

# Motivation: Biologische Neuronale Netzwerke

Neuronale Netze in der Künstlichen Intelligenz basieren auf der Inspiration durch das menschliche Gehirn.

## Biologische Neuronale Netzwerke

Das Gehirn besteht aus Milliarden von Neuronen, die durch Synapsen miteinander verbunden sind. Diese Neuronen kommunizieren durch elektrische und chemische Signale.

## Hauptmerkmale biologischer Netzwerke:

- Knoten im Netzwerk = Neuronen
- Verbindungen = Synapsen (*Gewichte* zwischen den Neuronen)
- Signale = elektrische Impulse (*Aktivierungen* durch Eingangswerte)
- Lernen = Anpassung der Synapsengewichte durch Erfahrungen (ähnlich zu ML-Trainingsprozessen)

# Artificial Neural Network (ANN)

**1** Verbindungen (Connections) → Lineare Gewichte 

Ein künstliches neuronales Netz besteht aus einzelnen Neuronen (Knoten), die über gewichtete Verbindungen miteinander verbunden sind.

 Lineares Modell der Verbindungen:

- Jede Verbindung zwischen zwei Neuronen hat ein Gewicht  $w$ , das bestimmt, wie stark die Information weitergegeben wird.
- Die Eingangswerte  $x$  werden mit den Gewichten multipliziert.
- Summe aller gewichteten Eingaben wird an das Neuron übergeben.

$$z = \sum w_i x_i + b$$

- ◆  $w$  = Gewichte zwischen den Neuronen (Verbindungsstärke)
- ◆  $x$  = Eingangswerte (Inputs)
- ◆  $b$  = Bias-Term (steuert die Aktivierungsschwelle)

## 2 Neuronen → Summation & Aktivierung ⚡

Jedes künstliche Neuron verarbeitet eingehende Signale durch zwei Hauptschritte:

### 📌 1 Summation (Input-Summe berechnen)

$$z = \sum w_i x_i + b$$

Der Input wird aufsummiert, bevor entschieden wird, ob das Neuron aktiviert wird.

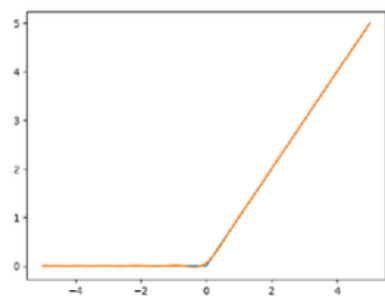
### 📌 2 Aktivierung (Entscheidung über die Aktivität des Neurons)

Das Neuron benötigt eine Aktivierungsfunktion, um eine Nicht-Linearität ins System einzubauen.

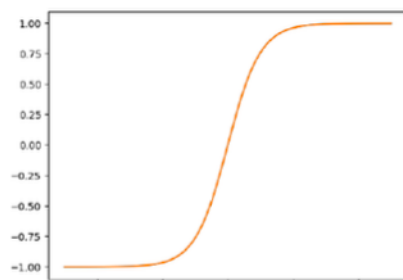
### 3 Aktivierungsfunktion → Einfache Nicht-Linearität

Da eine rein lineare Netzstruktur nicht genug erklärende Kraft hat, benötigen wir nichtlineare Aktivierungsfunktionen.

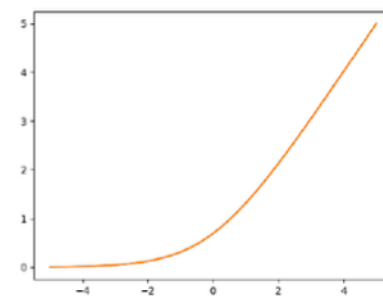
 Wichtige Aktivierungsfunktionen:



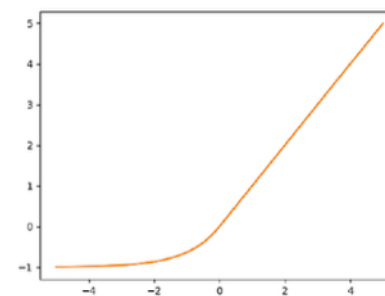
relu



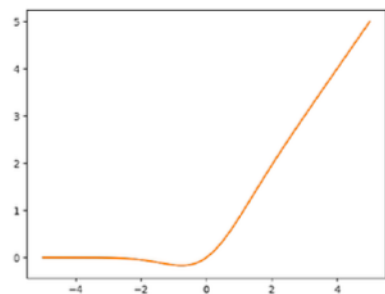
tanh



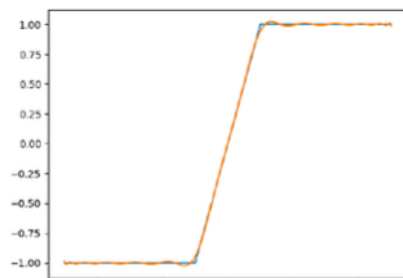
softplus



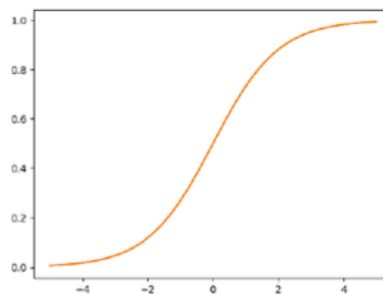
elu



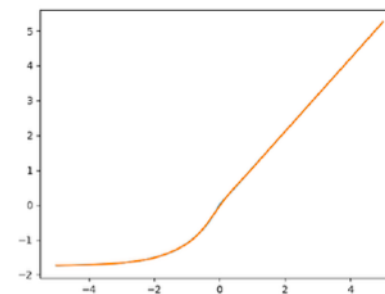
gelu



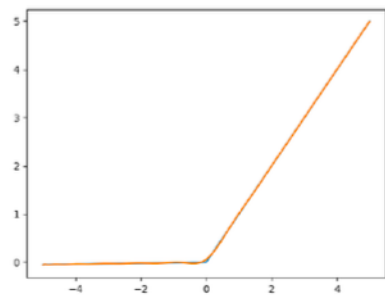
htanh



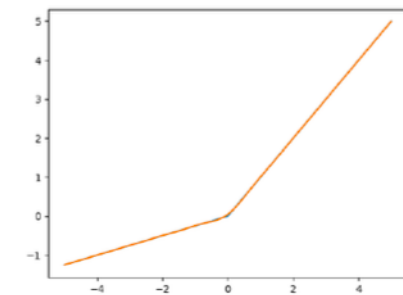
sigmoid



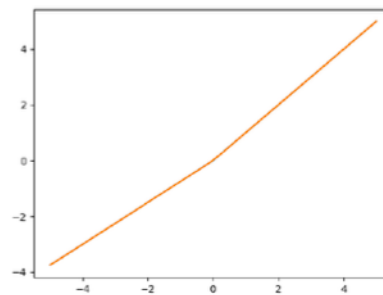
selu



lrelu

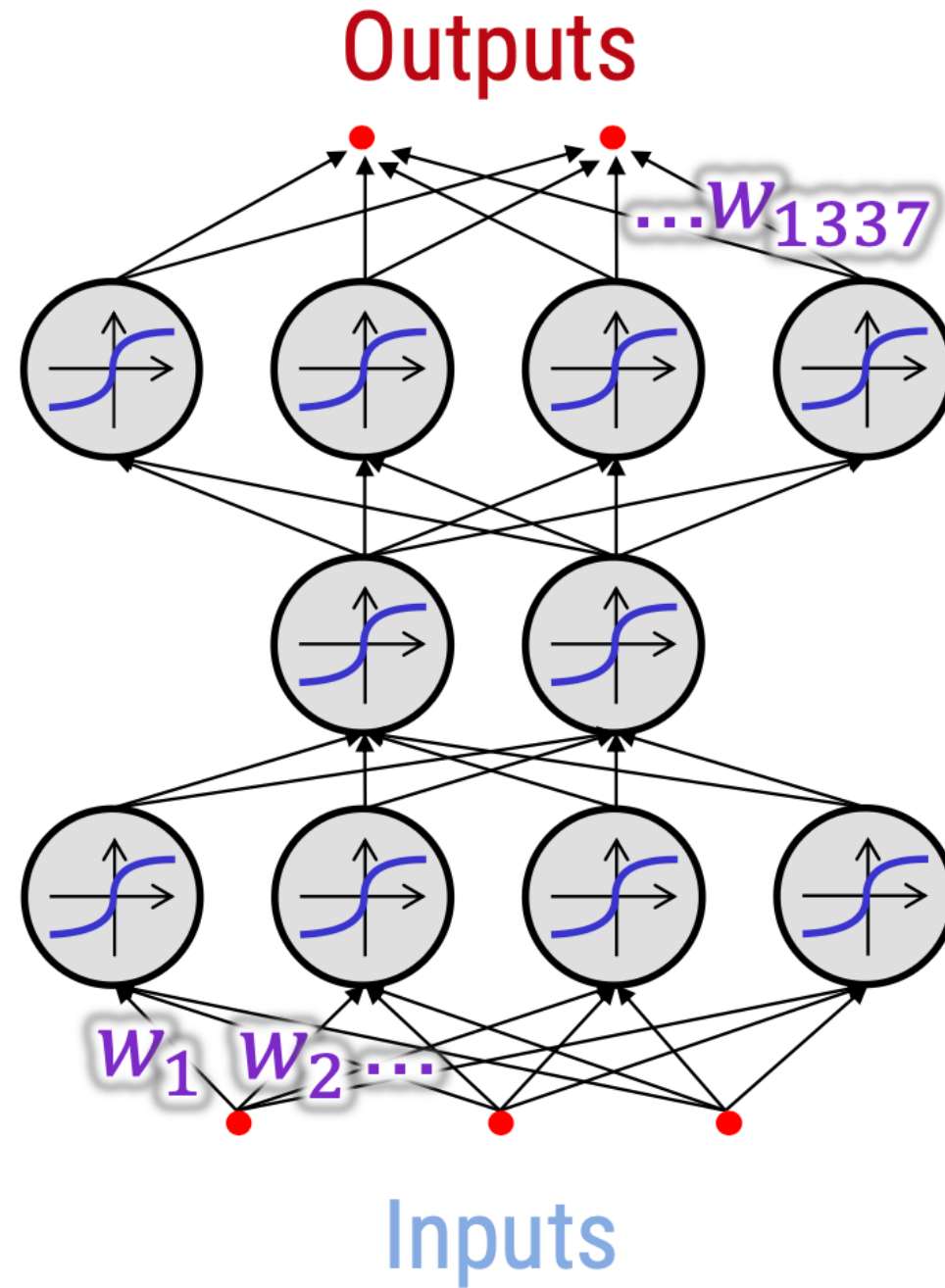


lrelu ( $\alpha = 0.25$ )



lrelu ( $\alpha = 0.75$ )

...



#### 4 Graph-Struktur → Einfaches Muster, oft „Feed-Forward“

Ein typisches neuronales Netz hat eine hierarchische, „Feed-Forward“-Struktur:

 Feed-Forward bedeutet:

✓ Eingaben werden von einer Eingabeschicht (Input Layer) über versteckte Schichten (Hidden Layers) bis zur Ausgabeschicht (Output Layer) verarbeitet.

✓ Es gibt keine Rückkopplungsschleifen (wie z. B. in rekurrenten neuronalen Netzen, RNNs).

✓ Neuronen summieren ihre *gewichteten Eingaben* 

✓ Aktivierungsfunktionen sorgen für *Nicht-Linearität* 

✓ Feed-Forward-Struktur ermöglicht Informationstransfer von *Input* → *Output* 

## 1 Vollständig verbundene Netzwerkschicht (Fully Connected Layer) 🏗️

Ein Fully Connected (FC) Network, auch als Dichtes Netzwerk (Dense Layer) bezeichnet, verbindet jedes Neuron aus einer Schicht mit jedem Neuron in der nächsten Schicht.

### 📌 Eigenschaften & Merkmale:

- ✅ Globale Verbindungen – Jedes Neuron ist mit allen anderen verbunden.
- ✅ Ableitung globaler Abhängigkeiten – Nützlich, wenn jede Eingabe jedes Ausgangs beeinflussen kann.
- ✅ Einsatzbereich: Merkmalsklassifikation (Feature Classification), z. B. MLPs (Multilayer-Perceptrons).

### 💡 Beispiel:

Klassische Feedforward-Netzwerke (MLPs).

Anwendung: Handgeschriebene Ziffernklassifikation (MNIST), allgemeine ML-Aufgaben.



## 2 Convolutional Neural Networks (CNNs)

- ◆ CNNs sind für die lokale Verbindung von Merkmalen spezialisiert.
- ◆ Sie nutzen Faltungsoperationen (Convolutions), um lokale Korrelationen in Daten zu erkennen.

### Eigenschaften & Vorteile von CNNs:

- ✓ Lernen lokaler Strukturen – Besonders gut für Bilder, Audiodaten & NLP (z.B. Texte).
- ✓ Reduzierte Anzahl an Parametern – CNNs sind durch ihre Filter sparsamer strukturiert als Fully Connected Layers.
- ✓ Translation Invariance – Identifiziert Muster an beliebigen Positionen im Bild.

### Beispiel:

Bilderkennung z.B. in ResNet, VGG, AlexNet

Textdaten-Aufbereitung in NLP-Anwendungen

Spracherkennung (Audio), z.B. WaveNet für Sprachsynthese

### 3 Rekurrente neuronale Netzwerke (RNNs)

RNNs unterscheiden sich von Feedforward-Netzen dadurch, dass sie eine gedächtnisähnliche Speicherung vorheriger Werte bieten.

 Eigenschaften von RNNs:

- ✓ Erinnerung an vergangene Zustände – Informationen aus vorherigen Zeitschritten beeinflussen zukünftige Berechnungen.

- ✓ Einsatz für sequentielle Daten (!)

- ✓ Anwendung für Zeitreihen, NLP & Vorhersagemodelle

 Beispielhafte Anwendungen:

Sprachverarbeitung (NLP) → Machine Translation

Zeitreihenanalysen → Aktienmarktvorhersage

# Sprache der Neuronalen Netzwerke – Grundbegriffe

## Input – Eingabe

 Was ins Netzwerk eingeht

- Dies sind die Rohdaten oder Merkmale (Features), die dem Netzwerk bereitgestellt werden.
- Z. B. ein Bild, ein Satz oder ein Tondatensatz.
- Repräsentation oft als Vektor oder Matrix.

 Beispiel:

- In der Bildverarbeitung: Jeder Pixelwert eines Bildes.
- In der Spracherkennung: Eine Zahlenrepräsentation von Wörtern (Word Embeddings).

## 2 Output – Ausgabe

 Was das Netzwerk nach der Verarbeitung ausgibt.

- Am Ende eines neuronalen Netzes erfolgt eine Entscheidung oder eine Berechnung.
- Es kann sein:
  - Eine Klassifikation: "Katze" oder "Hund"?
  - Eine numerische Vorhersage: Aktienkurs steigt oder fällt?
  - Ein generierter Satz in der maschinellen Übersetzung.

### 3 Features & Hidden Layers – versteckte Merkmale 🏛️

📌 Werte in den inneren Neuronen des Netzwerks.

- "Features" sind abstrahierte Informationen aus den Daten (= Merkmale), die in verdeckten Schichten entwickelt werden.
- "Hidden Layers" → Zwischenschichten, in denen das Netzwerk Merkmale extrahiert. Jede Schicht hat eine eigene Größe (z.B. 128, 512, 1024 Neuronen).

💡 Warum?

- Ein einfaches Modell erkennt nur grobe Muster.
- Mehr Hidden Layers = "Tiefe" im Lernen, sodass komplexe Muster erkannt werden können.

#### 4 Feed Forward Network – sequentielle Verarbeitung

 Einfachste Art neuronaler Netzwerke → Daten fließen nur "vorwärts".

Struktur:

- Input-Layer → Hidden Layers → Output-Layer
- Es gibt keine Rückkopplung oder Speicherung vorheriger Werte (wie bei RNNs).

 Beispiel für Anwendung

- Objektklassifikation (z.B. Erkennung von Hund vs. Katze aus Bildern).
- Kreditrisikobewertung (basierend auf statistischen Daten).

**5** Layer – eine Verarbeitungsebene in einem Feedforward-Netzwerk 

 Jede Schicht im NN führt eine Berechnung durch.

- Eine Schicht nimmt Eingaben entgegen, verarbeitet sie, gibt Ergebnisse weiter.
- In tiefen Netzen: Viele Layer = Hierarchische Abstraktion der Daten.

 Typen von Layern:

- ✓ Fully Connected Layer (dichte Verbindung)
- ✓ Convolutional Layer (für Bildverarbeitung)
- ✓ Recurrent Layer (für Zeit- und Sequenzdaten)

6 Preactivation – Zahlen, bevor die Nicht-Linearität angewendet wird ⚙️

📌 "Rohwert" am Eingang der Aktivierungsfunktion.

- Jeder Neuron berechnet:

$$z = \sum w_i x_i + b$$

- "Preactivation": die Summe, bevor die Aktivierungsfunktion angewendet wird.



## 7 Activation – Aktivierungswert nach Nicht-Linearität ⚡

📌 Nicht-lineare Transformation des neuronalen Outputs.

💡 Es gibt zwei Perspektiven:

"Activation" kann bedeuten:

- Zahlen, die nach der Aktivierungsschicht herauskommen.
- Beispiel: Sigmoid oder ReLU verändert  $z$  zu einem neuen Wert.

"Activation" kann auch bedeuten, ob ein ReLU aktiv wurde

- Bei ReLU ( $\text{ReLU}(x) = \max(0, x)$ ) wird alles unter 0 auf 0 gesetzt.
- Man kann entscheiden, ob ein Neuron "aktiviert" wurde oder nicht.

# NN-Talk

Begriff	Bedeutung
<b>Input</b>	Die Ausgangsdaten, die das NN verarbeitet.
<b>Output</b>	Das Endergebnis des Netzwerks.
<b>Feature / Hidden Layer</b>	Zwischenergebnisse abstrakte Merkmale
<b>Layer</b>	Berechnungsschritt innerhalb des NN.
<b>Feed Forward</b>	Der normale, sequenzielle Datenfluss eines Netzwerks.
<b>Preactivation</b>	Wert, bevor Aktivierungsfunktion
<b>Activation</b>	Der transformierte Wert nach der Aktivierungsfunktion.

## 1 Stack of Matrices – Mehrere Matrixberechnungen 🏗️

📌 Jede Schicht im neuronalen Netz kann als Matrixmultiplikation betrachtet werden:

$$Z = WX + b$$

Hierbei gilt:

$X$  = Eingangsdaten (Feature-Werte)

$W$  = Gewichtungsmatrix (lernbare Parameter)

$b$  = Bias (Steuerung der Schwellenwerte)

$Z$  = Summierte Eingangswerte (Preactivation)

💡 Was passiert hier?

✅ Jede Schicht transformiert ihre Eingabe durch eine Matrixmultiplikation

✅ Mehrere Schichten = Hierarchische Kombinationen einfacher Berechnungen ⌚

## 2 Non-Linearities – Nicht-Linearitäten für komplexe Muster 🧐

### 📌 Warum Nicht-Linearitäten?

Wenn wir nur lineare Transformationen nutzen, wäre unser Netzwerk nur eine komplexe lineare Funktion – und das wäre zu limitiert.

🌟 Deshalb führen wir Nicht-Linearitäten (Activation Functions) ein, z.B.:

#### ◆ ReLU (Rectified Linear Unit) 💡

$$f(z) = \max(0, z)$$

- Einfache, schnelle und effiziente Nicht-Linearität
- Wandelt negative Werte in 0 um → Spart Rechenzeit & verbessert Training
- In fast allen modernen Deep Learning-Modellen verwendet!

Andere Nicht-Linearitäten:

- ✓ Sigmoid: Wahrscheinlichkeitsprobleme (eher veraltet wegen Saturationseffekt)
- ✓ Tanh: Gut für Werte um 0, funktioniert ähnlich wie Sigmoid

Die Qualität des maschinellen Lernens hängt stark von der Datenvorbereitung ab. Ein berühmtes Zitat in der KI lautet:

- "Garbage in, garbage out"   

Schlechte Daten führen zu schlechten Modellen, egal wie gut der Algorithmus ist.

Daher sind Datenvorbereitung, Feature Engineering und Visualisierung entscheidende Schritte im Machine Learning-Prozess.

# 1 Datenvorbereitung – Quellen, Bereinigung und Transformation

## a) Datenquellen

Strukturierte Daten aus verschiedenen Quellen, z. B.:

- CSV-Dateien / Excel 
- Datenbanken (SQL, NoSQL) 
- APIs & Web Scraping 
- Sensorsysteme (z. B. IoT-Geräte, Kamerabilder, Audiodaten) 
- Open Data (z. B. Kaggle, UCI Machine Learning Repository)

## b) Datenbereinigung (Data Cleaning)

Rohdaten sind oft unvollständig, inkonsistent oder fehlerhaft. Typische Bereinigungsaufgaben:

- ✓ Fehlende Werte behandeln (Imputieren mit Median, Mittelwert oder Entfernen)
- ✓ Doppelte Einträge entfernen
- ✓ Outlier (Ausreißer) identifizieren & behandeln
- ✓ Konsistenz überprüfen (z. B. Datumsformate, Codierungen)

## ✅ Fehlende Werte behandeln (Imputation oder Entfernung)

In fast jedem Datensatz gibt es fehlende Werte (Missing Values), die durch verschiedene Gründe entstehen können, z. B.:

- Fehlende Sensorwerte in IoT-Daten 🌡️
- Nicht ausgefüllte Felder in Umfragen 📋
- Datenverluste beim Scraping oder bei der Erhebung 🌐

Fehlende Werte können Probleme verursachen:

- ❌ Mathematische Operationen können fehlschlagen
- ❌ Maschinelle Lernmodelle können keine fehlenden Werte verarbeiten
- ❌ Statistische Verzerrungen können entstehen



Methode	Beschreibung	Wann verwenden?
Einfaches Löschen	Entfernen von Zeilen oder Spalten mit fehlenden Werten	Bei wenigen fehlenden Werten 🔥
Mittelwert (Mean)	Ersetzen durch den Durchschnitt des Features	Wenn die Werte <b>gleichmäßig verteilt</b> sind ✅
Median	Ersetzen durch den Wert in der Mitte der Datenverteilung	Wenn <b>Ausreißer</b> vorhanden sind 🛡️ (z.B. Einkommen)
Modus	Ersetzen fehlender Werte durch den häufigsten Wert in der Spalte	Für <b>kategorische Daten</b> (z.B. Farben, Kategorien) 🎨
Vorhersagebasiert	Maschinelles Lernen zur Schätzung nutzen	Wenn viele Werte fehlen und es genügend Daten für Vorhersagen gibt 🔍

## 2 Fehlende Daten in Pandas-DataFrames finden

```
import pandas as pd

# Beispieldaten
data = {'Name': ['Anna', 'Ben', 'Carlos', 'Diana'],
        'Alter': [25, None, 30, 22],
        'Einkommen': [50000, 48000, None, 52000]}

df = pd.DataFrame(data)

# Fehlende Werte pro Spalte zählen
print(df.isnull().sum())
```

## Fehlende Werte entfernen (Drop)

```
df_cleaned = df.dropna() # Löscht jede Zeile mit fehlenden Werten
df_cleaned = df.dropna(axis=1) # Löscht jede Spalte mit fehlenden Werten
```

✓ Imputation mit Mittelwert / Median / Modus (Numerische Werte)

```
df['Alter'].fillna(df['Alter'].mean(), inplace=True) # Mittelwert  
df['Alter'].fillna(df['Alter'].median(), inplace=True) # Median
```

✓ Imputation mit Modus (Kategoriale Werte)

```
df['Kategorie'].fillna(df['Kategorie'].mode()[0], inplace=True)
```

# Doppelte Einträge entfernen

```
import pandas as pd

# Beispiel-Datenframe
data = {'Name': ['Anna', 'Ben', 'Carlos', 'Anna'],
        'Alter': [25, 30, 22, 25],
        'Einkommen': [50000, 48000, 52000, 50000]}

df = pd.DataFrame(data)

# Prüfen, ob doppelte Zeilen existieren
print(df.duplicated())
```

```
df_cleaned = df.drop_duplicates()
```

```
df_cleaned = df.drop_duplicates(subset=['Name'])
```

## ✅ Outlier (Ausreißer) identifizieren & behandeln

- Ausreißer (Outlier) sind Werte, die stark von den meisten anderen Datenpunkten abweichen.
- Sie können durch Messfehler, extreme Ereignisse oder echte Besonderheiten (z. B. Millionäre in Einkommenstabellen) entstehen.

## 1 Warum sind Ausreißer ein Problem?

- ✗ Beeinflussen Mittelwerte & Standardabweichungen stark
- ✗ Können Machine-Learning-Modelle verzerren
- ✗ Falsche Entscheidungsfindung, wenn nicht richtig behandelt

## 2 Outlier identifizieren 🕵️

### ✓ a) Visuelle Methoden

Histogramme & Boxplots helfen, Ausreißer zu sehen

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Beispiel-Datensatz
data = {'Alter': [25, 30, 22, 99, 27, 26, 21, 200, 32]}
df = pd.DataFrame(data)

# Boxplot erstellen
sns.boxplot(x=df['Alter'])
plt.show()
```

! Hier erkennt man Werte wie 99 oder 200 als potenzielle Ausreißer.

## ✅ b) IQR-Methode (Interquartilsabstand)

Outlier-Definition:

Ein Datenpunkt ist ein Ausreißer, wenn er außerhalb des Intervalls

$[Q1 - 1.5 * IQR, Q3 + 1.5 * IQR]$  liegt

$$IQR = Q3 - Q1$$

```
import numpy as np

Q1 = df['Alter'].quantile(0.25) # 25%-Quantil
Q3 = df['Alter'].quantile(0.75) # 75%-Quantil
IQR = Q3 - Q1 # Interquartilsabstand bestimmen

# Nach Outlier-Kriterien filtern
outlier_mask = (df['Alter'] < (Q1 - 1.5 * IQR)) | (df['Alter'] > (Q3 + 1.5 * IQR))
df_outliers = df[outlier_mask]

print(df_outliers) # Zeigt nur die Ausreißer an
```

### ✓ c) Z-Score-Methode (Standardabweichung)

Wenn Daten normalverteilt sind, kann der Z-Score helfen:

- Z-Score misst, wie viele Standardabweichungen ein Wert über/unter dem Mittelwert liegt.
- Werte mit  $|Z| > 3$  gelten oft als Ausreißer.

```
from scipy import stats

df['Z-Score'] = stats.zscore(df['Alter'])
df_outliers = df[(df['Z-Score'].abs() > 3)]
print(df_outliers)
```



### Beispiel – Outlier entfernen

```
df_no_outliers = df[~outlier_mask]  # ~ bedeutet "nicht" in Pandas
```

### Beispiel – Clipping auf 5%- & 95%-Quantil

```
low, high = df['Alter'].quantile([0.05, 0.95])  
df['Alter'] = df['Alter'].clip(lower=low, upper=high)
```

*Aufpassen: Nicht jeder Ausreißer ist ein Fehler – manchmal sind sie die spannendsten Daten!*

# Konsistenzprobleme in Daten

Problem	Beispiel
Uneinheitliche Datums-/Zeitformate	01/02/2023 vs. 2023-02-01 vs. 02-01-23
Unterschiedliche Groß-/Kleinschreibung	"Berlin" vs. "berlin"
Rechtschreibfehler in Kategorien	"Frau" , "fraU" , "Fr"
Verschiedene Einheiten	3.5 km vs. 3500 m vs. 3.5 Kilometer
Problematische Codierungen	ÄÖÜ wird als Ã,,Ã-Ãæ gespeichert
Mehrere fehlerhafte Werte für NULL	NULL , N/A , "undefined" , NaN
Doppelte Einträge mit Abweichungen	"Müller GmbH" vs. "Müller GmbH."

### ⚙️ c) Daten-Transformation und -Skalierung

Maschinen lernen besser, wenn Daten in einer gleichmäßigen Skala vorliegen. Dafür nutzen wir:

- ✅ Normierung (MinMax Scaling): Werte zwischen 0 und 1 skalieren
- ✅ Standardisierung (Z-Transformation): Werte auf Mittelwert 0 und Standardabweichung 1 transformieren
- ✅ One-Hot-Encoding für Kategorische Daten: Kategorien in binäre Werte umwandeln

✅ Normierung (MinMax Scaling): Werte zwischen 0 und 1 skalieren 🚀

📌 Warum Normierung?

- Maschinelle Lernmodelle arbeiten oft besser, wenn Features auf ähnliche Skalen gebracht werden.
- MinMax-Skalierung transformiert die Werte auf ein festes Intervall  $[0,1]$  oder  $[-1,1]$

Normierte Daten = Stabilere, schneller konvergierende Modelle!

✓ Ideal für neuronale Netze & optimierungsbasierte Algorithmen (z. B. SGD, K-Means, SVMs)

## Warum Standardisierung?

Manche Modelle (z. B. lineare Regression, kNN, PCA) funktionieren besser mit standardisierten Daten.

◆ Z-Transformation (Standardisierung) bringt Daten auf eine Normalverteilung und sorgt für:

✓ Mittelwert = 0

✓ Standardabweichung = 1

## ✅ One-Hot-Encoding für Kategorische Daten

### ◆ 1. Ausgangsdaten (Bevor OHE)

ID	Farbe
1	Rot
2	Blau
3	Grün
4	Rot
5	Blau

## ◆ 2. Nach One-Hot-Encoding (OHE)

ID	Farbe_Rot	Farbe_Blau	Farbe_Grün
1	1	0	0
2	0	1	0
3	0	0	1
4	1	0	0
5	0	1	0

### ◆ 3. Label-Encoding (Alternative zu OHE)

ID	Farbe	Farbe_LabelEncoded
1	Rot	2
2	Blau	0
3	Grün	1
4	Rot	2
5	Blau	0




# Bedeutung und Techniken des Feature Engineering

Ein gut designtes Feature(**Merkmal**) kann den Unterschied zwischen einem schlechten und einem exzellenten Modell ausmachen.

 Was ist Feature Engineering?

 Feature Engineering = Erstellung neuer oder besserer Merkmale für Modelle

 Ziel: Die zugrunde liegenden Muster in den Daten besser verständlich machen!

 a) Funktionale Transformationen von Features

✓ Log-Transformation: Um extreme Verteilungen (z.B. Einkommen) zu glätten.

✓ Polynomiale Merkmale: Kombinierte Features (z.B.  $x_1^2$ ,  $x_2^2$ ,  $x_1 \cdot x_2$ )

✓ Bucketizing (Binning): Beispiele in Gruppen einteilen (z. B. Altersklassen "Jung", "Mittel", "Alt")

## b) Feature Selektion (Auswahl wichtiger Features)

Nicht alle Features bringen einen Mehrwert – zu viele Features können sogar das Modell verschlechtern.

- ✓ Korrelation berechnen – Features mit niedriger Korrelation entfernen
- ✓ PCA (Principal Component Analysis) – Dimensionsreduktion
- ✓ LASSO-Regression – Selektiert relevante Merkmale durch Regularisierung

## c) Feature Generierung aus bestehenden Daten

Manchmal können neue Features aus bestehenden generiert werden:

- ✓ Zeitbasierte Features: Wochentag/Monat/Jahreszeit extrahieren aus Datum
- ✓ Relationsfeatures: Preis-zu-Qualität-Verhältnis in einem Online-Shop
- ✓ Textanalyse (NLP): Anzahl der Wörter, Sentiment-Score

# Datenvisualisierung für bessere Analysen

Datenvisualisierung hilft uns, Muster zu erkennen und Datenprobleme zu finden.

📌 a) Histogramme & Boxplots – Verteilungen anzeigen

Hilft zu erkennen, ob Features normalisiert oder skaliert werden müssen.

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.histplot(df['Feature1'])
plt.show()

sns.boxplot(x=df['Feature1'])
plt.show()
```

## b) Korrelationsmatrix – Wie hängen Features zusammen?


```
import seaborn as sns

plt.figure(figsize=(10,6))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm")
plt.show()
```

### c) Scatter Plots – Beziehung zwischen Features analysieren

```
sns.scatterplot(x=df['Feature1'], y=df['Feature2'], hue=df['Target'])  
plt.show()
```

Fazit: Warum ist Datenvorverarbeitung so wichtig?

- ✓ Verunreinigte Daten = Schlechtes Modell 
- ✓ Gutes Feature Engineering = Stärkeres Modell
- ✓ Datenvisualisierung zeigt uns Muster, die wir übersehen hätten!

## ✅ Hauptkomponentenanalyse (PCA) – Dimensionen reduzieren

### 📌 Warum PCA?


Die Hauptkomponentenanalyse (PCA) ist eine Technik zur Dimensionsreduktion.  
Ziel: Daten mit weniger Features (Dimensionen) repräsentieren, ohne dabei zu viel Information zu verlieren!

#### ◆ Anwendungsfälle:

- ✓ Datenvisualisierung: Reduktion auf 2D/3D für Plots
- ✓ Overfitting vermeiden: Entfernt unnötige Features
- ✓ Effizienz steigern: Weniger Dimensionen = Schnellere Modelle
- ✓ Feature Extraction: Kombiniert vorhandene Features zu neuen repräsentativen Komponenten

## Wie funktioniert PCA?

- 1 Standardisierung der Daten (Mittelwert = 0, Std.-Abw. = 1)
- 2 Berechnung der Kovarianzmatrix (zeigt, wie Features zusammenhängen)
- 3 Eigenwertzerlegung (Um Eigenvektoren zu berechnen)
- 4 Transformation auf die Hauptkomponenten (rotation des Raumes zu einer neuen Achse)

PCA wandelt ursprüngliche Features in neue, unkorrelierte Achsen um, wobei die erste Hauptkomponente (PC1) die maximale Varianz enthält! 

# Random Forest Klassifikation

## Theorie & Grundlagen

- **Random Forest** ist ein **Ensemble-Lernverfahren**, das mehrere Entscheidungsbäume kombiniert.

 **Ziel:** Ein robustes Modell erstellen, das Overfitting vermeidet!



## 1 Wie funktioniert Random Forest? 🌲

### 🎯 Grundidee:

- **Mehrere Entscheidungsbäume** gemeinsam nutzen (*Ensemble Learning*).
- Jeder Baum trifft seine eigene Entscheidung → **Mehrheit gewinnt** (Klassifikation)
- **Stichproben mit Zurücklegen** ( `Bootstrapping` ), um verschiedene Trainingssätze zu erzeugen.

# Wie funktioniert ein Entscheidungsbaum?

Ein Entscheidungsbaum zerlegt die Eingabedaten hierarchisch in Teilmengen, indem er an Knotenschnittpunkten Entscheidungen trifft, bis jede Beobachtung einer bestimmten Klasse zugeordnet ist.

💡 Hauptbestandteile eines Entscheidungsbaums:

- Wurzelknoten (Root): Der Startpunkt des Baums (Gesamtdaten).
- Zweig (Branch): Verbindung zwischen den Knoten.
- Entscheidungsknoten (Internal Node): Stellt eine Bedingung (z. B.  $X > 5$ ?) dar.
- Blätter (Leaf Nodes): Endpunkte, die eine finale Klasse oder Vorhersage ausgeben.

# Entscheidungsbaum

```

    Ist es Wochenende? (root)
      /      \      (branch)
    Ja      Nein (node)
    /      \
  Ist Wetter gut?  Bleib zu Hause (node)
  /      \
Ja      Nein (node)
/      \
Geh hin  Bleib zu Hause (leaf)

```

## 2 Entscheidungsbäume in Random Forest

Ein einzelner Entscheidungsbaum nutzt folgendes Prinzip:

1. Starte mit der gesamten Datenmenge.
2. Teile die Daten an einem **optimalen Split-Punkt** auf.
3. Wiederhole den Vorgang für jede Teilmenge (rekursive Aufteilung).
4. Blätterknoten enthalten finale Klassifikation/Zielwert.

### Problem:

- Einzelne **Entscheidungsbäume** **overfitten oft!**
- Lösung? → **Random Forest: Kombiniere mehrere zufällige Bäume!**

### 3 Wichtige Konzepte

#### Bootstrapping & Bagging

- **Bootstrapping:** Ziehe zufällige Stichproben aus den Trainingsdaten (mit Zurücklegen).
- **Bagging (Bootstrap Aggregation):**
  - Trainiere **mehrere Entscheidungsbäume** auf unterschiedlichen Datenproben.
  - Finalentscheidung: **Mehrheitsvoting (Klassifikation) / Durchschnitt (Regression)**.

$$y_{pred} = \frac{1}{N} \sum_{i=1}^N Tree_i(X)$$

 Dadurch ist **Random Forest robuster gegenüber Overfitting!**

## 4 Vor- & Nachteile von Random Forest

### ✓ Vorteile:




- Sehr genau dank Ensemble-Ansatz.
- Robust gegenüber Overfitting.
- Funktioniert mit vielen Datentypen (numerisch, kategorisch).
- Kann Feature-Wichtigkeit bewerten!

### ⚠ Nachteile:

- Langsamer als einfache Entscheidungsbäume.
- Speicher- & CPU-intensiv bei großen Datenmengen.

# Unüberwachtes Lernen

## Was ist unüberwachtes Lernen?

- Keine Labels  $y$ , nur Eingabedaten  $X$
- Das Modell **findet Muster selber**
- Einsatzgebiete:
  -  **Clustering** (Gruppierung von Daten)
  -  **Dimensionalitätsreduktion**
  -  **Anomalie-Erkennung**

# 1 Clustering – K-Means Algorithmus

- ◆ **Ziel:** Gruppiere Datenpunkte in  $k$  Gruppen

Gruppenbildung geschieht automatisch anhand der Ähnlichkeit von Merkmalen.

- ◆ **Anwendungsfälle:**

- Datensegmentierung
- Medizinische Diagnosen
- Produktempfehlungen
- Anomalieerkennung



## 2 Wie funktioniert K-Means?

Die K-Means-Algorithmus folgt diesen Schritten:

- 📌 1.  $k$  definieren: Anzahl der Cluster  $k$  wird festgelegt
- 📌 2. Zufällige Startpunkte:  $k$  zufällige Punkte als Cluster-Zentren setzen
- 📌 3. Zuweisung von Punkten: Jeder Punkt kommt zu seinem nächsten Cluster-Zentrum
- 📌 4. Zentren aktualisieren: Mittelwert der zugeordneten Punkte berechnen
- 📌 5. Wiederholen, bis sich die Zentren nicht mehr ändern

🔄 Das Verfahren wiederholt sich, bis die Cluster stabil bleiben oder eine Abbruchbedingung erreicht wird.

