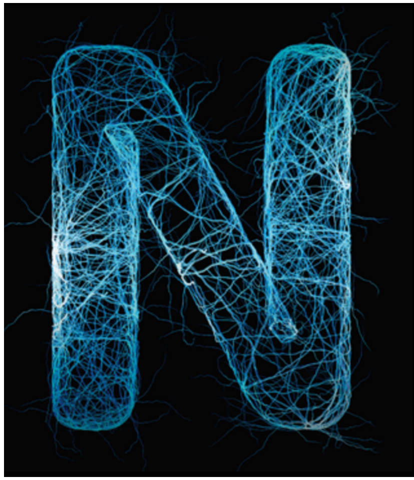


Künstliche neuronale Netze

and the rest …

Dr. Schwaiger Roland

Vorstellung

Dr. Roland Schwaiger

Located

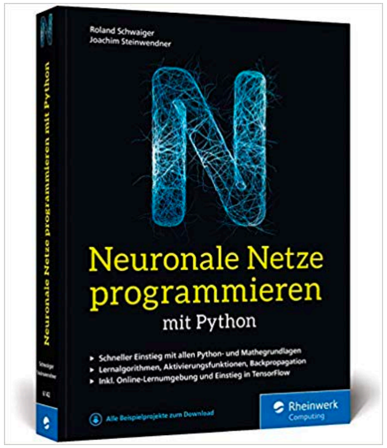
Bad Dürrnberg, Hallein, AT

Background

Mathematics (University Salzburg)
Computer Sciences (University Salzburg, Bowling Green State University)
Project & Process Management (SMBS – University of Salzburg Business School)

Profession

SAP Technical Consultant (Cert. SAP Development Consultant)
SAP Trainer
Project Coach (Cert. Scrum Master)
Software Architect
Software Developer (SAP AG, Walldorf, DE and Customer Development Projects)
Author (check out Amazon and/or www.citeseer.com)
Lecturer (University Salzburg, FH Salzburg)



Motivation

Ist es nur ein Hype-Thema?

Wie geht das? … im Detail …

Was machen wir? Was machen wir nicht?

Inhalte/Organisation

- Einleitung
- Technik (Python, Anaconda, …)
- Einfaches neuronales Netz (Perceptron)
- Lernen im einfachen Netz (Perceptron, Adaline)
- Mehrschichtiges neuronales Netz (MLP)
- Idee des Lernens im mehrschichtigen Netz (BackProp)
- Convolutional Neural Networks (CNN)
- CNN mit TensorFlow
- Evolution der neuronalen Netze
- Der Machine-Learning Prozess
- Lernverfahren

Einbettung

Einleitung

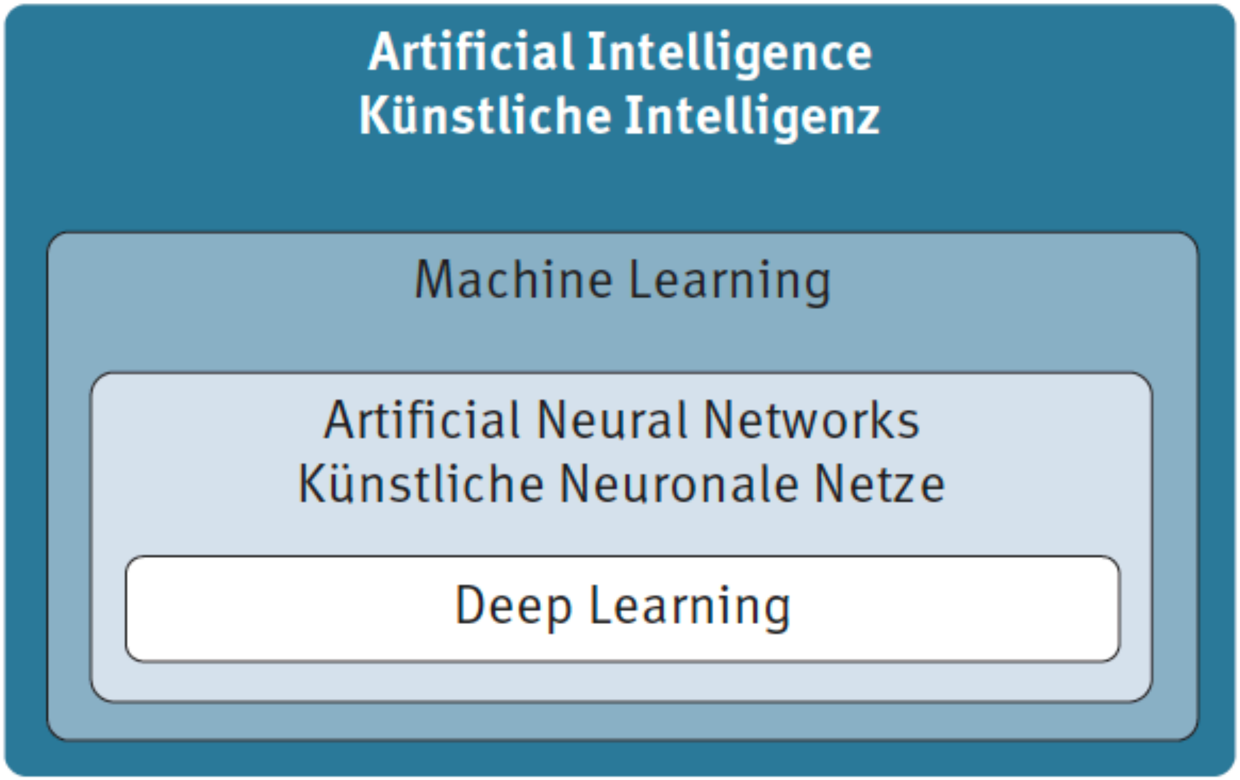


Abbildung 1.11 Begriffszwiebel für künstliche Intelligenz

Einleitung
AI

1956 John McCarthy (Dartmouth College) zusammen mit Marvin Minsky, Claude Shannon und Nathaniel Rochester organisierte eine zweimonatigen Workshop, der sich mit Themen zur künstlichen Intelligenz beschäftigen sollten, so dass jeder Aspekt des Lernens oder generell von Intelligenz so präzise beschrieben wird, dass eine Maschine gebaut werden kann, dass er simuliert werden kann.

- NLP
- Wissenspräsentation
- Logisches Schließen
- Machine Learning
- Robotik

Einleitung ML

- ML hat das Ziel Muster in Daten zu erkennen und Daten neue Erkenntnisse zu entlocken
- ML lernt aus Beispielen und kann noch nicht bekannte Beispiele verallgemeinern

-> Induktion

Einleitung ML

Induktion/Deduktion

Induktion und Deduktion sind zwei unterschiedliche Verfahren, wissenschaftliche Erkenntnisse zu gewinnen.

Die *Deduktion*geht vom Allgemeinen zum Speziellen, das heißt, von der Regel und dem Fall wird das Resultat abgeleitet.

Es folgt ein Beispiel für Freunde der österreichischen Backkunst:

- Regel: Alle Torten schmecken gut.
- Fall: Eine Sachertorte ist eine Torte.
- Resultat: Eine Sachertorte schmeckt gut.

Die *Induktion*geht vom Speziellen zum Allgemeinen, das heißt, von einzelnen Fällen und den Resultaten wird die Regel abgeleitet.

Beispiel:

- Fall: Eine Sachertorte ist eine Torte. Eine Linzertorte ist eine Torte.
- Resultat: Eine Sachertorte schmeckt gut. Eine Linzertorte schmeckt gut.
- Regel: Alle Torten schmecken gut.

Einleitung
ML

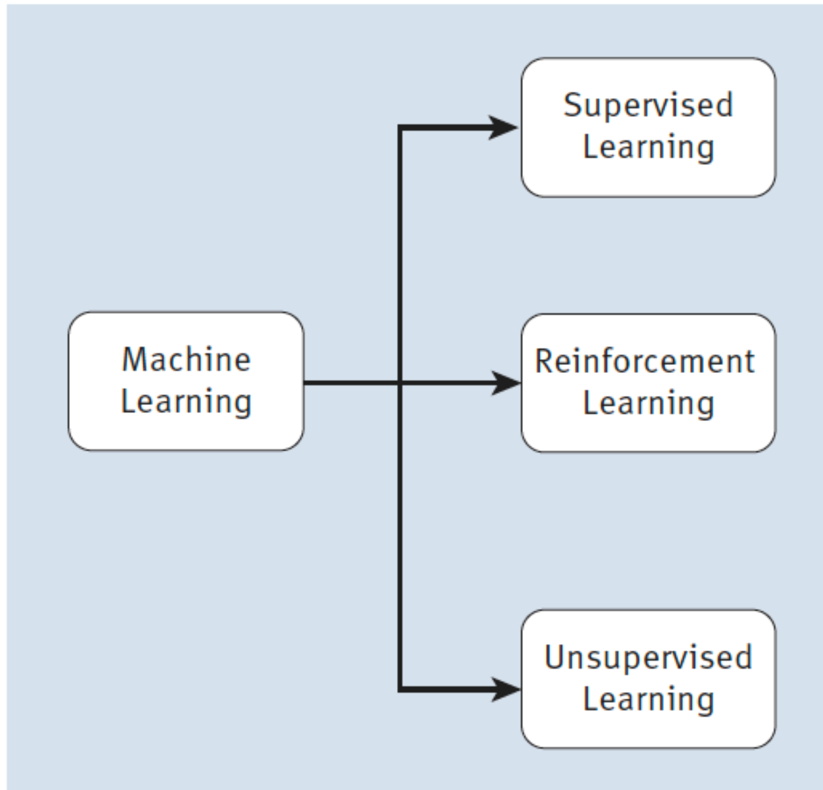
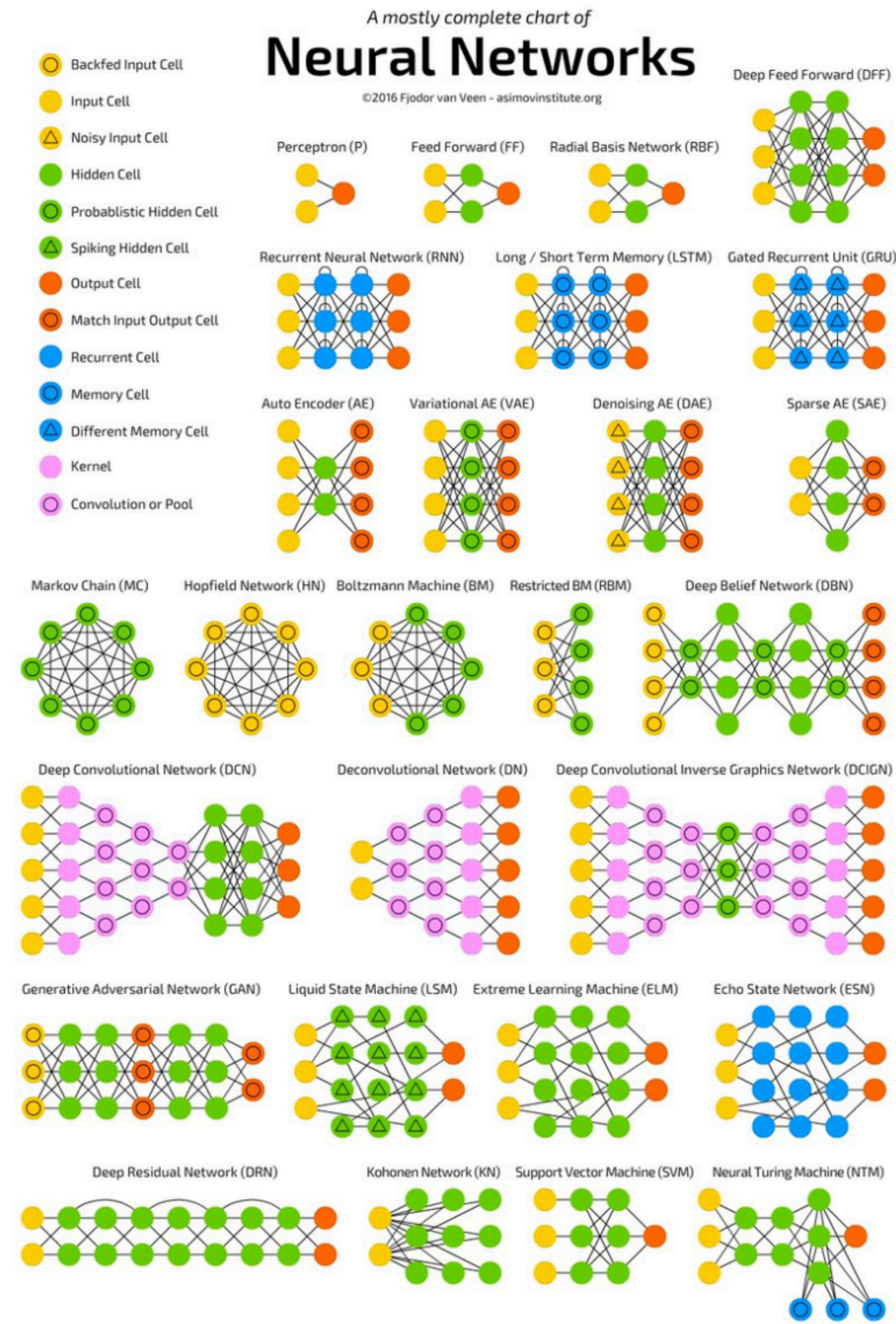


Abbildung 1.12 Lernstrategien im Machine Learning

Einleitu
KNN



NōR GmbH
empower the evolution of now

Einleitung

Einleitung
Brain

[Zum Beispiel: Visual Cortex](https://slideplayer.com/slide/1663078/)
https://slideplayer.com/slide/1663078/

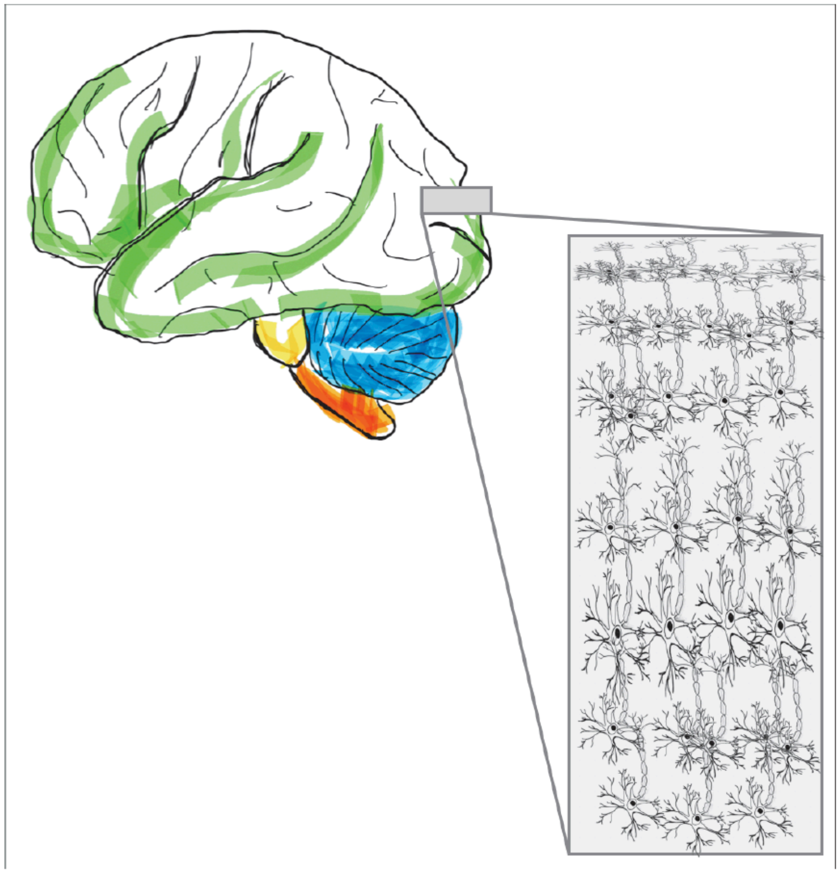
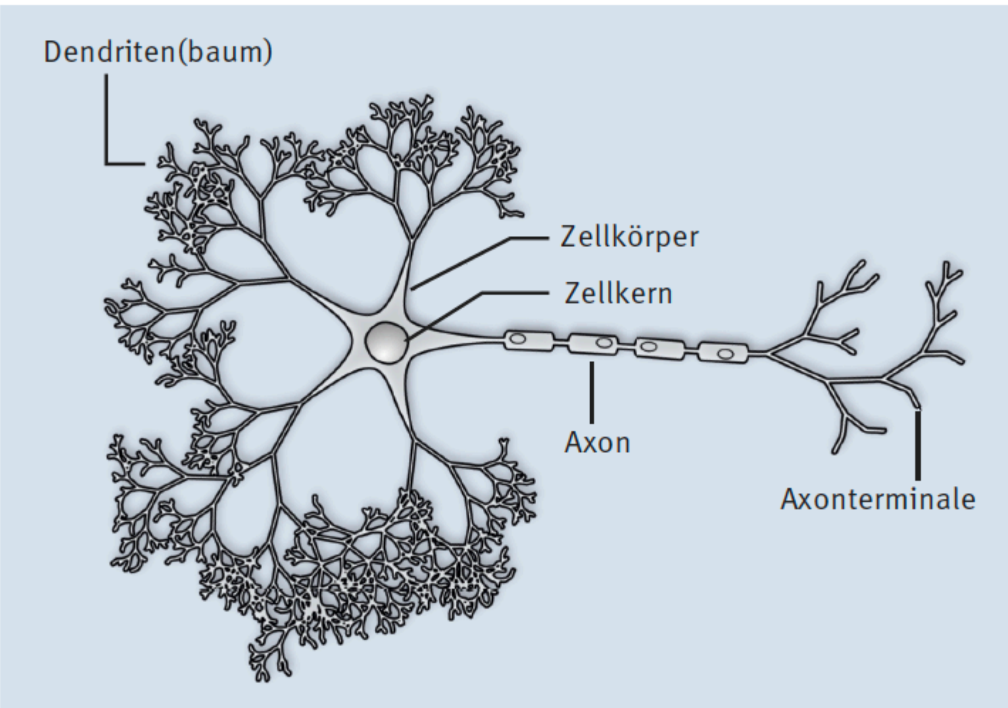


Abbildung 9.3 Eine schematische Darstellung eines Schnitts durch den Cortex

Einleitung



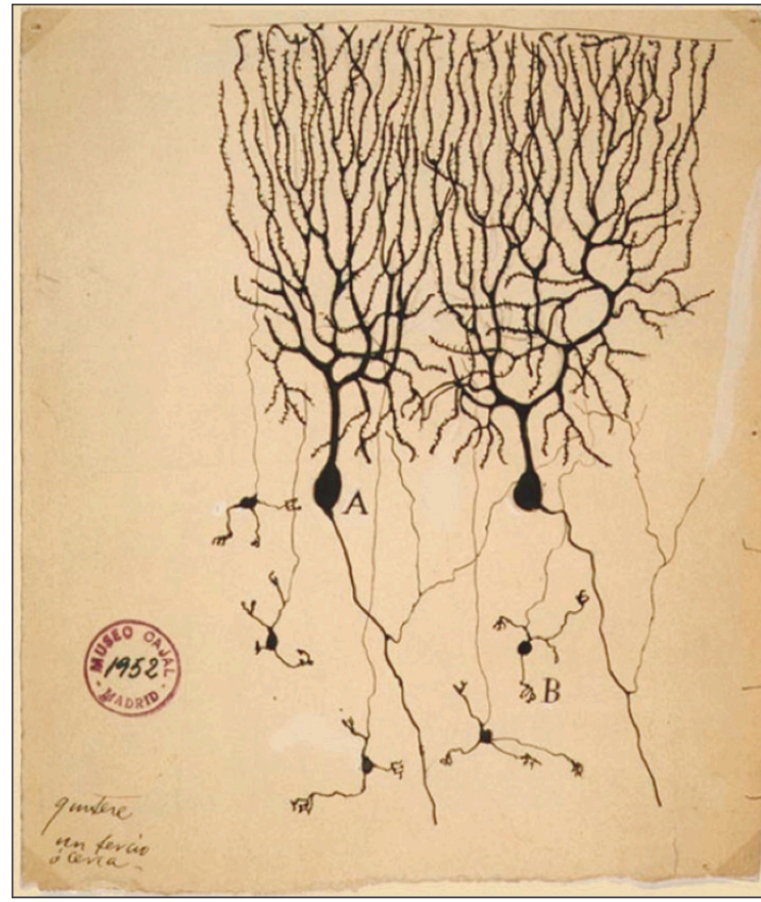


Abbildung 9.8 Die Zeichnung zweier Purkinjezellen (A) und fünf Körnerzell
Kleinhirn einer Taube von Ramón y Cajal, 1899[1]

Abbildung 1.6 Eine schematische Darstellung eines Neurons (von Nicolas.Rougier/CC-BY-SA-3.0, https://commons.wikimedia.org/w/index.php?curid=2192116)

Einleitung

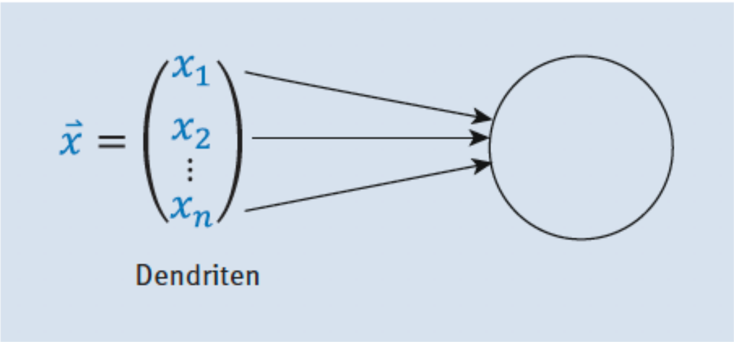


Abbildung 1.7 Künstliche Dendriten

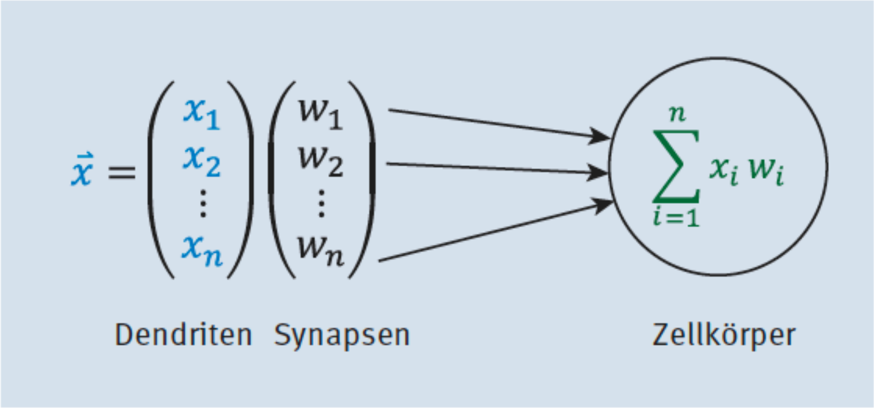


Abbildung 1.8 Künstliche Dendriten mit Synapsen(gewichten)

Einleitung

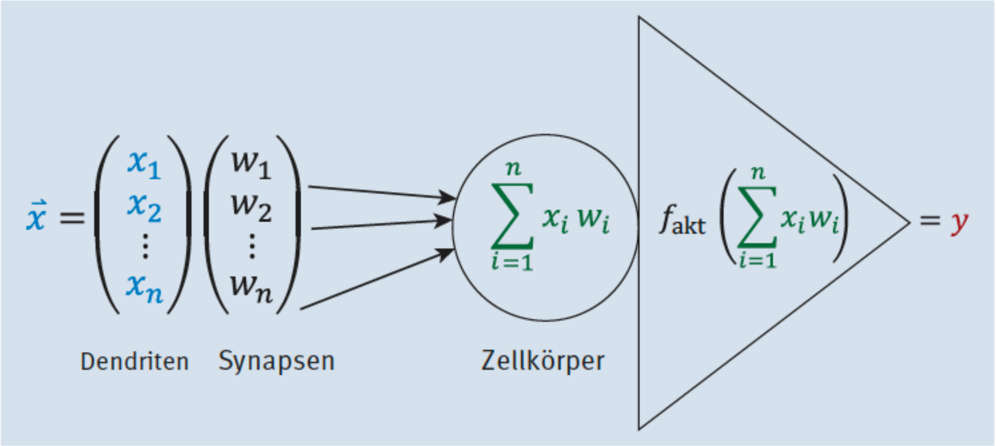


Abbildung 1.10 Das künstliche Neuron

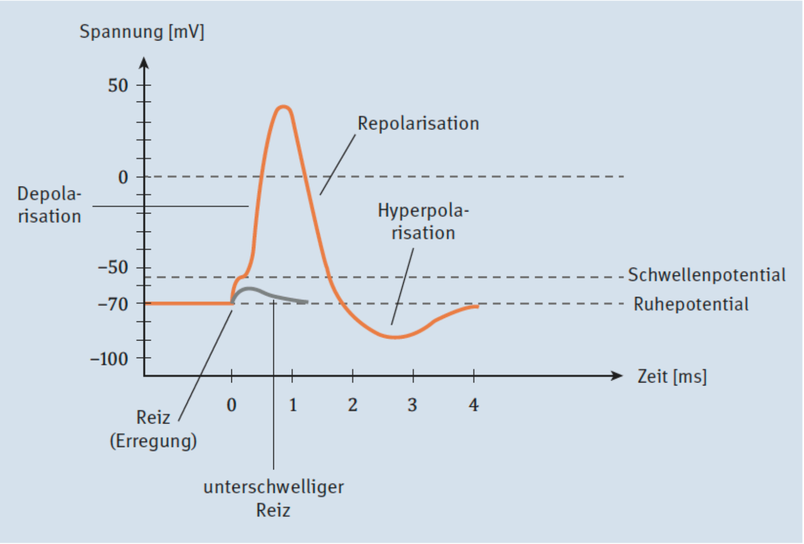


Abbildung 9.7 Aktionspotential

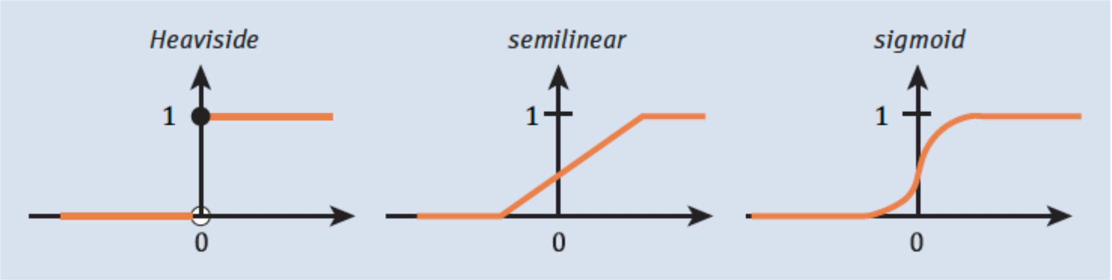


Abbildung 1.9 Beispiele für die Aktivierungsfunktion f_{akt}

Einleitung

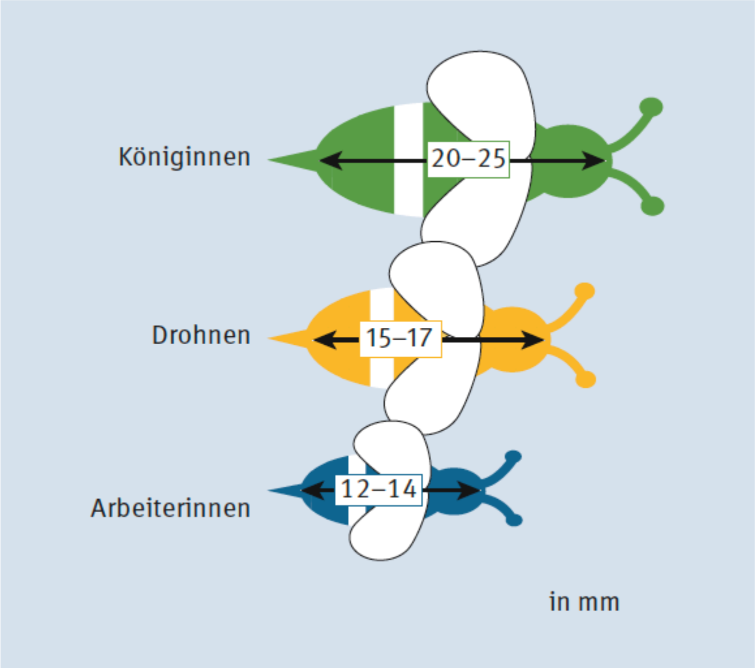


Abbildung 1.1 Die Bienenklassifikation

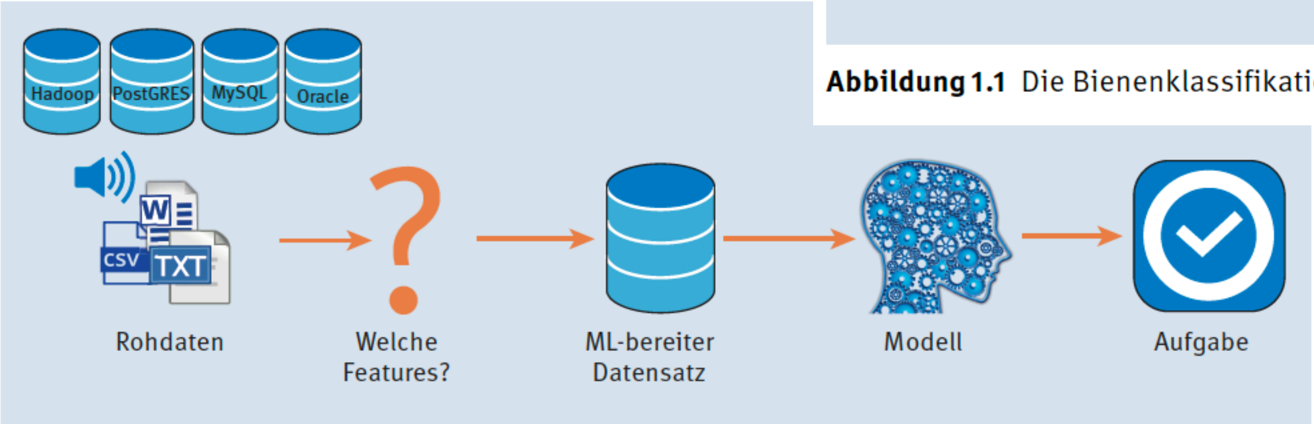
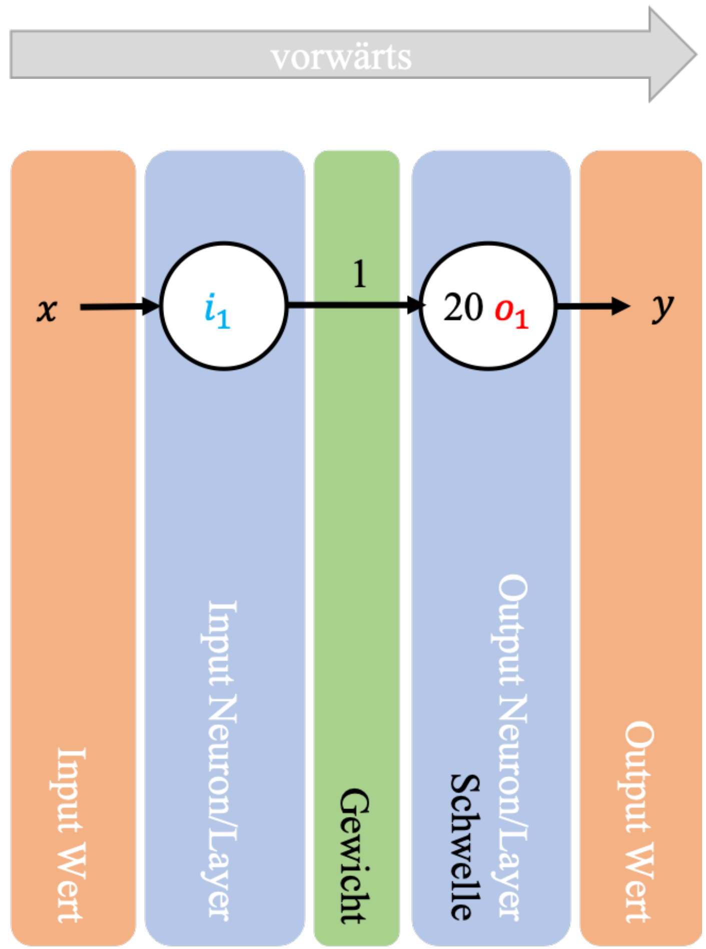


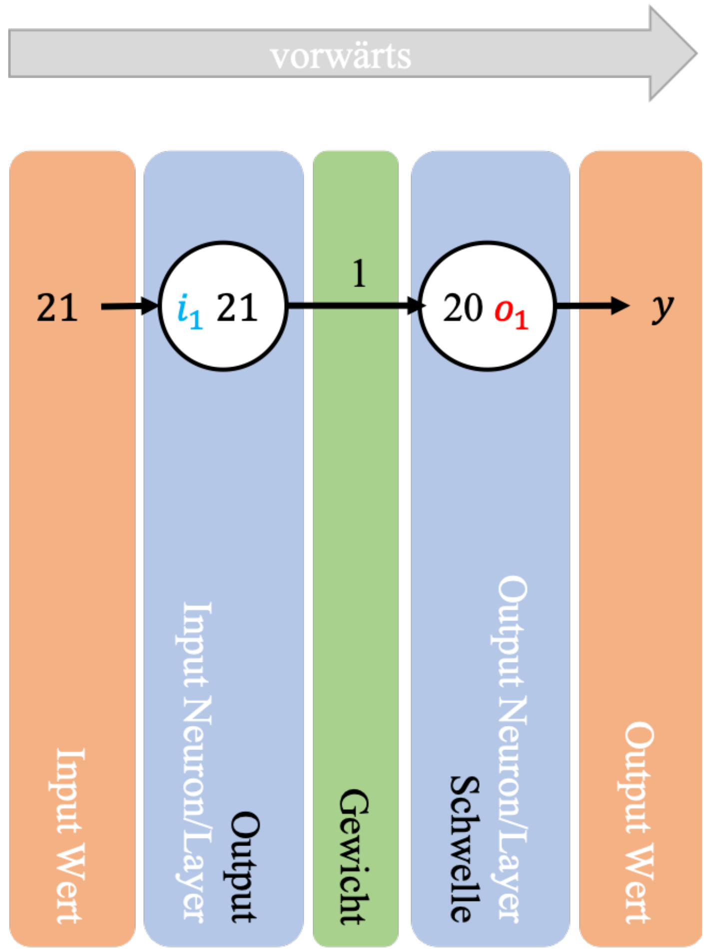
Abbildung 11.3 Realität eines Machine-Learning-Projekts

Einleitung



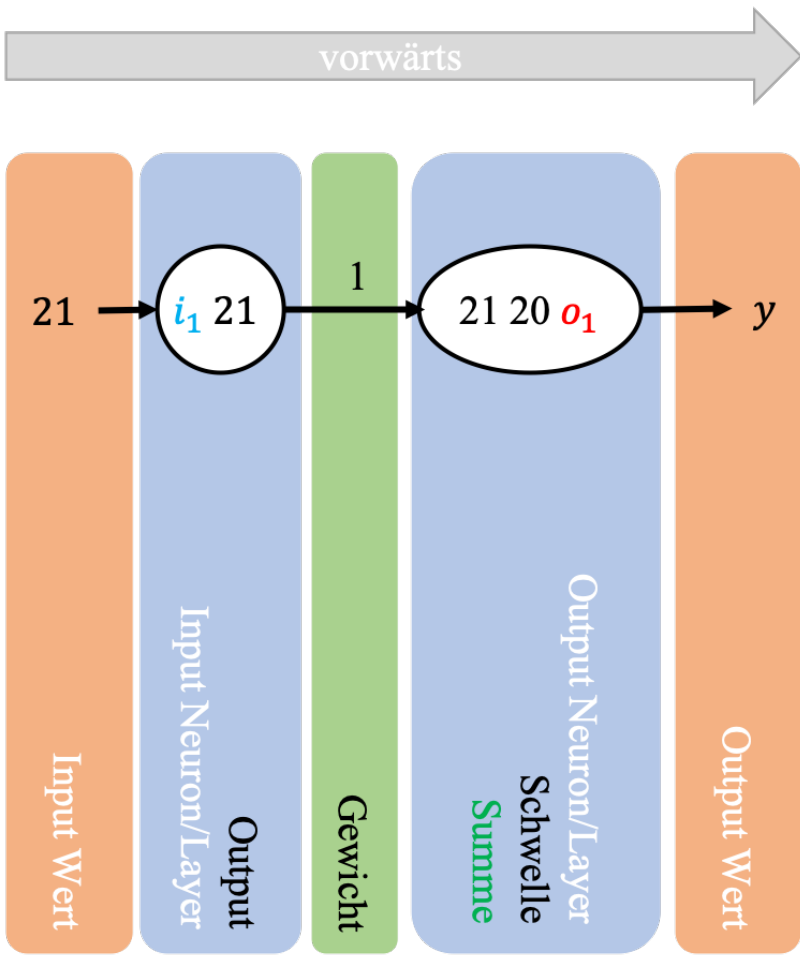
vorwärts
x
i_1
1
20 o_1
y
Input Wert
Input Neuron/Layer
Gewicht
Schwelle
Output Neuron/Layer
Output Wert

Einleitung

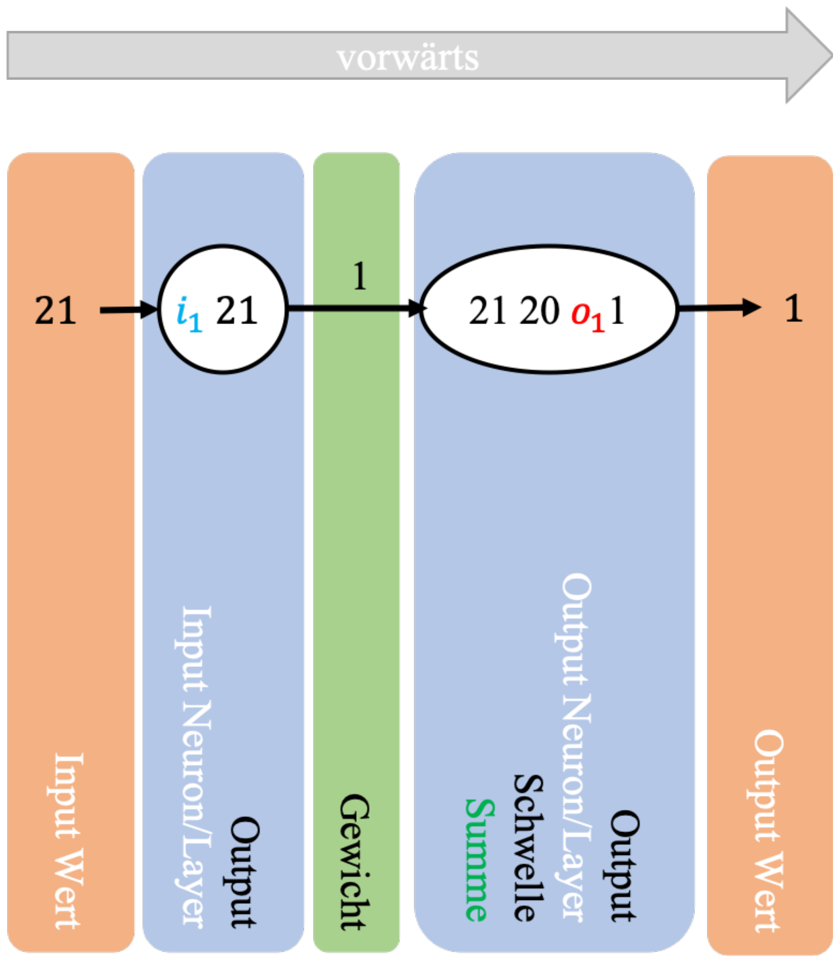

vorwärts
21
i_1 21
1
20 o_1
y
Input Wert
Input Neuron/Layer
Output
Gewicht
Output Neuron/Layer
Schwelle
Output Wert

NōR GmbH
empower the evolution of now

Einleitung



Einleitung



vorwärts
21
i_1 21
1
21 20 o_1 1
1
Input Wert
Input Neuron/Layer
Output
Gewicht
Output Neuron/Layer
Output
Schwelle
Summe
Output Wert

Einleitung

Aufgabe

Nun sind Sie an der Reihe. Probieren Sie es doch einmal mit einem Input von 14. Ist die Biene eine Königin? Was ist die Antwort des KNN?

Kapitel 2
Technik

Technik

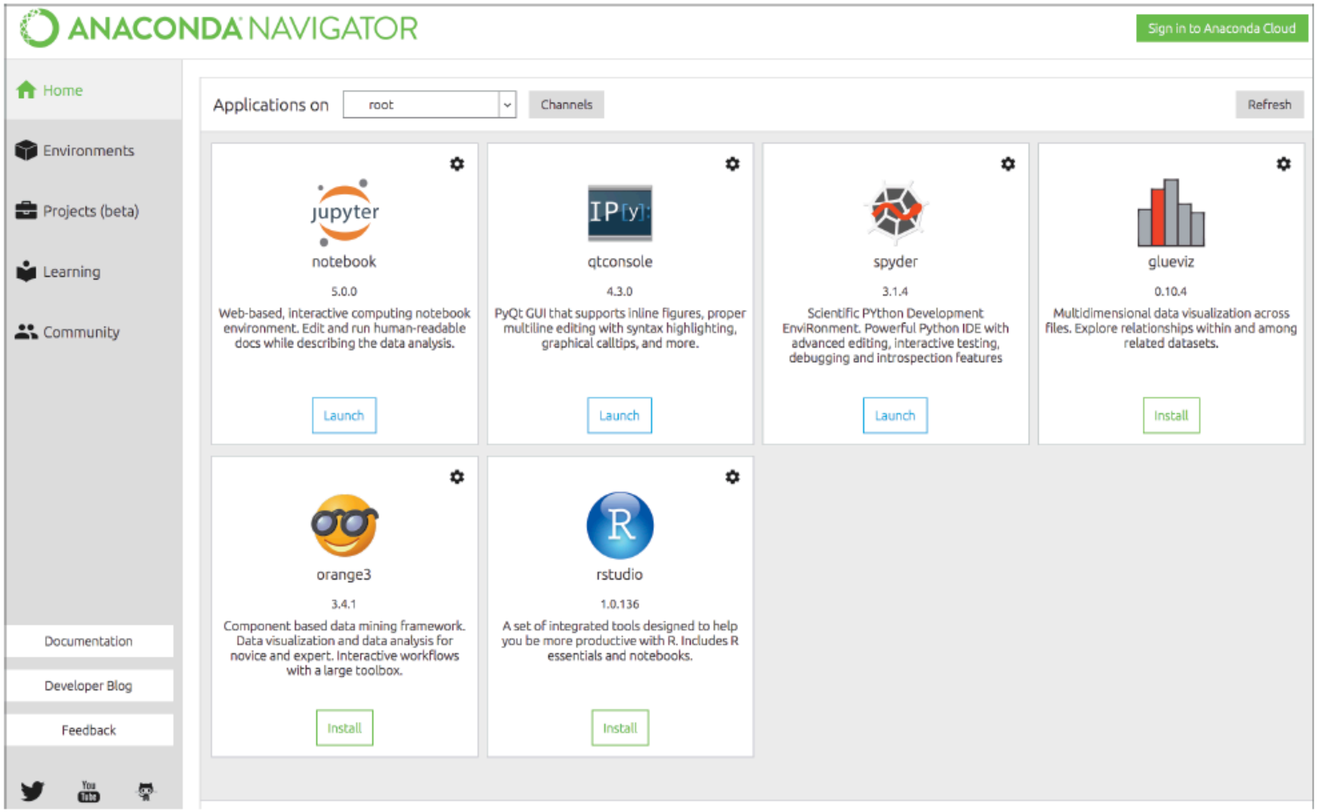


Abbildung 2.1 Der Anaconda Navigator

Technik

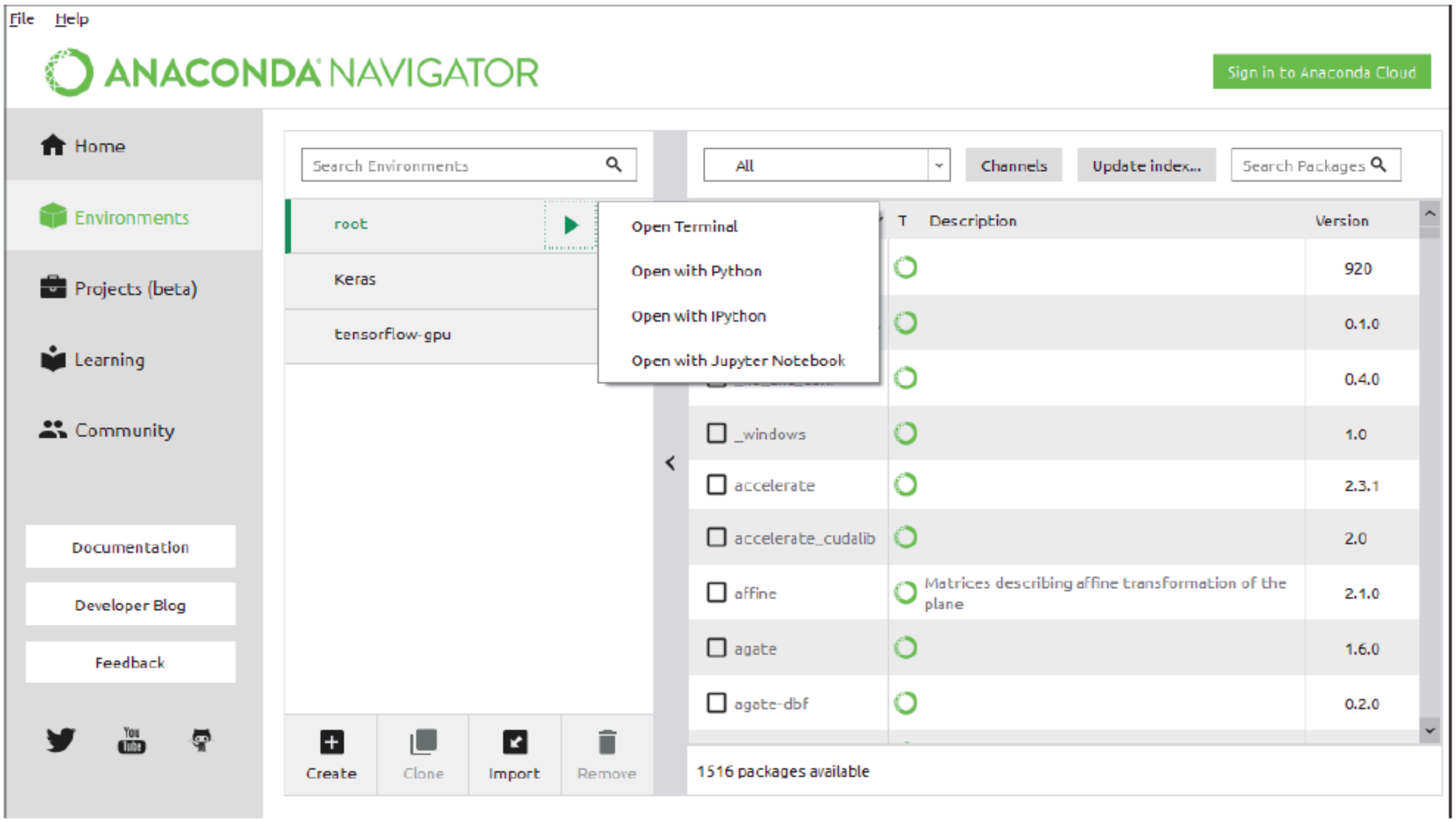


Abbildung 2.3 Das »Environment«-Fenster

Technik

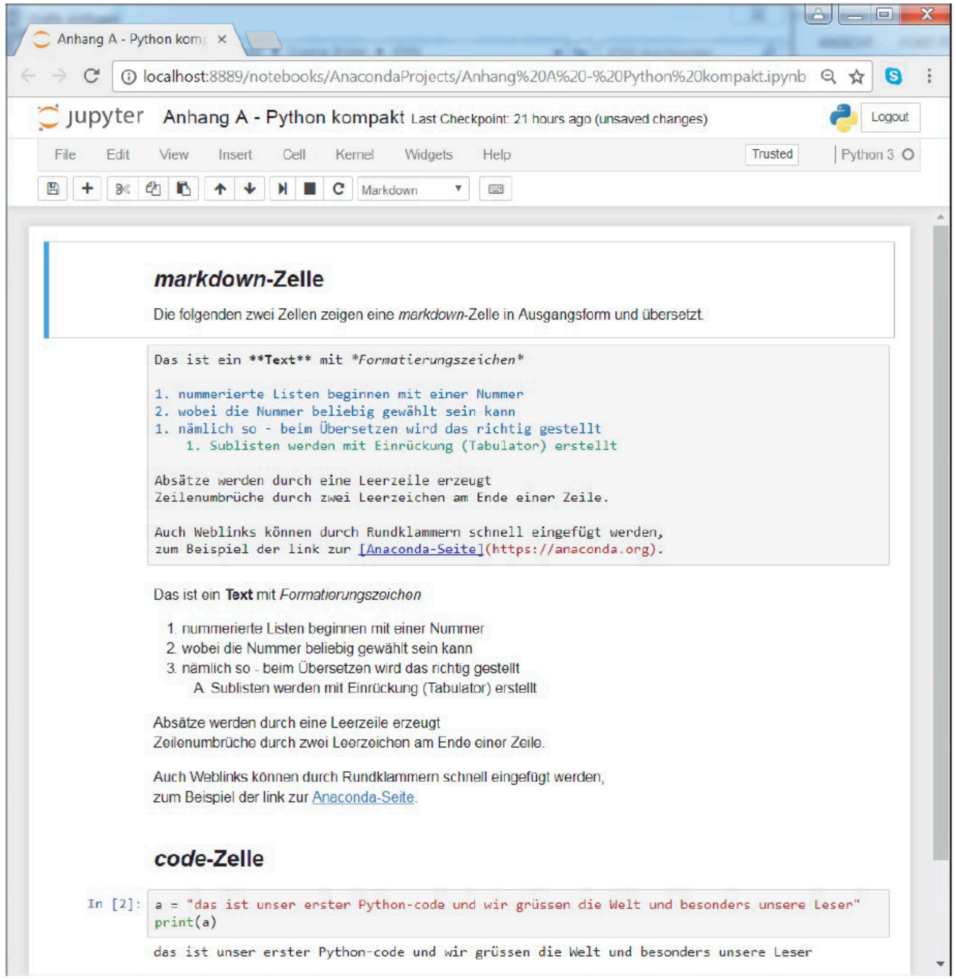


Abbildung 2.5 Markdown- und Code-Zellen in einem Jupyter-Notebook-Dokument

Technik

```
# der Docstring kann mit der Funktion help aufgerufen werden
help(print)
Out[1]:
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Listing 2.1 Docstring-Ausgabe

```
In [17]: # einfacher und schneller ist das Fragezeichen ?
         print?
```

```
Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file:  a file-like object (stream); defaults to the current sys.stdout.
sep:   string inserted between values, default a space.
end:   string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
Type:      builtin_function_or_method
```

Abbildung 2.6 Ausgabebereich im Jupyter Notebook

NōR GmbH
empower the evolution of now

Technik

```
# das Fragezeichen ? kann vielfältig eingesetzt werden
myList = ['NumPy', 'scikit', 'Keras', 'TensorFlow']
myList.append?
Out[2]:
Docstring: L.append(object) -> None -- append object to end
Type:      builtin_function_or_method

# oder einfach Information über ein Objekt
myList?
Out[3]:
Type:        list
String form: ['NumPy', 'scikit', 'Keras', 'TensorFlow']
Length:      4
Docstring:
list() -> new empty list
list(iterable) -> new list initialized from iterable's items
```

Listing 2.2 Das Fragezeichen

den Cursor auf den Funktions- oder Objektnamen und drücken Sie die Tastenkombination ⇧ + ⇆, erscheint eine Ballonhilfe, wie in Abbildung 2.7 gezeigt.

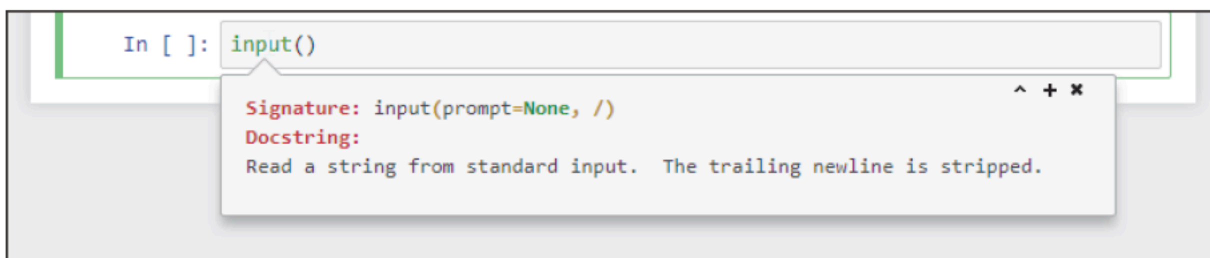


Abbildung 2.7 Kontexthilfe zur Builtin-Funktion »input()«

Technik

Debugging – Fun Fact

Unter diesem Begriff versteht man in der Welt der Programmierer das Nachverfolgen des Anwendungsablaufs. Das wird vor allem dann gemacht, wenn die Anwendung nicht das tut, was sie sollte, beziehungsweise einen Fehler (also einen *Bug*) generiert. Der Ursprung des Wortes wird oft auf den 9. September 1945 gesetzt. Man fand eine Motte (Bug) in einem Relais eines Computers. Diese Motte war verantwortlich für einen Fehler, der diesen Computer lahmlegte. Abbildung 2.10 zeigt das damalige Logbuch mit der eingelegten Motte. Allerdings wurde der Begriff schon vorher in diesem Sinne verwendet, der damalige Mitarbeiter fand es, wie wir, witzig, dass ein echter Bug zu einem Computerausfall führte.

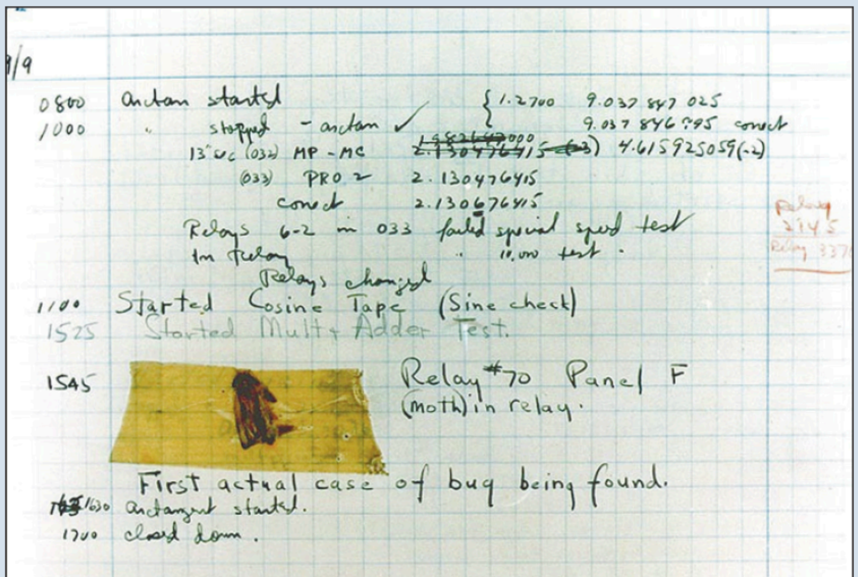


Abbildung 2.10 Der erste Bug in der Computergeschichte

Technik

Seite 60

- %xmode Steuert die Menge an Fehlinformationen
  - %xmode Context # Standard
  - %xmode Plain # Kompakt
  - %xmode Verbose # Ausführlich
- %debug Interaktiven Debugger im Kommandozeilenfenster starten (Prompt ipdb>), um z.B. Variableninhalte anzuzeigen
  - quit beendet

Technik
Python Module

- NumPy - Zum Rechnen
- Matplotlib - Zum Zeichnen
- scikit-learn - Zum ML
- Pandas - Zum Datenbearbeiten
- TensorFlow - Googles DeepLearning
- Keras - Ein Layer über ML Frameworks

Technik
Python Frameworks 01.2019

Die Frameworks im Überblick

Framework	Lizenz	Programmiersprache	Betriebssysteme	Vorteile	Nachteile
TensorFlow	Apache 2.0	Python, C/C++, Java, Go, R, Swift, JavaScript	Linux, macOS, Windows, Android	große Community, Bindings für viele Programmiersprachen, Long-Term Support	Einstieg im Vergleich schwerer
Keras	MIT-Lizenz	Python, R	Linux, macOS, Windows	schnell zu erlernen, einfache Handhabung, mobile Betriebssysteme	in einigen Fällen langsamer, kein Einblick „unter die Haube"
Microsoft Cognitive Toolkit	MIT-Lizenz	Python, C++, C#/.NET, Java, BrainScript	Windows, Linux, macOS (über Docker-Container)	nativ lauffähig in der Azure-Cloud, Support für Apache Spark	kleine Community
Torch	BSD-Lizenz	Lua, C	Linux, macOS, Windows, Android, iOS	Schnittstelle zu C über Lua, auf mobilen Betriebssystemen implementierbar	Verwendung von Lua
PyTorch	BSD-Lizenz	Python	Linux, macOS, Windows	Long-Term Support, einsetzbar in Cloud-Umgebungen, hohe Flexibilität	keine stabile Version
Caffe/Caffe2	BSD-Lizenz	Python, MATLAB, C++	Linux, macOS, Windows (Community-Support)	MATLAB-Interface, CaffeOnSpark	Caffe2 jetzt Teil von PyTorch, wenig Input-Formate, nur ein Output-Format
Theano	BSD-Lizenz	Python	Linux, macOS, Windows	Viele Feature-Requests sind umgesetzt.	Support und Weiterentwicklung eingestellt

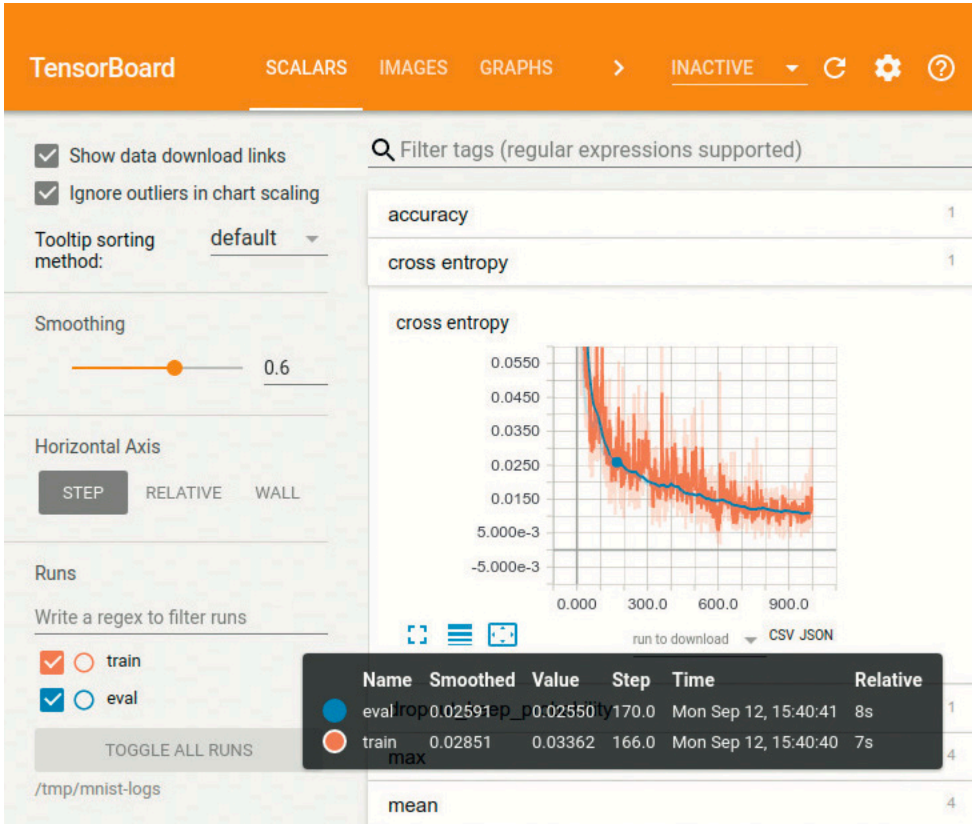
© Heise https://www.heise.de/select/ix/2019/1/1545999823788057

Technik TensorFlow

- Google Open-Source-Projekt
- TensorFlow Graph: Mathematische Operationen
  - Knoten sind Operatoren
  - Daten werden in Tensoren gespeichert

© Heise https://www.heise.de/select/ix/2019/1/1545999823788057

Technik TensorFlow



© Heise https://www.heise.de/select/ix/2019/1/1545999823788057

Technik
Keras

- Aus 2015 von Google Entwickler François Chollet
- Fokussiert auf einfache Bedienbarkeit
- Setzt auf TensorFlow oder Theano auf
- Ist in TensorFlow integriert
- Stellt Interface zwischen TensorFlow, Theano und MS Cognitive Toolkit dar

© Heise https://www.heise.de/select/ix/2019/1/1545999823788057

Technik
MS Cognitive Toolkit (CNTK)

- Deep-Learning-Framework
- Erste Beta 2016
- Als Backend für Keras
- Auf GPUs und Rechner verteilt ausführbar
- Modelle lassen sich in Azure-Cloud trainieren
- Funktionen als gerichtete Graphen
  - Knoten = Eingabewert, Parameter
  - Kanten = Matrix oder Operation

© Heise https://www.heise.de/select/ix/2019/1/1545999823788057

Technik
Torch

- Das Framework basiert auf der Skriptsprache Lua und auf **CUDA**, der von NVIDIA bereitgestellten Compute Unified Device Architecture.
- Durch die CUDA-Unterstützung lassen sich Modelle nicht nur auf CPUs, sondern vor allem auf GPUs schnell und performant trainieren.

© Heise https://www.heise.de/select/ix/2019/1/1545999823788057

Technik CUDA

CUDA (früher auch **Compute Unified Device Architecture** genannt) ist eine von Nvidia entwickelte Programmier-Technik, mit der Programmteile durch den **Grafikprozessor (GPU)** abgearbeitet werden können.

In Form der GPU wird zusätzliche Rechenkapazität bereitgestellt, wobei die GPU im Allgemeinen bei hochgradig parallelisierbaren Programmabläufen (hohe Datenparallelität) signifikant schneller arbeitet als die CPU. CUDA wird vor allem bei wissenschaftlichen und technischen Berechnungen eingesetzt.

https://www.nvidia.de/object/cuda-parallel-computing-de.html

Technik PyTorch

NōR GmbH
empower the evolution of now

- PyTorch gilt in der Community als Ersatz für die Python-Bibliothek NumPy, wenn Berechnungen GPU-Unterstützung brauchen. Das ermöglichen Tensoren, die in PyTorch multidimensionale Arrays abbilden, ähnlich wie in NumPy.

© Heise https://www.heise.de/select/ix/2019/1/1545999823788057

Technik Caffe/2

NōR GmbH
empower the evolution of now

- Das Deep-Learning-Framework Caffe entstand im Jahr 2014 am Vision and Learning Center der Berkeley University. Als Programmierschnittstellen sind Python und MATLAB vorgesehen. Neben der Unterstützung von CPUs und GPUs ist die Verwendung mit CUDA möglich.

© Heise https://www.heise.de/select/ix/2019/1/1545999823788057

Kapitel 3
Ein einfaches Netz

Einfaches Netz

Frau Karotte	Herr Lauch	→	Montag
abwesend	abwesend	→	ko
anwesend	abwesend	→	ok
abwesend	anwesend	→	ok
anwesend	anwesend	→	ok

Tabelle 3.1 Erster Personalplan

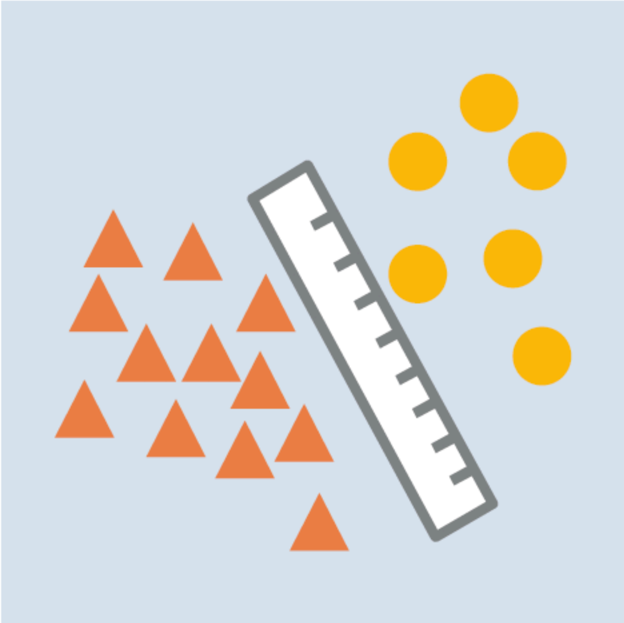


Abbildung 3.1 Linea(l/r)e Trennung

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

Einfaches Netz

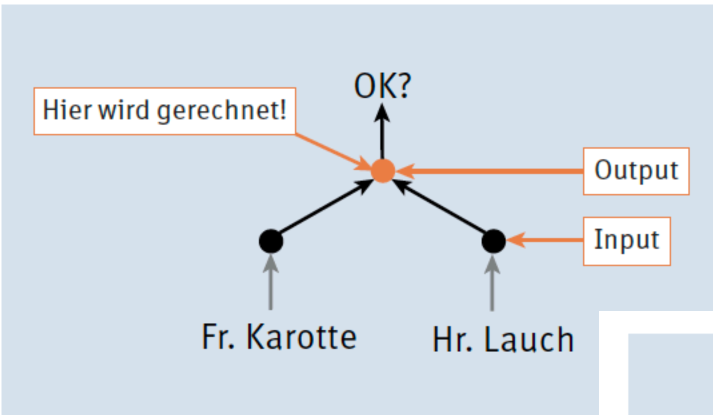


Abbildung 3.2 Die Entscheidung für Frau Ka

$$Ergebnis = Entscheidung(Summe)$$

wobei

$$Entscheidung = \begin{cases} 1, & \text{falls } Summe \geq 1 \\ 0, & \text{falls } Summe < 1 \end{cases}$$

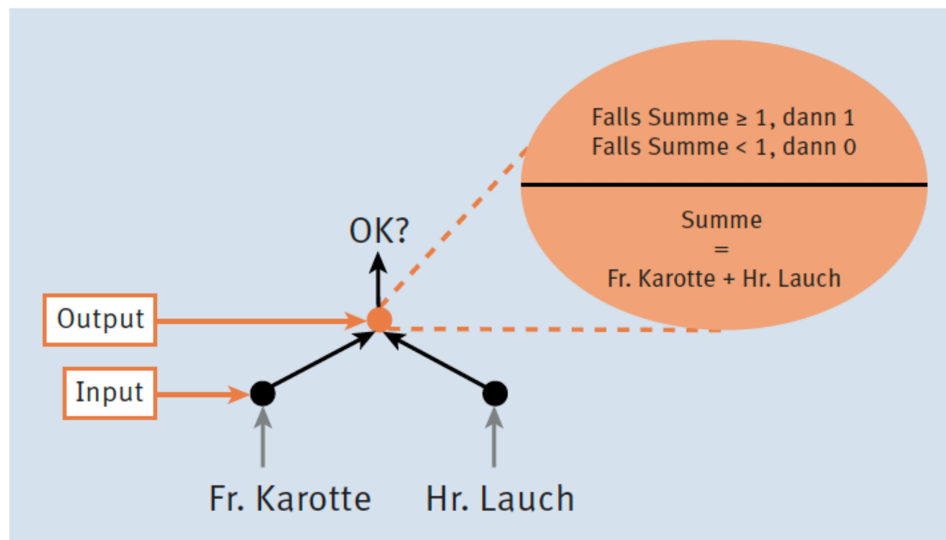


Abbildung 3.3 Ergebnisberechnung für Frau Karotte und Herrn Lauch im Detail

Einfaches Netz

Kapitel 3

```
def entscheidung( summe ):
    """Berechnung der Entscheidung zum Wert summe
        Input:  summe
        Output: 1, falls summe >= 1,
                0 sonst
    """

    if summe >= 1:
        return 1
    else:
        return 0
#-----------------------------
# Berechnung der entscheidung
ergebnis = entscheidung(1)
# Ausgabe in Zelle
print(ergebnis)
# Die Ausgabe
1
```

Listing 3.1 Die Entscheidung als Stufenfunktion

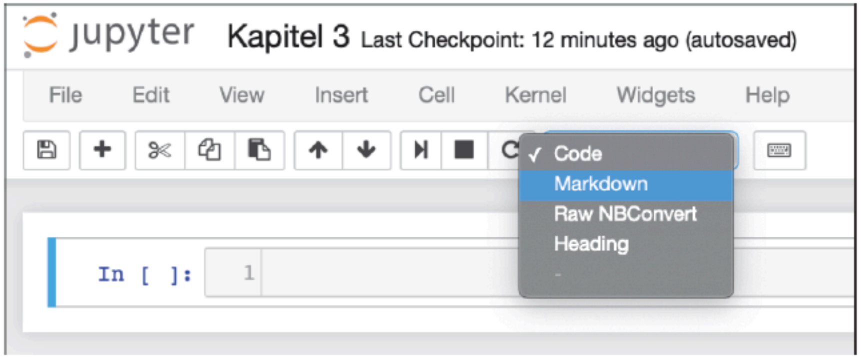


Abbildung 3.5 Eine Zelle mit »Markdown« markieren

NōR GmbH
empower the evolution of now

Einfaches Netz

Kapitel 3

jupyter Kapitel 3 Last Checkpoint: 19 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

Code

Kapitel 3

Die Entscheidung

In [26]:

```
def entscheidung( summe ):
    """Berechnung der Entscheidung zum Wert summe
        Input:  summe
        Output: 1, falls summe >= 1,
                0 sonst
    """
    if summe >= 1:
        return 1
    else:
        return 0
#------------------------------
# Berechnung der entscheidung
ergebnis = entscheidung(1)
# Ausgabe in Zelle
print(ergebnis)

```

```
1
```

Dokumentationsstring

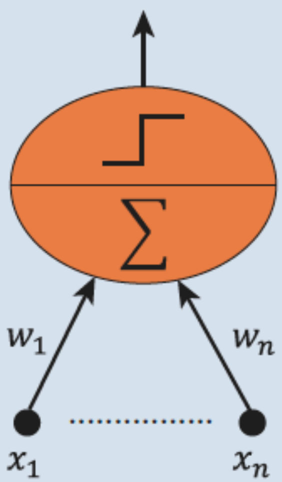
In [30]:

```
print(entscheidung.__doc__)
```

```
Berechnung der Entscheidung zum Wert summe
        Input:  summe
        Output: 1, falls summe >= 1,
                0 sonst
```

Abbildung 3.7 Programm und Ausgaben für die Funktion »entscheidung«

Einfaches Netz



Output

5 $f_{\text{akt}}(\vec{x}) = step(\vec{w}^{t} \cdot \vec{x})$

Stufenfunktion

4 $step(s)$

gewichtete Summe

$$s = x_1 \cdot w_1 + \cdots + x_n \cdot w_n = \sum_{i=1}^{n} x_i \cdot w_i = \vec{w}^{t} \cdot \vec{x}$$

3

Gewicht

$$\vec{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix} \quad \vec{w}^{t} = (w_1 \ldots w_n)$$

2

Input

1 $\vec{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$

Abbildung 3.9 Perceptron – Bestandteile und Berechnungsbausteine

Einfaches Netz

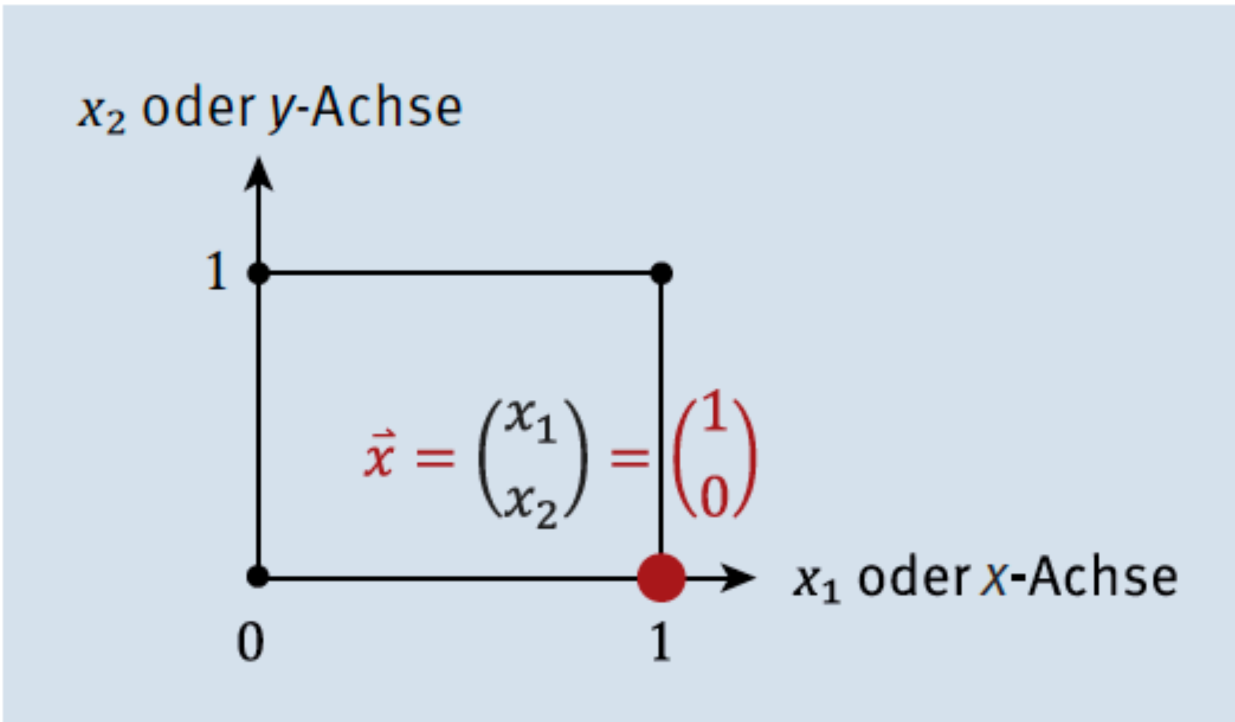


Abbildung 3.10 Vektor im kartesischen Koordinatensystem

Einfaches Netz Iris

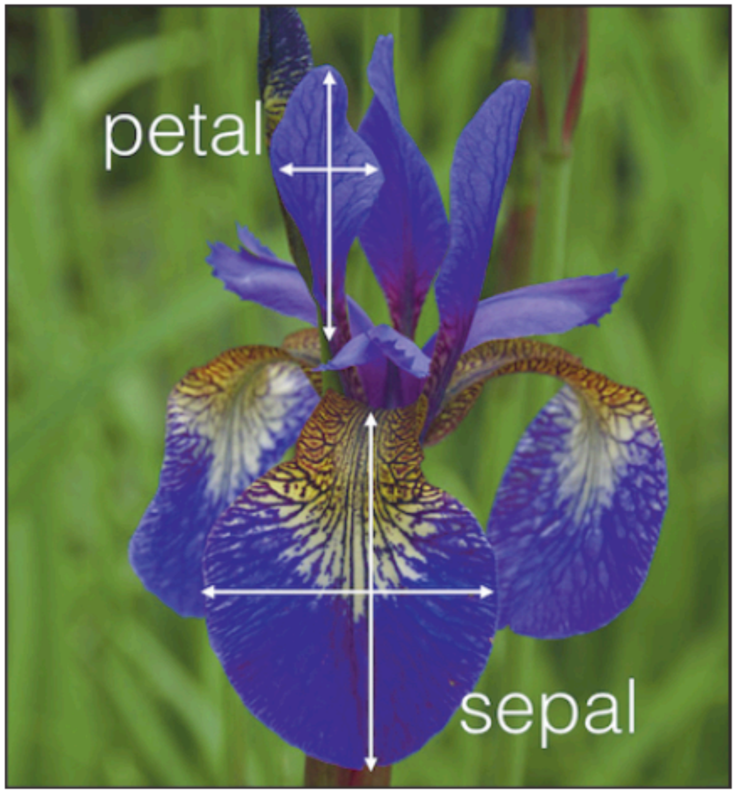


Abbildung 3.13 Blattmaße für Schwertlilien © Kaggle

Einfaches Netz

Kapitel 3

Lösung:

```
# Die benötigten Module importieren
import matplotlib.pyplot as plt
import numpy as np
# Ganz wichtig, sonst wird der Plot nicht angezeigt
%matplotlib inline

# File einlesen
fobj = open("iris.csv", "r")
# x1 sind die Koordinaten der x-Achse, x2 die der y-Achse
x1    = []
x2    = []
# Farben für die Datenpunkte
colors = []
# Mapping der Schwertlilien zu Farben mit
# einem Python-Dictionary
iris_colors = { 'Iris-setosa' : 'red',
            'Iris-versicolor' : 'green',
            'Iris-virginica' : 'blue'}
# Den Datensatz zeilenweise verarbeiten
for line in fobj:
#   Split
    words = line.rstrip().split(",")
#   Leerzeilen auslassen
    if len(words) != 5:
        continue
#   SepalLength
    x1.append(words[0])
#   SepalWidth
    x2.append(words[1])
#   Farbe
    colors.append(iris_colors[words[4]])
# Schließen der File Handle
fobj.close()
#-----------------------------------
# Gitter im Scatter-Plot zeichnen
plt.style.use('seaborn-whitegrid')
# Achsenbeschriftung und Titel
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('Scatter-Plot')
# Den Plot ausgeben
plt.scatter(np.array(x1), np.array(x2), color=colors )
```

Listing 3.4 Sepal Length und Sepal Width als Scatter-Plot

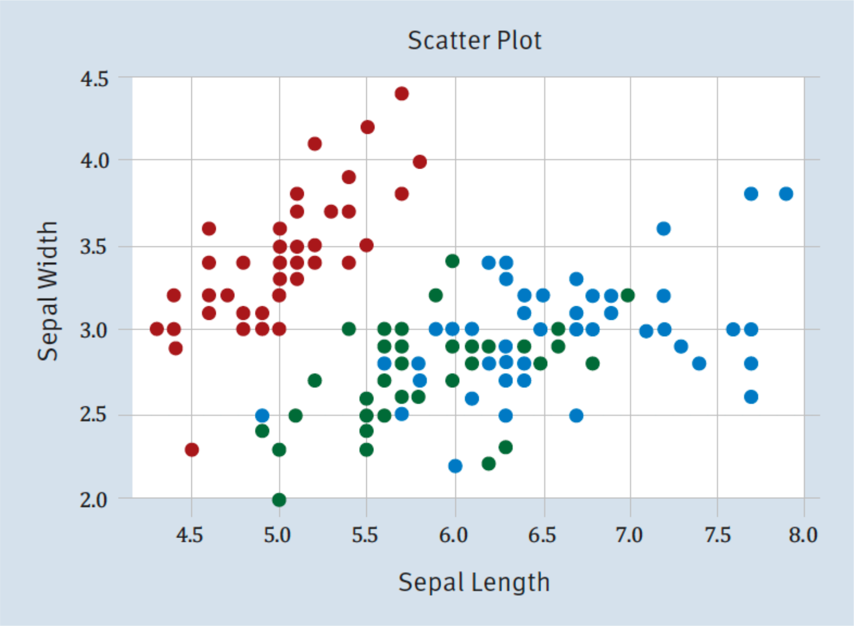
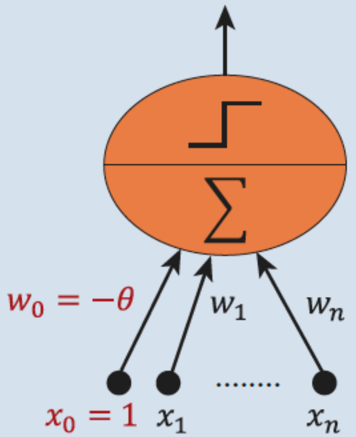


Abbildung 3.14 Die Lilien als Scatter-Plot mit den Koordinaten Sepal Length und Sepal Width

Einfaches Netz



Output

5 $f_{\text{akt}}(\vec{x}) = step(\vec{w}^t \cdot \vec{x})$

Stufenfunktion

4 $step(s)$

gewichtete Summe

$$s = 1 \cdot w_0 + x_1 \cdot w_1 + \cdots + x_n \cdot w_n = \sum_{i=0}^{n} x_i \cdot \underbrace{w_i}_{3} = \vec{w}^t \cdot \vec{x}$$

Gewicht

$$\vec{w} = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{pmatrix} \quad \vec{w}^t = (w_0 \quad w_1 \quad \ldots \quad \underbrace{w_n}_{2})$$

Input

$$\textcircled{1}\ \vec{x} = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{pmatrix}$$

Abbildung 3.15 Bausteine des Perceptrons

Einfaches Netz


```
# Perceptron Forward Path
import matplotlib.pyplot as plt
import numpy as np
# 3-dimensionaler Input = Bias-Neuron, Fr. Karotte, Hr. Lauch
# 4 Inputvektoren
X = np.array([
    [1,0,0],
    [1,0,1],
    [1,1,0],
    [1,1,1]])
# Die 4 gewünschten Ergebniswerte
y = np.array([0,1,1,1])
# Heaviside-Funktion
def heaviside( summe ):
    """Berechnung der Entscheidung zum Wert summe
        Input:  summe
        Output: 1, falls summe >= 0,
                0 sonst
    """
    if summe >= 0:
        return 1
    else:
        return 0
# Perceptron-Berechnung (Forward Path)
def perceptron_eval(X,y):
#   Der Gesamtfehler
    gesamtfehler = 0;
#   Die Gewichte so wählen, dass das OR-Problem gelöst werden kann
    w = np.array([-1,1,1])
#   Index i und Element-x-Ermittlung vom Array X
    for i, x in enumerate(X):
#   x = Zeile für Zeile verwenden
#   Inneres Produkt zwischen x und w
        summe = np.dot(w,x)
        ergebnis = heaviside(summe)
#       Fehler
        fehler = np.abs(ergebnis - y[i])
#.      Gesamtfehler
        gesamtfehler += fehler
#       Ausgabe
        print("Fr. Karotte = {}, Hr. Lauch = {}, Ergebnis = {}, Fehler = {}".
              format(x[1], x[2], y[i],fehler))
#   Gesamtfehler pro Epoche über ganzen Trainingsdatensatz
    return gesamtfehler
#---------------------------------------
# Core Function zum Auswerten des Inputs
gesamtfehler = perceptron_eval(X,y)
print("Gesamtfehler = %1d" % (gesamtfehler))
```

```
# Ausgabe
Fr. Karotte = 0, Hr. Lauch = 0, Ergebnis = 0, Fehler = 0
Fr. Karotte = 0, Hr. Lauch = 1, Ergebnis = 1, Fehler = 0
Fr. Karotte = 1, Hr. Lauch = 0, Ergebnis = 1, Fehler = 0
Fr. Karotte = 1, Hr. Lauch = 1, Ergebnis = 1, Fehler = 0
Gesamtfehler = 0
```

Listing 3.6 Lösung für das sehr einfache Personalplanungsproblem

Einfaches Netz

Kapitel 3

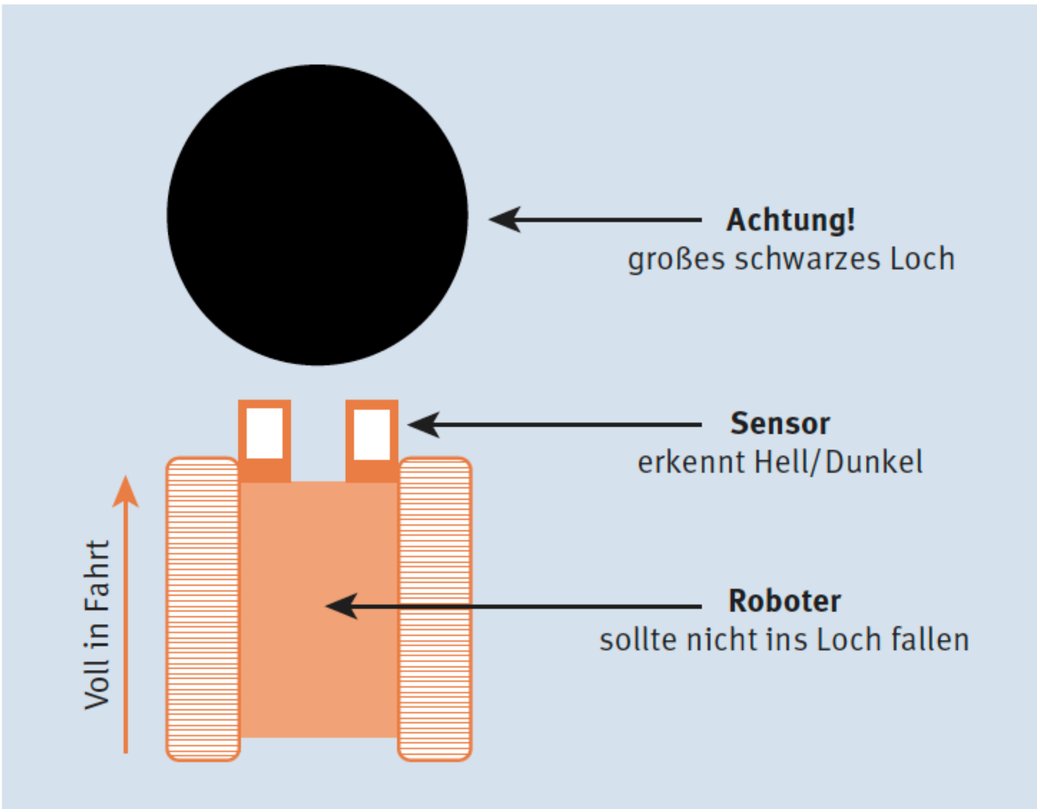
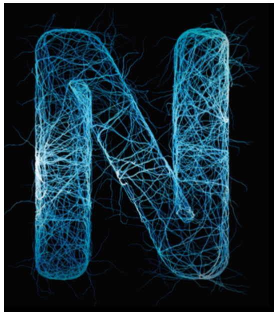


Abbildung 3.16 Einfache Robotersensorik

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

Tabelle 3.5 Locherkennung mit Hilfe der Sensorwerte



Kapitel 4

Lernen im einfachen Netz

Lernen im einfachen Netz

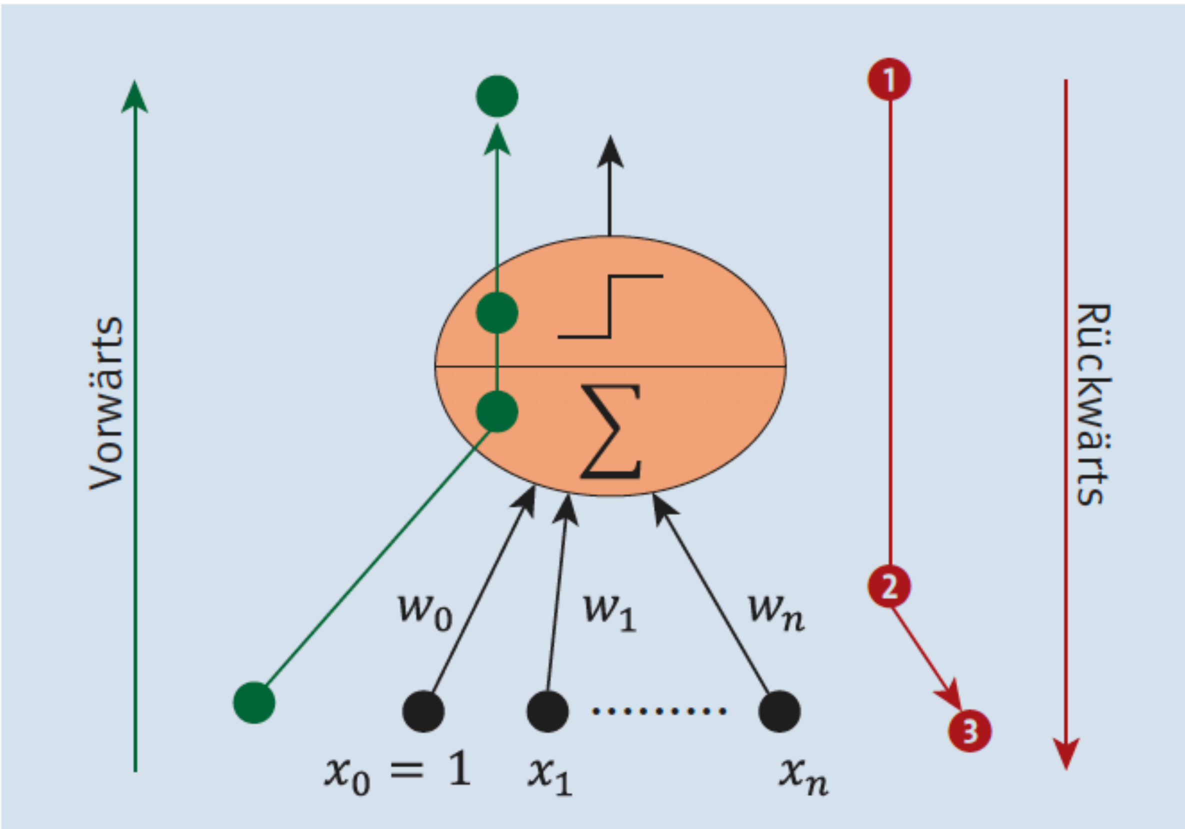


Abbildung 4.1 Vorwärts und rückwärts im Perceptron

1) Ermittle Fehler
2) Ändere Gewicht
3) Input berücksichtigen

Lernen im einfachen Netz

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

wobei

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

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

w_i^{neu} ist das Gewicht nach einer Änderung dels alten Gewichts, also falls zum Beispiel 0.1 zum alten Gewicht dazu gezählt wird. Das wäre dann bei unserem Beispiel –0.3 + 0.1 = –0.2.

Δw_i wird als »Delta w_i« gesprochen. Das kleine Dreieck wird als Delta bezeichnet und soll ausdrücken, dass eine Änderung stattfindet, umgangssprachlich mit »ein wenig« oder auch mit »ein bisserl« bezeichnet, je nachdem, aus welcher deutschsprachigen Ecke man kommt. Das bedeutet somit, dass mit jedem Schritt jedes Gewicht verändert wird, und das höchstens um –1 oder +1.

Lernen im einfachen Netz

$\hat{y}$ ... *der errechnete Wert*

y ... *der gewünschte Wert*

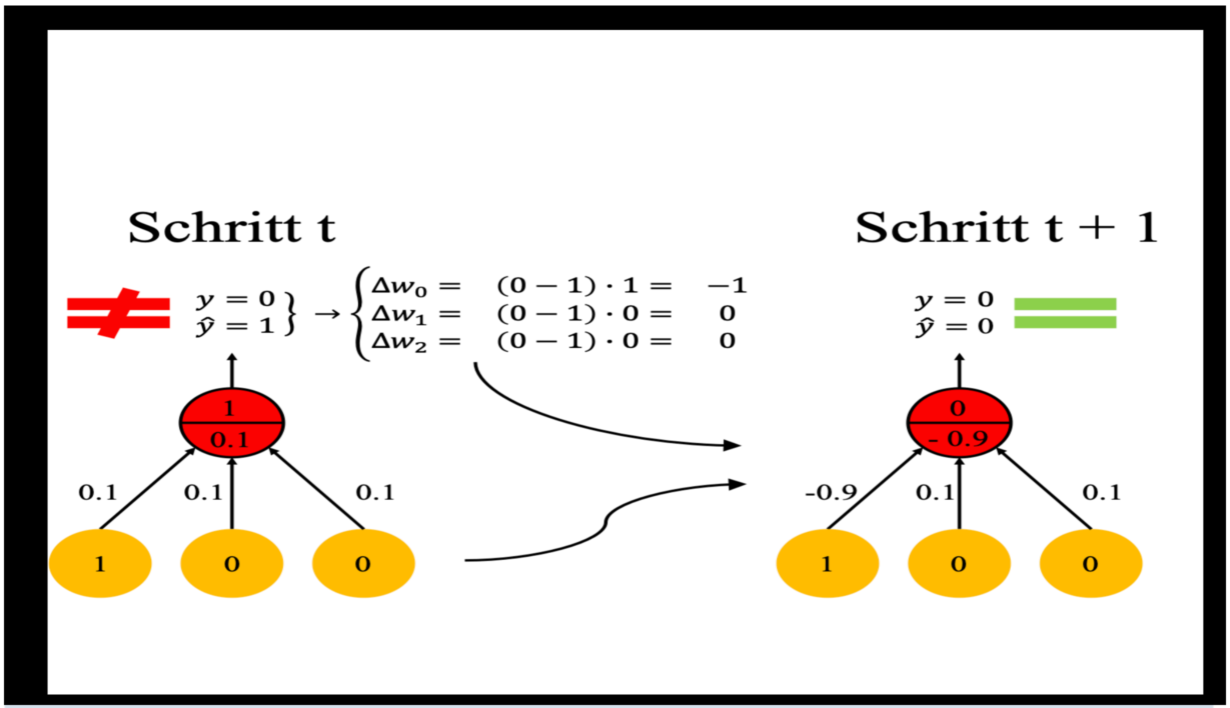


Abbildung 4.2 Ein Perceptron-Lernschritt

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

wobei

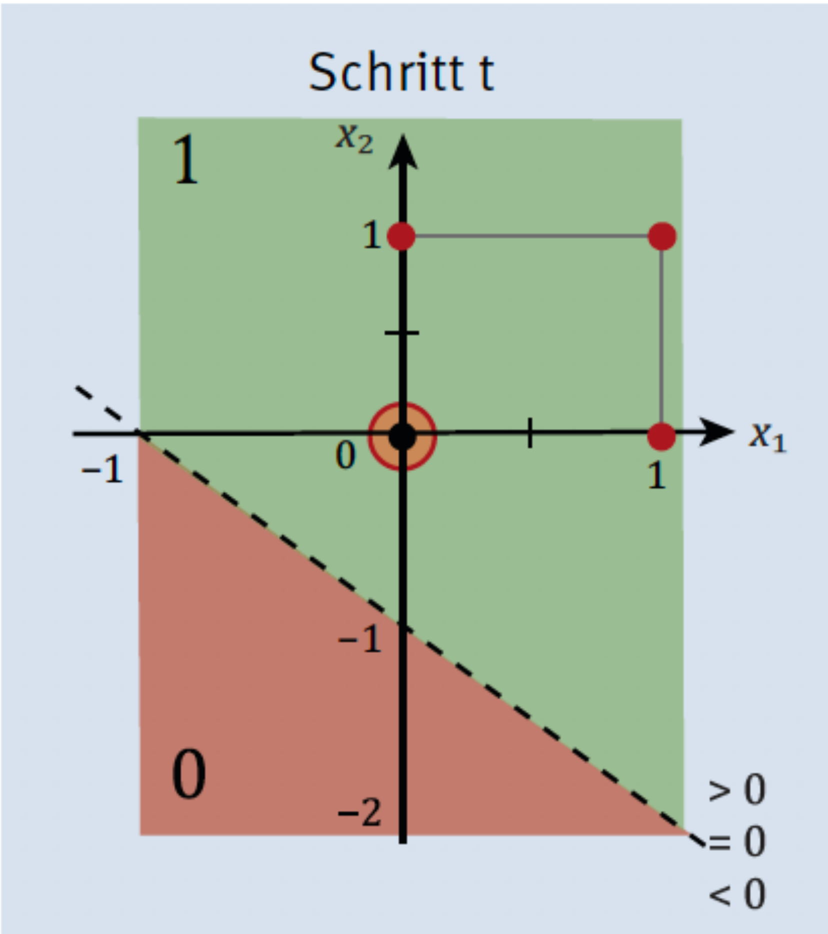
$$\boxed{\Delta w_i = (y_i - \hat{y}_i) \cdot x_i}$$

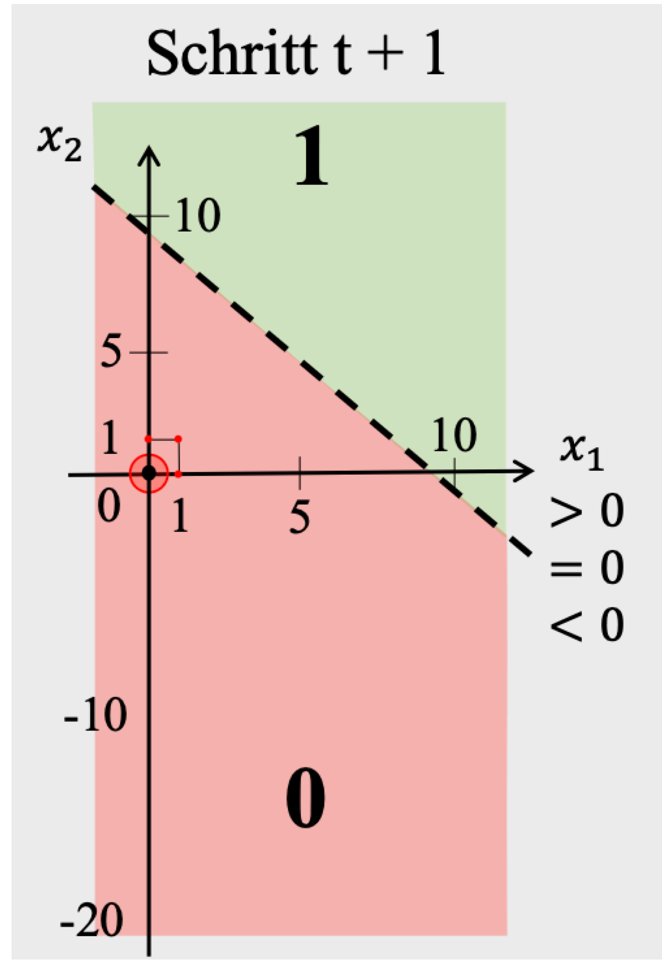
Lernen im einfachen Netz

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

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

Lernen im einfachen Netz





Lernen im einfachen Netz Perceptron Lernalgo 1/3

📟 Kapitel 4

```
# Grafische Darstellung
import matplotlib.pyplot as plt
# Zufall
from random import choice
# Für die mathematischen Operationen
from numpy import array, dot, random, linspace, zeros
# Ganz wichtig, sonst wird der Plot nicht angezeigt
%matplotlib inline

# Trainingsdaten
# Pro Zeile: die binären Inputdaten und die gewünschte binäre Ausgabe
# in einer Liste von Tupeln.
# An der Stelle 0 des Inputvektors ist das Bias-Neuron
training_data_set = [
    (array([1,0,0]), 0),
    (array([1,0,1]), 1),
    (array([1,1,0]), 1),
```

Lernen im einfachen Netz Perceptron Lernalgo 2/3

Kapitel 4

```
    (array([1,1,1]), 1),
]

# Die Heaviside-Stufenfunktion als Lambda-Funktion
heaviside = lambda x: 0 if x < 0 else 1

# Anfangsinitialisierung des Zufallgenerators wegen
# Reproduzierbarkeit der Ergebnisse
random.seed( 18 ) # irgendein Wert

# Array von Länge 3 mit 0 initialisieren
w = zeros(3)
# Die Anzahl der Durchläufe. Erfahrungswert durch Probieren
iterations = 25

# Start des Trainierens
def fit(iterations, training_data_set,w):
    """ Lernen im Perceptron
            iterations: Ein Vorwärts- und Rückwärtslauf aller Trainingsbeispiele
            trainings_data_set: Die Trainingsbeispiele
            w: Die Gewichte zum Starten
    """
    errors = []
    weights = []
    for i in range(iterations):
#       zufällige Auswahl eines Trainingsbeispiels
        training_data = choice(training_data_set)
        x = training_data[0]
        y = training_data[1]
# Den errechneten Output ermitteln: Gewichtete Summe mit
# nachgelagerter Stufenfunktion
        y_hat = heaviside(dot(w, x))
# Fehler berechnen als Differenz zwischen gewünschtem und
# aktuellem Output
        error = y - y_hat
# Fehler sammeln für die Ausgabe
        errors.append(error)
# Gewichte sammeln für spätere Ausgabe
        weights.append(w)
# Gewichtsanpassung = Das Lernen… x_i ist entweder 0 oder 1
        w += error * x
```

Lernen im einfachen Netz Perceptron Lernalgo 3/3

Kapitel 4

```
# Rückgabe der Fehler und Gewichte
    return errors, weights

# Trainieren
# Wir sammeln die Fehler/Gewichte in jedem Schritt für die grafische Ausgabe
errors, weights = fit(iterations, training_data_set,w)
# Den letzten Gewichtsvektor ausgeben
w = weights[iterations-1]
print("Gewichtsvektor am Ende des Trainings:")
print((w))

# Auswertung nach dem Trainieren
print("Auswertung am Ende des Trainings:")
for x, y in training_data_set:
    y_hat = heaviside(dot(x, w))
    print("{}: {} -> {}".format(x, y, y_hat))
#-------------------------------------------------------
# Grafik für Fehler pro Lernbeispiel :-)
# Figure-Nummern Start
fignr = 1
# Druckgröße in inch
plt.figure(fignr,figsize=(10,10))
# Ausgabe Fehler als Plot
plt.plot(errors)
# Raster
plt.style.use('seaborn-whitegrid')
# Labels
plt.xlabel('Iterationen')
plt.ylabel(r"$(y - \hat y)$")
# Ausgabe: Perfekt ?
Gewichtsvektor am Ende des Trainings:
[-1.  1.  1.]
Auswertung am Ende des Trainings:
[1 0 0]: 0 -> 0
[1 0 1]: 1 -> 1
[1 1 0]: 1 -> 1
[1 1 1]: 1 -> 1
```

Listing 4.1 Perceptron-Lernalgorithmus

Lernen im einfachen Netz

Kapitel 4

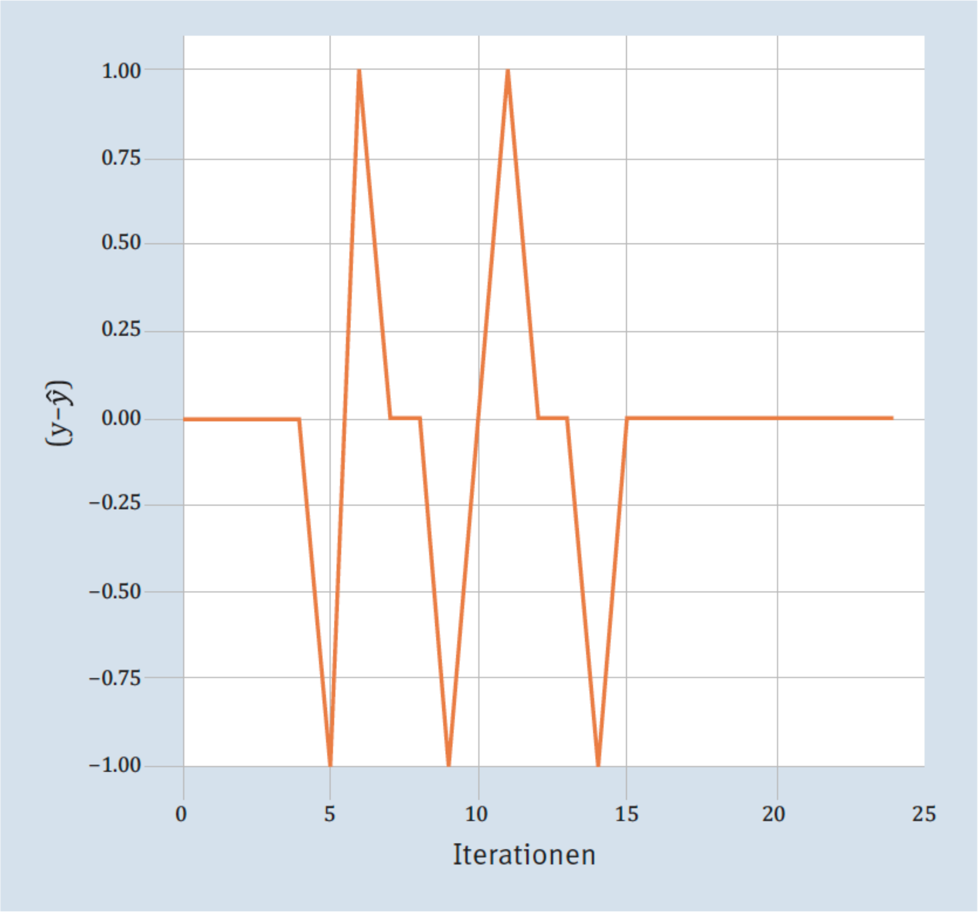


Abbildung 4.4 Die Differenz zwischen gewünschtem und errechnetem Output $(y - \hat{y})$ beim Lernen, pro zufällig gewähltem Trainingsbeispiel

Lernen im einfachen Netz

Aufgabe: Korrektur der Gewichte

Nehmen wir an, dass der Gewichtsvektor die Werte (–0.28, 0.02, 0.05) hat. Als Input wird dem KNN der Vektor (1,0,1) übergeben mit dem gewünschten Output 1. Wie sehen die neuen Gewichte aus, nach der Korrektur? Bitte um Ihre Berechnungen!

Lösung:

1. Zuerst $\hat{y}$ berechnen:

$$-0.28 \cdot 1 + 0.02 \cdot 0 + 0.05 \cdot 1 = -0.23 < 0 \rightarrow 0$$

2. Dann das Δw:

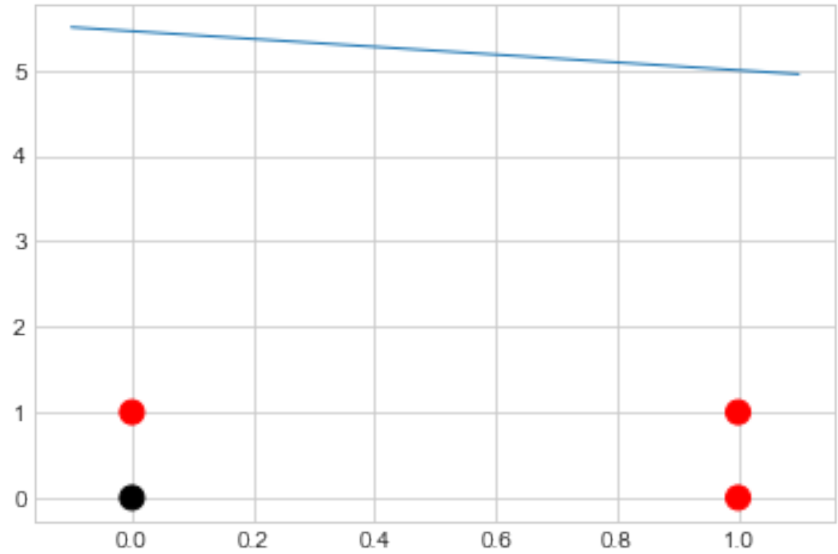
$$(1 - 0) \cdot x = 1 \cdot (1{,}0{,}1) = (1{,}0{,}1)$$

3. Großes Finale, das neue Gewicht:

$$(-0.28, 0.02, 0.05) + (1{,}0{,}1) = (0.72, 0.02, 1.05)$$

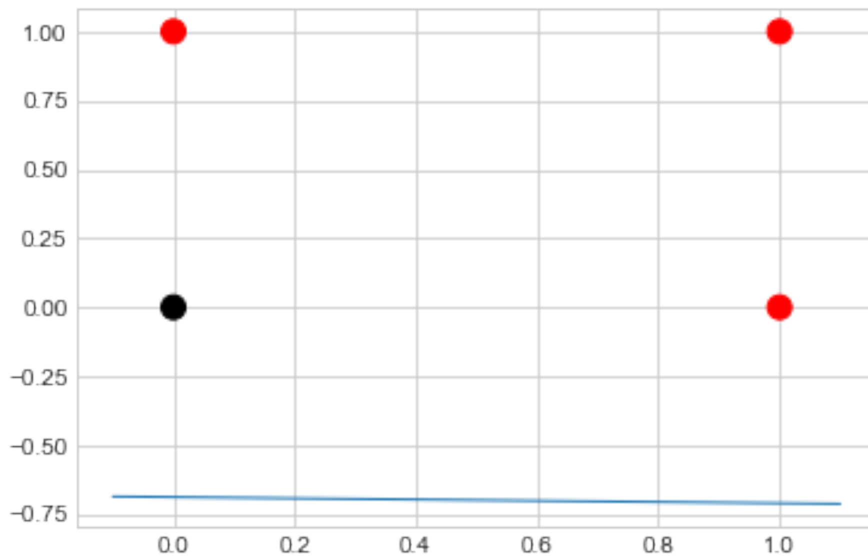
Lernen im einfachen Netz

$$Schritt = 0$$
$$x = (1, 0, 0)$$
$$y = 0$$
$$w_a = (-0.28, 0.02, 0.05)$$
$$\hat{y} = w_a * x = (-0.28, 0.02, 0.05) * (1, 0, 0) = -0.28 \rightarrow 0$$
$$Fehler = (y - \hat{y}) = (0 - 0) = 0$$
$$Korrektur = (y - \hat{y}) * x = 0 * (1, 0, 0) = (0, 0, 0)$$
$$w_n = w_a + Korrektur = (-0.28, 0.02, 0.05)$$



Lernen im einfachen Netz

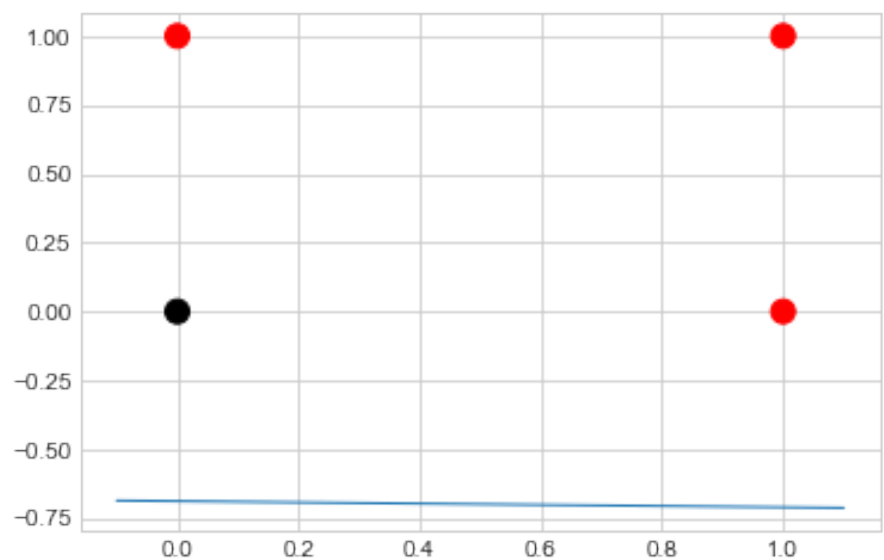
$$Schritt = 1$$
$$x = (1, 0, 1)$$
$$y = 1$$
$$w_a = (-0.28, 0.02, 0.05)$$
$$\hat{y} = w_a * x = (-0.28, 0.02, 0.05) * (1, 0, 1) = -0.23 \rightarrow 0$$
$$Fehler = (y - \hat{y}) = (1 - 0) = 1$$
$$Korrektur = (y - \hat{y}) * x = 1 * (1, 0, 1) = (1, 0, 1)$$
$$w_n = w_a + Korrektur = (0.72, 0.02, 1.05)$$



NōR GmbH
empower the evolution of now

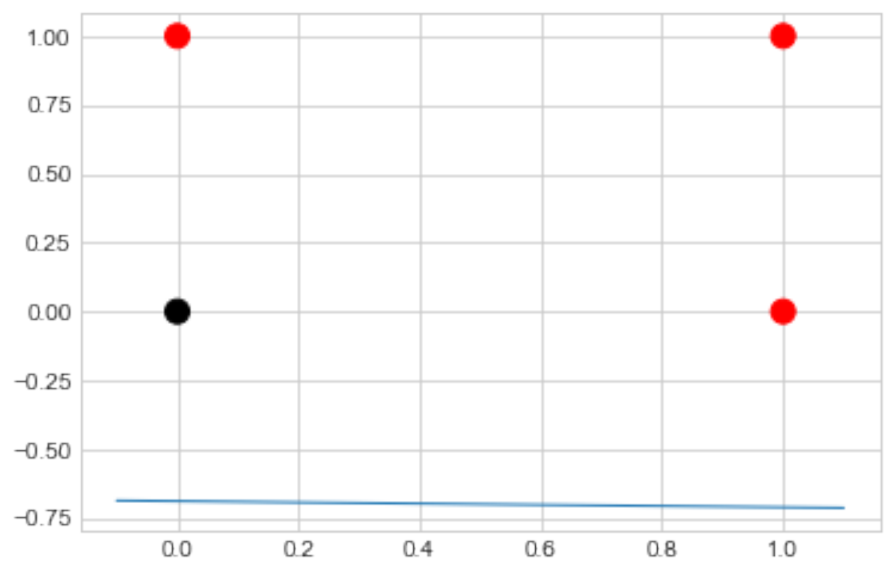
Lernen im einfachen Netz

$$Schritt = 2$$
$$x = (1, 0, 1)$$
$$y = 1$$
$$w_a = (0.72, 0.02, 1.05)$$
$$\hat{y} = w_a * x = (0.72, 0.02, 1.05) * (1, 0, 1) = 1.77 \rightarrow 1$$
$$Fehler = (y - \hat{y}) = (1 - 1) = 0$$
$$Korrektur = (y - \hat{y}) * x = 0 * (1, 0, 1) = (0, 0, 0)$$
$$w_n = w_a + Korrektur = (0.72, 0.02, 1.05)$$



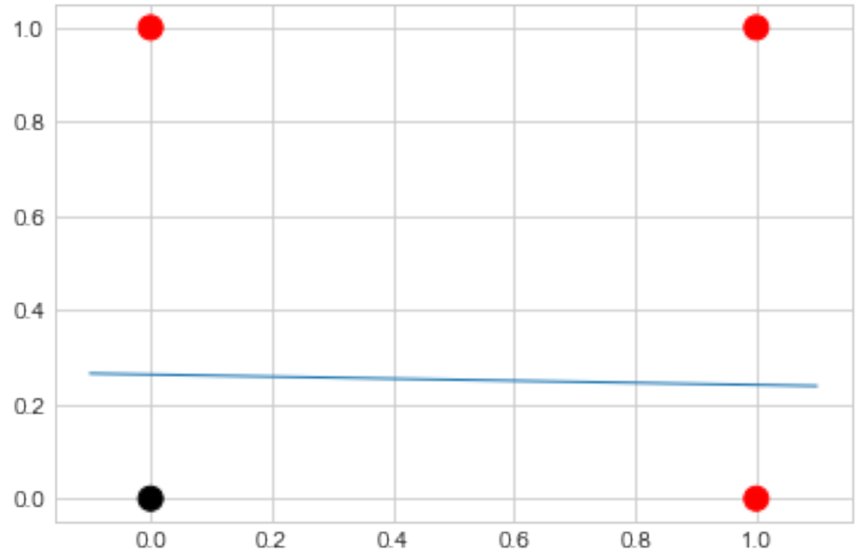
Lernen im einfachen Netz

$$Schritt = 3$$
$$x = (1, 1, 0)$$
$$y = 1$$
$$w_a = (0.72, 0.02, 1.05)$$
$$\hat{y} = w_a * x = (0.72, 0.02, 1.05) * (1, 1, 0) = 0.75 \rightarrow 1$$
$$Fehler = (y - \hat{y}) = (1 - 1) = 0$$
$$Korrektur = (y - \hat{y}) * x = 0 * (1, 1, 0) = (0, 0, 0)$$
$$w_n = w_a + Korrektur = (0.72, 0.02, 1.05)$$



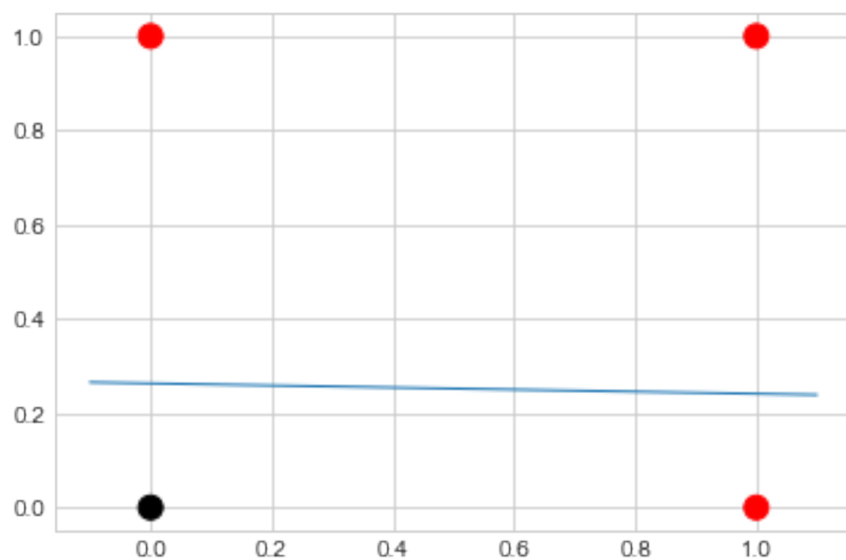
Lernen im einfachen Netz

$$Schritt = 4$$
$$x = (1,0,0)$$
$$y = 0$$
$$w_a = (0.72, 0.02, 1.05)$$
$$\hat{y} = w_a * x = (0.72, 0.02, 1.05) * (1,0,0) = 0.72 \rightarrow 1$$
$$Fehler = (y - \hat{y}) = (0 - 1) = -1$$
$$Korrektur = (y - \hat{y}) * x = -1 * (1,0,0) = (-1,0,0)$$
$$w_n = w_a + Korrektur = (-0.28, 0.02, 1.05)$$



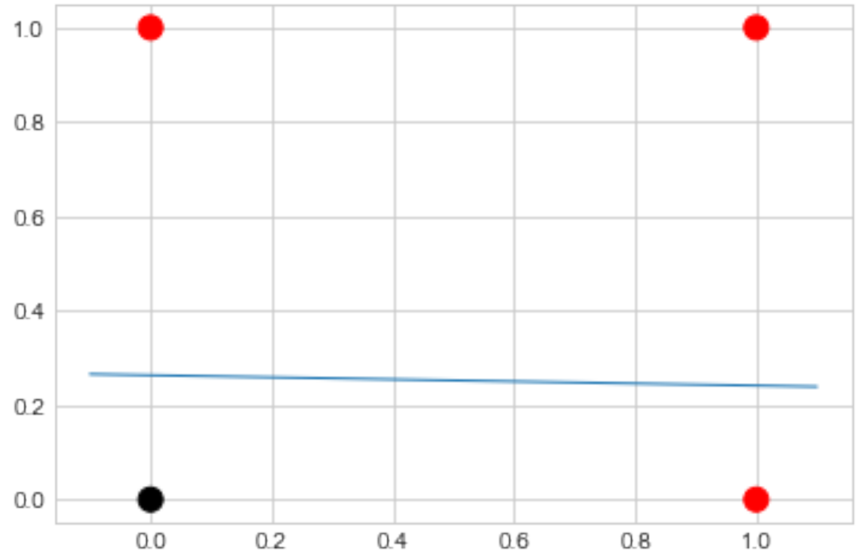
Lernen im einfachen Netz

$$Schritt = 5$$
$$x = (1, 0, 0)$$
$$y = 0$$
$$w_a = (-0.28, 0.02, 1.05)$$
$$\hat{y} = w_a * x = (-0.28, 0.02, 1.05) * (1, 0, 0) = -0.28 \rightarrow 0$$
$$Fehler = (y - \hat{y}) = (0 - 0) = 0$$
$$Korrektur = (y - \hat{y}) * x = 0 * (1, 0, 0) = (0, 0, 0)$$
$$w_n = w_a + Korrektur = (-0.28, 0.02, 1.05)$$



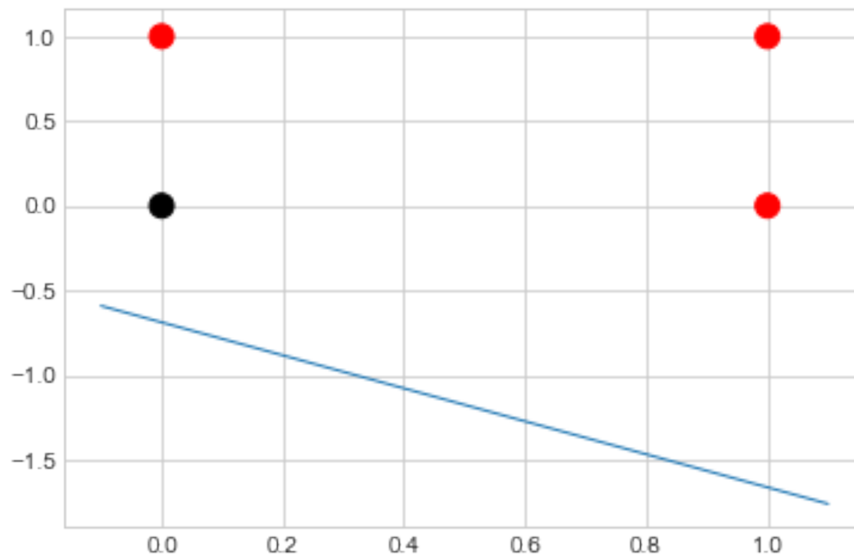
Lernen im einfachen Netz

$$Schritt = 6$$
$$x = (1, 1, 1)$$
$$y = 1$$
$$w_a = (-0.28, 0.02, 1.05)$$
$$\hat{y} = w_a * x = (-0.28, 0.02, 1.05) * (1, 1, 1) = 0.80 \rightarrow 1$$
$$Fehler = (y - \hat{y}) = (1 - 1) = 0$$
$$Korrektur = (y - \hat{y}) * x = 0 * (1, 1, 1) = (0, 0, 0)$$
$$w_n = w_a + Korrektur = (-0.28, 0.02, 1.05)$$



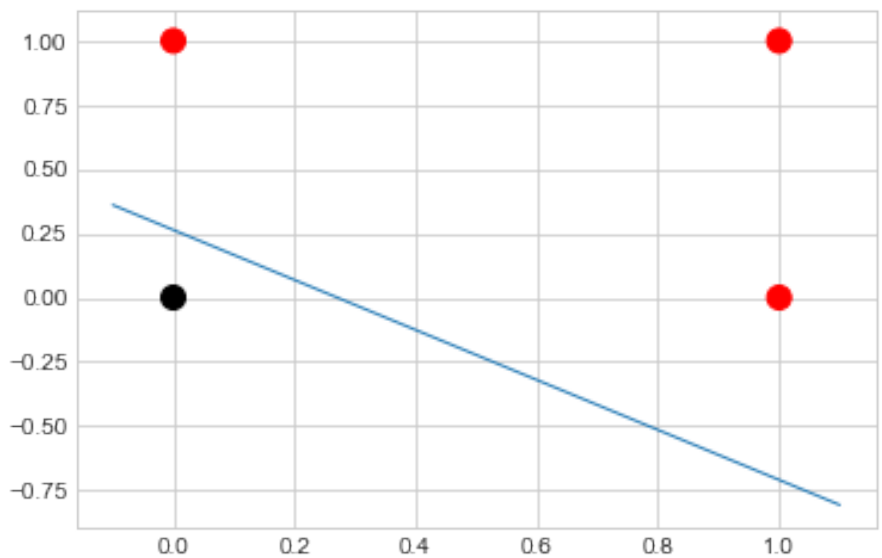
Lernen im einfachen Netz

$$Schritt = 7$$
$$x = (1, 1, 0)$$
$$y = 1$$
$$w_a = (-0.28, 0.02, 1.05)$$
$$\hat{y} = w_a * x = (-0.28, 0.02, 1.05) * (1, 1, 0) = -0.25 \to 0$$
$$Fehler = (y - \hat{y}) = (1 - 0) = 1$$
$$Korrektur = (y - \hat{y}) * x = 1 * (1, 1, 0) = (1, 1, 0)$$
$$w_n = w_a + Korrektur = (0.72, 1.02, 1.05)$$



Lernen im einfachen Netz

$$Schritt = 8$$
$$x = (1, 0, 0)$$
$$y = 0$$
$$w_a = (0.72, 1.02, 1.05)$$
$$\hat{y} = w_a * x = (0.72, 1.02, 1.05) * (1, 0, 0) = 0.72 \rightarrow 1$$
$$Fehler = (y - \hat{y}) = (0 - 1) = -1$$
$$Korrektur = (y - \hat{y}) * x = -1 * (1, 0, 0) = (-1, 0, 0)$$
$$w_n = w_a + Korrektur = (-0.28, 1.02, 1.05)$$



Lernen im einfachen Netz

Index	x	y	$\hat{y}$	Fehler $(y-\hat{y})$	Korrektur $(y-\hat{y})\cdot x$	w	Abbildung
						(−0.28, 0.02, 0.05)	
0	(1,0,0)	0	0	0	(−1,0,0)	(−0.28, 0.02, 0.05)	Abbildung 4.5
1	(1,0,1)	1	0	1	(1,0,1)	(0.72, 0.02, 1.05)	Abbildung 4.5
2	(1,0,1)	1	1	0	(0,0,0)	(0.72, 0.02, 1.05)	Abbildung 4.6
3	(1,1,0)	1	1	0	(0,0,0)	(0.72, 0.02, 1.05)	Abbildung 4.6
4	(1,0,0)	0	1	−1	(−1,0,0)	(−0.28,0.02,1.05)	Abbildung 4.7
5	(1,0,0)	0	0	0	(0,0,0)	(−0.28,0.02,1.05)	Abbildung 4.7
6	(1,1,1)	1	1	0	(0,0,0)	(−0.28,0.02,1.05)	Abbildung 4.8
7	(1,1,0)	1	0	1	(1,1,0)	(0.72,1.02,1.05)	Abbildung 4.8
8	(1,0,0)	0	1	−1	(−1,0,0)	(−0.28,1.02,1.05)	Abbildung 4.9

Tabelle 4.4 Lernschritte und Anpassungen (Forts.)

Lernen im einfachen Netz

Kapitel 4

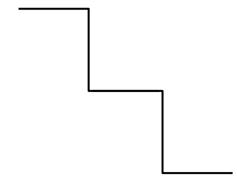
```
# scikit-learn-Perceptron
import numpy as np
# Das bereits bekannte Iris-Dataset
from sklearn.datasets import load_iris
# Ladies and Gentlemen - das Perceptron
from sklearn.linear_model import Perceptron
# Den Iris-Datensatz laden
iris = load_iris()
# Die Eingabevektoren für das Lernen
# 150 Vektoren mit 5 Spalten
X = iris.data[:,(2,3)] # petal length, petal width
# Die gewünschten Werte
y = iris.target
# Das Perceptron instanziieren
# random_state = Seed für Zufallsgenerator
# max_iter = maximale Anzahl an Iterationen
# tol = Stoppkriterium
Perceptron = Perceptron(random_state=49,max_iter=100000,tol=None)
# Lernen bitte
Perceptron.fit(X,y)
# Und auswerten: Iri-setosa, Iris -versicolor, Iris -virginca
y_prediction = Perceptron.predict([ [1.4,0.2], [3.5,1.0], [6.0,2.5]])
# Natürlich die Ausgabe nicht vergessen
print(y_prediction)
# Ausgabe
[0 1 2]
```

Listing 4.10 Die scikit-learn-Implementierung des Perceptrons

Lernen im einfachen Netz Adaline

Kapitel 4

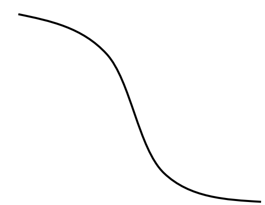
Rosenblatt: Perceptron



$$\Delta w_{ji} = \eta \cdot \left(y_j - \hat{y}_j\right) \cdot x_i \text{, bzw.}$$

$$\Delta w_{ji} = \left(y_j - \hat{y}_j\right) \cdot x_i \text{, für } \eta = 1$$

Widrow-Hoff: Adaline



$$\Delta w_{ji} = \eta \cdot (y_j - net_j) \cdot x_i$$

Lernen im einfachen Netz

- Kapitel 4

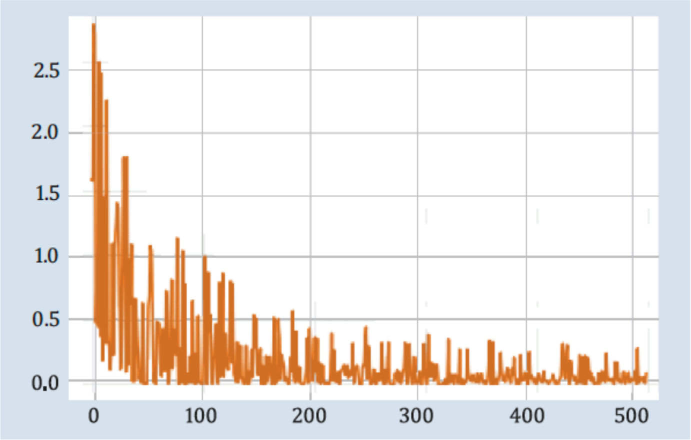


Abbildung 4.16 Adaline-Lernkurve

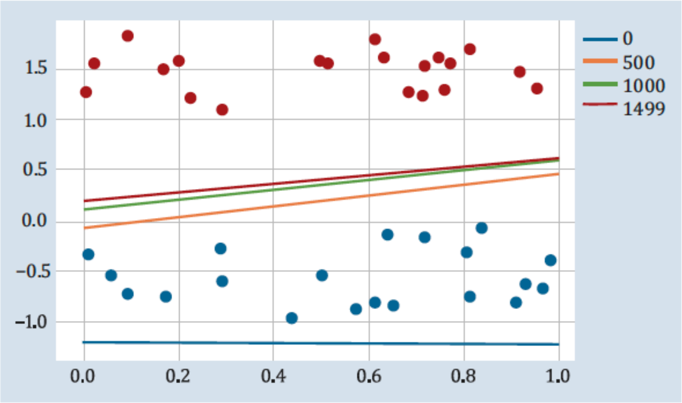


Abbildung 4.17 Adaline-Trenngeraden in unterschiedlichen Schritten

Lernen im einfachen Netz

Kapitel 4

Aufgabe: Lernrate

Ändern Sie die Lernrate von 0.1 bis 0.001, und beobachten Sie dabei die Lernkurve und die Trenngeraden.

Lösung:

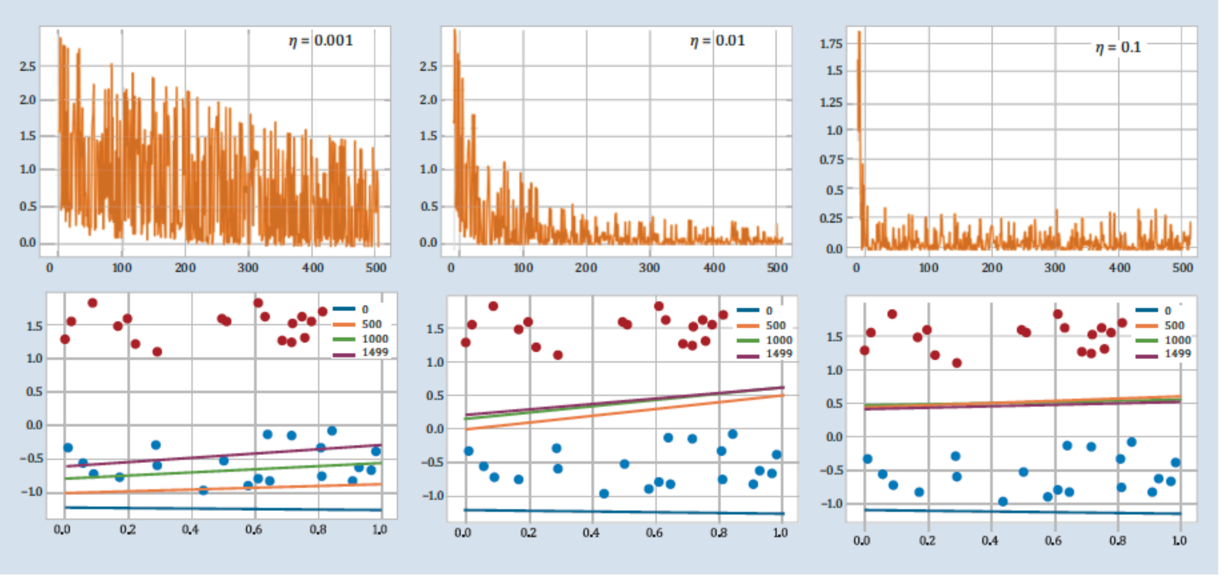


Abbildung 4.18 Lernratenvergleich

Lernen im einfachen Netz

Kapitel 4

Aufgabe: Ändern Sie den Überlappungsfaktor

Beim Wert $y + 1.0$ und $y - 1.0$ haben Sie eine optimale Trennung, also auch kein Problem für ein Perceptron. Verändern Sie den Trennwert, und beobachten Sie die Trenngeraden.

Lösung:

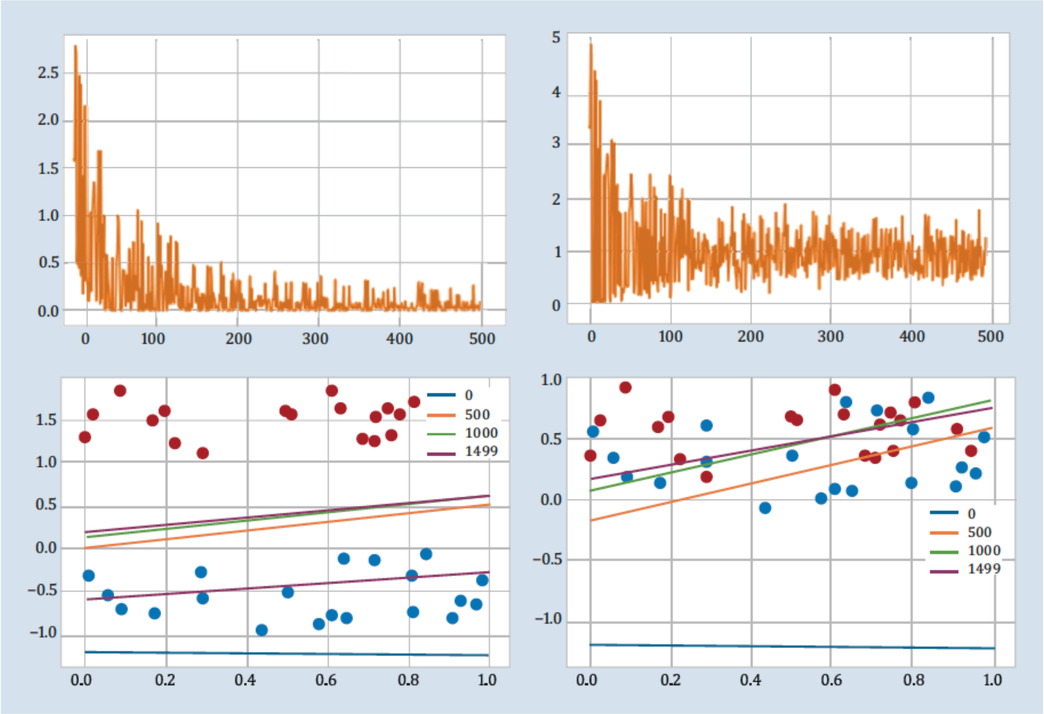
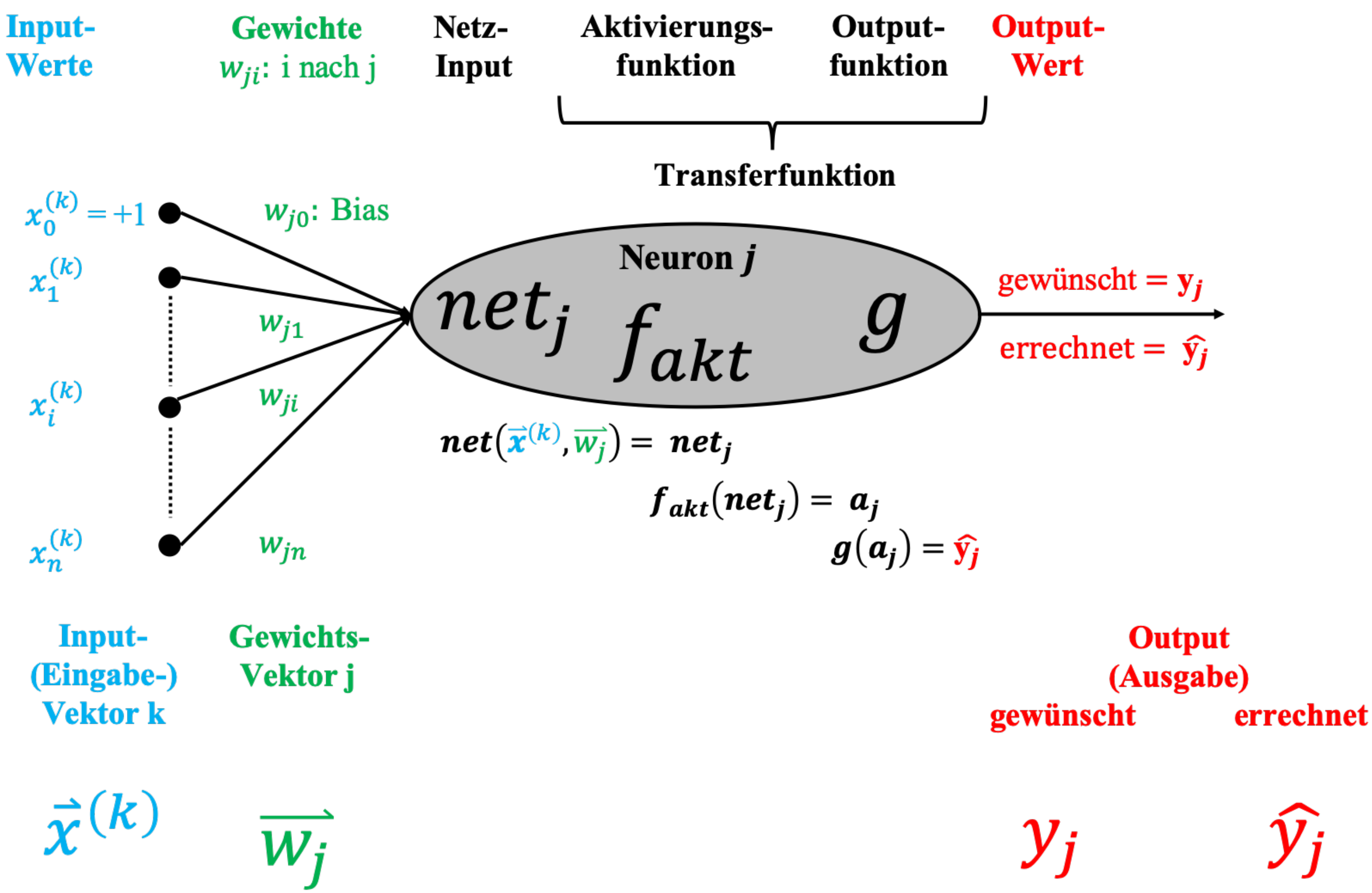


Abbildung 4.19 Überlappungsfaktor der zwei Klassen

Lernen im einfachen Netz



Kapitel 5
Mehrschichtige Netze (MLP)

Mehrschichtiges Netz (MLP)

P1	P2	→	E
0	0	→	0
1	0	→	1
0	1	→	1
1	1	→	0

Tabelle 5.1 Ein verstecktes XOR-Problem

Mehrschichtiges Netz (MLP)

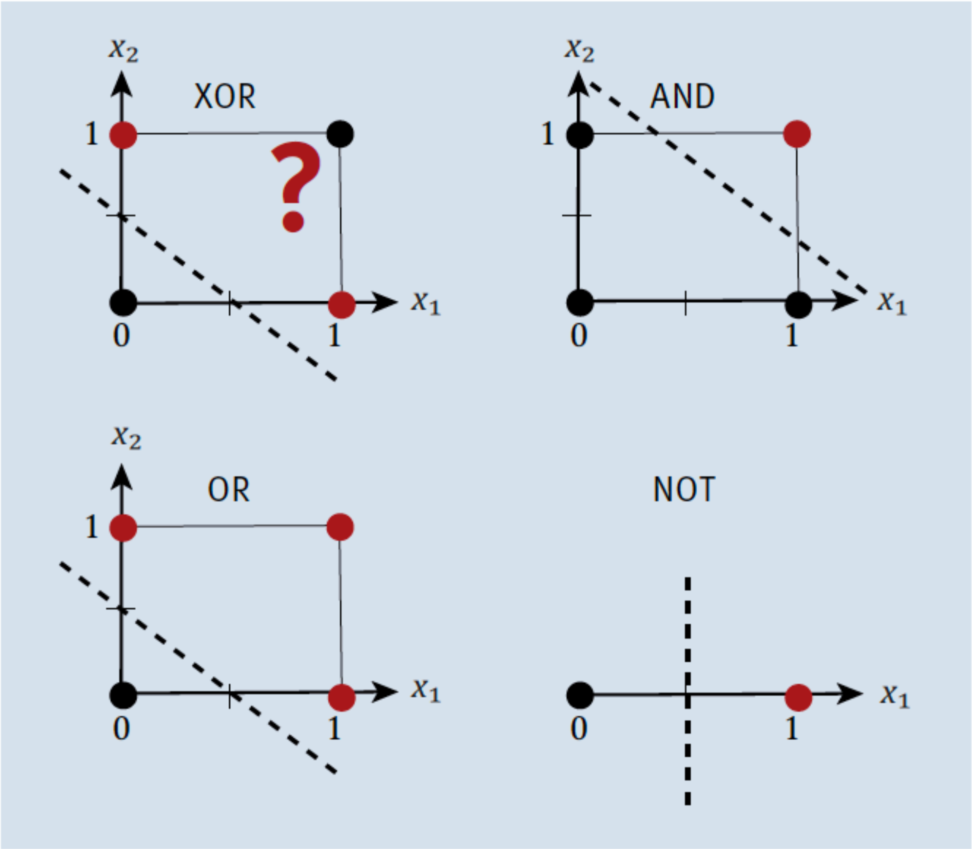


Abbildung 5.1 Wie soll man diese Planungsaufgabe lösen?

Mehrschichtiges Netz (MLP)

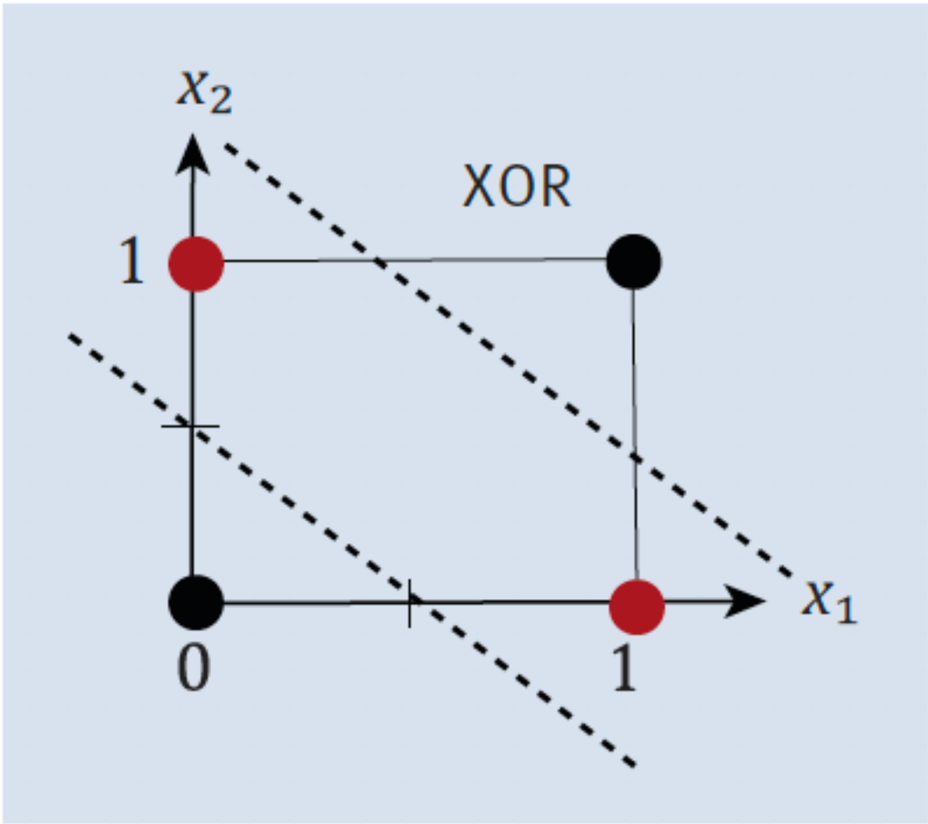


Abbildung 5.2 XOR gelöst

Mehrschichtiges Netz (MLP)

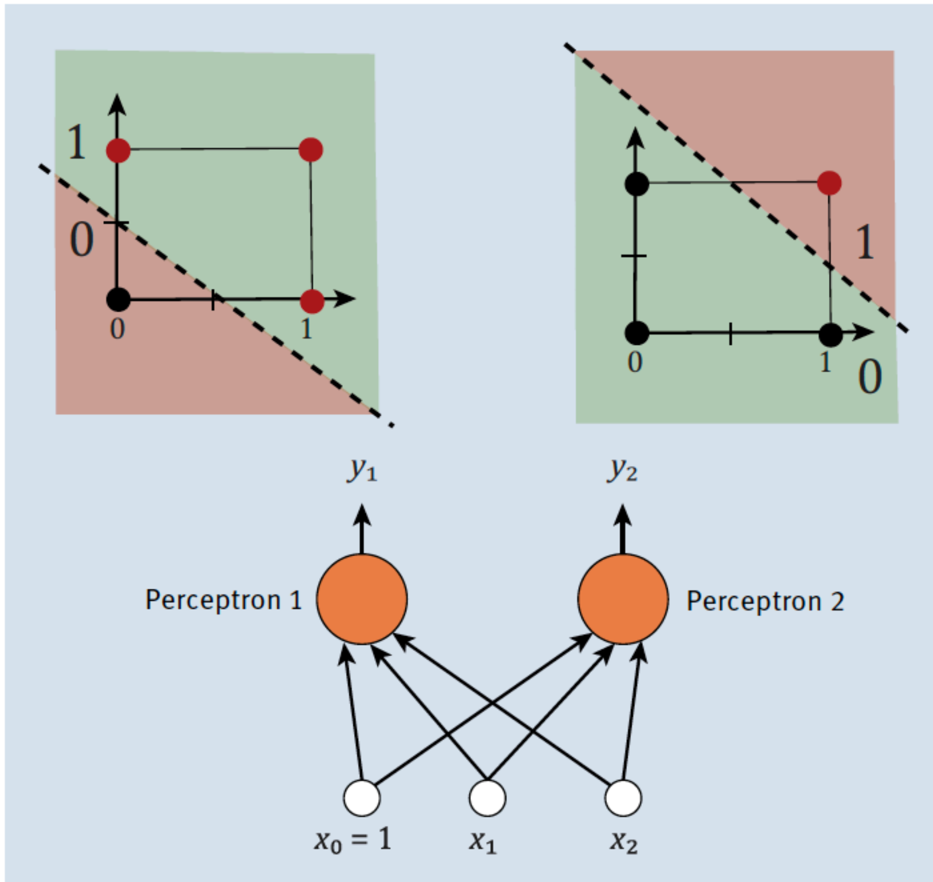


Abbildung 5.4 Lernaufgaben von Perceptron 1 und Perceptron 2

Mehrschichtiges Netz (MLP)

$x1$	$x2$	Perceptron 1	Perceptron 2	Perceptron 3
0	0	0	0	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	0

Tabelle 5.3 Perceptron 3 muss auch lernen.

Mehrschichtiges Netz (MLP)

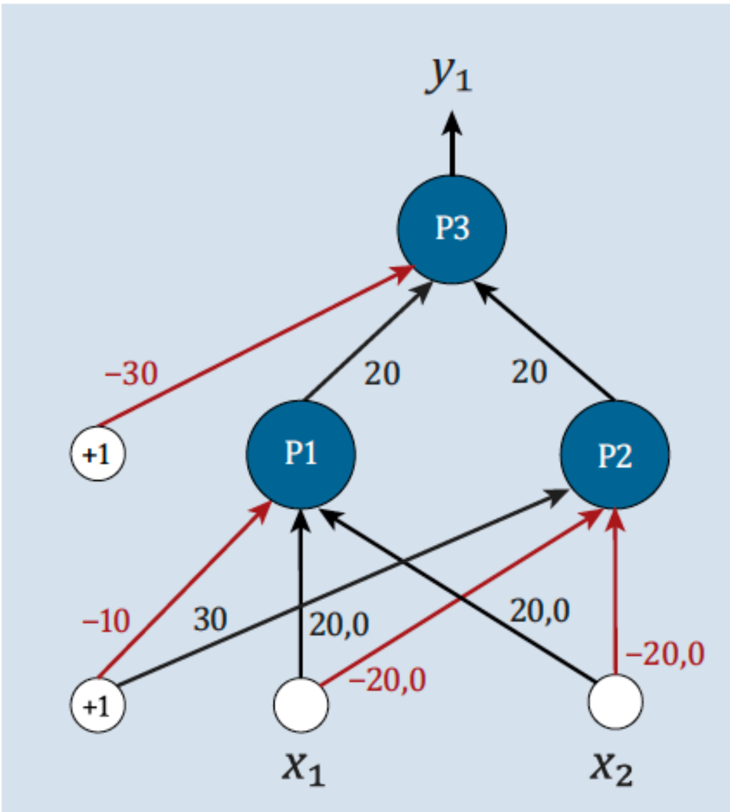
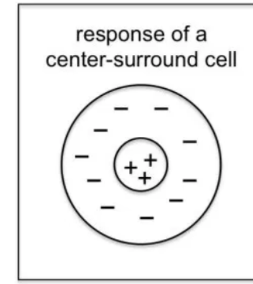
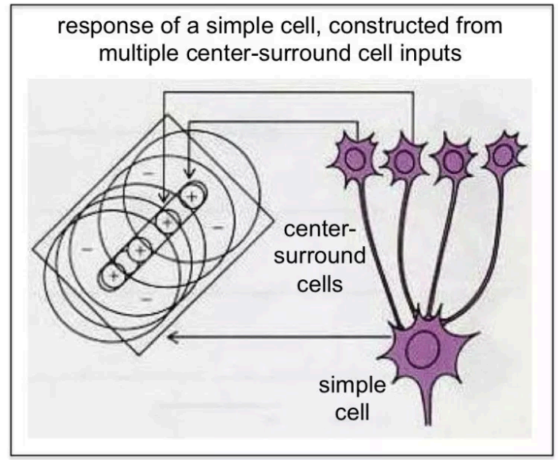
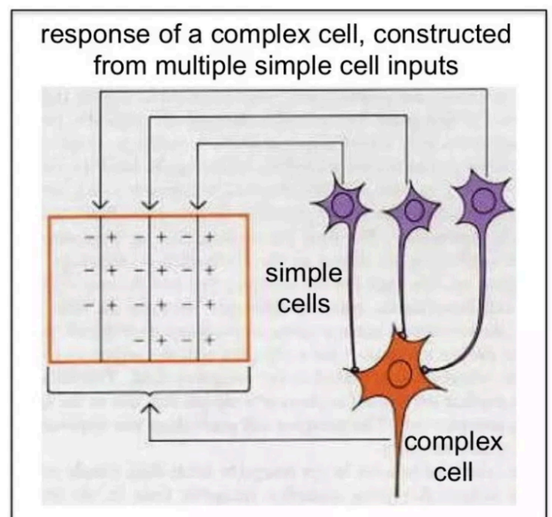


Abbildung 5.6 Das Netzwerk für das XOR-Problem

Hubel: "Eye, Brain, and Vision")







Mehrschichtiges Netz (MLP)

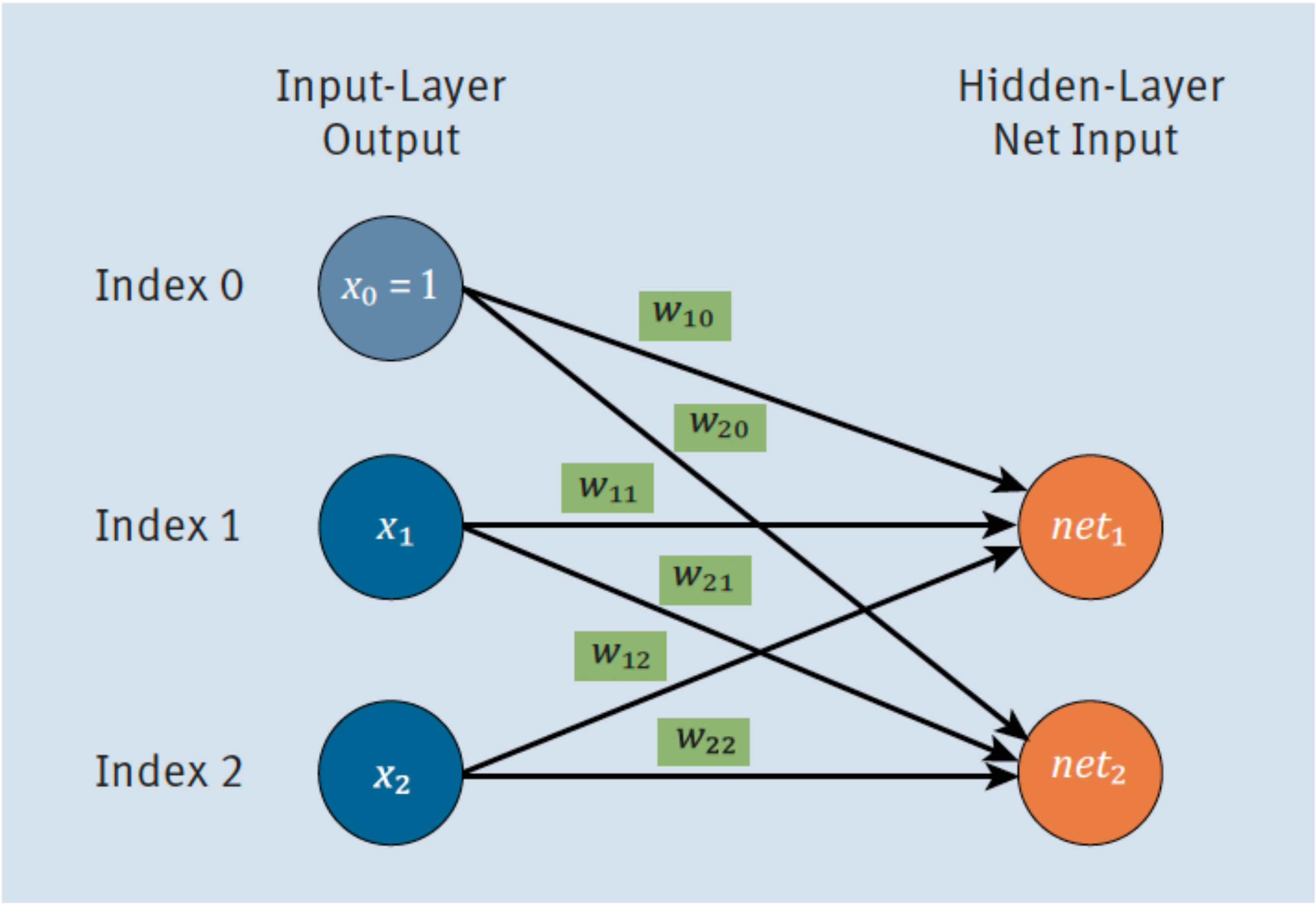


Abbildung 5.7 Die Berechnung im mehrschichtigen Netz

Mehrschichtiges Netz (MLP)

$$net_1 = w_{10} \cdot x_0 = 1 + w_{11} \cdot x_1 + w_{12} \cdot x_2$$

$$net_2 = w_{20} \cdot x_0 = 1 + w_{21} \cdot x_1 + w_{22} \cdot x_2$$

Abbildung 5.8 Auf dem Weg zur Matrixmultiplikation

Mehrschichtiges Netz (MLP)

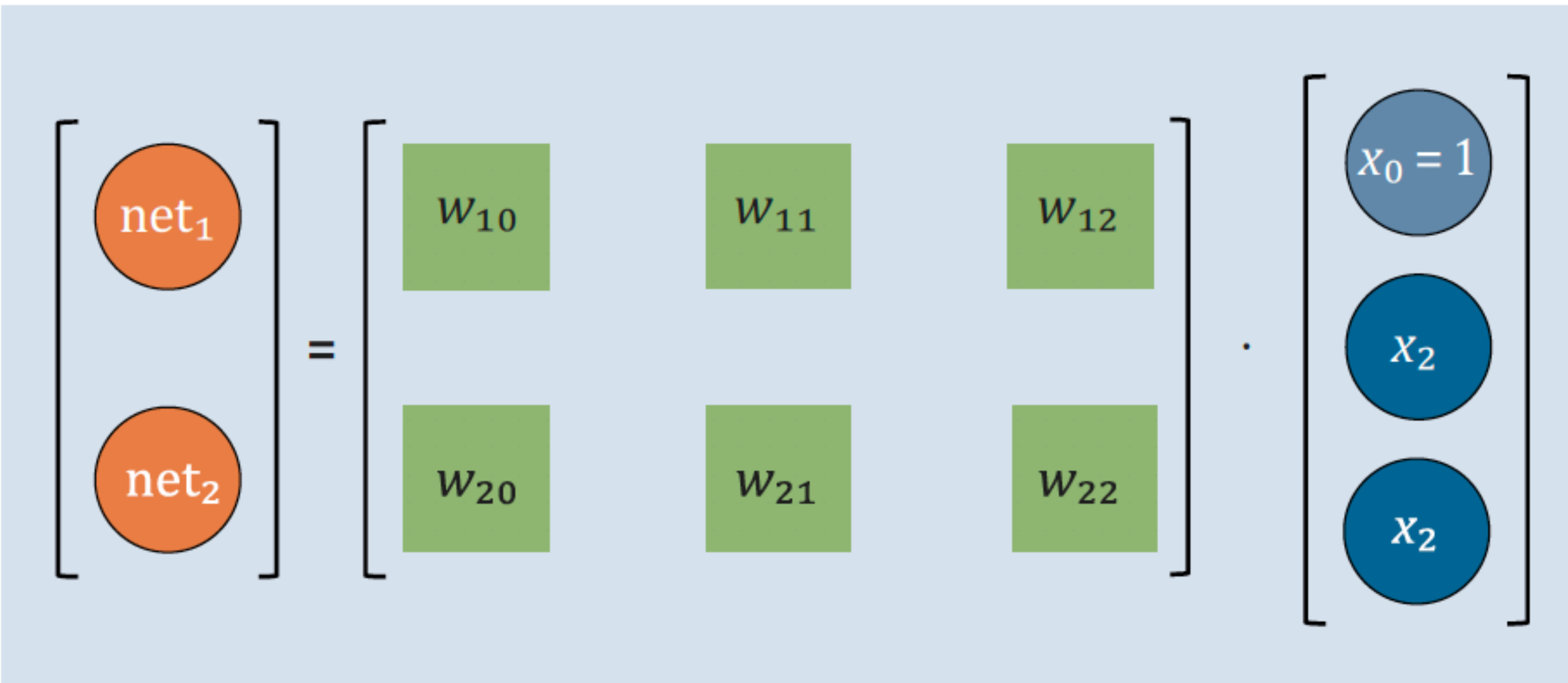


Abbildung 5.9 Matrixmultiplikation, super geeignet für »numpy«

Mehrschichtiges Netz (MLP)

📟 Kapitel 5

Check the Code!

Mehrschichtiges Netz (MLP)

Kapitel 5

```
###########################
# Initialisierung der Gewichte
W_IH = np.matrix([[0.0,0.0,0.0],[-10,20.0,20.0],[30,-20.0,-20.0]])
W_HO = np.matrix([[0.0,0.0,0.0],[-30,20.0,20.0]])
weights=[]
weights.append(W_IH)
weights.append(W_HO)
nn = MLP(weights=weights)
# Netzwerk ausgeben
nn.print()
# Test
X=np.array([[1.0,1.0,1.0],[1.0,0,1.0],[1.0,1.0,0],[1.0,0,0]])
y=np.array([0,1.0,1.0,0])
print('Predict:')
for idx,x in enumerate(X):
    print('{} {} -> {}'.format(x,y[idx],nn.predict(x)))
```

Listing 5.6 Die Verwendung der Klasse »MLP«

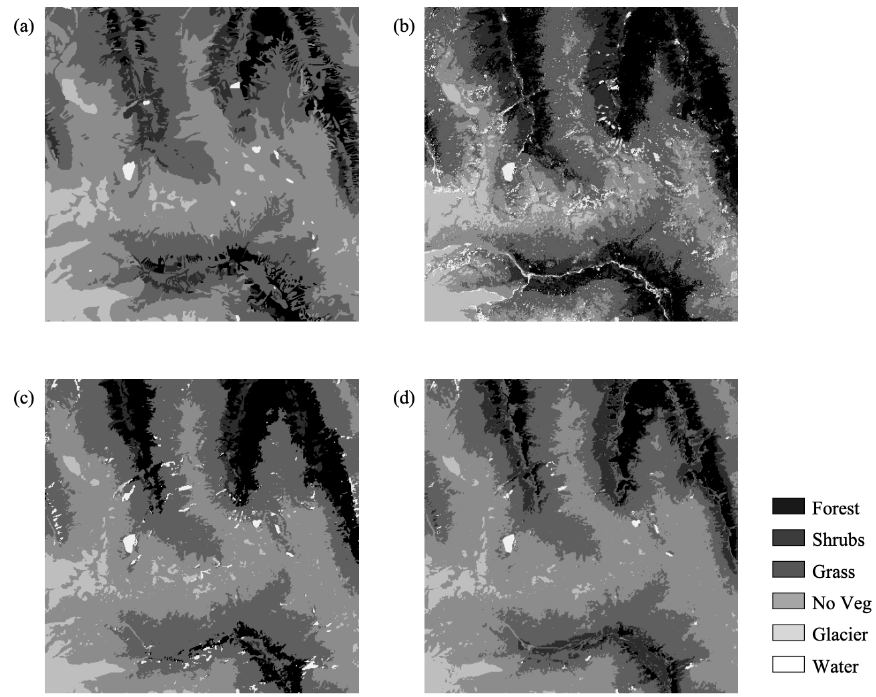
Mehrschichtiges Netz (MLP)

Kapitel 5

```
Multi-Layer-Perceptron - Netzwerkarchitektur
[[ 1.]
 [ 0.]
 [ 0.]]
----------v----------
[[  0.   0.   0.]
 [-10.  20.  20.]
 [ 30. -20. -20.]]
----------v----------
[[ 1.  1.  1.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]
----------v----------
[[  0.   0.   0.]
 [-30.  20.  20.]]
----------v----------
[[ 0.  0.  0.]
 [ 0.  0.  0.]]
----------v----------
Predict:
[ 1.  1.  1.] 0.0 -> [  4.54391049e-05]
[ 1.  0.  1.] 1.0 -> [ 0.99995452]
[ 1.  1.  0.] 1.0 -> [ 0.99995452]
[ 1.  0.  0.] 0.0 -> [  4.54391049e-05]
```

Listing 5.7 Ausgabe der Architektur mit Gewichten und der Vorhersage

Satellitenbild Klassifikation!



Kapitel 6
Lernen im MLP

Lernen im MLP

- Wie misst man einen Fehler?
  - Quadratisch

$$E = (y - \hat{y})^2$$

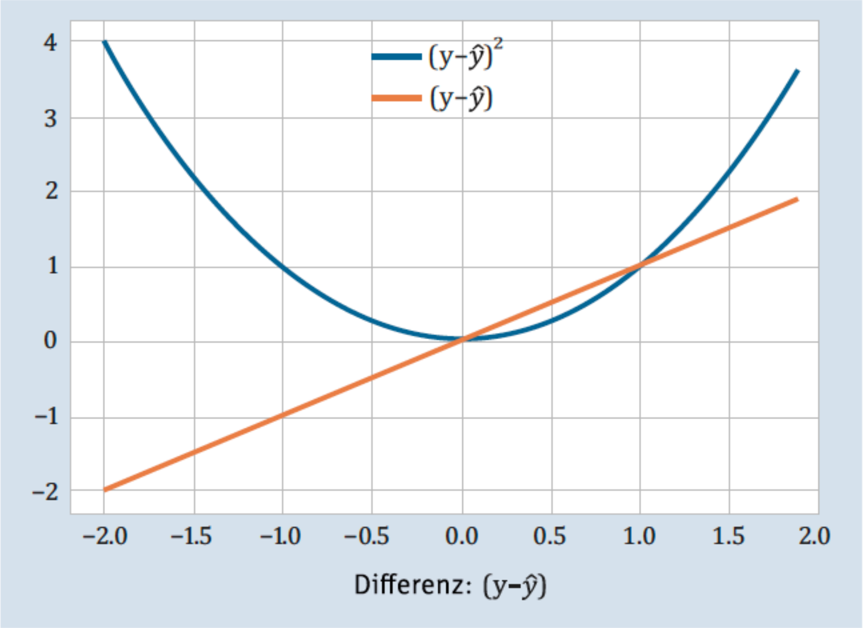


Abbildung 6.1 Fehlerkurven

Frage: Wie kann ich den Fehler zwischen Soll (Wunschwert) und Ist (berechnet) Minimieren? Z.B. Satellitenbild

Lernen im MLP

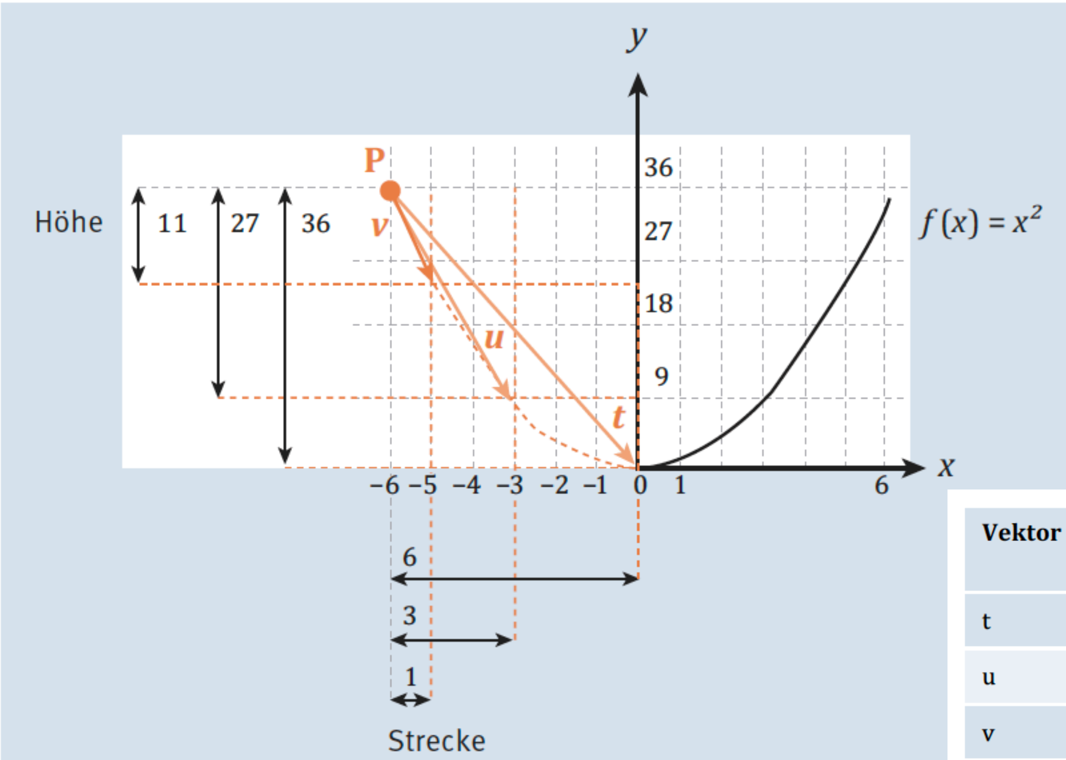


Abbildung 6.2 Der Gradient

Aufgabe: Finde die Richtung In der der Fehler abnimmt!

Oder

Wie komme ich blind den Berg runter?

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

Lernen im MLP

Kapitel 6

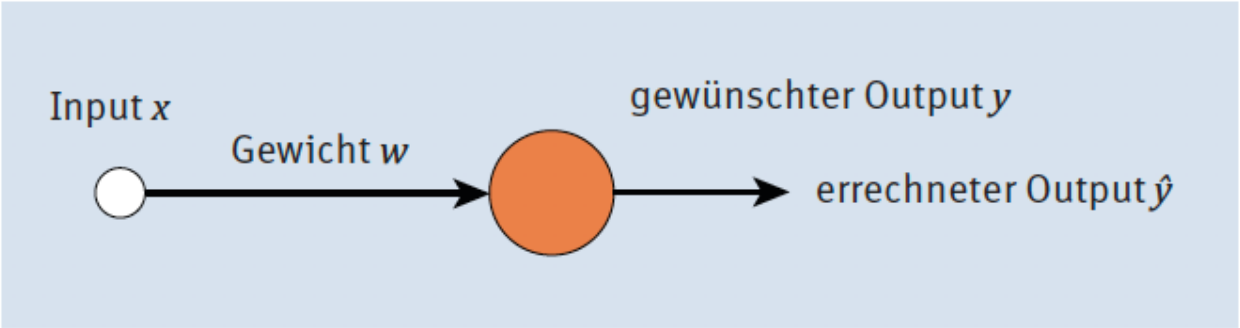


Abbildung 6.3 Einfaches KNN

Aktivitätsfunktion f(x) = x
Input x = 0.2
Output y = 0.2

Gewicht ?

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

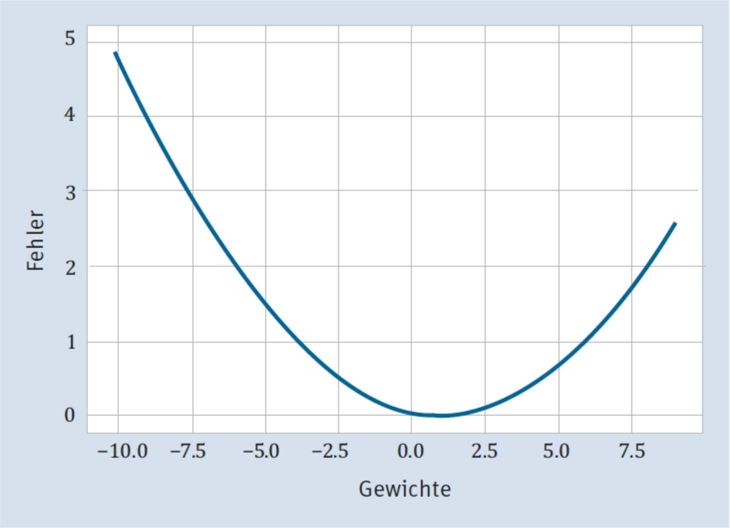


Abbildung 6.4 Fehler je nach Gewicht

Lernen im MLP

Gradientenabstiegsverfahren

Kapitel 6

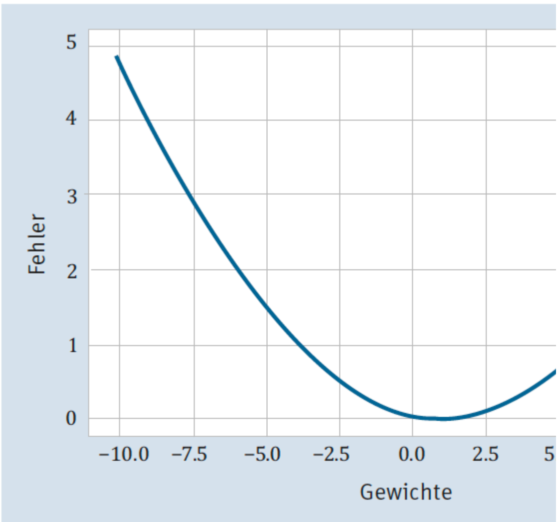


Abbildung 6.4 Fehler je nach Gewicht

1. Wir starten bei einem beliebigen Wert für das Gewicht w.
2. Dann ermitteln wir den negativen Gradienten zu diesem Gewicht. Wie oben beschrieben zeigt der ja zum niedrigsten Punkt.
3. Als Nächstes wandern wir den negativen Gradienten entlang in Richtung Minimum, aber nur für eine gewisse Schrittlänge (wir Konnektionisten nennen das die *Lernrate*).
4. Im neuen Punkt, den wir ermittelt haben, beginnt alles wieder von vorn (gehe zu Schritt 2.), außer wir haben einen Fehler erreicht, der für uns ausreichend ist, z. B. kleiner als 0.001 oder so, oder falls sich der Fehler nicht mehr verkleinern lässt.

Wir müssen Ihnen noch die Formel für die Ermittlung des Gradienten an der Stelle w verraten: Für die Funktion

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

lautet der Gradient:

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

Da wir nicht in die Richtung des stärksten Anstieges gehen wollen, sondern in die entgegengesetzte Richtung, müssen wir den Gradienten $E'(w)$ noch mit (-1) multiplizieren:

$$(-1) \cdot E'(w) = (-1) \cdot (-1) \cdot x \cdot (y - \hat{y}) = x \cdot (y - \hat{y})$$

Lernen im MLP
Gradientenabstiegsverfahren

Kapitel 6

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

Tabelle 6.2 Gradientenabstieg für das einfache KNN

Lernen im MLP
Gradientenabstiegsverfahren

- Lernrate

Aufgabe

Erweitern Sie das Programm so, dass eine Lernrate gesetzt werden kann, und experimentieren Sie damit.

Testen Sie Ihr Programm für die Lernraten 2.0, 1.5, 1.0, 0.01.

- Möglicherweise müssen Sie die Anzahl der Iterationen anpassen.
- Möglicherweise müssen Sie die Ausgabegenauigkeit der Spalte »E« anpassen ({:.6f})

Lösung:

```
# Initialisierungen
x = 0.2
y = x
eta = 0.1
```

und

```
# Delta für Gewichtsanpassung
    w_delta = (-1)*derivative*eta
```

Lernen im MLP

Gradientenabstiegsverfahren

Kapitel 6

- Sigmoide

Transferfunktion

Aufgabe

Ändern Sie den Gradientenabstieg so ab, dass die sigmoide Aktivierungsfunktion verwendet wird.

Lösungshinweise:

Natürlich gilt: Ob Sie es erst einmal selbst probieren oder sich gleich den ersten Hinweis ansehen, sei Ihnen überlassen.

- **Erster Hinweis**: Definition der Sigmoiden (bitte auch numpy importieren):

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

```
def func_sigmoid(x):
    # Wichtig: Nicht math.exp,
    # sondern np.exp wegen array Operationen verwer
    return 1.0 / (1.0 + np.exp(-x))
```

- **Zweiter Hinweis**: Aktivierung berechnen:

```
# Aktivierung (identische Funktion)
#    activation = func_id(net_i)
# Aktivierung (sigmoide)
    activation = func_sigmoid(net_i)
```

- **Dritter Hinweis**: Derivative neu berechnen. Das ist der schwerste Teil, da man die Ableitung der Sigmoide kennen muss: $sigmoide' = sigmoide \cdot (1 - sigmoide)$.

```
# Gradient
#    derivative = (-1.0)*x*(y - y_hat)
    derivative = (-1.0)*activation*(1.0-activation)*(y - y_hat)
```

- **Letzter Hinweis**: Möglicherweise müssen Sie die Anzahl der Iterationen und die Lernrate anpassen.

Die letzte Übung hatte es in sich. Aber wie Sie sehen werden, war das die optimale Vorbereitung für den *Backpropagation-Algorithmus*, den wir im nächsten Abschnitt besprechen.

Lernen im MLP

Kapitel 6

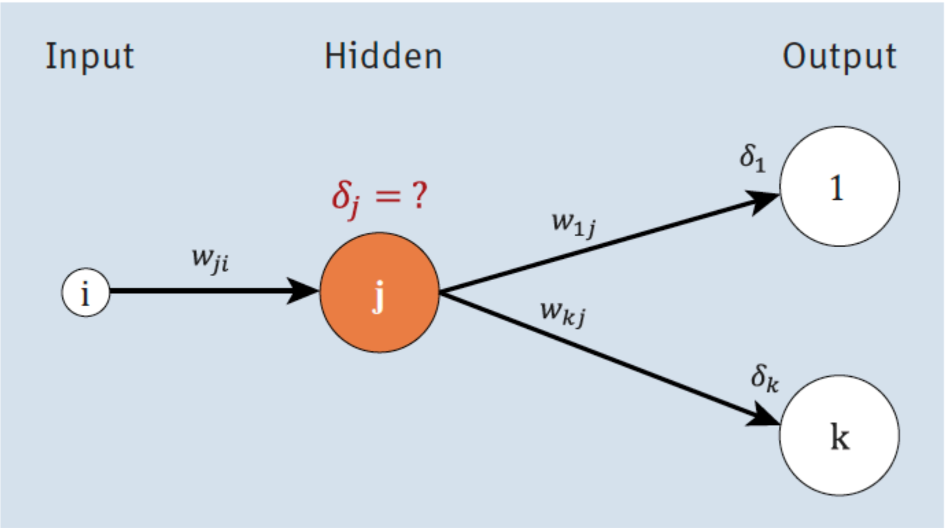


Abbildung 6.6 Vom Output zur versteckten Schicht

Mean Squared Error (MSE)

$$E = \frac{1}{2}\sum_i (y_i - \hat{y}_i)^2$$

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

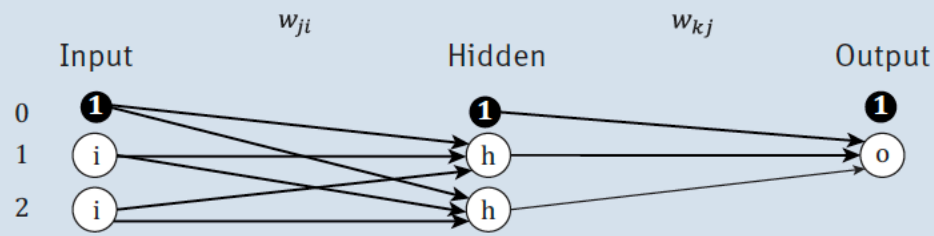
$$\Delta w_{ji} = -\eta \cdot \delta_j \cdot o_i$$

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

Geheimnis

Wir haben die Berechnungsvorschrift für δ_j für die Sigmoide angepasst. Natürlich könnten auch andere Aktivierungsfunktionen verwendet werden, wobei dann in der Berechnungsvorschrift das $f(net_j) \cdot (1 - f(net_j))$ ersetzt werden müsste. Die Details dazu finden sich wieder, für Unerschrockenene, in Anhang B.

Lernen im MLP



net	act	out	der	delta
–	–	1.0	–	
–	–	1.0	–	
–	–	1.0	–	

net	act	out	der	delta
–	–	1.0	–	

net	act	out	der	delta
–	–	–	–	–

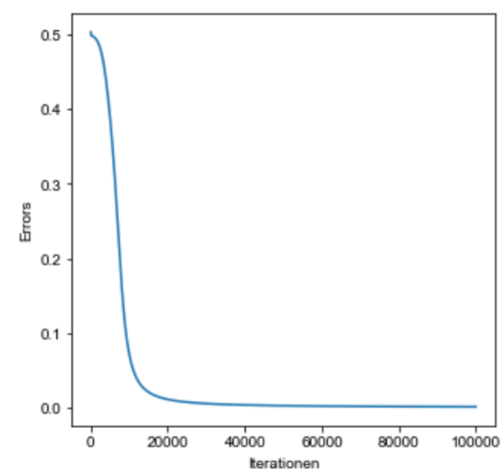
w_{ji}			
von / nach	0	1	2
0	–	–	–
1			
2			

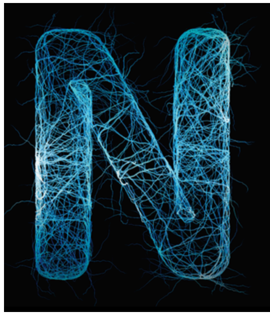
w_{kj}			
von / nach	0	1	2
0	–	–	–
1			

Abbildung 6.12 Netzaufbau für die Iteration

Lernen im MLP

```
Multi Layer Perceptron - Netzwerk Architektur
[[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----------
[[9.877 -4.719 -5.788]
 [2.338 -6.036 -6.033]
 [-7.233 4.650 4.650]]
----------v----------
[[1.000 1.000 1.000 1.000 -0.002]
 [2.338 0.912 0.912 0.080 0.000]
 [-7.233 0.001 0.001 0.001 0.000]]
----------v----------
[[0.416 -0.959 0.940]
 [4.590 -9.243 -9.399]]
----------v----------
[[0.000 0.000 0.000 0.000 0.000]
 [-3.847 0.021 0.021 0.020 -0.000]]
----------v----------
Predict:
[1.000 1.000 1.000] 0.0 -> [0.023]
[1.000 0.000 1.000] 1.0 -> [0.976]
[1.000 1.000 0.000] 1.0 -> [0.976]
[1.000 0.000 0.000] 0.0 -> [0.021]
```





Kapitel 7
Convolutional Neural Networks (CNN)

CNN

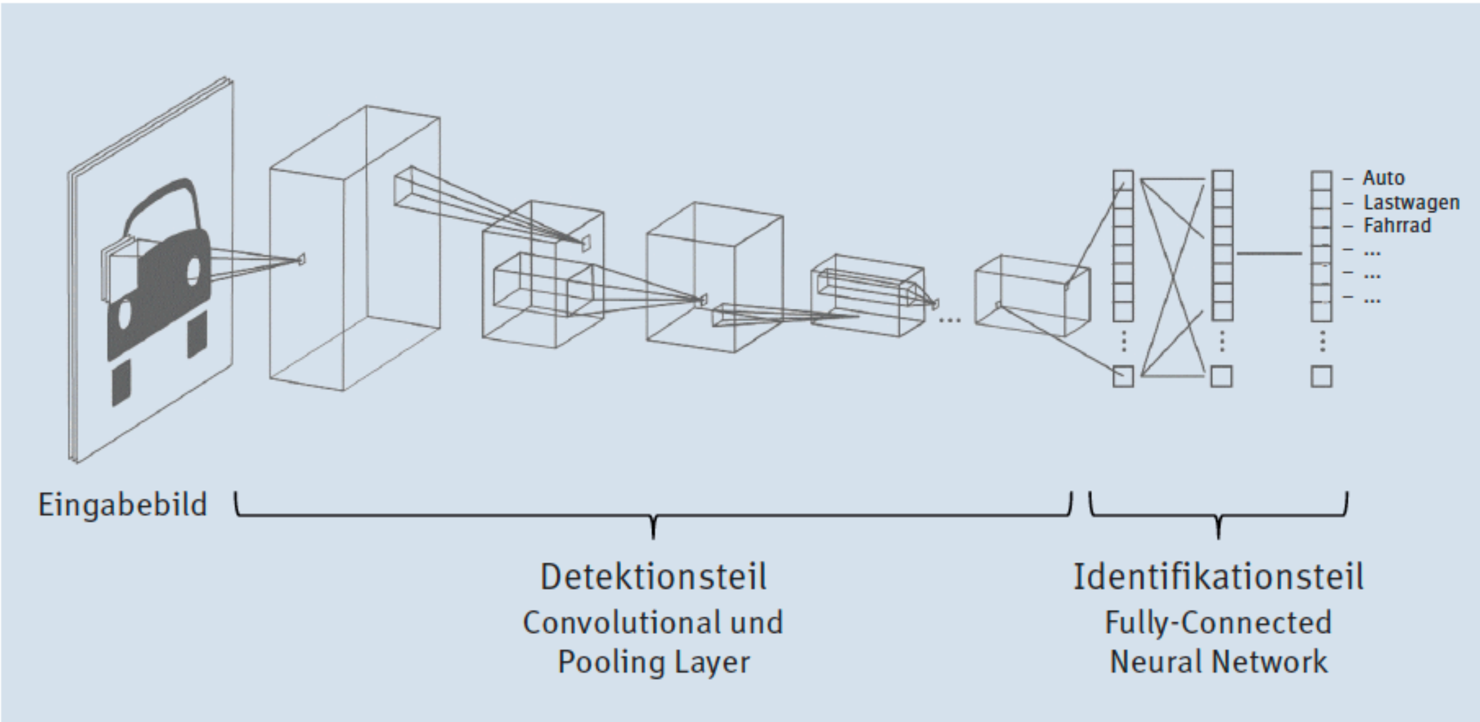


Abbildung 7.2 Die Struktur eines CNN mit Detektions- und Identifikationsteil

CNN

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.).

CNN

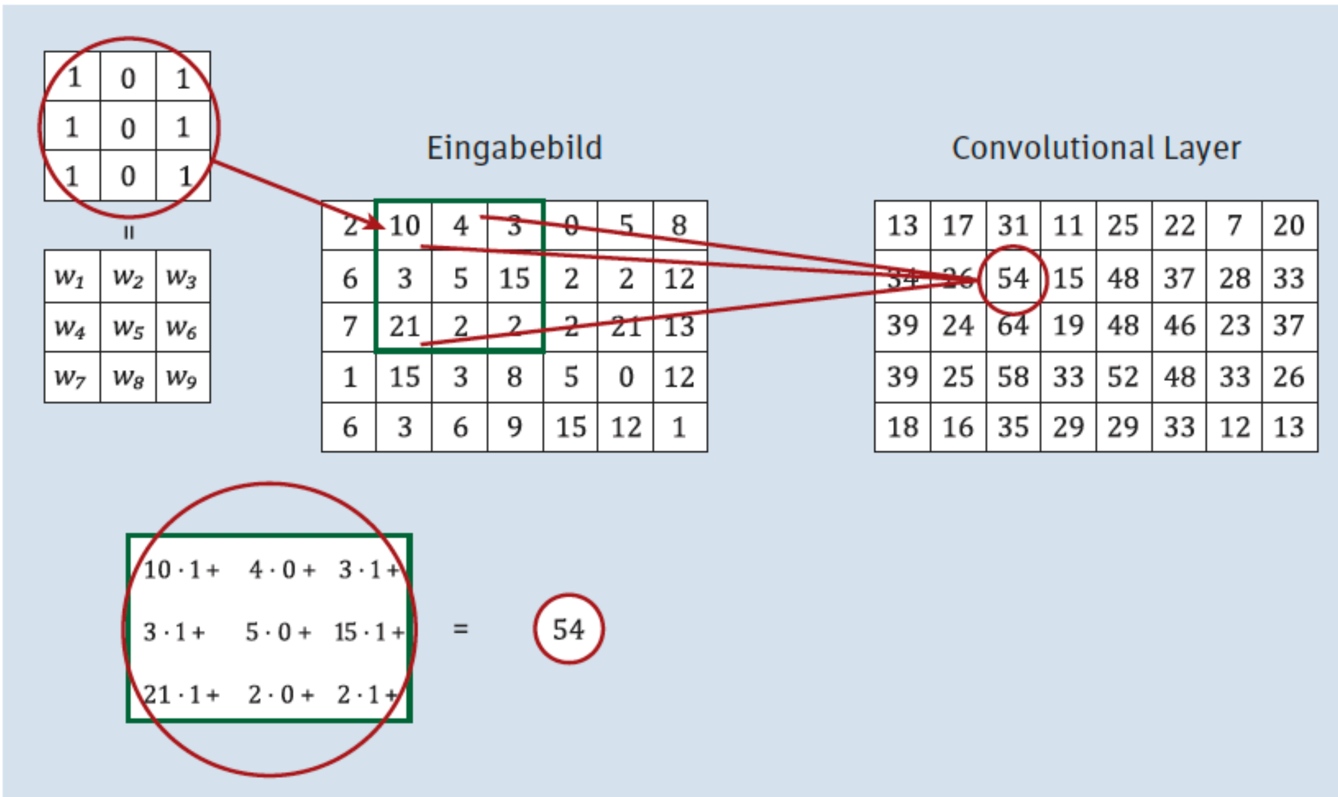


Abbildung 7.3 Eine diskrete Faltung mit einem vertikalen Linienfilter

CNN
Convolutional Layer

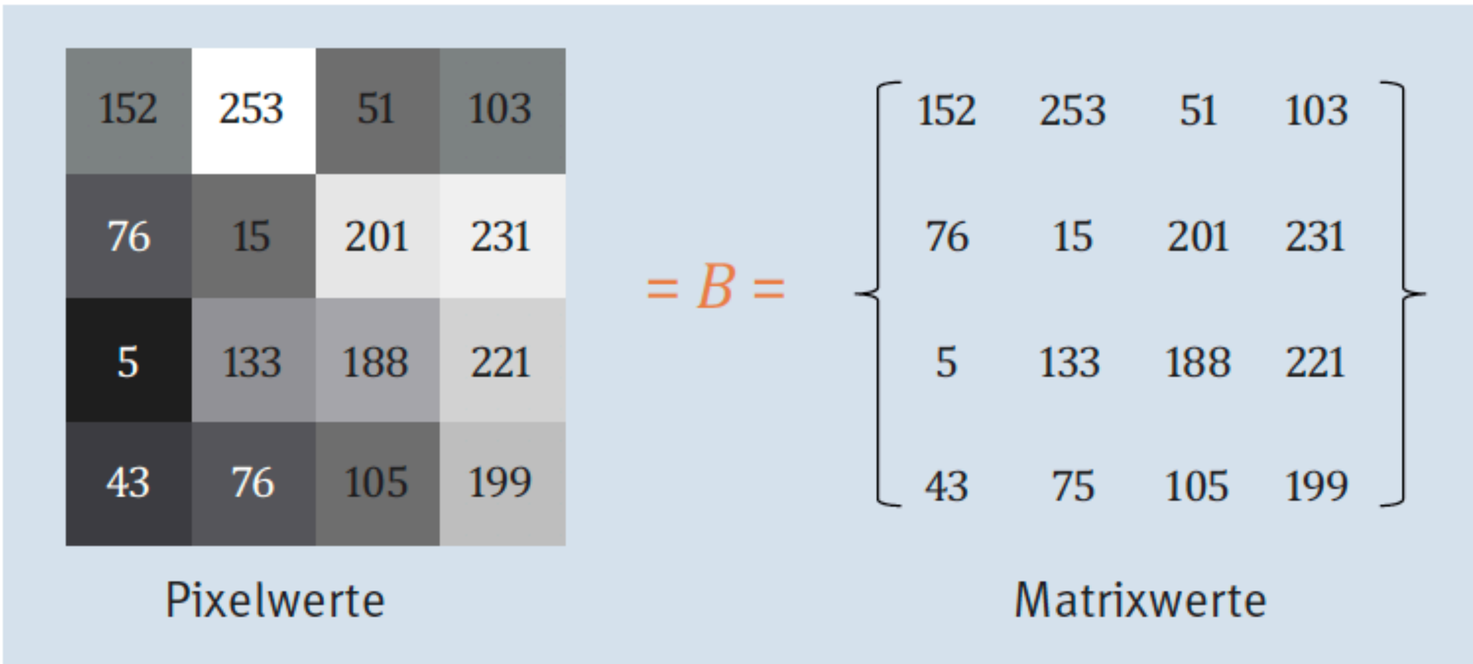


Abbildung 7.4 Vom Bild (Pixelwerte) zur Matrix

CNN

- Erster Convolutional Layer, jedes Neuron reagieren nur auf bestimmtes Muster, zB horizontale Linie

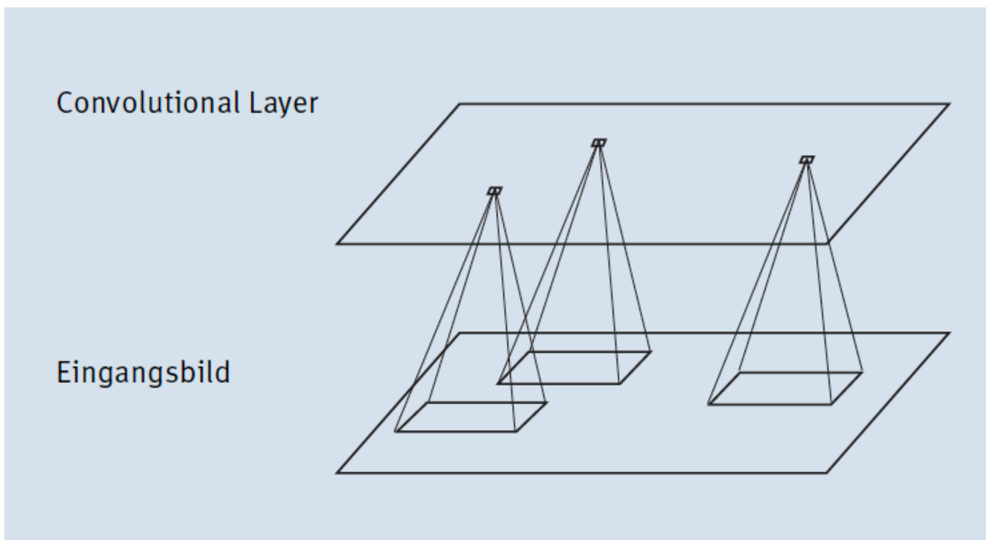


Abbildung 7.5 Der erste Convolutional Layer

CNN

- Im ersten Convolutional Layer wird pro Feature eine Ebene verwendet = Feature Maps

Beachten Sie, dass die Gewichte für alle Neuronen einer Ebene des Convolutional Layers gleich sind. Beim Training werden daher für eine Feature Map nur diese Gewichte trainiert.

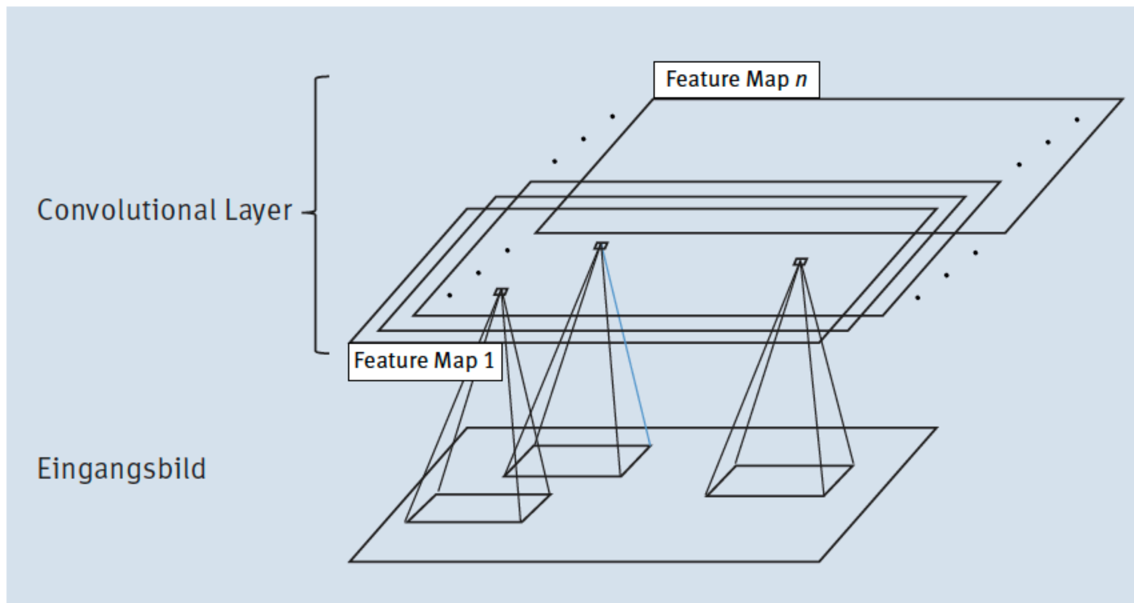


Abbildung 7.6 Ein Convolutional Layer besteht aus mehreren Feature Maps.

CNN
Pooling Layer

- Die Berechtigung für das Pooling stammt aus der Neurobiologie und dem Begriff der lateralen Hemmung, der ein Verschaltungsprinzip von Nervenzellen beschreibt, bei dem eine aktive Nervenzelle die Aktivität der benachbarten Zellen hemmt.
- Das Ziel des Poolings ist eine Reduktion der Dimensionalität

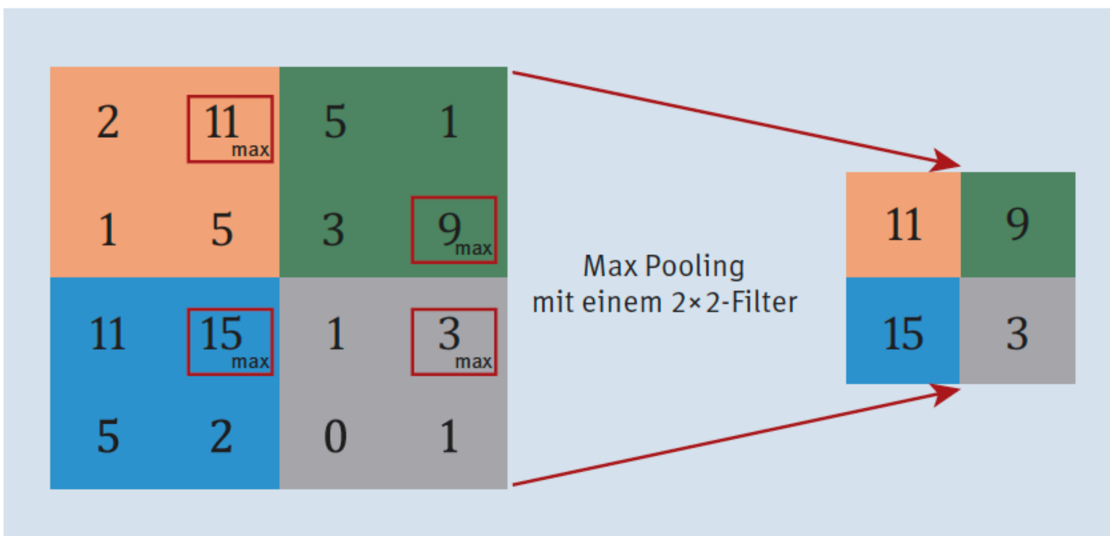


Abbildung 7.8 Beispiel für ein Max Pooling mit einem 2×2-Filter

Nicht überlappend!

Keine Gewichte

Es gibt auch Average Pooling

CNN
Hyper Parameter - Padding

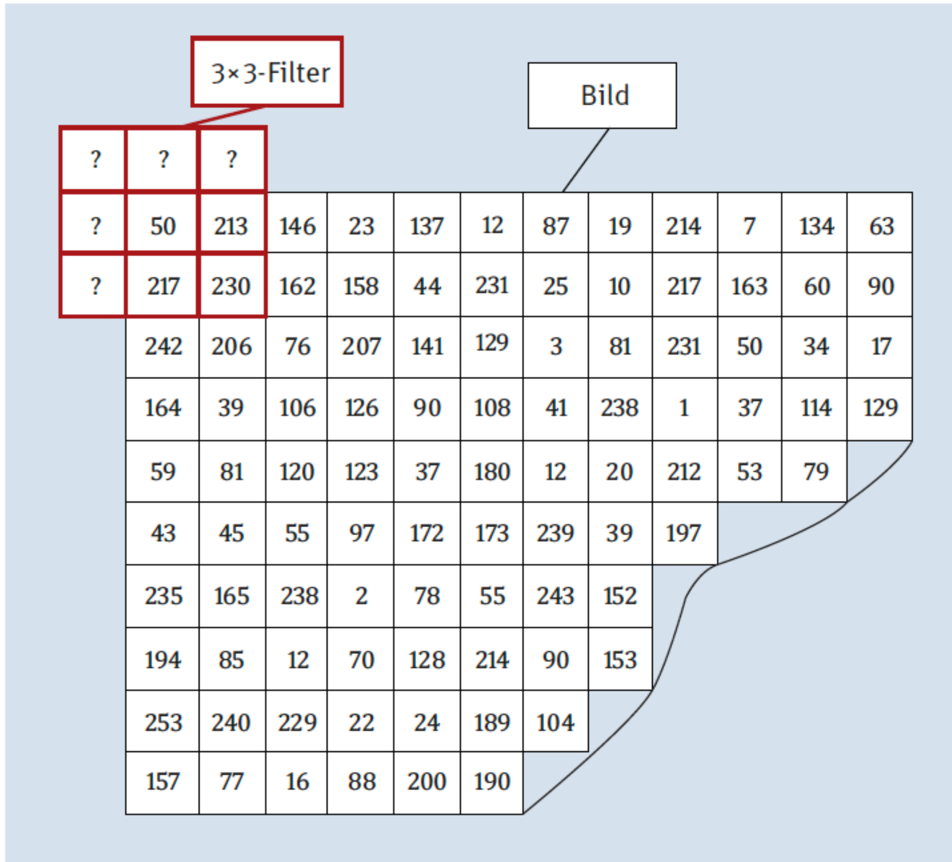
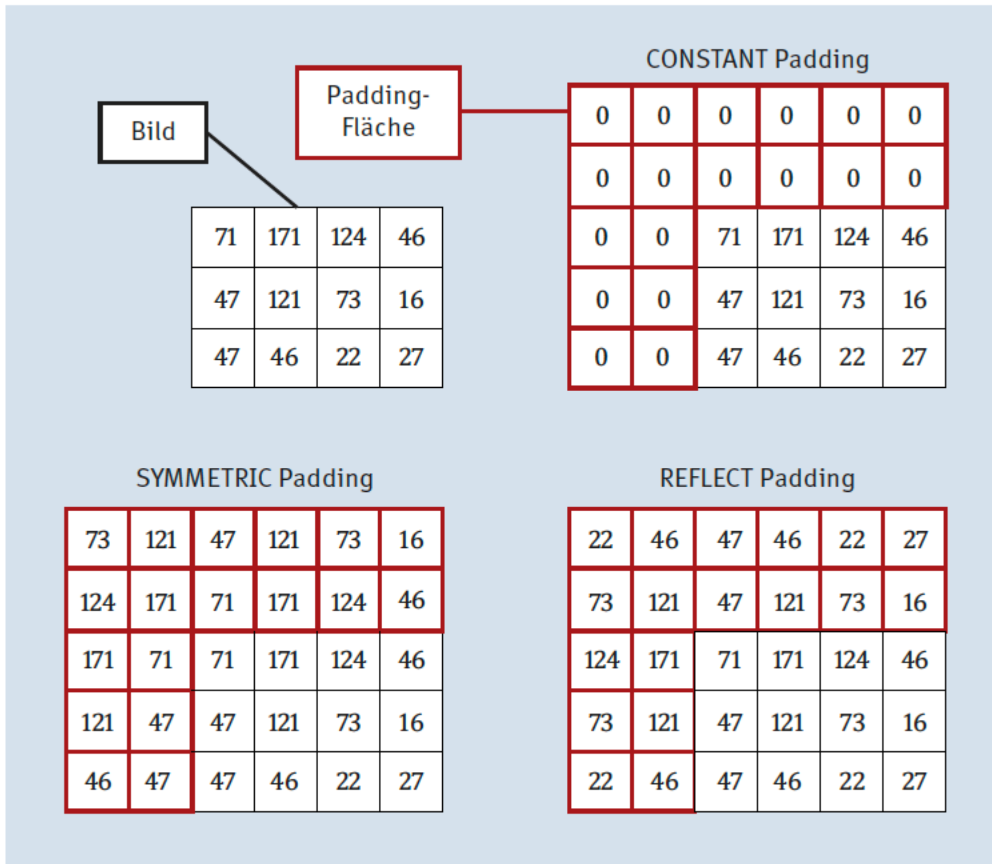


Abbildung 7.9 Filter – was tun mit Randpixeln?

CNN
Hyper Parameter - Padding



Zero Padding

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

CNN
Hyper Parameter - Stride

	115	142	13	18	40	156	5	106	21	135	87
	116	70	82	89	243	247	232	24	36	43	208
Stride = 1 (mit Überlappen)	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
	180	84	71	0	63	89	190	75	186	207	121
Stride = 2 (mit Überlappen)	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
	213	131	58	69	236	152	53	129	226	13	218
Stride = 3 (mit Überlappen)	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

CNN
Identifikationsteil - Flatten

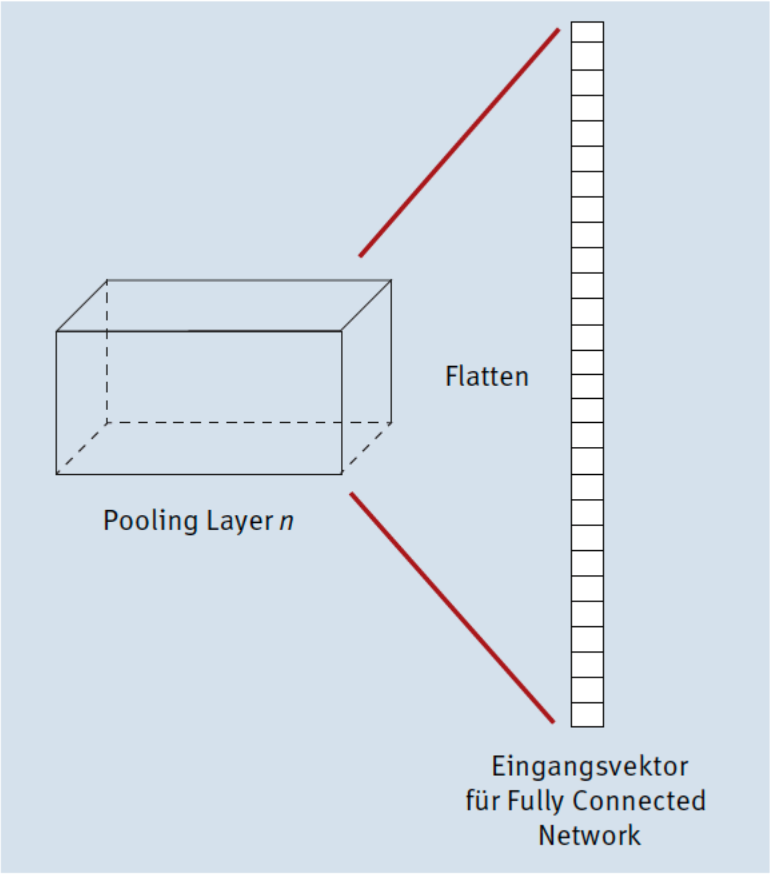


Abbildung 7.12 Flattening des letzten Blocks im Detektionsteil

CNN Identifikationsteil - Softmax

Mit welcher Wahrscheinlichkeit gehört ein Bild zu einer Klasse, und zwar für alle vorgegebenen Klassen?

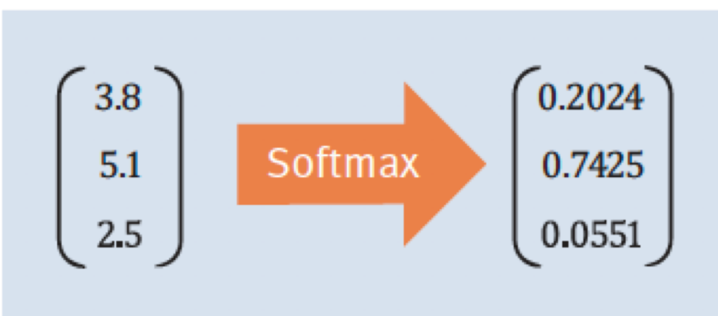


Abbildung 7.13 Beispiel für Softmax für drei Klassen

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

CNN Training

Massiv!

- Herausforderungen
  - Explodierender/verschwindender Gradient
  - Overfitting („Auswendiglernen“)
  - Sättigung („wenig Verbesserung beim Lernen“)

Größe!!!

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

CNN Training - Gradient

- Unangenehmerweise werden die 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.
- 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) und der verwendeten **Aktivierungsfunktion** (bis dahin war die Sigmoidfunktion sehr weit verbreitet) liegen.

CNN
Lernen - Initialisierung

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	$v = \sqrt{2}\sqrt{\frac{6}{n_{\text{ein}} + n_{\text{aus}}}}$	$\sigma = \sqrt{2}\sqrt{\frac{2}{n_{\text{ein}} + n_{\text{aus}}}}$

Tabelle 7.2 Gleichverteilung und Normalverteilung für die zufällige Initialisierung der Gewichte

Anzahl der Eingabeverbindungen, n_{ein}, und der Ausgabeverbindungen, n_{aus}

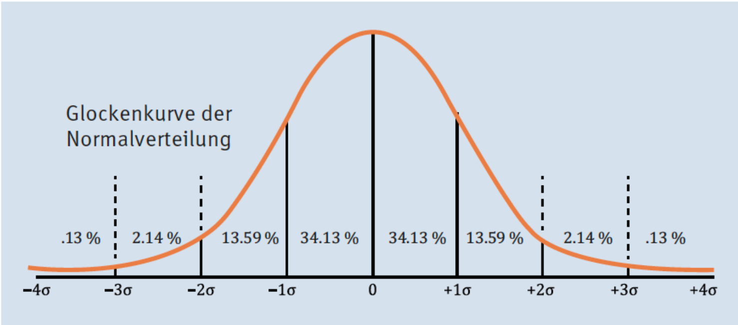


Abbildung 7.14 Typische Normalverteilung

CNN
Training - Aktivierungsfunktion

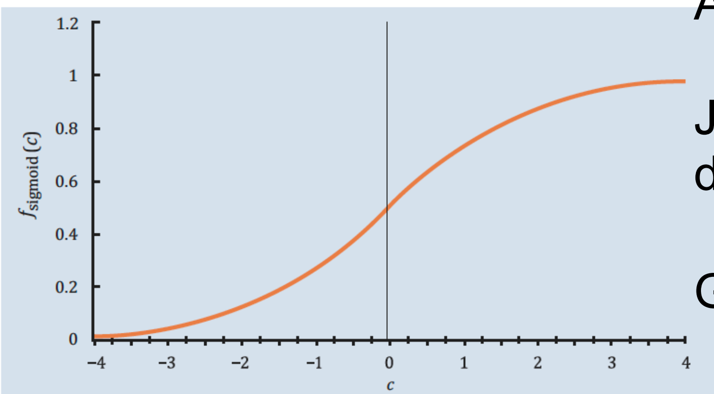


Abbildung 7.15 Sigmoidfunktion

Ableitung bei großen Werten sehr klein

Je näher zur Eingabeschicht,
desto kleiner die Änderung

Große Werte -> Sättigung

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

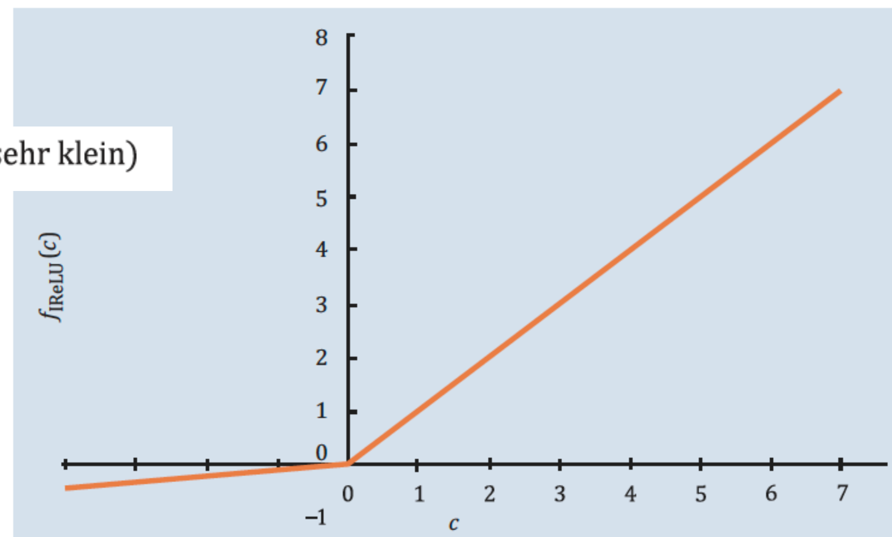


Abbildung 7.16 Die Leaky-ReLU-Aktivierungsfunktion

CNN
Training - Overfitting

Aufgrund der großen Anzahl an Parametern kann ein weiteres Problem auftreten, nämlich dass das Netz genau auf meinen Trainingsdatensatz trainiert wird.

Wird das trainierte Netz dann auf neue unbekannte Datensätze angewendet, versagt es. Hier spricht man auch von **Overfitting**, was im Gegenzug bedeutet, dass das Netz keine Generalisierungsfähigkeit besitzt.

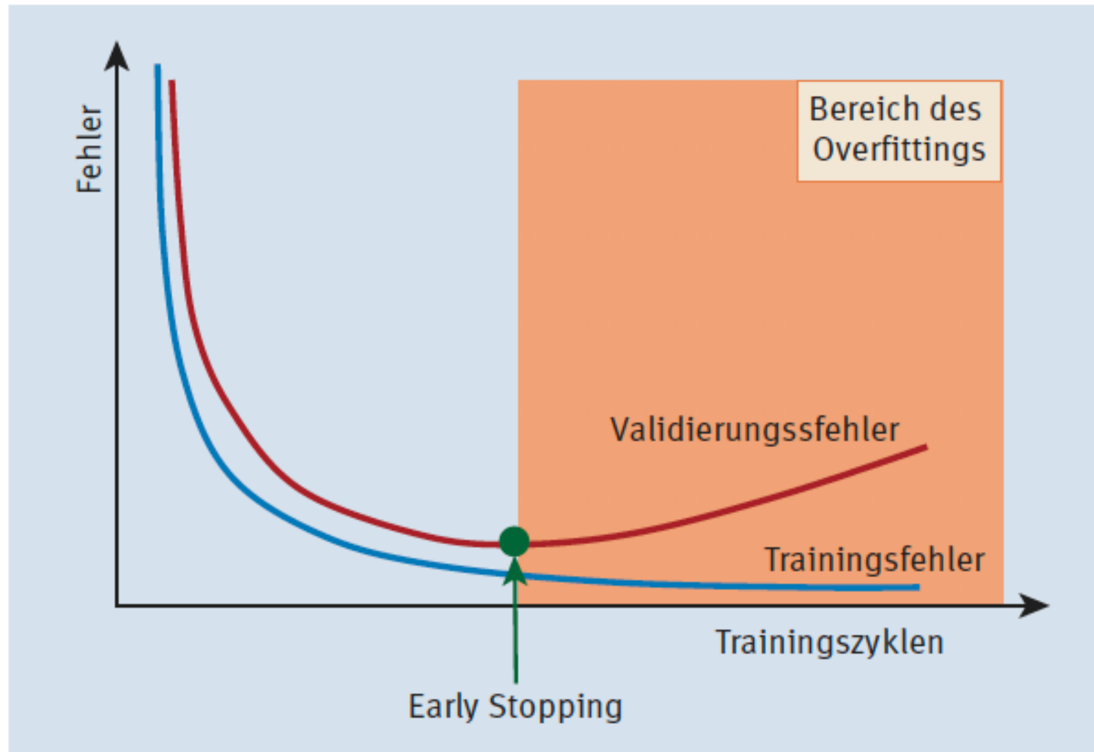


Abbildung 7.18 Fehlerkurve bei Early Stopping

CNN
Training - Dropout

Bei dieser Methode lässt man in jedem Trainingszyklus zufällig einen gewissen Teil der Neuronen nicht mitspielen, das heißt, sie werden für das Gewichtsupdate einfach nicht berücksichtigt.

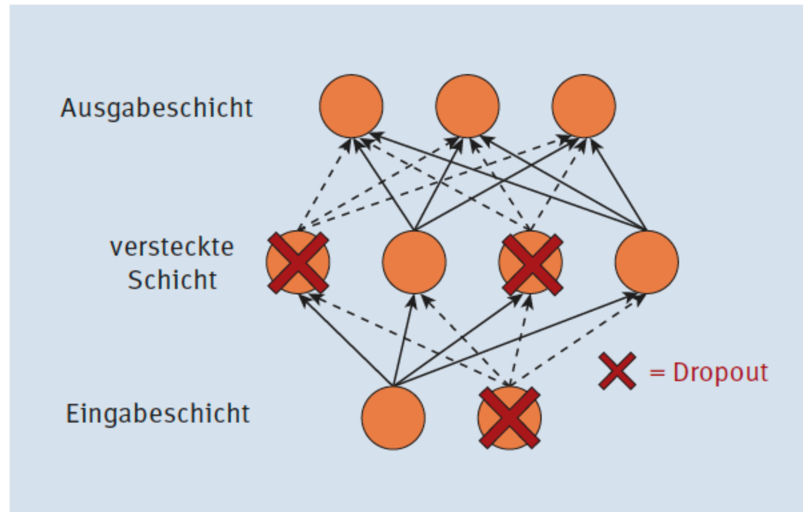


Abbildung 7.19 Netz mit Dropout-Neuronen (Dropout-Rate p = 50 %)

Kapitel 8
CNN mit TF

CNN mit TF

- TF einrichten, d.h. neues Environment

• Jeder TensorFlow-Code enthält zwei wesentliche Elemente:
  1. Baue einen (Berechnungs-)Graphen, der den Datenfluss der Berechnungen repräsentiert.
  2. Führe eine Session aus, die die Berechnungen aus dem Graphen ausführt und natürlich auch die entsprechenden Ressourcen (Speicher, CPU, GPU) zur Verfügung stellt.

NōR GmbH
empower the evolution of now

SimpleTF

$$f(a,b) = a * b + 2$$

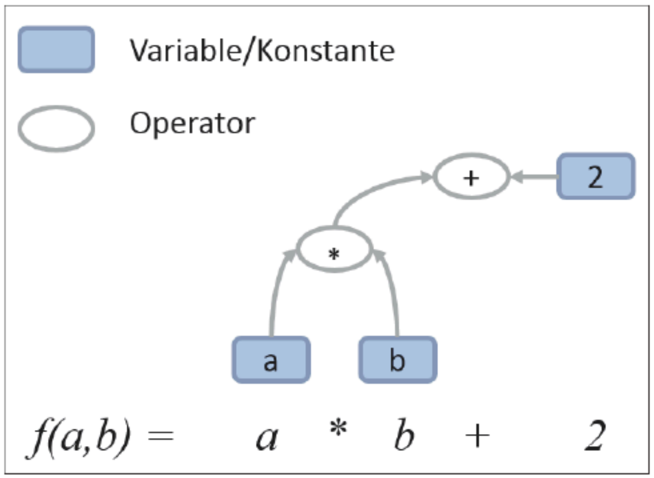


Abbildung 8.1 Berechnungsgraph einer Funktion

```
import tensorflow as tf

a = tf.Variable(10, name = 'a')
b = tf.Variable(4, name = 'b')
two = tf.constant(2)

f = tf.add(tf.multiply(a,b),two)
# f = a*b + 2 - so wäre das auch möglich
```

Listing 8.1 Erstellung eines Berechnungsgraphen in TensorFlow

```
sess = tf.Session()
init = tf.global_variables_initializer()
```

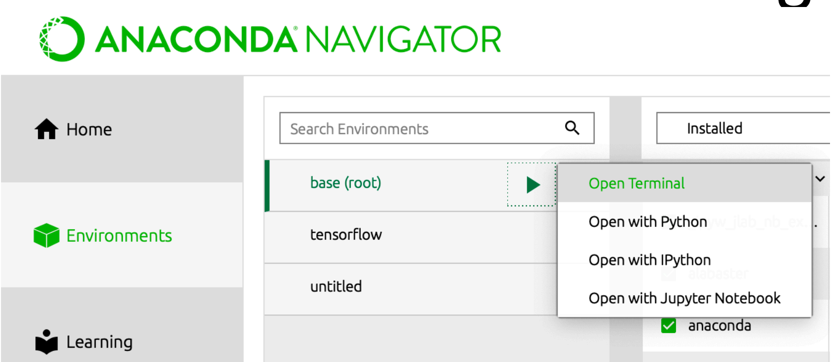
Listing 8.2 Session eröffnen und Variableninitialisierung vorbereiten

```
sess.run(init)
result = sess.run(f)
print(result)
sess.close()
Out[1]:
42
```

Listing 8.3 Ausführung der Session

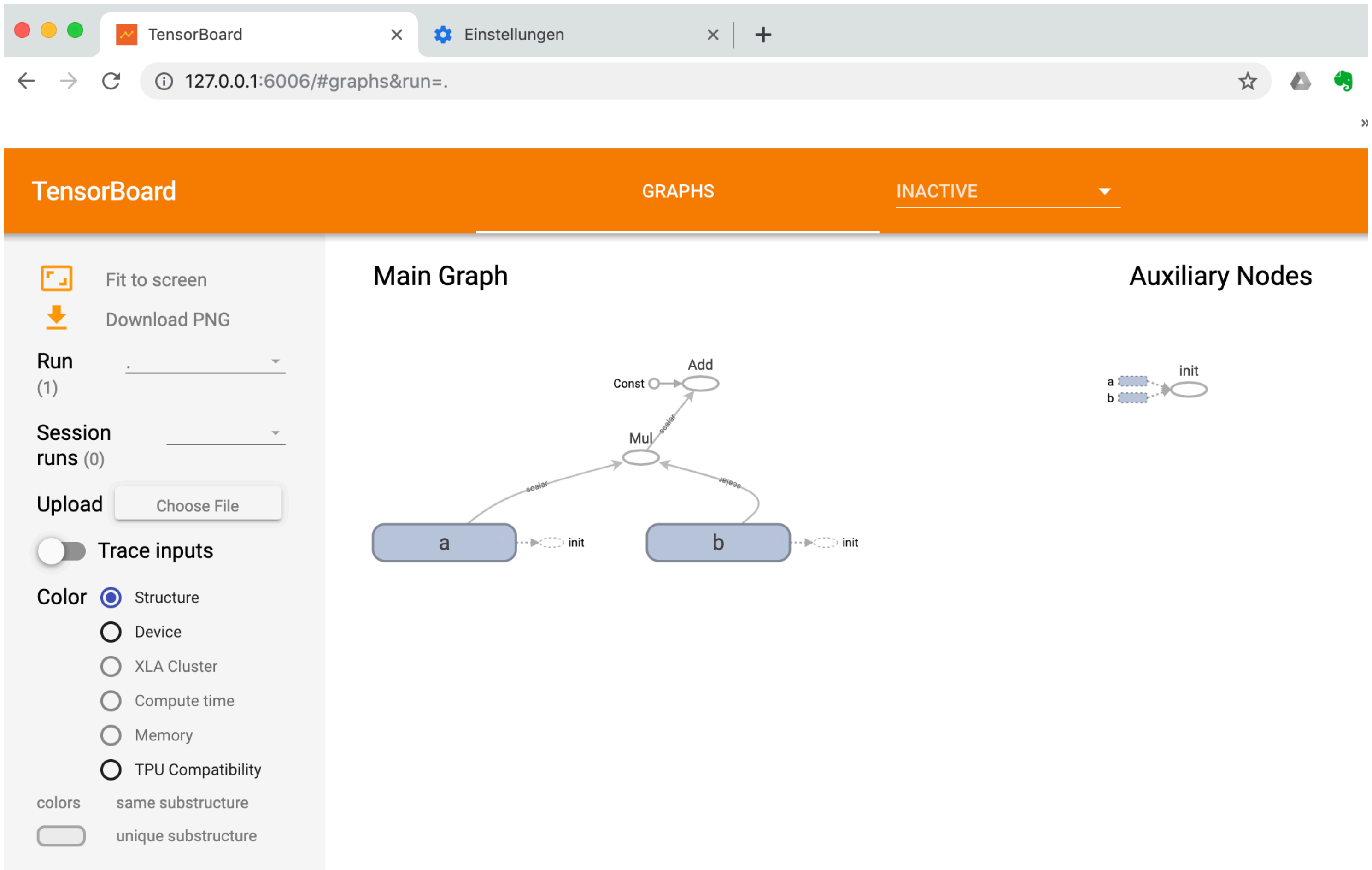
CNN mit TF

1. Starten von TensorBoard
   – Im Anaconda Navigator Terminal öffnen
   – Dann Befehl tensorboard - - logir=„D:/logs“ eingeben
   – URL im Browser eingeben (anstatt PC kann auch 127.0.0.1 eigegeben werden)



```
rolandschwaiger — a.tool — tensorboard --logdir=./logs — 80×24
/Users/rolandschwaiger/anaconda3
(base) bash-3.2$ cd 2019_Psychologie/
(base) bash-3.2$ ls
Bildklassifikation.ipynb        Kapitel 4.ipynb
DiesUndDas.ipynb                Kapitel 5.ipynb
Erträumte Bilder.ipynb          Kapitel 6.ipynb
FeatureEngineering.ipynb        Kapitel13_Features.ipynb
FehlerDebugging.ipynb           Kapitel13_labelonehot.ipynb
HonyBee_BumbleBee.ipynb         SimpleCNN_MNIST.ipynb
KNN.zip                         SimpleTF.ipynb
Kapitel 10.ipynb                Warmup.ipynb
Kapitel 12.ipynb                iris.csv
Kapitel 3.ipynb                 logs
(base) bash-3.2$ tensorboard --logdir="./logs"
/Users/rolandschwaiger/anaconda3/lib/python3.6/importlib/_bootstrap.py:219: Runt
imeWarning: compiletime version 3.5 of module 'tensorflow.python.framework.fast_
tensor_util' does not match runtime version 3.6
  return f(*args, **kwds)
TensorBoard 0.4.0rc3 at http://Rolands-MacBook-Pro-2.local:6006 (Press CTRL+C to
 quit)
W0529 11:08:35.580678 Reloader tf_logging.py:86] Found more than one graph event
 per run, or there was a metagraph containing a graph_def, as well as one or mor
e graph events.  Overwriting the graph with the newest event.
```

CNN mit TF


TensorBoard
Einstellungen
127.0.0.1:6006/#graphs&run=.
TensorBoard
GRAPHS
INACTIVE
Fit to screen
Download PNG
Run
(1)
Session runs (0)
Upload
Choose File
Trace inputs
Color
Structure
Device
XLA Cluster
Compute time
Memory
TPU Compatibility
colors
same substructure
unique substructure
Main Graph
Auxiliary Nodes
Add
Const
Mul
a
b
init

CNN mit TF

SimpleCNN_MNIST

- Der MNIST-Datensatz basiert auf einer modifizierten Datenbank des National Institute of Standards and Technology (NIST)
- Yann LeCun setzte 1998 praktisch den Startpunkt für Convolutional Neural Networks in der Bilderkennung

Abbildung 8.4 Ausschnitt aus dem MNIST-Datensatz

CNN mit TF

📟 SimpleCNN_MNIST

Check the code!

CNN mit TF
Einfaches CNN

SimpleCNN_MNIST

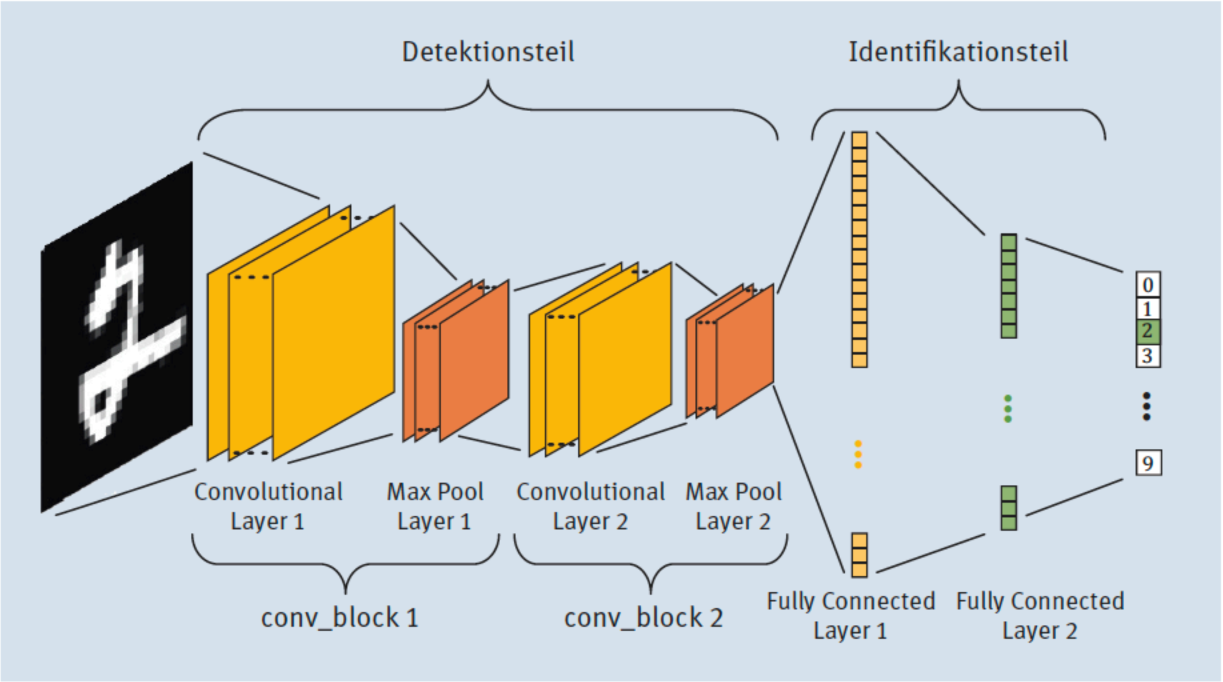


Abbildung 8.6 Die Struktur unseres CNN für den MNIST-Datensatz

CNN mit TF

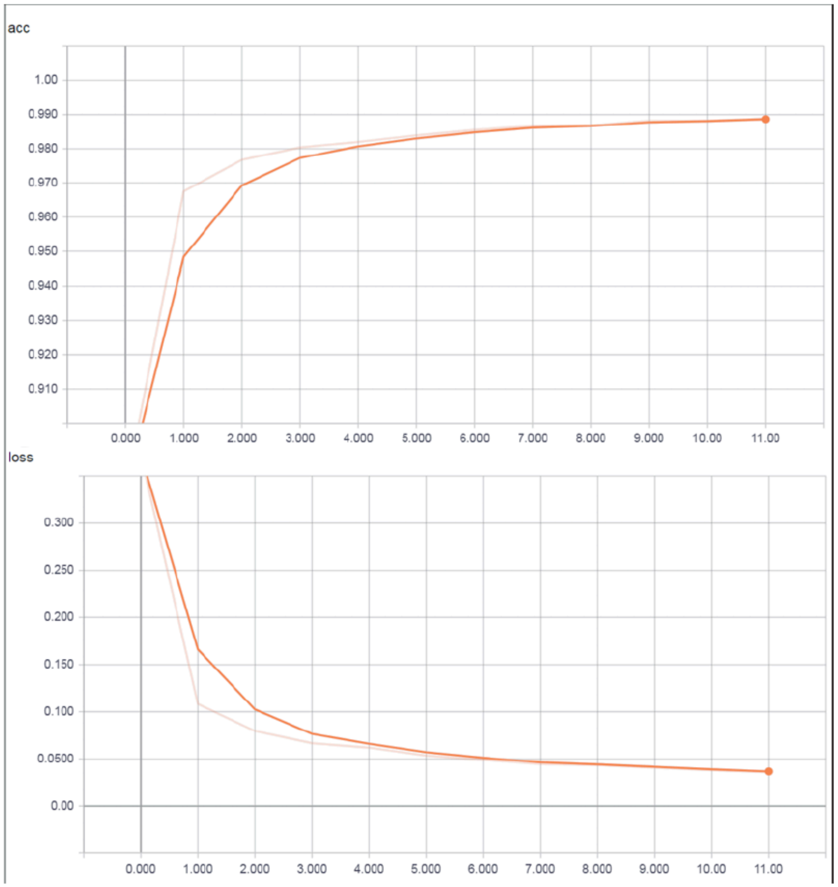
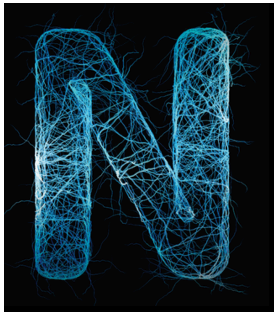


Abbildung 8.9 Verlauf Genauigkeit (»acc«) und Verlust (»loss«) der Trainingsdaten

Abbildung 8.10 Verlauf Genauigkeit (»acc«) und Verlust (»loss«) der Testdaten

Zwischen Perceptron und MLP



Kapitel 10
Evolution der Netze

Evolution der Netze

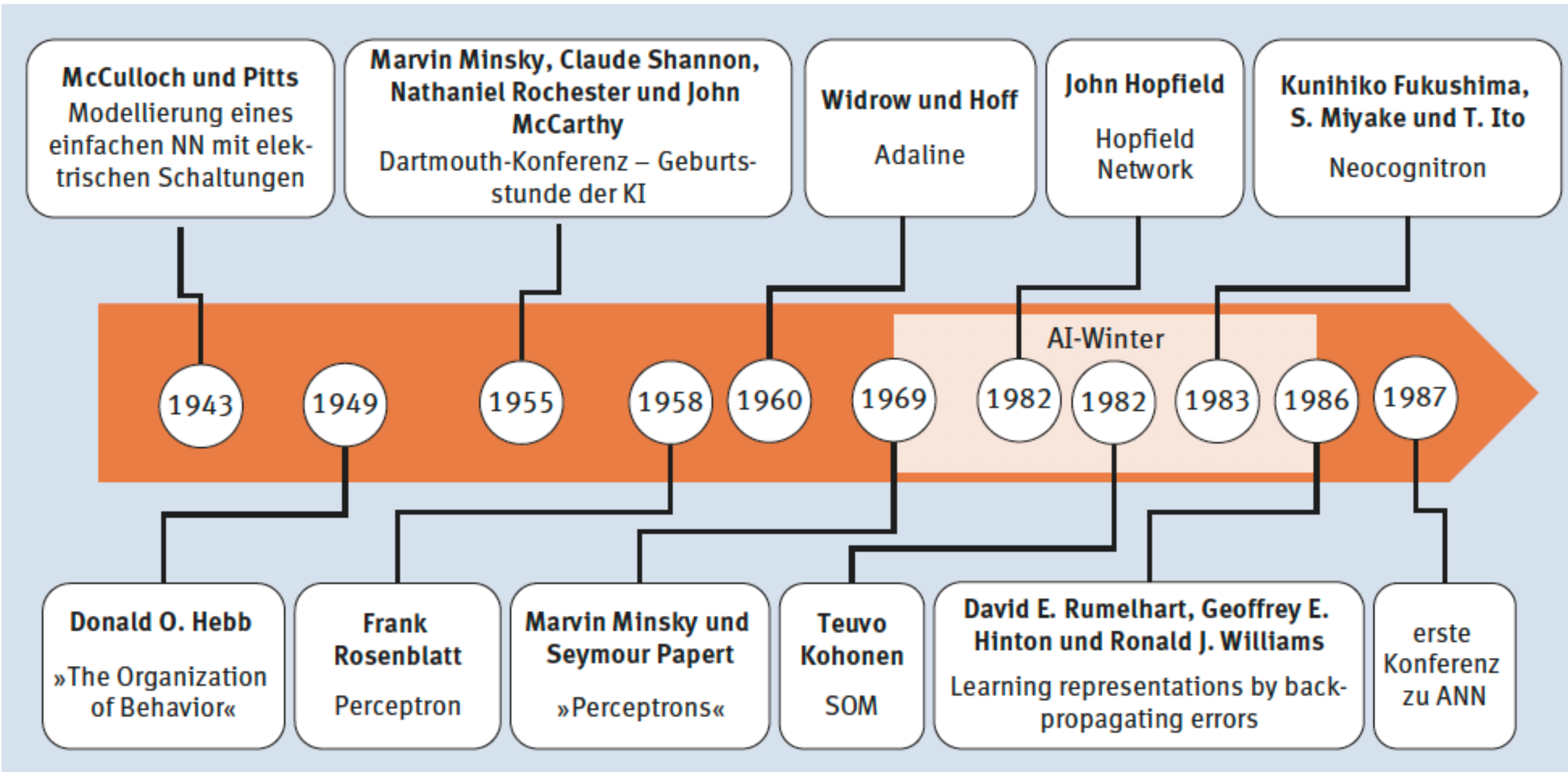


Abbildung 10.1 KNN-Geschichte Teil 1

Evolution der Netze

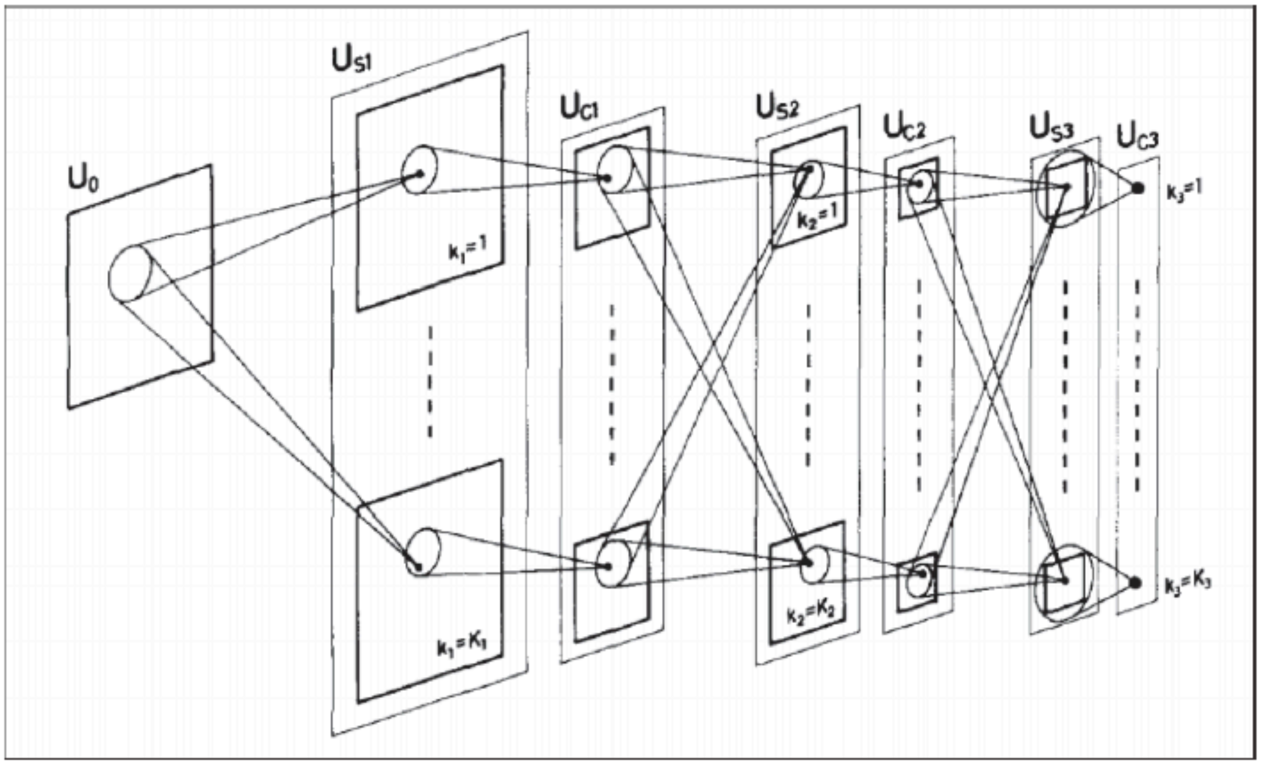


Abbildung 10.5 Schematisches Bild der Verschaltungen im Neocognitron von Fukushima (Quelle: https://www.rctn.org/bruno/public/papers/Fukushima1980.pdf)

Evolution der Netze

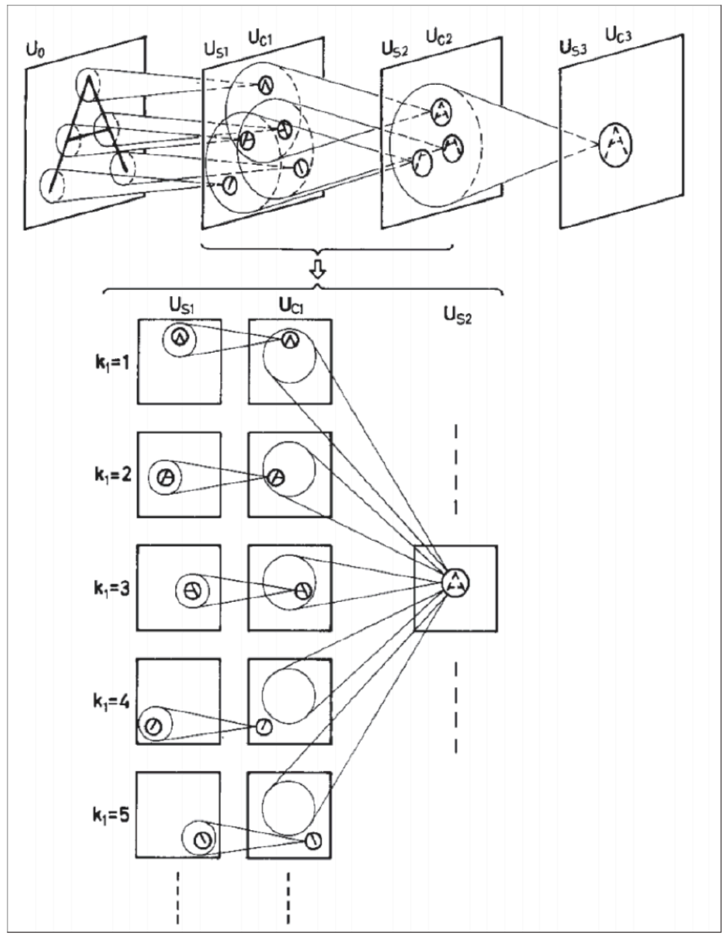


Abbildung 10.6 Neocognitron-Beispiel für die Extraktion von Features[2]

Evolution der Netze
Hopfield Netze

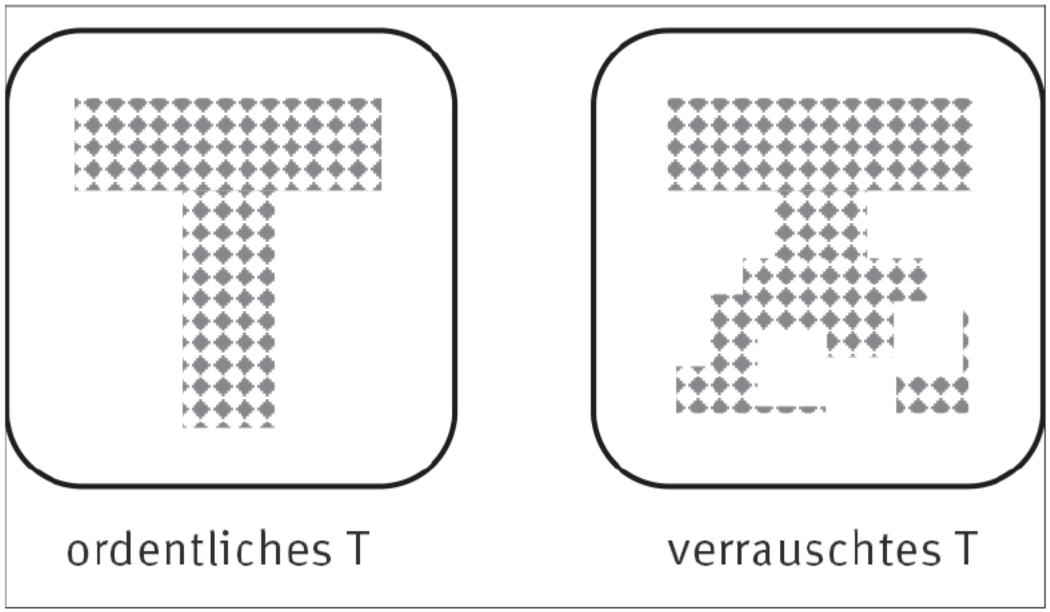


Abbildung 10.8 Assoziativer Speicher

Evolution der Netze

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

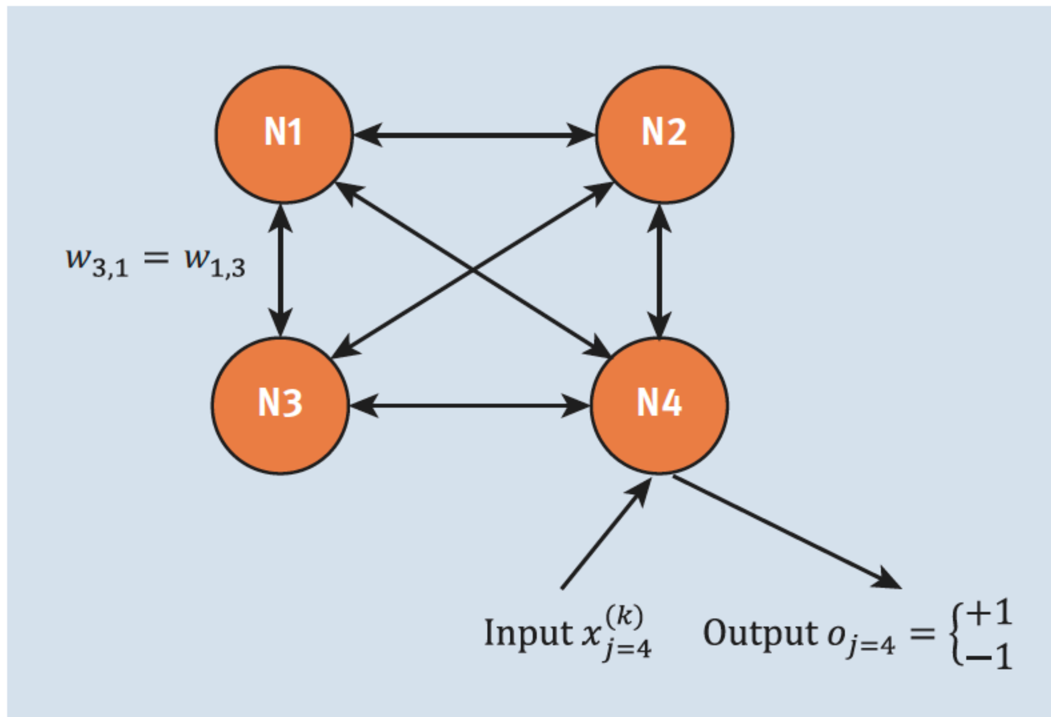
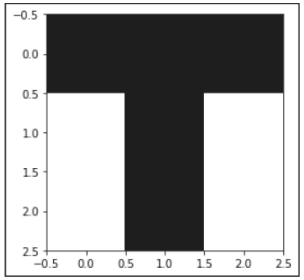


Abbildung 10.9 Beispiel eines binären Hopfield-Netzwerks

Evolution der Netze

Die Ausgabe sieht so aus:

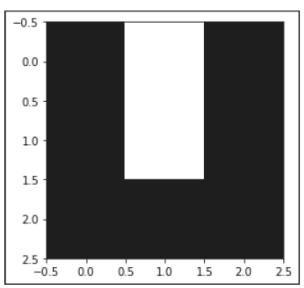
T - Muster



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 - Muster



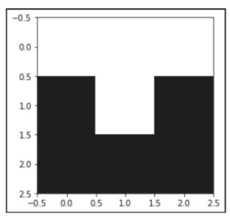
U - Matrix minus Einheitsmatrix

```
[[ 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.]]
```

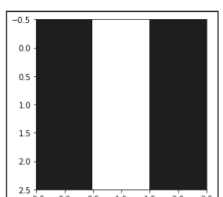
T + U - Matrix

```
[[ 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.]]
```

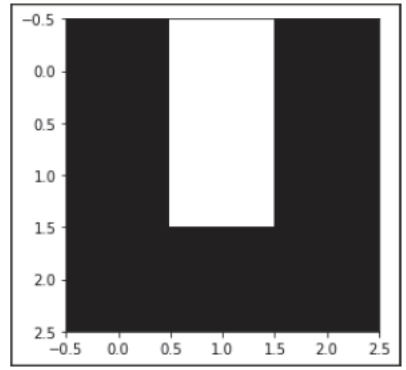
Inputvektor



Schritt 0



Schritt 1



Evolution der Netze

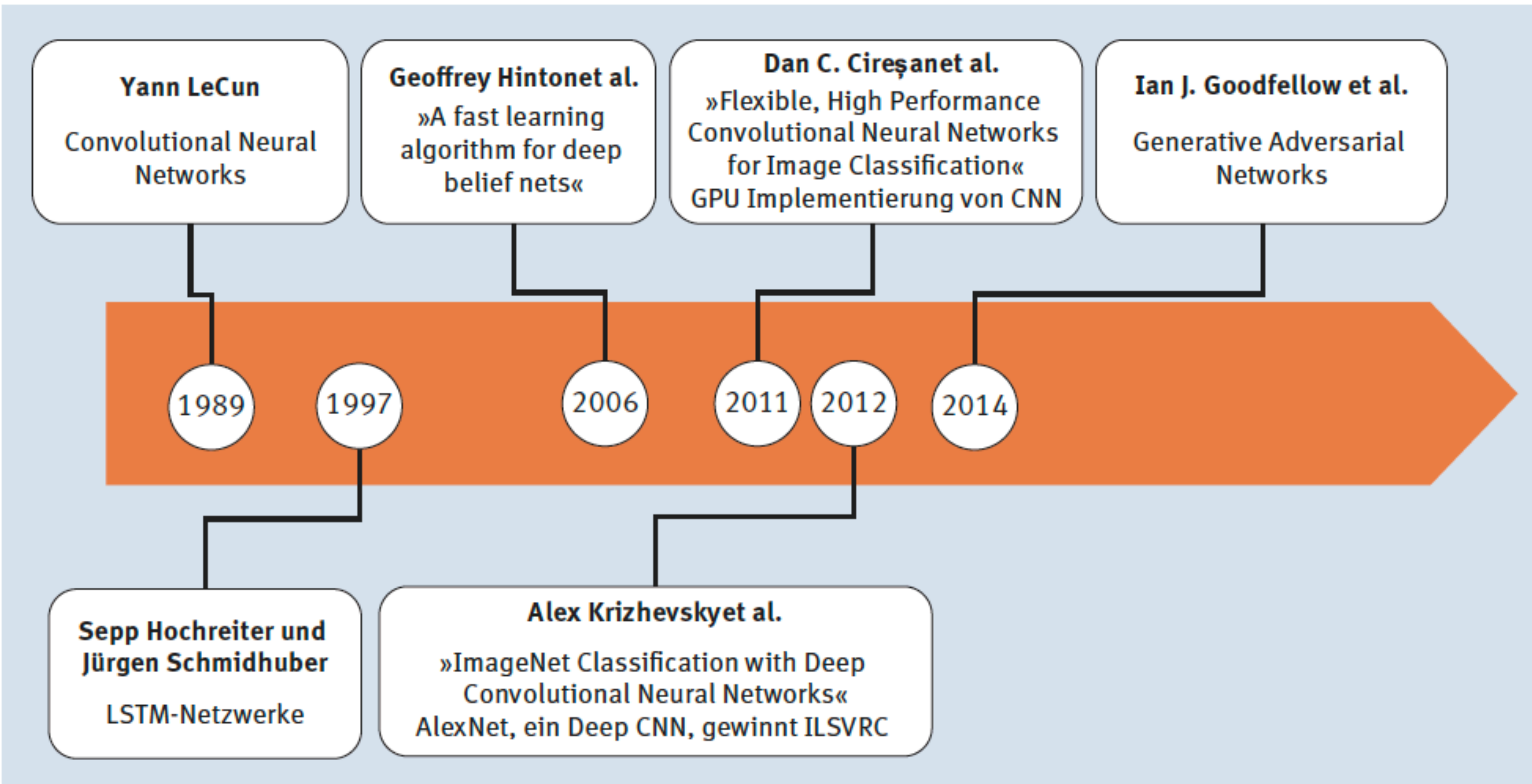


Abbildung 10.11 KNN-Geschichte Teil 2

Evolution der Netze
GAN

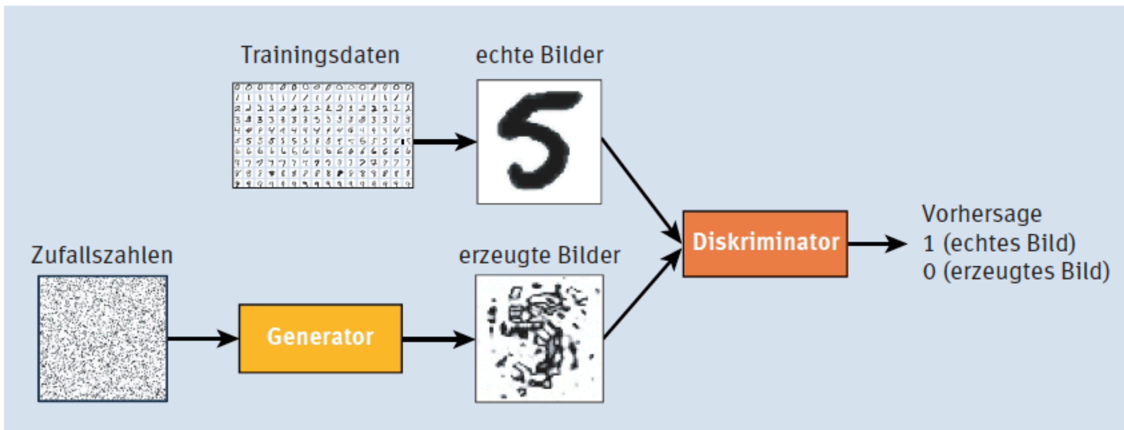


Abbildung 10.12 GAN – prinzipieller Aufbau

Evolution der Netze
GAN

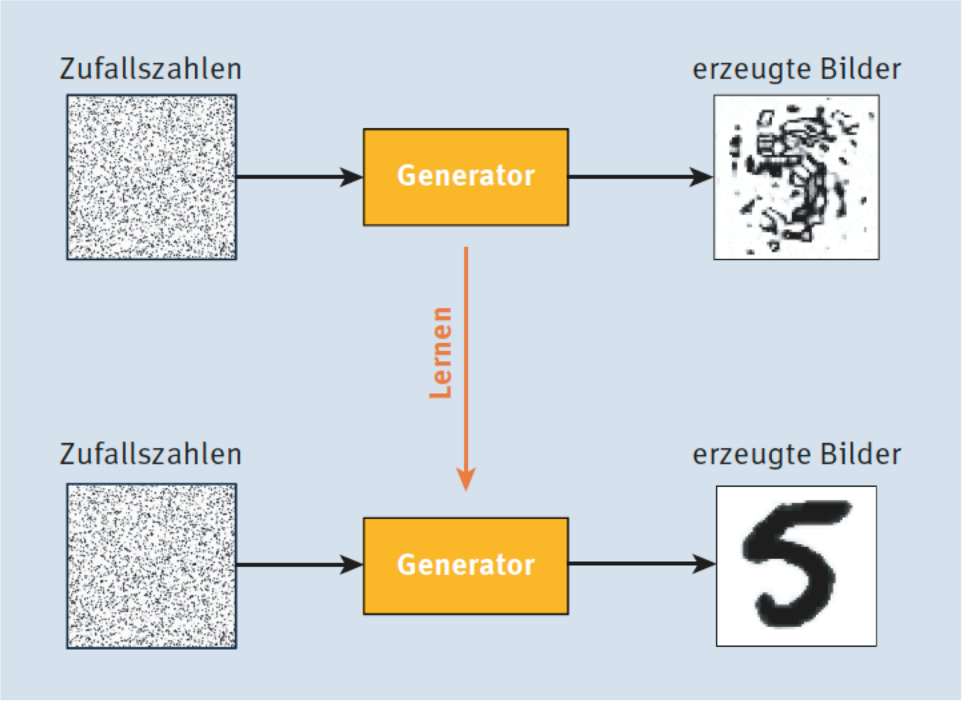


Abbildung 10.13 Der Generator lernt, Bilder zu erzeugen.

Evolution der Netze
GAN

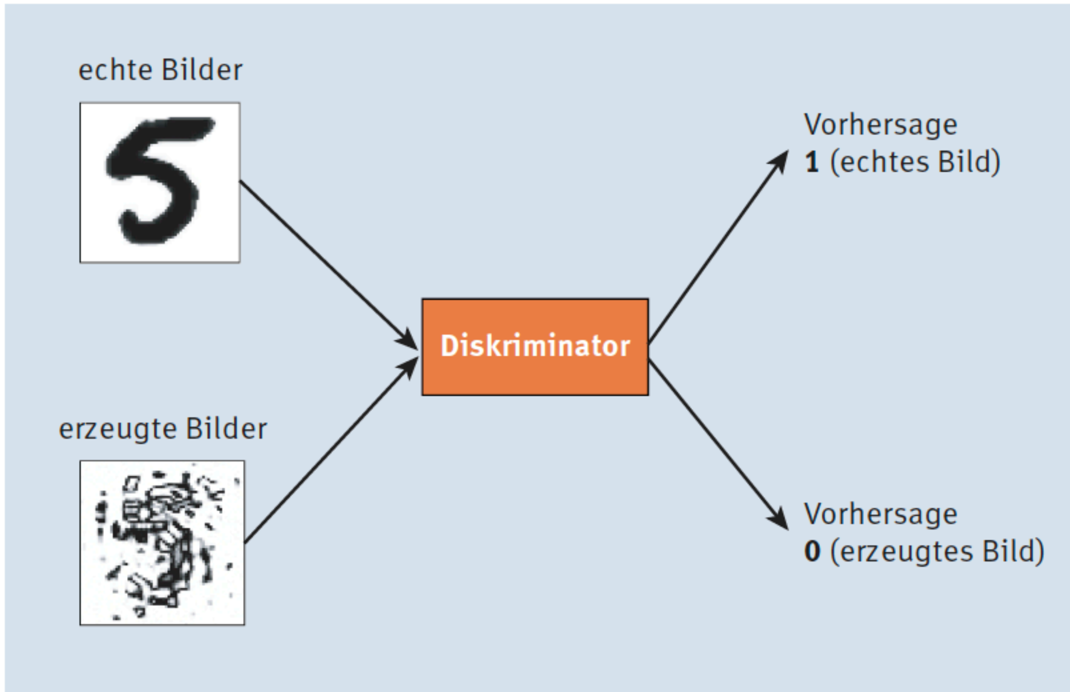


Abbildung 10.14 Der Diskriminator lernt, Bilder zu unterscheiden.

Kapitel 11
ML Prozess

ML Prozess

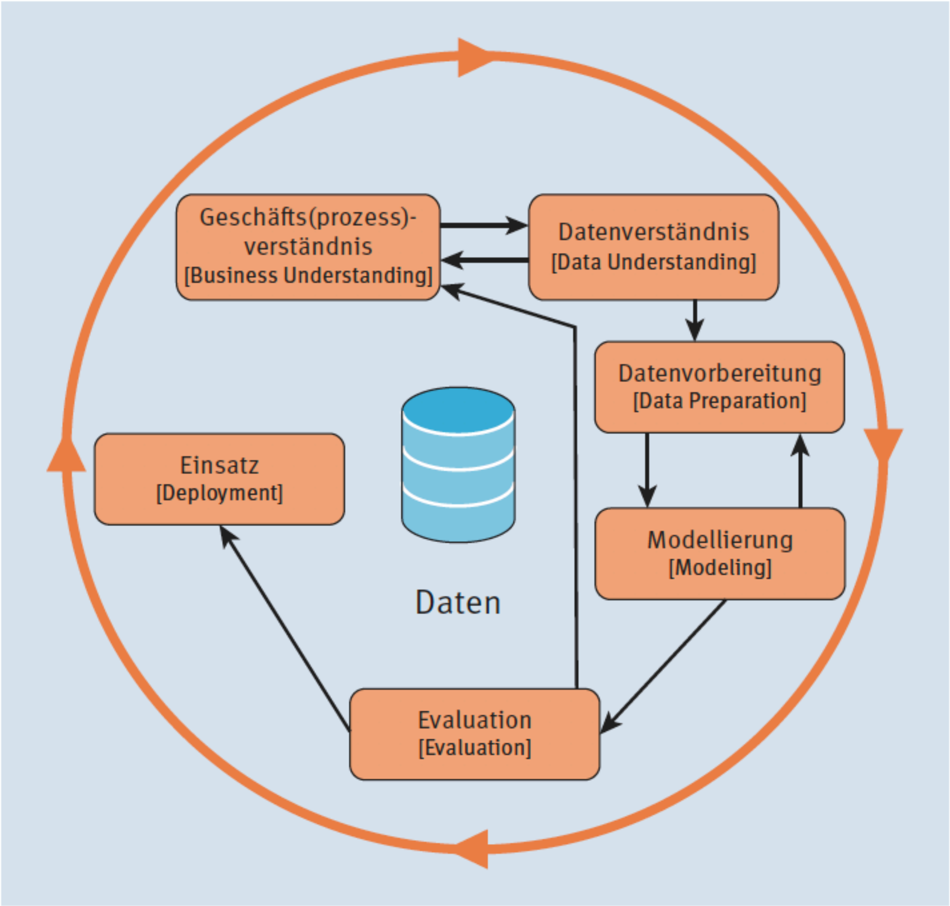


Abbildung 11.1 Das CRISP-DM-Modell (Quelle: Wikimedia Commons)

Kapitel 12
Lernverfahren

Lernverfahren

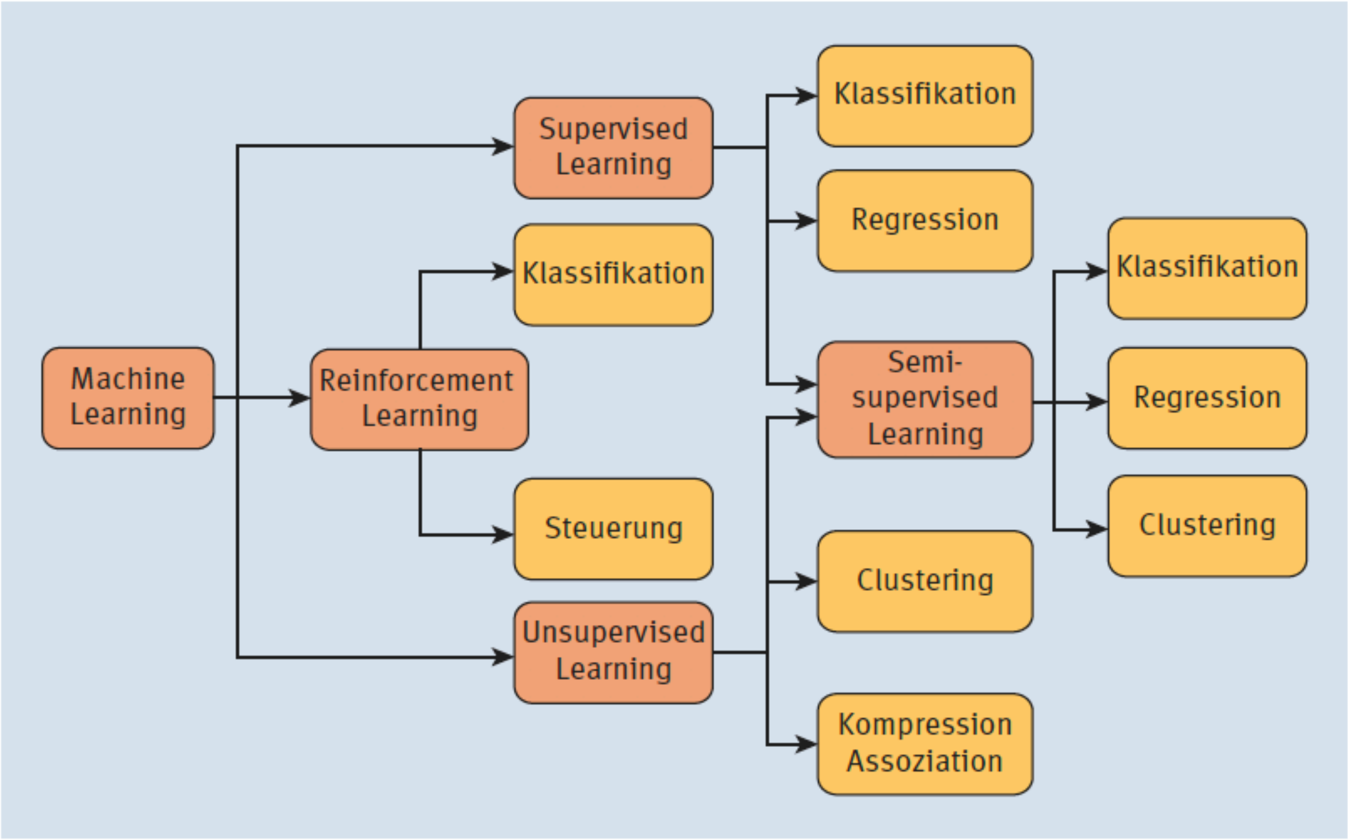


Abbildung 12.1 Lernstrategien und ihre möglichen Anwendungen

Lernverfahren

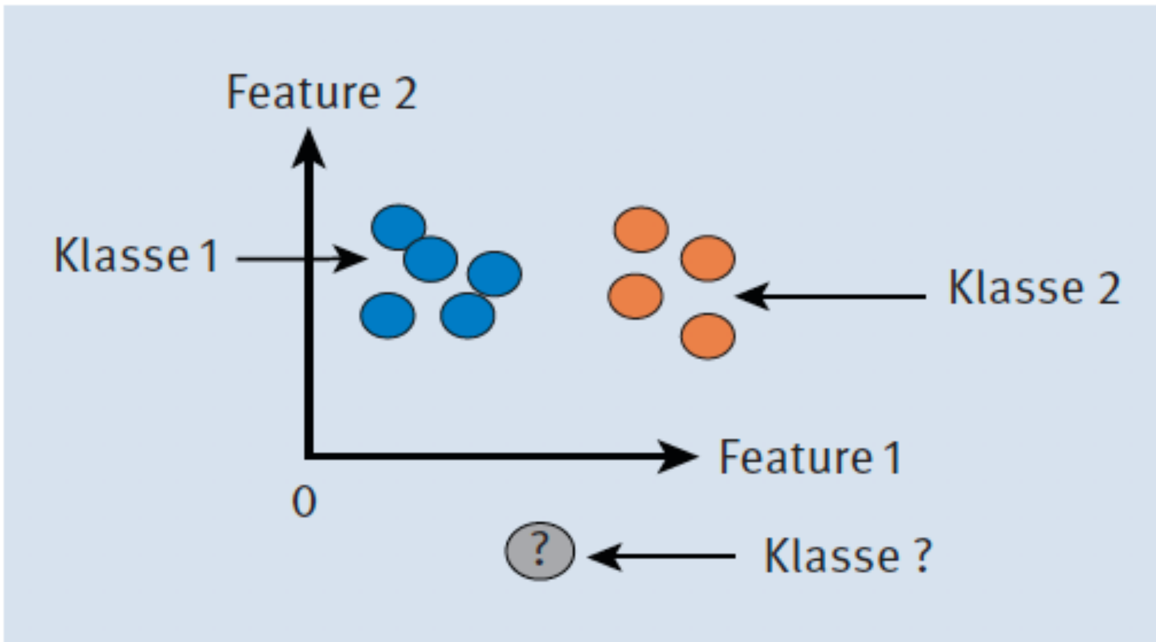


Abbildung 12.2 Klassifikation

Lernverfahren

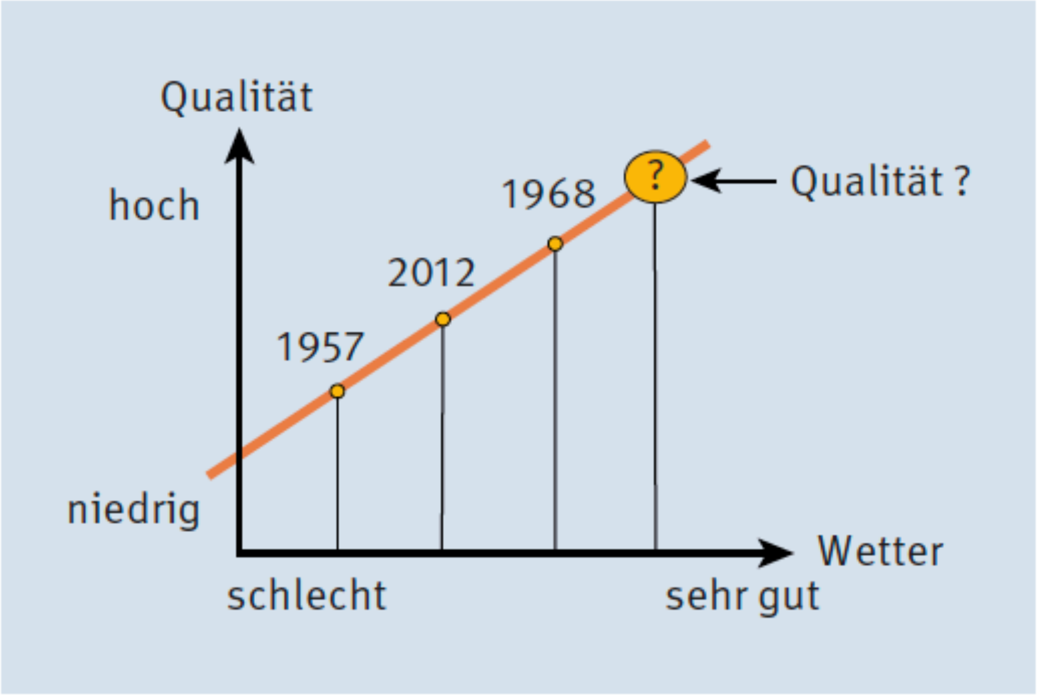


Abbildung 12.3 Wein-Qualitäts-Regression

Lernverfahren

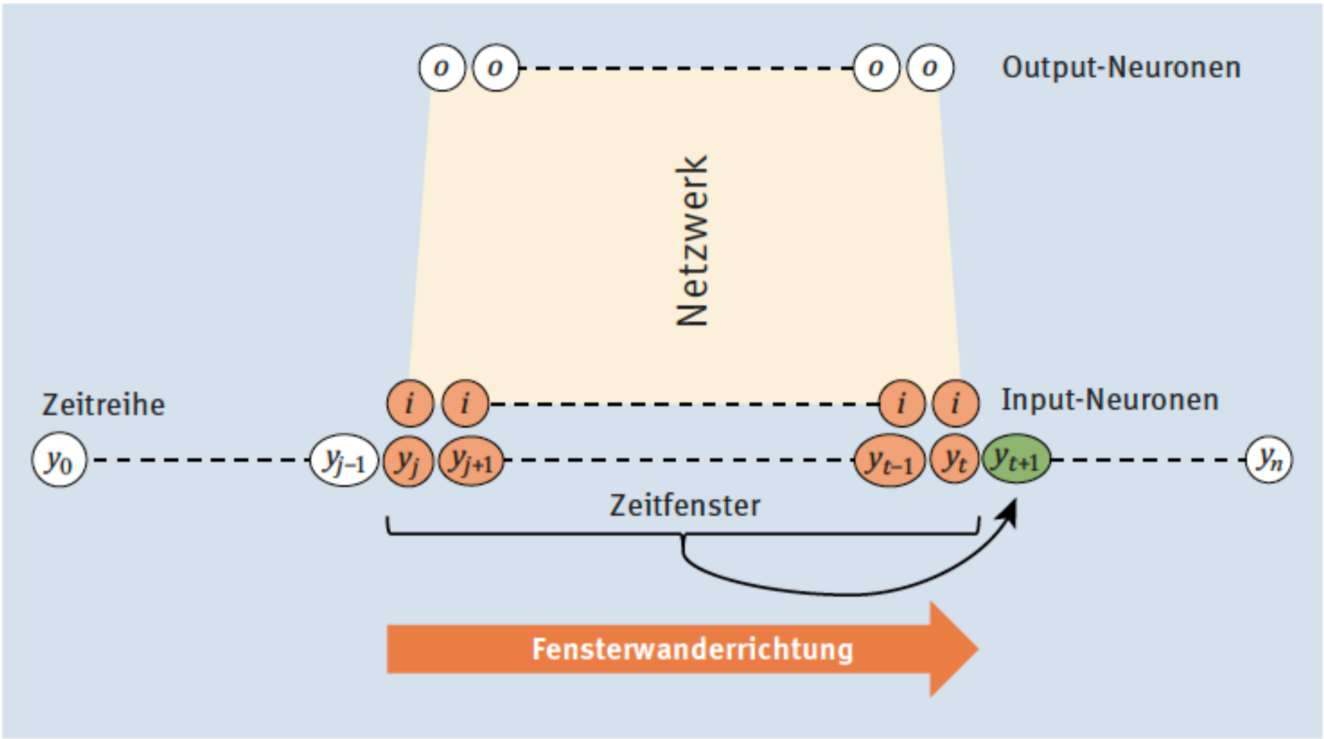


Abbildung 12.5 Zeitreihe mit Netzwerk

Lernverfahren

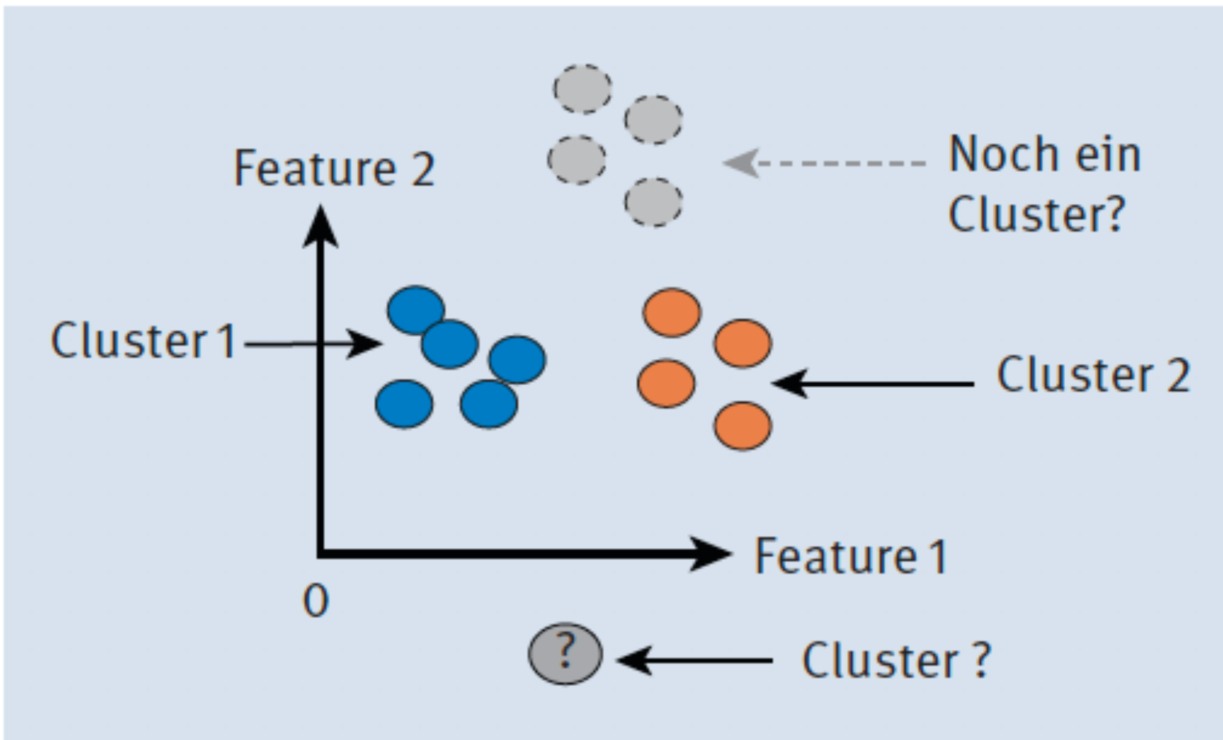


Abbildung 12.6 Cluster

Lernverfahren Kohonen Netze

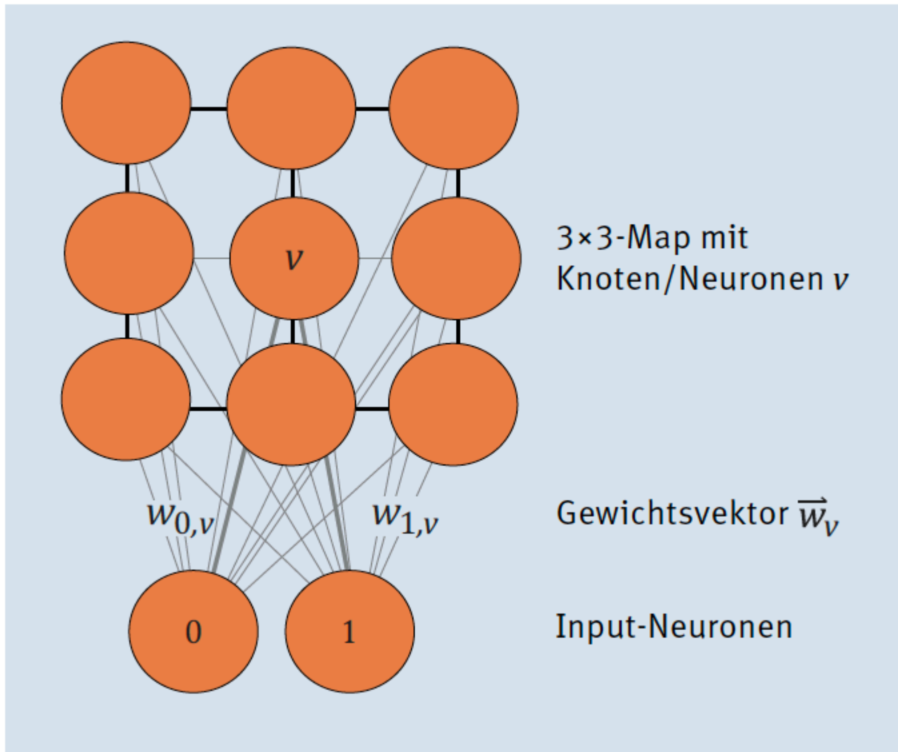


Abbildung 12.7 SOM mit zwei Input-Neuronen und einer 3×3-Map

Lernverfahren

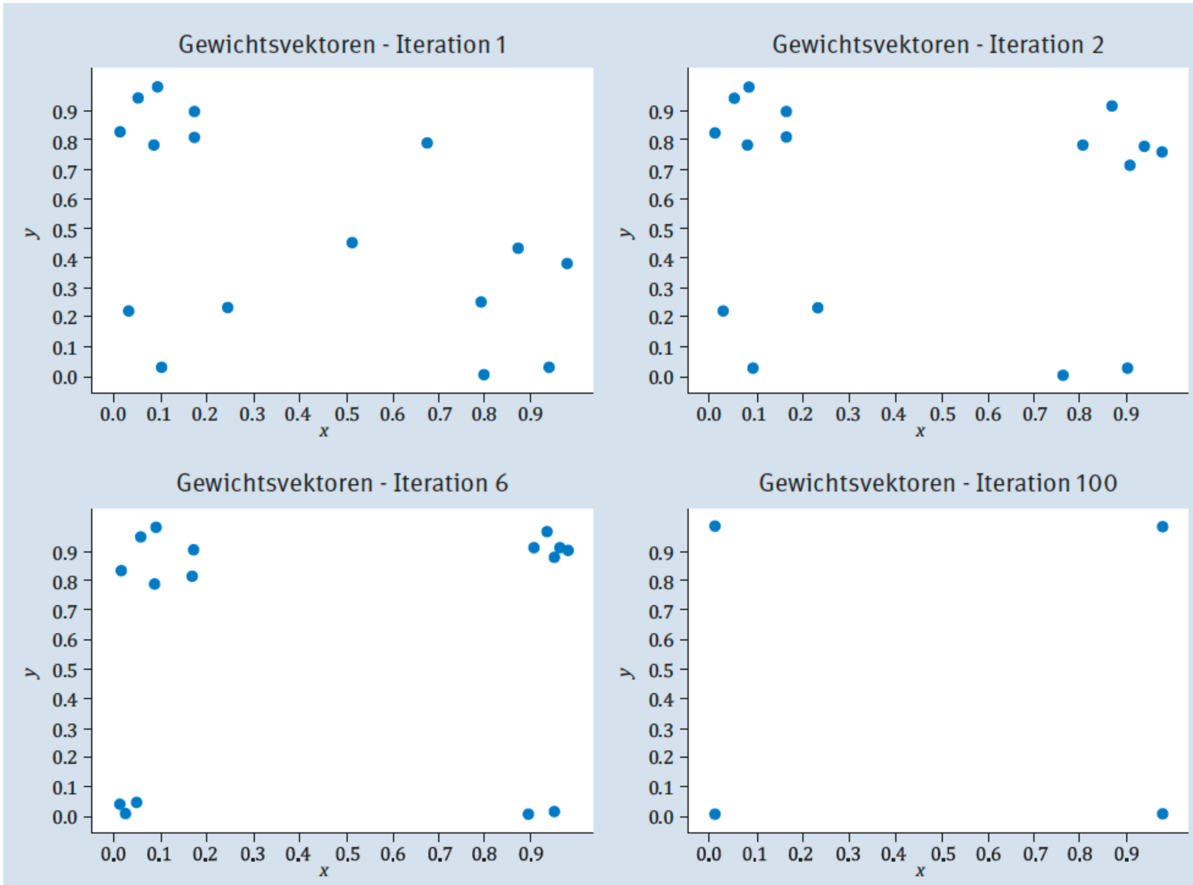


Abbildung 12.14 SOM und die Clusterbildung während des Lernvorgangs

Lernverfahren Reinforcement

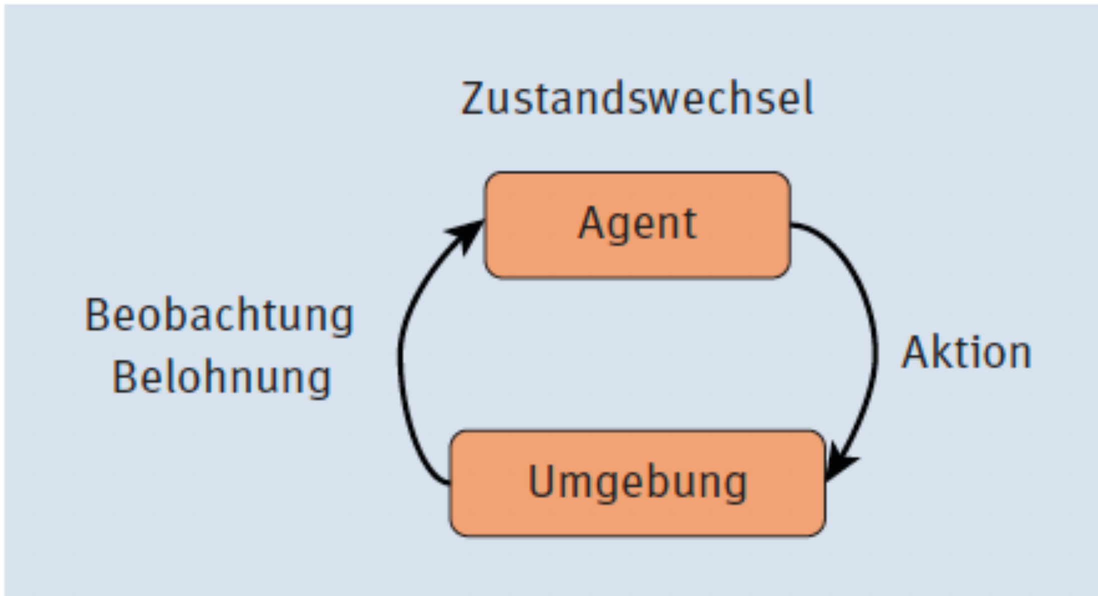


Abbildung 12.15 Reinforcement-System

Lernverfahren

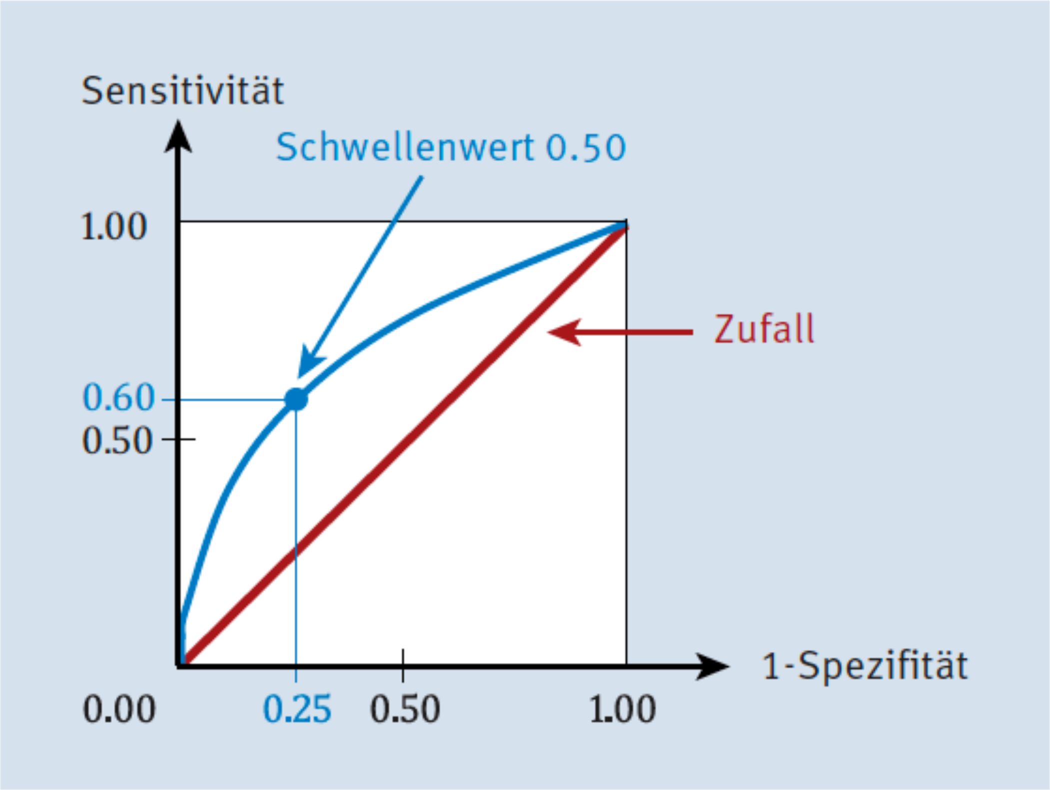


Abbildung 12.25 ROC-Kurve

Lernverfahren

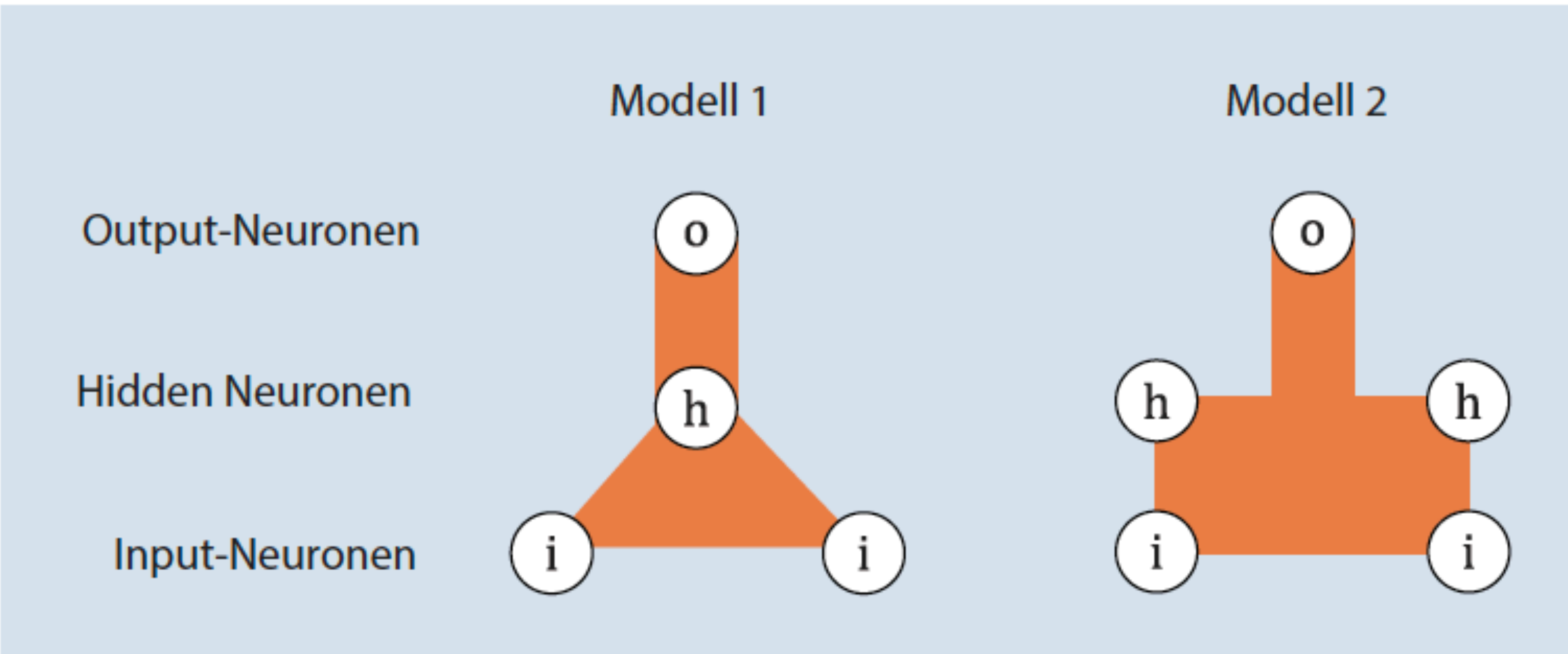


Abbildung 12.26 Modellvergleich

Lernverfahren

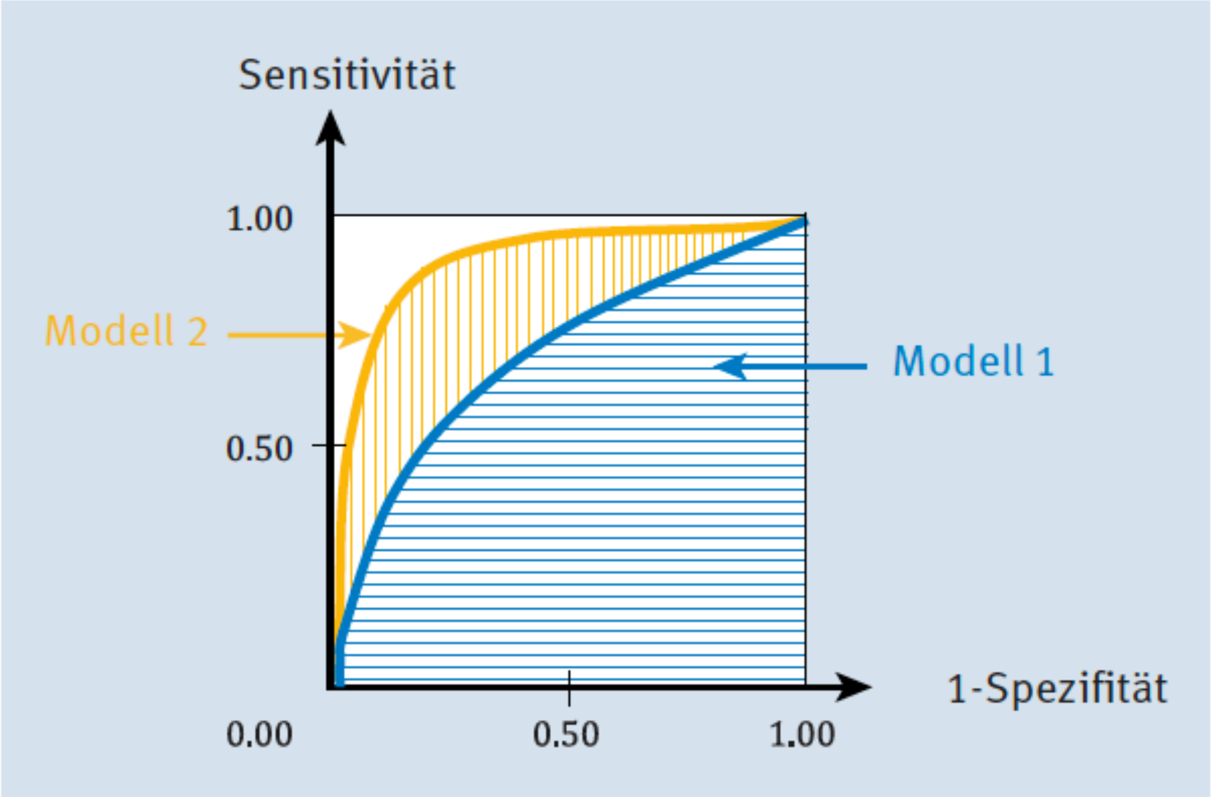
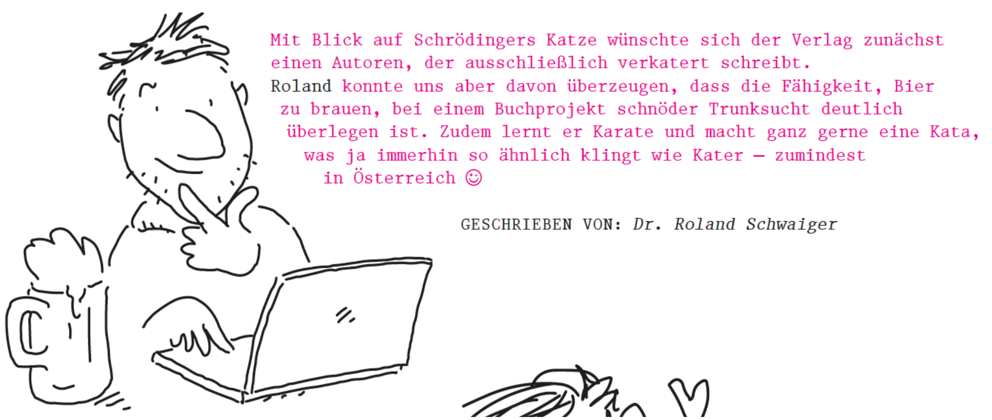


Abbildung 12.27 Zwei ROC-Curves und ihre AUROC

Literaturliste

- www.rolandschwaiger.at

- (2019) Roland Schwaiger und Joachim Steinwendner (2019). “Neuronale Netze programmieren mit Python”
- (2008) Roland Schwaiger (2008). "**Sprachen und Standards für IST- und SOLL-Prozessbeschreibungen im betrieblichen Umfeld**", Books on Demand, 2008, ISBN(13) 978-3-8370-6322-6
- (2015) Roland Schwaiger und Martin Schwaiger (2015). “**Agile Prozesserfassung**”, Books on Demand, 09.2015, ISBN(13) 978-3-8391-6919-3



Möge die Inspiration mit dir sein!

Kapitel 9
Vom Hirn zum Netz

Vom Hirn zum Netz

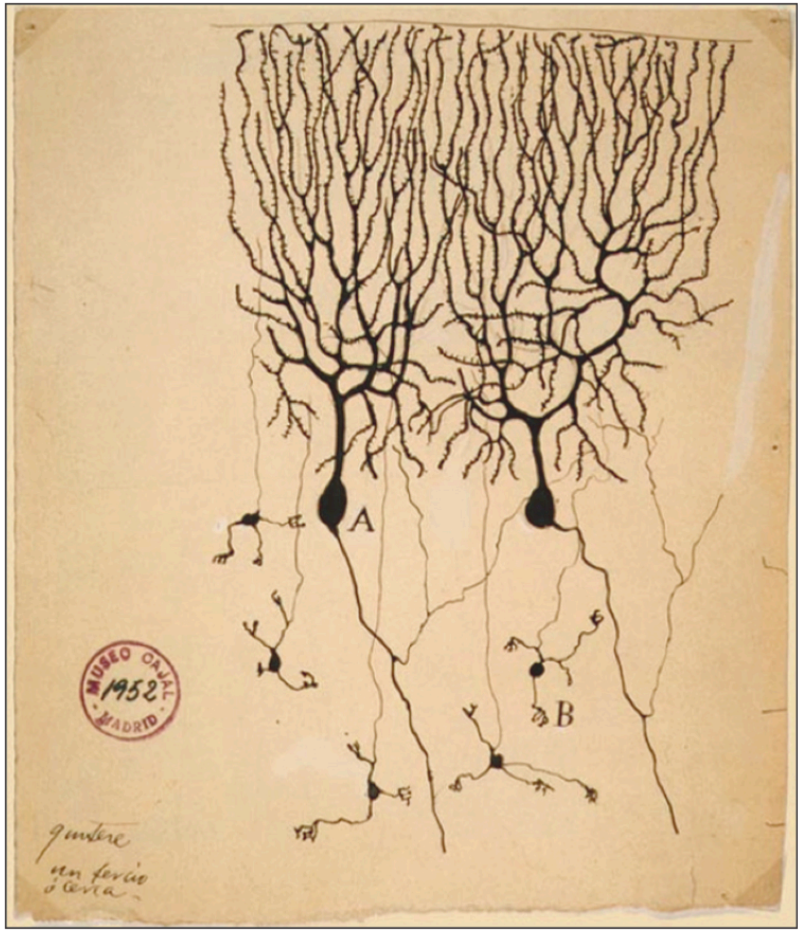


Abbildung 9.8 Die Zeichnung zweier Purkinjezellen (A) und fünf Körnerzellen (B) aus dem Kleinhirn einer Taube von Ramón y Cajal, 1899[1]

Alternative Theorie

Scientific American