

Datenanalyse mit KI

Dr.-Ing. Grigory Devadze

Was ist Künstliche Intelligenz?

- **Definition und grundlegende Konzepte:**

Künstliche Intelligenz (KI) bezeichnet Systeme oder Maschinen, die Aufgaben ausführen, die normalerweise menschliche Intelligenz erfordern – z.B. Lernen, Problemlösen, Wahrnehmung und Sprachverstehen.

- **Unterschied zwischen KI, Machine Learning und Deep Learning:**

- **KI (Künstliche Intelligenz):** Überbegriff für Maschinen, die „intelligent“ handeln.
- **Machine Learning (ML):** Teilbereich der KI, bei dem Systeme aus Daten lernen, um Aufgaben ohne explizite Programmierung zu lösen.
- **Deep Learning:** Spezielles Teilgebiet des ML, das mit künstlichen neuronalen Netzen arbeitet.

Kurzer Überblick: Geschichte der KI

1. Von den Anfängen bis zu Expertensystemen

- **1956:** Begriff „Künstliche Intelligenz“ erstmals verwendet
→ Dartmouth Conference (John McCarthy u.a.), Ziel: Maschinen zum intelligenten Handeln befähigen
- **1950er–1970er:** Erste KI-Programme
→ Schachprogramme, Beweistheorie, viel Optimismus, doch erste Rückschläge („KI-Winter“)
- **1980er:** Erste Expertensysteme
→ Systeme wie MYCIN unterstützen z.B. medizinische Diagnosen, KI-Einsatz in der Industrie;
Beschränkung: Wissen muss manuell eingegeben werden, geringe Flexibilität

2. Moderne Durchbrüche: Machine Learning & LLMs

- **2000er:** Durchbrüche mit Machine Learning dank leistungsfähiger Hardware
→ Mehr Daten, bessere Algorithmen, „Lernen aus Erfahrung“
- **2010er–Heute:** KI im Alltag
→ Sprachassistenten (Siri, Alexa), Bild- und Gesichtsanalyse, Produktempfehlungen, autonomes Fahren
→ Zentrale Rolle von Deep Learning (tiefe neuronale Netze)
- **LLMs (Large Language Models):**
→ Neuste KI-Entwicklung auf Basis von Deep Learning (z.B. GPT, BERT)
→ Können natürliche Sprache verstehen & generieren
→ Anwendungen: Chatbots, Text-Assistenz, automatische Übersetzungen
- KI und LLMs verändern Alltag, Wirtschaft und Forschung grundlegend und rasant

Wie lernt eine KI?

- **Training mit Daten:** Algorithmen erkennen Muster in großen Datensätzen.
- **Überwachtes Lernen:** Training mit Daten, bei denen die Lösung bekannt ist.
- **Unüberwachtes Lernen:** Finden von Mustern ohne vorgegebene Lösungen
- **Bestärkendes Lernen:** Lernen durch Belohnung und Bestrafung

Praxisbeispiele für KI

- Sprachübersetzung (z.B. Google Translate)
- Bilderkennung (z.B. Gesichtserkennung bei Smartphones)
- Chatbots & virtuelle Assistenten
- Medizinische Diagnosesysteme
- Autonomes Fahren, Robotik

Potenziale der KI in der Datenanalyse

- **Automatisierte Mustererkennung:**

KI-Modelle entdecken selbstständig verborgene Zusammenhänge und Trends in großen Datenmengen, die für Menschen oft nicht erkennbar sind.

- **Schnellere und präzisere Analysen:**

KI beschleunigt die Datenverarbeitung erheblich und liefert häufig genauere Ergebnisse, indem sie Fehlerquellen minimiert und Daten aus verschiedenen Quellen kombiniert.

- **Entscheidungsunterstützung durch Vorhersagen:**
Predictive Analytics mittels KI ermöglicht datenbasierte Entscheidungsfindung (z.B. Prognose von Nachfrage, Ausfallwahrscheinlichkeit, Risikobewertung).
- **Verarbeitung unstrukturierter Daten:**
KI kann auch mit Texten, Bildern oder Audiodaten umgehen – etwa mittels Natural Language Processing (NLP) oder Bilderkennungsalgorithmen.

Überblick über Tools und Technologien

- **Programmiersprachen:**
 - **Python:** Am weitesten verbreitete Sprache für KI und datenbasierte Analysen, mit vielen leistungsfähigen Bibliotheken.
 - **R:** Speziell für statistische Analysen und Visualisierung häufig genutzt.
- **Entwicklungsumgebungen:**
 - **Jupyter Notebooks:** Für interaktive Analysen, Visualisierung und Dokumentation des Workflows.

- **Frameworks & Libraries:**
 - **TensorFlow & PyTorch:** Für Deep Learning und komplexe neuronale Netze.
 - **Scikit-learn:** Für klassische Machine-Learning-Verfahren (Clustering, Klassifikation, Regression etc.).
 - **Pandas & NumPy:** Für Datenaufbereitung und -verarbeitung.
- **Weitere Tools:**
 - **Power BI, Tableau** zur Visualisierung
 - **AutoML-Plattformen** zur Vereinfachung von KI-Workflows
 - **MLOps:** Deployment

Was ist Maschinelles Lernen?


- Ein Teilgebiet der Künstlichen Intelligenz
- Systeme lernen selbstständig aus Daten
- Verbesserung ohne explizite Programmierung

Arten von ML


1 Überwachtes Lernen (Supervised Learning)

- Gelabelte Daten
- Beispiel: Spam-Erkennung 

2 Unüberwachtes Lernen (Unsupervised Learning)

- Struktur in unlabeled Daten erkennen
- Beispiel: Kundensegmentierung 

3 Bestärkendes Lernen (Reinforcement Learning)

- Lernen durch Belohnung & Bestrafung
- Beispiel: AlphaGo 

Beispiel: Spam-Erkennung mit Maschinellem Lernen

Wie erkennt ein ML-Algorithmus Spam?

- Ein Machine-Learning-Modell kann helfen, unerwünschte E-Mails (Spam) zu klassifizieren, indem es Muster in den Nachrichten analysiert.

1 Datensammlung und Vorbereitung

- Datenquellen: E-Mails mit Label „Spam“ oder „Nicht-Spam“
- Merkmale (Features):
 - Bestimmte Wörter („Gewonnen“, „Gratis“, „Schnell Geld“)
 - Anzahl der Links, Verwendung von Großbuchstaben
 - Absender-Adresse, Länge und Struktur der E-Mail

2 Training eines Modells

Supervised Learning:

- Das System wird mit vielen gelabelten E-Mails trainiert.
- Ziel: Regeln lernen, die Spam von legitimen E-Mails unterscheiden.

Beispielhafte Vorgehensweise:

- Wortanalyse: Welche Wörter sind typischerweise in Spam-Nachrichten?
- Wahrscheinlichkeitsberechnung: Wie oft tauchen bestimmte Wörter in Spam-/Nicht-Spam-E-Mails auf?
- Gewichtung der Merkmale: Welche Eigenschaften weisen am stärksten auf Spam hin?

3 Anwendung: Klassifikation neuer E-Mails

- Neue E-Mail geht ein → Modell analysiert Inhalte → Entscheidung: Spam oder Nicht-Spam?
- Falls viele „Spam-indikative“ Wörter enthalten sind → Wahrscheinlichkeit für Spam hoch
- Falls seriöse Muster erkannt werden → E-Mail bleibt im Posteingang

4 Optimierung des Modells

- Vermeidung von Fehlern:
- False Positives → Legitime Mails fälschlicherweise als Spam markiert
- False Negatives Spam wird nicht erkannt und landet im Posteingang
- Verbesserung durch kontinuierliches Lernen & Feedback: Nutzer markieren Mails als „Kein Spam“ oder „Als Spam melden“.

Häufig verwendete ML-Algorithmen

- ◆ Lineare Regression
- ◆ Entscheidungsbäume & Random Forests
- ◆ Support Vector Machines (SVM)
- ◆ Künstliche Neuronale Netze (Deep Learning)
- ◆ k-Means Clustering
- ◆ Hauptkomponentenanalyse (PCA)

Warum ist maschinelles Lernen besser als fest definierte Regeln?

1 Flexibilität & Anpassungsfähigkeit

◆ Regelbasierte Systeme:

- Müssen ständig von Menschen aktualisiert werden, wenn sich Spam-Taktiken ändern.
- Beispiel: Früher war „Gratis“ ein typisches Spam-Wort – heute tarnen Spammer es mit „Gr@tis“ oder „G.r.a.t.i.s“.

◆ ML-Systeme (wenn die gut sind):

- Lernen automatisch aus neuen Spam-Mustern.
- Erkennen auch veränderte Wortschreibweisen oder neue Tricks von Spammern.

2 Erkennung von versteckten Mustern

Regelbasierte Ansätze:

- Prüfen nur offensichtliche Kriterien (z. B. „Hat die E-Mail das Wort ‚Casino‘?“).
- Können komplexe Zusammenhänge nicht erkennen.

ML-Ansatz:

- Findet schwer erkennbare Muster durch statistische Analysen.
- Kann Spam sogar dann identifizieren, wenn kein typisches Spam-Wort vorhanden ist - etwa durch Satzstruktur, Absenderverhalten oder Ähnlichkeiten zu anderen Spam-Mails.

3 Höhere Genauigkeit & weniger Fehler

Regeln machen oft Fehler („False Positives“):

- Beispiel: Eine legitime Hotel-Buchungsbestätigung enthält das Wort „Gratis WLAN“ – könnte fälschlicherweise als Spam aussortiert werden.
- Ein Modell lernt, den Kontext zu analysieren: „Ist es echtes Hotelmarketing oder Fake-Gewinnspiel?“

4 Skalierbarkeit & Automatisierung

Regelbasierte Systeme:

- Erfordern menschliche Experten, die ständig neue Regeln schreiben und testen müssen.
- Werden bei hohen Datenmengen ineffizient.

Maschinelles Lernen:

- Wächst mit den Daten – je größer die Spam-Menge, desto besser wird das Modell.
- Erfordert weniger Wartung, nachdem es trainiert wurde.

5 Schutz vor „intelligenten“ Spammern

Spammer entwickeln ständig neue Tricks, zum Beispiel:

- Bilder statt Text (Spam-Infos als Bild eingebettet)
- Obfuscation (Verschleierung) („C.a.s.i.n.o“ statt „Casino“)
- Schädliche Links mit Kurz-URLs (z. B. bit.ly/xyz)

ML-Filter sind lernfähig:

- Finden Spam selbst dann, wenn er sich äußerlich stark verändert hat.
- Können künstliche Intelligenz gegen künstliche Intelligenz einsetzen.

Herausforderungen

Vergangenheit kann die Zukunft nicht immer vorhersagen

- Die Verwendung historischer Daten zur Vorhersage der Zukunft setzt voraus, dass es bestimmte **konstante Bedingungen** oder **Steady-State-Bedingungen** in einem komplexen System gibt.
- Dies ist **fast immer falsch**, wenn das System Menschen involviert.
- **Beispiel:**
 - Finanzkrisen können nicht allein durch historische Daten vorhergesagt werden, da sich Marktbedingungen und menschliches Verhalten ständig ändern.

Das Problem der unbekannten Merkmale

- Bei der Datenakquise definiert der Benutzer zunächst die Variablen, für die Daten erhoben werden.
- Es besteht jedoch immer die Möglichkeit, dass **kritische Variablen** nicht berücksichtigt oder sogar definiert wurden.
- **Beispiel:**
 - In der Medizin können unbekannte genetische Faktoren oder Umweltbedingungen den Ausgang einer Behandlung beeinflussen, obwohl sie nicht in den ursprünglichen Daten enthalten waren.

Selbstzerstörung von Algorithmen

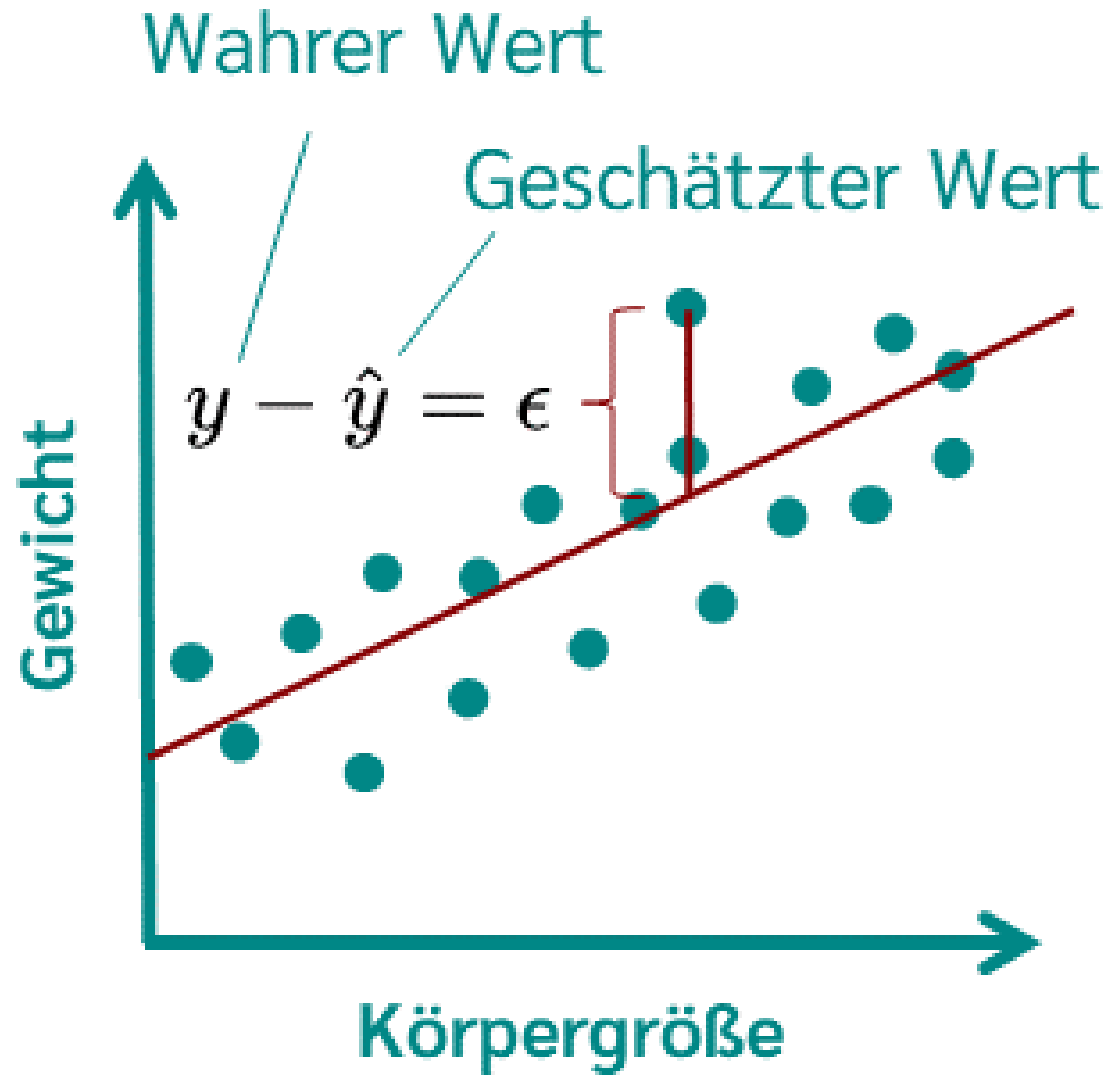
- Wenn ein Algorithmus zum akzeptierten Standard wird, kann er von Personen ausgenutzt werden, die den Algorithmus verstehen und ein Interesse daran haben, das Ergebnis zu manipulieren.
- Dies führt zu einer **Selbstzerstörung** des Algorithmus, da er nicht mehr zuverlässig ist.
- Beispiel:
 - CDO-Ratings vor der Finanzkrise 2008:
 - CDO-Händler manipulierten die Eingabevariablen, um AAA-Ratings für ihre Produkte zu erhalten, was zur Finanzkrise beitrug.

Lineare Regression

Die lineare Regression ist ein **grundlegender** (statistischer) Algorithmus, der den Zusammenhang zwischen einer abhängigen Variable (Zielvariable) und einer oder mehreren unabhängigen Variablen (Merkmale/Features) modelliert.

Wichtig für Predictive Analytics:

- Einfachheit und Interpretierbarkeit
- Grundlage für komplexere Modelle
- Schnelles Modell
- Breite Anwendbarkeit



Fehler epsilon

$$y = b \cdot x + a + \epsilon$$

Was ist die Idee hinter der linearen Regression?

Sie versucht, eine gerade Linie durch die Datenpunkte zu finden, die den besten Zusammenhang beschreibt.

$$y = m \cdot x + b$$

Bedeutung der Parameter:

- ✓ x = Eingangsvariable (z. B. Werbebudget)
- ✓ y = vorhergesagte Zielvariable (z. B. Umsatz)
- ✓ m = Steigung der Linie (zeigt den Einfluss von x auf y)
- ✓ b = Achsenabschnitt (Wert von y , wenn $x = 0$)

Mehrfache lineare Regression (multiple regression):

Falls mehrere unabhängige Variablen existieren:

$$y = b + m_1x_1 + m_2x_2 + \dots + m_nx_n$$

- ◆ Beispiel: Umsatzvorhersage basierend auf Werbebudget, Anzahl der Geschäfte und Marktanalysen.

Problemformulierung: Modellgleichung

Allgemeine Gleichung der einfachen (univariaten) linearen Regression:

$$y = m \cdot x + b$$

Das Ziel ist, die **besten** Werte für m und b zu finden!

Die gängigste Metrik ist der Mittlere Quadratische Fehler (Mean Squared Error, MSE):

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

- ◆ y_i = tatsächlicher Wert der i -ten Datenprobe
- ◆ \hat{y}_i = vorhergesagter Wert durch das Modell
- ◆ n = Anzahl der Datenpunkte

Ziel: Den Fehler so klein wie möglich machen → die Linie passt sich besser an die echten Daten an!

Matrixmultiplikation

Definition:

- Die Multiplikation zweier Matrizen \mathbf{A} (Größe $m \times n$) und \mathbf{B} (Größe $n \times p$) ergibt eine neue Matrix \mathbf{C} (Größe $m \times p$).
- Das Element C_{ij} wird berechnet als:

$$C_{ij} = \sum_{k=1}^n A_{ik} \cdot B_{kj}$$

Beispiel: Matrixmultiplikation

Gegeben:

- Matrix A (2x3):

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

- Matrix B (3x2):

$$B = \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix}$$

Schritt-für-Schritt-Berechnung

1. Berechne C_{11} :

$$C_{11} = (1 \cdot 7) + (2 \cdot 9) + (3 \cdot 11) = 7 + 18 + 33 = 58$$

2. Berechne C_{12} :

$$C_{12} = (1 \cdot 8) + (2 \cdot 10) + (3 \cdot 12) = 8 + 20 + 36 = 64$$

3. Berechne C_{21} :

$$C_{21} = (4 \cdot 7) + (5 \cdot 9) + (6 \cdot 11) = 28 + 45 + 66 = 139$$

4. Berechne C_{22} :

$$C_{22} = (4 \cdot 8) + (5 \cdot 10) + (6 \cdot 12) = 32 + 50 + 72 = 154$$

Ergebnis

Die resultierende Matrix C (2x2) ist:

$$C = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix}$$

Matrixinversion

Definition:

- Die Inverse einer quadratischen Matrix A (Größe $n \times n$) ist eine Matrix A^{-1} , sodass:

$$A \cdot A^{-1} = A^{-1} \cdot A = I$$

wobei I die Identitätsmatrix ist:

$$I = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ . & . & 1 & . \\ . & . & . & . \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Bedingung:

- Eine Matrix ist nur dann invertierbar, wenn ihre Determinante ungleich null ist.

Beispiel: Matrixinversion

Gegeben:

- Matrix A (2x2):

$$A = \begin{bmatrix} 4 & 7 \\ 2 & 6 \end{bmatrix}$$

Spezialfall für Adjunkte:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, adj(A) = \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Schritt-für-Schritt-Berechnung

1. Berechne die Determinante:

$$\det(A) = (4 \cdot 6) - (7 \cdot 2) = 24 - 14 = 10$$

2. Berechne die Adjunkte:

$$\text{adj}(A) = \begin{bmatrix} 6 & -7 \\ -2 & 4 \end{bmatrix}$$

3. Berechne die Inverse:

$$A^{-1} = \frac{1}{\det(A)} \cdot \text{adj}(A) = \frac{1}{10} \begin{bmatrix} 6 & -7 \\ -2 & 4 \end{bmatrix} = \begin{bmatrix} 0.6 & -0.7 \\ -0.2 & 0.4 \end{bmatrix}$$

Zusammenfassung

1. Matrixmultiplikation:

- Multipliziere Zeilen der ersten Matrix mit Spalten der zweiten Matrix.

$$A \cdot B = C.$$

2. Matrixinversion:

- Berechne die Inverse einer quadratischen Matrix, falls sie existiert.

$$A^{-1} \cdot A = I.$$

Direkte Lösung der Linearen Regression mittels Normalengleichung

Da die MSE (Mean Squared Error)-Fehlermetrik eine konvexe Funktion ist, kann man ihre Ableitung direkt setzen und nach den optimalen Parametern m und b auflösen.

Für die mehrdimensionale lineare Regression lautet die optimale Lösung:

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

- ◆ \mathbf{X} = Design-Matrix (enthält alle Eingangsvariablen)
- ◆ \mathbf{y} = Zielvariablen-Vektor (bekannte Werte)
- ◆ \mathbf{w} = gesuchter Parametervektor (enthält Gewichte m und b)

Warum ist das möglich?

- Die Mean-Squared-Error-Kostenfunktion ist eine quadratische Funktion der Parameter, die eine einfache Ableitung erlaubt.
- Weil es eine sog. konvexe Optimierungsaufgabe ist, existiert genau eine eindeutige Lösung, die direkt berechnet werden kann.

Beispiel: Einfache Lineare Regression mit analytischer Lösung

Für eine einfache Regression mit $w_1 = m$ und $w_0 = b$ können wir direkt die sog. Normalengleichung verwenden:

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Hiermit erhalten wir die optimalen Werte für m (Steigung) und b (Achsenabschnitt)

Vorteile der analytischen Lösung

- ✓ Exakte Lösung – keine Approximation nötig
- ✓ Kein Hyperparameter (z. B. Lernrate α) notwendig
- ✓ Schnell berechenbar für kleine bis mittlere Datensätze

Nachteile der analytischen Lösung

- ✗ Rechenaufwändig für große Datensätze

Die Matrix-Inversion $(\mathbf{X}^T \mathbf{X})^{-1}$ hat eine Komplexität von $O(n^3)$ → Sehr langsam für Millionen von Datenpunkten.

Alternative: Gradient Descent

Kostenfunktion (Fehlermetrik: Mean Squared Error, MSE)

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

Hierbei gilt:

y_i = Tatsächlicher Wert

\hat{y}_i = Vorhergesagter Wert anhand der Modellgleichung $\hat{y}_i = mx_i + b$

n = Anzahl der Datenpunkte

Partielle Ableitungen der MSE-Kostenfunktion

Gradientenabstieg benötigt die partiellen Ableitungen des Fehlers nach m (Steigung) und b (Y-Achsenabschnitt)

Partielle Ableitung nach m

$$\frac{\partial}{\partial m} MSE = -\frac{2}{n} \sum_{i=1}^n x_i (y_i - \hat{y}_i)$$

Partielle Ableitung nach b

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

☞ Diese Ableitungen bestimmen, wie sich m und b in Richtung eines besseren Modells anpassen müssen.

Aktualisierungsregel für den Gradientenabstieg

$$\begin{aligned}m &:= m - \alpha \cdot \frac{\partial}{\partial m} MSE \\b &:= b - \alpha \cdot \frac{\partial}{\partial b} MSE\end{aligned}$$

α = Lernrate (Hyperparameter, der bestimmt, wie große Schritte in Richtung der Optimierung gemacht werden)

Der Gradientenabstieg nutzt die Ableitung der MSE-Funktion, um m und b schrittweise anzupassen. Das wiederholt sich so lange, bis die Änderungen minimal sind (Konvergenz).

Grundprinzip hinter Gradient Descent (GD) und Stochastic Gradient Descent (SGD)

Methode	Berechnet Gradienten auf...	Vorteile	Nachteile
Batch Gradient Descent (BGD)	Allen Datenpunkten	Sehr stabile und genaue Konvergenz	Rechenintensiv bei großen Datenmengen
Stochastic Gradient Descent (SGD)	Nur einem zufälligen Datenpunkt pro Schritt	Sehr effizient bei großen Datenmengen	Kann stark schwanken (hohe Varianz)
Mini-Batch Gradient Descent (MBGD)	Kleine Gruppe von Beispielen (z. B. 32/64 Punkte)	Balance zwischen GD und SGD, stabil & effizient	Erfordert sorgfältige Wahl der Batch- Größe



Evaluationsmetriken für Regressionsmodelle

1. Mean Absolute Error (MAE)

- Formel:

$$MAE = \frac{1}{m} \sum_{i=1}^m |y_i - \hat{y}_i|$$

- Beschreibung:
 - Durchschnittlicher absoluter Fehler zwischen den vorhergesagten Werten \hat{y} und den tatsächlichen Werten y .
- Interpretation:
 - Je **niedriger** der MAE, desto genauer ist das Modell.
 - Beispiel: **MAE = 1000** bedeutet, dass die Vorhersagen im Durchschnitt um **1000** abweichen.

2 Mean Squared Error (MSE)

- Formel:

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

- Beschreibung:
 - Berechnet den **durchschnittlichen quadratischen Fehler**.
 - Höhere Fehler werden stärker gewichtet als bei MAE.
- Interpretation:
 - Niedrigere Werte bedeuten genauere **Modellvorhersagen**.
 - Da Fehler quadriert werden, hat MSE einen größeren Einfluss bei größeren Abweichungen.

3. Root Mean Squared Error (RMSE)

- Formel:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{m} \sum_{i=1}^m (y_i - \hat{y}_i)^2}$$

- Beschreibung:
 - RMSE ist die **Wurzel des MSE**, wodurch der Fehler in der gleichen Einheit wie (y) bleibt.
- Interpretation:
 - **Vergleichbarer mit den tatsächlichen Werten** als MSE.
 - Kleinere Werte deuten auf **besseres Modell** hin.

4. Bestimmtheitsmaß (R²-Score)

- Formel:

$$R^2 = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2}$$

wobei \bar{y} das Mittel der tatsächlichen Werte ist.

- Beschreibung:
 - Misst, wie viel der Variation von y durch die Prädiktoren erklärt wird.
 - Werte liegen zwischen **0 und 1** (oder negativ, wenn das Modell schlechter ist als bloßes Mittelwert-Raten).

Interpretation:

- $R^2 = 1 \rightarrow$ Perfektes Modell.
- $R^2 \approx 0 \rightarrow$ Modell erklärt kaum etwas.
- $R^2 < 0 \rightarrow$ Modell ist schlechter als eine naive Schätzung.
- Beispiel: $R^2 = 0.75$ bedeutet, dass **75% der Zielvariablen-Varianz** durch das Modell erklärt wird.

Wichtig

✓ MAE vs. MSE:

- MSE bestraft große Fehler stärker als **MAE** (weil quadriert).

✓ RMSE:

- Gut interpretierbar, weil gleiche Einheit wie die Zielvariable.

✓ R^2 :

- Zeigt die Modellgüte, kann aber täuschen – ein zu hohes R^2 kann auf Overfitting hindeuten!

Immer mehrere Metriken nutzen, um ein Modell korrekt zu bewerten!

Klassifikationsproblem

Die Klassifikation ist eine Problemstellung im überwachten maschinellen Lernen, bei der das Ziel darin besteht, eine Abbildung

$$f : \mathbb{R}^n \rightarrow \{C_1, C_2, \dots, C_k\}$$

zu lernen, die Eingabedaten $\mathbf{x} \in \mathbb{R}^n$ einer von k vordefinierten Klassen C_i zuordnet.

Angenommen, wir haben eine Menge von Trainingsbeispielen:

$$D = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$$

wobei

- $\mathbf{x}_i \in \mathbb{R}^n$ der Feature-Vektor der i -ten Beobachtung ist,
- $y_i \in \{C_1, C_2, \dots, C_k\}$ das Label der i -ten Beobachtung ist.

Das Ziel der Klassifikation ist es, eine hypothetische Funktion f zu erlernen:

$$f(\mathbf{x}) = y$$

wobei f die unbekannte wahre Entscheidungsfunktion (Hypothese) ist, die ein ML-Modell approximiert.

Wahrscheinlichkeitsmodell der Klassifikation

Im klassischen ML-Ansatz versucht man, die bedingte Wahrscheinlichkeitsverteilung $P(y|\mathbf{x})$ zu approximieren:

$$P(y = C_k|\mathbf{x}) = f(\mathbf{x}).$$

Ein Modell gibt dann die Klasse mit der höchsten Wahrscheinlichkeit aus:

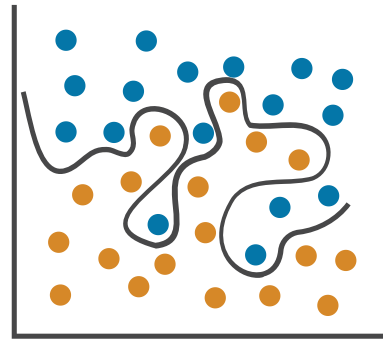
$$\hat{y} = \arg \max_{C_k} P(y = C_k|\mathbf{x}).$$

Beispiele für wahrscheinlichkeitsbasierte Modelle:

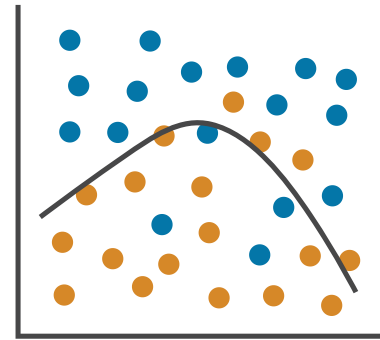
- ✓ Logistische Regression
- ✓ Naive Bayes
- ✓ Neuronale Netze

Classification

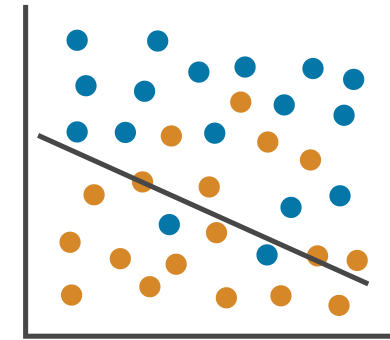
Overfitting



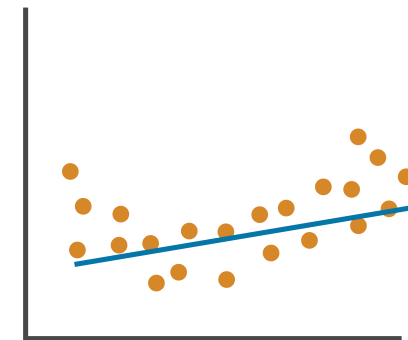
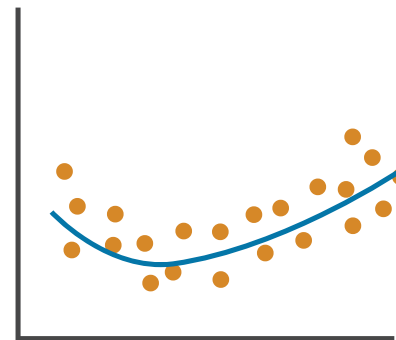
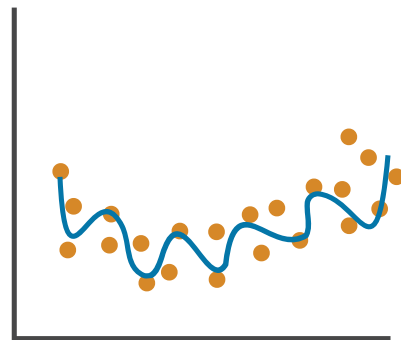
Right Fit



Underfitting



Regression



Bewertungskriterien für Modelle und Systeme

Genauigkeit (Accuracy)

- Definition:
 - Bezieht sich auf die Bewertung, die angewendet wird, um das qualifizierteste Modell zur Identifizierung von Zusammenhängen in einem Datensatz basierend auf Eingabe- oder Trainingsdaten auszuwählen.
- Formel (Fawcett 2006):

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{False Positives} + \text{True Negatives} + \text{False Negatives}}$$

Aktualität (Timeliness)

- Bezieht sich auf die Verfügbarkeit und Zugänglichkeit von Daten für die Entscheidungen.
- Klare, gut organisierte Daten ermöglichen gute Entscheidungen und ein besseres Verständnis zukünftiger Erwartungen.

Kosten (Cost)

- Der Preis für den Dienst.

Skalierbarkeit (Scalability)

- Bezieht sich auf die Messung, ob ein Algorithmus/Framework/Plattform schnelle Veränderungen im Datenwachstum bewältigen kann.

Zuverlässigkeit (Reliability)

- Bezieht sich auf die Wahrscheinlichkeit, dass ein System eine bestimmte Aufgabe in einer spezifischen Umgebung und zu einer bestimmten Zeit ausführen kann.

Leistung (Performance)

- Die Menge an nützlicher Arbeit, die in einer bestimmten Zeit erledigt wird.

Gültigkeit (Validity)

- Überprüft, ob Modelle wie erwartet und gemäß ihren Designzwecken und geschäftlichen Anwendungen funktionieren.

Ressourcennutzung (Resource Utilization)

- Bezieht sich auf den prozentualen Anteil der Zeit, in der eine Komponente genutzt wird, im Vergleich zur Gesamtzeit, in der die Komponente verfügbar ist.

Zeit

- Faktoren, die sich auf die Zeit beziehen, wie Verarbeitungszeit, Gesamtzeit zur Bereitstellung einer Ausgabe und Ausführungszeit.

Energie

- Die Gesamtmenge an Energie, die verbraucht wird, um die angewandten Anfragen auszuführen.

Throughput

- Die maximale Menge an verarbeiteten Daten in einem System zu einem bestimmten Zeitpunkt.

Nachhaltigkeit (Sustainability)

- Die Fähigkeit des Modells, auf einem bestimmten Niveau gehalten zu werden, ohne zukünftige Updates zu benötigen.

Machbarkeit (Feasibility)

- Die Möglichkeit, dass eine Aussage oder ein Modell komfortabel umgesetzt werden kann.

Sicherheit (Security)

- Der Grad, in dem ein System frei von Bedrohungen ist oder irreparable Konsequenzen vermeidet.
- Besonders kritisch in Smart Cities und im Smart Healthcare.

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:

Summation (Input-Summe berechnen)

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

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

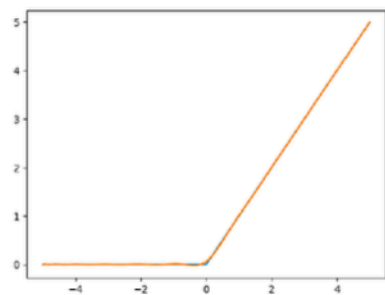
📌 2 Aktivierung (Entscheidung über die Aktivität des Neurons)

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

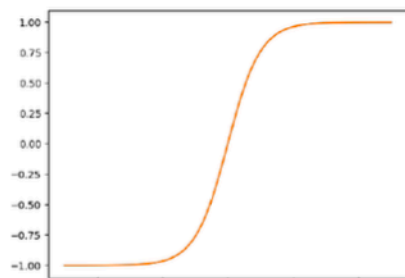
3 Aktivierungsfunktion → Einfache Nicht-Linearität

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

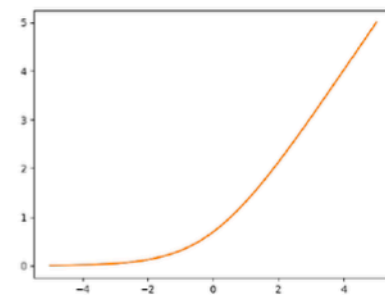
Wichtige Aktivierungsfunktionen:



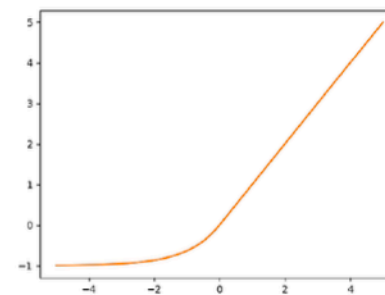
relu



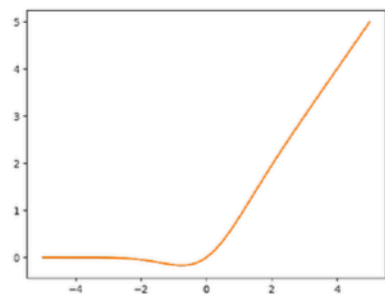
tanh



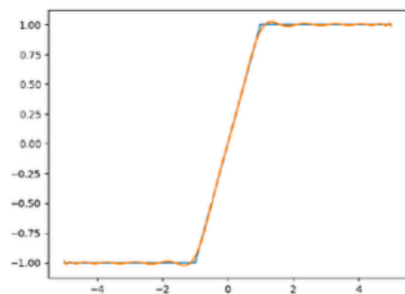
softplus



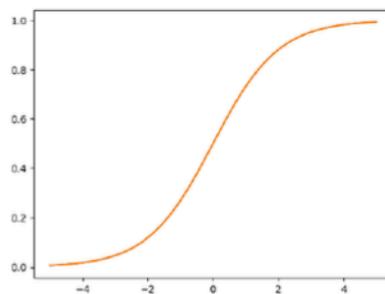
elu



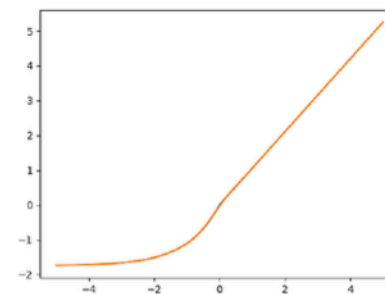
gelu



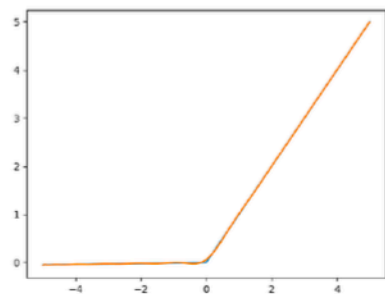
htanh



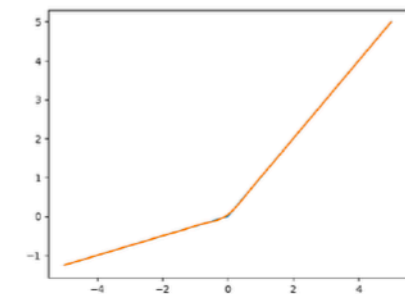
sigmoid



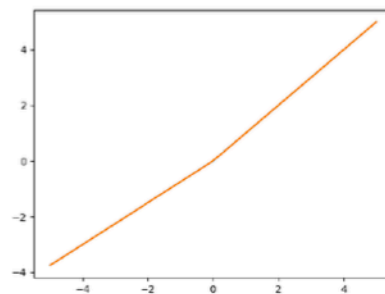
selu



lrelu

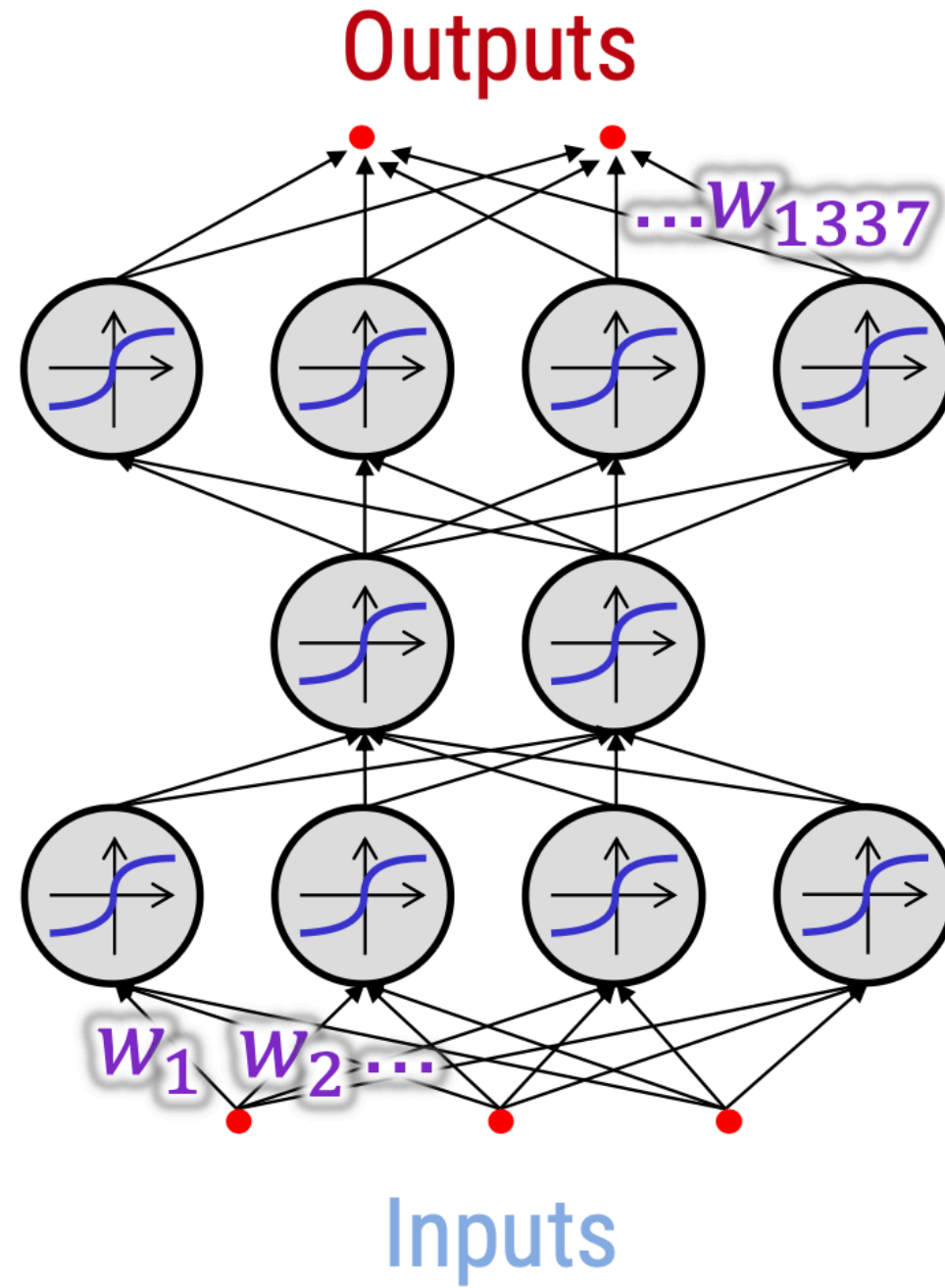


lrelu ($\alpha = 0.25$)



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).

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:

Bildererkennung 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

- 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

- 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

- "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

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

- 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

"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
Input	Die Ausgangsdaten, die das NN verarbeitet.
Output	Das Endergebnis des Netzwerks.
Feature / Hidden Layer	Zwischenergebnisse abstrakte Merkmale
Layer	Berechnungsschritt innerhalb des NN.
Feed Forward	Der normale, sequenzielle Datenfluss eines Netzwerks.
Preactivation	Wert, bevor Aktivierungsfunktion
Activation	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.

◆ 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

Weitere wichtige Kostenfunktionen für DNNs

Binary Cross-Entropy

- Die Binary Cross-Entropy (BCE) ist eine Verlustfunktion für **binäre Klassifikationsprobleme**.
- Sie misst die Differenz zwischen den **vorhergesagten Wahrscheinlichkeiten** und den **tatsächlichen Labels**.

$$\text{BCE} = -\frac{1}{N} \sum_{i=1}^N [y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)]$$

- y_i : Tatsächliches Label (0 oder 1).
- p_i : Vorhergesagte Wahrscheinlichkeit (zwischen 0 und 1).
- N : Anzahl der Samples.

Warum BCE?

- **Eigenschaften:**
 - Straft große Abweichungen zwischen Vorhersage und tatsächlichem Label stark.
 - Gut geeignet für Probleme mit zwei Klassen (z. B. Churn-Vorhersage, Spam-Erkennung).
- **Vorteile:**
 - Einfach zu berechnen.
 - Gut interpretierbar.
 - Funktioniert gut mit Sigmoid-Aktivierungsfunktionen.

Beispiel

- Tatsächliche Labels:

$$y = [1, 0, 1, 1]$$

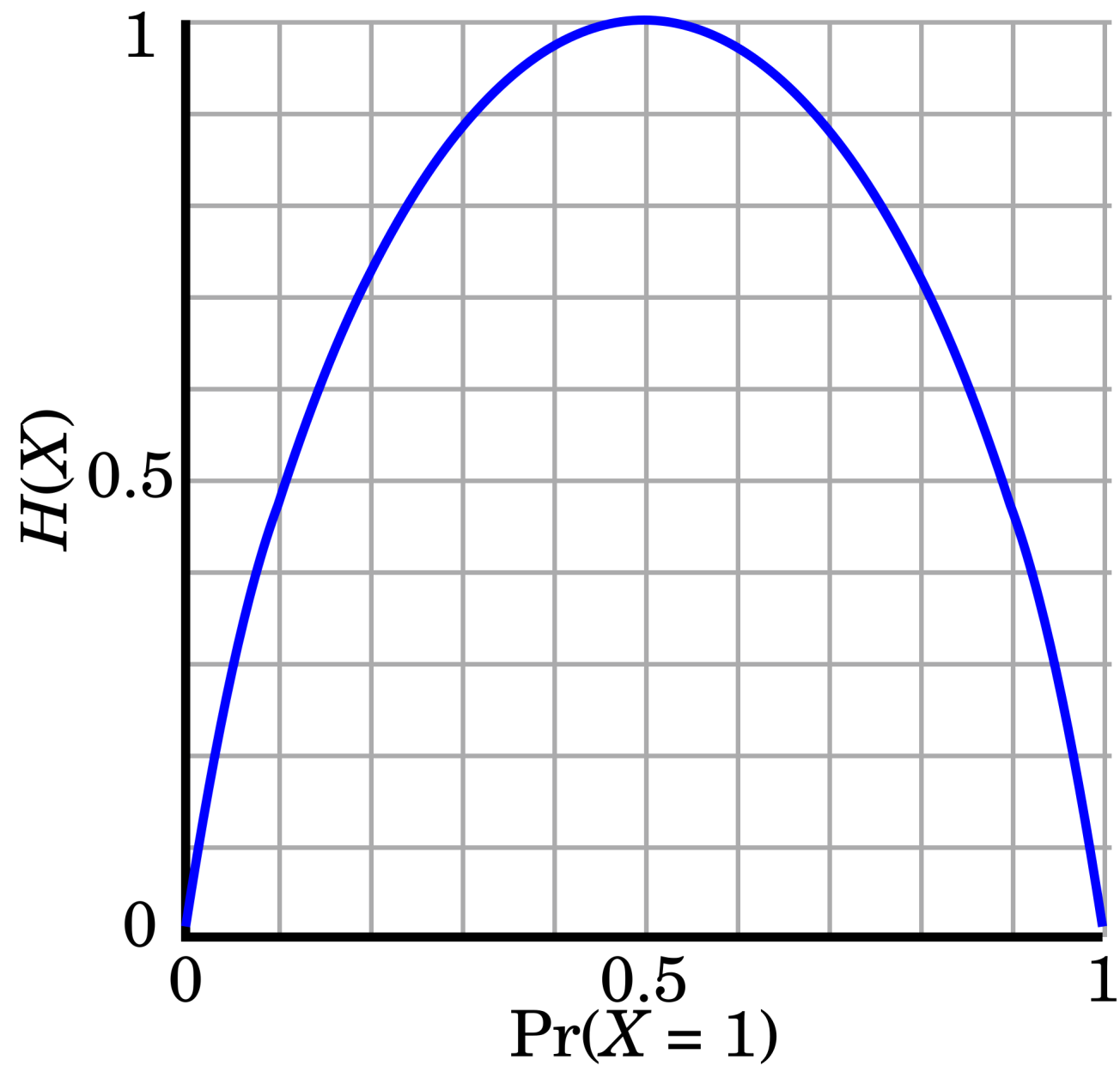
- Vorhergesagte Wahrscheinlichkeiten:

$$p = [0.9, 0.2, 0.8, 0.4]$$

- BCE berechnen:

$$\text{BCE} = -\frac{1}{4} [1 \cdot \log(0.9) + 1 \cdot \log(0.8) + 1 \cdot \log(0.8) + 1 \cdot \log(0.4)] \approx 0.3667.$$

relativ niedrig, d.h. die Vorhersagen sind akzeptabel



Was ist Cross-Entropy?

- Die Cross-Entropy misst die Differenz zwischen zwei Wahrscheinlichkeitsverteilungen.
- In Machine Learning wird sie verwendet, um die Differenz zwischen den **vorhergesagten Wahrscheinlichkeiten** und den **tatsächlichen Labels** zu messen.

$$\text{Cross-Entropy} = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^C y_{ij} \cdot \log(p_{ij})$$

- y_{ij} : Tatsächliches Label (One-Hot-Encoded).
- p_{ij} : Vorhergesagte Wahrscheinlichkeit für Klasse (j).
- N : Anzahl der Samples, C : Anzahl der Klassen.

- **Eigenschaften:**
 - Straft große Abweichungen zwischen Vorhersage und tatsächlichem Label stark.
 - Gut geeignet für **Multi-Klassen-Klassifikation**.
- **Vorteile:**
 - Einfach zu berechnen.
 - Gut interpretierbar.
 - Funktioniert gut mit Softmax-Aktivierungsfunktionen.

Anwendung in PyTorch

```
import torch
import torch.nn as nn

# Beispiel: Vorhergesagte Wahrscheinlichkeiten und tatsächliche Labels
predictions = torch.tensor([[0.9, 0.1, 0.0], [0.2, 0.7, 0.1], [0.1, 0.2, 0.7], [0.8, 0.1, 0.1]])
labels = torch.tensor([[1, 0, 0], [0, 1, 0], [0, 0, 1], [1, 0, 0]])

# Cross-Entropy Loss
criterion = nn.CrossEntropyLoss()
loss = criterion(predictions, labels)
print(f"Cross-Entropy Loss: {loss.item()}")
```

Recurrent Neural Network

- Ein RNN ist ein neuronales Netzwerk, das für die Verarbeitung von **sequenziellen Daten** entwickelt wurde.
 - Es hat eine **Gedächtnisfunktion**, die es ermöglicht, Informationen aus früheren Schritten zu speichern.
- **Anwendung:**
 - Zeitreihenvorhersage (z. B. Aktienkurse, Wetter).
 - Textverarbeitung (z. B. Textgenerierung, Sentiment-Analyse).
 - Spracherkennung.

Aufbau eines RNN

- Ein RNN besteht aus einer **wiederholten Zelle**, die Informationen über die Zeit weiterreicht.
 - Jede Zelle nimmt zwei Eingaben:
 - a. Den aktuellen Eingabewert x_t .
 - b. Den versteckten Zustand h_{t-1} aus dem vorherigen Schritt.

$$h_t = \tanh(W_h \cdot h_{t-1} + W_x \cdot x_t + b)$$

$$y_t = W_y \cdot h_t + b_y$$

Problem herkömmlicher RNNs

- **Vanishing Gradient:**
 - Bei langen Sequenzen "verschwinden" die Gradienten während des Backpropagation.
 - Dies führt dazu, dass das Netzwerk nicht lernt.
- **Exploding Gradient:**
 - Die Gradienten können auch explodieren, was zu instabilen Updates führt.

LSTM

- Long short-term memory
- LSTM ist eine spezielle Art von **Recurrent Neural Network (RNN)**.
- Es wurde entwickelt, um das Problem des **vanishing gradient** in herkömmlichen RNNs zu lösen.
- **Anwendung:**
 - Zeitreihenvorhersage (z. B. Aktienkurse, Wetter).
 - Sequenz-zu-Sequenz-Modelle (z. B. Maschinelle Übersetzung, Textgenerierung).

LSTM vs. RNNs

- Vanishing Gradient:
 - Bei langen Sequenzen "verschwinden" die Gradienten während des Backpropagation.
- Lösung:
 - LSTM führt **Gedächtniszellen (Memory Cells)** ein, die Informationen über lange Zeiträume speichern können.

Aufbau einer LSTM-Zelle

1. **Forget Gate:** Entscheidet, welche Informationen verworfen werden.
2. **Input Gate:** Fügt neue Informationen hinzu.
3. **Output Gate:** Steuert, welche Informationen ausgegeben werden.

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

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

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

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

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

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

Übersicht der Komponenten

Komponente	Formel	Funktion
Forget Gate	$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$	Entscheidet, welche Informationen aus C_{t-1} verworfen werden.
Input Gate	$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$	Entscheidet, welche neuen Informationen zu C_t hinzugefügt werden.
Kandidatenzustand	$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$	Berechnet potenzielle neue Informationen für C_t .

Komponente	Formel	Funktion
Zellzustand	$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$	Aktualisiert den Zellzustand.
Output Gate	$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$	Entscheidet, welche Informationen aus C_t als h_t ausgegeben werden.
Versteckter Zustand	$h_t = o_t \cdot \tanh(C_t)$	Gibt die gefilterte Version des Zellzustands aus.

Erklärung der Symbole

- h_{t-1} : Versteckter Zustand aus dem vorherigen Zeitschritt.
- x_t : Eingabe zum aktuellen Zeitschritt.
- W_f, W_i, W_C, W_o : Gewichtsmatrizen für die jeweiligen Gates.
- b_f, b_i, b_C, b_o : Bias-Werte für die jeweiligen Gates.
- σ : Sigmoid-Funktion (Werte zwischen 0 und 1).
- \tanh : Tangens hyperbolicus (Werte zwischen -1 und 1).
- C_t : Aktualisierter Zellzustand.
- h_t : Versteckter Zustand zum aktuellen Zeitschritt.

Datenfluss

1. **Eingabe:** h_{t-1} und x_t .
2. **Forget Gate:** Bestimmt, was aus C_{t-1} behalten wird.
3. **Input Gate:** Bestimmt, welche neuen Informationen zu C_t hinzugefügt werden.
4. **Zellzustand:** C_t wird aktualisiert.
5. **Output Gate:** Bestimmt, was aus C_t als h_t ausgegeben wird.
6. **Ausgabe:** h_t wird an den nächsten Zeitschritt weitergegeben.