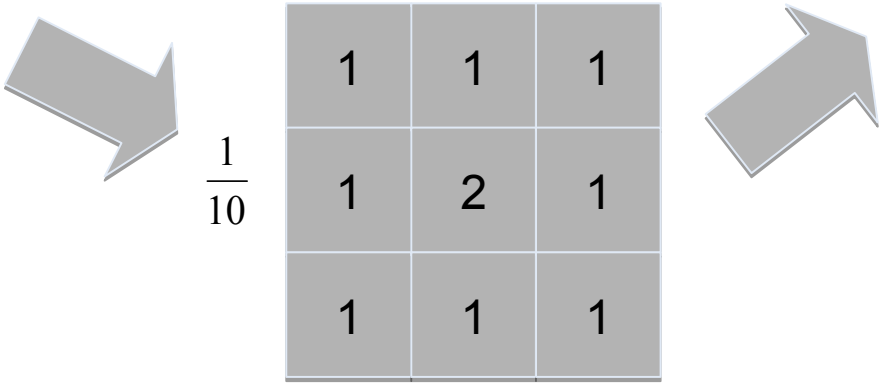
Example

Input

0	2	2	2	0
1	10	2	10	1
2	2	20	2	2
1	10	5	10	1
0	2	10	2	1

Output

?	?	?	?	?
?	**5,1**	**5,4**	**5,1**	?
?	**5,5**	**9,1**	**5,5**	?
?	**6,2**	**6,8**	**6,3**	?
?	?	?	?	?



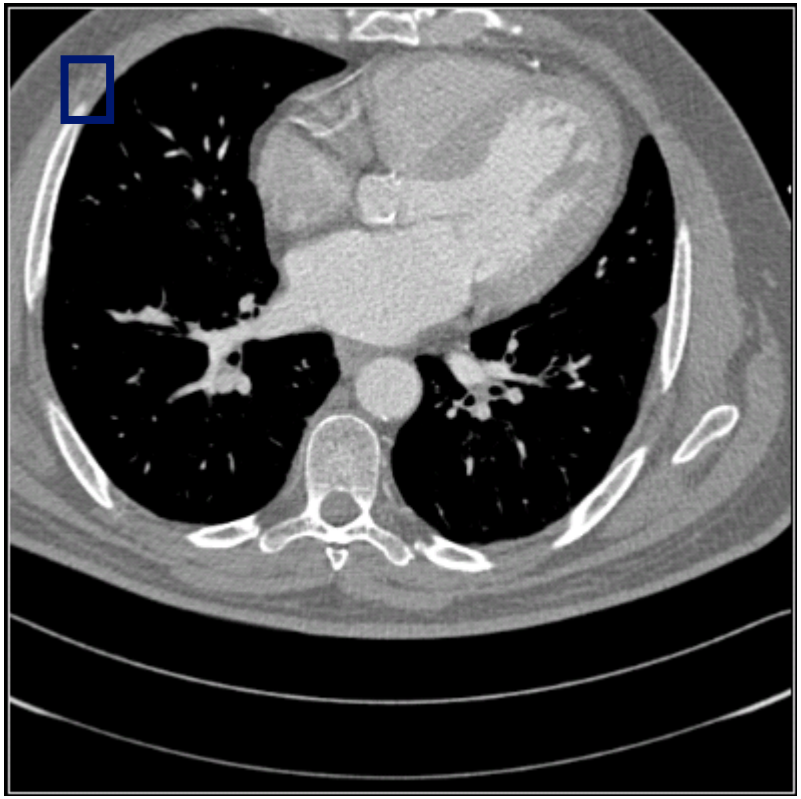
Gaussian filter

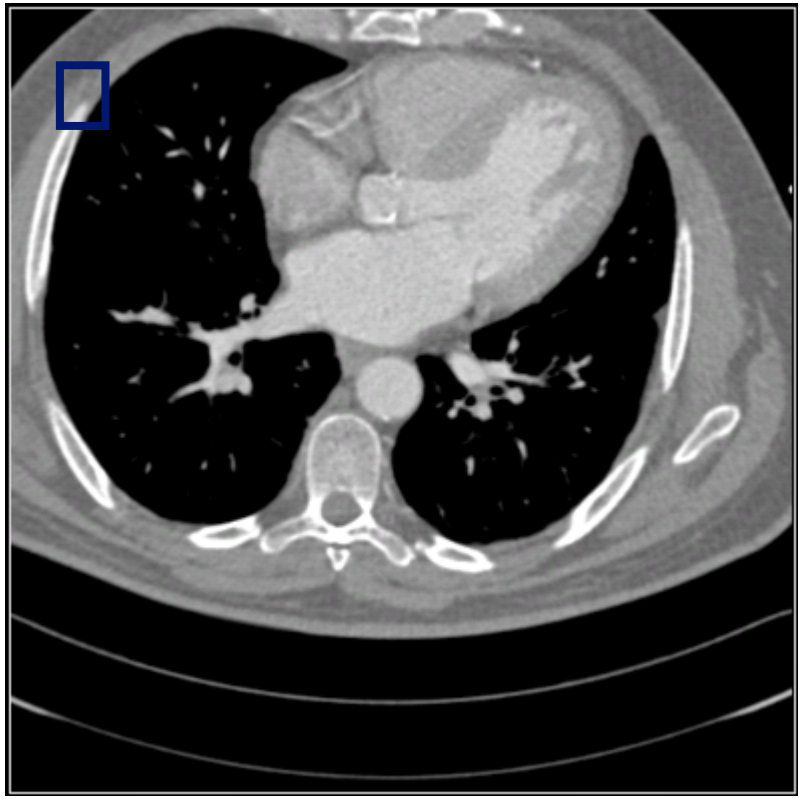
- Better smoothing filter than box filter
- Structure is inspired by Gaussian normal distribution
- Normal distribution can be approximated via binomial distribution
- Influence of the environment in dependence to the distance to the image center => no strong flattening of edges

$$\frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \cdot \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}}$$

$$\frac{1}{16} \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} = \frac{1}{4} \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array} \cdot \frac{1}{4} \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline \end{array}$$

Example: Gaussian filter





Sobel Filter

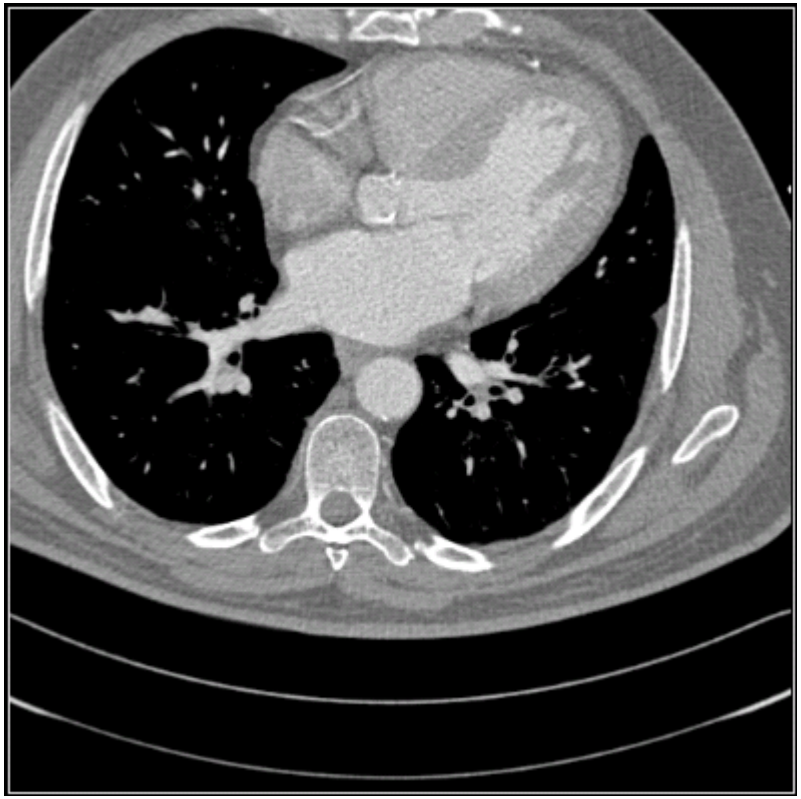
- Gaussian-based weighting the difference of the pixel values to enhance edges
- Sobel-X filter enhances vertical, Sobel-Y Filter horizontal edges

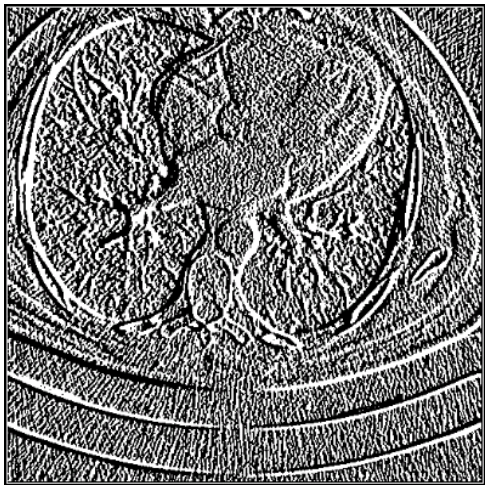
$$S_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

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

NCT

Example Sobel Filter





Convolutional Neural Networks (CNNs)

Input

0	2	2	2	0
1	10	2	10	1
2	2	20	2	2
1	10	5	10	1
0	2	10	2	1

Output

?	?	?	?	?
?	**5,1**	**5,4**	**5,1**	?
?	**5,5**	**9,1**	**5,5**	?
?	**6,2**	**6,8**	**6,3**	?
?	?	?	?	?

$\frac{1}{10}$

1	1	1
1	2	1
1	1	1

Gauss

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

Prewitt

$P_x =$

-1	0	1
-1	0	1
-1	0	1

$P_y =$

-1	-1	-1
0	0	0
1	1	1

Sobel

$S_x =$

-1	0	1
-2	0	2
-1	0	1

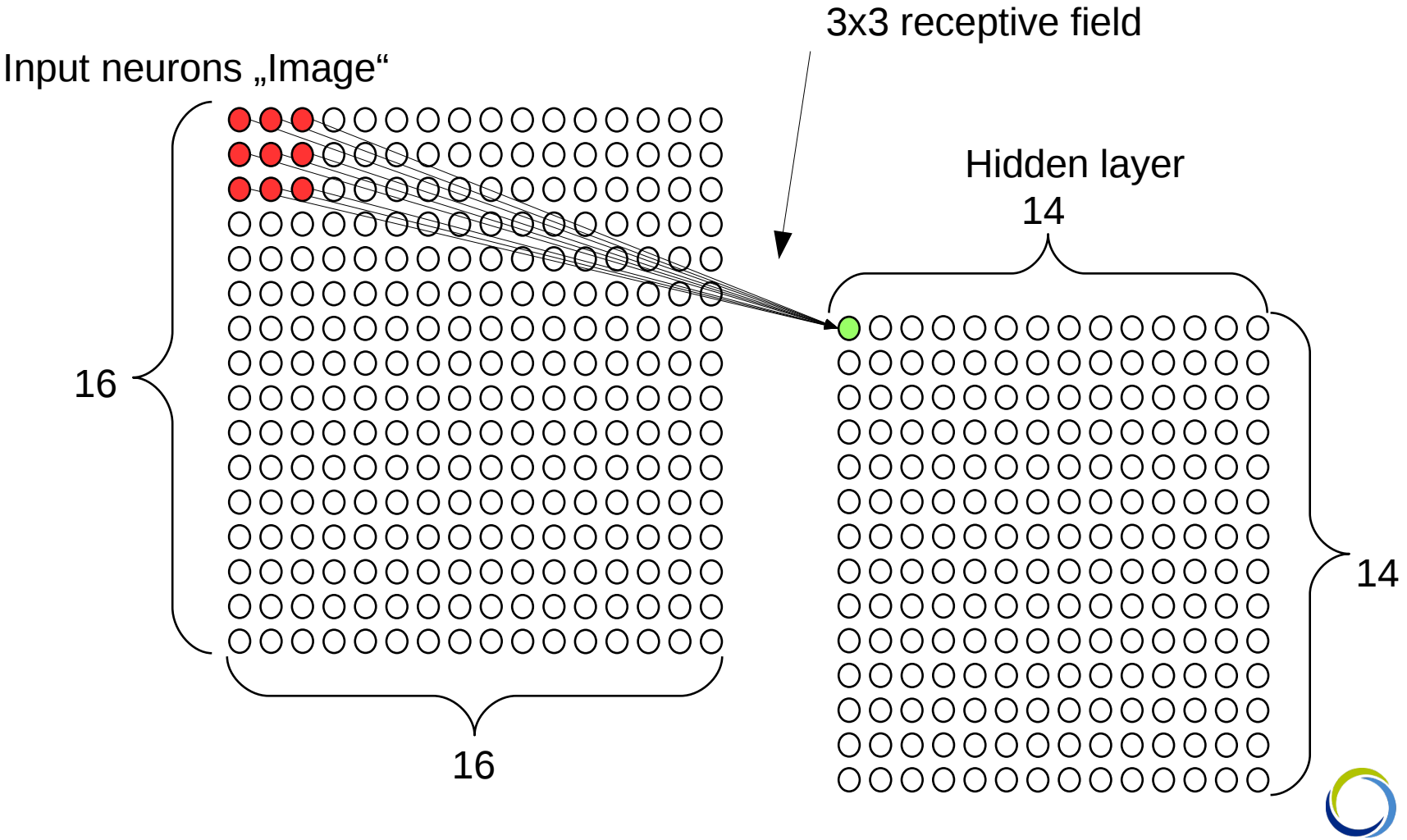
$S_y =$

-1	-2	-1
0	0	0
1	2	1

NCT

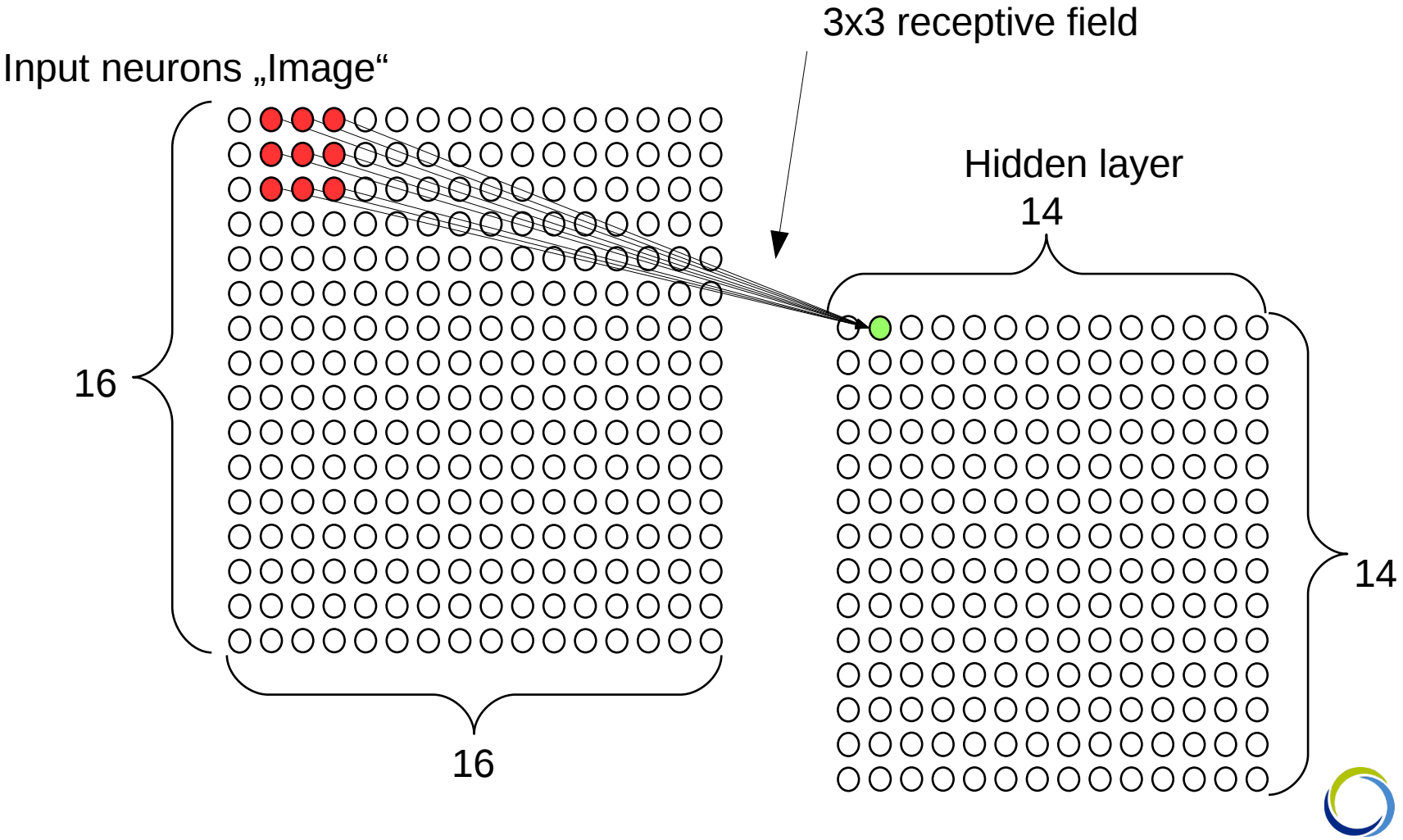
CNNs – Convolutional Layer

- Variant of neural network inspired by filters
  - No complete interconnectedness



CNNs – Convolutional Layer

- Variant of neural network inspired by filters
  - No complete interconnectedness



CNNs – Convolutional Layer

- Variant of neural network inspired by filters
  - No complete interconnectedness
  - Shared weights and biases
    - Combination of weights and bias is called kernel
  - Weight is a 3D-Tensor (# channels x height x width)
  - Output for the *j*,*k*th *n*euron of layer *l* for a 1x3x3 kernel

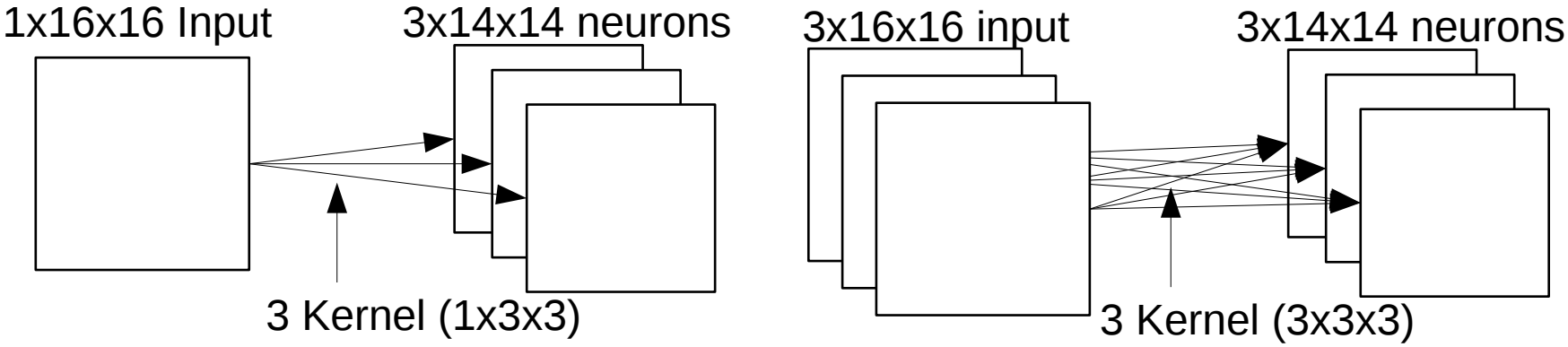
$$a_{l,j,k} = \sigma\left(b_l + \sum_{m=0}^{2}\sum_{n=0}^{2} w_{l,m,n} \cdot a_{l-1,j+m,k+n}\right)$$

  - All neurons of a receptive field compute the same feature, but on a different position
    - Feature can be found independent of position
  - Output matrix is called feature map

NCT

CNNs – Convolutional Layer

- For image processing multiple, varying feature maps are used

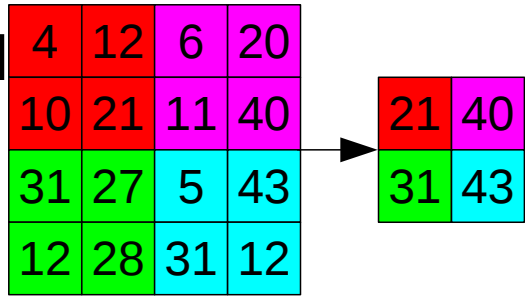


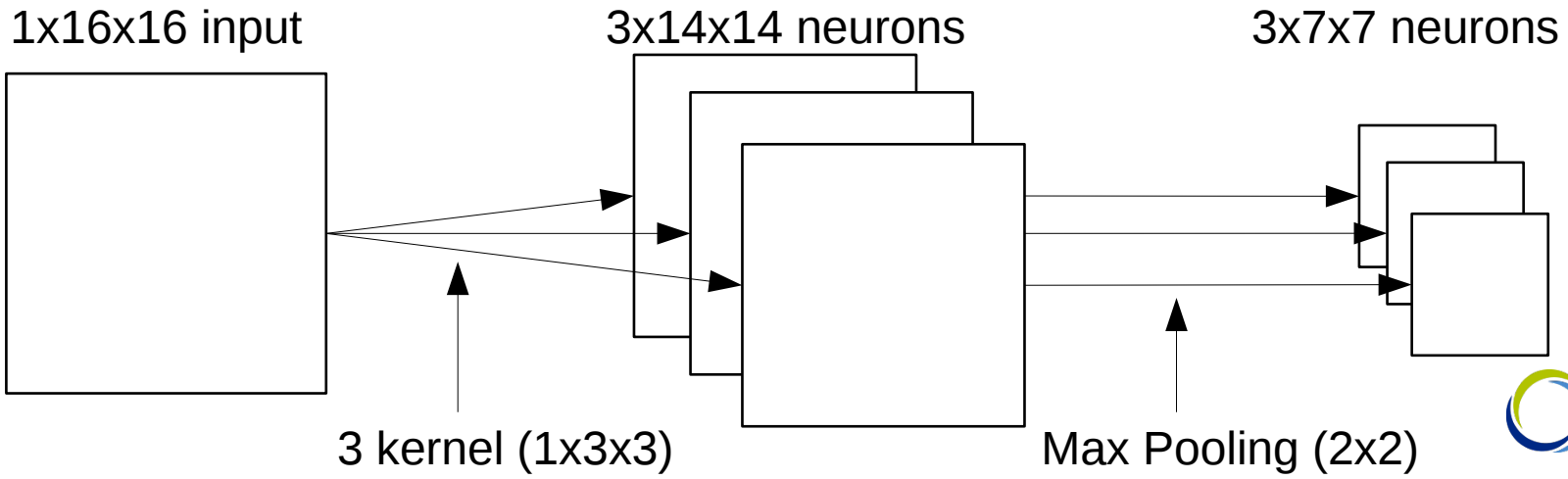
- Feature map for 3x3x3 Kernel has 28 parameters
  - 9 weights per channel: 27, plus 1 bias
  - For 20 feature maps 560 parameters
    - Independent of height and width of input image
  - Compare fully connected: 50372

NCT

CNNs – Max Pooling Layer

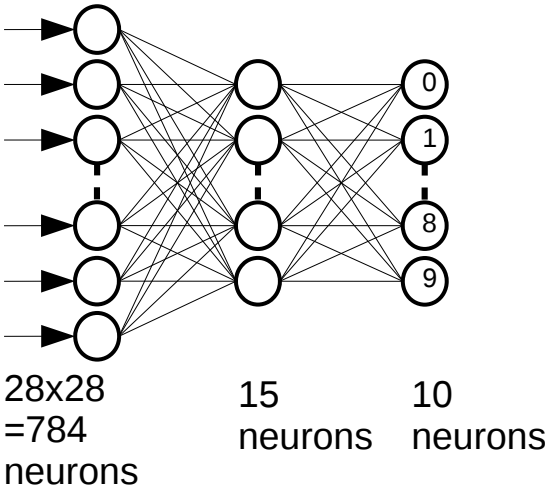
- Reduction of the feature map
  - Retains information if a feature was found
  - Smaller map for following steps
- Selects maximum value from a neighborhood
- Applied to each feature map

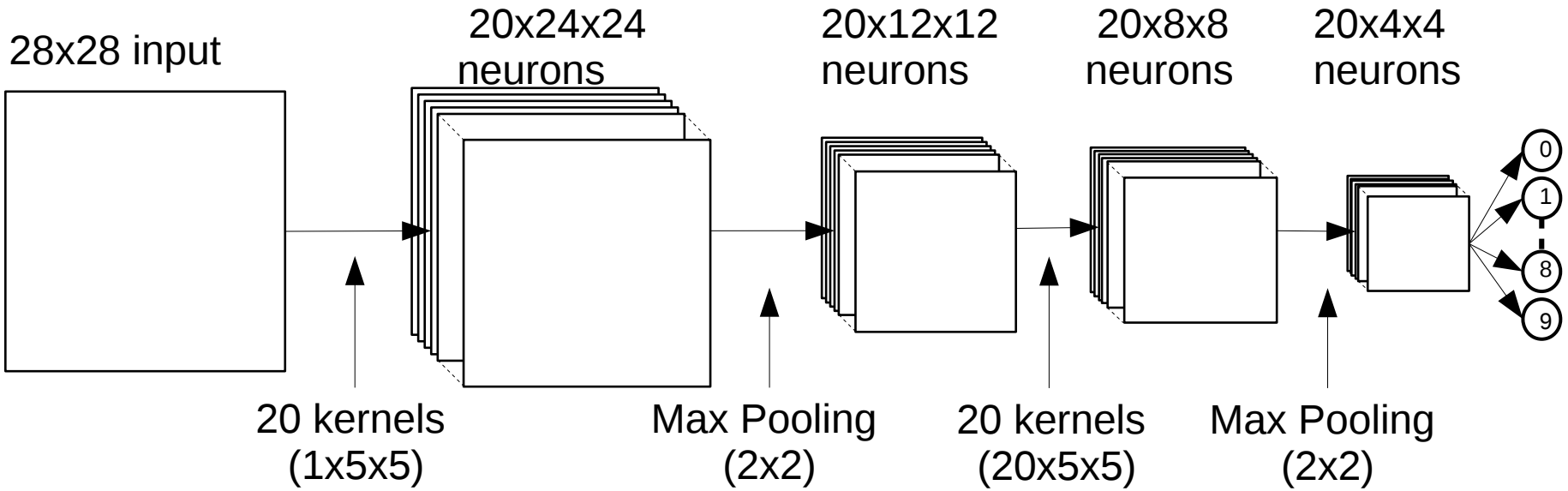




Classification with CNNs

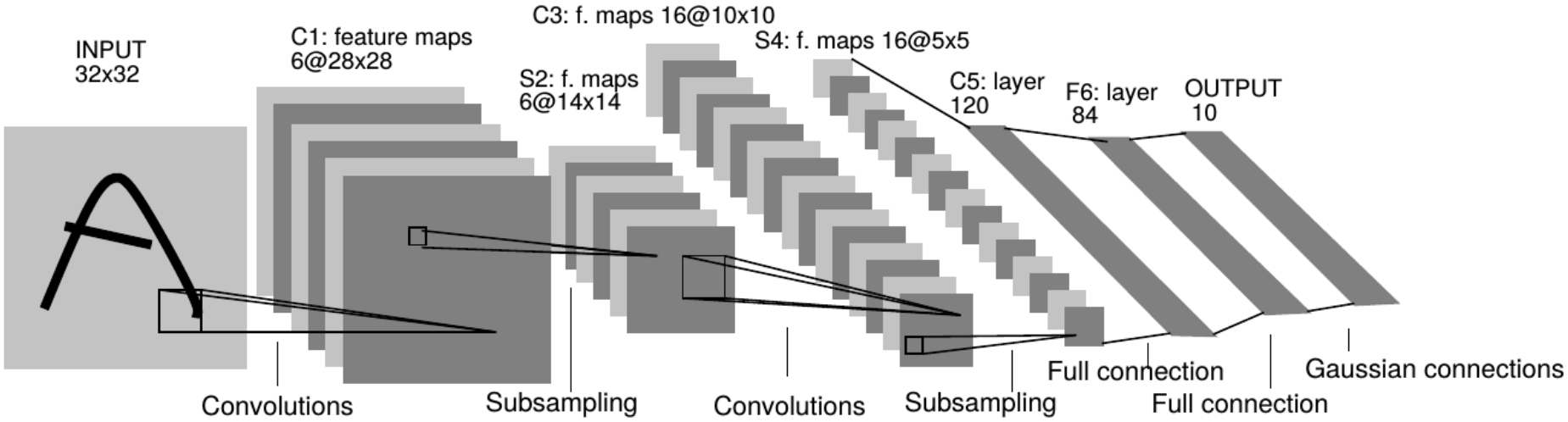
- Recognition of handwritten digits
  - Number parameters
    - Fully connected:11 935
    - CNN: 13 750





CNN: Example architectures

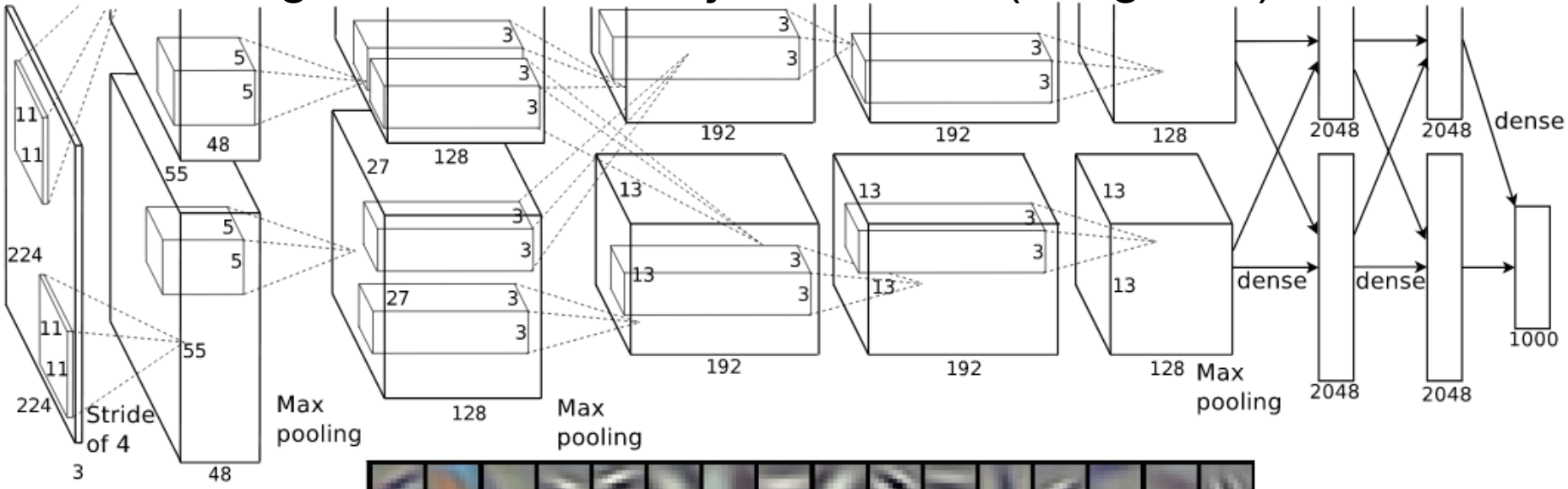
- LeNet: First CNN architectures
  - Recognition of handwritten characters

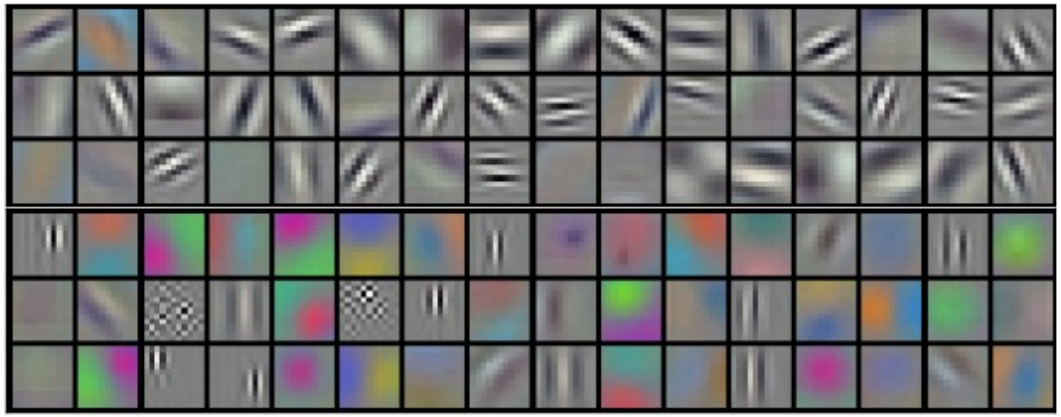


Y. LeCun, et al., Gradient-based learning applied to document recognition. Proceedings of the IEEE, november 1998

CNN: Example architectures

- AlexNet: Popularization of CNNs, first deep CNN
  - Recognition of 1000 object classes (ImageNet)



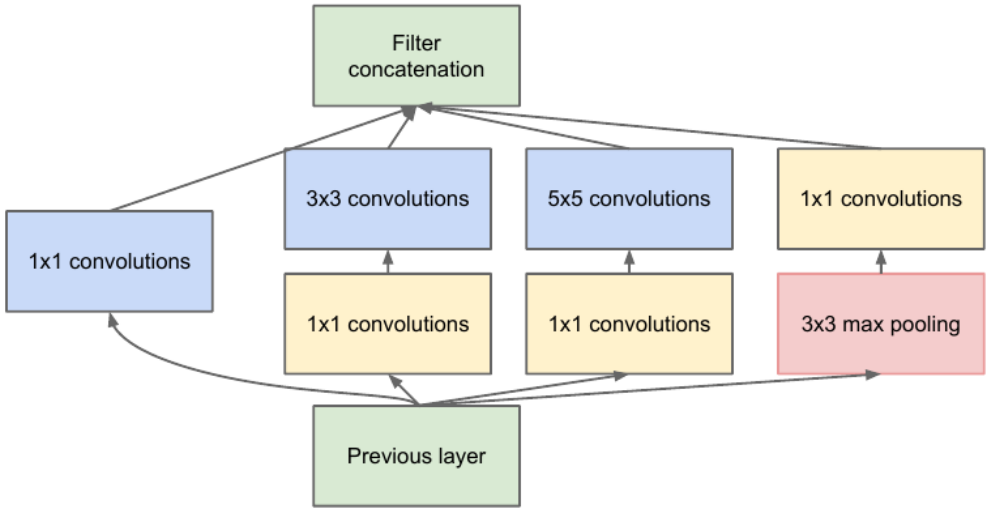


A. Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing System, 2012

NCT

CNN: Example architectures

- GoogleNet: Especially deep CNN
  - Recognition of 1000 object classes (ImageNet)
  - Only 4M parameters (Compare AlexNet: 60M)

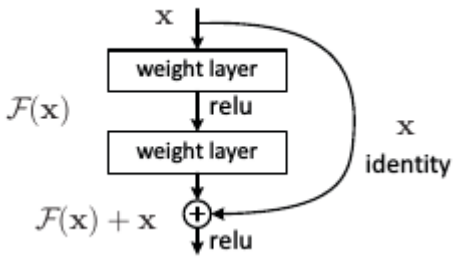


C. Szegedy, et al. "Going deeper with convolutions."
Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

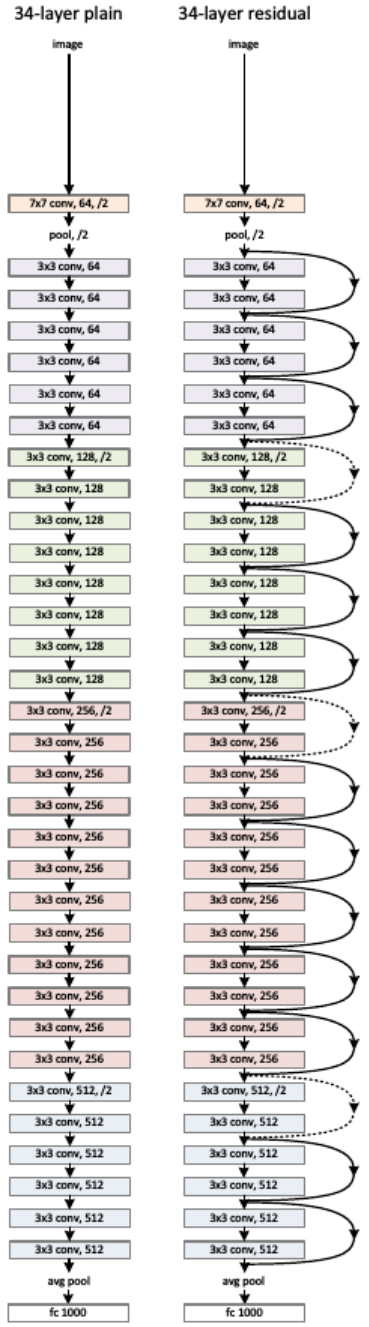
NCT

CNN: Example architectures

- ResNet (State of the Art): Even deeper CNNs
  - Recognition of 1000 object classes (ImageNet)
  - Residual Units
    - Skip Chains



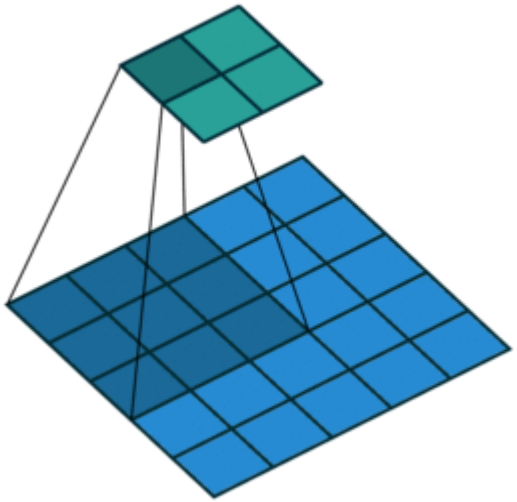
  - Allows deep networks with few parameters
    - ResNet32: 0.46M
    - ResNet44: 0.66M
    - ResNet110: 1.7M



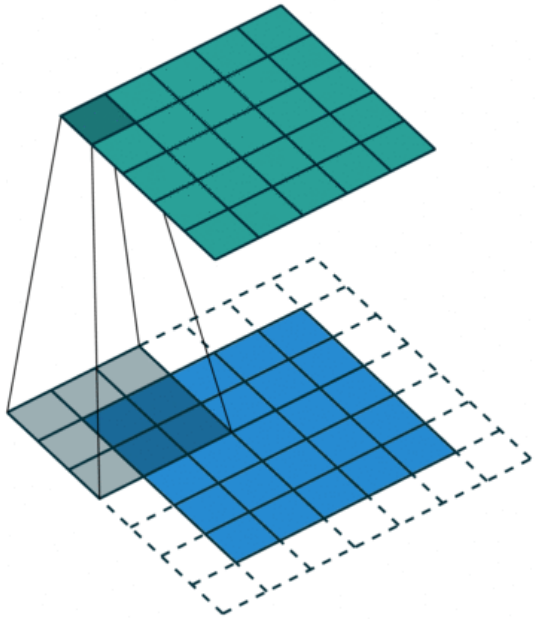
K. He, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

Segmentation with CNNs

- Up to now: Convolution



Convolution with 3x3 Kernel, step size 1, no padding



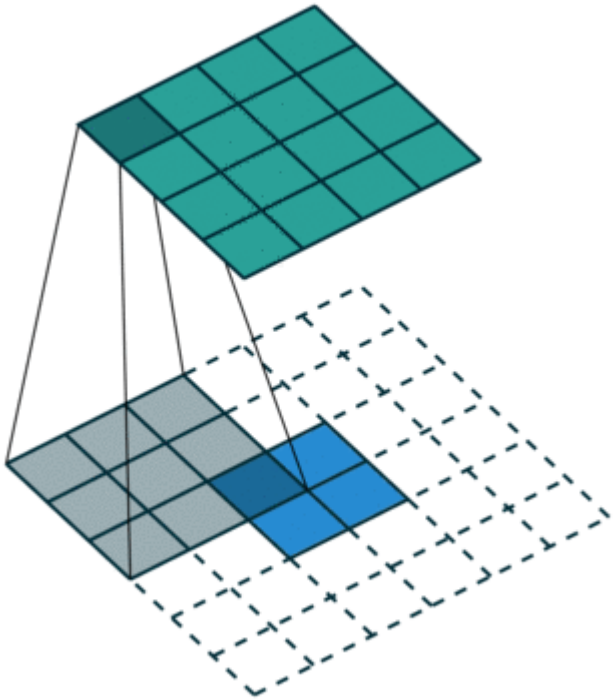
Convolution with 3x3 Kernel, step size 1, padding

- Resolution is reduced or stays the same
  - Necessary for efficient computation
  - For segmentation output has to have the same size as input

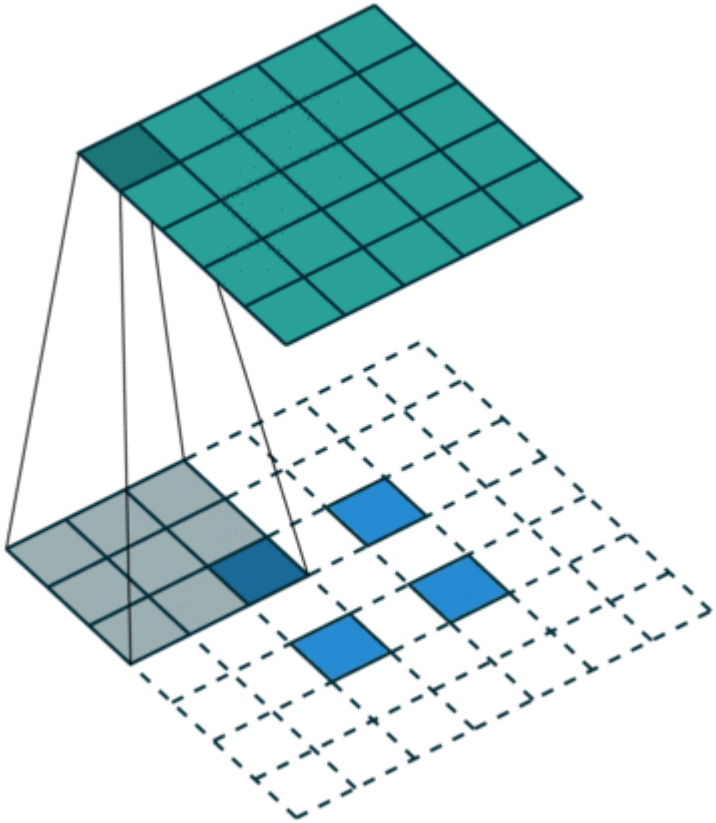
NCT

Segmentation with CNNs

- Transposed Convolution



Convolution with 3x3 Kernel, step size 1, padding of 2

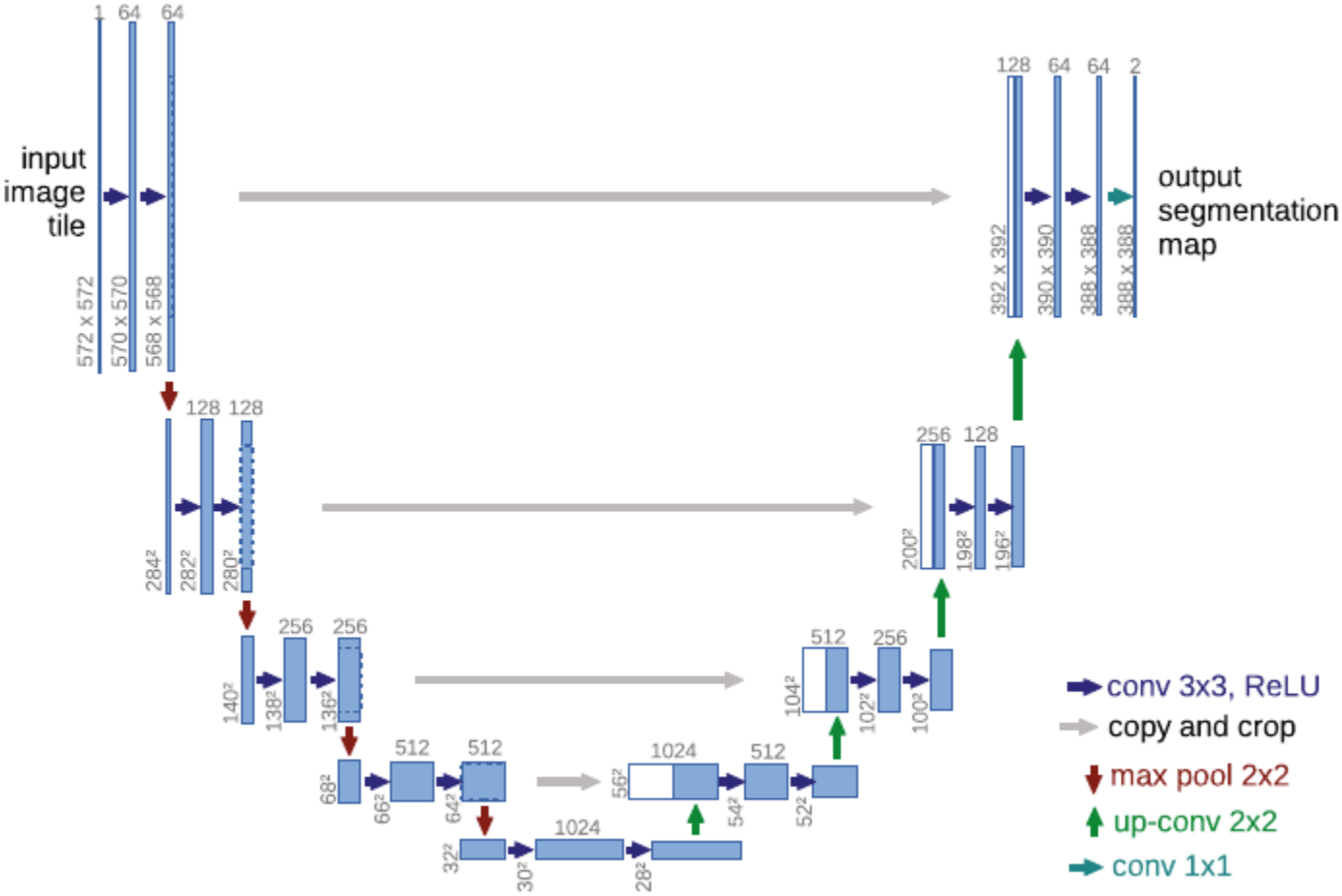


Convolution with 3x3 Kernel, step size 2, Padding

- Can be used to increase resolution

Segmentation with CNNs

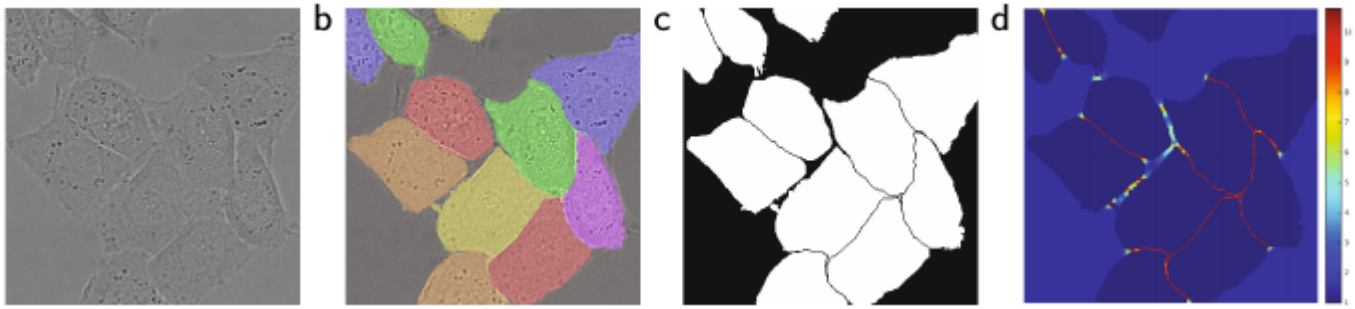
- UNet: Pure CNN for segmentation

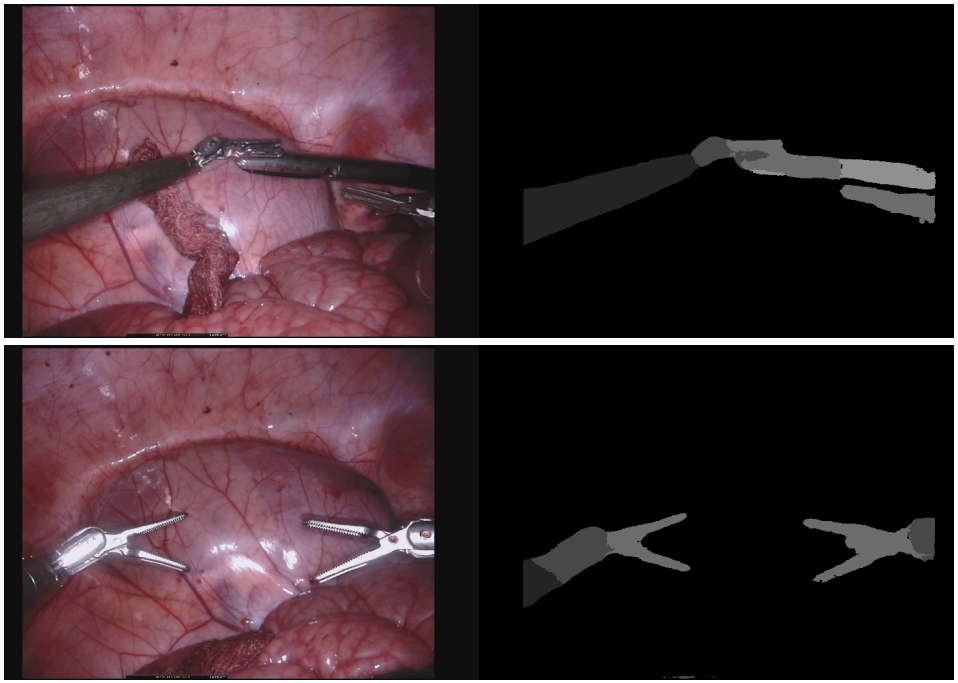


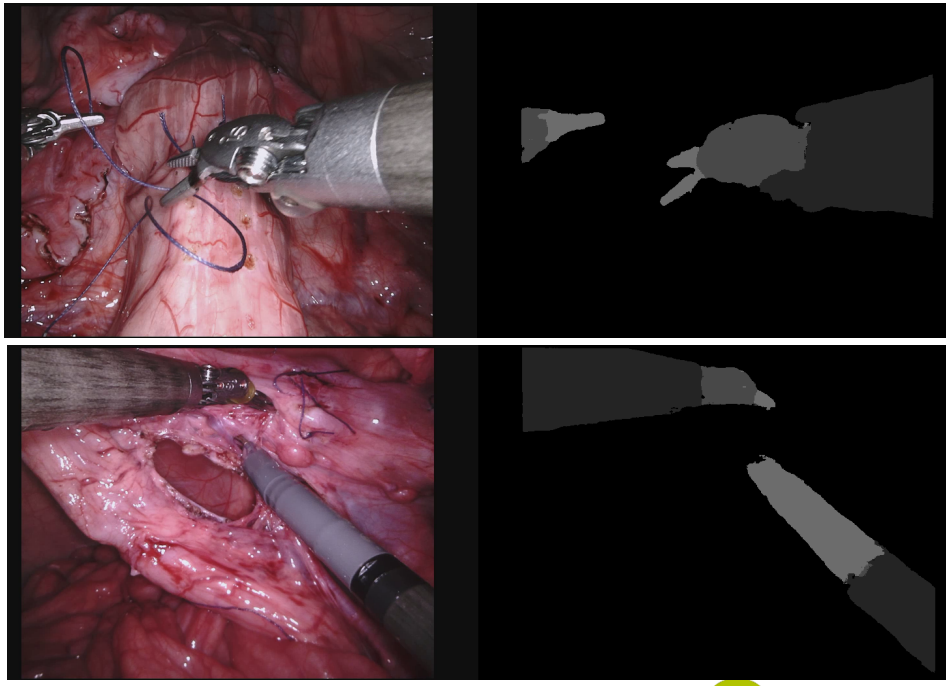
O. Ronneberger, et al., „U-Net: Convolutional Networks for Biomedical Image Segmentation“ Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015

Segmentation with CNNs

- UNet: Pure CNN for segmentation

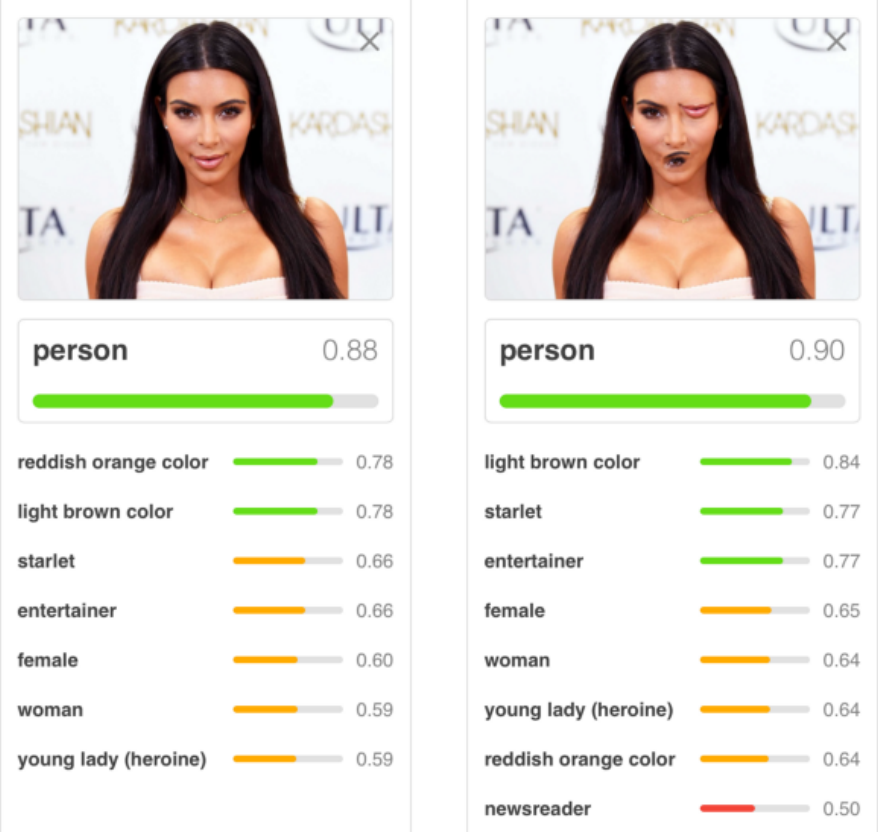






Problem CNNs/Outlook

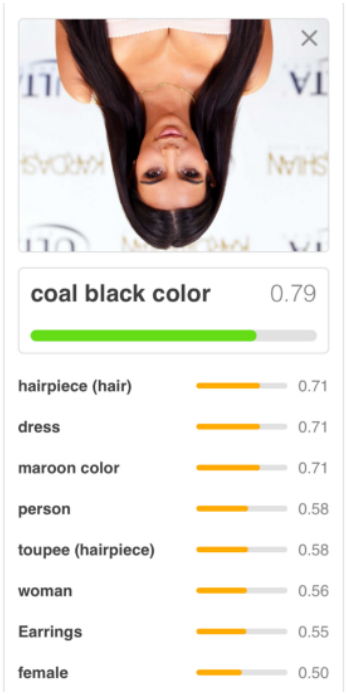
- CNNs produce State of the Art results for many problems
- Few drawbacks
  - Geometric relations of the features is not explicitly viewed
  - E.g.: A face consists of 2 eyes, a nose and a mouth



NCT

Problem CNNs/Outlook

- CNNs produce State of the Art results for many problems
- Few drawbacks
  - Geometric relations of the features is not explicitly viewed
  - Not invariant to rotations



Problem CNNs/Outlook

- CNNs produce State of the Art results for many problems
- Few drawbacks
  - Geometric relations of the features is not explicitly viewed
  - Not invariant to rotations
- Capsule Network
  - Extension of CNNs
    - Saves geometric relations of features
    - Detection via voting
    - E.g.: A face consists of 2 eyes that lie besides each other, a nose below the eyes and a mouth below the nose

NCT

Summary

Deep Learning: How can images be segmented and classified using artificial neural networks?

- Perceptron
- Neural Networks
  - Training via Backpropagation
- Convolutional Neural Networks
  - Image classification (what can be seen in an image)
  - Image segmentation (where can something be seen in an image)

NCT

Tutorials/Sources

- Great tutorials:
  - http://neuralnetworksanddeeplearning.com (very well written)
  - http://cs231n.github.io/convolutional-networks/

Are there any questions?