



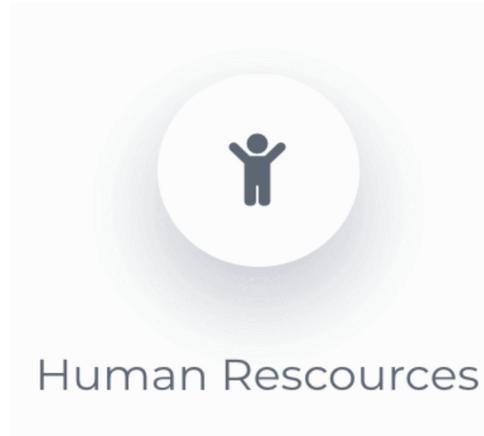
DRAFT

ML + ANN

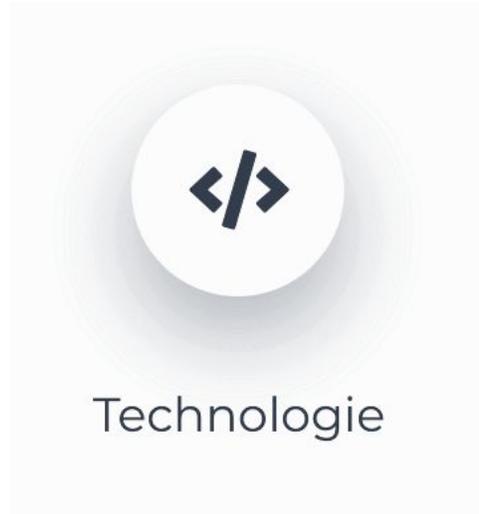
Psychologie 2021
Dr. Roland Schwaiger



NoR Business Units



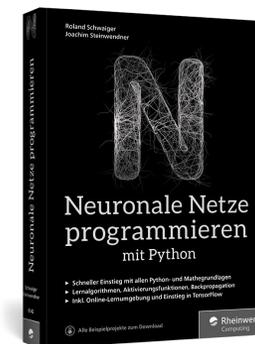
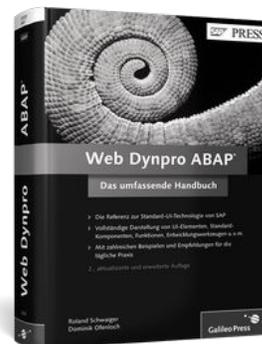
Human Resources



Technologie



Künstliche Neuronale Netze
& Evolutionäre Algorithmen





Agenda

1. Einleitung
2. Starter Kit
3. Ein einfaches neuronales Netz
4. Lernen im einfachen Netz
5. Mehrschichtige neuronale Netze
6. Lernen im mehrschichtigen Netz
7. Convolutional Neural Networks (CNN)
8. Programmieren von CNN mit TensorFlow2 and Keras
9. (Vom Hirn zum Netz)
10. Die Evolution der neuronalen Netze
11. (Der Machine-Learning-Prozeß) ? Feature aspects
12. Lernverfahren
13. Anwendungsgebiete und Praxisbeispiele



Erste Schritte

- Download Material von <https://www.rheinwerk-verlag.de/neuronale-netze-programmieren-mit-python/>
- Für Kapitel 8:
 - Öffnen und ausführen Kapitel_08-InstallBibliotheken
 - “...mit conda” hat bei mir nicht funktioniert (Mac + alte TensorFlow Lib)
 - “...mit pip” hat funktioniert



Kapitel 1

Einleitung



1.4 Ist diese Biene eine Königin?

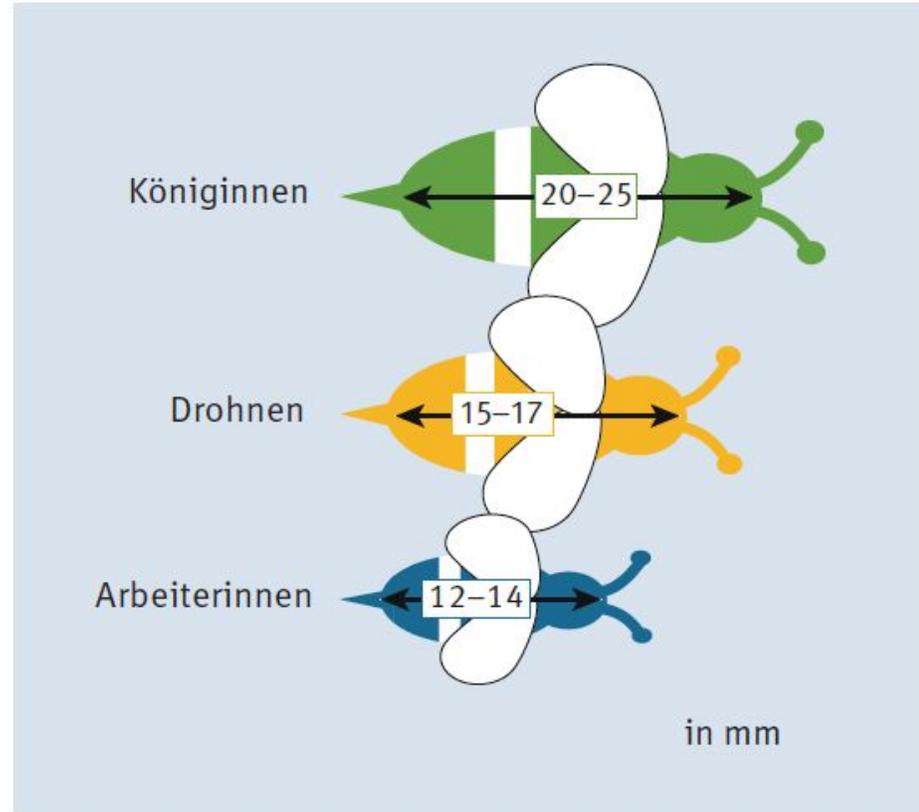


Abbildung 1.1 Die Bienenklassifikation



1.5 Ein künstliches neuronales Netz für den Bienenstaat

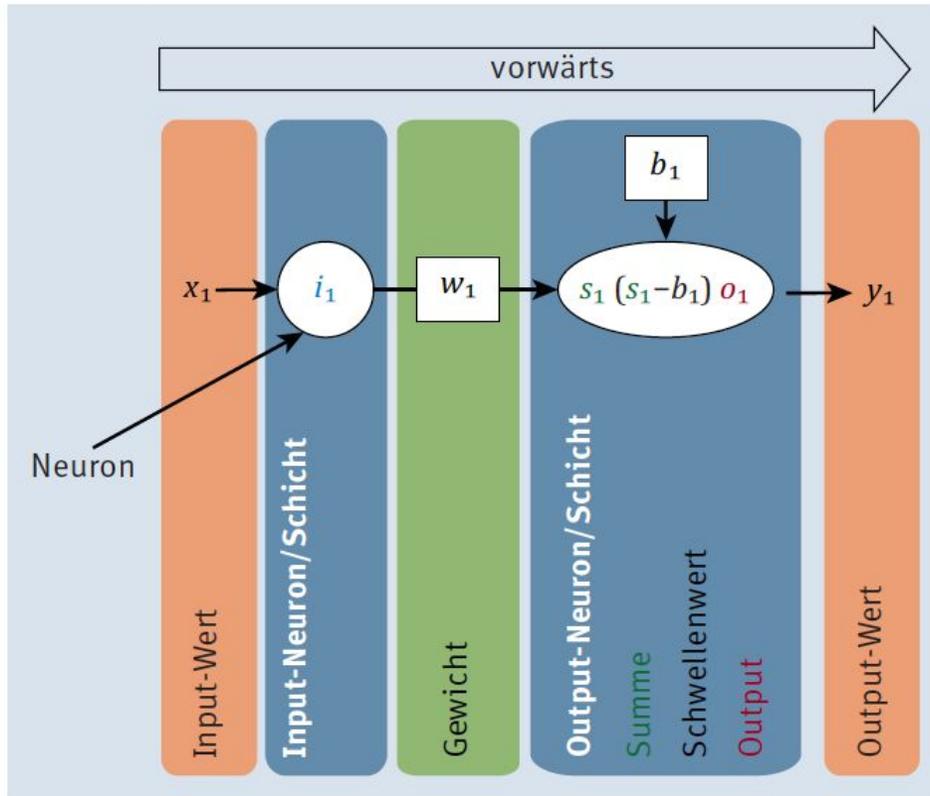


Abbildung 1.2 Ein einfaches KNN

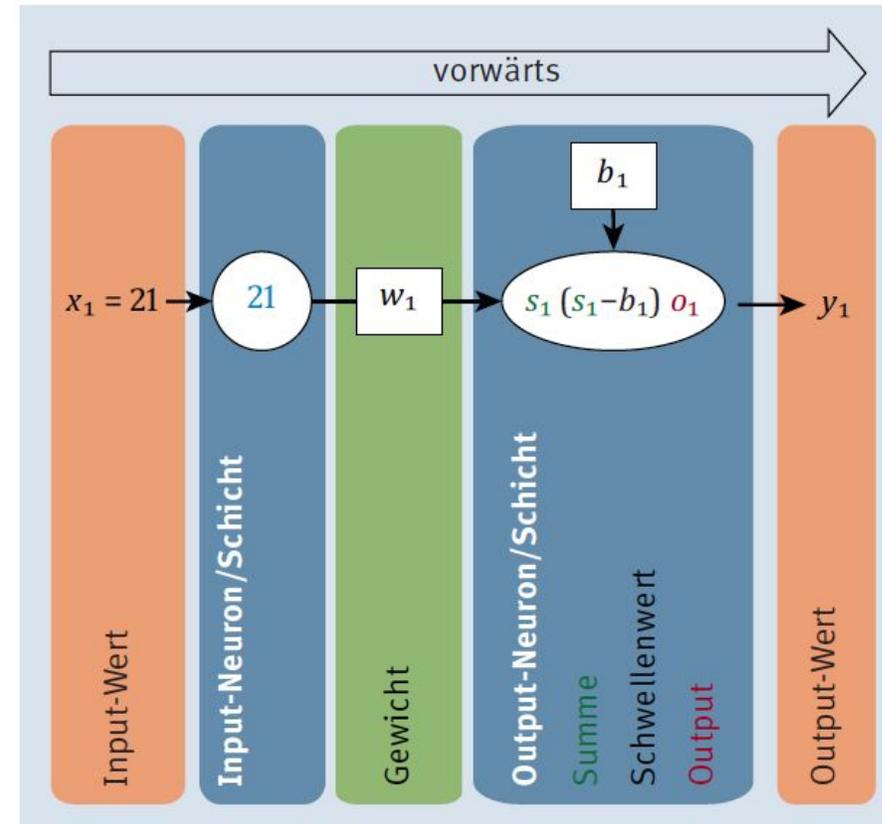


Abbildung 1.3 Der Input-Wert 21 wird an das Netz übergeben.



1.5 Ein künstliches neuronales Netz für den Bienenstaat

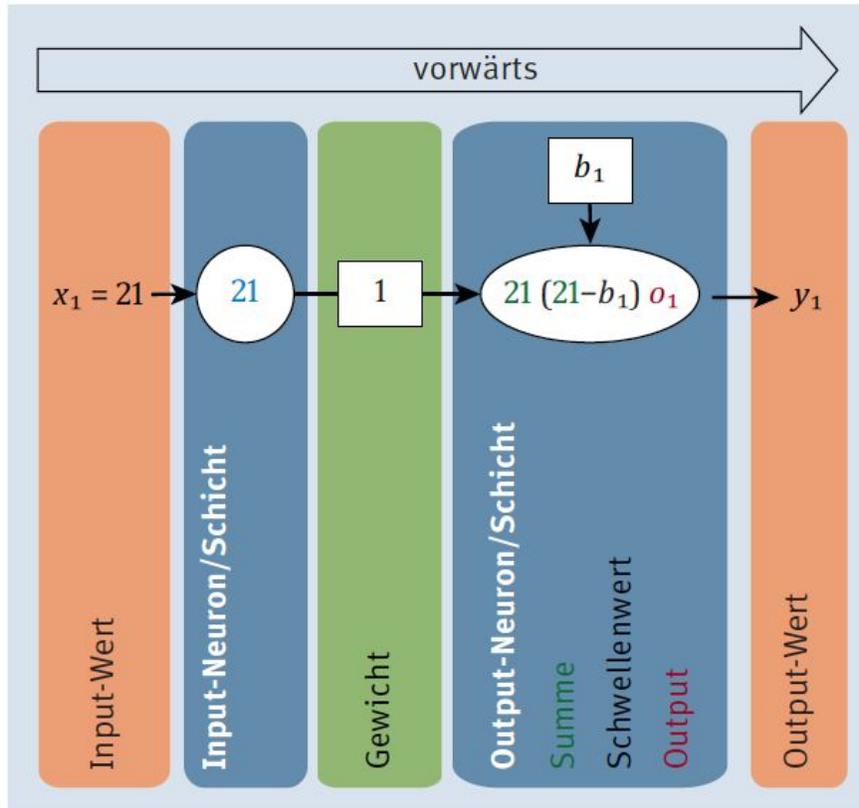


Abbildung 1.4 Der gewichtete Input zum Output-Neuron wird berechnet.

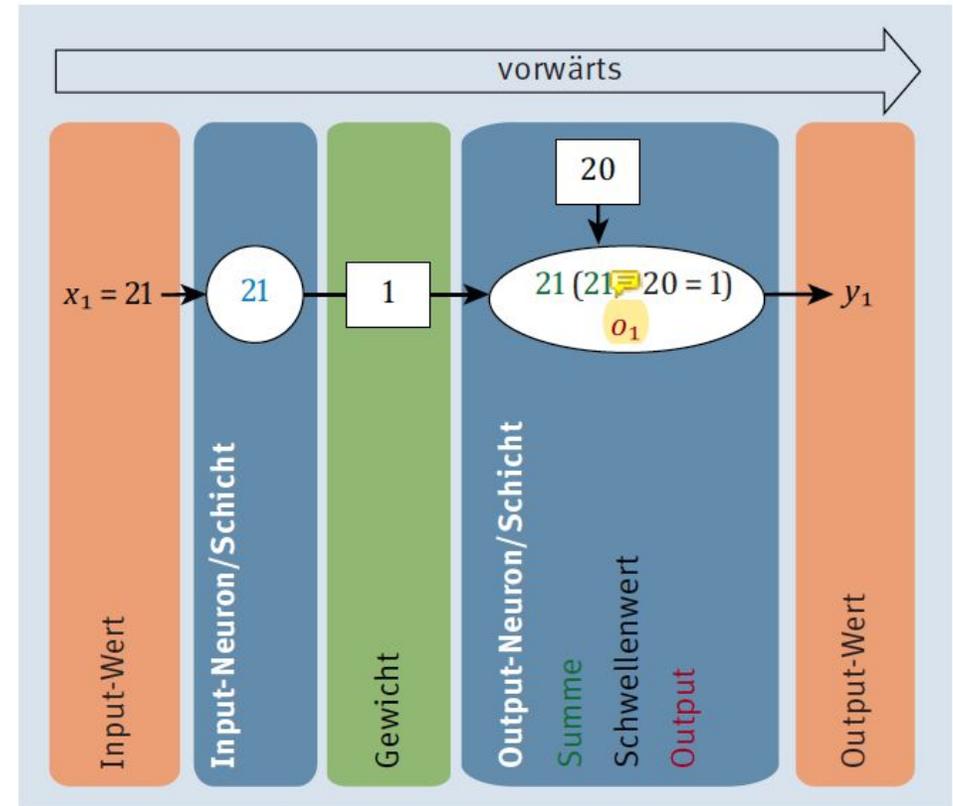


Abbildung 1.5 Schwellenwertberechnung



1.5 Ein künstliches neuronales Netz für den Bienenstaat

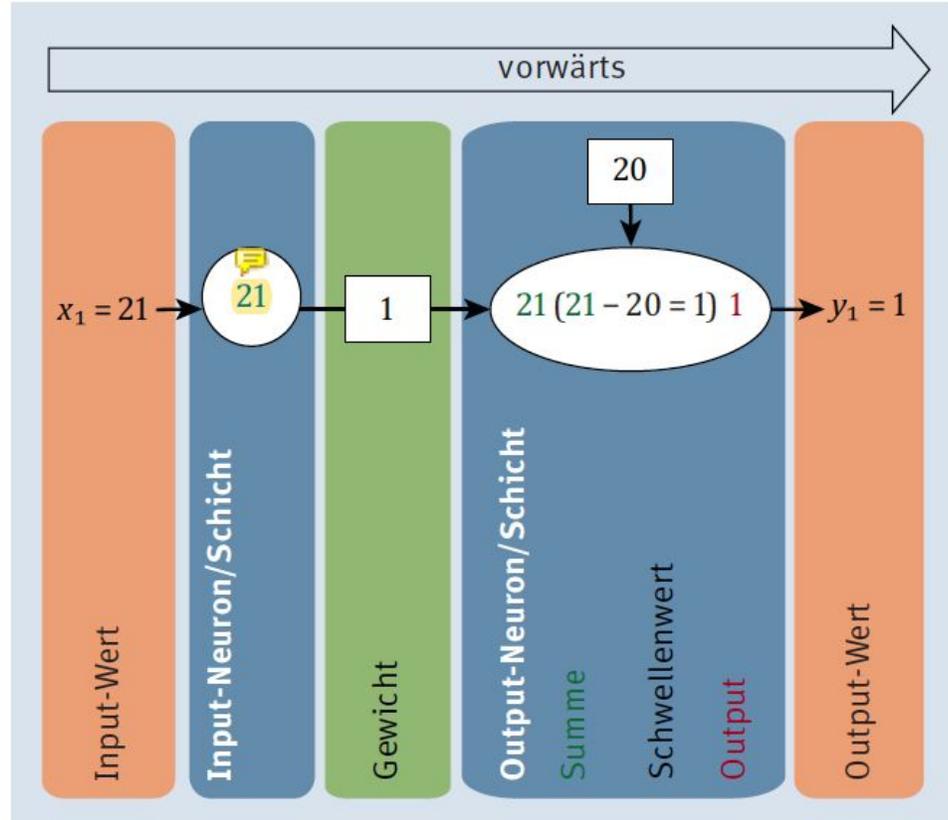


Abbildung 1.6 Output-Berechnung



1.6.1 Das biologische Neuron und seine technische Kopie

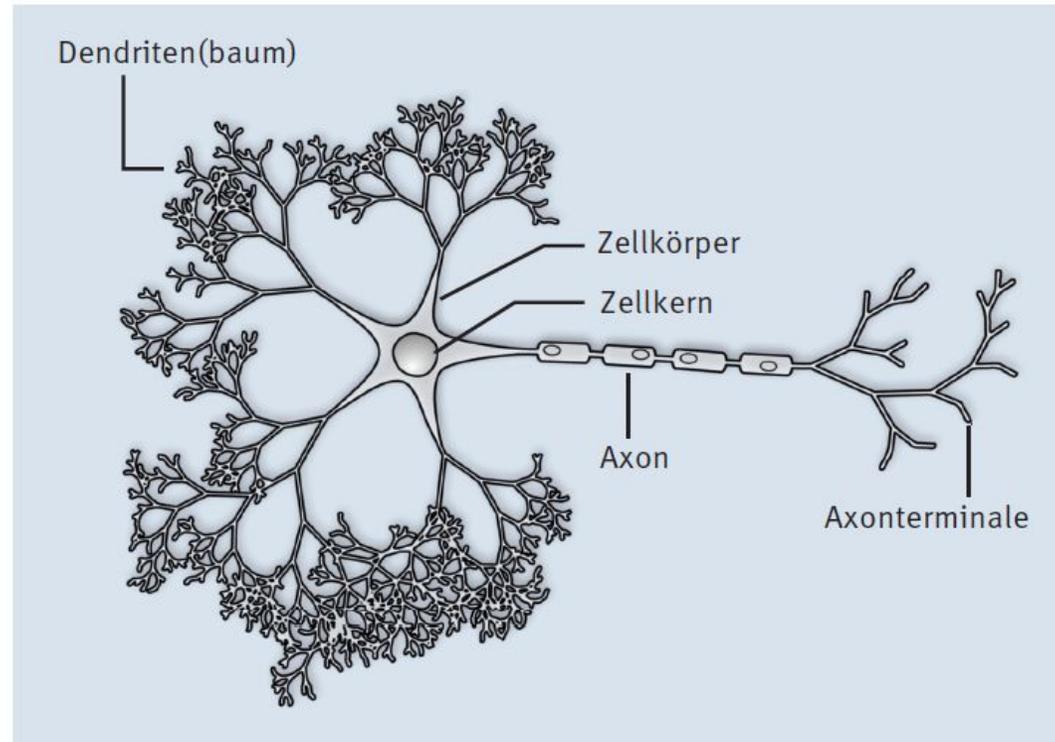


Abbildung 1.7 Schematische Darstellung eines Neurons (von Nicolas.Rougier/CC-BY-SA-3.0, <https://commons.wikimedia.org/w/index.php?curid=2192116>)



1.6.2 Das künstliche Neuron und seine Elemente

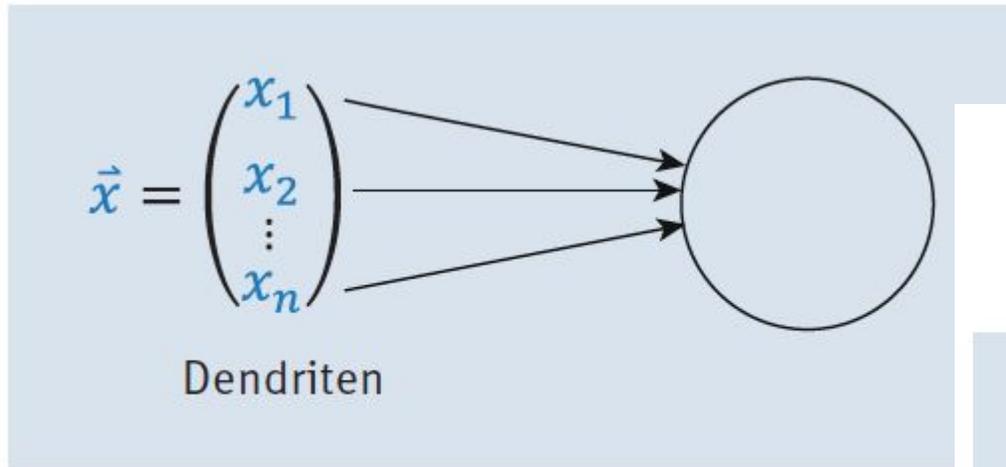


Abbildung 1.8 Künstliche Dendriten

$$\sum_{i=1}^n x_i \cdot w_i$$

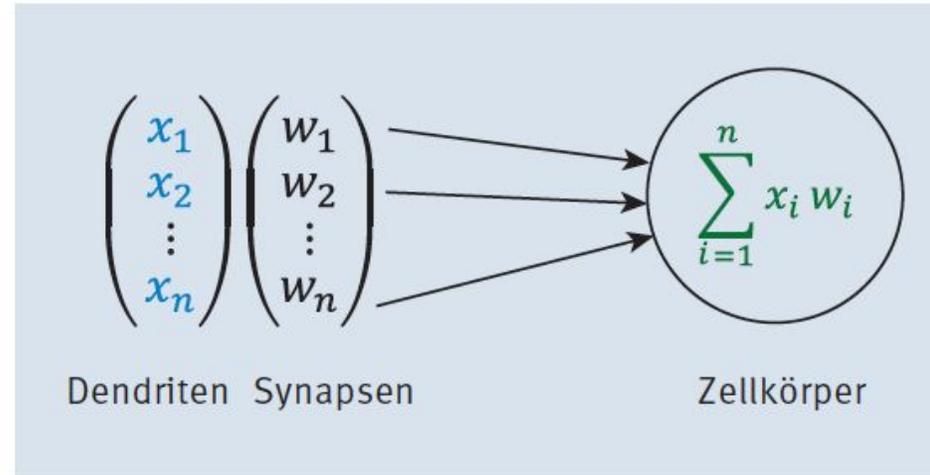


Abbildung 1.9 Künstliche Dendriten mit Synapsen(gewichten)



1.6.2 Das künstliche Neuron und seine Elemente

Activation Function

$$y = f_{\text{akt}} \left(\sum_{i=1}^n x_i w_i \right)$$

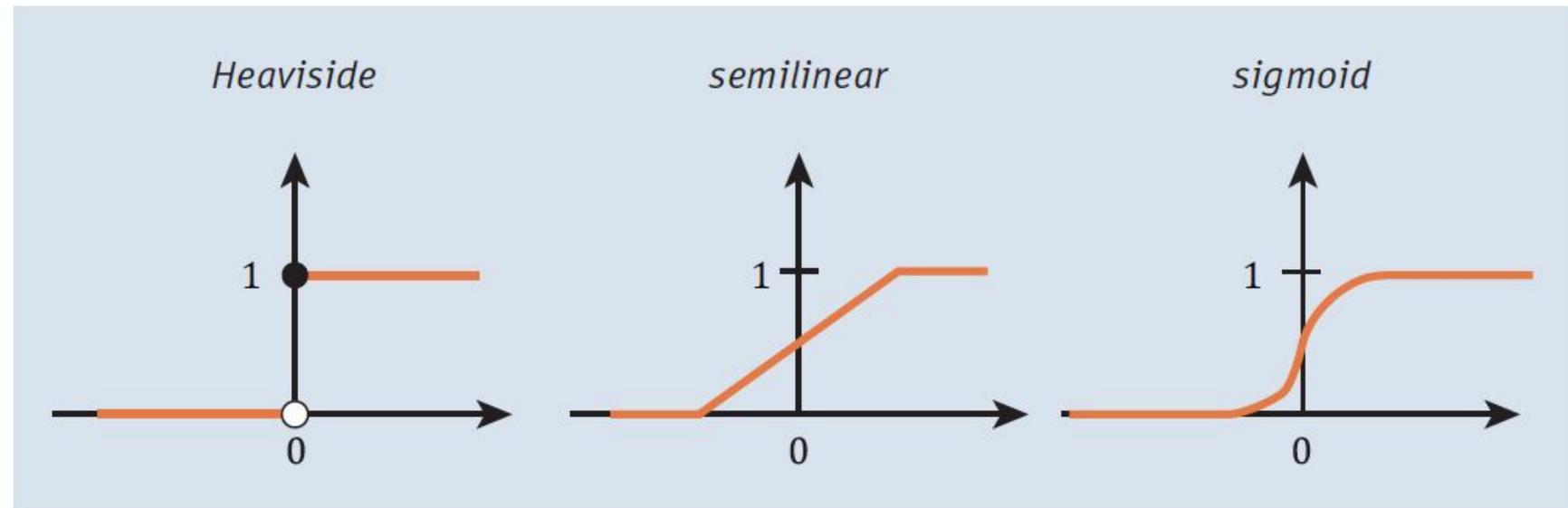


Abbildung 1.10 Beispiele für die Aktivierungsfunktion fakt



1.6.2 Das künstliche Neuron und seine Elemente Artificial Neuron

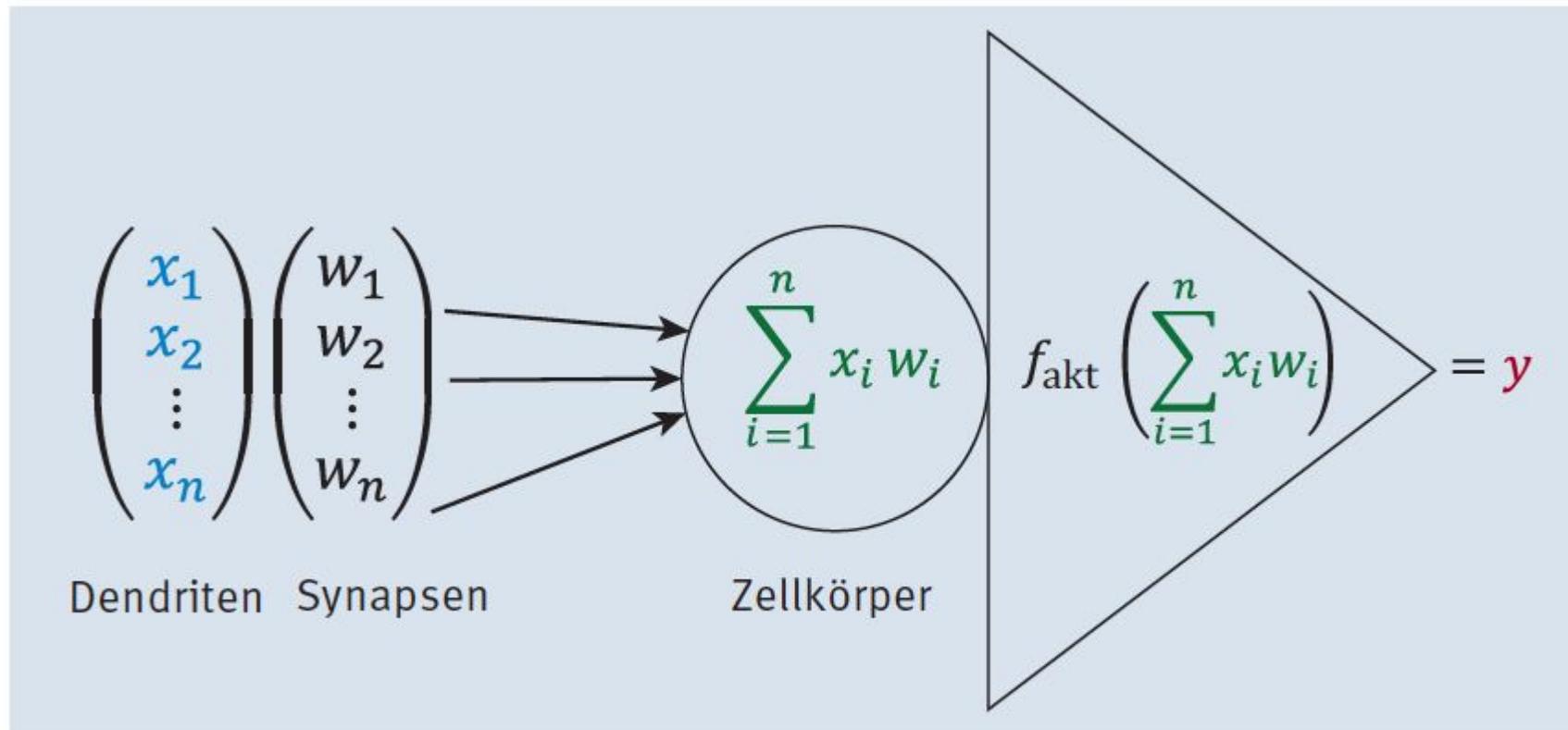


Abbildung 1.11 Das künstliche Neuron



1.7 Einordnung und der Rest

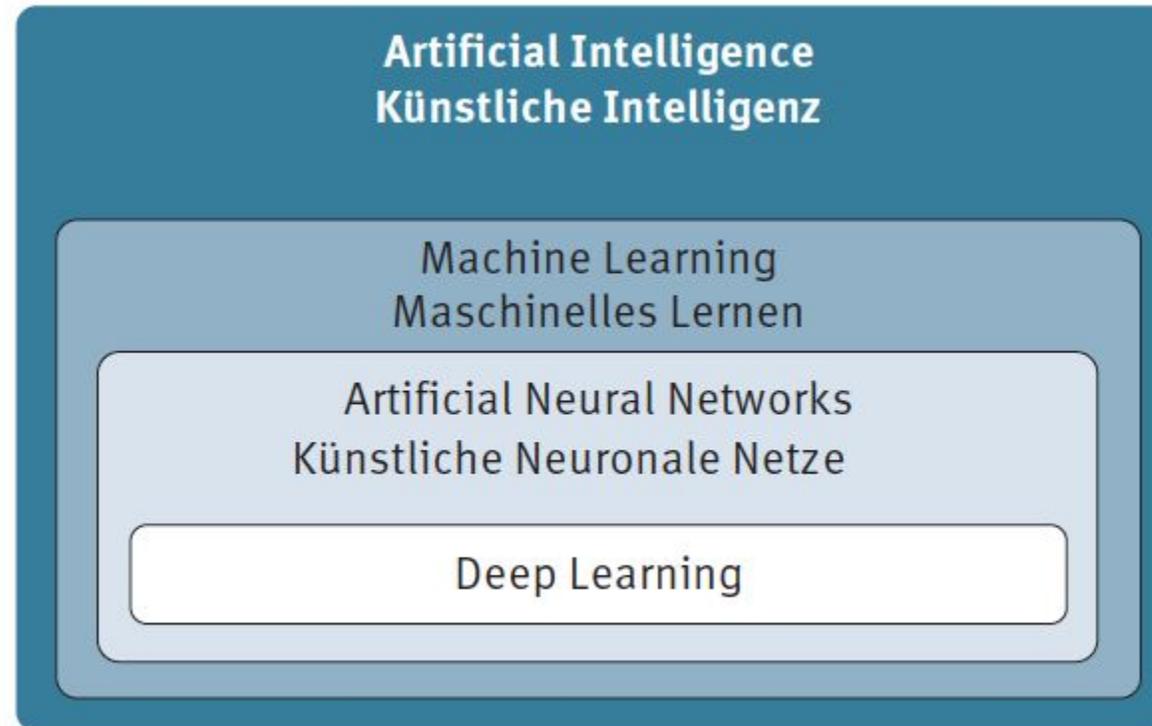


Abbildung 1.12 Begriffszwiebel für künstliche Intelligenz



1.7.4 Machine Learning (maschinelles Lernen)

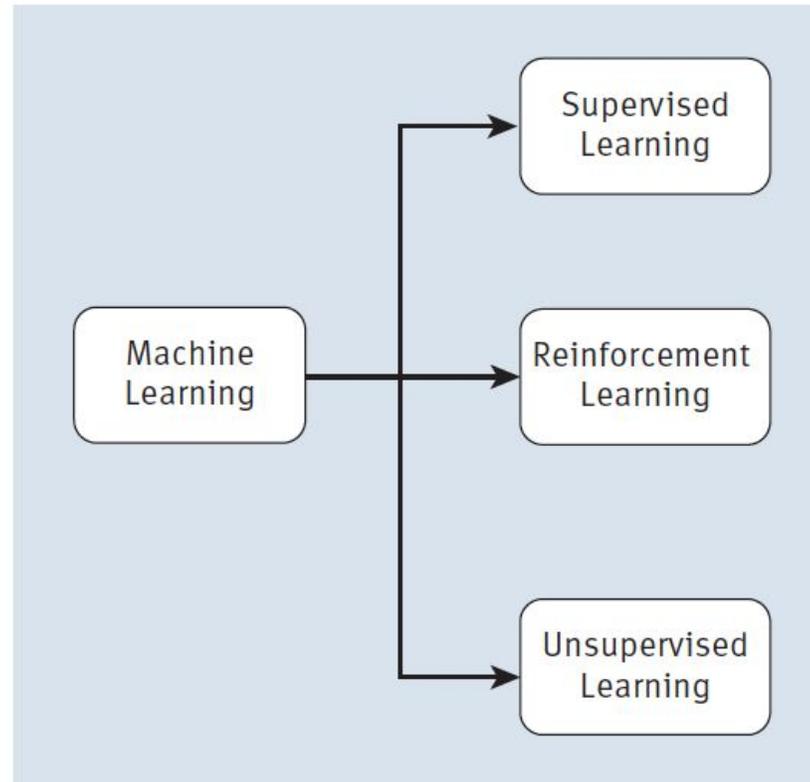


Abbildung 1.13 Lernstrategien im Machine Learning



1.7.5 Deep Neural Networks

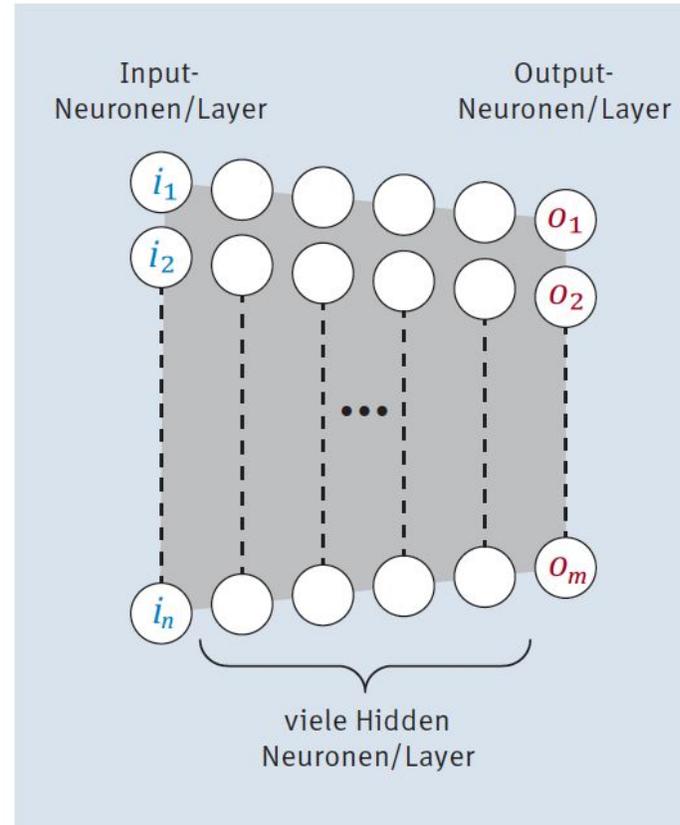


Abbildung 1.14 Deep-Neural-Network-Schema



Kapitel 2

Das minimale Starterkit für die Entwicklung von neuronalen Netzen mit Python



Ingredients

- Anaconda
- Jupyter Notebook
- Python
 - NumPy
 - matplotlib
 - scikit-learn
 - pandas
- + TensorFlow 2
- + Keras



Kapitel 3

Ein einfaches neuronales Netz



3.2 Her mit dem neuronalen Netz!

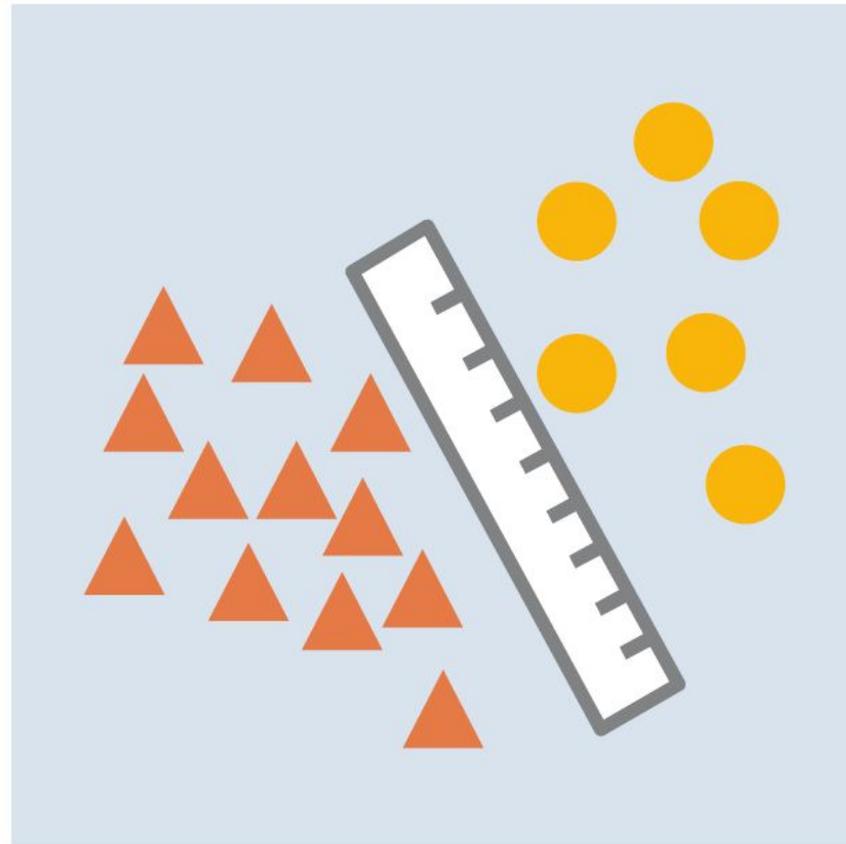


Abbildung 3.1 Linea(l/r)e Trennung



Frau Karotte	Herr Lauch	-->	Montag
nicht anwesend	nicht anwesend	-->	nicht ok
anwesend	nicht anwesend	-->	ok
nicht anwesend	anwesend	-->	ok
anwesend	anwesend	-->	ok

Tabelle 3.1 Erster Personalplan

Frau Karotte	Herr Lauch	-->	Montag
0	0	-->	0
1	0	-->	1
0	1	-->	1
1	1	-->	1

Tabelle 3.2 Zweiter Personalplan mit KNN-geeigneter Codierung

Frau Karotte	Berechnung	Herr Lauch	-->	Summe	Ergebnis
0	+	0	=	0	0
1	+	0	=	1	1
0	+	1	=	1	1
1	+	1	=	2	1

Tabelle 3.3 Errechneter Personalplan aus den Anwesenheiten

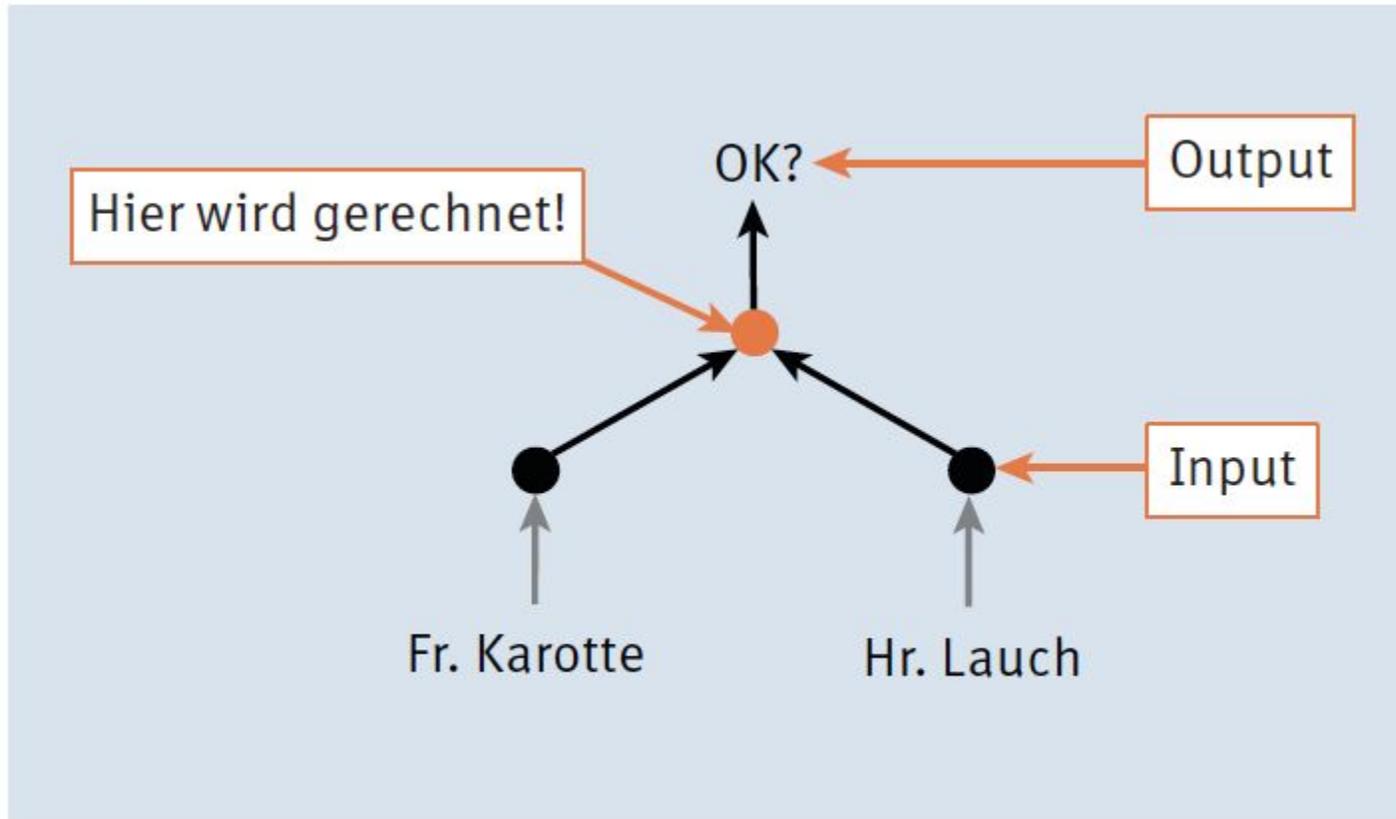


Abbildung 3.2 Die Entscheidung für Frau Karotte und Herrn Lauch berechnen



3.3 Neuron-Zoom-in

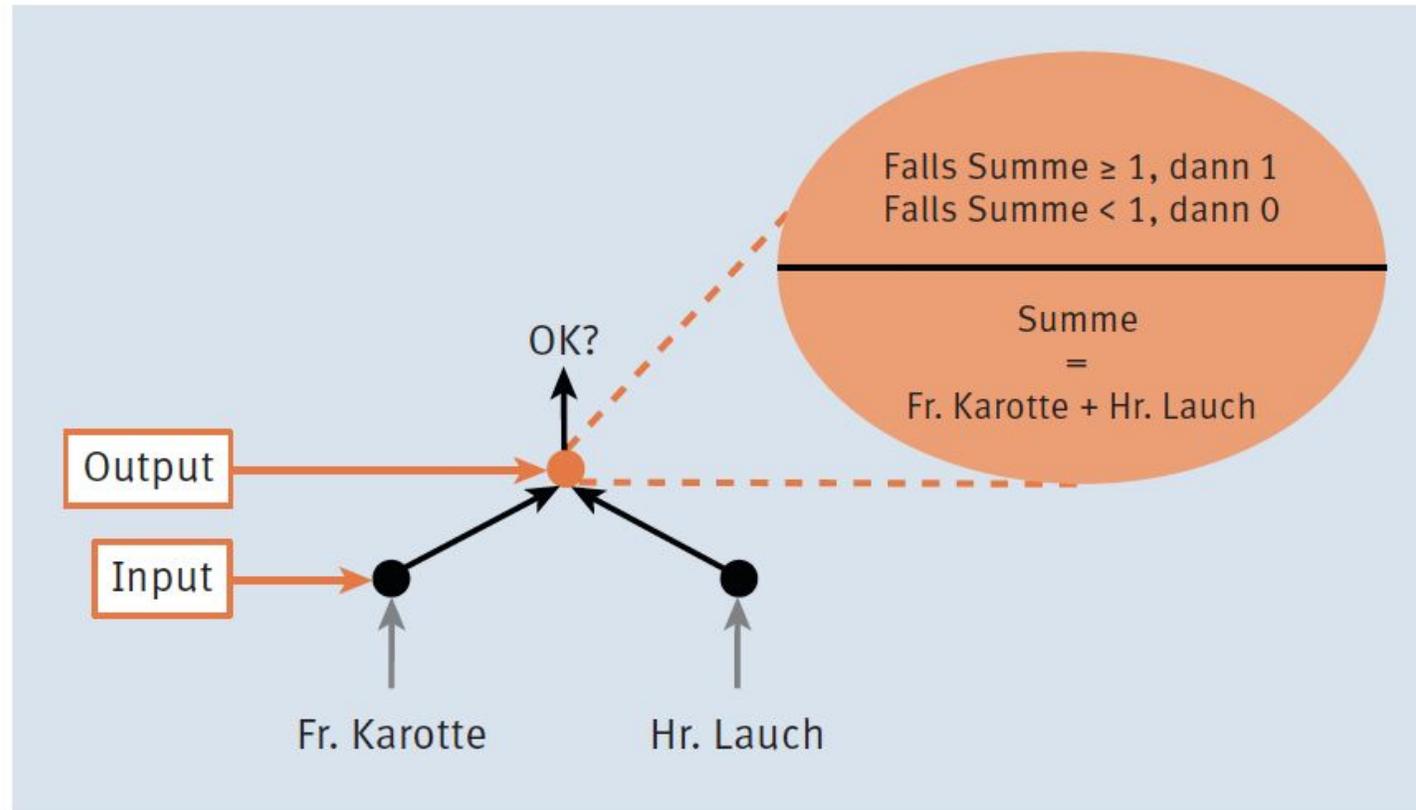


Abbildung 3.3 Ergebnisberechnung für Frau Karotte und Herrn Lauch im Detail



Listing 3.1 Die Entscheidung als Stufenfunktion (like heaviside function as seen before)





3.4 Stufenfunktion

Listing 3.2 Verwendung von »pyplot«





3.4 Stufenfunktion

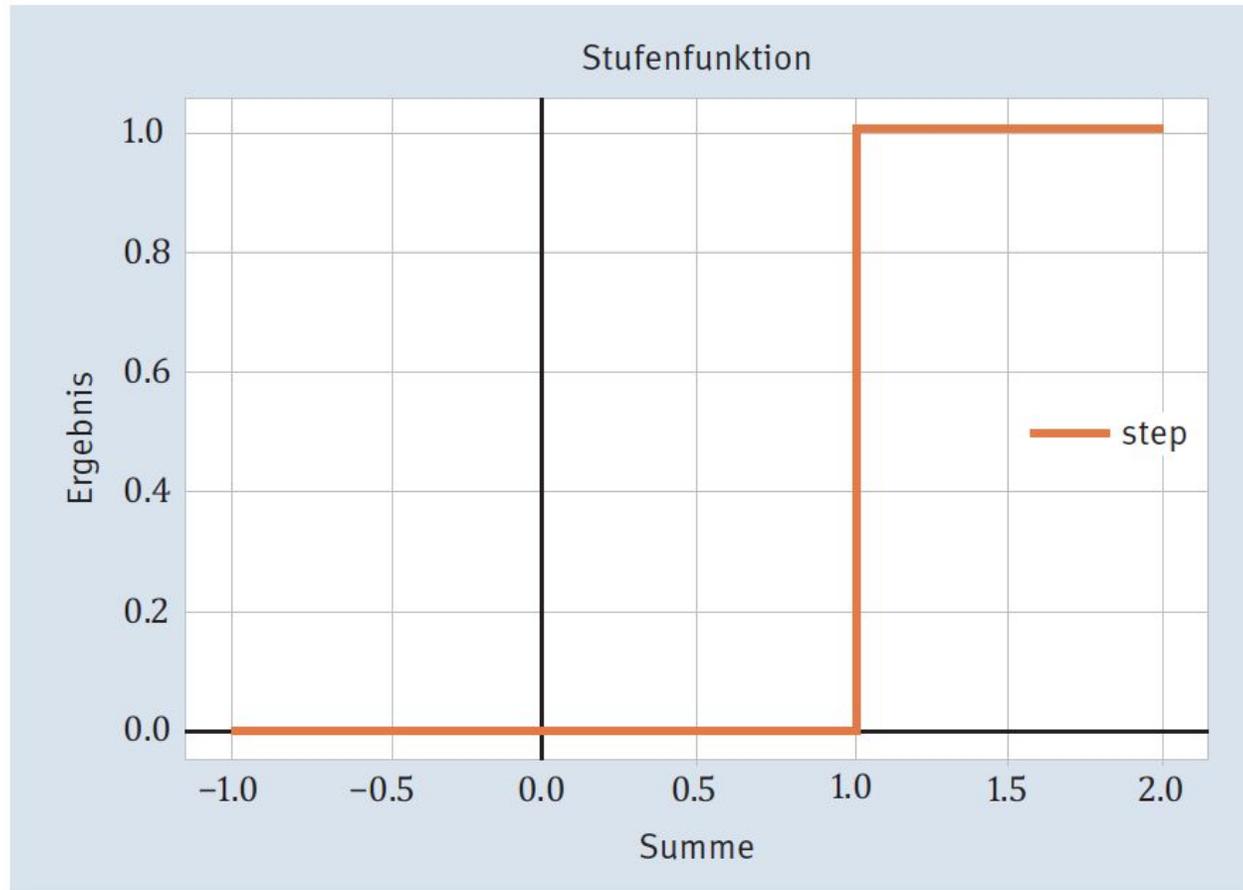


Abbildung 3.8 Stufenfunktion



3.5 Perceptron

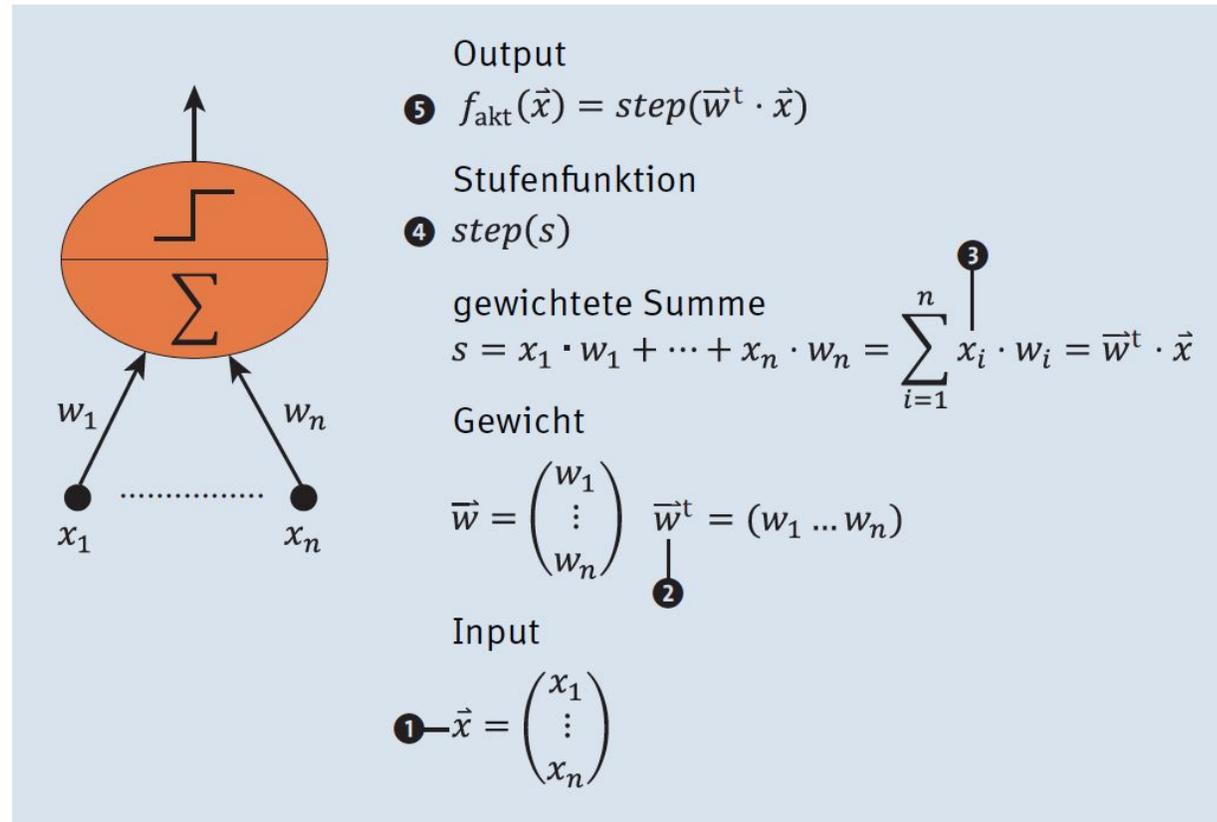


Abbildung 3.9 Perceptron – Bestandteile und Berechnungsbausteine



3.6 Punkte im Raum – Vektorrepräsentation

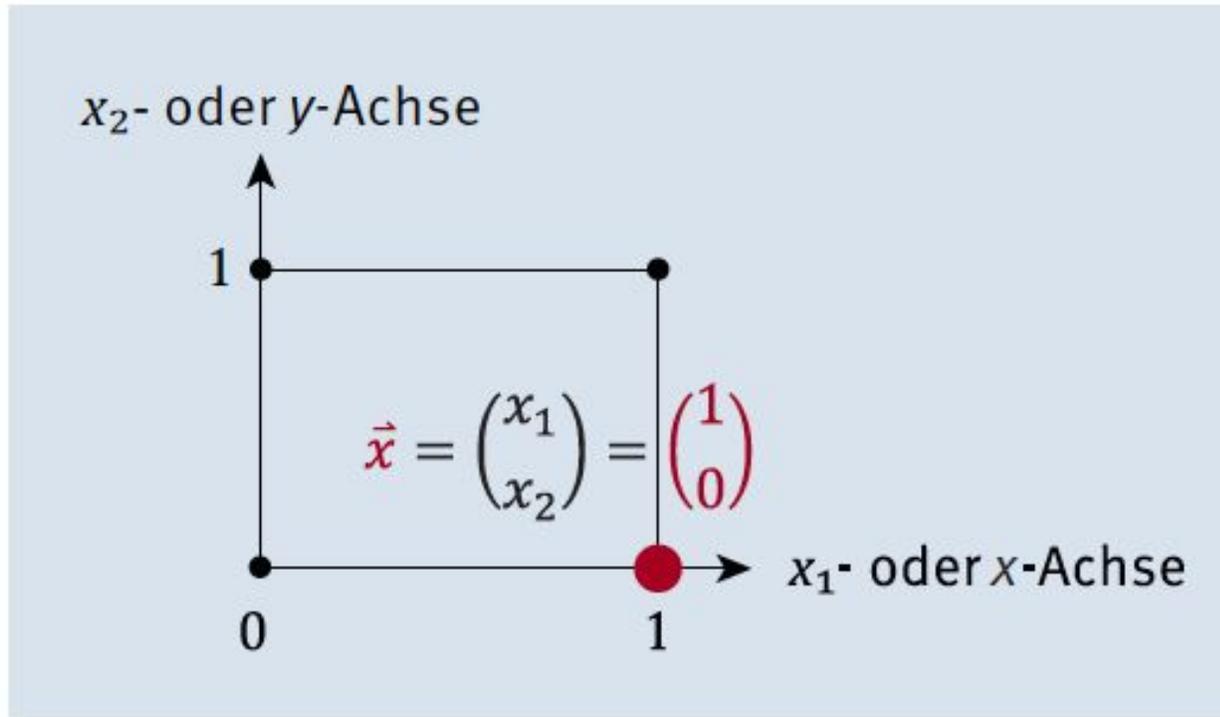


Abbildung 3.10 Punkt im kartesischen Koordinatensystem

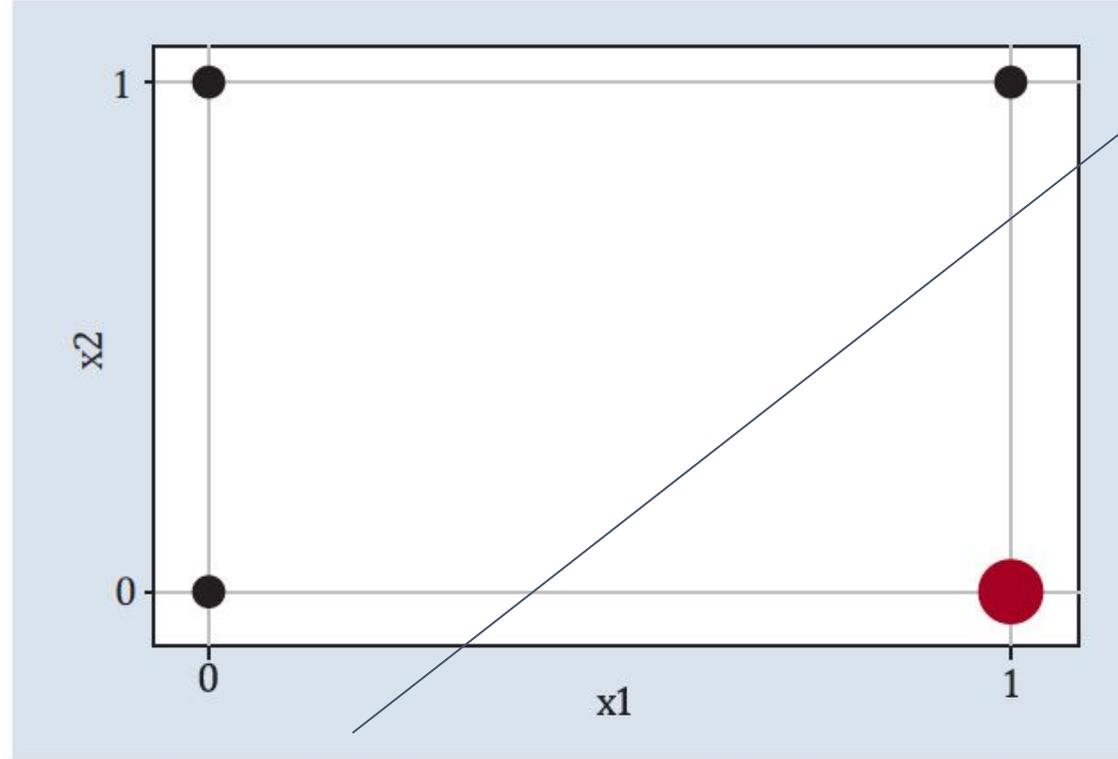


Abbildung 3.12 Scatter-Plot für die Planungspunkte



3.3 Scatter-Plot





Abbildung 3.13 Blattmaße für Schwertlilien (© Kaggle)

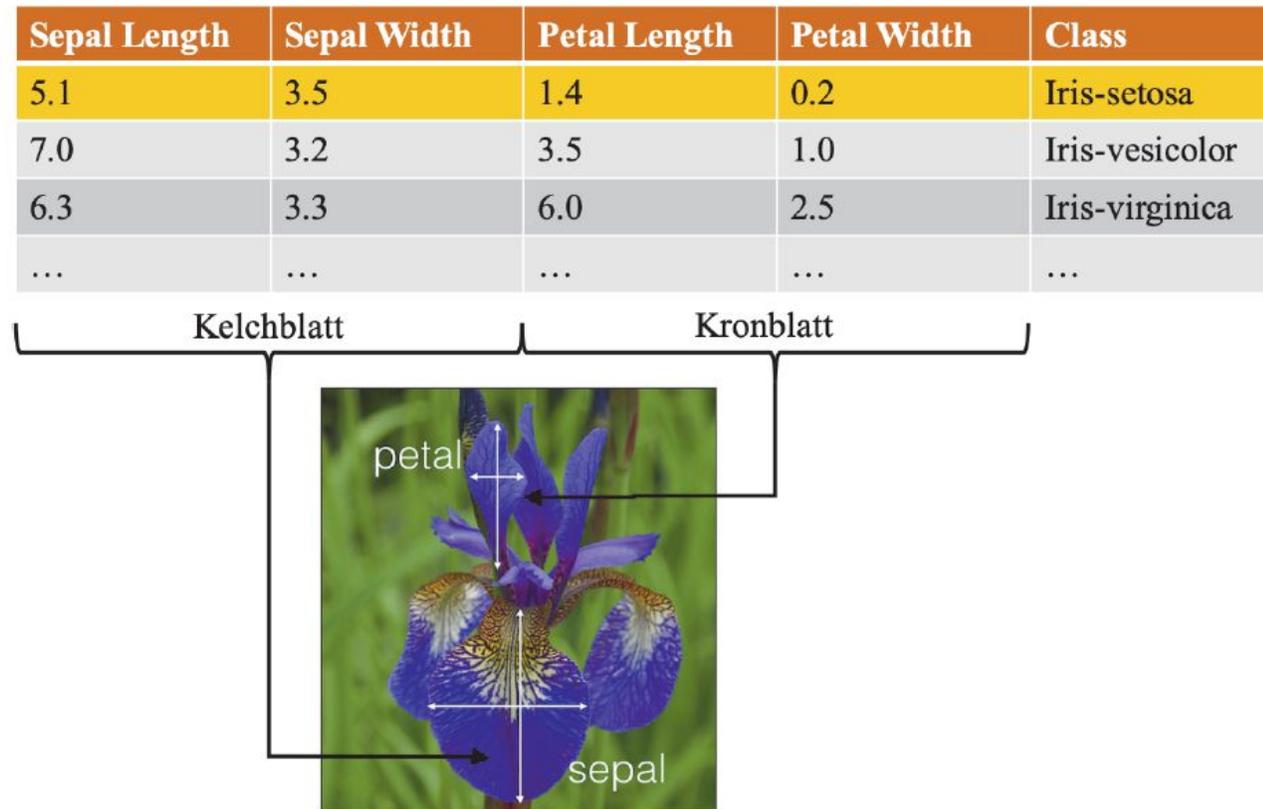


Abbildung 3.13 Blattmaße für Schwertlilien (© Kaggle)



Listing 3.4 Sepal Length und Sepal Width als Scatter-Plot





3.6.2 Aufgabe: Iris-Datensatz als Scatter-Plot ausgeben

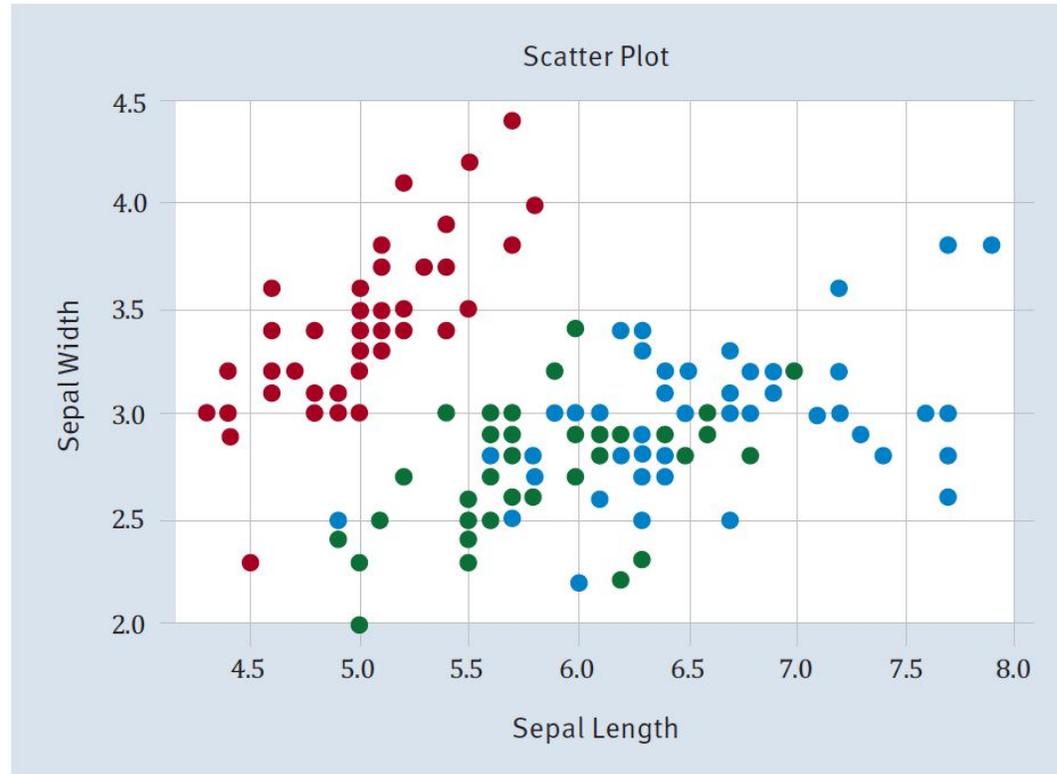


Abbildung 3.14 Die Lilien als Scatter-Plot mit den Koordinaten »Sepal Length« und »Sepal Width«



Scalar Product

$$\vec{w}^t \cdot \vec{x} = (w_1 w_2 \dots w_n) \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = w_1 \cdot x_1 + w_2 \cdot x_2 + \dots + w_n \cdot x_n$$



Listing 3.5 Skalarprodukt mit Python





3.10 Die gewichtete Summe reloaded

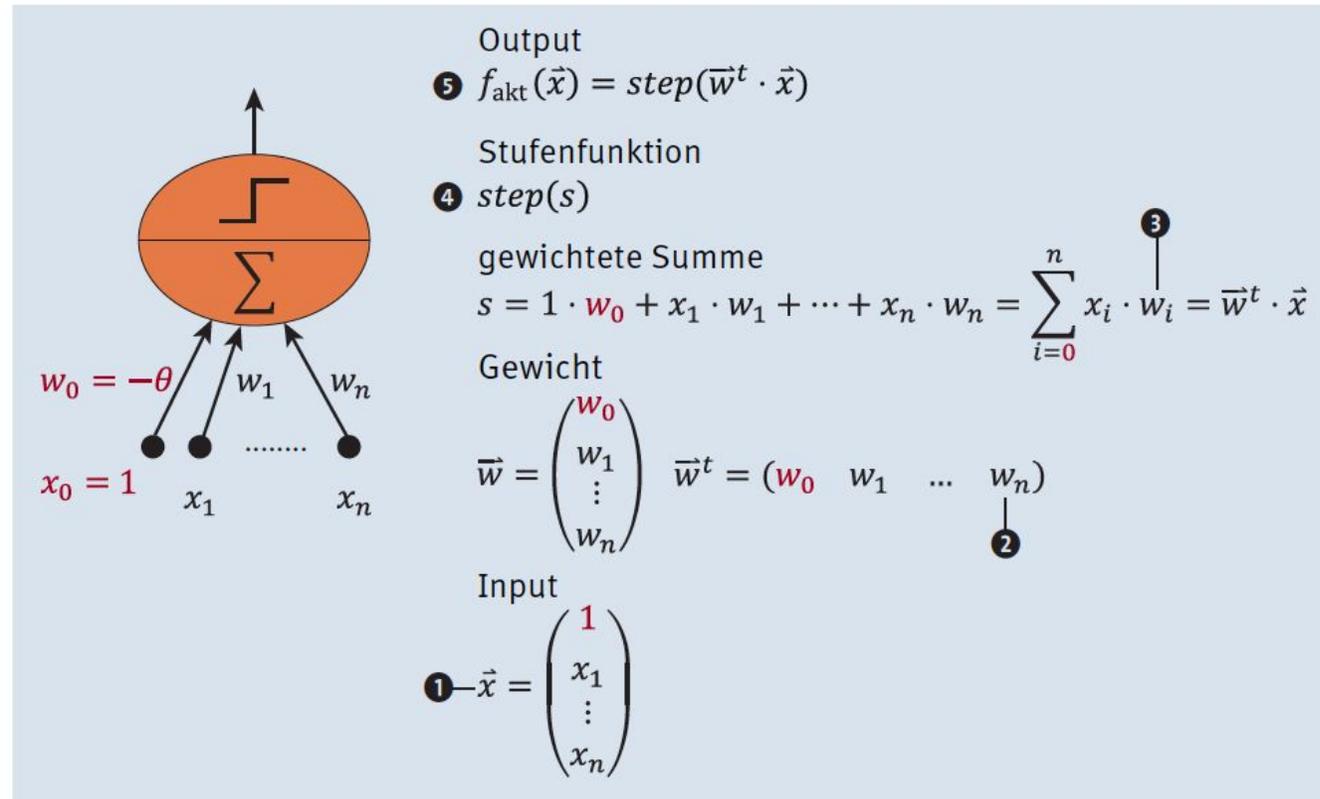


Abbildung 3.15 Bausteine des Perceptrons



3.11 Alles zusammen

Bias-Neuron	Frau Karotte	Herr Lauch	-->	Montag	Input-Vektor
1	0	0	-->	0	1
1	0	1	-->	1	2
1	1	0	-->	1	3
1	1	1	-->	1	4

Tabelle 3.4 Der erweiterte Input-Vektor für das Planungsproblem



Listing 3.6 Lösung für das sehr einfache Personalplanungsproblem





3.12 Aufgabe: Roboterschutz

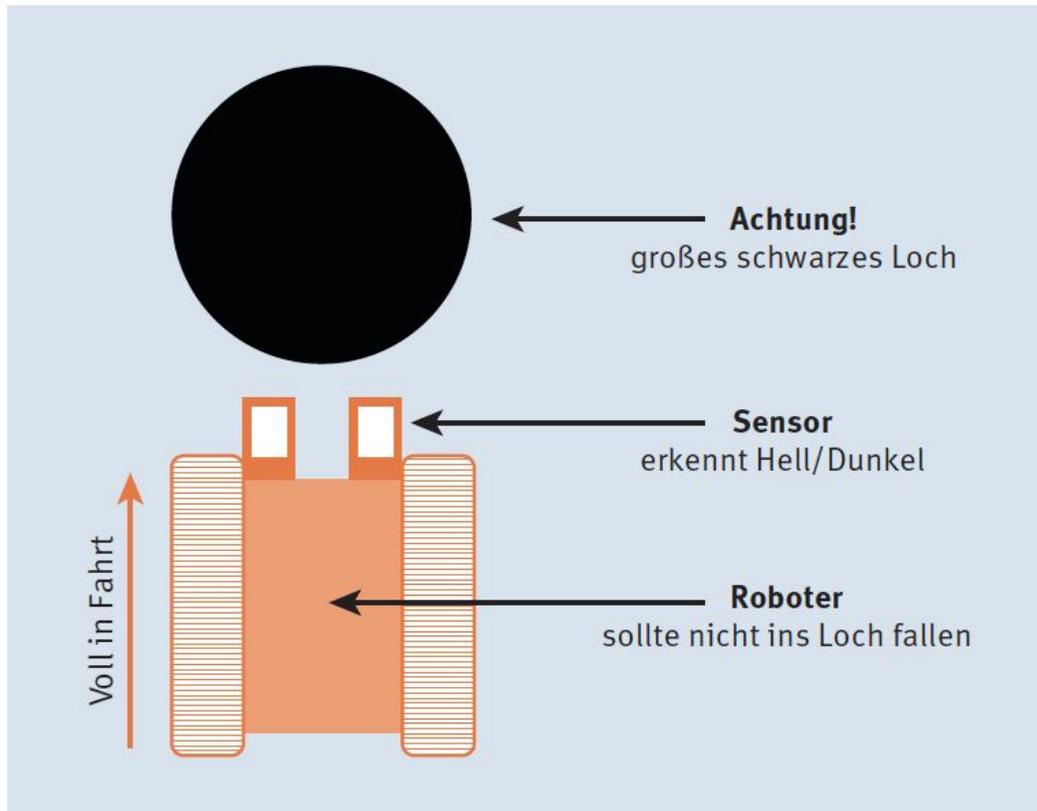


Abbildung 3.16 Einfache Robotersensorik

Sensor links	Sensor rechts	Loch
0	0	0
1	0	0
0	1	0
1	1	1

Tabelle 3.5 Locherkennung mithilfe der Sensorwerte



Kapitel 4

Lernen im einfachen Netz



4.1 Vorgeschichte: Man lässt planen

P1	P2	-->	E
0	0	-->	0
0	1	-->	1
1	0	-->	1
1	1	-->	1

Tabelle 4.1 Das altbekannte Planungsproblem

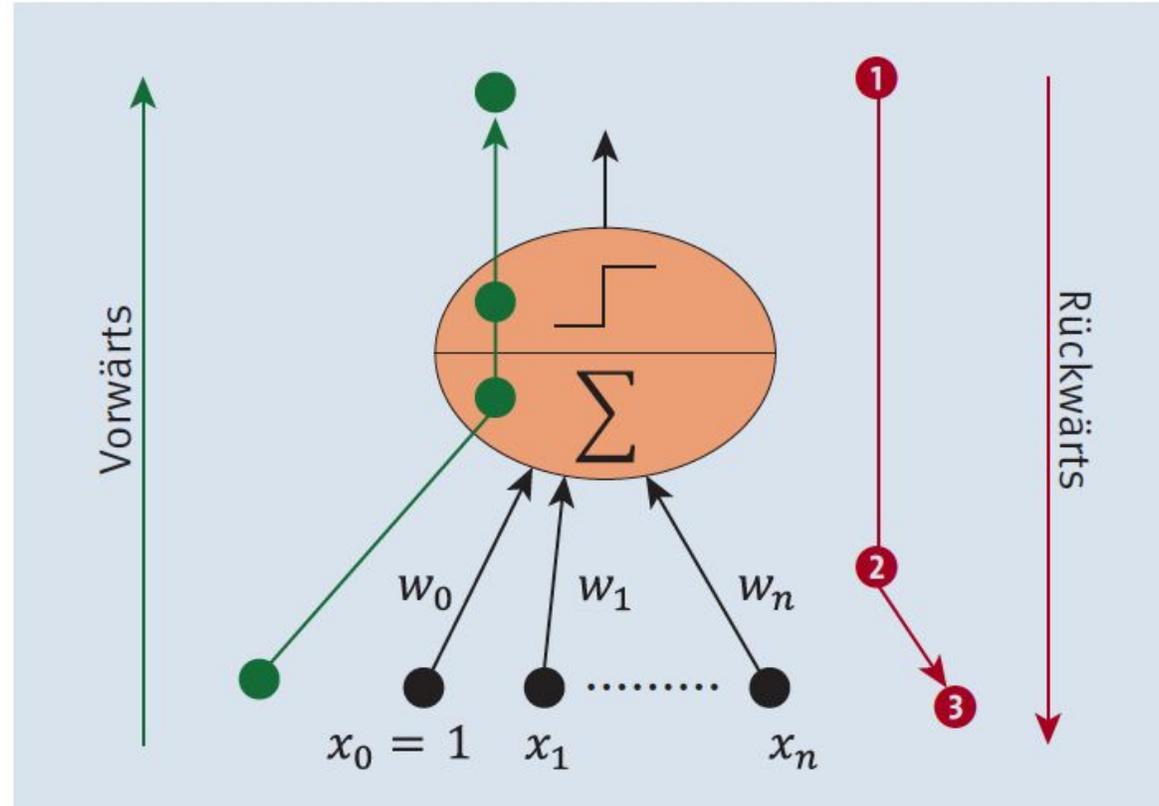


Abbildung 4.1 Vorwärts und rückwärts im Perceptron



4.3 Perceptron-Lernen

$$s = \sum_{i=0}^n w_i \cdot x_i = \vec{w}^t \cdot \vec{x}$$

$$s = 0.1 \cdot x_0 + 0.1 \cdot x_1 + 0.1 \cdot x_2$$

Und für den Input $\vec{x} = (1.0, 0.0, 0.0)$ ergibt das:

$$s = 0.1 \cdot 1.0 + 0.1 \cdot 0.0 + 0.1 \cdot 0.0 = 0.1$$

Nun sind Sie an der Reihe!

Aufgabe

Wie sieht die Gleichung für die Gewichtswerte $\vec{w}^t = (-0.9, 0.1, 0.1)$ aus?

Wie sieht der Wert für $\vec{x} = (1.0, 0.0, 0.0)$ aus?



Weight Change

$$w_i^{\text{neu}} = w_i^{\text{alt}} + \Delta w_i$$

wobei

$$\Delta w_i = (y - \hat{y}) \cdot x_i$$

w_i^{alt} bezeichnet das Gewicht mit dem aktuellen Wert, also zum Beispiel -0.3 .

w_i^{neu} ist das Gewicht nach einer Änderung des alten Gewichts, also falls zum Beispiel 0.1 zum alten Gewicht dazugezählt wird. Das wäre dann bei unserem Beispiel $-0.3 + 0.1 = -0.2$.



Learn step

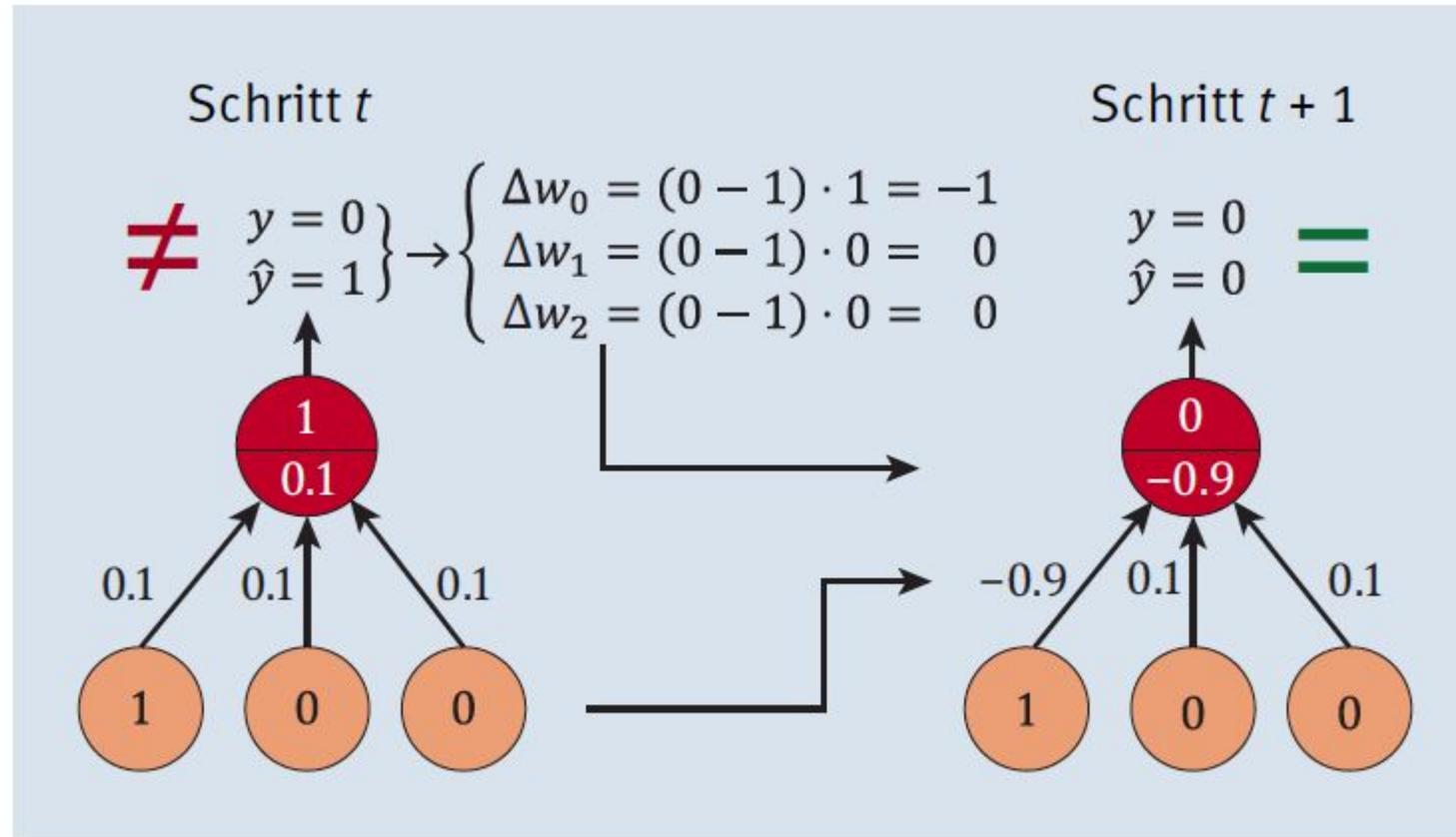


Abbildung 4.2 Ein Perceptron-Lernschritt



Weight Change

y_i	\hat{y}_i	$(y - \hat{y})$	$(y - \hat{y}) \cdot x_i$
0	0	0	0
0	1	-1	$-x_i$
1	0	1	x_i
1	1	0	0

Mit diesen Erkenntnissen können Sie nun wieder in die Formel $w_i^{\text{neu}} = w_i^{\text{alt}} + \Delta w_i$ gehen und Ersetzungen durchführen:

- ▶ w_i^{neu} ist der Wert im Schritt $t + 1$, also können wir $w_i(t + 1)$ schreiben.
- ▶ w_i^{alt} ist der Wert im Schritt t , also können wir $w_i(t)$ schreiben.

Somit ergeben sich die folgenden Berechnungsschritte für die Veränderung der Gewichte:

- ▶ $w_i(t + 1) = w_i(t)$ bei korrekter Ausgabe
- ▶ $w_i(t + 1) = w_i(t) + x_i$ bei Ausgabe 0 und gewünschter Ausgabe 1
- ▶ $w_i(t + 1) = w_i(t) - x_i$ bei Ausgabe 1 und gewünschter Ausgabe 0

Tabelle 4.2 Mögliche Fehler im Perceptron und die dazu passende Änderung des Gewichts



4.4 Trenngerade für einen Lernschritt

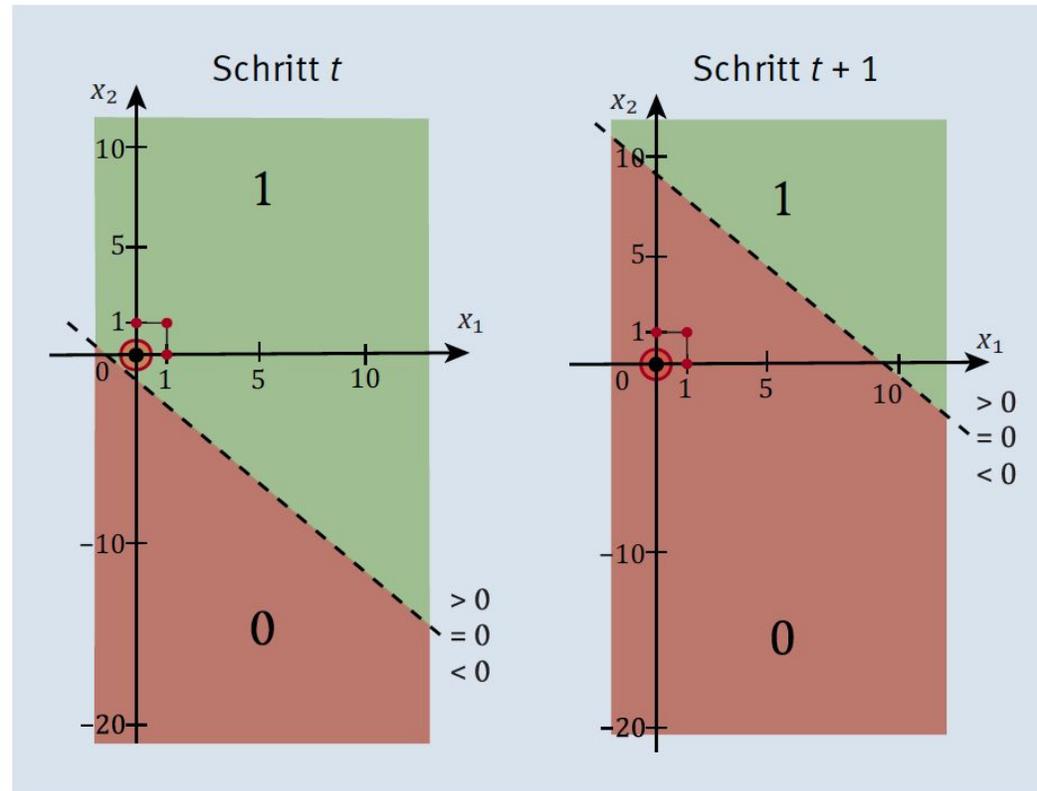


Abbildung 4.3 Die Veränderung der Lage der Trenngeraden durch einen Lernschritt, in dem die Gewichte angepasst werden



Listing 4.1 Perceptron-Lernalgorithmus





Error

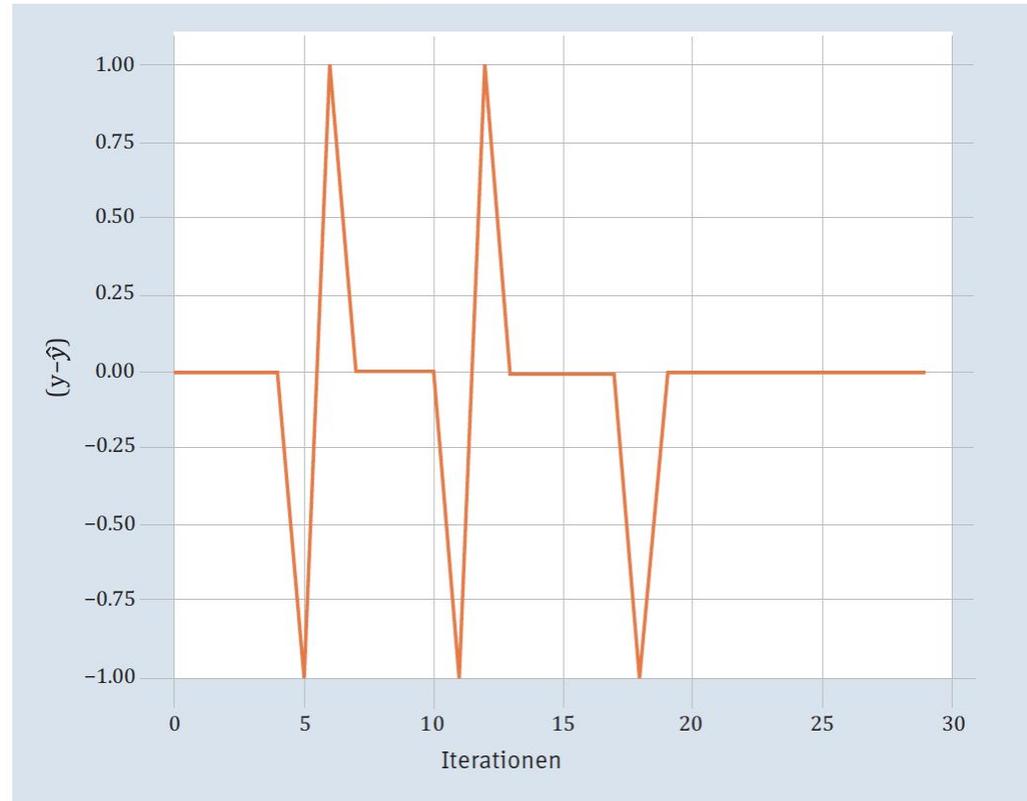


Abbildung 4.4 Die Differenz zwischen gewünschtem und errechnetem Output ($y - \hat{y}$) beim Lernen, pro zufällig gewähltem Trainingsbeispiel



4.6 Die Trenngeraden bzw. Hyperplanes oder auch Hyperebenen für das Beispiel

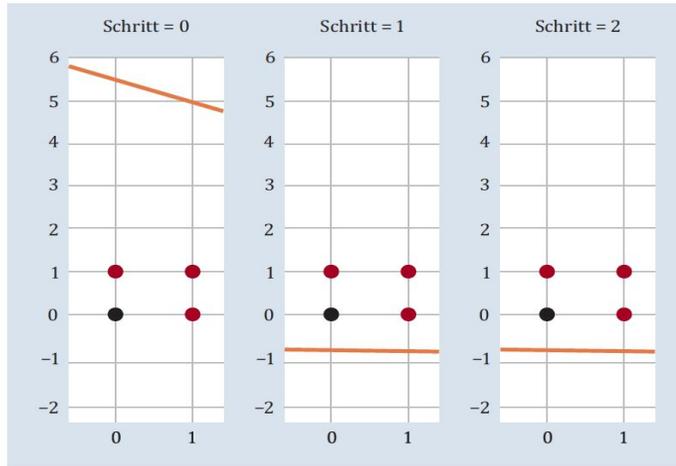


Abbildung 4.5 Schritt 0, Schritt 1 und Schritt 2

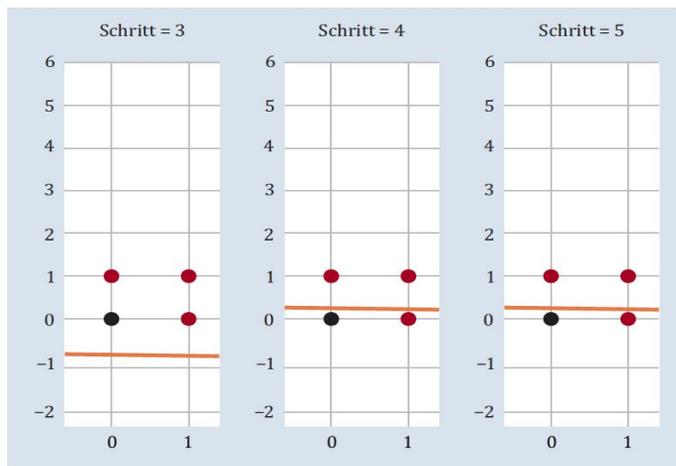


Abbildung 4.6 Schritt 3, Schritt 4 und Schritt 5

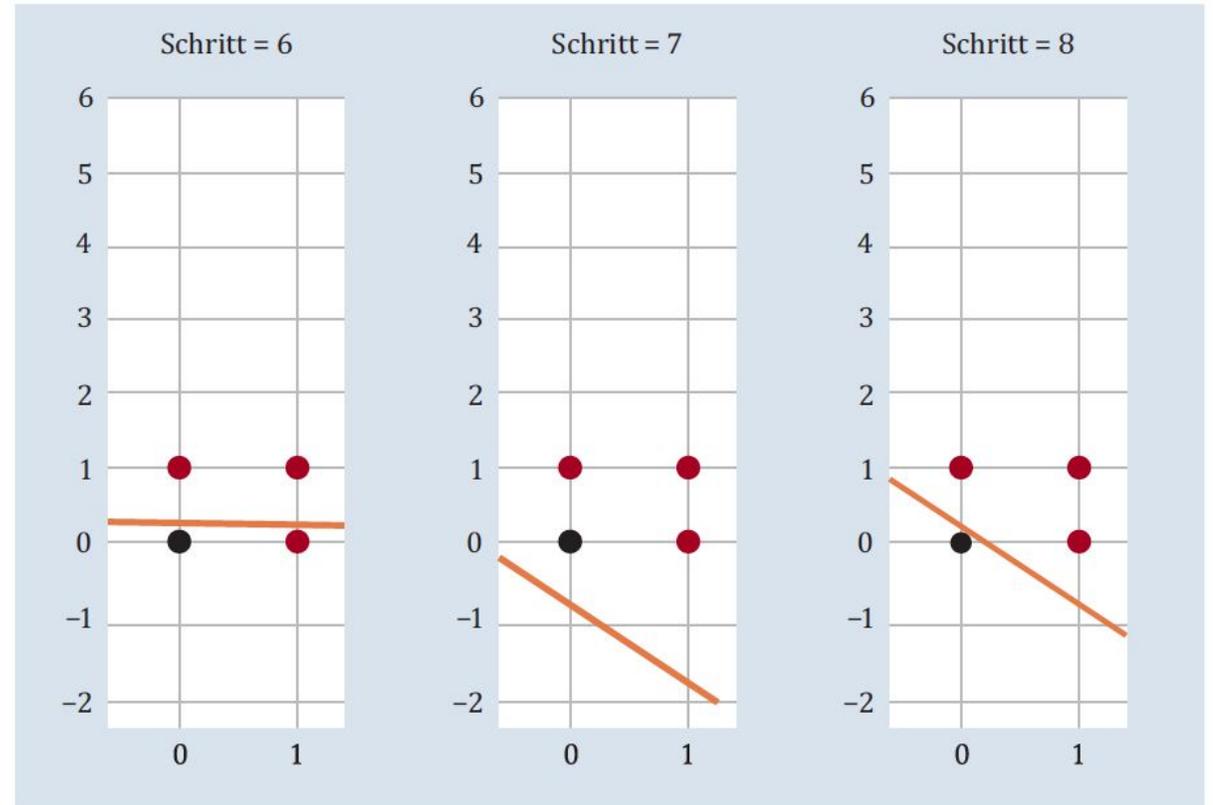


Abbildung 4.7 Schritt 6, Schritt 7 und Schritt 8



Lernschritte

Schritt	x	y	\hat{y}	Fehler ($y - \hat{y}$)	Korrektur ($y - \hat{y}$) · x	w	Abbildung
						(-0.28, 0.02, 0.05)	
0	(1,0,0)	0	1	-1	(-1,0,0)	(-0.28, 0.02, 0.05)	Abb. 4.5
1	(1,0,1)	1	0	1	(1,0,1)	(0.72, 0.02, 1.05)	Abb. 4.5
2	(1,0,1)	1	1	0	(0,0,0)	(0.72, 0.02, 1.05)	Abb. 4.5
3	(1,1,0)	1	1	0	(0,0,0)	(0.72, 0.02, 1.05)	Abb. 4.6

Schritt	x	y	\hat{y}	Fehler ($y - \hat{y}$)	Korrektur ($y - \hat{y}$) · x	w	Abbildung
4	(1,0,0)	0	1	-1	(-1,0,0)	(-0.28,0.02,1.05)	Abb. 4.6
5	(1,0,0)	0	0	0	(0,0,0)	(-0.28,0.02,1.05)	Abb. 4.6
6	(1,1,1)	1	1	0	(0,0,0)	(-0.28,0.02,1.05)	Abb. 4.7
7	(1,1,0)	1	0	1	(1,1,0)	(0.72,1.02,1.05)	Abb. 4.7
8	(1,0,0)	0	1	-1	(-1,0,0)	(-0.28,1.02,1.05)	Abb. 4.7

Tabelle 4.4 Lernschritte und Anpassungen (Forts.)



4.2,3,4,5,6,7,8,9 Scikit compatible Estimator





Error

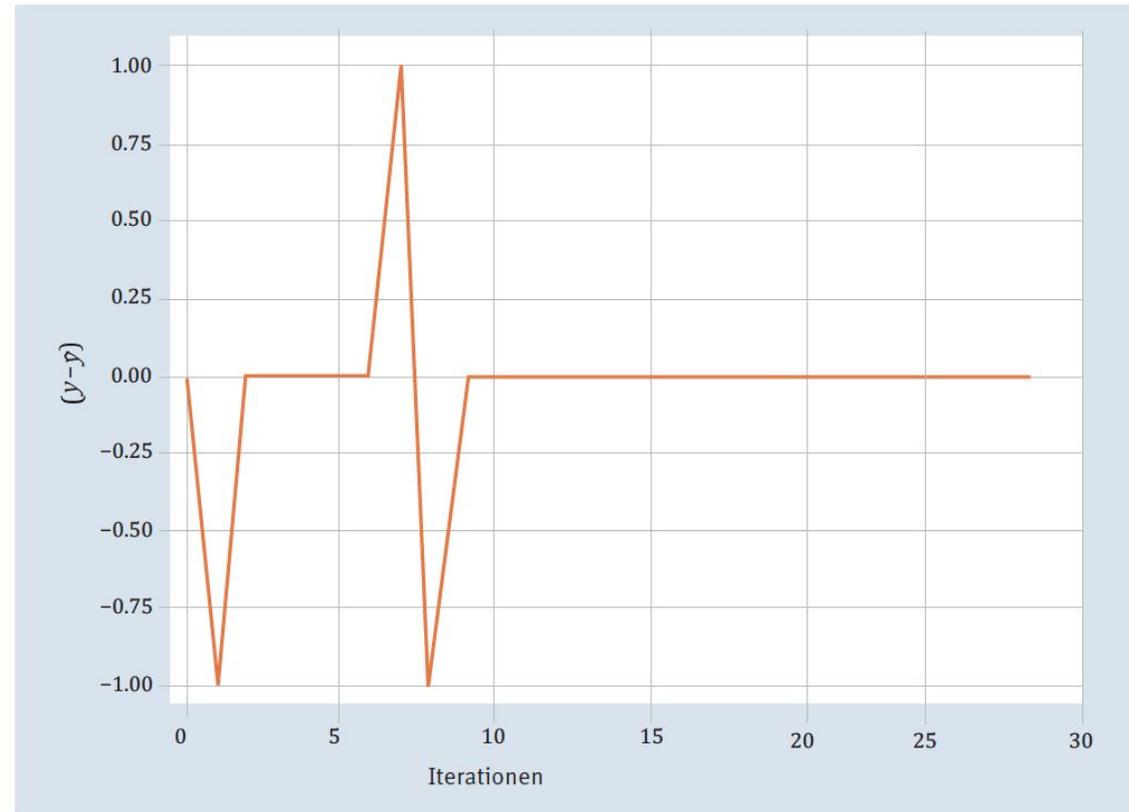


Abbildung 4.8 Differenzen beim Output des Perceptron-Estimators



4.10 scikit-learn-Perceptron-Estimator





Auswertung und Datenpunkte

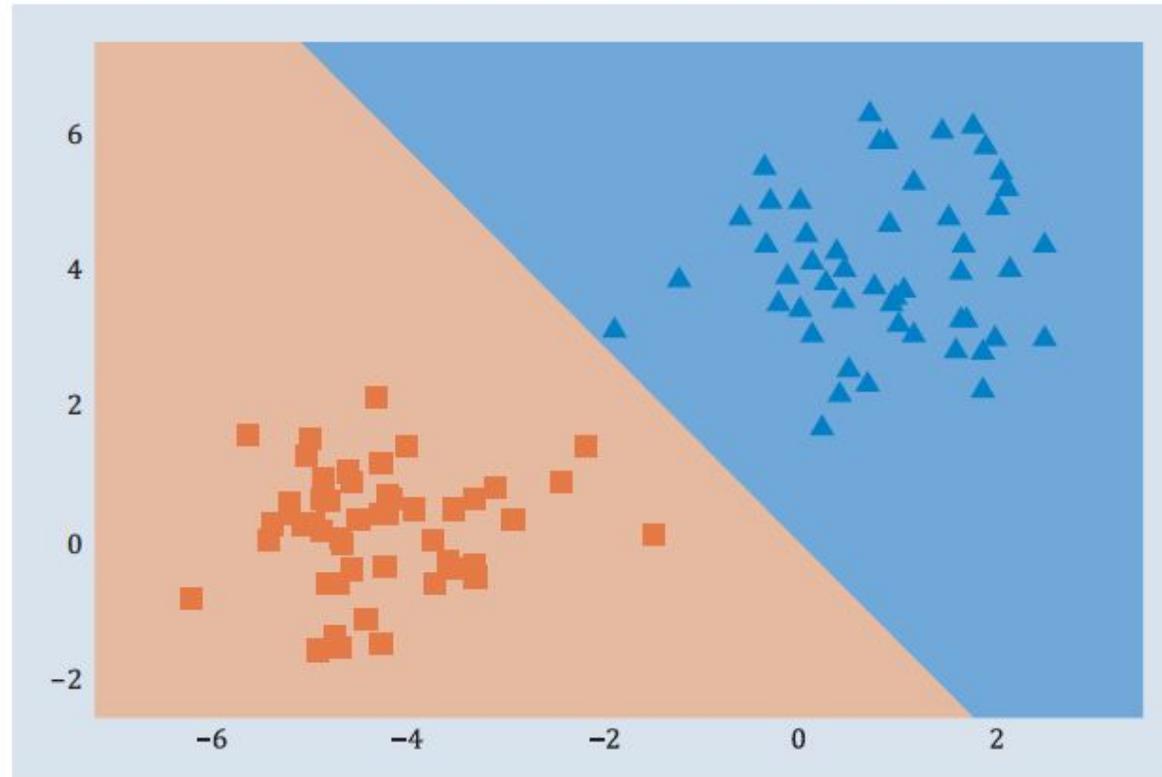


Abbildung 4.9 scikit-learn-Perceptron-Auswertung und Datenpunkte



4.9 Adaline

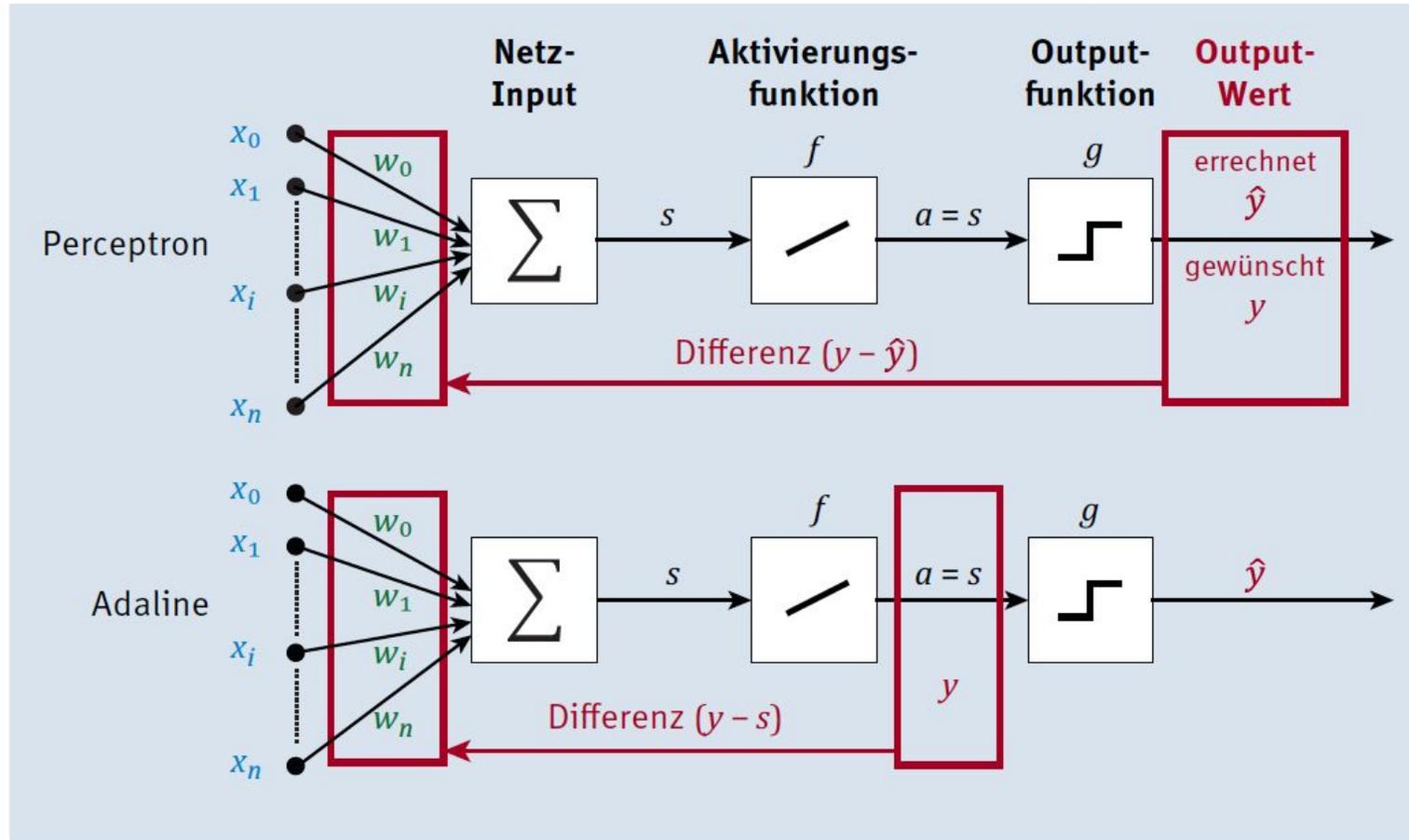
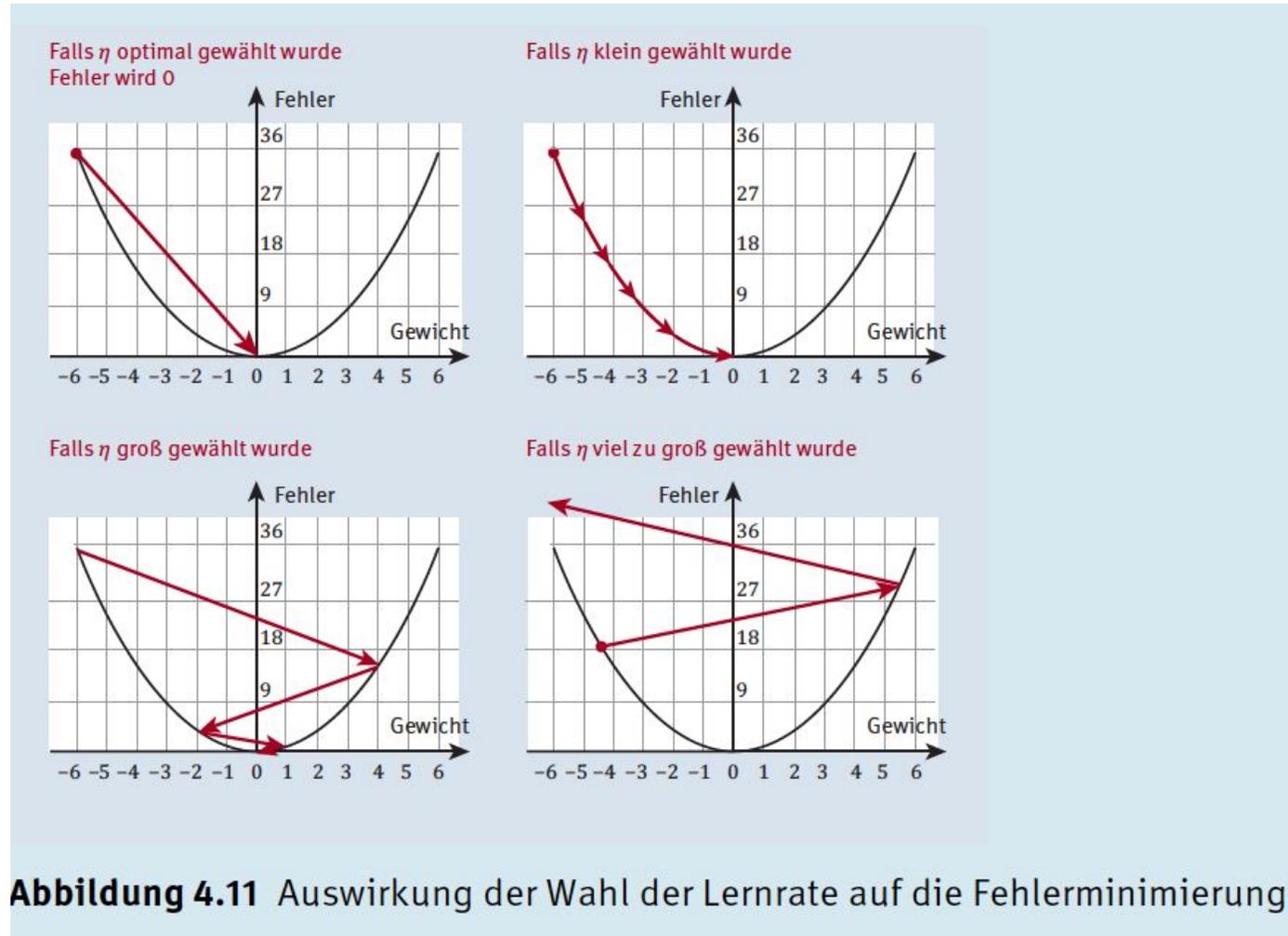


Abbildung 4.10 Differenzermittlung für das Perceptron und Adaline im Vergleich



Lernrate





4.11,12,13,14,15 Adaline-Estimator





Adaline Lernen

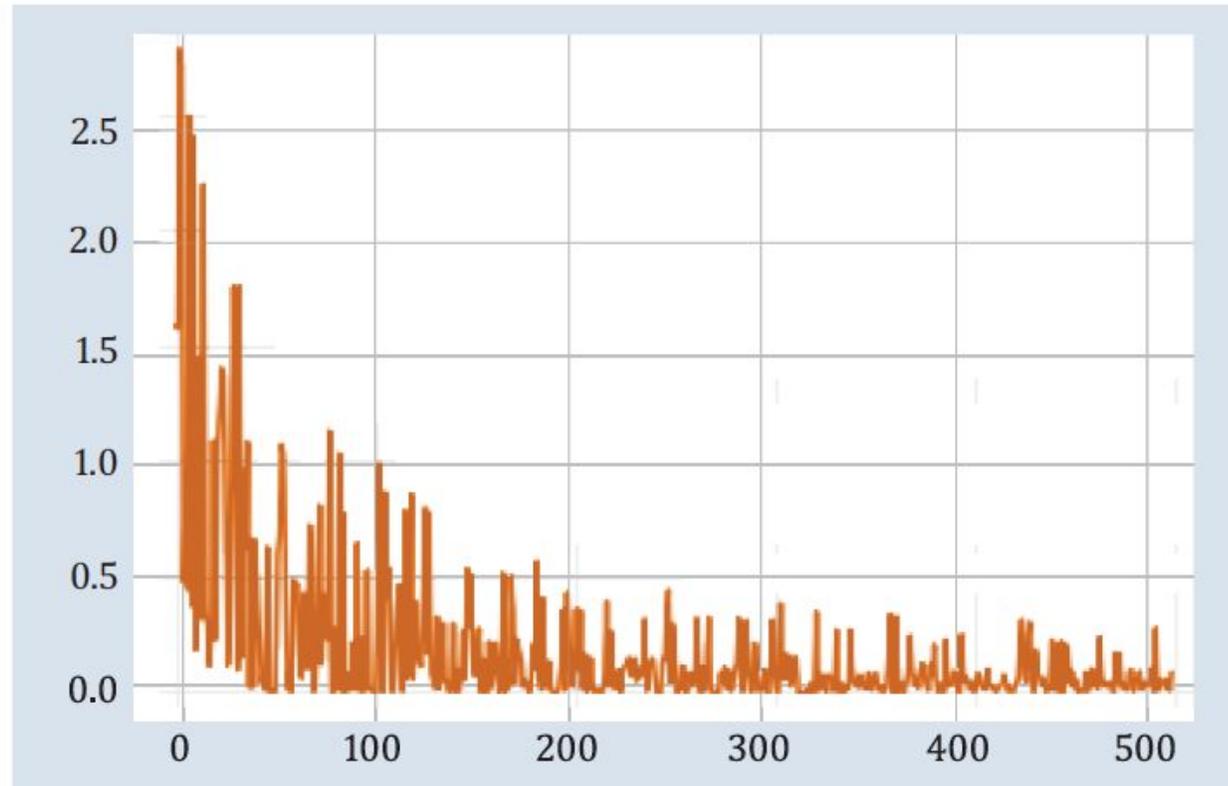


Abbildung 4.12 Adaline-Lernkurve



Trenngeraden

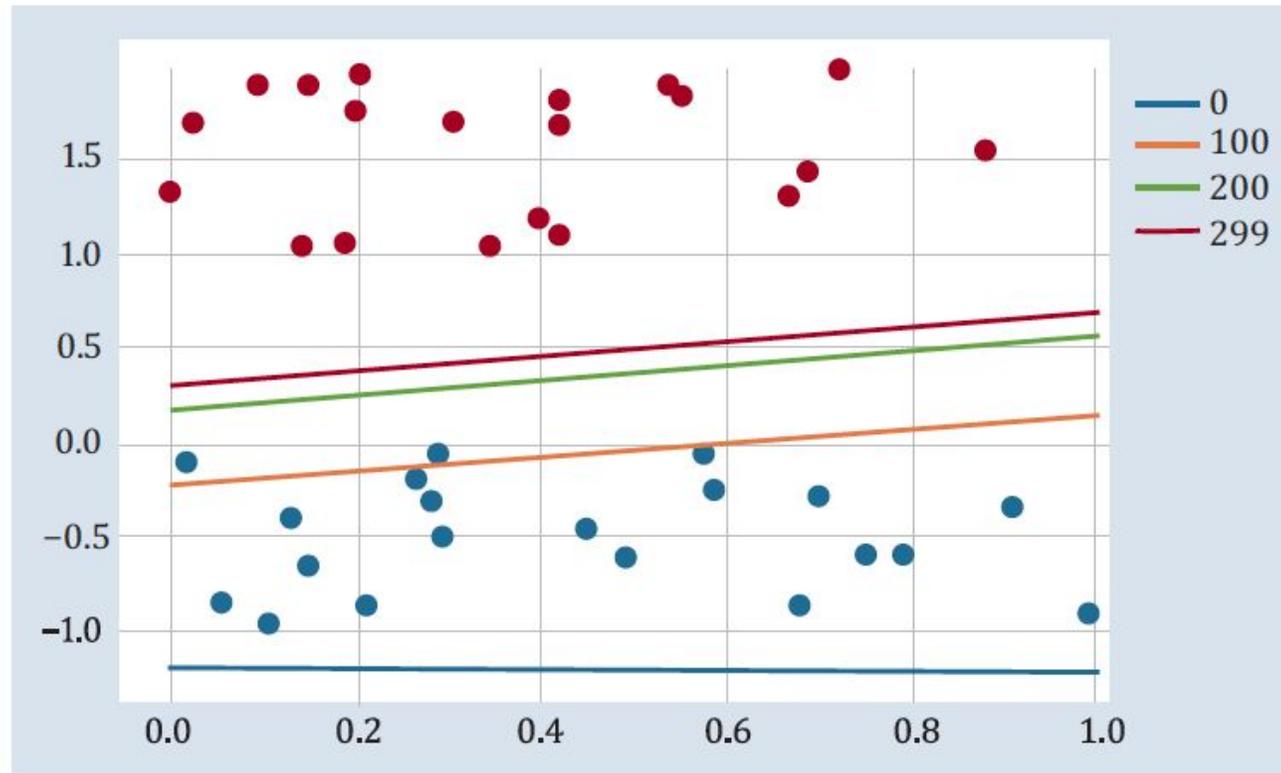


Abbildung 4.13 Adaline-Trenngeraden in unterschiedlichen Schritten



Lernraten

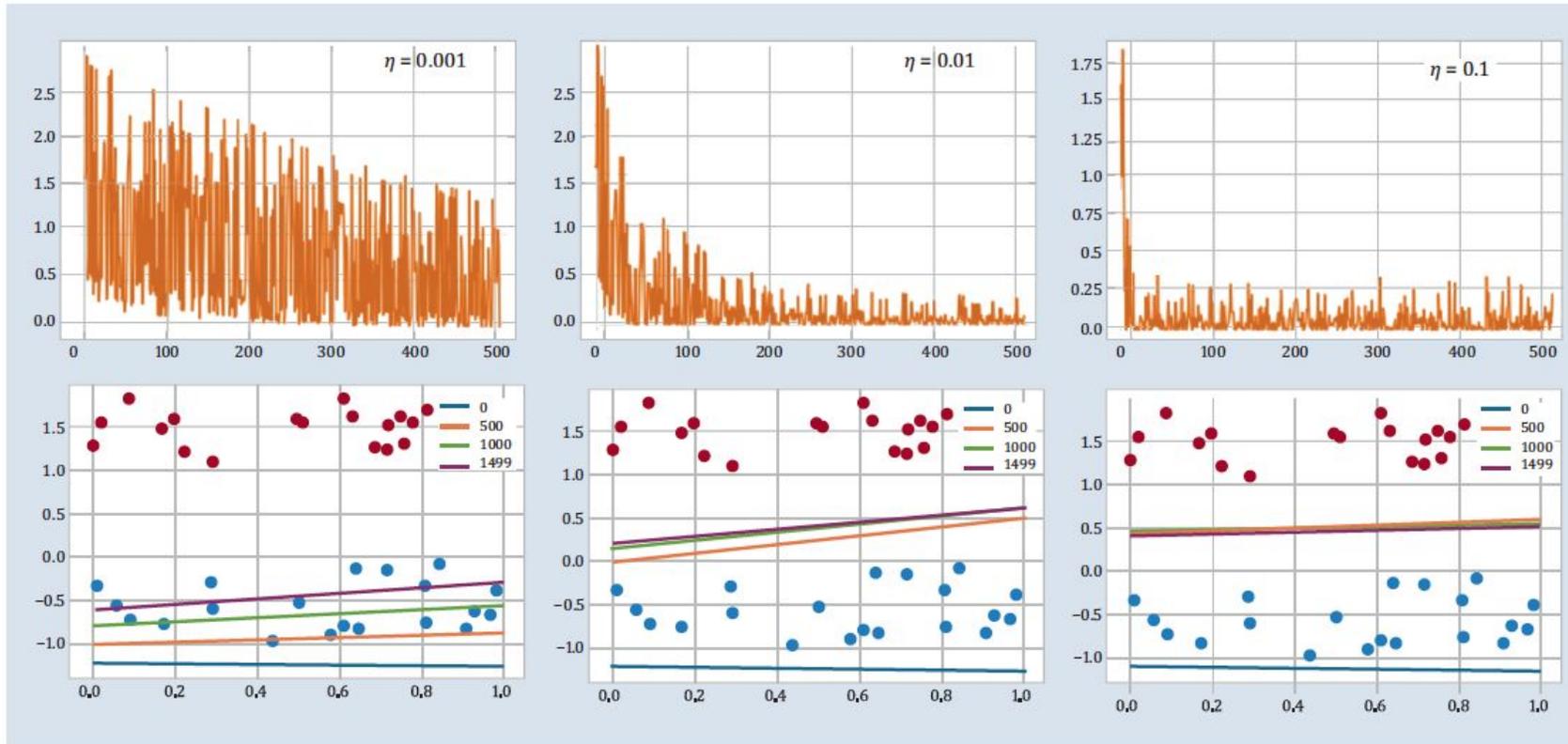


Abbildung 4.14 Lernratenvergleich



Stabile Lösung

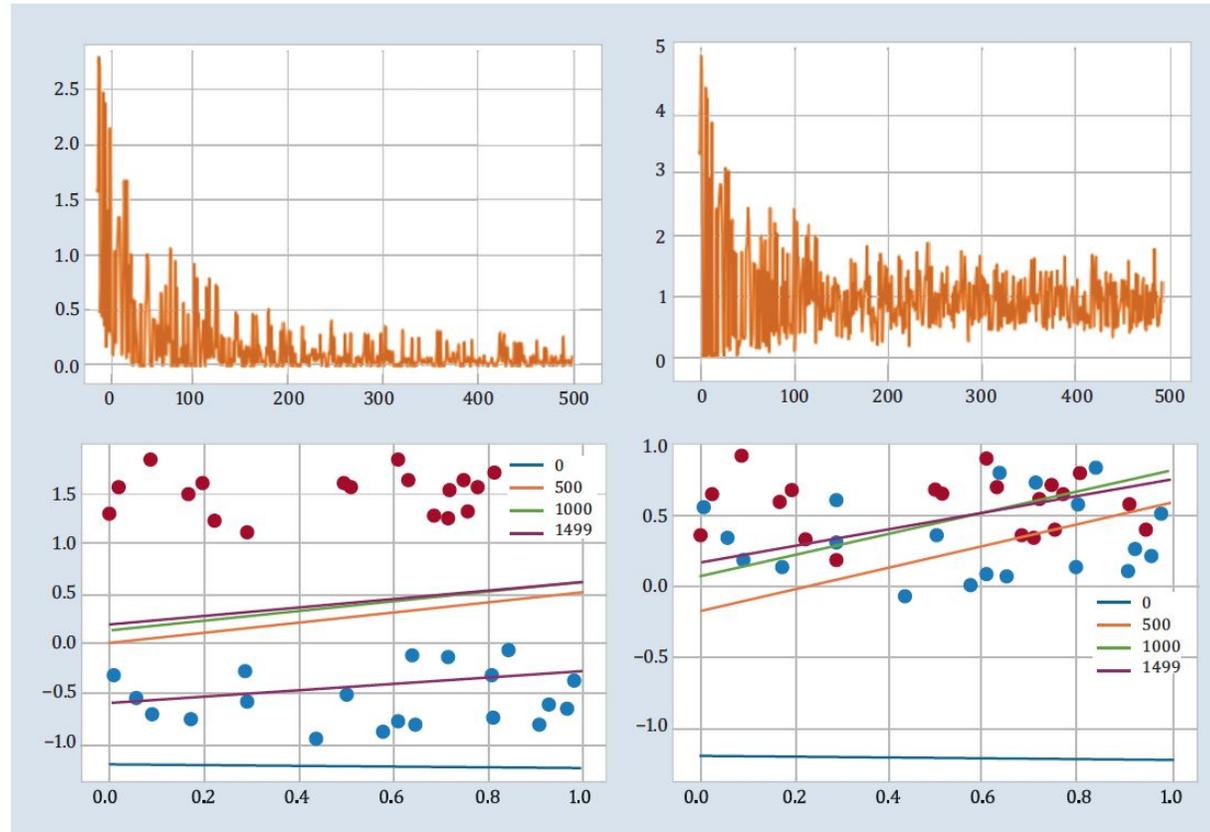
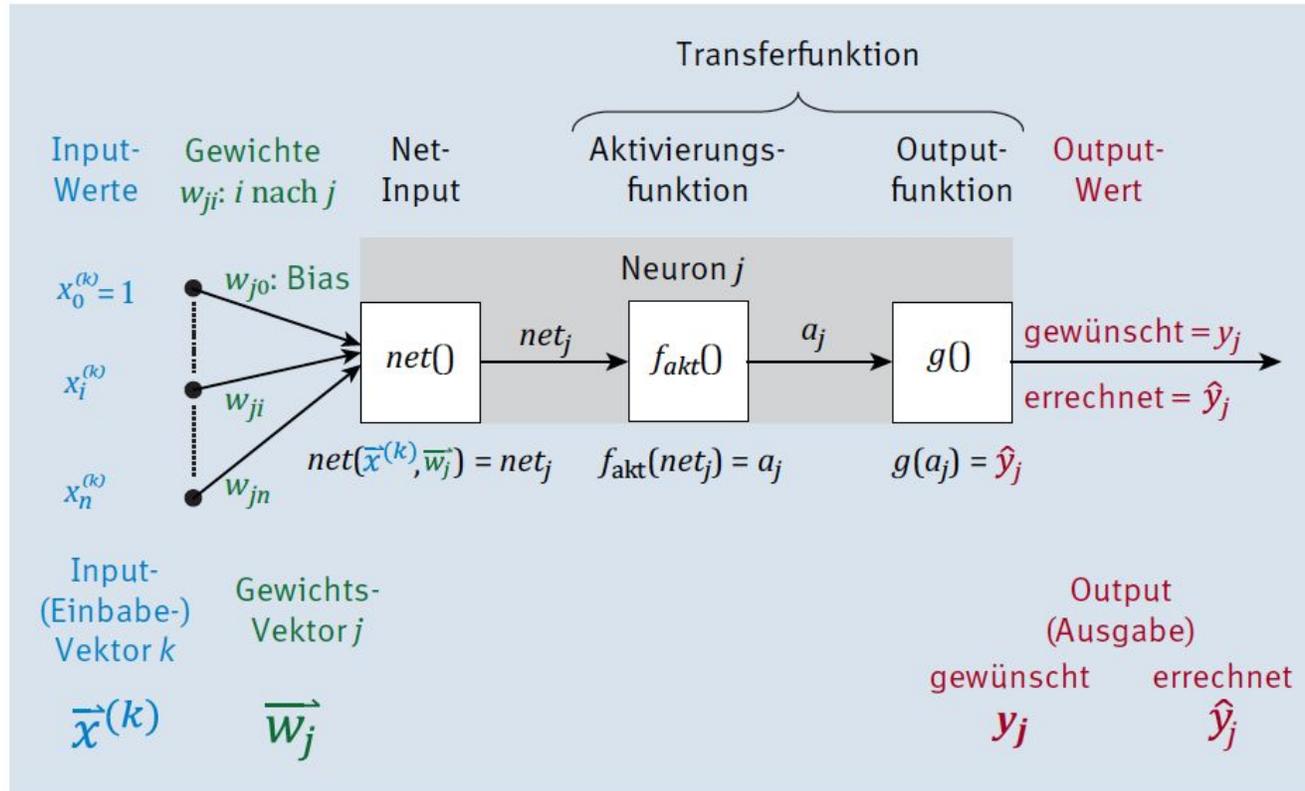


Abbildung 4.15 Überlappungsfaktor der zwei Klassen



Neuronenmodell



Rosenblatt:

$$\Delta w_i = \eta \cdot (y - \hat{y}) \cdot x_i, \text{ bzw.}$$

$$\Delta w_i = (y - \hat{y}) \cdot x_i, \text{ für } \eta = 1$$

Widrow-Hoff:

$$\Delta w_i = \eta \cdot (y - s) \cdot x_i$$

Abbildung 4.16 Verallgemeinertes Neuronenmodell mit Details aus diesem Kapitel



Kapitel 5

Mehrschichtige neuronale

Netze



5.1 Ein echtes Problem

P1	P2	-->	E
0	0	-->	0
1	0	-->	1
0	1	-->	1
1	1	-->	0

Tabelle 5.1 Ein verstecktes XOR-Problem



Lösung?

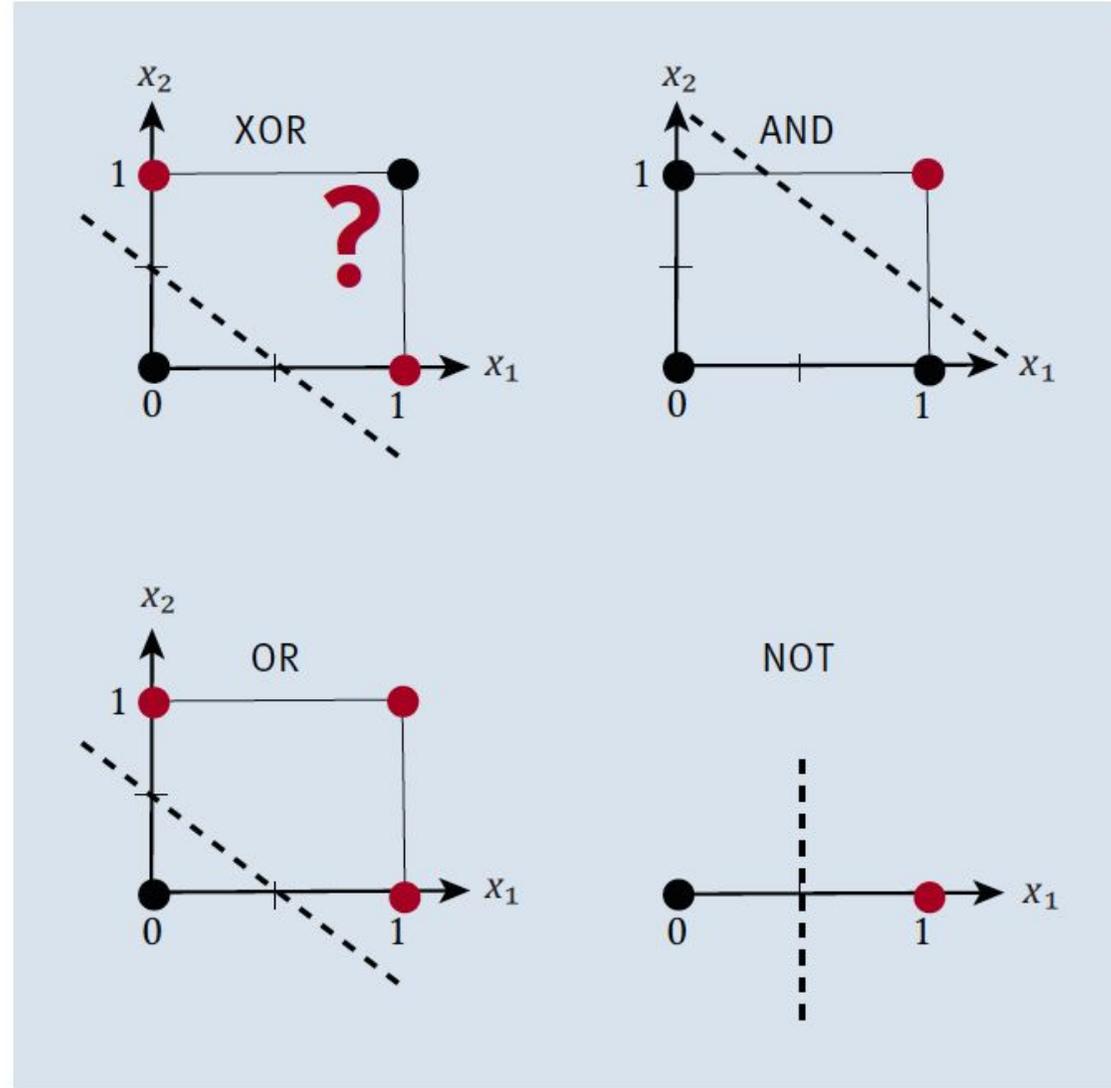


Abbildung 5.1 Wie soll man diese Planungsaufgabe lösen?



5.2 XOR kann man lösen

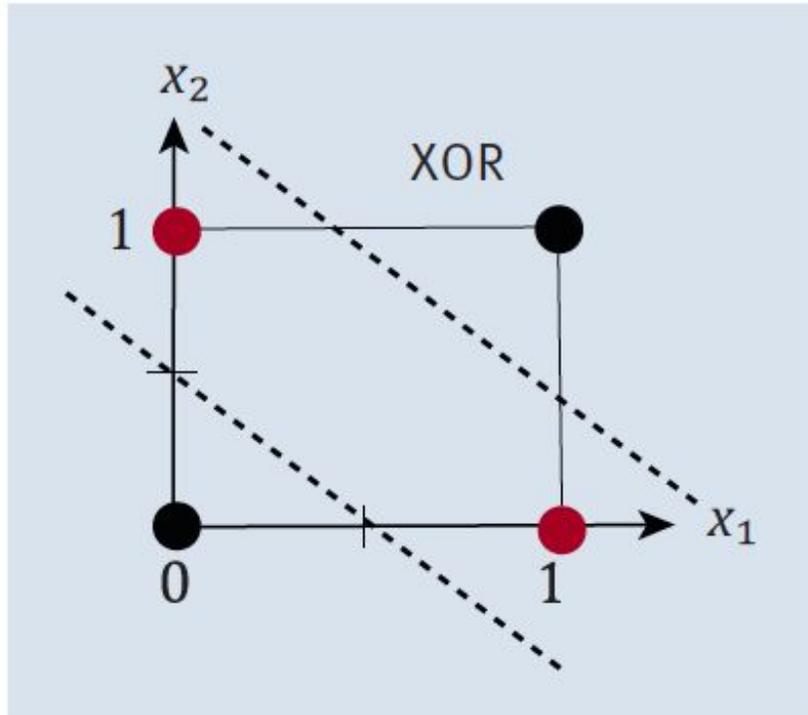


Abbildung 5.2 XOR gelöst

Lösung:

$$(\text{NOT } x_1 \text{ AND } x_2) \text{ OR } (x_1 \text{ AND NOT } x_2)$$

x_1	x_2	NOT x_1	NOT x_2	(NOT x_1 AND x_2)	(x_1 AND NOT x_2)	OR
0	0	1	1	0	0	0
0	1	1	0	1	0	1
1	0	0	1	0	1	1
1	1	0	0	0	0	0

Tabelle 5.2 Wahrheitstafel für das XOR-Problem



2 Perceptrons

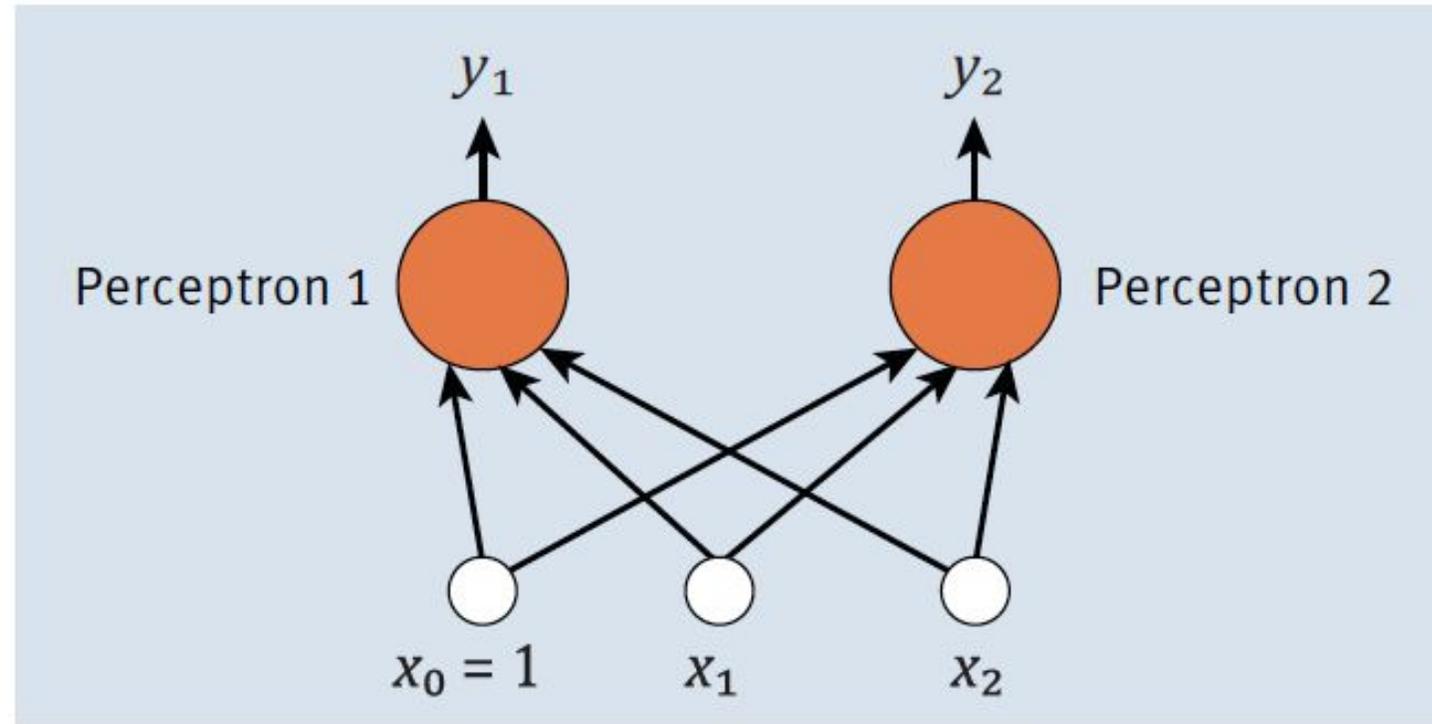


Abbildung 5.3 Zwei Perceptrons



Lösung mit 2 Perceptrons

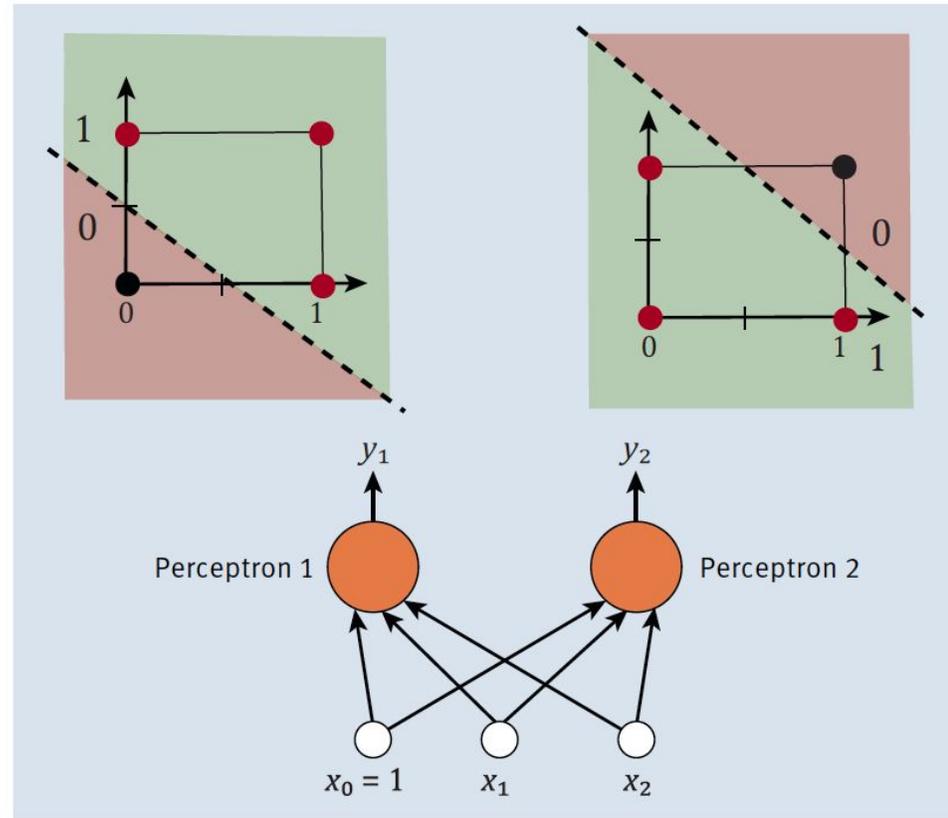


Abbildung 5.4 Lernaufgaben von Perceptron 1 und Perceptron 2



Lösung mit 2 Perceptrons

x_1	x_2	Perceptron 1 (y_1)	Perceptron 2 (y_2)	x_1	x_2	Perceptron 1 (y_1)	Perceptron 2 (y_2)	Perceptron 3 (\hat{y})
0	0	0	1	0	0	0	1	0
1	0	1	1	1	0	1	1	1
0	1	1	1	0	1	1	1	1
1	1	1	0	1	1	1	0	0

Tabelle 5.3 Perceptron 1 und Perceptron 2

Tabelle 5.4 Perceptron 3 muss auch lernen.

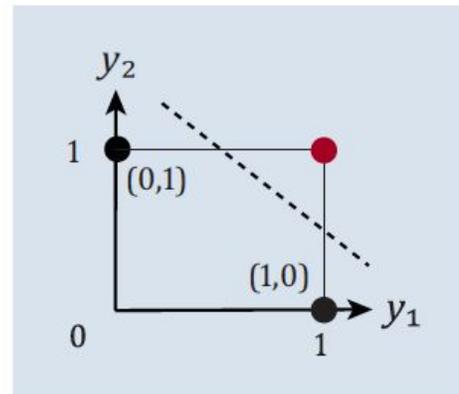


Abbildung 5.5 Perceptron 3 hat gelernt.



Netzwerk und Gewichte

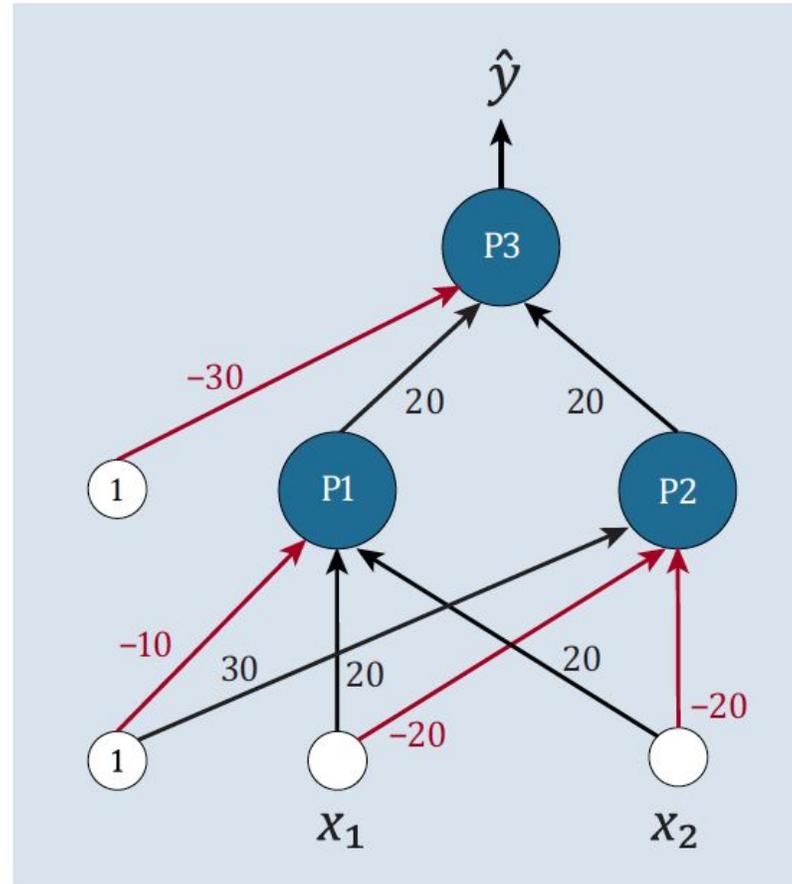


Abbildung 5.6 Das Netzwerk für das XOR-Problem



Berechnungen

Input ($\mathbf{1}, x_1, x_2$)	Output P1	Output P2	Output P3 (\hat{y})
(1,0,0)	$-10 \cdot 1 + 20 \cdot 0 + 20 \cdot 0 = -10$ → 0	$30 \cdot 1 - 20 \cdot 0 - 20 \cdot 0 = 30$ → 1	$-30 \cdot 1 + 20 \cdot 0 + 20 \cdot 1 = -10$ → 0
(1,0,1)	$-10 \cdot 1 + 20 \cdot 0 + 20 \cdot 1 = 10$ → 1	$30 \cdot 1 - 20 \cdot 0 - 20 \cdot 1 = 10$ → 1	$-30 \cdot 1 + 20 \cdot 1 + 20 \cdot 1 = 10$ → 1
(1,1,0)	$-10 \cdot 1 + 20 \cdot 1 + 20 \cdot 0 = 10$ → 1	$30 \cdot 1 - 20 \cdot 1 - 20 \cdot 0 = 10$ → 1	$-30 \cdot 1 + 20 \cdot 1 + 20 \cdot 1 = 10$ → 1
(1,1,1)	$-10 \cdot 1 + 20 \cdot 1 + 20 \cdot 1 = 30$ → 1	$30 \cdot 1 - 20 \cdot 1 - 20 \cdot 1 = 10$ → 0	$-30 \cdot 1 + 20 \cdot 1 + 20 \cdot 0 = -10$ → 0

Tabelle 5.5 Ermittlung der Outputs der unterschiedlichen Neuronen



Netzwerk und Gewichte II

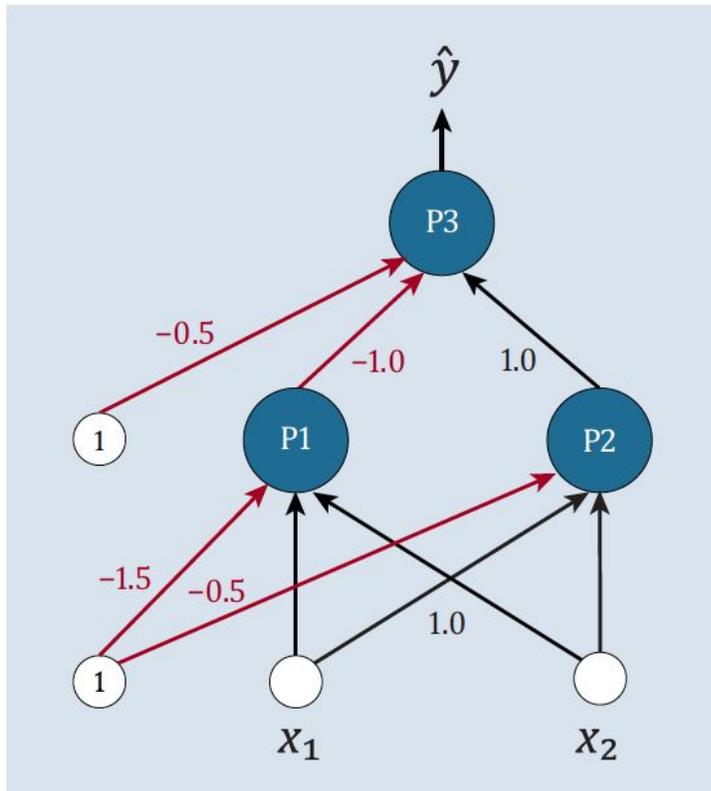


Abbildung 5.7 Lösungsvorschlag zu den Gewichten im MLP

Input $(1, x_1, x_2)$	Output P1	Output P2	Output P3 (\hat{y})
(1,0,0)	$-1.5 \cdot 1 + 1.0 \cdot 0 + 1.0 \cdot 0 = -1.5$ -> 0	$-0.5 \cdot 1 + 1.0 \cdot 0 + 1.0 \cdot 0 = -0.5$ -> 0	$-0.5 \cdot 1 - 1.0 \cdot 0 + 1.0 \cdot 0 = -0.5$ -> 0
(1,0,1)	$-1.5 \cdot 1 + 1.0 \cdot 0 + 1.0 \cdot 1 = -0.5$ -> 0	$-0.5 \cdot 1 + 1.0 \cdot 0 + 1.0 \cdot 1 = 0.5$ -> 1	$-0.5 \cdot 1 - 1.0 \cdot 0 + 1.0 \cdot 1 = 0.5$ -> 1
(1,1,0)	$-1.5 \cdot 1 + 1.0 \cdot 1 + 1.0 \cdot 0 = -1.5$ -> 0	$-0.5 \cdot 1 + 1.0 \cdot 1 + 1.0 \cdot 0 = 0.5$ -> 1	$-0.5 \cdot 1 - 1.0 \cdot 0 + 1.0 \cdot 1 = 0.5$ -> 1
(1,1,1)	$-1.5 \cdot 1 + 1.0 \cdot 1 + 1.0 \cdot 1 = 0.5$ -> 1	$-0.5 \cdot 1 + 1.0 \cdot 1 + 1.0 \cdot 1 = 1.5$ -> 1	$-0.5 \cdot 1 - 1.0 \cdot 1 + 1.0 \cdot 1 = -0.5$ -> 0

Tabelle 5.6 Lösungsvorschlag für die Gewichtung im MLP



5.3 Vorbereitungen für den Start

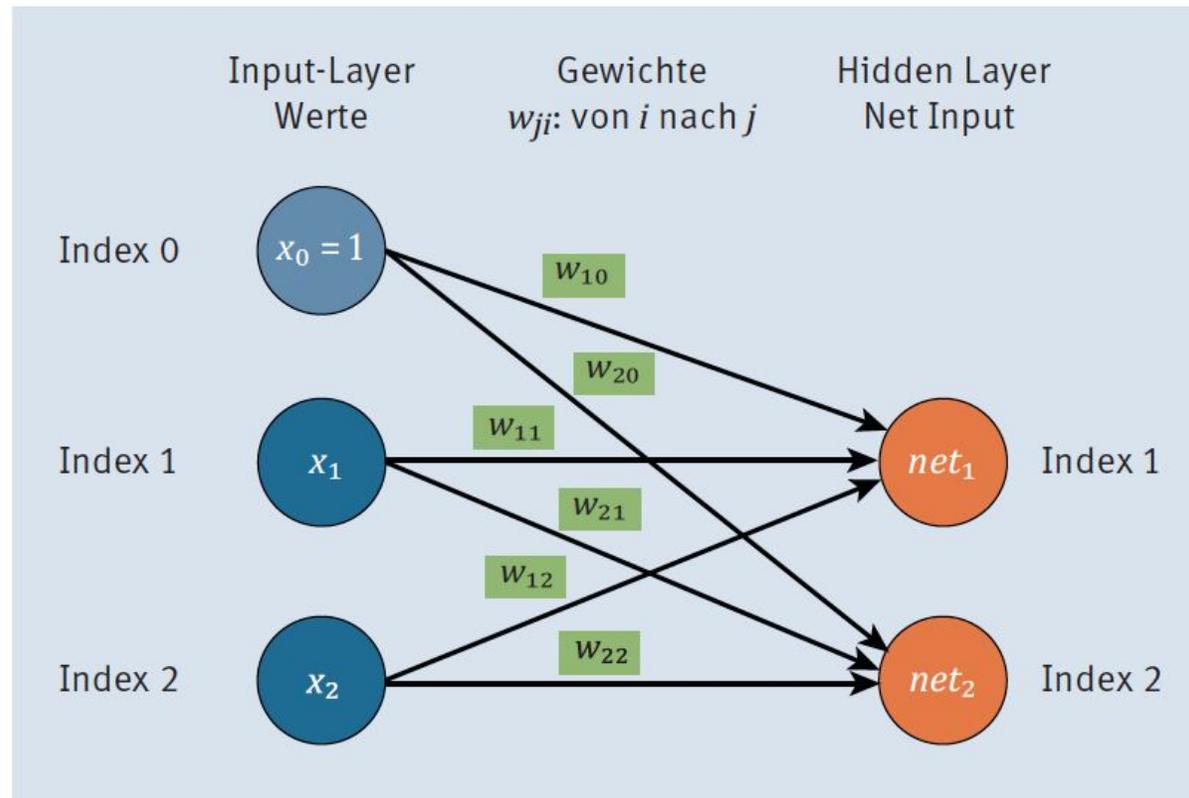


Abbildung 5.8 Die Berechnung im mehrschichtigen Netz



Matrixmultiplikation I

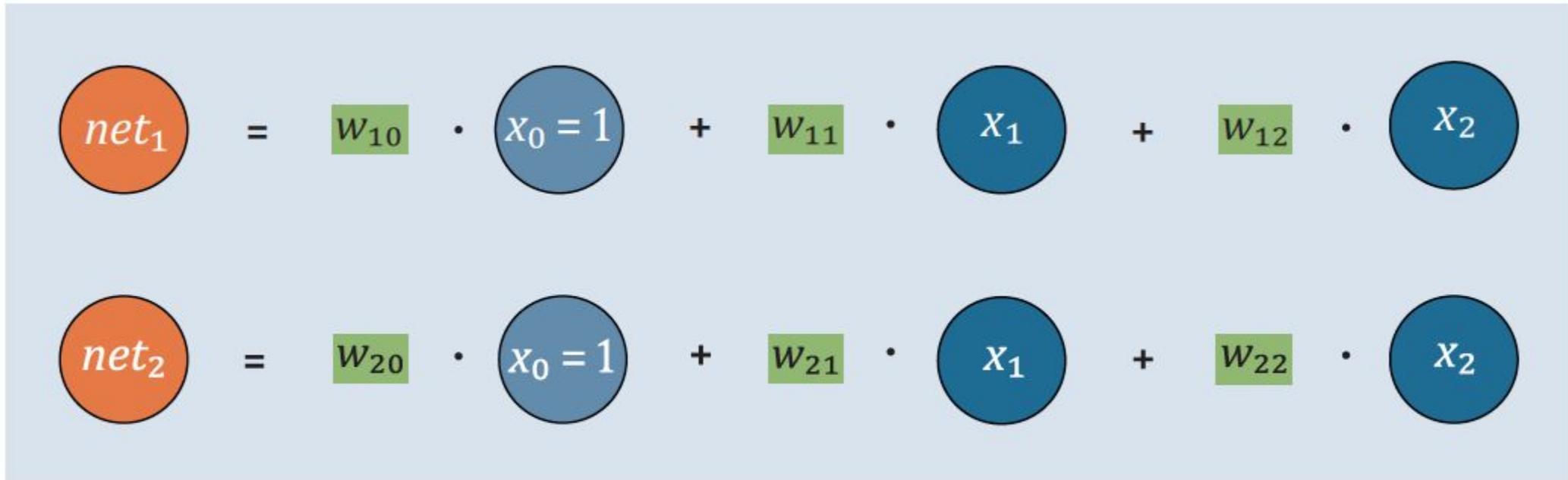


Abbildung 5.9 Auf dem Weg zur Matrixmultiplikation



Matrixmultiplikation II

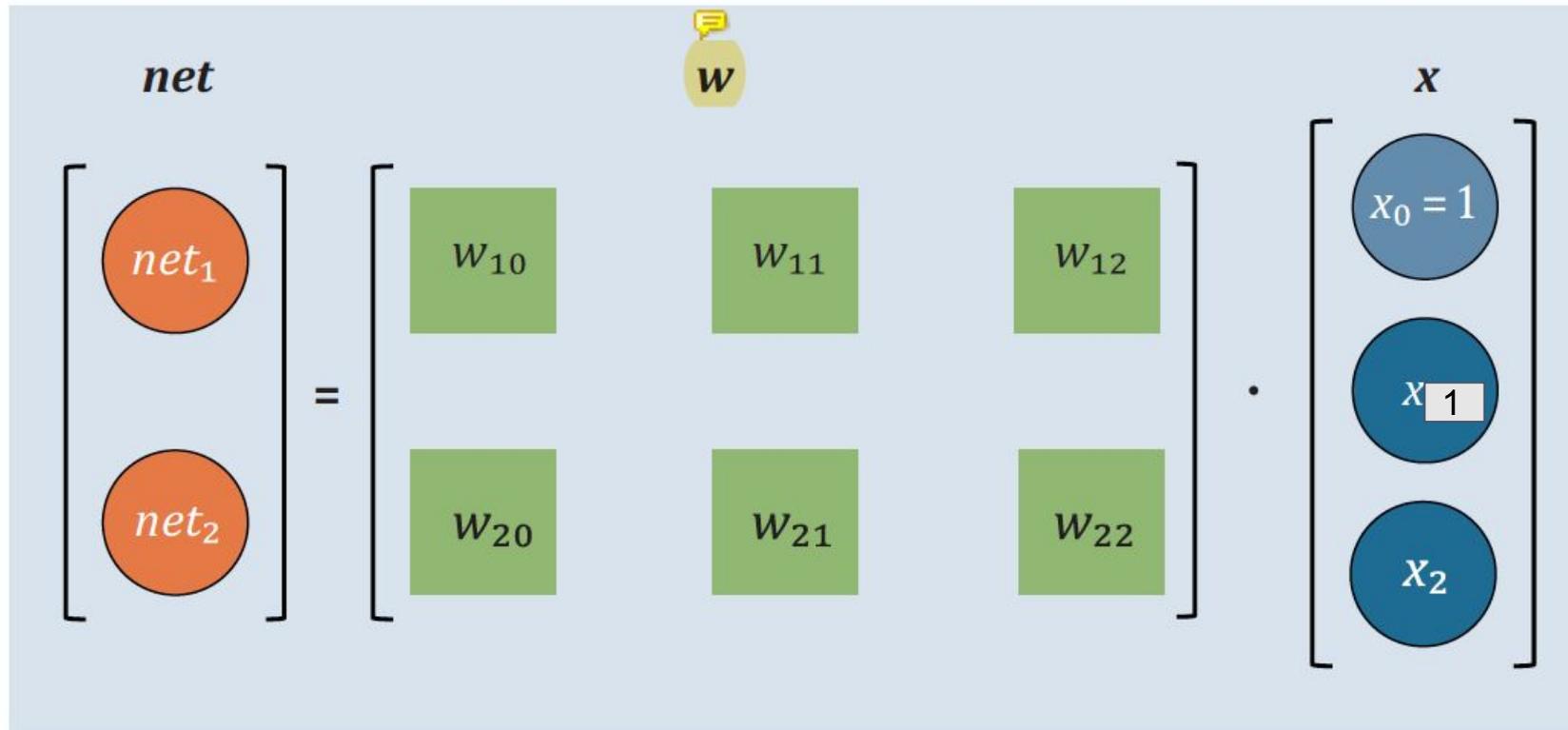


Abbildung 5.10 Matrixmultiplikation, supergeeignet für »numpy«



5.4 Der Plan für die Umsetzung

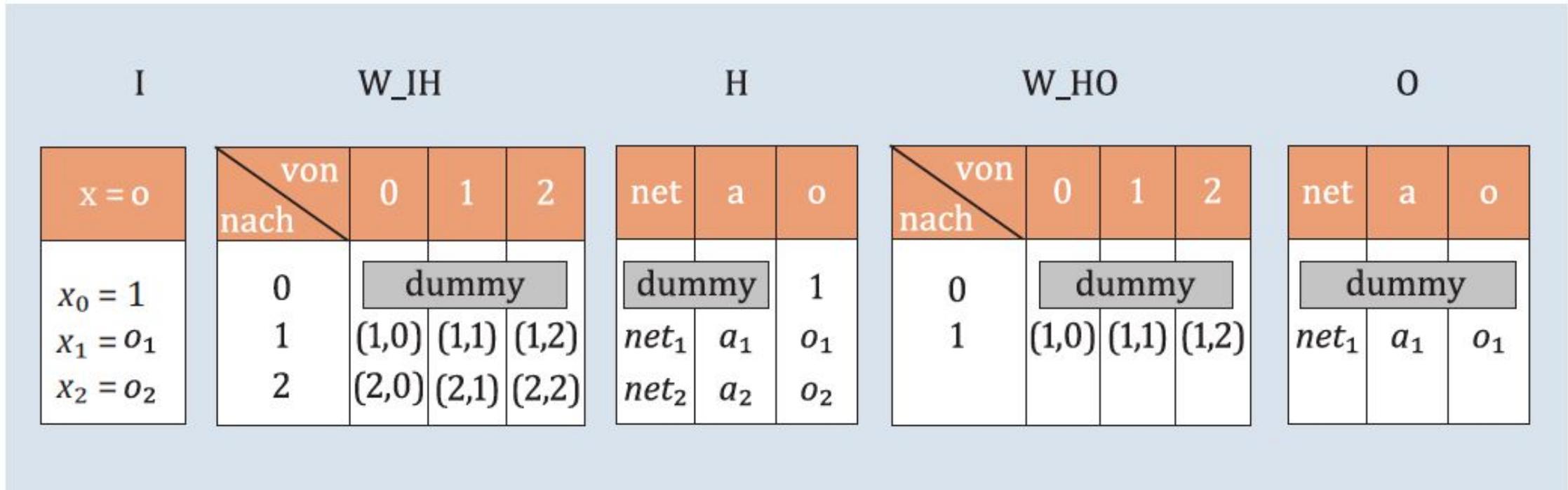


Abbildung 5.11 Die Bausteine des Netzwerkes



5.1,2,3,4,5,6 MLP





5.9 Die Verwendung

```
# Ausgabe:
Multi-Layer Perceptron - Netzwerkarchitektur
[[ 1.000]
 [ 0.000]
 [ 0.000]]
-----v-----
[[ 0.000  0.000  0.000]
 [-10.000 20.000 20.000]
 [ 30.000 -20.000 -20.000]]
-----v-----
[[ 1.000  1.000  1.000]
 [ 0.000  0.000  0.000]
 [ 0.000  0.000  0.000]]
-----v-----
[[ 0.000  0.000  0.000]
 [-30.000 20.000 20.000]]
-----v-----
[[ 0.000  0.000  0.000]
 [ 0.000  0.000  0.000]]
-----v-----
Predict:
[ 1.000  1.000  1.000] 0.0 -> [ 0.000]
[ 1.000  0.000  1.000] 1.0 -> [ 1.000]
[ 1.000  1.000  0.000] 1.0 -> [ 1.000]
[ 1.000  0.000  0.000] 0.0 -> [ 0.000]
```

Listing 5.7 Ausgabe der Architektur mit Gewichten und der Vorhersage



Kapitel 6

Lernen im mehrschichtigen

Netz



6.1 Wie misst man einen Fehler?

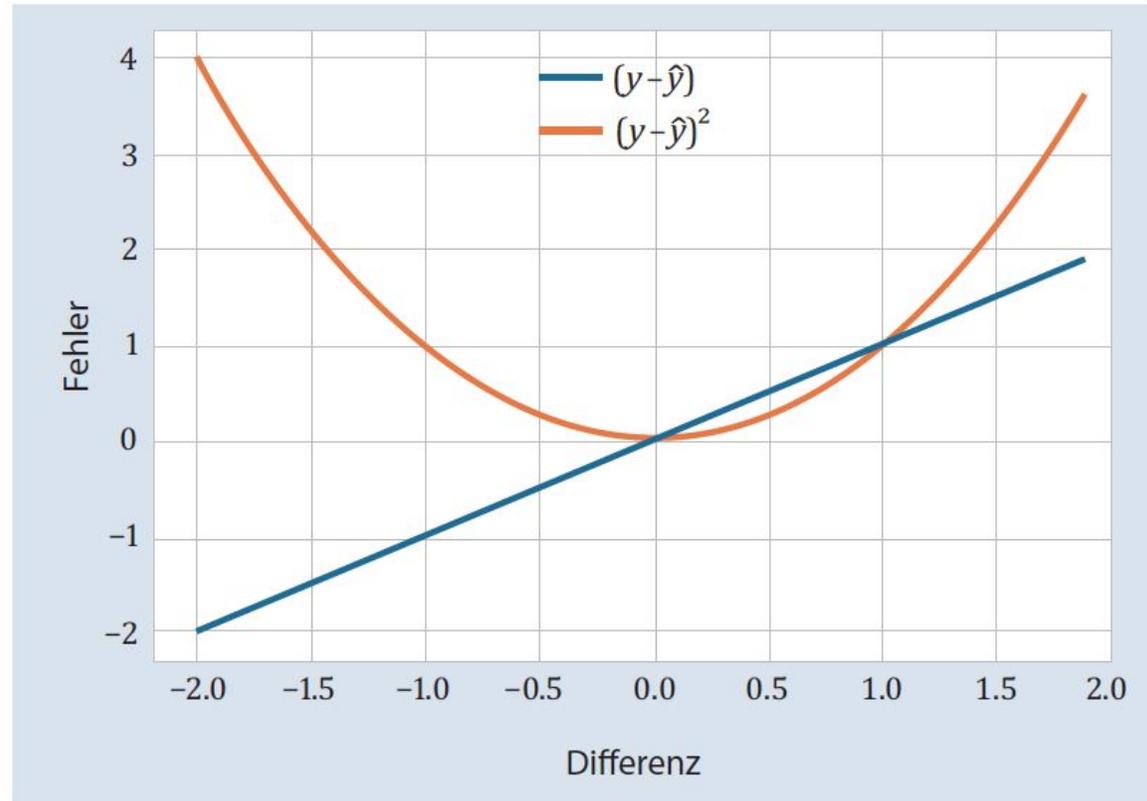


Abbildung 6.1 Fehlerkurven



6.2.1 Gradientenabstieg – die Idee

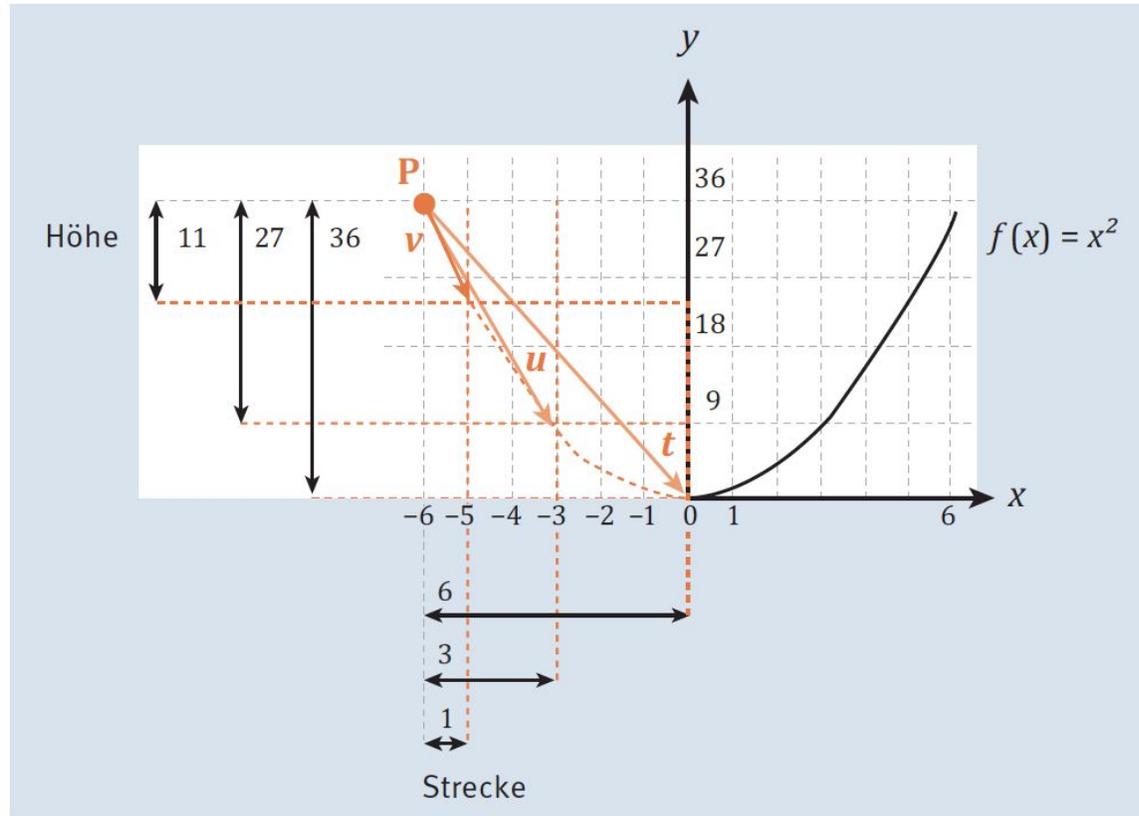


Abbildung 6.2 Der Gradient

Vektor	Höhe	Strecke	Gefälle (Höhe/Strecke)
t	36	6	6
u	27	3	9
v	11	1	11
	5,75	0,5	11,5
	1,19	0,1	11,9
	immer kleiner	immer kleiner	12

Tabelle 6.1 Die Annäherung – Schritt für Schritt zum Gradienten



6.2.2 Algorithmus für den Gradientenabstieg

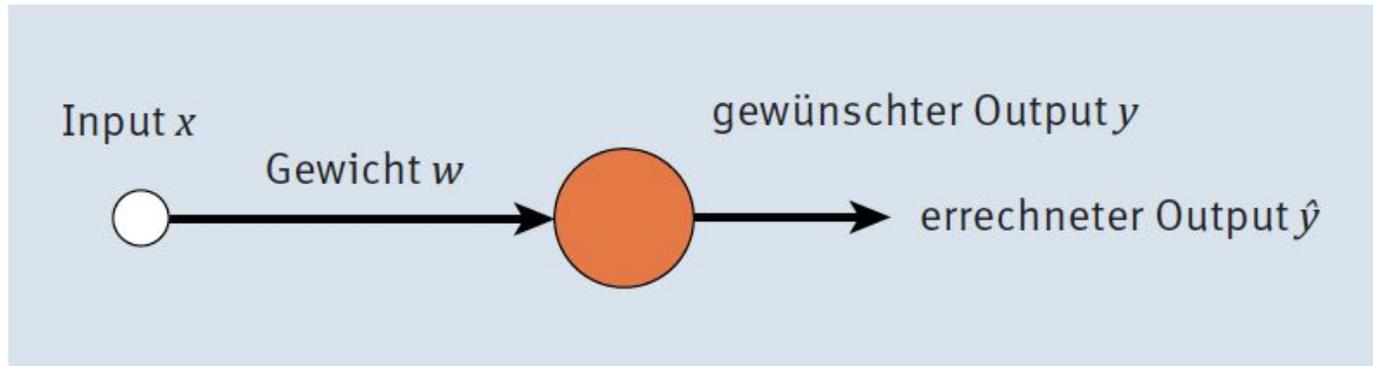


Abbildung 6.3 Einfaches KNN

$$E = \frac{1}{2} \cdot (y - \hat{y})^2 = \frac{1}{2} \cdot (y - w \cdot x)^2$$

lautet der Gradient:

$$\nabla E(w) = \frac{1}{2} \cdot 2 \cdot (y - w \cdot x) \cdot (-1) \cdot x = (-1) \cdot x \cdot (y - \hat{y})$$

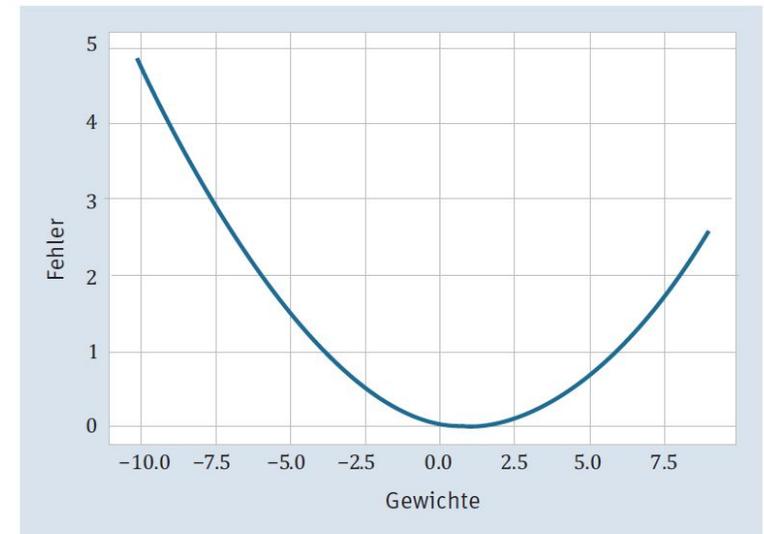


Abbildung 6.4 Fehler je nach Gewicht



Listing 6.1 Gradientenabstieg für das einfache KNN





Tabelle 6.2 Gradientenabstieg für das einfache KNN

Iteration	x	w	Net Input	a	y_hat	y	E	E'	w delta
0	0.2	-10.00	-2.00	-2.00	-2.00	0.20	2.42	-0.44	0.44
10	0.2	-6.31	-1.26	-1.26	-1.26	0.20	1.07	-0.29	0.29
20	0.2	-3.86	-0.77	-0.77	-0.77	0.20	0.47	-0.19	0.19
30	0.2	-2.23	-0.45	-0.45	-0.45	0.20	0.21	-0.13	0.13
40	0.2	-1.15	-0.23	-0.23	-0.23	0.20	0.09	-0.09	0.09
50	0.2	-0.43	-0.09	-0.09	-0.09	0.20	0.04	-0.06	0.06
60	0.2	0.05	0.01	0.01	0.01	0.20	0.02	-0.04	0.04
70	0.2	0.37	0.07	0.07	0.07	0.20	0.01	-0.03	0.03
80	0.2	0.58	0.12	0.12	0.12	0.20	0.00	-0.02	0.02
90	0.2	0.72	0.14	0.14	0.14	0.20	0.00	-0.01	0.01
100	0.2	0.81	0.16	0.16	0.16	0.20	0.00	-0.01	0.01
110	0.2	0.88	0.18	0.18	0.18	0.20	0.00	-0.00	0.00
120	0.2	0.92	0.18	0.18	0.18	0.20	0.00	-0.00	0.00

Tabelle 6.2 Gradientenabstieg für das einfache KNN

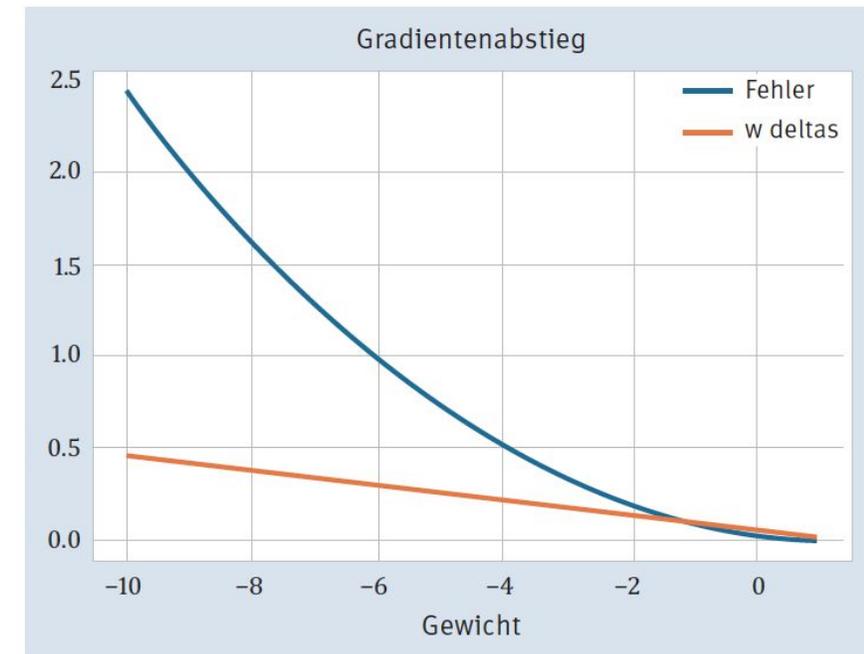


Abbildung 6.5 Fehler beim Gradientenabstieg und Schrittweite bei der Gewichtsangpassung



Sigmoide und ihre Ableitung

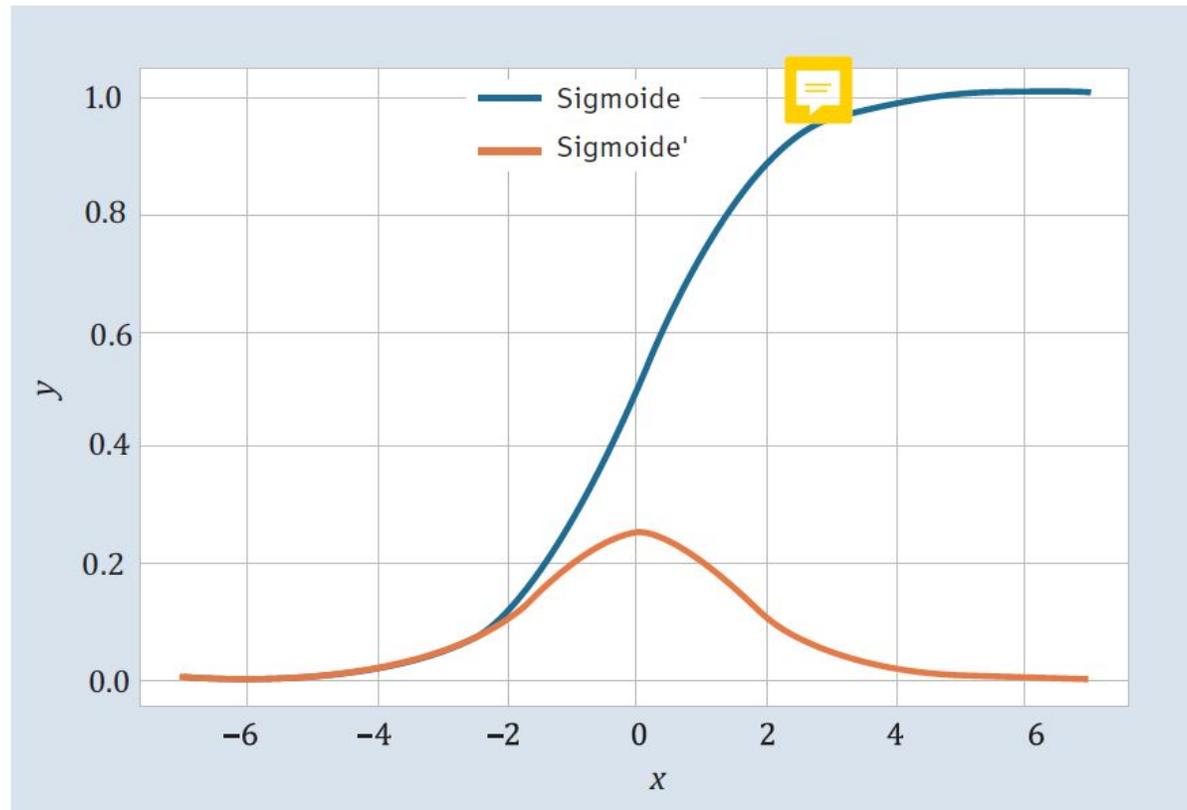


Abbildung 6.6 Sigmoide und die Ableitung der Sigmoide



6.3 Ein Netz aus sigmoiden Neuronen

$$E = \frac{1}{2} \cdot \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

$$w_{ji}^{\text{neu}} = w_{ji}^{\text{alt}} + \Delta w_{ji}, \text{ mit}$$

$$\Delta w_{ji} = -\eta \cdot \delta_j \cdot o_i$$

$$\delta_j = \begin{cases} o_j \cdot (1 - o_j) \cdot (y_j - \hat{y}_j), & \text{falls } j \text{ Output-Neuron} \\ o_j \cdot (1 - o_j) \cdot \sum_k \delta_k \cdot w_{kj} & \text{falls } j \text{ Hidden Neuron} \end{cases}$$



Abbildung 6.7 Vom Output zur versteckten Schicht

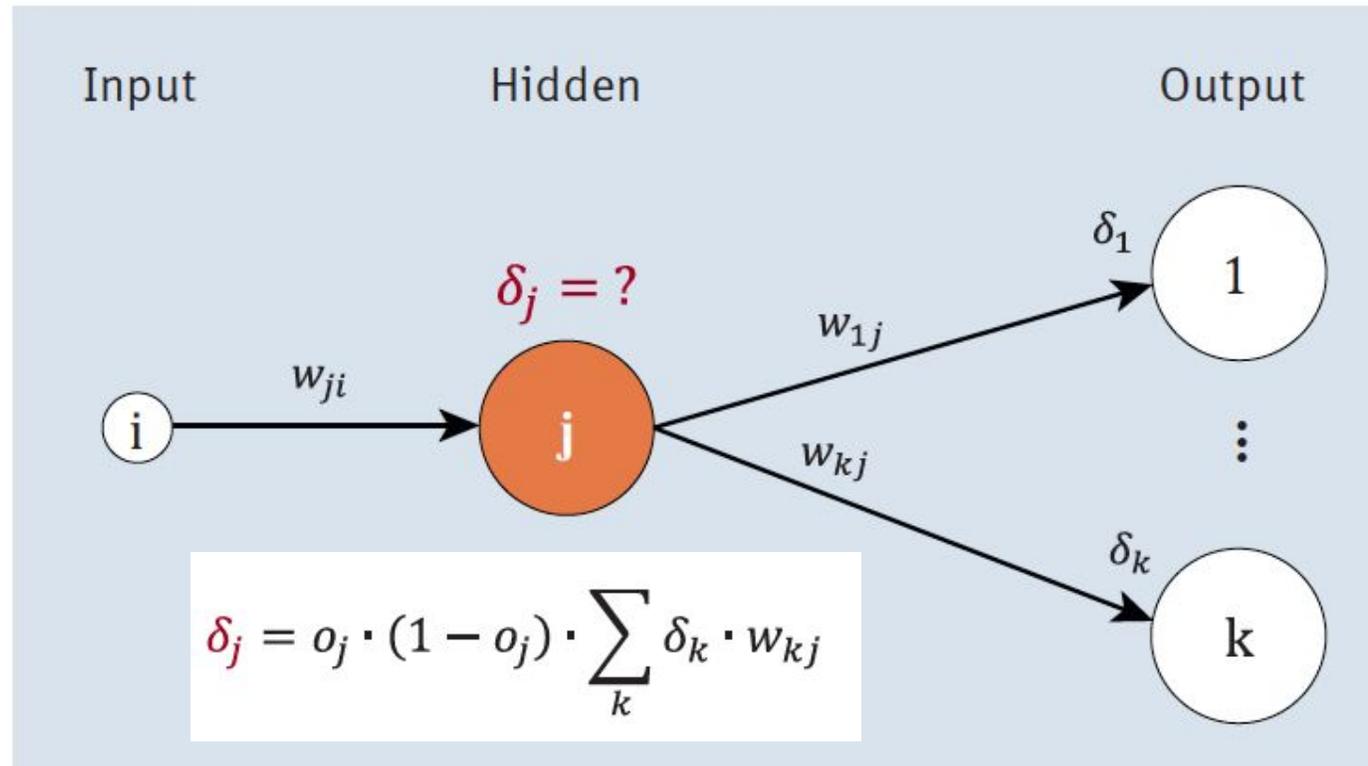


Abbildung 6.7 Vom Output zur versteckten Schicht



Abbildung 6.8 Vektor- und Matrixmultiplikationen für die Programmierung

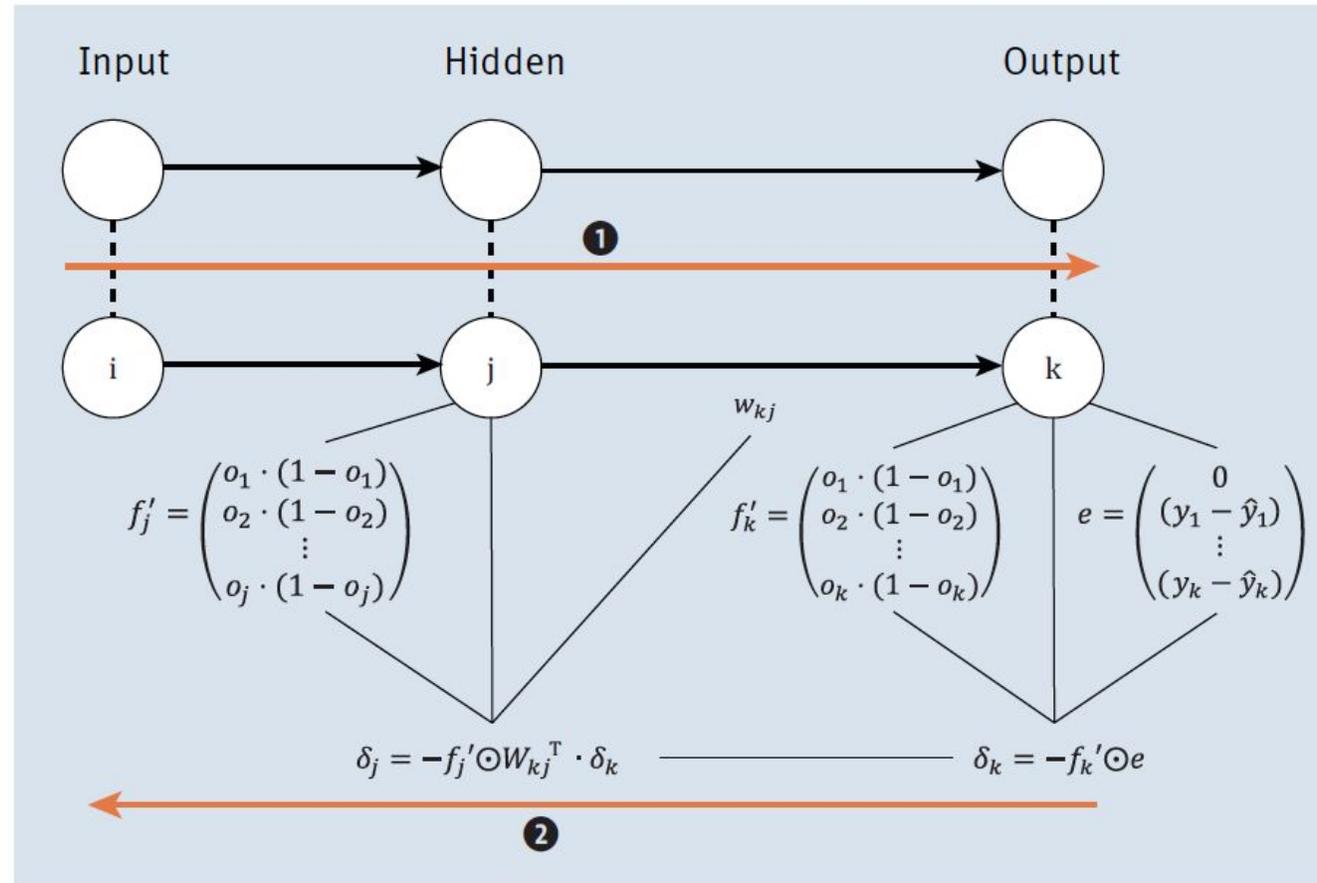


Abbildung 6.8 Vektor- und Matrixmultiplikationen für die Programmierung



6.2,3,4,5,6,7 BackProp





Fehlerkurve

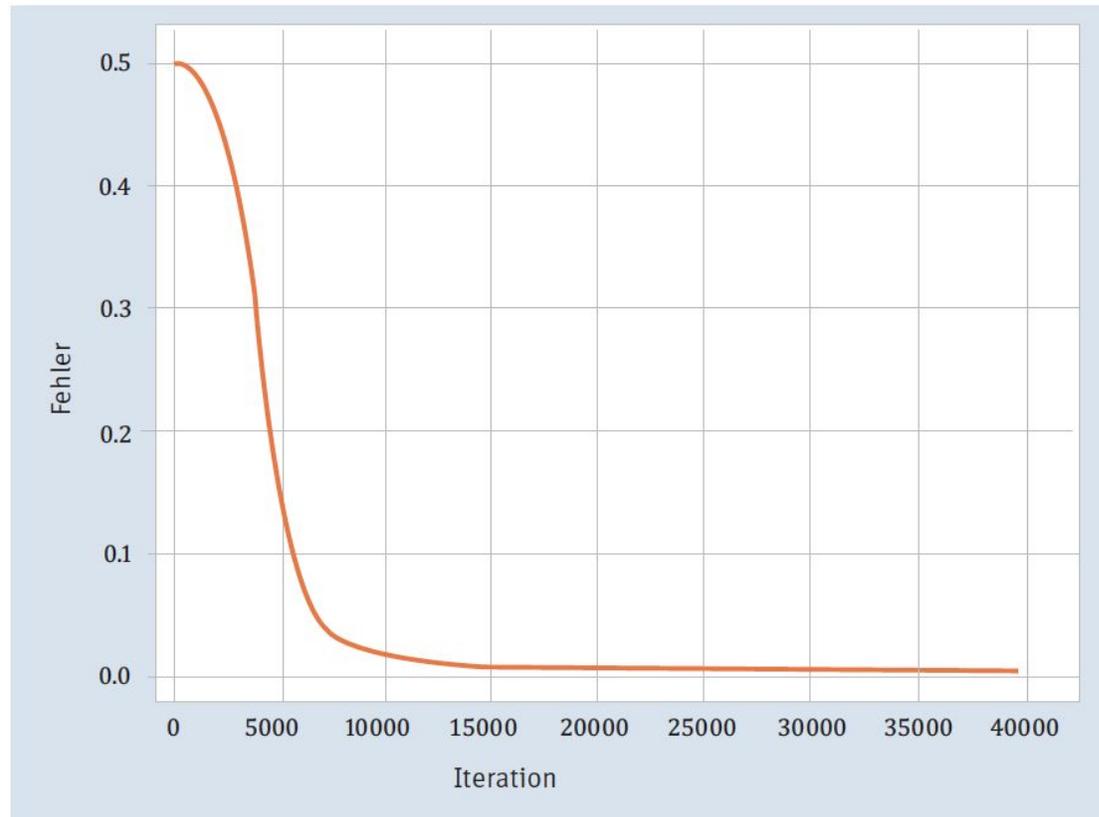


Abbildung 6.11 Was für eine schöne Fehlerkurve, wie aus dem Bilderbuch!



Listing 6.8 Output des Programms

```
# Ausgabe:
Multi-Layer-Perceptron - Netzwerkarchitektur
[[ 1.000  1.000  1.000  1.000  1.000]

[ 0.000  0.000  0.000  0.000  0.000]
[ 0.000  0.000  0.000  0.000  0.000]]
-----v-----
[[ 7.495 -5.248 -6.321]
 [ 2.108 -5.657 -5.653]
 [-6.578  4.224  4.223]]
-----v-----
[[ 1.000  1.000  1.000  1.000 -0.005]
 [ 2.108  0.892  0.892  0.097  0.001]
 [-6.578  0.001  0.001  0.001  0.000]]
-----v-----
[[ 0.416 -0.959  0.940]
 [ 4.037 -8.164 -8.261]]
-----v-----
[[ 0.000  0.000  0.000  0.000  0.000]
 [ -3.253  0.037  0.037  0.036 -0.001]]
-----v-----
Predict:
[ 1.000  1.000  1.000] 0.0 -> [ 0.042]
[ 1.000  0.000  1.000] 1.0 -> [ 0.957]
[ 1.000  1.000  0.000] 1.0 -> [ 0.957]
[ 1.000  0.000  0.000] 0.0 -> [ 0.037]
```

Listing 6.8 Output des Programms



Abbildung 6.12 Netzaufbau für die Iteration

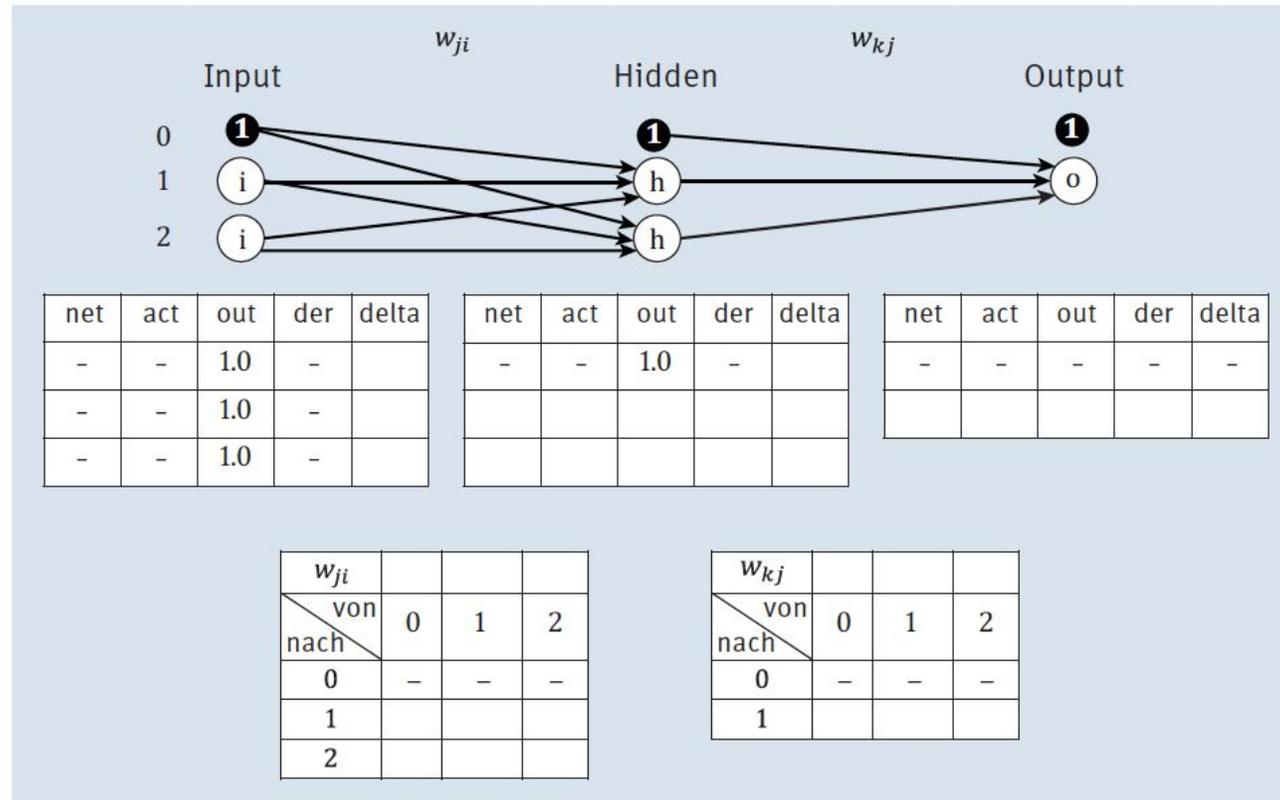


Abbildung 6.12 Netzaufbau für die Iteration



6.9 Learning Step





Ein Iterationsschritt

Input	net	act	out	der	delta	0	Input	net	act	out	der	delta
0	1.000	1.000	1.000	1.000	1.000		0	1.000	1.000	1.000	1.000	1.000
1	0.000	0.000	1.000	0.000	0.000		1	0.000	0.000	1.000	0.000	0.000
2	0.000	0.000	1.000	0.000	0.000	2	0.000	0.000	1.000	0.000	0.000	
W _{ij}						1	W _{ij}					
von							von					
nach	0	1	2				nach	0	1	2		
0	-0.251	0.901	0.464			0	-0.254	0.899	0.461			
1	0.197	-0.688	-0.688			1	0.198	-0.688	-0.688			
2	-0.884	0.732	0.202			2	-0.883	0.733	0.203			
Hidden	net	act	out	der	delta	2	Hidden	net	act	out	der	delta
0	1.000	1.000	1.000	1.000	1.000		0	1.000	1.000	1.000	1.000	-0.091
1	-1.179	0.235	0.235	0.180	0.000		1	-1.179	0.235	0.235	0.180	0.014
2	0.051	0.513	0.513	0.250	0.000	2	0.051	0.513	0.513	0.250	0.022	
W _{jk}						3	W _{jk}					
von							von					
nach	0	1	2				nach	0	1	2		
0	0.416	-0.959	0.940			0	0.416	-0.959	0.940			
1	0.665	-0.575	-0.636			1	0.661	-0.576	-0.638			
Output	net	act	out	der	delta	4	Output	net	act	out	der	delta
0	0.000	0.000	0.000	0.000	0.000		0	0.000	0.000	0.000	0.000	0.000
1	0.203	0.551	0.551	0.247	0.000		1	0.203	0.551	0.551	0.247	-0.136

Abbildung 6.13 Von alt zu neu, ein Iterationsschritt



6.10/11/12/13 Initialisierung





Output Schicht

ALT						Matrixindex	NEU					
Input	net	act	out	der	delta		Input	net	act	out	der	delta
0	1.000	1.000	1.000	1.000	1.000	0	0	1.000	1.000	1.000	1.000	1.000
1	0.000	0.000	1.000	0.000	0.000		1	0.000	0.000	1.000	0.000	0.000
2	0.000	0.000	1.000	0.000	0.000		2	0.000	0.000	1.000	0.000	0.000
W _{ji}						1	W _{ji}					
von					nach		0	1	2			
0	-0.251	0.901	0.464				0	-0.254	0.899	0.461		
1	0.197	-0.688	-0.688				1	0.198	-0.688	-0.688		
2	-0.884	0.732	0.202			2	-0.883	0.733	0.203			
Hidden	net	act	out	der	delta	2	Hidden	net	act	out	der	delta
0	1.000	1.000	1.000	1.000	1.000		0	1.000	1.000	1.000	1.000	-0.091
1	-1.179	0.235	0.235	0.180	0.000		1	-1.179	0.235	0.235	0.180	0.014
2	0.051	0.513	0.513	0.250	0.000	2	0.051	0.513	0.513	0.250	0.022	
W _{kj}						3	W _{kj}					
von					nach		0	1	2			
0	0.416	-0.959	0.940				0	0.416	-0.959	0.940		
1	0.665	-0.575	-0.665				1	0.661	-0.576	-0.638		
Output	net	act	out	der	delta	4	Output	net	act	out	der	delta
0	0.000	0.000	0.000	0.000	0.000		0	0.000	0.000	0.000	0.000	0.000
1	0.203	0.551	0.551	0.247	0.000		1	0.203	0.551	0.551	0.247	-0.136

diff = y - out

Abbildung 6.14 Berechnung der Output-Schicht



Delta der Hidden Schicht

ALT						Matrixindex	NEU					
Input	net	act	out	der	delta		Input	net	act	out	der	delta
0	1.000	1.000	1.000	1.000	1.000	0	0	1.000	1.000	1.000	1.000	1.000
1	0.000	0.000	1.000	0.000	0.000		1	0.000	0.000	1.000	0.000	0.000
2	0.000	0.000	1.000	0.000	0.000		2	0.000	0.000	1.000	0.000	0.000
W _{ji}						1	W _{ji}					
von \ nach							von \ nach					
0							0					
1							1					
2						2						
Hidden						2	Hidden					
net							net					
act							act					
out						out						
der						der						
delta						delta						
0	1.000	1.000	1.000	1.000	1.000	0	1.000	1.000	1.000	1.000	-0.091	
1	-1.179	0.235	0.235	0.180	0.000	1	-1.179	0.235	0.235	0.180	0.014	
2	0.051	0.513	0.513	0.250	0.000	2	0.051	0.513	0.513	0.250	0.022	
W _{kj}						3	W _{kj}					
von \ nach							von \ nach					
0							0					
1						1						
Output						4	Output					
net							net					
act							act					
out						out						
der						der						
delta						delta						
0	0.000	0.000	0.000	0.000	0.000	0	0.000	0.000	0.000	0.000	0.000	
1	0.203	0.551	0.551	0.247	0.000	1	0.203	0.551	0.551	0.247	-0.136	

Abbildung 6.15 Delta-Berechnung in der Hidden-Schicht



Gewichtsdeltas

ALT						Matrixindex	NEU					
Input	net	act	out	der	delta		Input	net	act	out	der	delta
0	1.000	1.000	1.000	1.000	1.000	0	0	1.000	1.000	1.000	1.000	1.000
1	0.000	0.000	1.000	0.000	0.000		1	0.000	0.000	1.000	0.000	0.000
2	0.000	0.000	1.000	0.000	0.000		2	0.000	0.000	1.000	0.000	0.000
W _{ji}						1	W _{ji}					
von							von					
nach							nach					
0	-0.251	0.901	0.464				0	-0.254	0.899	0.461		
1	0.197	-0.688	-0.688			1	0.198	-0.688	-0.688			
2	-0.884	0.732	0.202			2	-0.883	0.733	0.203			
Hidden						2	Hidden					
0	1.000	1.000	1.000	1.000	1.000		0	1.000	1.000	1.000	1.000	-0.091
1	-1.179	0.235	0.235	0.180	0.000		1	-1.179	0.235	0.235	0.180	0.014
2	0.051	0.513	0.513	0.250	0.000	2	0.051	0.513	0.513	0.250	0.022	
W _{kj}						3	W _{kj}					
von							von					
nach							nach					
0	0.416	-0.959	0.940			0	0.416	-0.959	0.940			
1	0.665	-0.575	-0.636			1	0.661	-0.576	-0.638			
Output						4	Output					
0	0.000	0.000	0.000	0.000	0.000		0	0.000	0.000	0.000	0.000	0.000
1	0.203	0.551	0.551	0.247	0.000	1	0.203	0.551	0.551	0.247	-0.136	

Abbildung 6.16 Berechnung des Gewichtsdeltas für W_{kj}



Kapitel 7

Convolutional Neural Networks



7.1 Aufbau eines CNN

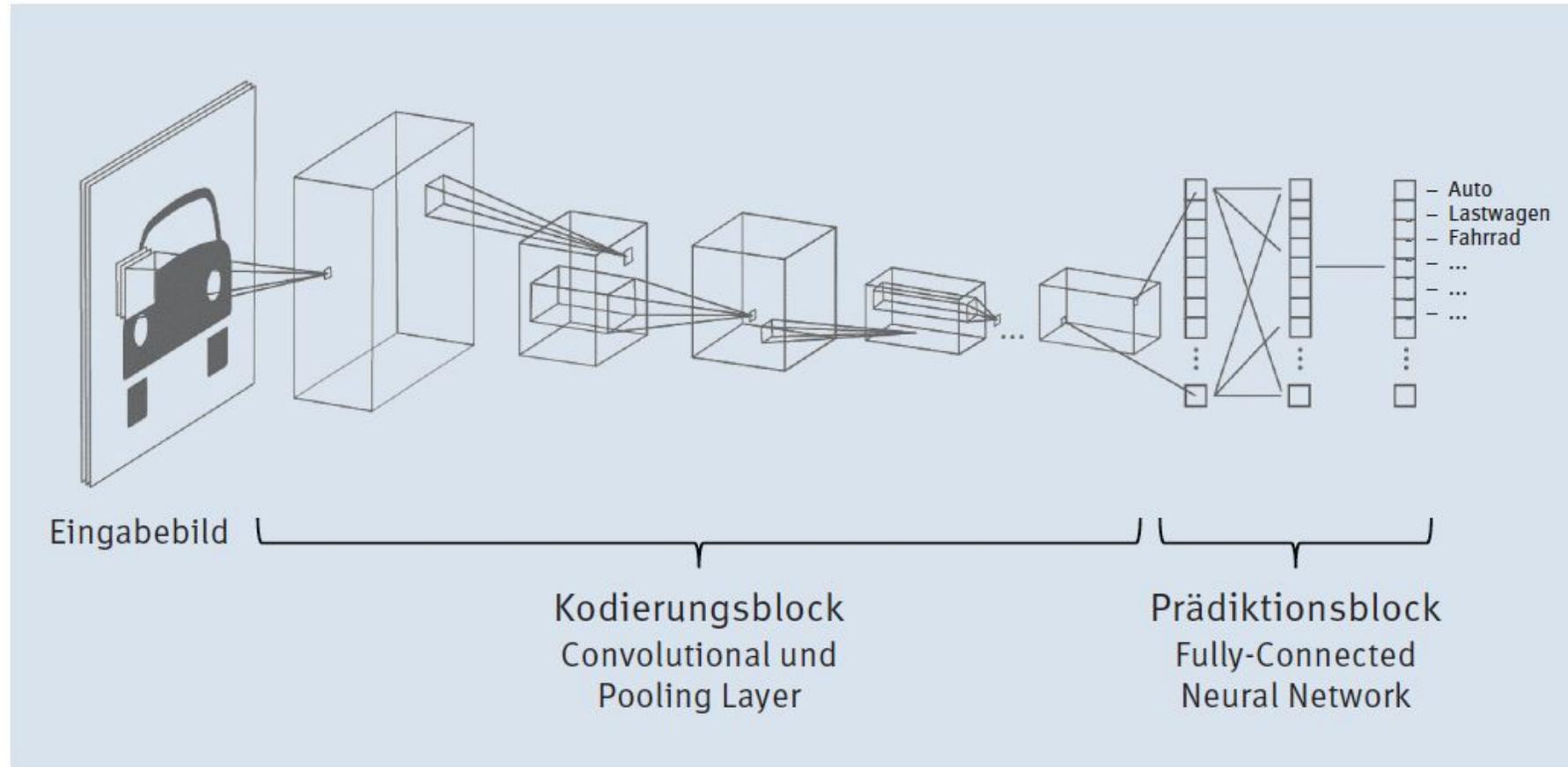


Abbildung 7.2 Die Struktur eines CNN mit Kodierungs- und Prädiktionsblock



7.1 Aufbau eines CNN

Convolution

Convolution?

Was bedeutet *Convolution*? Die deutsche Übersetzung wäre der mathematische Begriff der Konvolution oder Faltung, in unserem Fall genauer die diskrete Faltung. Mathematisch kann die Faltung als Produkt von zwei Funktionen verstanden werden. In der Bildverarbeitung bedeutet die diskrete Faltung das »Filtern« eines Bildes mit einer 3×3 - oder 5×5 -Matrix (siehe Abbildung 7.3), wobei es unterschiedliche Filtertypen gibt (Linienfilter, Kantenfilter, Weichzeichner etc.). Beim Convolutional Neural Network passiert in den ersten Ebenen genau das – das Eingangsbild wird »gefiltert«, um gewisse Merkmale zu betonen (Linien, Kanten, Punkte, Ecken etc.).



7.1 Aufbau eines CNN

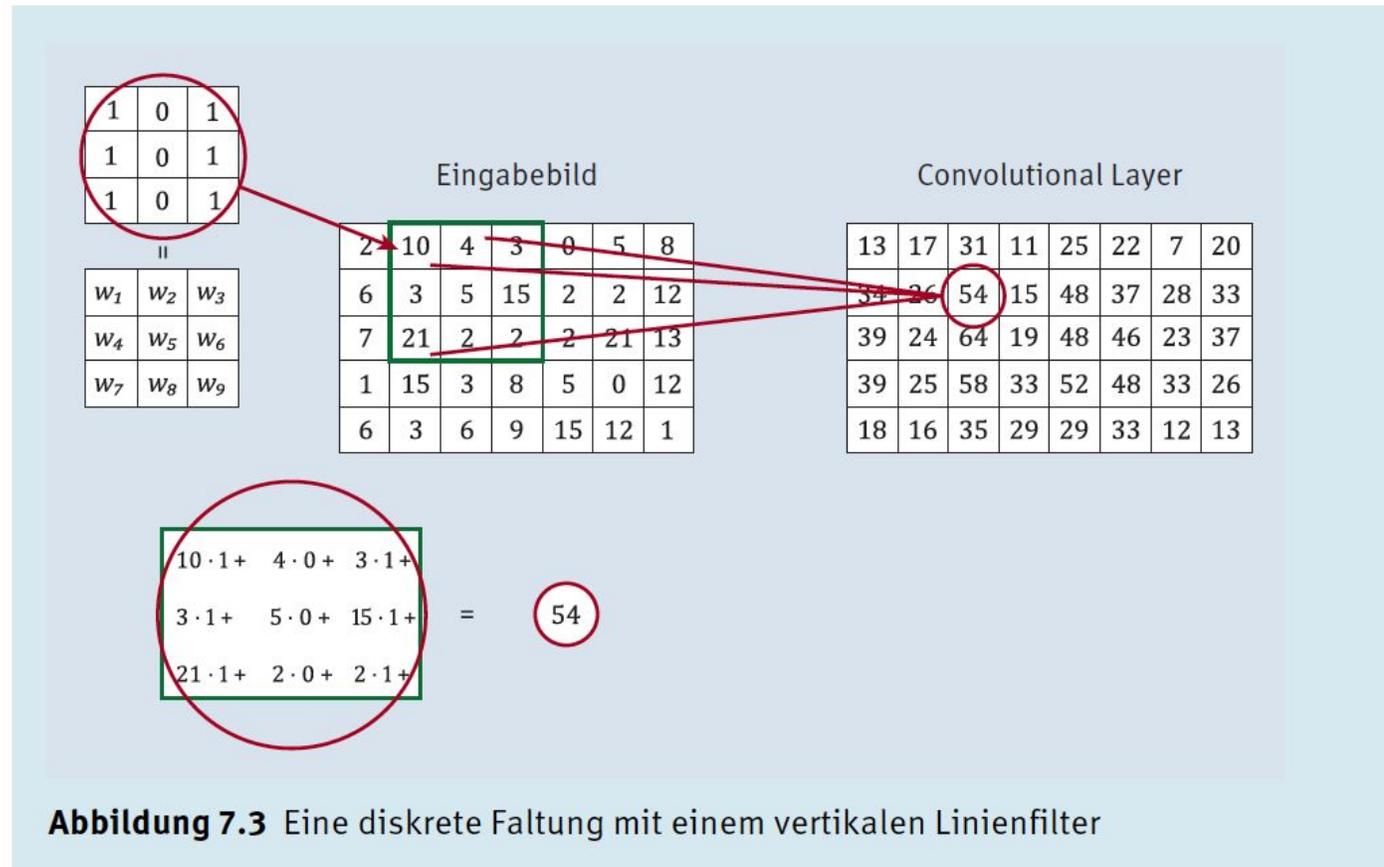


Abbildung 7.3 Eine diskrete Faltung mit einem vertikalen Linienfilter



7.2 Der Kodierungsblock

7.2.1 Convolutional Layer

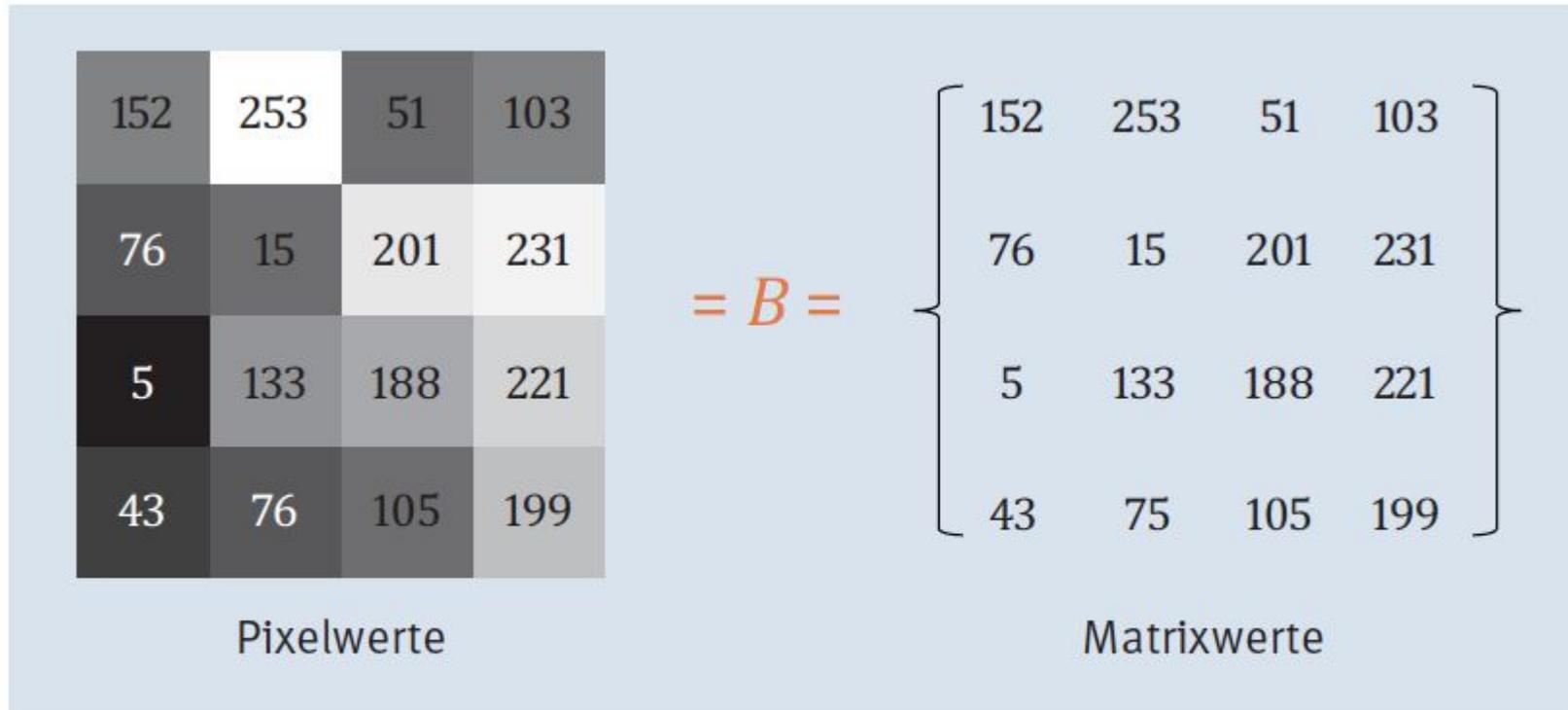


Abbildung 7.4 Vom Bild (Pixelwerte) zur Matrix



7.2.1 Convolutional Layer

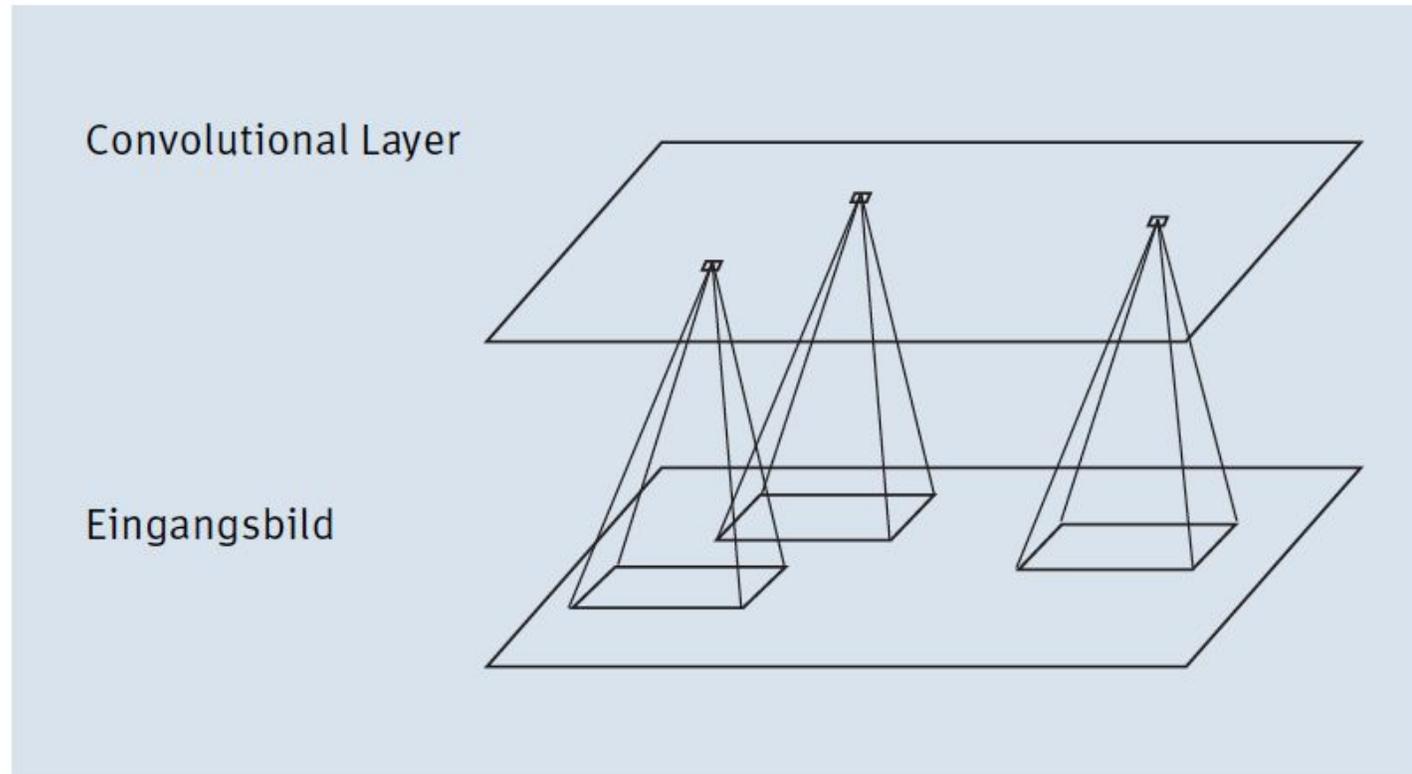


Abbildung 7.5 Der erste Convolutional Layer



7.2.1 Convolutional Layer

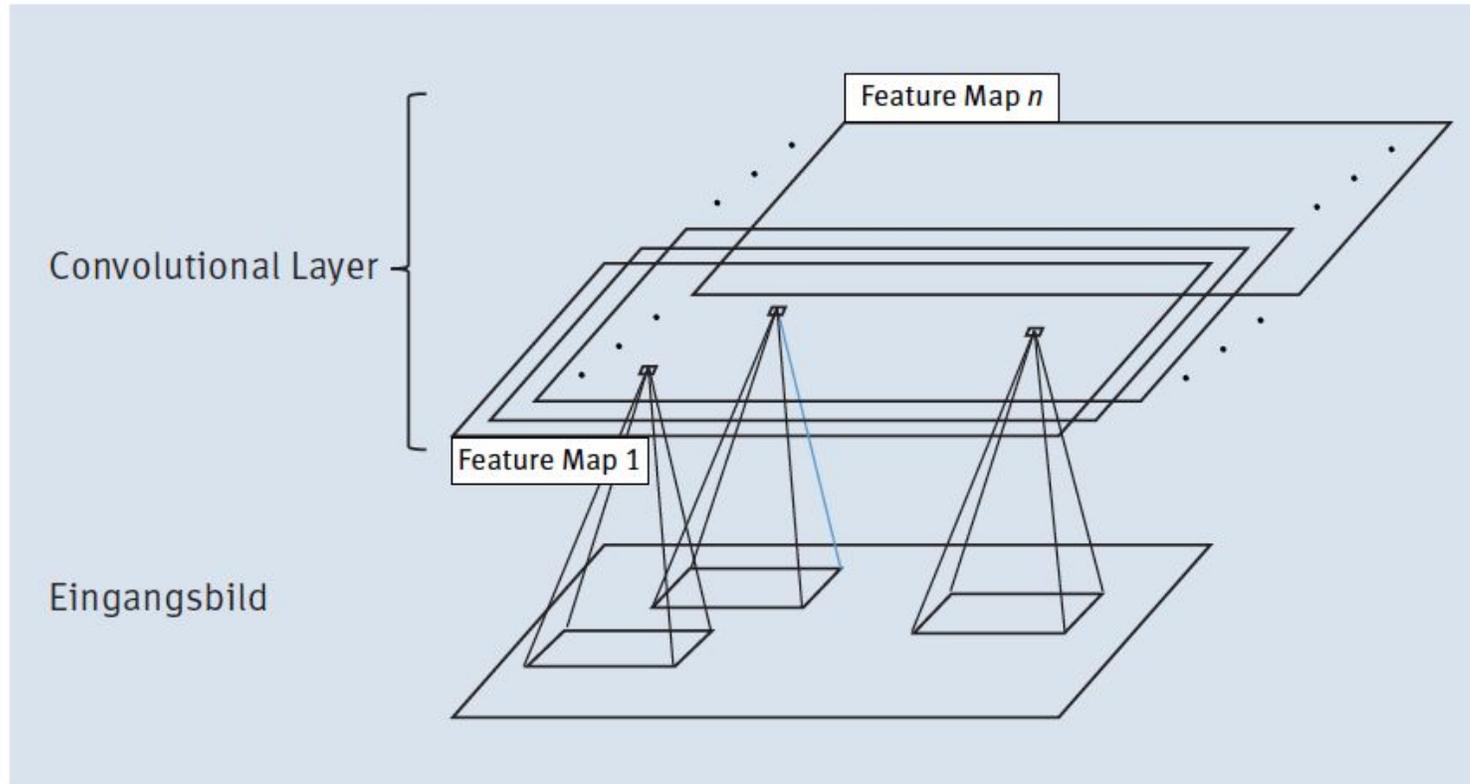


Abbildung 7.6 Ein Convolutional Layer besteht aus mehreren Feature Maps.



7.2.1 Convolutional Layer

Wie viele Feature Maps?

Woher weiß ich, wie viele Feature Maps ich brauche und auf welches Muster im Bild eine Feature Map reagiert? Nun, die richtige Anzahl zu finden, ist nicht so einfach und eine Sache der Erfahrung bzw. des Experimentierens. Die gute Nachricht ist, dass die Muster (also die Gewichte, die durch den 3×3 -Filter repräsentiert werden) durch das CNN selbstständig gelernt werden! Allerdings bedarf es dazu einer großen Anzahl an Bildern in der Trainingsphase – aber dazu später.



7.2.2 Activation Function

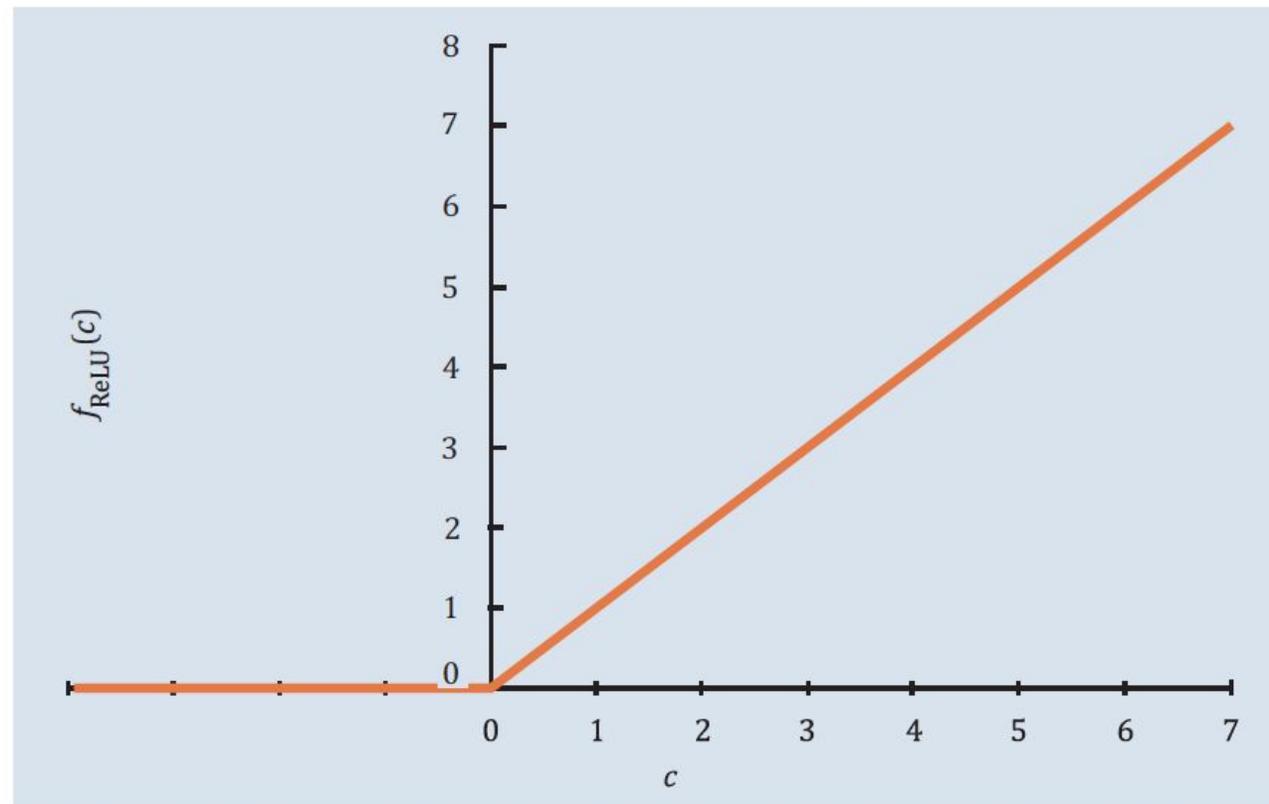


Abbildung 7.7 Darstellung der ReLU-Aktivierungsfunktion



Hyperparameter

Was ist ein Hyperparameter?

Jeder Algorithmus braucht gewisse Einstellungen – das können Parameter zur Definition der Aktivierungsfunktion sein oder überhaupt die Festlegung, welche Aktivierungsfunktion verwendet wird; auch die Bestimmung, welche Fehlerfunktion verwendet wird, oder ganz spezifische Parameter, die vom Algorithmus abhängen. Man kann sie mit Schaltern, Rädchen, Schraubchen vergleichen, die vor dem Start eingestellt werden müssen, aber das Ergebnis wesentlich beeinflussen können. Solche Parameter nennen wir *Hyperparameter*.

Es gibt natürlich Verfahren, die versuchen, selbstständig die optimalen Einstellungen zu finden. Die sind allerdings sehr aufwendig, da für jede Änderung eines Hyperparameters ein neues Training durchgeführt werden muss, und eignen sich daher nur für Aufgaben mit nicht allzu großer Datenmenge oder wenn die entsprechenden Rechenkapazitäten zur Verfügung stehen.



7.2.3 Pooling Layer

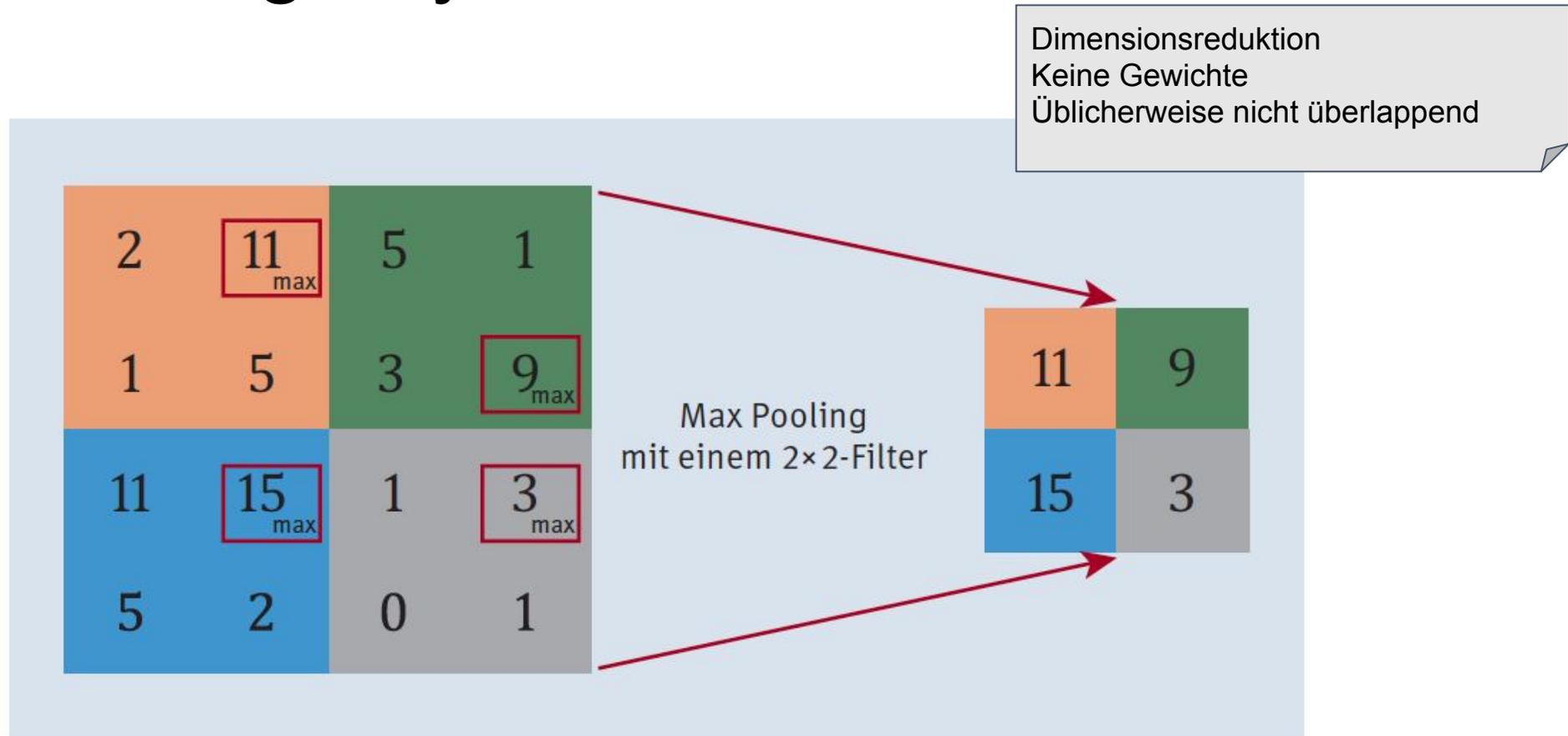


Abbildung 7.8 Beispiel für ein Max Pooling mit einem 2×2-Filter



7.2.4 Überlappen, ausfüllen und Schrittlänge Padding

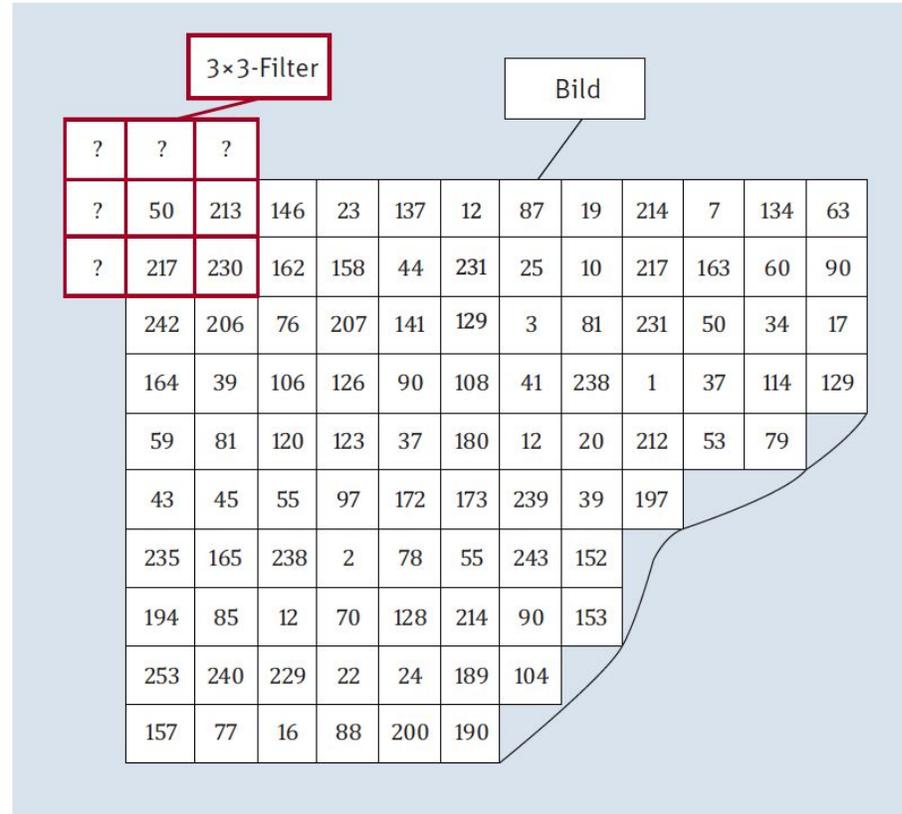


Abbildung 7.9 Filter – was tun mit Randpixeln?



7.2.4 Überlappen, ausfüllen und Schrittlänge Padding

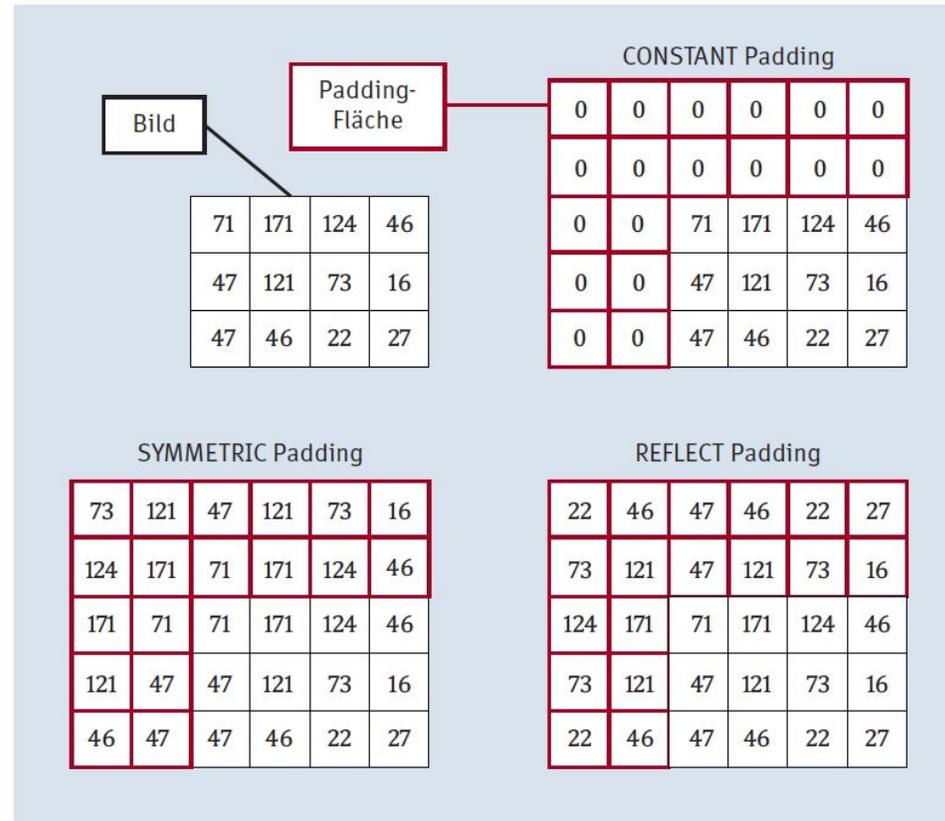


Abbildung 7.10 Padding-Beispiele für einen 5x5-Filter – TensorFlow



7.2.4 Überlappen, ausfüllen und Schrittlänge Stride für (Convolution und Pooling)

	115	142	13	18	40	156	5	106	21	135	87
Stride = 1 (mit Überlappen)	116	70	82	89	243	247	232	24	36	43	208
	50	55	198	219	115	54	205	66	216	159	129
	87	50	29	6	239	31	186	214	11	191	195
	36	112	158	26	147	32	96	185	96	7	69
Stride = 2 (mit Überlappen)	180	84	71	0	63	89	190	75	186	207	121
	12	253	1	226	108	88	53	62	20	52	207
	159	232	199	129	196	70	220	6	112	208	138
	59	2	34	159	214	175	55	24	10	159	62
Stride = 3 (ohne Überlappen)	213	131	58	69	236	152	53	129	226	13	218
	113	117	218	77	11	21	156	40	2	119	43
	250	242	91	241	29	245	208	14	255	17	99
	2	232	26	225	168	215	113	156	149	203	76

Abbildung 7.11 Die Schrittlänge = Stride eines Filters



7.3 Der Prädiktionsblock

7.3.1 Flatten

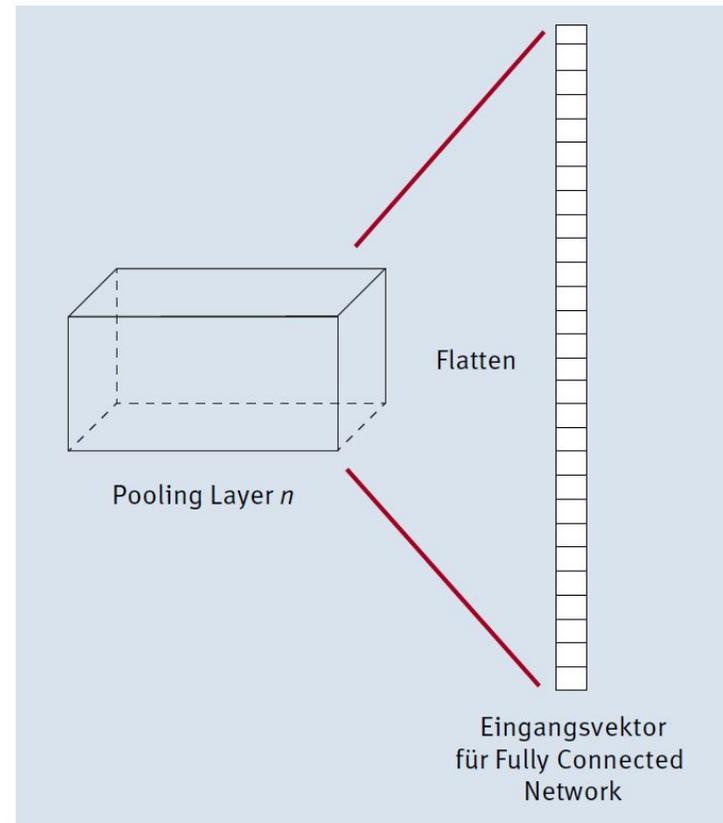


Abbildung 7.12 Flattening des letzten Blocks im Kodierungsblock



7.3.2 Softmax

Wenn wir nun die Klasse bestimmen wollen, dann nehmen wir einfach die Klasse mit dem höchsten Wert. Sind wir aber an der Klassenwahrscheinlichkeit interessiert, so empfiehlt sich Softmax als Aktivierungsfunktion für den Logits Layer. Softmax transformiert die Werte aus dem Logits Layer in einen Wertebereich von 0 und 1, aber so, dass die Summe aller Werte gleich 1 ergibt.



Abbildung 7.13 Beispiel für Softmax für drei Klassen

Softmax kann auch als Aktivierungsfunktion gesehen werden und wird bei k Klassen und $i = 1, \dots, k$ für die Klasse c_j folgendermaßen definiert:

$$\text{Softmax: } f_{\text{akt}}(c_j) = f_{\text{softmax}}(c_j) = \frac{e^{c_j}}{\sum_{i=1}^k e^{c_i}}$$



7.4 Trainieren von Convolutional Neural

Viele dieser Herausforderungen resultieren aus der Größe dieser Convolutional Neural Networks. Wobei mit »Größe« hier die Anzahl der Neuronen, Gewichte oder Verbindungen gemeint ist. Tabelle 7.1 zeigt einige bekannte CNN und die Anzahl ihrer Parameter.

Name des Convolutional Neural Networks	Anzahl Parameter
LeNet5 (1998)	~ 60.000
AlexNet (2012)	~ 60.000.000
VGG (2014)	~ 138.000.000
GoogleNet (2014)	~ 4.000.000
ResNet50 (2015)	~ 2.400.000

Tabelle 7.1 Vergleich von CNN anhand der Anzahl der Parameter



7.4.1 Das Problem der explodierenden/verschwindenden Gradienten Initialisierung

SCHNITT FÜR SCHNITT RICHTUNG EINGABESCHICHT UND BERECHNET DABEI DEN FEHLERGRADIENTEN. Wird der Fehlergradient nach jeder Schicht bestimmt, dann können die Gewichte des Netzes entsprechend angepasst werden. Unangenehmerweise werden diese Gradienten immer kleiner, je näher man der Eingabeschicht kommt, was dazu führt, dass das Training nicht oder nur sehr langsam zu einer Lösung konvergiert. In diesem Fall spricht man vom *verschwindenden Gradienten*. Umgekehrt kann es auch – wenn auch seltener und nur in bestimmten Netzformen – dazu kommen, dass die Gewichte der Schichten extrem stark verändert werden und dadurch das Training divergiert, was dann auch als Problem *explodierender Gradienten* bezeichnet wird.

Aktivierungsfunktion	Gleichverteilung $(-v, v)$	Normalverteilung
Sigmoidfunktion	$v = \sqrt{\frac{6}{n_{\text{ein}} + n_{\text{aus}}}}$	$\sigma = \sqrt{\frac{2}{n_{\text{ein}} + n_{\text{aus}}}}$
Tangens hyperbolicus	$v = 4 \sqrt{\frac{6}{n_{\text{ein}} + n_{\text{aus}}}}$	$\sigma = 4 \sqrt{\frac{2}{n_{\text{ein}} + n_{\text{aus}}}}$
ReLU	$v = \sqrt{2} \sqrt{\frac{6}{n_{\text{ein}} + n_{\text{aus}}}}$	$\sigma = \sqrt{2} \sqrt{\frac{2}{n_{\text{ein}} + n_{\text{aus}}}}$

Wegen dieser Probleme verschwanden die Deep-Learning-Netze lange Zeit in der Schublade. Die Autoren Xavier Glorot und Yoshua Bengio trugen mit ihrer Arbeit dazu bei, die Ursachen dieses Verhaltens besser zu verstehen, die in einer ungünstigen Kombination von Initialisierung (die Gewichte müssen ja initial irgendeinen Wert haben)

Tabelle 7.2 Gleichverteilung und Normalverteilung für die zufällige Initialisierung der Gewichte

Die Zufallswerte sollen so gewählt werden, dass sie einer Normalverteilung mit Mittelwert 0 und einer Standardabweichung bzw. einer Gleichverteilung zwischen $-v$ und $+v$ entsprechen. Dabei sind sowohl die Standardabweichung als auch der Wert v von der



Gaußsche Glockenkurve

Jeder Generator von Zufallswerten muss so funktionieren, dass die Verteilung eingehalten wird. Das bedeutet für eine Normalverteilung mit Mittelwert 0 und den in Tabelle 7.2 definierten Standardabweichungen, dass 64,26 % der Gewichtswerte zwischen -1σ und $+1\sigma$ liegen müssen.

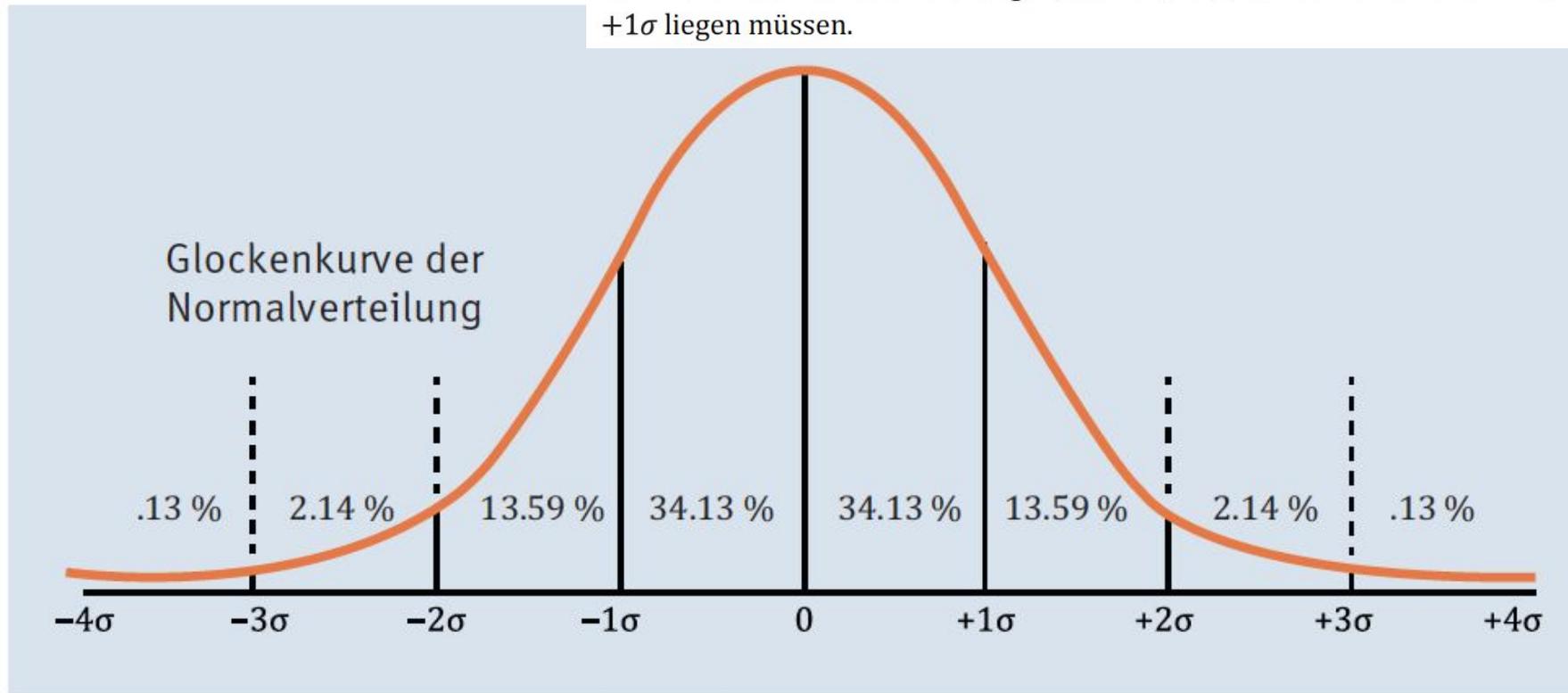


Abbildung 7.14 Typische Normalverteilung



Aktivierungsfunktion

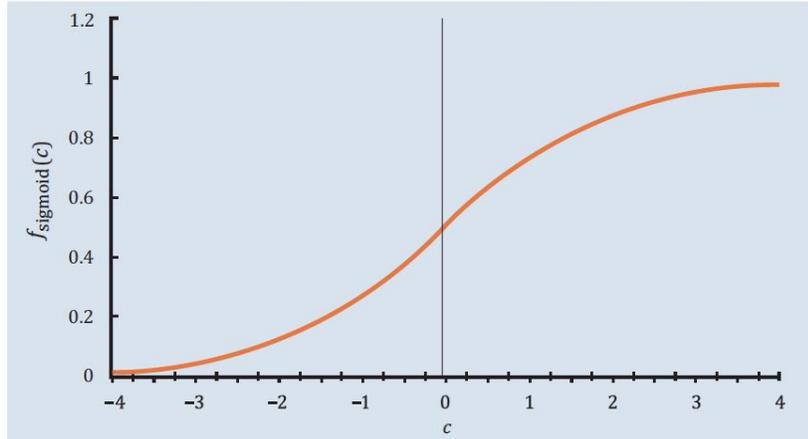


Abbildung 7.15 Sigmoidfunktion

lReLU: $f_{\text{akt}}(x) = f_{\text{lReLU}}(x) = \max(\alpha x, x)$, wo $\alpha = 0.01$ (sehr klein)

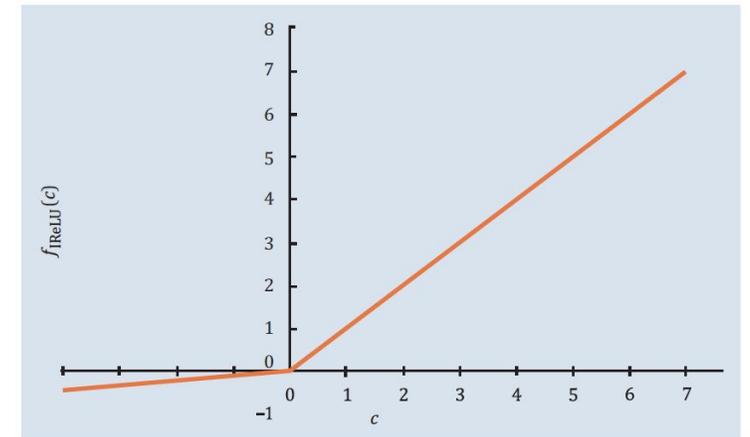


Abbildung 7.16 Die Leaky-ReLU-Aktivierungsfunktion

$$\text{ELU: } f_{\text{akt}}(x) = f_{\text{ELU}}(x) = \begin{cases} \alpha(e^x - 1), & \text{falls } x < 0 \\ x, & \text{falls } x \geq 0 \end{cases}$$

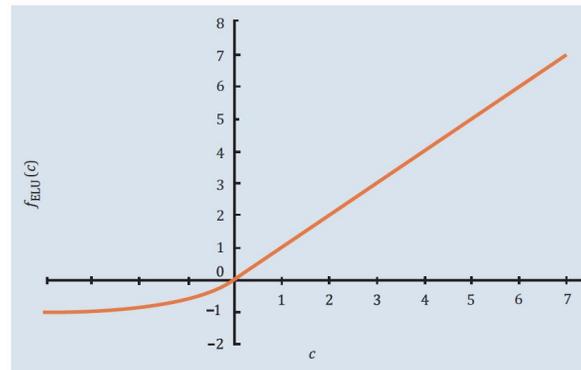


Abbildung 7.17 Die ELU-Aktivierungsfunktion



7.4.2 Das Optimierungsverfahren

Momentum-Optimierung

Adam-Optimierung

$$\nabla_W C(W) = \begin{pmatrix} \frac{\partial}{\partial W_0} C(W) \\ \frac{\partial}{\partial W_1} C(W) \\ \vdots \\ \frac{\partial}{\partial W_n} C(W) \end{pmatrix}$$

Das bedeutet also, vom Gewichtsvektor wird ein Veränderungsvektor \vec{h} abgezogen. Dieser Veränderungsvektor sieht beim Standard-Gradientenverfahren folgendermaßen aus:

$$\vec{h} = \eta \nabla_W C(W)$$

Dabei ist η unsere schon bekannte Lernrate. Die neuen Gewichtsvektoren erhalten wir also durch Subtraktion des Veränderungsvektors:

$$W^{\text{neu}} = W^{\text{alt}} - \vec{h}$$

Die folgenden beiden Beispiele von Optimierungsvarianten modifizieren diesen Veränderungsvektor, um das Training schneller konvergieren zu lassen.

$$\vec{h}_{\text{neu}} = \eta_{\text{momentum}} \vec{h}_{\text{alt}} + \eta \nabla_W C(W)$$
$$W_{\text{neu}} = W_{\text{alt}} - \eta_{\text{Adam}}(h_{\text{alt}}, h_{\text{alt}}^2)$$



7.4.3 Verhindern von Overfitting

Early Stopping

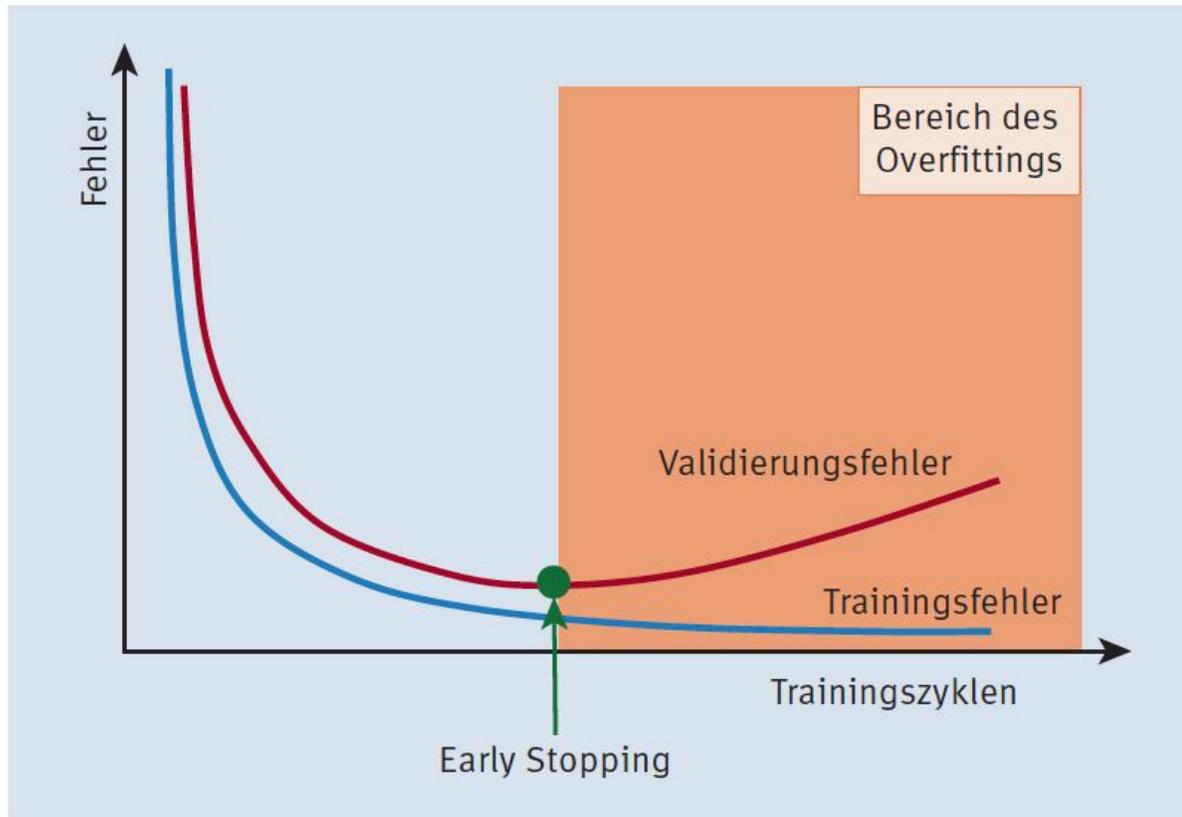
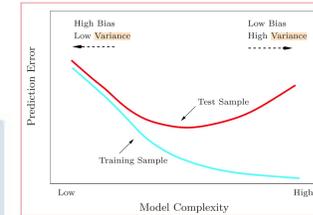


Abbildung 7.18 Fehlerkurve bei Early Stopping



Bias-Variance Tradeoff
Underfitting-Overfitting

Bias is the difference in the model prediction and the actual value.

Bias is the difference in the model prediction and the actual value.

High Bias (UNDESIRABLE): This type of model always predicts the same output every time or takes a random guess while prediction.

Low Bias/High Variance (UNDESIRABLE): A model that overfits the training data. This performs poorly on unseen data.

Low Bias/Low Variance (DESIRABLE): A model that almost always gives the best results. This model performs well on seen and unseen examples.



7.4.3 Verhindern von Overfitting

Dropout

Graceful degradation

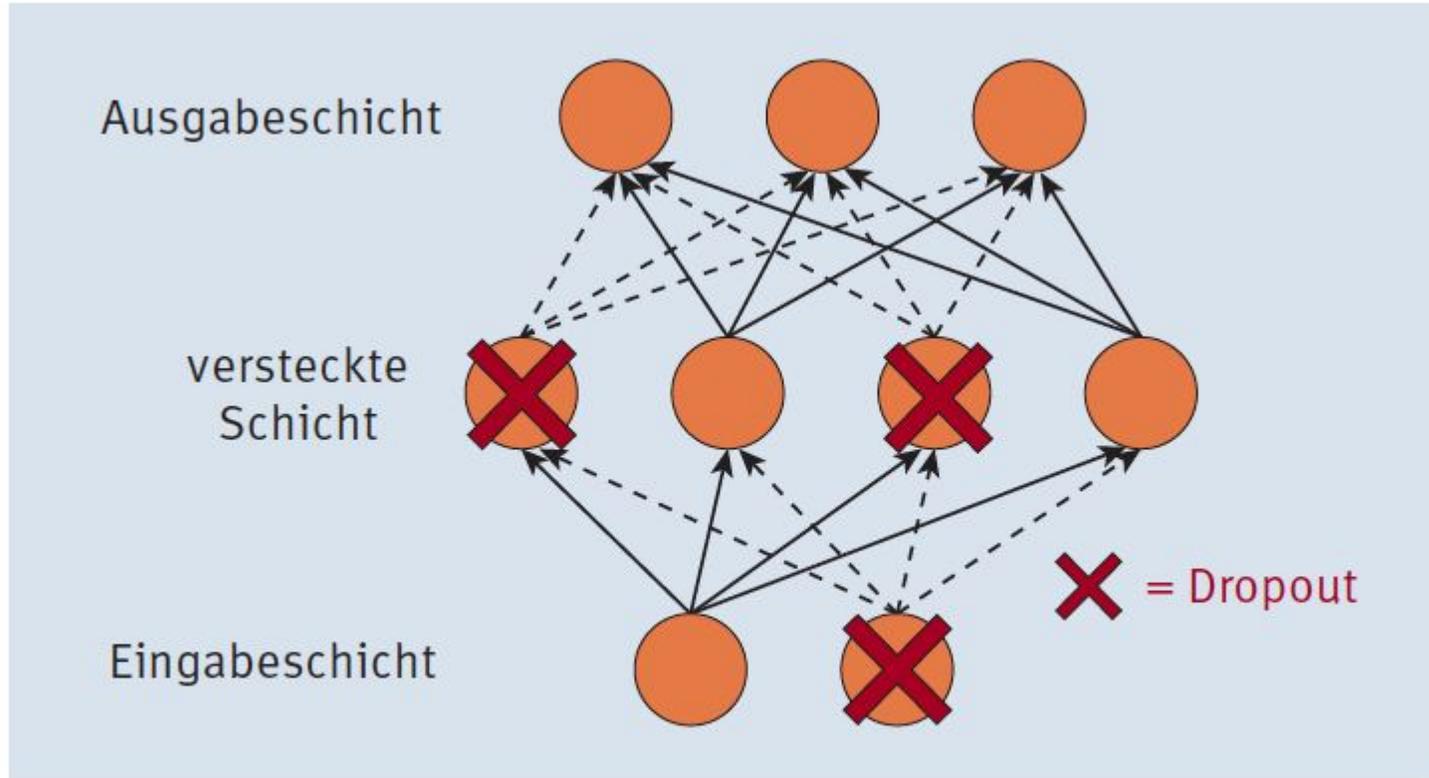


Abbildung 7.19 Netz mit Dropout-Neuronen (Dropout-Rate $p = 50\%$)



Kapitel 8

Programmierung von Convolutional Neural Networks mit TensorFlow 2.0



8.1 Convolutional Networks zur Handschriftenerkennung

8.1.1 Der Datensatz



Abbildung 8.1 Ausschnitt aus dem MNIST-Datensatz



Keras

Die Keras-Bibliothek verdient eine besondere Erwähnung. Sie wurde von François Chollet entwickelt. Keras ist eine Python-Bibliothek, die eine einheitliche Schnittstelle für verschiedene Bibliotheken für neuronale Netzwerke bietet, wie TensorFlow, CNTK oder Theano. Das Ziel von Keras ist, die Anwendung dieser Bibliotheken so einsteiger- und benutzerfreundlich wie möglich zu machen. Obwohl Keras als eigenständige Bibliothek erhalten bleibt, wurde sie ab der TensorFlow-Version 1.4 integriert und mit der Version 2.0 noch mehr an TensorFlow angepasst und erweitert.



8.1 Laden des MNIST-Datensatzes aus der » tensorflow.keras«-Bibliothek





8.2 Laden und Vorverarbeitung des MNIST-Datensatzes





Listing 8.3 Format des modifizierten Trainings- und Testdatensatzes





Listing 8.4 Darstellung eines zufällig ausgewählten MNIST-Bildes





8.1.2 Ein einfaches CNN

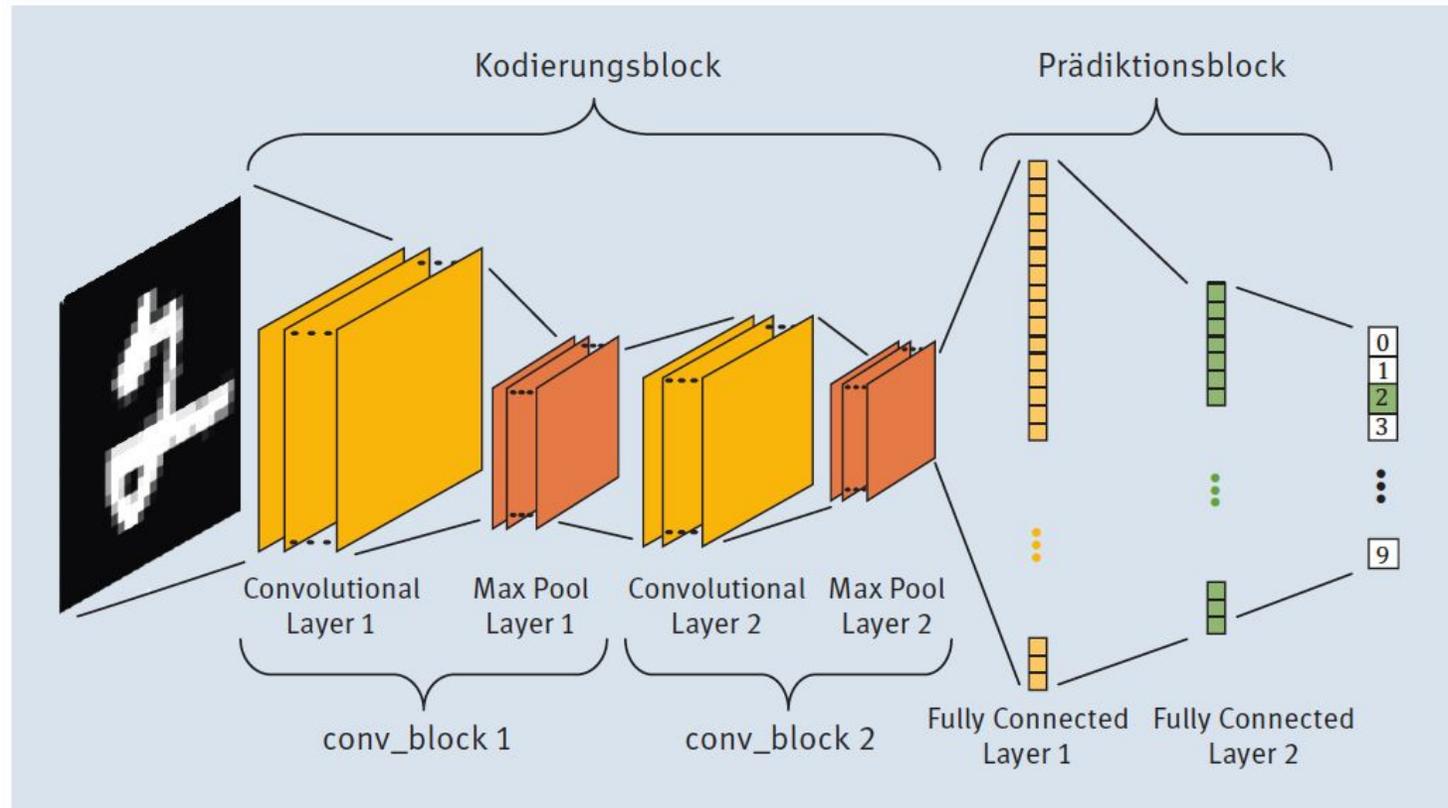


Abbildung 8.3 Die Struktur unseres CNN für den MNIST-Datensatz



Listing 8.5 Import der Bibliotheken





Das Modell

Listing 8.6 Codeabschnitt für die Netzwerkarchitektur





Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 24, 24, 32)	832
max_pooling2d_8 (MaxPooling2)	(None, 12, 12, 32)	0
conv2d_9 (Conv2D)	(None, 8, 8, 64)	51264
max_pooling2d_9 (MaxPooling2)	(None, 4, 4, 64)	0
dropout_8 (Dropout)	(None, 4, 4, 64)	0
flatten_4 (Flatten)	(None, 1024)	0
features (Dense)	(None, 128)	131200
dropout_9 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 64)	8256
dense_8 (Dense)	(None, 10)	650
Total params: 192,202		
Trainable params: 192,202		
Non-trainable params: 0		

Abbildung 8.4 Ausgabe der Funktion »model.summary()«



Listing 8.7 Festlegung Verlustfunktion und Optimierung





Das Training

Listing 8.8 Training des Modells mit »
`model.fit()`«





8.1.3 Die Ergebnisse

Tensorboard

Um die Ergebnisse selbst mittels TensorBoard nachzuvollziehen, können Sie es in einem Terminal (am besten aus Anaconda Navigator in der entsprechenden Environment) selbst starten mit dem Befehl:

```
tensorboard -logdir "logs"
```

Achtung! Sie müssen im Terminal zuerst in das Verzeichnis wechseln, in dem sich auch das Jupyter Notebook befindet. Statt "logs" müssen Sie das Verzeichnis nehmen, das Sie in [Listing 8.8](#) für die Variable LOG_DIR gewählt haben.

Danach öffnen Sie einen Webbrowser und starten die TensorBoard-Webapplikation mit der URL <http://localhost:6006/> (oder, wenn das nicht funktioniert, mit <http://rechner-name:6006>)

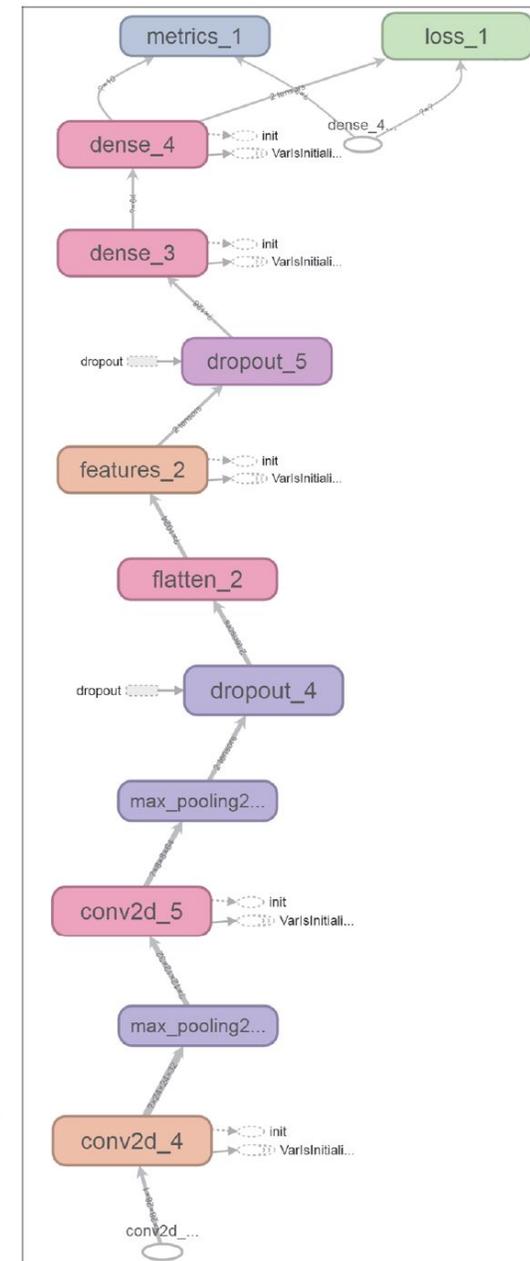


Abbildung 8.5 Unser Convolutional Neuronal Network im TensorBoard (Ausschnitt)



Listing 8.9 Verlust und Genauigkeit unseres trainierten CNN



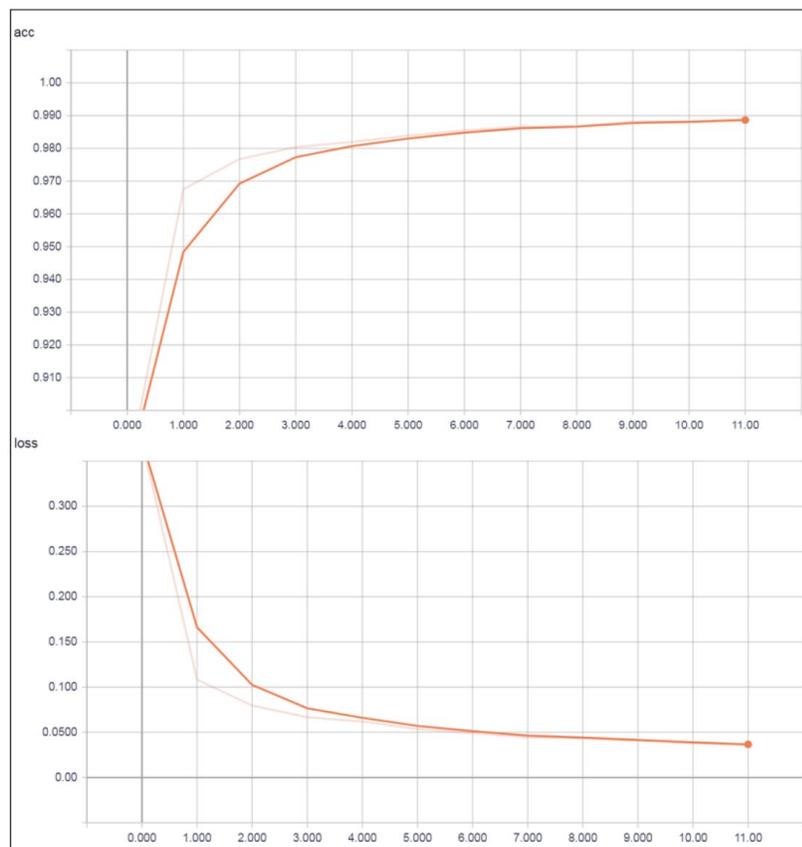


Abbildung 8.6 Verlauf Genauigkeit (»acc«) und Verlust (»loss«) der Trainingsdaten



Abbildung 8.7 Verlauf Genauigkeit (»acc«) und Verlust (»loss«) der Testdaten



Listing 8.10 Speichern des Modells und der Gewichte (h5-format)





Listing 8.11 Speichern des Modells und der Gewichte (pb-format)





Listing 8.12 Laden und Verwenden des vortrainierten Modells





Listing 8.13 Vergleich der Ergebnisse und Prüfung des trainierten Netzes



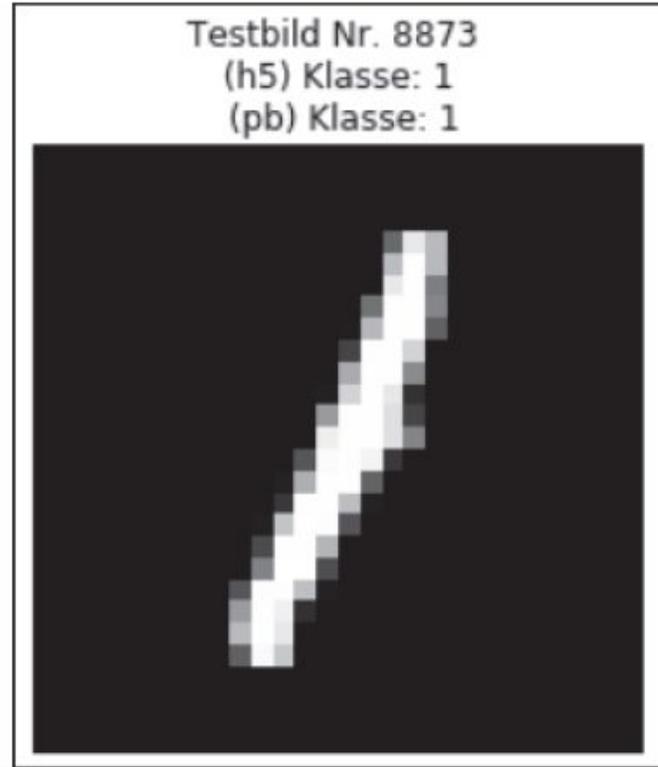


Abbildung 8.8 Klassifikationsergebnis eines Testbildes



8.2 Transfer Learning mit Convolutional Neural Networks

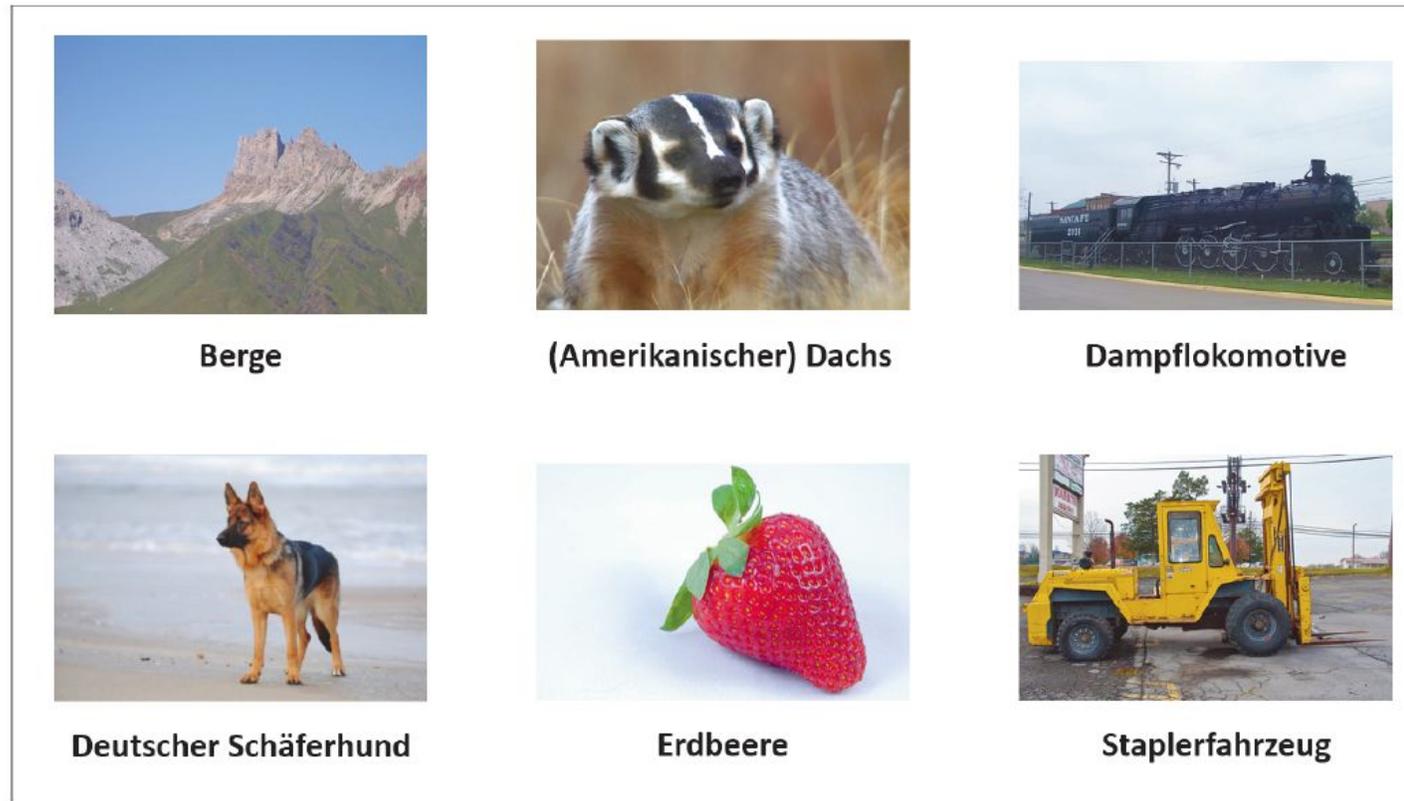


Abbildung 8.9 Beispielbilder für unsere Aufgabe

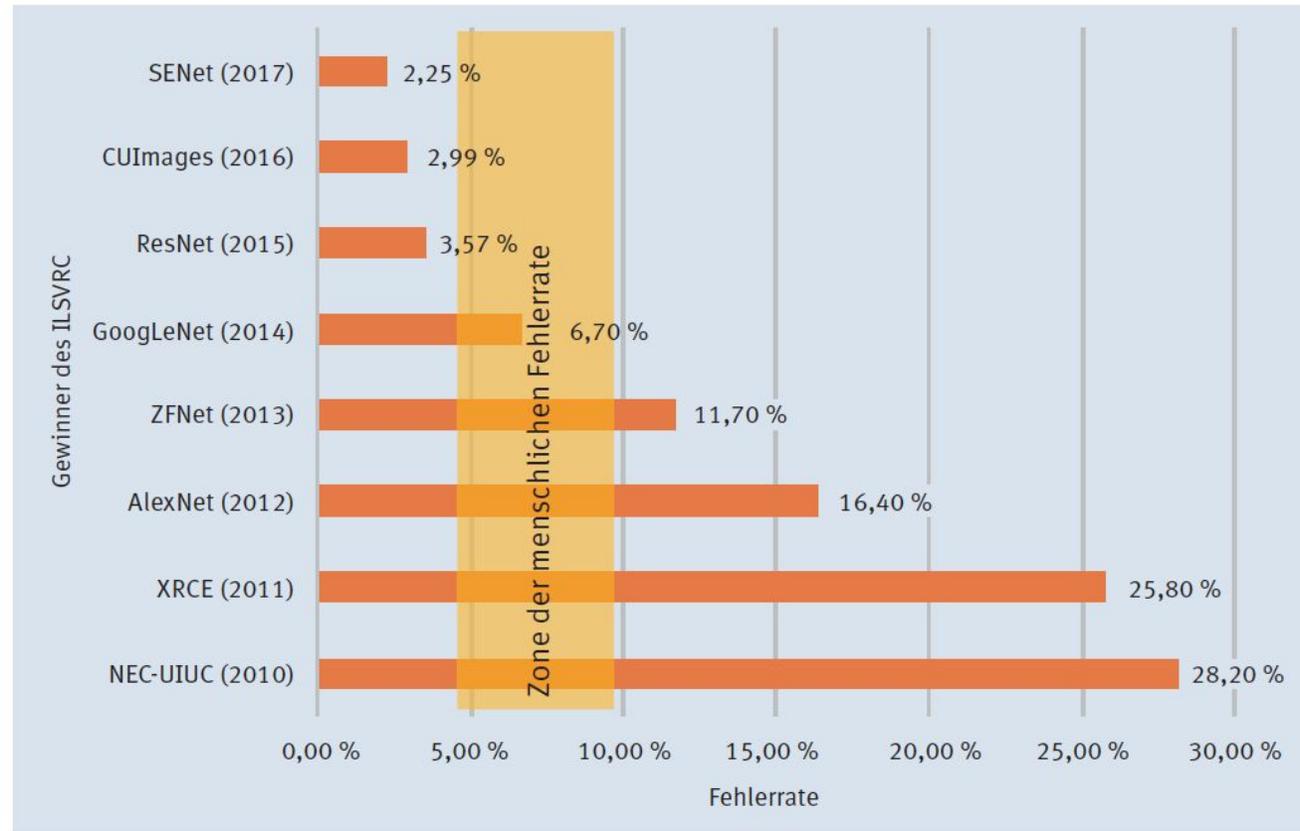


Abbildung 8.10 Gewinner des ILSVRC seit 2010 mit Angabe der Fehlerrate



8.2.1 Das vortrainierte Netzwerk

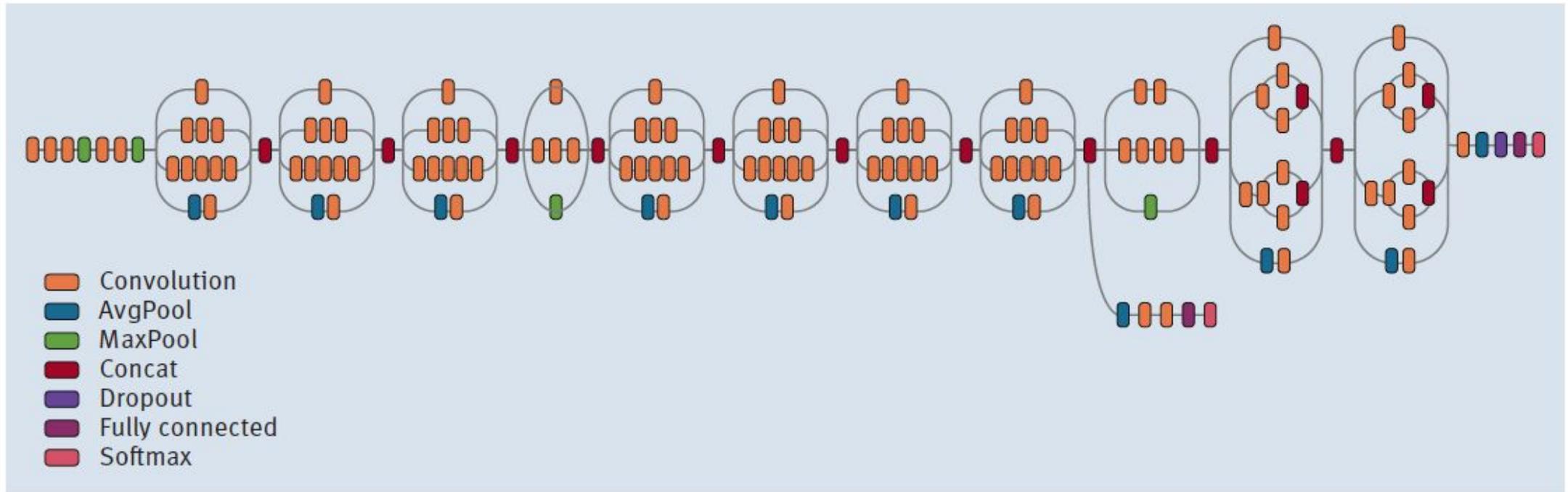


Abbildung 8.11 Die Netzwerkstruktur von Inception-v3

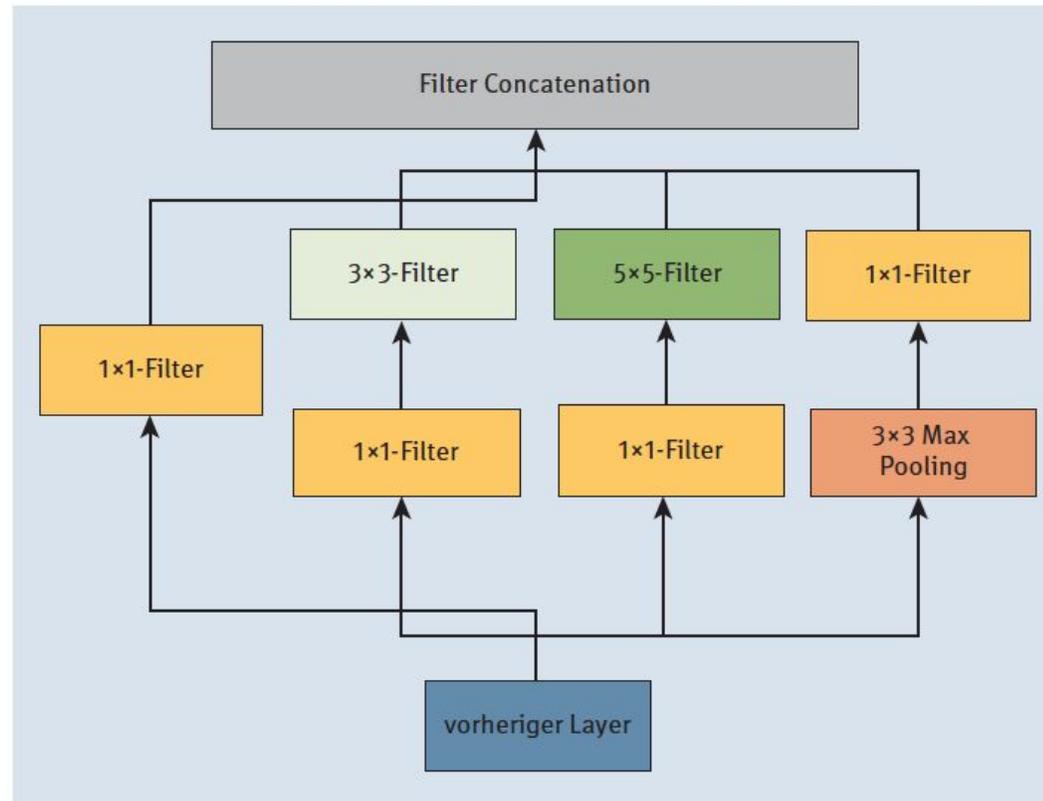


Abbildung 8.13 Inception-Modul, aus dem Inception-v3 aufgebaut ist



8.2.2 Datenvorbereitung

Listing 8.14 Import der Bibliotheken für neuronale Netze





Listing 8.15 Wichtige Helferlein für die Bildbearbeitung





8.2.3 Das vortrainierte Netz

Listing 8.16 Laden und Anzeigen des Modells Inception-v3





Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, None, None, 3 0)		
conv2d_1 (Conv2D)	(None, None, None, 3 864)		input_1[0][0]
batch_normalization_1 (BatchNor	(None, None, None, 3 96)		conv2d_1[0][0]
activation_1 (Activation)	(None, None, None, 3 0)		batch_normalization_1[0][0]
conv2d_2 (Conv2D)	(None, None, None, 3 9216)		activation_1[0][0]
batch_normalization_2 (BatchNor	(None, None, None, 3 96)		conv2d_2[0][0]
activation_2 (Activation)	(None, None, None, 3 0)		batch_normalization_2[0][0]
conv2d_3 (Conv2D)	(None, None, None, 6 18432)		activation_2[0][0]
concatenate_2 (Concatenate)	(None, None, None, 7 0)		activation_92[0][0] activation_93[0][0]
activation_94 (Activation)	(None, None, None, 1 0)		batch_normalization_94[0][0]
mixed10 (Concatenate)	(None, None, None, 2 0)		activation_86[0][0] mixed9_1[0][0] concatenate_2[0][0] activation_94[0][0]
avg_pool (GlobalAveragePooling2	(None, 2048)	0	mixed10[0][0]
predictions (Dense)	(None, 1000)	2049000	avg_pool[0][0]
Total params: 23,851,784			
Trainable params: 23,817,352			
Non-trainable params: 34,432			

Abbildung 8.14 Ausschnitt der Inception-v3-Netzwerkstruktur



Listing 8.17 Laden des zu klassifizierenden Bilds





Abbildung 8.15 Bild eines Deutschen Schäferhunds (Budinho, CC BY-SA 3.0, https://commons.wikimedia.org/wiki/File:Grauer_Deutscher_Schäferhund_Standbild.jpg)



Listing 8.18 Weitere Bildvorverarbeitung und Ausgabe der Form und der Pixelwerte





8.2.4 Die Ergebnisse

Listing 8.19 Klassifikation des Bildes





Listing 8.20 Das klassifizierte Bild mit Klassenlegende



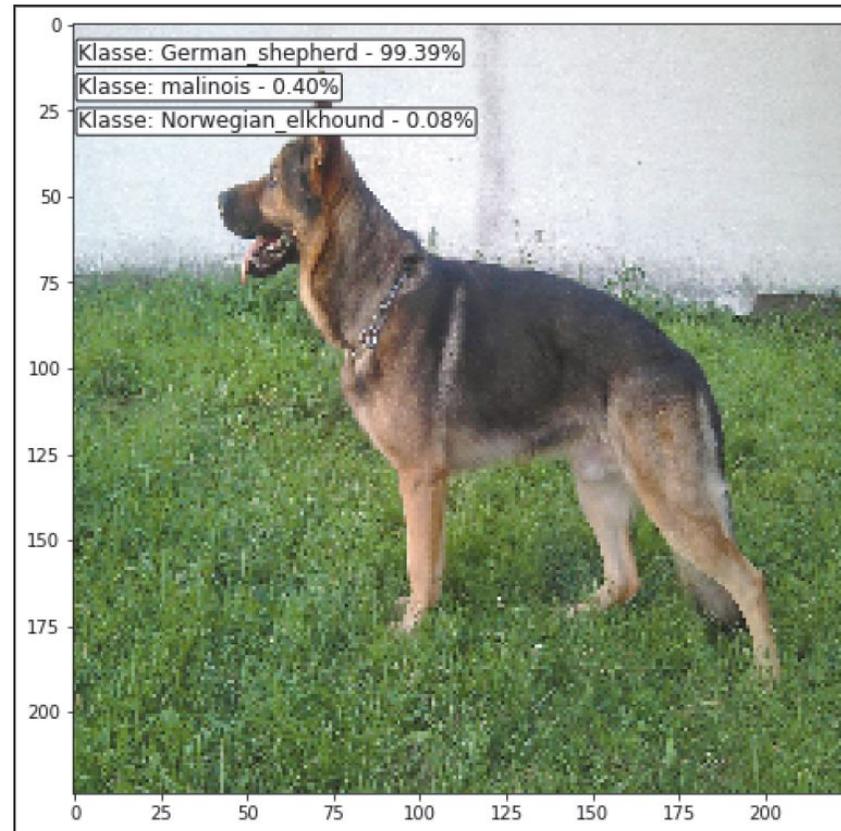


Abbildung 8.16 Klassifikationsergebnis



Kapitel 10

Die Evolution der neuronalen Netze

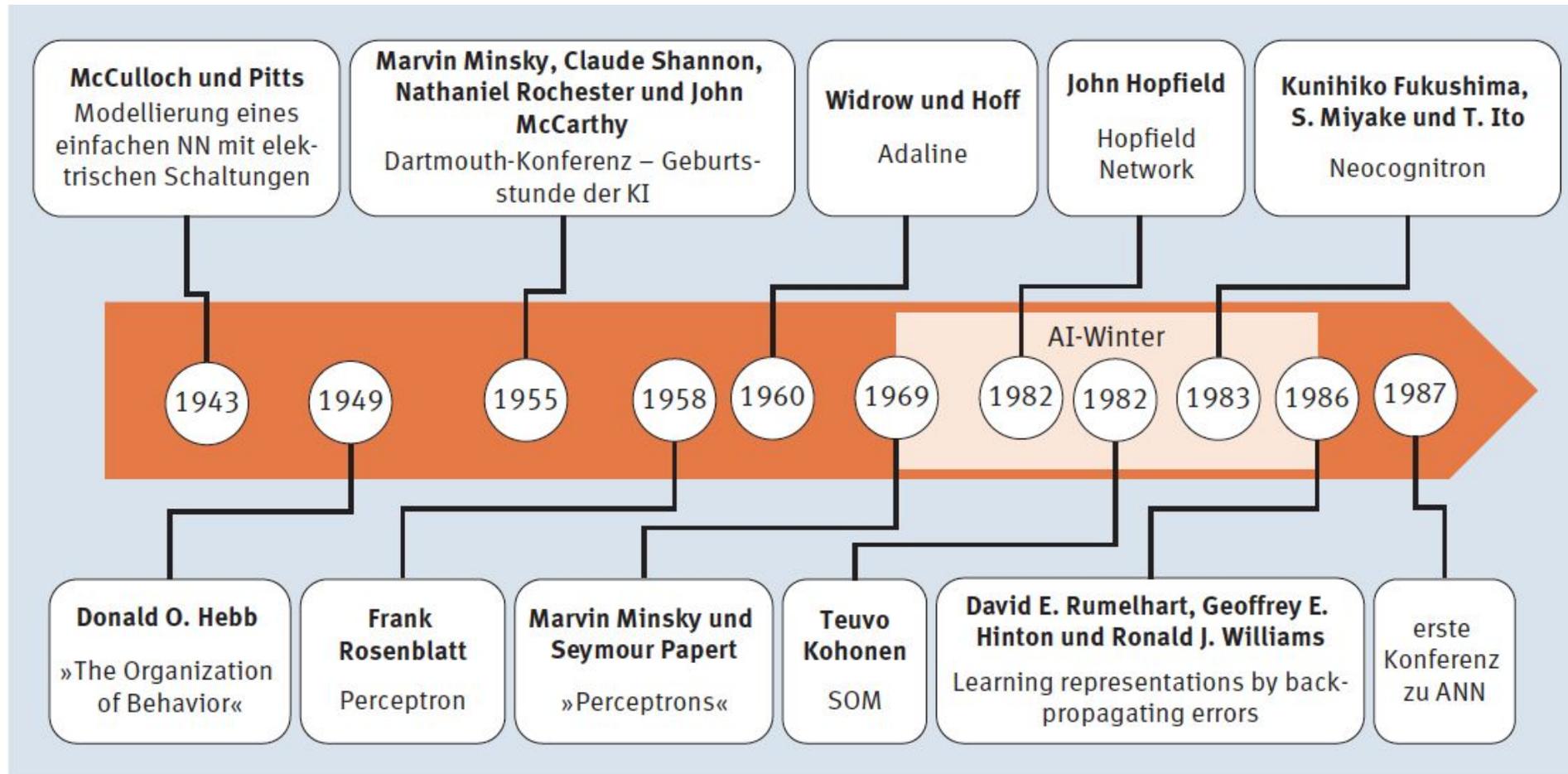


Abbildung 10.1 KNN-Geschichte, Teil 1



10.1.1 1943: McCulloch-Pitts Neurons



Abbildung 10.2 Grafische Darstellung (Ausschnitt) von McCulloch-Pitts-Netzen aus deren Originalpaper von 1943: »A logical calculus of the ideas immanent in nervous activity«

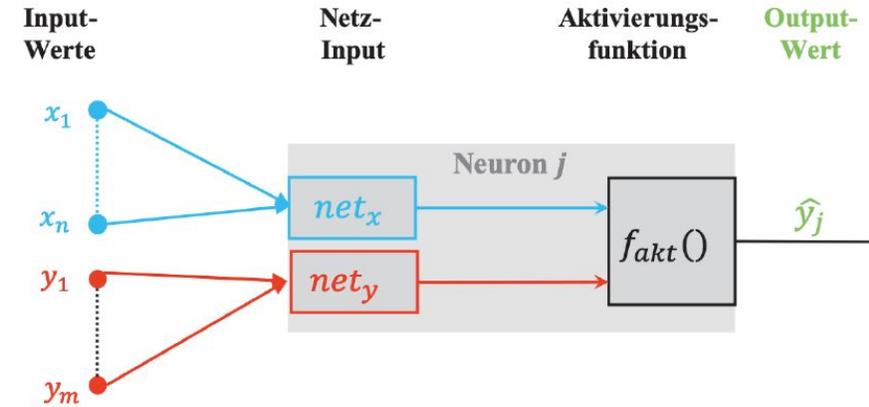


Abbildung 10.3 Darstellung eines McCulloch-Pitts-Neurons

Wenn man nun eher der formale Typ ist, dann wird man sich eine kompakte Darstellung wünschen, die wir, farblich angereichert, gerne liefern:

$$net_x = \sum_{i=1}^n x_i$$

$$net_y = \sum_{i=1}^m y_i$$

$$f_{\text{akt}}(net_x, net_y, \theta) = \hat{y}_j = \begin{cases} 1, & \text{falls } net_x \geq \theta \text{ und } net_y = 0 \\ 0, & \text{sonst} \end{cases}$$



10.2.4 1959: Bernard Widrow und Marcian Hoff – Adaline und Madaline

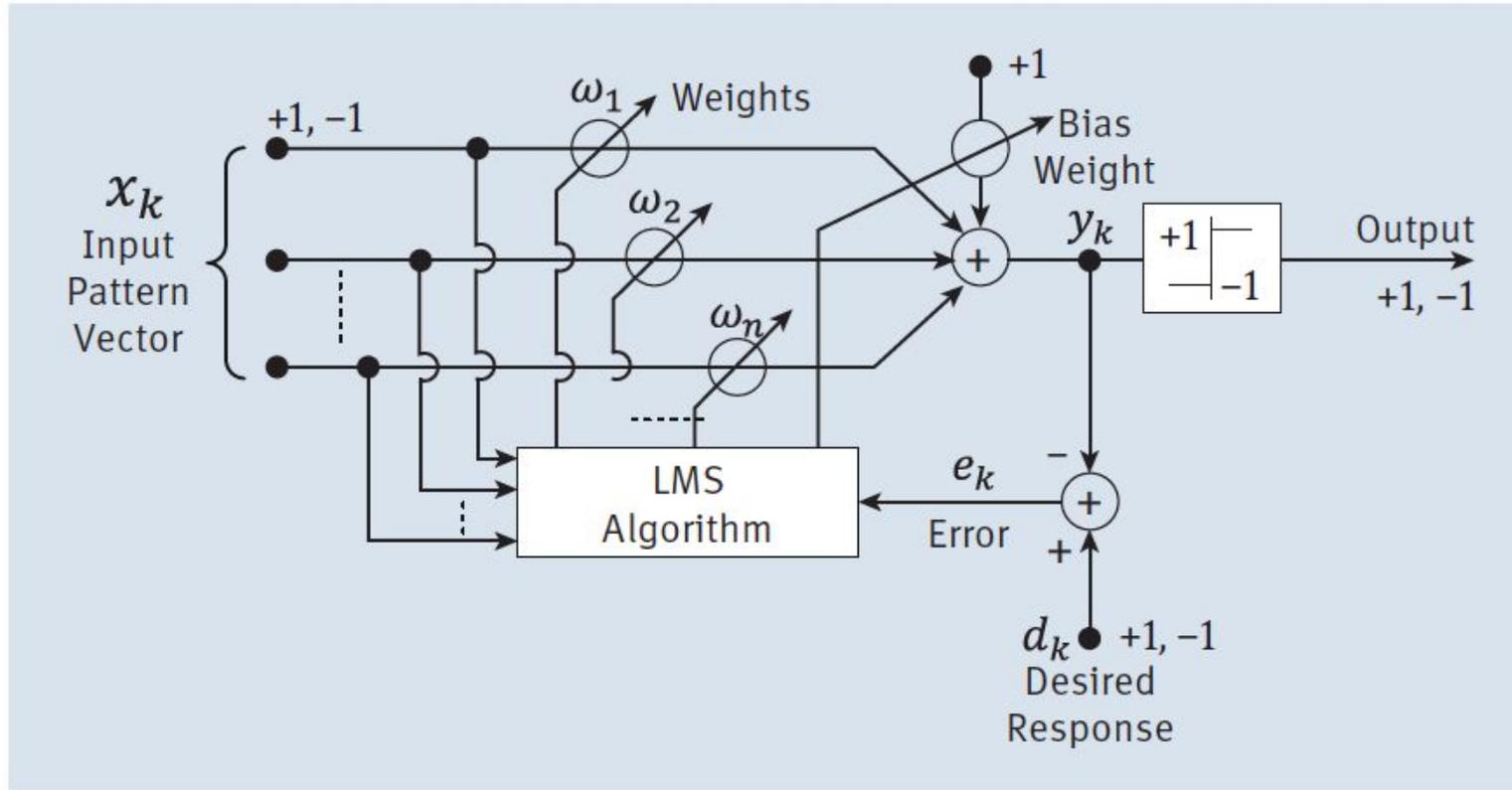


Abbildung 10.4 Adaline-Schaltdiagramm von Widrow und Hoff¹



10.5.1 1980: Fukushimas Neocognitron

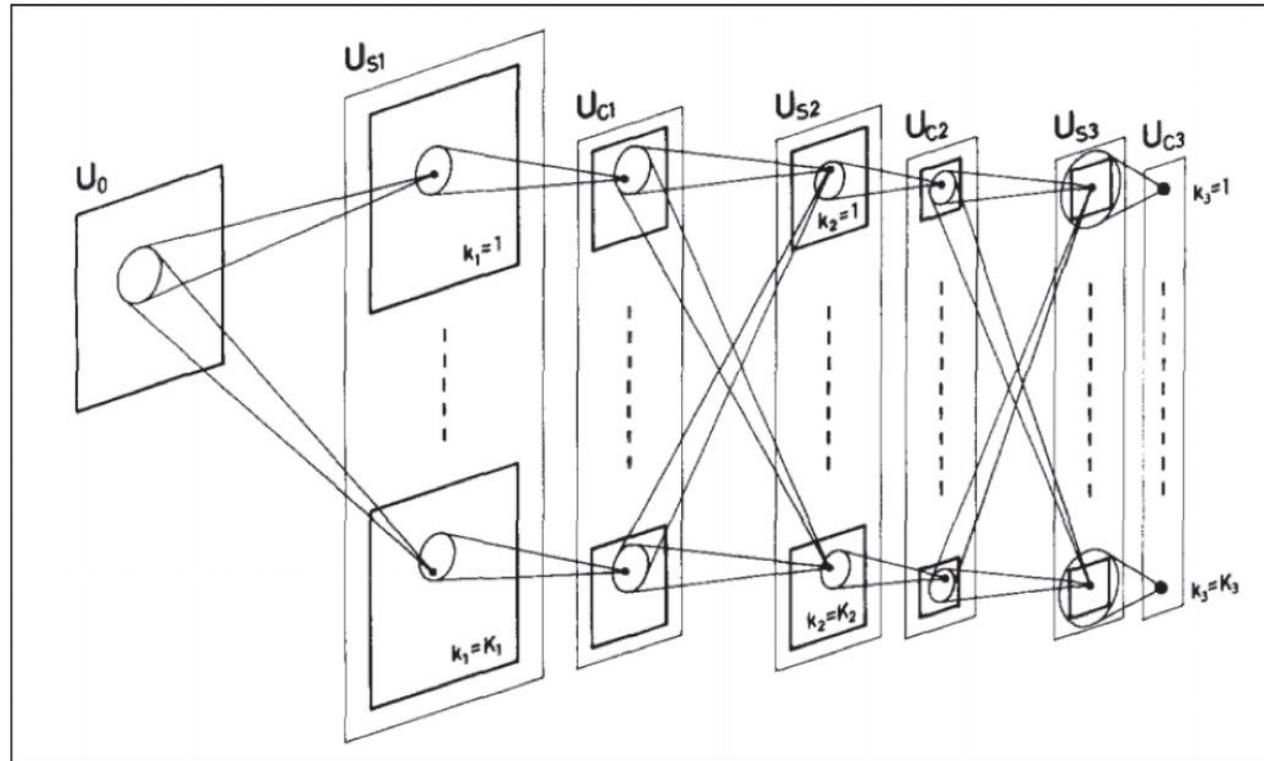


Abbildung 10.5 Schematisches Bild der Verschaltungen im Neocognitron von Fukushima
(Quelle: <https://www.rctn.org/bruno/public/papers/Fukushima1980.pdf>)



10.5.1 1980: Fukushimas I

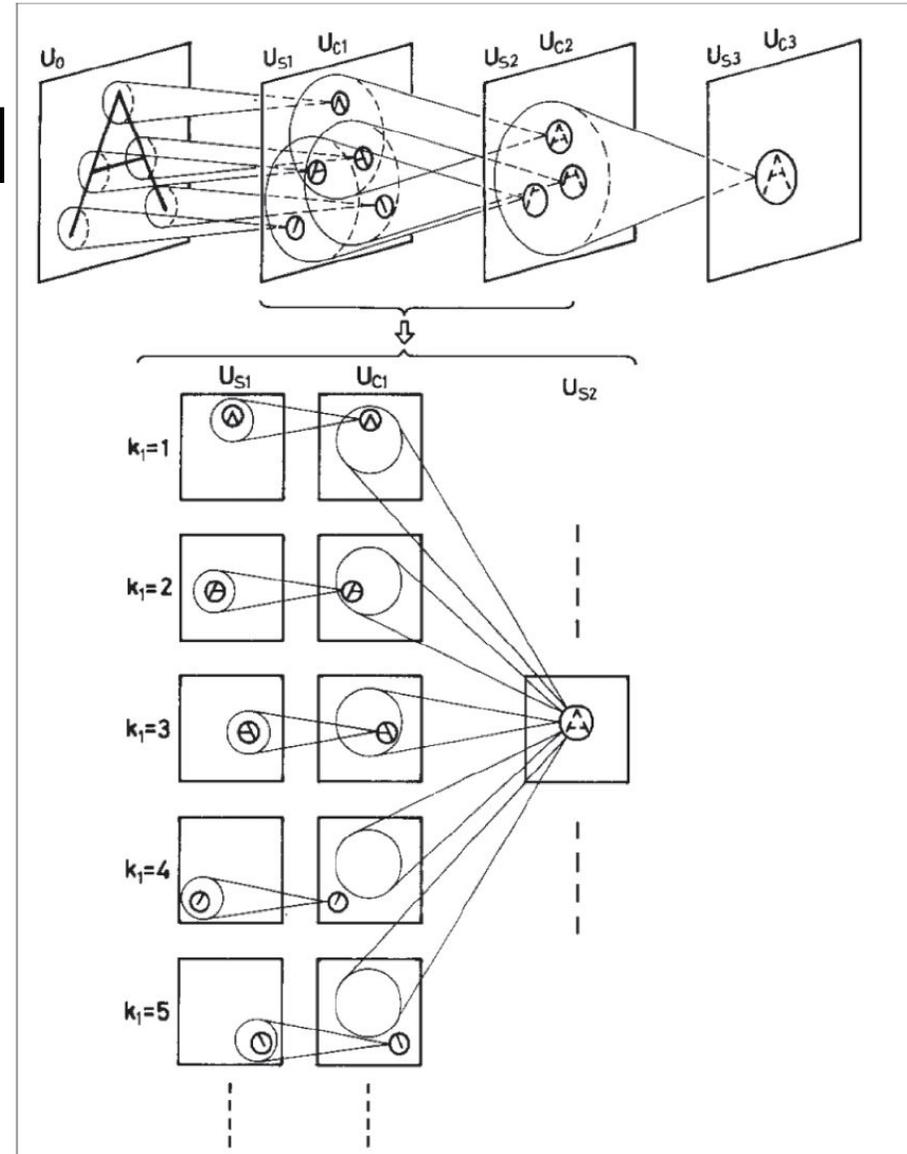


Abbildung 10.6 Neocognitron-Beispiel für die Extraktion von Features²



10.5.1 1980: Fukushimas Neocognitron

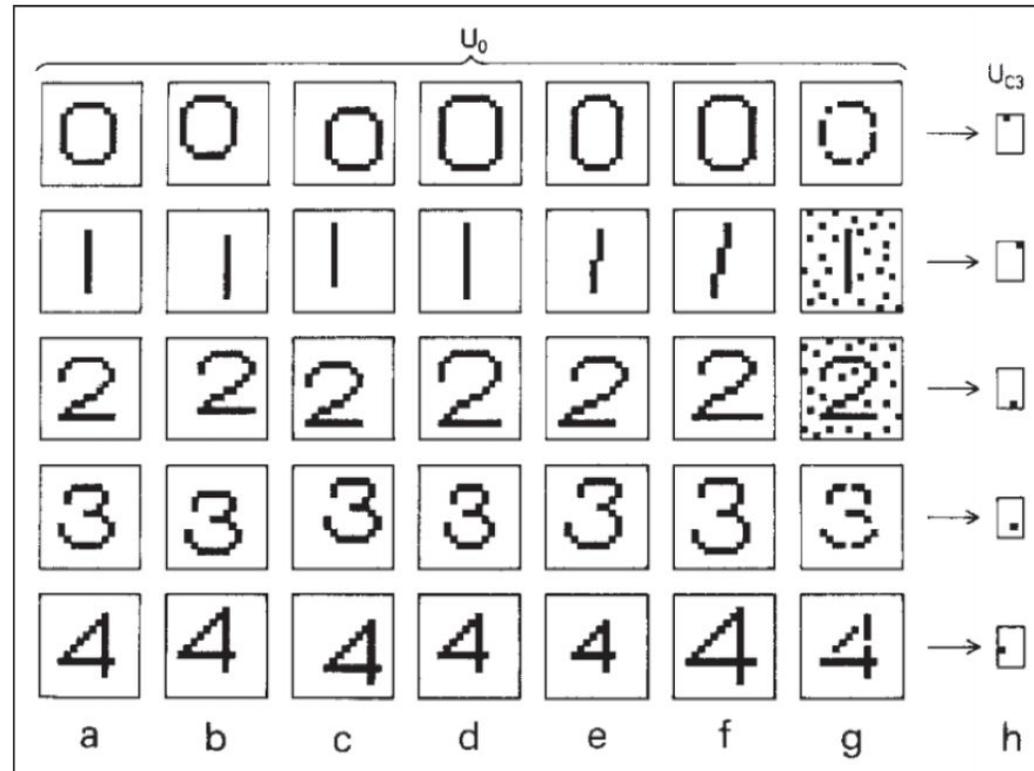


Abbildung 10.7 Neocognitron-Beispiel von unterschiedlichen Variationen und die Antwort im höchsten Layer³



10.5.2 1982: John Hopfield

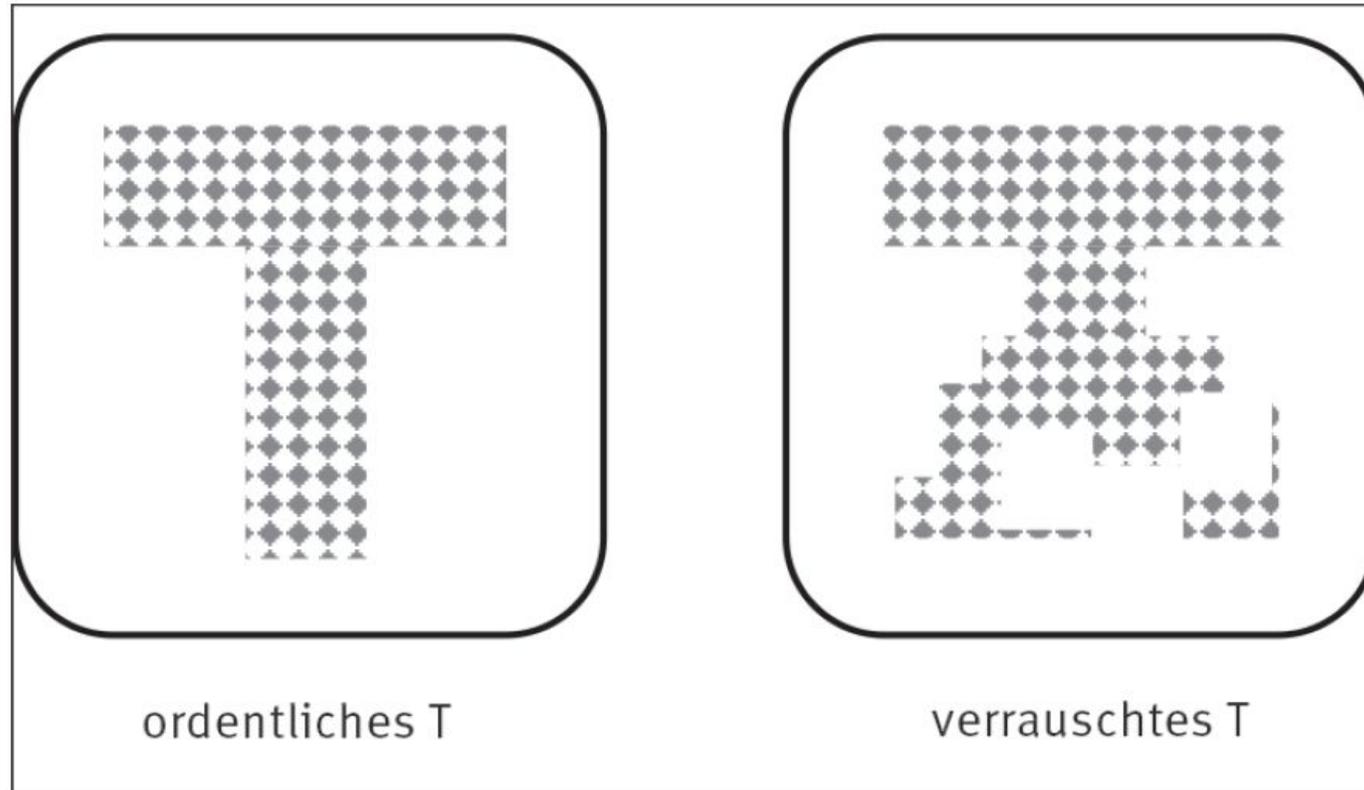
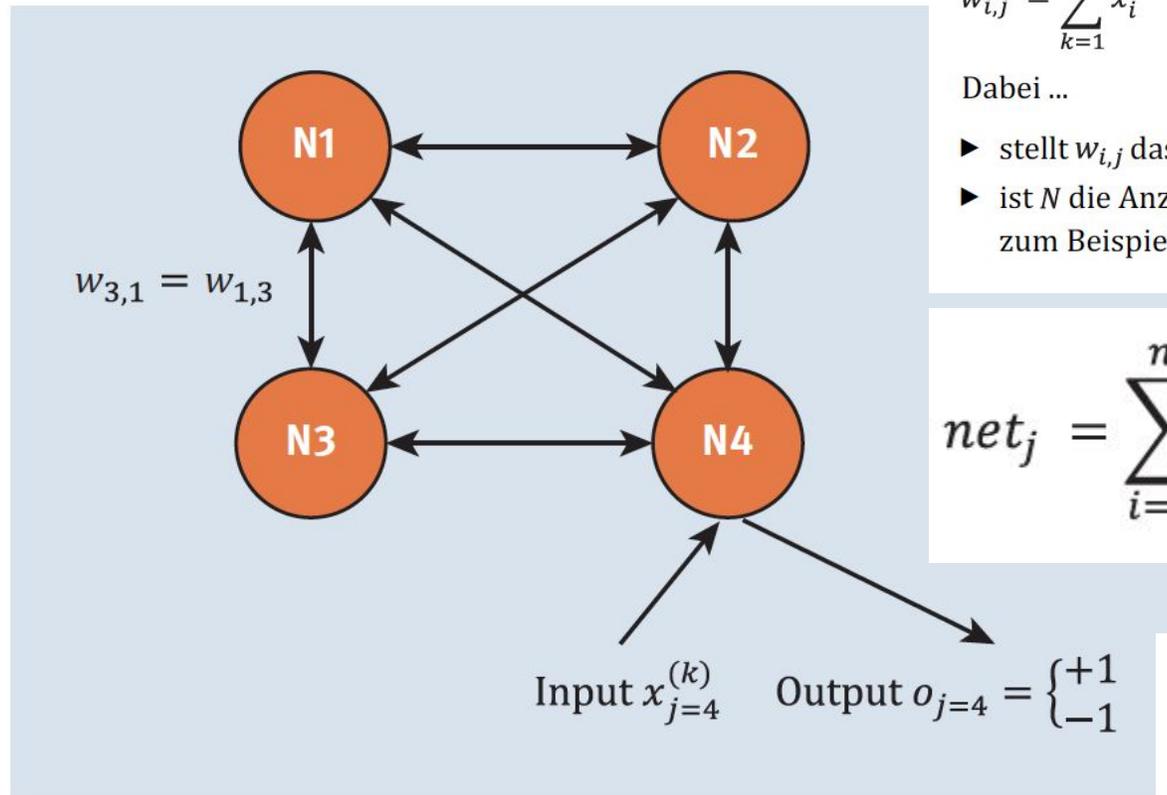


Abbildung 10.8 Assoziativer Speicher



10.5.2 1982: John Hopfield



$$w_{i,j} = \sum_{k=1}^N x_i^{(k)} \cdot x_j^{(k)}$$

Dabei ...

- ▶ stellt $w_{i,j}$ das Gewicht der Verbindung von Neuron i nach Neuron j dar,
- ▶ ist N die Anzahl der Inputs, die wie bereits gewohnt als Vektor repräsentiert werden, zum Beispiel der k -te Input $\vec{x}^{(k)} = (x_1^{(k)}, x_2^{(k)}, \dots)$.

$$net_j = \sum_{i=1}^n w_{i,j} \cdot o_i$$

$$f_{\text{step}}(net_j) = o_j = \begin{cases} +1, & \text{falls } net_j \geq \theta_j \\ -1, & \text{falls } net_j < \theta_j \end{cases}$$

Abbildung 10.9 Beispiel eines binären Hopfield-Netzwerks



10.5.2 1982: John Hopfield Beispiel

$$\vec{x}^{(T)} = (+1, +1, +1, -1, +1, -1, -1, +1, -1)$$

und das U durch den Vektor

$$\vec{x}^{(U)} = (+1, -1, +1, +1, -1, +1, +1, +1, +1)$$

repräsentiert.

0	1	2
3	4	5
6	7	8

0	1	2
3	4	5
6	7	8

Abbildung 10.10 Links ein T und rechts ein U mit den Indizes für den Vektor



10.5.2 1982: John Hopfield Beispiel

$$W_T = \begin{pmatrix} 0 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 \\ 1 & 0 & 1 & -1 & 1 & -1 & -1 & 1 & -1 \\ 1 & 1 & 0 & -1 & 1 & -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 0 & -1 & 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 & 0 & -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 0 & 1 & -1 & 1 \\ -1 & -1 & -1 & 1 & -1 & 1 & 0 & -1 & 1 \\ 1 & 1 & 1 & -1 & 1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 0 \end{pmatrix}$$



Listing 10.1/2/3 Die Multiplikation des transponierten T/U-Vektors mit sich selbst ergibt eine Matrix und $W = WT + WU$





Listing 10.4 Lernen und Auswertung mit Hopfield-Netzen

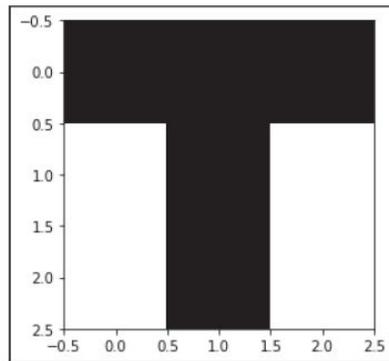




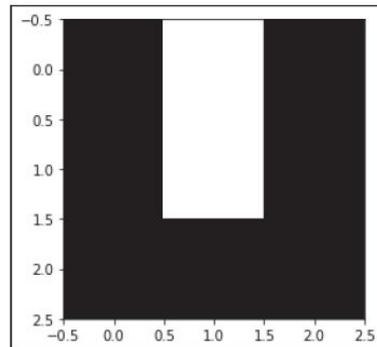
10.5.2 1982: John Hop

Die Ausgabe sieht so aus:

T - Muster



U - Muster



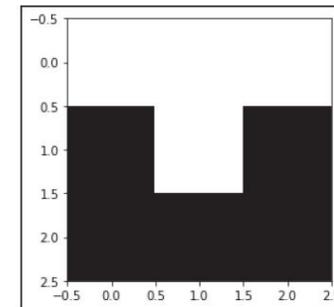
T + U - Matrix

```

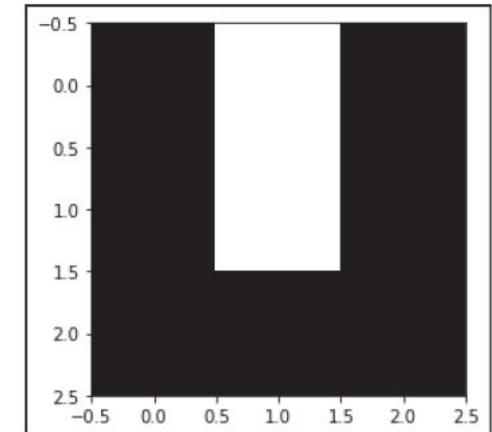
[[ 0.  0.  2.  0.  0.  0.  0.  2.  0.]
 [ 0.  0.  0. -2.  2. -2. -2.  0. -2.]
 [ 2.  0.  0.  0.  0.  0.  0.  2.  0.]
 [ 0. -2.  0.  0. -2.  2.  2.  0.  2.]
 [ 0.  2.  0. -2.  0. -2. -2.  0. -2.]
 [ 0. -2.  0.  2. -2.  0.  2.  0.  2.]
 [ 0. -2.  0.  2. -2.  2.  0.  0.  2.]
 [ 2.  0.  2.  0.  0.  0.  0.  0.  0.]
 [ 0. -2.  0.  2. -2.  2.  2.  0.  0.]]

```

Inputvektor



Schritt 1



T - Matrix minus Einheitsmatrix

```

[[ 0.  1.  1. -1.  1. -1. -1.  1. -1.]
 [ 1.  0.  1. -1.  1. -1. -1.  1. -1.]
 [ 1.  1.  0. -1.  1. -1. -1.  1. -1.]
 [-1. -1. -1.  0. -1.  1.  1. -1.  1.]
 [ 1.  1.  1. -1.  0. -1. -1.  1. -1.]
 [-1. -1. -1.  1. -1.  0.  1. -1.  1.]
 [-1. -1. -1.  1. -1.  1.  0. -1.  1.]
 [ 1.  1.  1. -1.  1. -1. -1.  0. -1.]
 [-1. -1. -1.  1. -1.  1.  1. -1.  0.]]

```

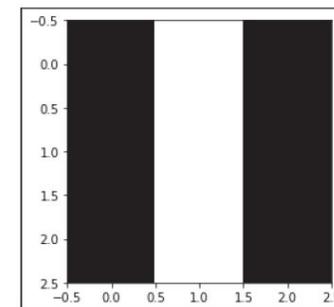
U - Matrix minus Einheitsmatrix

```

[[ 0. -1.  1.  1. -1.  1.  1.  1.  1.]
 [-1.  0. -1. -1.  1. -1. -1. -1. -1.]
 [ 1. -1.  0.  1. -1.  1.  1.  1.  1.]
 [ 1. -1.  1.  0. -1.  1.  1.  1.  1.]
 [-1.  1. -1. -1.  0. -1. -1. -1. -1.]
 [ 1. -1.  1.  1. -1.  0.  1.  1.  1.]
 [ 1. -1.  1.  1. -1.  1.  0.  1.  1.]
 [ 1. -1.  1.  1. -1.  1.  1.  0.  1.]
 [ 1. -1.  1.  1. -1.  1.  1.  1.  0.]]

```

Schritt 0



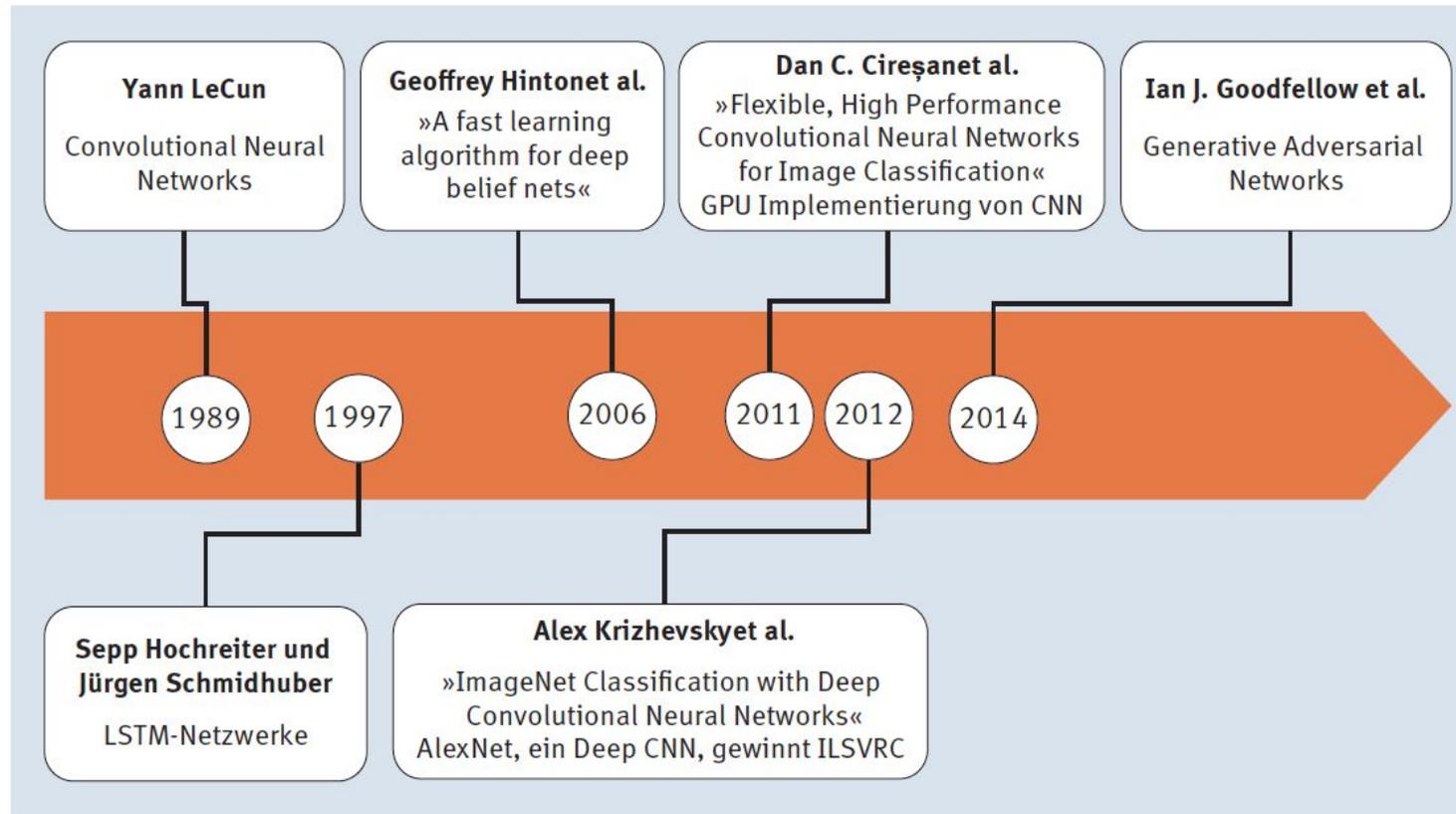


Abbildung 10.11 KNN-Geschichte, Teil 2



10.8.1 2014: Ian J. Goodfellow et al. – Generative Adversarial Networks (GAN)

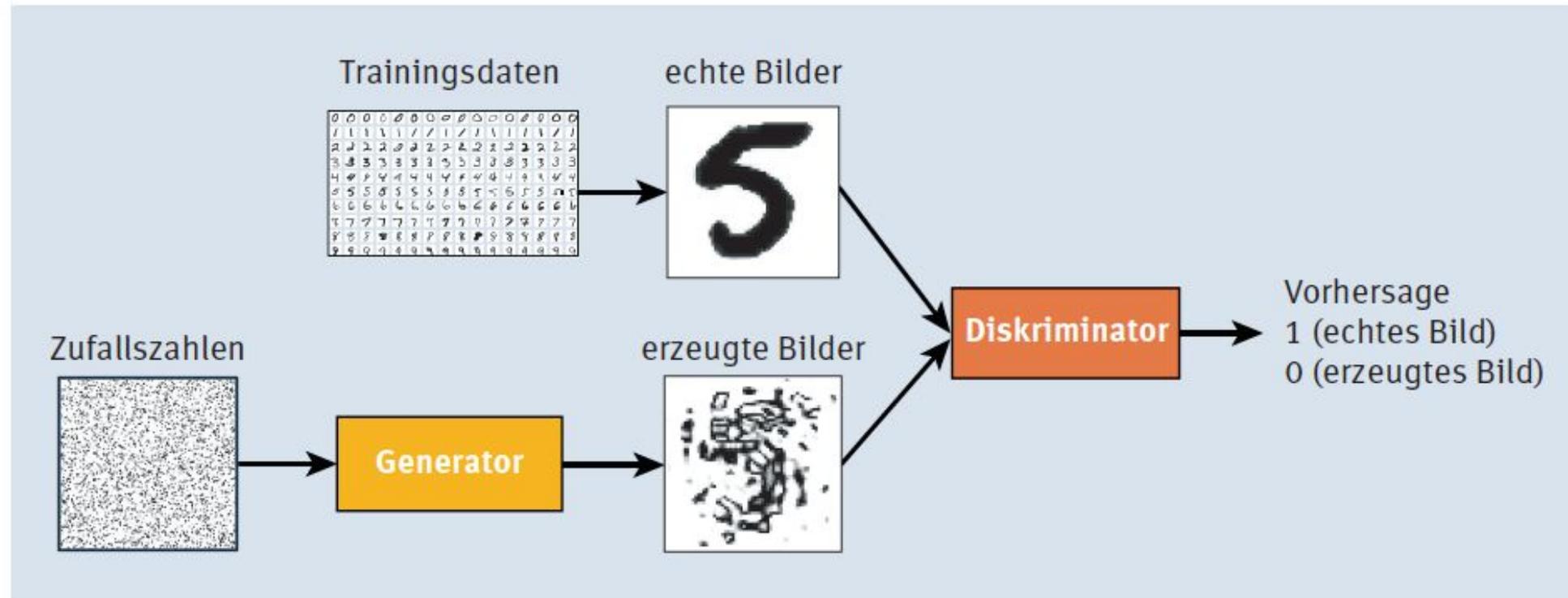


Abbildung 10.12 GAN – prinzipieller Aufbau



10.8.1 2014: Ian J. Goodfellow et al. – Generative Adversarial Networks (GAN)

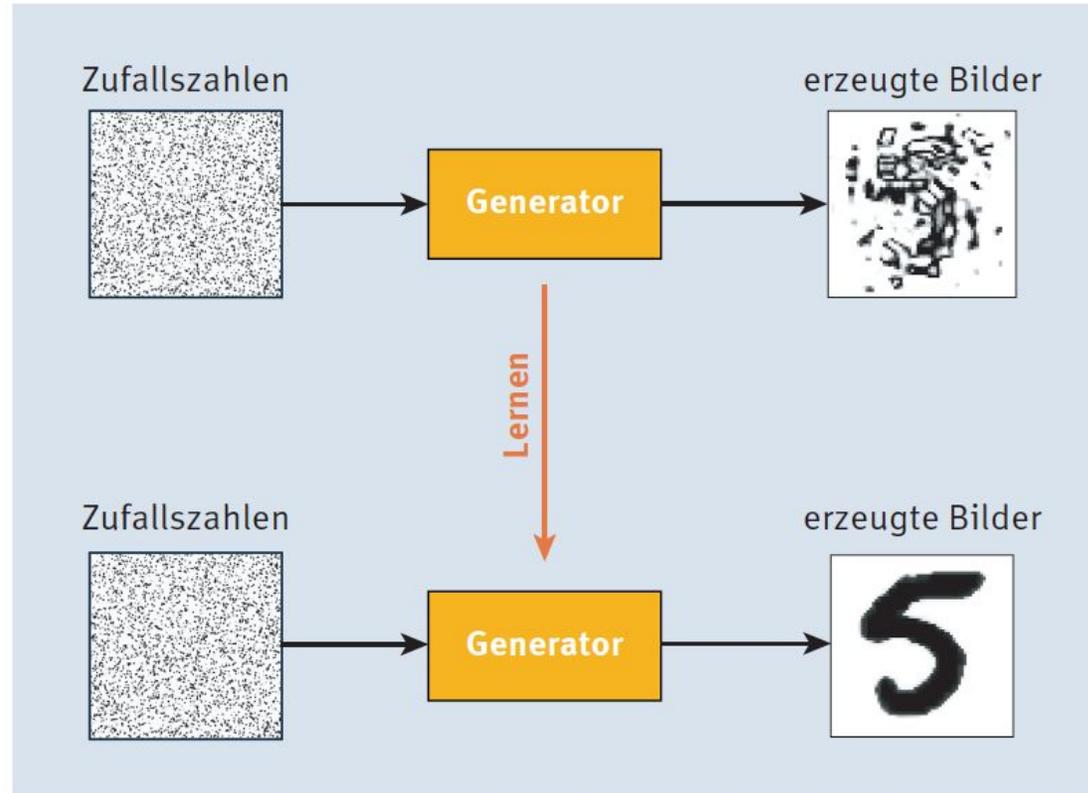


Abbildung 10.13 Der Generator lernt, Bilder zu erzeugen.



10.8.1 2014: Ian J. Goodfellow et al. – Generative Adversarial Networks (GAN)

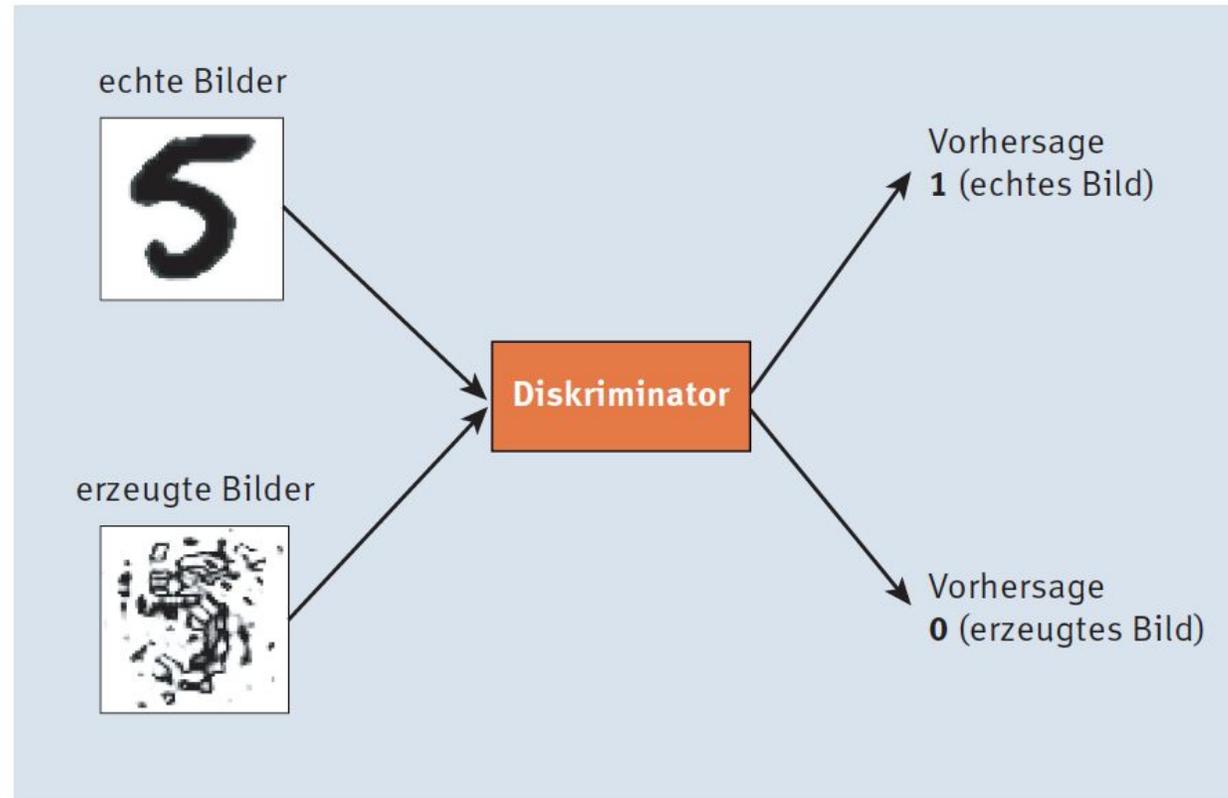


Abbildung 10.14 Der Diskriminator lernt, Bilder zu unterscheiden.



Kapitel 11

Der Machine-Learning-Prozess



11.1 Das CRISP-DM-Modell

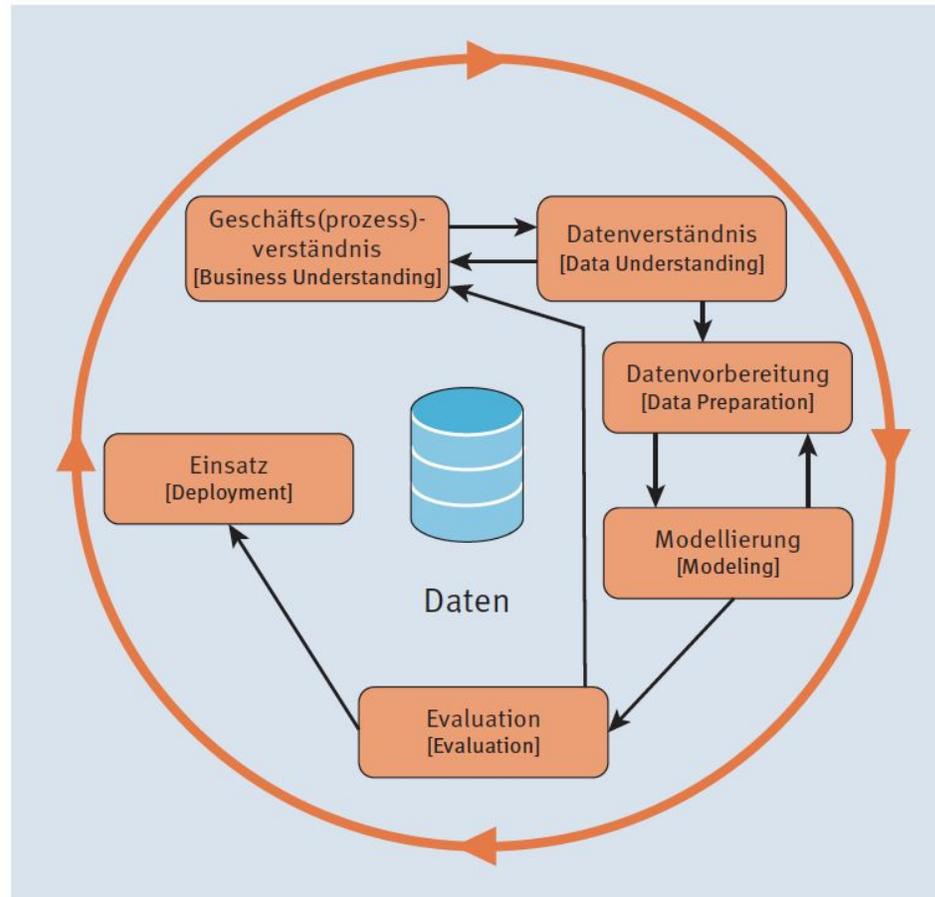


Abbildung 11.1 Das CRISP-DM-Modell (Quelle: Wikimedia Commons)



11.2 Feature Engineering

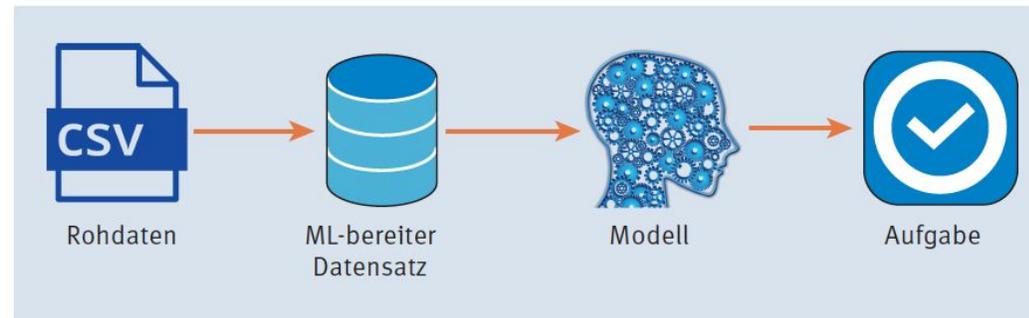


Abbildung 11.2 Gewünschter Ablauf eines Machine-Learning-Projekts

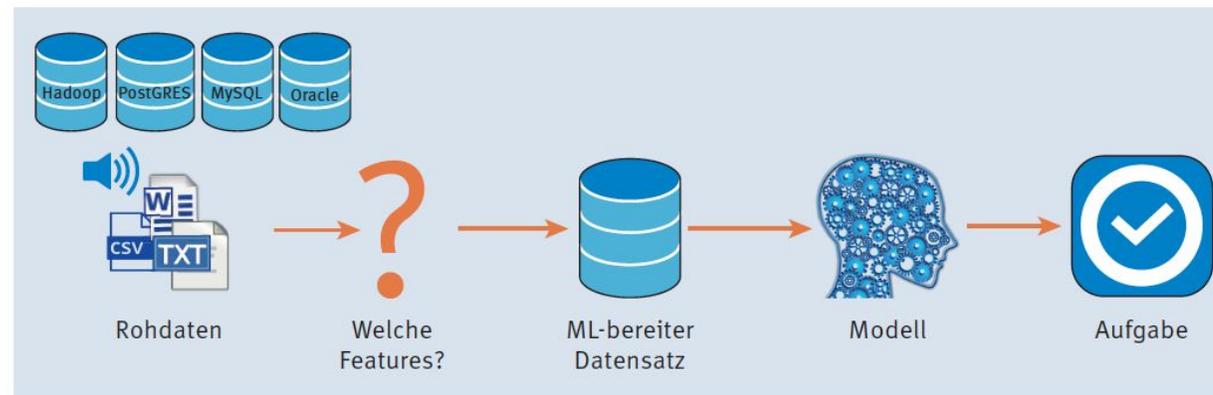


Abbildung 11.3 Realität eines Machine-Learning-Projekts

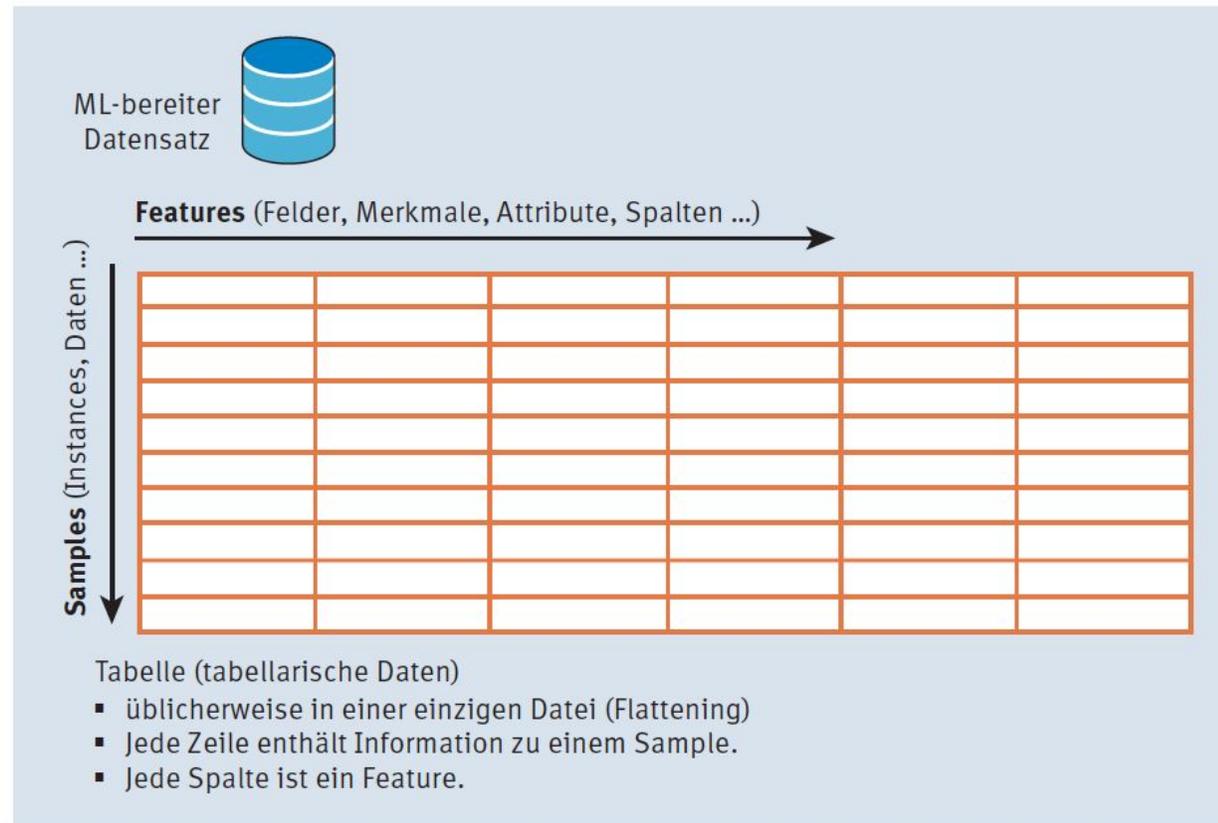


Abbildung 11.4 Für ML (Machine Learning) bereiter Datensatz



11.2.1 Feature-Kodierung

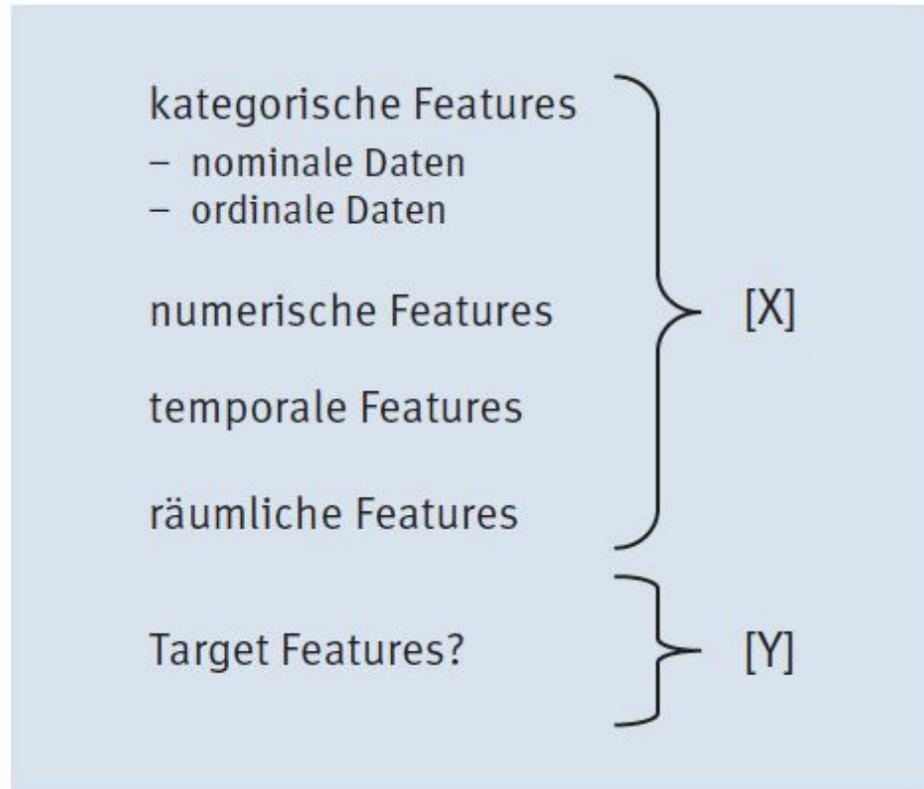


Abbildung 11.5 Einteilung des Formats der Rohdaten

Kategorisches Feature

- Label Encoding

sehr gut	1
gut	2
befriedigend	3
genügend	4
nicht genügend	5

Tabelle 11.1 Beispiel für Label Encoding (Schulnoten in Österreich)

Kategorisches Feature

- One-Hot-Encoding

Numerische Feature

- Binning (in Kategorien)
 - Quantil-Binning
 - Logarithmic-Binning
- Feature-Skalierung (-Normalisierung)

Temporales Feature

- Time Binning

Räumliches Feature

- geopy



Listing 11.1 Beispiel für Label Encoding der Haarfarbe der Schüler durch »sklearn «





Listing 11.2 One-Hot-Encoding mit der scikit-learn-Bibliothek





Listing 11.3 Vorbereitung der Daten für » category_encoders«



	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03

Abbildung 11.6 Ausgabe zu [Listing 11.3](#)



Listing 11.4 Verwendung weiterer Encoder



	CRIM	ZN	INDUS	CHAS_0	CHAS_1	NOX	RM	AGE	DIS	RAD_0	RAD_1	RAD_2	RAD_3	RAD_4	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0	1	0.538	6.575	65.2	4.0900	0	0	0	0	1	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0	1	0.469	6.421	78.9	4.9671	0	0	0	1	0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0	1	0.469	7.185	61.1	4.9671	0	0	0	1	0	242.0	17.8	392.83	4.03

Abbildung 11.7 Ausgabe zu Listing 11.4



Listing 11.5 Binning von Altersangaben mit fixer Bereichslänge

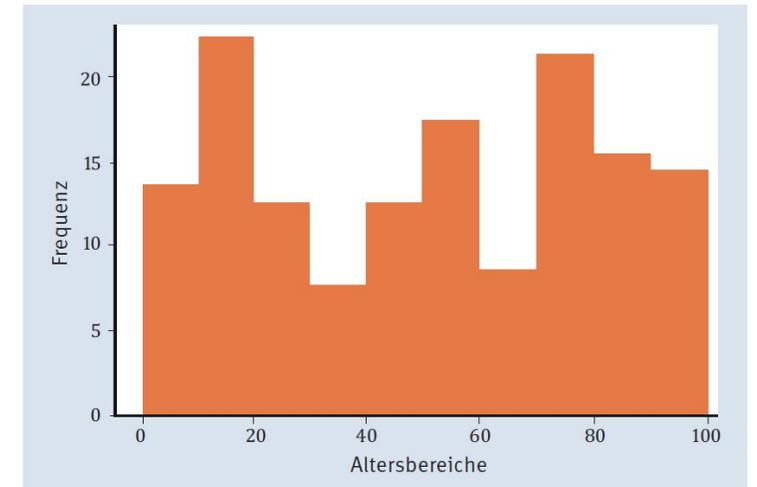


Abbildung 11.8 Ausgabe zu [Listing 11.5](#)



Listing 11.6 Quantiles Binning





Listing 11.7 Logarithmisches Binning

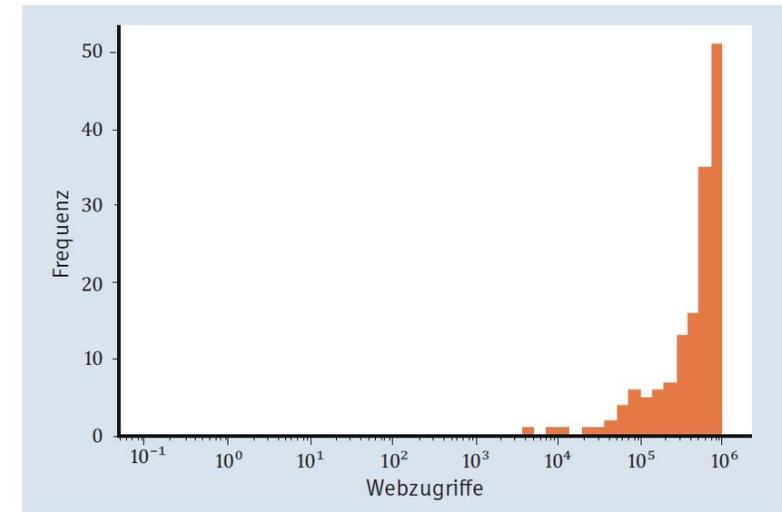


Abbildung 11.9 Bildausgabe zu [Listing 11.7](#)



Feature Normalisierung

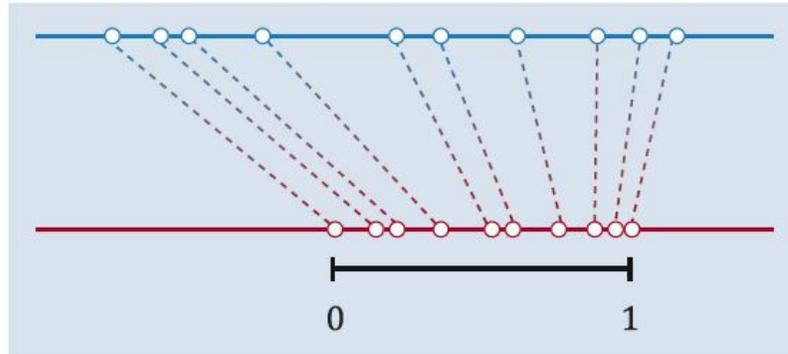


Abbildung 11.10 Skizze einer Skalierung in den Wertebereich [0,1]

Die einfachste Möglichkeit der Skalierung eines Wertes x_i ist die *Min-Max-Skalierung*, wobei der minimale Wert aller Daten auf 0 abgebildet wird und der maximale auf 1 mit folgender Formel:

$$x_{\text{minmax_sc}_i} = \frac{x_i - \min(\vec{x})}{\max(\vec{x}) - \min(\vec{x})}$$

Diese Formel wird für alle Elemente x_i angewandt, wobei $\vec{x} = (x_1, x_2, \dots, x_n)$.



Feature Varianzskalierung

Eine weitere Möglichkeit der Skalierung ist die Varianzskalierung, die mithilfe folgender Formel durchgeführt wird:

$$x_{var_sc_i} = \frac{x_i - \text{mean}(\vec{x})}{\sqrt{\text{var}(\vec{x})}}$$

Ein Feature, das so transformiert wird, hat dann einen Mittelwert von 0 und eine Varianz von 1.



Listing 11.8 Min-Max- und Varianzskalierung mit »sklearn«





Temporales Feature

Tageszeit (Stunde)	Kategorie ID	Kategorie Beschreibung
[5, 8)	1	frühmorgens
[8, 11)	2	morgens
[11, 14)	3	mittags
[14, 19)	4	nachmittags
[19, 22)	5	abends
[22, 24) und [0, 5)	6	nachts

Tabelle 11.2 Time Binning der Tageszeit



Listing 11.9 Umwandlung von Adresse in Höhen- und Breitengrad





11.2.2 Feature-Extraktion

- ▶ Datenbanken
- ▶ Textdateien
- ▶ Bilddaten
- ▶ Audiodaten

Mit *Feature-Extraktion* bezeichnet man Methoden, die Rohdaten in einen Eingabevektor \vec{x} transformieren. Allein darüber könnte man wieder ein eigenes Buch schreiben, deshalb konzentrieren wir uns auf wesentliche Elemente.



11.2.2 Feature-Extraktion Datenbank

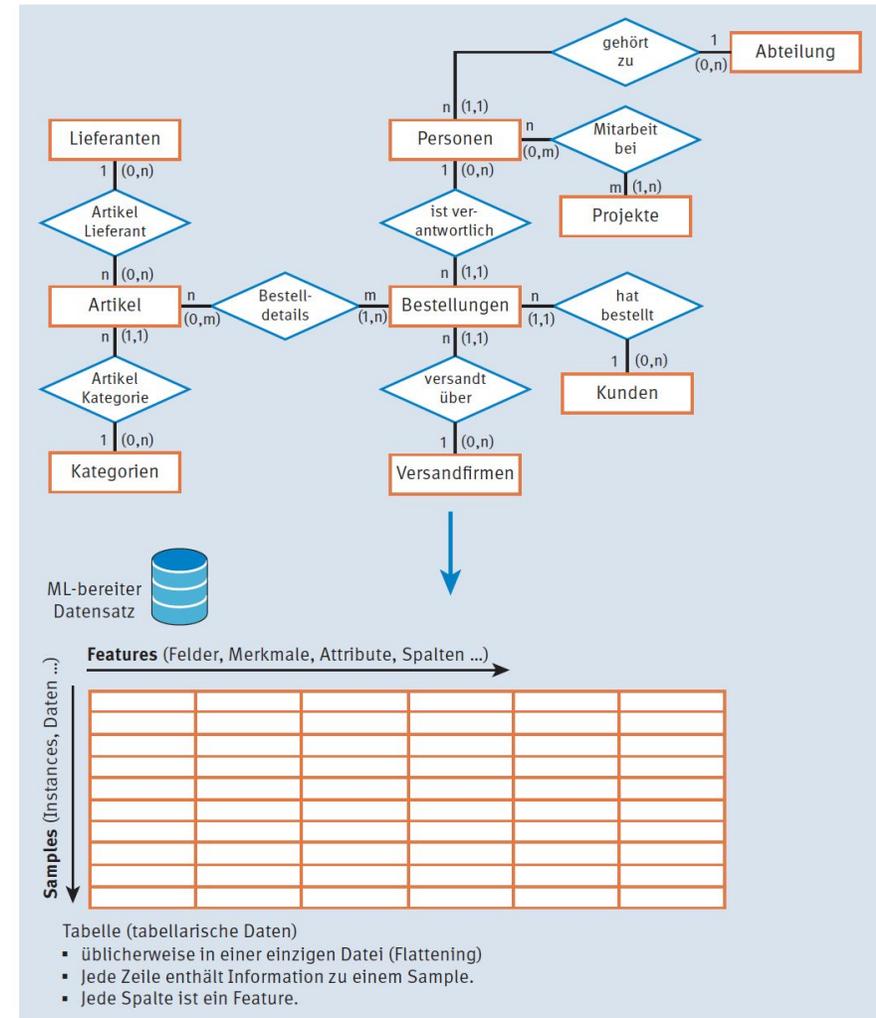


Abbildung 11.12 Datenverflachung einer Datenbank



11.2.2 Feature-Extraktion Textdaten

- ▶ Säubern (Cleaning)
- ▶ Entfernen (Removal)
- ▶ Wortwurzel (Stem, Chop)
- ▶ Tokenisierung (Tokenization)



Listing 11.10 Befehl zum Download der »nltk«-Daten

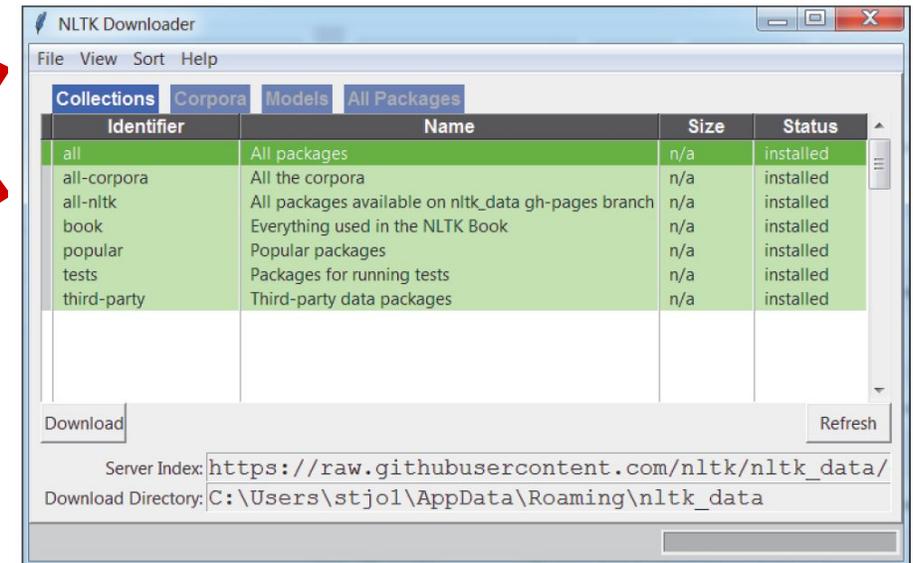


Abbildung 11.13 Das interaktive »nltk«-Download-Fenster



Listing 11.11/12/13/14/15 Tokenisierung/Klein und Interpunktion/Stopppwörter/Wortstamm/Bag-of-Words eines Textes





TF-IDF. TF steht für Term Frequency, IDF für Inverse Document Frequency, $\log(\text{Anzahl aller Dokumente} / \text{Anzahl aller Dokumente mit Wort } w)$.

Wort	TF		IDF	TF*IDF	
	Satz 1	Satz 2		Satz 1	Satz 2
Das	1/7	1/7	$\log(2/2)=0$	0	0
Auto	1/7	0	$\log(2/1)=0,3$	0,043	0
Fahrrad	0	1/7	$\log(2/1)=0,3$	0	0,043
fährt	1/7	1/7	$\log(2/2)=0$	0	0
man	1/7	1/7	$\log(2/2)=0$	0	0
auf	1/7	1/7	$\log(2/2)=0$	0	0
einer	1/7	1/7	$\log(2/2)=0$	0	0
Straße	0	1/7	$\log(2/1)=0,3$	0	0,043
Autobahn	1/7	0	$\log(2/1)=0,3$	0,043	0

Abbildung 11.14 TF-IDF-Beispiel



11.2.2 Feature-Extraktion Bilddaten

- ▶ Translation (Verschieben)
- ▶ Rotation (Drehen)
- ▶ Skalierung (Vergrößern, Verkleinern)
- ▶ Scheren (Skalieren, aber in x- und y-Richtung unterschiedlich)
- ▶ horizontales und/oder vertikales Spiegeln



Listing 11.16 Datenaugmentation für Bilddaten





11.2.2 Feature-Extraktion

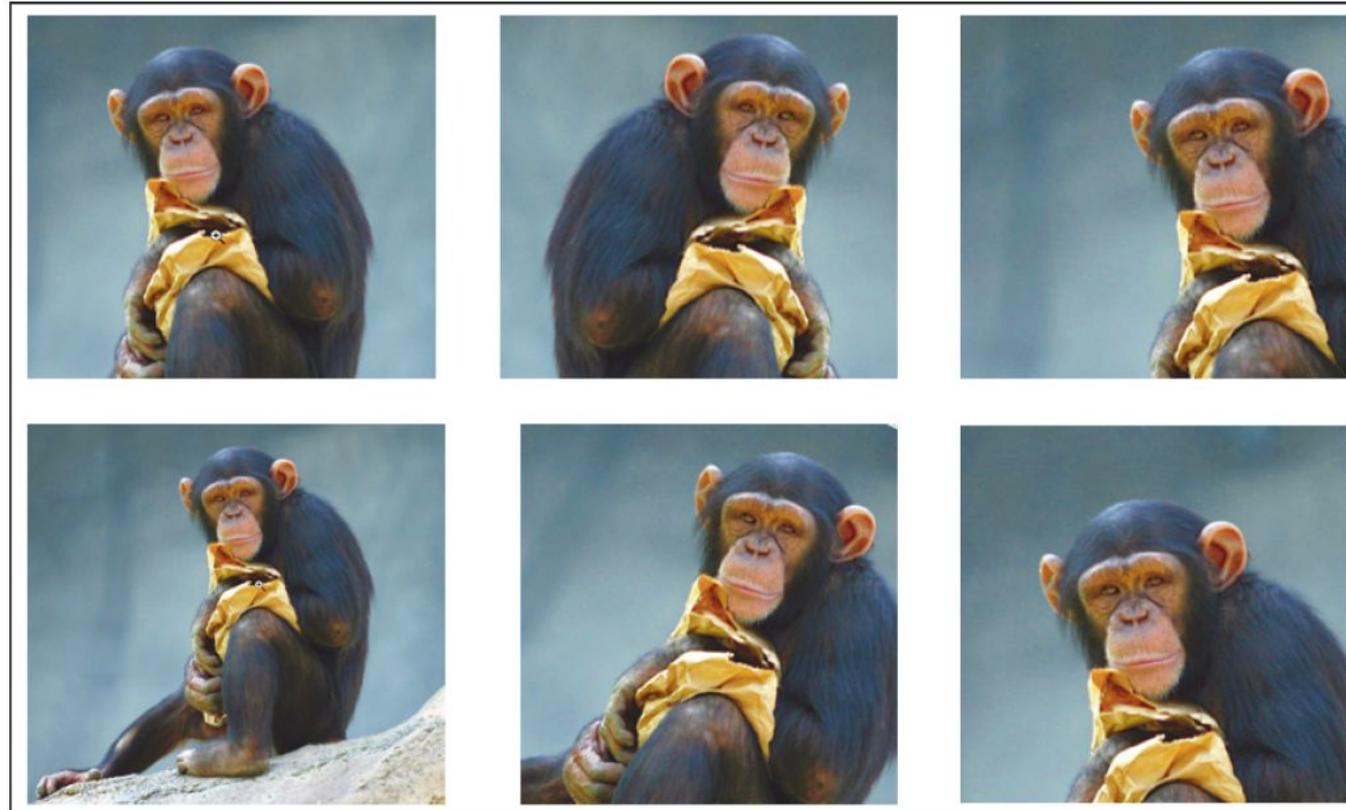


Abbildung 11.15 Beispiel für Data Augmentation von Bildern (Quelle: Wikimedia Commons)



11.2.3 Der Fluch der Dimensionalität

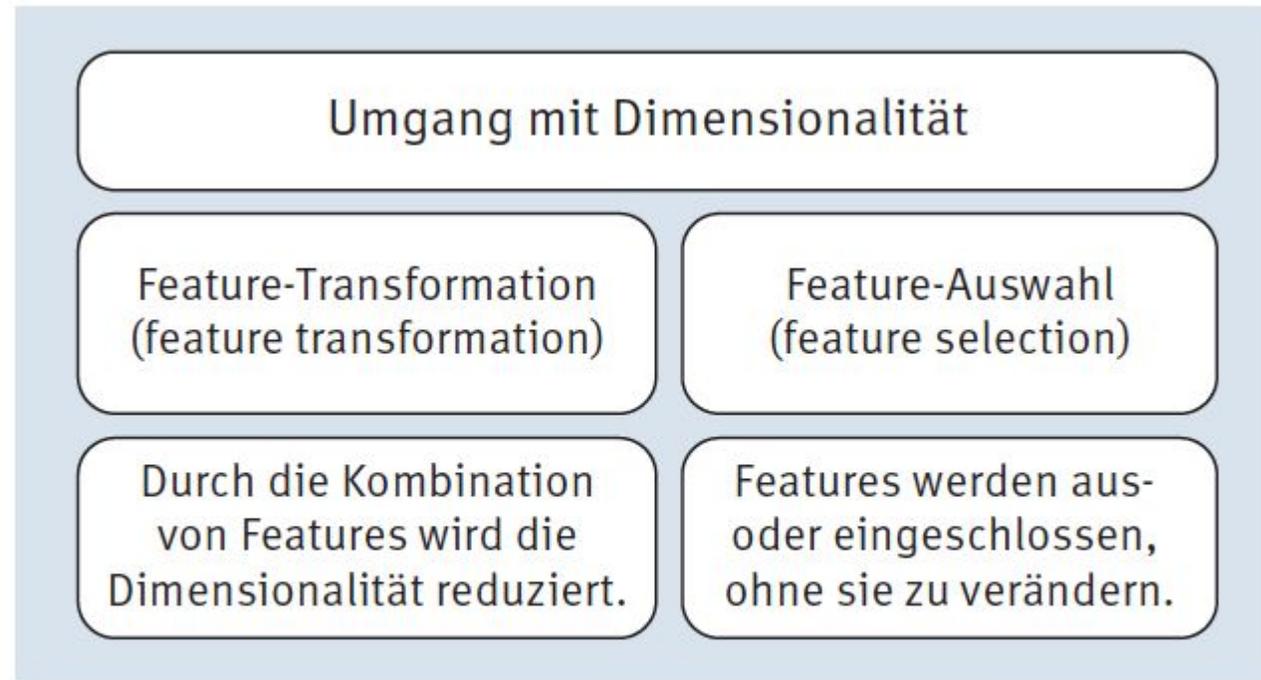


Abbildung 11.16 Umgang mit Dimensionalität



11.2.4 Feature-Transformation

Wissensbasiert

$$BMI = \frac{kg(\text{Körpergewicht})}{m(\text{Körpergröße})^2}$$

Hier würden die beiden Features Körpergewicht und Körpergröße durch ein neues kombiniertes Feature BMI ersetzt.



11.2.4 Feature-Transformation Automatisch

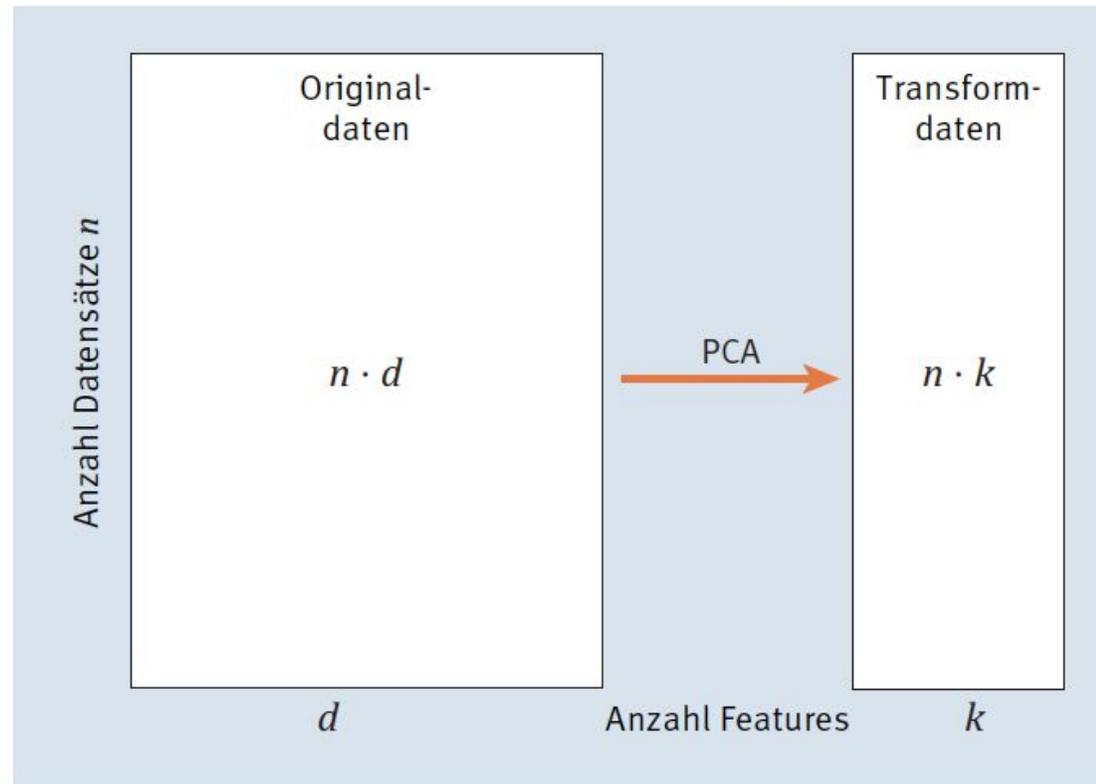


Abbildung 11.17 Funktionsweise der Principal-Components-Analyse



Listing 11.17/18/19/20 Laden des Iris-Datensatzes/PCA von scikit-learn/Scatter Plot





Abbildung 11.18 Scatter-Plot der Original-Iris-Daten

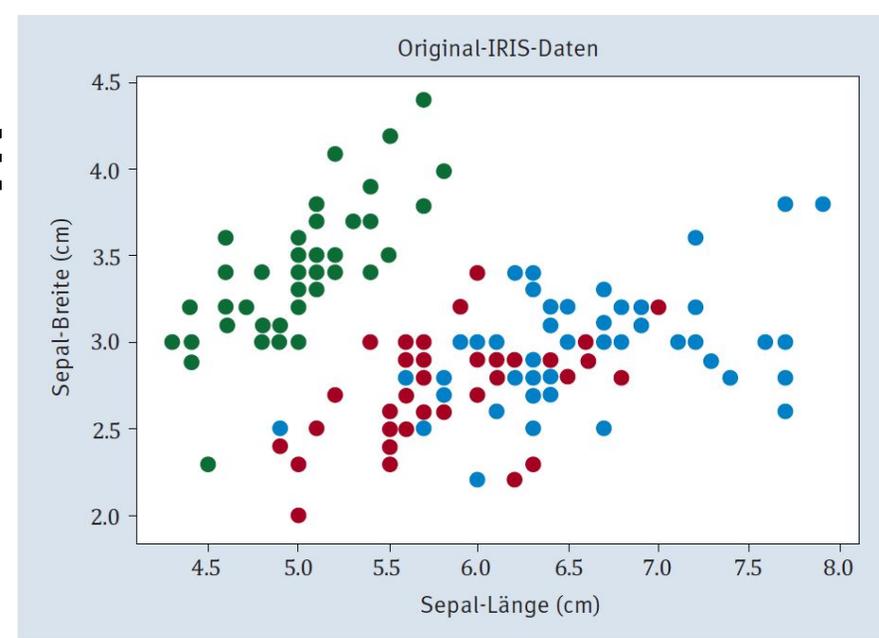


Abbildung 11.18 Scatter-Plot der Original-Iris-Daten

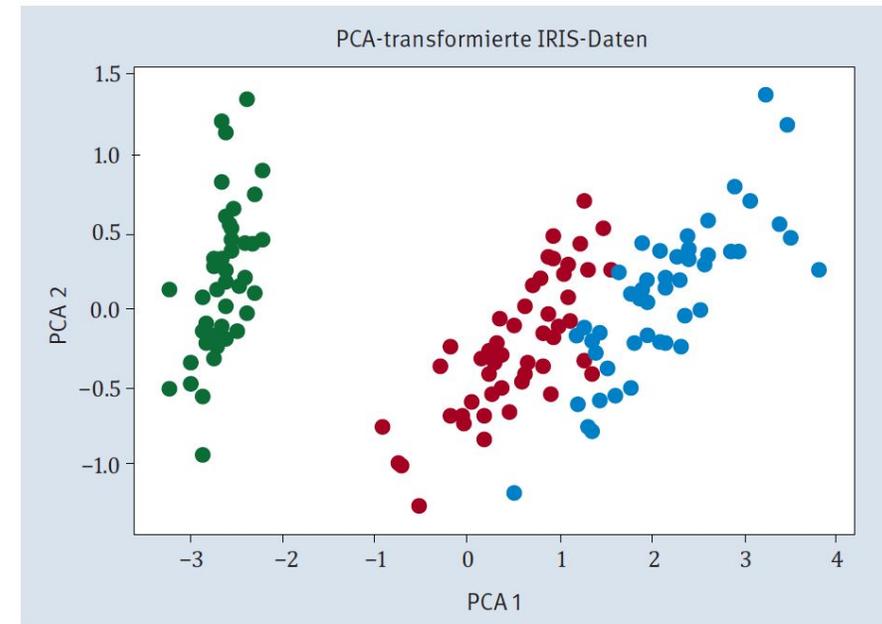


Abbildung 11.19 Streudiagramm der PCA-transformierten Iris-Daten



11.2.5 Feature-Auswahl Filtermethoden



Listing 11.21 Filtermethode zur Feature-Selektion für den Datensatz X, y



Chi-Quadrat-Test

Der Chi-Quadrat-Test ist ein statistisches Maß, das bestimmt, ob es zwischen zwei beliebigen (kategorischen) Features einen Zusammenhang gibt. Der Name kommt daher, dass eine quadrierte Differenz von erwarteten und beobachteten Häufigkeiten von Features zur Berechnung verwendet wird.



11.2.5 Feature-Auswahl Wrapper-Methoden



Listing 11.22 Wrapper-Methode zur Feature-Selektion für den Datensatz X,y





Kapitel 12

Lernverfahren



12.1 Lernstrategien

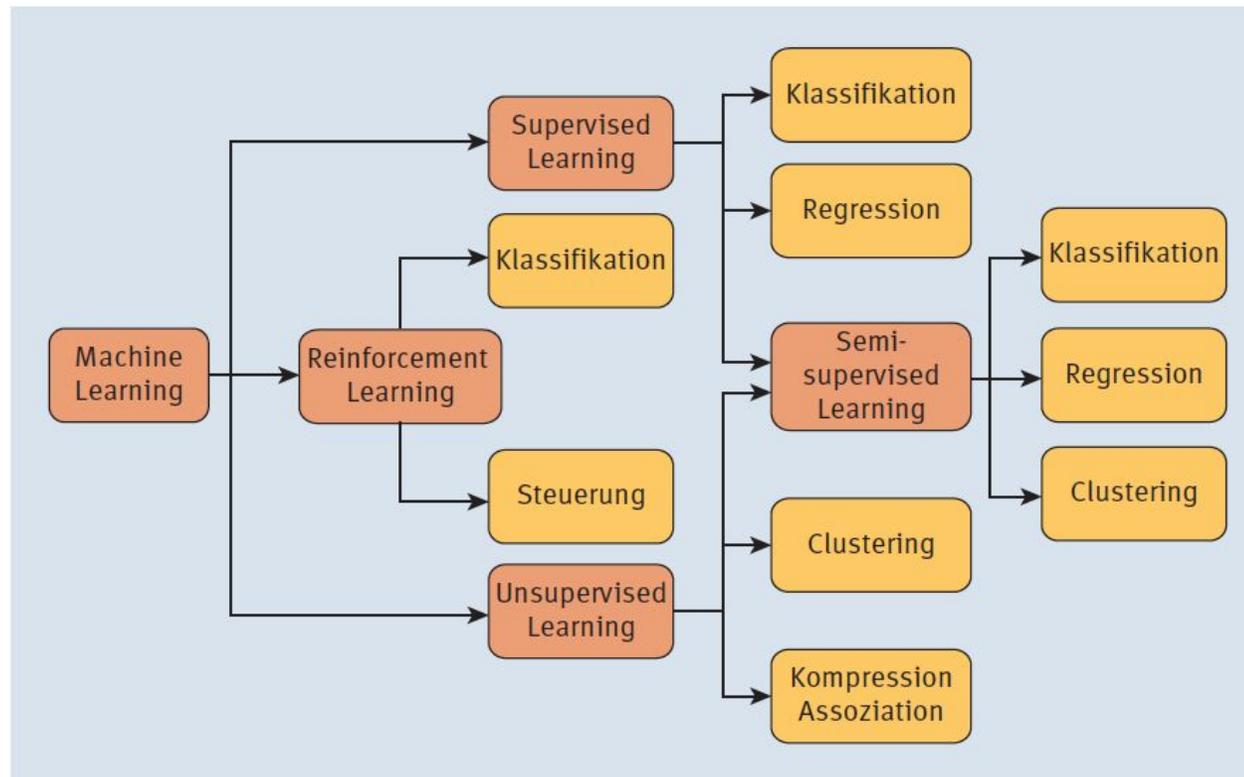


Abbildung 12.1 Lernstrategien und ihre möglichen Anwendungen



12.1.1 Überwachtes Lernen (Supervised Learning)

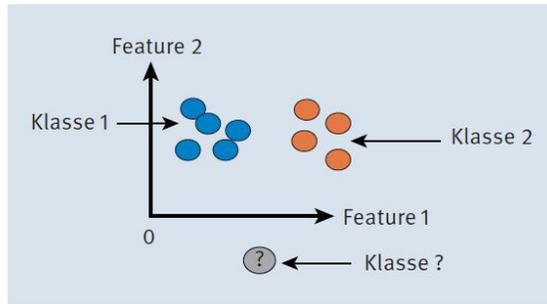


Abbildung 12.2 Klassifikation

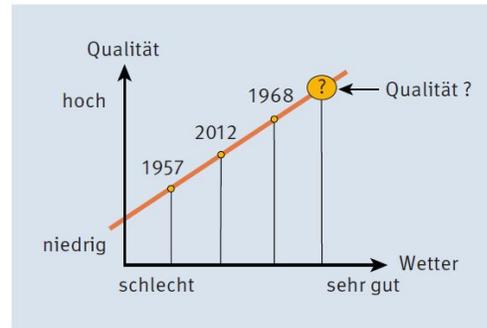


Abbildung 12.3 Wein-Qualitäts-Regression

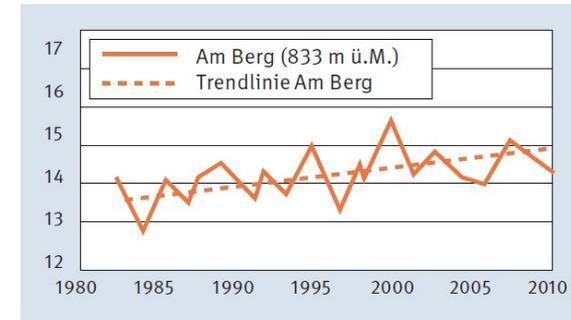


Abbildung 12.4 Schematische Darstellung der Trendanalyse des Mittelwertes der Temperatur für den Ort Am Berg

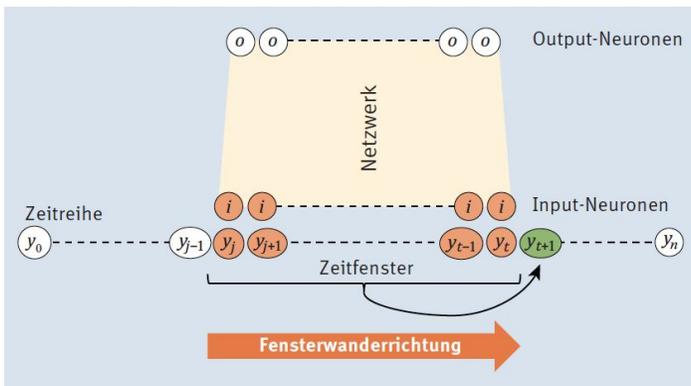


Abbildung 12.5 Zeitreihe mit Netzwerk

Lernalgorithmen

Bekannte Algorithmen für das Supervised Learning sind neben den neuronalen Netzen, die wir auch in die Liste eingefügt haben:

Algorithmus	Einsatzgebiet
Classification Rule Learners	Klassifikation
Decision Trees	Klassifikation
Naive Bayes	Klassifikation
Nearest Neighbor	Klassifikation
Linear Regression	Regression
Model Trees	Regression
Regression Trees	Regression
Neural Networks	Klassifikation/Regression
Support Vector Machines	Klassifikation/Regression

Tabelle 12.1 Lernalgorithmen überwachtes Lernen



12.1.2 Unüberwachtes Lernen (Unsupervised Learning)

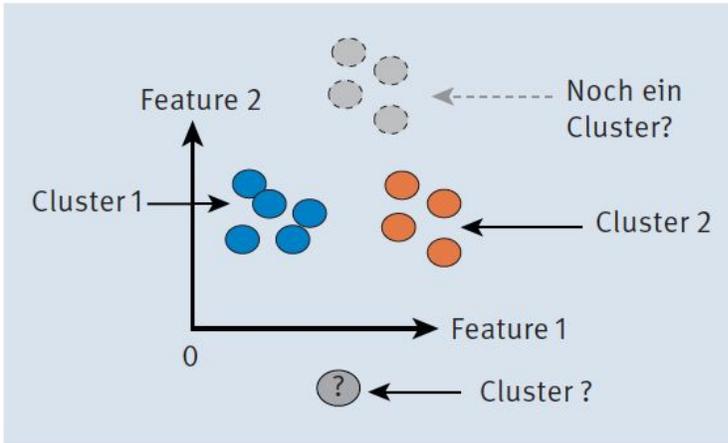


Abbildung 12.6 Cluster

Algorithmus	Einsatzgebiet
k-means	Clustering
Gaussian Mixture	Clustering
Hierarchical Clustering	Clustering
PCA-Hauptkomponentenanalyse (Dimensionsreduzierung)	Clustering
Hidden Markov Model	Clustering
Neural Networks	Clustering
Hopfield Networks	Assoziation
Self-Organizing Map (SOM)	Clustering
Association Rules	Pattern Discovery

Tabelle 12.2 Lernalgorithmen für unüberwachtes Lernen



Beispiel: SOM

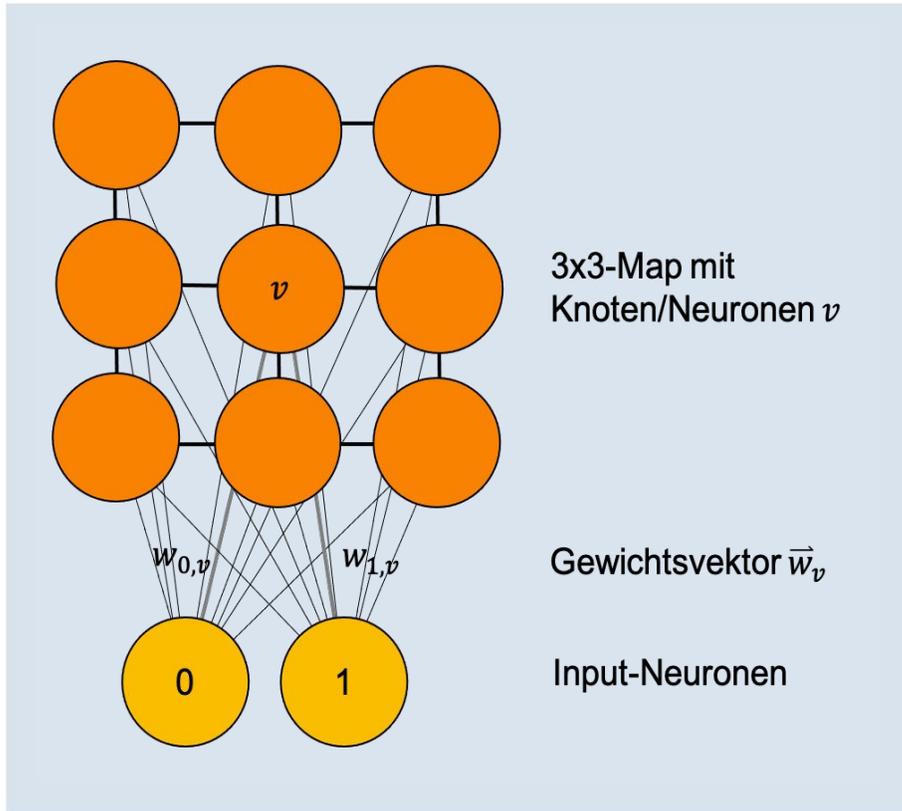


Abbildung 12.7 SOM mit zwei Input-Neuronen und einer 3x3-Map

Algorithmus

1. Initialisiere die Gewichte der Gewichtsvektoren \vec{w}_v , zu jedem Knoten v mit kleinen zufälligen Werten aus dem Intervall $[0,1]$.
2. Wiederhole für eine bestimmte Anzahl an Iterationen I :
 - Wähle einen Input-Vektor $\vec{x}(s)$ aus der Menge der Trainingsbeispiele X zufällig aus, wobei s der Iterationsschritt ist.
 - Berechne die Distanz d vom Input-Vektor zu den Gewichtsvektoren aller Knoten mit $d(\vec{x}(s), \vec{w}_v(s))$.
 - Bestimme den Knoten u mit der kleinsten Distanz; dieser wird auch als *Best Matching Unit (BMU)* bezeichnet.
 - Passe den BMU-Gewichtsvektor $\vec{w}_u(s)$ und den der Knotennachbarn in Richtung des Input-Vektors an. Die Anpassung der Gewichte erfolgt anhand folgender Formel:

$$\vec{w}_v(s+1) = \vec{w}_v(s) + \theta(u, v, s) \cdot \alpha(s) \cdot (\vec{x}(s) - \vec{w}_v(s))$$

Dabei haben die unterschiedlichen Elemente der Formel folgende Bedeutung:

- ▶ $\vec{w}_v(s)$ – der Gewichtsvektor für Knoten mit Index v zum Iterationsschritt s
- ▶ s – der Iterationsschritt
- ▶ u – der Index für die BMU bezüglich des Input-Vektors $\vec{x}(s)$
- ▶ $\vec{x}(s)$ – der Input-Vektor
- ▶ $\alpha(s)$ – eine Lernrate, die im Laufe der Zeit abnimmt
- ▶ $\theta(u, v, s)$ – die Nachbarschaftsfunktion zur Ermittlung der Nachbarn mit Index v für BMU mit Index u zum Iterationsschritt s (Abbildung 12.10). Mit zunehmendem Iterationsschritt s schrumpft die Nachbarschaft und bekommt Werte im Intervall $[0,1]$.



SOM: Euklidische Distanz

$$d(\vec{x}, \vec{w}) = \|\vec{x} - \vec{w}\|$$
$$= \sqrt{(x_1 - w_1)^2 + (x_2 - w_2)^2 + \dots} = \sqrt{\sum_i (x_i - w_i)^2}$$

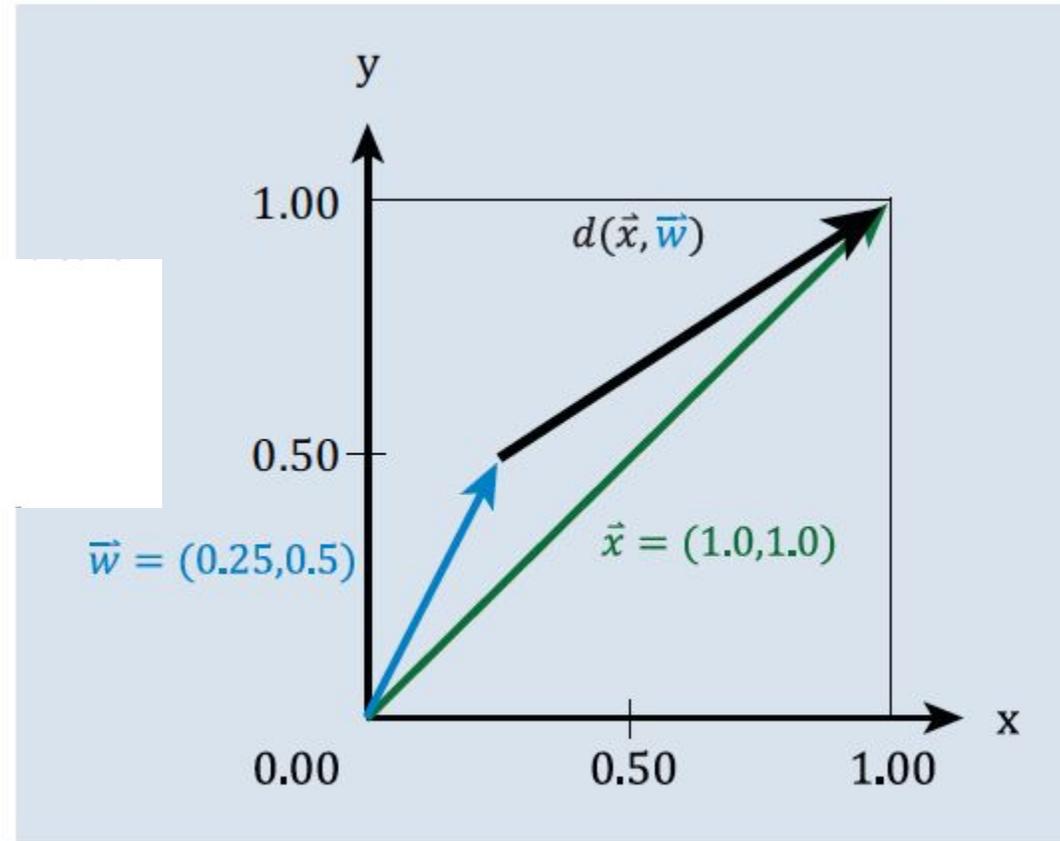


Abbildung 12.8 Euklidische Distanz von zwei Vektoren



SOm: Lernrate

$$\alpha(s) = \alpha(0) \cdot \frac{1}{s}$$

$$\alpha(0) = 0.80$$

$$\alpha(s, I) = \alpha(0) \cdot \left(1 - \frac{s}{I}\right)$$

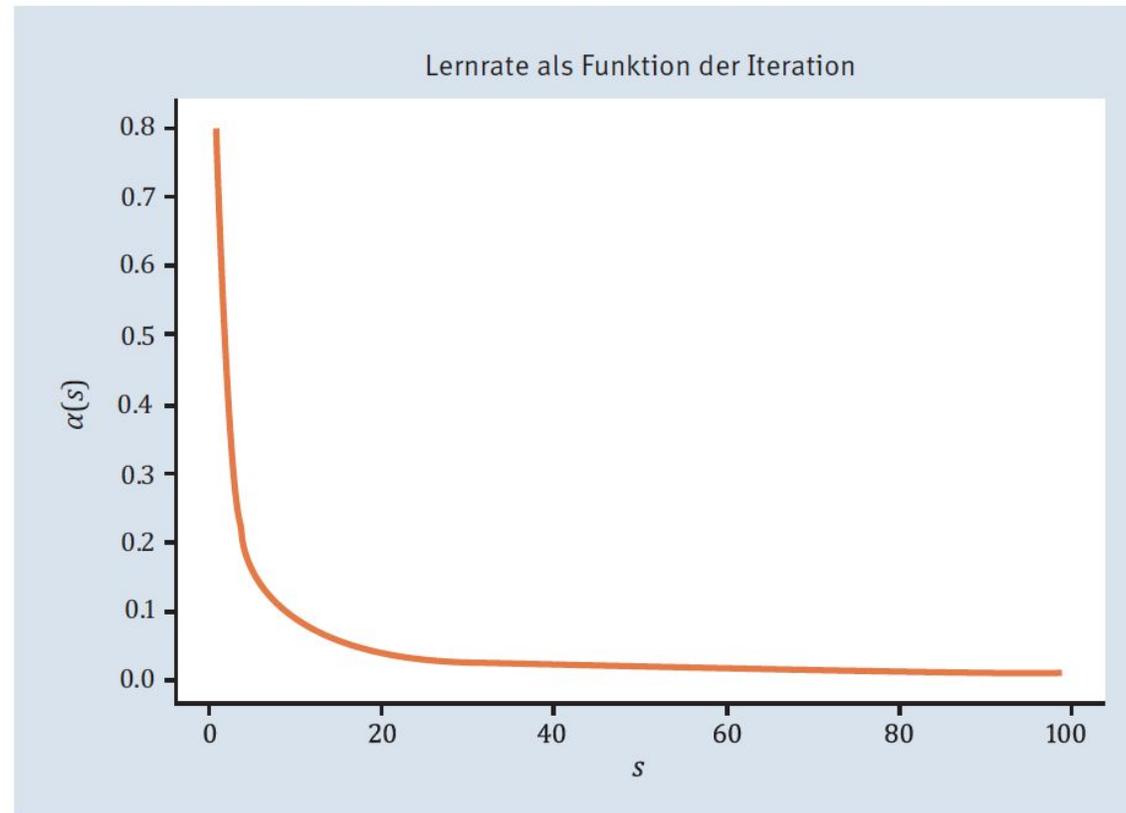


Abbildung 12.9 Abfall der Lernrate im Laufe der Zunahme der Iterationen



SOM: Nachbarschaft

Diesen Sachverhalt gießen wir wieder in eine berechenbare Form. Für die Berechnung, die wir für die **Nachbarschaft** zwischen Knoten u und v verwenden werden, gilt die folgende hübsche Formel:

$$\theta(u, v, s) = \begin{cases} h_{u,v}(s), & \text{falls } d(u, v) \leq r(s, I) \\ 0, & \text{sonst} \end{cases}$$

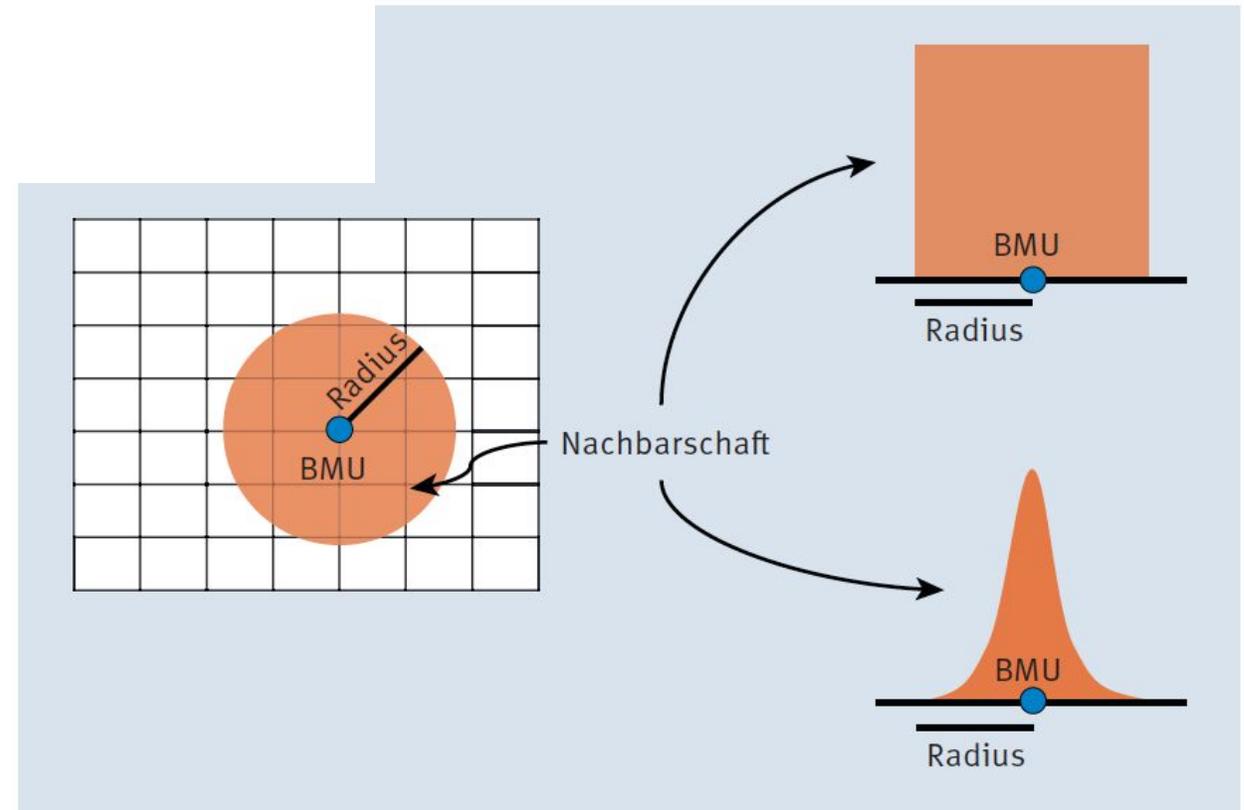


Abbildung 12.10 Unterschiedliche Nachbarschaftsfunktionen



SOM: Radius

$$r(s, I) = r(0) \cdot e\left(\frac{s}{I}\right)$$

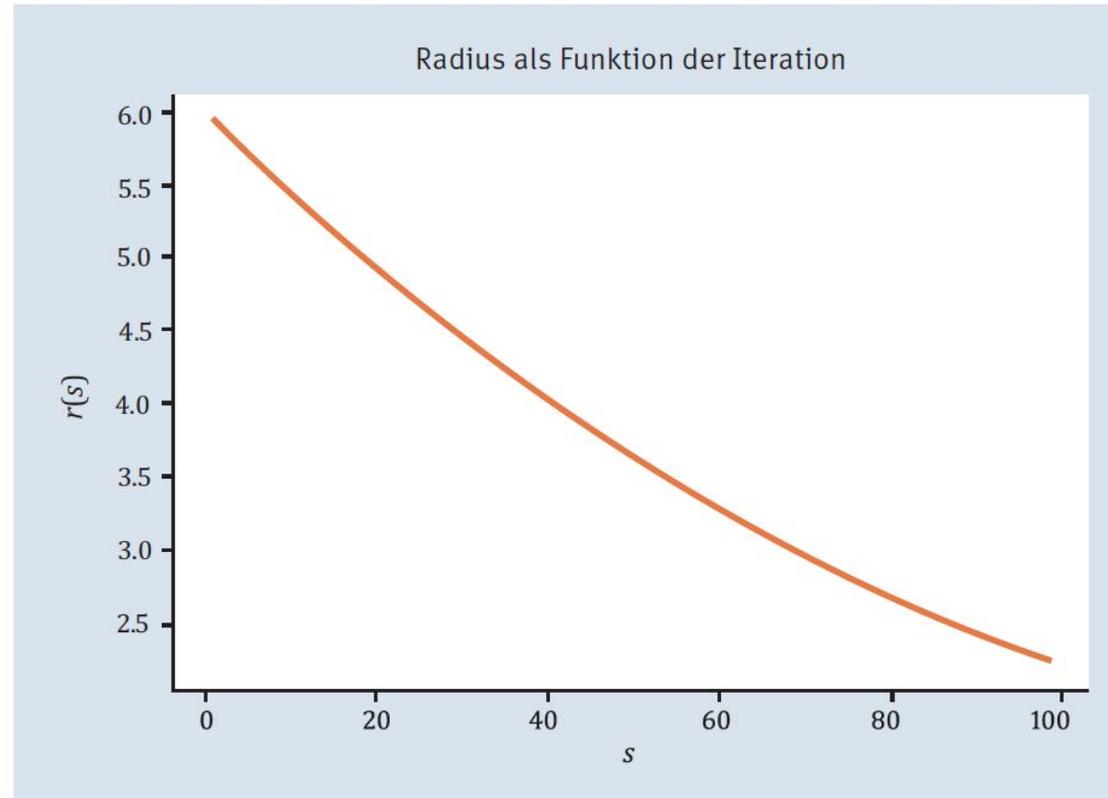


Abbildung 12.11 Radius als Funktion der Iteration



SOM: Lernen

Bevor wir zur Implementierung schreiten, sehen wir uns noch einmal die Lernformel für die SOM mit dem bisher Gehörten an:

$$\vec{w}_v(s+1) = \vec{w}_v(s) + \underbrace{\theta(u, v, s)}_{\text{kleiner 1}} \cdot \underbrace{\alpha(s)}_{\text{kleiner 1}} \cdot \underbrace{(\vec{x}(s) - \vec{w}_v(s))}_{\text{Vektor zur BMU}}$$

Wir können die Formel nun folgendermaßen verbal zusammenfassen:

SOM-Lernen

Ändere pro Iterationsschritt das Gewicht eines Map-Knotens in Richtung der BMU.

Die Richtung vom Gewichtsvektor zur BMU ergibt sich aufgrund der Differenz der Vektoren $(\vec{x}(s) - \vec{w}_v(s))$, die im Anhang B detailliert erläutert wird.

Noch eine Anmerkung: Wenn zum Vektor $\vec{w}_v(s)$ der Differenzvektor dazugezählt wird, dann zeigt der neue Vektor $\vec{w}_v(s+1)$ mehr in Richtung Input-Vektor.



SOM: Programm

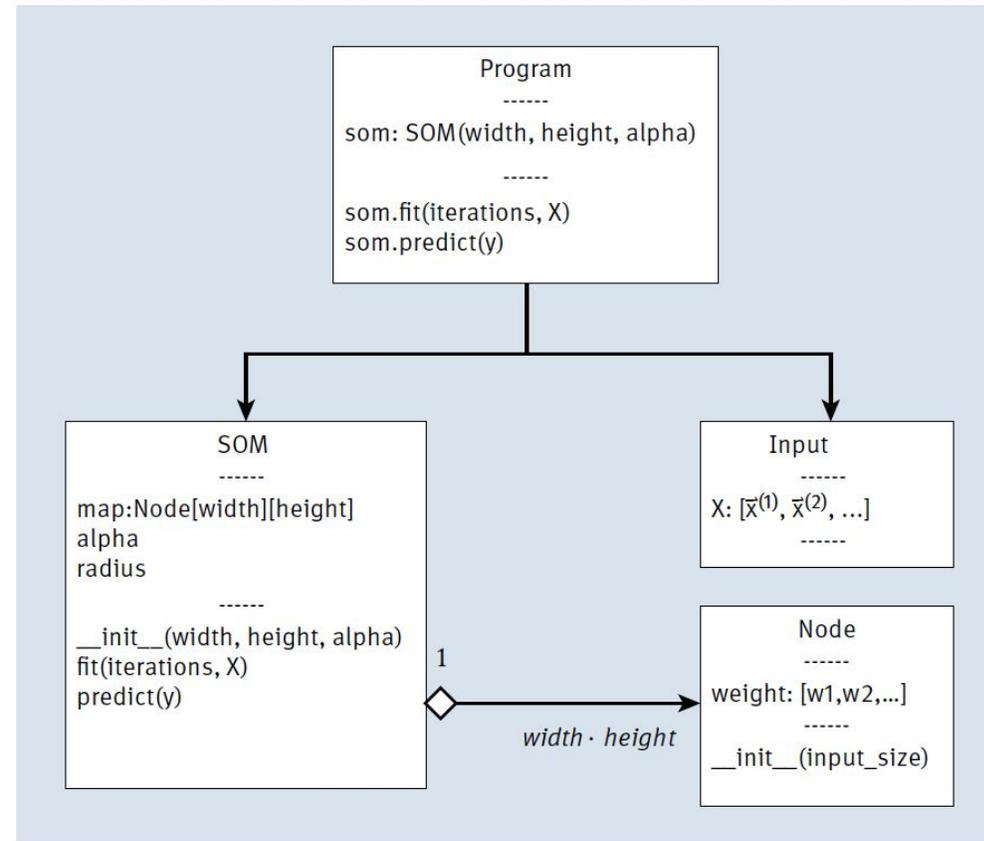


Abbildung 12.12 Die Bausteine zur SOM und ihre Zusammenhänge



Listing 12.2 SOM-Implementierung





SOM: Clustering

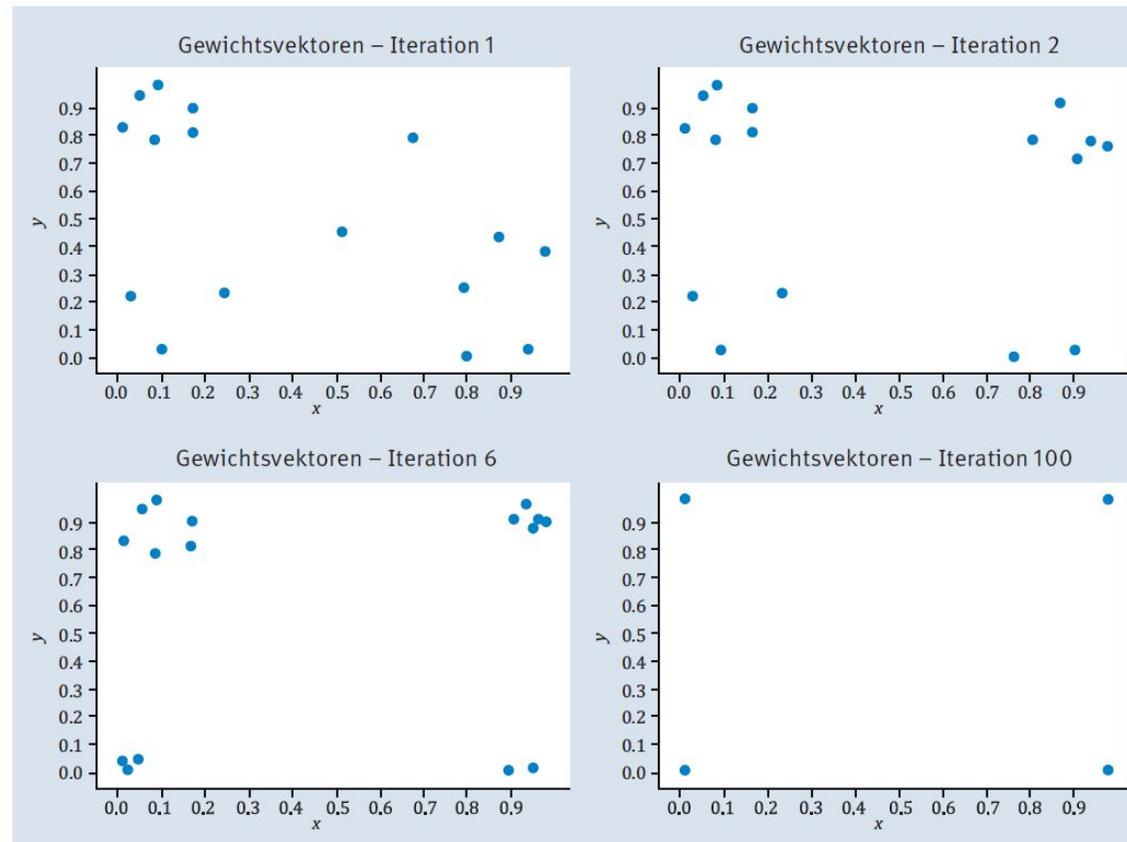
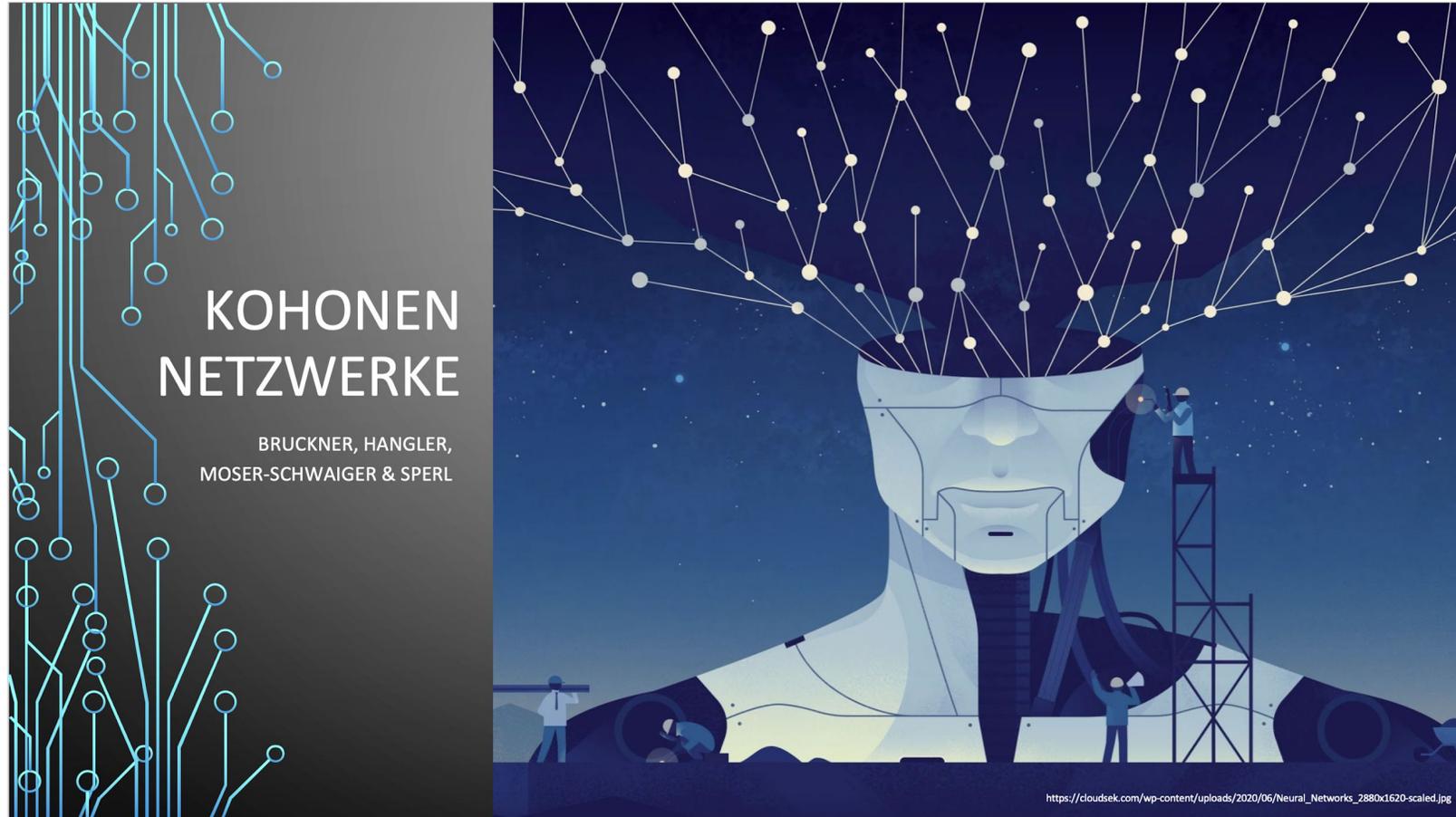


Abbildung 12.13 SOM und die Clusterbildung während des Lernvorgangs



Kohonen COWI presentation





EEG signal based Modified Kohonen Neural Networks for Classification of Human Mental Emotions

D. Jude Hemanth^{1,*}

¹Department of ECE, Karunya Institute of Technology and Sciences, Coimbatore, India

*Corresponding Author: D. Jude Hemanth, Email: judehemanth@karunya.edu

Computers and Electrical Engineering 68 (2018) 170–180



Contents lists available at [ScienceDirect](https://www.sciencedirect.com)

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng



Brain signal based human emotion analysis by circular back propagation and Deep Kohonen Neural Networks[☆]



D. Jude Hemanth^a, J. Anitha^a, Le Hoang Son^{b,c,*}

^a Department of ECE, Karunya University, Coimbatore, India

^b Division of Data Science, Ton Duc Thang University, Ho Chi Minh City, Vietnam

^c Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh City, Vietnam



12.1.3 Verstärkendes Lernen (Reinforcement Learning)

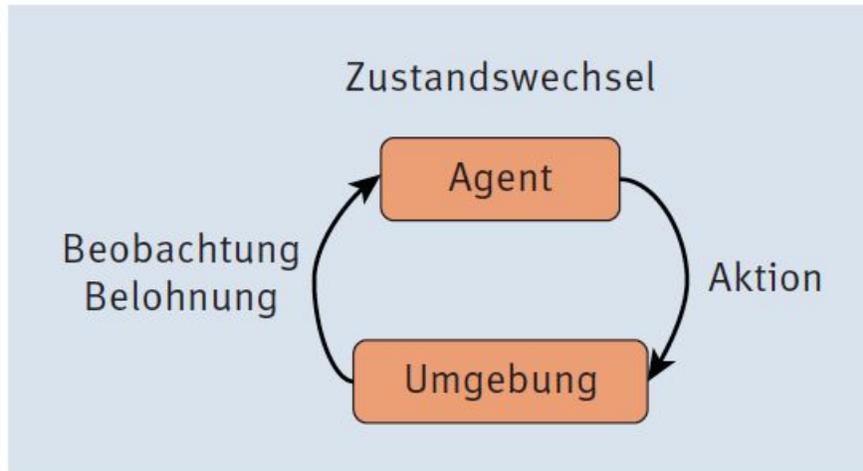


Abbildung 12.14 Reinforcement-System

Abschlagsfaktor γ («Gamma«)
oder auch *Diskontierungsfaktor* genannt.

$$G_t = \sum_{k=1}^T \gamma^k \cdot r_{t+k} \text{ mit } 0 \leq \gamma < 1$$



Q-Learning: Programm

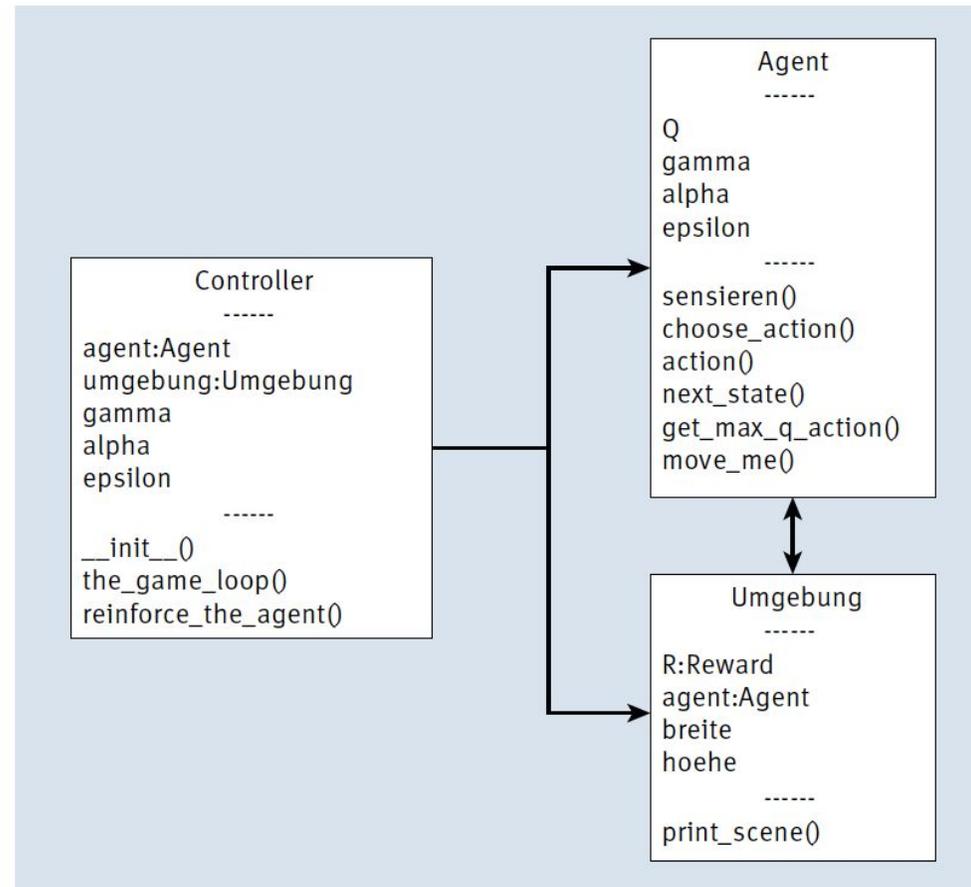


Abbildung 12.15 Klassendiagramm zum Q-Learning



Q-Learning: Game-Loop

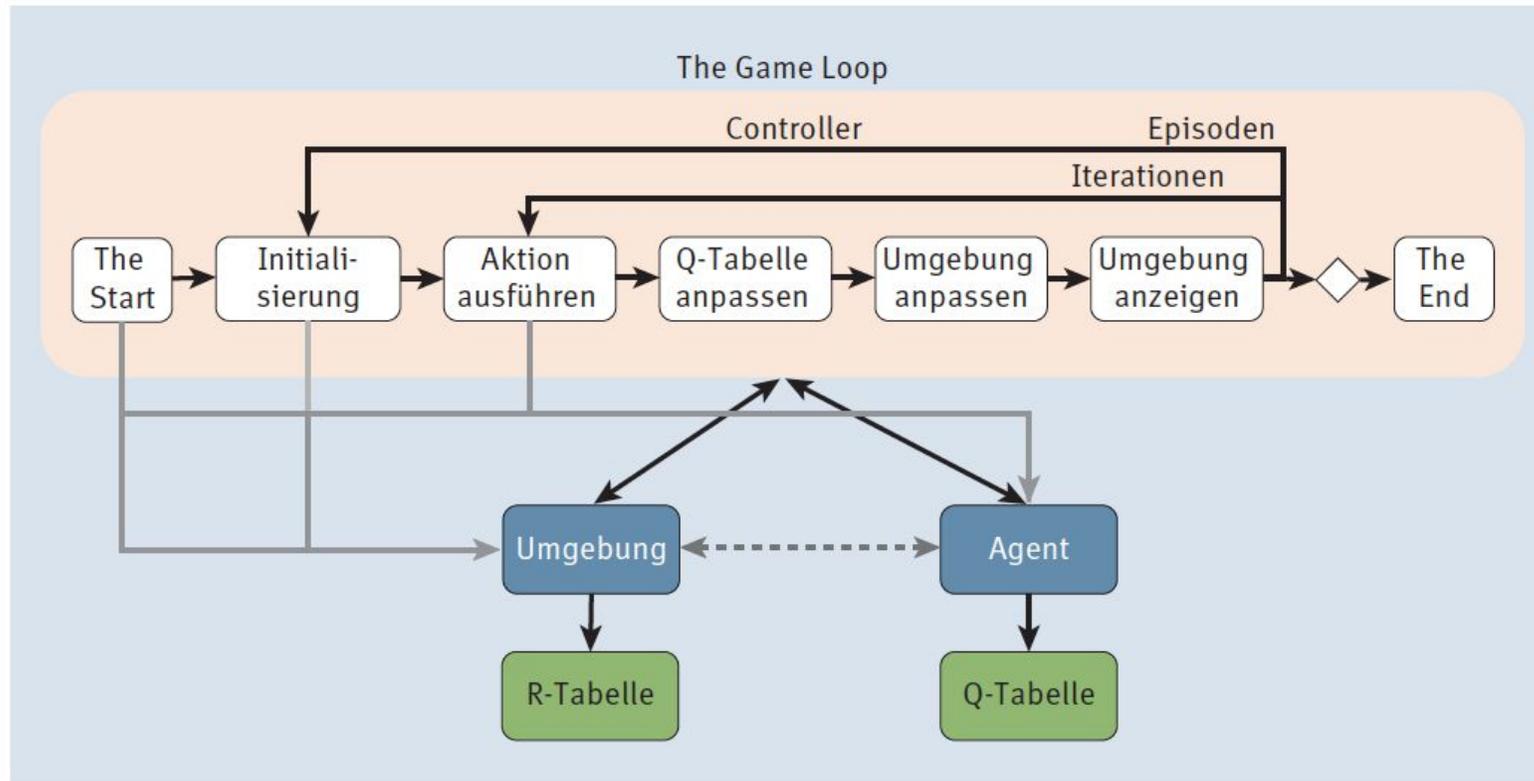


Abbildung 12.16 Game-Loop für das Q-Learning



Q-Learning: Algorithmus

Der Game-Loop sieht übersetzt in einen Algorithmus dann folgendermaßen aus:

Q-Learning-Algorithmus

1. Initialisiere den *Gamma-Wert* γ , der den Abschlag für zukünftige Belohnungen festlegt, die *Lernrate* und die *Belohnungstabelle* R , die für Zustände und Aktionen die resultierende Belohnung enthält.
2. Initialisiere die Q-Tabelle mit beliebigen Werten, zum Beispiel 0.
3. Für jede Episode:
 - Wähle einen zufälligen Initialzustand.
 - Wiederhole, bis der Zielzustand erreicht ist:
 - ▶ Wähle eine Aktion aus der Menge der möglichen Aktionen für den derzeitigen Zustand aus.
 - ▶ Berechne den neuen Q-Wert.
 - ▶ Ermittle die Belohnung zum aktuellen Zustand und zur Aktion.
 - ▶ Ermittle den maximalen Q-Wert für alle möglichen Aktionen im nächsten Zustand.
 - ▶ Setze den nächsten Zustand als den aktuellen Zustand.

$$Q^{\text{neu}}(s_t, a_t) \leftarrow Q^{\text{alt}}(s_t, a_t) + \underbrace{\alpha}_{\text{LR}} \cdot \left(r_t + \underbrace{\gamma}_{\text{DF}} \cdot \underbrace{\max_{a \in A} Q(s_{t+1}, a)}_{\text{ZW}} - Q^{\text{alt}}(s_t, a_t) \right)$$

Eine alternative Formulierung der Formel ist die folgende:

$$Q^{\text{neu}}(s_t, a_t) \leftarrow (1 - \alpha) \cdot Q^{\text{alt}}(s_t, a_t) + \alpha \cdot \left(r_t + \underbrace{\gamma}_{\text{DF}} \cdot \underbrace{\max_{a \in A} Q(s_{t+1}, a)}_{\text{ZW}} \right)$$



Q-Learning: Auswertung

Sind die Epochen absolviert, kann der Agent mittels der Q-Tabelle auf sein Wissen zugreifen und es für jede Startsituation nutzen. Dabei folgt der Agent seinem Gedächtnis von seinem aktuellen Zustand zum Zielzustand, und das könnte in algorithmischer Form folgendermaßen lauten:

1. Setze den aktuellen Zustand als Startzustand.
2. Wiederhole, bis der Zielzustand erreicht ist:
 - Suche in der Q-Tabelle in der Zeile mit dem aktuellen Zustand die Aktion mit dem maximalen Q-Wert.
 - Ermittle aus dem aktuellen Zustand und der Aktion den neuen Zustand und setze diesen als aktuellen Zustand.



Listing 12.3/4/5/6 Q-Learning





Q-Learning: Exploration vs. Exploitation

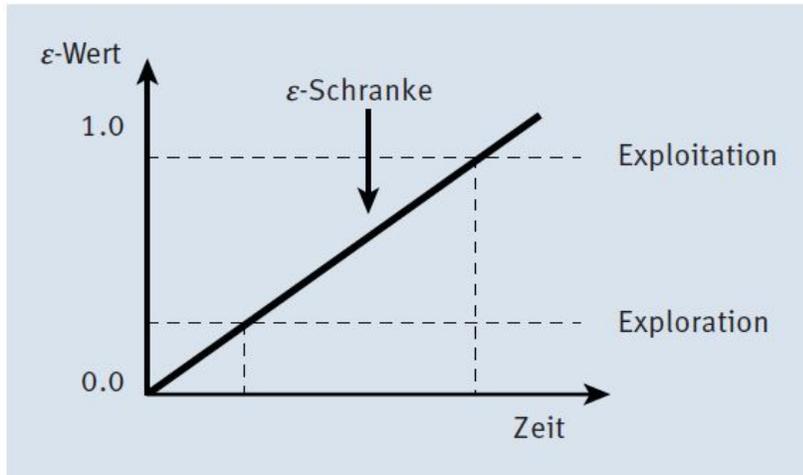


Abbildung 12.17 Exploration versus Exploitation

- Am Anfang ist die Auswahl der Aktion nicht einfach, da ja noch nichts gelernt wurde. Darum gibt es ein **Auswahlverfahren** für Aktionen, das auch die Lerndauer berücksichtigt. Dazu wird die sogenannte ϵ -Schranke (gesprochen *Epsilon*, E wie in Exploration oder Exploitation) definiert, zum Beispiel 0.2. Dann wird bei der Aktionsauswahl ein Zufallswert zwischen 0 und 1 ermittelt. Falls dieser Wert größer als die ϵ -Schranke ist, dann wird eine zufällige Aktion ermittelt. Falls der Wert kleiner als die ϵ -Schranke ist, dann wird die Aktion mit dem maximalen Q-Wert verwendet. Mit der Anzahl der Iterationen wird die Schranke angehoben, und die Wahrscheinlichkeit für eine zufällige Auswahl einer Aktion sinkt. Damit trägt man der Tatsache Rechnung, dass der Agent im Laufe der Zeit dazulernt.



Q-Learning: Ein Zahlenbeispiel

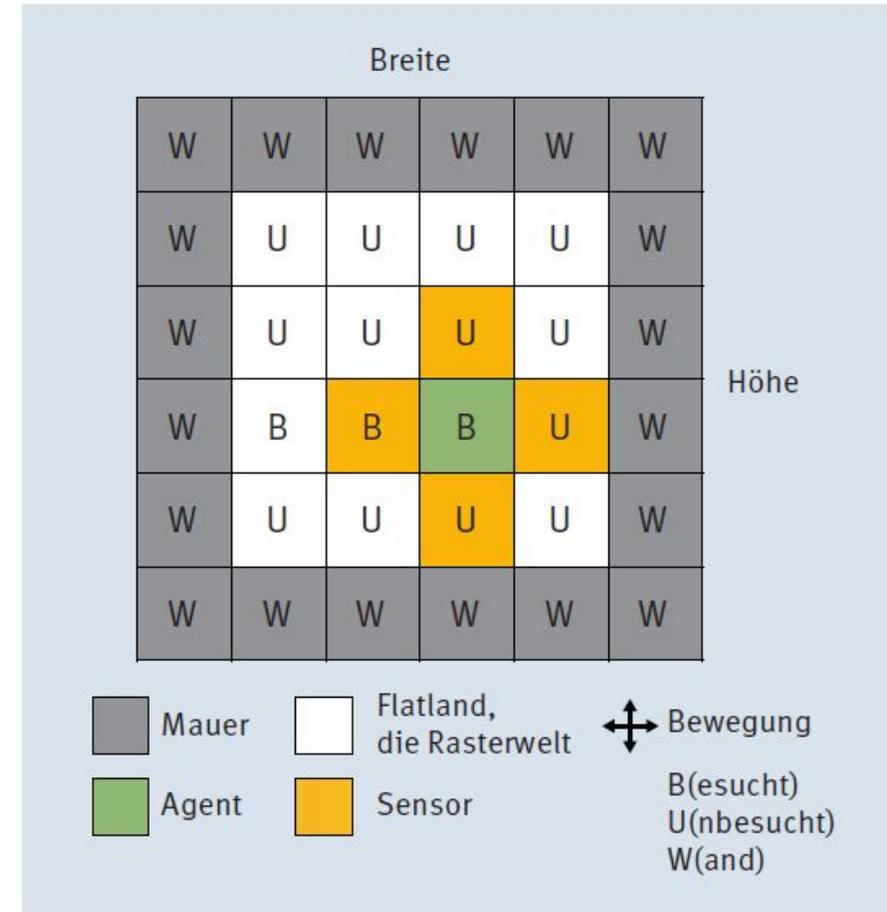


Abbildung 12.18 Flatland



Q-Learning: Initialisierung der R-Tabelle

R-Tabelle				
Zustände S	Aktionen			
(R,U,L,O)	rechts (0)	unten (1)	links (2)	oben (3)
(W,W,W,W)	-10.0	-10.0	-10.0	-10.0
...				
(U,U,W,W)	+5.0	+5.0	-10.0	-10.0
...				
(B,B,B,B)	-3.0	-3.0	-3.0	-3.0

Tabelle 12.3 Initialisierung der R-Tabelle; für Zustand (U,U,W,W) und Aktion »rechts« gibt es die Belohnung 5.0.



Q-Learning: Step

	W/-10.0				
W/-10.0	B	U/+5.0			
	U/+5.0				

Abbildung 12.19 Initialisierung (Schritt $t = 0$), mit Zustand (U,U,W,W)

		W/-10.0			
	B/-3.0	B	U/+5.0		
		U/+5.0			

Abbildung 12.20 Schritt $t = 1$, mit Zustand (U,U,B,W)



Q-Learning: Initialisierung der Q-Tabelle

Q-Tabelle				
Zustände	Aktionen			
(R,U,L,O)	rechts (0)	unten (1)	links (2)	oben (3)
...				
(U,U,W,W)	0.1	0.2	0.3	0.4
...				
(U,U,B,W)	0.1	0.2	0.3	0.4
...				

Tabelle 12.4 Initialisierung der Q-Tabelle



Q-Learning: Neuer Q-Wert

Q-Tabelle				
Zustände	Aktionen			
(R,U,L,O)	rechts (0)	unten (1)	links (2)	oben (3)
...				
(U,U,W,W)	5.32	0.2	0.3	0.4
...				
(U,U,B,W)	0.1	0.2	0.3	0.4
...				

Tabelle 12.5 Ermittlung der neuen Q-Werte für den ersten Schritt



Q-Learning: Wanderung

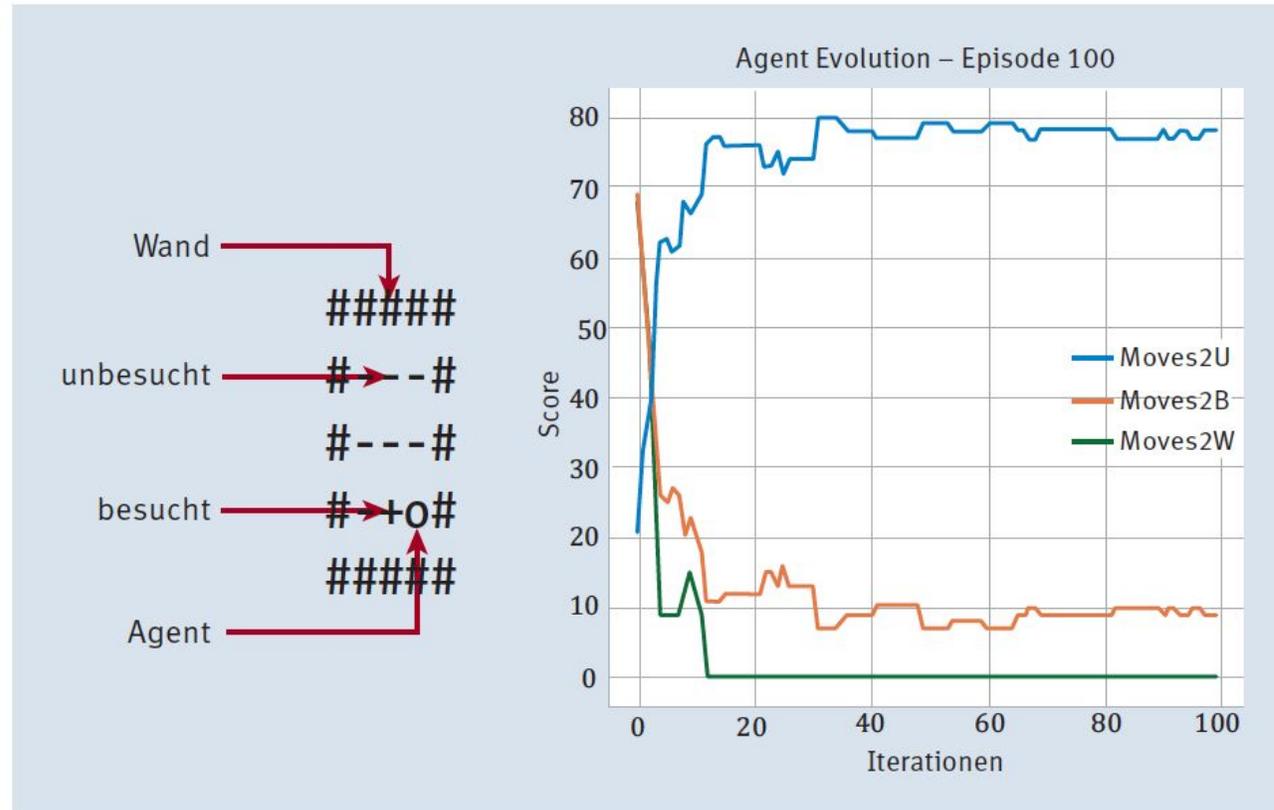


Abbildung 12.21 Der gelernte Agent in Episode 100 durchwandert die Umgebung.



Q-Learning: KNN

- ▶ Die **Anzahl** haben wir in Klammern geschrieben.
- ▶ **One-Hot(-Encoding) 4** bedeutet, dass vier Inputs verwendet werden und jeder den Wert 0 oder 1 annehmen kann, wobei immer nur genau einer der Inputs den Wert 1 annehmen kann.
- ▶ **Max 4** bedeutet, dass der maximale Wert gewinnt.
- ▶ Nun zu den Parametern:
- ▶ Input-Layer = aktueller State (4) + Aktion (One-Hot, 4) + Belohnung + nächster State (4)
- ▶ Hidden Layer = gleiche Dimension wie Input-Layer
- ▶ Output-Layer = entweder Q-Werte pro Aktion (maximal 4) oder einen Q-Wert
- ▶ Transferfunktion: ReLU
- ▶ Lernen: MSE
- ▶ Lernalgorithmus: Adam

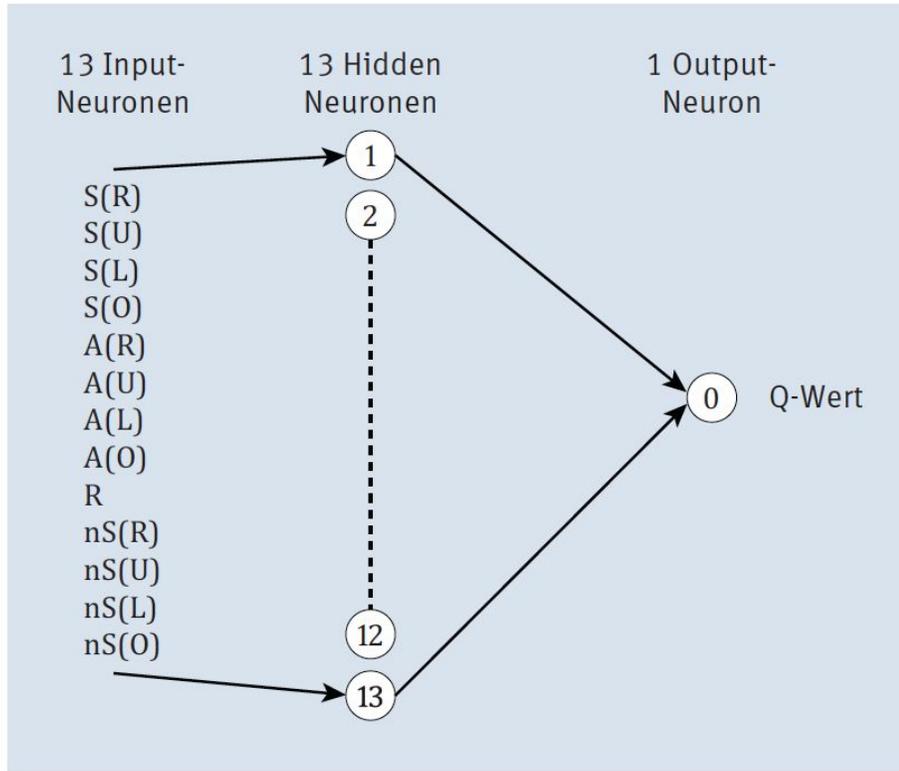


Abbildung 12.22 Q-NN-Architektur

wird diese Art von Speicher als *Ringspeicher* bezeichnet. Den Speicheraufbau, der gleichbedeutend mit dem Input für das KNN ist, haben wir folgendermaßen definiert:

1. Zustand/Sensor (4)
2. Aktion (4)
3. Belohnung
4. neuer Zustand (4)

Die zu optimierende Funktion, das heißt, die Fehlerfunktion, die durch Lernen zu minimieren ist, ist abgeleitet von der Q-Funktion:

$$MSE = \frac{1}{2} \cdot \sum \left(\underbrace{r_t + \gamma \cdot \max_{a \in A} Q(s_{t+1}, a)}_{\text{Zielwert}} - \underbrace{Q^{\text{alt}}(s_t, a_t)}_{\text{Vorhersage}} \right)^2$$



12.1.4 Teilüberwachtes Lernen (Semi-supervised Learning)

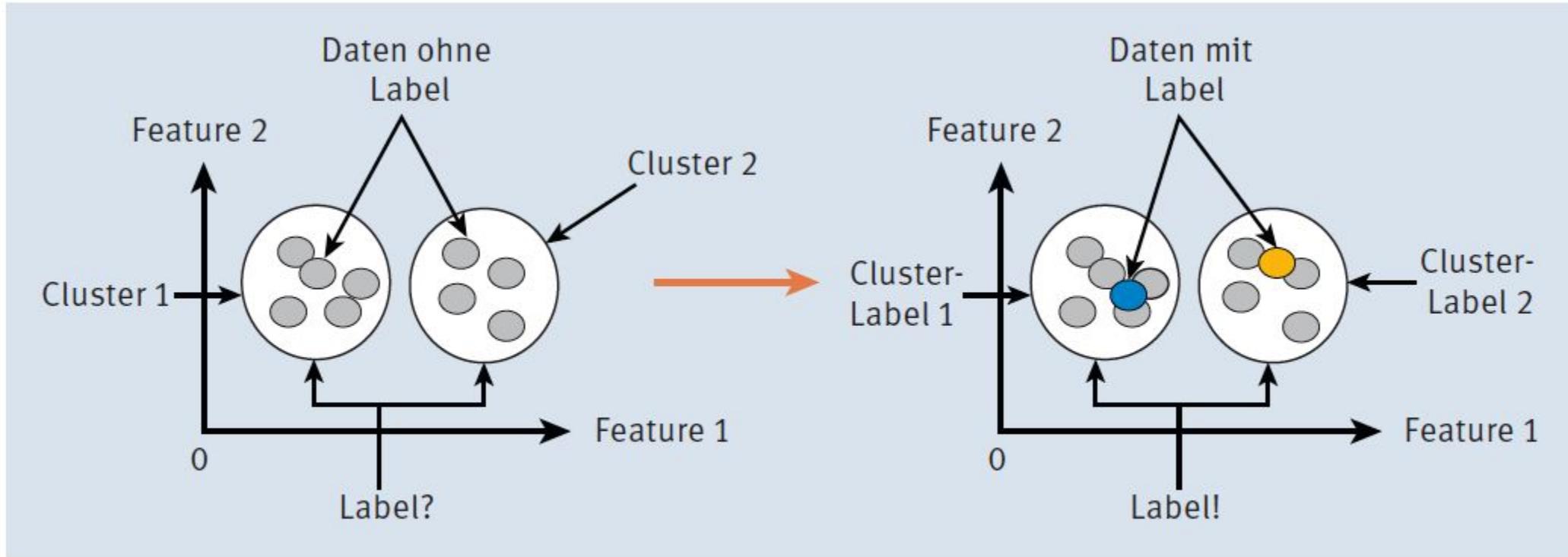


Abbildung 12.23 Semi-supervised Learning



12.2.1 Confusion Matrix

		tatsächliche Klasse		Summe
		A	nicht A	
vorhergesagte Klasse	A	True Positive (<i>TP</i>)	False Positive (<i>FP</i>)	<i>TP + FP</i>
	nicht A	False Negative (<i>FN</i>)	True Negative (<i>TN</i>)	<i>FN + TN</i>
	Summe	<i>TP + FN</i>	<i>FP + TN</i>	

Tabelle 12.6 Konfusionsmatrix



12.2.1 Confusion Matrix Beispiel

		tatsächliche Klasse		Summe
		A	nicht A	
vorhergesagte Klasse	A	9 (<i>TP</i>)	3 (<i>FP</i>)	12 (<i>TP</i> + <i>FP</i>)
	nicht A	6 (<i>FN</i>)	9 (<i>TN</i>)	15 (<i>FN</i> + <i>TN</i>)
Summe		15 (<i>TP</i> + <i>FN</i>)	12 (<i>FP</i> + <i>TN</i>)	27

Tabelle 12.7 Konfusionsmatrix für Stoppschild-Klassifikator

Die *Genauigkeit* als erstes einfaches Beispiel einer Kennzahl ergibt den Anteil der als korrekt klassifizierten Ereignisse (*TP*) zu der Gesamtheit der als Klasse »A« klassifizierten (*TP* + *FP*), d. h.

$$\text{Genauigkeit} = \frac{TP}{TP + FP}$$

Dadurch ergibt sich die *Genauigkeit* = $\frac{9}{12} = 0.75$.

Die *Sensitivität* (Empfindlichkeit) ist der Anteil der als korrekt klassifizierten Ereignisse (*TP*) zu der Gesamtheit aller als Klasse »A« zu klassifizierenden bzw. tatsächlich der Klasse »A« zugehörigen (*TP* + *FN*), d. h.

$$\text{Sensitivität} = \frac{TP}{TP + FN}$$

Dadurch ergibt sich die *Sensitivität* = $\frac{9}{15} = 0.6$.

Die *Spezifität* ist der Anteil der korrekt klassifizierten Nicht-Stoppschilder (*TN*) zu der Gesamtheit aller als Klasse »nicht A« klassifizierten (*FP* + *TN*), d. h.

$$\text{Spezifität} = \frac{TN}{FP + TN}$$

Dann ergibt sich dadurch die *Spezifität* = $\frac{9}{12} = 0.75$.



12.2.2 ROC-Curves

Schwellenwert	1-Spezifität	Sensitivität
0.00	0.00	0.00
0.10	0.00	0.13
0.20	0.01	0.37
...
0.50	0.25	0.60

1. Zuerst benötigen wir ein künstliches neuronales Netz, das eine Klassifikationsaufgabe im Sinne des vorherigen Abschnitts erfüllt (zum Beispiel Stoppschilder vs. Nicht-Stoppschilder).
2. Dann folgt die Ermittlung der **Sensitivität** und **Spezifität** für unterschiedliche Schwellenwerte für die Klassifikation der Netze in der Output-Schicht (z. B. Softmax) plus die Berechnung von *1-Spezifität* (False Positive Rate), also der Anteil der False-Positive-Klassifizierten aller »nicht A«, d. h. die als Stoppschild klassifiziert wurden, in der Menge aller als nicht als Stoppschild klassifizierten.
3. Mit diesen x/y-Wertepärchen (1-Spezifität, Sensitivität) kann die ROC-Kurve gezeichnet werden.

Tabelle 12.8 Spezifität und Sensitivität für die Ermittlung der ROC-Kurve

Schwellenwert	1-Spezifität	Sensitivität
...		
1.00	1.00	1.00

Tabelle 12.8 Spezifität und Sensitivität für die Ermittlung der ROC-Kurve (Forts.)



ROC-Kurve

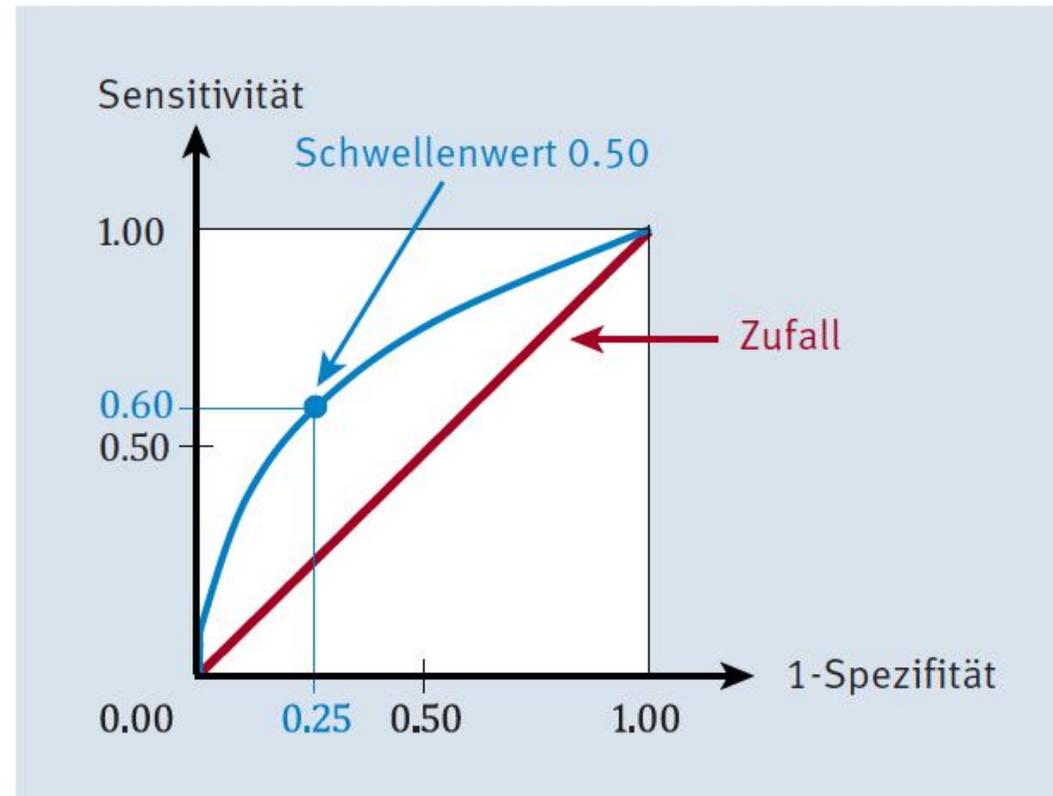


Abbildung 12.24 ROC-Kurve



Modellvergleich

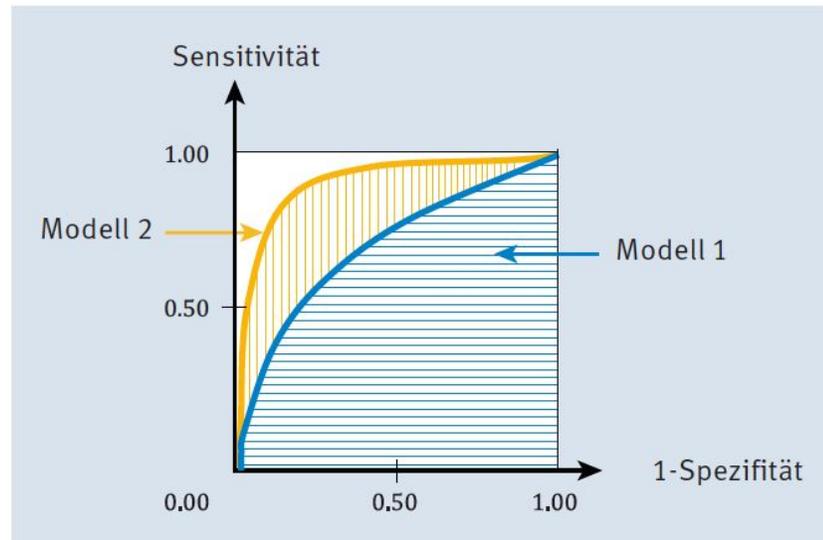


Abbildung 12.26 Zwei ROC-Kurven und ihre AUROC

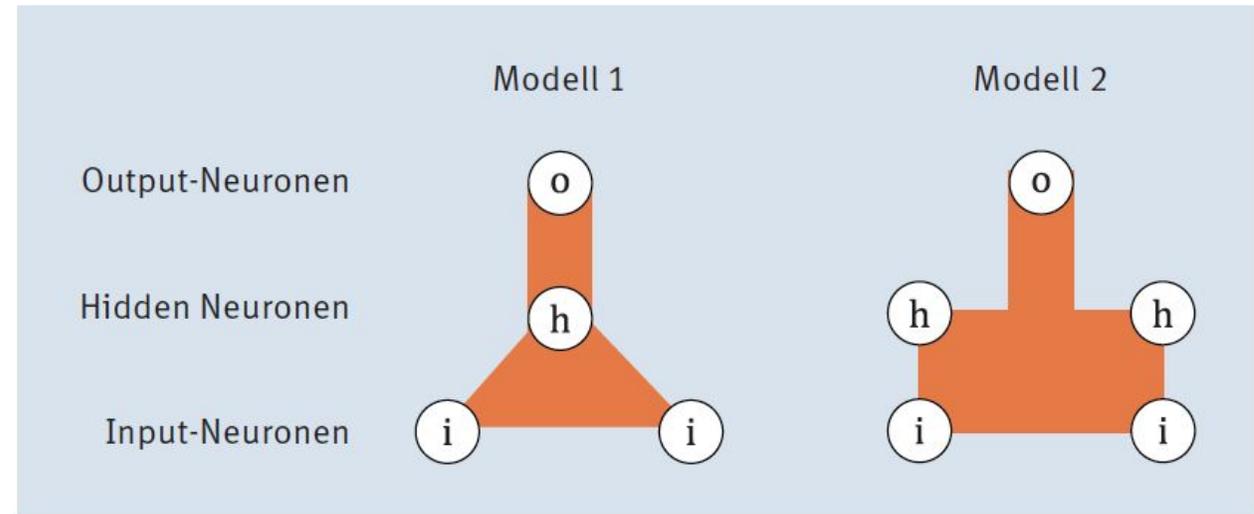


Abbildung 12.25 Modellvergleich



Kapitel 13

Anwendungsbereiche und Praxisbeispiele



13.1 Warmup

Listing 13.1/2/3/4/5 TensorFlow





13.1 Warmup

Wir haben also ein neuronales Netz mit 13 Eingabe-Features; die erste Schicht besteht aus 32 Neuronen, die zweite Schicht aus 64 Neuronen, und wir haben genau ein Ausgabeneuron, das uns den geschätzten Median-Hauswert liefert.

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 32)	448
dense_4 (Dense)	(None, 64)	2112
dense_5 (Dense)	(None, 1)	65
Total params: 2,625		
Trainable params: 2,625		
Non-trainable params: 0		

Abbildung 13.1 Netzwerkmodell für den Boston-Housing-Datensatz



13.2 Bildklassifikation



Abbildung 13.2 Beispiel für Bildklassifikation

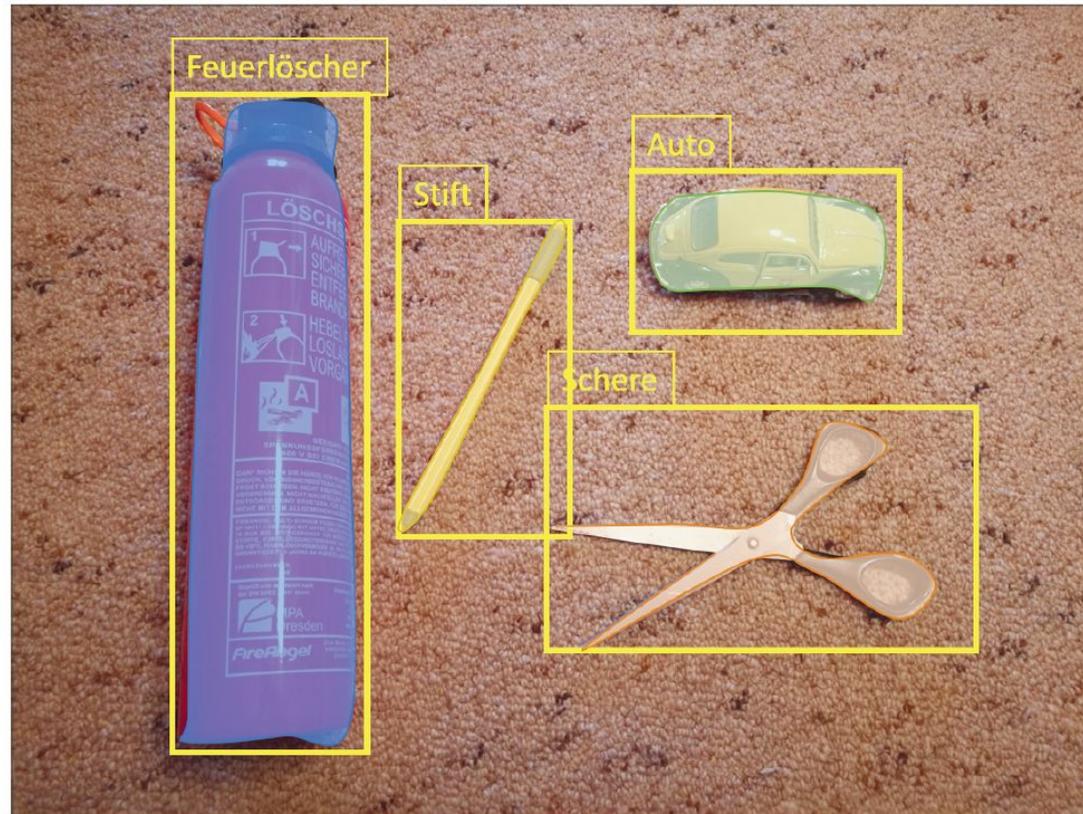


Abbildung 13.4 Beispiel für Objektsegmentierung



Abbildung 13.3 Beispiel für Objektidentifikation



13.2.2 Von Bienen und Hummeln

Listing 13.7 Biene & Hummel



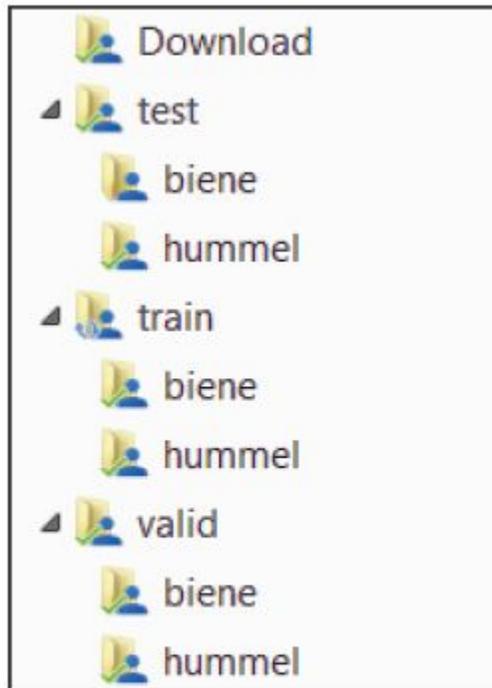


Abbildung 13.5 Verzeichnisstruktur für den Naive-Bees-Datensatz



Listing 13.8 Download der Datei » train_labels.csv«





Listing 13.9/10/11/12 Übertragen der Datei in einen DataFrame





Biene und Hummel



Abbildung 13.6 Bild Nummer 8 (Biene oder Hummel? Nicht nur wegen der Unschärfe schwer zu entscheiden)



Listing 13.13/14 Verteilung der Bilddaten auf die Netzwerkstruktur





Summary

The summary is textual and includes information about:

- The layers and their order in the model.
- The output shape of each layer.
- The number of parameters (weights) in each layer.
- The total number of parameters (weights) in the model.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 198, 198, 32)	896
max_pooling2d (MaxPooling2D)	(None, 99, 99, 32)	0
conv2d_1 (Conv2D)	(None, 97, 97, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 48, 48, 64)	0
conv2d_2 (Conv2D)	(None, 46, 46, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 23, 23, 128)	0
conv2d_3 (Conv2D)	(None, 21, 21, 128)	147584
max_pooling2d_3 (MaxPooling2D)	(None, 10, 10, 128)	0
flatten (Flatten)	(None, 12800)	0
dense (Dense)	(None, 512)	6554112
dense_1 (Dense)	(None, 1)	513
=====		
Total params: 6,795,457		
Trainable params: 6,795,457		
Non-trainable params: 0		

Abbildung 13.7 Schichten des Modells



Listing 13.15/16/17/18 Erzeugung der Trainings-, Test- und Validierungsdaten aus den Bilddaten



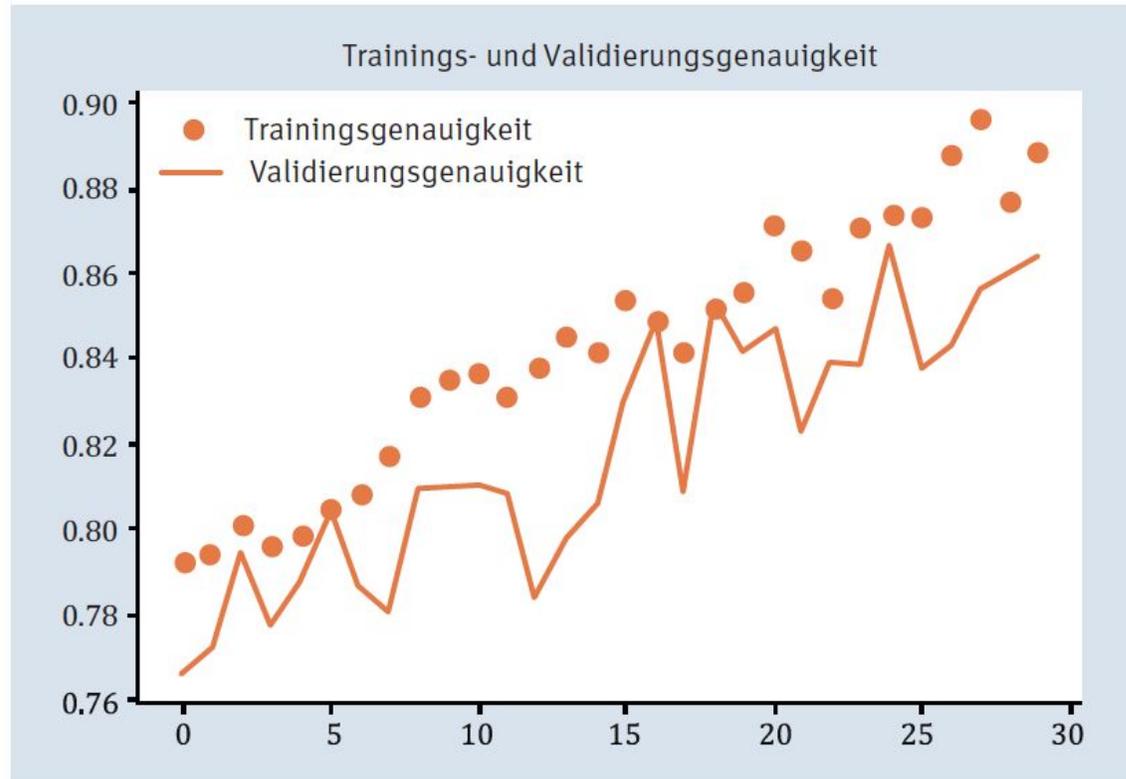


Abbildung 13.8 Verlaufskurve Trainings- und Validierungsgenauigkeit

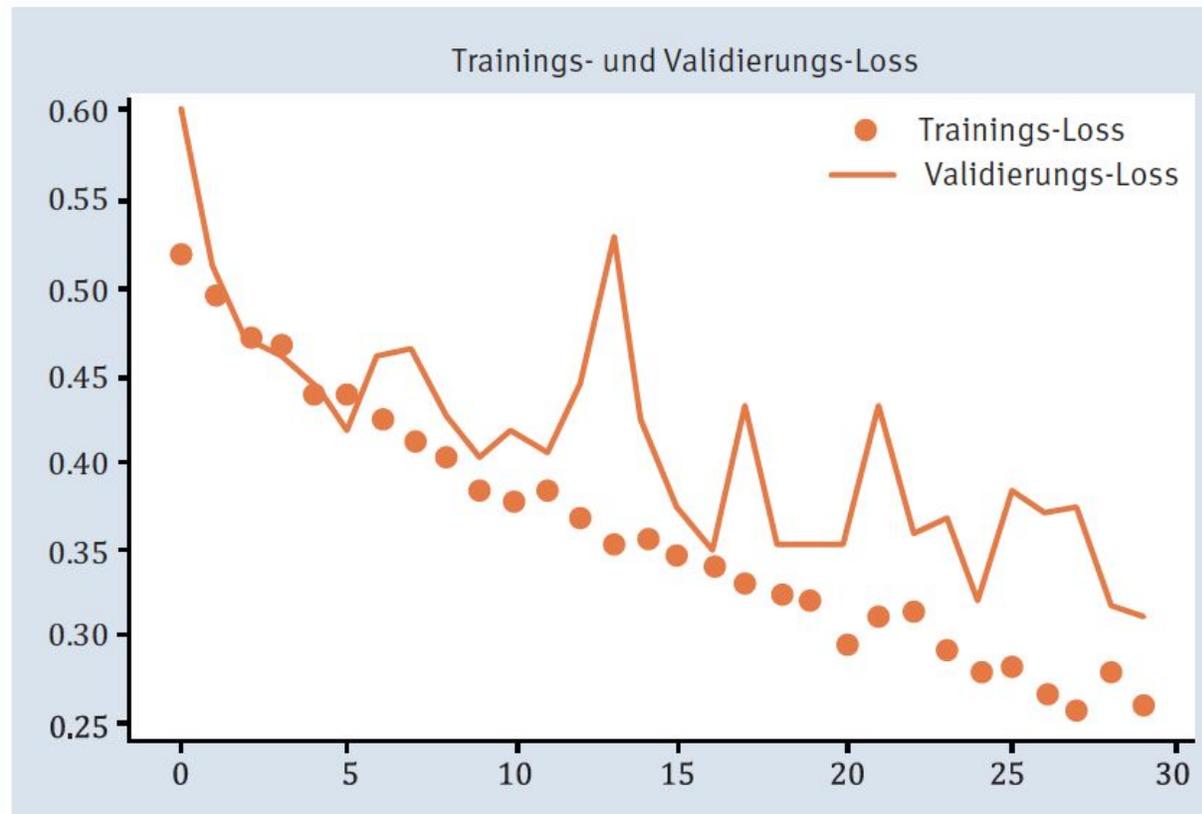


Abbildung 13.9 Verlaufskurve Trainings- und Validierungs-Loss



Abbildung 13.10 Verzeichnisstruktur des Katzen- und Hundedatensatzes



Listing 13.19/20/21 Import der Bibliotheken





Dogs & Cats



Abbildung 13.11 Auszug aus dem Testdatensatz für Katzen



Abbildung 13.12 Auszug aus dem Testdatensatz für Hunde



Listing 13.22/23 Import und Aufbau des vortrainierten CNN





Model Summary

Layer (type)	Output Shape	Param #
inception_v3 (Model)	(None, 4, 4, 2048)	21802784
flatten (Flatten)	(None, 32768)	0
dense (Dense)	(None, 256)	8388864
dense_1 (Dense)	(None, 512)	131584
dense_2 (Dense)	(None, 1)	513
Total params: 30,323,745		
Trainable params: 8,520,961		
Non-trainable params: 21,802,784		

Abbildung 13.13 Ausgabe von »model.summary()« unseres Netzes



Listing 13.24/25/26 Import und Aufbau des vortrainierten CNN



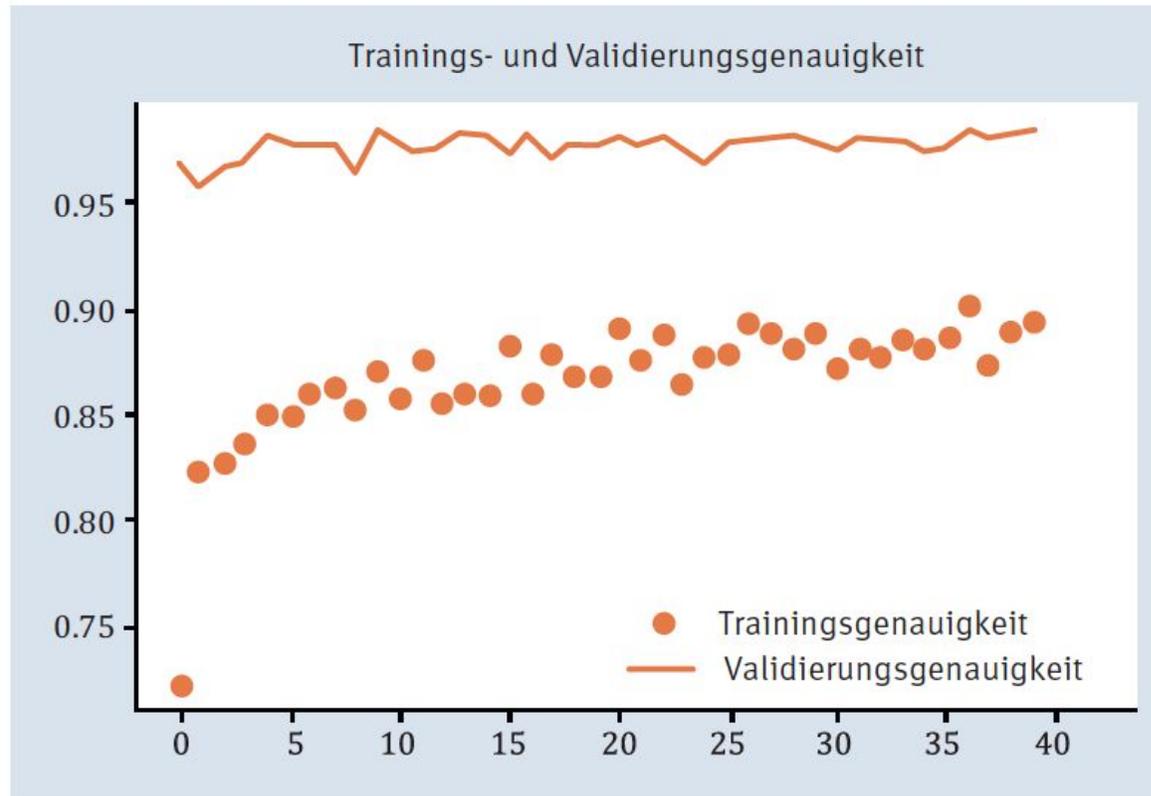


Abbildung 13.14 Genauigkeitsentwicklung über 40 Trainingsepochen

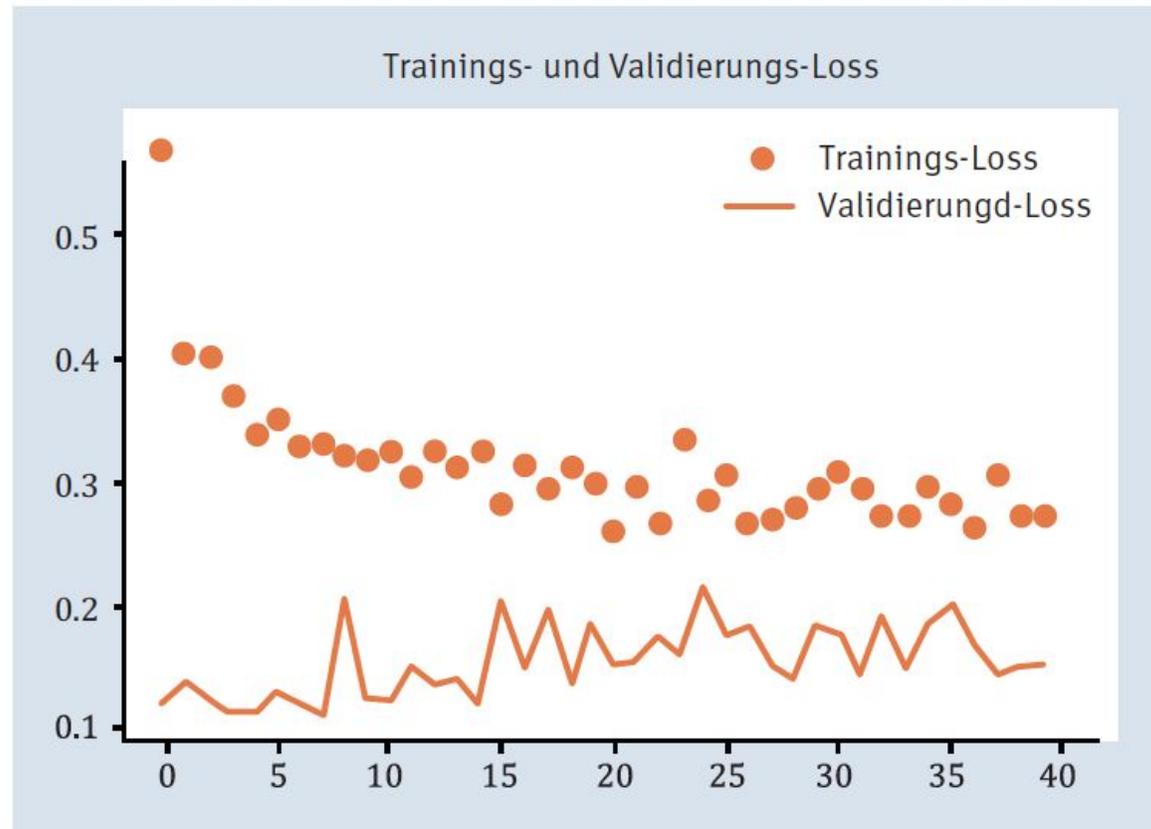


Abbildung 13.15 Verlustentwicklung über 40 Trainingsepochen

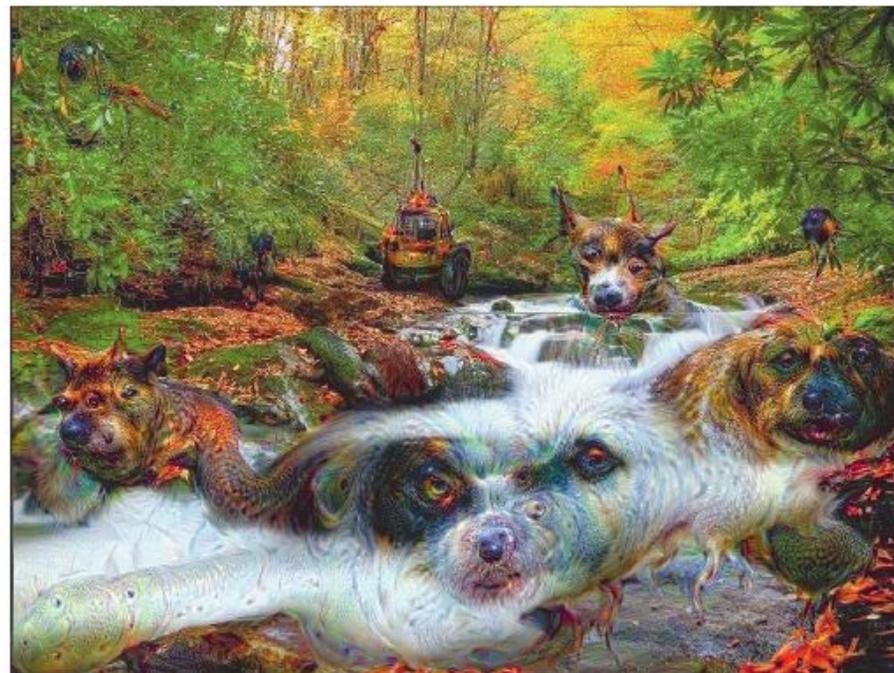
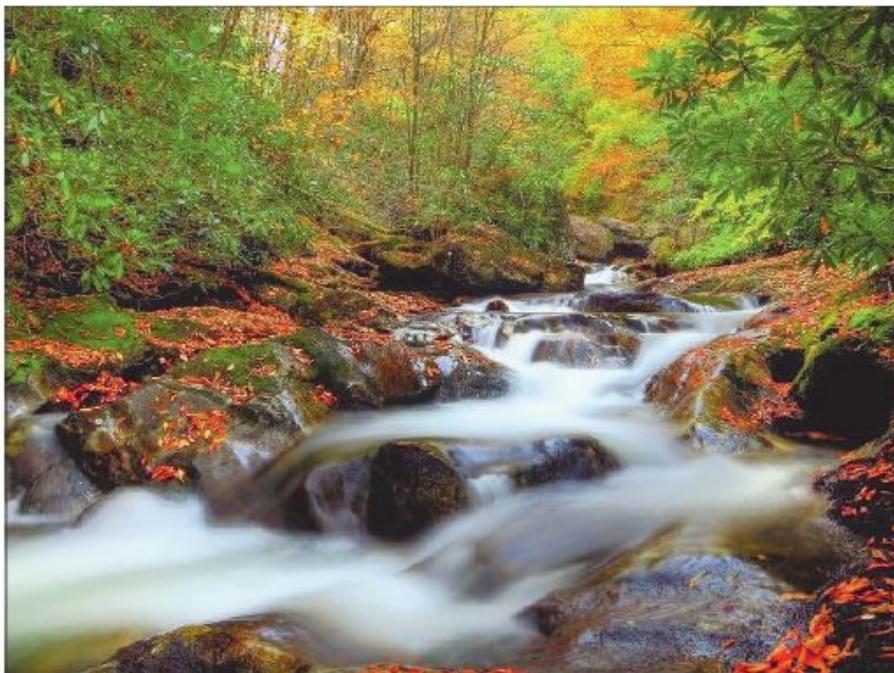


Abbildung 13.16 Bild einer Herbstlandschaft und die Traumvariante (Quelle: Wikimedia Commons)

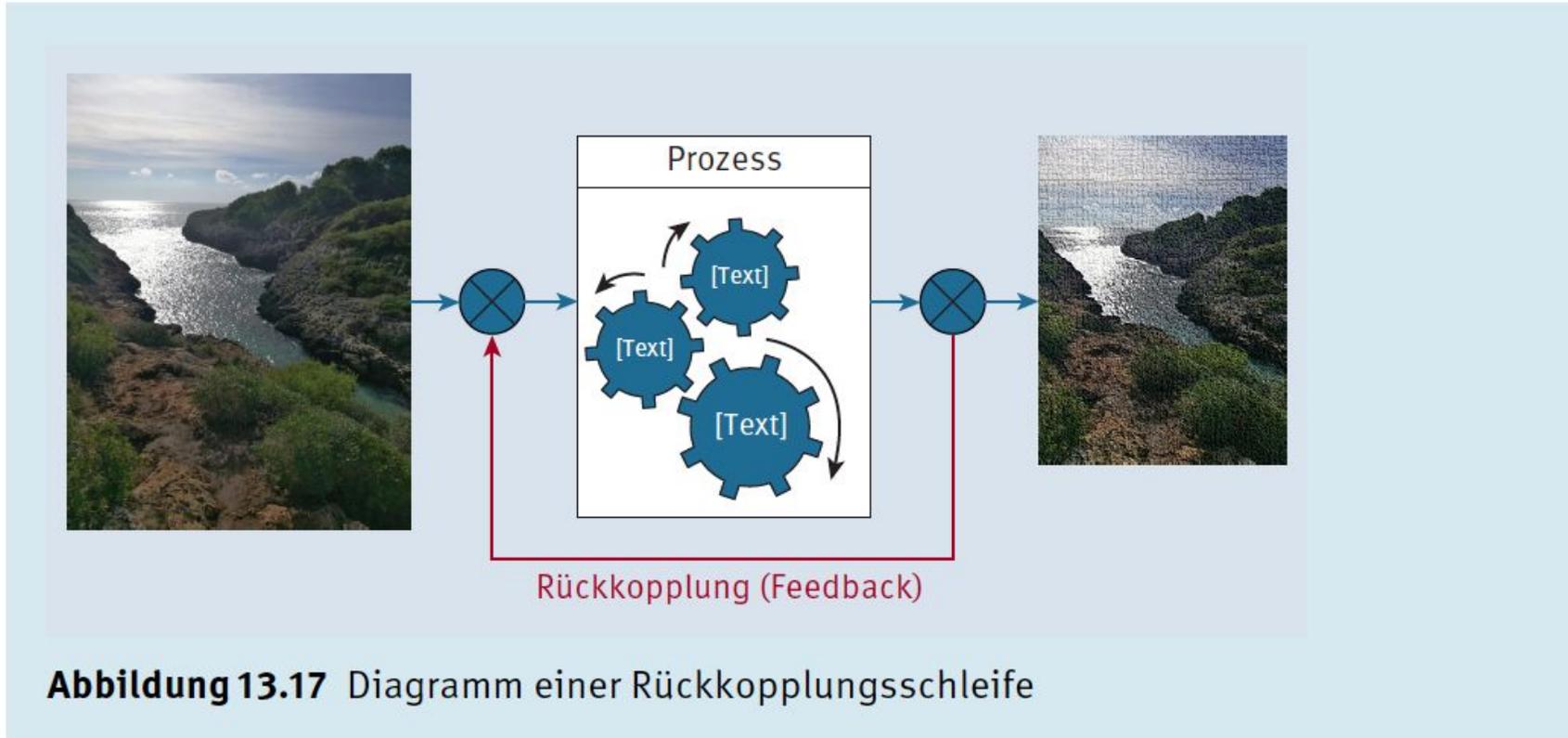


Abbildung 13.17 Diagramm einer Rückkopplungsschleife



Abbildung 13.18 Fünfstufige Bildpyramide

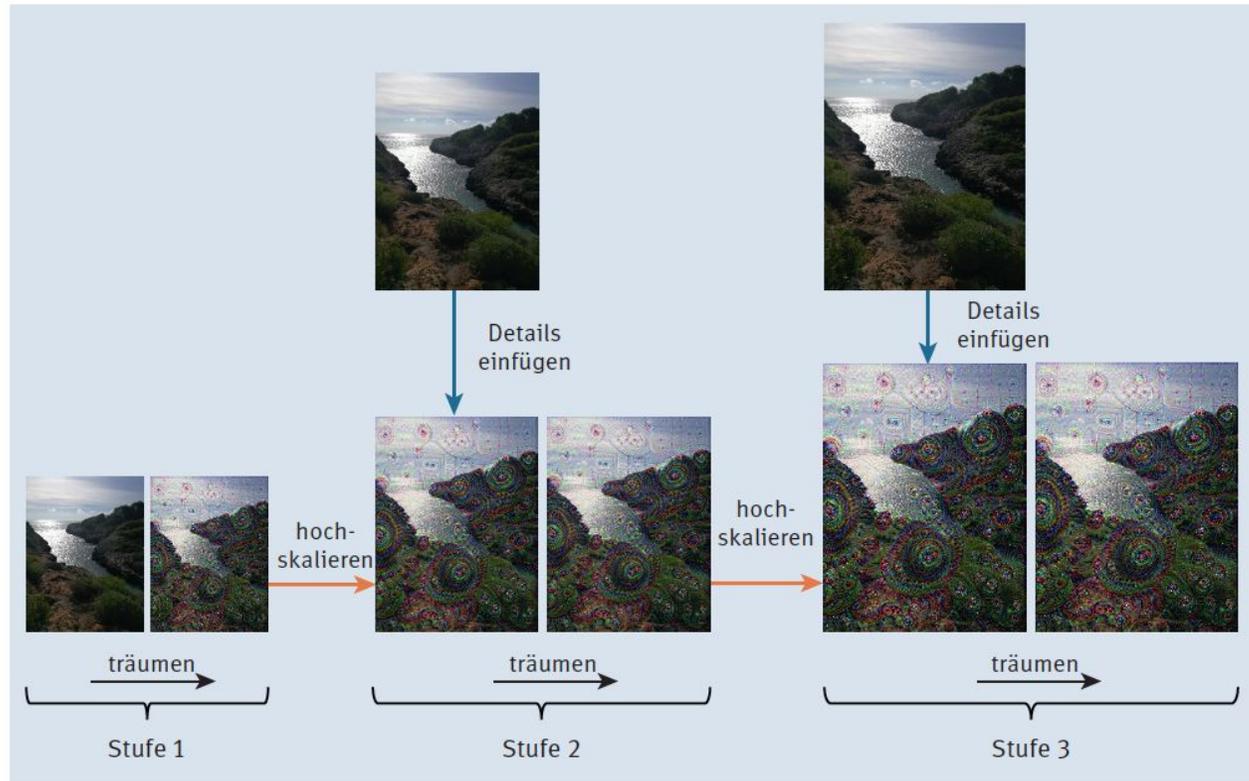


Abbildung 13.19 Bildhafte Darstellung des stufenweisen DeepDream-Algorithmus



Listing 13.28/29/30/31/32 Import der Bibliotheken und -Funktionen





Layer (type)	Output Shape	Param #
input_3 (InputLayer)	(None, None, None, 3)	0
block1_conv1 (Conv2D)	(None, None, None, 64)	1792
block1_conv2 (Conv2D)	(None, None, None, 64)	36928
block1_pool (MaxPooling2D)	(None, None, None, 64)	0
block2_conv1 (Conv2D)	(None, None, None, 128)	73856
block2_conv2 (Conv2D)	(None, None, None, 128)	147584
block2_pool (MaxPooling2D)	(None, None, None, 128)	0
block3_conv1 (Conv2D)	(None, None, None, 256)	295168
block3_conv2 (Conv2D)	(None, None, None, 256)	590080
block3_conv3 (Conv2D)	(None, None, None, 256)	590080
block3_conv4 (Conv2D)	(None, None, None, 256)	590080
block3_pool (MaxPooling2D)	(None, None, None, 256)	0
block4_conv1 (Conv2D)	(None, None, None, 512)	1180160
block4_conv2 (Conv2D)	(None, None, None, 512)	2359808
block4_conv3 (Conv2D)	(None, None, None, 512)	2359808
block4_conv4 (Conv2D)	(None, None, None, 512)	2359808
block4_pool (MaxPooling2D)	(None, None, None, 512)	0
block5_conv1 (Conv2D)	(None, None, None, 512)	2359808
block5_conv2 (Conv2D)	(None, None, None, 512)	2359808
block5_conv3 (Conv2D)	(None, None, None, 512)	2359808
block5_conv4 (Conv2D)	(None, None, None, 512)	2359808
block5_pool (MaxPooling2D)	(None, None, None, 512)	0
Total params: 20,024,384		
Trainable params: 20,024,384		
Non-trainable params: 0		

Abbildung 13.20 Ein Ausschnitt der Netzwerkstruktur von Inception_v3 (Ausgabe von »model.summary()«)



Listing 13.33/34/35/36/37 Loss- und Gradientenfunktion für den Optimierungsprozess



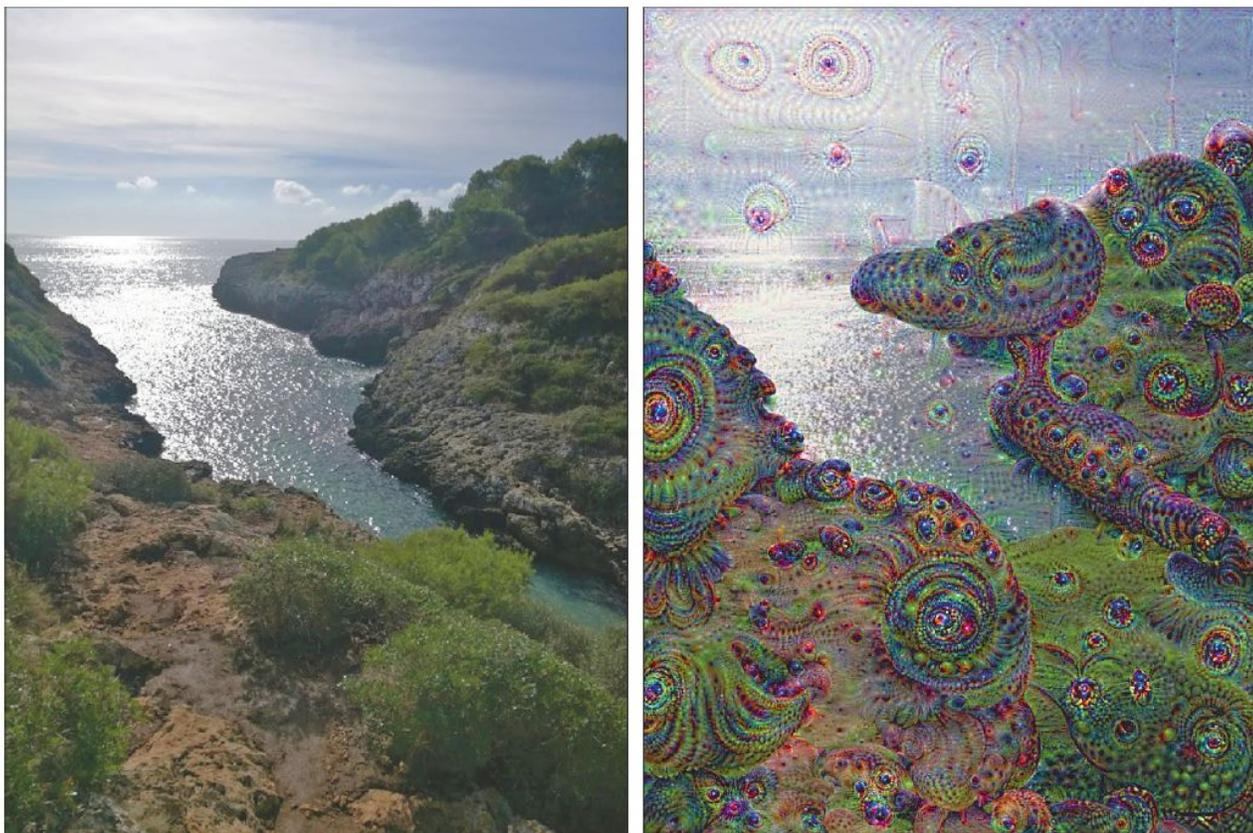


Abbildung 13.21 Original und Traumbild auf Basis von Inception_v3



Anhang C

TensorFlow 2 und Keras

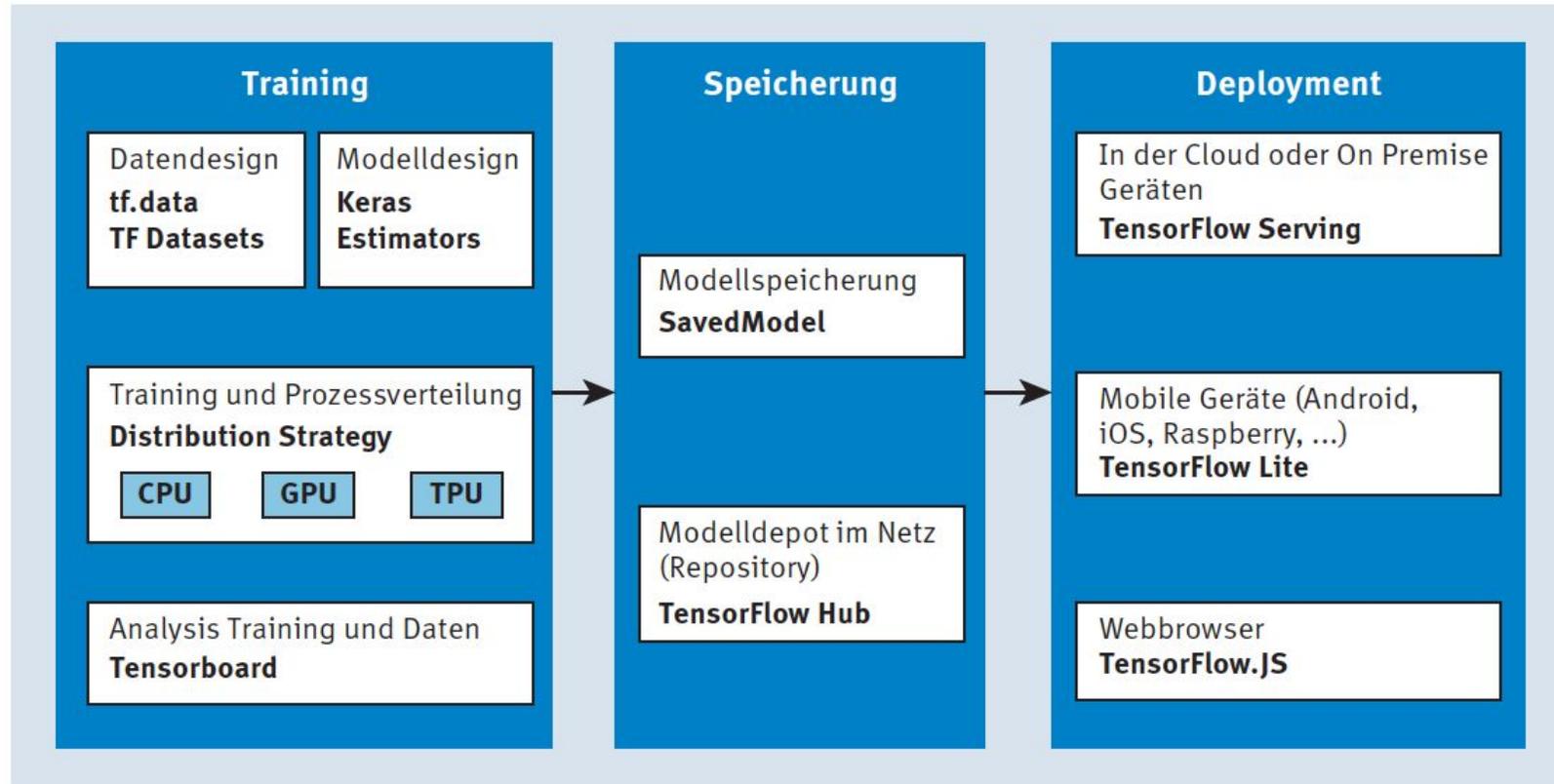


Abbildung C.1 Das TensorFlow-Ökosystem



Listing C.6 Einfaches neuronales Netz nach dem sequenziellen Modell (Keras)





Listing C.7 Neuronales Netz, mit funktionalem Paradigma erstellt (Keras)





Noch mehr ..



Beispiel

<https://www.scribbr.de/statistik/varianz/>

Wir haben 5 Personen gefragt, wie viele Stunden Sport sie pro Woche treiben.

Person	1	2	3	4	5
Stunden Sport/Woche	2	3	7	5	3

Mittelwert: $\frac{2+3+7+5+3}{5} = 4$

Varianz: $\frac{(2-4)^2 + (3-4)^2 + (7-4)^2 + (5-4)^2 + (3-4)^2}{5-1} = \frac{4+1+9+1+1}{4} = \frac{16}{4} = 4$

Merke

Die Standardabweichung erhalten wir, indem wir die Wurzel aus der Varianz ziehen.

Die Varianz berechnen

Formel zur Stichprobenvarianz

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

s^2	Varianz
n	Gesamtzahl der Beobachtungen
x_i	Beobachtungswert
\bar{x}	arithmetisches Mittel (Mittelwert) der Stichprobe

Formel zur Varianz der Grundgesamtheit

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2$$

σ^2	Varianz
N	Gesamtzahl
x_i	Beobachtungswert
μ	arithmetisches Mittel (Mittelwert) der Grundgesamtheit

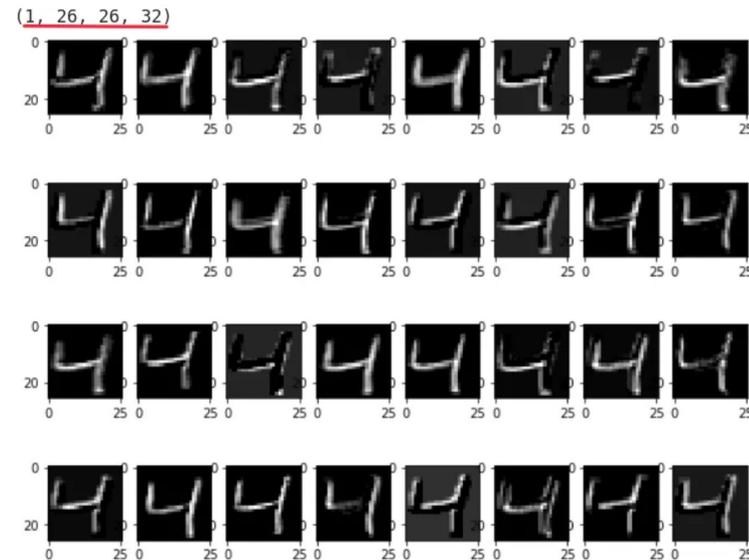


Keras filters (batch_size)

<https://androidkt.com/get-output-of-intermediate-layers-keras/>

```
visualize_conv_layer('conv_0')
```

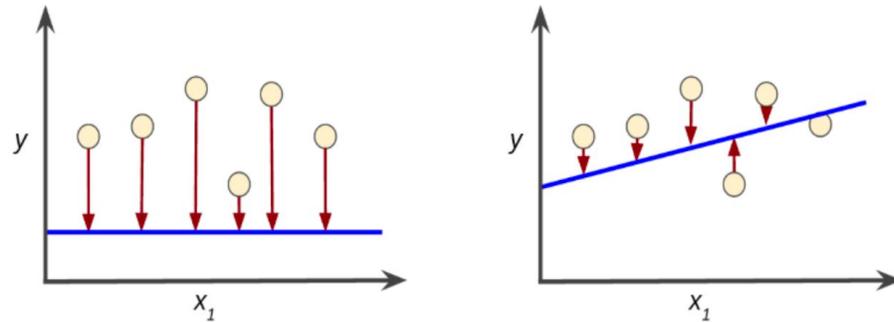
Our model has 32 filters. In the first layer, we can get some sense for what these layers are looking for by simply visualizing layer. We can just visualize that layer as a little 26x26x1 image with one channel. Because there are 32 of these filters we just visualize 32 little 26x26 images.





Loss and Accuracy

<https://intellipaat.com/community/368/how-to-interpret-loss-and-accuracy-for-a-machine-learning-model>



- The arrows represent a loss.
- The blue lines represent predictions.

They are two different metrics to evaluate your model's performance usually being used in different phases:

- **Loss** is often used in the training process to find the "best" parameter values for your model (e.g. weights in neural network). It is what you try to optimize in the training by updating weights.
- **Accuracy** is more from an applied perspective. Once you find the optimized parameters above, you use this metrics to evaluate how accurate your model's prediction is compared to the true data.



Wie geht es weiter?



Anhang