

# Algorithmik zur Optimierung in neuronalen Netzwerken

## Gradient Descent und Backpropagation

---

Tim Hilt

Date: tbd

Hochschule Esslingen — University of Applied Sciences

Supervised Learning

Künstliche Neuronale Netze

Training

- Loss-Funktion

- Gradient Descent

- Backpropagation

# Supervised Learning

---

# Machine Learning Workflow

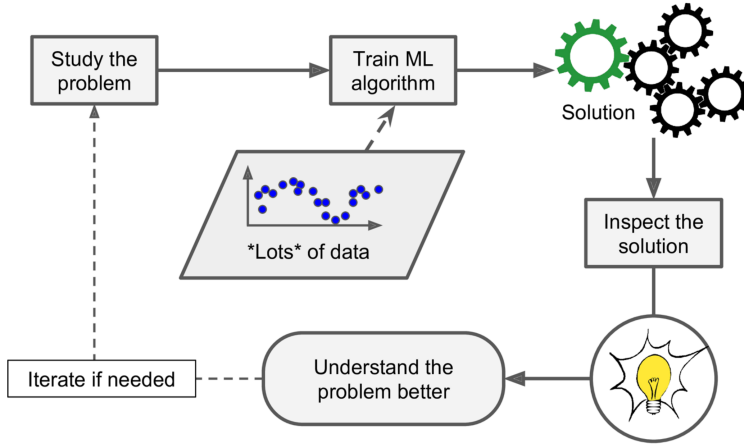


Abbildung 1: Machine Learning Workflow [1]

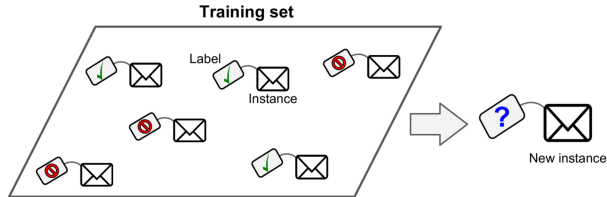


Abbildung 2: Struktur der Daten bei Supervised Learning [1]

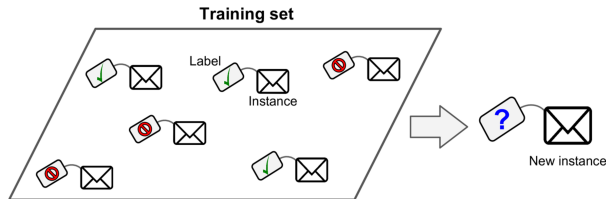
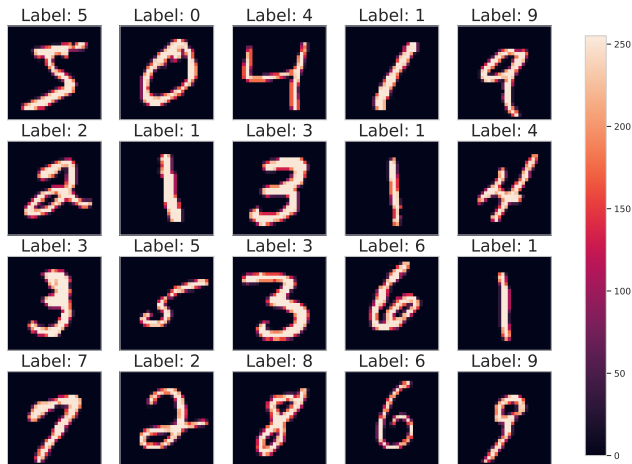


Abbildung 2: Struktur der Daten bei Supervised Learning [1]

## Definition Supervised Learning

„In supervised learning, the dataset is the collection of labeled examples  $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ . Each element  $\mathbf{x}_i$  among  $N$  is called a feature vector. A feature vector is a vector in which each dimension  $j = 1, \dots, D$  contains a value that describes the example somehow [...]. The goal of a supervised learning algorithm is to use the dataset to produce a model, that takes a feature vector  $\mathbf{x}$  as input and outputs information that allow deducing the label  $\hat{y}$  for this feature vector.“ [2]

# Beispiel: Datensatz für Supervised Learning



- Insgesamt 70000 Bilder
- Bildgröße:  $28 \times 28$  Pixel
- Abgebildet: Handgeschriebene Ziffern von 0 bis 9
- Quelle: Yann LeCun et al [3]

# Beispiel: Datensatz für Supervised Learning



- Insgesamt 70000 Bilder
- Bildgröße:  $28 \times 28$  Pixel
- Abgebildet: Kleidungsstücke
- Quelle: Zalando Research [4]

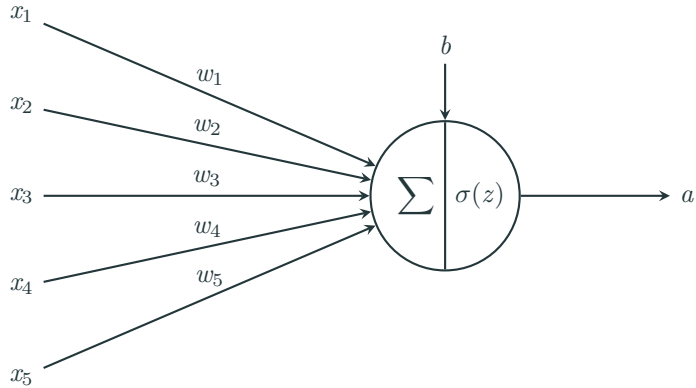
Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot



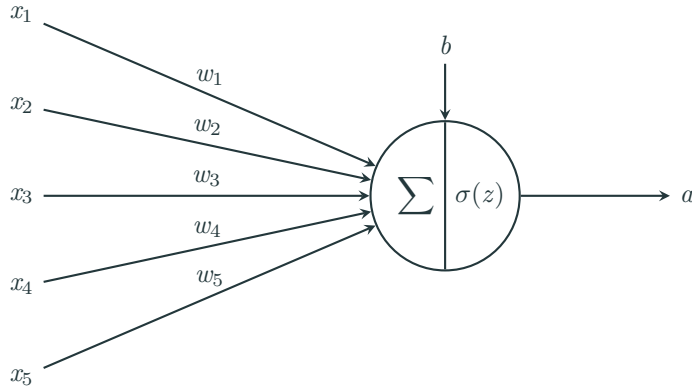
# Künstliche Neuronale Netze

---

# Künstliches Neuron



# Künstliches Neuron

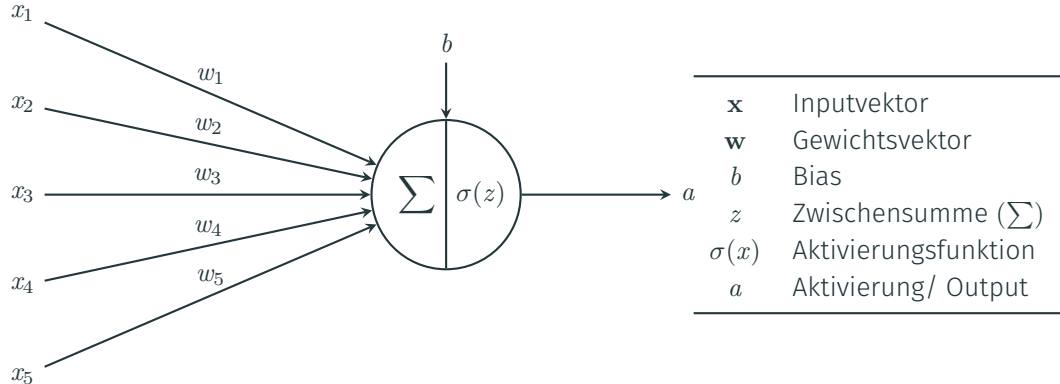


---

$\mathbf{x}$	Inputvektor
$\mathbf{w}$	Gewichtsvektor
$b$	Bias
$z$	Zwischensumme ( $\Sigma$ )
$\sigma(x)$	Aktivierungsfunktion
$a$	Aktivierung/ Output

---

# Künstliches Neuron



$$z = \sum_i x_i w_i + b = \mathbf{xw} + b$$

$\Rightarrow z$  wird für spätere Parameteroptimierung benötigt

⇒ Es gibt eine Vielzahl verschiedener Aktivierungsfunktionen für unterschiedliche Problemstellungen, für uns soll jedoch lediglich die **Sigmoid-Funktion** relevant sein:

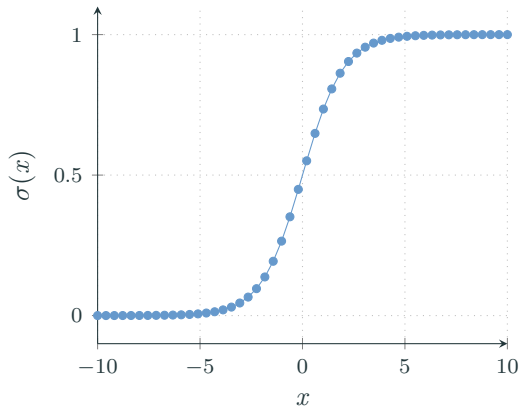
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

# Aktivierungsfunktion $\sigma(x)$

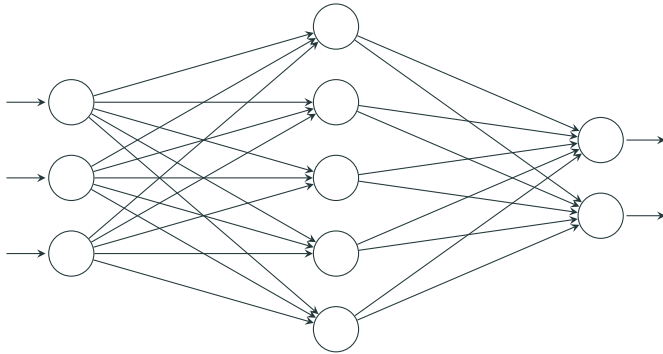
⇒ Es gibt eine Vielzahl verschiedener Aktivierungsfunktionen für unterschiedliche Problemstellungen, für uns soll jedoch lediglich die **Sigmoid-Funktion** relevant sein:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

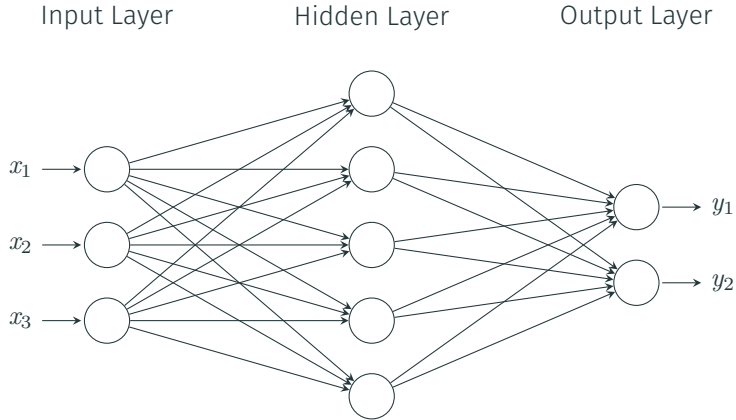
Sigmoid-Funktion



# Architektur eines Neuronalen Netzwerks

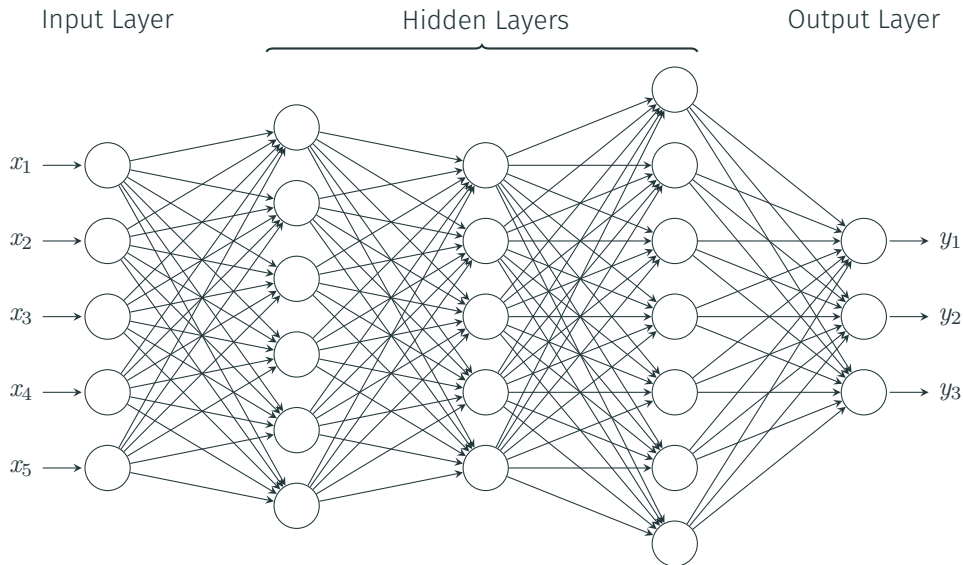


# Architektur eines Neuronalen Netzwerks

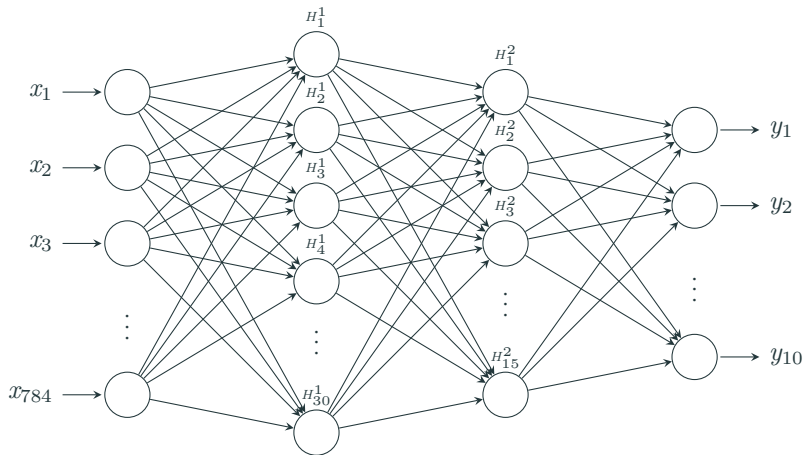




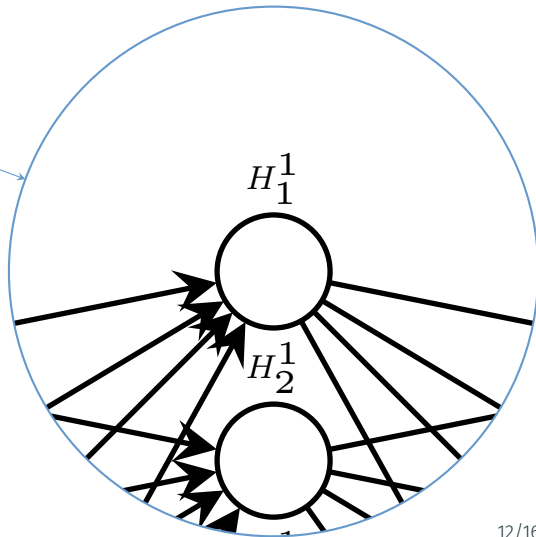
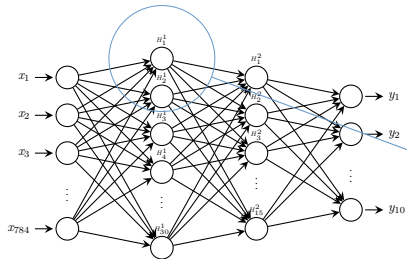
# Deep Neural Network



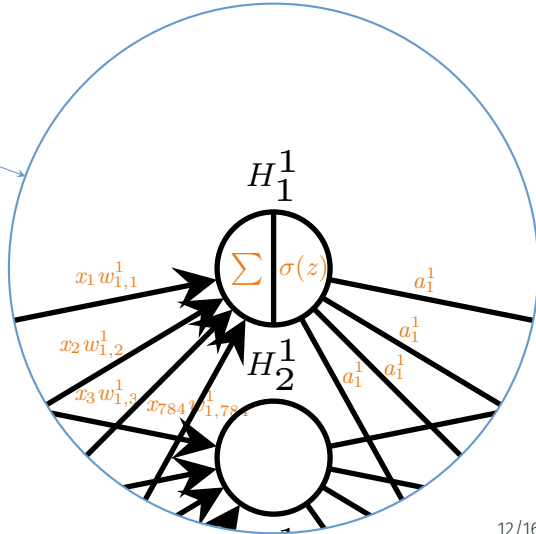
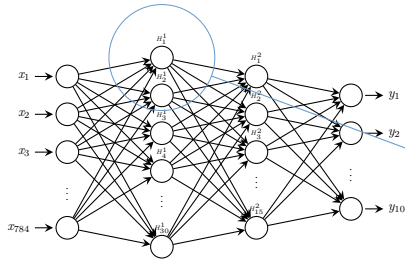
# Target-Architektur zur Klassifikation von MNIST



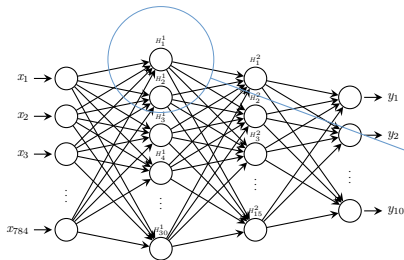
# Vektorisierung der Gewichte $w$ und der Biases $b$



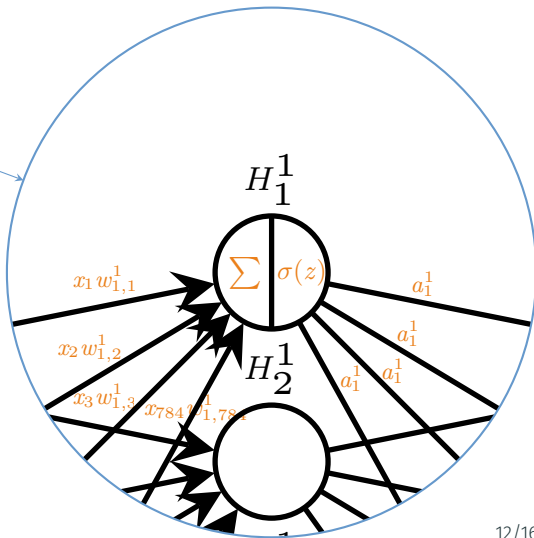
# Vektorisierung der Gewichte $w$ und der Biases $b$



# Vektorisierung der Gewichte $w$ und der Biases $b$



$$\mathbf{W}^1 = \begin{bmatrix} w_{1,1}^1 & w_{2,1}^1 & \cdots & w_{30,1}^1 \\ w_{1,2}^1 & w_{2,2}^1 & \cdots & w_{30,2}^1 \\ \vdots & \vdots & \cdots & \vdots \\ w_{1,784}^1 & w_{2,784}^1 & \cdots & w_{30,784}^1 \end{bmatrix}; \mathbf{b}^1 = \begin{bmatrix} b_1^1 \\ b_2^1 \\ \vdots \\ b_{30}^1 \end{bmatrix}$$



# Training

---

# Backpropagation

Es werden vier Gleichungen benötigt:

Error im Output-Layer:

Error einzelner Neuronen:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L)$$

Vektorisiert:

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

Aufgelöst, wenn MSE benutzt:

$$\delta^L = (a^L - y) \odot \sigma'(z^L)$$

Error im Layer  $l$  hinsichtlich Error im nächsten Layer  $\delta^{l+1}$

$$\delta^l = \left( (w^{l+1})^T \delta^{l+1} \right) \odot \sigma' (z^l)$$

- Rekursive Definition durch Verwendung von  $\delta^l$  in Abhängigkeit von  $\delta^{l+1}$
- Wenn anfangs  $\delta^L$  in die Gleichung gegeben wird kann der Error rekursiv für jeden vorhergehenden Layer berechnet werden



$$\delta^L = \nabla_a C \odot \sigma' (z^L)$$

$$\delta^l = \left( (w^{l+1})^T \delta^{l+1} \right) \odot \sigma' (z^l)$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$





$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

Pass

Fragen?

# Literaturverzeichnis

---

-  GÉRON, Aurélien. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.
-  BURKOV, Andriy. *The hundred-page machine learning book*. Andriy Burkov Quebec City, Can., 2019.
-  LECUN, Yann; BOTTOU, Léon; BENGIO, Yoshua; HAFFNER, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*. 1998, Jg. 86, Nr. 11, S. 2278–2324.
-  XIAO, Han; RASUL, Kashif; VOLLGRAF, Roland. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*. 2017.