

# Grundlagen Supervised & Unsupervised Learning

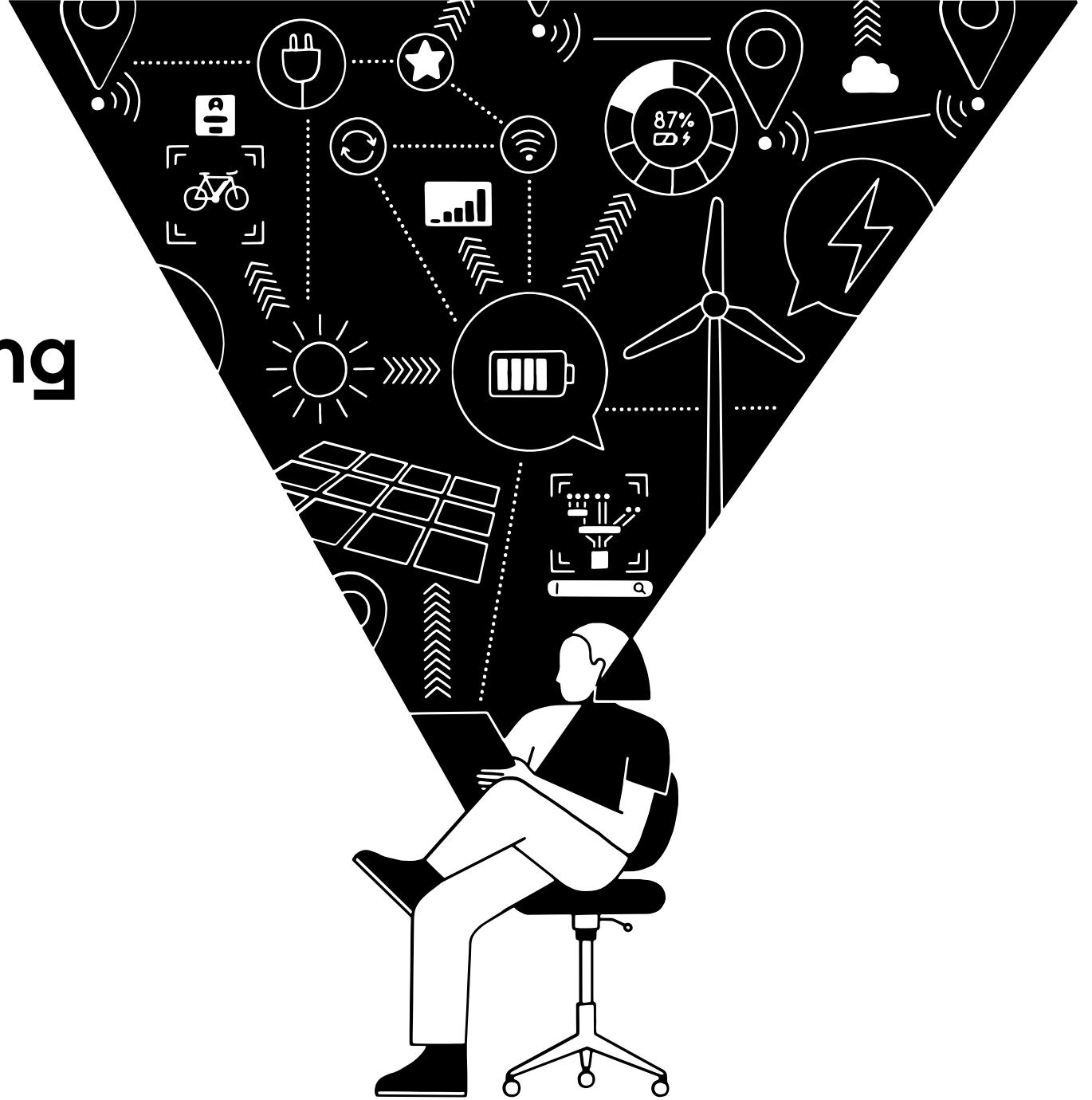
## Tag 1

Mit Anwendungsbeispielen in TensorFlow Keras



Tobias Krebs

**exeta**





# TOBIAS KREBS

## Associate Data Engineer

Tobias Krebs hat seit mehr als 3 Jahren Erfahrung in der Verarbeitung von Daten auf unterschiedlichste Weise. Diese sammelte er u.a. an der Informatikfakultät in Kopenhagen, dem Information Systems Chair der Humboldt Universität Berlin und in internen Projekten. Der Fokus liegt dabei im Bereich des Machine Learnings und Data Engineerings.

### Biografie

- EXXETA
- M.Sc Economics Humboldt Universität zu Berlin
- Information Systems Chair Humboldt Universität zu Berlin
- Computer Science Fakultät Universität Kopenhagen

### Beratungskompetenz

- Data Science, Machine Learning, Deep Learning, Data Engineering
- IT Expertise u.a. in Python (Pandas, scikit-learn, tensorflow, keras, pytorch, numpy)
- AWS Cloud: Elastic Computing 2 (EC2)

### Sprachen

- Deutsch, Englisch

### Auszug relevante Projekterfahrung

#### **Data Engineer | Automatisierte Bereitstellung von GPU Cloud Umgebungen**

- Auswahl geeigneter Umgebungen
- Automatisierte Erstellung von passenden Machine Images für Data Science Anwendungen
- Erstellung eines konfigurierbaren Skripts zum Starten benötigter Computing Umgebungen

#### **Data Scientist | Masterarbeit: Predicting media bias of news articles using deep learning**

- Ausführliche Literatur Recherche zum Thema
- Aufbereiten der Daten für NLP Usecase
- Anwenden und vergleichen verschiedener Machine Learning Modelle (Random Forest, LSTM, BERT)

#### **Data Scientist | Erstellen von Seminaraufgaben für Python und Machine Learning**

- Erstellen einer Übersicht aller benötigten Aufgaben
- Schreiben der einzelnen Jupyter Notebooks mit Aufgaben und Lösungen

# Exxeta in Zahlen



Gegründet 2005



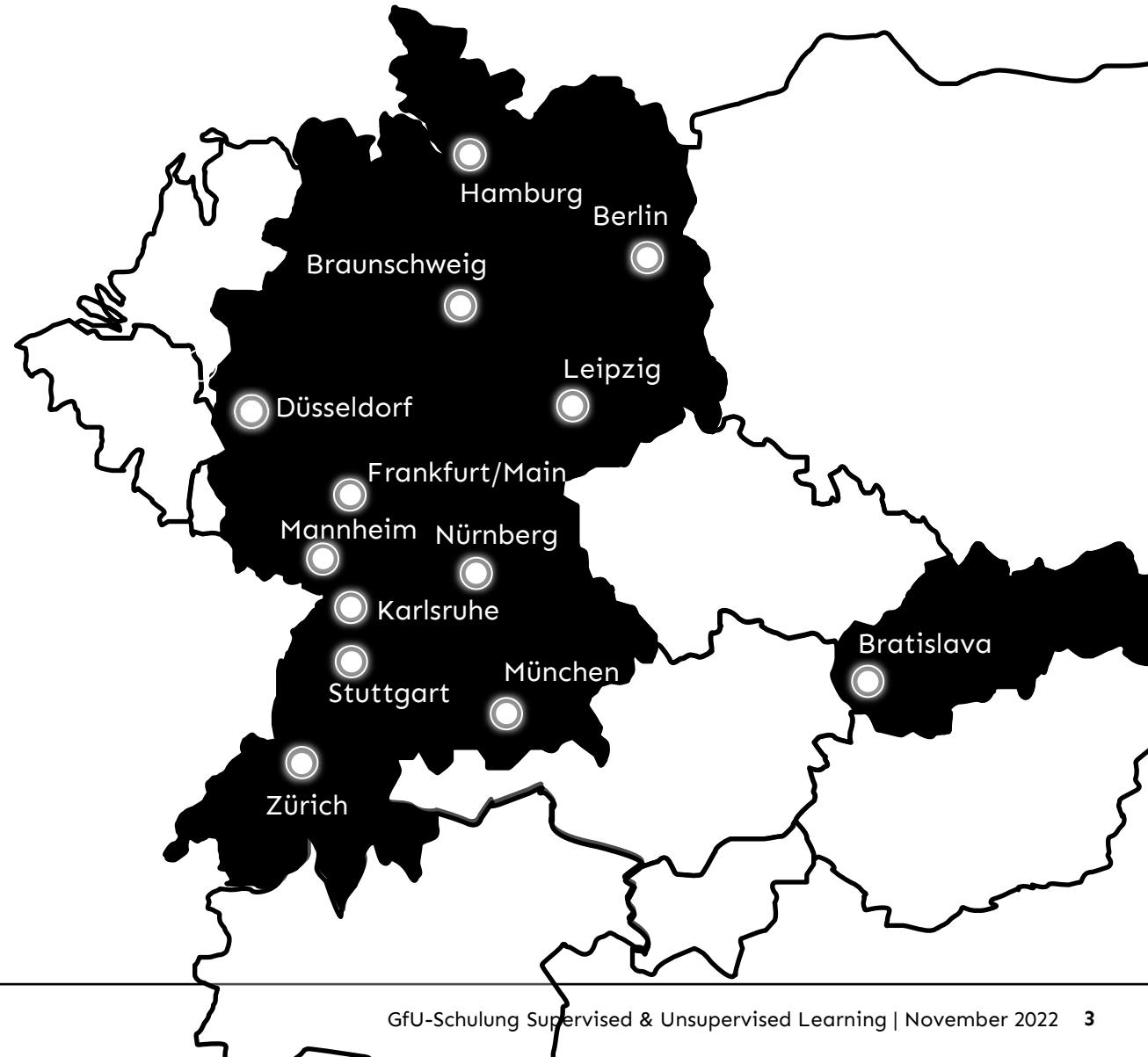
Mitarbeitende > 1.100



Umsatz > €100M

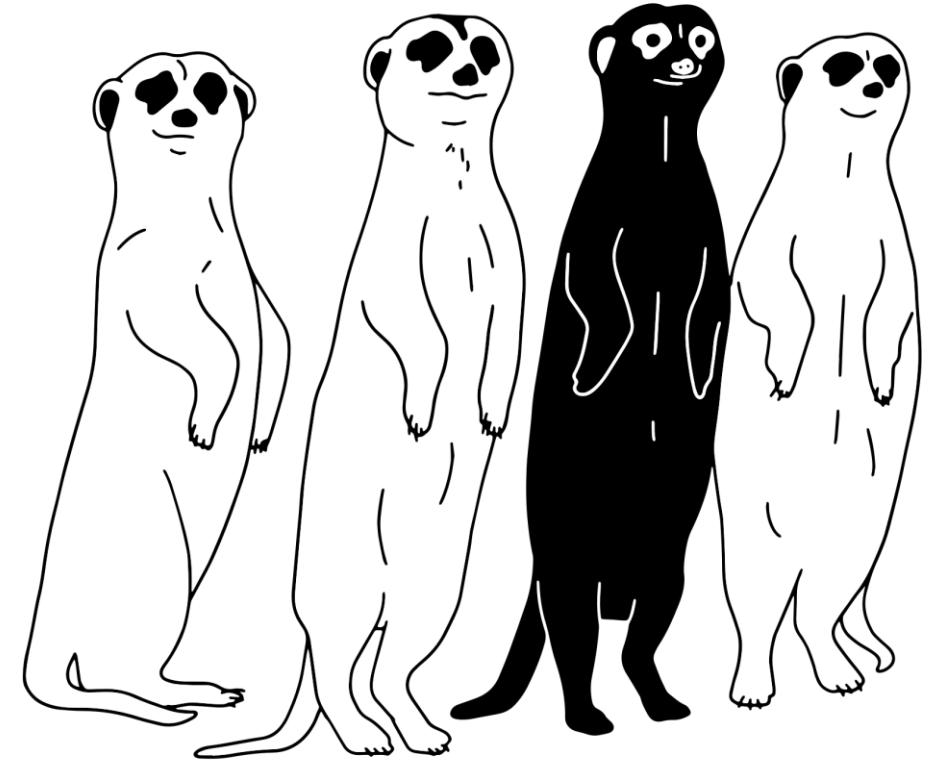


Standorte 13 (D, CH, SK)



# Vorstellungsrunde

- Hintergrund – Was mache ich in meinem Beruf/Tätigkeit?
- Habe ich bereits Erfahrungen mit Machine Learning gemacht?
- Warum besuche ich diese Schulung?
- Welche Vorkenntnisse habe ich? (Allgemein ML, Python, tensorflow/keras oder ähnliche Tools bspw.: PyTorch)
- Was erwarte ich mir von dieser Schulung?



# **Agenda - Tag 1**

**Vorstellungsrunde**

---

**Grundlagen & Überblick Machine Learning**

---

**Supervised Learning**

---

**Neural Networks (MLPs)**

---

**Deep Neural Networks (CNNs)**

---

# **Agenda - Tag 2**

## **Offene Themen von Tag 1**

---

**Klassisches Unsupervised Learning**

**Deep Neural Networks (RNNs + LSTMs)**

---

# **Agenda - Tag 3**

**Offene Themen von Tag 2**

---

**Deep Learning Unsupervised Learning**

---

**Offene Themen & Fragen**

---

**Feedback**

---

# Lernziele

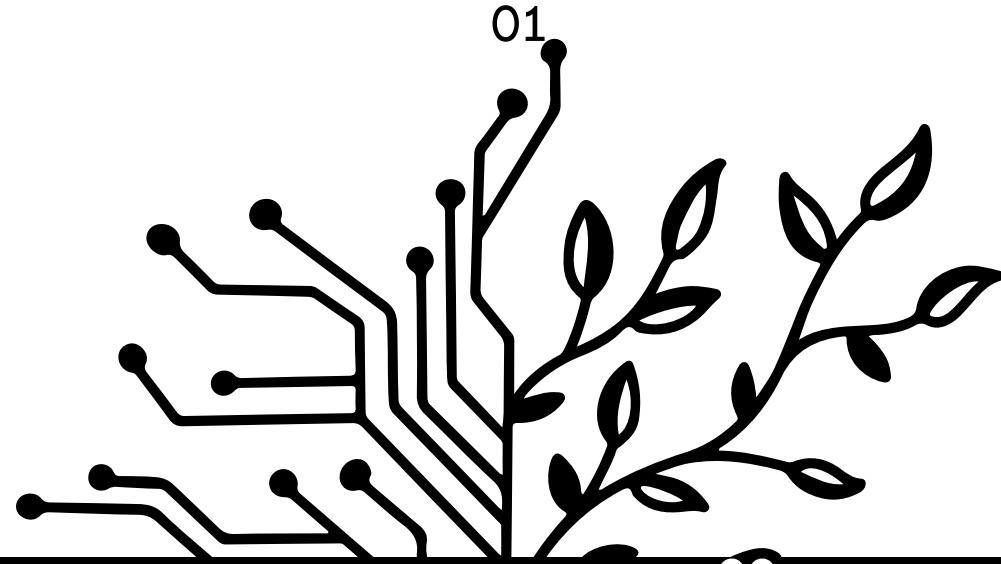
- Grundlegendes Verständnis über die Methoden und Konzepte des Supervised und Unsupervised Learning
- Überblick über Theorie und Praxis von neuronalen Netzen und deren Erweiterung: Deep Learning Methoden wie beispielsweise Convolutional Neural Networks (CNNs) oder Recurrent Neural Networks (RNNs), und deren Einsatzfelder
- Einführung in das Package Tensorflow/tensorflow.keras zum Trainieren der Modelle
- Ausblick auf weitere Methoden

# Literatur

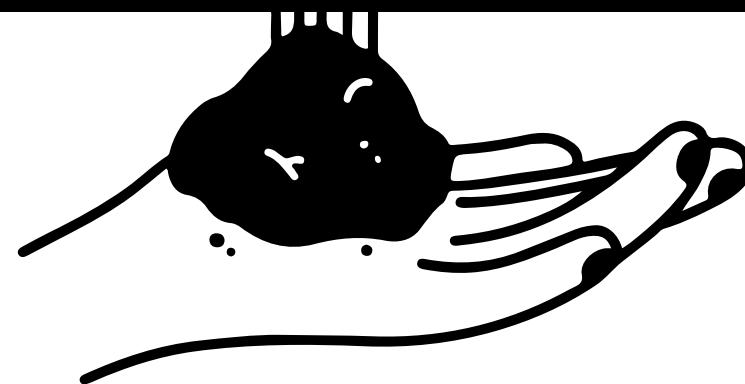
- Provost, Foster, and Tom Fawcett. Data Science für Unternehmen : Data Mining und datenanalytisches Denken praktisch anwenden, mitp, 2017. → Eher Business
- Géron, A., Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, O'Reilly, 2019. → Eher technisch
- Raschka, S., Machine Learning mit Python: Das Praxis-Handbuch für Data Science, Predictive Analytics und Deep Learning, mitp, 2016 → Eher technisch

# Unterlagen herunterladen

- Öffnen der bevorzugten Git Konsole (z.B Git Bash)
- Auswahl des Ordners in dem das Repository abgelegt werden soll
- Klonen des Repositorys: **git clone https://gitfront.io/r/user-2025188/sv5fADUekGjL/supervised-unsupervised-learning-keras/**



# Grundlagen & Überblick Machine Learning



# Was ist Machine Learning?

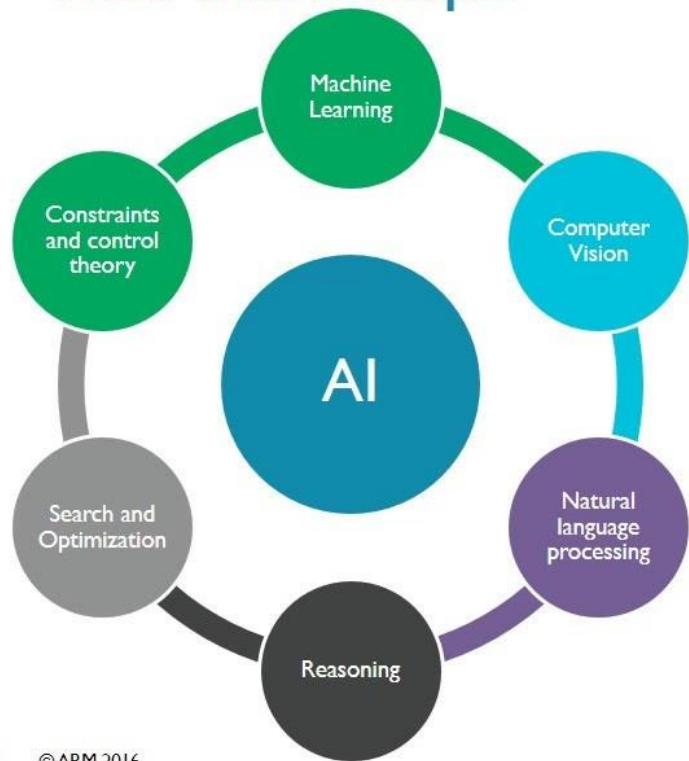


**„Machine Learning is the field of study  
that gives computers the ability to learn  
without being explicitly programmed“**

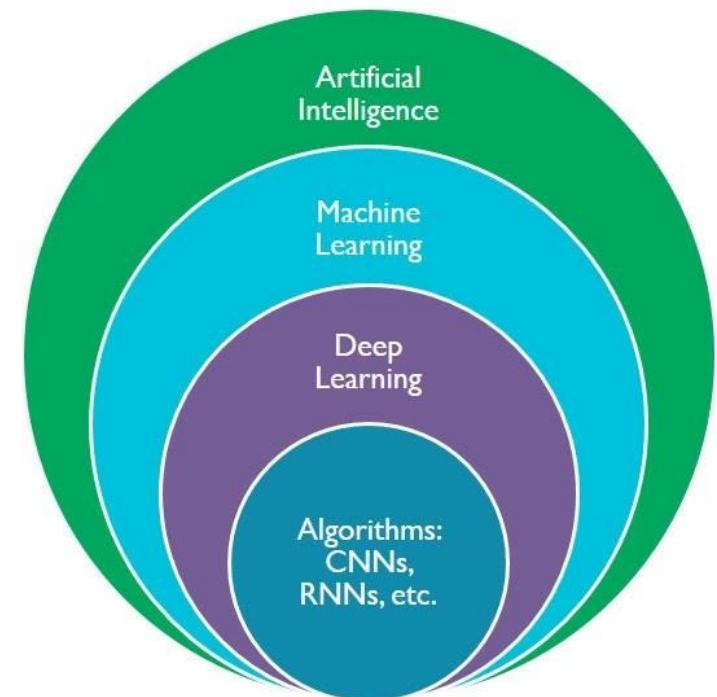
**Arthur Samuel, 1959**

# Artificial Intelligence und Machine Learning

The AI landscape



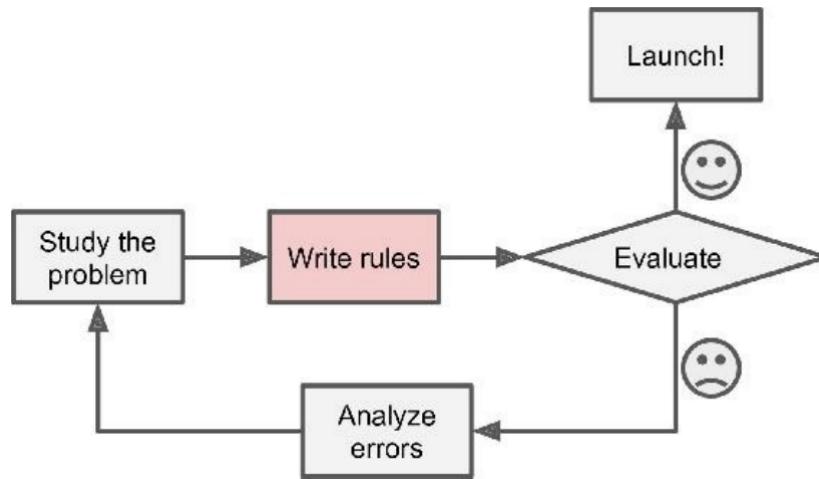
6 ©ARM 2016



ARM

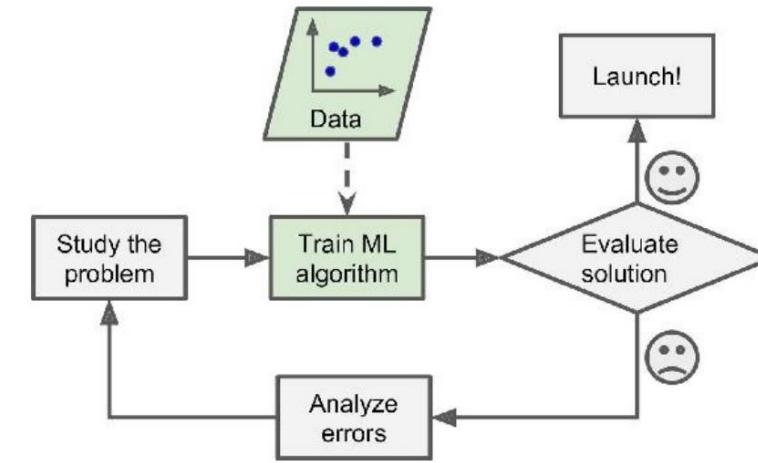
# Warum Machine Learning?

## Motivation – Spam Filter



### Traditioneller Ansatz

- Komplex, hard-codiert
- Schwer zu warten



### Machine learning Ansatz

- Automatisches lernen aus Daten
- Automatisches re-trainieren

# Beispiele aus dem Alltag

## **Sentiment-Analyse (NLP):**

Hat ein bestimmter Post/Review eine positive oder negative Aussage → Automatische Erstellung von Tags für Nutzerreviews

## **Gaming-Bot:**

Intelligente Systeme, welche mithilfe von Reinforcement Learning trainiert werden. Beispiel ist das berühmte AlphaGo, welche den World Champion besiegte.

## **Entdecken von Hirntumoren in MRT Scans:**

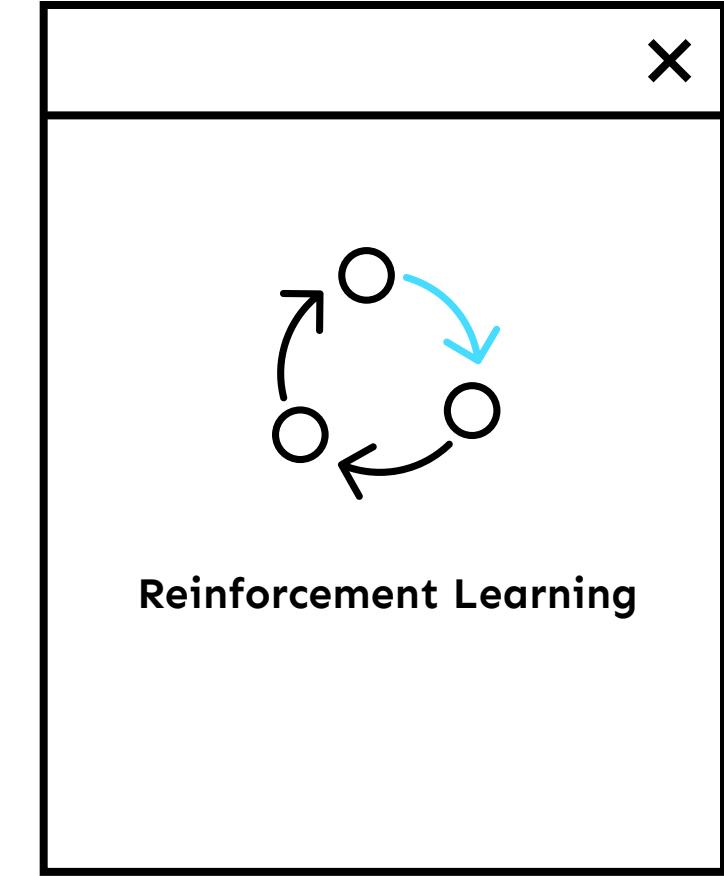
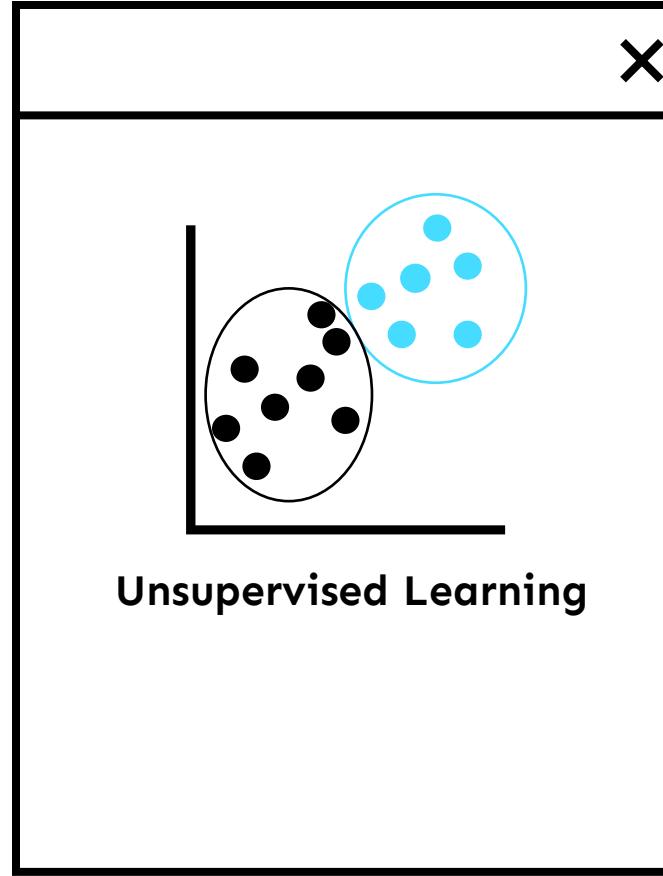
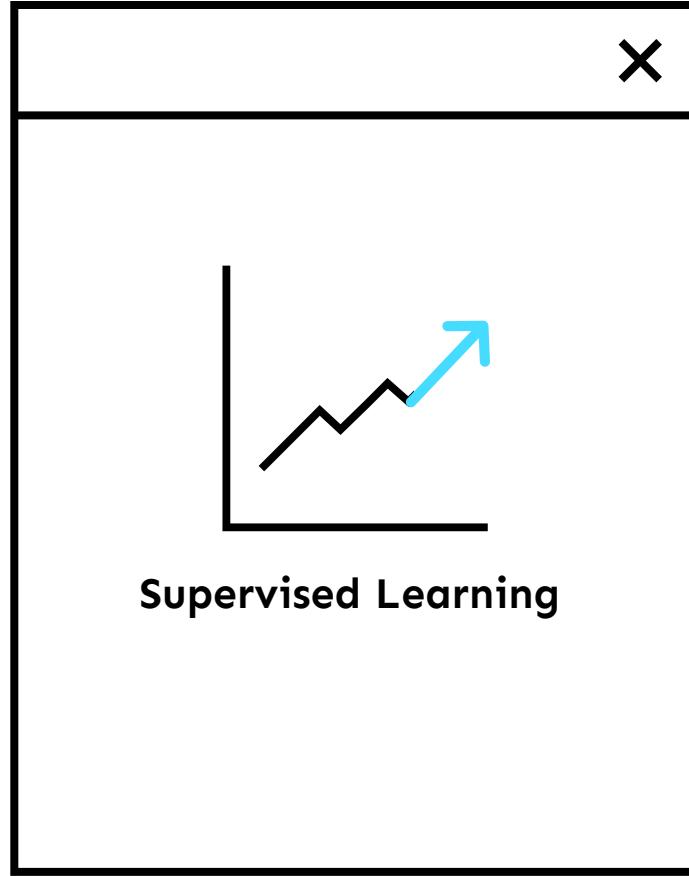
Bestimmen der Größe und Position von Tumoren in MRT Scans mithilfe von Image Classification (CNNs)

## **Vorhersage des nächsten Jahresumsatzes:**

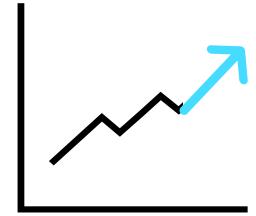
Regressionsmodelle, welche beispielsweise aus historischen Daten trainiert werden. Diese können beispielsweise neuronale Netze sein.

## **Fallen Ihnen weitere Beispiele ein?**

# Supervised vs. Unsupervised vs. Reinforcement Learning



# Supervised Learning



„Überwachtes Lernen“: In den historischen Daten ist das Ergebnis (**Label**), das in der Zukunft vorhergesagt werden soll, bekannt.

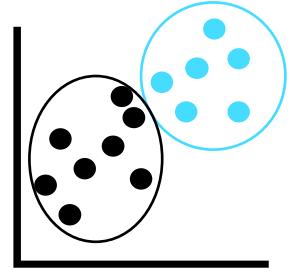
- Die historischen Daten nutzt der Machine-Learning-Algorithmus, um das Ergebnis in der Zukunft bestmöglich vorherzusagen.
- Das Modell wird zielgerichtet **trainiert**, d.h. das Muster, das den Zusammenhang zwischen Eingangsdaten und Ergebnis darstellt, wird erlernt.

Prediktive Fragestellungen (**Predictive Analytics**):

- **Klassifikation:** Das vorherzusagende Merkmal ist **kategorisch**
- **Regression:** Das vorherzusagende Merkmal ist **numerisch** mit nicht klar abgrenzbaren Werten

Die **Güte** des Modells ist durch die **Abweichung** zwischen Prognosewert und Label direkt messbar.

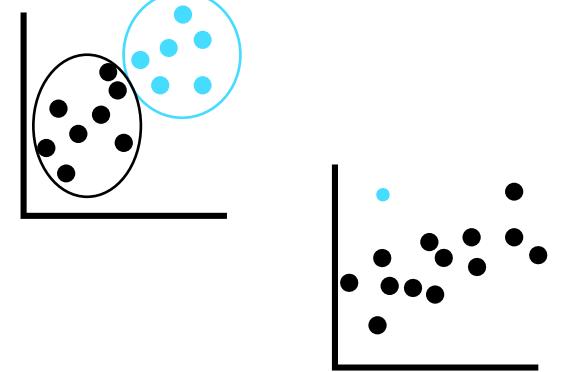
# Unsupervised Learning



„**Unüberwachtes Lernen**“: Es gibt keine Vergleichswerte aus der Historie; allgemeine Muster in den Daten werden erkannt.

- **Clustering:** Das Modell sucht nach gemeinsamen Merkmalen z.B. Kunden werden in Gruppen mit ähnlichem Bestellverhalten geclustert
- **Anomaly Detection:** Das Modell sucht nach Auffälligkeiten oder Ausreißern im Datenset.

z.B. Kunden mit unüblichem Bestellverhalten werden als auffällig gekennzeichnet

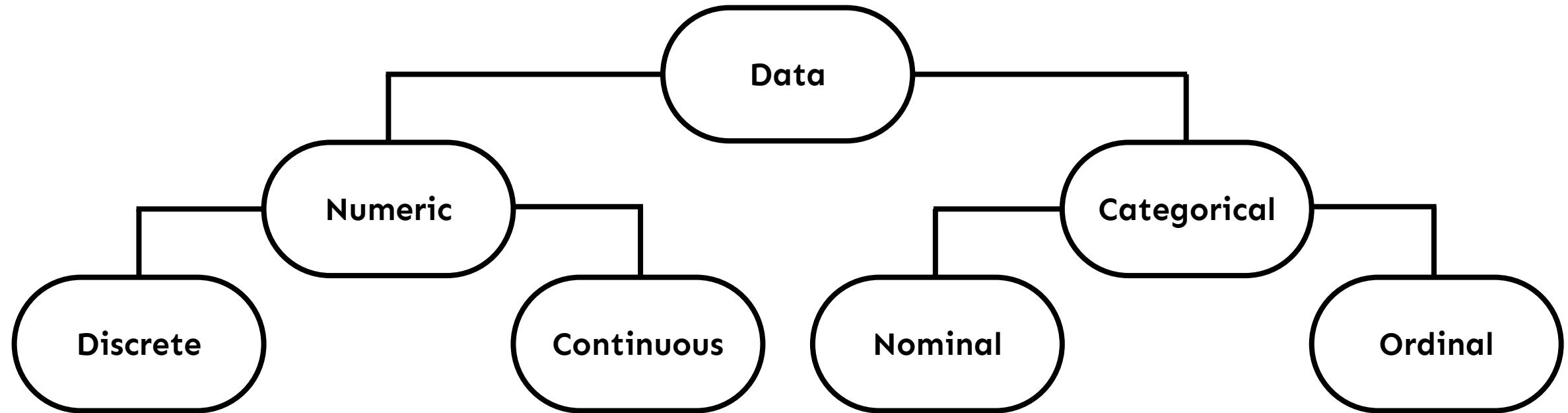


Die **Güte** des Modells ist nicht direkt messbar, vielmehr liegt diese im Auge des Betrachters

- Müssen die Cluster feingranularer gebildet werden? Können Cluster zusammengelegt werden? Hilft das Clustering für weiterführende Analyseschritte?
- Handelt es sich bei den Auffälligkeiten tatsächlich um problematische Anomalien? Oder handelt es sich nur um sehr seltene, aber dennoch nachvollziehbare Ausreißer?

Die **Fachlichkeit** spielt eine große Rolle. Das Ergebnis muss im fachlichen Kontext **interpretiert** und bewertet werden.

# Datentypen Machine Learning



Werte sind Integer:

- Anzahl Studierende
- Alter

Werte können jeden

- Wert annehmen,  
üblicherweise innerhalb  
einer Range:
- Temperatur
  - Alter

Keine natürliche

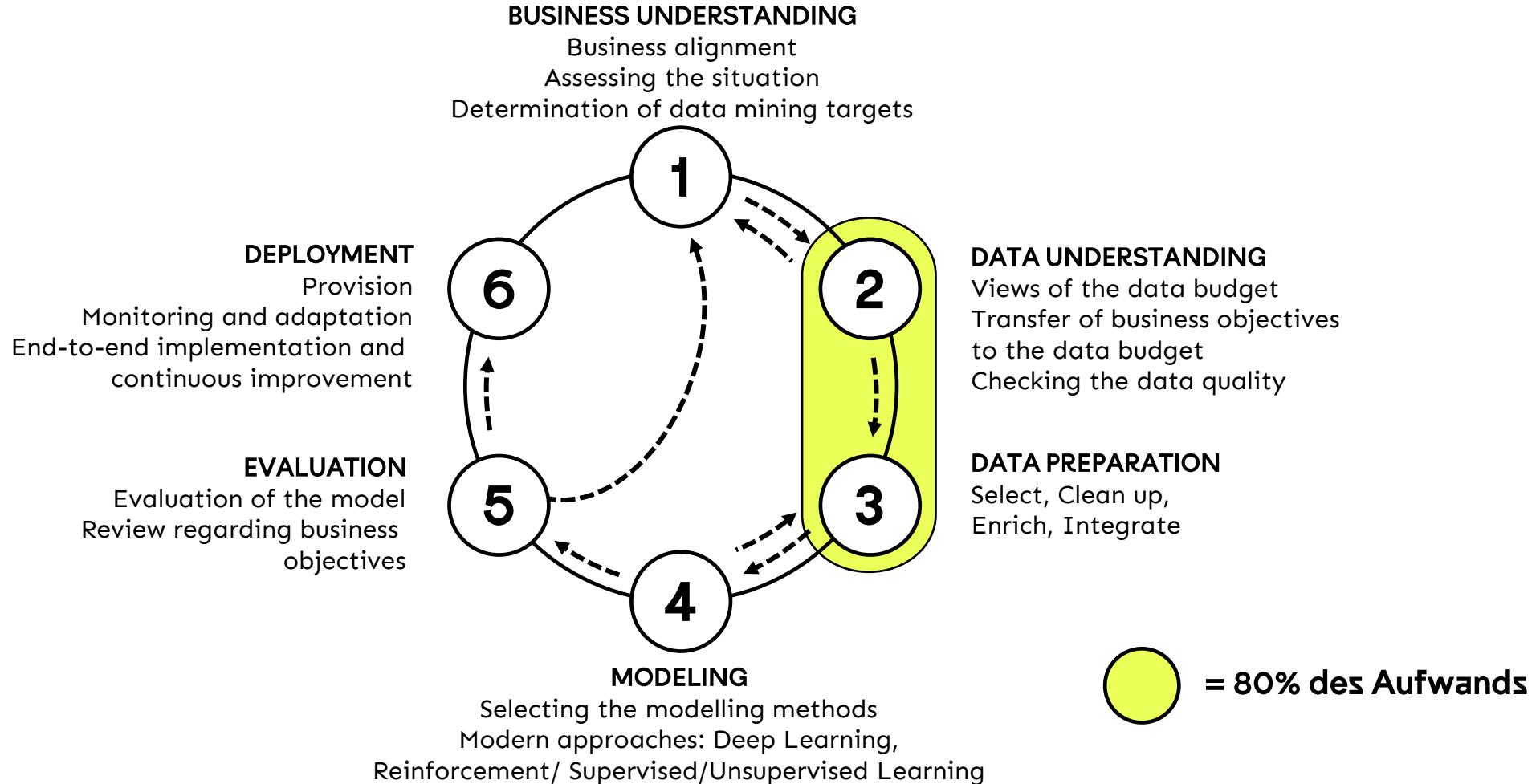
- Reihenfolge zwischen  
Kategorien:
- Geschlecht
  - Länder
  - Farbnamen

Eine Reihenfolge

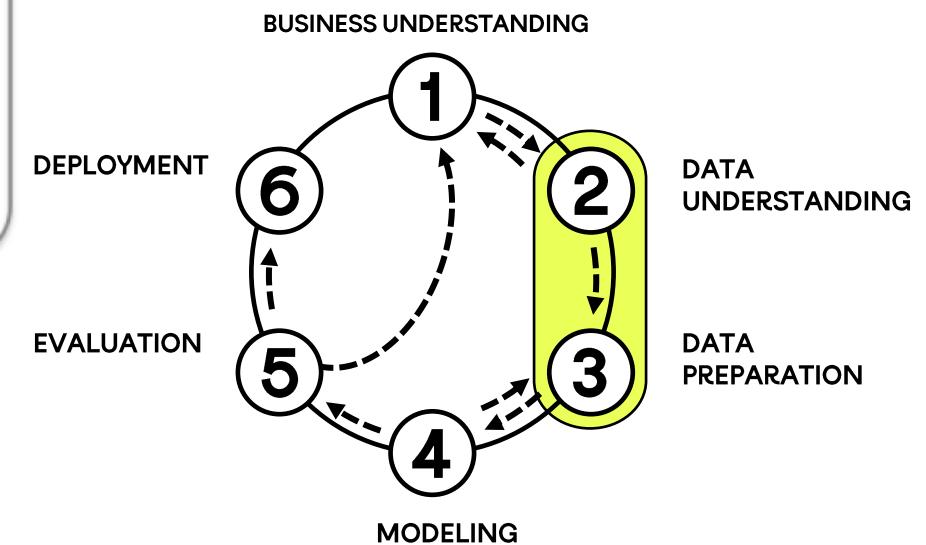
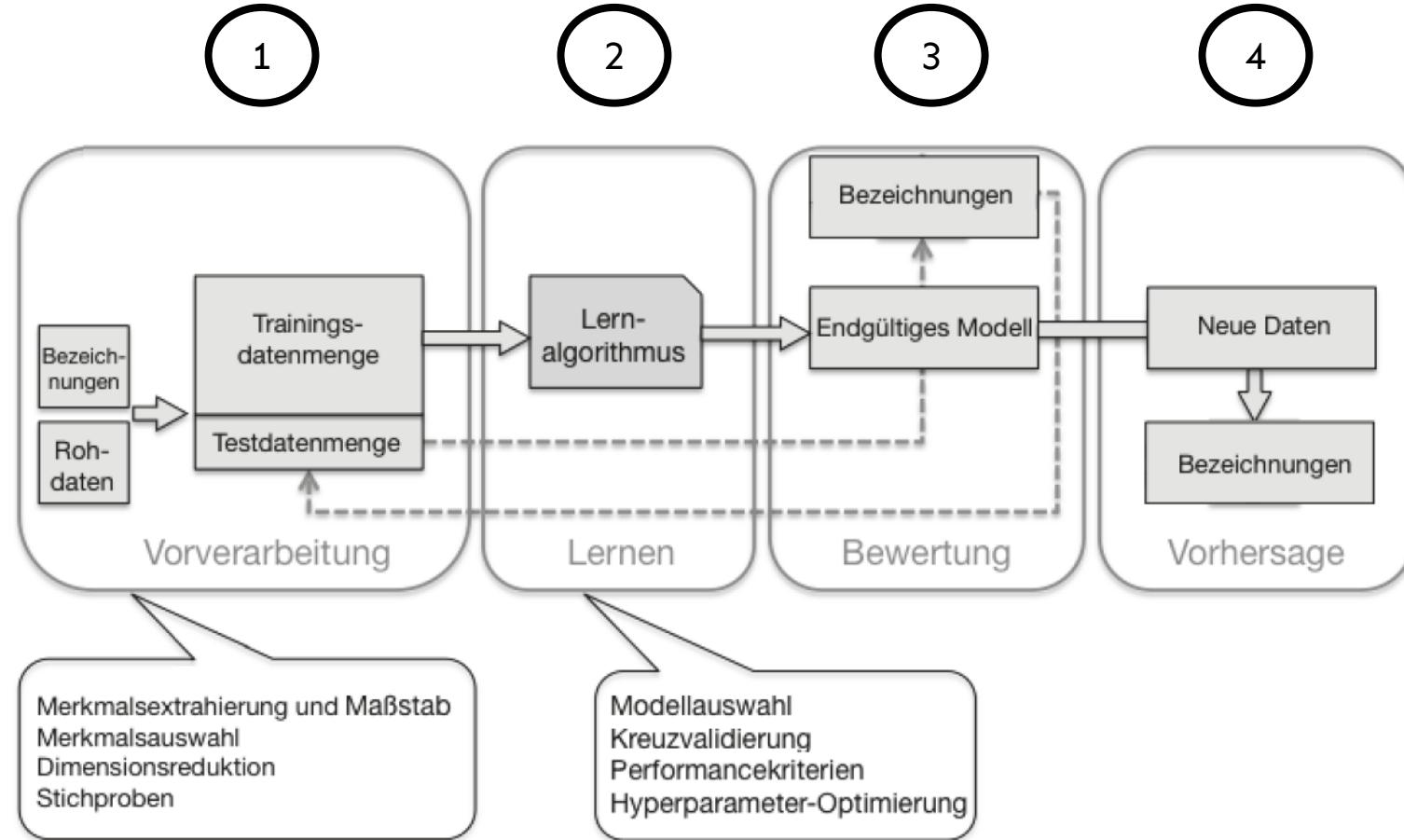
- zwischen Kategorien:
- Tshirt Größen (S, M, L)
  - Tageszeit (morgens,  
mittags, abends)

# Cross Industry Standard Process of Data Mining (Crisp-DM)

Der Data Science / Machine Learning Workflow



# Machine Learning Pipeline



Raschka, S., Machine Learning mit Python: Das Praxis Handbuch für Data Science, Predictive Analytics und Deep Learning, 2016, S.31 f.

# Typen von Machine Learning Problemen in dieser Schulung

## 1. Klassifikation und Wahrscheinlichkeitsabschätzung der Klassenzugehörigkeit

- Unterteilung der Daten in Klassen z.B.: männlich/weiblich, Spam/kein Spam
- Ziel ist es, jeden Datenpunkt einer Klasse zuzuordnen
- Zugehörigkeit zu mehreren Klassen meist ausgeschlossen
- Fragestellung: Welche Kunden haben hohes Abwanderungspotenzial?

## 2. Regression

- Vorhersagen eines numerischen Wertes z.B.: Aktienkurs, Umsatz
- Fragestellung: Wieviel Umsatz werden wir im nächsten Jahr voraussichtlich machen?

## 3. Clustering

- Zusammenfassen von Datenpunkten zu Gruppen anhand ihrer Ähnlichkeit
- Es wird vorrangig kein richtiges „Ziel“ verfolgt
- Fragestellung: Finden wir ähnliche Gruppen in unseren Daten wieder?

# Herausforderungen Machine Learning

Machine Learning bringt im Vergleich zu traditionellen Software Systemen **neue Herausforderungen** mit sich:

- Unzureichende Trainingsdaten
- Trainingsdaten sind nicht repräsentativ (Sampling Bias)
- Schlechte Datenqualität (Fehlwerte → oder Fehlwerte, die nicht als solche erkennbar sind Bsp.: als „unknown“ gekennzeichnet)
- Irrelevante oder unzureichende Features
- Overfitting → grundlegendes Problem ist Bias/Variance Trade-Off
- Generell gilt „**Garbage in, garbage out**“!
  - Besonders bei der Automatisierung von Geschäftsentscheidungen



# Installation Miniconda

1. Miniconda herunterladen:

[https://repo.anaconda.com/miniconda/Miniconda3-latest-  
Windows-x86\\_64.exe](https://repo.anaconda.com/miniconda/Miniconda3-latest-Windows-x86_64.exe)



2. Und Installieren: [https://conda.io/projects/conda/en/stable/user-  
guide/install/windows.html](https://conda.io/projects/conda/en/stable/user-guide/install/windows.html)

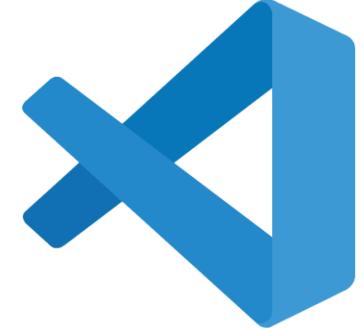
# Anlegen einer Virtuel Environment

1. Anaconda Prompt öffnen
2. **conda create -n <myenv> python=3.10**
3. Entweder **pip install -r requirements.txt** oder **pip install package**  
(z.B. tensorflow, matplotlib, ...)
4. Aktivieren der VEnv mit : **conda activate <myenv>**



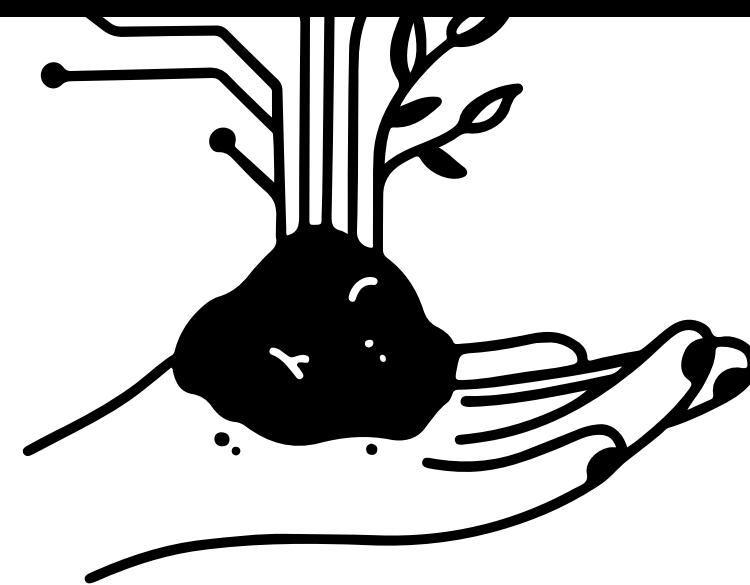
# Installation VS Code

- Setup exe herunterladen und installieren:  
<https://code.visualstudio.com/>
- Python Extension installieren
- Jupyter Notebook öffnen:
  - Ctrl+Shift+P
  - Create: New Jupyter Notebook
- Wähle Python Interpreter deiner Conda Environment in der rechten oberen Ecke aus
- `print("hello world!")` eingeben und ausführen



01

# Supervised Learning



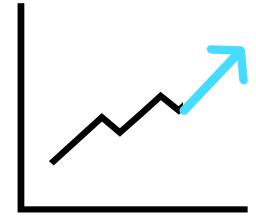
# Supervised Learning

## Fragestellung Supervised Learning:

z.B.: Gibt es Kunden, welche eine hohe Wahrscheinlichkeit besitzen ihren Vertrag zu kündigen?

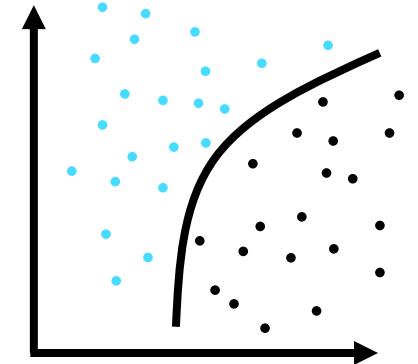
- Training erfolgt auf historischen Daten, welche alle gelabelt sind.
- Ziel ist es durch die vorliegenden Daten (Features), die Zielvariable so genau wie möglich vorhersagen zu können und ggf. eine Wahrscheinlichkeit der Klassenzugehörigkeit zu ermitteln.
- Vorhersage erfolgt mit Daten bei der die Klasse (das Target) fehlt.

# Typen des Supervised Learnings



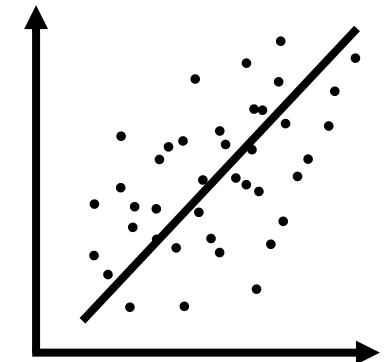
## 1. Klassifikation und Wahrscheinlichkeitsabschätzung der Klassenzugehörigkeit

- Unterteilung der Daten in Klassen z.B.: Betrug/kein Betrug, Spam/kein Spam
- Ziel ist es, jeden Datenpunkt einer Klasse zuzuordnen
- Zugehörigkeit zu mehreren Klassen meist ausgeschlossen
- Fragestellung: Welche Kunden haben hohes Abwanderungspotenzial?
- Vorhersagen durch Trennung des Entscheidungsraums



## 2. Regression

- Vorhersagen eines numerischen Wertes z.B.: Aktienkurs, Umsatz
- Fragestellung: Wieviel Umsatz werden wir im nächsten Jahr voraussichtlich machen?
- Aufstellung einer Funktion, welche eine Art Approximation darstellt



# Bias Variance Trade-Off



# Bias- Variance Tradeoff

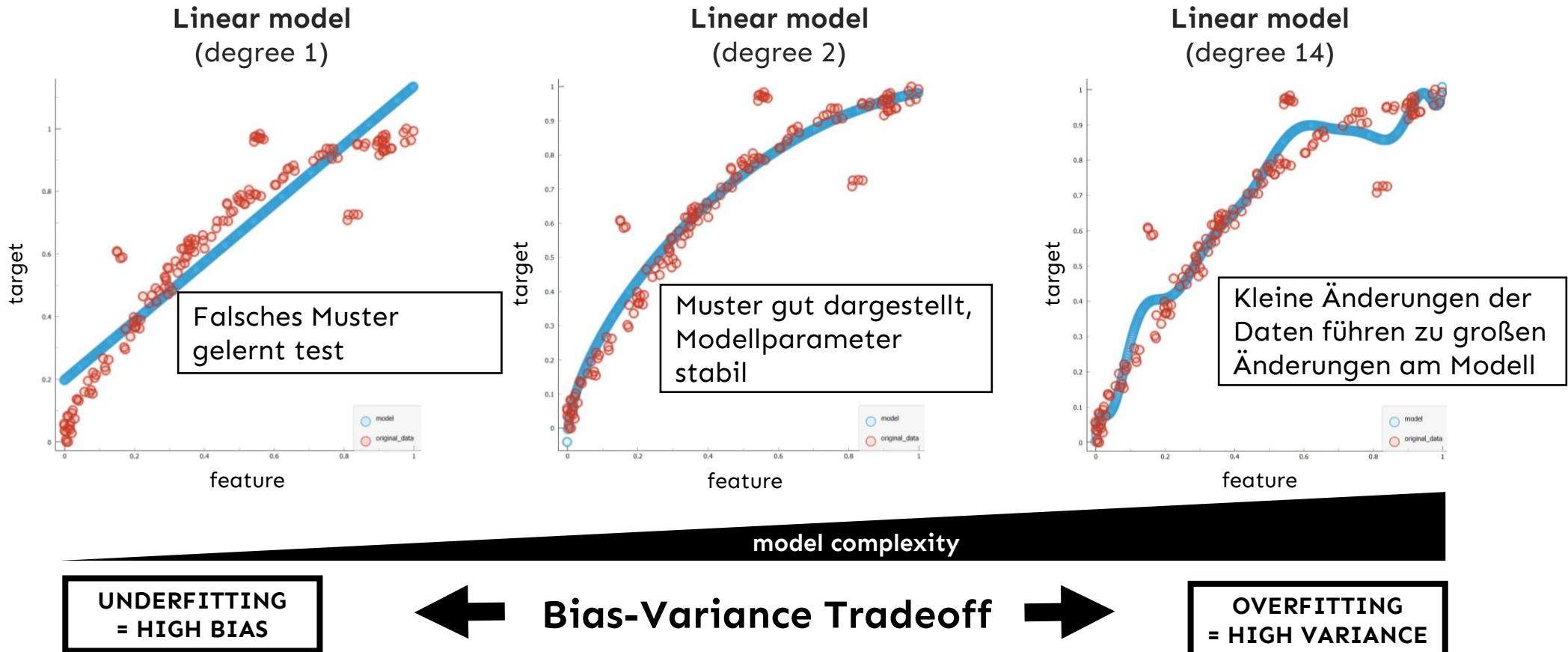
## Over- und Underfitting

### Grundlegende Herausforderung im Machine Learning:

Wie kann ein Modell

- a. die richtigen Muster in den historischen Daten finden
- b. gut performen auf neuen Daten (Generalisierung)

# Bias-Variance Tradeoff



# Bias-Variance Tradeoff

Low Bias, Low Variance (so gut wie unmöglich):

- Modell hat informative/relevante Features
- Modellparameter sind stabil
- Modell performt gut auf unbekannten Daten (Generalisierung)

Low Bias, High Variance:

- Overfitting
- Sprunghafte Modellparameter bei der Verwendung unterschiedlicher Trainingssets (inkonsistent)
- Modell lernt eventuell zu viel Noise oder Daten auswendig (keine Generalisierung möglich)

High Bias, Low Variance:

- Underfitting
- Vorhersagen sind konsistent, aber inakkurat im Durchschnitt

High Bias, High Variance:

- Modell hat weder informative Features, noch ist es richtig dimensioniert.
- Vorhersagen sind inkonsistent und inakkurat im Durchschnitt

→ Keine Vorhersagen auf das Target möglich

Source. <https://medium.com/analytics-vidhya/difference-between-bias-and-variance-in-machine-learning> ; eigene Übersetzung

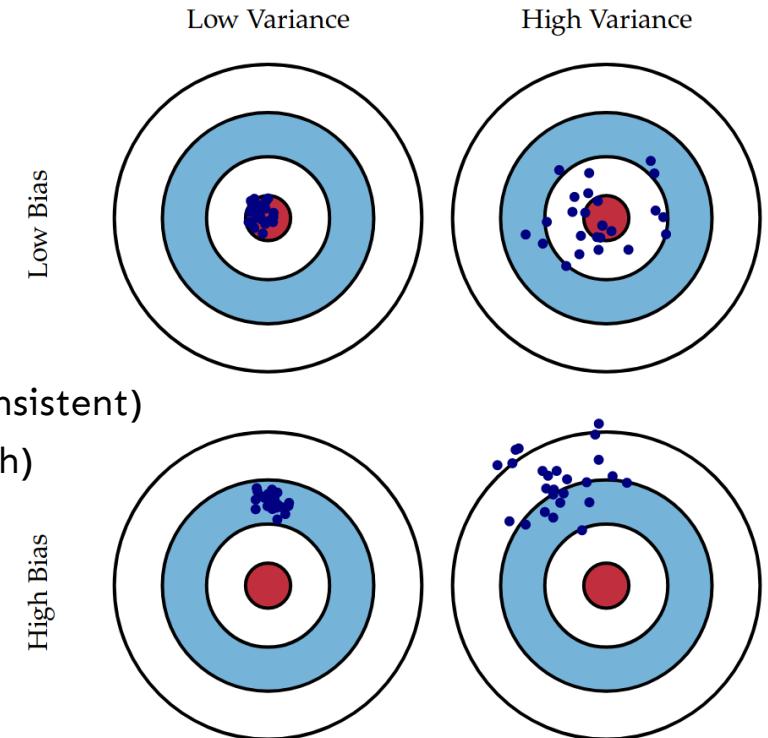


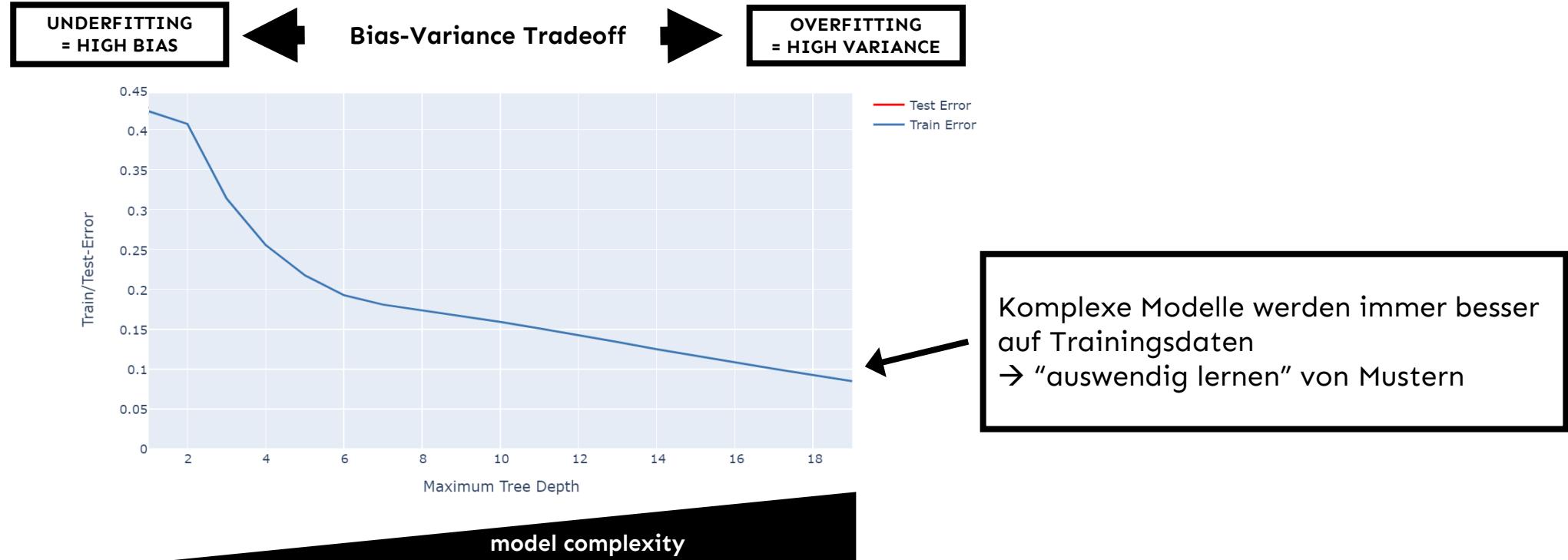
Fig. 1 Graphical illustration of bias and variance.

Sources: Fortmann-Roe, Scott. 2012. "Understanding the Bias-Variance Tradeoff."

# Bias-Variance Tradeoff

## Train/Test-Error

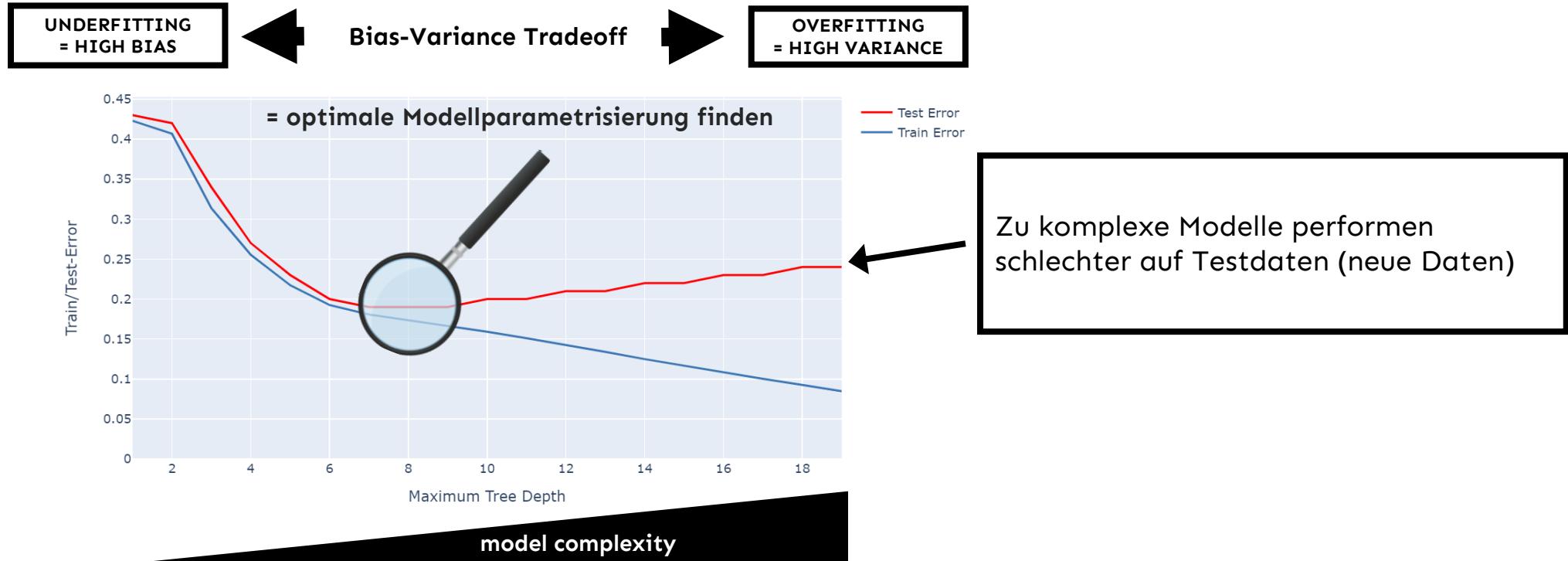
Eine numerische Betrachtung des Bias-Variance Tradeoffs - Variieren von Modellparametern & messen des train-test-errors



# Bias-Variance Tradeoff

## Train/Test-Error

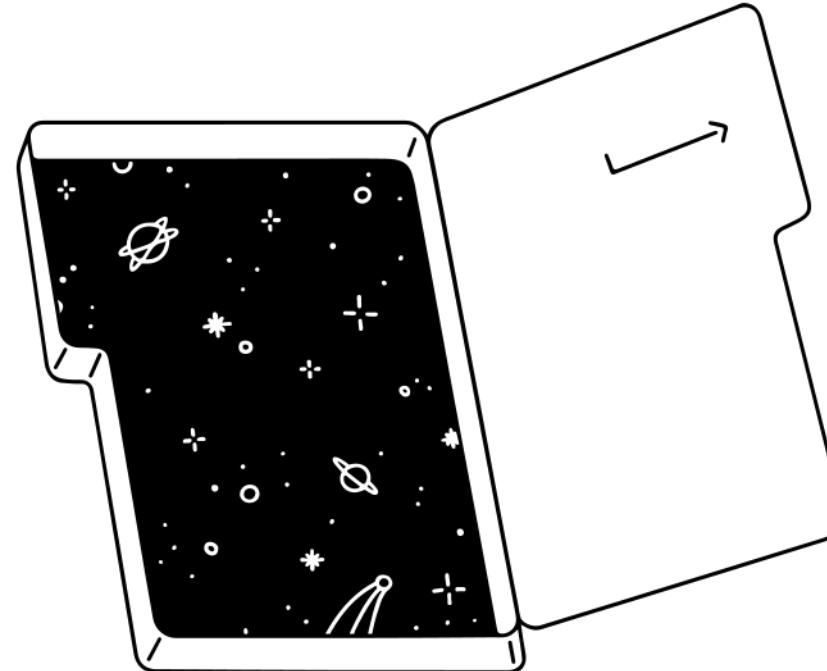
Eine numerische Betrachtung des Bias-Variance Tradeoffs - Variieren von Modellparametern & messen des train-test-errors



# Python Basics

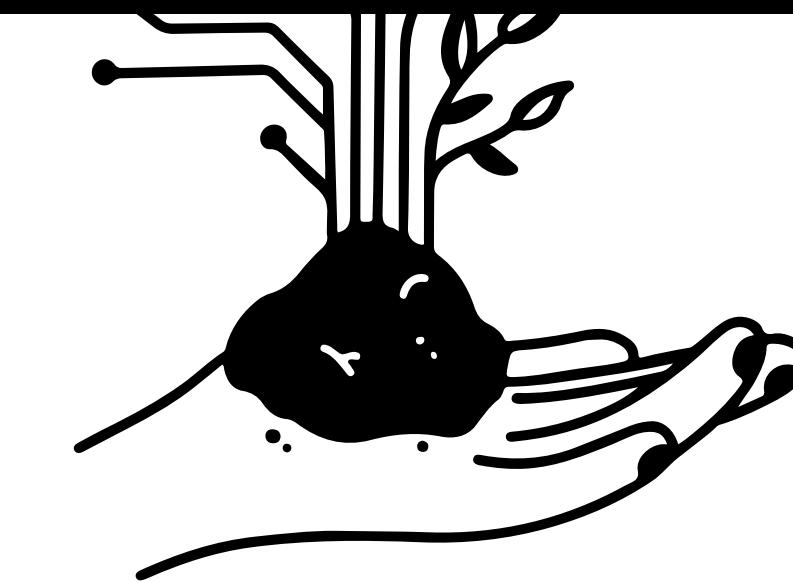
- Möglichkeit, Python Basiswissen aufzufrischen: [A Whirlwind Tour of Python](#) von Jake VanderPlas
- Wiederholung der wichtigsten Python Basics
- [Übung\\_00\\_Python\\_Auffrischung](#)

## PRAXIS



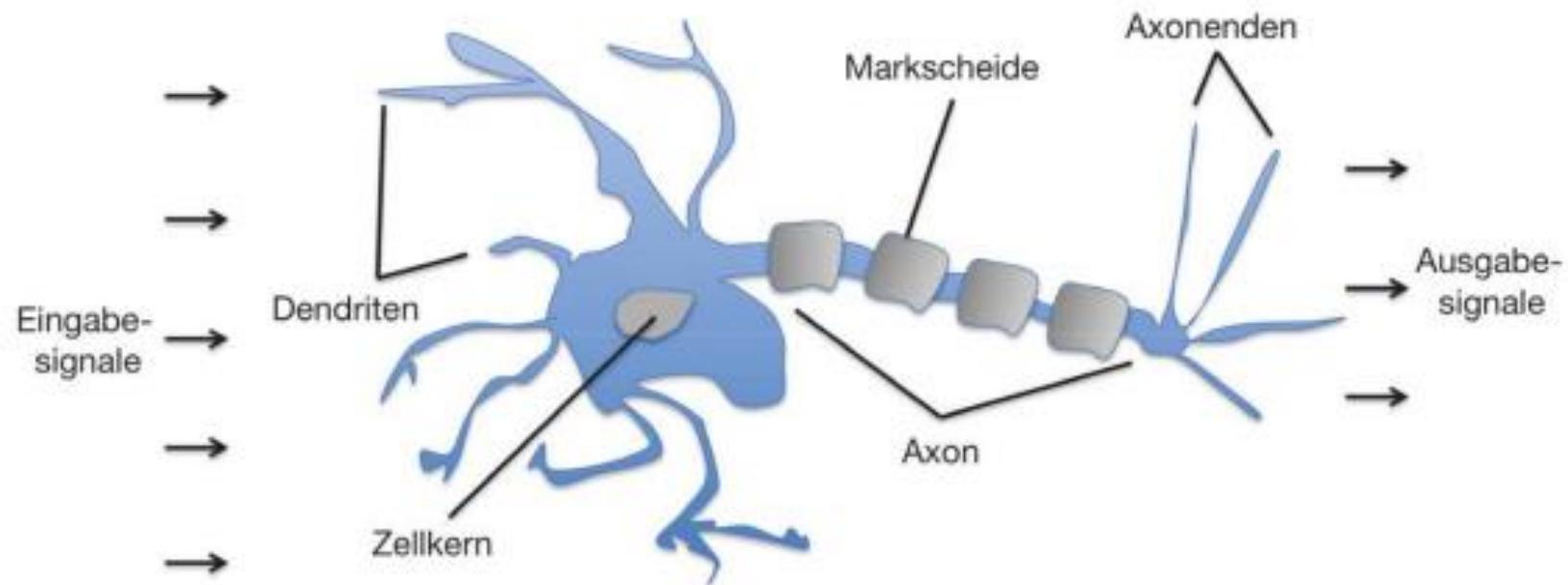
01

# Neural Networks (MLP's)



# Ursprung Neural Networks

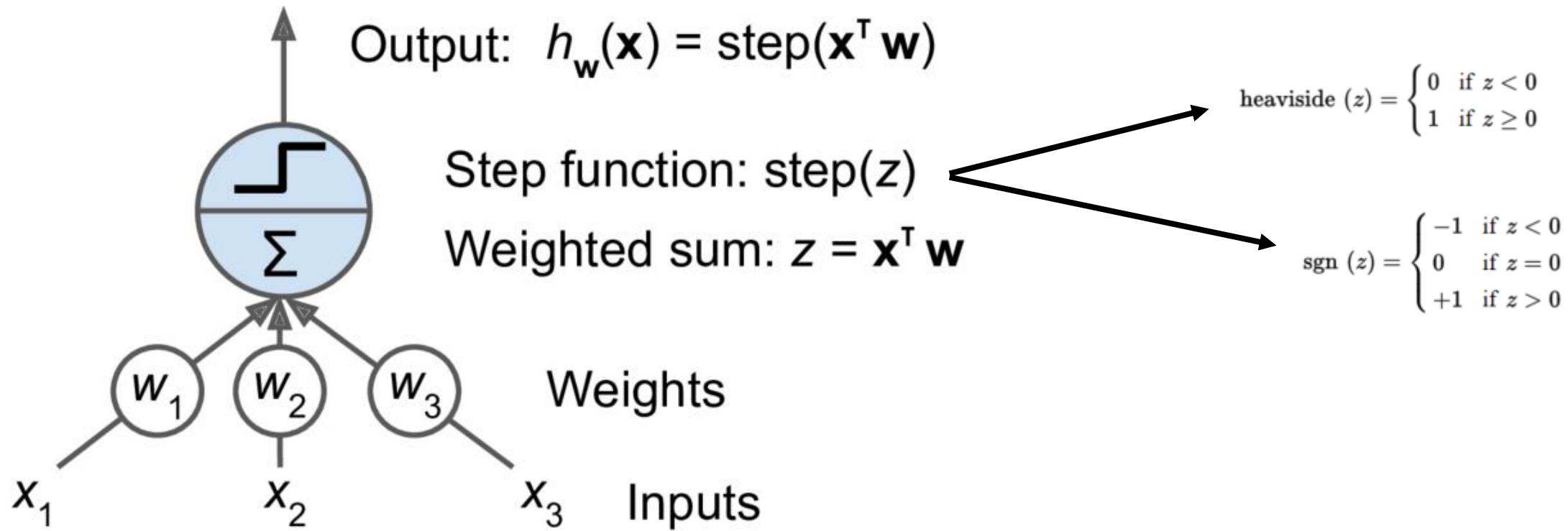
## Biologisches Neuron



Cf. Raschka 2016, 38

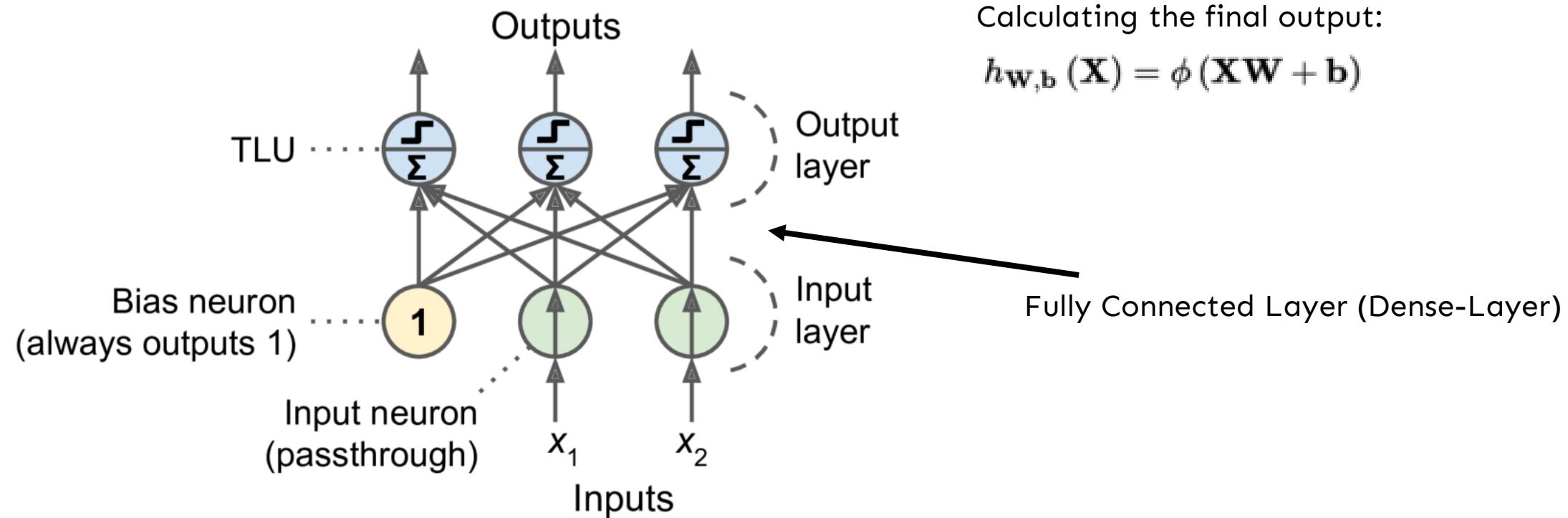
# Ursprung Neural Networks

Perceptron (TLU → Threshold Logical Unit)



# Ursprung Neural Networks

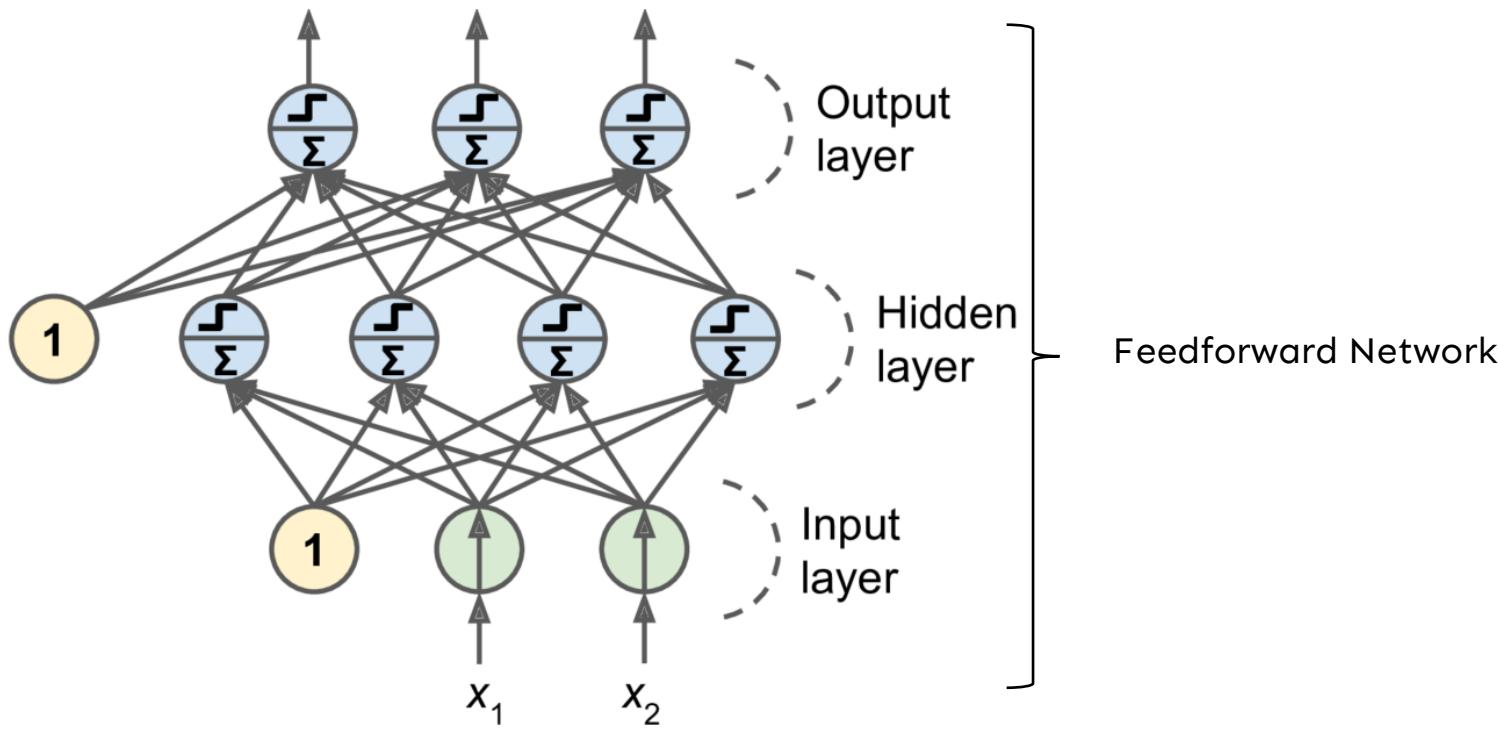
## Perceptron



# Neural Networks

## Multi-Layer Perceptron (MLP's)

Normale Perceptrons können keine XOR Funktion abbilden, jedoch durch stapeln mehrerer Perceptrons, kann dies erreicht werden → deshalb Multi-Layer Perceptrons

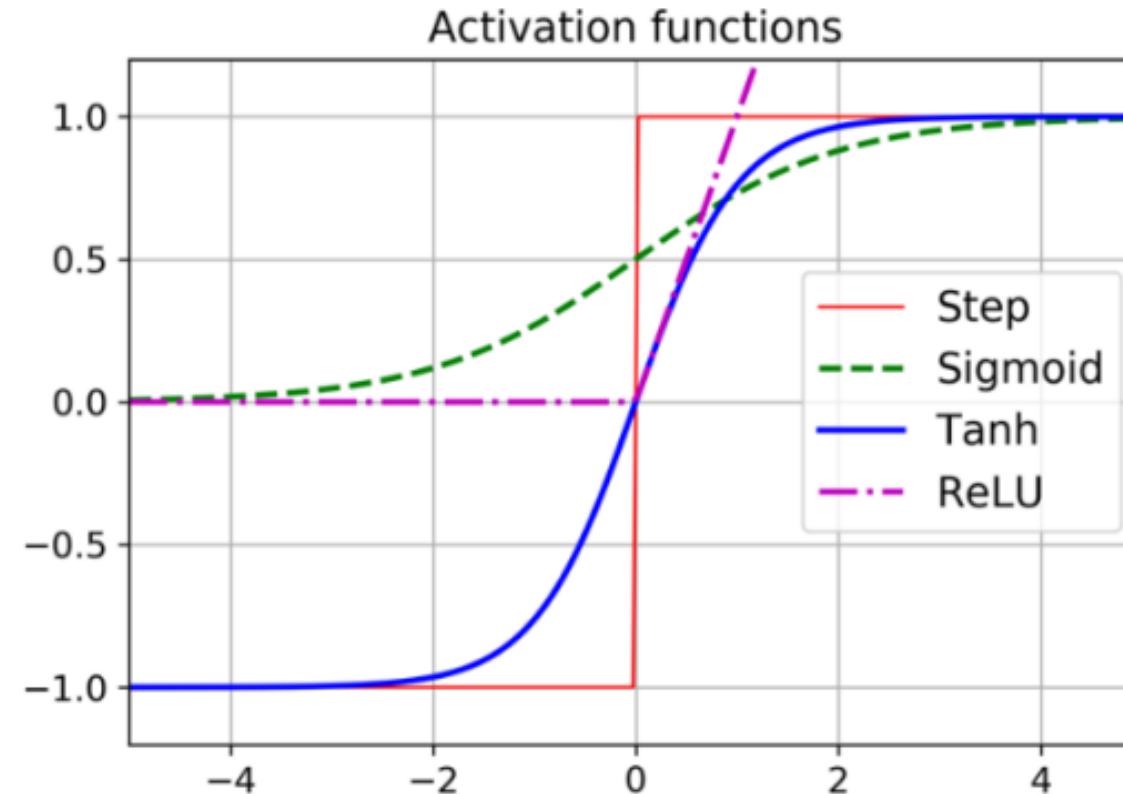


Géron, 2019, p.291 f.

# Neural Networks

## Aktivierungsfunktionen

- Tensorflow-Keras bietet verschiedene Arten von Aktivierungsfunktionen an, welche auf den Output von Neuronen angewandt werden
- Überblick:  
<https://keras.io/api/layers/activations/#available-activations>
- Wir werden vor allem ReLU benutzen (Rectified Linear Unit), die meistgenutzte Aktivierungsfunktion



# Neural Networks

## Backpropagation of Error – How neural networks learn

- Basierend auf Paper von Rumelhart, Hinton & Williams über backpropagation training algorithm 1986
- Benötigt zwei Schritte: forward pass & reverse pass
- Ist in der Lage den Gradient des „Network errors“ in Bezug auf jeden einzelnen Modellparameter zu berechnen → ermöglicht es herauszufinden wie die Weights & Biases angepasst werden müssen um den Error zu minimieren
- Sobald der Gradient berechnet ist, kann ein Gradient Descent Step durchgeführt werden

Gute Zusammenfassung von Géron:

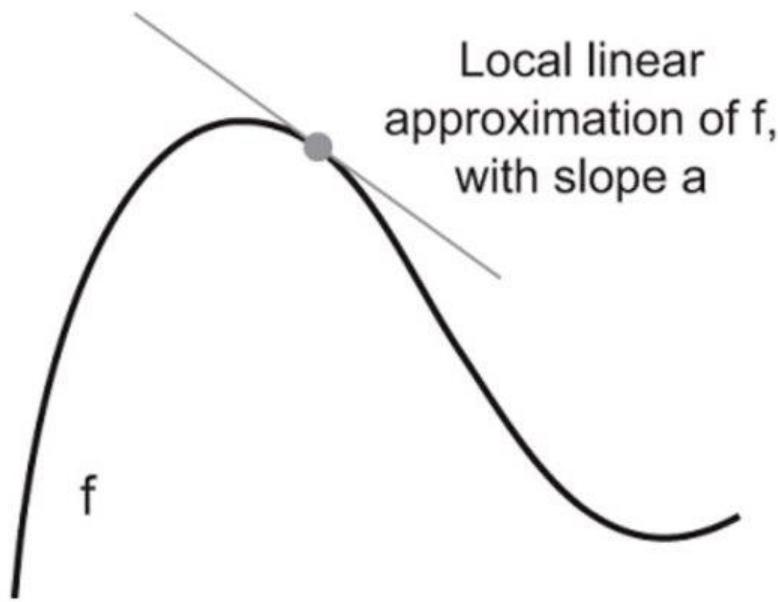
„...for each training instance, the backpropagation algorithm first makes a prediction (forward pass) and measures the error, then goes through each layer in reverse to measure the error contribution from each connection (reverse pass), and finally tweaks the connection weights to reduce the error (Gradient Descent step).“

# Gradientenabstieg

Die Ableitung einer Funktion sagt uns in welche Richtung sie lokal kleinere Werte hat.

Im mehrdimensionalen nennt man die Ableitung Gradient.

Der Gradient zeigt in Richtung der größten Steigung.



$$\text{Gradient: } \mathbf{g} = \nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

Abstieg:  $\mathbf{X}_{\text{new}} = \mathbf{X} - \epsilon \mathbf{g}$

$\epsilon$  : Learning rate (Parameter)

# Gradientenabstieg

## Learning Rate

Zu kleine Learning Rate

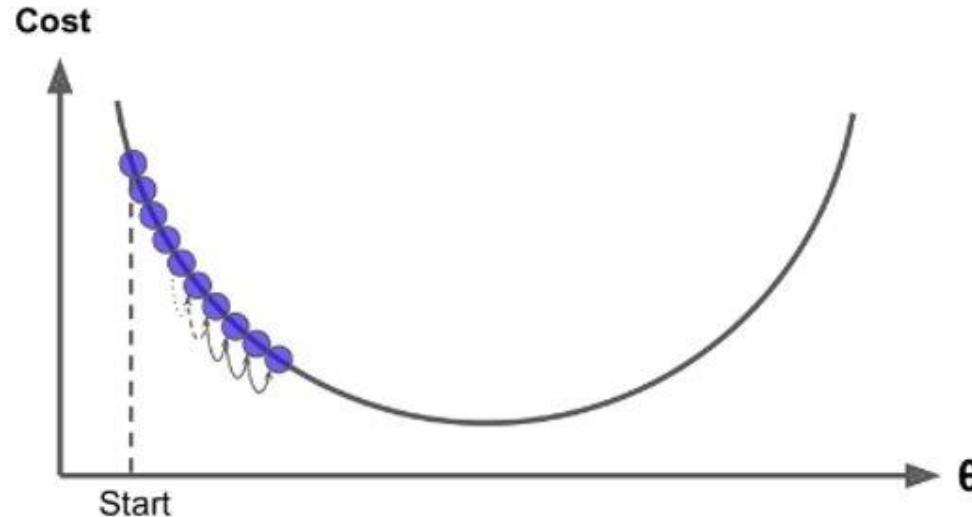


Figure 4-4. Learning rate too small

Zu große Learning Rate

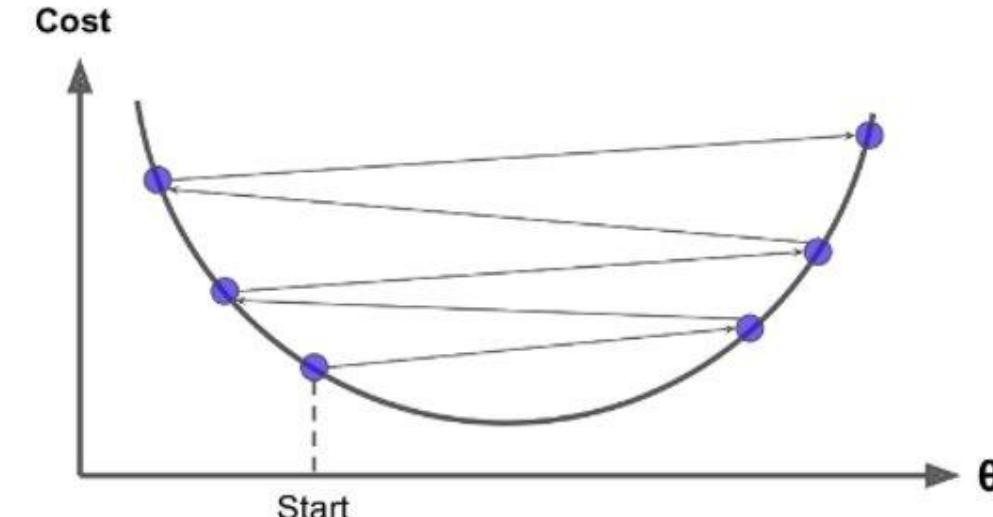


Figure 4-5. Learning rate too large

# Tensorflow Playground

Experimentieren mit unterschiedlichen Neural Network Parametern

<https://playground.tensorflow.org>

# Grundlagen TensorFlow

## Was ist ein Tensor?

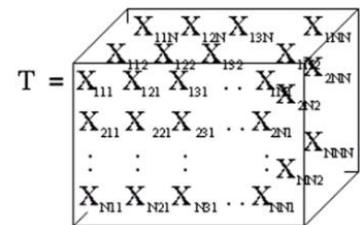
In diesem Kontext, ein multi-dimensionaler Array aus Zahlen

Rank 0: Einfaches Skalar ( $1, 2, 3, 1/2, \pi$ )

Rank 1: Vektor  $\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$

Rank 2: Matrizen  $\begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ m_{21} & m_{22} & \cdots & m_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \cdots & m_{nn} \end{bmatrix}$

Rank3: 3D-Array  $\rightarrow$

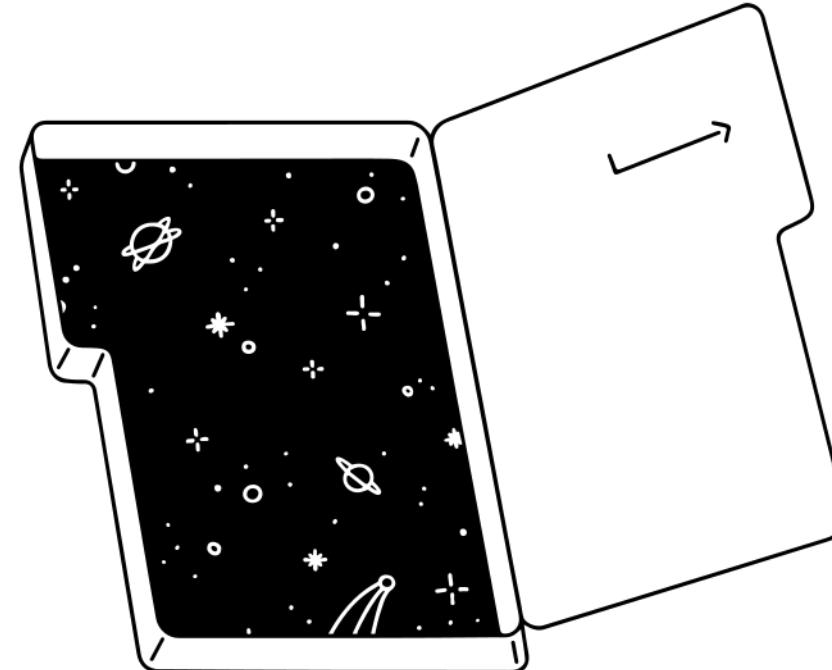


<https://www.youtube.com/watch?v=TvxmkZmBa-k>

# Numpy/Tensorflow Basics

- Vorschau:  
[01\\_Einführung\\_tensorflow\\_numpy](#)
- [Übung\\_01\\_Einführung\\_tensorflow\\_numpy](#)

## PRAXIS



# Hyperparameter - Regression

## Output Neurons

- 1 Output Neuron falls lediglich 1 numerischer Wert vorhergesagt werden soll
- 2 Output Neuronen für bspw. Koordinaten (1 Neuron pro Output Dimension) → exakte Position in einem Bild
- 4 Output Neuronen für Bounding Box: exakte Position + Höhe und Breite der Box (Object Detection)

# Hyperparameter - Regression

## Output Activation

- „linear“ → Wert wird unverändert weitergegeben
- „ReLU“ → Nur Werte  $\geq 0$  werden ausgegeben
- Logistic function → range 0 bis 1 (Min-Max Skalierung des Labels nötig)
- hyperbolic tangent → range -1 bis 1 (Labels müssen dementsprechend skaliert werden)

# Hyperparameter - Regression

Loss

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (x_i - \hat{x}_i)^2$$

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |x_i - \hat{x}_i|$$

# Hyperparameter - Klassifikation

## Output Neurons + Activation functions

- Binary Classification:
  - 1 Neuron für Binary Classification: Spam or no Spam
  - Logistic Activation Function
- Multilabel Binary Classification:
  - 1 Neuron per Label, bspw. 1 Neuron für Spam or no Spam und 1 Neuron für urgent or not urgent  
(Achtung nicht exklusiv → Prediction könnte Spam + urgent sein )
  - Logistic Activation Function
- Multiclass Prediction (exklusiv):
  - 1 Neuron per Class
  - Softmax Activation Function (Wahrscheinlichkeitsverteilung)  
→ Werte normiert zwischen 0-1 und addieren sich zu 1 auf

# Hyperparameter - Klassifikation

## Loss

- Categorical Crossentropy (One-Hot kodiert)

```
>>> y_true = [[0, 1, 0], [0, 0, 1]]  
>>> y_pred = [[0.05, 0.95, 0], [0.1, 0.8, 0.1]]
```

- Sparse categorical crossentropy

```
>>> y_true = [1, 2]  
>>> y_pred = [[0.05, 0.95, 0], [0.1, 0.8, 0.1]]
```

# Merktabellen

## Regression

Hyperparameter	Typical value
# input neurons	One per input feature (e.g., $28 \times 28 = 784$ for MNIST)
# hidden layers	Depends on the problem, but typically 1 to 5
# neurons per hidden layer	Depends on the problem, but typically 10 to 100
# output neurons	1 per prediction dimension
Hidden activation	ReLU (or SELU, see <a href="#">Chapter 11</a> )
Output activation	None, or ReLU/softplus (if positive outputs) or logistic/tanh (if bounded outputs)
Loss function	MSE or MAE/Huber (if outliers)

# Merktabellen

## Klassifikation

Hyperparameter	Binary classification	Multilabel binary classification	Multiclass classification
Input and hidden layers	Same as regression	Same as regression	Same as regression
# output neurons	1	1 per label	1 per class
Output layer activation	Logistic	Logistic	Softmax
Loss function	Cross entropy	Cross entropy	Cross entropy

# Einführung tensorflow Keras

## Sequential API

- Modellbildung anhand von Layer-Architekturen (`keras.layers`)
- Hinzufügen diverser Layer durch `model.add()` oder Sequential-Array
- Überblick über alle verfügbaren Layer: <https://keras.io/api/layers/>
- Outputs werden wie Name schon sagt sequentiell von Layer zu Layer weitergereicht
- Falls Individualisierung erforderlich Functional API nötig
  - Outputs und Inputs können individuell angepasst werden

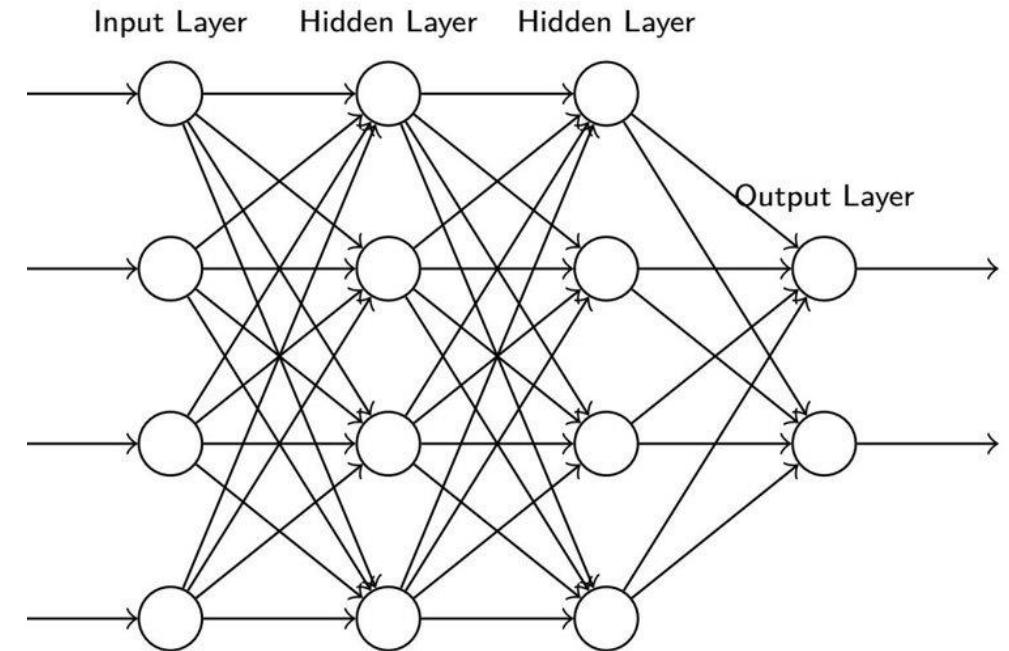
# Einführung tensorflow Keras

## Sequential API

Code

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[4]),
    keras.layers.Dense(4, activation="relu"),
    keras.layers.Dense(4, activation="relu"),
    keras.layers.Dense(2, activation='softmax')
])
```

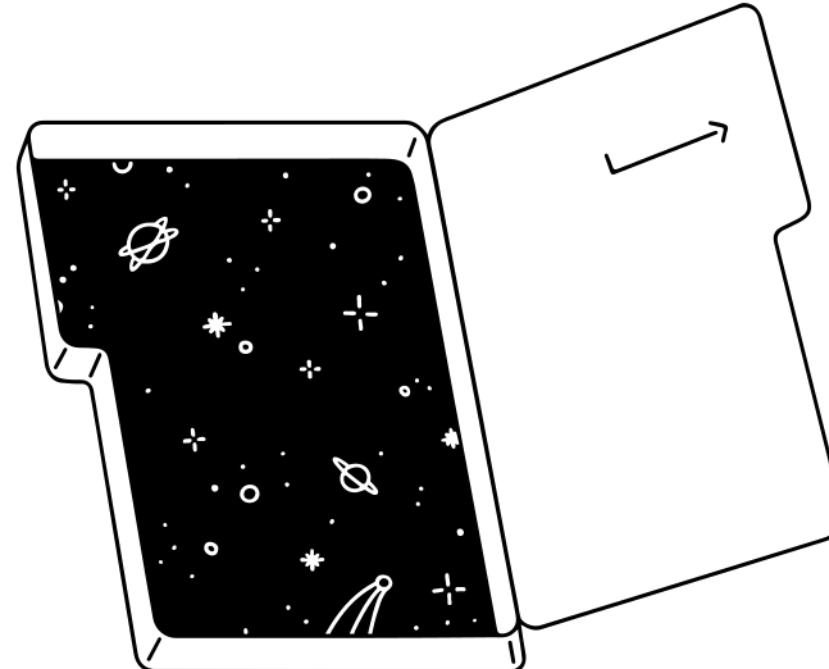
Repräsentation



# Regression & Klassifikation mit MLPs

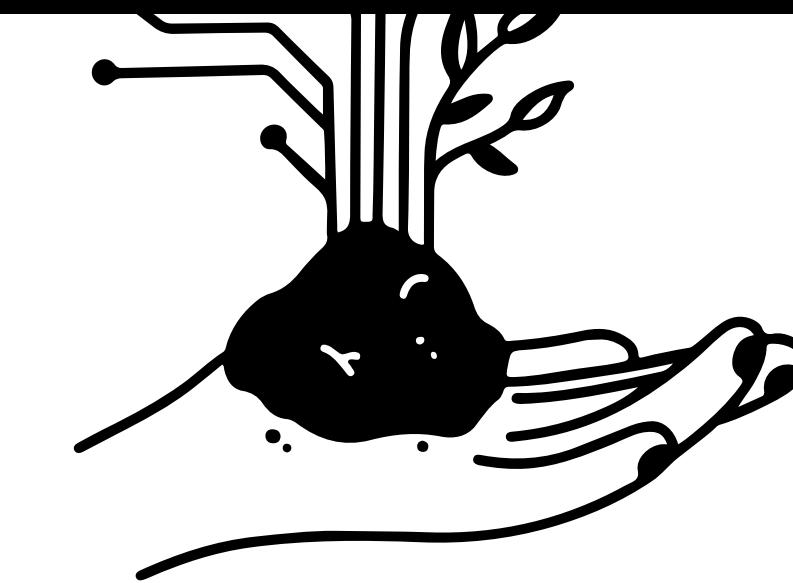
- Vorschau:  
[\*\*02\\_Supervised\\_Learning\\_Regession\\_Classification\*\*](#)
- [\*\*Übung\\_02\\_Regession\*\*](#)
- [\*\*Übung\\_03\\_Klassifikation\*\*](#)

## PRAXIS



01

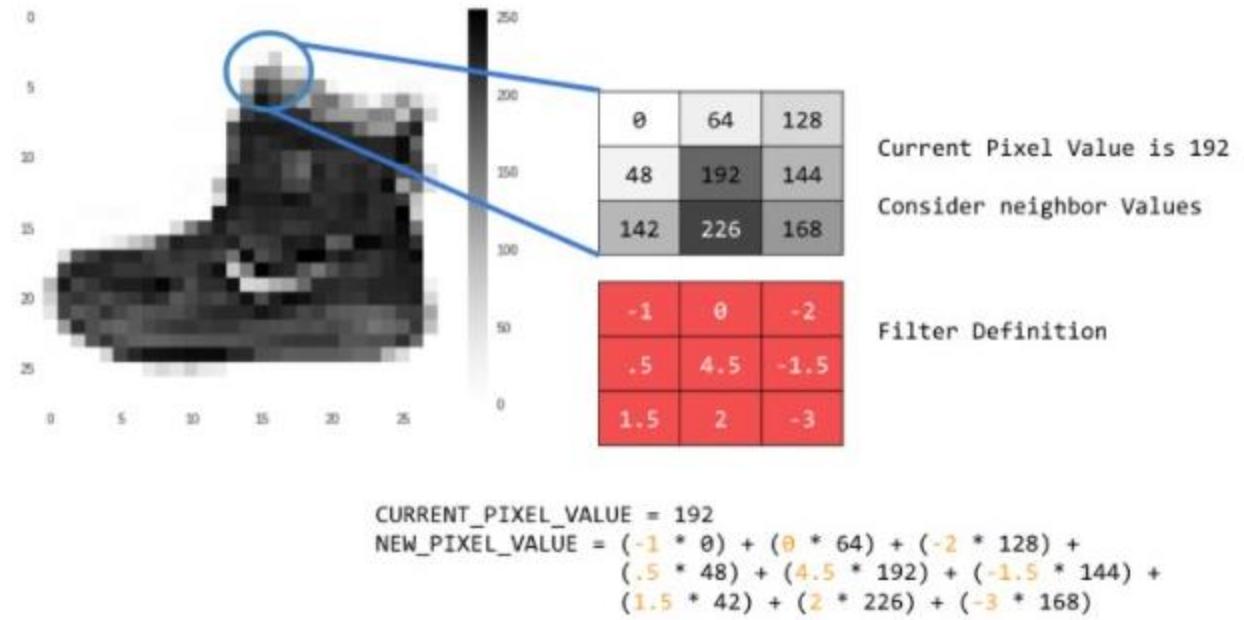
# Neural Networks (CNNs)



# Convolutional Neural Networks (CNNs)

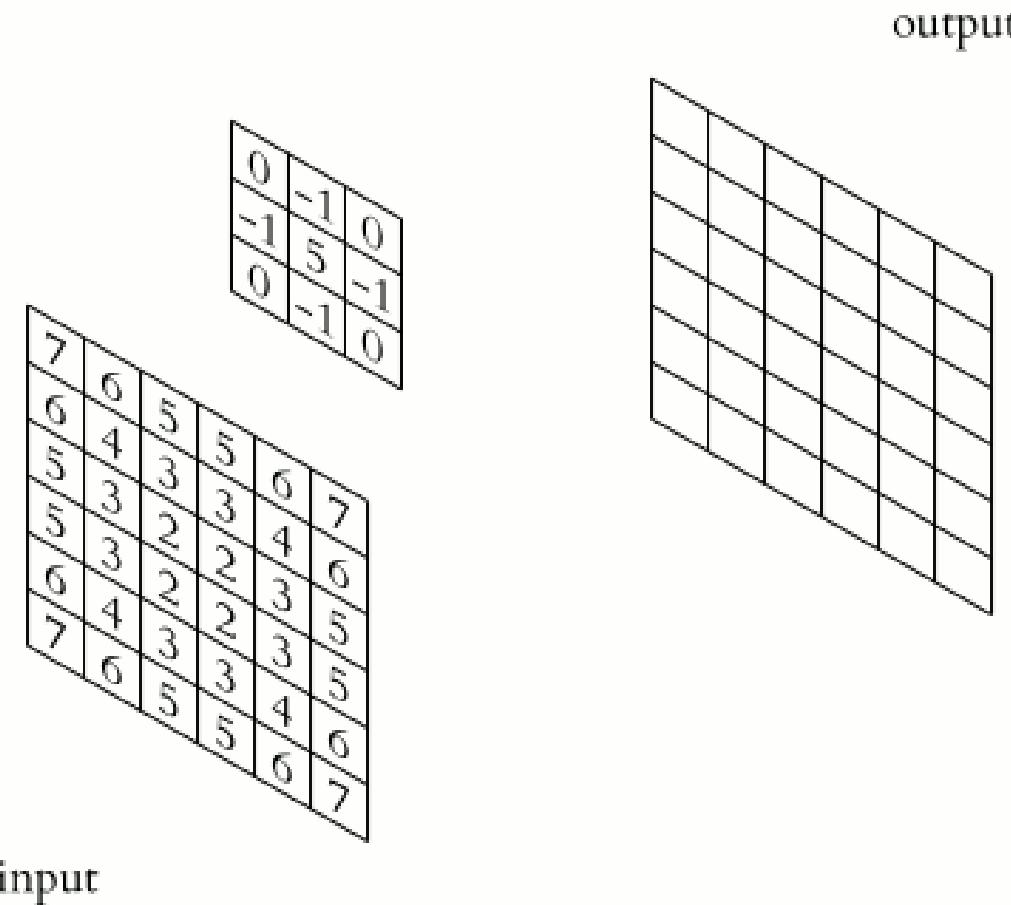
## Grundidee

- Filter, welcher über Bild fährt und daraus wichtige Features extrahiert  
→ Feature Mapping
- Jeder Pixelwert in einem Bild wird gescannt und dabei auf dessen „Nachbarwerte“ geachtet
- Werte werden mit Filter/Kernel multipliziert und zu einem neuen Pixelwert aufsummiert (gewichtete Summe).



# Convolutional Neural Networks (CNNs)

## Convolutional-Operation



# Convolutional Neural Networks (CNNs)

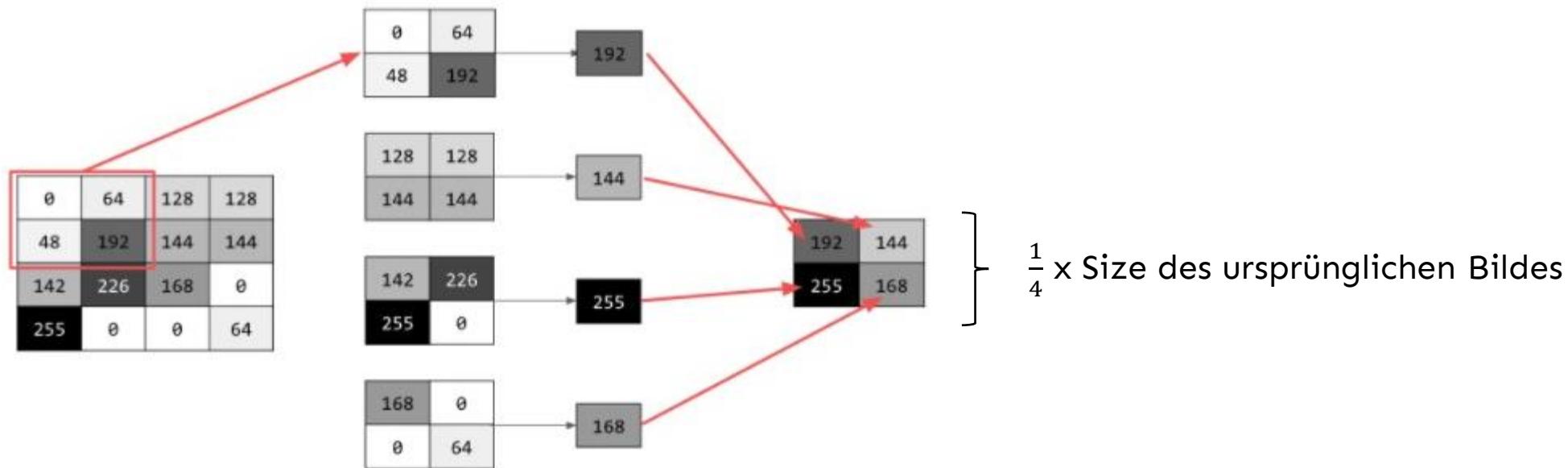
## Pooling-Layer

- Nachdem alle essenziellen Features eines Bildes extrahiert wurden, wird Pooling durchgeführt
- Informationen des Bildes werden komprimiert und auf die „wichtigen“ Features reduziert
- Es gibt mehrere Arten von Pooling, besonders beliebt ist Maximum oder Average Pooling
- Arbeitet wie Convolutions auch mit einer Filter Size

# Convolutional Neural Networks (CNNs)

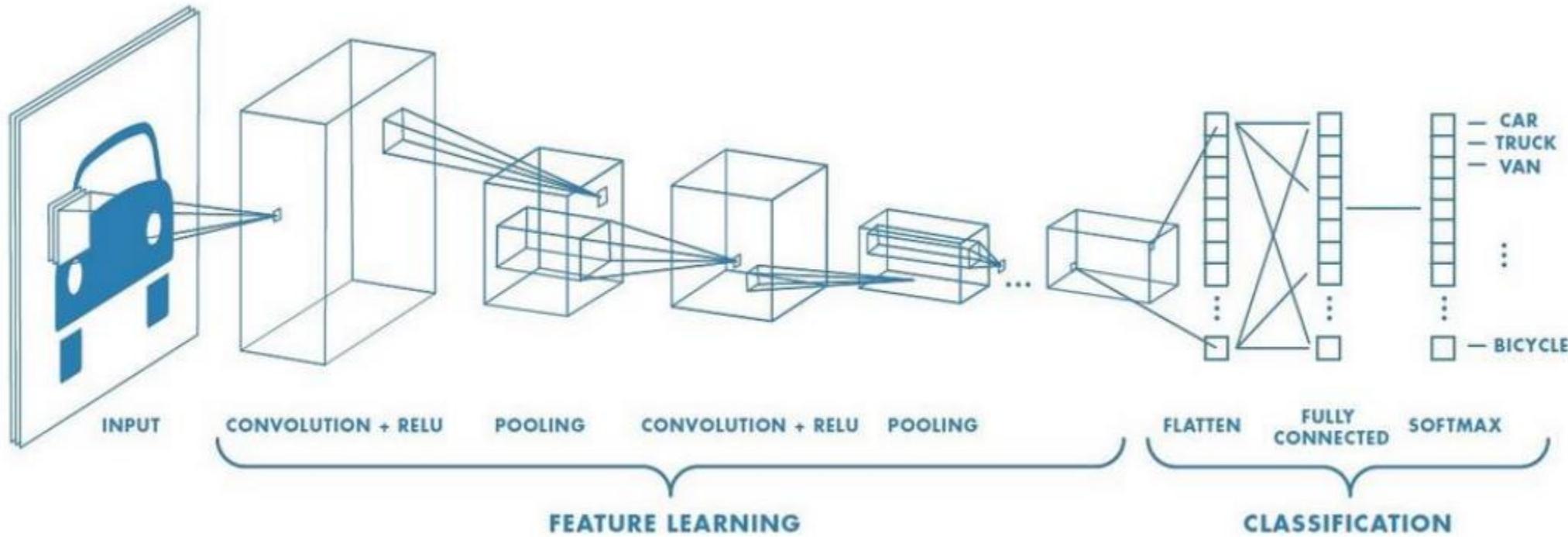
## Pooling-Layer

Max-Pooling mit einem (2,2) Filter:



# Convolutional Neural Networks (CNNs)

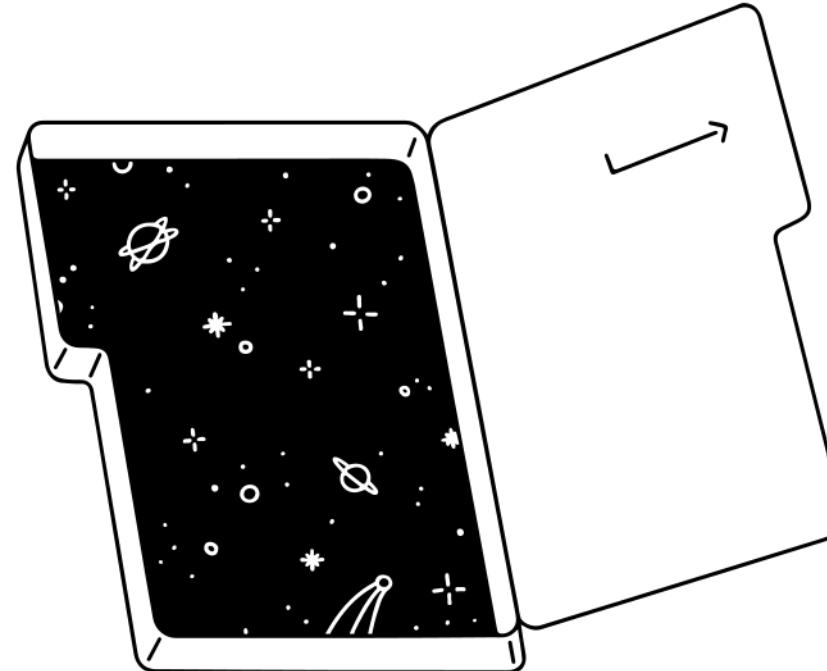
## Architektur



# CNNs

- Vorschau: **03\_MLP's vs CNNs**
- **Übung\_04\_CNNs**

## PRAXIS



# Grundlagen Supervised & Unsupervised Learning

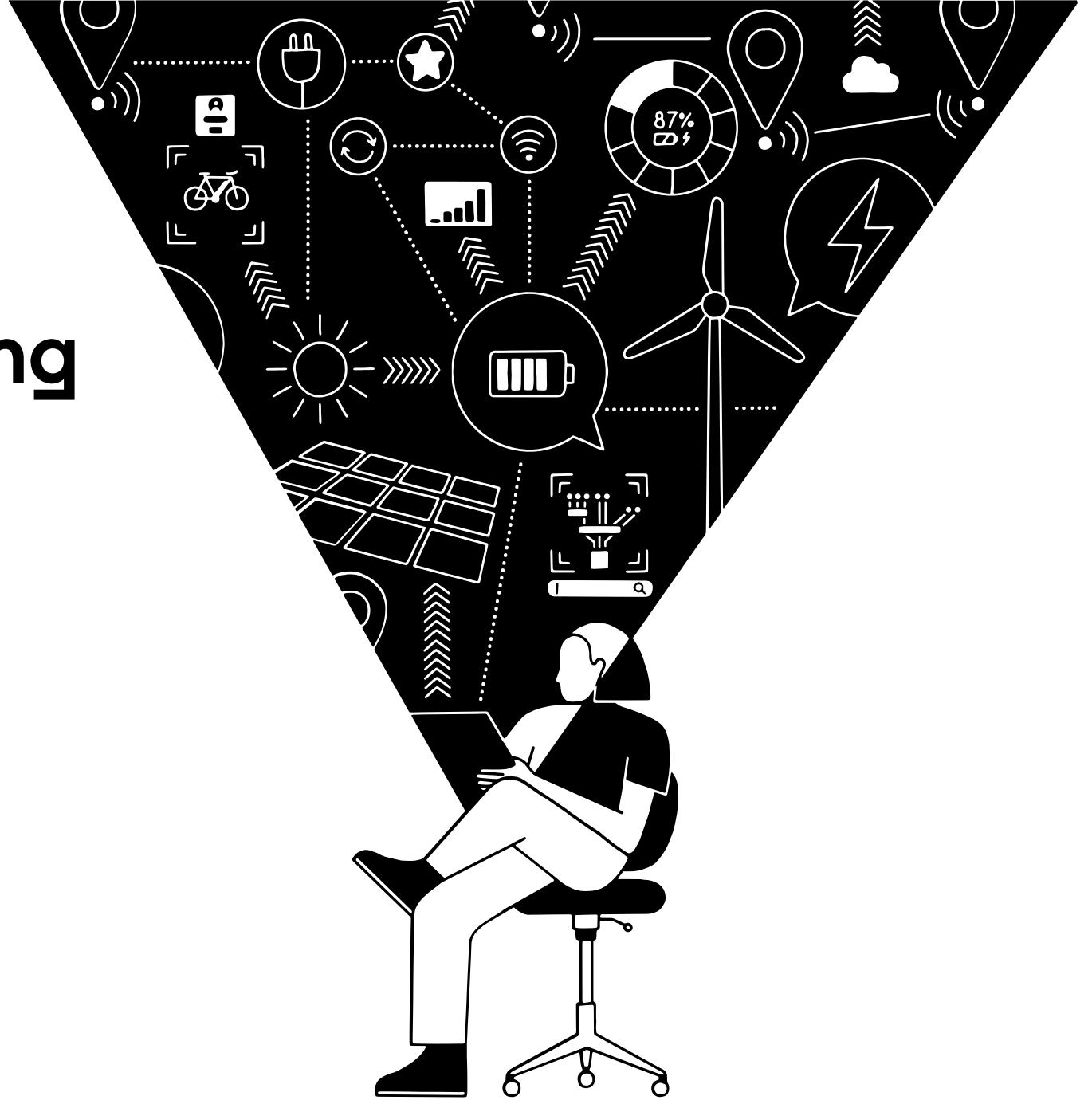
## Tag 2

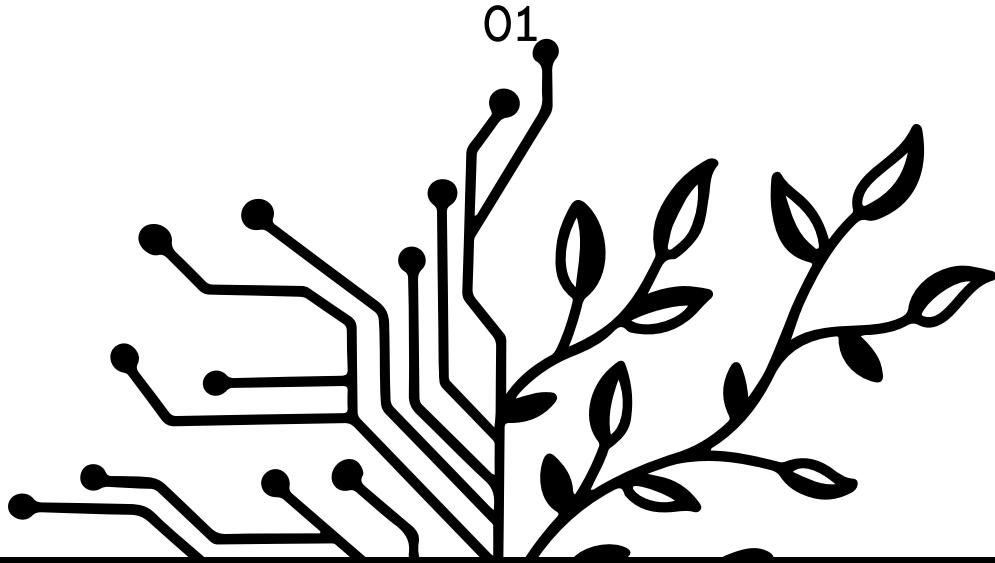
Mit Anwendungsbeispielen in TensorFlow Keras



Tobias Krebs

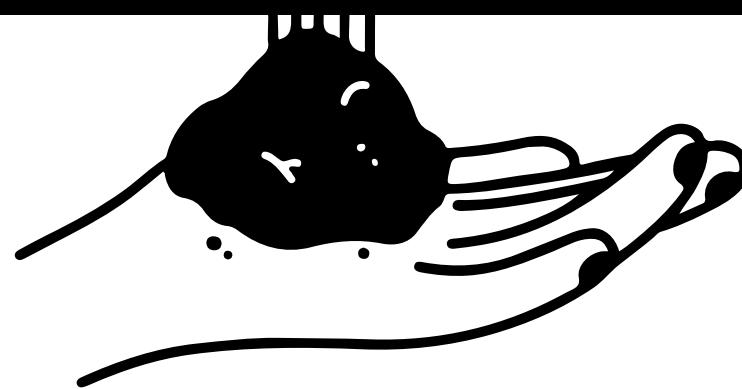
**exeta**





# Unsupervised Learning

## Basic Clustering

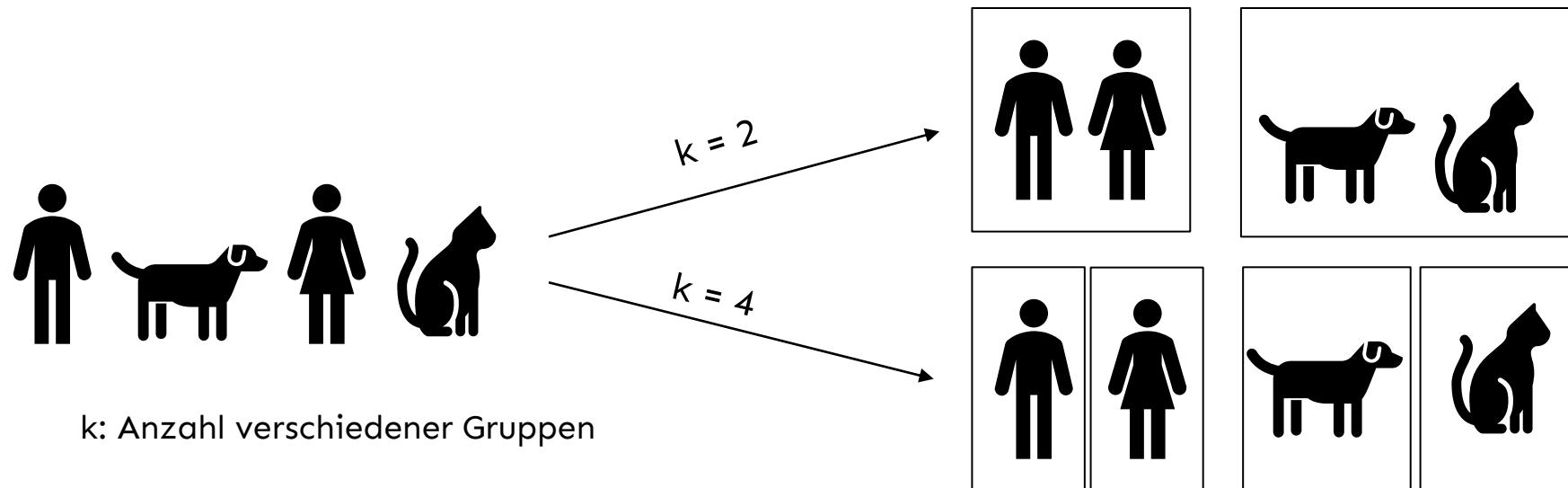


# Unsupervised Learning

## Fragestellung Unsupervised Learning:

Können Accounts verschiedener Nutzer in bestimmte Gruppen aufgeteilt werden?

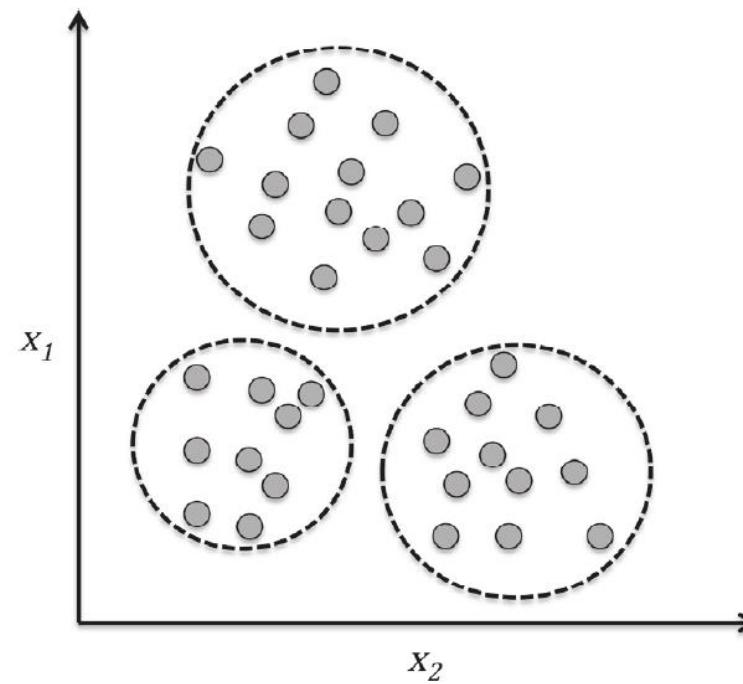
- Keine Angaben über ein bestimmtes Ziel in den Daten vorhanden
- Ziel wird allein anhand vorliegender Daten und deren Ähnlichkeit zueinander bestimmt



# Unsupervised Learning

## Clustering

**Clustering:** Aufteilung von nicht gelabelten Daten in ähnliche Gruppen

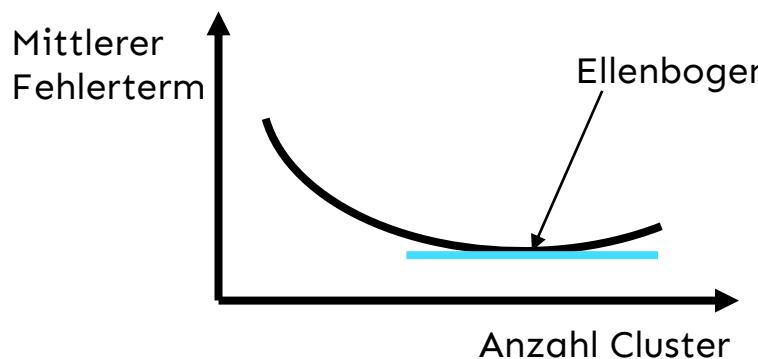


# Clustering

## K-Means

- Prototype-based Clustering - jedes Cluster wird von einem zentralen Datenpunkt ("centroid") repräsentiert
- Leichte Implementierung und hohe Effizienz
- Anzahl der Cluster ( $k$ ) müssen vorher definiert werden

→ Beurteilung der Güte beispielsweise mit "Ellenbogenkriterium" oder Silhouettendiagramm



Raschka, S., Machine Learning mit Python, 2016, S.314 ff.

# Clustering

## K-Means

### Algorithmus:

1. Auswahl der  $k$  Zentroiden aus Objekten als anfängliche Clusterzentren.
2. Alle Objekte dem nächsten Zentroiden zuweisen.
3. Neuberechnung des Zentroiden mit den aus Schritt 2 zugewiesenen Objekten.
4. Wiederholung von Schritt 2 und 3, bis sich die Zuordnung nicht mehr ändert (entweder Schwellenwert oder maximale Iterationen werden vorgegeben).

Aber was ist unser Ähnlichkeitsmaß ?

# Clustering

## K-Means

### Euklidische Distanz:

Datenobjekte die sich ähnlich sind, sind nahe beieinander platziert

$$\text{Formel: } d(x, y)^2 = \sum_{j=1}^m (x_j - y_j)^2 = \|x - y\|_2^2$$

→ Durch euklidische Distanz kann k-Means Algorithmus als Optimierungsaufgabe formuliert werden

Summe der quadrierten Abweichungen innerhalb eines Clusters soll minimiert werden

$$\text{Formel: SSE} = \sum_{i=1}^n \sum_{j=1}^k w^{(i,j)} \|x^{(i)} - \mu^{(j)}\|_2^2$$

# Clustering

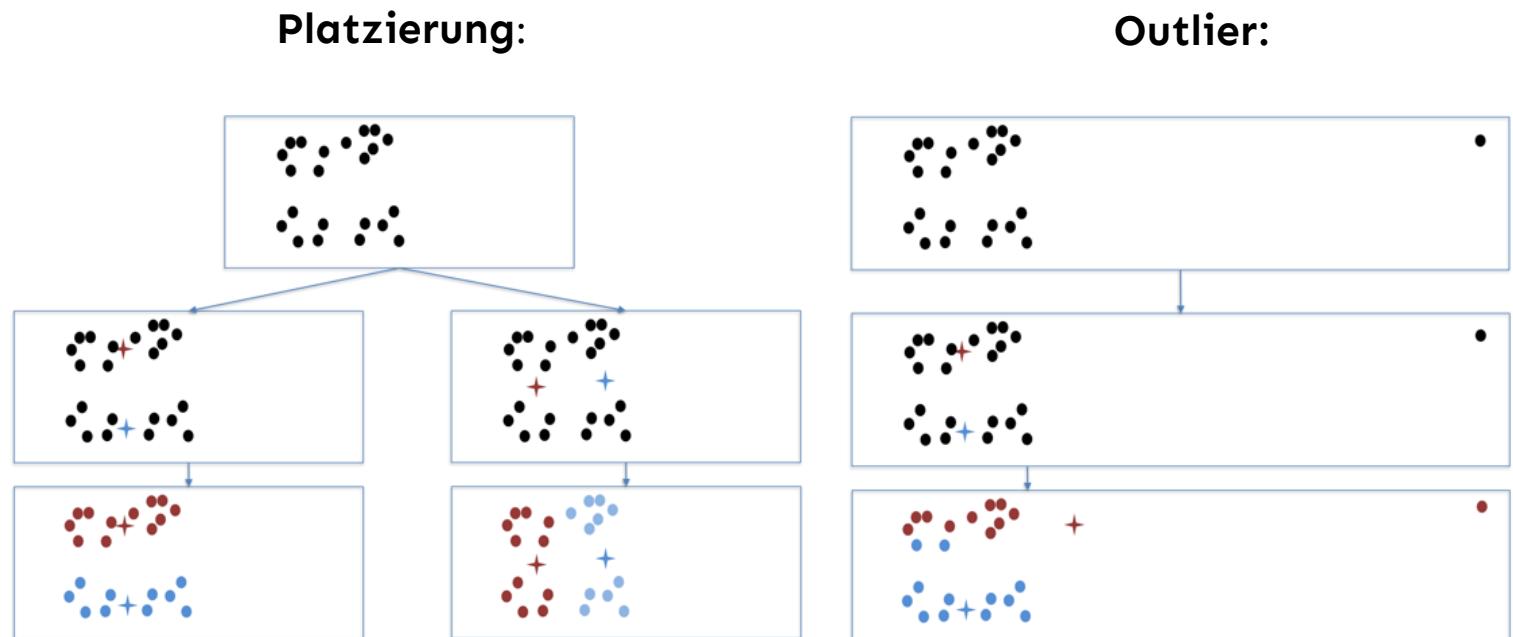
## K-Means

Pros:

- Leicht zu implementieren
- Äußerst effektiv

Cons:

- Initiale Platzierung der Zentroide ist wichtig
- Einfluss von Outliern



# Clustering Visualisierung

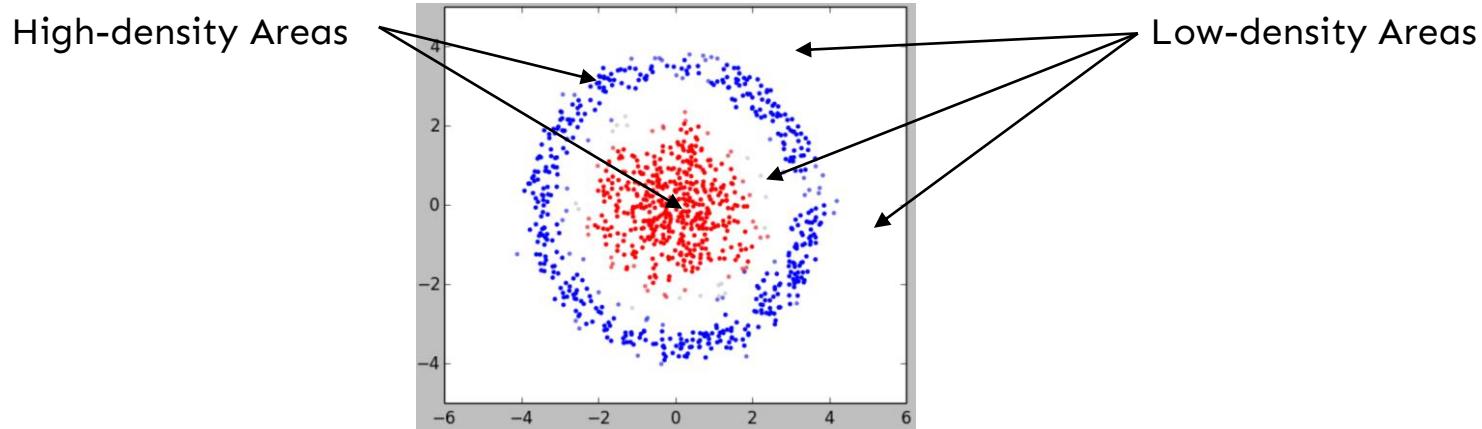
## K-Means

<https://www.naftaliharris.com/blog/visualizing-k-means-clustering/>

# Clustering

## DBSCAN

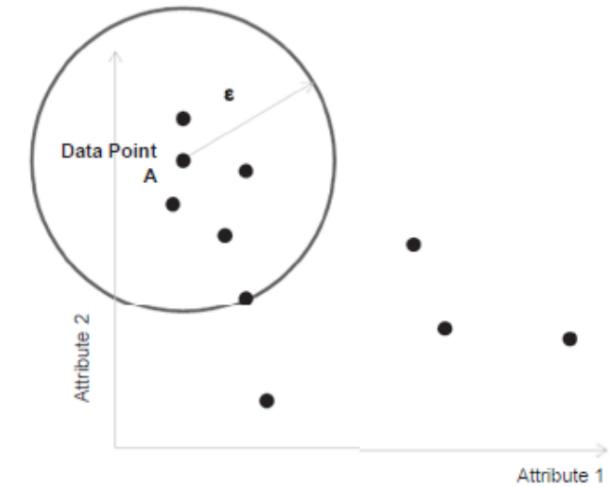
- In vielen Anwendungsfällen sind die Anzahl an Clustern vorher nicht bekannt  
→ Problem bei K-Means
- Außerdem können komplexe Datenstrukturen für K-Means ein Problem werden
- Für solche Situationen, kann man auf Density-Based Clustering zurückgreifen:



# Clustering

## DBSCAN

- Dichte messen
  - Dichte = Anzahl von Punkten innerhalb eines gewissen Bereichs mit dem Radius  $\varepsilon$  (epsilon)
  - Beispiel: Dichte um den Datenpunkt A ist 6
- Basierend auf dieser Idee identifiziert der DBSCAN Algorithmus Dichteregionen mit den Hyperparametern radius ( $\varepsilon$ ) und den „minimum number of points“



# Clustering

## DBSCAN

### DBSCAN

#### (Density-Based Spatial Clustering of Applications with Noise)

1. Berechnen der Density für jeden Datenpunkt abhängig von Epsilon  
→ Density > MinPts = High-Density Area

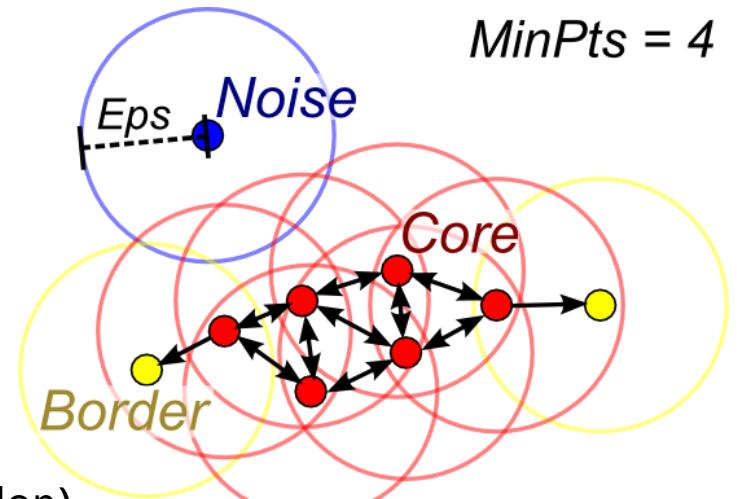
2. Wenn Datenpunkt = High-Density Area  
→ Core Point

Falls Datenpunkt ≠ High-Density Area, aber Core Point innerhalb Radius (Epsilon)

→ Border Point

3. Falls Datenpunkt ≠ High-Density Area und kein Core Point innerhalb Radius (Epsilon)

→ Noise



Source: <https://medium.com/@elutins/dbscan-what-is-it-when-to-use-it-how-to-use-it-8bd506293818>

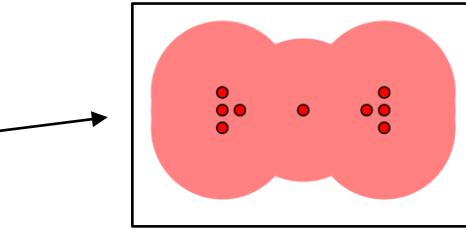
# Clustering

## DBSCAN

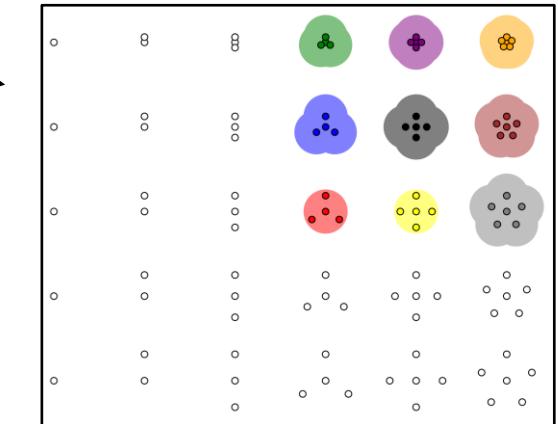
### Probleme

- Wenn ein Dataset Regionen mit variierender Dichte enthält kann DBSCAN diese nicht identifizieren

- Entweder setzen wir MinPoints zu klein/Epsilon zu hoch  
→ Worst Case: Es wird nur 1 Cluster identifiziert



- Oder wir setzen MinPoints zu hoch/Epsilon zu klein  
→ Worst Case: Alle Punkte werden als „Noise“ identifiziert,  
Low-density Areas werden übergangen.



→ K-Means Clustering ist passender in diesem Fall

# Clustering Visualisierung

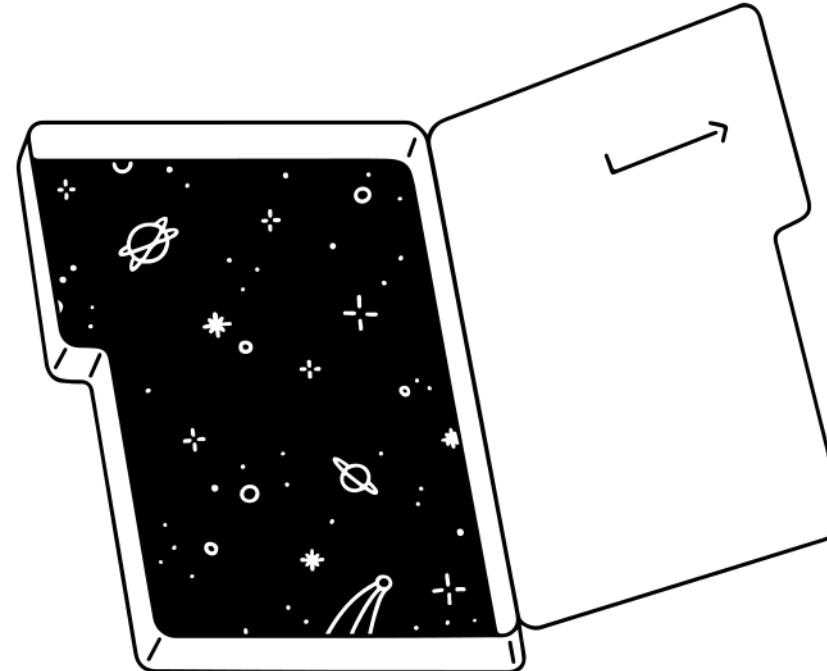
DBSCAN

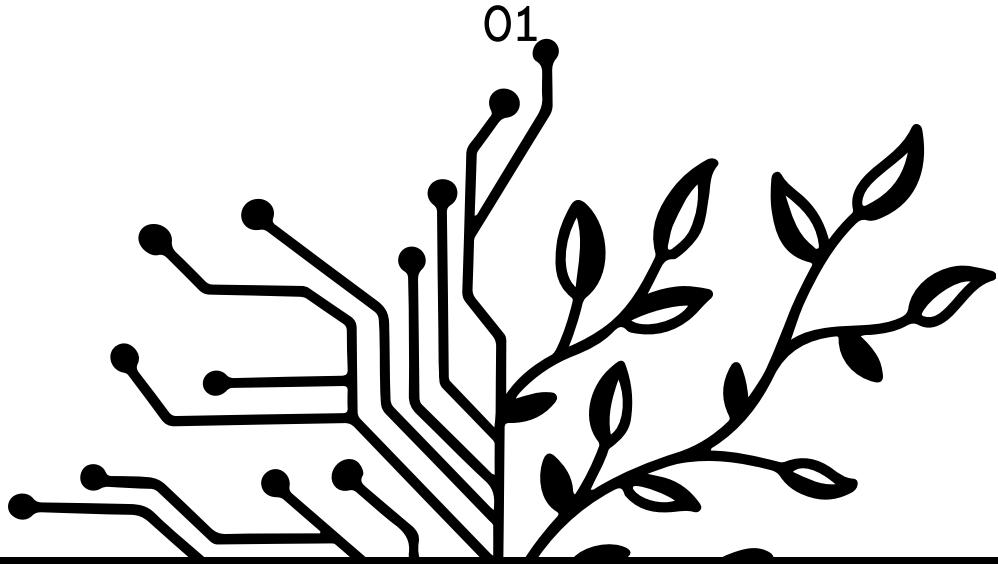
<https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/>

# Basics Clustering

- Vorschau: [04\\_Basic\\_Clustering](#)
- [Übung\\_05\\_Clustering](#)

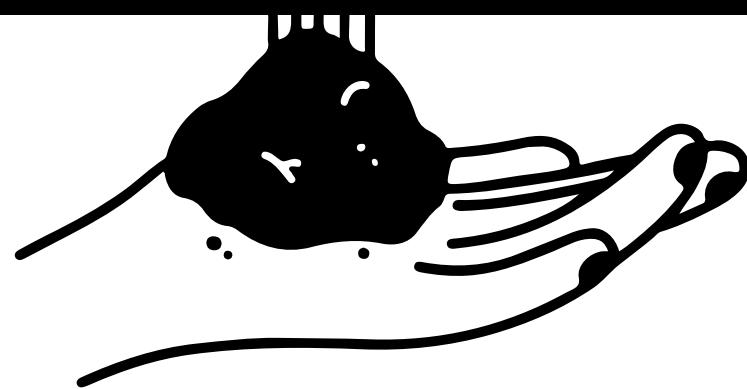
PRAXIS





01

# Neural Networks RNNs + LSTMs

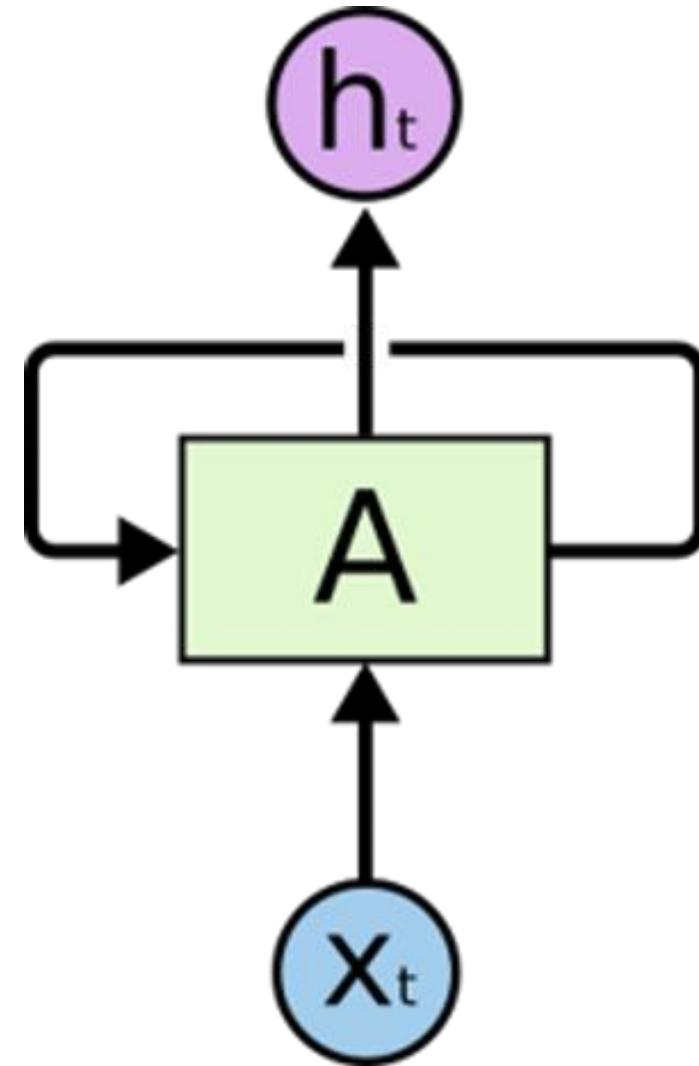


# Einfache Rekurrente Neurone

Supervised Learning

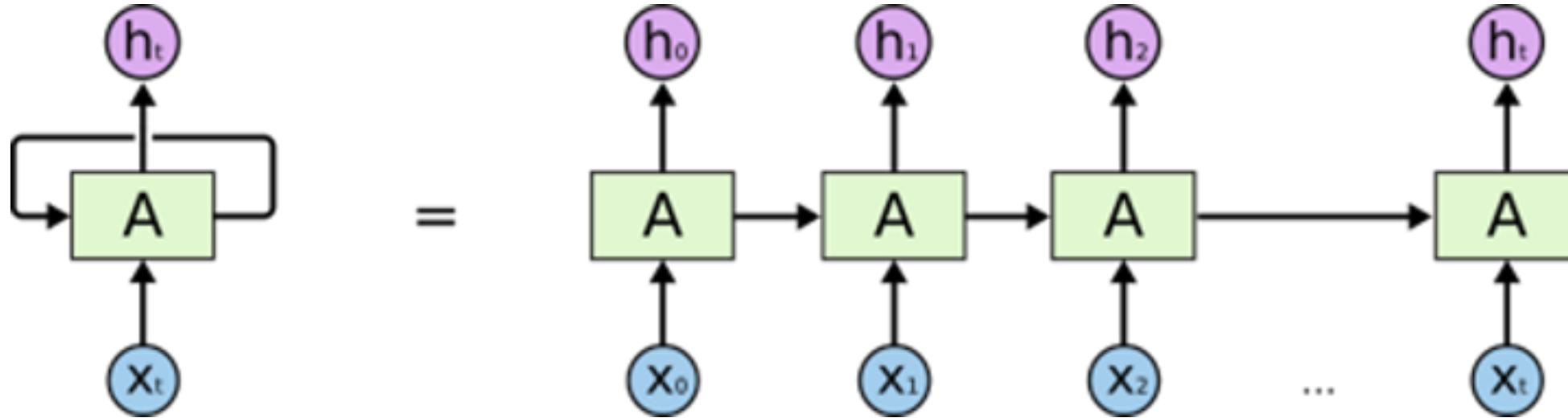
- RNNs haben einen Feedback Loop, welche den Output zurück ins Netzwerk führt
- Input:  $X_t, h_{t-1}$
- Output:  $h_t$

$h_t$  ist der Output der Hidden Layer zum Zeitpunkt t



# Rekurrente neuronale Netze

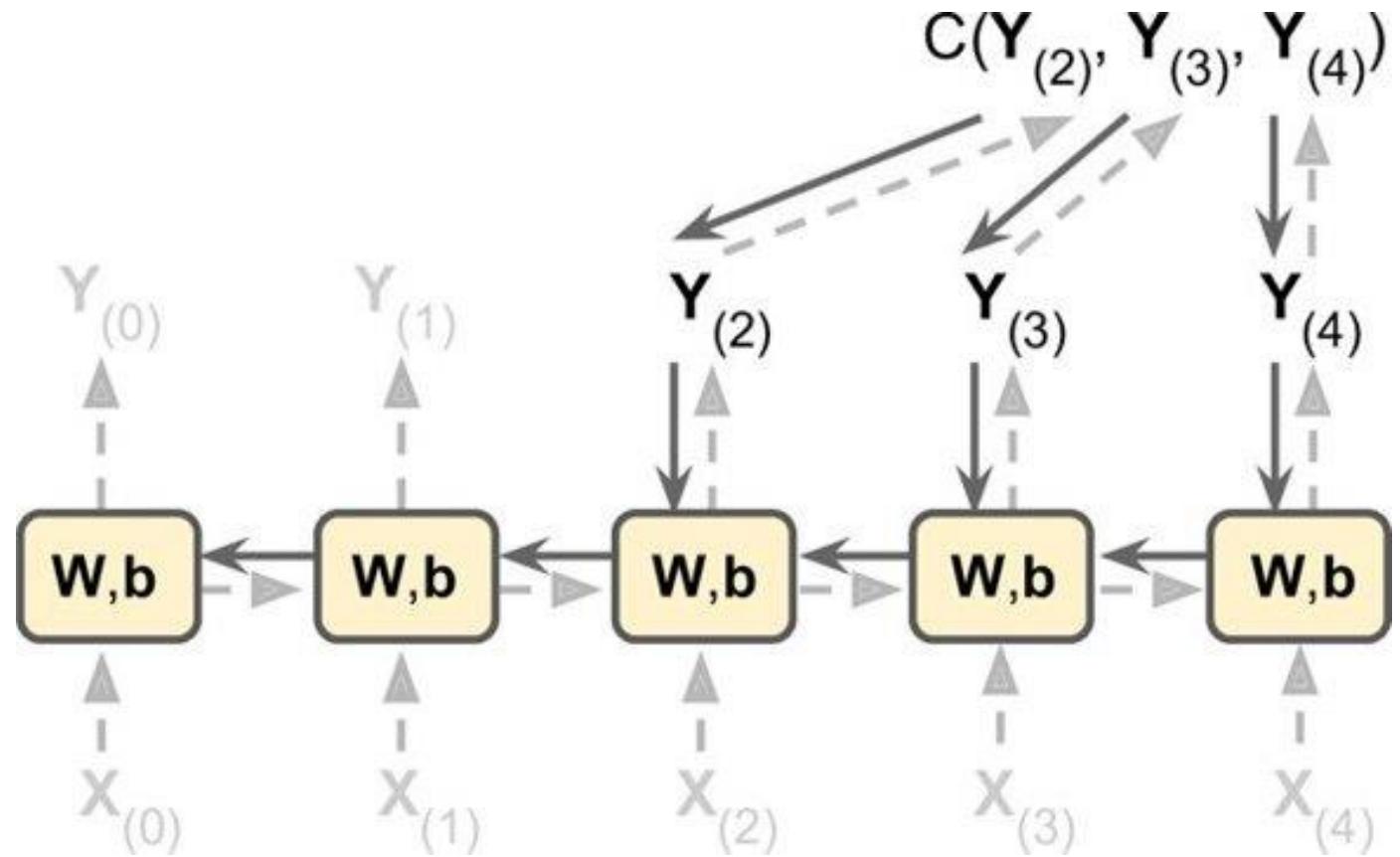
- RNNs können in der Zeit aufgerollt werden
- Das aufgerollte Netzwerk ist ähnlich zu einem tiefen Feedforward Netzwerk
- Prinzip der Backpropagation bleibt gleich (BPTT)
- Zum ersten Zeitschritt existiert kein Hidden State: Initialisiere mit Nullen



<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

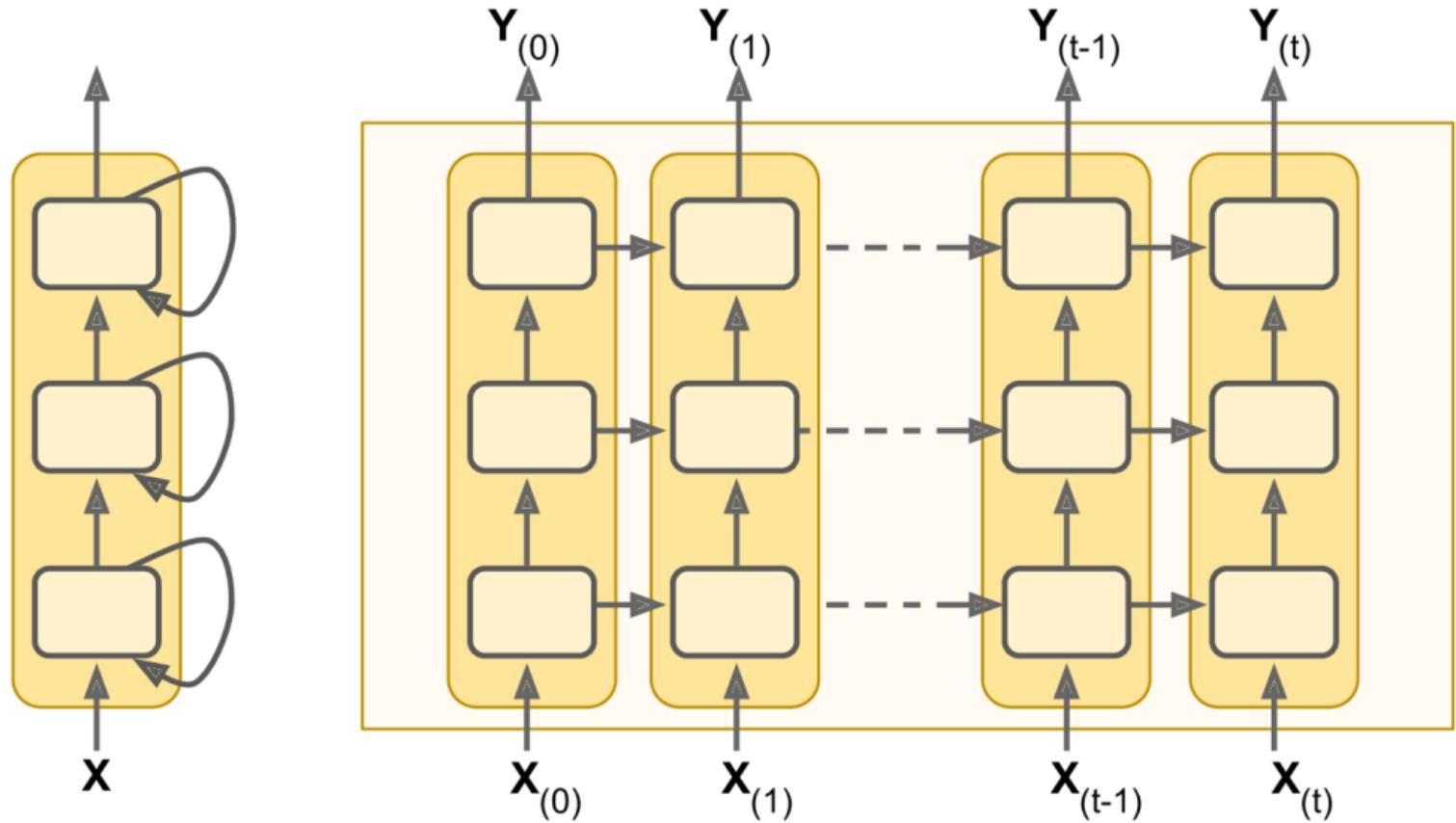
# Backpropagation through time (BPTT)

- Aufgerollte RNNs werden mit normaler Backpropagation trainiert
- Outputs können von der Loss Funktion ignoriert werden
- Vanishing/Exploding Gradient Problem:  
→ Ebenso wie tiefe Feedforward Netze haben auch RNNs Probleme mit instabilen Gradienten

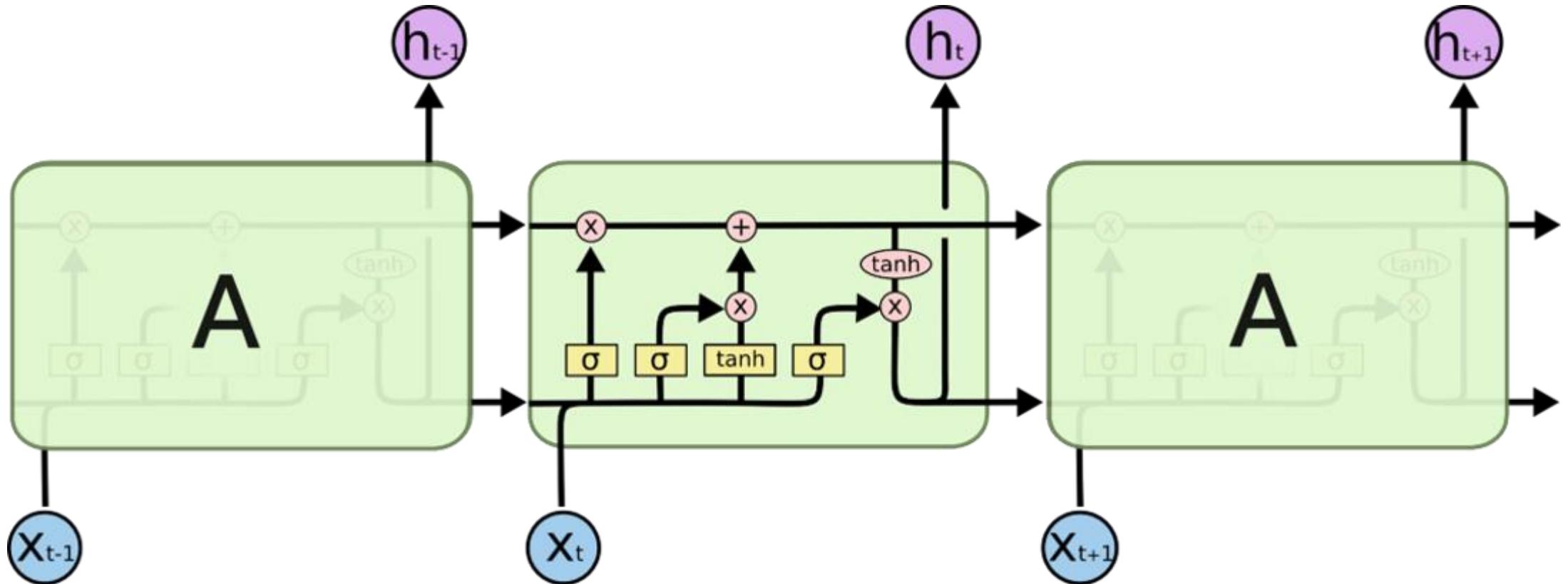


# Multilayer RNNs

- Output einer RNN layer kann als Input der nächsten RNN layer dienen
- In keras muss man das keyword **return\_sequences = True** setzen damit die Layer eine Sequenz zurückgibt



# LSTMs (Long Short-Term Memory Network)

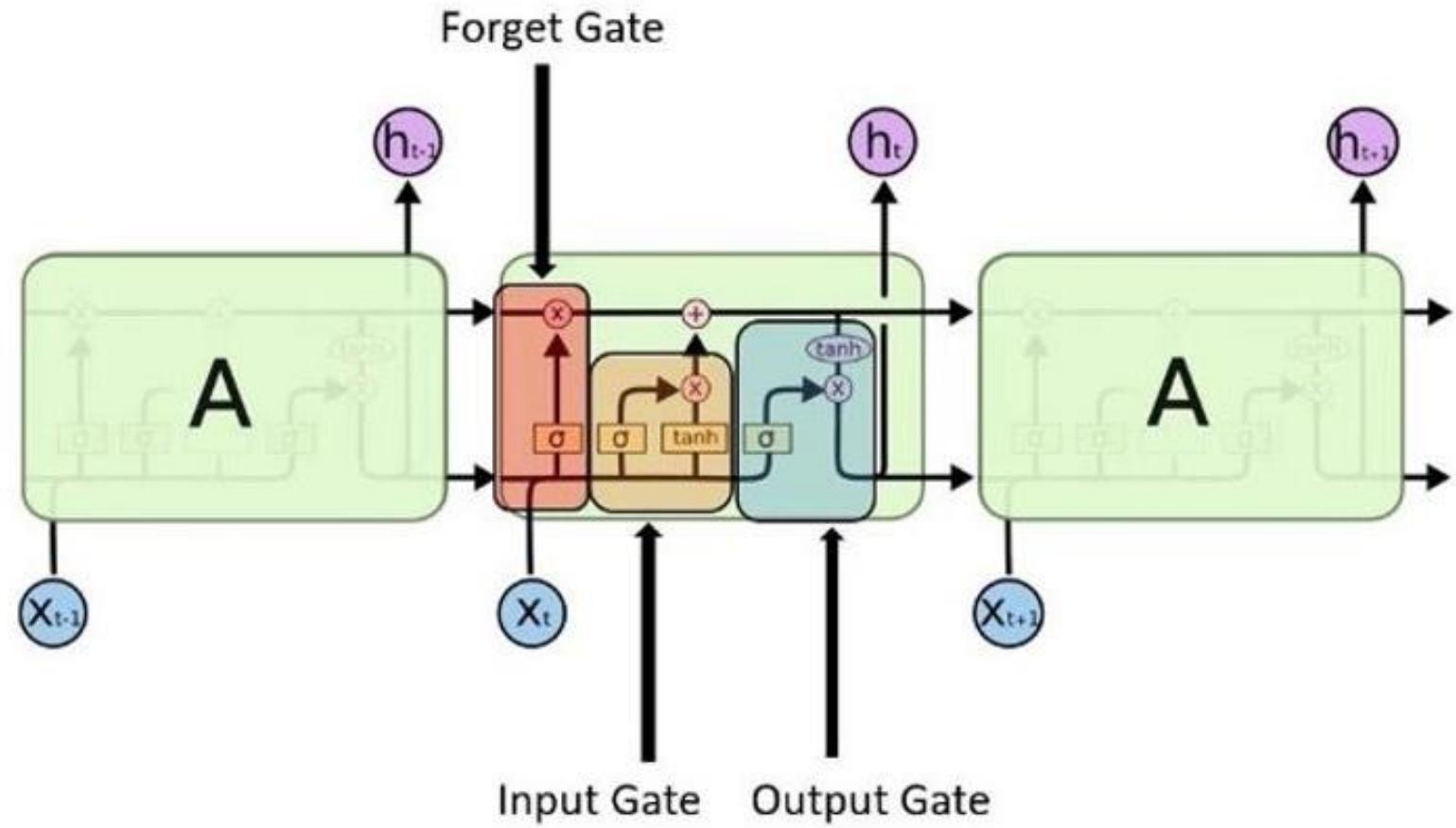


- LSTM (Long Short-Term Memory) (Hochreiter & Schmidhuber, 1997)
- LSTMs besitzen einen Cell State zum Speichern von Informationen, um das Short-Term Memory Problem von RNNs in den Griff zu bekommen

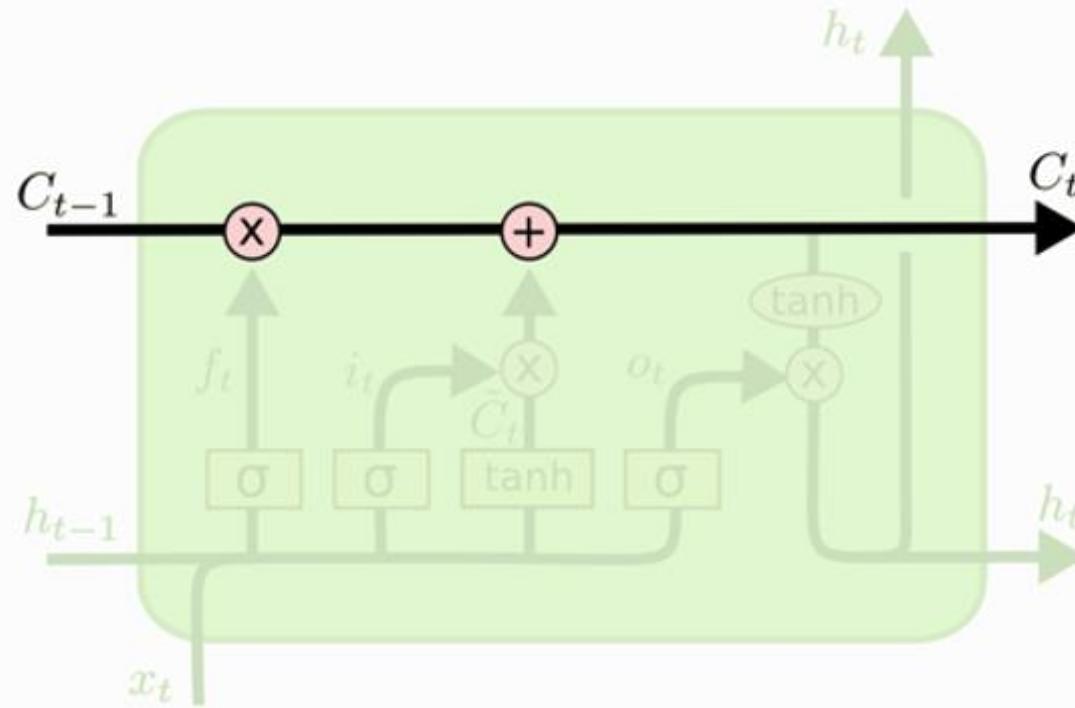
<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# LSTM Zelle

- Die LSTM Zelle besitzt drei Gates, welche den Informationsfluss steuern:
  - Forget Gate
  - Input Gate
  - Output Gate



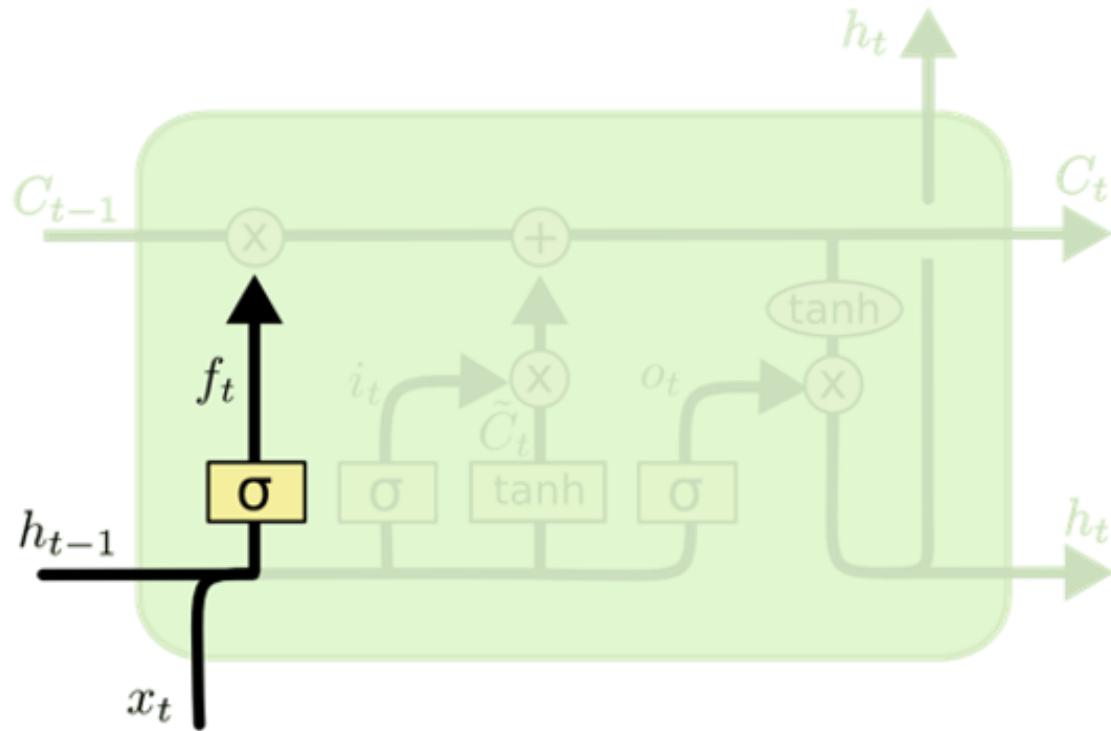
# Cell state



- Cell state durchläuft die LSTM Zelle und dient als Gedächtnis
- Zwei Operationen:
  - Forget Gate: Multiplikation mit  $[0, 1]$  (steuert welche Informationen vergessen werden sollen)
  - Input Gate: Addition (steuert welche Informationen hinzugefügt werden sollen)

<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# Forget Gate

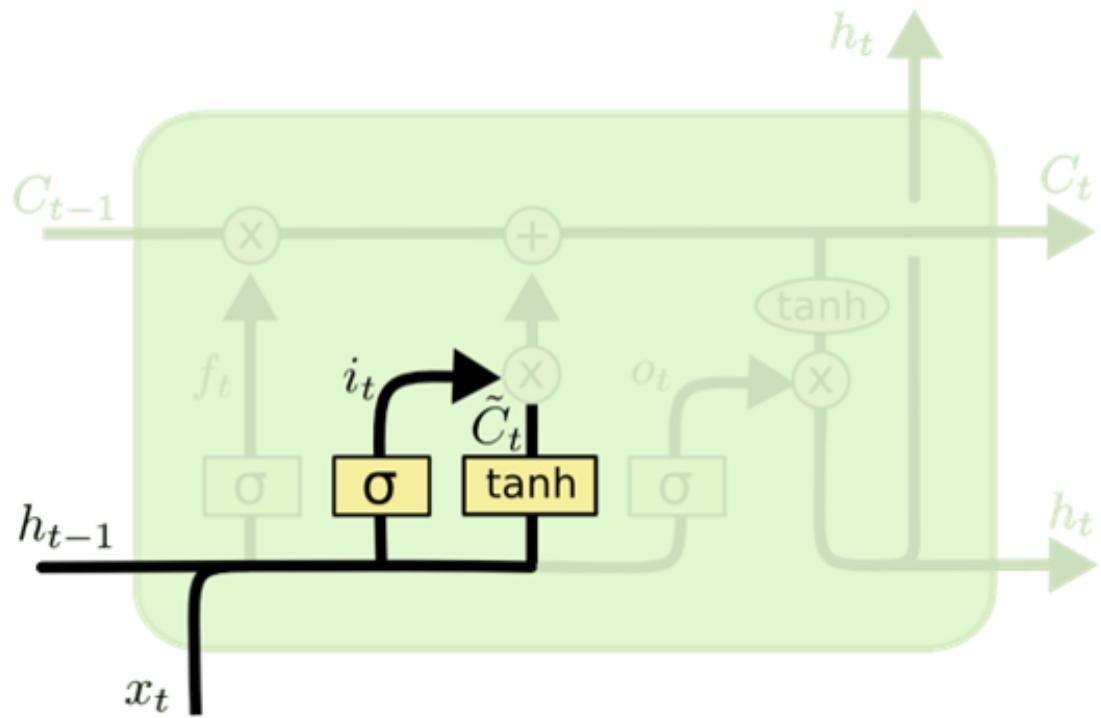


$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

- Input:  $x_t, h_{t-1}$
- Sigmoid bildet auf Intervall  $(0, 1)$  ab:
  - 0 = vergiss alles
  - 1 = erhalte alles

<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# Input Gate

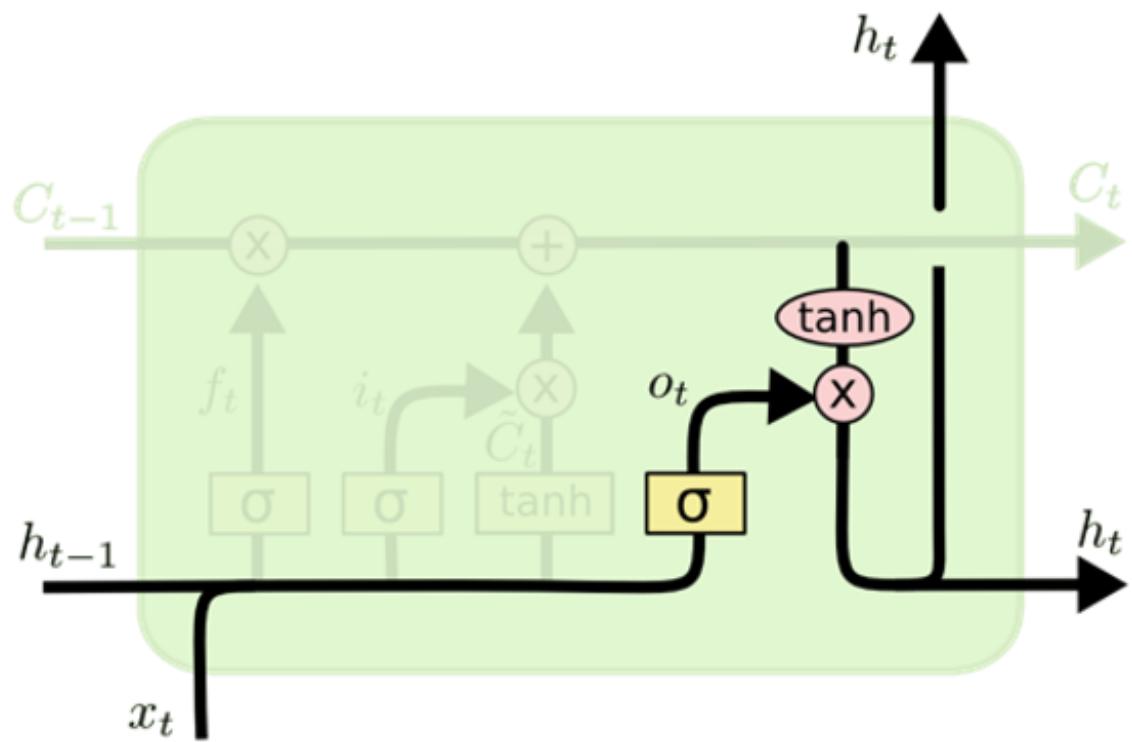


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

- Input:  $x_t, h_{t-1}$
- $i_t$  steuert welche Teile des Cell State ein Update erfahren
- Addition auf den Cell State steuert welche neuen Informationen hinzugefügt werden sollen

<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# Output Gate



$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

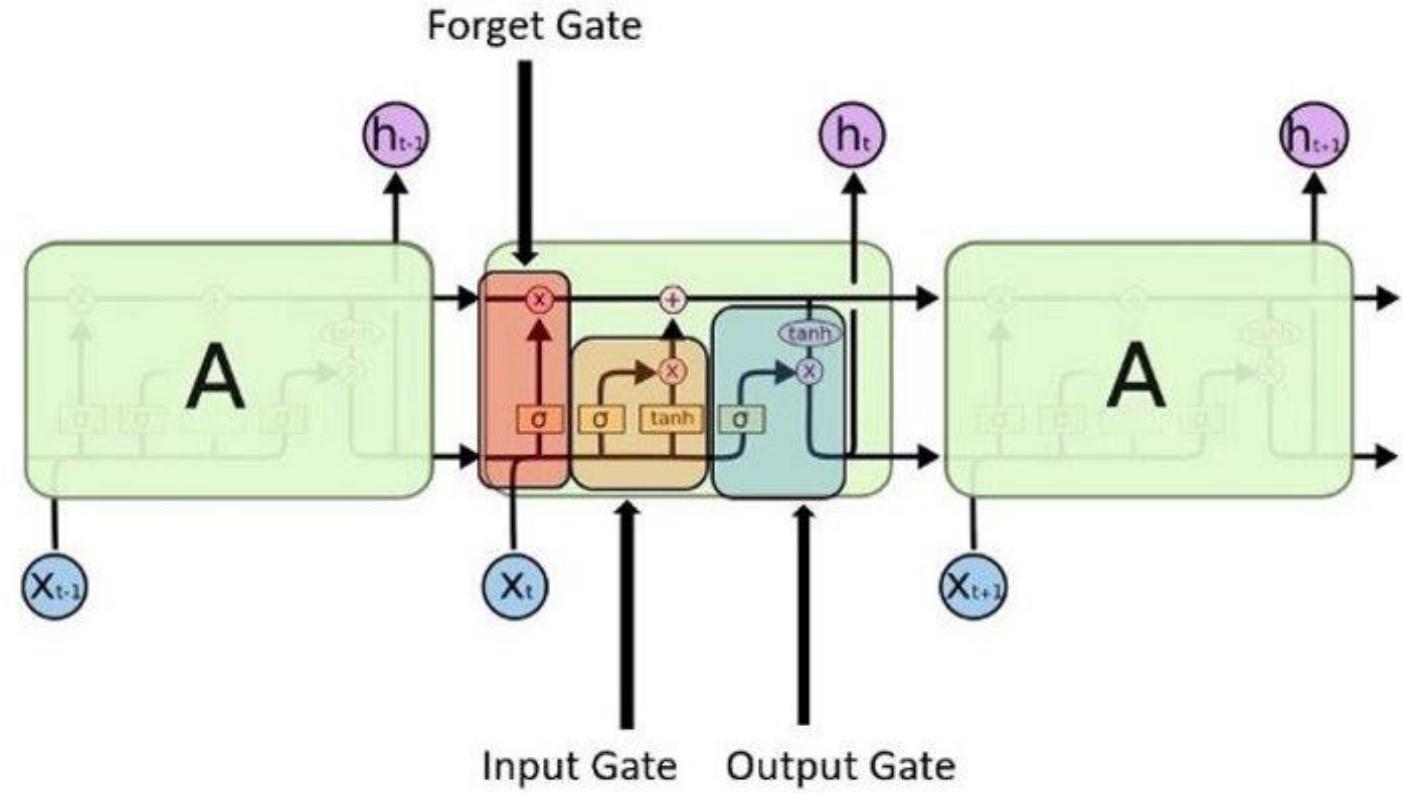
$$h_t = o_t * \tanh (C_t)$$

- Input:  $C_t, x_t, h_{t-1}$
- $O_t$  steuert welche Teile des Cell States als Output weiter gegeben werden

<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# LSTM Wrap-Up

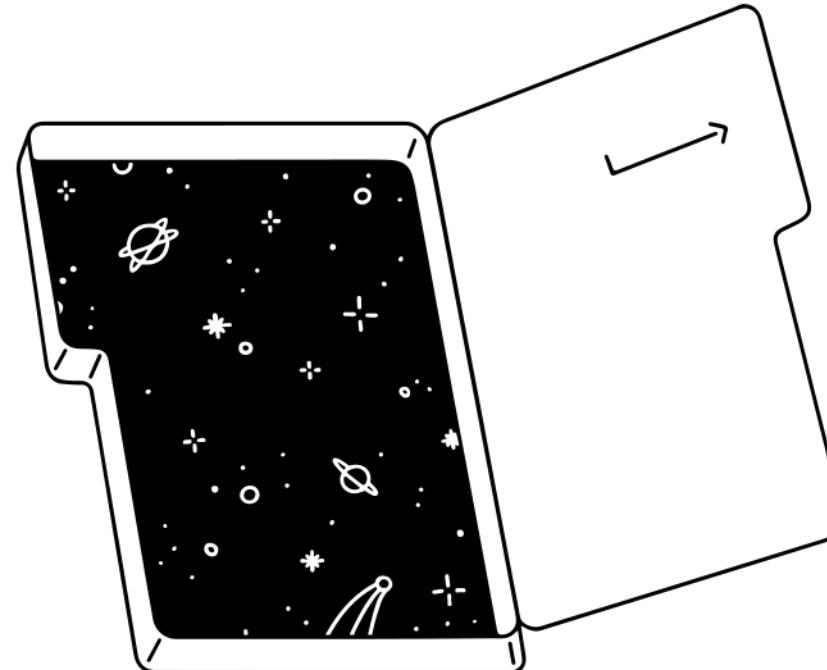
- Die LSTM Zelle besitzt drei Gates (Sigmoids):
  - Input Gate: kontrolliert wie viel des neuen Werts in den Cell State fließt
  - Forget Gate: kontrolliert was zum Cell State hinzugefügt wird, bzw. vergessen wird
  - Output Gate: kontrolliert welcher Teil des Cell States als Output ausgegeben wird



# RNNs und LSTMs

- Vorschau: [05\\_TimeSeries\\_LSTMS](#)
- [Übung\\_06\\_TimeSeries\\_LSTMS](#)

## PRAXIS



# Grundlagen Supervised & Unsupervised Learning

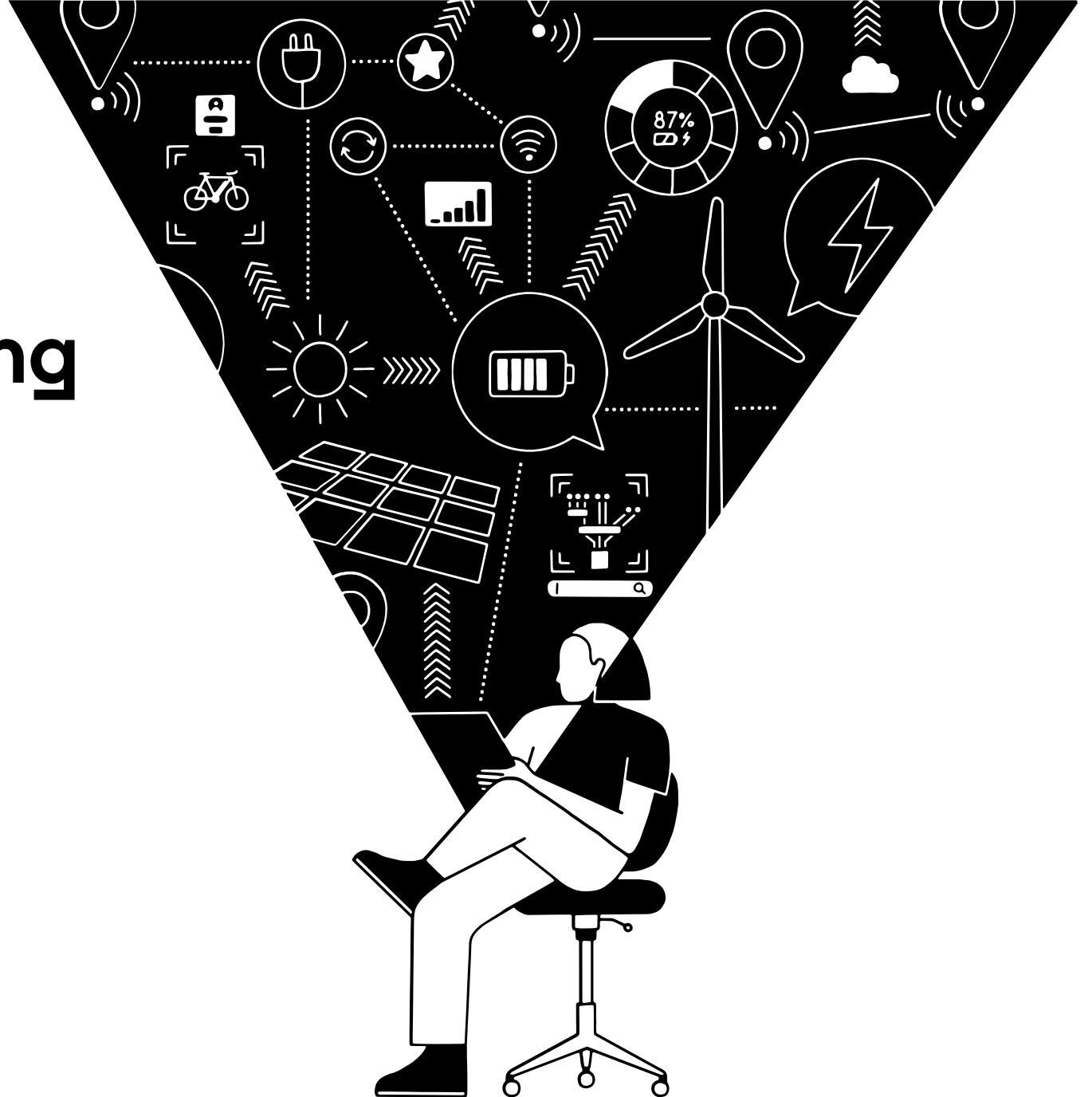
## Tag 3

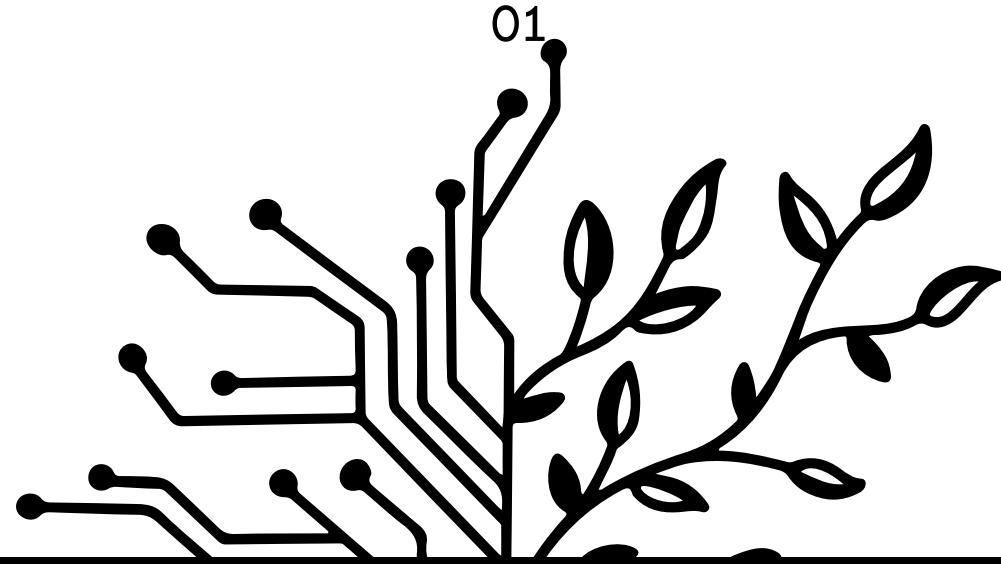
Mit Anwendungsbeispielen in TensorFlow Keras



Tobias Krebs

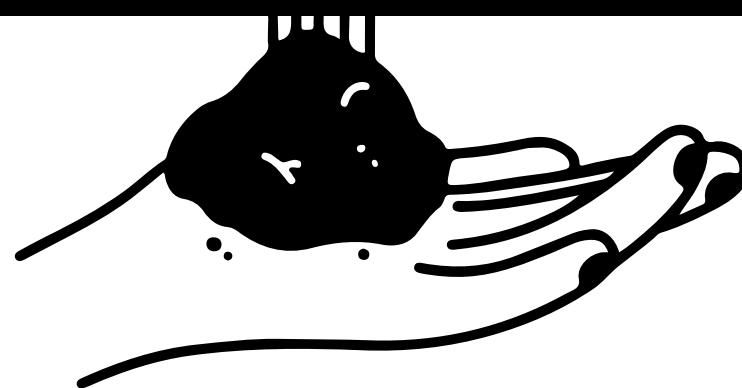
**exeta**





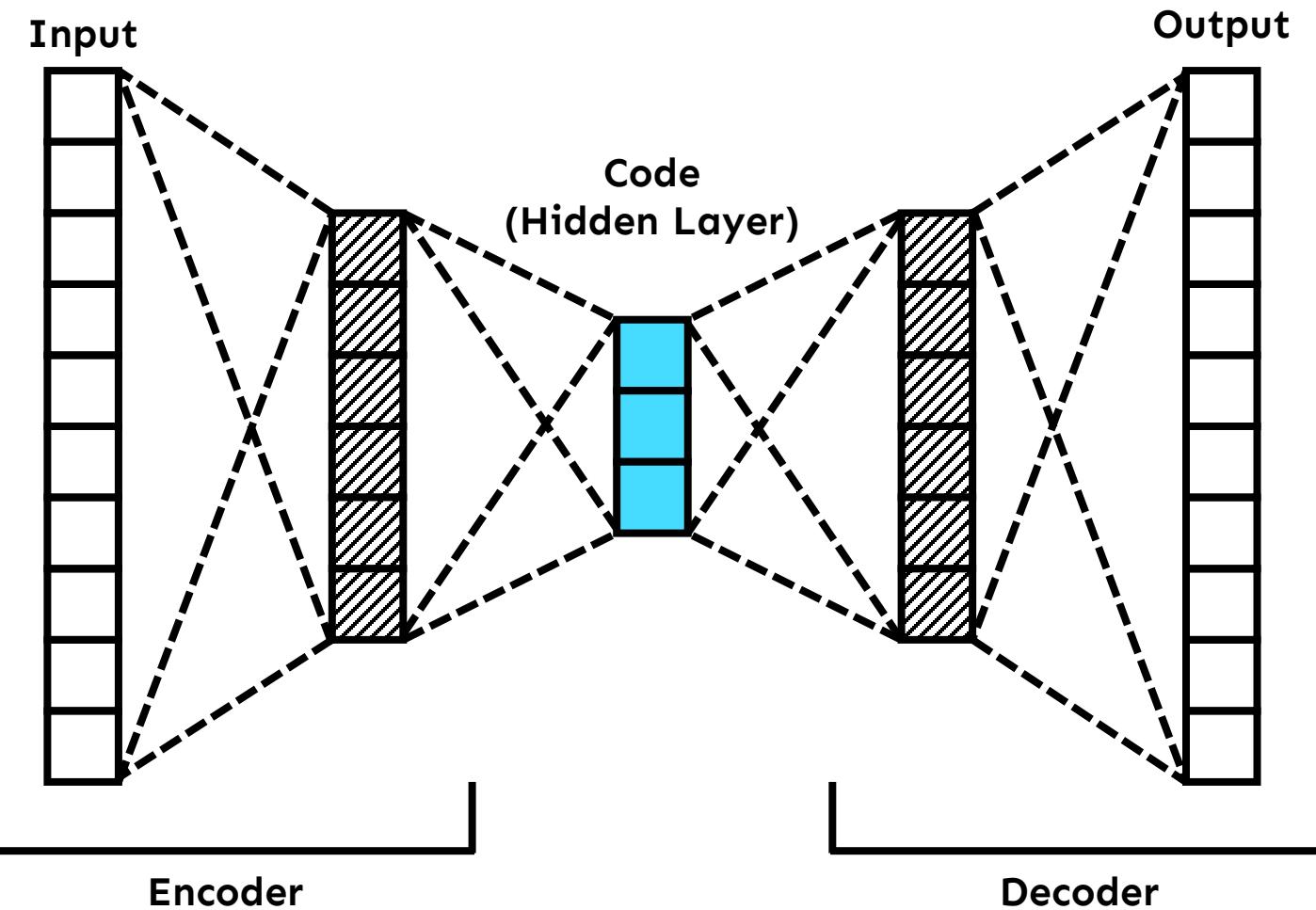
01

# Unsupervised Learning Autoencoders

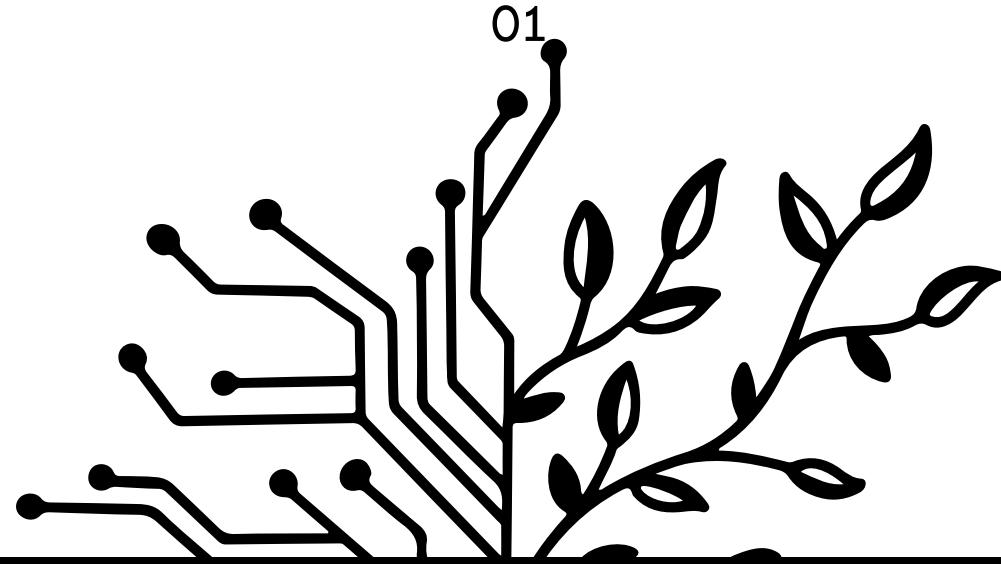


# Traditioneller Autoencoder

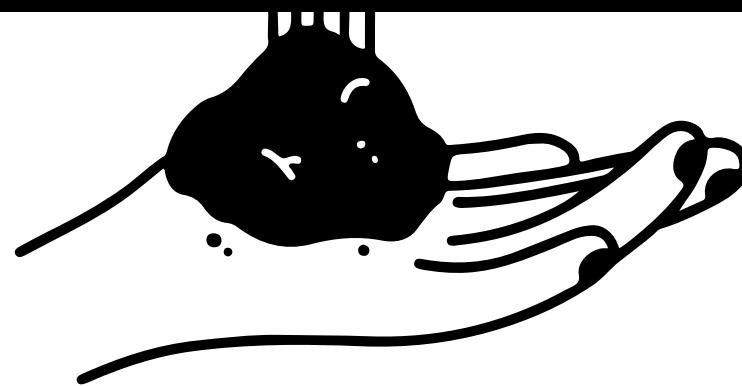
- Eine Form von Komprimierung
- Wird auch in die Kategorie des self-supervised learning eingeordnet
- Ziel: Encoder und Decoder so wählen, dass so wenig Information wie möglich benötigt wird um Input als Output wiederherzustellen
- Wenn das Bottleneck zu klein ist, geht zu viel Information verloren.
- Ist es zu groß, ist der Nutzen gering



Quelle: <https://towardsdatascience.com/generating-images-with-autoencoders-77fd3a8dd368>

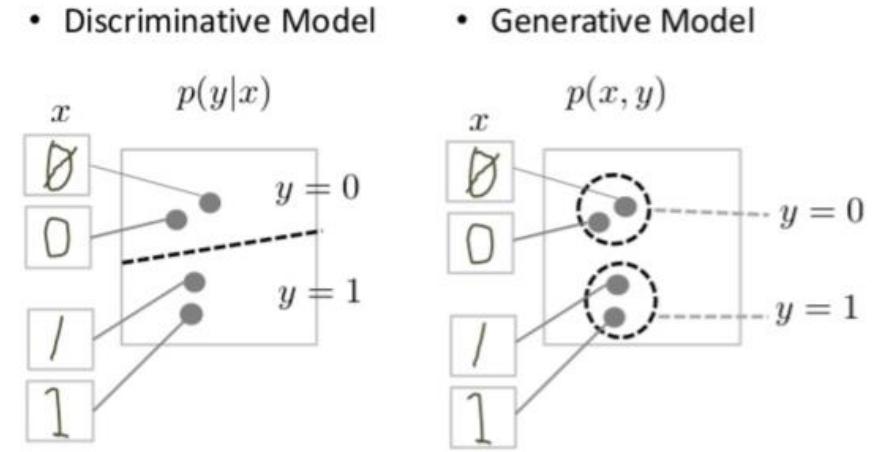


# Unsupervised Learning GANs



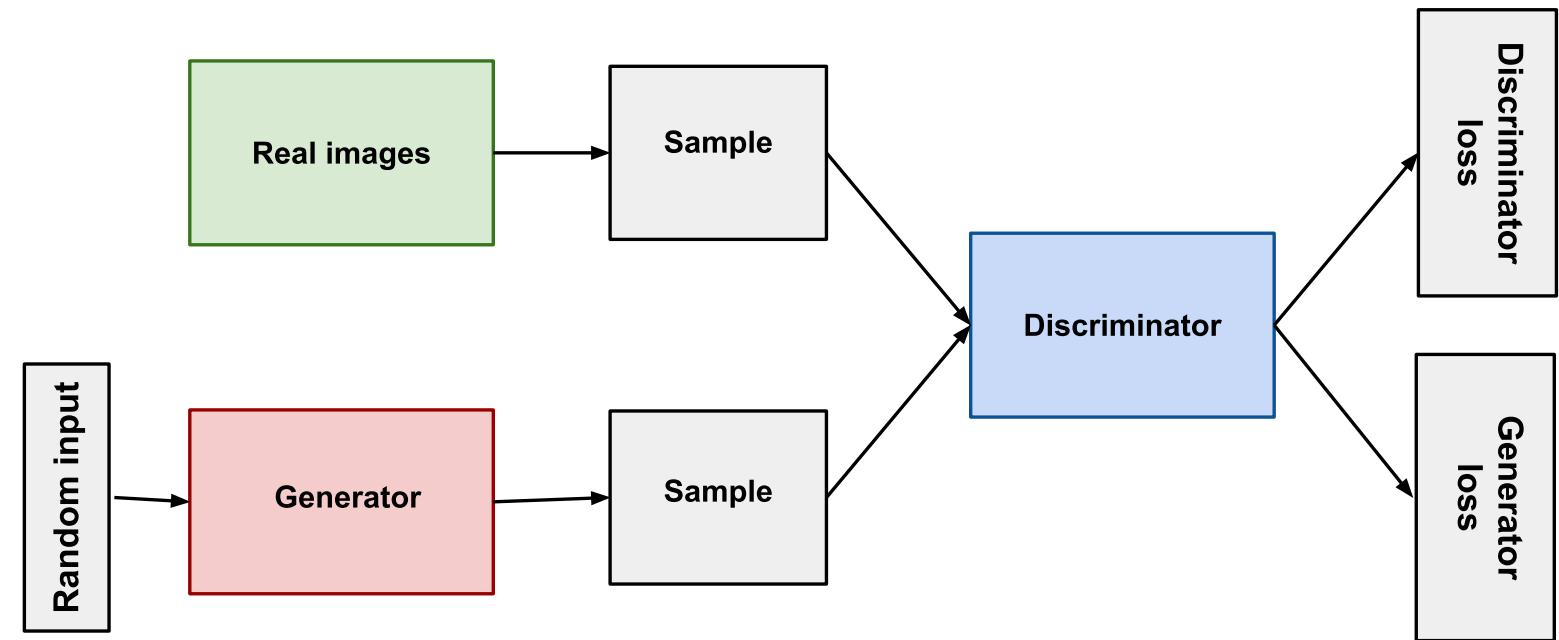
# Generative Modelle

- **Generative Modelle** generieren neue Datenpunkte
- **Diskriminative Modelle** diskriminieren zwischen unterschiedlichen Arten von Datenpunkten
- Formal bedeutet das:
  - Generative Modelle finden eine Representation von  $p(X, Y)$  bzw.  $p(X)$
  - Diskriminative Modelle modellieren die bedingte Wahrscheinlichkeit  $p(Y | X)$
- Zwei Klassen generativer Modelle:
  - **Explizite Modelle** spezifizieren die Datenerzeugendenfunktion  $p(X, Y)$
  - **Implizite Modelle** definieren einen stochastischen Prozess mit dem sich Samples erzeugen lassen



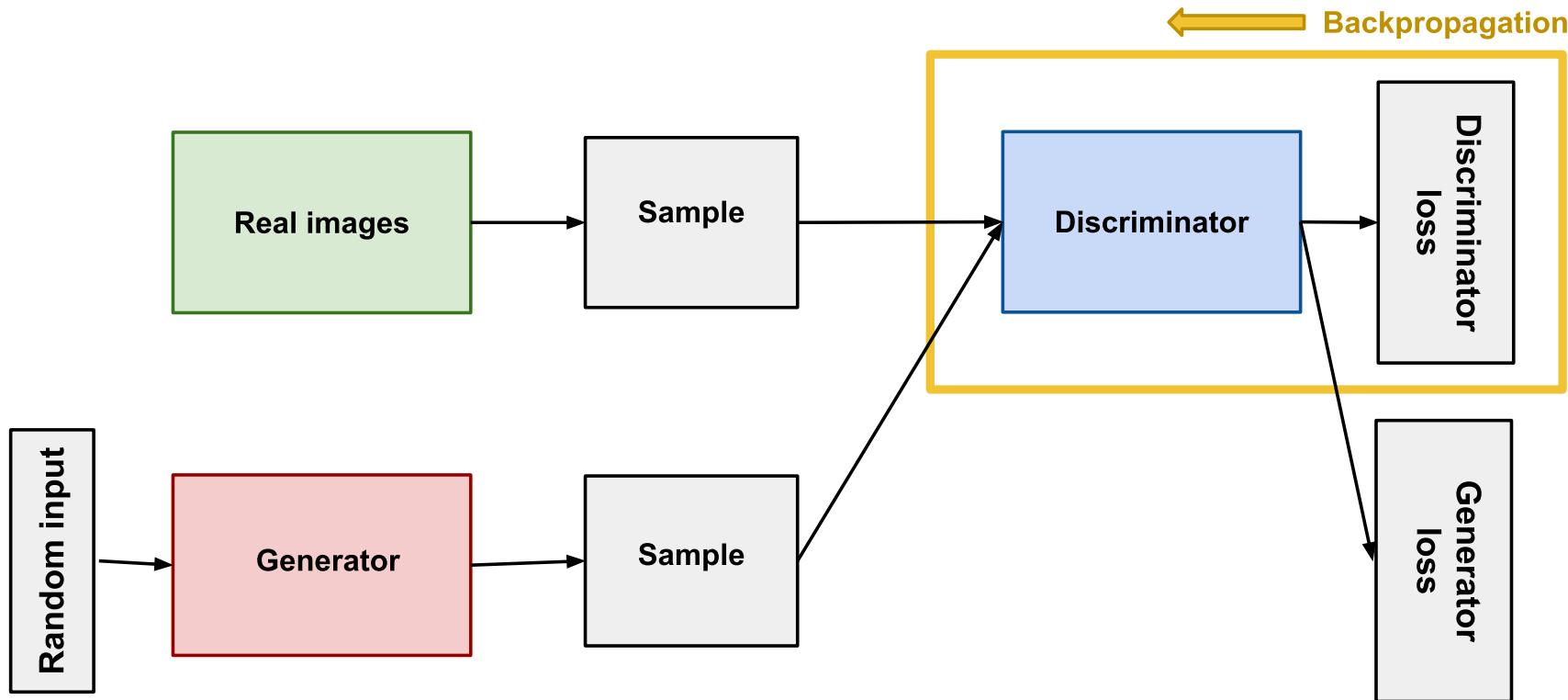
# Generative Adversarial Networks (GANs)

- Wie lassen sich generative Modelle trainieren?
- Ian Goodfellow et al. (2014): "Generative Adversarial Networks"
- GANs sind implizite generative Modelle
- Es werden zwei Netzwerke trainiert:
  - Generator: erzeugt Daten
  - Diskriminator: versucht echte von fake Daten zu unterscheiden
- **Training:**
  - Diskriminator und Generator werden abwechselnd epochenweise trainiert



# Training von GANs: Diskriminator Loss

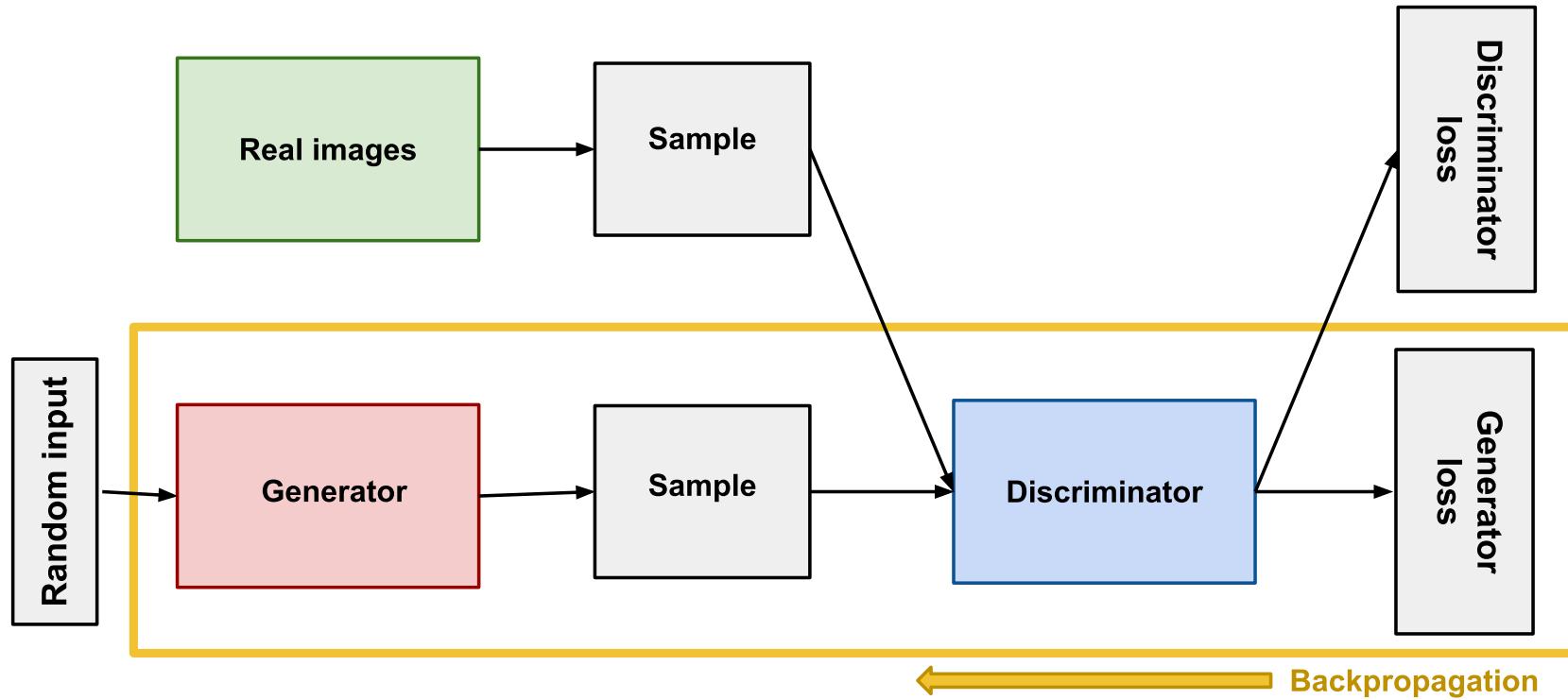
- Diskriminator ist ein Klassifizierer und versucht  $\mathbb{E}_x[\log(D(x))] + \mathbb{E}_z[\log(1 - D(G(z)))]$  zu maximieren (vgl. Cross entropy)



<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# Training von GANs: Generator Loss

- Generator versucht den Diskriminator auszutricksen und minimiert daher  $\mathbb{E}_z[\log(1 - D(G(z)))]$



<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# Training von GANs: Minimax Loss

- Insgesamt ergibt sich so, dass der Generator folgenden Loss minimiert, während der Diskriminatord den Loss maximiert:

```
- Insgesamt ergibt sich so, dass der Generator folgenden Loss minimiert, während der Diskriminatord den Loss maximiert.  
- D(x) ist die Output Wahrscheinlichkeit des Diskriminators, dass das reale Beispiel x echt ist.  
- G(z) ist der Output des Generators gegeben dem Noise z.  
- G(z) ist der Output des Generators zum fake Beispiel, erzeugt vom Generator  
- Ex ist der Erwartungswert bzgl. der Noise Verteilung, welche als Input des Generators dient.
```

- $D(x)$  ist die Output Wahrscheinlichkeit des Diskriminators, dass das reale Beispiel  $x$  echt ist
- $E_x$  ist der Erwartungswert über alle Trainingsbeispiele
- $G(z)$  ist der Output des Generators gegeben dem Noise  $z$
- $D(G(z))$  ist die Output Wahrscheinlichkeit des Diskriminators zum fake Beispiel, erzeugt vom Generator
- $E_z$  ist der Erwartungswert bzgl. der Noise Verteilung, welche als Input des Generators dient

<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# This X does not exist

Link: <https://thisxdoesnotexist.com/>

## This Baseball Player Does Not Exist

Using generative adversarial networks (GAN), we can learn how to create realistic-looking fake versions of almost anything, as shown by this collection of sites that have sprung up in the past month. Learn [how it works](#).



### This Person Does Not Exist

The site that started it all, with the name that says it all. Created using a style-based generative adversarial network (StyleGAN), this website had the tech community buzzing with excitement and intrigue and inspired many more sites.



### This Cat Does Not Exist

These purr-fect GAN-made cats will freshen your feeline-gs and make you wish you could reach through your screen and cuddle them. Once in a while the cats have visual deformities due to imperfections in the model – beware, they can cause nightmares.



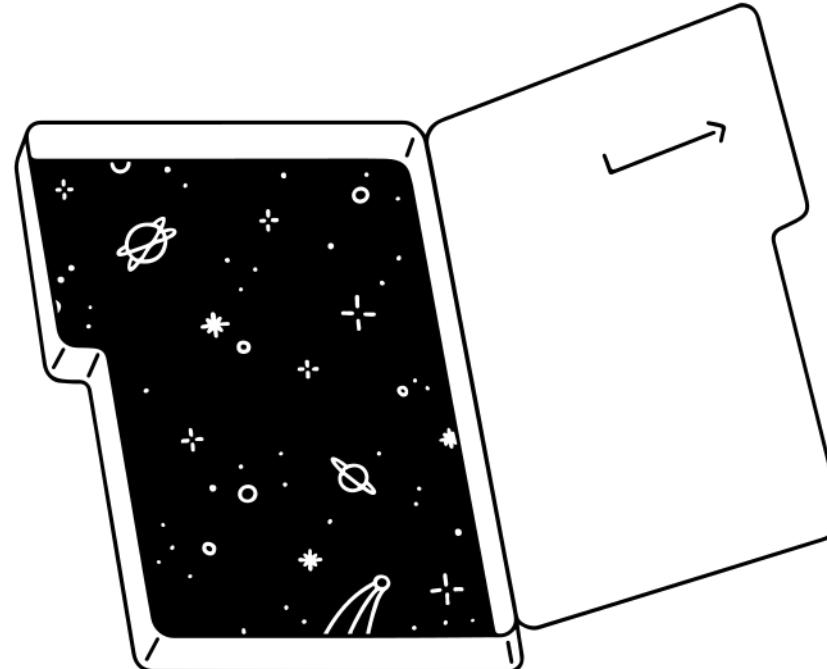
### This Rental Does Not Exist

Why bother trying to look for the perfect home when you can create one instead? Just find a listing you like, buy some land, build it, and then enjoy the rest of your life.

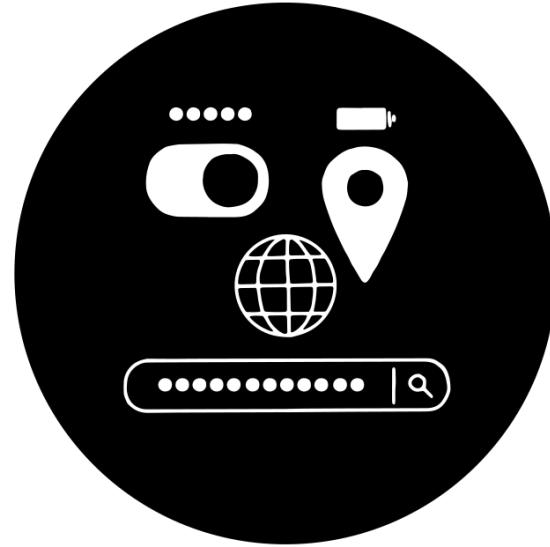
# GANs

- Vorschau: **05\_TimeSeries\_LSTMS**
- **Übung\_06\_TimeSeries\_LSTMS**

## PRAXIS



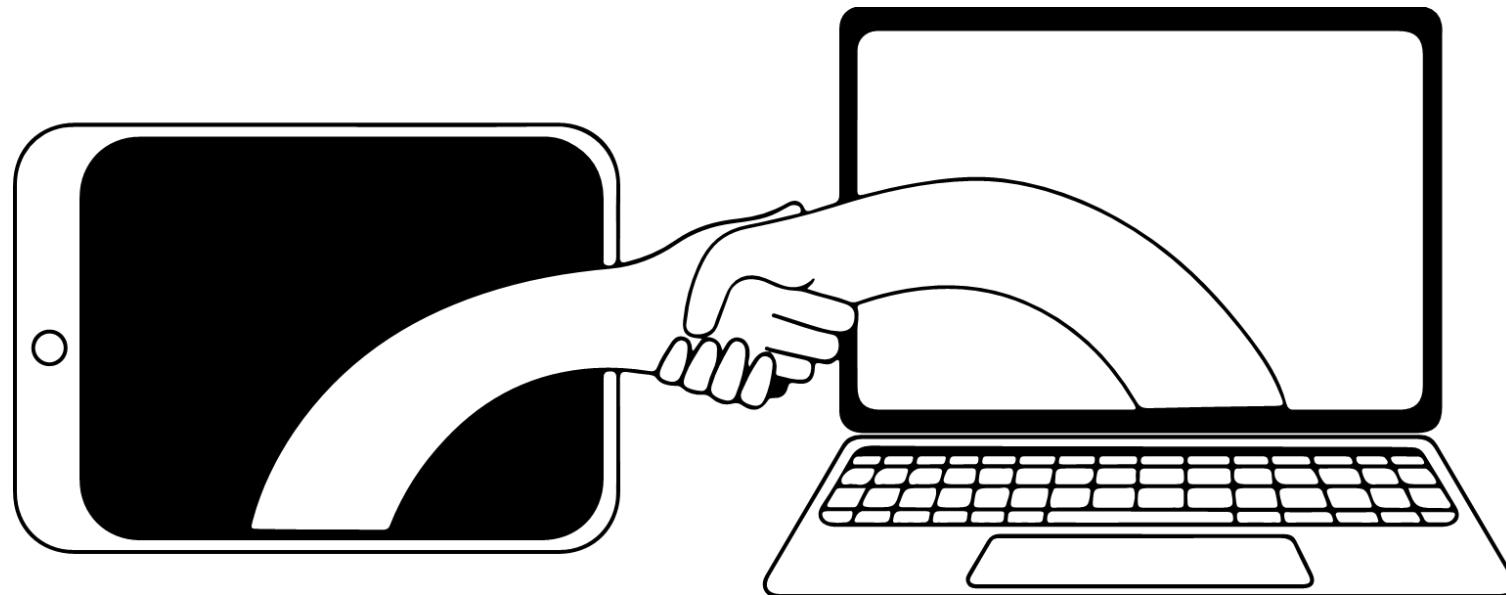
# Ehrliches Feedback



[Link zur Bewertung](#)

# Fragen und Feedbackrunde

- Haben Sie Fragen zu den behandelten Themen?
- Welche Themen haben Sie vermisst?
- Sonstiges Feedback zum Seminar/Verbesserungsvorschläge für die Zukunft?



**LET'S  
STAY IN  
TOUCH**

EXXETA GMBH  
Albert-Nestler-Straße 19  
D-76131 Karlsruhe  
[Tobias.krebs@exxeta.com](mailto:Tobias.krebs@exxeta.com)

**exeta**