

Entwicklungsstufen, Auswirkung und Definition

Freitag, 16. September 2022 13:40

HTL Krems – Informationstechnologie Felix Schneider KI 4

Künstliche Intelligenz – Entwicklungsstufen, Auswirkung und Definition

Künstliche Intelligenz (engl. *Artificial Intelligence*, AI) sorgte vor allem in den letzten Jahren für einen weitreichenden Medienrummel. In unzähligen Artikeln war von selbstfahrenden Autos, virtuellen Assistenten oder Anwendungen im Gesundheitsbereich zu lesen. Man möchte fast glauben, dass der Einsatz von KI die Lösung für jede Aufgabenstellung ist. Doch was verbirgt sich hinter diesem Begriff und in welcher Beziehung steht dieser zu weiteren zentralen Begriffen wie *Machine Learning* oder *Deep Learning*? ← nein

Hypo: 1. 60er HW unausgereift
2. 80er HW aufgerollt
3. 100er KI Demokratization
Laptops, Matrizenberechnung (40er-10er CPU x 5000)
Daten, bspw. Daten entwickeln, verarbeiten

Künstliche Intelligenz

Machine Learning

...
Deep Learning

Bereits in den 1950er-Jahren stellten sich einige Pioniere der Informatik die Frage, ob es möglich ist, einem Computer das „Denken“ beizubringen. Daraus lässt sich folgende kompakte Definition dieses Fachgebietes ableiten:

KI ist der Versuch, normalerweise von Menschen erledigte geistige Aufgaben automatisiert zu lösen.
(gleiche Fähigkeiten wie der Mensch (Emotionen, (gibt's heutzutage noch nicht) soziale Komplexität))

1.1 Was versteht man unter starker bzw. schwacher KI?

KI als Fachgebiet umfasst neben Machine Learning auch noch andere Ansätze, die aber nichts mit Lernen im klassischen Sinn zu tun haben. Hierzu zählen bspw. die Expertensysteme der 1980er-Jahre. Die Idee war, menschliches Denken mit einem festgeschriebenen Regelsatz nachzubilden. Dieser Ansatz ist unter *symbolische KI* bekannt. Das funktionierte bei wohldefinierten Aufgaben zwar gut (z.B. Schachprogramme), jedoch gelang es nicht, Regelwerke zur Lösung komplexer, weniger deutlich umrissener Aufgaben zu finden (wie z.B. Klassifizierung von Bildern o. die Erkennung natürlicher Sprache). So ergab sich mit Machine Learning ein neuer Ansatz.

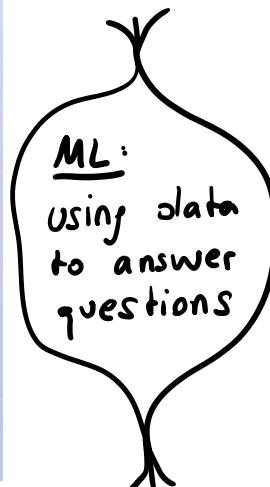
Machine Learning folgt einem anderen Paradigma, nämlich, dass ein Computer automatisch Regeln lernt, indem er vorhandene Daten durchsucht. Es braucht also keine expliziten Datenverarbeitungsregeln durch den Programmierer, wie das bei der symbolischen KI der Fall ist.

Regeln → Klassisches Programmieren → Antworten
Daten ↓
Stark limitiert

effizient
Antworten ⇒ DATEN (Beispiele)
⇒ Machine Learning
allgemein gültige Regeln (vergleiche Parameter)

! Deep Learning mit Python und Keras; Francois Chollet, mitp Verlag; 2018; ISBN 978-3-95845-838-3

KI4_B01-Entwicklung-Auswirkung-Definition-v0.2.docx 1/2

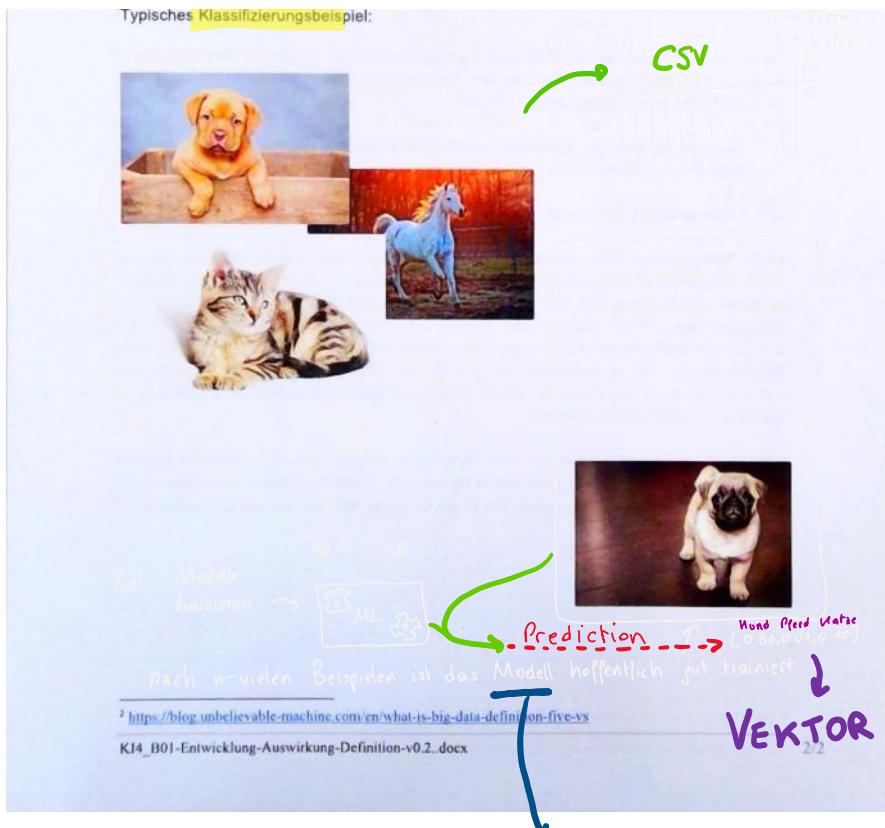


Beim Machine Learning werden sowohl Daten als auch die dazugehörigen Antworten vorgegeben. Ergebnis sind die *Regeln*, die dann in weiterer Folge auf neue Daten angewendet werden können (siehe Bsp. Bildklassifizierung) und so eigenständig Antworten liefern. Und dieses „eigenständig“ kann, wenn man möchte, als künstliche Intelligenz verstanden werden.

Zusammenfassend ist festzuhalten: Machine Learning-Systeme werden nicht explizit programmiert, sondern vielmehr trainiert. Training heißt, es werden viele aussagekräftige Beispiele (die für die zu lösende Aufgabe repräsentativ sind) hinsichtlich Muster und Gesetzmäßigkeiten durchsucht. Sind diese gefunden, können daraus allgemeine Regeln zur Automatisierung der Aufgabe abgeleitet werden. Unerlässliche Voraussetzung ist, dass es viele, viele repräsentative Beispiele in entsprechender Qualität gibt. Schlussendlich sind wir an brauchbaren Ergebnissen interessiert. In diesem Zusammenhang ist die Bedeutung von Big Data zu erwähnen. Denn erst durch die Entwicklungen im Bereich von Big Data konnte Maschinelles Lernen diesen Reifegrad erzielen.

1.2 Nehmen Sie den referenzierten Blog-Bericht „What is Big Data?“ durch.

Typisches Klassifizierungsbeispiel:
Handwritten digit recognition
CSV
0 → Hund
1 → Pferd
2 → Katze
3 → Auto
4 → Vogel
5 → Boot
6 → Motorrad
7 → Auto
8 → Vogel
9 → Boot



Modell besteht aus sogenannten
PARAMETERN



Grundlagen des Machine Learnings

Machine Learning (ML) ist mehr als nur *Deep Learning*. Wie der Abbildung 1 zu entnehmen ist, umfasst ML neben Deep Learning auch noch andere Algorithmen. Je nach Aufgabenstellung liefert ein Algorithmus gute oder weniger gute Antworten. Auch wenn Deep Learning das „angesagte Zeug“ sein mag, haben bspw. SVM oder Random Forest ebenso praktische Bedeutung.

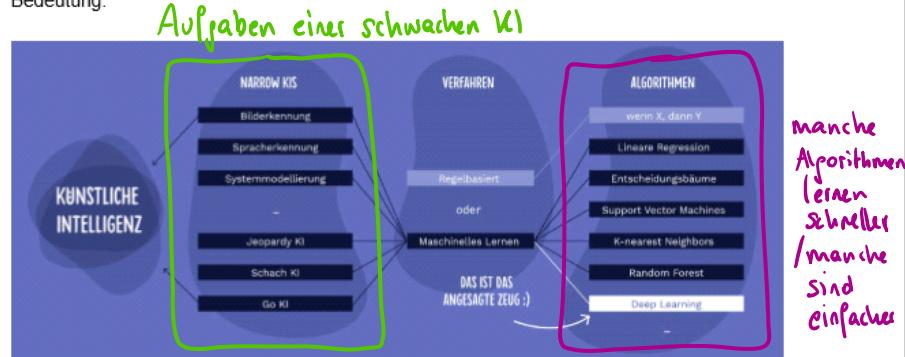


Abbildung 1: KI – Verfahren und Algorithmen im Überblick¹

Die in Abbildung 1 (auszugsweise) angeführten Algorithmen lassen sich grob in drei grund-sätzliche Lernmethoden des ML unterteilen:

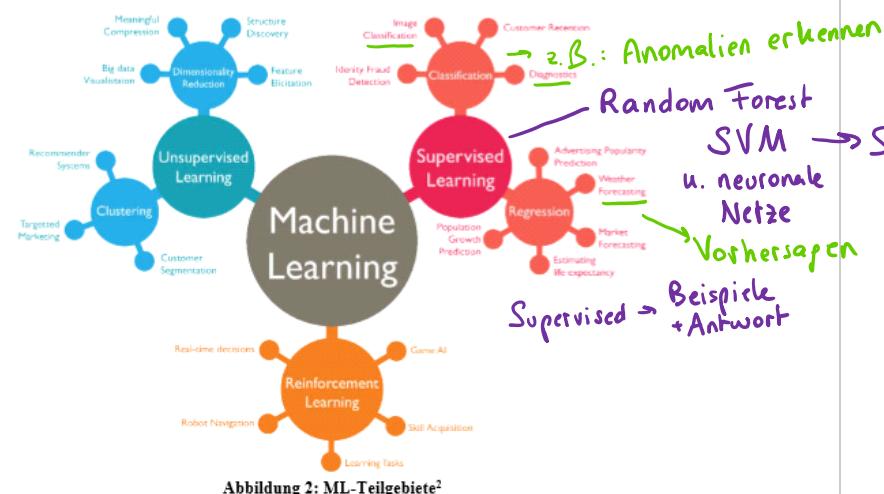
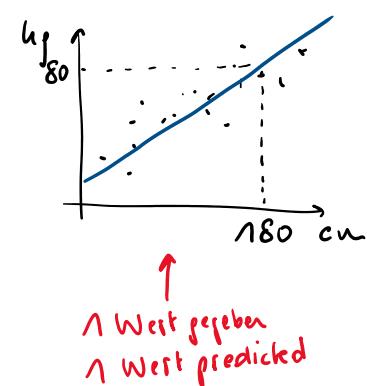


Abbildung 2: ML-Teilgebiete²



¹ Quelle: <https://news.microsoft.com/uploads/prod/sites/40/2018/11/Deep-Learning-einfach-gemacht.pdf>

² Quelle: <https://wordstream-files-prod.s3.amazonaws.com/s3fs-public/machine-learning.png>

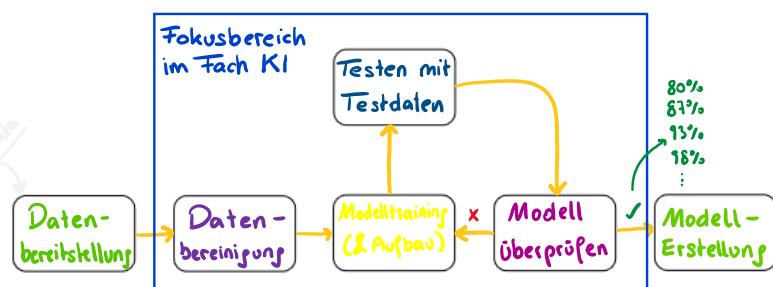
Supervised Learning: Ist die bei Weitem häufigste Form des Lernens. Das Bsp. der letzten Einheit (**Mehrfachklassifizierung** der Tierfotos) ist ein Standardbeispiel für überwachtes Lernen. Kurz um: Die Eingabedaten bestehen aus einer Reihe von **Beispielen** mit den dazugehörigen **Antworten** (die i. d. Regel von Menschen geschaffen werden). Mit überwachtem Lernen kann man etwas klassifizieren oder prognostizieren (Regression).

Unsupervised Learning: Beim unüberwachten Lernen lernt der Algorithmus auf eigene Faust. Es werden **keine Zielvorgaben** gemacht (vgl. Anmerkung „Katze“, „Hund“ etc.). Der Algorithmus wird ausschließlich mit Daten gefüttert, was uns Entwicklern theoretisch Zeit spart. Andererseits werden in der Regel aber viel mehr Daten und vor allem Rechen-Power benötigt. Deswegen findet diese Art des Lernens wenig praktische Anwendung.

Reinforcement Learning: Beim sog. bestärkenden Lernen werden vorab keine Daten benötigt. Der Algorithmus **generiert Lösungen und Strategien auf Basis von Belohnungen** im *Trial-and-Error*-Verfahren. Das Algorithmus Ziel ist es, die Belohnung zu maximieren. Wichtig: Die einzelnen Aktionen sind nicht vorgegeben.

Der Machine Learning Prozess

Wie geht man bei der Umsetzung eines Projektes vor? Auf jeden Fall methodisch! Nachfolgendes Diagramm fasst die typische Vorgehensweise zusammen:



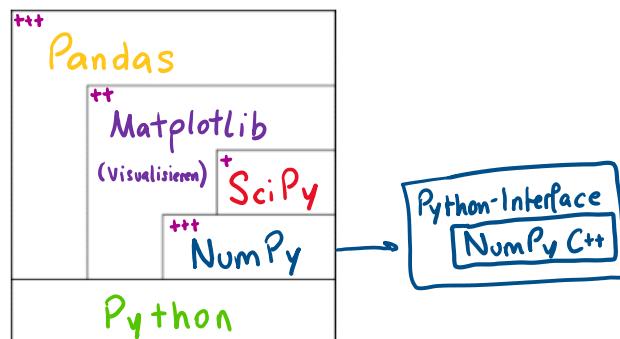
- 1. **Daten besorgen**: z.B. von Kunden, Sensoren o. Datasets aus dem Netz (frei oder kostenpflichtig) etc.
- 2. **Datenbereinigung**: fehlerhafte Daten entfernen (z.B. Nullwerte, Felder auffüllen) und Vektorisierung der Daten
- 3. **Split der Daten**
 - a. **Modelltraining und Aufbau**
 - b. **Testdaten**
- 4. **Modell überprüfen**
- 5. **Modellerstellung**: Rollout in das Produktivsystem



Python, NumPy, Matplotlib, SciPy und Pandas

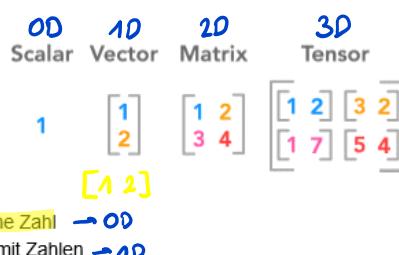
Python ist eine universelle Programmiersprache, die sich in unterschiedlichen Bereichen einsetzen lässt. Hierzu zählen bspw. die Systemadministration, die Webentwicklung oder die Computerlinguistik. Python kann aber auch zum Lösen numerischer Probleme herangezogen werden. Die Krux bei der Sache liegt aber in der Laufzeit als auch im Speicherverbrauch. „Reines“ Python – also ohne den Einsatz irgendwelcher Spezialmodule – würde sich beim Lösen numerischer Probleme nicht unbedingt auszeichnen. Aber glücklicherweise gibt es diese Spezialmodule.

Verwendung:
 + ... wenig / selten
 ++ ... mehrfach
 +++ ... häufig



NumPy¹ ermöglicht ein besonders effizientes Arbeiten mit **Arrays**. Hierzu stellt das Modul grundlegenden Datenstrukturen (`numpy.ndarray`, N-dimensional array) bereit, die von **SciPy**, **Matplotlib** und **Pandas** verwendet werden. Die performante und speicherschonende Verarbeitung von **Vektoren**, **Matrizen** oder **mehrdimensionalen Arrays** ist darauf zurückzuführen, dass ein großer Teil des Moduls in **C** entwickelt worden ist. Durch die kompilierten Funktionen wird **größtmögliche Ausführungsgeschwindigkeit** **garantiert**. Ferner bietet das Modul eine **riesige Anzahl** von mathematischen Funktionen, um mit diesen Arrays alles möglich anzustellen. Ein Blick in diese Quelle² verdeutlicht das.

`>!pip list`
 numpy



- Skalar: **einzelne Zahl** → **0D**
- Vektor: Array mit Zahlen → **1D**

¹ <https://numpy.org/>

² <https://numpy.org/doc/stable/reference/routines.html#routines>

- Matrix: **2-D Array** → **2D**
- Tensor: **n-dimensionales Array (n>2)** → **4D**

SciPy³ baut auf NumPy auf, d.h. es benutzt die Datenstrukturen, die NumPy bereitstellt. Es erweitern die Leistungsfähigkeit von NumPy mit nützlichen Funktionalitäten wie bspw. **Regression** oder im Bereich der **deskriptiven Statistik**.

„Ein Bild sagt mehr als tausend Worte“ lautet ein bekanntes Sprichwort. Die praktische Umsetzung erfolgt in unserem Fall mit **Matplotlib**⁴. Mit diesem Modul kann man Daten anschaulich und (hoffentlich auch) aussagekräftig grafisch darstellen.

„Ein Bild sagt mehr als tausend Worte“ lautet ein bekanntes Sprichwort. Die praktische Umsetzung erfolgt in unserem Fall mit **Matplotlib⁴**. Mit diesem Modul kann man Daten ansprechend und (hoffentlich auch) aussagekräftig grafisch darstellen.

Pandas⁵ ist das neueste der hier vorgestellten Module. Es stellt Datenstrukturen (z.B. DataFrame o. Series) und die zur **Bearbeitung von Tabellen und Zeitreihen notwendigen Operatoren zur Verfügung**. In der Praxis bedeutet das: immer dann, wenn ein Datensatz als csv-Datei (vgl. music.csv) vorliegt, laden wir diesen mithilfe von Pandas in das Notebook (vgl. `pandas.read_csv(...)`). Wie der nachfolgenden Abbildung zu entnehmen ist, bildet Pandas den Inhalt der CSV-Datei als Tabelle ab. Das ist eine Hauptaufgabe des Moduls.

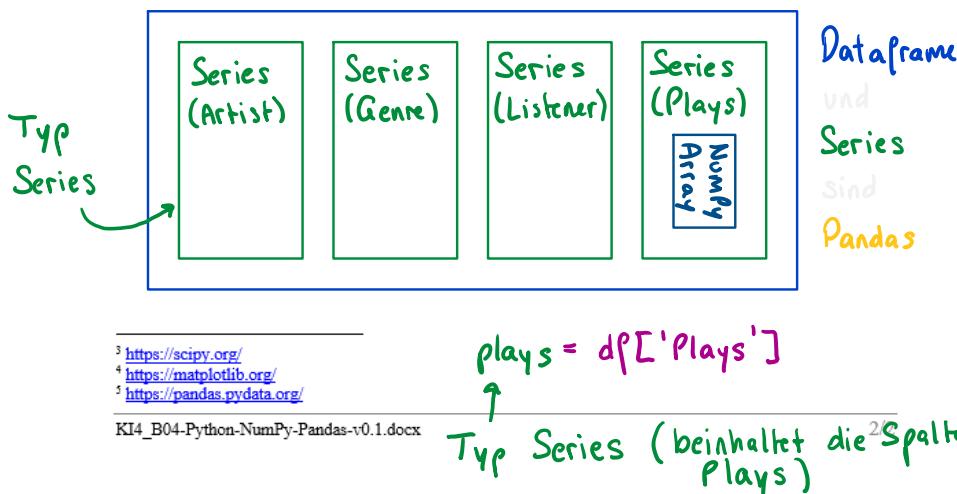
`import pandas as pd`

music.csv

	A	B	C	D
1	Artist	Genre	Listeners	Plays
2	Billie Holiday	Jazz	1,300,000	27,000,000
3	Jimi Hendrix	Rock	2,700,000	70,000,000
4	Miles Davis	Jazz	1,500,000	48,000,000
5	SIA	Pop	2,000,000	74,000,000
6				

pd.read_csv('music.csv')

Typ
Dataframe
 $df = pd.read_csv('music.csv')$



³ <https://scipy.org/>
⁴ <https://matplotlib.org/>
⁵ <https://pandas.pydata.org/>

NumPy Arrays im Detail

Die wichtigsten Attribute eines NumPy-Arrays:

- Anzahl der Achsen:** Ein 3D-Tensor verfügt bspw. über drei Achsen, eine Matrix über 2. Der Wert ist über `ndim` verfügbar. Details siehe Seite 2, unten.
- Shape** (dt. auch Gestalt oder Form): Hierbei handelt es sich um ein Tupel von Ganzzahlen, die angeben, wie viele Dimensionen das Array entlang der verschiedenen Achsen besitzt. Eine Matrix besitzt z.B. zwei Werte, ein Vektor einen. Die Shape eines Skalars ist leer. Die Shape ist über `shape` verfügbar.
- Datentyp:** Via `dtype` abruf- bzw. änderbar. Es stehen aufgrund der C-Codebasis verschiedene Typen (z.B. `uint8`, `uint16`, `float32`, `float64`) zur Verfügung.

Beispiel für Vektordaten

Eine Versicherungsdatenbank, die Alter, PLZ und Einkommen der darin gespeicherten Personen enthält. Alle Personen können durch einen Vektor, der diese drei Werte enthält, beschrieben werden. Bei z.B. 100.000 Personen kann die gesamte Datenmenge in einem 2-D-Array mit der Shape (100000, 3) gespeichert werden.

drei Beispiele: $\begin{bmatrix} [28, 3500, 50000], & \rightarrow \text{einzelne} \\ [32, 3512, 65000], & \text{Vektoren} \\ [54, 1010, 120000] \end{bmatrix} \rightarrow \text{Shape}(3, 3)$

Vektordaten – 2D-Arrays mit der allg. Shape: (`samples, features`)

Zeitreihen oder sequentielle Daten

Einmal pro Minute werden der aktuelle Börsenkurs sowie der höchste und niedrigste Kurs der letzten Minuten gespeichert. Jede Minute wird als ein 3D-Vektor erstellt, sodass sich für den ganzen Handelstag ein 2D-Array mit der Shape (390, 3) ergibt (Ann.: ein Handelstag dauert 390 Min.). Die Daten von 250 Handelstagen resultieren wiederum in einem Tensor mit der Shape (250, 390, 3). Oder anders formuliert: Jedes Sample enthält die Daten eines Handelstages.

Jede Minute ein Vektor:

[133.21, 135.14, 132.88]

Nach einem Handelsjahr mit 250 Tagen:

$\begin{bmatrix} \text{MATRIX HT1}, & \leftarrow \text{Reihen-} \\ \text{MATRIX HT2}, & \text{folge} \\ \text{MATRIX HTX}, & \text{ist} \end{bmatrix}$ von
Bedeutung

Nach einem Handelstag eine Matrix:

$\begin{bmatrix} [133.21, 135.14, 132.88] \\ [133.28, 135.10, 133.18] \\ [132.90, 133.40, 132.01] \\ \dots \\ [132.91, 133.30, 132.80] \end{bmatrix} \rightarrow \text{Vektor d. zweiten Minute}$
 $\rightarrow \text{Vektor d. dritten Minute}$
 $\rightarrow \text{Vektor d. 390ten Minute}$

Shape(250,390,3)

Zeitreihen – 3D-Tensor mit der allg. Shape: (`samples, timestamps, features`)

KI4_B04-Python-NumPy-Pandas-2-v0.1.docx

Anzahl HT (Anzahl Kurse) Kurse

Bilddaten

Bilder weisen typischerweise 3 Dimensionen auf: Höhe, Breite u. Farbtiefe. Graustufenbilder besitzen zwar nur einen Farbkanal und könnten daher auch als 2D-Array gespeichert werden, aber gemäß Konvention werden Bilddaten immer als 3D-Tensor gespeichert. Bei Graustufenbildern ist der Farbkanal also eindimensional. Ein als z.B. 128 Graustufenbilder der Größe 3x3 bestehender Datenstapel könnte demnach in einem (128, 3, 3, 1)-Tensor gespeichert werden.



3

3
Wert: 0-255

array([[[255, 255, 255],
[255, 0, 255],
[255, 255, 255]]])

1 Bild

shape(1)

shape(3, 1)

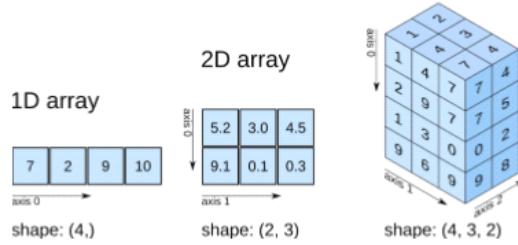
shape(3, 3, 1)

import numpy as np
a = np.array ([1, 2, 3])
↓
Type: NumPy

Merkmal,
Feature,
Vektor

↗ ↓
 Wert: 0-255
 $\text{shape}(3, 1)$
 $\text{shape}(3, 3, 1)$
 $\text{shape}(128, 3, 3, 1)$
 #rrggbbaa
 eher unüblich

Bilder – 4D-Tensoren mit der allg. Shape: (samples, height, width, channels)
 Achsen bei 1D, 2D und 3D Arrays im Überblick:
 Bilder 128 3 3 1 Farbe Graustufenbild
 1D array px px 1 3D array



KI4_B04-DeskriptiveStat-v0.2

Friday, January 27, 2023 1:18 PM



Idee
wichtige Merkmale einer
Datenbank zusammenfassen
daraus folgen

Methode

Deskriptive Statistik

Als Data Scientists sind wir an der Beschaffenheit der vorliegenden Daten interessiert. Das umfasst nicht nur bekannte **Lageparameter** (wie Mittelwert¹ o. Median), sondern auch Streuungsmaße, die Auskunft über die Streuung der Daten in unserem Datensatz geben.

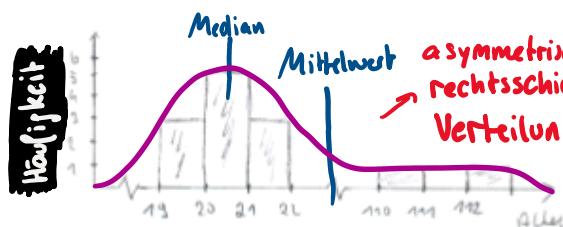
Mittelwert vs. Median

gesucht ist ein typischer Wert für die Datenreihe

Alter in einem Kung-Fu-Kurs

Alter	19	20	21	110	112
Häufigkeit	3	5	3	1	1

n=13

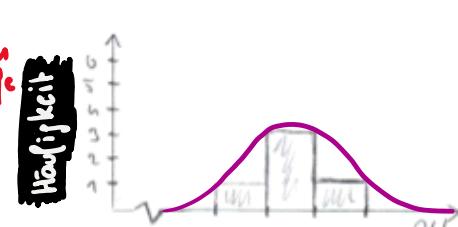


Mittelwert = 34
Median = 20

1

Alter	19	20	21
Häufigkeit	1	3	1

n=5



Mittelwert = 20
Median = 20

2

Median Vorgehensweise:

Ausreißer
Mittelwert wird negativ beeinflusst

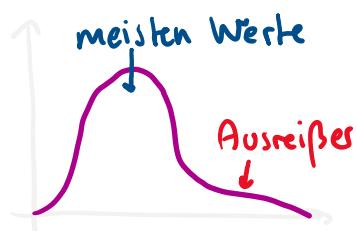
- sortieren
- mittleren Wert nehmen

1 19 19 20 20 20 20 20 21 21 21 110 112

2 19 20 20 20 21

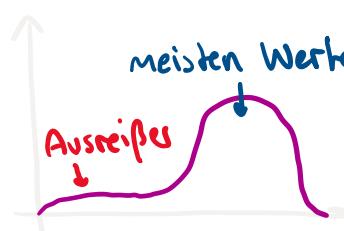
Ausreißer: Ein extrem hoher o. niedriger Wert, der vom Rest der Daten stark abweicht.

Rechtsschiefe Daten

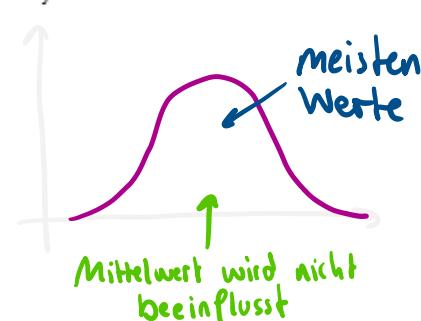


Modalwert (Modus) – jener Wert, mit der größten Häufigkeit

Linksschiefe Daten



Symmetrische Daten



Alter	1	2	3	31	32	33
Häufigkeit	3	4	2	2	4	3

→ Sonderfall

¹ Genau genommen ist hiermit das arithmetische Mittel gemeint.

Mittelwert = 17

Median = 17

1 1 1 2 2 2 2 3 3 31 31 32 32 32 32 33 33

Was soll man mit solchen Daten machen?

2 Modalwerte (2,32) \rightarrow bimodal

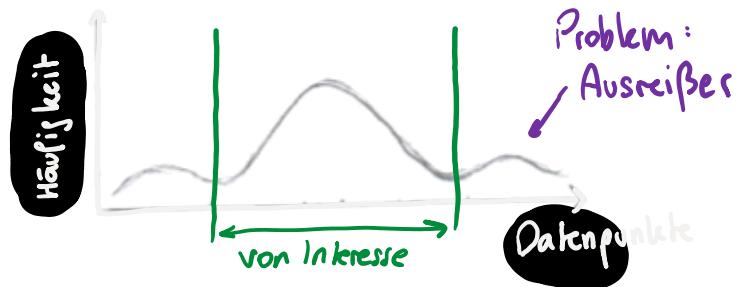
Alter	1	2	3	31	32	33
Häufigkeit	3	4	2	2	4	3

splitten

Streuungsmaße

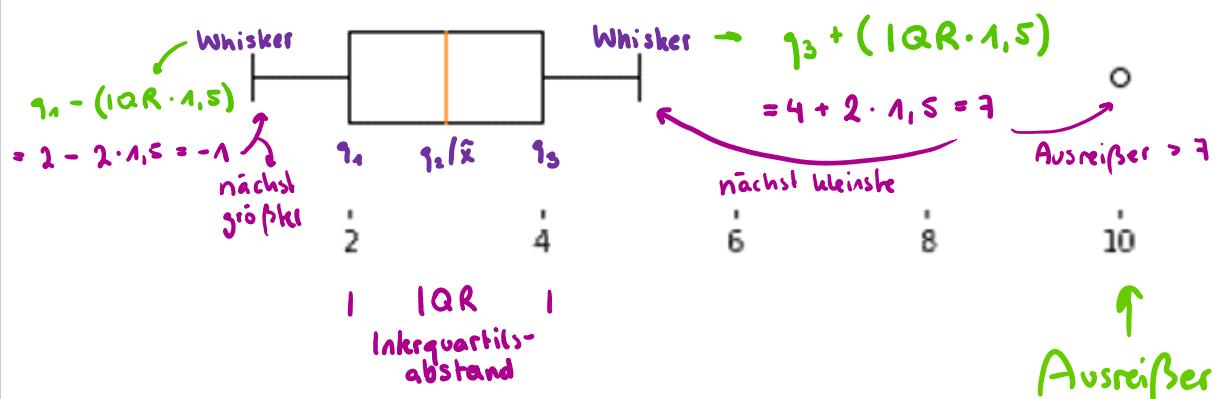
Um die Verteilung (Streuung) von Daten zu verstehen und ungewöhnliche oder auffällige Werte (Ausreißer, vgl. Werte des Kung-Fu-Bsp.) zu identifizieren, ist ein sog. Boxplot überaus hilfreich. Der Interquartilsabstand (IQR) ist die Differenz zwischen dem dritten Quartil (Q3) und dem ersten Quartil (Q1). Dieser gibt einen schnellen (grafischen) Überblick über die Verteilung der Daten und ist robust gegen Ausreißer.

Ein weiteres Streuungsmaß ist die Standardabweichung, die angibt, wie stark die Werte vom Mittelwert abweichen. Je größer die Standardabweichung, desto größer die Streuung der Daten um den Mittelwert. Eine kleine Standardabweichung deutet auf eine enge Verteilung der Daten hin, während eine größere auf breitere Verteilung hindeutet.



Quartile helfen uns. Werte werden hierbei aufsteigend sortiert und in vier gleich große Teile geteilt.

$n=20$ q_1 q_2/\bar{x} q_3 $q_1 \text{ Position} = n \cdot 0,25$
1 1 1 2 2 | 2 2 3 3 3 | 3 3 4 4 4 | 4 5 5 5 10 $q_2 \text{ P.} = n \cdot 0,5$
 | | | | | | | | | |
 2 4 6 8 10



$$x_p = \begin{cases} \frac{1}{2}(x_{\lfloor np \rfloor} + x_{\lfloor np+1 \rfloor}) & \text{falls ganzzahlig} \\ x_{\lfloor np+1 \rfloor} & \text{falls nicht ganzzahlig} \end{cases}$$

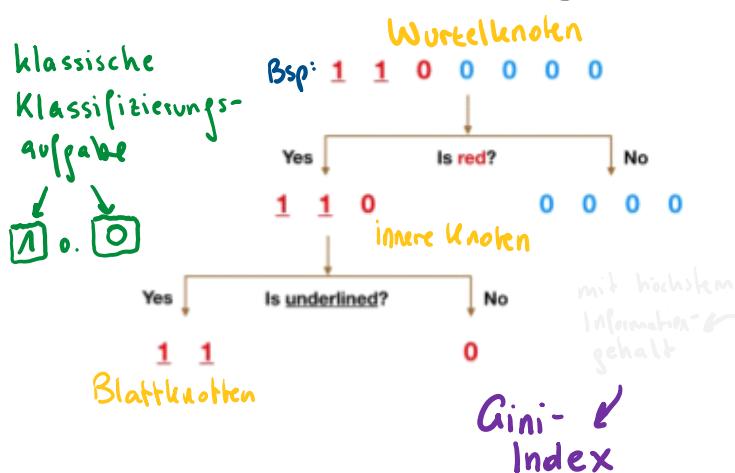
KI4_B06-RandomForest-v0.2

Friday, February 3, 2023 1:20 PM



Random Forest

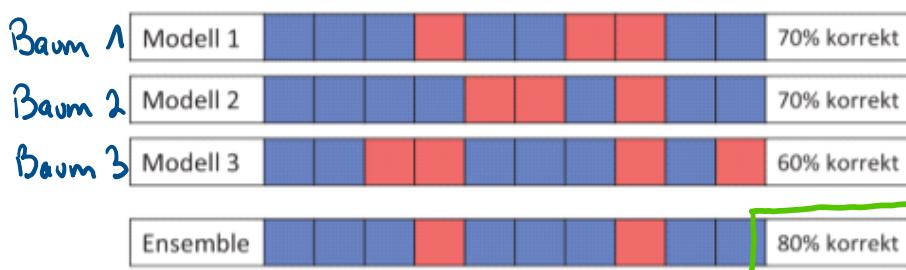
Random Forests könnte man am besten mit „Die Weisheit der Crowd“ charakterisiert. Aber der Reihe nach: Random Forest ist ein ML-Algorithmus, der sich für Klassifizierungs- als auch Regressionsaufgaben eignet. Nachdem der Algorithmus anhand von Beispielen und Antworten lernt, zählt dieser zu den überwachten Lernverfahren (*Supervised Learning*). Für die Vorhersagen nutzt der Algorithmus die Ergebnisse einer Vielzahl an Entscheidungsbäumen. Wie funktioniert ein Entscheidungsbaum?



Unsere Daten bestehen aus **1**, **0** und **0**. Die Idee ist, dass der Entscheidungsbaum **1** und **0** erkennt, also vorhersagen kann, ob es sich um eine **1** oder **0** handelt. Um das zu bewerkstelligen, muss der Algorithmus **aussagekräftige** Merkmale (**Features**) in den Daten erkennen, anhand welcher der Baum aufgebaut wird. Genau das geschieht beim sog. **Training**.

Jetzt ist es aber so, dass die Vorhersagegenauigkeit eines einzelnen Entscheidungsbaumes einem bestimmten Bias (Fehler) unterliegt. Daher ist es ratsam, auf die Vorhersagen mehrerer Bäume – in diesem Zusammenhang spricht man von einem Ensemble, also dem Wald – zurückzugreifen.

Forest mit $n=3$:



gute
Ergebnis ist
der Weisheit
der Crowd
geschuldet

Ein Ensemble mit **Mehrheitsentscheid** liefert präzisere Vorhersagen als die einzelnen Bäume (Modelle), die es enthält. Das liegt daran, dass korrekte Vorhersagen einander bestärken, während Fehler sich gegenseitig aufheben. Das Ensembling funktioniert allerdings nur dann, wenn die darin enthaltenen Modelle nicht alle dieselben Fehler machen. Um das zu vermeiden, sollten die Modelle in hohem Maße unkorreliert sein. Ein systematischer Weg, um unkorrelierte Bäume zu erzeugen, ist das sog. Bootstrap Aggregating, kurz Bagging.

Bagging

Entscheidungsbäume reagieren sehr empfindlich auf die Daten, anhand welcher sie trainiert werden – kleine Änderungen im Trainingsdatensatz können zu deutlich unterschiedlichen Baustrukturen führen. Diesen Umstand macht sich der Random Forest Algorithmus zunutze, indem jeder Baum aus einer zufällig gewählten Untermenge von Daten und Features erzeugt wird. So entstehen viele unterschiedliche Entscheidungsbäume, die jedoch alle eine gewisse Vorhersagekraft besitzen. Indem die Zahl der möglichen Features für jede Verzweigung beschränkt wird, entstehen immer wieder andere Bäume.

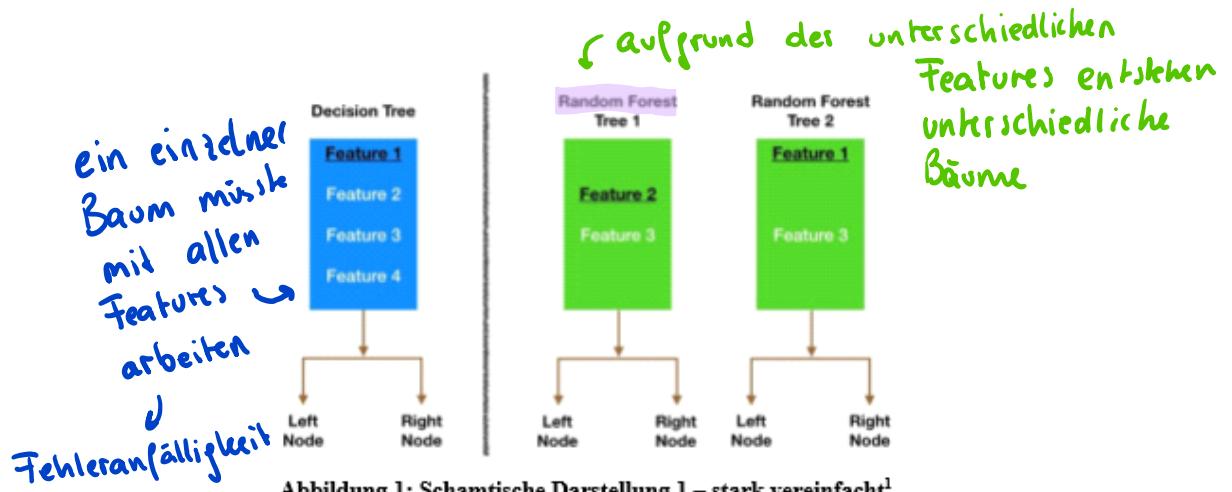


Abbildung 1: Schamtische Darstellung 1 – stark vereinfacht¹

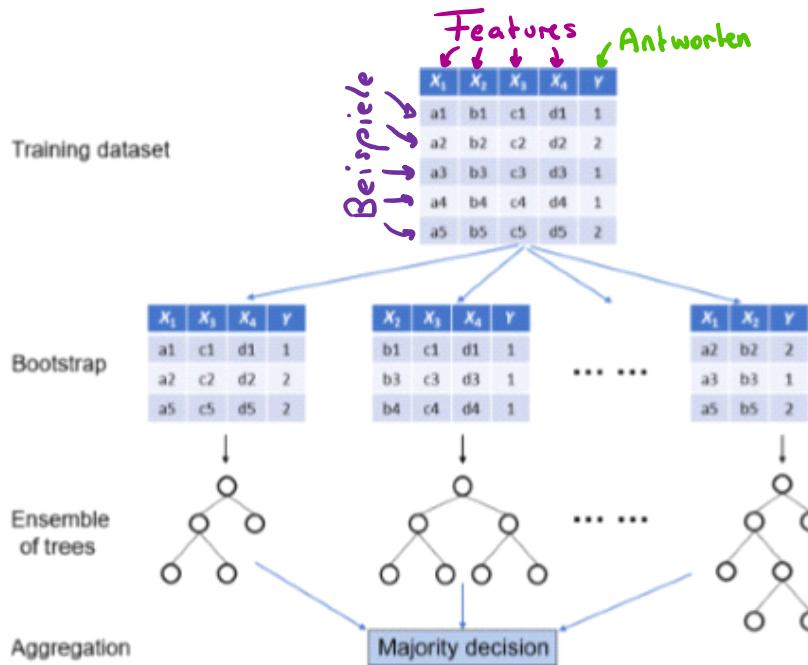


Abbildung 2: Schematische Darstellung 2 – detailliert(er)²

Die praktische Umsetzung eines *Random Forest Classifier* erfolgt mit der Scikit-learn-Implementierung `sklearn.ensemble.RandomForestClassifier`³.

¹ <https://towardsdatascience.com/understanding-random-forest-58381e0602d2>

² https://pages.cms.hu-berlin.de/EOL/geo_rs/S09_Image_classification2.html

³ <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

KI4_B06-Datasets-Iris-Notes-v0.1

Friday, February 17, 2023 1:21 PM



Iris dataset

↳ Beispiele ↳ Antworten

Grundsätzlich gestaltet sich der Aufbau des Datasets so, dass jede Zeile für genau eine Blume steht. Ziel: Klassifizierung, ob Blume (vgl. Beispiel) vom Typ Setosa, Versicolor o. Virginica ist.

Features:

Beispiele: →



:

→

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
...					
49	5.0	3.3	1.4	0.2	0
50	7.0	3.2	4.7	1.4	1
51	6.4	3.2	4.5	1.5	1
99	5.7	2.8	4.1	1.3	1
100	6.3	3.3	6.0	2.5	2
101	5.8	2.7	5.1	1.9	2

Answers
Klasse 0

Target
/Labels

Klasse 1

Klasse 2

shape (150, 5)

Dataset:

X-train (Beispiele für das Training)	Y-train (Antworten → Training)
X-test (Beispiele für die Validierung)	Y-test (Antworten)

~100
Datensätze
(70%)

// RFC trainieren

clf.fit(x-train, y-train)

// Prediction

y-pred = clf.predict(x-test)

↳ : shape(50,)

		y-pred	y-test	correct?
~50	Datensätze (30%)	1	0	0
2		2	2	1
0		0	1	0

PROZENTSATZ
ermitteln

Confusion Matrix I & II

Friday, February 24, 2023 1:19 PM



Confusion Matrix I

Bei dem Heart (oder Iris) Dataset haben wir die sog. **Accuracy** (Korrektklassifizierungsrate), also wie viele Einträge **prozentual richtig klassifiziert** wurden, ermittelt.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Number of all predictions}}$$

Die Accuracy ist jedoch **mit Vorsicht zu genießen**, man sollte dieser nicht immer allzu viel Glauben schenken, aber dazu später mehr. Vorerst interessiert uns die Funktionsweise einer **Confusion Matrix (CM)**.

↳ bei welchen Daten hat der **Classifier von meinem Modell Fehler gemacht.**

ACHTUNG:

manchmal 0/1
vertauscht

		y-pred PREDICTIVE VALUES		← vom Classifier oder Neuronalem Netz vorhergesagt
		POSITIVE (1)	NEGATIVE (0)	
ACTUAL VALUES	y-test	TP ☺	FN ☹ ↴	Fehler sollte minimalst sein
	POSITIVE (1)	true positiv	false negativ	Heart-Dataset: im Heart-DS
		FP ☹	TN ☺	y-test [0110100011010-]
		false positiv	true negativ	0... gesund 1... potentielles Risiko

2x2 Matrix

Abbildung 1: Confusion Matrix für die **Binärklassifizierung**¹

Es werden die vorhergesagten Werte (*Predicted Values*) den tatsächlichen (*Actual Values*) gegenübergestellt, und zwar Eintrag für Eintrag. Grundlage bilden hierbei jene 30% des Datasets, die für die Validierung zurückgehalten wurden. Hierbei wird Datensatz für Datensatz ausgewertet, ob der vorhergesagte Wert dem tatsächlichen entspricht – oder eben nicht! Das Ergebnis ist dann einem der vier Quadranten (siehe Abbildung 1) zuzuordnen, die da lauten: *True Positiv*, *False Negative*, *False Positive* und *True Negativ*. Wie die die Zuordnung im Detail erfolgt, ist folgendermaßen definiert:

- **TP:** Vorhersage ist „1“ (Positive), was wahr (True) ist, weil der tatsächliche Wert ebenfalls „1“ (Positive) ist. Daraus folgt „True Positive“.
- **FN:** Vorhersage ist „0“ (Negative), was aber falsch (False) ist, weil der tatsächliche Wert „1“ (Positive) ist. Daraus folgt „False Negative“.
- **FP:** Vorhersage ist „1“ (Positive), was aber falsch (False) ist, weil der tatsächliche Wert „0“ (Negative) ist. Daraus folgt „False Positive“.
- **TN:** Vorhersage ist „0“ (Negative), was richtig (True) ist, weil der tatsächliche Wert „0“ (Negative) ist. Daraus folgt „True Negative“.

¹ <https://towardsdatascience.com/decoding-the-confusion-matrix-bb4801decbb>

Betrachtung im Kontext des Heart Datasets

0 ... nicht krank

1 ... krank

		predicted_values	
		1	0
actual_values	1	Patient weiblich krank	Krankheit nicht erkannt
	0	unnötige Medikamenten- vergabe	Patient krankgesund

Übungsbeispiel Cat & Dog

predicted_values = ['dog', 'dog', 'dog', 'cat', 'dog', 'dog', 'cat', 'cat', 'cat', 'cat']

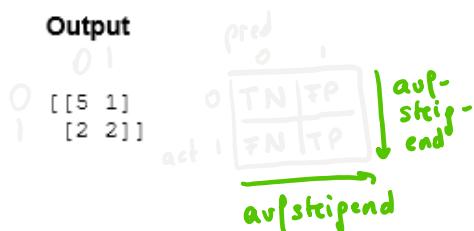
actual_values = ['dog', 'cat', 'dog', 'cat', 'dog', 'dog', 'cat', 'dog', 'cat', 'dog']

		PREDICTIVE VALUES	
		POSITIVE (CAT)	NEGATIVE (DOG)
ACTUAL VALUES	POSITIVE (CAT)	TRUE POSITIVE  YOU ARE A CAT	FALSE NEGATIVE  YOU ARE A DOG
	NEGATIVE (DOG)	FALSE POSITIVE  YOU ARE A CAT TYPE I ERROR	TRUE NEGATIVE  YOU ARE NOT A CAT

- **Fall 0:** „True Negative“, weil die Vorhersage „Dog“ dem tatsächlichen Wert („Dog“) entspricht. „Negative“ ist jetzt als Dog zu verstehen.
- **Fall 1:** „False Negative“, weil eine Katze irrtümlicherweise als „Dog“ erkannt wurde, daher „False...“.
- ...

Erstellung der Confusion Matrix mit `sklearn.metrics.confusion_matrix`²:

```
# predicted values  
predicted = [1, 0, 0, 1, 0, 0, 0, 1, 0, 0]  
  
# actual values  
actual = [1, 0, 0, 1, 0, 0, 1, 0, 0, 1]  
  
print(confusion_matrix(actual, predicted))
```



Iris:

y-pred [1 0 0 2 1 0 1 ...] → predicted values
y-test [1 0 1 2 0 1 1 ...] → actual values
✓✓✗✓✗✗✓

Accuracy: Prozentsatz der richtigen Vorhersagen zu ermitteln!

Klasse 0: 5 Bsp.
Klasse 1: 5 Bsp.
Klasse 2: 140 Bsp.

Sehr gut auf Klasse 2 trainiert, aber nicht auf 1 und 0.
Accuracy: 97%
in Wahrheit schlechter (für 0 und 1)

² https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html



Confusion Matrix II

Ergebnis der Evaluierung eines Modells mit 1000 Beispielen:

		Predicted/Classified	
		Negative	Positive
Actual	Negative	998	TN
	Positive	1	FN
			1 FP
			1 TP

$$\text{Accuracy} = 999/1000 = 0,999$$

Wie ist dieses Ergebnis zu werten?

Precision

Die Precision (Genauigkeit) misst, wie präzise unsere *positiven* Vorhersagen waren:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

Recall

Der Recall (Sensitivität) misst den Anteil der durch das Modell identifizierten Positiven:

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

F1 Score

Kombiniert man Precision und Recall erhält man das F1-Maß, das folgendermaßen definiert ist (harmonische Mittel):

$$F1 = 2 \times \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

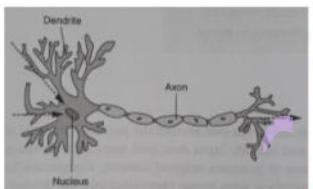
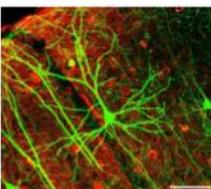
Normalerweise bedingt die Auswahl des Modells einen Kompromiss zwischen Genauigkeit und Sensitivität. Ein Modell, das schon beim kleinsten Anzeichen „Ja“ vorhersagt, wird voraussichtlich eine hohe Sensitivität haben, aber eine geringe Genauigkeit. Ein Modell, das nur „Ja“ sagt, wenn es sich absolut sicher ist, wird voraussichtlich eine geringe Sensitivität und eine hohe Genauigkeit aufweisen.

Am Ende des Tages ist es ein Kompromiss zw. falsch Positiven und falsch Negativen. Zu häufig „Ja“ sagen, führt zu mehr falsch Positiven, zu häufig „Nein“ sagen, führt zu einer Menge falsch Negativen.



Das Perzeptron

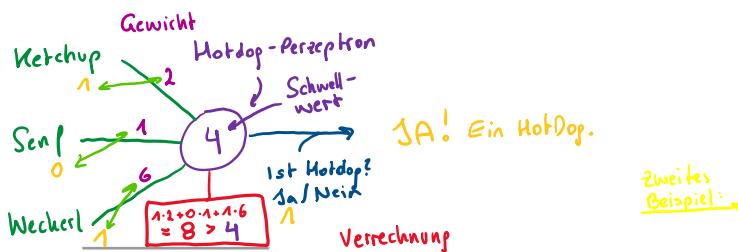
Das Grundkonzept neuronaler Netze liegt in der groben Nachahmung der Funktionsweise einer **Nervenzelle (Neuron)**. Jede Nervenzelle (siehe Abbildung 1) erhält über von vielen **Dendriten** eine Eingabe in seinen Zellkörper (Nucleus). Wenn das Signal, das an einem Dendriten entlang übermittelt wird, den Zellkörper erreicht, löst es eine **kleine Spannungsänderung** in diesem aus. Manche Dendriten verursachen eine kleine positive Spannungsänderung, andere eine negative. Wenn der **kumulierte Effekt** dieser Änderung einen Anstieg der Spannung vom Ruhezustand (-70mV) **über die Schwelle von rund -55mV erreicht**, feuert die Zelle ein sog. **Aktionspotenzial** vom Kern aus in sein **Axon** und überträgt damit ein **Signal** an andere Neuronen im Netzwerk.

Abbildung 1: Nervenzelle (schematisch)¹Abbildung 2: Nervenzellen (mikroskopische Aufnahme)²

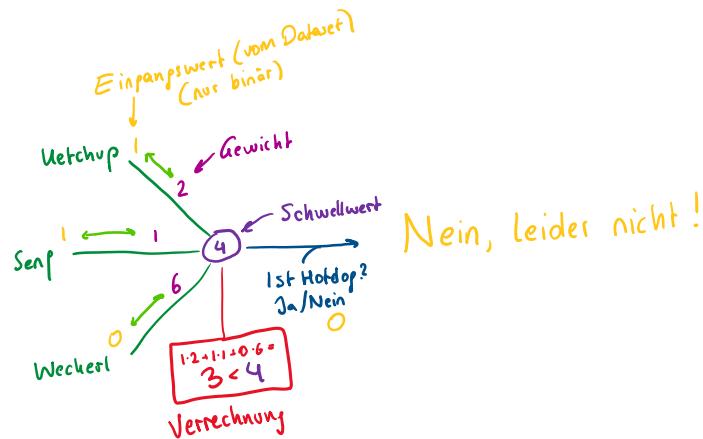
Ende der 50er-Jahre hat F. Rosenblatt einen Artikel über sein **Perzeptron** veröffentlicht, el-nen Algorithmus, der von der Funktionsweise des biologischen Neurons inspiriert war und somit das erstes künstliche Neuron darstellt. Dieses konnte

- Eingaben von mehreren anderen Neuronen empfangen,
- diese Eingaben verrechnen (gewichtete Summe) und
- eine Ausgabe erzeugen, falls die gewichtete Summe den Schwellwert überschreitet.

Hotdog-Detektor – ein Perzeptron, dass erkennen kann, ob ein Objekt ein Hotdog ist.



¹Natural Language Processing in Action: Hobson L., Howard C., Hapka H.M.; Manning Publications; ISBN9781617294631; 2019
²Lee W.-C.A, Huang H, Feng G, Sanez JE, Brown EN, So PT, et al. (2006) Dynamic Remodeling of Dendritic Arbons in GABAergic Interneurons of Adult Visual Cortex. *PLoS Biol* 4(2): e29. <https://doi.org/10.1371/journal.pbio.0040029>



Nach Verallgemeinerung sieht die Berechnung der gewichteten Summe wie folgt aus:

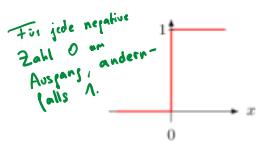
$$\Theta = \sum_{i=1}^n w_i x_i$$



Letzter Schritt - Ausgabe erzeugen:

$$\Theta = \sum_{i=1}^n w_i x_i \begin{cases} \geq \text{Schwellenwert, Ausgabe 1} \\ < \text{Schwellenwert, 0} \end{cases}$$

Das Perzeptron verwenden als Aktivierungsfunktion die sog. **Heavisidefunktion** (Sprungfunktion) mit dem Schwellenwert (engl. *Threshold*) θ :



Theta

$$o = \begin{cases} 1, \text{ wenn } \sum_{i=1}^n x_i w_i \geq \theta \\ 0, \text{ wenn } \sum_{i=1}^n x_i w_i < \theta \end{cases}$$

Um eine allgemeine Perzeptron-Gleichung formulieren zu können, führen wir den Begriff des sog. **Bias** ein, den wir mit **b** bezeichnen. Der Bias ist ein zusätzlicher Eingang, an dem immer „1“ anliegt. Dieser hat ein eigenes Gewicht, das es ebenfalls zu erlernen gilt. Der Bias kann als „Korrekturfaktor“ verstanden werden.

- Es gilt: $b = \text{-Schwellenwert}$

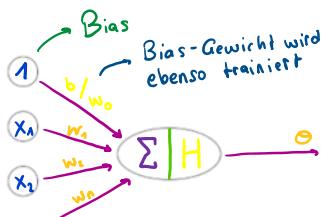
↳ wird genauso trainiert

$$o = \begin{cases} 1, \text{ wenn } \sum_{i=1}^n w_i x_i + b \geq 0 \\ 0, \text{ wenn } \sum_{i=1}^n w_i x_i + b < 0 \end{cases}$$

Angepasst:

$$o = \begin{cases} 1, \text{ wenn } \sum_{i=1}^n w_i x_i \geq \theta \\ 0, \text{ wenn } \sum_{i=1}^n w_i x_i < \theta \end{cases}$$

Schwellwert θ



KI4_B09-NeuronaleNetze-v0.1

Sunday, April 30, 2023 10:58 AM



Künstliche Neuronale Netze

Moderne **künstliche Neurone** sind keine Perzeptronen. Die Perzeptron offensichtlichste Einschränkung ist, dass es ausschließlich **binäre Eingaben** verarbeitet und auch **nur binäre Antworten** liefert. In vielen Fällen wollen wir jedoch Vorhersagen treffen, die auf kontinuierliche Variablen (als Eingangswerte) basieren. Ein weiterer Faktor ist der Übergang des Perzeptrons bei der Ausgabe von „0“ auf „1“, dieser erfolgt ganz plötzlich. Das Perzeptron kennt keine Finesse: Entweder es „schreit“ oder es „schweigt“.

Das Sigmoid-Neuron

Jedes künstliche Neuron, das die *Sigmoid*-Funktion als seine Aktivierungsfunktion benutzt, wird als Sigmoid-Neuron bezeichnet ab. Diese bietet uns eine Alternative zum sprunghaften Verhalten des Perzeptrons: es erfolgt ein **sanfter Übergang von „0“ nach „1“ bei der Ausgabe**. Dadurch wird auch das Lernen der Parameter entscheidend erleichtert.

Es gilt: $z = w \cdot x + b$

Dot product (Punkt-/Skalarprodukt \Rightarrow 1 Wert)

Problem d. Sättigung der Neuronen beim asymptotischen Verlauf.

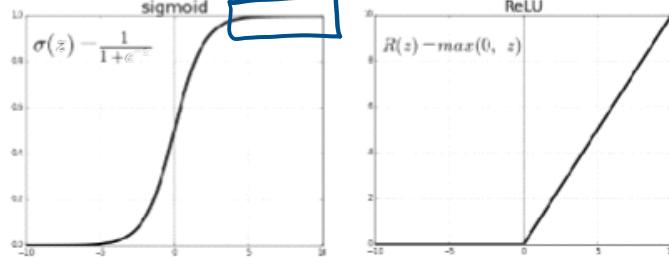


Abbildung 1: Sigmoid und ReLU Funktionen¹

↳ Aktivierungsfunktionen

ReLU – Rectified Linear Units

Das ReLU-Neuron bzw. die ReLU-Funktion ist eine der einfachsten Funktionen. Im Prinzip werden zwei verschiedene lineare Funktionen kombiniert: eine, bei der negative z-Werte „0“ und eine, bei den positiven z-Werten „z“ zurückliefern. Diese **Nichtlinearität** ist eine wichtige Eigenschaft aller Aktivierungsfunktionen, die in Neuronalen Netzen zum Einsatz kommen. Die relativ einfache Form von Nichtlinearität bei der ReLU ist beim Rechnen ein (**Performanz**-) **Vorteil**. ReLU-Neuronen sind daher die am häufigsten eingesetzten Neuronen in den verborgenen Schichten.

Schichten eines künstlichen neuronalen Netzwerkes

Ein neuronales Netz besteht zumindest aus **3 Schichten**:

- **Eingabeschicht:** Führt keine Berechnungen durch, sondern dient lediglich als **Platzhalter für die Eingabedaten**.
- **Verborgene Schicht(en):** Es gibt viele Arten, der allgemeinste Type ist jedoch die **vollständig verbundene Schicht (Dense Layer)**. Es gilt: Jedes Neuron dieser Schicht ist mit jedem anderen Neuron der Schicht davor verknüpft.

Deep Learning
↳ mehrere Schichten



¹ <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>

- **Ausgabeschicht:** In dieser Schicht sind **Sigmoid-Neuronen die typische Wahl**, wenn es ein **Klassifizierungsproblem** zu lösen gilt. Denn die Sigmoid gibt Aktivierungen aus, die zw. „0“ und „1“ liegen. Somit wird es uns möglich, die vom Netzwerk geschätzten **Wahrscheinlichkeiten** zu erhalten.

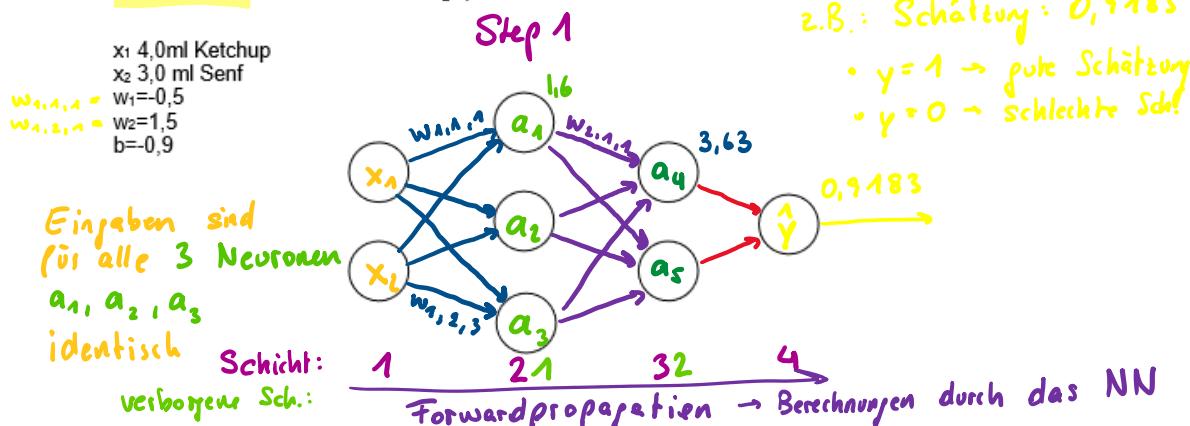
$$42.8 : 0.4327 \dots = 43\% \text{ Klasse } 0$$



die zw. „0“ und „1“ liegen. Somit wird es uns möglich, die vom Netzwerk geschätzten Wahrrscheinlichkeiten zu erhalten.

Netzwerk zum Erkennen von Hotdogs

Netzwerkarchitektur: Eingabeschicht mit 2 Neuronen², zwei vollständig verbundene verborgene Schichten mit drei bzw. zwei ReLU-Neuronen und die Ausgabeschicht mit einem Sigmoid-Neuron, weil Klassifizierungsproblem.



Forwardpropagation durch die erste verborgene Schicht:

$$z = w \cdot x + b = (w_{1,1,1,1} \cdot x_1 + w_{1,2,1,1} \cdot x_2) + b =$$

$$a = \max(0, z) = (-0,5 \cdot 4 + 1,5 \cdot 3) + (-0,9) =$$

ReLU $z = 2 + 4,5 - 0,9 = 1,6$

$$a = \max(0, z) = \max(0, 1,6) = 1,6$$

Aktivierung von a_1 = 1,6
 Es wirkt sich auf a_4 und a_5 aus...

Forwardpropagation durch die nachfolgende Schicht:

Berechnung	Aktivierung a_4	$\max(0, w \cdot a_1 + w \cdot a_2 + w \cdot a_3 + b_4)$
------------	-------------------	--

$$a_4 = \max[0, w_{2,1,1,1} \cdot \max(0, w_{1,1,1,1} \cdot x_1 + w_{1,2,1,1} \cdot x_2 + b_1) +$$

$$+ w_{2,2,1,1} \cdot \max(0, w_{1,1,2,1} \cdot x_1 + w_{1,2,2,1} \cdot x_2 + b_2) +$$

$$+ w_{2,3,1,1} \cdot \max(0, w_{1,1,3,1} \cdot x_1 + w_{1,2,3,1} \cdot x_2 + b_3) + b] = 3,63$$

Werte:
 $w_{1,1,1,1} = -0,5$
 $w_{1,2,1,1} = 1,5$
 $w_{1,1,2,1} = -2,8$
 $w_{1,2,2,1} = 0,4$
 $w_{1,1,3,1} = -8,5$
 $w_{1,2,3,1} = -1,3$
 $x_1 = 4$
 $x_2 = 3$
 $b_1 = -0,9$
 $b_2 = 12,1$
 $b_3 = -1,2$
 $b_4 = 0,3$

Das Ausgabeneuron ist ein Sigmoid-Neuron. Um also seine Aktivierung a zu berechnen, übergeben wir seinen z -Wert an die Sigmoid-Funktion:

$$a = \sigma(z)$$

$$a_4 = 3,63 \quad w_{3,1,1,1} = 1 \quad w_{3,2,1,1} = -0,5 \quad b = -0,3 \quad z = 2,42$$

$$a_5 = 1,82 \quad w_{3,1,2,1} = -0,5$$

$$a = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-2,42}} = 0,9183 \dots \approx 91,8\% \text{ ein Hotdog}$$

² Der Einfachheit halber um das „Weckerl“ reduziert!

KI4_B09-NN-MINST-v0.2

Friday, April 21, 2023 1:13 PM



Neuronales Netzwerk - MNIST

Beim MNIST-Dataset¹ handelt es sich um 70.000 Bildern von handgeschriebenen Ziffern; wobei 60.000 Beispiele im Trainings- und 10.000 Beispiele im Testdatensatz enthalten sind. Jedes der Gaußtiefenbilder hat eine Dimension von 28 x 28 Pixeln.

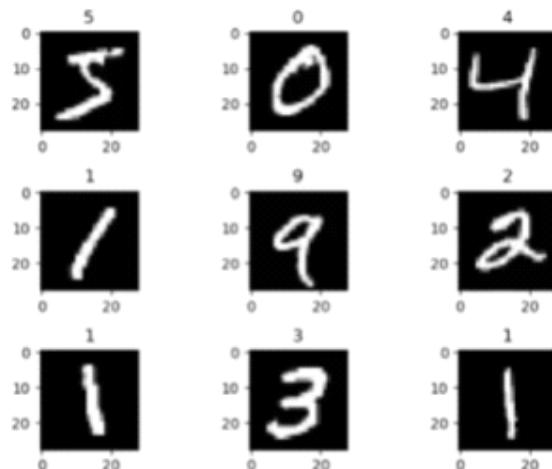


Abbildung 1: Die ersten 9 Bilder des Datasets

Die Werte eines Bildes liegen zw. 0 und 255. Das ergibt 256 „Graustufen“. Die Ausgabe eines einzelnen Bildes (Auszug) liefert ein Array mit der Shape (28,28). Damit kein falscher Eindruck entsteht: Von 28 Zeilen sind nur 3 zu sehen, es gibt tatsächlich Werte zw. 0 und 255.

- 1
- 2
- 3

Image als Matrix (28, 28)

Ziel ist es, ein Neuronales Netz (mit **(zwei)** Layern) zu trainieren, das imstande ist, handgeschriebene Ziffern zu klassifizieren. Der Einfachheit halber streben wir eine Netzstruktur mit 3 Layern an – also sonderlich Deep ist das nicht!



¹ <http://yann.lecun.com/exdb/mnist/>

Mehr Details liefert nachfolgende **Model Summary**. Die gelb markierten Bereiche sind ausführungsbedingt und haben – zumindest jetzt – keine Bedeutung.

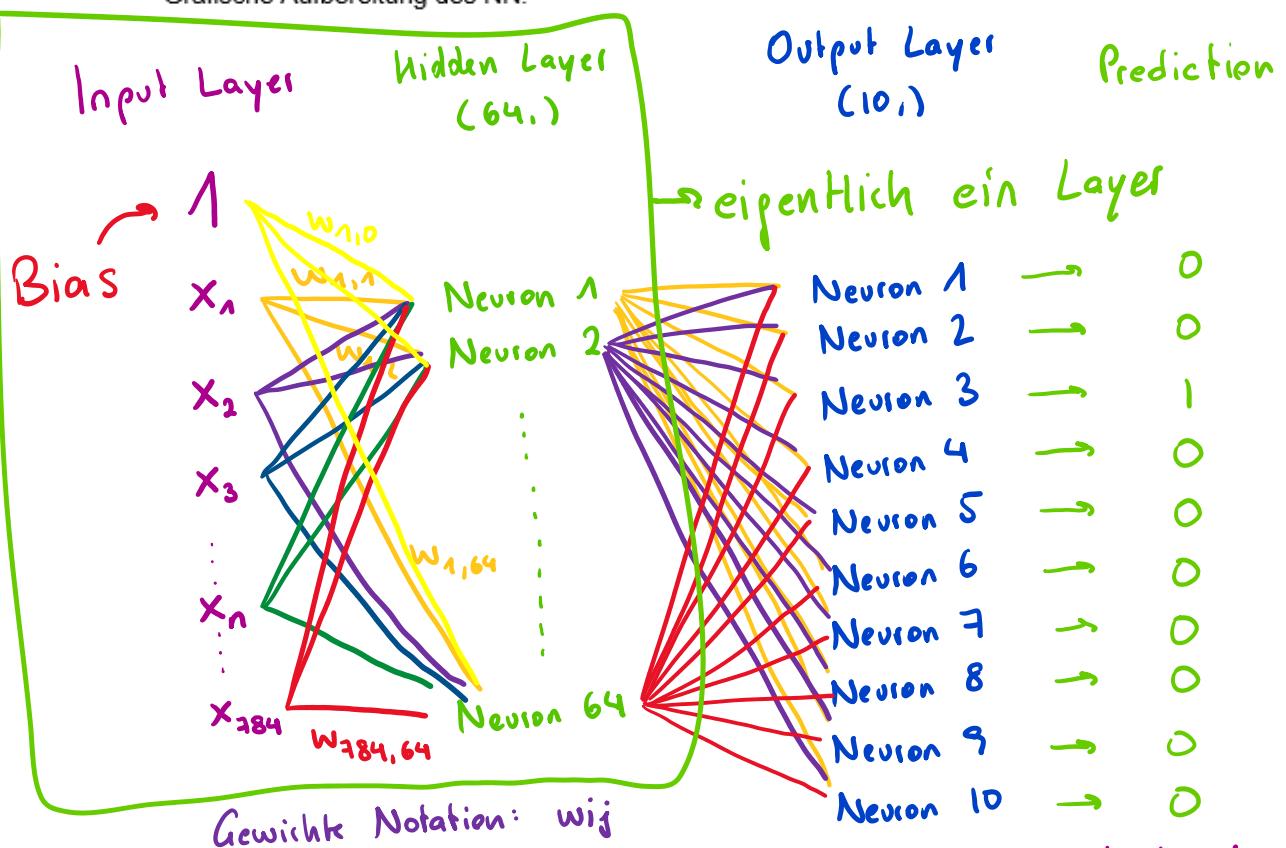
Model: "sequential_16" fortlaufend ↗ Methoden Output

Layer (type)	Output Shape	Param #
dense_32 (Dense)	(None, 64)	50240
dense_33 (Dense)	(None, 10)	650
Total params:	50,890	
Trainable params:	50,890	
Non-trainable params:	0	

↳ Output Shape resulted

↳ Output Shape resultiert aus der Neuronenanzahl d. jeweiligen Layers

Grafische Aufbereitung des NN:



- Dense Layer realisiert die Verdriftung der Layer untereinander

$$384 \times 64 = 50176 \quad (\text{50240 Parameter fehlen noch})$$

$$+64 \text{ Bias} = \boxed{50240} \quad \hookrightarrow \text{Bias fehlt noch}$$

↳ Parameter für Hidden Layer

$64 \times 10 + 10 \text{ Bias} = \boxed{650}$ Parameter für Output Layer

Lo in Summe 50890 Parameter



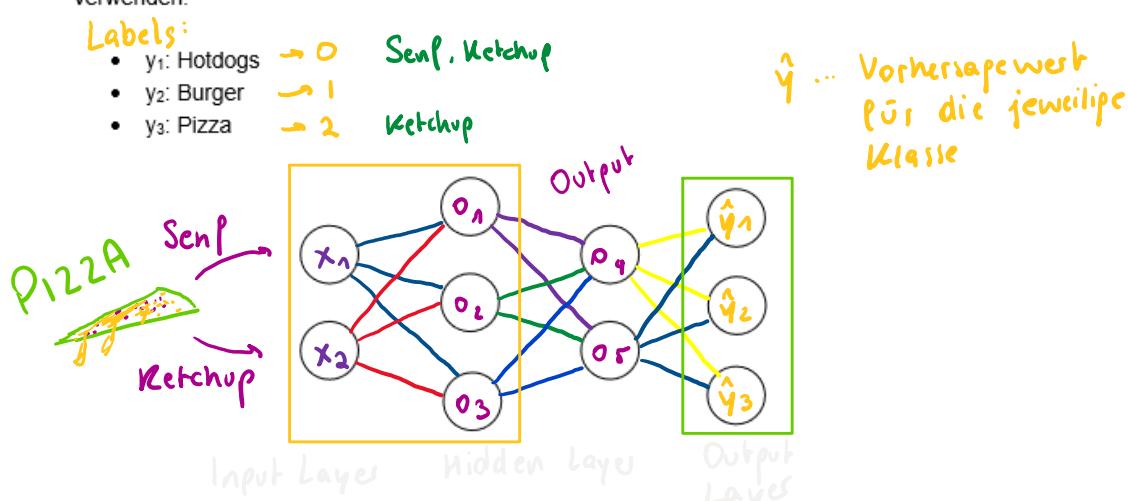
Künstliche Neuronale Netze - Mehrklassenproblem

Die Sigmoid-Aktivierungsfunktion eignet sich ganz gut, um die Wahrscheinlichkeit einer binären Ausgabe (z.B. ja oder nein) vorherzusagen. Bei einem Mehrklassenproblem, wie das beim MNIST-Datensatz der Fall ist, braucht es jedoch Softmax-Neuronen in der Ausgabeschicht. Die Softmax-Aktivierungsfunktion erzeugt eine Wahrscheinlichkeitsverteilung über alle Klassen hinweg. Hierbei wird die Wahrscheinlichkeit für jede Klasse berechnet. Die Summe aller Wahrscheinlichkeiten ergibt 1.

Sigmoid Neuron

Netzwerk zum Erkennen von Hotdogs, Burger oder Pizza

Zur Erinnerung: Beim binären Hotdog-Klassifikator hatten wir 1 Sigmoid-Ausgabeneuron. Im vorliegenden Fall soll das Netzwerk zwischen 3 Klassen (Hotdogs, Burger u. Pizza) unterscheiden können. Folglich handelt es sich hierbei um ein sog. Mehrklassenproblem. Demnach braucht es 3 Neuronen in der Ausgabeschicht, die die Softmax-Aktivierungsfunktion verwenden.



Die einzige Abhängigkeit bei Softmax ist die exp-Funktion (Exponentialfunktion), die die eulersche Zahl „e“ als Basis verwendet. Wenn wir mit dem Befehl `exp(x)` einen Wert x an die Funktion übergeben, erhalten wir e^x zurück.

Angenommen, wir präsentieren dem oben vorgestellten Netz eine Pizza, dann enthält diese ein wenig Ketchup, aber keinen Senf, sodass x_1 bspw. 0,5 und x_2 sicherlich 0 ist. Mit diesen Eingaben erfolgt die Forward-Propagation, um Informationen bis zur Ausgabeschicht zu propagieren. Durch diese ergibt die Berechnung der z-Werte folgendes Ergebnis:

$$\left. \begin{array}{l} y_1: z = -1,0 \\ y_2: z = 1,0 \\ y_3: z = 5,0 \end{array} \right\} \text{Inputs d. Aktivierungsfunktion (Softmax)} \quad \text{↳ wahrscheinlich Pizza}$$

Diese Werte lassen bereits vermuten, dass das präsentierte Objekt höchstwahrscheinlich eine Pizza ist.

// ↑ für Hotdog

$$P(\text{Hotdog}) = \exp(-1.0) / (\exp(-1.0) + \exp(1.0) + \exp(5.0)) = 0.002428$$

// ↑ für Burger

$$P(\text{Burger}) = \exp(1.0) / (\exp(-1.0) + \exp(1.0) + \exp(5.0)) = 0.01794$$

// ↑ für Pizza

$$P(\text{Pizza}) = \exp(5.0) / (\exp(-1.0) + \exp(1.0) + \exp(5.0)) = 0.9799$$

$$\hookrightarrow [0.002428, 0.01794, 0.9799].\text{sum}() == 1$$

„Softmax“ kann somit so verstanden werden: Die Funktion liefert **z** mit dem höchsten Wert zurück (dem *max*), aber tut dies ganz sanft, indem nicht 100% bzw. 0% ausgegeben werden.

Die Gleichung¹:

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

¹ Quelle <https://www.analyticsvidhya.com/blog/2021/04/introduction-to-softmax-for-neural-network/>

KI4_B09-NeuronaleNetze-CostfunctionBackprop-v0.1

Friday, May 5, 2023 1:18 PM



Neuronale Netze – Lernen, um die Kosten zu minimieren

Trainingsdaten (vgl. X_{train}) werden in einzelne Batches aufgeteilt und dann Batch für Batch per **Forwardpropagation** durch das Netzwerk geleitet. Als Ergebnis der Forwardpropagation kann die Vorhersage in \hat{y} angesehen werden. Wäre nun ein Netzwerk perfekt kalibriert, würden die \hat{y} -Werte exakt gleich den Label-Werten y sein. Dieser Goldstandard ($\hat{y} = y$) wird jedoch bei praxisrelevanten Aufgabenstellungen faktisch nicht erreicht.

Die Genauigkeit der Vorhersagen des Netzwerks wird durch die Kostenfunktion gemessen. Ziel ist es, die Parameter (Gewichte) so anzupassen, dass die **Kostenfunktion** – also die Differenz zw. Vorhersage \hat{y} und dem Label-Wert y – minimiert wird.

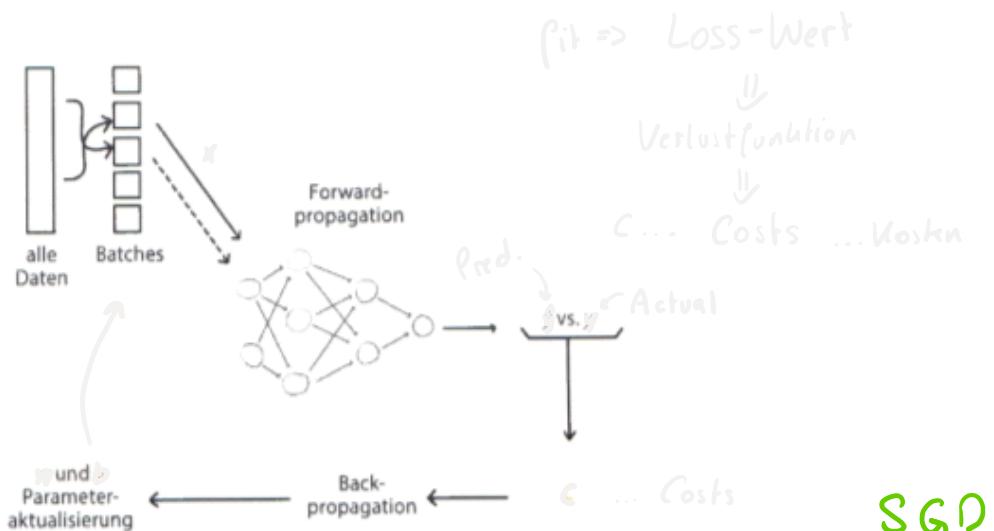


Abbildung 1: Überblick über das Training eines NN¹

→ Vektor, der in Richtung der max. Steigung zeigt

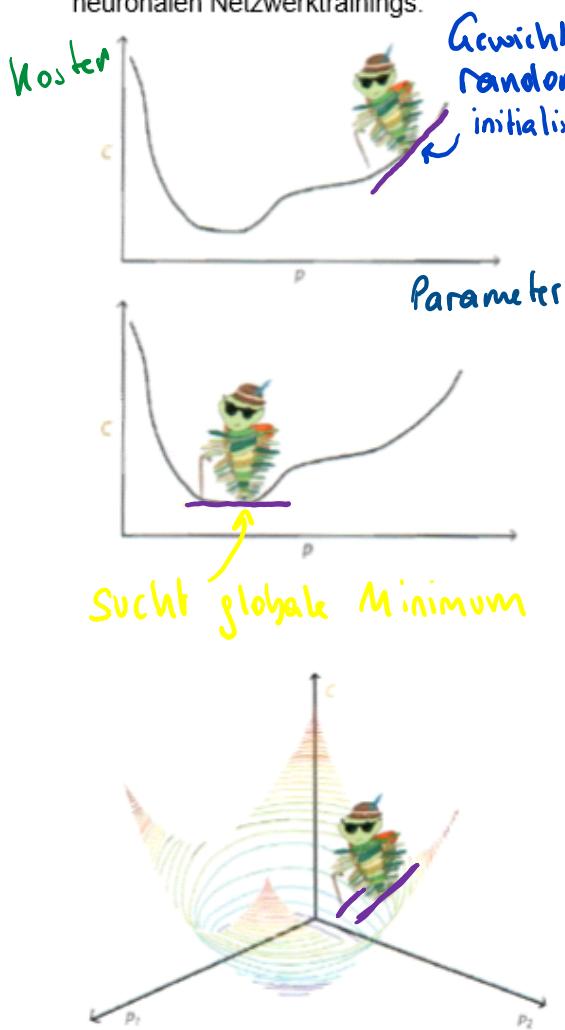
Hierbei findet das sog. **Gradientenabstiegsverfahren** Anwendung. Das Gradientenabstiegsverfahren ist ein **iterativer Optimierungsalgorithmus**, der die **Parameter des Netzwerks anpasst** mit dem Ziel, die Kosten zu minimieren. Theoretisch ist es möglich, das Minimum einer (differenzierbaren) Funktion analytisch zu bestimmen. Bei einem Neuronalen Netz mit einigen Tausend oder Millionen Parametern ist das jedoch nicht mehr machbar. Stattdessen wird der Gradient der Kostenfunktion an bestimmten Punkten bestimmt, um so Schritte in **Richtung des lokalen Minimums** zu machen. Der Gradient ist ein **Vektor, der die Richtung der maximalen Steigung einer Funktion an einem bestimmten Punkt angibt**. Um die Richtung der minimalen Steigung zu erhalten, wird einfach das **Vorzeichen des Gradienten umgekehrt**.

Bei neuronalen Netzen wird die Ableitung der Kostenfunktion nach den Gewichten des Netzwerks berechnet, um die Richtung zu bestimmen, in der die Gewichte angepasst werden müssen. Diese Richtung wird als Gradient bezeichnet. Für die Interessierten liefert diese Quelle² weiterführende Informationen.

¹ Deep Learning illustriert; J. Krohn, G. Beyleveld, A. Bassens; dpunkt.Verlag

² <https://www.youtube.com/watch?v=odIgtjXduVg>

Zusammen bilden diese Konzepte das sog. Backpropagation und somit das Fundament des neuronalen Netzwerktrainings.



Beispiele des Gradientenabstiegs entlang einer eindimensionalen Verlustfunktion, um jenen Wert des Parameters p zu finden, der die **wenigsten Kosten verursacht**. Die gestrichelte Linie kennzeichnet den Wert des Anstieges, der an der jeweiligen Stelle berechnet wurde. Bei einem weiteren Schritt nach links würden die Kosten wieder steigen.

In der Praxis hat ein Modell nicht nur einen Parameter. Bei zwei Modellparametern (p_1 und p_2) ergibt sich quasi bereits ein „Gebirge“ - p_1 und p_2 sind als Längen- und Breitengrade zu verstehen, während die Höhe die Kosten C repräsentiert. Hierbei gilt: je geringer die Höhe, umso besser!

Wie groß die einzelnen Schritte des Gradientenabstiegs sind, wird durch die sog. **Lernrate η (Eta)** bestimmt. Die Lernrate ist als ein **weiterer Hyperparameter** zu verstehen. Die richtige Lernrate zu finden, ist gar nicht so einfach, denn bei zu großen oder zu kleinen Werten kann das Neuronale Netz nicht wirklich lernen.

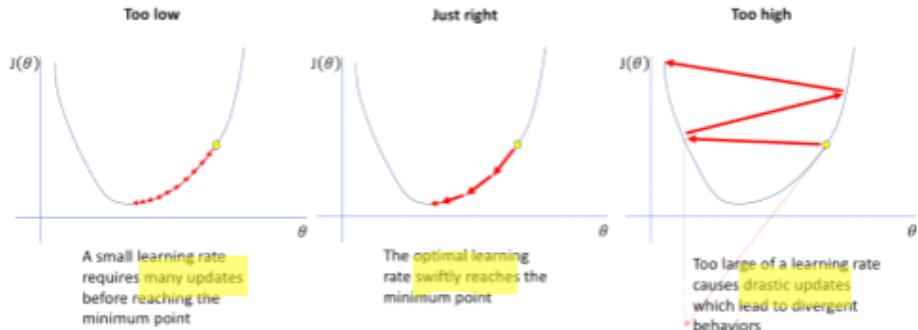


Abbildung 2: Lernrate und deren Auswirkung³

Quadratische Kosten

Die quadratischen Kosten C (mittlerer quadratischer Fehler; mean squared error (MSE)) sind wohl die am einfachsten zu berechnende Kostenfunktion.

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

Für jede Instanz i berechnen wir die Differenz (den Fehler) zwischen dem wahren Wert in y_i und der Vorhersage in \hat{y}_i . Anschließend bilden wir das Quadrat dieser Differenz.

- Somit ist die Differenz zw. den beiden Werten immer positiv, egal welcher Wert größer oder kleiner ist.
- Durch das Quadrieren werden größer Fehler (Differenzen) zw. y_i und \hat{y}_i viel stärker „bestraft“ als kleinere.

Um die mittleren Kosten C zu erhalten, sind alle Instanzen zu addieren und durch deren Anzahl zu dividieren.

Die quadratischen Kosten haben jedoch einen Nachteil, und zwar dann, wenn sich ein Neuron (bzw. deren Aktivierungsfunktion) in der Sättigung befindet. Diese entsteht, wie wir bereits wissen, durch hohe z-Werte. Unter diesen Umständen lernt das NN sehr langsam. Abhilfe schaffen die Kreuzentropie-Kosten.

Kreuzentropie Kosten Crossentropy

Durch die Kreuzentropie-Kosten wird die Wirkung gesättigter Neuronen auf die Lerngeschwindigkeit minimiert. Deshalb ist diese Kostenfunktion viel beliebter. Die Gleichung, die man sich nicht merken muss:

$$C = \frac{-1}{n} \sum_{i=1}^n [y_i \ln \hat{y}_i + (1 - y_i) \ln (1 - \hat{y}_i)]$$

Kurz um: Die Kreuzentropie-Kosten sind so strukturiert, dass das ein Neuron umso schneller lernt, je größer der Unterschied zw. y_i und \hat{y}_i ist.

³ <https://www.jeremyjordan.me/nn-learning-rate/>

KI4_B10-AktivierungsfunktionenUeberblick-v0.1 - Kopie

Friday, May 12, 2023 1:16 PM



Aktivierungsfunktionen im Überblick

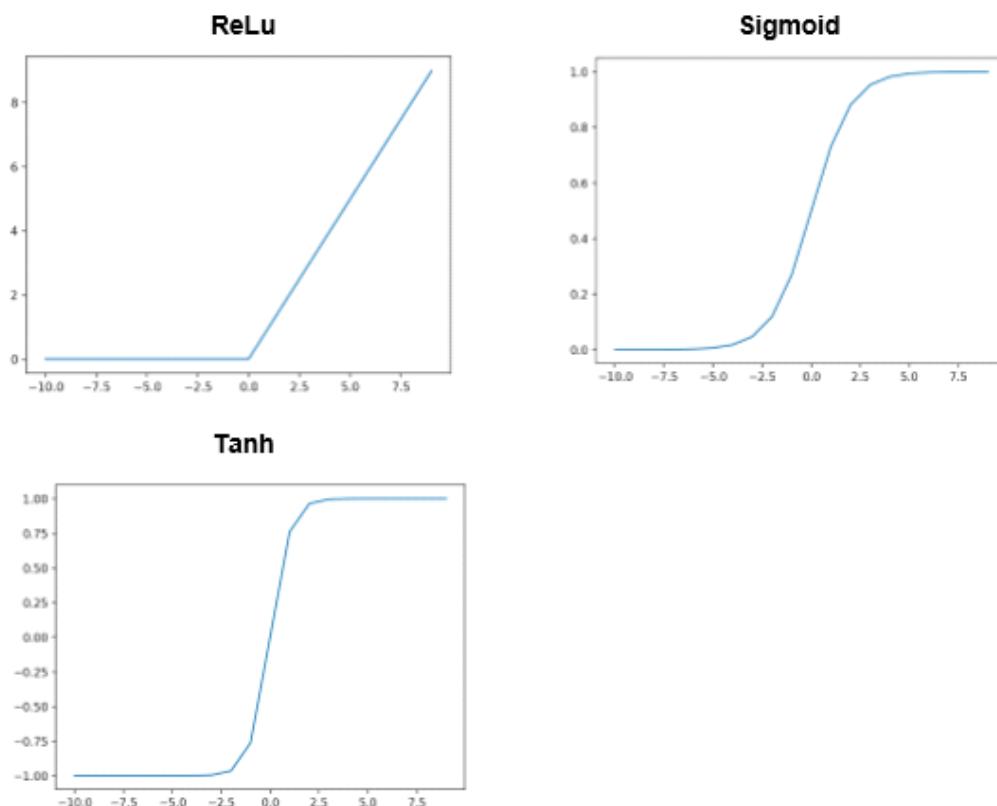
Die Wahl der Aktivierungsfunktion für den jeweiligen Layer ist von zentraler Bedeutung. Im Falle eines Hidden Layer wird damit bestimmt, wie gut das Netz anhand der Trainingsdaten lernen kann. Was den Output-Layer betrifft, definiert die Aktivierungsfunktion die Art und Weise wie das Modell Vorhersagen ausgibt. Die Wahl der Aktivierungsfunktion für einen Layer (z.B. Dense) betrifft immer alle Neuronen/Units des Layers.

In der Regel nutzen alle *Hidden Layer* dieselbe Aktivierungsfunktion. Ein Wechsel der Funktion ist eher unüblich. Der Output Layer hingegen nutzt üblicherweise eine andere Aktivierungsfunktion. Konkret hängt die Wahl von Aufgabenstellung ab – also ob es sich bspw. um eine Binär- oder Mehrfachklassifizierung handelt.

Hidden Layer

Beim Hidden Layer kommt in der Regel eine der drei vorgestellten¹ Aktivierungsfunktionen zum Einsatz:

- Rectified Linear Unit (ReLU)
- Sigmoid
- Hyperbolic Tangent (Tanh)



¹ Selbstverständlich gibt es auch noch andere Aktivierungsfunktionen, die relevant sind. Hierbei handelt es sich um einen Forschungsbereich, der laufend neue Erkenntnisse zu Tage befördert.

Nachdem die Sigmoid- als auch die Tanh-Funktion Probleme beim Lernen bereiten können, ist die ReLU quasi die Standard-Empfehlung beim Hidden Layer.

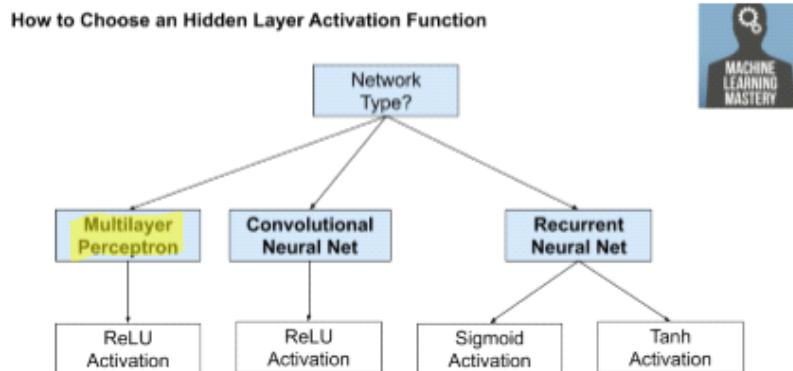


Abbildung 1: Aktivierungsfunktionen für den Hidden Layer²

Output Layer

Der Output Layer ist jener Layer in einem Neuronalen Netzwerk, der die Vorhersage ausgibt. Folgende Aktivierungsfunktionen kommen am häufigsten zum Einsatz:

- Linear
- Sigmoid
- Softmax

Welche konkret einzusetzen ist, hängt vom Aufgabentyp ab.

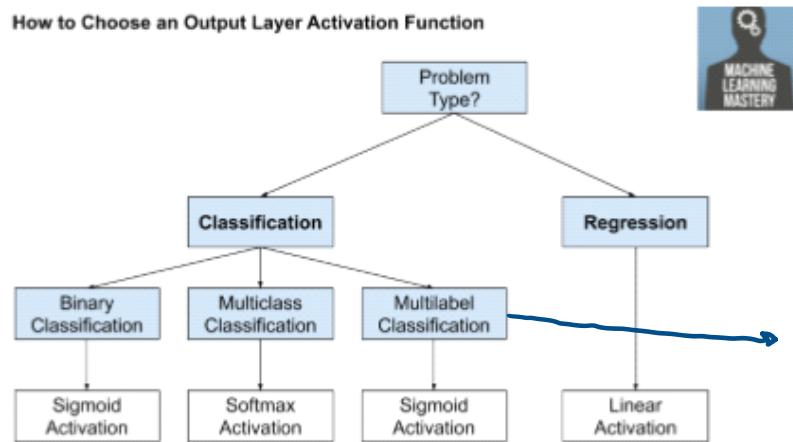


Abbildung 2: Aktivierungsfunktionen für den Output Layer²

Aufgabentyp	Aktivierung des Output-Layers	Verlustfunktion
Binärklassifizierung	sigmoid	binary_crossentropy
Singel-Label-Mehrzahlklassifizierung	softmax	categorical_crossentropy
Multi-Label-Mehrzahlklassifizierung	sigmoid	binary_crossentropy

² <https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/>

Regression (beliebige Werte)	keine	mse
Regression (zw. 0 und 1)	sigmoid	mse oder binary_crossentropy

CNN-TotalParams

Friday, May 12, 2023 1:22 PM



Jeder Filter erzeugt Featuremap

Training:

jeder Filter enthält Infos über bestimmte Merkmale



Echte

Welche im Bild erkannt wurde.



Produktion:

Jeder Filter reagiert auf unterschiedliche Merkmale.

1. Conv2D-Layer "conv2d_4":
 - Output Shape: (None, 26, 26, 32)
 - Anzahl der trainierbaren Parameter: $32 * (3 * 3 * 1 + 1) = 320$
 - 32: Anzahl der Filter
 - 3 * 3: Größe des Filters
 - 1: Anzahl der Kanäle im Eingang (Graustufenbild)
 - 1: Bias pro Filter
2. Conv2D-Layer "conv2d_5":
 - Output Shape: (None, 24, 24, 64)
 - Anzahl der trainierbaren Parameter: $64 * (3 * 3 * 32 + 1) = 18\,496$
 - 64: Anzahl der Filter
 - 3 * 3: Größe des Filters
 - 32: Anzahl der Kanäle im Eingang (Ausgabe des vorherigen Layers)
 - 1: Bias pro Filter
3. MaxPooling2D-Layer "max_pooling2d_2":
 - Output Shape: (None, 12, 12, 64)
 - Dieser Layer enthält keine trainierbaren Parameter.
4. Dropout-Layer "dropout_4":
 - Output Shape: (None, 12, 12, 64)
 - Dieser Layer enthält keine trainierbaren Parameter.
5. Flatten-Layer "flatten_2":
 - Output Shape: (None, 9216)
 - Dieser Layer enthält keine trainierbaren Parameter.
6. Dense-Layer "dense_4":
 - Output Shape: (None, 128)
 - Anzahl der trainierbaren Parameter: $9216 * 128 + 128 = 1\,179\,776$
 - 9216: Eingangsgröße des Layers (Output des vorherigen Layers)
 - 128: Anzahl der Neuronen im Dense-Layer
 - 128: Bias im Dense-Layer
7. Dropout-Layer "dropout_5":
 - Output Shape: (None, 128)
 - Dieser Layer enthält keine trainierbaren Parameter.
8. Dense-Layer "dense_5":
 - Output Shape: (None, 10)
 - Anzahl der trainierbaren Parameter: $128 * 10 + 10 = 1\,290$
 - 128: Eingangsgröße des Layers (Output des vorherigen Layers)
 - 10: Anzahl der Neuronen im Dense-Layer
 - 10: Bias im Dense-Layer

Filtrgröße Kanäle
je Filter ein Bias

damit Konsistenz: Batch Size

10% Dropout würde 6 Filter deaktivieren

Matrix Platten

$12 \times 12 \times 64$

Bias

Kappt Verbindung

10 Output Neuronen



Die Gesamtzahl der trainierbaren Parameter im Modell ergibt sich durch die Summe der Parameter in den einzelnen Schichten:

$$\text{Total params} = 320 + 18\,496 + 1\,179\,776 + 1\,290 = 1\,199\,882$$

Transfer

not t.a.

Faltungsteil: Parameter aus anderem Modell nicht überschreiben

Transfer
Learning

not t.a.
trainable

Faltungsteil : Parameter aus anderem
Modell nicht überschreiben
Dense Layer

KI4_B10-CNN1-v0.2

Friday, May 12, 2023 1:34 PM



Convolutional Neural Network I

→ Computer Vision

Um aus einem „herkömmlichen“ Neuronalen Netzwerk, wie wir es kennen, ein sog. **Convolutional Neural Network** (CNN) zu machen, braucht es zumindest eine weitere Zutat: einen *Convolutional Layer*¹. Dieser führt sog. Faltungen aus, um wichtige Merkmale (Features) im Bild zu erkennen. Üblicherweise ist dann auch noch ein *Pooling Layer* enthalten, der weniger wichtige Informationen verwirft. Von Layer zu Layer möchte man also immer konkreter werden, nur wichtige und sinnvolle Informationen behalten. Die Ausgabeschicht wird wiederum durch einen *Dense Layer* realisiert, der die eigentliche Klassifizierung vornimmt.

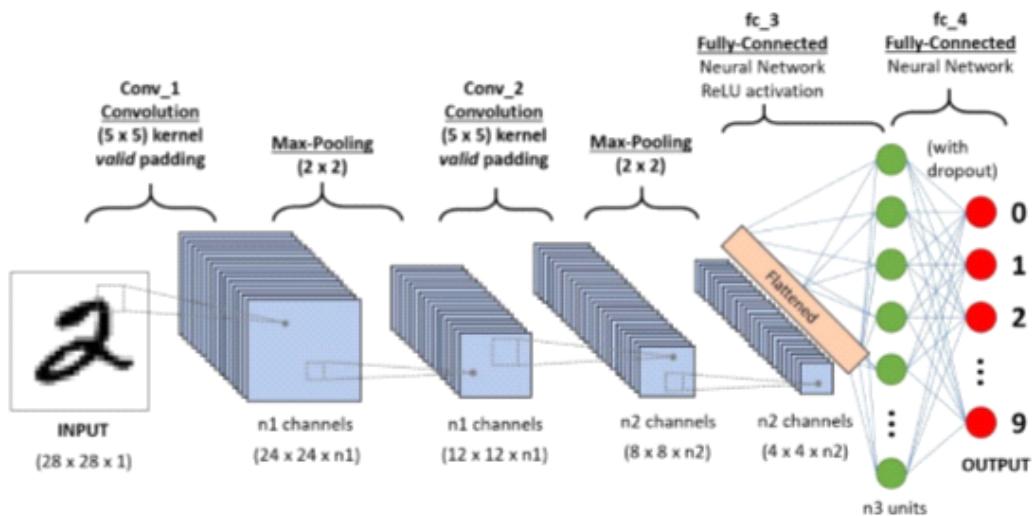


Abbildung 1: CNN-Beispiel für MNIST²

Ein CNN, das dem Namen gerecht werden will, beinhaltet also zumindest folgende Layer:

- Convolutional Layer
- Pooling Layer
- Dense Layer
- *DropOut Layer*

Ein Convolutional Layer lernt lokale Muster (*Patterns*), die von zentraler Bedeutung zu sein scheinen. Hierbei handelt es sich i.d.R. um 3x3 Pixel große Fenster (Filter, siehe Abbildung 2, Pfeil³), die aus trainierbaren Gewichten bestehen. Somit können die Filter als Ergebnis des Trainingsprozesses beim Convolutional Layer verstanden werden.

CNNs sind bei der Bildverarbeitung deshalb so effizient, weil es keine Rolle spielt, an welcher Stelle ein Muster erlernt wird. Ein Muster, das bspw. in der unteren linken Ecke erlernt wurde, kann überall erkannt werden – egal wo⁴!

¹ Eine mögliche Keras-Implementierung: https://keras.io/api/layers/convolution_layers/convolution2d/

² <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

³ Es sei an dieser Stelle erwähnt, dass das in der Regel mehr als drei sind.

⁴ Muster sind daher translationsinvariant, so der Fachbegriff.

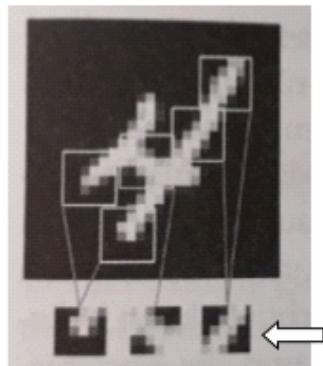


Abbildung 2: Muster

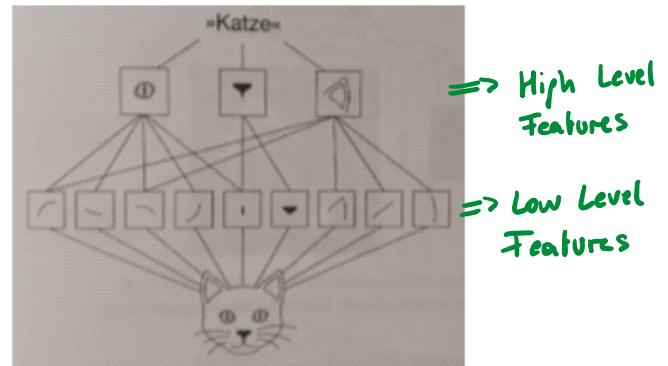


Abbildung 3: Räumliche Hierarchien

Ein weiterer interessanter Aspekt: CNNs können räumliche Hierarchien eines Musters erlernen (siehe Abbildung 3). Der erste Convolutional Layer lernt bspw. einfache Muster wie Kanten oder Formen, der zweite Layer lernt wiederum Muster, die sich aus dem Output des vorhergehenden Layers zusammensetzen.

Nehmen Sie diese Quelle <https://www.youtube.com/watch?v=FmpDlaiMleA> durch. => *Faltung*

Während der Forwardpropagation wird eine multidimensionale Variante der wichtigsten Gleichung des Unterrichts $z = w \cdot x + b$ an jeder Position berechnet, die der Filter während der Faltung entlang eines Bildes einnimmt.

0,01	0,09	0,22
-1,36	0,34	-1,59
0,13	-0,69	1,02

•

0,53	0,34	0,06
0,37	0,82	0,01
0,62	0,91	0,34

Kernel-Gewichte Pixel-Eingabe

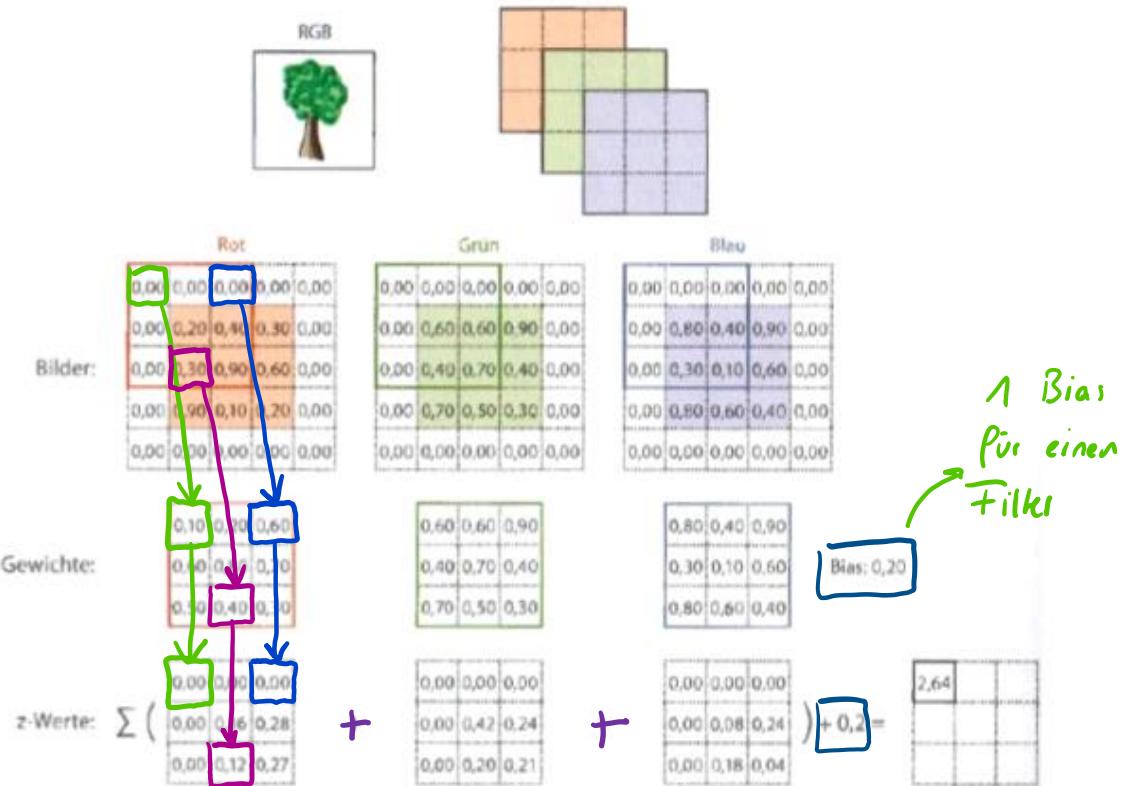
$$\begin{aligned}
 w \cdot x &= 0,01 * 0,53 + 0,09 * 0,34 + 0,22 * 0,06 \\
 &\quad + (-1,36) * 0,37 + 0,34 * 0,82 + (1,59) * 0,01 \\
 &\quad + 0,13 * 0,62 + (-0,69) * 0,91 + 1,02 * 0,34 = -0,3917
 \end{aligned}$$

Bias addieren – sagen wir z.B. 0,2: $w \cdot x + b = -0,3917 + 0,2 = -0,1917$

Mit z können wir schließlich den Aktivierungswert a berechnen, indem wir z z.B. der Aktivierungsfunktion ReLU übergeben. Die grundlegenden Rechenoperationen haben sich also nicht verändert.

Weiteres Bsp.: RGB-Bild eines Baumes von 5x5 Pixeln Größe⁵ (genau genommen 3x3 Pixel mit Padding). Wir wählen eine Filtergröße von 3x3, und da es drei Farbkanäle gibt, ist die Gewichtungsmatrix ein Array mit der Dimension (3,3,3).

⁵ Ja, das ist nicht sehr realistisch!



	Rot	Grün			Blau		
Bilder:	0,00 0,00 0,00 0,00 0,00	0,00 0,00 0,00 0,00 0,00			0,00 0,00 0,00 0,00 0,00		
	0,00 0,20 0,40 0,30 0,00	0,00 0,60 0,60 0,90 0,00			0,00 0,80 0,40 0,90 0,00		
	0,00 0,30 0,90 0,60 0,00	0,00 0,40 0,70 0,40 0,00			0,00 0,30 0,10 0,60 0,00		
	0,00 0,90 0,10 0,20 0,00	0,00 0,70 0,50 0,30 0,00			0,00 0,80 0,60 0,40 0,00		
	0,00 0,00 0,00 0,00 0,00	0,00 0,00 0,00 0,00 0,00			0,00 0,00 0,00 0,00 0,00		
Gewichte:	0,10 0,20 0,60 0,60 0,80 0,70 0,50 0,40 0,30	0,60 0,60 0,90 0,40 0,70 0,40 0,70 0,50 0,30			0,80 0,40 0,90 0,30 0,10 0,60 0,80 0,60 0,40		
							Bias: 0,20
z-Werte:	$\sum \left(\begin{array}{ c c c } \hline 0,09 & 0,12 & 0,00 \\ \hline 0,06 & 0,16 & 0,00 \\ \hline 0,00 & 0,60 & 0,00 \\ \hline \end{array} \right) + \begin{array}{ c c c } \hline 0,42 & 0,24 & 0,00 \\ \hline 0,20 & 0,21 & 0,00 \\ \hline 0,00 & 0,00 & 0,00 \\ \hline \end{array} + \begin{array}{ c c c } \hline 0,08 & 0,24 & 0,00 \\ \hline 0,18 & 0,04 & 0,00 \\ \hline 0,00 & 0,00 & 0,00 \\ \hline \end{array} \right) + 0,2 = \begin{array}{ c c c } \hline 2,64 & 4,67 & 3,58 \\ \hline 5,19 & 8,75 & 4,90 \\ \hline 3,80 & 4,66 & 2,24 \\ \hline \end{array}$						

Berechnung der Aktivierungs-Map (= Faltungs-Output) eines ConvLayers:

$$\text{Aktivierungs Map} = \frac{D - F + 2P}{S} + 1$$

D... Größe des Bildes (entweder Breite o. Höhe)

→ 5x5

F... Größe des Filters

→ 3x3

P... wie viel Padding

→ evH. 1x1

S... Schrittänge (i.d.R. 1) → default: 1

Beispiel: $\frac{5-3+0}{1} + 1 = 3$

KI4_B12-CNN2-v0.1

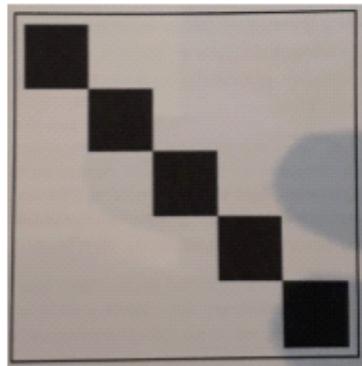
Friday, May 12, 2023 1:46 PM



Convolutional Neural Network 2

Um die Funktionsweise des Convolutional Layers im Detail zu verstehen, „arbeiten“ Sie das referenzierte Video¹ durch.

Gegeben ist ein 5x5 Pixel großes Image (unten, links). Die dazugehörigen Werte des Graustufenbildes sind der Tabelle (unten, rechts) zu entnehmen. Es gilt: 0 entspricht einem schwarzen Bildpunkt, 255 einem weißen.



0	255	255	255	255
255	0	255	255	255
255	255	0	255	255
255	255	255	0	255
255	255	255	255	0

Keras hat einen Filter (Kernel-Size = 3 x 3) wie folgt initialisiert:

x 0	x 0	x 0
x 0	x 1	x 0
x 0	x 0	x 0

Dieser wird nun mit dem Bild (vgl. Eingangsdaten), das eine Größe von 5 x 5 aufweist, gefaltet. Die Faltungsergebnis (Feature Map) weist eine Größe von 3 x 3 auf. Wie man deren Größe berechnet, haben wir bereits gelernt! Hinweis: Normalisierung – also durch 9 – ist nicht notwendig!

Faltung 1:

0	255	255	255	255
x 0	x 0	x 0		
255	0	255	255	255
x 0	x 1	x 0		
255	255	0	255	255
x 0	x 0	x 0		
255	255	255	0	255
255	255	255	255	0

=

0		

$$(0 \times 0) + (255 \times 0) + \dots + (0 \times 0) = 0$$

¹ <https://www.youtube.com/watch?v=FmpDIaiMieA>

$| = 255$

Faltung 2:

0	255	255	255	255
255	0	255	255	255
255	255	0	255	255
255	255	255	0	255
255	255	255	255	0

=

0	1	

Faltung 3:

0	255	255	255	255
255	0	255	255	255
255	255	0	255	255
255	255	255	0	255
255	255	255	255	0

=

0	1	1

Faltung 4:

0	255	255	255	255
255	0	255	255	255
255	255	0	255	255
255	255	255	0	255
255	255	255	255	0

=

0	1	1
1		

Faltung 5:

0	255	255	255	255
255	0	255	255	255
255	255	0	255	255
255	255	255	0	255

=

0	1	1
1	0	

255	255	255	255	0
-----	-----	-----	-----	---

$$l = 255$$

Faltung 6:

0	255	255	255	255
255	0	255	255	255
255	255	0	255	255
255	255	255	0	255
255	255	255	255	0

=

0	1	1
1	0	1
1	1	0

Faltung 7:

0	255	255	255	255
255	0	255	255	255
255	255	0	255	255
255	255	255	0	255
255	255	255	255	0

=

0	1	1
1	0	1
1	1	0

Faltung 8:

0	255	255	255	255
255	0	255	255	255
255	255	0	255	255
255	255	255	0	255
255	255	255	255	0

=

0	1	1
1	0	1
1	1	0

Faltung 9:

0	255	255	255	255
255	0	255	255	255
255	255	0	255	255

=

0	1	1
1	0	1
1	1	0

255	255	255	0	255
255	255	255	255	0

$$l = 255$$

Grafische Darstellung:

0	1	1
1	0	1
1	1	0

12.2.1 Nehmen Sie die 9 Faltungsoperationen vor und überführen Sie das Ergebnis in die grafische Darstellungsform, indem Sie alle jene Pixel schwarz ausmalen, die laut Feature Map auszumalen sind.

Max-Pooling

Im nächsten Schritt gilt es die Aufgabe des Max-Pooling-Layers zu übernehmen. Übertragen Sie hierzu das Ergebnis der Faltung, und zwar den jeweiligen Zahlenwert 0 oder 255:

0	1	1
1	0	1
1	1	0

Als Pool-Size (Fenstergröße) ist 2×2 zu verwenden:

Schritt 1

0	1	1
1	0	1
1	1	0

=

1

Schritt 2

0	1	1
1	0	1
1	1	0

=

1
1

Schritt 3

0	1	1
1	0	1
1	1	0

=

1
1
1

$I = 255$

Schritt 4



12.2.2 Führen Sie das Max-Pooling durch.

