Künstliche Intelligenz kapieren und programmieren

Teil 6: Aus Fehlern lernen – Error Backpropagation

Michael Weigend
Universität Münster

mw@creative-informatics.de
www.creative-informatics.de

2024

Materialien bei GitHub:
https://github.com/mweigend/ki-workshop

Tag 2

Zeit	Thema	Inhalte
9.00	Perzeptron	Neuron, Aktivierungsfunktion, Daten visualisieren mit Matplotlib, Rosenblatt-Perzeptron für logische Operationen
11.00	**Aus Fehlern lernen**	**Error-Backpropagation, einfaches künstliches neuronales Netz (KNN) mit verborgenen Knoten**
12.45	*Mittagspause*	
13.45	Ziffern erkennen	NumPy, KNN mit Array-Operationen, das Ziffern erkennen kann
15.00	Anwendung von KI	Verkehrsschilder erkennen, Gesichter erfassen, Experimente mit OpenCV
16.00	*Ende*	

6.1 Die Grenzen des Rosenblatt-Perzeptrons

Exklusives ODER (XOR)

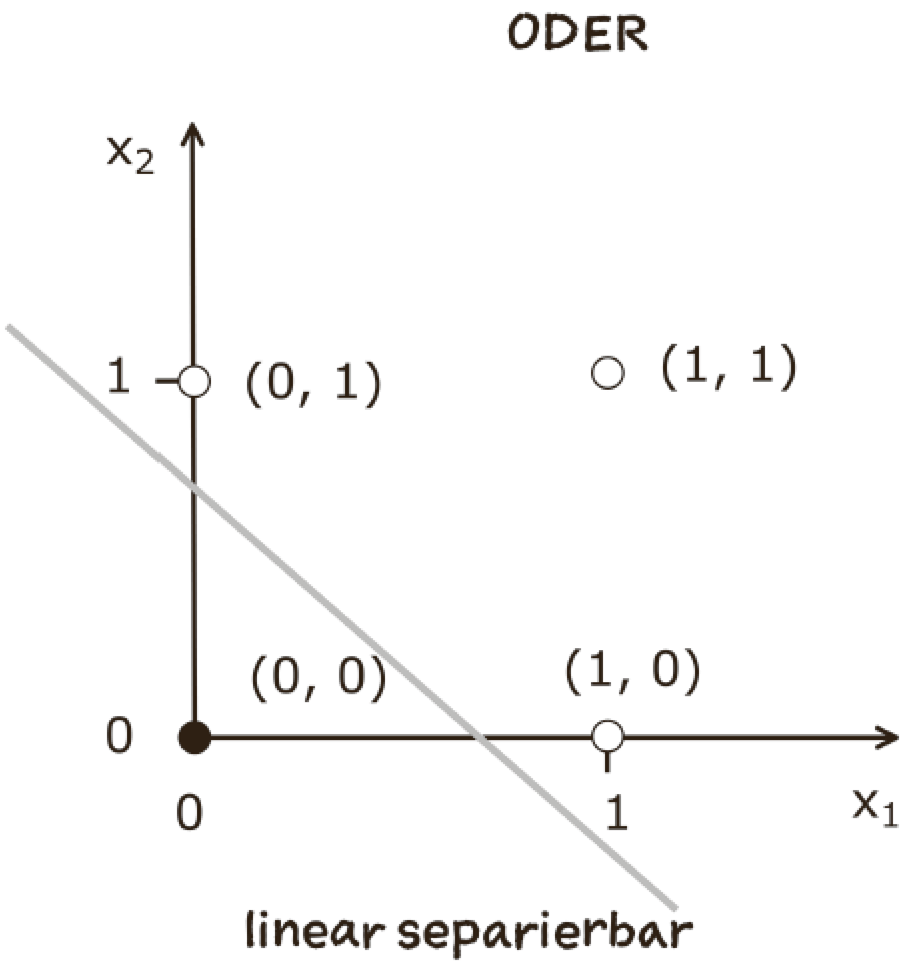
a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

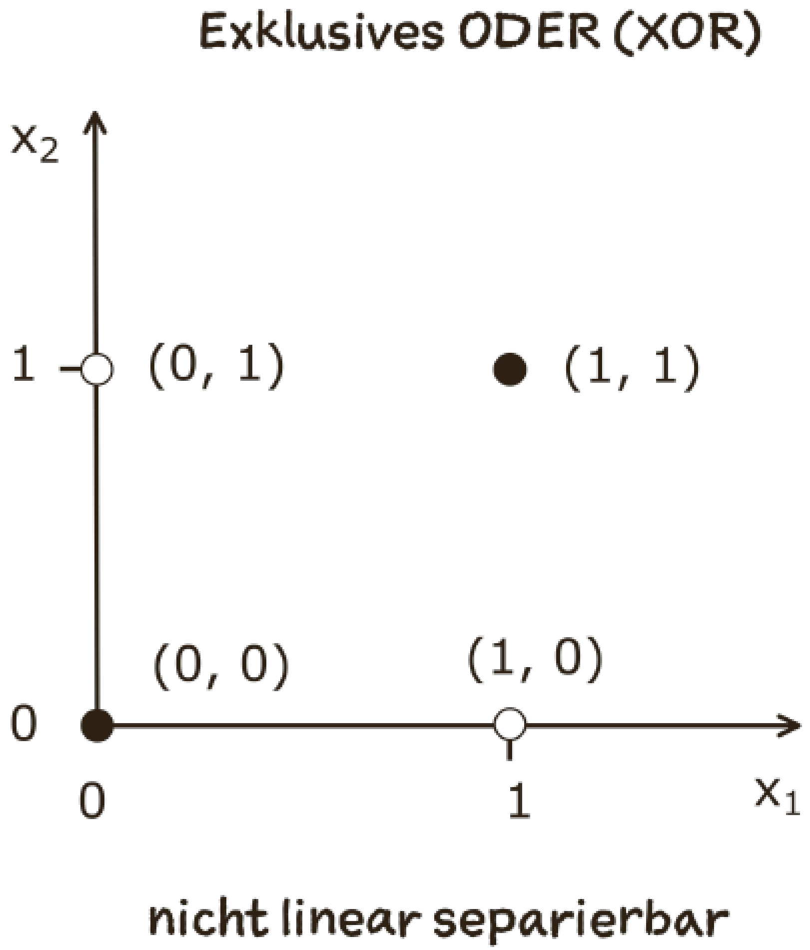
Die Grenzen des Rosenblatt-Perzeptrons

Seymour Papert und Marvin Minsky (1969):

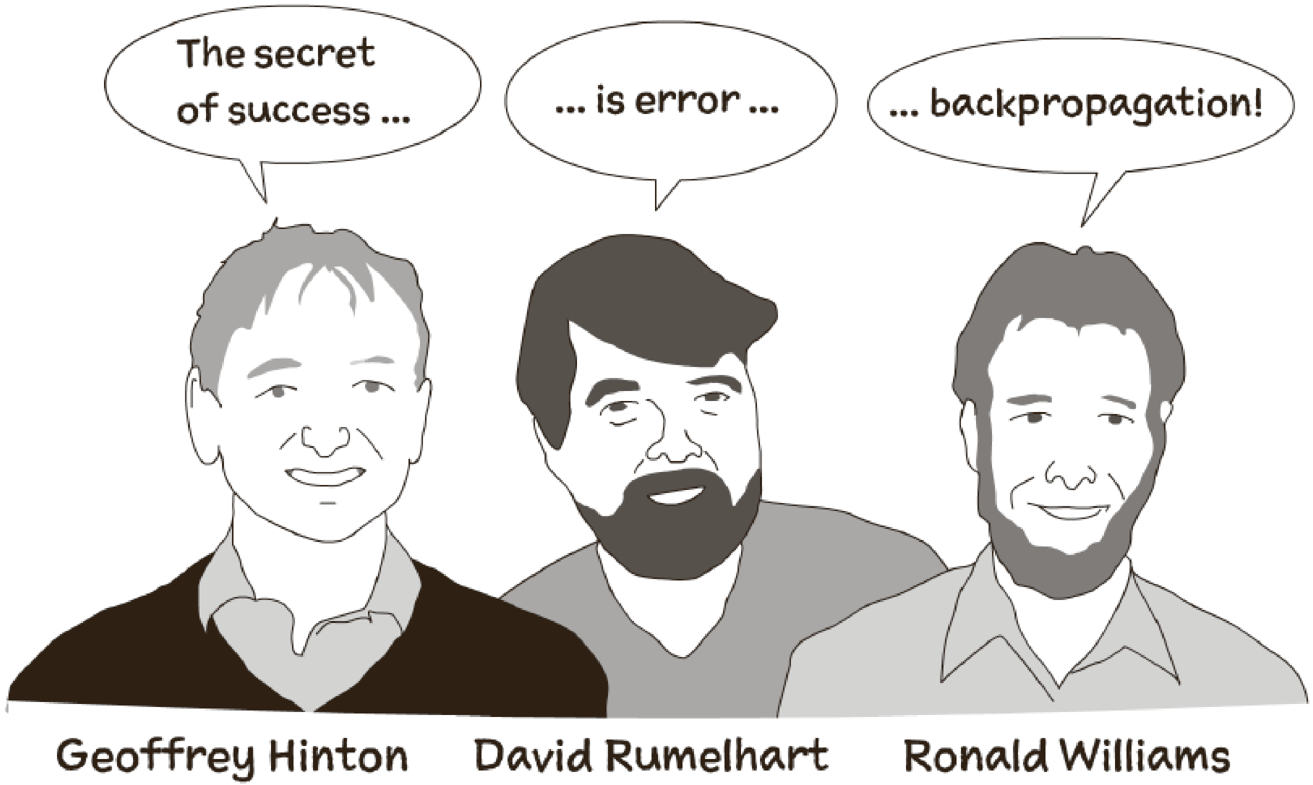
Ein Perzeptron, das nur kann nur aus Eingabe- und Ausgabeknoten besteht, kann nur solche Datenpunkte klassifizieren, die linear separierbar sind

Das XOR-Problem





6.2 Error Backpropagation

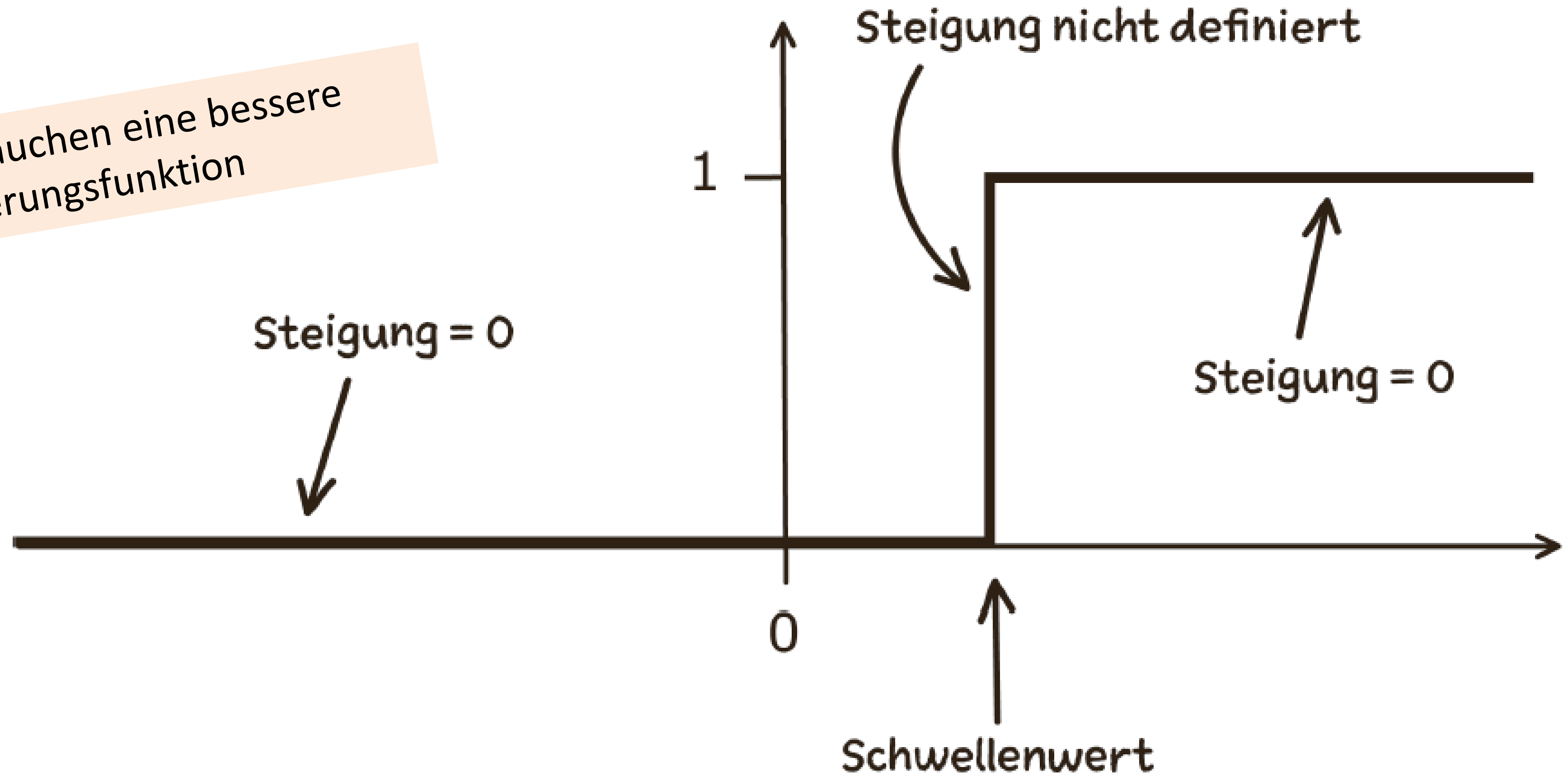


Learning representations by back-propagating errors (1987)

- »Verborgene Knoten«, die zwischen Eingabe- und Ausgabeknoten liegen
- Error Backpropagation (auf Deutsch »Fehlerrückführung«), ein neuer Mechanismus für die Anpassung der Gewichte während des Trainings.

Das Problem der Schwellenwertfunktion

Wir brauchen eine bessere Aktivierungsfunktion

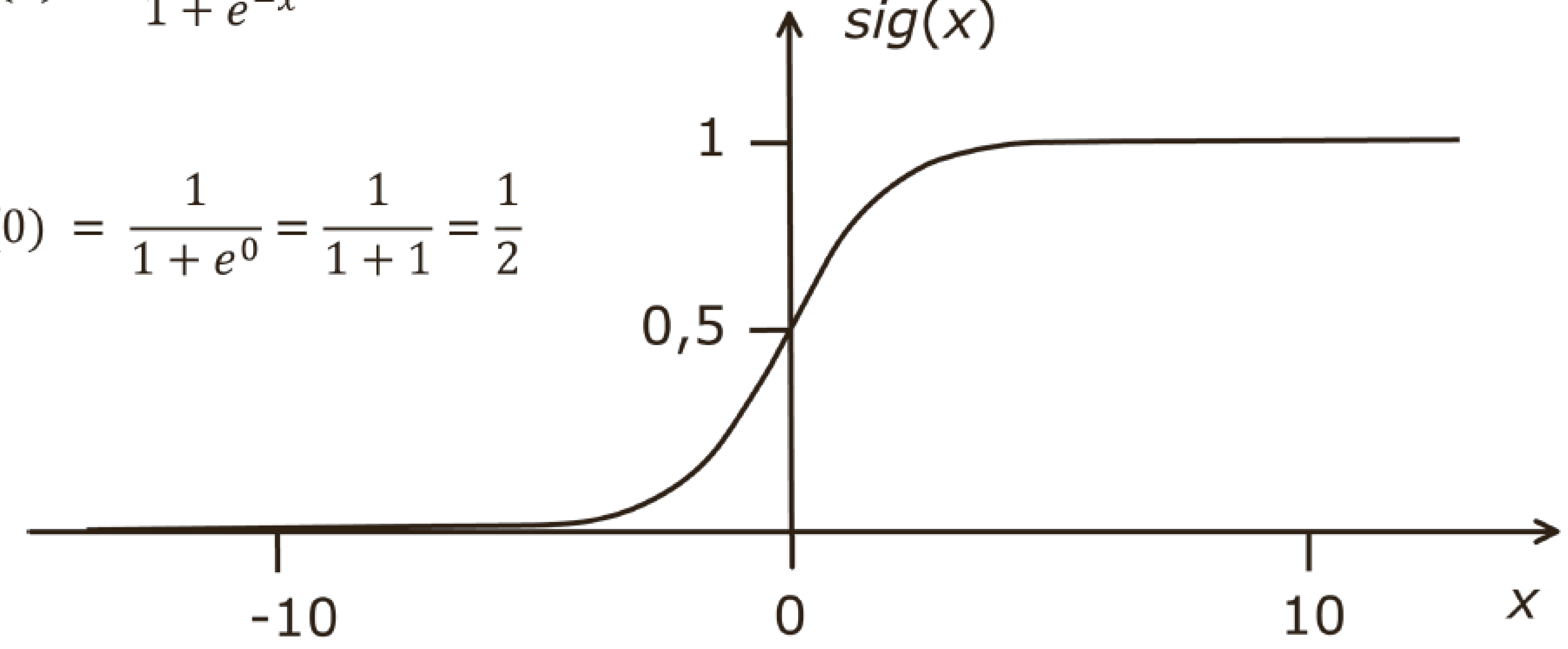


Die Sigmoid-Funktion

Eulersche Zahl e =2,7182818 …

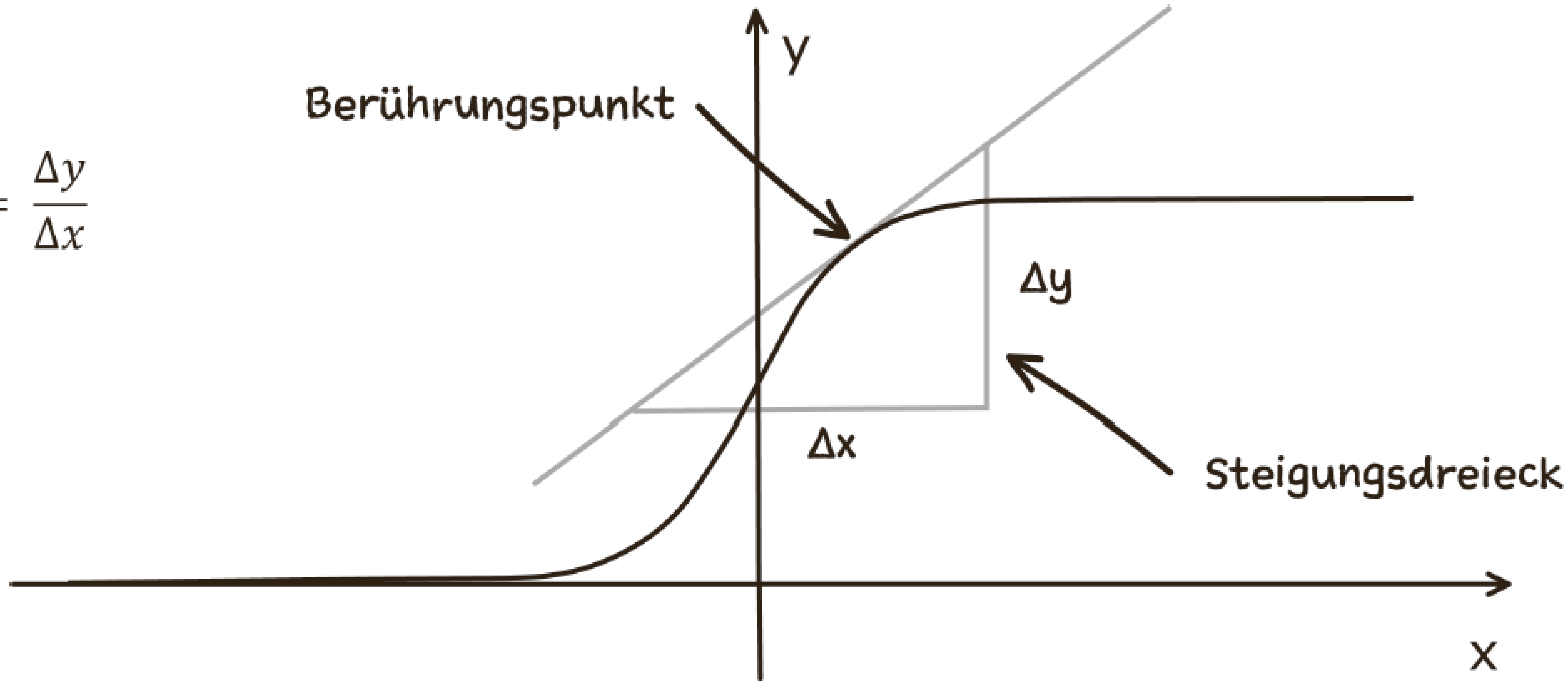
$$sig(x) = \frac{1}{1+e^{-x}}$$

$$sig(0) = \frac{1}{1+e^{0}} = \frac{1}{1+1} = \frac{1}{2}$$

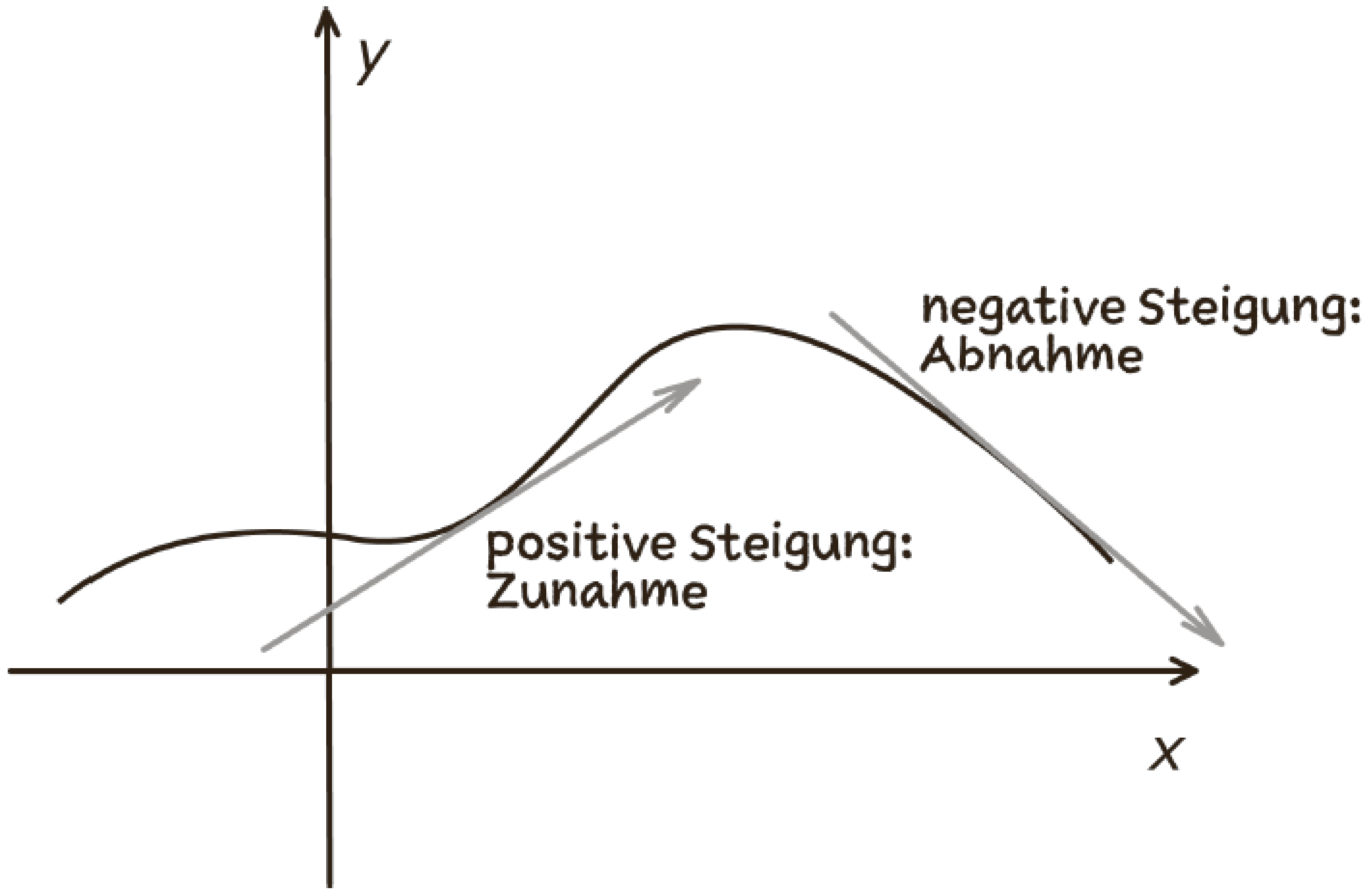


Steigung

$$Steigung = \frac{\Delta y}{\Delta x}$$

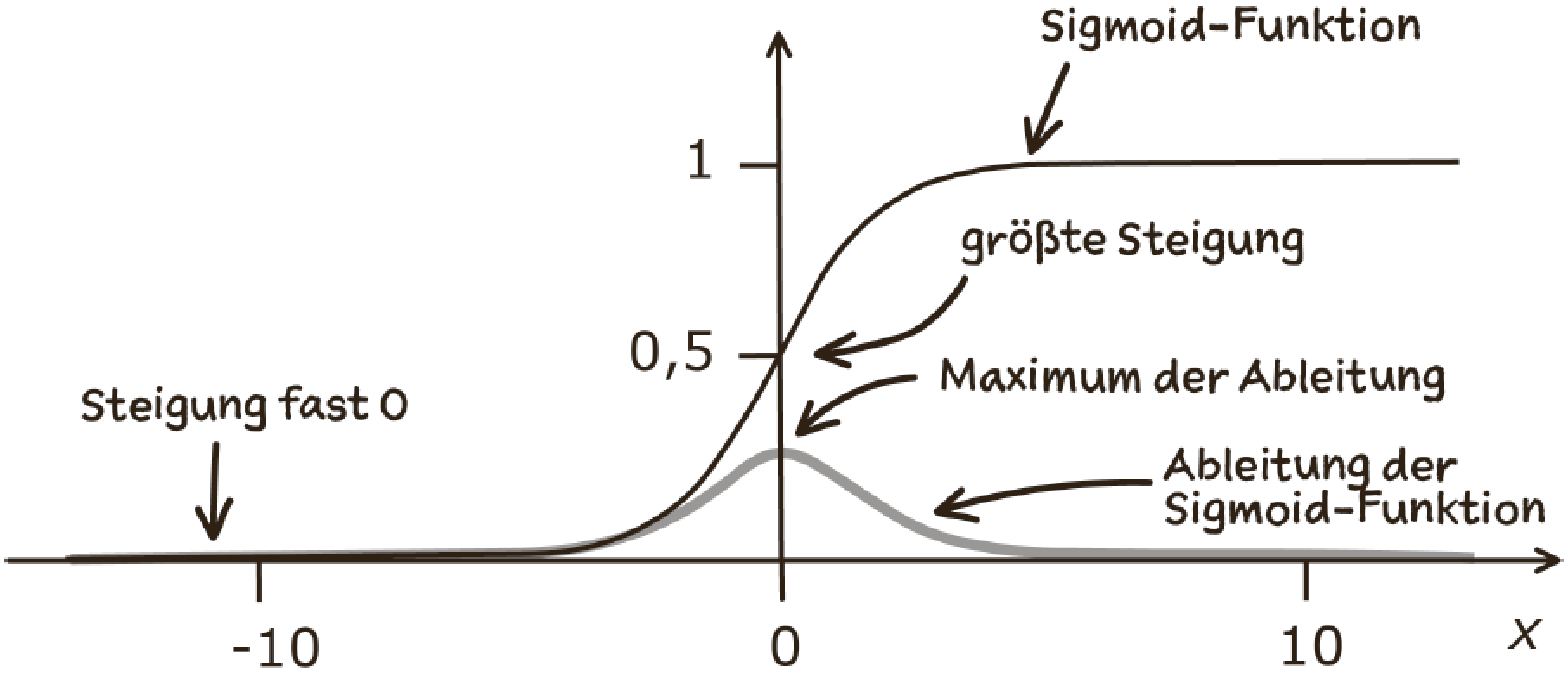


Steigung im Alltag



Ableitung der Sigmoid-Funktion

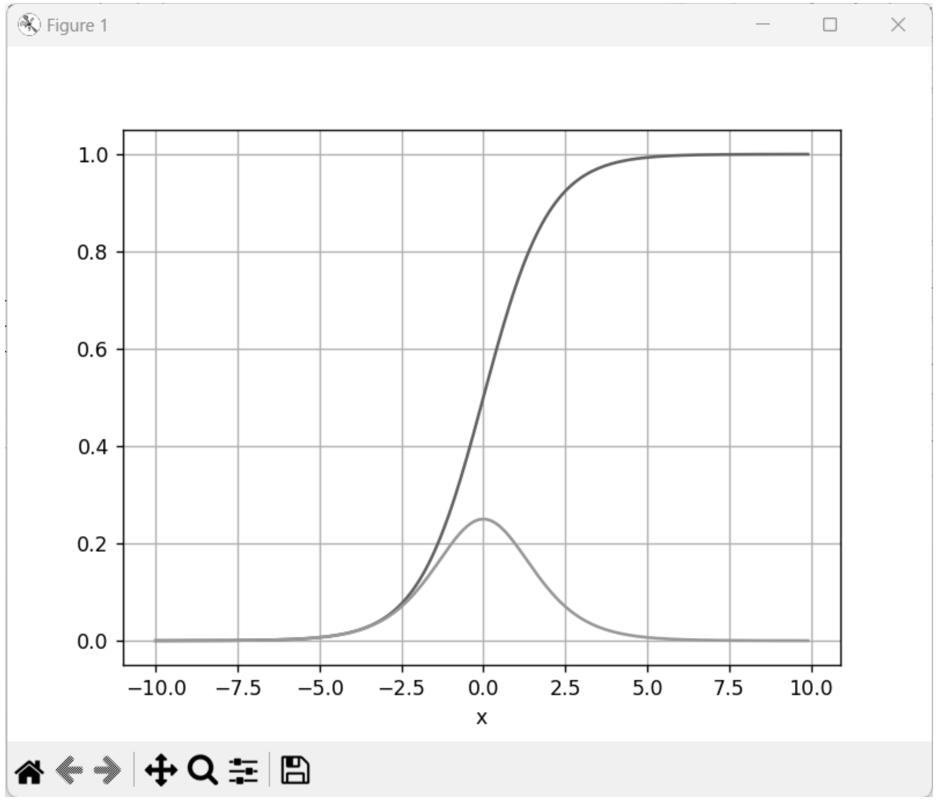
$$sig'(x) = \frac{d\ sig(x)}{dx} = sig(x) \cdot \left(1 - \ sig(x)\right)$$



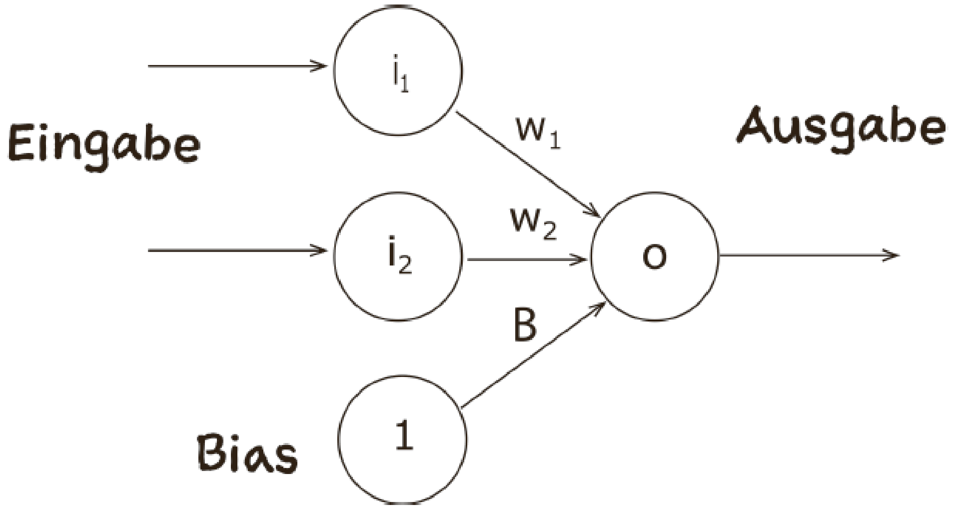
Übung 6.1 (10 min)

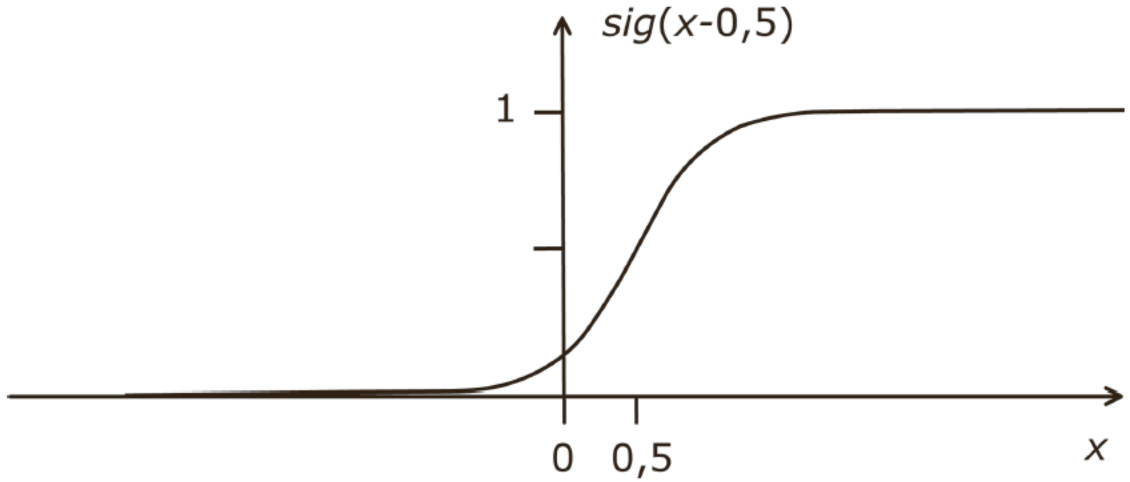
Erweitern Sie das folgende Programm, so dass sig(x) und die Ableitungsfunktion dargestellt werden.
Tipp: Zum Plotten der Ableitungsfunktion fügen Sie einen zweiten Aufruf von plot() ein.

```
# sigmoid_plot_starter.py
from matplotlib.pyplot import *
from math import e

def sig(x):
    return 1 / (1 + e**-x)

x_ = [x/10 for x in range(-100, 100)]
y_1 = [sig(x) for x in x_]
plot(x_, y_1)
xlabel('x')
grid()
show()
```



Perzeptron mit Bias und Aktivierungsfunktion sig()

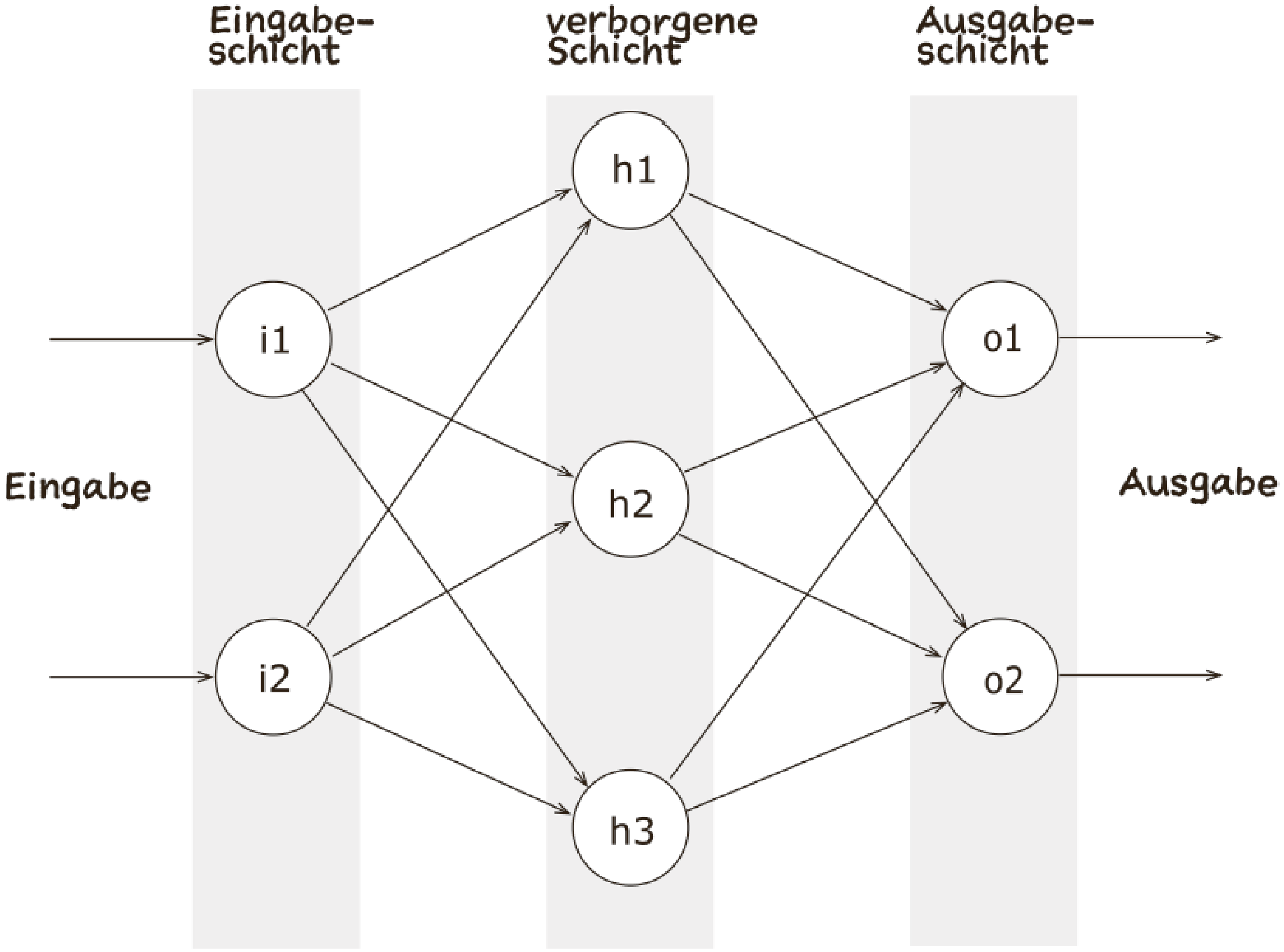




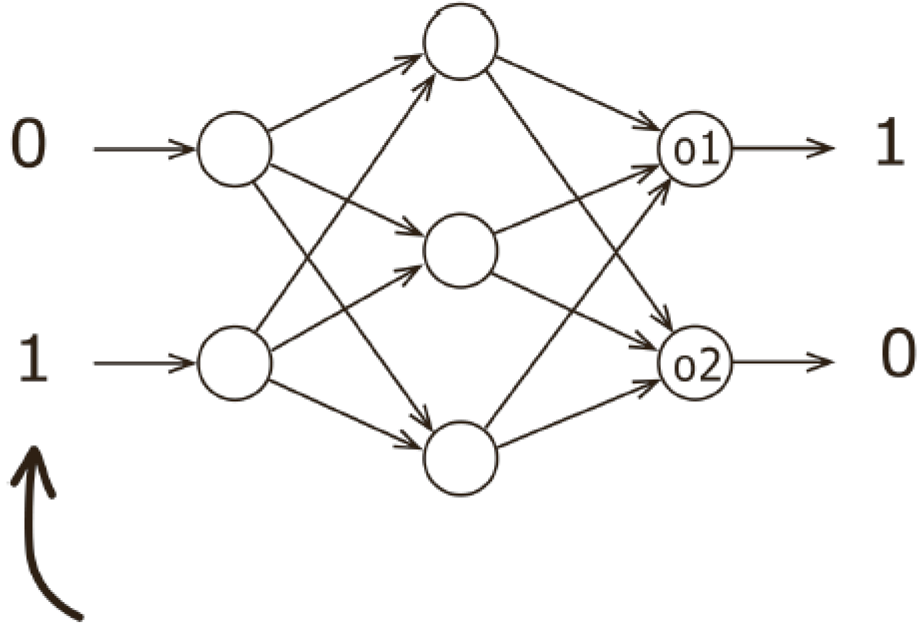
Gewichtete Eingabe:

$$x = i_1 \cdot w_1 + i_2 \cdot w_2 + B$$

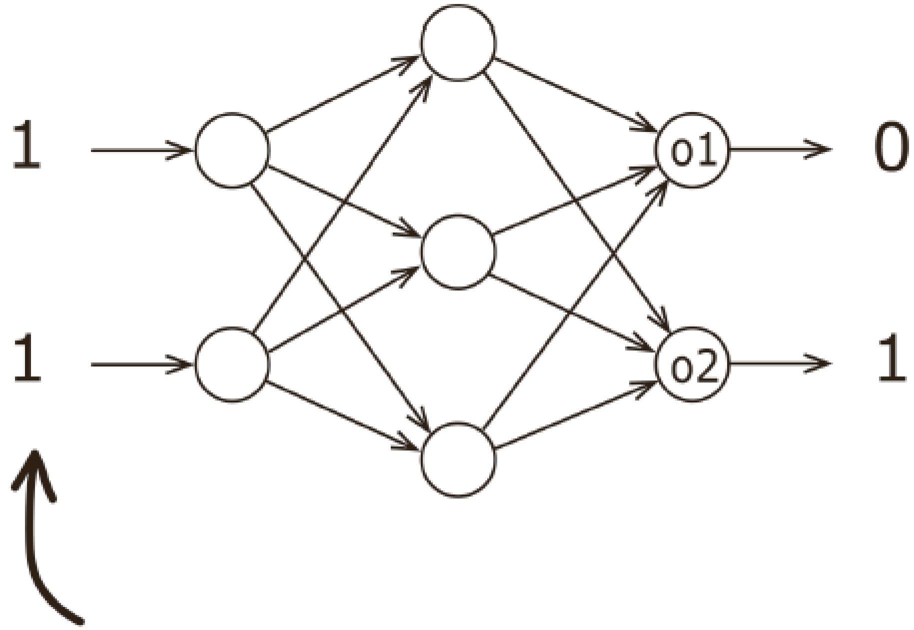
Neuronales Netz mit verborgenen Knoten



XOR-Detektor



ungleiche Eingabewerte
XOR



gleiche Eingabewerte
nicht XOR

Trainingsdaten

Zufällige Eingabedaten (Input)

i1	i2	t1	t2
0	0	0	1
0	1	1	0
1	1	0	1
1	0	1	0
0	1	1	0
...			

Erwartete Ausgabe (Target)

Die 4 Phasen eines Lernschritts

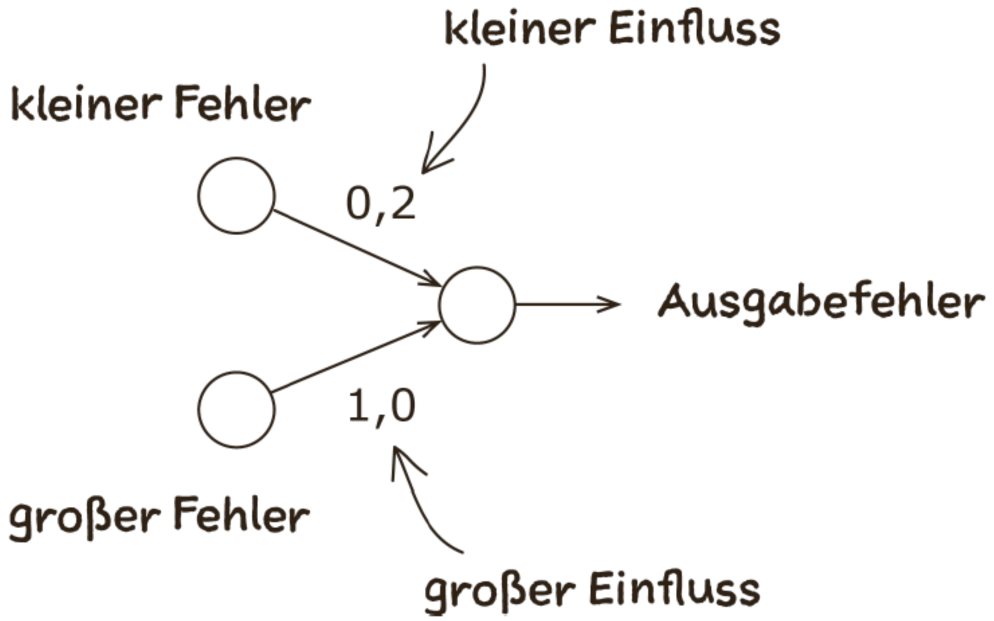
- Eingaben (i1 und i2) werden gelesen und im neuronalen Netz verarbeitet. Es entsteht eine Ausgabe o1 und eine Ausgabe o2.

- Die Ausgaben o1 und o2 werden mit der erwarteten Ausgabe t verglichen und es wird jeweils der Fehler an jedem Ausgabeknoten berechnet. Der Fehler ist die Differenz zwischen der tatsächlichen Ausgabe und der erwarteten Ausgabe (t).

- Auch für die anderen Knoten wird ein Ausgabefehler berechnet. Das ist die eigentliche Error Backpropagation.

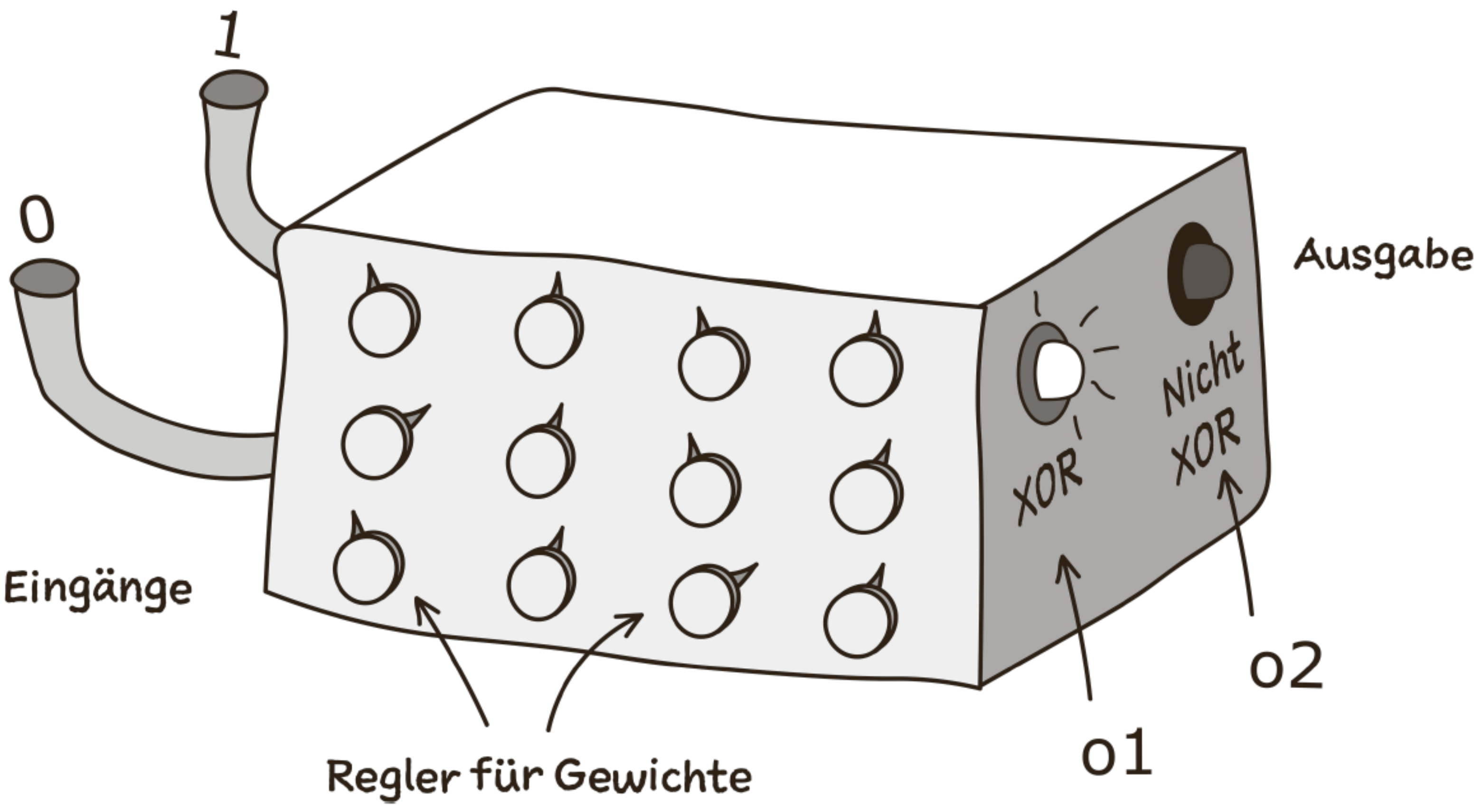
- Aufgrund der Fehler werden die Gewichte zwischen den Knoten etwas geändert.

Error Backpropagation (Fehlerrückführung)



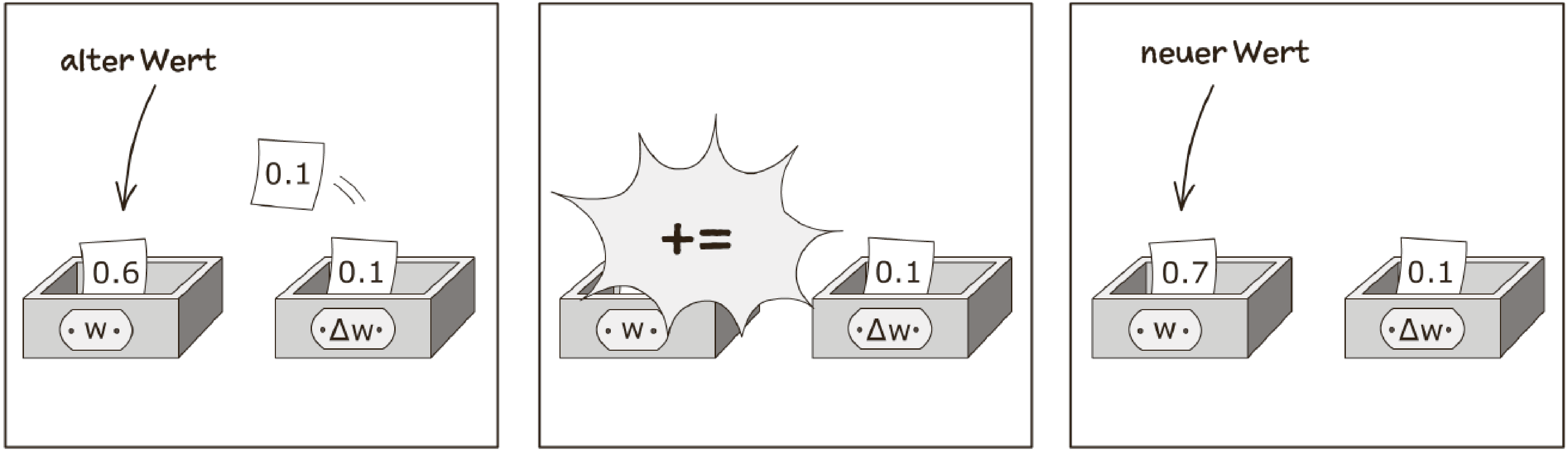
- Den inneren Knoten werden Fehler zugeordnet
- Das Gewicht bestimmt die Größe des Fehlers

Lernen = Gewichte ändern



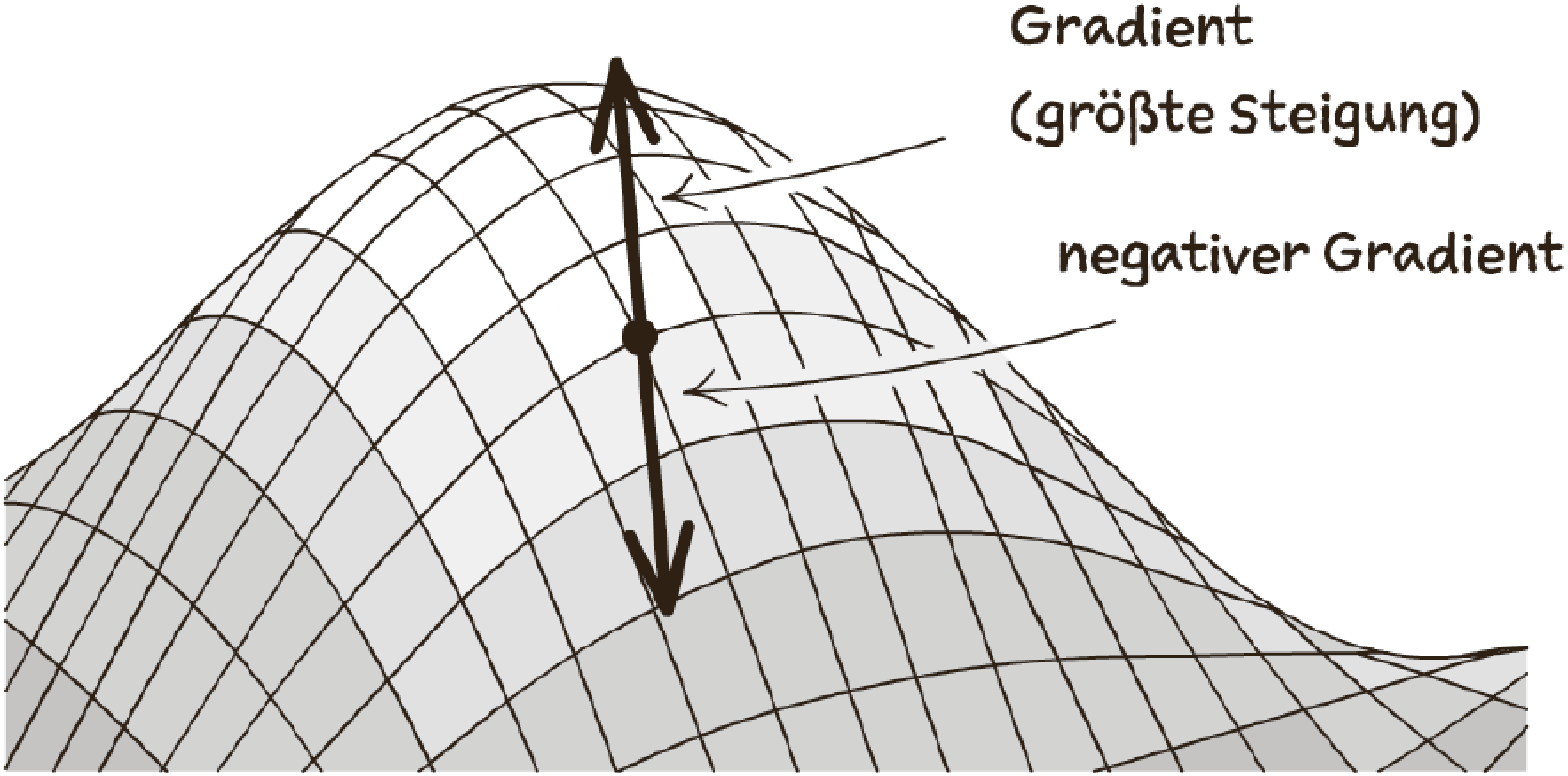
Ein Gewicht ändern

$$w += \Delta w$$

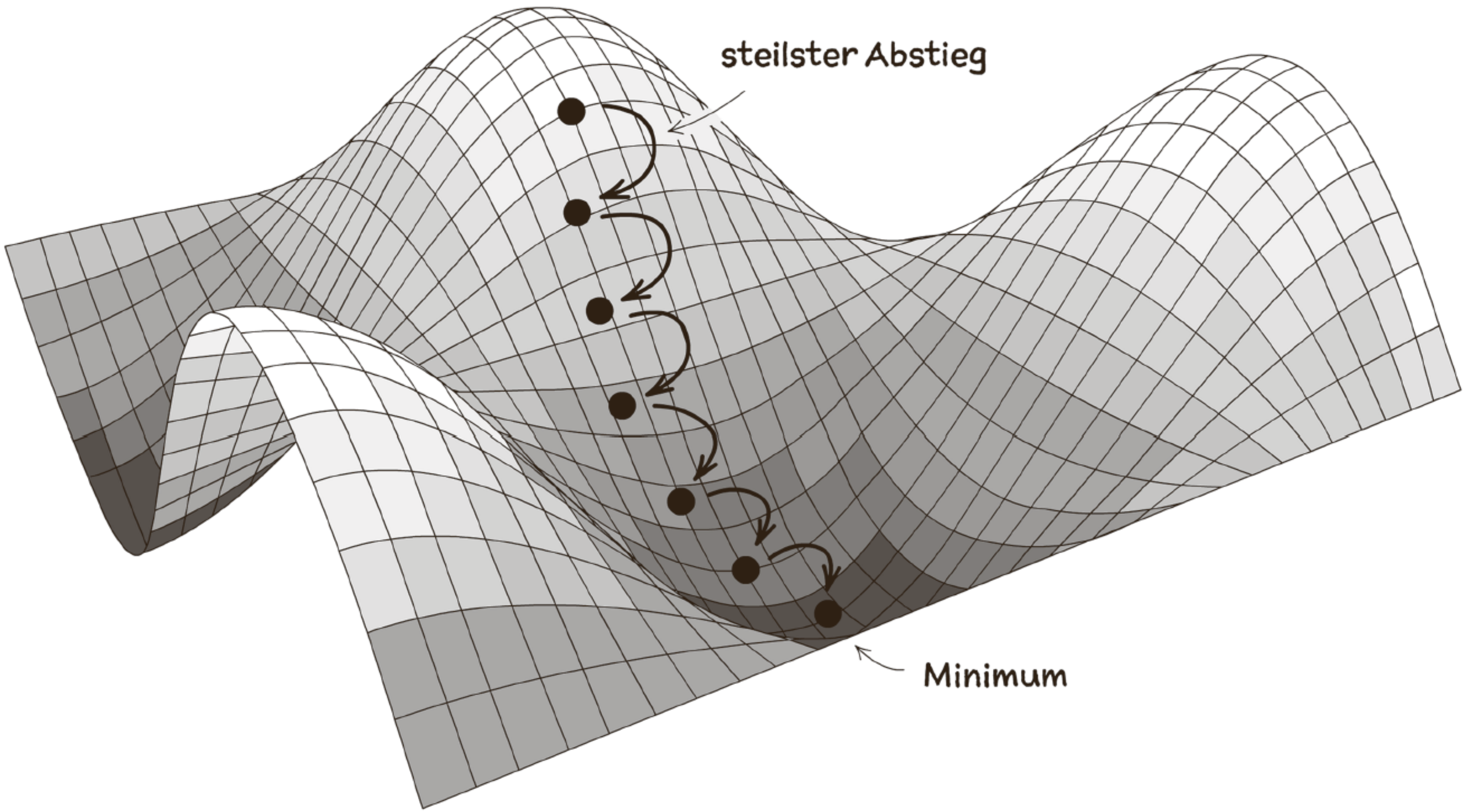


Gradienten-
verfahren

Bewegungsrichtung

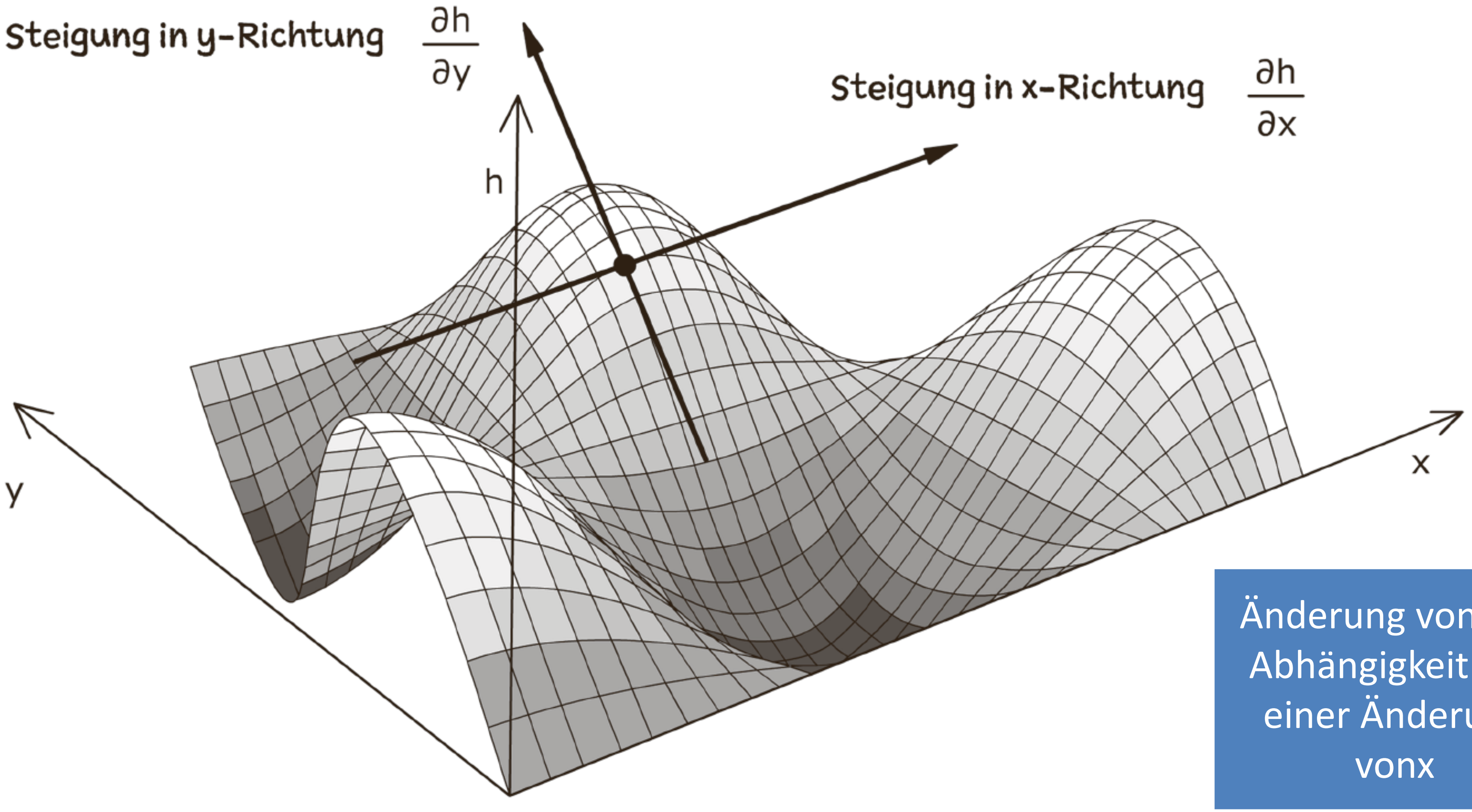


In mehreren Schritten zum tiefsten Punkt



Welche Problem könnte auftreten?

Partielle Ableitung



$$h = f(x, y)$$

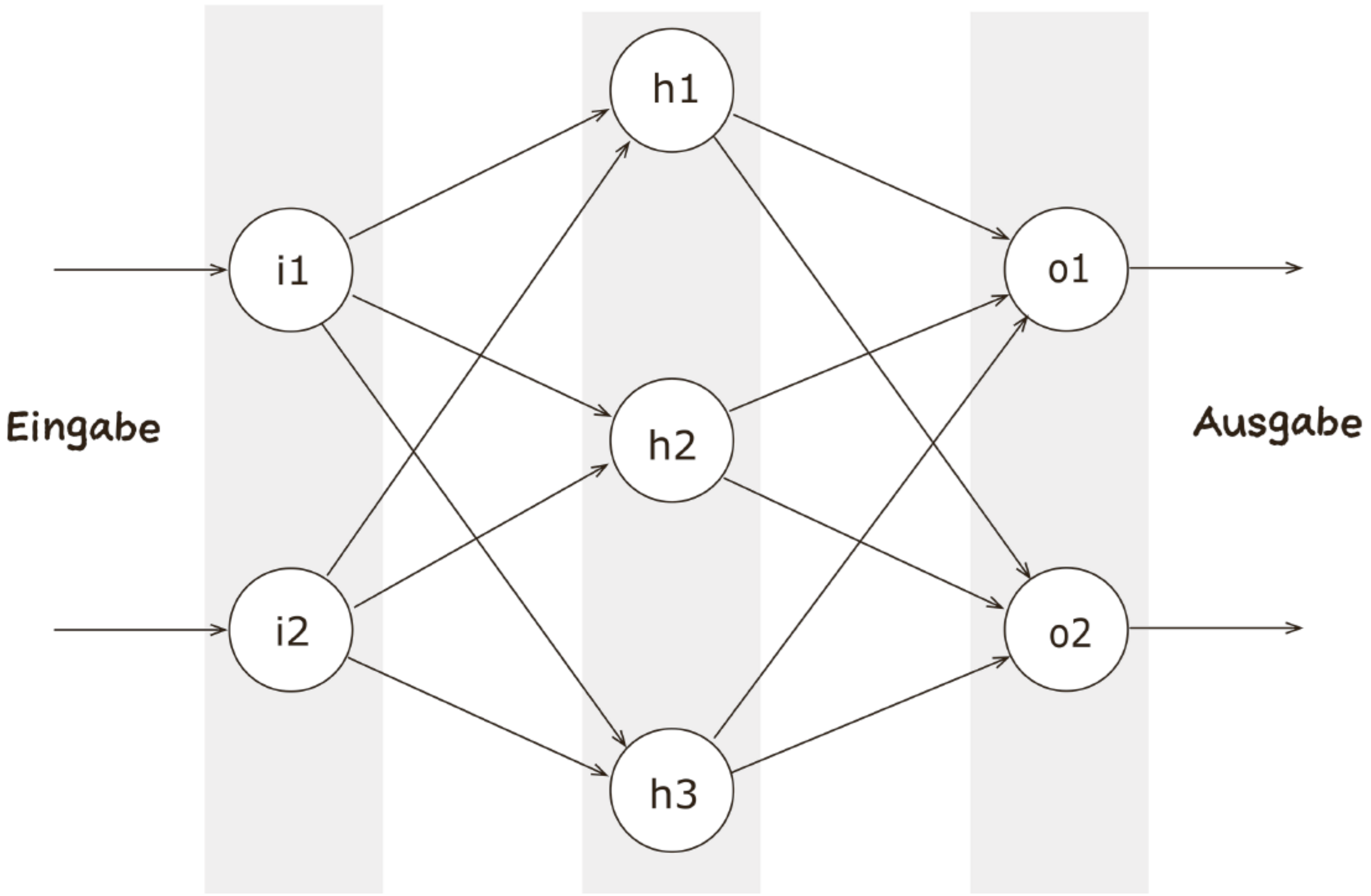
Höhenfunktion: Die Höhe ist eine Funktion
der Position auf der Karte.

Del Zeichen, steht für winzig kleine Änderung

$$\frac{\partial h}{\partial x}$$

Änderung von h in Abhängigkeit von einer Änderung vonx

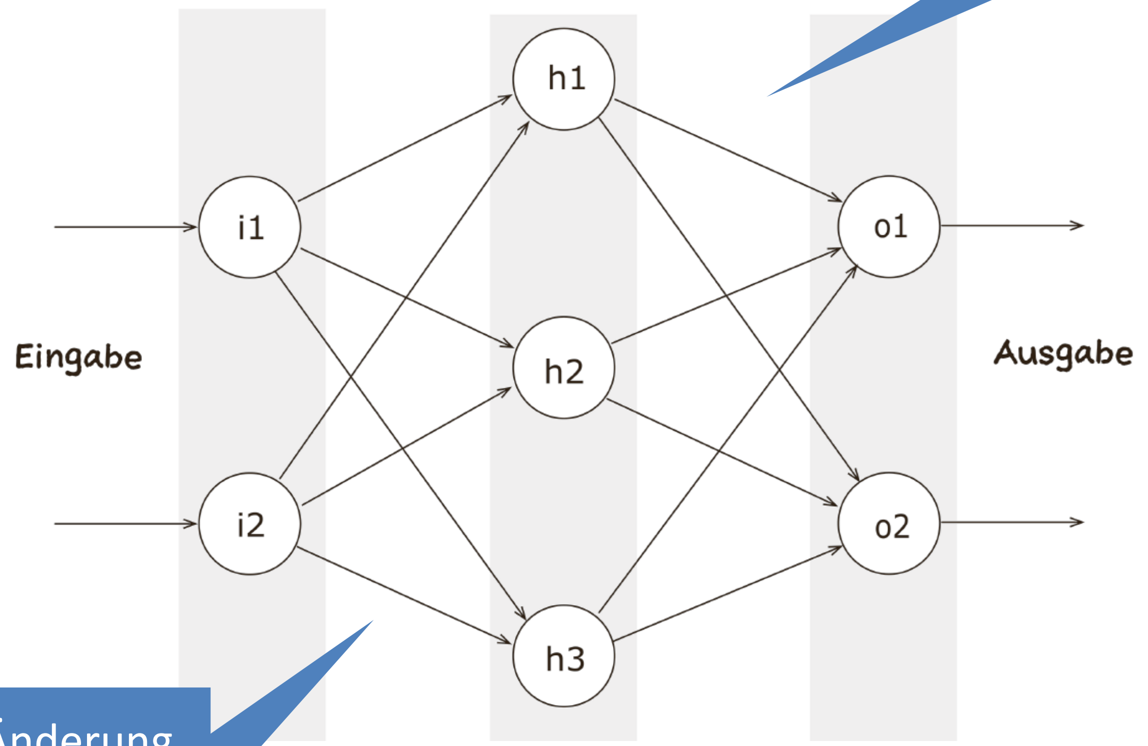
Zurück zum neuronalen Netz ...



Wie viele Gewichte?

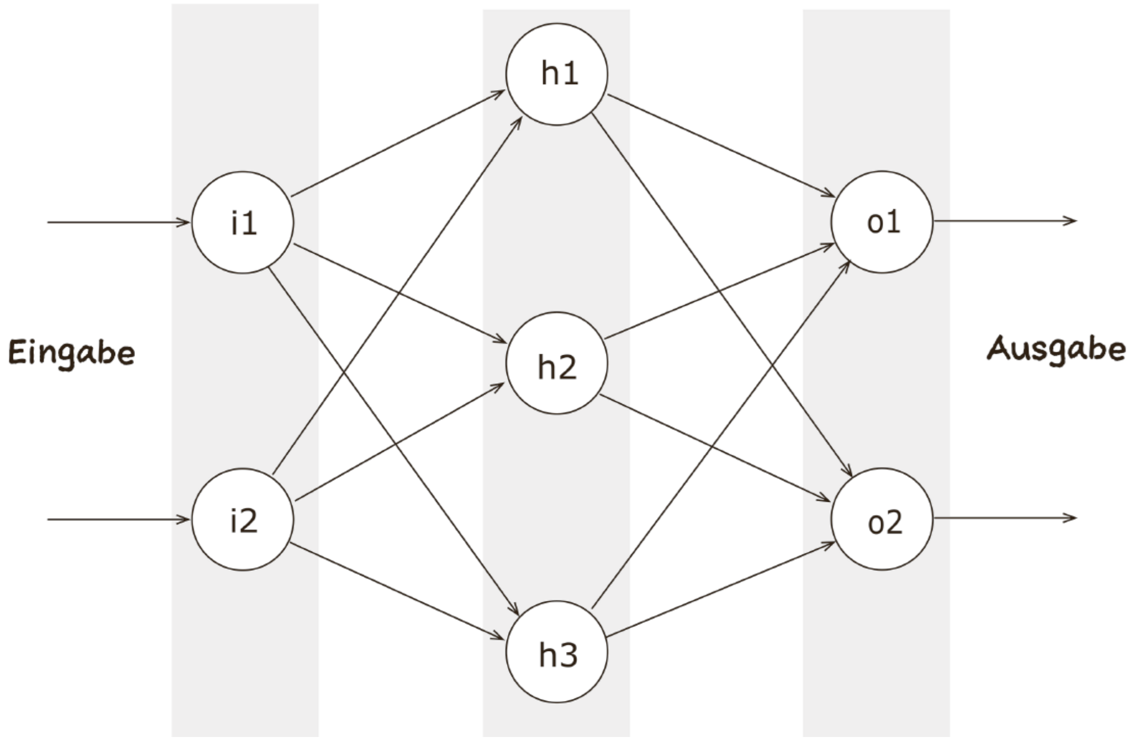
Das Ziel

Formeln für die Änderung der Gewichte zwischen verborgener Schicht und Ausgabeschicht



Formeln für die Änderung der Gewichte zwischen Engabeschicht und verborgener Schicht

Fehler



i1	i2	t1	t2
0	0	0	1
0	1	1	0
1	1	0	1
1	0	1	0
0	1	1	0
...			

Beim Training werden die Gewichte geändert, so dass der Fehler minimal wird.
Formel für den Fehler in der Ausgabe?

Fehlerfunktion

$$E = (t_1 - o_1)^2 + (t_2 - o_2)^2$$

Warum nimmt man das Quadrat der Fehler?

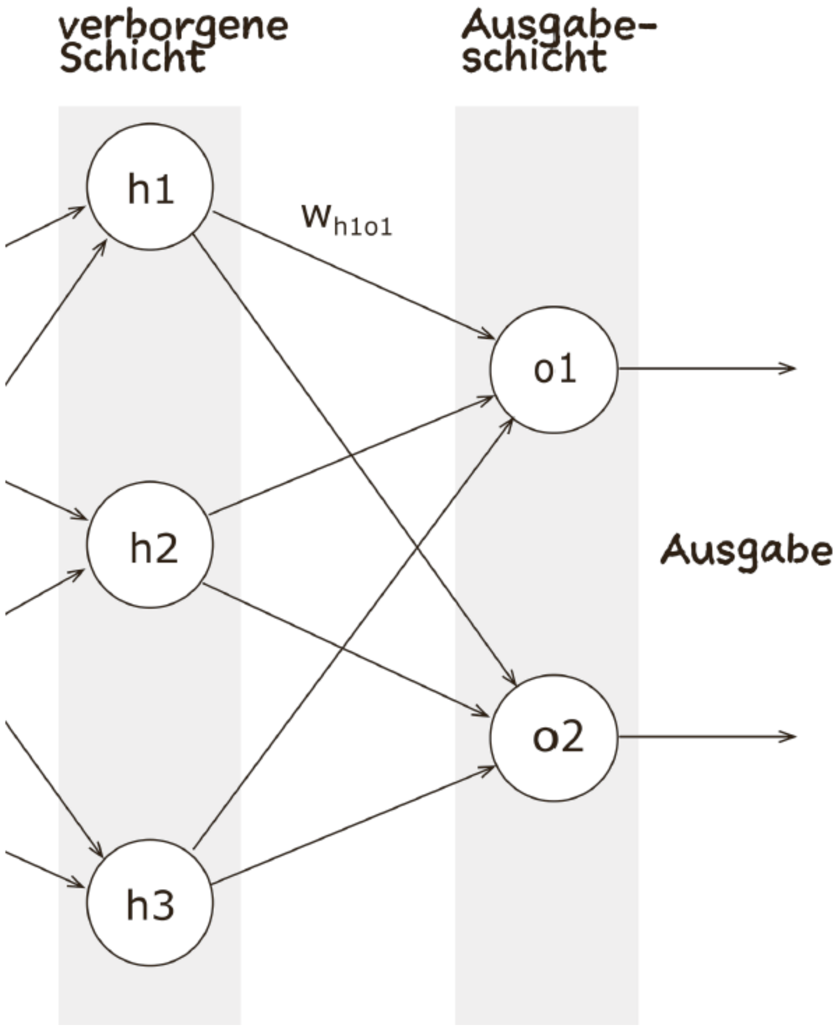
1. Fehler immer positiv
2. Ableitung leicht zu berechnen (sonst hätte man Fallunterscheidungen)
3. Wenn der Fehler klein ist, ist das Fehlerquadrat noch kleiner. Veränderungen in der Nähe des Minimums sind dann besonders vorsichtig.

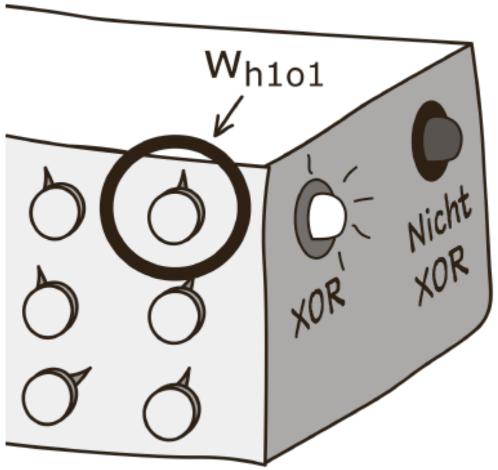
Änderung eines Gewichts in einem Trainingsschritt

$$w \mathrel{+}= \alpha \cdot \left(-\frac{\partial E}{\partial w}\right)$$

Lernrate

Änderung des Gewichts w_{h1o1}





$$\frac{\partial E}{\partial w_{h1o1}} = \frac{\partial((t_1 - o_1)^2 + (t_2 - o_2)^2)}{\partial w_{h1o1}}$$

Änderung des Fehlers in Abhängigkeit von der Änderung des Gewichts w_{h101}

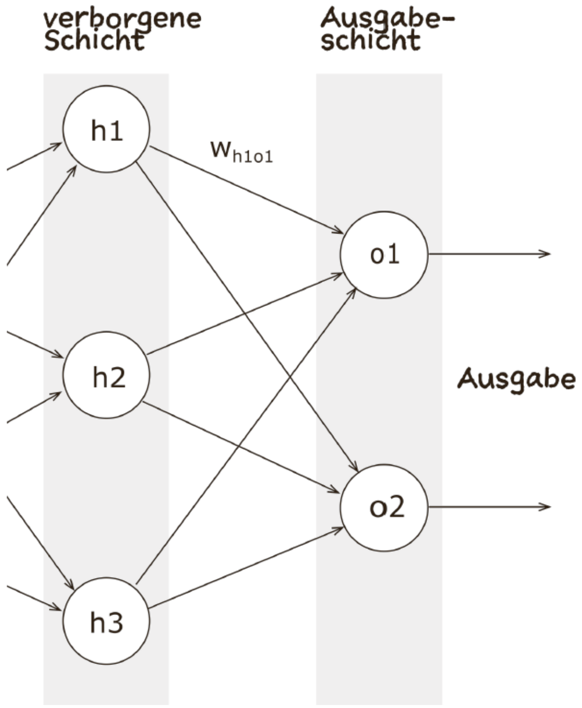
$$\frac{\partial E}{\partial w_{h1o1}} = \frac{\partial((t_1 - o_1)^2 + (t_2 - o_2)^2)}{\partial w_{h1o1}}$$

$$\frac{\partial E}{\partial w_{h1o1}} = \frac{\partial(t_1 - o_1)^2}{\partial w_{h1o1}} + \frac{\partial(t_2 - o_2)^2}{\partial w_{h1o1}}$$

gleich null

Änderung des Fehlerquadrats am Ausgabeknoten o1 bei Änderung von w_{h1o1}

Änderung des Fehlersquadrats am Ausgabeknoten o2 bei Änderung von w_{h1o1}



$$\frac{\partial E}{\partial w_{h1o1}} = \frac{\partial(t_1 - o_1)^2}{\partial w_{h1o1}}$$

$$\frac{\partial E}{\partial w_{h1o1}} = \frac{\partial (t_1 - o_1)^2}{\partial w_{h1o1}}$$

Nach der Kettenregel für partielle Ableitungen

$$\frac{\partial E}{\partial w_{h1o1}} = \frac{\partial (t_1 - o_1)^2}{\partial o_1} \cdot \frac{\partial o_1}{\partial w_{h1o1}}$$

Gleichung 1

1. Faktor

2. Faktor

1. Faktor $\frac{\partial(t_1 - o_1)^2}{\partial o_1}$

Änderung des Fehlerquadrats in Abhängigkeit von der Ausgabe

ist die Ableitung der Funktion

$$\mathrm{E} = (t_1 - o_1)^2$$

Konstante

Variable

$$E \;= t_1{}^2 - 2 \cdot t_1 \cdot o_1 + o_1{}^2$$

$$\frac{\partial(t_1 - o_1)^2}{\partial o_1} = -2 \cdot (t_1 - o_1)$$

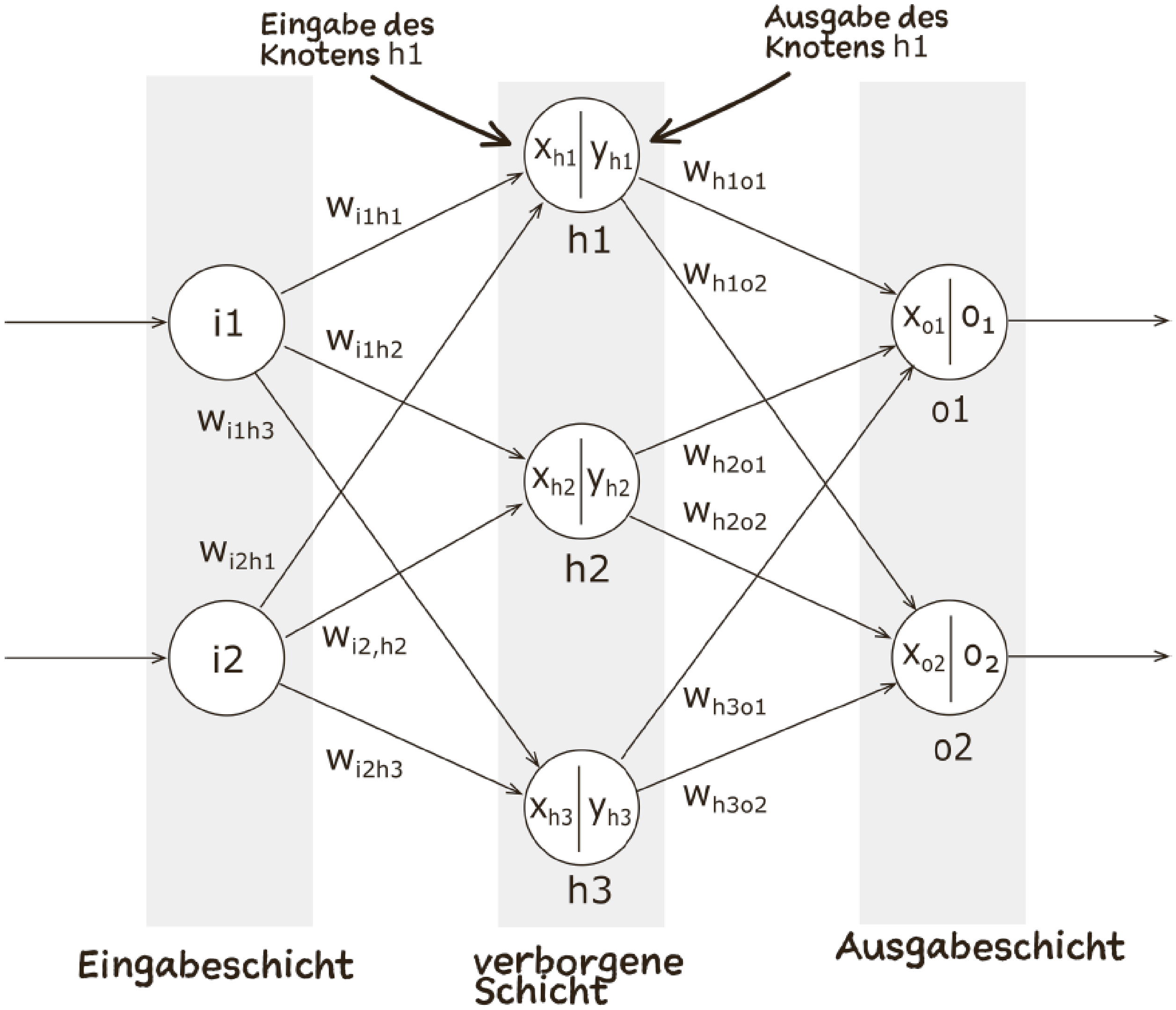
Gleichung 2

Ableitung nach den Ableitungsregeln für Polynome

2. Faktor $\frac{\partial o_1}{\partial w_{h1o1}}$

Änderung der Ausgabe in Abhängigkeit von der Änderung des Gewichts

Systematik der Variablennamen



2. Faktor $\frac{\partial o_1}{\partial w_{h1o1}}$

$$o_1 = sig(x_{o1})$$

$$\frac{\partial o_1}{\partial w_{h1o1}} = \frac{\partial sig(x_{o1})}{\partial w_{h1o1}}$$

Darum kümmern wir uns als nächstes

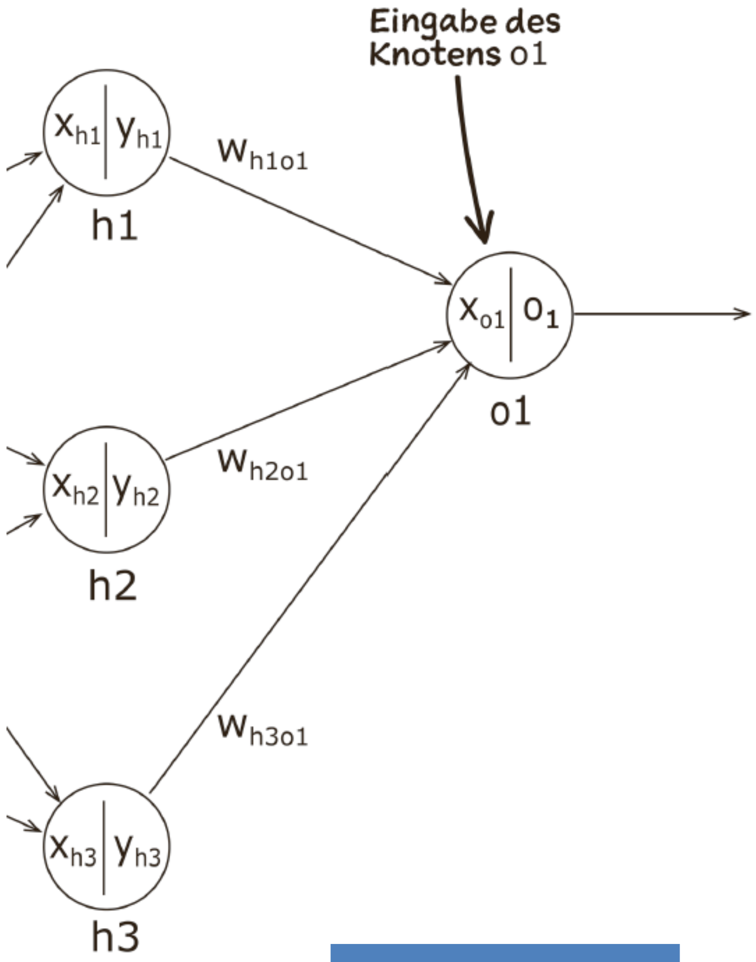
$$\frac{\partial o_1}{\partial w_{h1o1}} = \underbrace{sig(x_{o1}) \cdot (1 - sig(x_{o1}))}_{\substack{\text{Ableitung von } sig(x_{o1}) \\ \text{äußere Ableitung}}} \cdot \underbrace{\frac{\partial \boldsymbol{x_{o1}}}{\partial \boldsymbol{w_{h1o1}}}}_{\text{innere Ableitung}}$$

Gleichung 3

$$x_{o1} = y_{h1} \cdot w_{h1o1} + y_{h2} \cdot w_{h2o1} + y_{h3} \cdot w_{h3o1}$$

$$\frac{\partial x_{o1}}{\partial w_{h1o1}} = \frac{\partial (w_{h1o1} \cdot y_{h1} + w_{h2o1} \cdot y_{h2} + w_{h3o1} \cdot y_{h3})}{\partial w_{h1o1}}$$

$$\frac{\partial x_{o1}}{\partial w_{h1o1}} = \frac{\partial (w_{h1o1} \cdot y_{h1})}{\partial w_{h1o1}} + \underbrace{\frac{\partial (w_{h2o1} \cdot y_{h2})}{\partial w_{h1o1}}}_{\text{gleich null}} + \underbrace{\frac{\partial (w_{h3o1} \cdot y_{h3})}{\partial w_{h1o1}}}_{\text{gleich null}}$$



$$\frac{\partial x_{o1}}{\partial w_{h1o1}} = \frac{\partial (w_{h1o1} \cdot y_{h1})}{\partial w_{h1o1}}$$

$$\frac{\partial x_{o1}}{\partial w_{h1o1}} = y_{h1}$$

Einsetzen in Gleichung 3

$$\frac{\partial o_1}{\partial w_{h1o1}} = sig(x_{o1}) \cdot \left(1 - sig(x_{o1})\right) \cdot \frac{\boldsymbol{\partial x_{o1}}}{\boldsymbol{\partial w_{h1o1}}}$$

$$\frac{\partial o_1}{\partial w_{h1o1}} = sig(x_{o1}) \cdot \left(1 - sig(x_{o1})\right) \cdot y_{h1}$$

$$o_1 = sig(x_{o1})$$

$$\frac{\partial o_1}{\partial w_{h1o1}} = o_1 \cdot (1 - o_1) \cdot y_{h1}$$

Gleichung 4

$$\frac{\partial E}{\partial w_{h1o1}} = \frac{\partial (t_1 - o_1)^2}{\partial o_1} \cdot \frac{\partial o_1}{\partial w_{h1o1}}$$

einsetzen

$-2 \cdot (t_1 - o_1)$ aus Gleichung 2

$o_1 \cdot (1 - o_1) \cdot y_{h1}$ aus Gleichung 4

$$\frac{\partial E}{\partial w_{h1o1}} = -2 \cdot (t_1 - o_1) \cdot \ o_1 \cdot (1 - o_1) \cdot y_{h1}$$

Gleichung 5

Wie ändert sich das Fehlerquadrat $E = (t_1 - o_1)^2$, wenn man das Gewicht w_{h1o1} ändert?

Berechnung der Änderung des Gewichts beim Training

$$\Delta w_{h1o1} = -\alpha \cdot \frac{\partial E}{\partial w_{h1o1}}$$

Lernrate (Zahl zwischen 0 und 1)

$$\frac{\partial E}{\partial w_{h1o1}} = -2 \cdot (t_1 - o_1) \cdot o_1 \cdot (1 - o_1) \cdot y_{h1}$$

Gleichung 5

$$\Delta w_{h1o1} = -\alpha \cdot (-2) \cdot (t_1 - o_1) \cdot o_1 \cdot (1 - o_1) \cdot y_{h1}$$

$$\Delta w_{h1o1} = \alpha \cdot 2 \cdot (t_1 - o_1) \cdot o_1 \cdot (1 - o_1) \cdot y_{h1}$$

Berechnung der Änderung des Gewichts beim Training

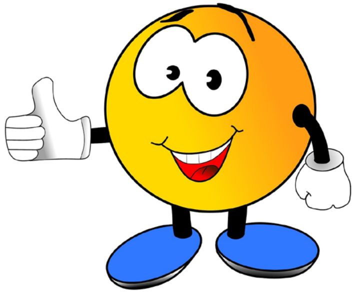
$$\Delta w_{h1o1} = \alpha \cdot 2 \cdot (t_1 - o_1) \cdot \; o_1 \cdot (1 - o_1) \cdot y_{h1}$$

weglassen

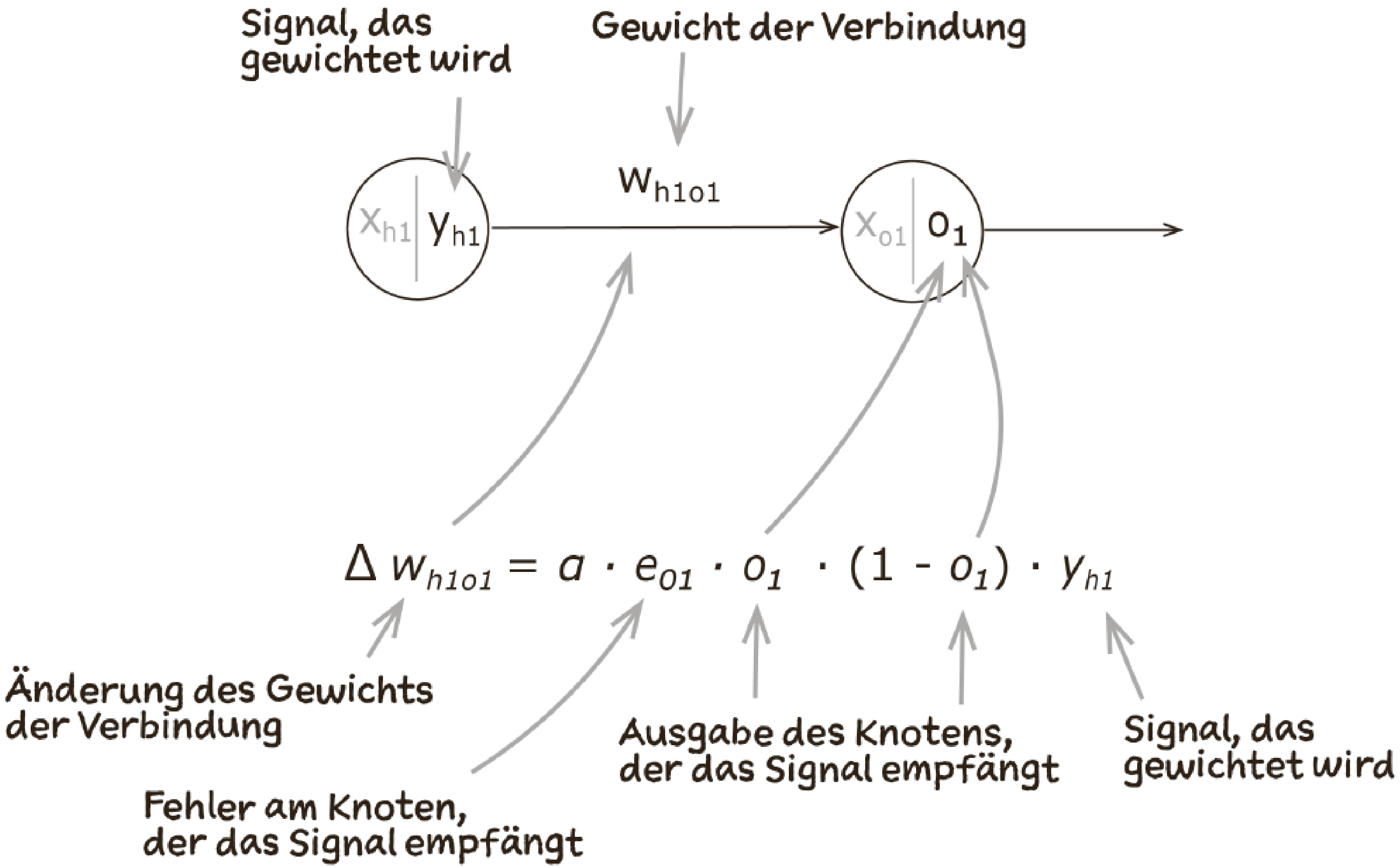
$$e_{o1} = t_1 - o_1$$

$$\Delta w_{h1o1} = \alpha \cdot \; e_{o1} \cdot \; o_1 \cdot (1 - o_1) \cdot y_{h1}$$

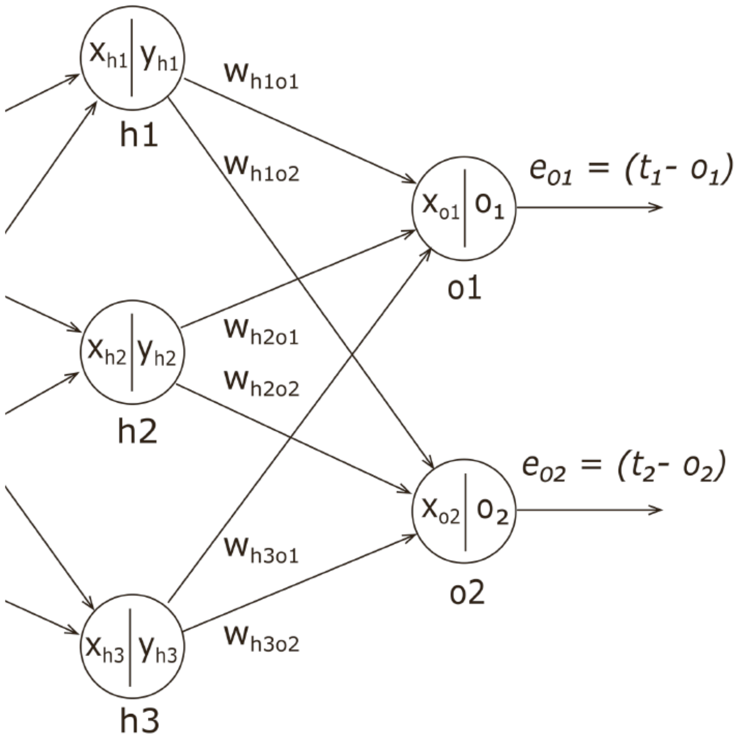
Gleichung 6



Aktualisierung der übrigen Gewichte



Aktualisierung der übrigen Gewichte



Fehler am Knoten,
der das Signal empfängt

Ausgabe des Knotens,
der das Signal empfängt

Signal,
das gewichtet wird

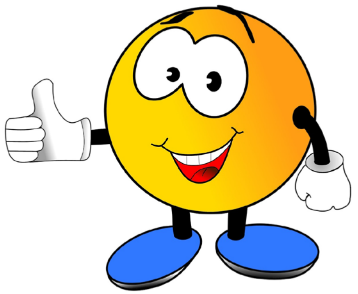
$$\Delta w_{h2o1} = \alpha \cdot e_{o1} \cdot o_1 \cdot (1 - o_1) \cdot y_{h2}$$

$$\Delta w_{h3o1} = \alpha \cdot e_{o1} \cdot o_1 \cdot (1 - o_1) \cdot y_{h3}$$

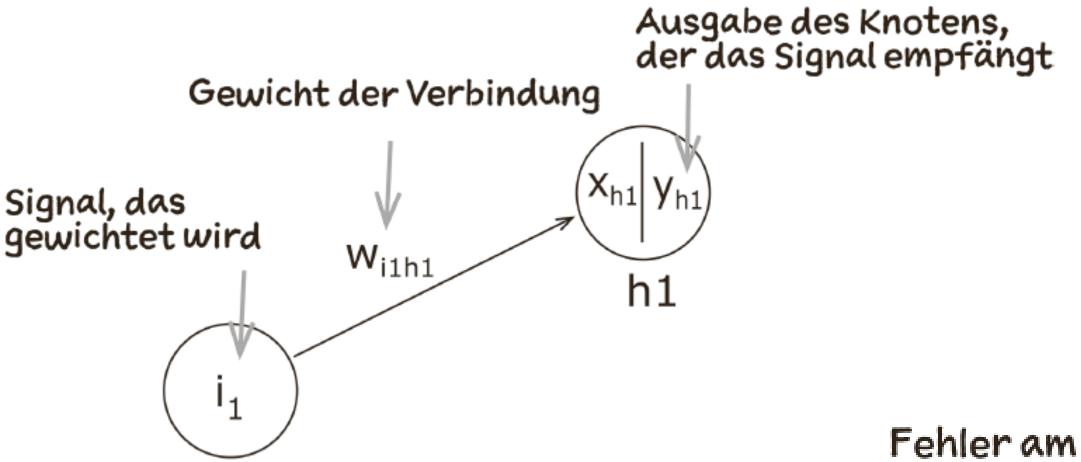
$$\Delta w_{h1o2} = \alpha \cdot e_{o2} \cdot o_2 \cdot (1 - o_2) \cdot y_{h1}$$

$$\Delta w_{h2o2} = \alpha \cdot e_{o2} \cdot o_2 \cdot (1 - o_2) \cdot y_{h2}$$

$$\Delta w_{h3o2} = \alpha \cdot e_{o2} \cdot o_2 \cdot (1 - o_2) \cdot y_{h3}$$



Gewichte zwischen Eingabeschicht und verborgener Schicht



Fehler am Knoten, der das Signal empfängt

Ausgabe des Knotens, der das Signal empfängt

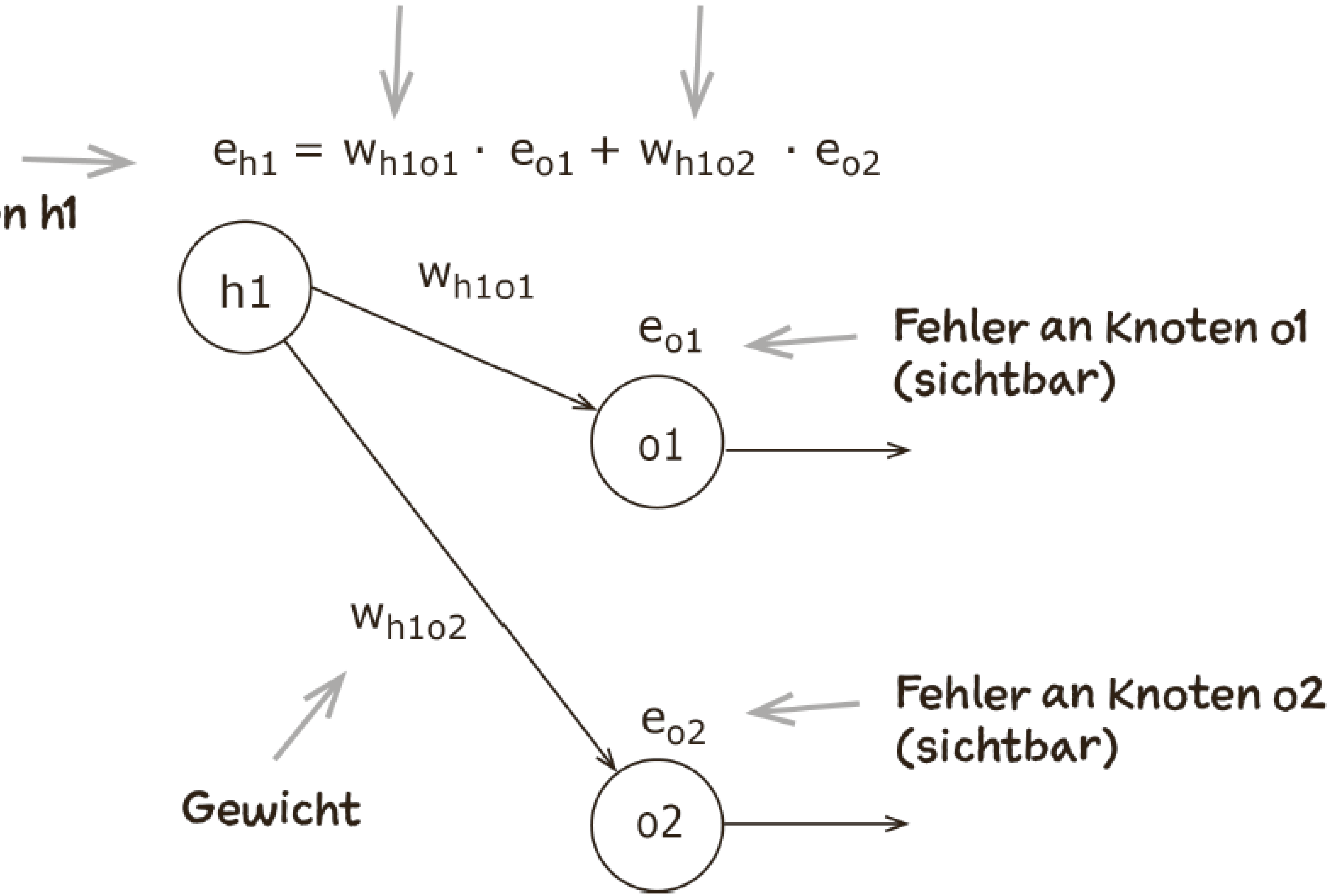
Signal, das gewichtet wird

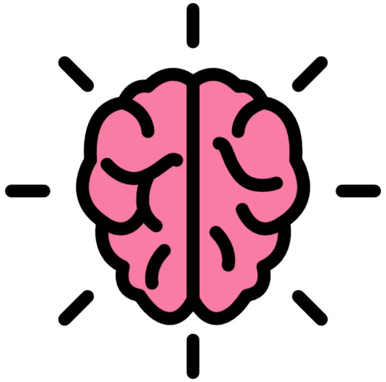
$$\Delta w_{i1h1} = \alpha \cdot e_{h1} \cdot y_{h1} \cdot (1 - y_{h1}) \cdot i_1$$

?

Gewichte werden in Rückwärtsrichtung verwendet

Fehler an verborgenem Knoten h1 (unsichtbar)





Aktualisierung der Gewichte

Ausgabe des Knotens, der das Signal empfängt

Signal, das gewichtet wird

Fehler am Knoten, der das Signal empfängt

$$e_{h1} = w_{h1o1} \cdot e_{o1} + w_{h1o2} \cdot e_{o2}$$

$$e_{h2} = w_{h2o1} \cdot e_{o1} + w_{h2o2} \cdot e_{o2}$$

$$e_{h3} = w_{h3o1} \cdot e_{o1} + w_{h3o2} \cdot e_{o2}$$

$$\Delta w_{i1h1} = \alpha \cdot e_{h1} \cdot y_{h1} \cdot (1 - y_{h1}) \cdot i_1$$

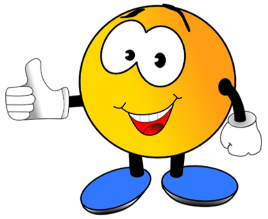
$$\Delta w_{i1h2} = \alpha \cdot e_{h2} \cdot y_{h2} \cdot (1 - y_{h2}) \cdot i_1$$

$$\Delta w_{i1h3} = \alpha \cdot e_{h3} \cdot y_{h3} \cdot (1 - y_{h3}) \cdot i_1$$

$$\Delta w_{i2h1} = \alpha \cdot e_{h1} \cdot y_{h1} \cdot (1 - y_{h1}) \cdot i_2$$

$$\Delta w_{i2h2} = \alpha \cdot e_{h2} \cdot y_{h2} \cdot (1 - y_{h2}) \cdot i_2$$

$$\Delta w_{i2h3} = \alpha \cdot e_{h3} \cdot y_{h3} \cdot (1 - y_{h3}) \cdot i_2$$



XOR-Detektor

Zufallszahlen (float) zwischen –W und + W

```
from random import uniform, shuffle
from math import e
W = 0.5  # obere Grenze für Anfangswerte der
Gewichte
LR = 0.2

# Initialisierung der Gewichte
wi1h1 = uniform(-W, W)
wi2h1 = uniform(-W, W)
wi1h2 = uniform(-W, W)
wi2h2 = uniform(-W, W)
wi1h3 = uniform(-W, W)
wi2h3 = uniform(-W, W)
wh1o1 = uniform(-W, W)
wh2o1 = uniform(-W, W)
wh3o1 = uniform(-W, W)
wh1o2 = uniform(-W, W)
wh2o2 = uniform(-W, W)
wh3o2 = uniform(-W, W)

def sig(x):
    return 1.0 / (1.0 + e**-x)
```

XOR-Detektor

Das neuronale Netz arbeitet: Aus der Eingabe wird die Ausgabe berechnet

```
def vorhersagen(i1,i2):
    xh1 = wi1h1 * i1 + wi2h1 * i2
    xh2 = wi1h2 * i1 + wi2h2 * i2
    xh3 = wi1h3 * i1 + wi2h3 * i2
    yh1 = sig(xh1)
    yh2 = sig(xh2)
    yh3 = sig(xh3)
    xo1 = yh1 * wh1o1 + yh2 * wh2o1 + yh3 * wh3o1
    xo2 = yh1 * wh1o2 + yh2 * wh2o2 + yh3 * wh3o2
    o1 = sig(xo1)
    o2 = sig(xo2)
    return o1, o2
```

XOR-Detektor

Globale Variablen für die Gewichte

```
def trainieren(i1, i2, t1, t2):
    global wi1h1, wi2h1, wi1h2, wi2h2, wi1h3, wi2h3
    global wh1o1, wh2o1, wh3o1, wh1o2, wh2o2, wh3o2
    # Berechnung der Ausgabe (Vorhersage)
    xh1 = wi1h1 * i1 + wi2h1 * i2
    xh2 = wi1h2 * i1 + wi2h2 * i2
    xh3 = wi1h3 * i1 + wi2h3 * i2
    yh1 = sig(xh1)
    yh2 = sig(xh2)
    yh3 = sig(xh3)
    xo1 = yh1 * wh1o1 + yh2 * wh2o1 + yh3 * wh3o1
    xo2 = yh1 * wh1o2 + yh2 * wh2o2 + yh3 * wh3o2
    o1 = sig(xo1)
    o2 = sig(xo2)
    # Aktualisierung der Gewichte
    # Verborgene Schicht - Ausgabeschicht
    eo1 = t1 - o1
    eo2 = t2 - o2
    wh1o1 += LR * eo1 * o1*(1-o1)*yh1
    wh2o1 += LR * eo1 * o1*(1-o1)*yh2
    wh3o1 += LR * eo1 * o1*(1-o1)*yh3
    wh1o2 += LR * eo2 * o2*(1-o2)*yh1
    wh2o2 += LR * eo2 * o2*(1-o2)*yh2
    wh3o2 += LR * eo2 * o2*(1-o2)*yh3
    # Eingabeschicht - verborgene Schicht
    eh1 = wh1o1 * eo1 + wh1o2 * eo2
    eh2 = wh2o1 * eo1 + wh2o2 * eo2
    eh3 = wh3o1 * eo1 + wh3o2 * eo2
    wi1h1 += LR * eh1 * yh1*(1-yh1)*i1
    wi2h1 += LR * eh1 * yh1*(1-yh1)*i2
    wi1h2 += LR * eh2 * yh2*(1-yh2)*i1
    wi2h2 += LR * eh2 * yh2*(1-yh2)*i2
    wi1h3 += LR * eh3 * yh3*(1-yh3)*i1
    wi2h3 += LR * eh3 * yh3*(1-yh3)*i2
    return eo1, eo2
```

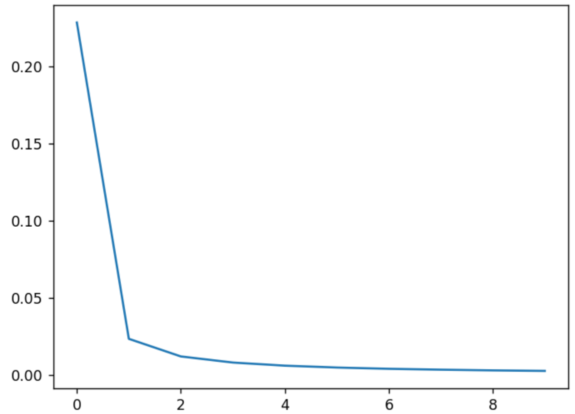
XOR-Detektor

Liste mit 8000 Tupeln

```
# Zufällige Trainingsdaten erzeugen
d =[(0, 0, 0, 1), (0, 1, 1, 0), (1, 0, 1, 0), (1, 1, 0, 1)]
daten = 2000 * d
shuffle(daten)

# Training
for ep in range(10):
    summeFehlerQuadrate = 0
    for i1, i2, t1, t2 in daten:
        eo1, eo2 = trainieren(i1, i2, t1, t2)
        summeFehlerQuadrate += eo1**2 + eo2**2
    mFehlerQuadrate = summeFehlerQuadrate / len(daten)
    print('Epoche:',ep, 'mittlere Fehlerquadratsumme:',
          mFehlerQuadrate)
# Testen
for i1, i2, t1, t2 in d:
    o1, o2 = vorhersagen(i1, i2)
    print('Eingabe:', i1, i2, 'Vorhersage: ,,
           round(o1, 4), round(o2, 4),
          'Target:', t1, t2)
```

Übung 6.2 XOR-Detektor (10 min)

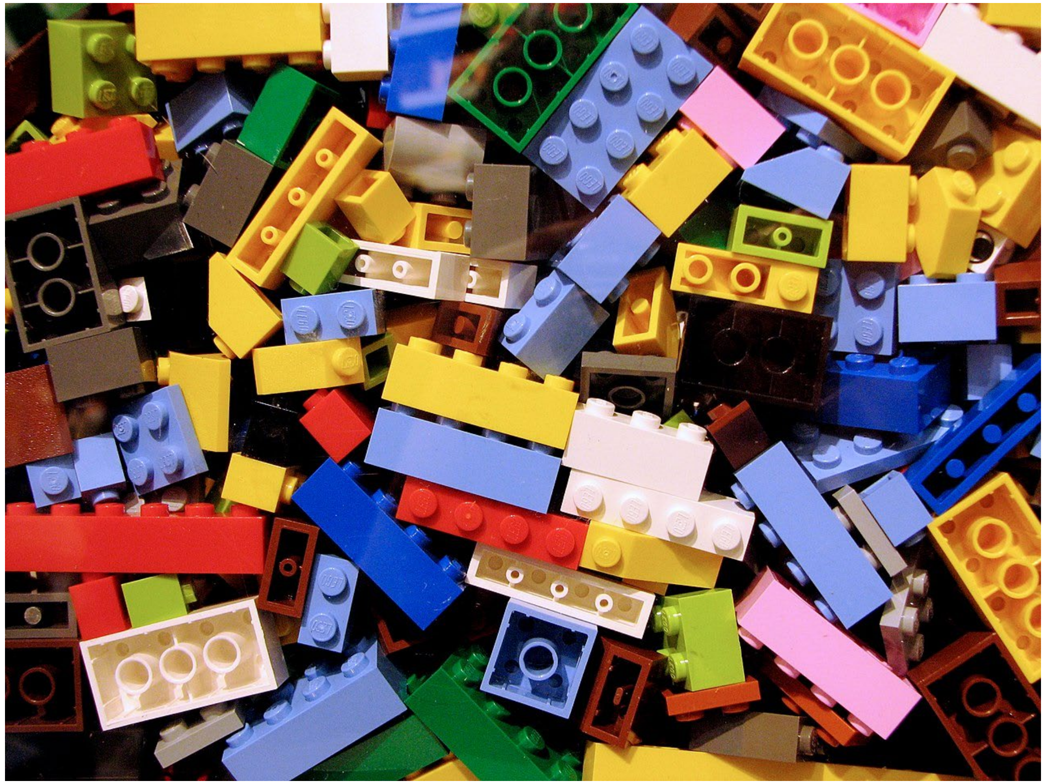
1. Testen Sie das Starterprojekt xor.detektor.py.
2. Experimentieren Sie mit dem Programm. Ändern Sie das Programm ab, beobachten Sie die Auswirkung und versuchen Sie die Auswirkungen zu erklären:
   - Lernrate auf 0,1 setzen
   - W verkleinern, sodass die zufälligen Anfangswerte für die Gewichte aus einem kleineren Intervall kommen, z.B. (-0.2, +0.2).
   - Anfangswerte für die Gewichte auf 0 setzen (W=0).
3. Ändern Sie die Trainingsdaten ab, sodass das Programm ODER erkennt.
4. Ändern Sie das Programm ab, dass es eine Lernkurve zeigt.
5. Überlegen Sie sich Ideen für weitere Aktivitäten mit dem Starterprojekt (Google Docs).



https://docs.google.com/document/d/140lNslEWA_AwUwtpVMCZqtH7_eOkU8rmvfrgWqE5M3E/edit?usp=sharing

Give your brain a hand - Visualisieren mit Lego

- Alles ist richtig!
- Wenn Ihnen nichts einfällt, beginnen Sie einfach und bauen Sie irgendetwas!



Übung 6.3 Visualisieren mit Lego

Dreier- oder Vierergruppe (ca. 10 min)

- Neuron
- Synapse
- Ausgabefehler (Unterschied zwischen tatsächlichem Wert und erwartetem Wert)
- Fehler an einem verborgenen Knoten
- Aktivierung eines Neurons
- Feuern eines Neurons
- Erregungspotenzial eines Neurons
- Eingabeknoten
- Ausgabeknoten
- Gewicht einer Verbindung
- Training
- Lernrate

Spielregeln:

1. Jede Person wählt einen Begriff aus der Liste. Verraten Sie nicht, was Sie gewählt haben.
2. Bauen Sie eine Lego-Skulptur, die diesen Begriff in irgendeiner Form darstellt. (5 min)
3. Reihum stellt jede Person ihre Skulptur vor:
   1. Zuerst raten die anderen, was es ist.
   2. Dann wird die Skulptur erklärt.

Rückblick

- Nur ein neuronales Netz mit einer **verborgenen Schicht** kann Punktmengen erkennen, die nicht linear separierbar sind (z.B. XOR).
- Als Aktivierungsfunktion verwendet man eine Sigmoid-Funktion. Sie liefert Werte zwischen 0 und 1. Sie ist glatt und überall differenzierbar.
- Beim **Gradientenverfahren** verwendet man als Fehlerfunktion die Summe der Fehlerquadrate. Der Funktionswert ist immer positiv und wird bei kleinen Fehlerwerten besonders klein.
- Beim **Training** eines neuronalen Netzes ändert man die Gewichte der Verbindungen zwischen den Knoten. Man ändert ein Gewicht so, dass der Fehler, soweit seine Änderung durch das Ändern des Gewichts beeinflusst wird, kleiner wird. Das ist die Grundidee des **Gradientenverfahrens**. Es ist vergleichbar mit einem Bergwanderer, der immer bergab geht, um ins Tal zu kommen.
- Direkt beobachten kann man nur die **Ausgabefehler** des neuronalen Netzes (Differenz zwischen erwartetem Wert t und berechnetem Wert o).
- Durch **Error Backpropagation** (Fehlerrückführung) kann man auch den verborgenen Knoten Fehler zuordnen.