

Algorithmik zur Optimierung in neuronalen Netzwerken

Gradient Descent und Backpropagation

Tim Hilt

19. Mai 2020

Hochschule Esslingen — University of Applied Sciences

- Supervised Learning

- Künstliche Neuronale Netze

 - Künstliches Neuron

 - Architektur eines Neuronalen Netzes

- Training

 - Gradient Descent

 - Backpropagation

- Umsetzung in Python (mit Keras)

Supervised Learning

Machine Learning Workflow

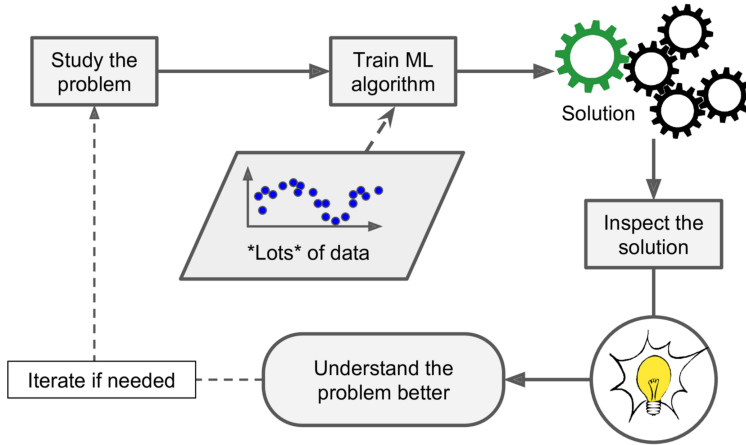


Abbildung 1: Machine Learning Workflow [1]

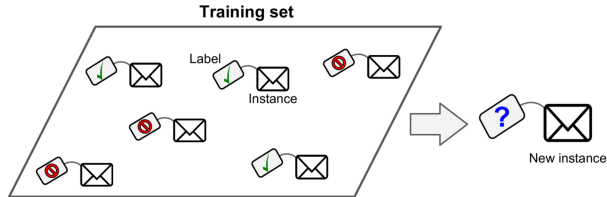


Abbildung 2: Struktur der Daten bei Supervised Learning [1]

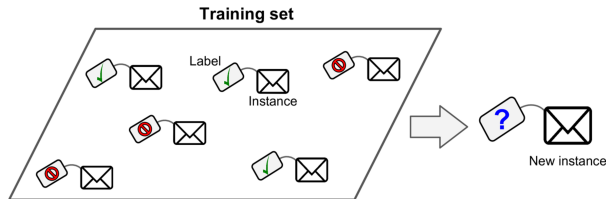
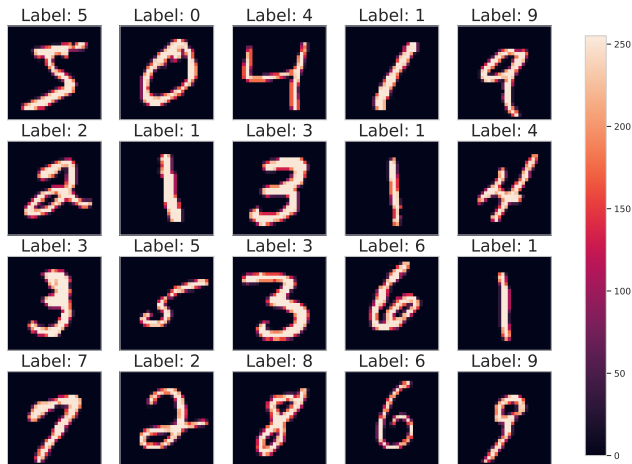


Abbildung 2: Struktur der Daten bei Supervised Learning [1]

Definition Supervised Learning

Bei Supervised Learning ist jeweils ein Datensatz gegeben, der *gelabelte* Beispiele enthält. Dabei wird das i -te Beispiel jeweils mit einem Vektor \mathbf{x}_i und das Label mit y_i benannt. Die Aufgabe des lernenden Algorithmus ist es, aufgrund der Beispielmatrix \mathbf{X} auf die Beispiellabel \mathbf{y} zu schließen. Hierzu wird ein sog. *Modell* trainiert, welches angewendet auf bisher unbekannte Daten $\mathbf{x}_{\text{unbekannt}}$ passende Werte $y_{\text{unbekannt}}$ vorhersagen kann. [2]

Beispiel: Datensatz für Supervised Learning



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

Beispiel: Datensatz für Supervised Learning

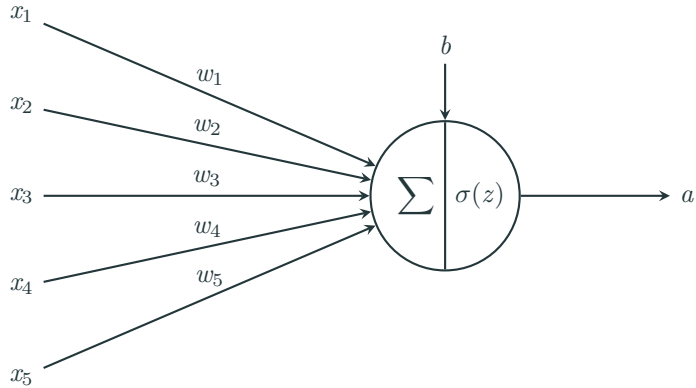


- Insgesamt 70000 Bilder
- Bildgröße: 28×28 Pixel
- Abgebildet: Kleidungsstücke
- Quelle: Zalando Research [4]

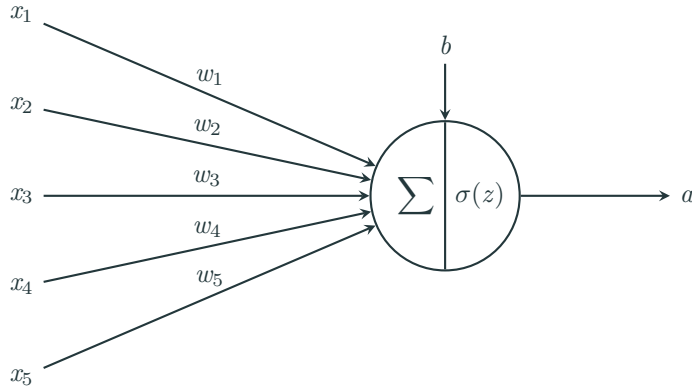
Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Künstliche Neuronale Netze

Künstliches Neuron

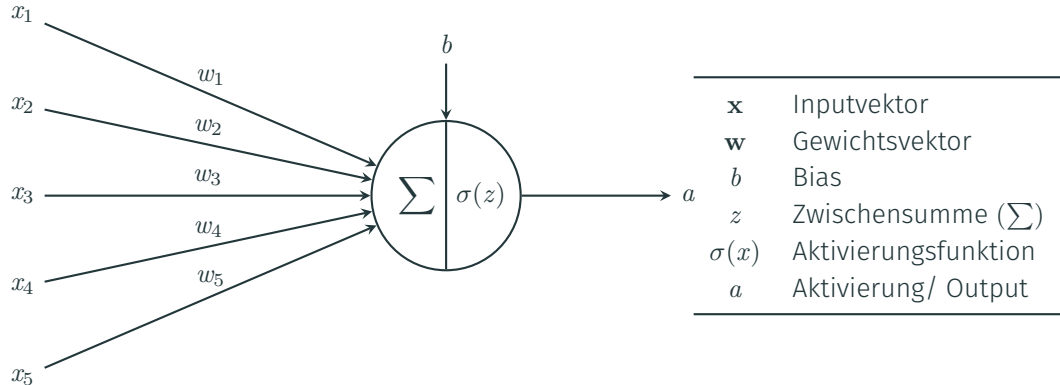


Künstliches Neuron



\mathbf{x}	Inputvektor
\mathbf{w}	Gewichtsvektor
b	Bias
z	Zwischensumme (Σ)
$\sigma(x)$	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:

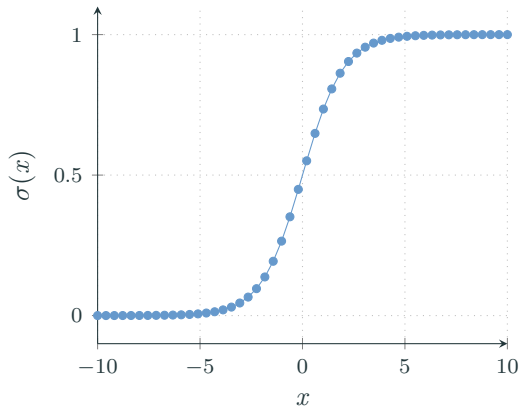
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Aktivierungsfunktion $\sigma(x)$

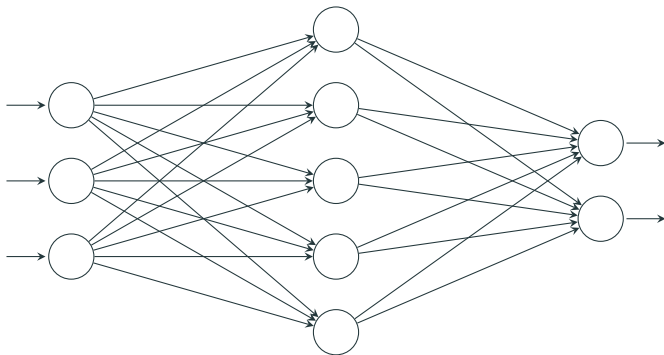
⇒ Es gibt eine Vielzahl verschiedener Aktivierungsfunktionen für unterschiedliche Problemstellungen, für uns soll jedoch lediglich die **Sigmoid-Funktion** relevant sein:

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

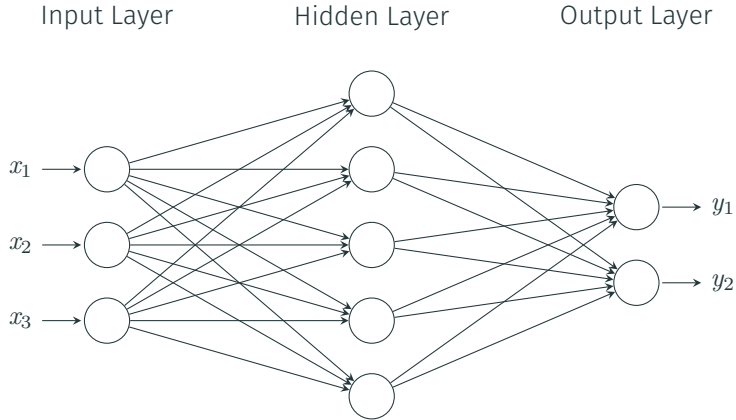
Sigmoid-Funktion



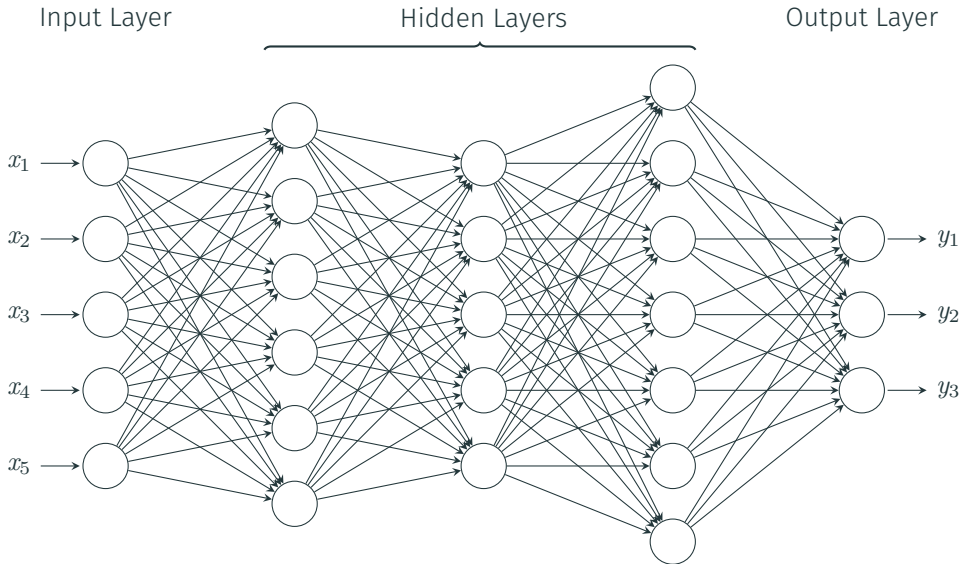
Architektur eines Neuronalen Netzwerks



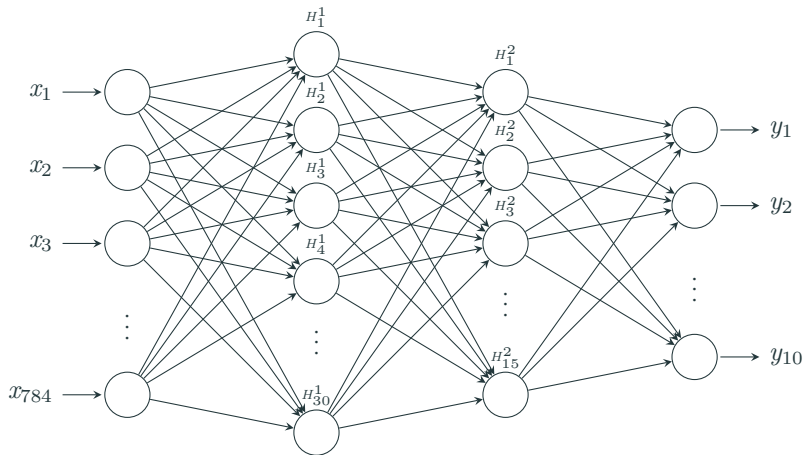
Architektur eines Neuronalen Netzwerks



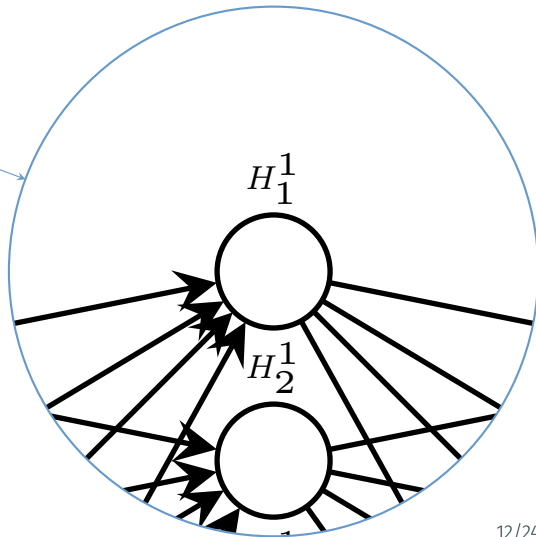
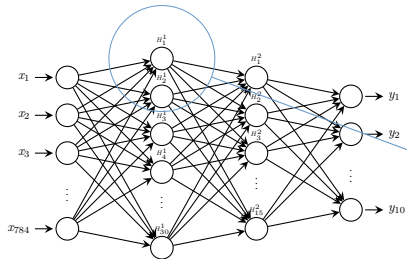
Deep Neural Network



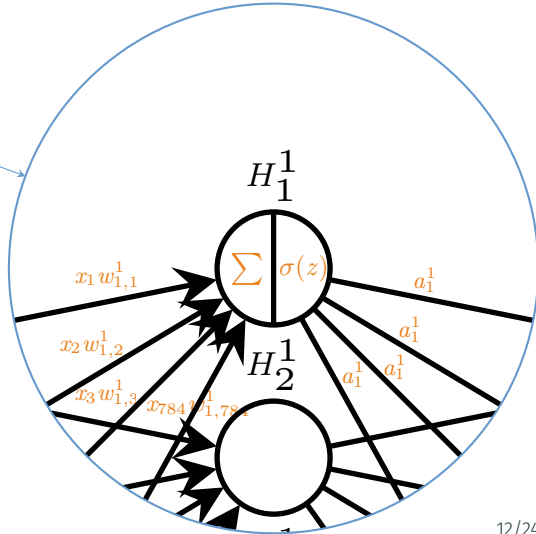
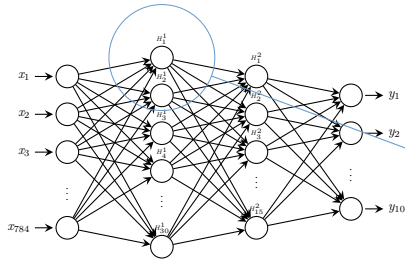
Target-Architektur zur Klassifikation von MNIST



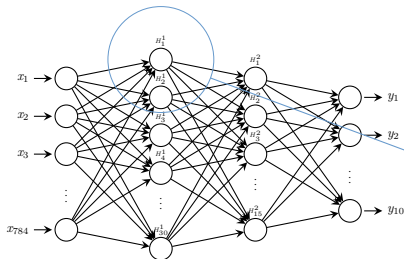
Vektorisierung der Gewichte w und der Biases b



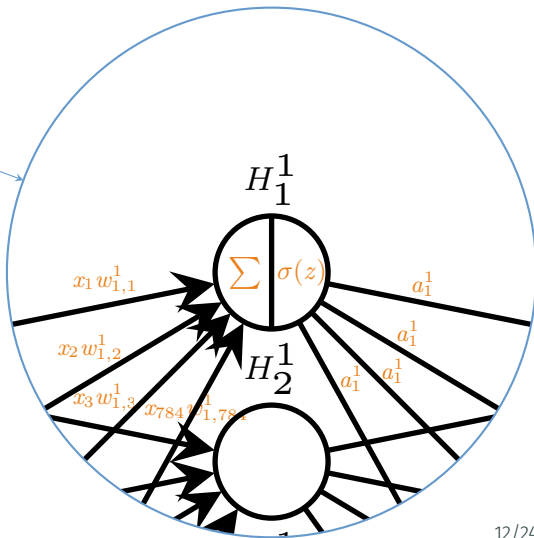
Vektorisierung der Gewichte w und der Biases b



Vektorisierung der Gewichte w und der Biases b



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



Output eines neuronalen Netzwerks berechnen

Gegeben:

- Inputvektor \mathbf{x}
- Gewichtsmatrizen $\mathbf{W}^{2...4}$
- Biasvektoren $\mathbf{b}^{2...4}$

$$\mathbf{a}^1 = \mathbf{x}$$

$$\mathbf{a}^2 = \sigma(\mathbf{W}^2 \mathbf{a}^1 + \mathbf{b}^2)$$

$$\mathbf{a}^3 = \sigma(\mathbf{W}^3 \mathbf{a}^2 + \mathbf{b}^3)$$

$$\mathbf{a}^4 = \sigma(\mathbf{W}^4 \mathbf{a}^3 + \mathbf{b}^4) = \hat{\mathbf{y}}$$

Training

- Dient zur Berechnung des Fehlers während dem Training
- Trainingsfehler soll minimiert werden
- \Rightarrow wir suchen den Punkt, an dem die Ableitung der Loss-Funktion 0 wird, der Fehler also nicht mehr abnimmt
- Es gibt eine Vielzahl an Loss-Funktionen, wir betrachten hier die „**Mean Squared Error (MSE)**“:

$$C(w, b) = \frac{1}{2m} \sum_{x=1}^m (y(x) - \hat{y}(x))^2$$

Loss-Funktion

- Dient zur Berechnung des Fehlers während dem Training
- Trainingsfehler soll minimiert werden
- \Rightarrow wir suchen den Punkt, an dem die Ableitung der Loss-Funktion 0 wird, der Fehler also nicht mehr abnimmt
- Es gibt eine Vielzahl an Loss-Funktionen, wir betrachten hier die „**Mean Squared Error (MSE)**“:

$$C(w, b) = \frac{1}{2m} \sum_{x=1}^m (y(x) - \hat{y}(x))^2$$

$C(w, b)$	Cost in Abhängigkeit von w und b
m	Anzahl der Trainingsinstanzen
$y(x)$	Gewünschter Output wenn x Input ist
$\hat{y}(x)$	Tatsächlicher Output des Netzwerkes

Gradient Descent

- Methode um die Weights w und Biases b zu optimieren
- Vorgehen:

Gradient Descent

- Methode um die Weights w und Biases b zu optimieren
- Vorgehen:
 1. Finde die Änderungsrate des Fehlers in Abhängigkeit von den Weights und Biases
($\partial C / \partial w$; $\partial C / \partial b$)

Gradient Descent

- Methode um die Weights w und Biases b zu optimieren
- Vorgehen:
 1. Finde die Änderungsrate des Fehlers in Abhängigkeit von den Weights und Biases ($\partial C/\partial w; \partial C/\partial b$)
 2. Multipliziere die Änderungsrate mit der Lernrate η

Gradient Descent

- Methode um die Weights w und Biases b zu optimieren
- Vorgehen:
 1. Finde die Änderungsrate des Fehlers in Abhängigkeit von den Weights und Biases ($\partial C/\partial w; \partial C/\partial b$)
 2. Multipliziere die Änderungsrate mit der Lernrate η
 3. Ziehe das Produkt aus Änderungsrate und Lernrate von den aktuellen Parametern ab

Gradient Descent

- Methode um die Weights w und Biases b zu optimieren
- Vorgehen:
 1. Finde die Änderungsrate des Fehlers in Abhängigkeit von den Weights und Biases ($\partial C/\partial w; \partial C/\partial b$)
 2. Multipliziere die Änderungsrate mit der Lernrate η
 3. Ziehe das Produkt aus Änderungsrate und Lernrate von den aktuellen Parametern ab
 4. Aktualisiere die alten Parameter durch das Ergebnis des letzten Schrittes

$$w_{k+1} = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_{k+1} = b_k - \eta \frac{\partial C}{\partial b_k}$$

Problem

Wie finde ich die Änderungsraten $\frac{\partial C}{\partial w}$; $\frac{\partial C}{\partial b}$, die ich für Gradient Descent benötige?

Problem

Wie finde ich die Änderungsraten $\frac{\partial C}{\partial w}$; $\frac{\partial C}{\partial b}$, die ich für Gradient Descent benötige?

Beispiel:

Gegeben:

$$f(a, b, c, d) = ((a + b) \cdot (c + d))^2$$

Gesucht:

$$\frac{\partial f}{\partial a}$$

Backpropagation — Einleitung

Beispiel:

Gegeben:

$$f(a, b, c, d) = ((a + b) \cdot (c + d))^2$$

Gesucht:

$$\frac{\partial f}{\partial a}$$

Analytische Lösung

Umformung zu Summanden:

$$f = (a + b)^2 \cdot (c + d)^2$$

$$f = (a^2 + 2ab + b^2) \cdot (c + d)^2$$

Ableitung:

$$\frac{\partial f}{\partial a} = (2a + 2b) \cdot (c + d)^2$$

- **Problem:**
 - Analytische Verfahren kommen bei komplizierten Funktionen an ihre Grenzen
 - Sind schwer zu implementieren
 - Sind ineffizient

Backpropagation — Idee

- **Problem:**
 - Analytische Verfahren kommen bei komplizierten Funktionen an ihre Grenzen
 - Sind schwer zu implementieren
 - Sind ineffizient
- **Idee:** Divide and conquer; Problem in kleinere, handhabbare Probleme zerlegen

- **Problem:**
 - Analytische Verfahren kommen bei komplizierten Funktionen an ihre Grenzen
 - Sind schwer zu implementieren
 - Sind ineffizient
- **Idee:** Divide and conquer; Problem in kleinere, handhabbare Probleme zerlegen

$$f(a, b, c, d) = ((a + b) \cdot (c + d))^2$$

- **Problem:**
 - Analytische Verfahren kommen bei komplizierten Funktionen an ihre Grenzen
 - Sind schwer zu implementieren
 - Sind ineffizient
- **Idee:** Divide and conquer; Problem in kleinere, handhabbare Probleme zerlegen

$$f(a, b, c, d) = ((a + b) \cdot (c + d))^2$$

$$g = a + b$$

Backpropagation — Idee

- **Problem:**
 - Analytische Verfahren kommen bei komplizierten Funktionen an ihre Grenzen
 - Sind schwer zu implementieren
 - Sind ineffizient
- **Idee:** Divide and conquer; Problem in kleinere, handhabbare Probleme zerlegen

$$f(a, b, c, d) = ((a + b) \cdot (c + d))^2$$

$$g = a + b$$

$$h = c + d$$

Backpropagation — Idee

- **Problem:**
 - Analytische Verfahren kommen bei komplizierten Funktionen an ihre Grenzen
 - Sind schwer zu implementieren
 - Sind ineffizient
- **Idee:** Divide and conquer; Problem in kleinere, handhabbare Probleme zerlegen

$$f(a, b, c, d) = ((a + b) \cdot (c + d))^2$$

$$g = a + b$$

$$h = c + d$$

$$i = g \cdot h$$

Backpropagation — Idee

- **Problem:**
 - Analytische Verfahren kommen bei komplizierten Funktionen an ihre Grenzen
 - Sind schwer zu implementieren
 - Sind ineffizient
- **Idee:** Divide and conquer; Problem in kleinere, handhabbare Probleme zerlegen

$$f(a, b, c, d) = ((a + b) \cdot (c + d))^2$$

$$g = a + b$$

$$h = c + d$$

$$i = g \cdot h$$

$$f = i^2$$

Backpropagation

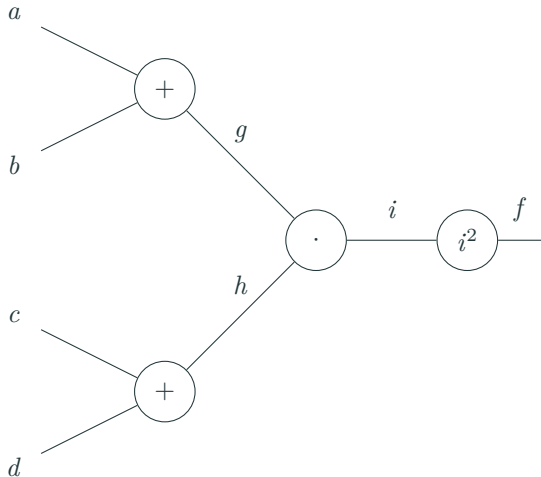
$$f(a, b, c, d) = ((a + b) \cdot (c + d))^2$$

$$g = a + b$$

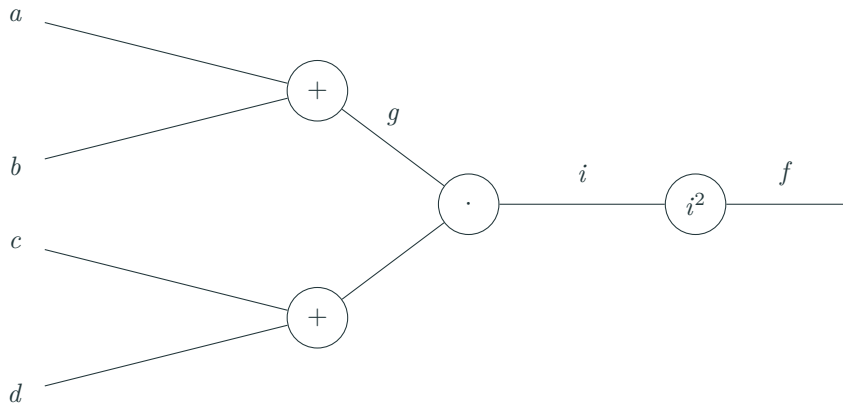
$$h = c + d$$

$$i = g \cdot h$$

$$f = i^2$$

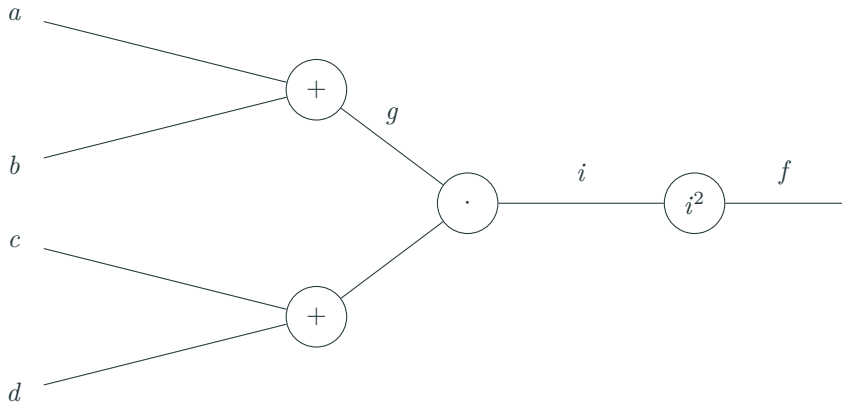


Backpropagation



Gesucht: $\frac{\partial f}{\partial a}$

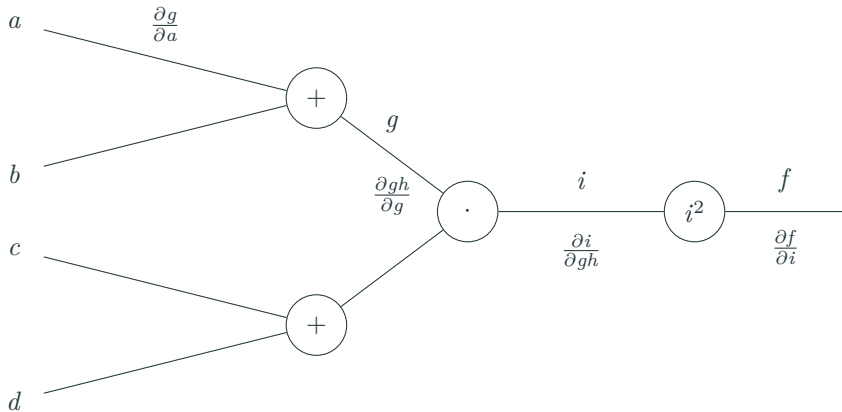
Backpropagation



Gesucht: $\frac{\partial f}{\partial a}$

$$\frac{\partial f}{\partial a} = \frac{\partial f}{\partial i} \cdot \frac{\partial i}{\partial g} \cdot \frac{\partial g}{\partial a}$$

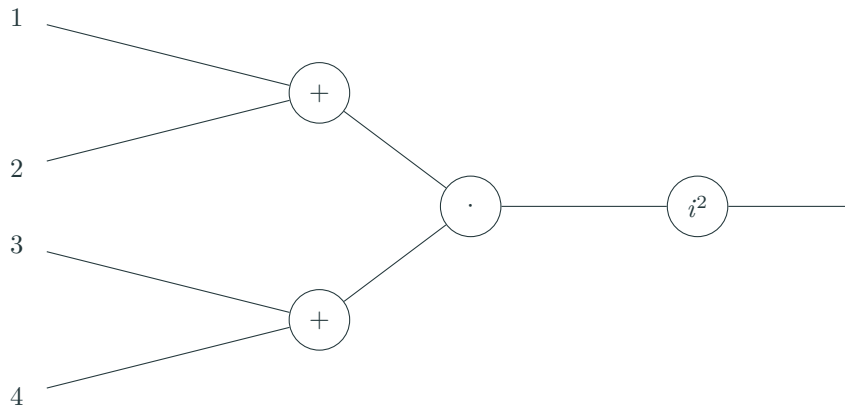
Backpropagation



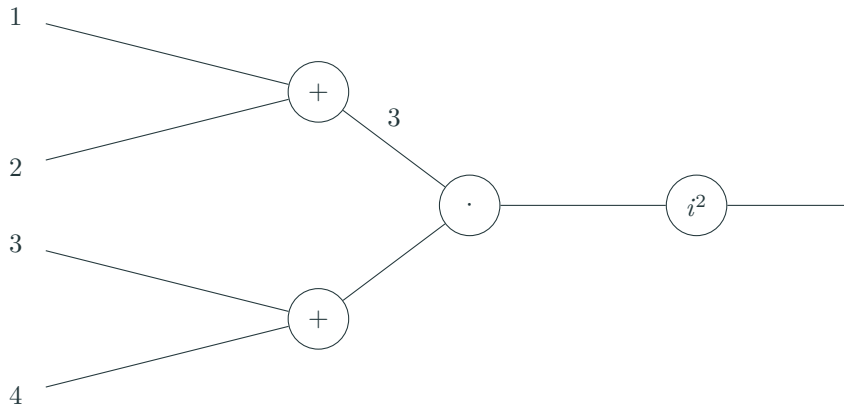
Gesucht: $\frac{\partial f}{\partial a}$

$$\frac{\partial f}{\partial a} = \frac{\partial f}{\partial i} \cdot \frac{\partial i}{\partial gh} \cdot \frac{\partial gh}{\partial g} \cdot \frac{\partial g}{\partial a}$$

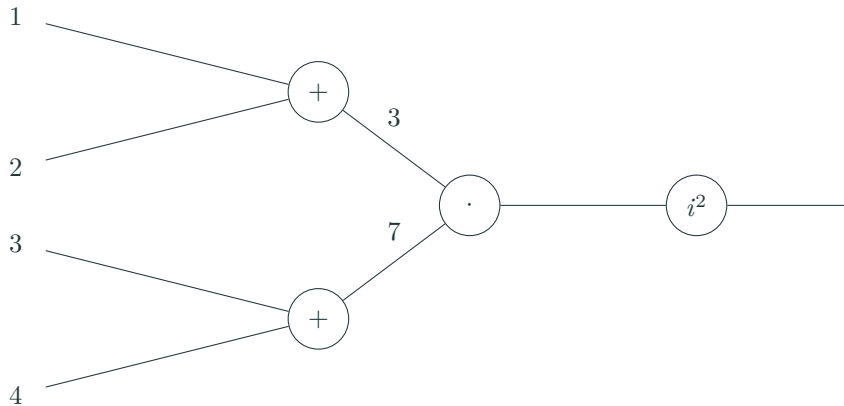
Beispiel



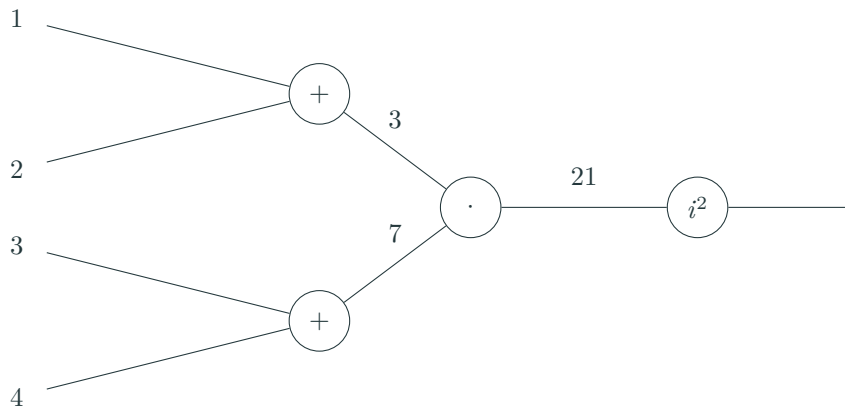
Beispiel



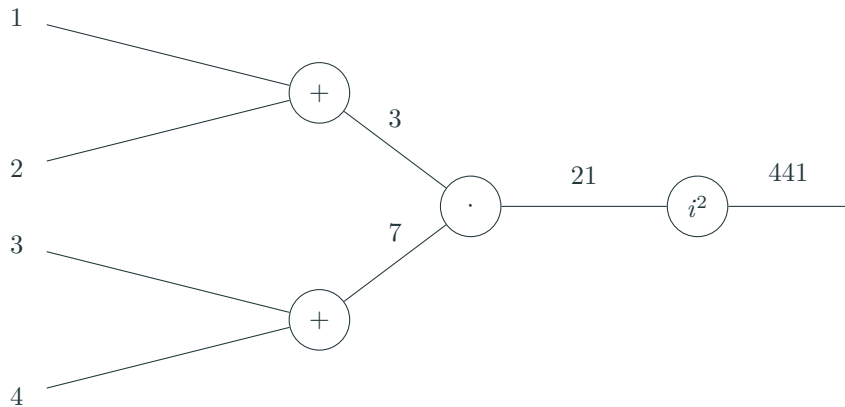
Beispiel



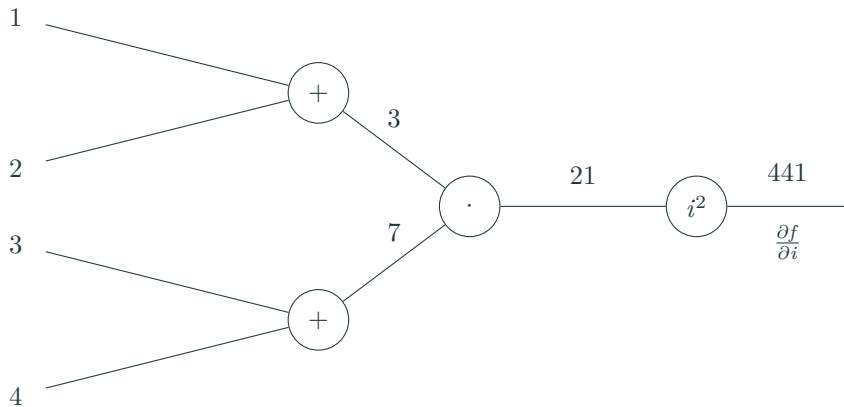
Beispiel



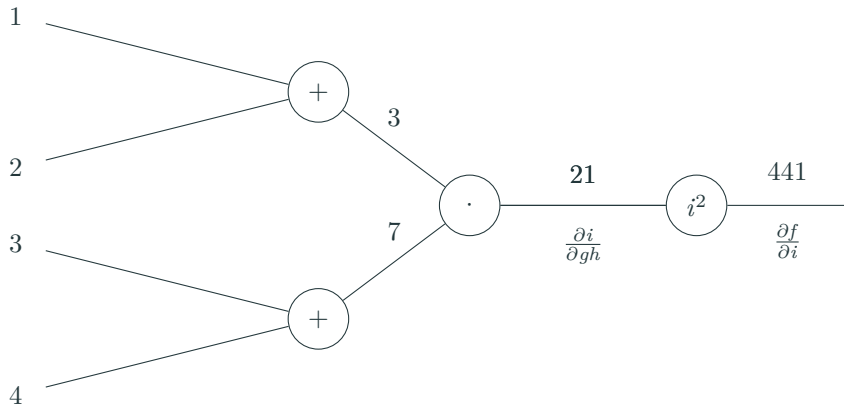
Beispiel



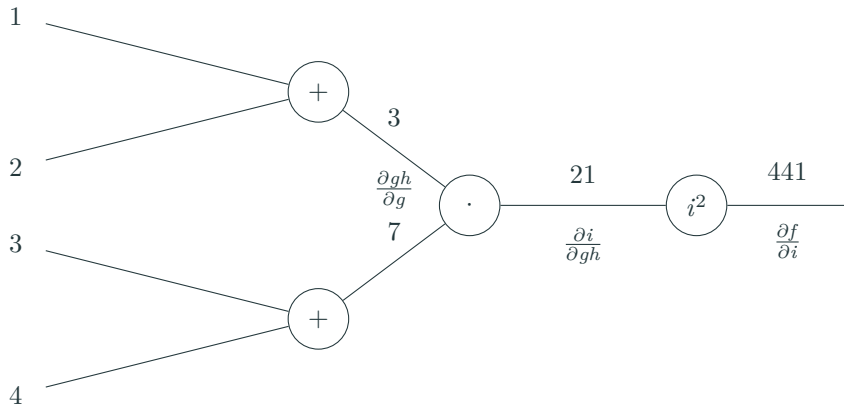
Beispiel



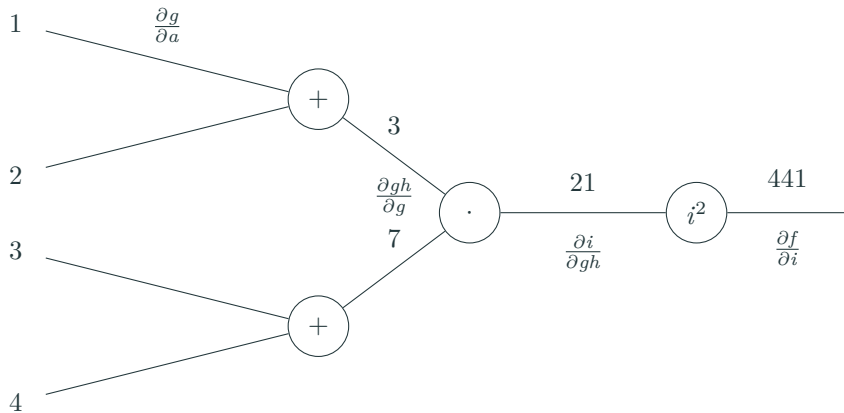
Beispiel



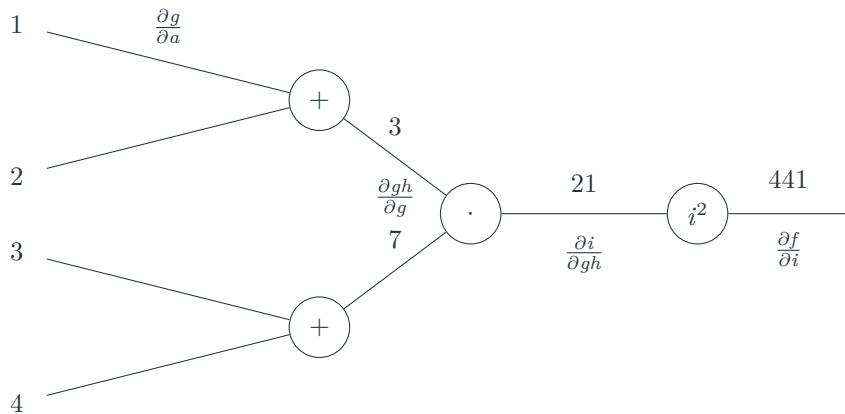
Beispiel



Beispiel



Beispiel



$$\frac{\partial f}{\partial i} = 2 \cdot 21 = 42; \quad \frac{\partial i}{\partial gh} = 1; \quad \frac{\partial gh}{\partial g} = h = 7; \quad \frac{\partial g}{\partial a} = 1 \quad \Rightarrow 7 \cdot 42 = 294$$

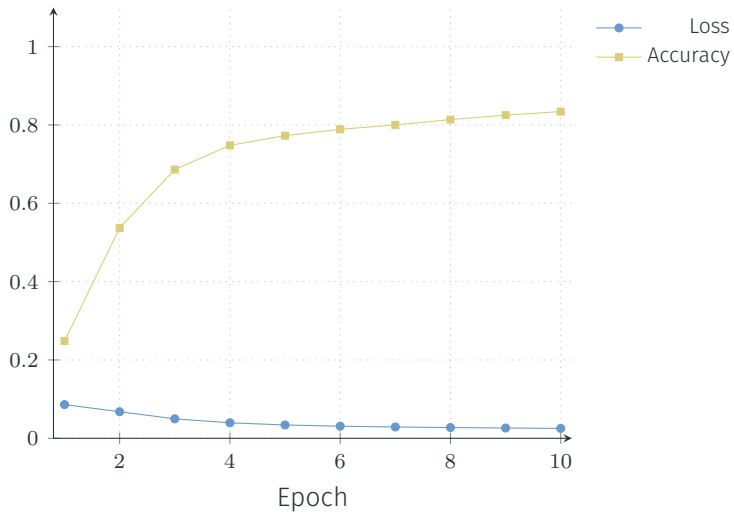
Umsetzung in Python (mit Keras)

```
from tensorflow import keras
...
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(30, activation='sigmoid'),
    keras.layers.Dense(15, activation='sigmoid'),
    keras.layers.Dense(10, activation='sigmoid'),
])

model.compile(loss='mse',
              optimizer=keras.optimizers.SGD(learning_rate=.8),
              metrics=['accuracy'])

history = model.fit(X_train, y_train_cat, epochs=10,
                   validation_data=(X_valid, y_valid_cat))
```

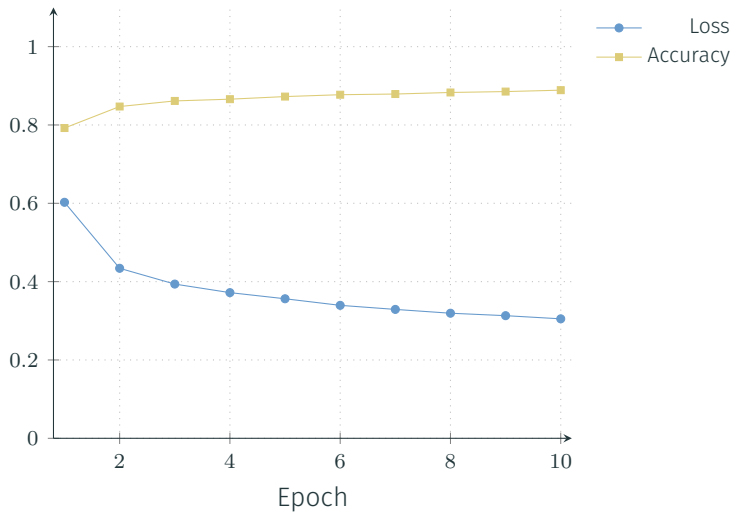

Verlauf während des Trainings



- Vorteil: Schnellere Konvergenz
- Verwendung von optimierter Cost-, Activation- und Gradient-Descent-Funktion





```
model = keras.models.Sequential([  
    keras.layers.Flatten(input_shape=(28, 28)),  
    keras.layers.Dense(30, activation='relu'),  
    keras.layers.Dense(15, activation='relu'),  
    keras.layers.Dense(10, activation='softmax'),  
)  
  
model.compile(loss='sparse_categorical_crossentropy',  
              optimizer='Adam',  
              metrics=['accuracy'])
```

Verlauf während des Trainings



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.