

Deep Learning und Text Mining - Textanalyse und - generierung

Schulung am 20.01.2025 bis 24.01.2025

Ihr Trainer: Nicolas Kuhaupt

Überblick Zeiten

9:00 – 10:30



10:45 – 12:00



13:00 – 14:45



15:00 – 16:30

16:30 – 17:30 **OpenSpace**

Vorstellungsrunde

Erwartungen an die Schulung?

Vorkenntnisse?

Wenn ein KI-Sprachmodell einen Teil eurer Arbeit automatisieren könnte, was würdet ihr als erstes automatisieren?



Materialien im Repository + Einrichtung in Pycharm

Daten, Code, Glossar

https://github.com/NKDataConv/schulung_deep_learning

Anmerkungen zur Schulung

- Jeder sollte coden und Code ausführen
- Spielerisch mit Code umgehen und Dinge ausprobieren
- Für Lösung von Übungsaufgaben ist an einigen Stellen die Zuhilfenahme der Dokumentation notwendig

Agenda

1. Grundlagen des Deep Learning
2. Erweiterte Konzepte und Architekturen
3. Verarbeitung von Textdaten (NLP-Grundlagen)
4. Transformer-Modelle
5. Trainingspraxis und Optimierung
- (6. MLOps und Deployment)
- (7. Anwendungsfälle und Projekte)

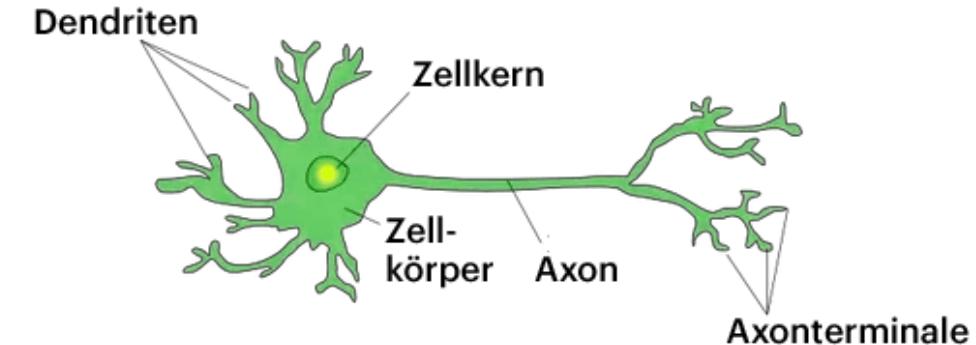
Agenda

1. Grundlagen des Deep Learning

2. Erweiterte Konzepte und Architekturen
3. Verarbeitung von Textdaten (NLP-Grundlagen)
4. Transformer-Modelle
5. Trainingspraxis und Optimierung
6. MLOps und Deployment
7. Anwendungsfälle und Projekte

Grundlagen biologischer Neuronen: Aufbau und Funktionsweise

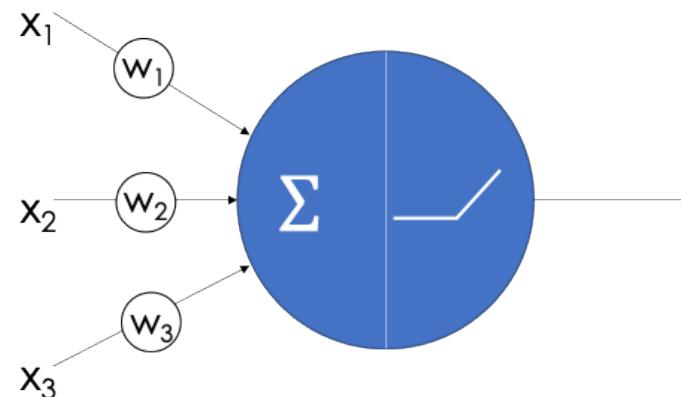
- Neuronale Grundstrukturen umfassen **Dendriten** (zur Aufnahme von Signalen), das **Soma** (Zellkörper) zur Verarbeitung der **eingehenden Informationen** und das **Axon zur Weiterleitung** elektrischer Impulse.
- Die **Informationsübertragung** erfolgt elektrisch entlang des Axons sowie chemisch über Neurotransmitter an den Synapsen.
- **Synapsen bilden Kontaktstellen** zwischen Neuronen und gewährleisten die chemische Signalweitergabe.
- Die Komplexität biologischer Neuronen dient als **Inspirationsquelle** für künstliche neuronale Netze.



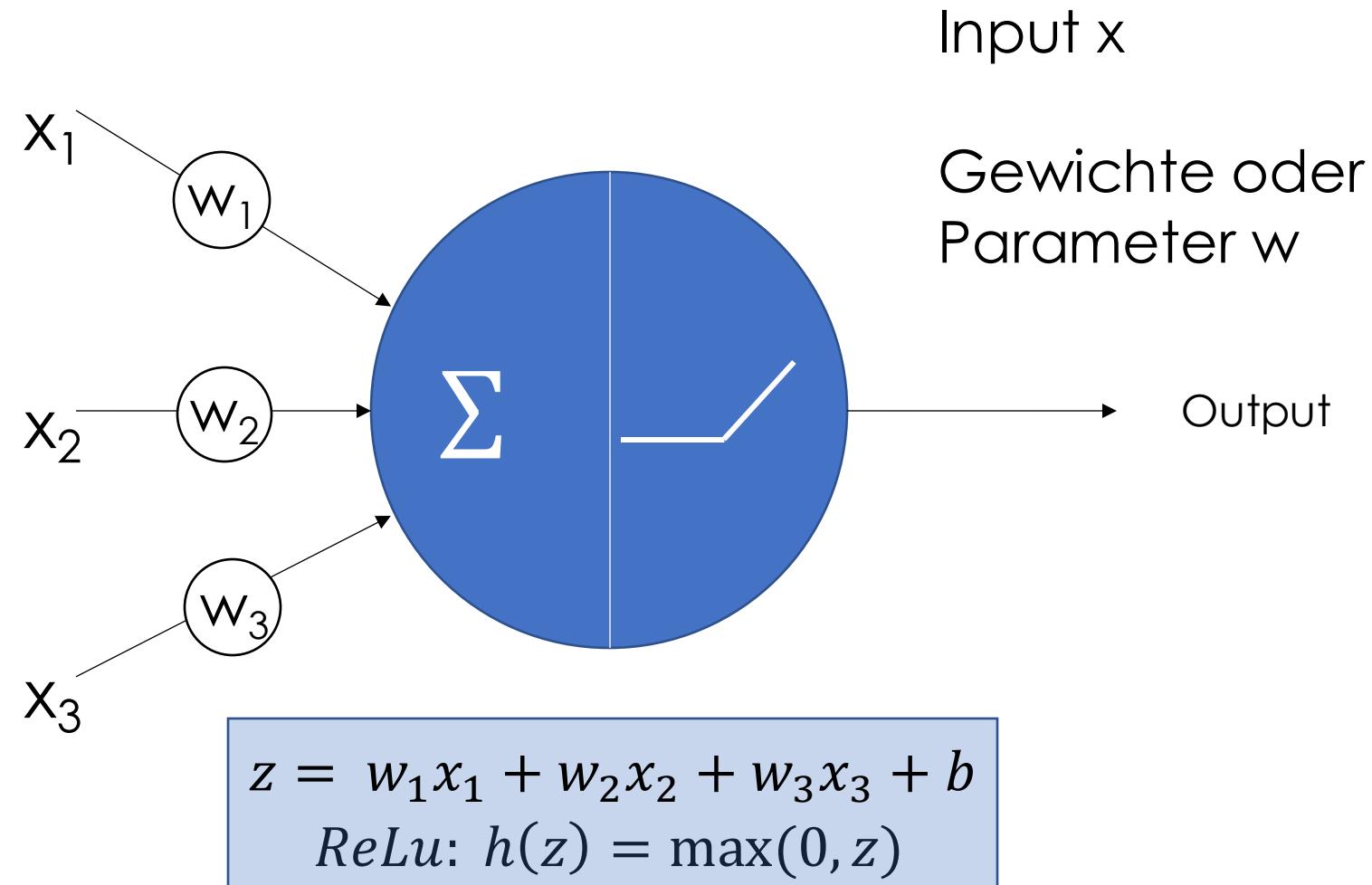
<https://i0.wp.com/katzlberger.ai/wp-content/uploads/2019/12/neuronales-netzwerk-mensch-maschine.png?fit=602%2C376&ssl=1>

Vergleich künstliches vs. biologisches Neuron: Ähnlichkeiten und Unterschiede

- Künstliche Neuronen sind **vereinfachte** mathematische **Modelle** biologischer Neuronen.
- Biologische Neuronen sind hochkomplex, weisen enorme Anpassungsfähigkeit auf und lernen kontinuierlich.
- In beiden Fällen werden Eingangssignale verarbeitet, um ein Ausgabe- bzw. Antwortsignal zu erzeugen.
- Während biologische Prozesse stark variieren und **chemisch-elektrisch** ablaufen, arbeiten künstliche Neuronen nach **deterministischen, reproduzierbaren** Berechnungen.



Neuronale Netze – das Neuron



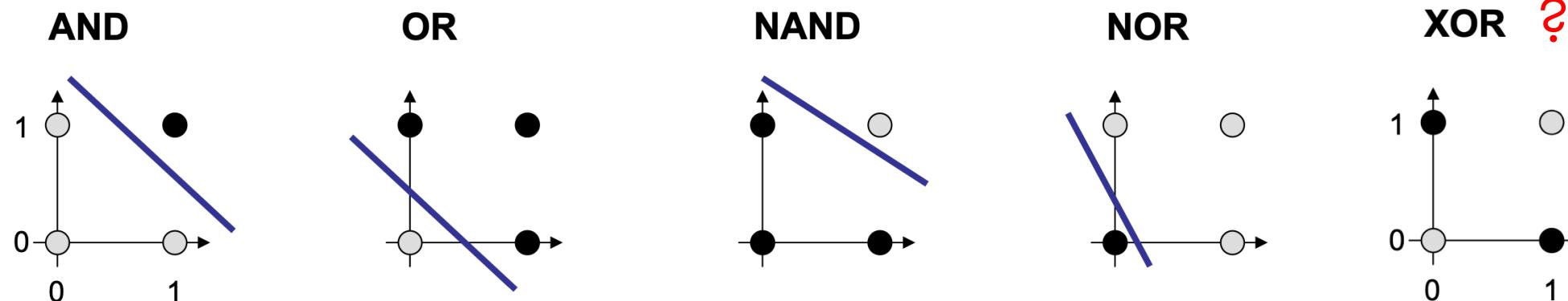


Code Demo: Einfaches Perceptron in NumPy implementieren

Grundlagen_Deep_Learning/Einfuehrung_neuronen/demo_perceptron.py

Historischer Überblick: Perceptron und frühe neuronale Netze

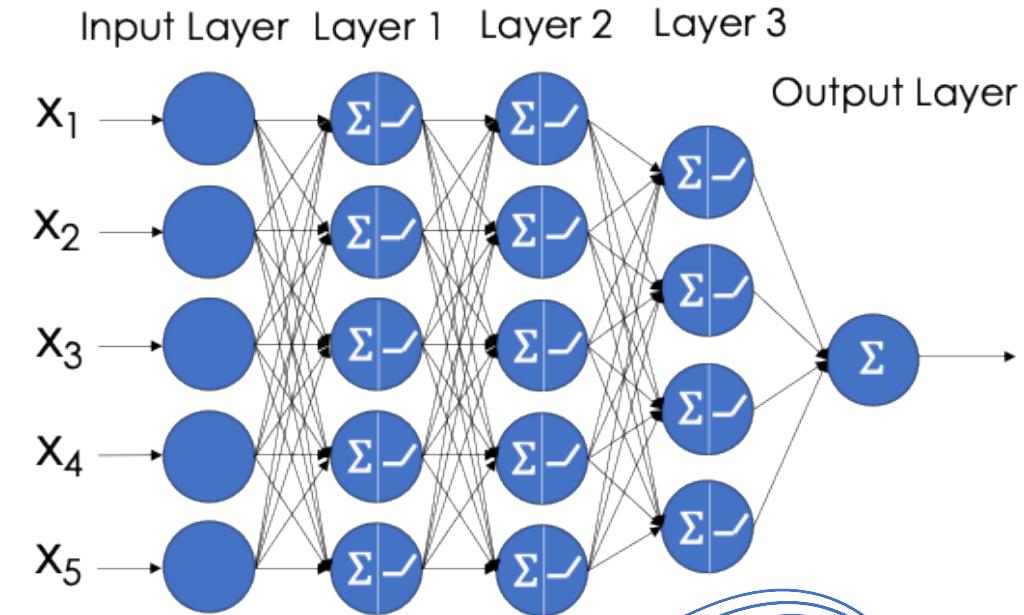
- Das **Perceptron** (1958) war ein einfaches, lineares Modell für **binäre Klassifikationsaufgaben**.
- Die **Unfähigkeit**, nichtlineare **Probleme wie XOR** zu lösen, führte zur Entwicklung von **Mehrschichtnetzwerken**.
- Der Durchbruch kam in den 1980er Jahren mit der **Backpropagation**, die tiefere Netze effektiv trainierbar machte.
- Diese frühen Ansätze legten die Grundlage für moderne Deep-Learning-Techniken.



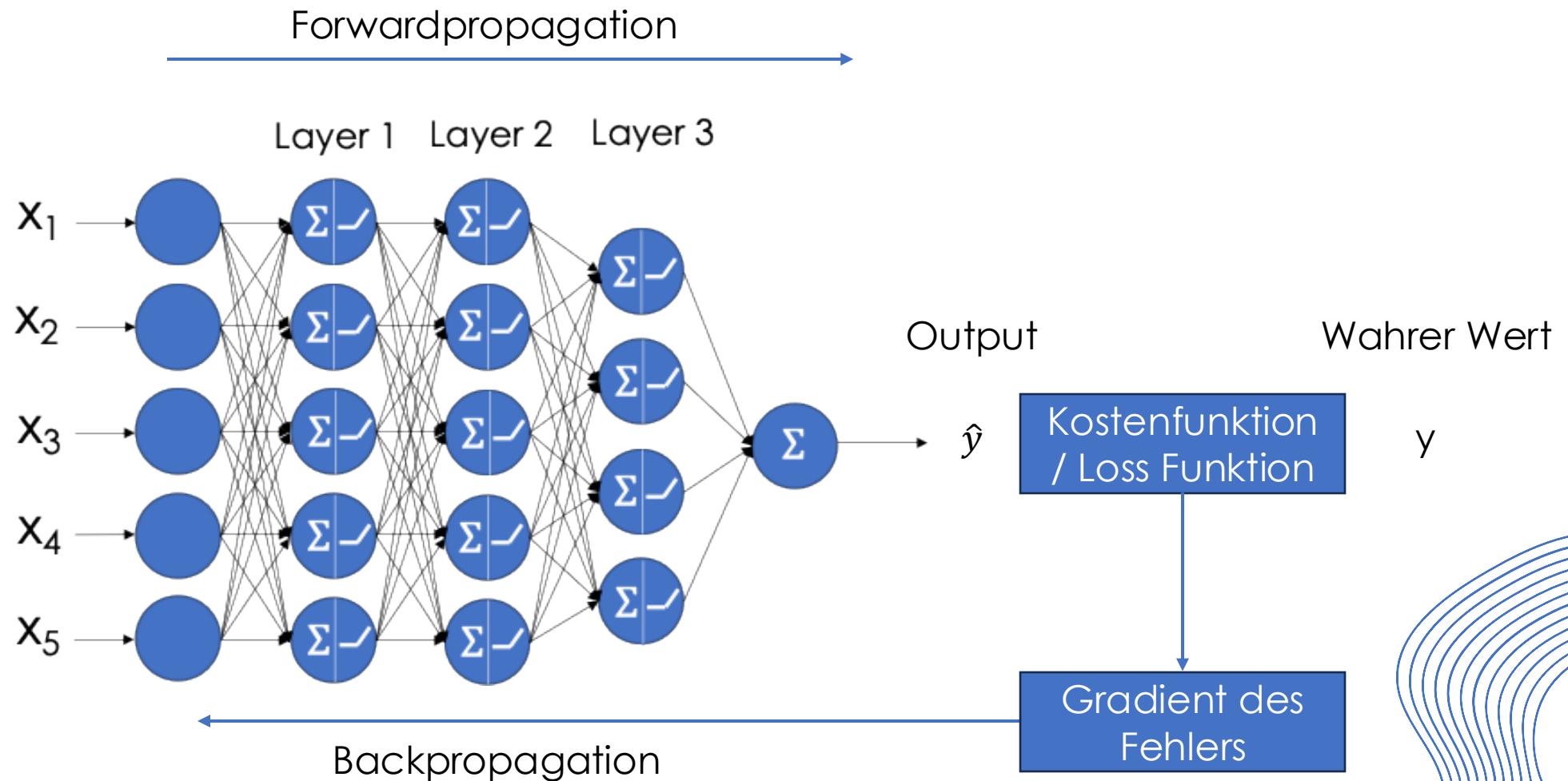
<https://ls11-www.cs.tu-dortmund.de/people/rudolph/teaching/lectures/POKS/WS2007-08/lecMeta.pdf>

Schichtenarchitektur: Input-, Hidden- und Output-Layer im Überblick

- Der **Input-Layer** nimmt die Rohdaten auf und gibt sie an die nächsten Schichten weiter.
- **Hidden-Layer** extrahieren schrittweise abstrakte Merkmale aus den Eingangsdaten.
- Der **Output-Layer** liefert letztlich die finale Vorhersage, beispielsweise eine Klasse oder einen numerischen Wert.
- Durch mehrere hintereinander geschaltete Schichten (Tiefe) können komplexe Muster erkannt werden.



Beispiel eines einfachen neuronalen Netzes: Datenfluss von Input bis Output

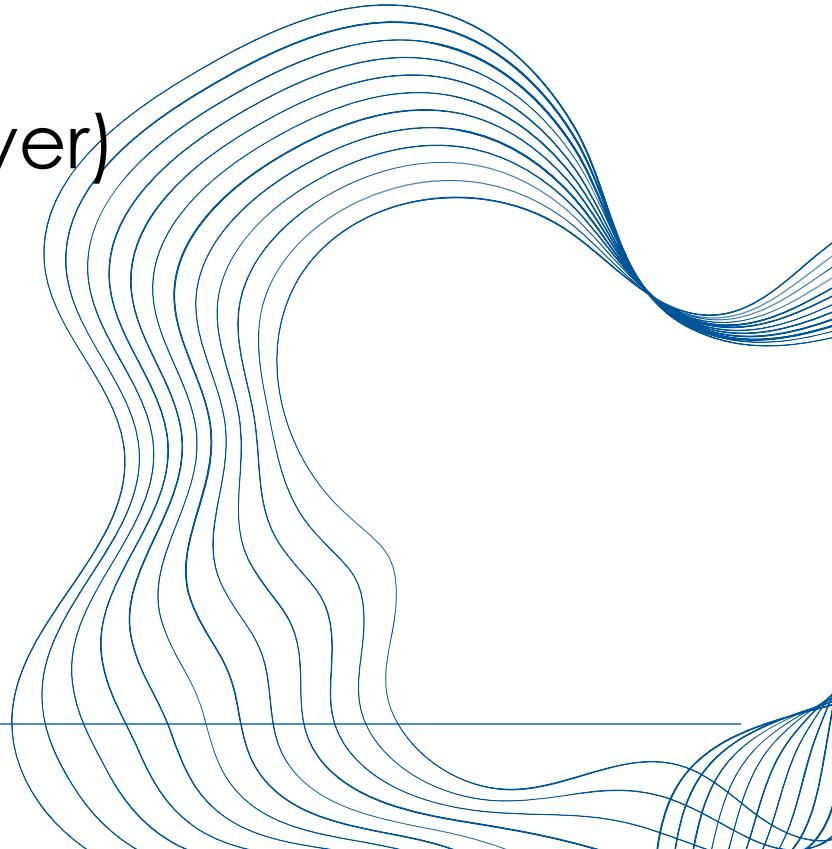


Beispiel eines einfachen neuronalen Netzes: Datenfluss von Input bis Output

- Die Eingabedaten passieren **Schicht für Schicht gewichtete Verbindungen**, die ihre Bedeutung anpassen.
- **Aktivierungsfunktionen** entscheiden, welche Signale weitergeleitet werden, um nichtlineare Beziehungen abzubilden.
- Über einen **Fehlervergleich (Loss)** zwischen Soll- und Ist-Ausgabe wird die Genauigkeit bestimmt.
- Durch **Anpassung der Gewichte anhand des Fehlers** lernt das Netz, seine Vorhersagen schrittweise zu verbessern.

Begriffe

- = Neuronales Netz
- = Neural Net (NN)
- = Multilayer Perceptron (MLP)
- = (Mehrlagen Perzeptron)
- = Feed Forward Network (nur Standard Layer)
(ab 2 Hidden Layers)
- = Tiefes Neuronales Netz
- = Deep Neural Net

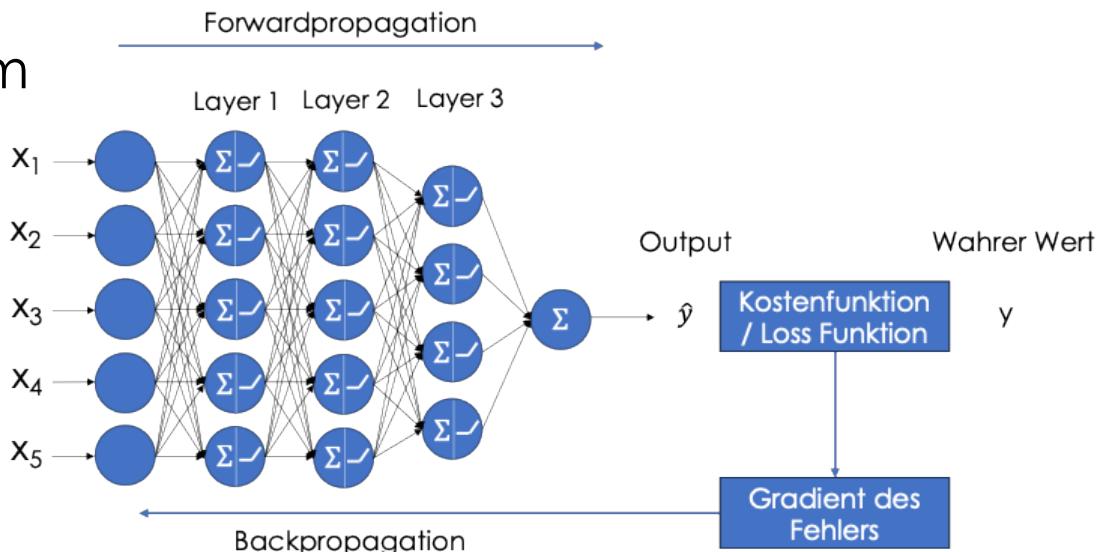


Playground

<https://playground.tensorflow.org/>

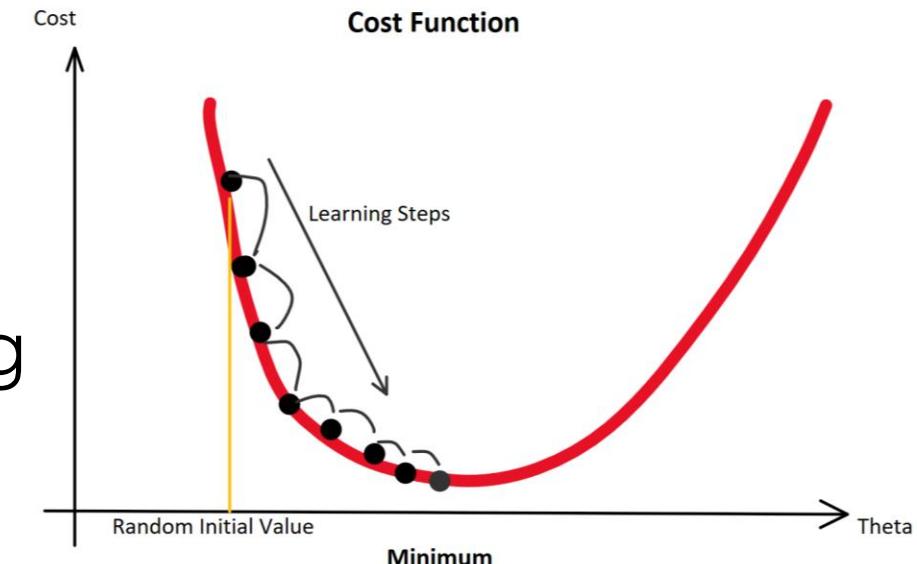
Backpropagation: Idee, mathematische Grundlagen und Intuition

- Bei der Backpropagation wird der am Ausgang **gemessene Fehler rückwärts durch das Netzwerk propagiert.**
- Mit Hilfe der **Kettenregel** lassen sich **Gradienten** der Gewichte berechnen.
- Gewichte werden **iterativ** korrigiert, um den Fehler kontinuierlich zu reduzieren.
- Backpropagation ist das **zentrale Verfahren** zum Trainieren tiefgreifender neuronaler Netze.



Gewichtsaktualisierung mit Gradientenabstieg: Von Partialableitungen zu neuen Gewichten

- Zunächst wird der **Gradient** der Loss-Funktion bezüglich der Gewichte berechnet.
- Die **Gewichte** werden dann in Richtung des abnehmenden Fehlers **angepasst**.
- Die **Lernrate** bestimmt die Größe der Gewichtsänderungen pro Schritt.
- Ziel ist es, den **Gesamtfehler durch kontinuierliche Updates zu minimieren**.

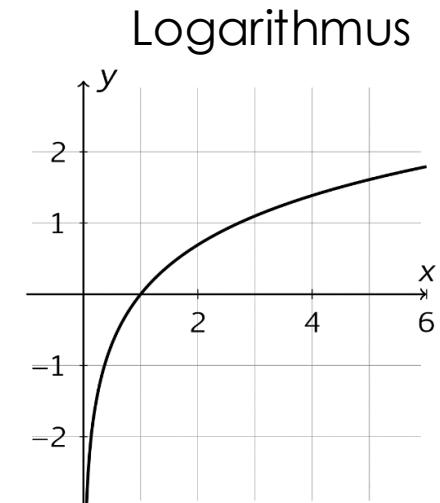


<https://data-science-blog.com/wp-content/uploads/2019/01/gradient-descent-cost-function.png>

Verständnis von Loss-Funktionen (MSE, Cross Entropy) im Kontext des Lernprozesses

- **MSE** (Mean Squared Error / Mittlere quadratische Abweichung) wird häufig bei Regressionsaufgaben als Fehlermaß eingesetzt.
- **Cross Entropy** (Kreuzentropie) ist ein gängiges Maß für Klassifikationsfehler und betrachtet Wahrscheinlichkeitsverteilungen.
- Die Loss-Funktion bestimmt über ihren Gradienten die Richtung des Lernschrittes.
- Die Wahl der **Loss-Funktion richtet sich nach der Art der Aufgabe** (Regression, Klassifikation, etc.).

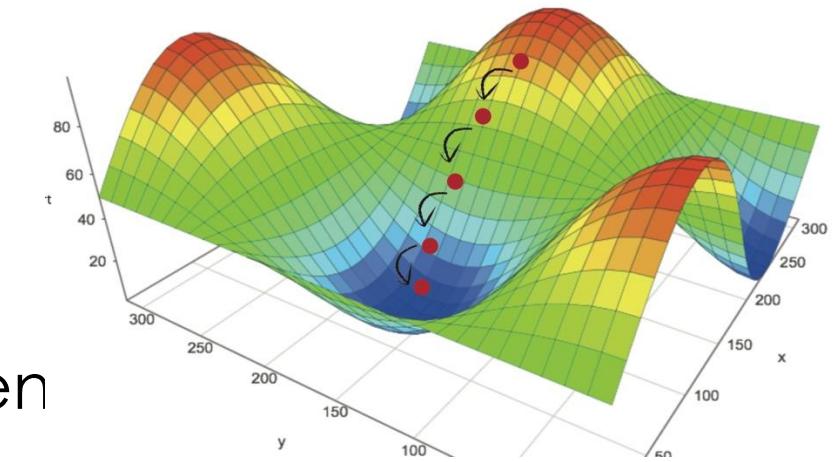
$$\sum \frac{(y - \hat{y})^2}{n}$$
$$-(y \log(\hat{y}) + (1-y) \log(1- \hat{y}))$$



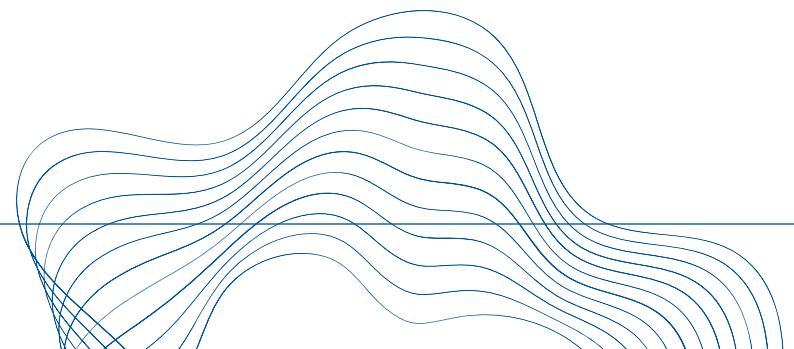
<https://www.schuelerhilfe.de/online-lernen/1-mathematik/690-die-logarithmusfunktion>

Wahl der Lernrate: Einfluss auf Konvergenz und Stabilität des Trainings

- Hohe Lernraten ermöglichen **schnelle Fortschritte**, bergen aber das Risiko **unkontrollierter Schwankungen**.
- Niedrige Lernraten sorgen für **stabiles**, aber möglicherweise **sehr langsames Training**.
- Adaptive Optimizer (Adam, RMSProp) passen die Lernrate automatisch an.
- Eine durchdachte Lernratenwahl ist **entscheidend für effizientes und stabiles Training**.

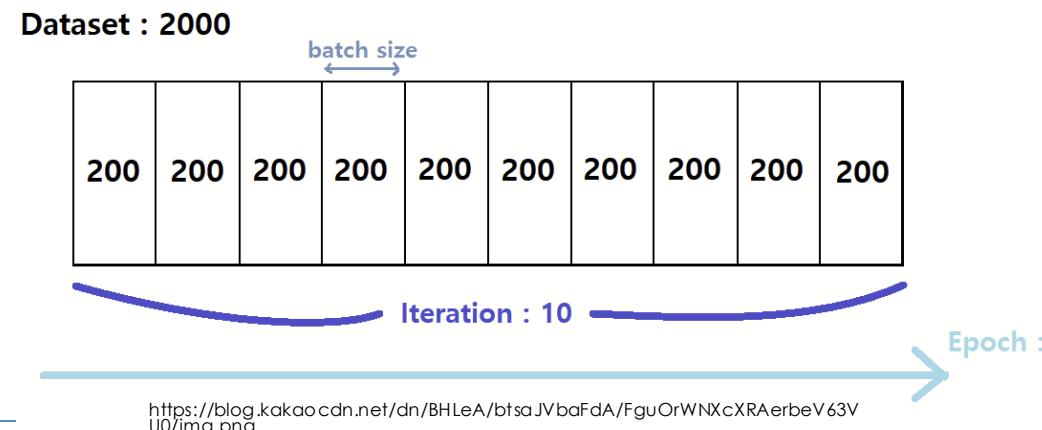


<https://oreillyblog.dpunkt.de/wp-content/uploads/2017/06/gradienten.jpg>



Überblick über einen Trainingszyklus (Epochen, Batches, Iterationen)

- Eine **Epoche** entspricht einem vollständigen Durchlauf des gesamten Trainingsdatensatzes.
- Ein **Batch** ist ein kleiner Auszug der Daten, auf dem jeweils ein einzelner Update-Schritt durchgeführt wird.
- Eine **Iteration** beschreibt genau diesen einzelnen Schritt der Gewichtsaktualisierung pro Batch.
- Mehrere Epochen verbessern schrittweise die Leistung des Modells.



Praktisches Beispiel mit einfacherem Datensatz: Schritt-für-Schritt-Durchlauf

- Zuerst werden die **Daten** geladen, bereinigt und normalisiert.
- Das Modell **berechnet Vorhersagen**, die mit den echten Werten verglichen werden, um den **Fehler zu bestimmen**.
- Anschließend passt **Backpropagation** mithilfe des Gradientenabstiegs die Gewichte an.
- Dieser Prozess wird **wiederholt**, bis die gewünschte Genauigkeit erreicht ist.

Python Bibliotheken



Hugging Face

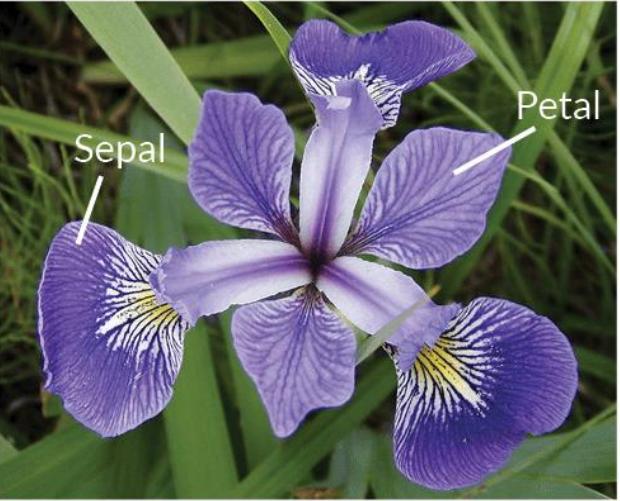
(PyTorch)



Code Demo: Einfaches neuronales Netz in Keras trainieren

Grundlagen_Deep_Learning/Trainingsprozess/demo_keras_training.py

Iris Datensatz



Iris Versicolor



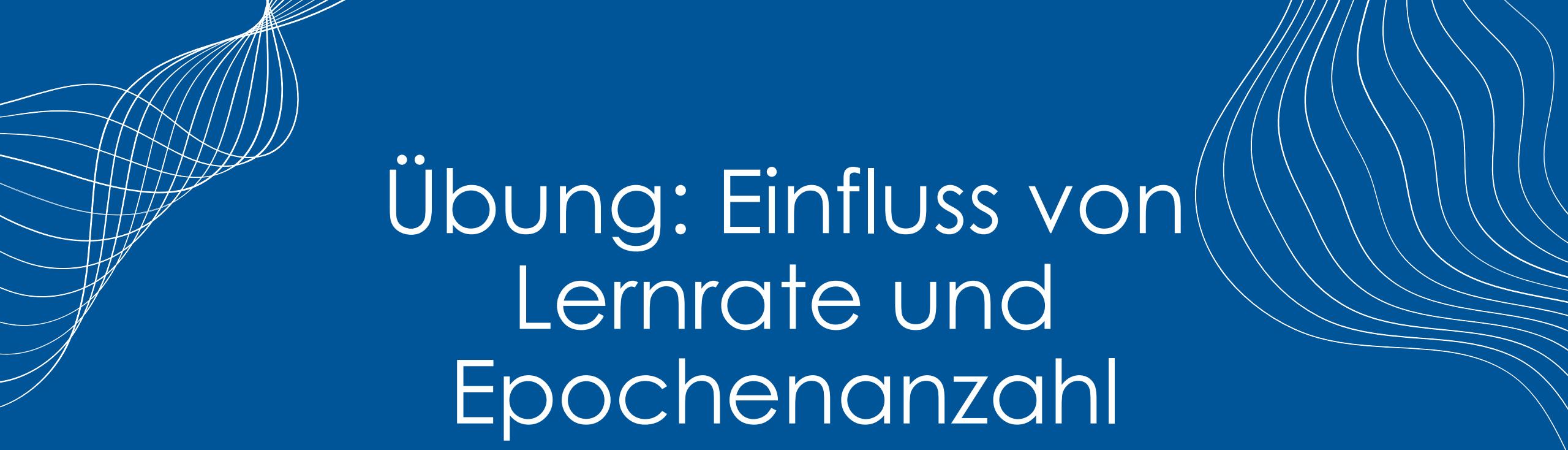
Iris Setosa



Iris Virginica

https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Machine+Learning+R/iris-machinelearning.png



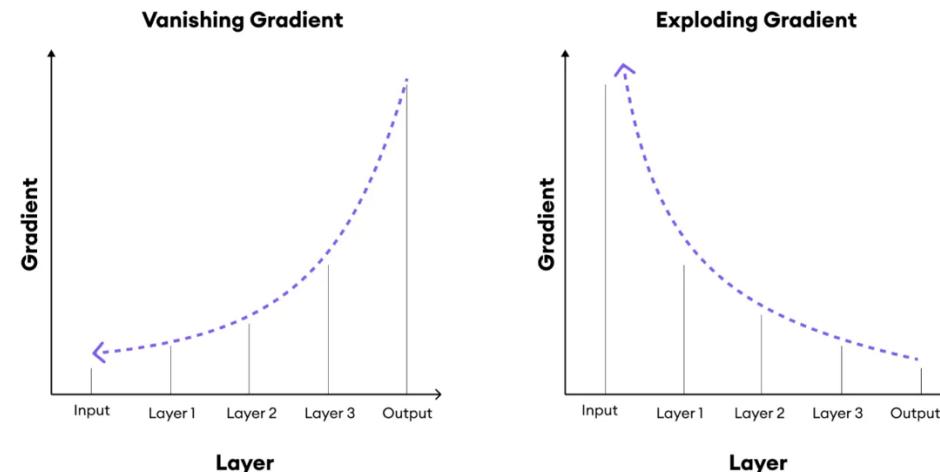


Übung: Einfluss von Lernrate und Epochenanzahl experimentell untersuchen

Grundlagen_Deep_Learning/Trainingsprozess/uebung_gradienstabstieg.py

Vanishing und Exploding Gradients

- Bei **Vanishing Gradients** werden die Fehler-Signale durch häufiges Multiplizieren kleiner Zahlen praktisch „ausgelöscht“.
- Vanishing Gradients führen dazu, dass tiefer liegende Schichten kaum noch lernrelevante Updates erhalten, wodurch das Training nahezu zum Stillstand kommt.
- Bei **Exploding Gradients** werden die Fehler-Signale durch häufiges Multiplizieren großer Zahlen so stark, dass sie das System „sprengen“.
- Exploding Gradients sorgen für instabile Gewichtsupdates, die zu extremen Modellparametern und chaotischem Lernverhalten führen.



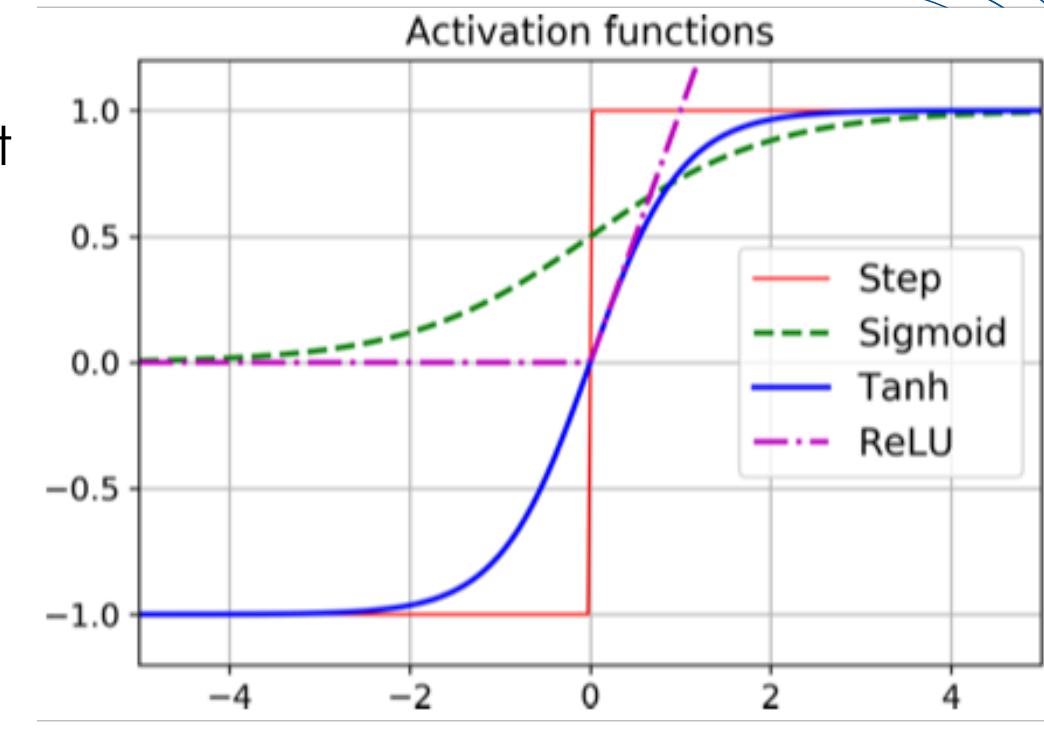
<https://aiml.com/wp-content/uploads/2023/11/vanishing-and-exploding-gradient-1.png>

Maßnahmen zur Entschärfung: Angepasste Initialisierung, andere Aktivierungsfunktionen

- Eine durchdachte **Gewichtsinitialisierung** (z. B. Glorot oder He) reduziert das Risiko des Vanishing Gradient.
- Der Einsatz von **ReLU-Varianten** (ReLU, Leaky ReLU, ELU) verhindert Aktivierungssättigungen, wie sie bei Sigmoid oder Tanh häufiger vorkommen.
- **Batch Normalization** stabilisiert Eingaben für nachfolgende Schichten, erleichtert das Training tiefer Modelle und ermöglicht oft höhere Lernraten.
- Die Kombination aus angepasster Initialisierung, geeigneter Aktivierungsfunktion und Normalisierungsschichten führt in der Praxis häufig zu deutlich **robusteren Trainingsprozessen**.

Aktivierungsfunktionen: Eigenschaften von Sigmoid, ReLU, Tanh

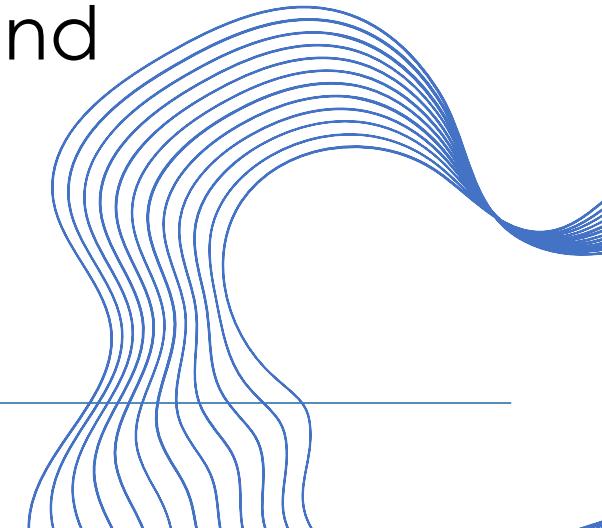
- **Sigmoid** wandelt Eingaben in Werte zwischen 0 und 1 um, ideal für Wahrscheinlichkeitsinterpretationen.
- **Tanh** gibt Werte zwischen -1 und 1 aus, ist um 0 zentriert und oft leistungsfähiger als Sigmoid.
- **ReLU** (Rectified Linear Unit) ist einfach, verhindert zum Teil das Verschwinden von Gradienten und ist sehr verbreitet.
- **Die Wahl der Aktivierungsfunktion hängt von Aufgabe, Daten und Modellarchitektur ab.**



https://miro.medium.com/v2/resize:fit:444/1*BrcwfzrjHoHf3jfSfmE5ww.png

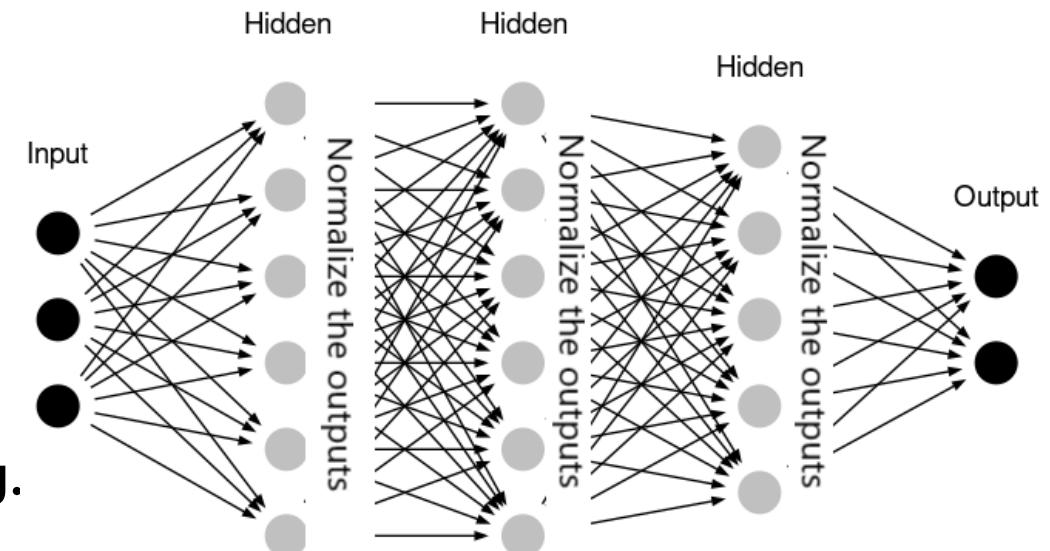
Gewichtsinitialisierungsmethoden (Glorot, He) und deren Einfluss auf Konvergenz

- Eine sorgfältige **Initialisierung der Gewichte** vermeidet Vanishing und Exploding Gradients.
- **Glorot-Initialisierung** (Xavier) sorgt für ausgeglichene Varianzverteilungen in den Schichten.
- **He-Initialisierung** ist speziell für ReLU-Schichten konzipiert und unterstützt schnelles, stabiles Training.
- Eine passende Initialisierung fördert schnellere und stabilere Konvergenz.



Batch Normalization: Normalisierung der Zwischenoutputs für stabileres Training

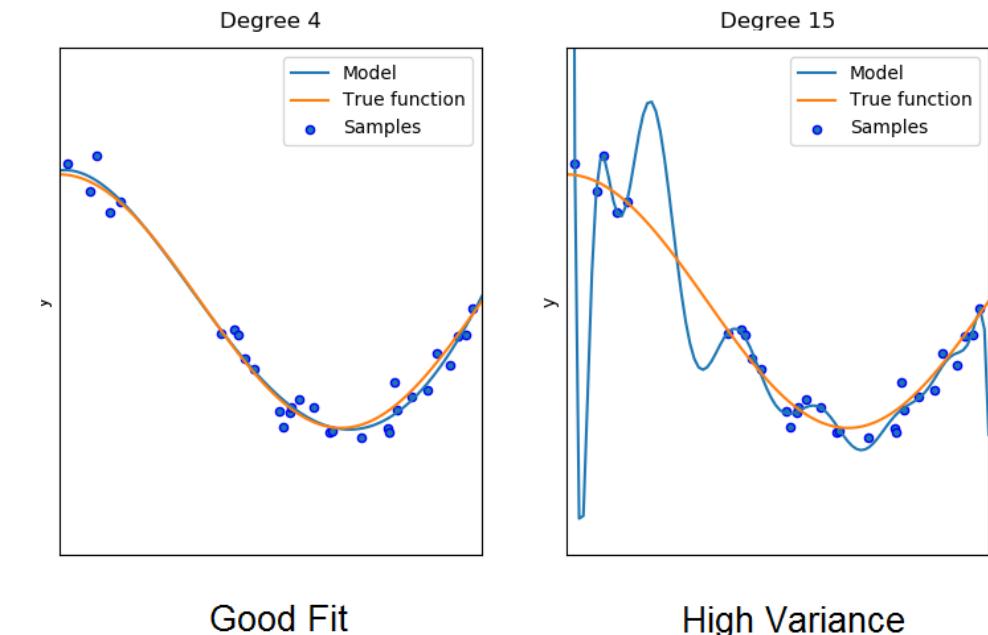
- **Batch Normalization** normalisiert die Aktivierungen jeder Schicht, um Schwankungen zu reduzieren.
- Dadurch können höhere Lernraten genutzt werden, was das **Training beschleunigt**.
- Das Verfahren minimiert interne Kovarianzverschiebungen und **stabilisiert den Lernprozess**.
- **Insgesamt führt Batch Normalization zu schnellerem, robusteren und präziserem Training.**



https://images.deeppi.org/glossary-terms/981e1ffea3814ae193c27461253faf63/batch_normalization.png

Regularisierungsmethoden: L1, L2, Dropout und deren Wirkung auf Generalisierung

- **L1- und L2-Regularisierung** fügen Strafen für große Gewichte hinzu, um Overfitting zu vermeiden.
- **Dropout** schaltet Neuronen zufällig aus und erhöht die Robustheit des Modells.
- Beide Methoden verbessern die Generalisierungsfähigkeit und reduzieren Überanpassung.
- **Diese Techniken sind Schlüsselemente, um ein gutes Gleichgewicht zwischen Komplexität und Generalisierung zu erreichen.**



https://miro.medium.com/proxy/0*seXiPgCCaH1N9sJJ

Klassifikation mit MNIST

10 Klassen, handgeschriebene Zahlen

Beliebter Datensatz für
Bildklassifikation



<https://de.wikipedia.org/wiki/MNIST-Datenbank>



Code Demo: Vergleich eines MLP mit und ohne Dropout

Grundlagen_Deep_Learning/Bausteine_DL_Modelle/demo_
dropout.py



Übung: Verschiedene Aktivierungsfunktionen in einem MLP testen

Grundlagen_Deep_Learning/Bausteine_DL_Modelle/uebung
_aktivierungsfunktionen.py

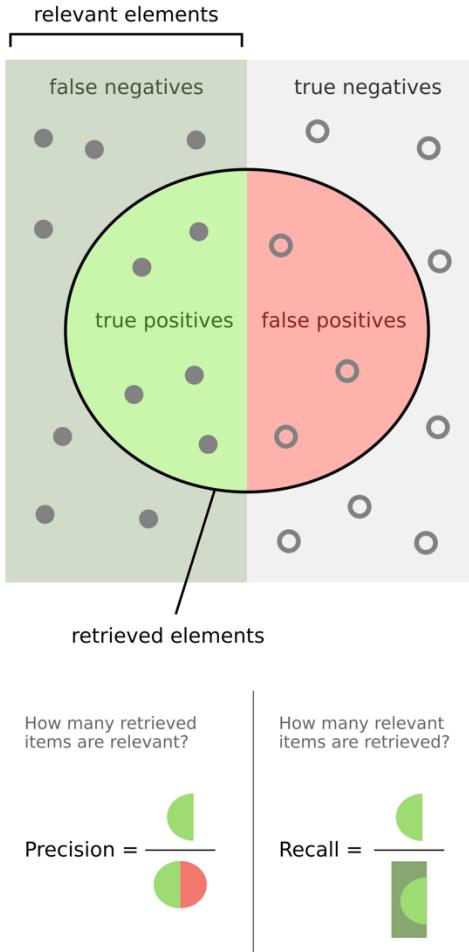
Wichtige Metriken Regression: MSE/MAE

$$\sum \frac{(y - \hat{y})^2}{n}$$

$$\sum \frac{|y - \hat{y}|}{n}$$

- **MSE** (Mean Squared Error) ist sensitiv gegenüber Ausreißern und oft bei Regressionen gebräuchlich.
- **MAE** (Mean Absolute Error) betrachtet den absoluten Fehler und ist robuster gegenüber Ausreißern.
- Die Wahl der Metrik hängt stark von der Aufgabenstellung und den Eigenschaften der Daten ab.

Wichtige Metriken Klassifikation: Accuracy, F1-Score, Precision und Recall

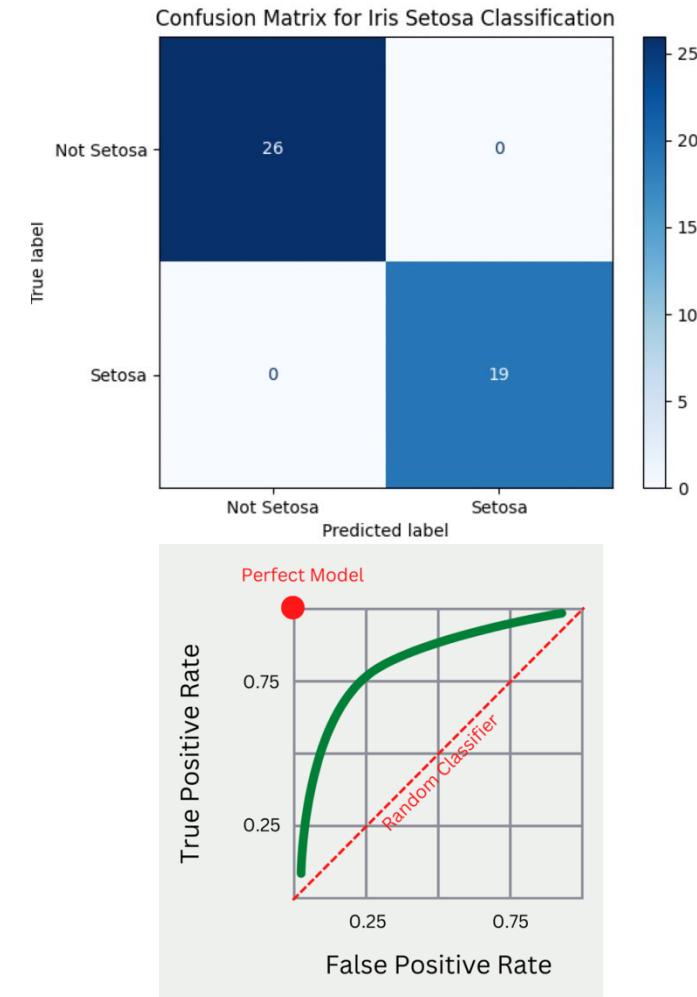


<https://upload.wikimedia.org/wikipedia/commons/thumb/2/26/Precisionrecall.svg/1200px-Precisionrecall.svg.png>

- **Accuracy** misst den Anteil korrekt klassifizierter Beispiele, sinnvoll bei **ausbalancierten** Datensätzen.
- **Precision** definiert den Anteil der als positiv vorhergesagten Beispiele, die tatsächlich positiv sind.
- **Recall** misst, wie viele der tatsächlich positiven Beispiele korrekt erkannt werden.
- Der **F1-Score** ist das harmonische Mittel aus Precision und Recall und hilft bei unausgewogenen Daten.
- Diese Metriken sind besonders relevant, wenn Klassen stark **unterschiedlich häufig** vorkommen.

Konfusionsmatrix und ROC-Kurve: Tiefergehende Evaluation von Klassifikationsmodellen

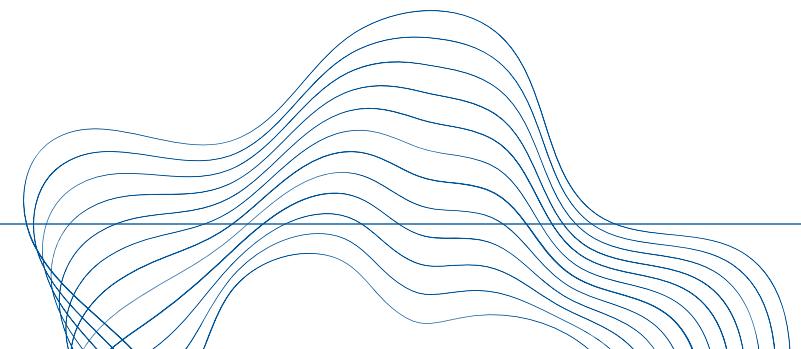
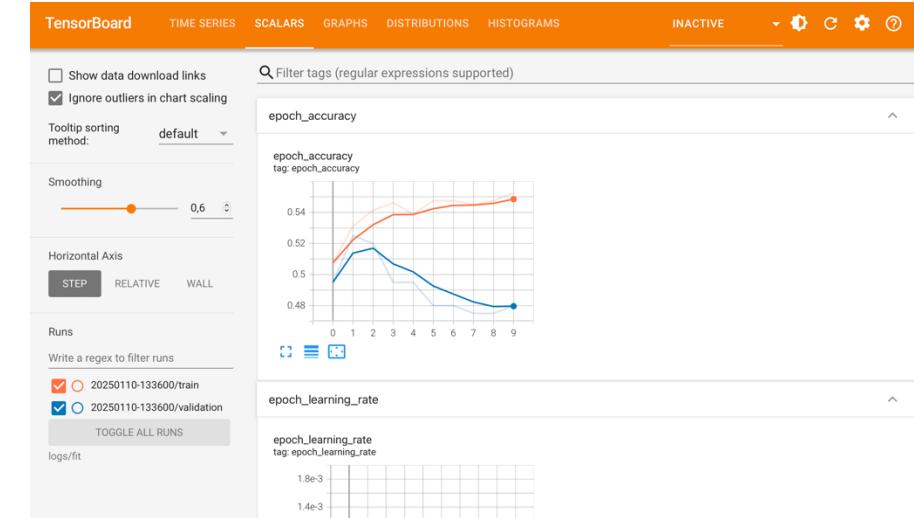
- Eine **Konfusionsmatrix** zeigt detailliert, welche Klassen oft verwechselt werden.
- Die **ROC-Kurve** illustriert das Verhältnis zwischen True Positive Rate und False Positive Rate.
- Die **AUC** (Area Under Curve) fasst die Modellgüte in einer einzelnen Kennzahl zusammen.
- Diese Visualisierungen erleichtern eine präzise Fehleranalyse und gezielte Modellverbesserung.



<https://datacamp.com/wp-content/uploads/ROC-Curve-Diagram-1024x709.png>

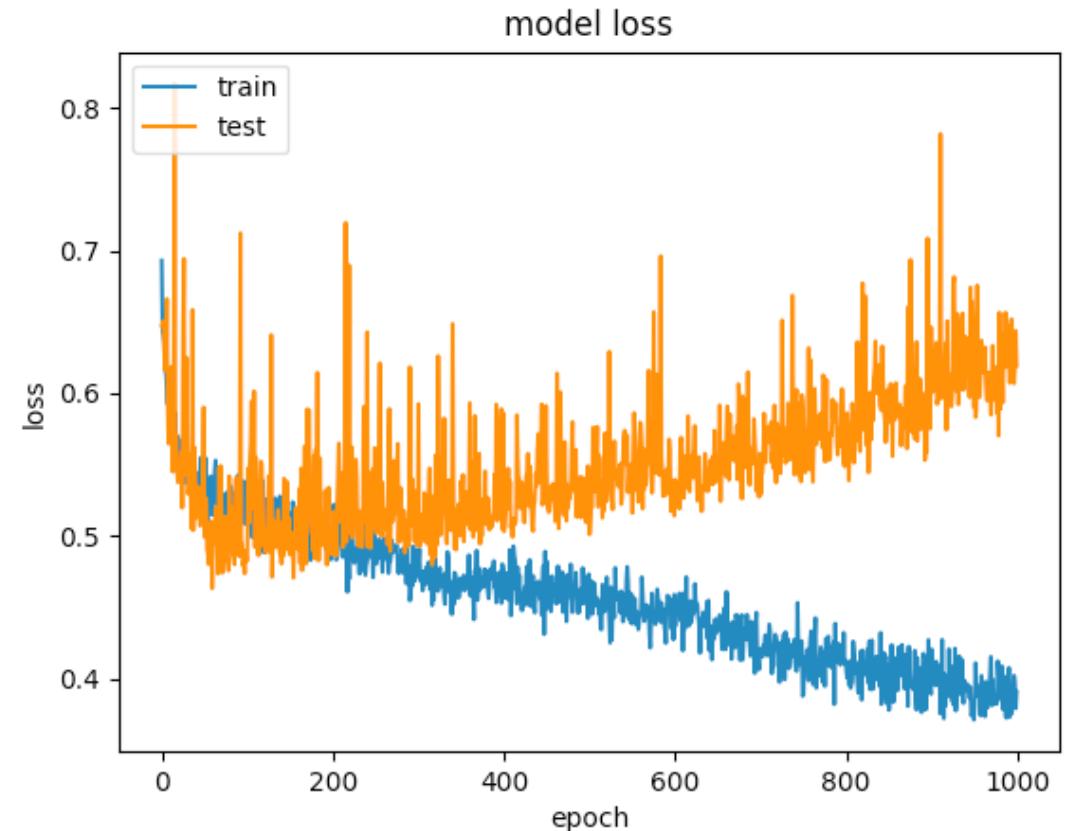
Visualisierung von Loss- und Metrikverläufen in TensorBoard

- Die Darstellung der **Loss-Kurve zeigt den Lernfortschritt über Epochen hinweg.**
- **Metrikplots** (z. B. Accuracy, F1) helfen, die Modellleistung besser einzuschätzen.
- Frühzeitiges Erkennen von **Over- oder Underfitting** ist durch die Visualisierung möglich.
- Unterschiedliche **Experimente und Modellvarianten** können direkt verglichen werden.



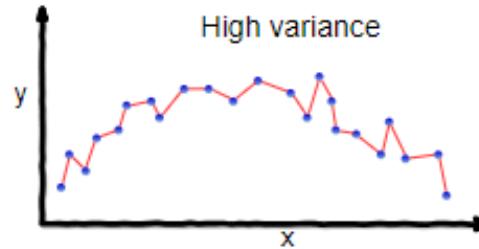
Vergleich von Trainings- und Validierungsmetriken zur Einschätzung der Modellgeneralisierung

- **Trainingsmetriken** fallen oft besser aus, da das Modell auf diese Daten abgestimmt ist.
- **Validierungsmetriken** zeigen, wie gut das Modell auf unbekannte Daten generalisiert.
- Große **Abweichungen** zwischen beiden Metriken deuten auf **Overfitting** hin.
- Diese Analysen erleichtern die gezielte Feinjustierung des Modells.

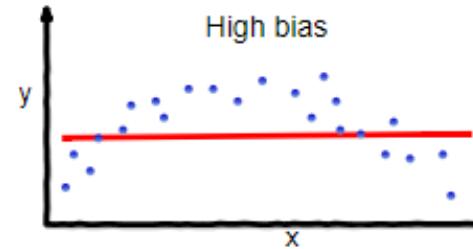


Overfitting, Underfitting und Bias-Variance Tradeoff: Grundlagen und Erkennungsmerkmale

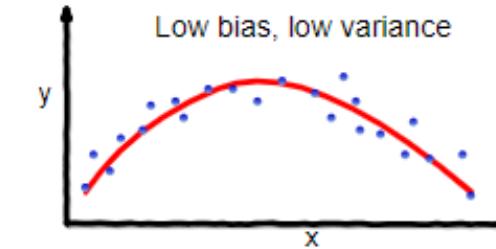
- **Overfitting:** Das Modell passt sich zu stark an die Trainingsdaten an und verfehlt die Generalisierung auf neue, unbekannte Daten. Erkennbar an stark divergierender Trainings- und Validierungsleistung.
- **Underfitting:** Das Modell ist zu einfach und erkennt die zugrundeliegenden Muster nicht ausreichend. Sowohl Training als auch Validierung zeigen schwache Performance.
- Der **Bias-Variance Tradeoff** beschreibt das Spannungsfeld zwischen Modellbias (zu einfache Modelle mit hohem Fehler) und Varianz (zu komplexe Modelle mit schwacher Generalisierung).
- Ziel: Ein **Gleichgewicht zwischen Modellkomplexität und Generalisierungsfähigkeit** finden, indem man Modellarchitektur, Regularisierung und Datenmenge ausbalanciert.



overfitting



underfitting



Good balance

Wie liest man TensorBoard-Plots richtig?

- Auf die korrekte **Skalierung der Achsen** achten, um Fehlinterpretationen zu vermeiden.
- **Trendlinien** sind aussagekräftiger als einzelne Ausreißer.
- Durch den **Vergleich verschiedener Runs** können unterschiedliche **Hyperparametereinstellungen** oder Architekturen bewertet werden.
- Die Identifikation von sinnvollen **Abbruchzeitpunkten** (Early Stopping) wird erleichtert.



Code Demo: TensorBoard zur Trainingsvisualisierung

Grundlagen_Deep_Learning/Evaluation_Visualisierung/demo_tensorboard.py



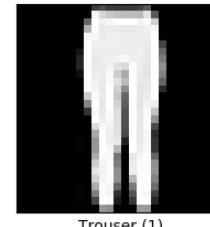
Demo: Erstellen einer Konfusionsmatrix für ein Klassifikationsmodell

Grundlagen_Deep_Learning/Evaluation_Visualisierung/
demo_konfusionsmatrix.py

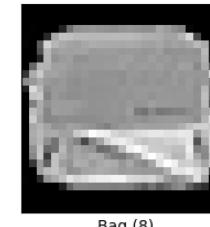
Datensatz Fashion MNIST



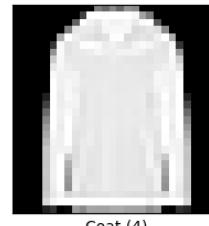
Pullover (2)



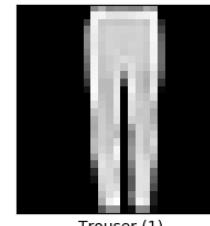
Trouser (1)



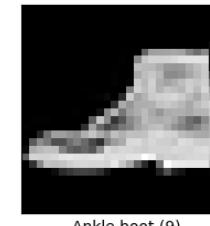
Bag (8)



Coat (4)



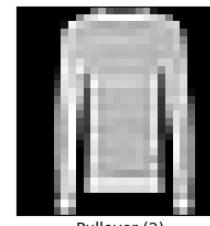
Trouser (1)



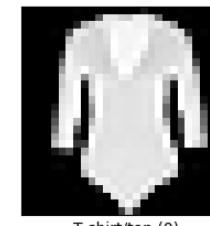
Ankle boot (9)



Pullover (2)



Pullover (2)



T-shirt/top (0)



Übung: Erstelle ein Klassifikationsmodell für Fashion MNIST.

Grundlagen_Deep_Learning/
uebung_klassifikation_fashion_mnist.py

Optimiere die Accuracy. Verfolge das Training auf Tensorboard.

Wie gut wirst du?

Agenda

1. Grundlagen des Deep Learning
- 2. Erweiterte Konzepte und Architekturen**
3. Verarbeitung von Textdaten (NLP-Grundlagen)
4. Transformer-Modelle
5. Trainingspraxis und Optimierung
6. MLOps und Deployment
7. Anwendungsfälle und Projekte

Optimizer und ADAM – Die Grundlagen

Was macht ein Optimizer?

- **Ziel:** Minimierung der **Kostenfunktion**, um ein Modell zu trainieren.
- Anpassung der **Modellgewichte** basierend auf Gradienten (Richtung des steilsten Abstiegs).
- Unterstützt das Modell, optimale Parameter für beste Vorhersagen zu finden.

ADAM Optimizer: Adaptive Moment Estimation

- **Kernidee:** Kombination von **Momentum** und **adaptiver Lernrate**.
- **Momentum:** Berücksichtigt vergangene Gradienten für beschleunigte Updates.
- **Adaptive Lernraten:** Passt die Lernrate für jedes Gewicht individuell an.

Vorteile:

- **Effiziente Konvergenz:** Stabil und schnell, selbst bei verrauschten Daten.
- **Flexibel:** Funktioniert gut mit Standard-Hyperparametern.

Merksatz: Optimizer sind die Motoren des Deep Learning, und ADAM kombiniert Geschwindigkeit mit Stabilität!

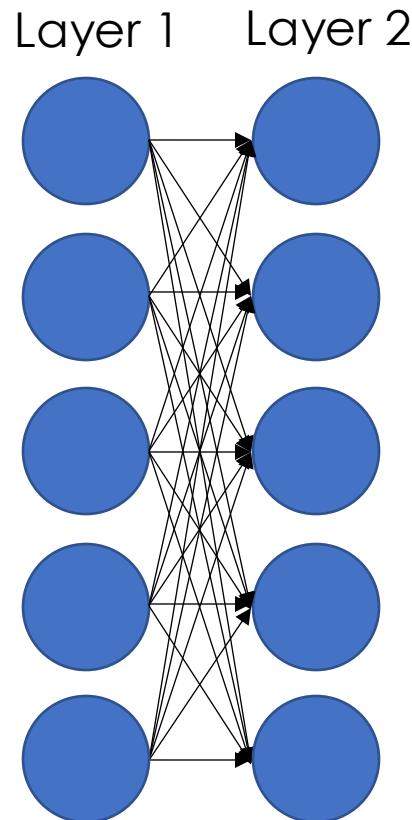


Demo: Auswirkungen unterschiedlicher Optimizer

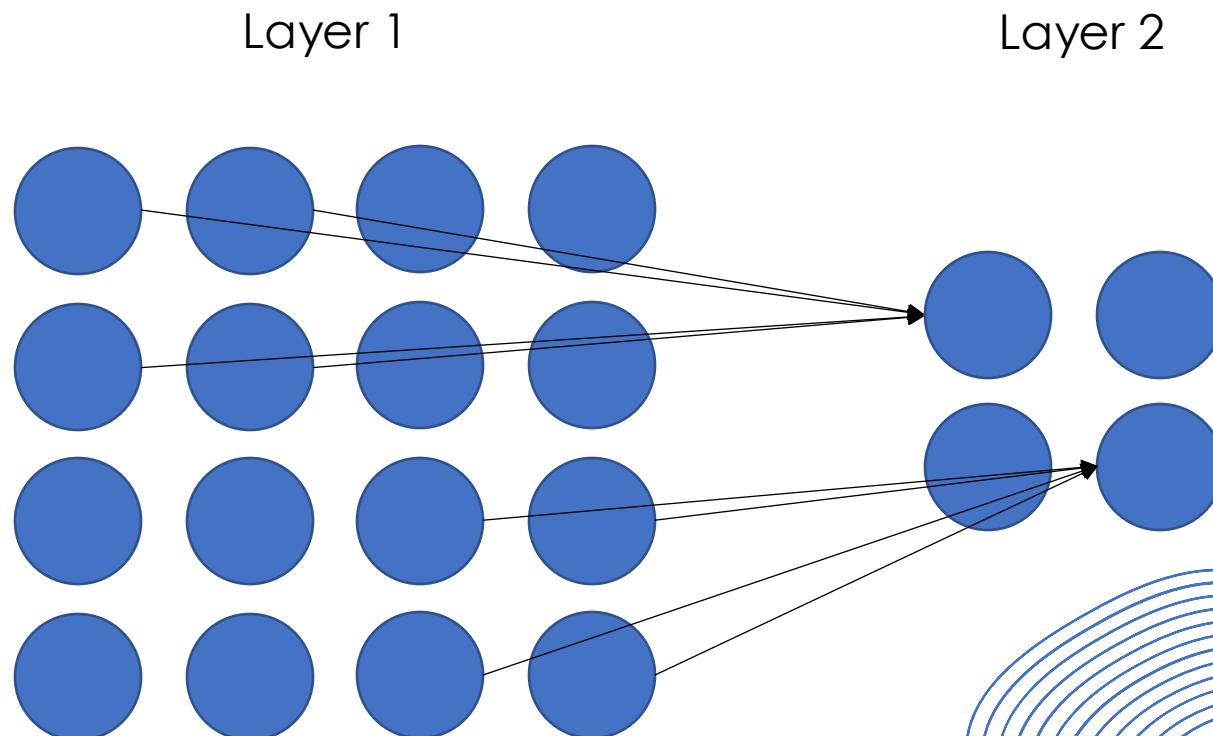
Erweiterte_Konzepte_und_Architekturen/Optimizer/demo_a
dam_optimizer.py

Architekturen – Convolutions

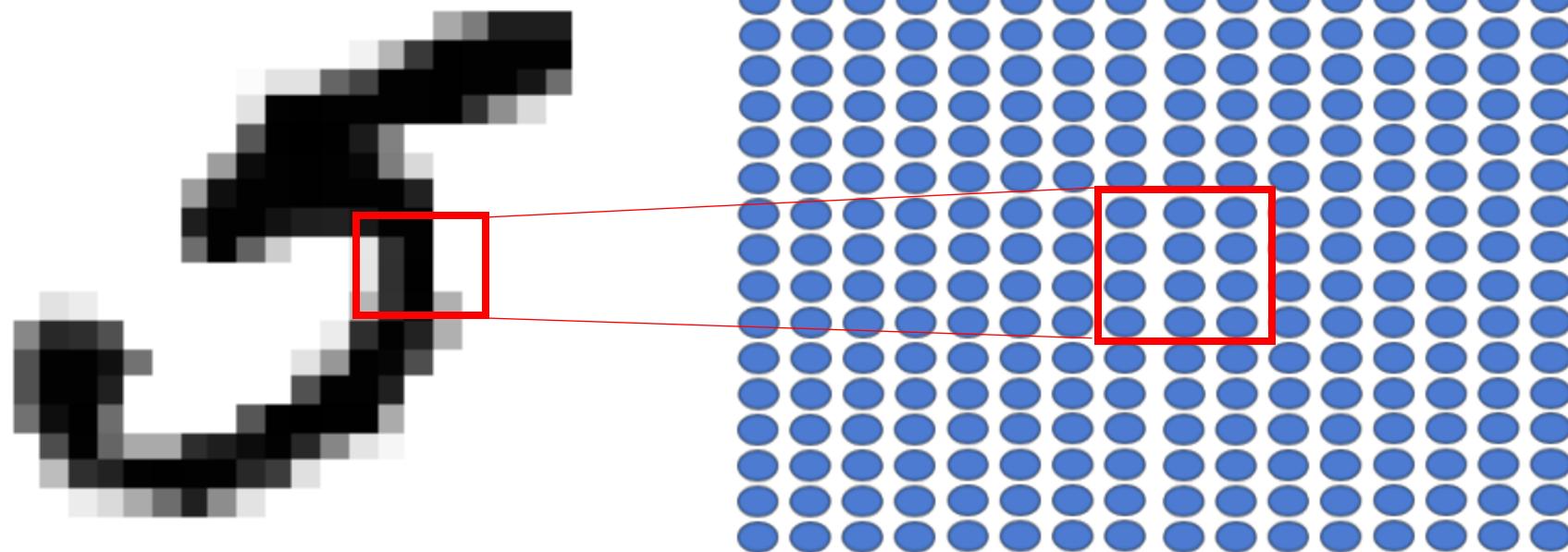
Bisher im Neuronalen Netz



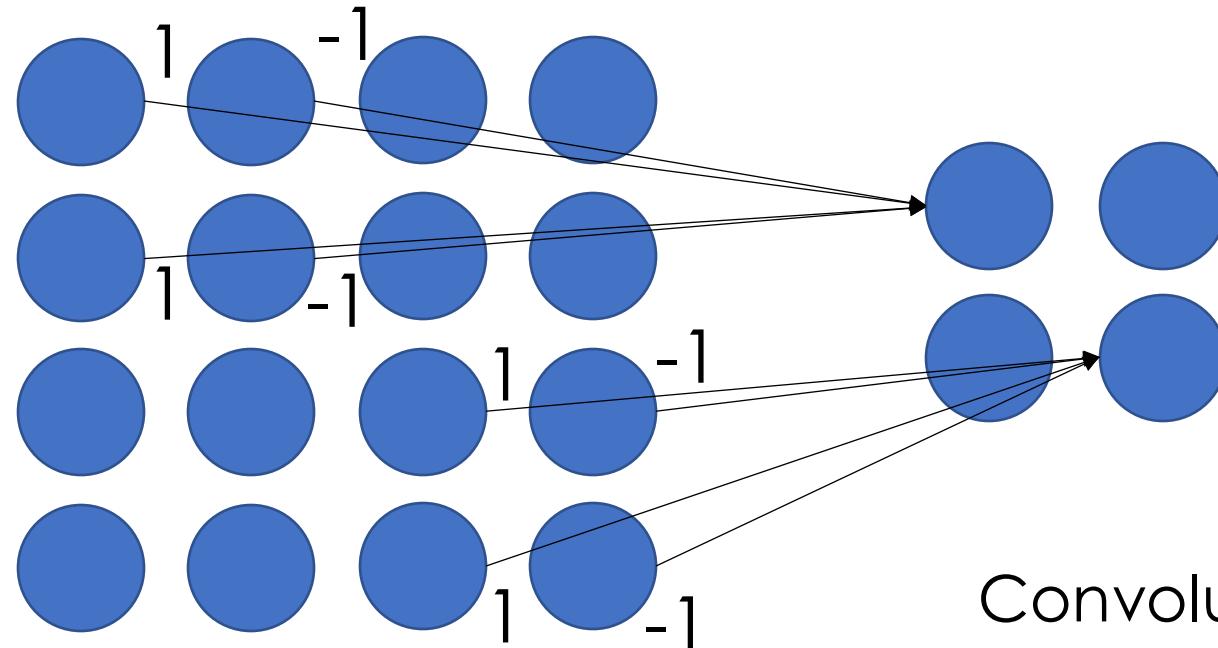
Im Convolutional Neuronal Net (CNN)



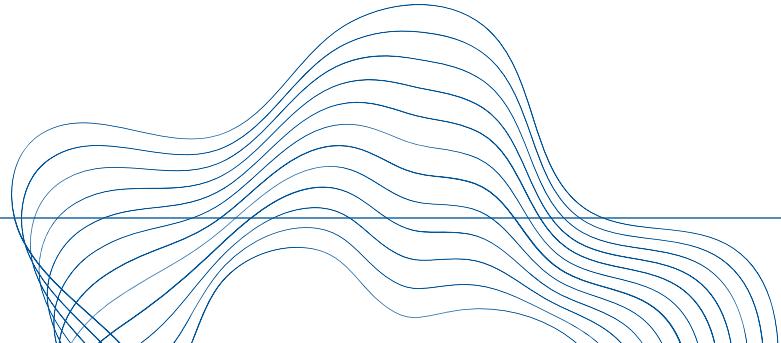
Architekturen – Convolutions



Architekturen – Convolutions als Filter

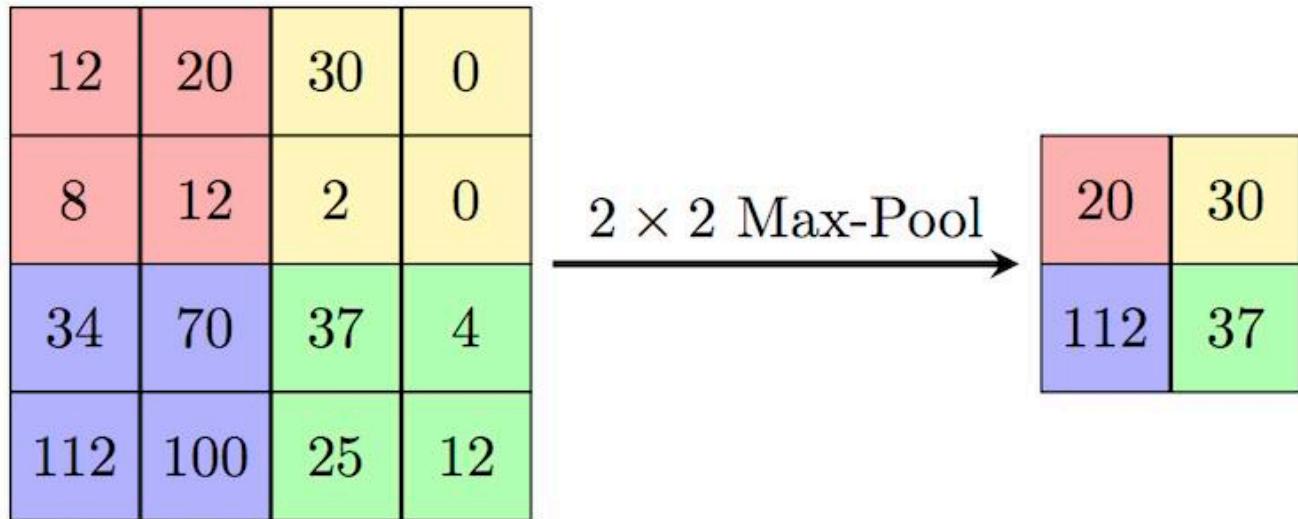


Convolutions können auch als Filter verstanden werden. In diesem Fall, um vertikale Kanten zu erkennen.

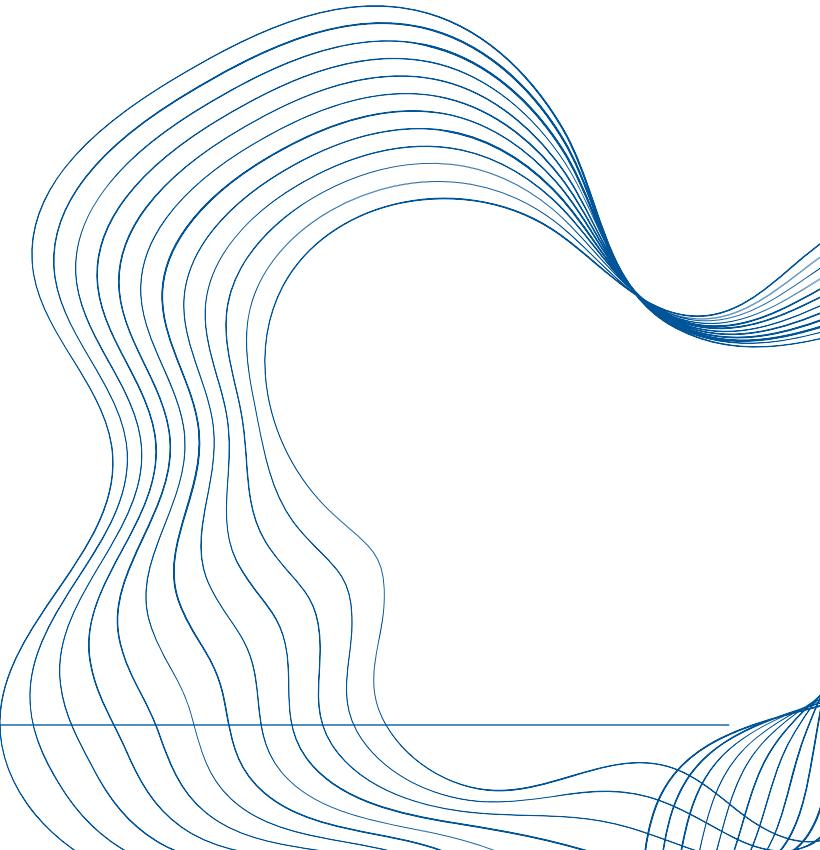


CNN – Pooling Layer

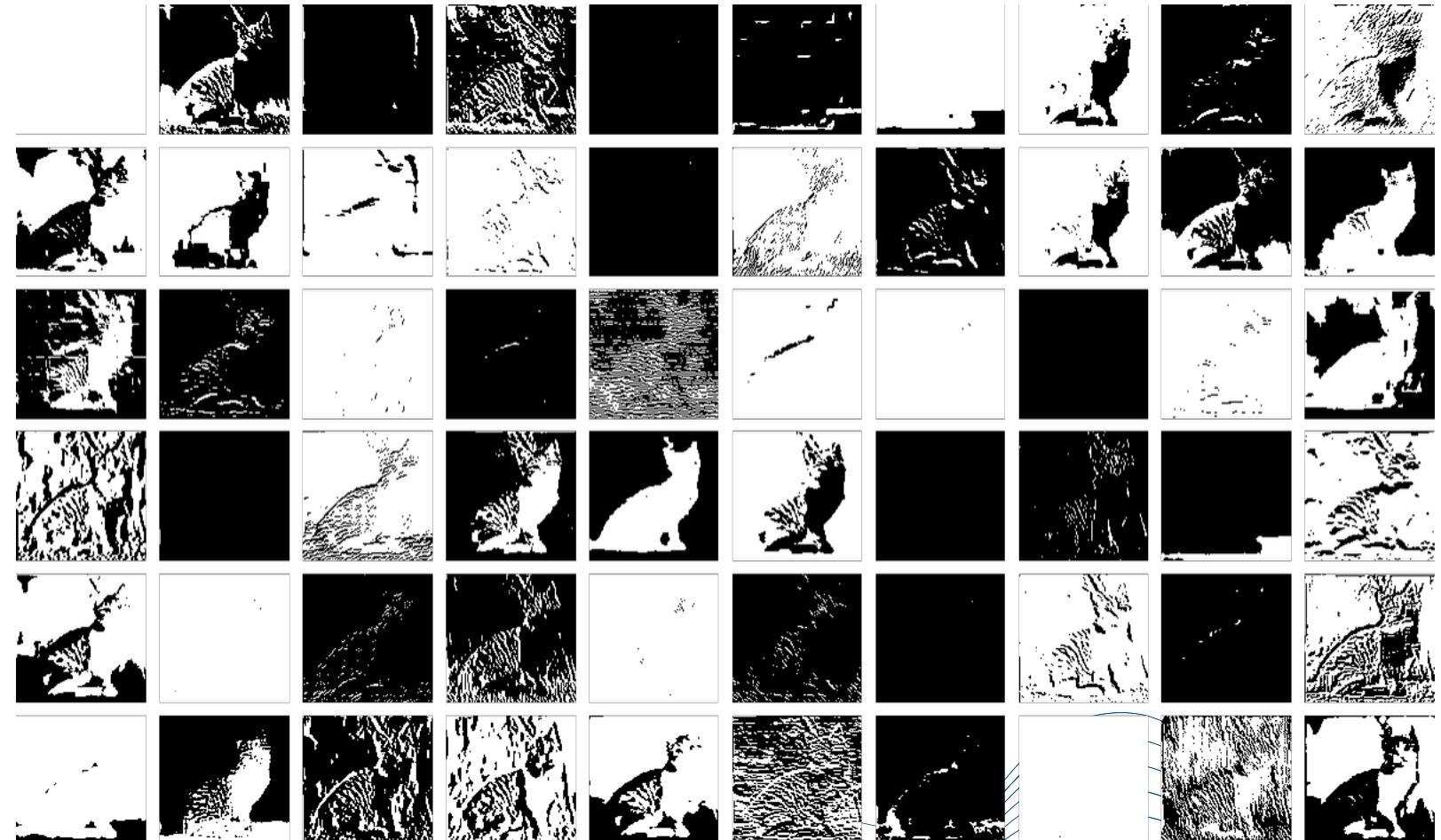
- **Pooling Layer** reduzieren die räumlichen Dimensionen, fassen Informationen zusammen und verringern damit Overfitting sowie Rechenaufwand.



<https://production-media.paperwithcode.com/methods/MaxpoolSample2.png>



Convolutional Neural Networks (CNNs)



<https://writings.stephenwolfram.com/2023/02/what-is-chatgpt-doing-and-why-does-it-work/>



CNNs visualisiert

<https://poloclub.github.io/cnn-explainer/>

CNNs: Aufbau, Convolutional Layer, Pooling Layer und typische Anwendungsbereiche

- **Convolutional Layer** extrahieren mithilfe von **Filtern lokale Muster** (Kanten, Texturen) aus Bildern und verarbeiten diese effizient.
- **Pooling Layer** reduzieren die räumlichen Dimensionen, fassen Informationen zusammen und verringern damit Overfitting sowie Rechenaufwand.
- Durch hintereinandergeschaltete Convolutionschichten und Pooling entsteht ein **hierarchisches Merkmalsmodell, das von einfachen zu immer komplexeren Mustern übergeht.**
- CNNs finden breite **Anwendung** in der Bildklassifikation, Objekterkennung, Gesichtserkennung und in der medizinischen Bildanalyse.

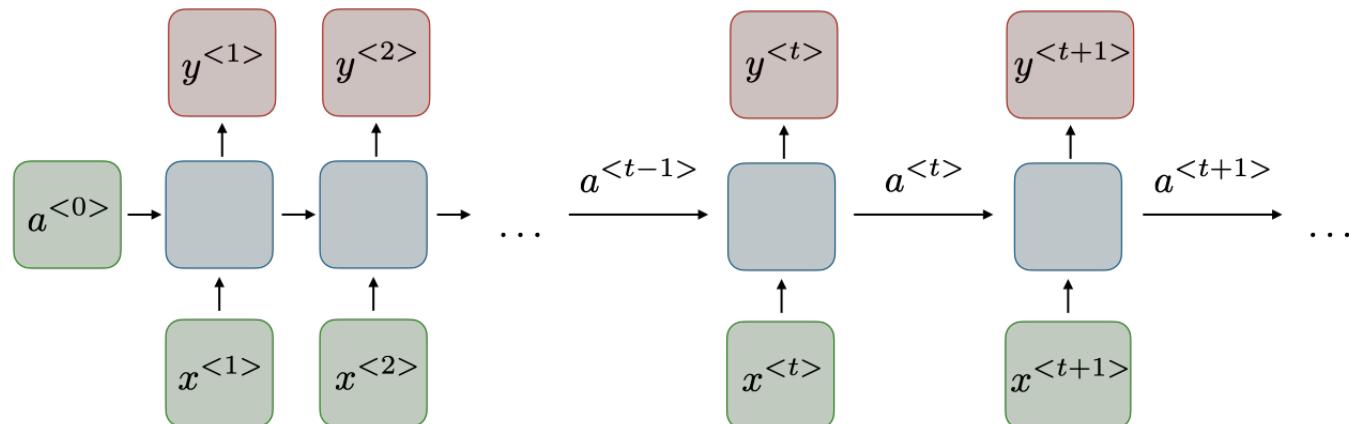


Code Demo: Einfaches CNN auf MNIST trainieren

Erweiterte_Konzepte_und_Architekturen/Erweiterte_Architekt
uren/demo_cnn_mnist.py

Zeitliche Abhängigkeiten modellieren mit Recurrent Neural Networks (RNN)

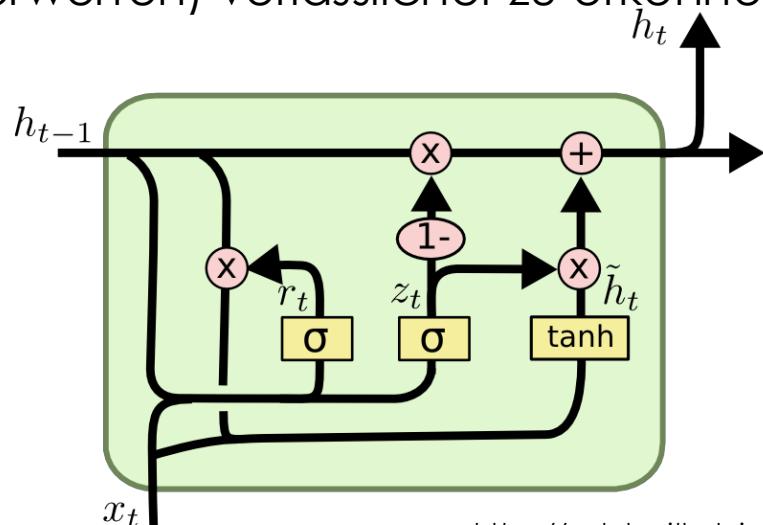
- **RNNs** (Recurrent Neural Networks) verarbeiten **Sequenzen** schrittweise, indem sie Informationen über zeitliche Zusammenhänge in einem internen Zustand speichern.
- Einfache RNNs leiden oft stark unter **Vanishing Gradients** bei langen Sequenzen.
- **LSTM** (Long Short-Term Memory) und **GRU** (Gated Recurrent Unit) sind erweiterte RNN-Varianten, die durch interne Gating-Mechanismen Langzeitabhängigkeiten besser erfassen.
- Diese Modelle wurden für **Anwendungen wie Textverarbeitung, Sprachmodellierung, maschinelle Übersetzung sowie Zeitreihenanalysen entwickelt**, sind aber heute weitestgehend von Transformer Netzwerken abgelöst.



<https://stanford.edu/~shervine/teaching/cs-230/illustrations/architecture-rnn-ltr.png?9ea4417fc145b9346a3e288801dbdfdc>

LSTMs: Detaillierter Einblick in Gating-Mechanismen

- **LSTMs** verfügen über **Input-, Forget- und Output-Gates**, um zu steuern, wie viele Informationen in den Zellzustand gelangen, wie viele vergessen werden und wie viel davon an die nächste Einheit weitergegeben wird.
- Durch den separaten Zellzustand in LSTMs wird ein expliziter **Speicher für langfristige Abhängigkeiten** bereitgestellt.
- LSTMs tragen dazu bei, Vanishing-Gradient-Probleme zu mildern und **Langzeitkontakte effektiver** nutzbar zu machen.
- Dadurch werden Modelle in der Lage, **komplexe zeitliche Muster** (z. B. in Texten, Audio, Sensorwerten) verlässlicher zu erkennen.



$$z_t = \sigma (W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma (W_r \cdot [h_{t-1}, x_t])$$

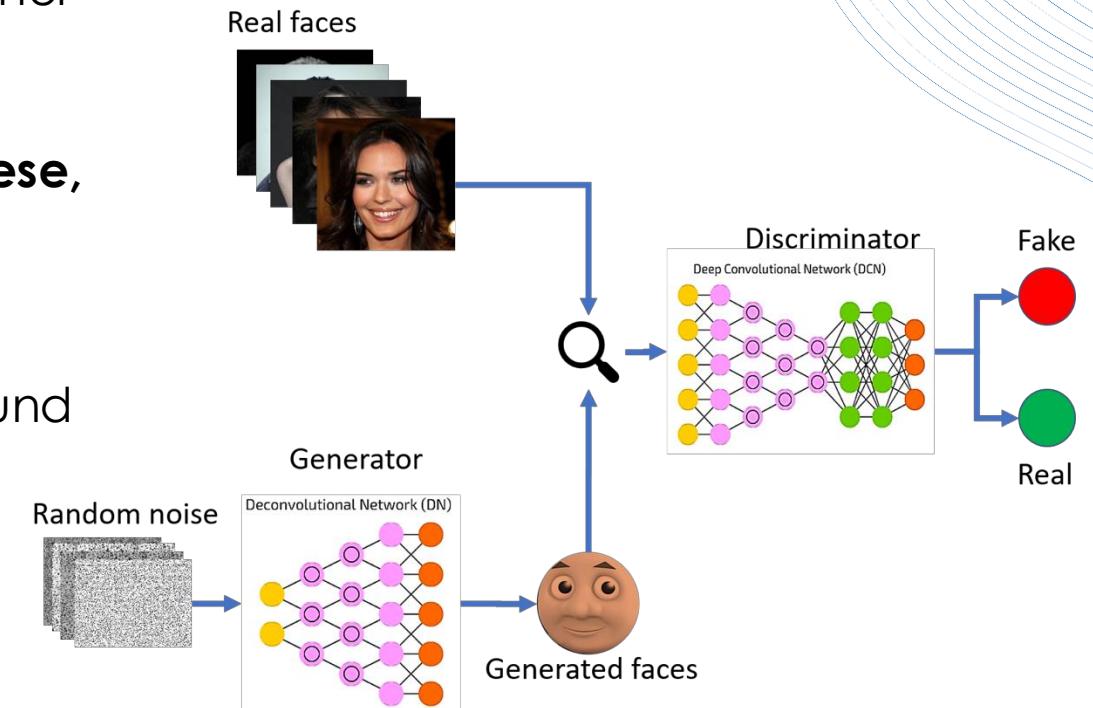
$$\tilde{h}_t = \tanh (W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

<https://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-var-GRU.png>

GANs: Generator und Discriminator im Wettstreit – Aufbau und Anwendungen

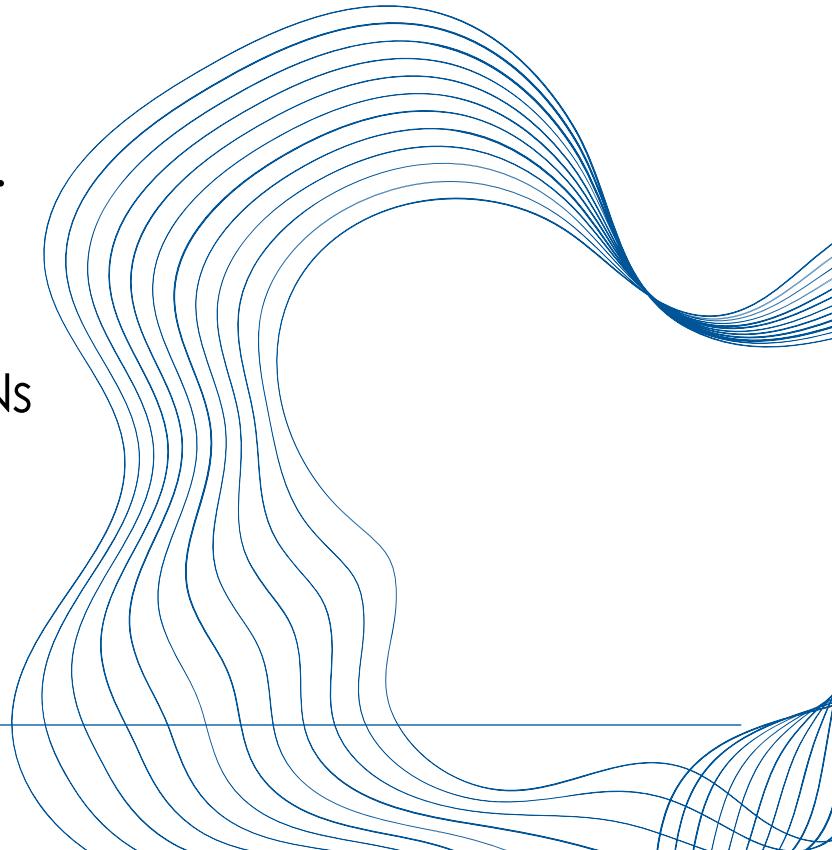
- In Generative Adversarial Networks (GANs) treten **zwei Modelle gegeneinander** an: Der Generator erzeugt künstliche Daten, der Discriminator entscheidet, ob diese echt oder künstlich sind.
- Durch den Wettstreit lernen beide Modelle: Der Generator wird immer besser im **Erzeugen** realistischer Daten, der Discriminator im **Erkennen von Fälschungen**.
- Anwendungen umfassen die **realistische Bildsynthese, Stiltransfer, das Auffüllen fehlender Bildbereiche (Inpainting) und Datenaugmentation** für knappe Datensätze.
- Herausfordernd ist dabei oft die **Trainingsstabilität** und die richtige Balance zwischen Generator und Discriminator.
- GANs sind heute weitestgehend abgelöst durch Diffusion Models.



https://d18rbf1v22mj88.cloudfront.net/wp-content/uploads/sites/3/2018/03/29200233/GAN_en.png

Typische Anwendungsfälle für jede Architektur

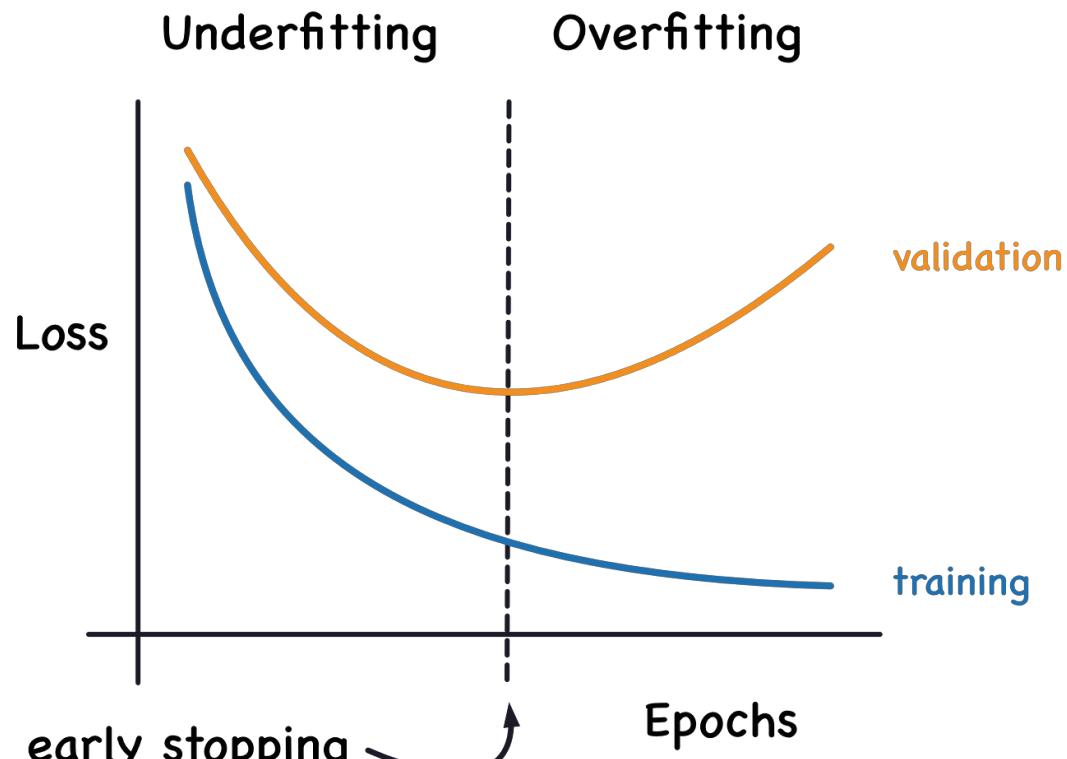
- **CNNs** eignen sich für Bildklassifikation, Objektdetektion, Segmentierung und visuelle Mustererkennung.
- **RNN/LSTM/GRU-Modelle** sind besonders effektiv für Aufgaben mit zeitlichen Abhängigkeiten, wie Sprachmodellierung, Übersetzung, Chatbots, Stimmungsanalyse und Zeitreihenprognosen.
- **GANs** werden häufig zur Bildgenerierung, Text-zu-Bild-Synthese, Stiltransfer und Datenaugmentation verwendet.
- Die Wahl der Architektur hängt eng von den Datenarten und Anwendungszwecken ab.
- **Hybride Ansätze** kombinieren gelegentlich CNNs und RNNs (z. B. für Videoanalyse) oder nutzen GANs für Datenaufbereitung.



Code Demo: RNN-basierte Textklassifikation mit LSTM

Erweiterte_Konzepte_und_Architekturen/Erweiterte_Architekt
uren/demo_lstm_textklassifikation.py

Frühzeitiges Abbrechen des Trainings: Wann und warum?



- **Early Stopping** beendet das Training bevor sich das Modell zu stark an die Trainingsdaten anpasst, um **Overfitting zu vermeiden**.
- Durch rechtzeitiges Abbrechen **spart man Rechenzeit und Ressourcen**, da unnötig lange Trainingsphasen vermieden werden.
- Die **Generalisierungsfähigkeit wird maximiert**, da das Modell an einem Punkt mit guter Balance zwischen Bias und Varianz stoppt.
- Early Stopping ist damit ein einfacher und effektiver Ansatz, um die **Trainingszeit zu verkürzen und gleichzeitig die Modellleistung auf Validierungsdaten zu verbessern**.

Beispielhafter Einsatz von Early Stopping und Einfluss auf Generalisierung

- Ein typisches Kriterium ist der **Validierungsfehler**: Steigt er über mehrere Epochen an, wird das Training frühzeitig beendet.
- Dadurch bleibt das Modell näher an einem idealen Punkt, an dem es noch nicht überangepasst ist.
- Das Ergebnis sind **stabilere und robustere Vorhersagen auf unbekannten Datensätzen**.
- Early Stopping ist ein **verbreiteter Standard** in der Praxis und kann oft ohne aufwendiges Fine-Tuning eingesetzt werden.
- Letztlich führt es zu Modellen, die im Schnitt **bessere Generalisierung** zeigen, ohne unnötig lange Trainingszeiten.



Code Demo: Early Stopping in Keras implementieren

Erweiterte_Konzepte_und_Architekturen/Early_Stopping/de
mo_early_stopping.py

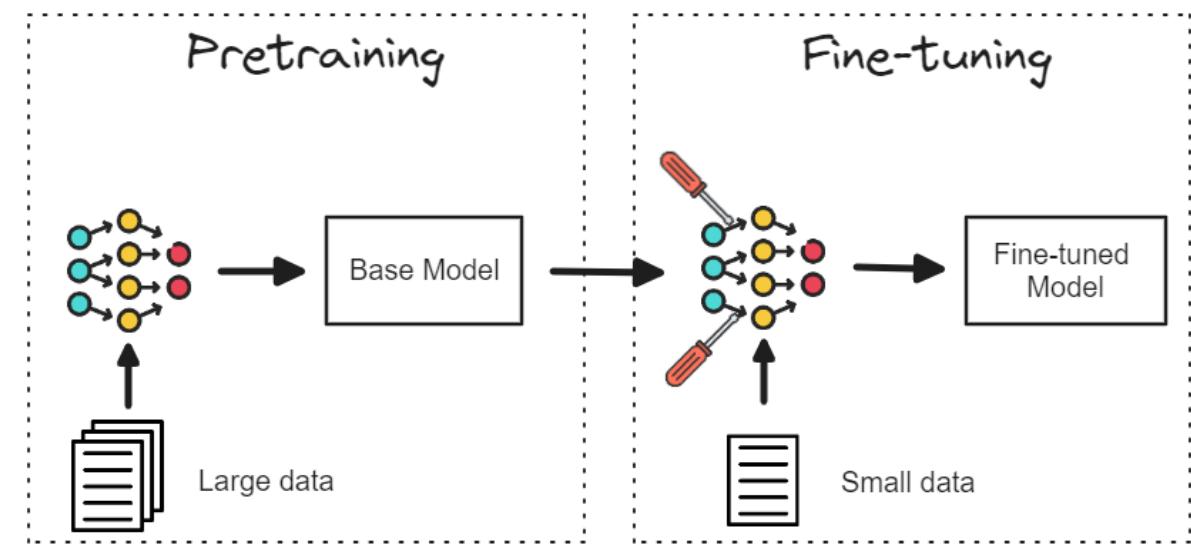


Übung: Optimiere die Klassifikation des Fashion MNIST Datensatz

Nutze bessere Optimierer, CNNs und Early Stopping.
Wie gut wirst du?

Wiederverwendung vortrainierter Modelle: Idee, Vorteile und Grenzen

- Transfer Learning nutzt bereits erlernte Merkmalsrepräsentationen aus einem vortrainierten Modell, um neue Aufgaben **schneller** und mit **weniger Daten** anzugehen.
- **Vorteile:** Schnellere Konvergenz, geringerer Bedarf an umfangreichen Trainingsdaten, Nutzung etablierter Architekturen.
- **Grenzen:** Falls sich die Zielaufgabe stark von der ursprünglichen Domäne unterscheidet, sinkt der Nutzen vortrainierter Gewichte.
- Trotz dieser Grenzen ist Transfer Learning ein äußerst effizienter und verbreiteter Ansatz in der Praxis.

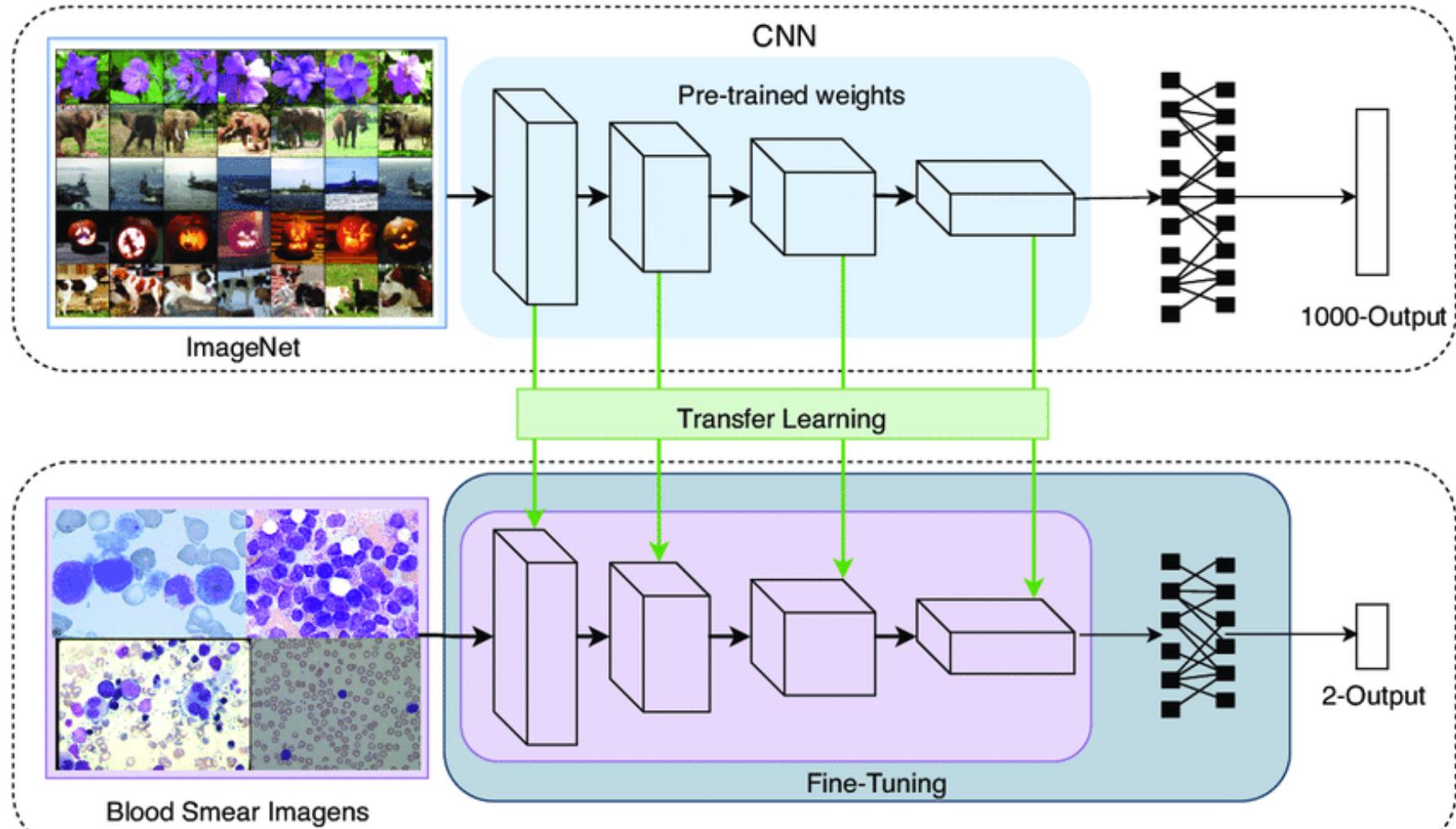


https://miro.medium.com/v2/resize:fit:1400/1*J058x7lmBME4fDT7V-rrZg.png

Fine-Tuning einer vortrainierten Basis: Wann lohnt es sich?

- Fine-Tuning lohnt sich, wenn bereits ein **ähnliches Problem** erfolgreich gelöst wurde und die dafür trainierten Features als Ausgangspunkt dienen.
- Es **spart erhebliche Ressourcen und Zeit**, da Teile des Modells schon ‚vorbereitet‘ sind.
- **Feinabstimmung der oberen Schichten** an die neue Zielaufgabe führt häufig zu deutlich besseren Ergebnissen als ein Training von Grund auf.
- Insbesondere bei **begrenzten Trainingsdaten** ist Fine-Tuning ein wertvolles Werkzeug.

Finetuning mit CNNs



<https://www.researchgate.net/publication/351085550/figure/fig3/AS:1016156517310464@1619282001253/The-transfer-learning-and-fine-tuning-techniques-used-in-the-development-of-the-proposed.png>

Beispiel: Vortrainiertes CNN für Bilderkennung auf neuem Datensatz

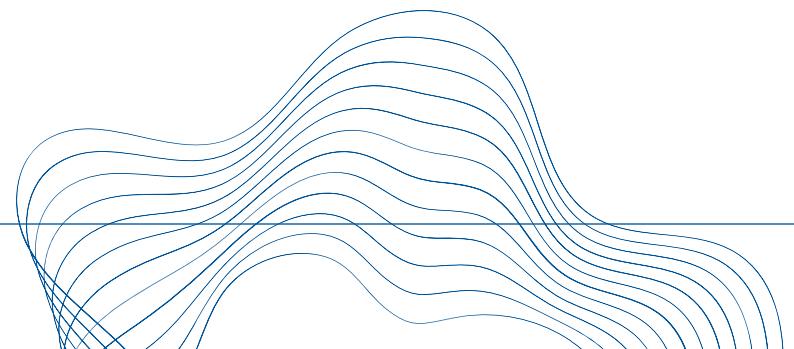
- Nutzung einer bekannten CNN-Architektur (z. B. VGG, ResNet), die bereits auf großen Bilddatensätzen (z. B. ImageNet) trainiert wurde.
- Die **unteren Schichten extrahieren generische Bildmerkmale**, die für viele visuelle Aufgaben nützlich sind.
- Nur die **oberen Schichten werden neu trainiert** oder feinjustiert, um an die speziellen Klassen des neuen Datensatzes anzupassen.
- Ergebnis: Gute Performance mit vergleichsweise geringem Aufwand und weniger benötigten Daten.
- Dieser Ansatz ist in vielen Industrieprojekten Standard und erlaubt schnelle Prototypenentwicklung.

Best Practices zur Anpassung von Hyperparametern beim Transfer Learning

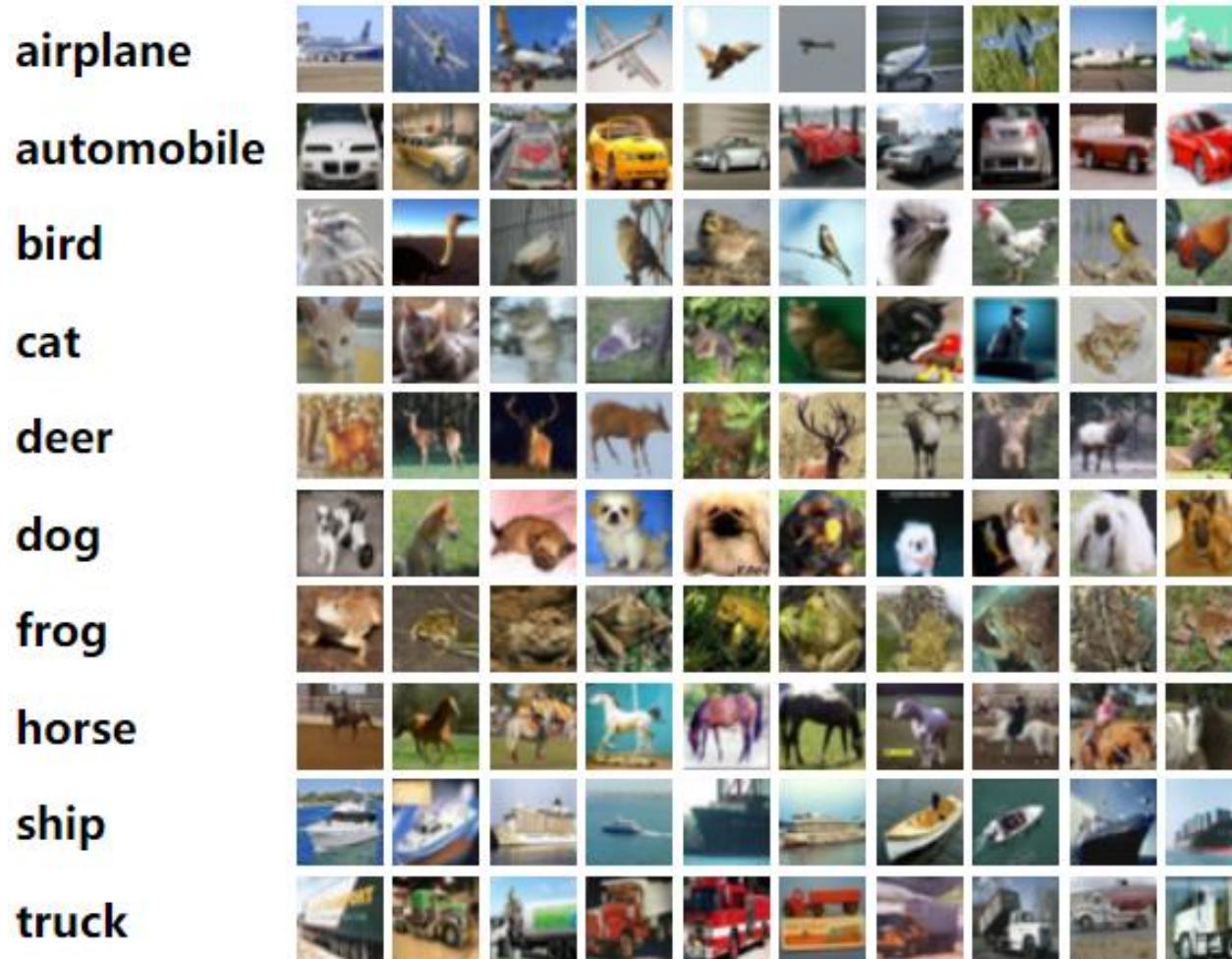
- Reduzieren Sie die **Lernrate** für vortrainierte Schichten, um bereits gut angepasste Gewichte nicht zu stark zu verändern.
- **Entsperrn Sie nur schrittweise weitere Schichten** für das Fine-Tuning, um Instabilitäten zu vermeiden.
- Nutzen Sie frühzeitiges **Monitoring** von Validierungsmetriken, um rechtzeitig zu erkennen, wann das Fine-Tuning abgeschlossen sein könnte.
- Passen Sie Batch-Größe, Optimierer und ggf. Regularisierungsmechanismen an, um optimale Ergebnisse zu erzielen.

Anwendungsfälle aus Industrie und Forschung

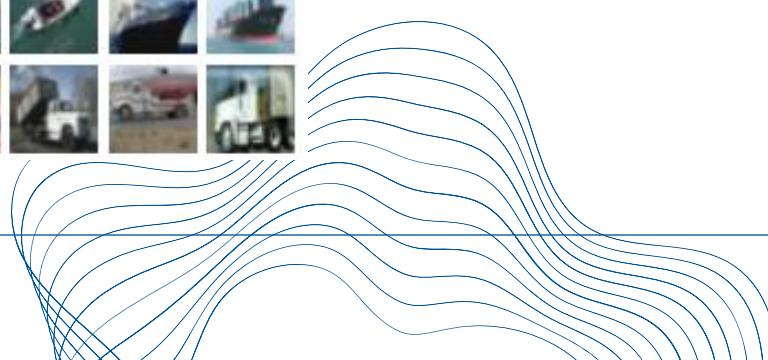
- **Medizinische Bildanalyse:** Vortrainierte CNNs auf allgemein verfügbaren Bildern, anschließend Fine-Tuning auf spezifische medizinische Scans.
- **Sprachmodelle wie BERT** können für Fachdomänen (z. B. juristische oder medizinische Texte) weitertrainiert und spezialisiert werden.
- Transfer Learning beschleunigt die Entwicklung neuer KI-Anwendungen, indem es bereits vorhandenes Wissen effektiv weiterverwendet.
- Sowohl in der Forschung als auch in der Industrie ist Transfer Learning ein wichtiger **Beschleuniger für erfolgreiche KI-Projekte**.



Cifar-10 Datensatz



<https://production-media.paperswithcode.com/datasets/4fd12b82-2bc3-4f97-ba51-400322b228b1.png>





Code Demo: Fine-Tuning eines vortrainierten CNN auf Custom-Dataset

Erweiterte_Konzepte_und_Architekturen/Transfer_Learning/d
emo_fine_tuning_cnn.py

Übung Fine Tuning

Analyse der Ergebnisse: Welche Klassen werden schlecht vorhergesagt? Gibt es Klassen, die oft verwechselt werden?



Übung: Erstelle eine Klassifikation für Wettertage

Nutze alle Möglichkeiten, die du kennst. Wie gut wirst du?
Erweiterte_Konzepte_und_Architekturen/uebung_classification_weather.py

Agenda

1. Grundlagen des Deep Learning
2. Erweiterte Konzepte und Architekturen
- 3. Verarbeitung von Textdaten (NLP-Grundlagen)**
4. Transformer-Modelle
5. Trainingspraxis und Optimierung
6. MLOps und Deployment
7. Anwendungsfälle und Projekte

Bereinigung (Lowercasing, Entfernen von Sonderzeichen, HTML-Tags)

- Konvertierung aller Texte in Kleinbuchstaben sorgt für **einheitliche und vergleichbare Eingaben**, verringert Fehlklassifikationen durch Groß- und Kleinschreibung.
- Entfernung störender **HTML-Tags** und Formatierungselemente, um reinen Text zu erhalten und Interpretationsfehler durch **Markup** zu vermeiden.
- Beseitigung unerwünschter **Sonderzeichen** und **irrelevanter Symbole** reduziert Störgeräusche im Korpus.
- Diese Schritte legen den Grundstein für einen sauberen und konsistenten Dateninput, der nachfolgende Verarbeitungsschritte vereinfacht.

Tokenisierung: Wort- und Subwortverfahren: BPE, WordPiece

- Die Aufspaltung in **Tokens** (Wörter oder Subwörter) bildet die Grundlage für alle weiteren NLP-Analysen.
- **Subwortverfahren** wie Byte-Pair Encoding (BPE) oder WordPiece **reduzieren das Problem unbekannter Wörter** (OOV), indem sie seltene oder komplexe Wörter in häufig vorkommende Bestandteile zerlegen.
- Dadurch entsteht eine **flexiblere Repräsentation des Vokabulars**, die es ermöglicht, mit **deutlich weniger Wortformen** auszukommen.
- Diese Techniken bilden die Basis für **moderne Embedding-Modelle** wie BERT, die kontextabhängige Bedeutungen besser erfassen.

“Das ist ein Beispiel” → [“Das”, “ist”, “ein”, “Bei”, “spiel”]

Lemmatisierung, Stemming: Wann anwenden?

- **Lemmatisierung** reduziert Wörter auf ihre **Grundform** (Lemma), ohne dabei die eigentliche Wortbedeutung zu verlieren. Beispielsweise wird „gelaufen“ auf „laufen“ zurückgeführt.
- **Stemming** verkürzt Wörter auf ihre **Wortstämme**, ohne dabei auf grammatische Korrektheit oder vollständige Semantik zu achten. Dies ist eine schnellere, aber oft weniger präzise Methode. Beispielsweise wird „runner“ und „running“ auf „runn“ zurückgeführt.
- Beide Verfahren **reduzieren die Formenvielfalt**, erleichtern statistische Analysen und verbessern die Generalisierung insbesondere bei **klassischen NLP-Modellen**.
- Lemmatisierung ist sinnvoll, wenn genaue Wortbedeutungen wichtig sind, während Stemming für schnelle, grobe Normalisierungen geeignet ist.

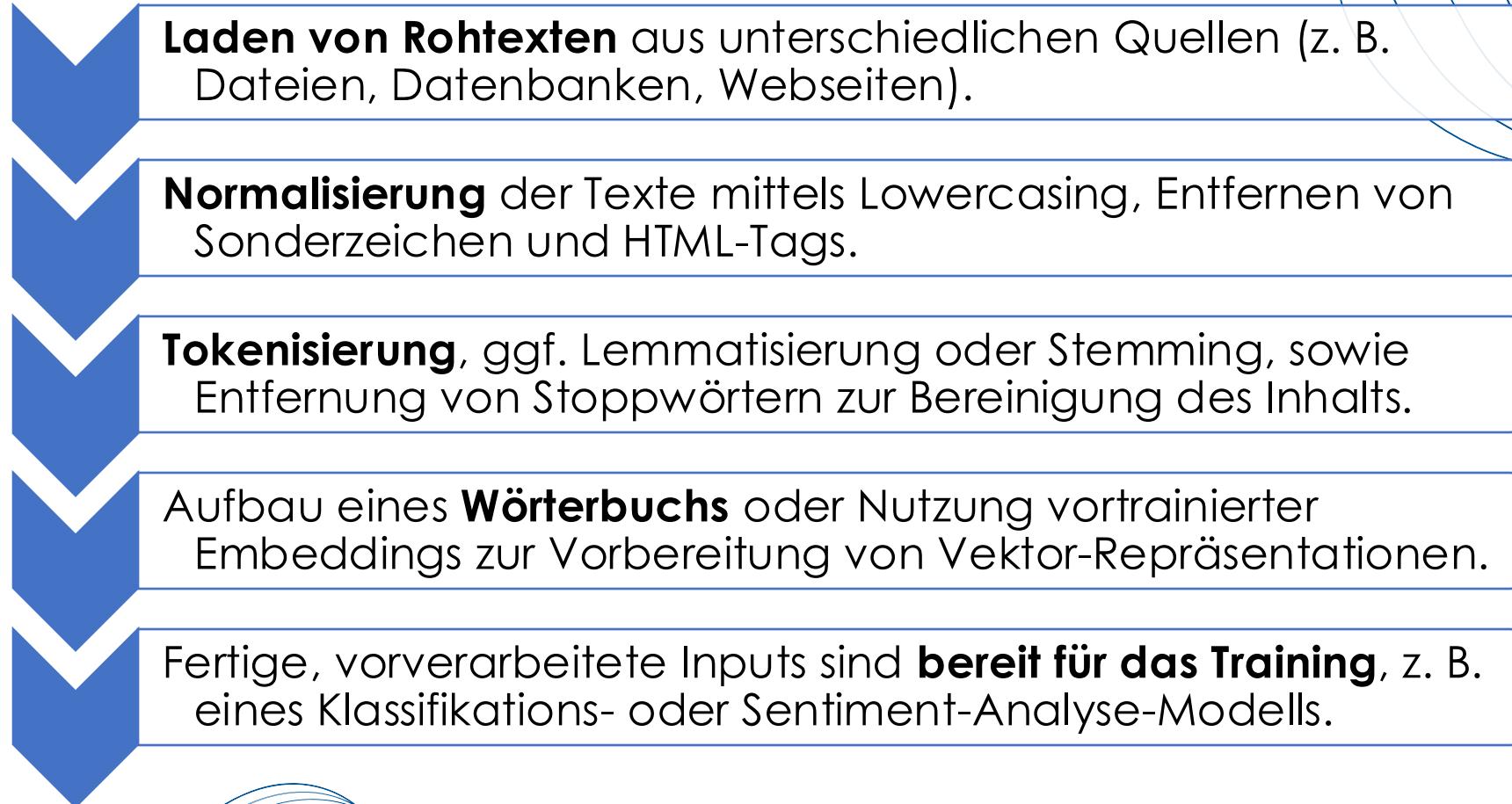
Padding und Masking bei variablen Sequenzlängen

- **Padding** fügt künstliche Platzhalter (spezielle Tokens) hinzu, um alle Sequenzen auf eine einheitliche Länge zu bringen, was die Batch-Verarbeitung vereinfacht.
- **Masking** markiert gepaddete Bereiche, sodass das Modell diese ignorieren kann und nicht fälschlich versucht, aus den Platzhalter-Tokens zu lernen.
- Beide Techniken sind essentiell, um **effizient mit Sequenzen unterschiedlicher Länge umzugehen**, besonders bei Modellen wie RNNs, LSTMs oder Transformern.
- Dadurch werden Trainingsprozesse standardisiert, robust und vergleichbar.

Beispiel:

- Eingabe: ["Hello", "world", "<PAD>", "<PAD>"]
- Maske: [1, 1, 0, 0]

Praxisbeispiel: Preprocessing-Pipeline für einen Beispieldoktorpus





Code Demo: Tokenisierung und Preprocessing mit Python- Tools

Verarbeitung_von_Textdaten_NLP_Grundlagen/Preprocessin
g/demo_tokenisierung.py



Übung: Erstelle eine Wordcloud von einer Webseite

Nutzt `requests.get()` um den Inhalt einer beliebigen Seite zu laden.

Nutze sinnvolle Preprocessing Steps.

`Verarbeitung_von_Textdaten_NLP_Grundlagen/Preprocessin
g/uebung_wordcloud.py`

Bag-of-Words, TF-IDF, n-Gramme: Klassische Repräsentationen

- **Bag-of-Words** (BoW) zählt die Häufigkeit von Wörtern ohne Rücksicht auf Reihenfolge oder Kontext. Einfach, aber kontextlos.
- **TF-IDF** (Term Frequency - Inverse Document Frequency) gewichtet seltene, aber aussagekräftige Wörter höher und reduziert den Einfluss von häufigen, aber wenig informativen Begriffen.
- **N-Gramme** (z. B. Bigramme, Trigramme) bilden Folgeeinheiten von mehreren Wörtern und berücksichtigen damit rudimentär die Wortreihenfolge.
- Diese klassischen Repräsentationen bilden die Grundlage vieler älterer NLP-Verfahren, sind jedoch **in ihrer Aussagekraft begrenzt**.

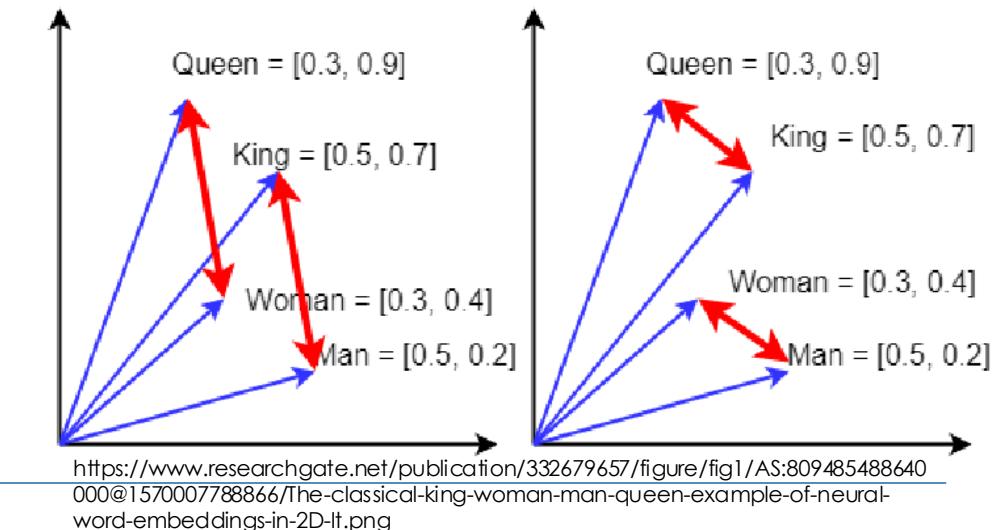
Übung: Ersetze das BOW Embedding durch ein TF-IDF Embedding

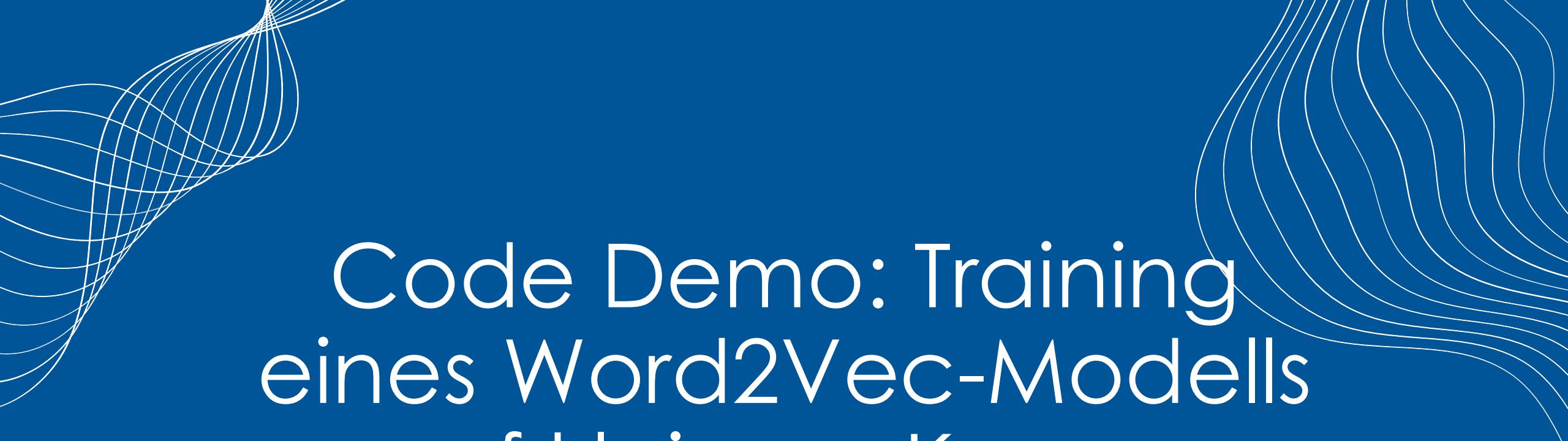
Verarbeitung_von_Textdaten_NLP_Grundlagen/Preprocessin
g/demo_tokenisierung.py

Hinweis: kurze Recherche für Verwendung von sklearn
preprocessing TF IDF notwendig.

Statische Wort-Embeddings (Word2Vec, GloVe): Idee, Training, Anwendung

- Statische Wort-Embeddings projizieren jedes Wort in einen kontinuierlichen Vektorraum, in dem **semantisch ähnliche Wörter nah beieinander liegen**.
- **Word2Vec** und **GloVe** lernen solche Vektoren aus großen Korpora, indem sie den **Kontext eines Wortes berücksichtigen, jedoch ohne Wortbedeutungen je nach Satzumgebung** dynamisch anzupassen.
- Diese vortrainierten Embeddings verbessern viele NLP-Aufgaben, wie Klassifikation oder Clustering, da sie semantisches Wissen bereitstellen.
- Allerdings bleiben **Wortbedeutungen für ein Wort immer gleich**, unabhängig vom Kontext.



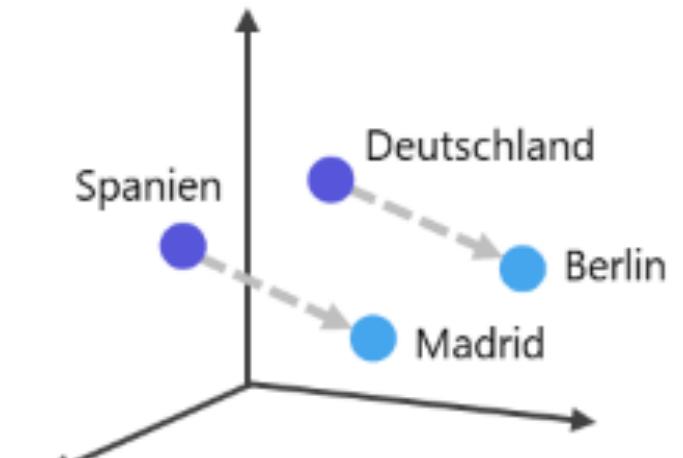


Code Demo: Training eines Word2Vec-Modells auf kleinem Korpus

Verarbeitung_von_Textdaten_NLP_Grundlagen/Textrepraes
entationen/demo_word2vec_training.py

Kontextabhängige Embeddings (BERT, GPT): Neue Ära der Sprachrepräsentation

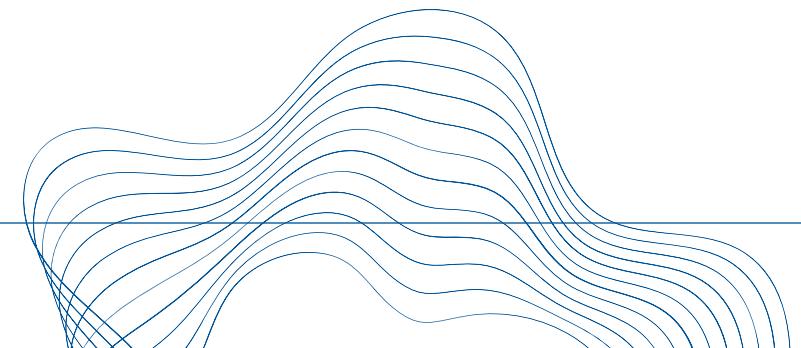
- **Kontextabhängige Embeddings** wie BERT oder GPT generieren **unterschiedliche Wortvektoren je nach Satzkontext** und erfassen damit feinere Bedeutungsnuancen.
- Die Transformer-Architektur ermöglicht es, **globale Kontextinformationen über Self-Attention-Mechanismen** zu berücksichtigen.
- Diese Modelle liefern **erheblich bessere Ergebnisse in komplexen NLP-Aufgaben**, wie Fragebeantwortung, Sentiment-Analyse oder Named Entity Recognition.
- Sie markieren einen Paradigmenwechsel, der statische Embeddings in vielen Bereichen ablöst.



[https://www.plusmeta.de/static/ac896624d937fc9a2b48e0b2eb117463/242a6/embeddings-werden-für-nlp-im-deep-learning-verwendet.png](https://www.plusmeta.de/static/ac896624d937fc9a2b48e0b2eb117463/242a6/embeddings-werden-fuer-nlp-im-deep-learning-verwendet.png)

Vergleich statischer und kontextabhängiger Embeddings an Beispielen

- **Statische Embeddings:** Das Wort „Bank“ erhält immer denselben Vektor, egal ob es um eine Sitzbank oder eine Finanzbank geht.
- **Kontextabhängige Embeddings:** Die Vektordarstellung von „Bank“ unterscheidet sich je nach Kontext („sich auf eine Bank setzen“ vs. „Geld von der Bank abheben“).
- Kontextabhängige Repräsentationen helfen bei polysemen Wörtern, Mehrdeutigkeiten besser aufzulösen.
- Als Folge steigen Genauigkeit, Robustheit und Nuanciertheit in einer Vielzahl von NLP-Aufgaben.





Demo: Einbinden von vortrainierten Embeddings in ein Klassifikationsmodell

Verarbeitung_von_Textdaten_NLP_Grundlagen/
Textrepräsentationen/demo_embeddings_integration.py



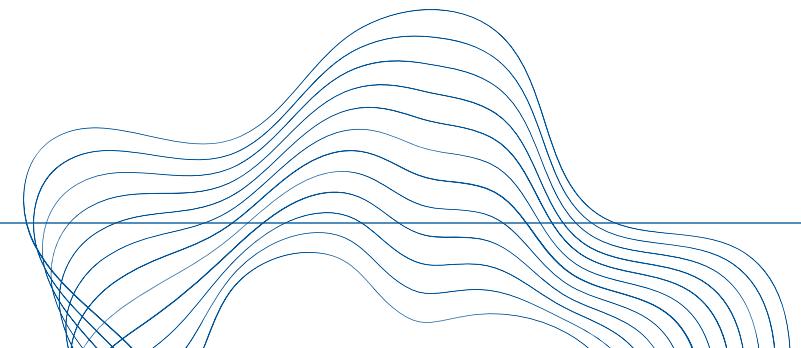
Übung / Diskussion: Wie
kann die Klassifikation
weiter verbessert werden?

Umgang mit mehrsprachigen Datensätzen: Zeichensätze, Tokenisierung

- Unterschiedliche Sprachen nutzen verschiedene **Schriftsysteme** (z. B. Lateinisch, Kyrillisch, Chinesisch), was Anpassungen bei Encoding und Normalisierung erfordert.
- Einheitliche Tokenisierung **über Sprachgrenzen hinweg ist herausfordernd**, da Sprachen unterschiedliche Wortbildungsregeln aufweisen.
- Gemeinsame Vokabulare oder **Subwortverfahren helfen**, mehrsprachige Modelle zu erstellen, die auf mehrere Sprachen gleichzeitig angewendet werden können.
- Wichtig für globale Anwendungen wie internationale Chatbots oder Übersetzungssysteme.

Mehrsprachige Embeddings (MUSE, XLM-R)

- Mehrsprachige Embeddings projizieren Wörter aus verschiedenen Sprachen in einen **gemeinsamen Vektorraum**, um semantische Ähnlichkeiten sprachübergreifend nutzbar zu machen.
- MUSE und XLM-R ermöglichen **direkten Vergleich** von Bedeutungen zwischen Sprachen, **ohne aufwändige Übersetzungsressourcen**.
- Diese Modelle erleichtern Cross-Lingual Transfer Learning, sodass ein auf Englisch trainiertes Modell auch für Deutsch oder Französisch nützlich sein kann.
- Ein wichtiger Schritt hin zu universellen Sprachmodellen für den internationalen Einsatz.



POS-Tagging, Dependency Parsing: Extraktion grammatischer Informationen

- **POS-Tagging** (Part-of-Speech-Tagging) weist jedem Wort seine Wortart (Verb, Nomen, Adjektiv usw.) zu, um syntaktische Strukturen besser zu verstehen.
- Diese Verfahren liefern **wertvolle syntaktische Informationen für komplexere Aufgaben** wie Textverständnis, Übersetzung oder Fragebeantwortung.
- Sie bilden eine Grundlage für tieferes linguistisches Verständnis.



https://miro.medium.com/v2/resize:fit:1376/1*iJfCrgV28T8HJXg1CjGziA.png

Tooling: SpaCy, NLTK, Stanza – Vor- und Nachteile

- **SpaCy** bietet schnelle, produktionsreife Pipelines für Tokenisierung, POS-Tagging und Parsing, mit Fokus auf einfache Integrationen.
- **NLTK** ist sehr umfangreich und stark im **akademischen Kontext**, jedoch oft langsamer und komplexer in der Handhabung.
- **Stanza** (von der Stanford NLP Group) nutzt moderne, **neuronale Modelle** und deckt mehrere Sprachen ab.

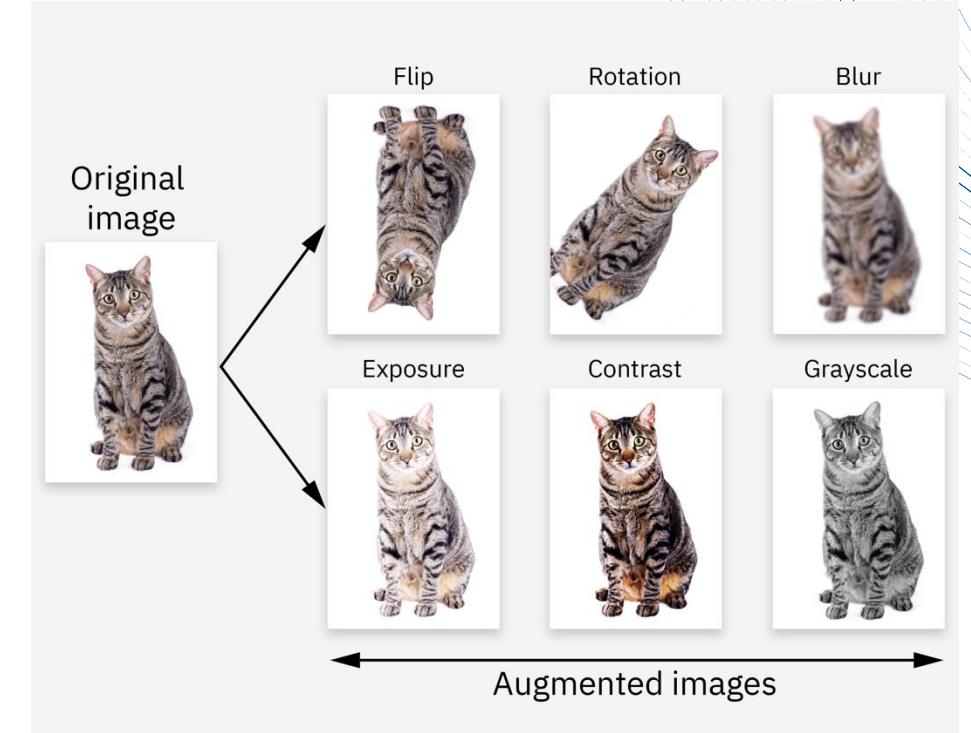
The SpaCy logo consists of the word "spaCy" in a bold, blue, sans-serif font. Behind the text are several thin, blue, wavy lines that curve upwards and outwards from the right side, creating a dynamic, flowing effect.The Stanza logo features a red feather icon followed by the word "Stanza" in a large, black, sans-serif font. The feather is positioned to the left of the text, pointing towards it.

Code Demo: POS- Tagging mit SpaCy

Verarbeitung_von_Textdaten_NLP_Grundlagen/POS/demo_
spacy_pos_tagging.py

Notwendigkeit und Ziele von Datenaugmentation

- Datenaugmentation **erweitert den Trainingsdatensatz** künstlich, ohne zusätzliche manuelle Annotationskosten.
- Sie erhöht die Datenvielfalt und **Robustheit eines Modells**, indem es unterschiedlich formulierte, aber semantisch ähnliche Trainingsbeispiele erhält.
- Dies hilft, **Overfitting zu reduzieren**, da das Modell nicht nur auf wenige starre Formulierungen angepasst wird.
- Ziel ist eine **bessere Generalisierung** auf neue, ungewohnte Texte.



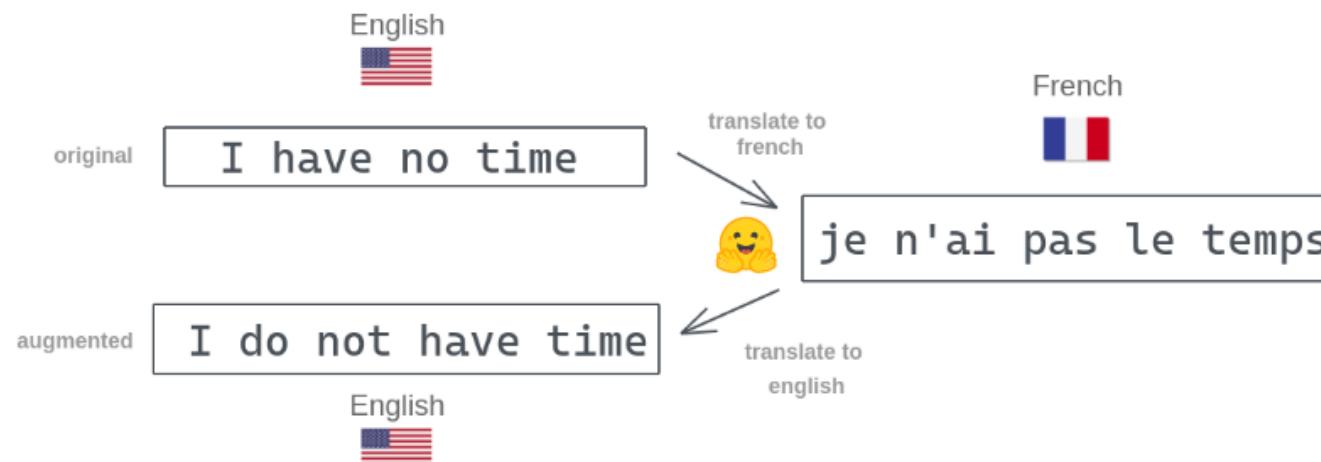
<https://www.ibm.com/content/dam/connectedassets-adobe-cms/worldwide-content/creative-assets/s-migr/ul/g/ea/ee/data-augmentation-image-augment.png>

Synonym Replacement: Wortersetzung für robustere Modelle

- Beim Synonym Replacement werden ausgewählte Wörter **durch ihre Synonyme ersetzt**, um neue Beispielsätze mit gleicher Bedeutung, aber unterschiedlicher Wortwahl zu erzeugen.
- Dies verhindert, dass sich ein Modell **zu stark auf bestimmte Schlüsselwörter einschießt**.
- Einfach umzusetzen, erfordert aber eine gute **Synonymquelle** (z. B. WordNet) und eine gewisse Vorsicht, da nicht alle Synonyme kontextuell passend sind.
- Ein effektives Mittel, um **linguistische Flexibilität** ins Modell zu bringen.

Back-Translation: Maschinelle Übersetzung zur Datenerweiterung

- Back-Translation übersetzt einen Satz **zunächst in eine andere Sprache** und anschließend **zurück in die Ausgangssprache**.
- Das Ergebnis ist ein **semantisch ähnlicher**, aber **anders formulierter** Satz.
- Diese Methode generiert völlig neue Varianten, was vor allem bei **geringem Datenvolumen** sehr nützlich ist.
- **Voraussetzung** ist ein gut funktionierendes maschinelles Übersetzungssystem und ein sinnvolles Sprachpaar für die Erzeugung vielfältiger Paraphrasen.



Risiken und Grenzen von Textaugmentation

- **Unnatürliche oder kontextfremde Sätze können entstehen**, besonders wenn Synonyme nicht sorgfältig ausgewählt werden.
- Übermäßige oder unsachgemäße Augmentation **kann zu Verzerrungen im Datensatz führen**.
- Nicht jede Aufgabe profitiert gleichermaßen von Augmentation – bei hochspezialisierten **Fachterminologien** könnten Sinnverschiebungen auftreten.
- **Eine sorgfältige Strategie und Evaluierung ist notwendig, um sicherzustellen, dass Augmentation die Modellleistung verbessert**.



Code Demo: Einsatz von Back-Translation mit MarianMT

Verarbeitung_von_Textdaten_NLP_Grundlagen/Datenaugmen
tation/demo_back_translation.py



Übung: Teste die
Datenaugmentation mit
Übersetzung in französisch.

Verarbeitung_von_Textdaten_NLP_Grundlagen/Datenaugm
entation/demo_back_translation.py

Agenda

1. Grundlagen des Deep Learning
2. Erweiterte Konzepte und Architekturen
3. Verarbeitung von Textdaten (NLP-Grundlagen)
- 4. Transformer-Modelle**
5. Trainingspraxis und Optimierung
6. MLOps und Deployment
7. Anwendungsfälle und Projekte

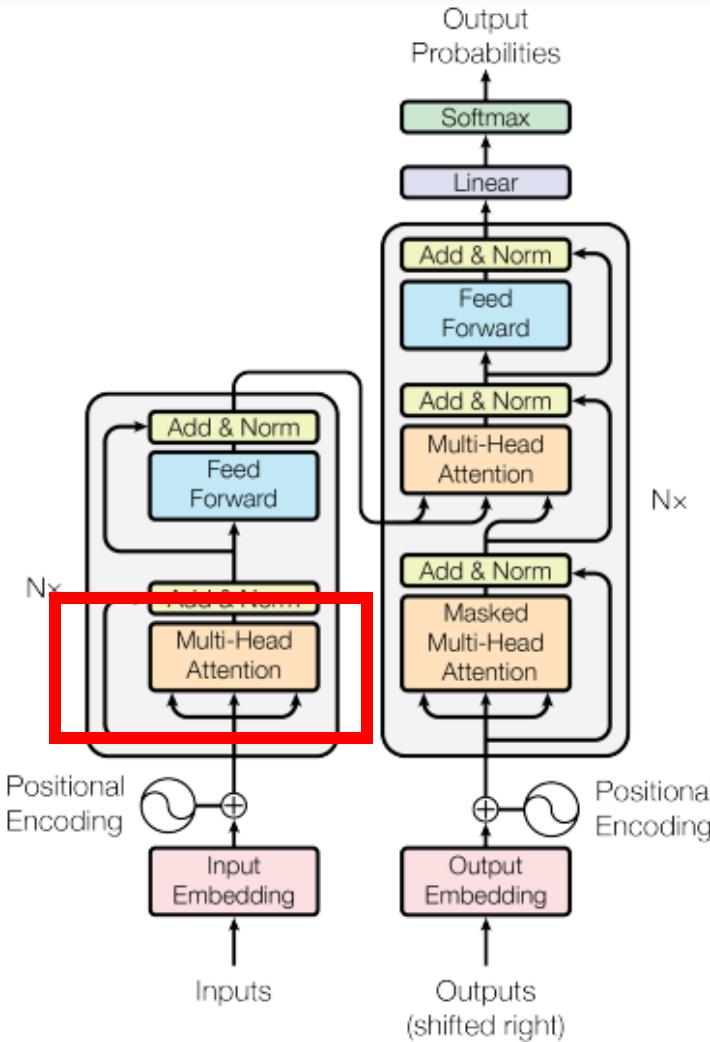
Probleme mit langen Kontexten und Abhängigkeiten

- **RNNs** haben Schwierigkeiten, über **lange Sequenzen** hinweg den Kontext aufrechtzuerhalten, da frühere Informationen mit zunehmender Sequenzlänge an Einfluss verlieren.
- **Vanishing- und Exploding-Gradient-Probleme** erschweren das effektive Training sehr tiefer oder langer RNN-Modelle.
- Durch serielle Verarbeitung ist **keine echte Parallelisierung** möglich, was die Trainingsgeschwindigkeit begrenzt.
- **Transformer-Modelle** setzen auf **Self-Attention** und umgehen somit viele dieser Nachteile, indem sie globale Zusammenhänge auf einmal berücksichtigen.

Memory-Bottlenecks und Schwierigkeiten bei paralleler Berechnung

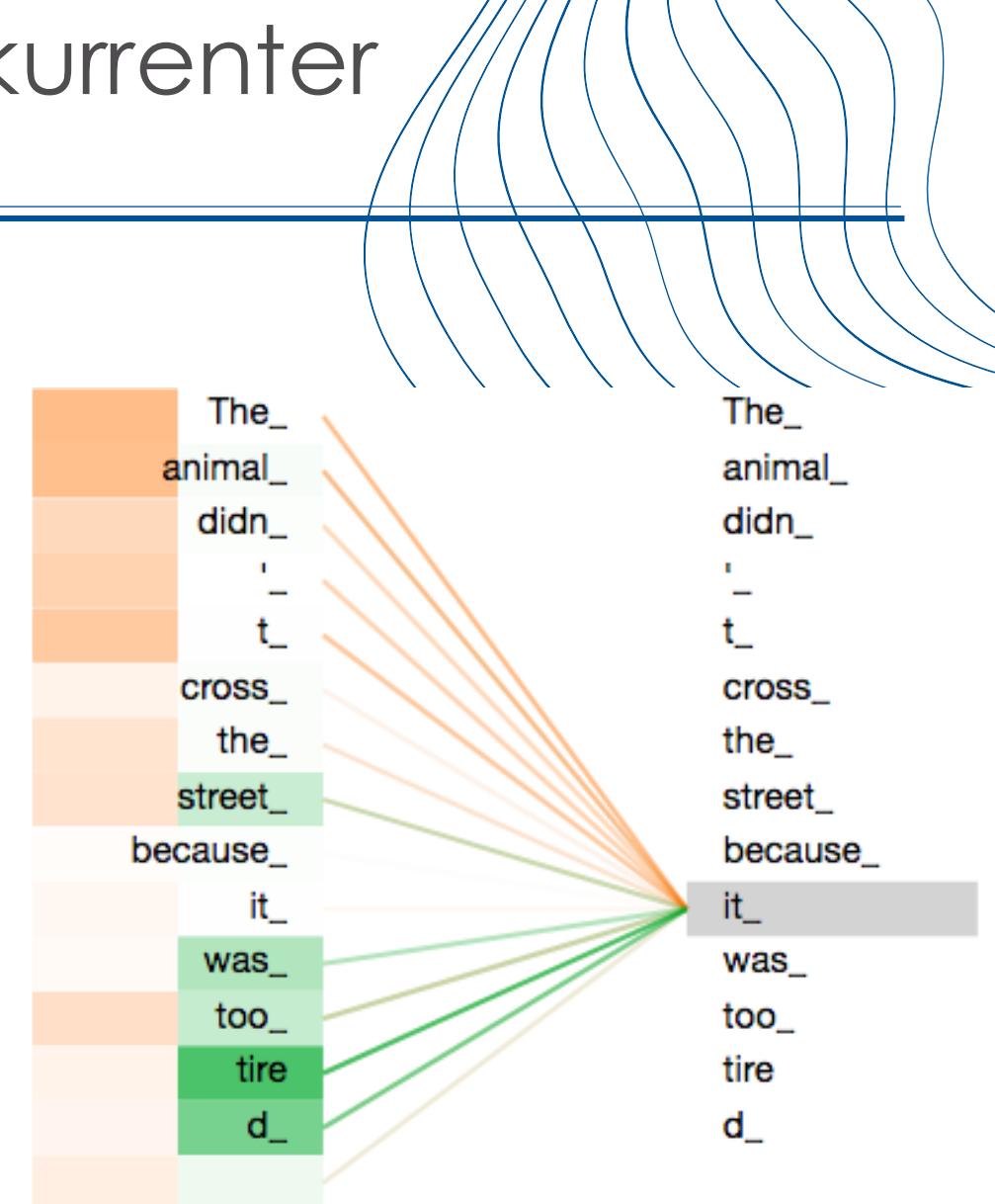
- RNNs müssen **Sequenzen schrittweise verarbeiten**, was bei langen Eingaben zu hohen Speicheranforderungen führt.
- Je länger die Sequenz, desto größer der **Memory-Bedarf**, da alle Zwischenzustände gehalten werden müssen.
- Die **fehlende Parallelisierung** hemmt die Effizienz: Große Datensätze können nur langsam verarbeitet werden.
- Der **Bedarf an effizienteren, parallelisierbaren Architekturen** wächst, um mit immer größeren Textkorpora umzugehen.

Attention is all you need



Warum Self-Attention statt rekurrenter Schleifen?

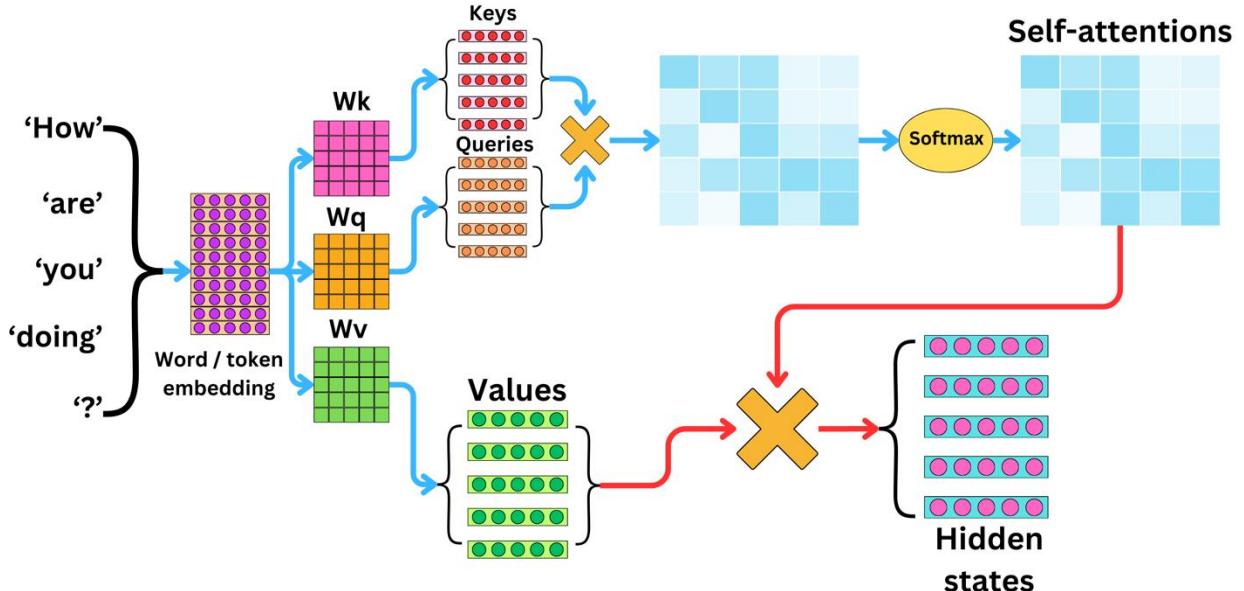
- **Self-Attention** ermöglicht, alle Tokens einer Sequenz **parallel** zu betrachten und relevante Zusammenhänge direkt zu gewichten.
- Dadurch können **globale Abhängigkeiten** mühelos erfasst werden, ohne die Probleme rekurrenter Schleifen.
- **Vanishing-Gradient-Probleme** werden reduziert, da die Berechnung nicht auf lange Kettenmultiplikationen angewiesen ist.
- **Self-Attention ist das Kernkonzept des Transformers und hat den Durchbruch in vielen NLP-Aufgaben ermöglicht.**



https://jalammar.github.io/images/t/transformer_self-attention_visualization_2.png

Self-Attention Mechanismus: Q, K, V und Berechnung der Gewichte

- Tokens werden in drei Projektionsräume abgebildet: **Queries** (Q), **Keys** (K) und **Values** (V).
- Die **Attention-Gewichte** entstehen aus der **Ähnlichkeit zwischen Query und Key** und bestimmen, welche Informationen aus den Values hervorgehoben werden.
- Dieser Mechanismus erlaubt es, **jedes Token kontextabhängig** neu zu gewichten, um relevante Beziehungen hervorzuheben.
- Die Self-Attention bildet damit den **Grundbaustein** des Transformers, um **komplexe Muster in der Eingabesequenz aufzudecken**.



https://substackcdn.com/image/fetch/f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2F0f41a56a-8b4a-4fb9-9124-82c01f95fc35_3399x1665.png

Attention Mechanismus

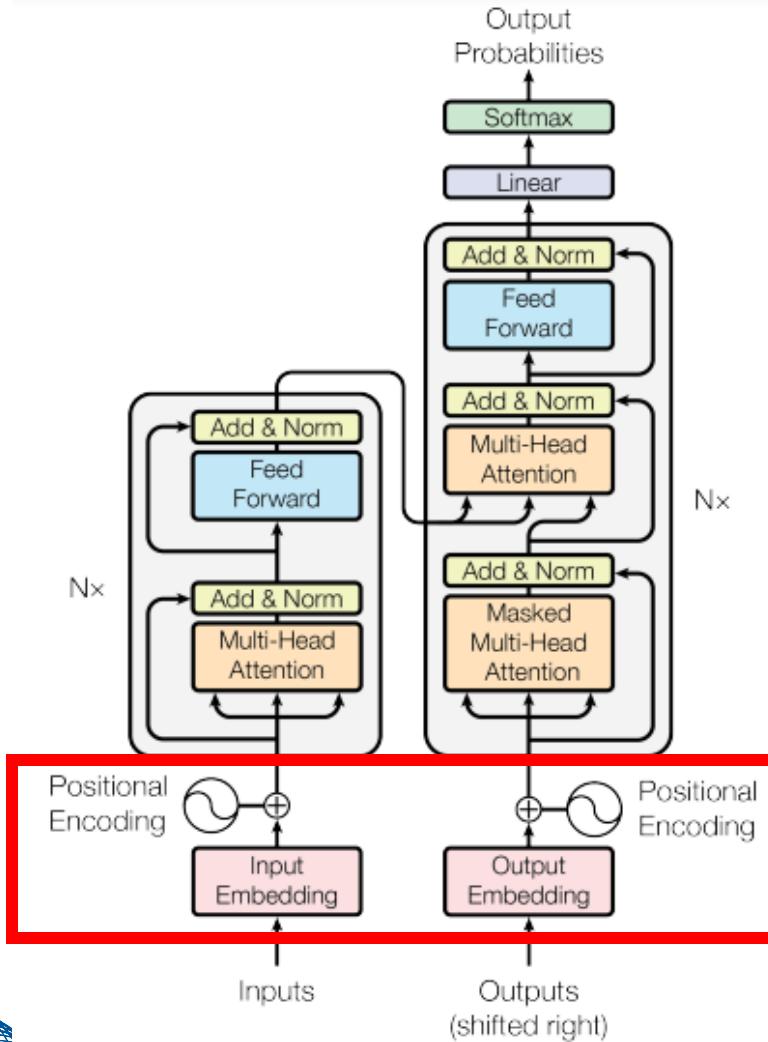
<https://poloclub.github.io/transformer-explainer/>



Code Demo: Berechnung von Self-Attention Schritt- für-Schritt in NumPy

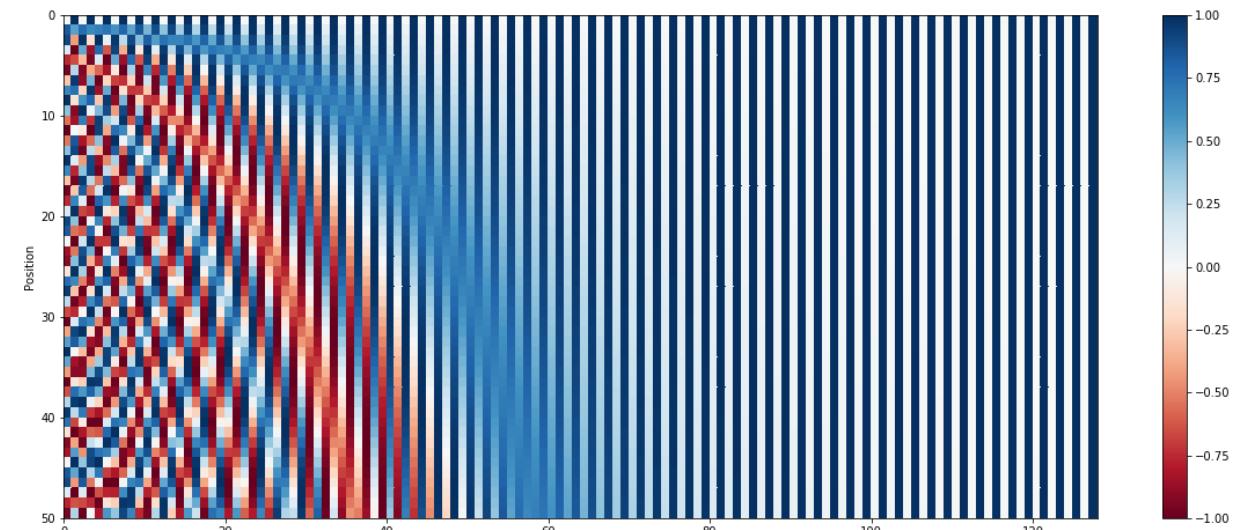
Transformer_Modelle/Architektur_des_Transformers/demo_se
lf_attention.py

Attention is all you need



Positional Encoding: Kodierung relativer Wortpositionen

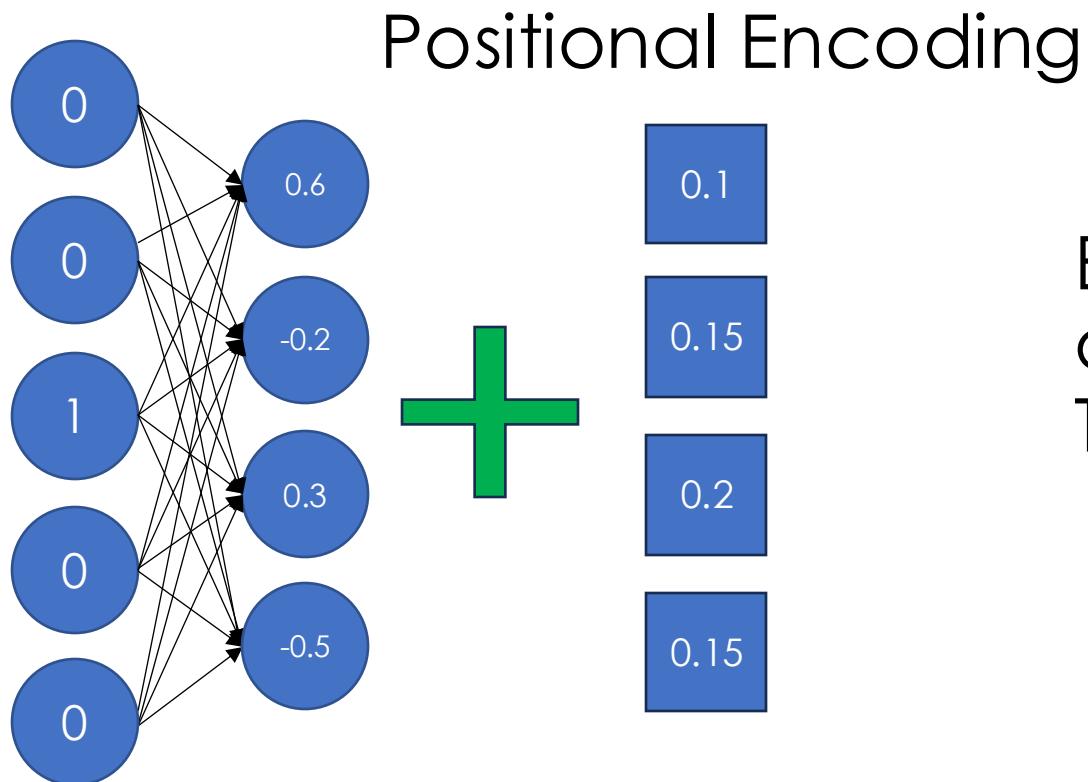
- Da Self-Attention **keine Sequenzordnung implizit speichert**, werden Positionsinformationen durch **Positional Encodings** ergänzt.
- **Periodische Funktionen** (z. B. Sinus- und Kosinus-Perioden) repräsentieren Positionen, sodass das Modell Reihenfolgen erkennen kann.
- Diese kontinuierlichen, trigonometrischen Encodings ermöglichen es, auch sehr lange Kontextfenster zu verarbeiten.
- **Positional Encoding ist essenziell, um den Transformers ein Verständnis für Wortreihenfolgen zu vermitteln.**



https://kazemnejad.com/img/transformer_architecture_positional_encoding/positional_encoding.png

Positional Encoding

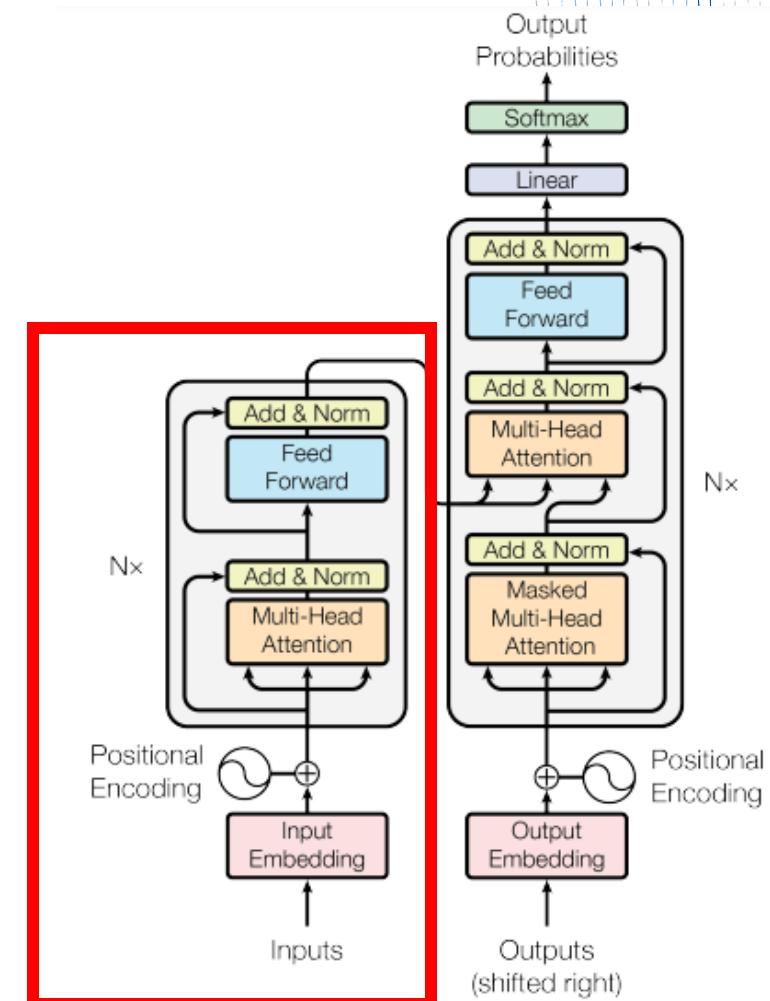
Von Token-Encoding zu Embeddings



Berechnungsvorschrift
aus der Position des
Tokens im Satz

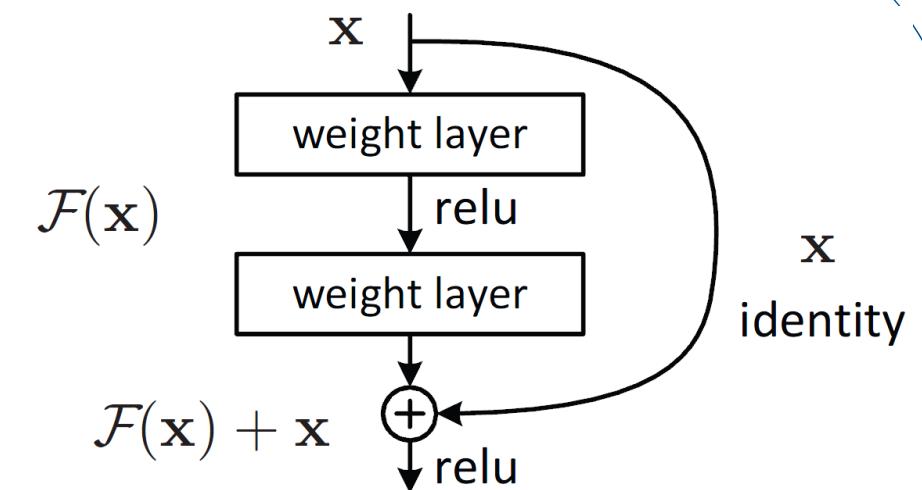
Encoder-Struktur: Gestapelte Attention- und Feed-Forward-Schichten

- Der **Encoder** besteht aus mehreren Blöcken, die jeweils eine **Multi-Head-Self-Attention-Schicht** und eine Positionwise-Feed-Forward-Schicht enthalten.
- Residual-Verbindungen und Layer Normalization sorgen für stabile Gradientenflüsse und erleichtern tiefes Training.
- Durch die **Mehrfachstapelung wird ein immer reichhaltigerer, kontextualisierter Merkmalsraum erzeugt.**
- Der Encoder transformiert **Eingabe-Embeddings in kontextreiche Repräsentationen, die für Downstream-Aufgaben genutzt werden.**



Residual Connections und Layer Normalization als Stabilitätsfaktoren

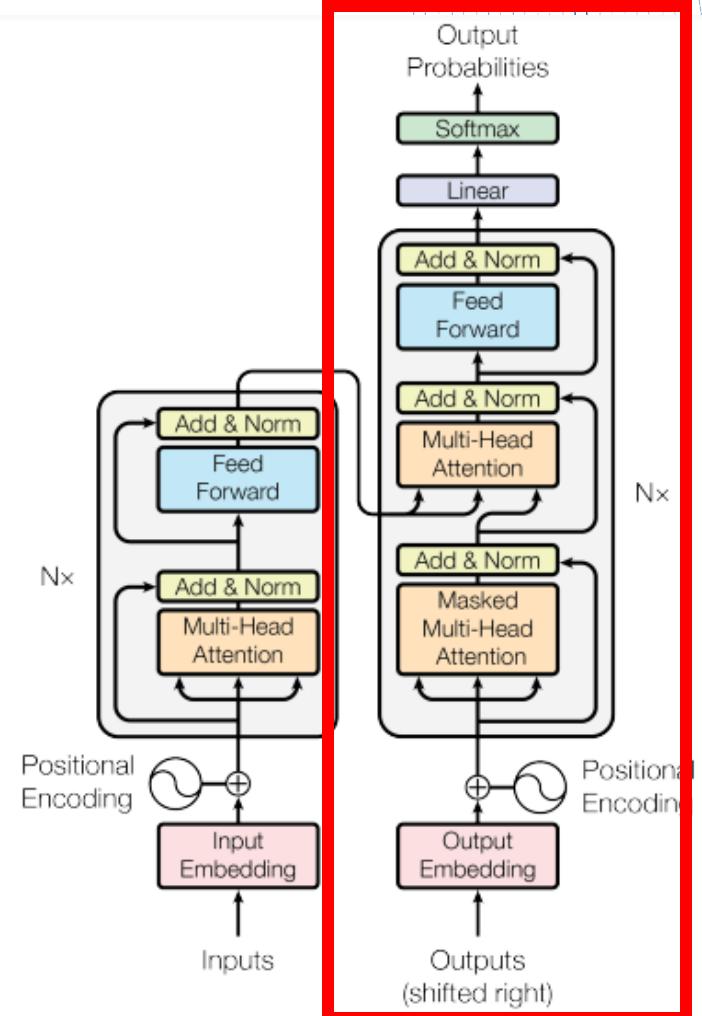
- **Residual Connections** leiten Informationen um tiefen Schichten herum und ermöglichen so stabileres Training.
- **Layer Normalization** normalisiert Eingaben jeder Schicht, stabilisiert Training und begünstigt Konvergenz.
- Durch diese Techniken können Transformer-Modelle **sehr tief** und gleichzeitig **effizient** trainiert werden.
- Das Ergebnis sind hochperformante Modelle, die in der Praxis state-of-the-art Ergebnisse liefern.



<https://production-media.paperwithcode.com/methods/resnet-e1548261477164.png>

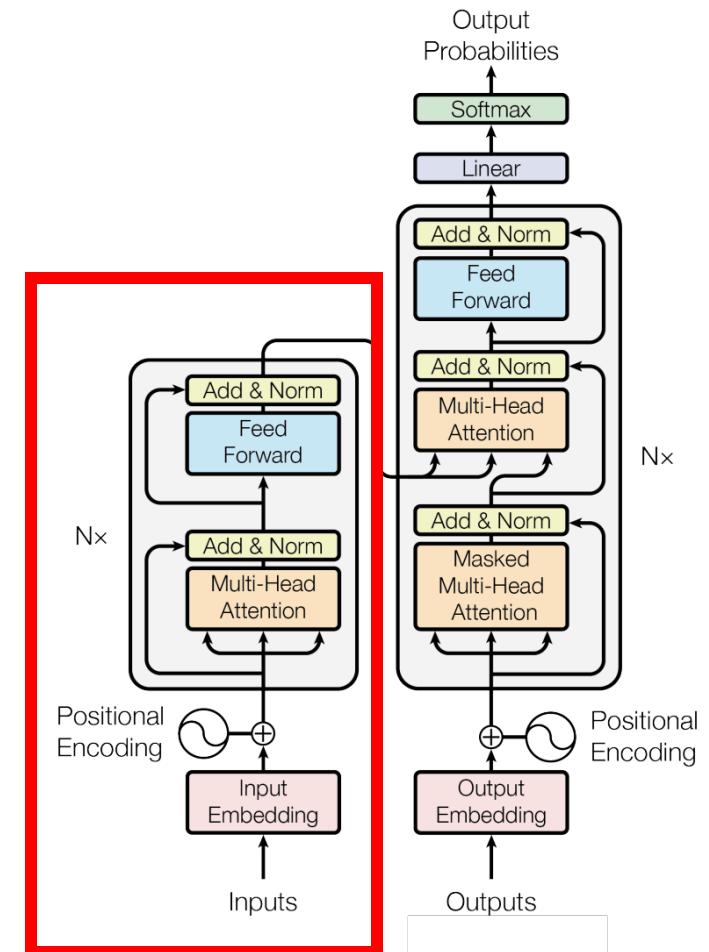
Decoder-Struktur: Maskierte Self-Attention und Cross-Attention

- Der **Decoder** nutzt **maskierte Self-Attention**, um bei der Generierung eines Tokens **nicht „in die Zukunft“** zu schauen und so autoregressives Verhalten zu gewährleisten.
- Cross-Attention-Schichten beziehen Informationen aus dem Encoder-Ausgaberaum ein, um den generierten Output auf den Input-Kontext abzustimmen.
- Der **Decoder ermöglicht die schrittweise Erzeugung von Ausgabetexten**, z. B. bei maschineller Übersetzung oder Textzusammenfassung.
- Durch diese Architektur wird eine flexible Input-Output-Transformation möglich, die deutlich leistungsfähiger ist als klassische RNN-basierte seq2seq-Modelle.



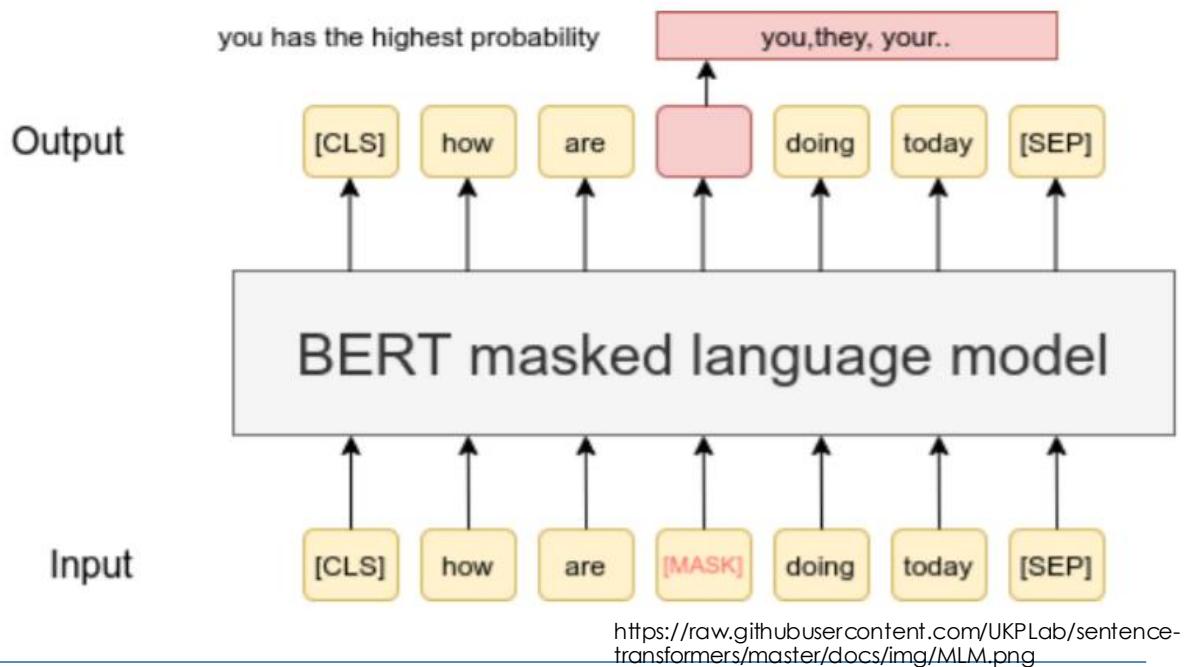
BERT: Bidirektionale Kontexte mit Masked Language Modeling

- BERT berücksichtigt simultan den **Kontext von links und rechts eines Tokens**, anstatt nur vorwärts oder rückwärts zu schauen.
- Durch das **Masked Language Modeling** lernt BERT tiefe semantische Zusammenhänge, indem es fehlende Wörter vorhersagen muss.
- BERT dient als **universeller Encoder**, der für vielfältige Aufgaben wie Sentimentanalyse, Named Entity Recognition oder Fragebeantwortung feingetunt werden kann.
- Mit **BERT wurden in vielen Benchmarks signifikante Leistungssteigerungen erreicht**.



Masked Language Modeling: Maskieren von Tokens zur Kontextmodellierung

- Bestimmte **Tokens im Input werden maskiert**, und das Modell muss diese rekonstruieren.
- Dies zwingt das Modell, kontextuelle Hinweise intensiver zu nutzen, um fehlende Informationen abzuleiten.
- **Masked Language Modeling ist zentrale Trainingsmethode von BERT**, um ein umfassendes kontextuelles Verständnis aufzubauen.
- Dadurch wird das Modell robust gegen unvollständige oder teils fehlende Eingaben.



<https://raw.githubusercontent.com/UKPLab/sentence-transformers/master/docs/img/MLM.png>

Code Demo: Tokenizer

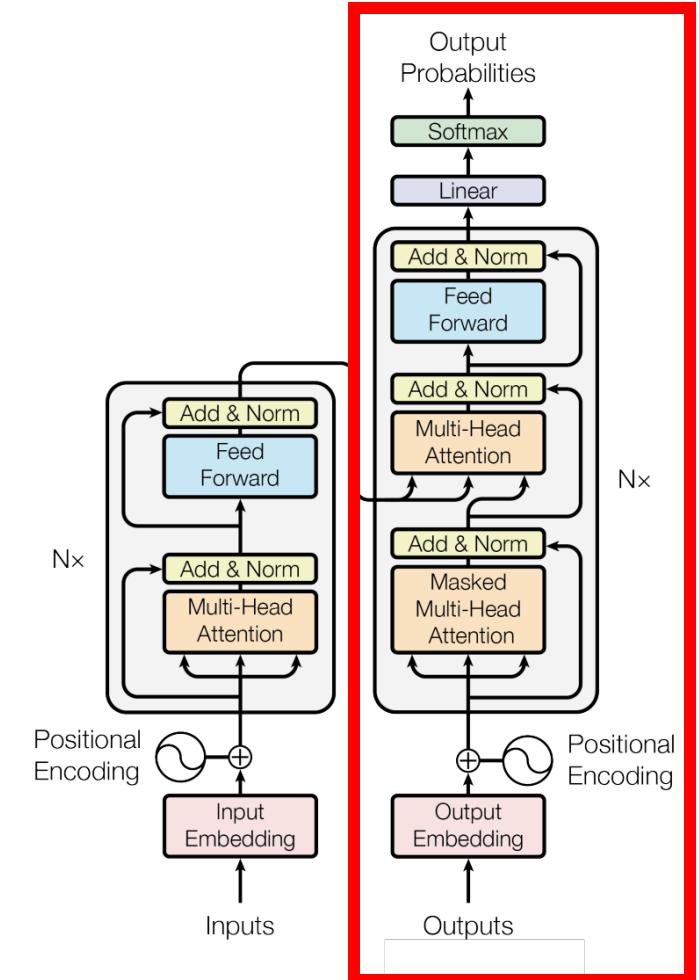
Transformer_Modelle/Architektur_des_Transformers/demo_bert_tokenizer.py

Übung Tokenizer

1. Ersetze den Input Text durch einen längeren Text. Was stellst du bei truncation=True fest?
2. Füge Sonderzeichen in den Text ein. Was beobachtest du?

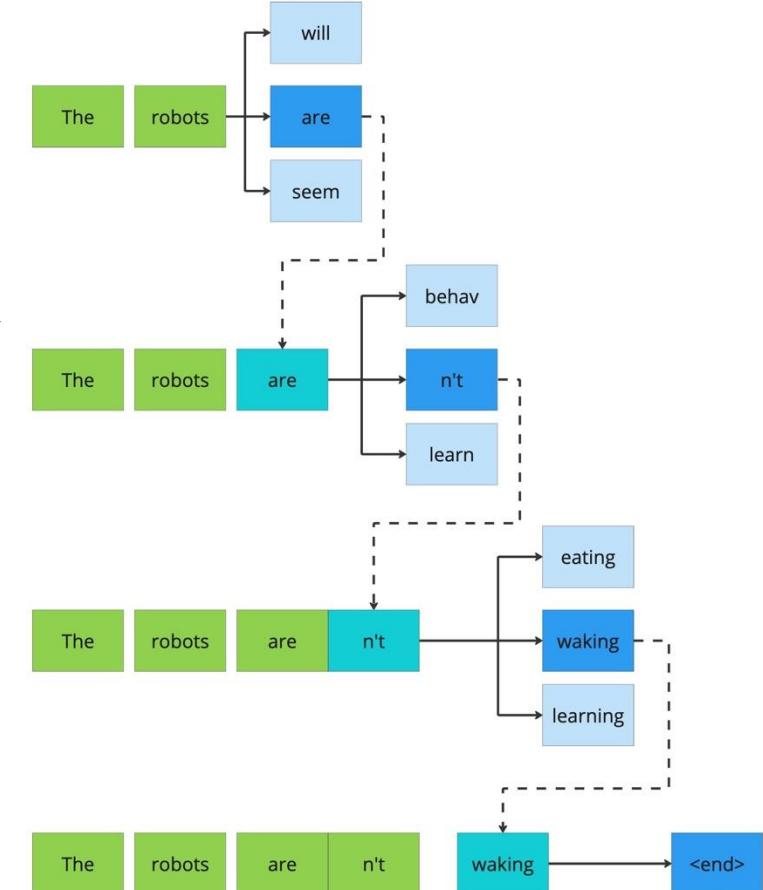
GPT: Autoregressive Modelle für Textgenerierung

- **GPT** (Generative Pre-trained Transformer) ist ein autoregressives Modell, das das **nächste Wort auf Basis der bisherigen Tokens vorhersagt**.
- Es eignet sich hervorragend für **generative Aufgaben** wie kreative Texterstellung, Dialogsysteme oder Storytelling.
- Größere GPT-Modelle (GPT-2, GPT-3, GPT-4) zeigen bemerkenswerte Fähigkeiten im freien Textgenerieren und bieten breites Sprachverständnis.
- Ihre Stärke liegt in der natürlich anmutenden, flüssigen Textproduktion.



Next-Token Prediction: Grundlagen der Textgenerierung

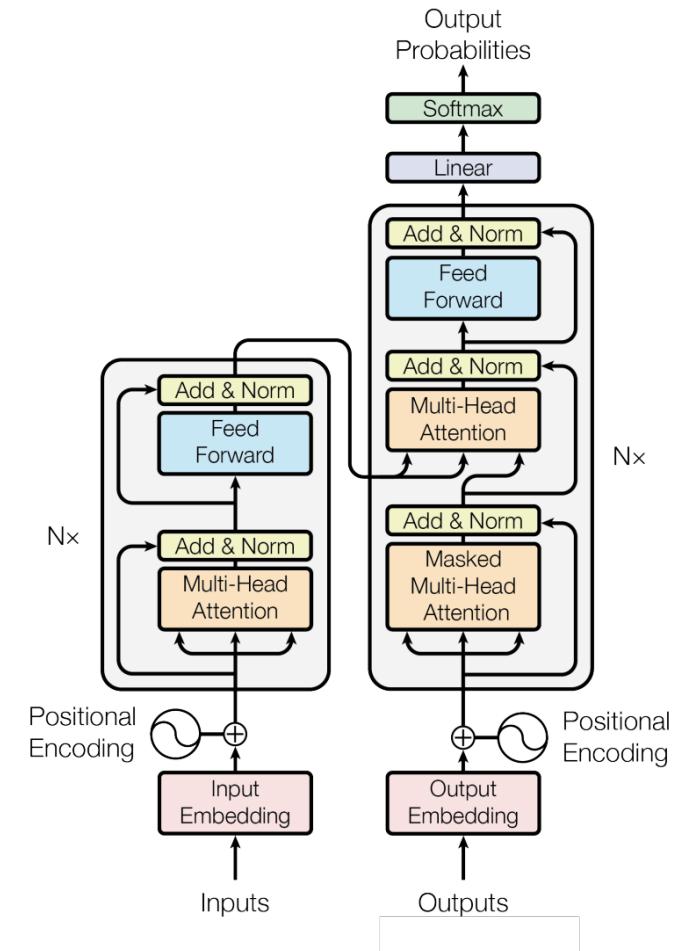
- Beim Next-Token-Prediction-Training sagt das Modell das **nächste Wort auf Basis des bisherigen Kontextes** vorher.
- Dieses unüberwachte Trainingsverfahren nutzt riesige Textkorpora, um statistische Sprachmuster zu erlernen.
- Das Modell entwickelt ein **implizites Verständnis für Grammatik, Syntax und semantische Muster**.
- Next-Token Prediction bildet das **Fundament für generative Sprachmodelle** wie GPT.



https://substackcdn.com/image/fetch/f_auto,q_auto:good,fl_progressive,steep/hhttps%3A%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2F943efc65-c436-4914-b6f9-13c39a8cedc5_1896x2109.jpeg

seq2seq-Transformer: Neuer Standard für maschinelle Übersetzung

- **seq2seq-Transformermodelle** nutzen ein **Encoder-Decoder-Design**, um Eingabesequenzen (z. B. Sätze in Sprache A) in Ausgabesequenzen (z. B. Sätze in Sprache B) zu transformieren.
- Durch parallele Berechnungen und Self-Attention erreichen sie hohe Effizienz und Präzision.
- Sie haben sich als neuer Goldstandard in der **maschinellen Übersetzung** etabliert und verdrängen RNN-basierte Ansätze.
- Auch in anderen Bereichen wie **Textzusammenfassung** oder **Fragebeantwortung** setzen seq2seq-Transformer Maßstäbe.



Vergleich der Modelle: Einsatzgebiete, Stärken und Schwächen

- **BERT**: Hervorragend in Verständnis- und Klassifikationsaufgaben, aber weniger in der freien Textgenerierung.
- **GPT**: Exzellent in kreativer Textproduktion, jedoch kontextuell meist nur unidirektional (von links nach rechts) und daher eingeschränkter im volumnfänglichen Sprachverständnis.
- **seq2seq-Transformer**: Spezialisieren sich auf Input-Output-Aufgaben (wie Übersetzung) und verbinden das Beste aus zwei Welten (Encoder für Verständnis, Decoder für Generierung).
- Die Modellwahl hängt stark von der jeweiligen Aufgabenstellung und den Anforderungen an Verständnis oder Generierung ab.

Vergleich der Modelle

Encoder

Erkennen der
Eigenschaften des
Textes

Klassifikation des
Textes,
Embeddings
erstellen

BERT

Decoder

Input Sequenz
erweitern

Textvorhersage,
Next Token
Prediction

GPT

Encoder Decoder

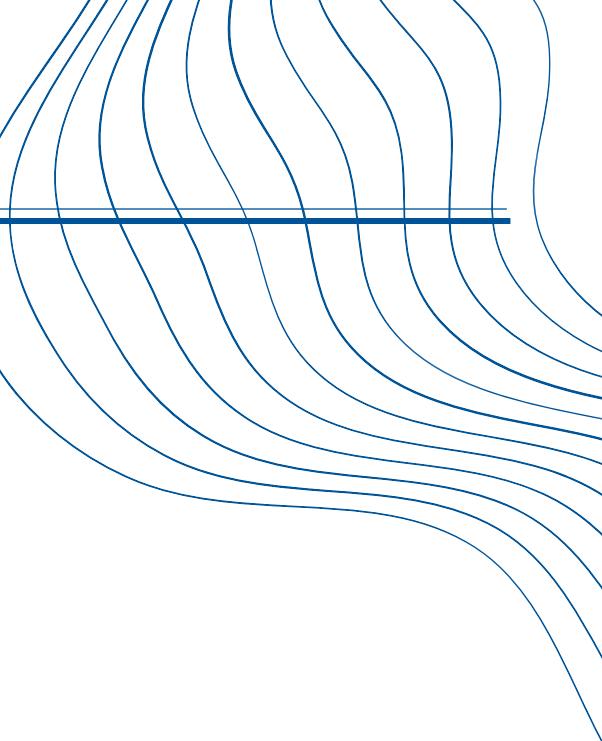
Sequenz zu
Sequenz

Übersetzung,
Zusammenfassung

seq2seq

TensorFlow/Keras: Beispielimplementierung eines einfachen Transformers

- Eigene **Custom Layers für Multi-Head-Attention und Feed-Forward-Netzwerke** können in Keras einfach erstellt werden.
- Einsatz von **Functional APIs** oder Subclassing, um eine modulare, wiederverwendbare Struktur aufzubauen.
- **Schritt-für-Schritt-Implementierung erleichtert Verständnis für interne Abläufe.**
- Mit diesem Fundament lassen sich komplexere Modelle und Fine-Tuning-Strategien verwirklichen.



TensorFlow





Code Demo: Einfachen Transformer in TensorFlow/Keras implementieren

Transformer_Modelle/Implementierung_und_Tools/demo_tra
nsformer_keras.py

Hugging Face: Nutzung und Fine-Tuning vortrainierter Modelle

- **Hugging Face Transformers** bietet eine breite Auswahl an vortrainierten Modellen (BERT, GPT, RoBERTa, T5) direkt an.
- **Einfaches Fine-Tuning** mit wenigen Codezeilen für spezifische Tasks, z. B. Klassifikation oder Named Entity Recognition.
- **Große Community und umfangreiche Dokumentation** erleichtern den Einstieg.
- Industriestandard für **schnellen Transfer** von Forschungsergebnissen in praktische Anwendungen.



Hugging Face

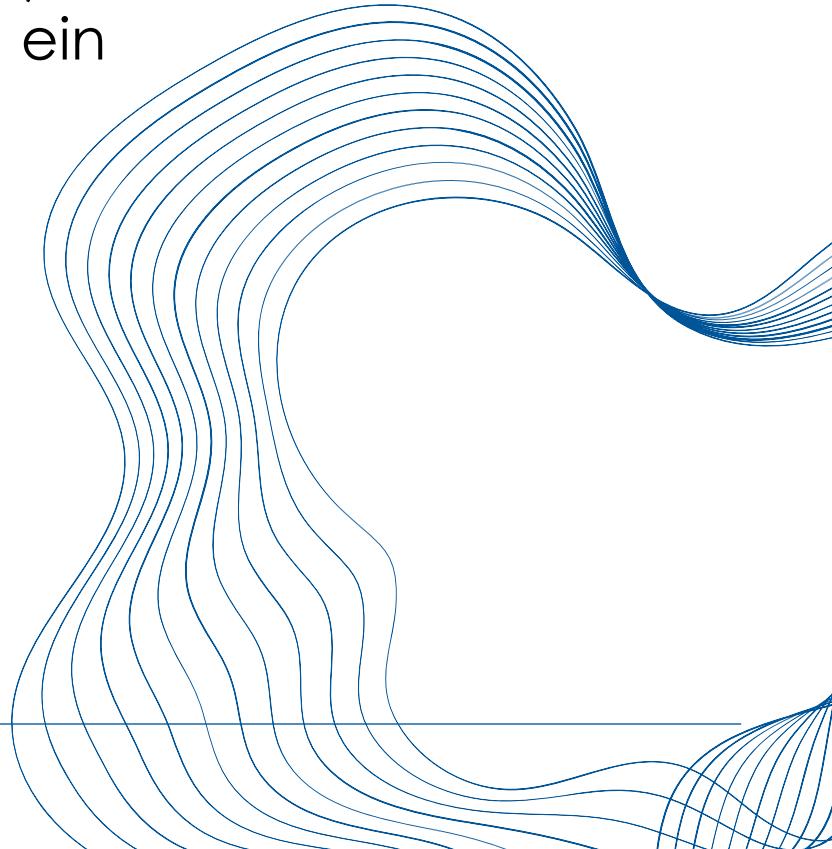


Code Demo: Inferenz mit BERT und GPT-Modellen

Transformer_Modelle/Bekannte_Modelle/demo_bert_gpt_inf
erenz.py

Auxiliary Tasks (z. B. POS-Tagging) zur Verbesserung der Repräsentation

- Neben Hauptzielen (z. B. Textklassifikation) können **zusätzliche Aufgaben wie POS-Tagging** oder Semantische Rollenklassifikation trainiert werden.
- Diese zusätzlichen Tasks liefern zusätzliche Lernsignale, **verbessern die Qualität der Embeddings** und fördern ein ganzheitliches Sprachverständnis.
- Der Ansatz **steigert die Robustheit** und kann helfen, auch bei begrenzten Datenmengen bessere Repräsentationen zu erzielen.
- Durch Auxiliary Tasks erhalten Modelle ein breiteres Wissen über linguistische Strukturen.



Multi-Task Learning: Effizienzsteigerung durch parallele Aufgaben

- Bei **Multi-Task Learning** werden mehrere verwandte Aufgaben gleichzeitig gelernt, um gemeinsame Strukturen besser auszunutzen.
- Dies führt zu effizienterem Training, da das Modell von **verschiedenen Perspektiven auf die Daten** profitiert.
- **Overfitting** auf eine Einzelaufgabe wird **reduziert**, da das Modell auf vielfältige Zielstellungen optimiert wird.
- Multi-Task Learning **fördert Generalisierung** und kann die **Performance in jeder einzelnen Aufgabe verbessern**.

Übung: BERT nutzen

1. Rufe die Dokumentation von dem Modell bert-base-uncased auf (<https://huggingface.co/google-bert/bert-base-uncased>) und verschaffe dir einen Überblick.
2. Nutze das Modell, um den folgenden Satz zu vervollständigen:
„Die Hauptstadt von Deutschland ist [MASK]“
3. Ersetze bert-base-uncased durch ein anderes Modell, das du auf huggingface findest und teste es.



Code Demo: Masked Language Modeling mit Hugging Face

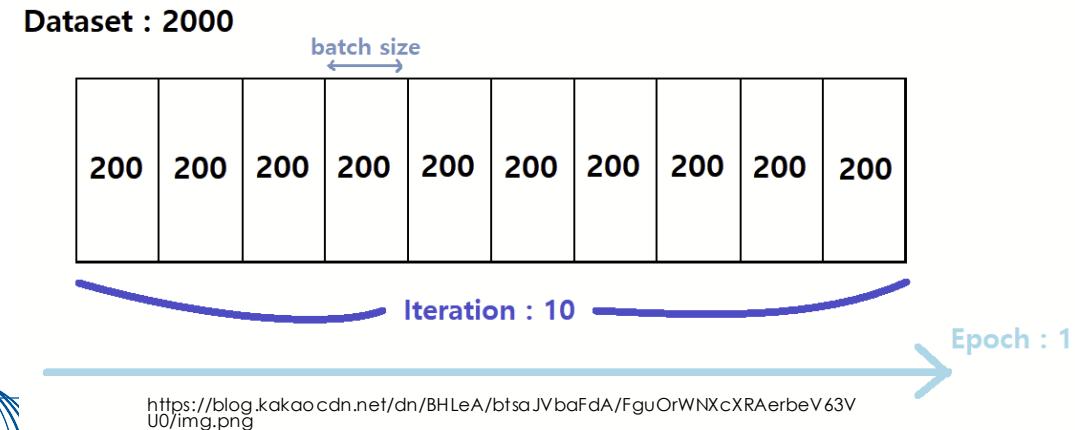
Transformer_Modelle/Trainingsziele_Auxiliary_Tasks/demo_m
asked_language_modeling.py

Agenda

1. Grundlagen des Deep Learning
2. Erweiterte Konzepte und Architekturen
3. Verarbeitung von Textdaten (NLP-Grundlagen)
4. Transformer-Modelle
- 5. Trainingspraxis und Optimierung**
6. MLOps und Deployment
7. Anwendungsfälle und Projekte

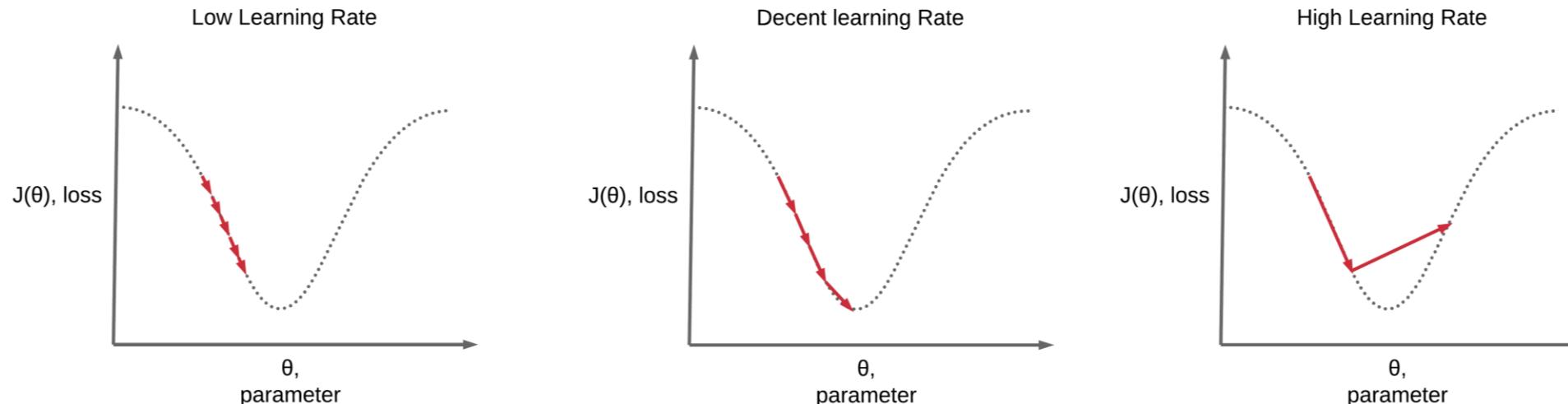
Batch Size: Einfluss auf Stabilität, Geschwindigkeit und Generalisierung

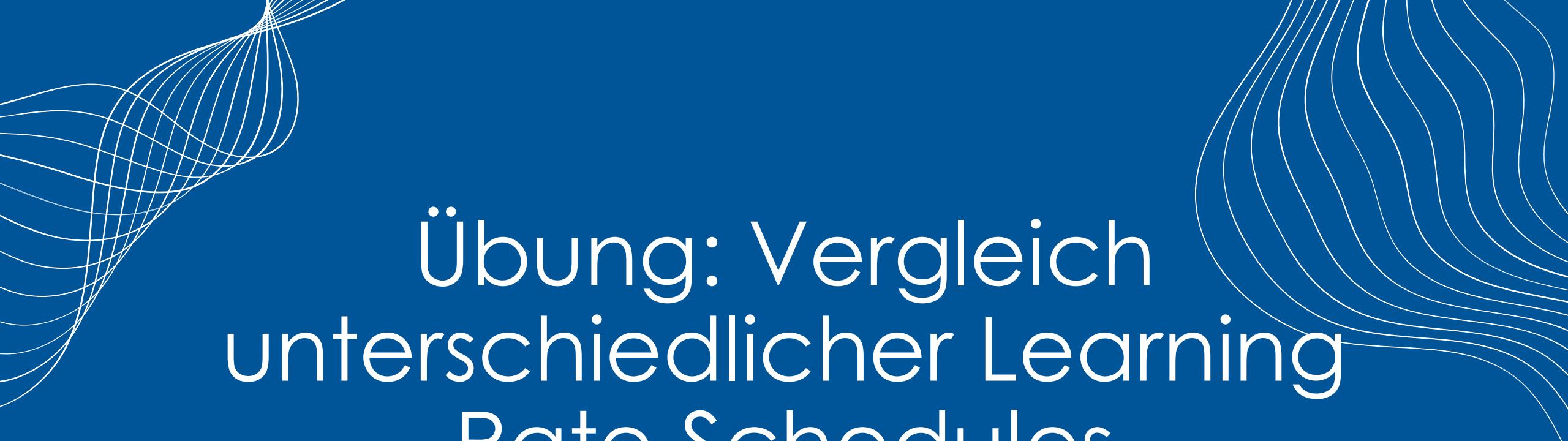
- Große Batches ermöglichen **schnelleres Training**, bergen aber ein **höheres Risiko für Overfitting**, da Updates seltener und weniger rauschbehaftet sind.
- Kleine Batches können zu **besserer Generalisierung** führen, doch die Trainingsgeschwindigkeit sinkt.
- Der richtige Kompromiss hängt von Datenmenge, Modellkomplexität und Hardware-Ressourcen ab.
- **Experimente** mit verschiedenen Batchgrößen und Monitoring von Metriken helfen, ein optimales Gleichgewicht zu finden.



Learning Rate Schedules: Step Decay, Warmup, Cyclical Learning Rates

- **Step Decay:** Die Lernrate wird nach festgelegten Epochen reduziert, um feineres Finetuning am Ende des Trainings zu ermöglichen.
- **Warmup:** Zu Beginn wird die Lernrate langsam angehoben, um Stabilität zu gewährleisten und anfängliche Explodierungen der Gradienten zu vermeiden.
- **Cyclical Learning Rates:** Periodische Schwankungen der Lernrate helfen, aus lokalen Minima herauszufinden und die Konvergenz zu verbessern.
- **Eine durchdachte Lernratensteuerung kann die Trainingszeit verkürzen und die Modellqualität erhöhen.**





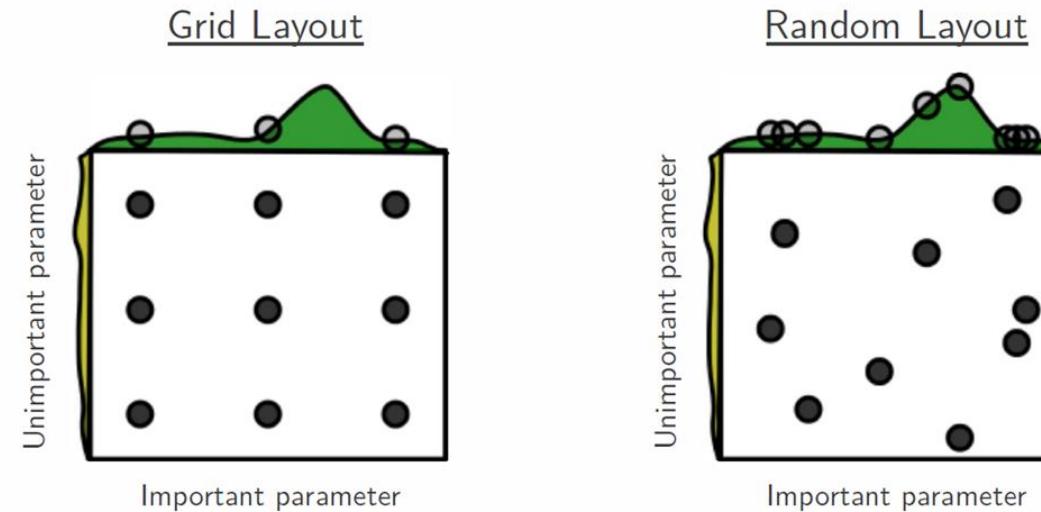
Übung: Vergleich unterschiedlicher Learning Rate Schedules

Trainingspraxis_und_Optimierung/Effizientes_Training/
uebung_lr_schedules.py

Vollziehe den Code nach. Findest du noch einen besseren
Scheduler?

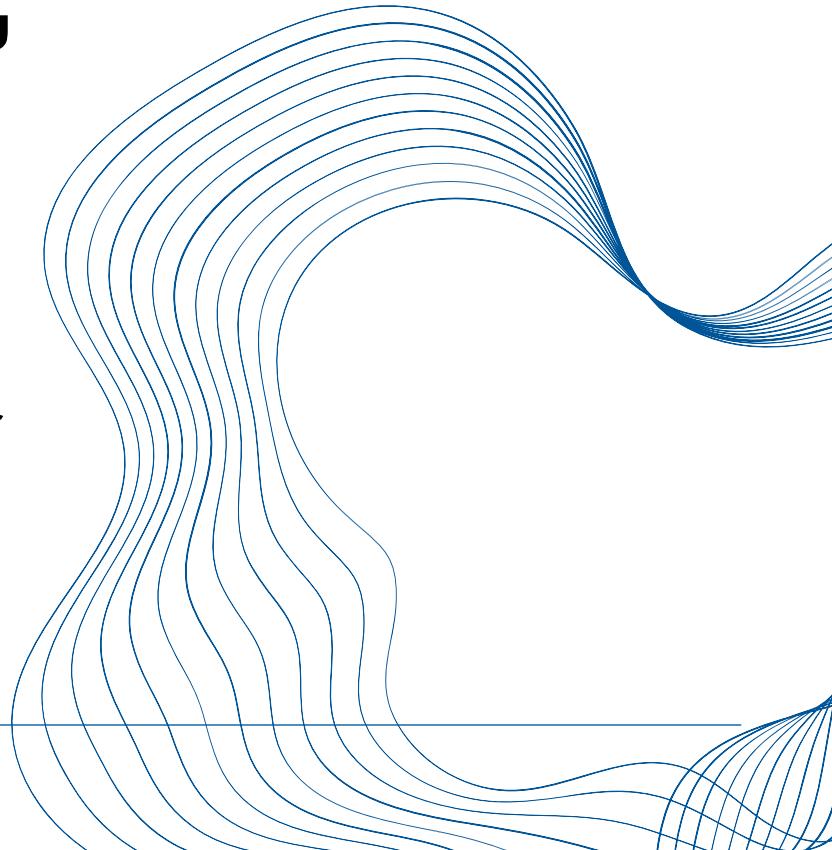
Grid Search, Random Search, Bayesian Optimization

- **Grid Search** durchsucht systematisch den Parameterraum, ist aber bei vielen Hyperparametern teuer.
- **Random Search** probiert zufällig gewählte Parameterkombinationen und findet oft schneller gute Lösungen.
- **Bayesian Optimization** nutzt Modellierungen des Parameterraums, um gezielt Bereiche mit hohem Potenzial zu erkunden.
- Die Wahl der Methode hängt von der Komplexität des Modells, Rechenressourcen und Zeit ab.



Wichtige Hyperparameter: Learning Rate, Layeranzahl, Dropout-Rate

- Die **Lernrate** beeinflusst maßgeblich die Konvergenzgeschwindigkeit und Stabilität des Trainings.
- Die **Anzahl der Layer** und deren Tiefe bestimmen die Modellkapazität; **zu flach führt zu Underfitting, zu tief birgt Overfitting-Gefahr.**
- **Dropout** reguliert das Modell und beugt Overfitting vor, indem es zufällig Neuronen deaktiviert.
- Eine ausgewogene Einstellung dieser Parameter ist entscheidend für hohe Genauigkeit und gute Generalisierung.





Code Demo: Nutzung von Optuna für Bayesian Optimization

Trainingspraxis_und_Optimierung/Hyperparameter_Tuning/demo_optuna_bayesian.py

Datenvorverarbeitung optimieren: Prefetching, Caching, Datengeneratoren

- **Prefetching lädt Daten im Hintergrund**, während das Modell trainiert, um Leerlaufzeiten der GPU zu minimieren.
- **Caching speichert bereits verarbeitete Daten im Arbeitsspeicher**, um erneute Berechnungen zu vermeiden und die Pipeline zu beschleunigen.
- **Effiziente Datengeneratoren** (z. B. `tf.data` in TensorFlow) reduzieren Overhead und verbessern die Gesamt-Performance.
- Eine optimierte Datenpipeline kann signifikant zur **Senkung der Trainingsdauer** beitragen.

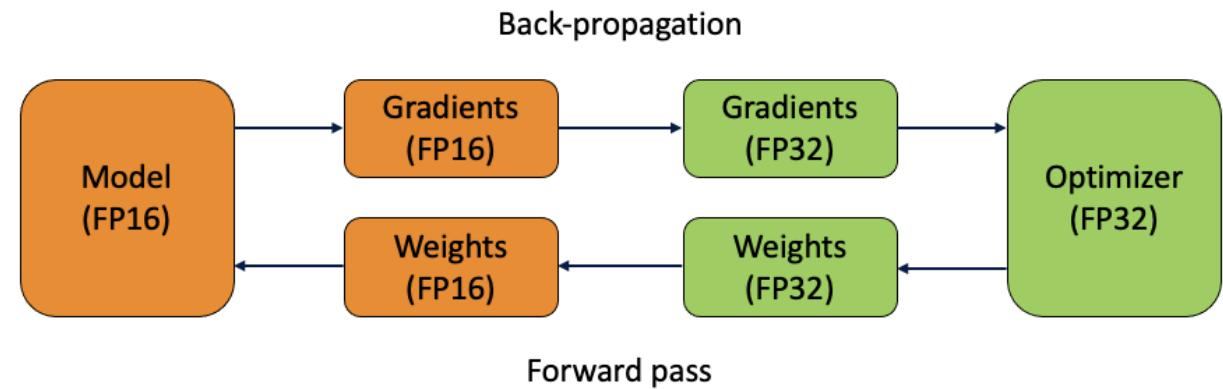


Code Demo: Datenpipeline- Optimierung mit tf.data

Trainingspraxis_und_Optimierung/Effizientes_Training/demo_tfdata_pipeline.py

FP16 vs. FP32: Unterschiede in Genauigkeit und Performance

- FP16 nutzt 16-Bit-Gleitkommazahlen und ist dadurch **schneller**, jedoch mit **geringerer numerischer Präzision**.
- FP32 (32-Bit) liefert **genauere Berechnungen**, ist aber **ressourcenintensiver**.
- Mixed Precision **kombiniert beide Ansätze**, um das Training stark zu beschleunigen, ohne dabei signifikant an Güte zu verlieren.
- Ermöglicht Nutzung moderner GPU-Funktionalitäten und reduziert Trainingszeiten.



https://miro.medium.com/v2/resize:fit:1400/1*htZ4PF2fZ0tJ5HdsIaAbQ.png

Nutzung von NVIDIA Tensor Cores zur Trainingsbeschleunigung

- NVIDIA Tensor Cores sind für **schnelle Matrixmultiplikationen optimiert** und profitieren stark von Mixed Precision.
- **Massive Beschleunigung**, insbesondere bei sehr tiefen und großen Modellen.
- Kostenersparnis durch effizienteren GPU-Einsatz.
- Standard in aktuellen GPU-Generationen für High-Performance-Deep-Learning.



Implementierung von Mixed Precision in TensorFlow/Keras

- Die Aktivierung erfolgt durch das **Setzen einer globalen Policy** (z. B.
``tf.keras.mixed_precision.set_global_policy('mixed_float16')``).
- Viele Operationen werden automatisch in FP16 ausgeführt, sensible Berechnungen bleiben in FP32.
- Resultiert in deutlich **schnellerem Training bei minimalen Genauigkeitsverlusten**.
- Ein **unkomplizierter Schritt** zu besserer Auslastung der Hardware.



Code Demo: Mixed Precision Training mit Tensorflow

Trainingspraxis_und_Optimierung/Mixed_Precision_Training/d
emo_mixed_precision.py

Systematische Dokumentation und Vergleich von Experimenten

- Sorgfältiges Logging von **Hyperparametern, Versionsinformationen und Metriken** ermöglicht reproduzierbare Forschung.
- Vergleich verschiedener Runs hilft, Verbesserungen klar zu quantifizieren.
- Gemeinsame **Dokumentation im Team** erleichtert Wissensaustausch und beschleunigt den Fortschritt.
- **Diese Praxis ist die Grundlage für professionelle MLOps-Workflows.**

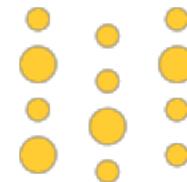
MLflow: Logging von Parametern, Metriken und Modellen



- MLflow bietet ein **zentrales UI, um Experimente, Parameter, Metriken und Artefakte zu verwalten.**
- Einfacher **Vergleich** zwischen verschiedenen Trainingsläufen.
- Langfristige Speicherung von Modellen und deren Metadaten fördert Nachhaltigkeit der Entwicklung.
- Unverzichtbar für **Reproduzierbarkeit** in professionellen ML-Projekten.

Weights & Biases: Kollaboratives Tracken und Visualisieren

- Weights & Biases (W&B) ist eine **cloudbasierte** Lösung, die es Teams ermöglicht, Experimente in Echtzeit zu verfolgen.
- **Interaktive Dashboards** und einfache Handhabung erleichtern die Analyse großer Mengen an Trainingsläufen.
- Schnelles **Teilen von Ergebnissen** und einfache Integration in CI/CD-Pipelines.
- Fördert **transparente Zusammenarbeit** und schnelle Iterationszyklen.



Weights & Biases

Reproduzierbare Pipelines durch klare Versionierung

- Verwendung von **Git für Codeversionierung** und **MLflow/W&B für Modell- und Parametertracking** bietet vollständige Transparenz.
- Reproduzierbarkeit ist sichergestellt, da alle relevanten Informationen gespeichert sind.
- Erleichtert die Wiederaufnahme abgebrochener Experimente oder die Skalierung auf neue Hardware.
- Fundament für professionelle **MLOps-Strukturen**.



Code Demo: MLflow- Integration in einen Trainings-Workflow

Trainingspraxis_und_Optimierung/Experiment_Tracking/dem
o_mlflow_integration.py

Übung: Fashion MNIST

Tracke deine Experimente mit mlflow im Fashion MNIST Experiment.

Hinweis: Recherche für das Loggen des Tensorflow Models notwendig.

Checkpoints: Absicherung von Modellzuständen während des Trainings

- Regelmäßige Checkpoints sichern **Zwischenzustände** des Modells, um Fortschritt nicht bei **Abstürzen oder Unterbrechungen** zu verlieren.
- Erleichtert die Analyse verschiedener Trainingsphasen durch Zugriff auf frühere Modellzustände.
- Senkt das **Risiko**, komplettne Trainings **neu starten** zu müssen.
- Besonders wertvoll bei langwierigen Trainings und knappen Hardware-Ressourcen.
- Laden des **letzten Checkpoints ermöglicht nahtlosen Anschluss** an den vorherigen Stand.
- Eine gute Praxis, um Zeit und Kosten zu sparen.



Code Demo: Laden von Checkpoints und Inferenz auf alten Modellversionen

Trainingspraxis_und_Optimierung/Checkpointing_Modellver
waltung/demo_checkpoint_laden.py

Übung: Fashion MNIST

Erstelle automatisiert Checkpoints für dein Modell im Fashion MNIST.

Bonus Übung Fashion MNIST

Optimiere dein Modell mit Hyperparameter Suche und Learning Rate Schedules.

Agenda

1. Grundlagen des Deep Learning
2. Erweiterte Konzepte und Architekturen
3. Verarbeitung von Textdaten (NLP-Grundlagen)
4. Transformer-Modelle
5. Trainingspraxis und Optimierung
- 6. MLOps und Deployment**
7. Anwendungsfälle und Projekte

Modellregistrierung und Metadatenverwaltung



- **Zentralisierte Speicherung** aller Modellversionen in einer Modellregistrierungslösung wie MLflow oder Weights & Biases.
- Speicherung und Verwaltung von **Metadaten wie Trainingsparametern, Datensätzen und Performance-Metriken** für jede Modellversion.
- Einfaches Auffinden und Wiederverwenden von Modellen durch eindeutige Kennzeichnungen, **Tags und Versionsnummern**.
- Ermöglicht die Nachverfolgung von Modellentwicklungen und erleichtert die Reproduzierbarkeit von Experimenten.
- Grundlage für **effizientes MLOps**, indem alle relevanten Informationen zentral verwaltet werden.



Wiederverwendung vortrainierter Modelle in neuen Projekten

- Nutzung **vortrainierter Modelle als Ausgangspunkt**, um Entwicklungszeiten zu verkürzen und Ressourcen zu sparen.
- **Ausnutzung bereits gelernter Features** und Muster zur Verbesserung der Modellleistung in neuen Aufgaben.
- **Beschleunigung des Prototyping-Prozesses** durch schnelle Implementierung bewährter Modelle.
- Häufig praktizierte Vorgehensweise in der Industrie, um auf etablierte Architekturen zurückzugreifen und deren Vorteile zu nutzen.
- Ermöglicht eine schnellere Markteinführung und bessere Nutzung vorhandenen Wissens und bestehender Modelle.

Automatisierte Retrainings bei Datenaktualisierungen

- Implementierung von **Pipelines**, die Modelle **automatisch neu trainieren**, sobald neue Daten verfügbar sind, um die Modellaktualität zu gewährleisten.
- Sicherstellung, dass Modelle kontinuierlich an sich **verändernde Daten angepasst** werden und ihre Performance aufrechterhalten bleibt.
- Vollautomatische ML-Pipelines ermöglichen einen nahtlosen Übergang von Datenaktualisierungen zum Modell-Update.
- Sicherung langfristiger Modellqualität und Anpassungsfähigkeit an dynamische Umgebungen durch kontinuierliches Lernen.
- Reduzierung manueller Eingriffe und **Förderung einer skalierbaren, robusten ML-Infrastruktur**.



Code Demo: MLflow Model Registry Integration in CI/CD Pipeline

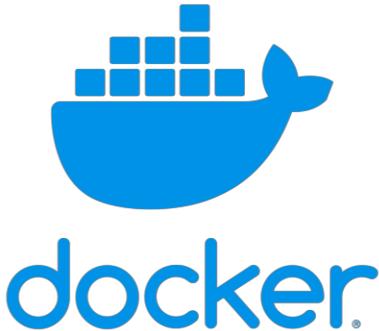
MLOps_und_Deployment/demo_mlflow_model_registry.py

FastAPI: Einfaches Bereitstellen von Inferenz-APIs

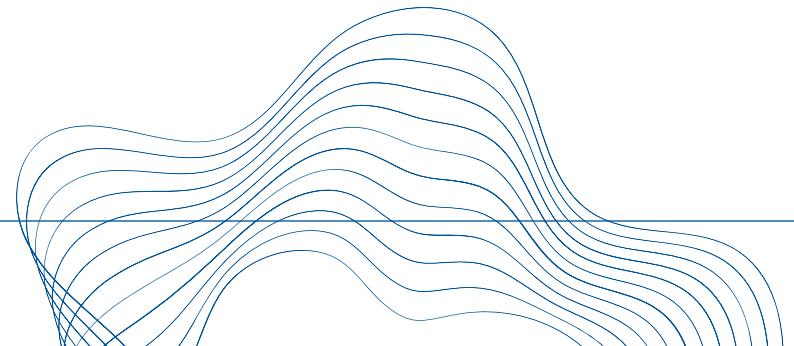
- FastAPI ist ein modernes, schnelles und leichtgewichtiges Python-Framework zur Erstellung von **RESTful APIs**, das auf Standards wie OpenAPI basiert.
- Ermöglicht die nahtlose **Integration von Machine-Learning-Modellen durch einfache Endpunktdefinitionen und automatische Dokumentation**.
- Unterstützt asynchrone Programmierung, was die Verarbeitungsgeschwindigkeit und die Skalierbarkeit von APIs erheblich verbessert.
- **Einfache Skalierbarkeit durch Containerisierung mit Docker** und Orchestrierung mit Kubernetes, was die Bereitstellung in großen Produktionsumgebungen erleichtert.
- Ermöglicht die **Bereitstellung** von Echtzeitvorhersagen mit niedrigen Latenzzeiten, ideal für Anwendungen wie Chatbots, Empfehlungssysteme und Echtzeit-Analysen.



Containerisierung mit Docker: Portabilität und Wiederverwendbarkeit



- Docker ermöglicht das Verpacken von Code, Bibliotheken und Abhängigkeiten in isolierte Container, die auf jeder Plattform konsistent laufen.
- Erleichtert die **Bereitstellung und den Betrieb von Anwendungen** durch Portabilität zwischen Entwicklungs-, Test- und Produktionsumgebungen.
- **Schnelle und reproduzierbare Umgebungen** durch die Verwendung von Docker-Images, die exakt definierte Zustände und Konfigurationen enthalten.
- **Standardmethode für CI/CD-Pipelines**, da Docker-Container nahtlos in automatisierte Build- und Deployment-Prozesse integriert werden können.
- Fördert die **Wiederverwendbarkeit von Komponenten** und erleichtert die **Zusammenarbeit in Teams** durch konsistente Entwicklungsumgebungen.





Code Demo: Modell mit FastAPI deployen

MLOps_und_Deployment/demo_fastapi_deployment.py

Übung Deployment

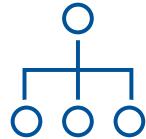
Verbessere das bestehende MNIST Model und lade es unter einer neuen Version in die Mlflow Models Registry.

Deploye und teste das Modell in fastapi.

Best Practices für das Deployment – Teil I



- **Autoscaling** passt die Anzahl der laufenden Instanzen automatisch basierend auf der aktuellen Nachfrage an, um eine konstante Performance sicherzustellen.



- **Load Balancer** verteilen den eingehenden Traffic gleichmäßig auf mehrere Server oder Instanzen, um Überlastungen einzelner Ressourcen zu verhindern.



- **Monitoring:** Kontinuierliches Tracking wichtiger Leistungskennzahlen wie Latenzzeiten, Durchsatz (Anzahl der Anfragen pro Sekunde) und Fehlerraten zur Bewertung der Modellperformance.



- **Incident-Management:** Einrichtung automatisierter Benachrichtigungen bei Überschreiten von Grenzwerten für wichtige Metriken

Best Practices für das Deployment – Teil II



- Implementierung von **Unit- und Integrationstests** zur Sicherung der **Codequalität** und Funktionalität.
- **Canary Deployments** führen neue Modellversionen schrittweise ein, indem sie zunächst an einem kleinen Nutzeranteil ausgerollt werden.
- Entwicklung und Implementierung von **Strategien zum schnellen Zurücksetzen** auf die letzte stabile Modellversion bei auftretenden Qualitätsproblemen.



Übung Fashion MNIST

1. Speichere dein bestes Fashion MNIST Modell in der Mlflow Model Registry
2. Binde das Modell in Fastapi ein.

Agenda

1. Grundlagen des Deep Learning
2. Erweiterte Konzepte und Architekturen
3. Verarbeitung von Textdaten (NLP-Grundlagen)
4. Transformer-Modelle
5. Trainingspraxis und Optimierung
6. MLOps und Deployment
- 7. Anwendungsfälle und Projekte**

Projekte

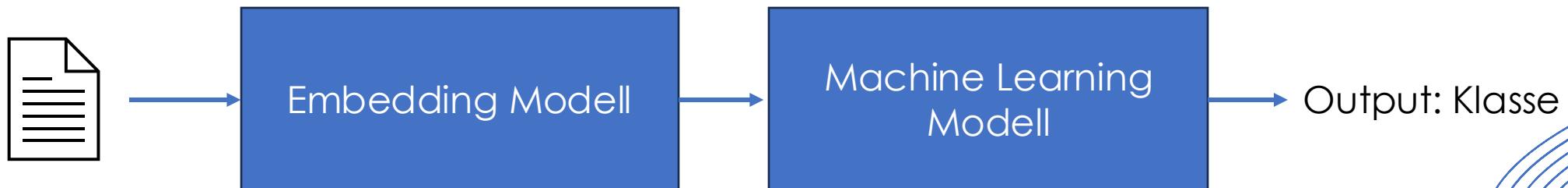
1. Klassifikation
2. Retrieval Augmented Generation
3. Named Entity Recognition

Projekt 1 - Klassifikation

- Klassifikation von Text in vorgegebene Klassen
- **Sentiment Analysis** ist Klassifikation in Klassen „positiv“, „negativ“, „neutral“
- **Themenklassifikation**
- **Spamklassifikation**
- Erkennung von potenziell **risikoreichen Inhalten**

Option 1: Klassifikation mit Embeddings

- **Vortrainiertes Modell** generiert Embeddings
- Auf Basis der Embeddings wird ein **separates Modell** trainiert
- Dies kann ein Neuronales Netz sein, aber auch ein klassisches Machine Learning Modell





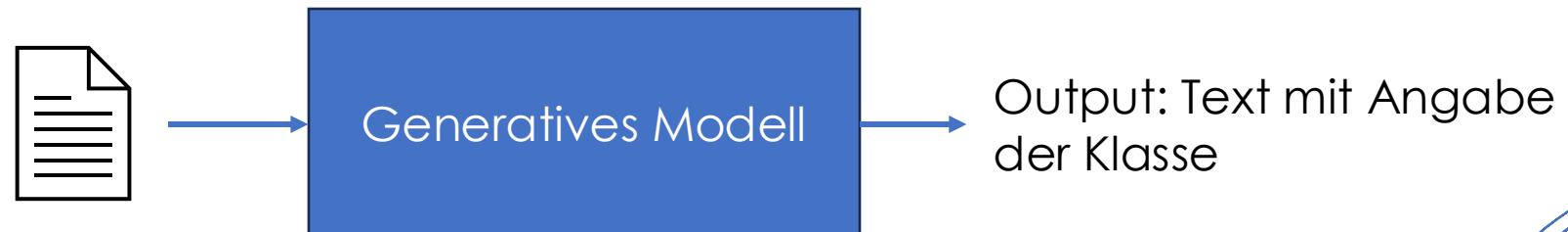
Code Demo: Klassifikation mit Embeddings

Anwendungsfaelle_und_Projekte/Klassifikation/demo_klassifi
kation_mit_embeddings.py

Option 2: Klassifikation mit Generativen Modellen

- Generatives Modell mit **Prompting** nutzen
- **Zero Shot** Prompting – ohne Beispiele im Prompt
- **Few Shot** Prompting: Beispiele im Prompt angeben

Teile den Text in eine der folgenden Klassen ein...



Option 3: Klassifikation mit Fine Tuning

- Training eines Large Language Modells für die Klassifikation
- **Klassifikations-head** auf das Transformer Netzwerk setzen und Modell trainieren
- Im Besten Fall kann ein **vortrainiertes Modell fine-tuned** werden





Code Demo: Fine Tuning eines Klassifikationsmodells

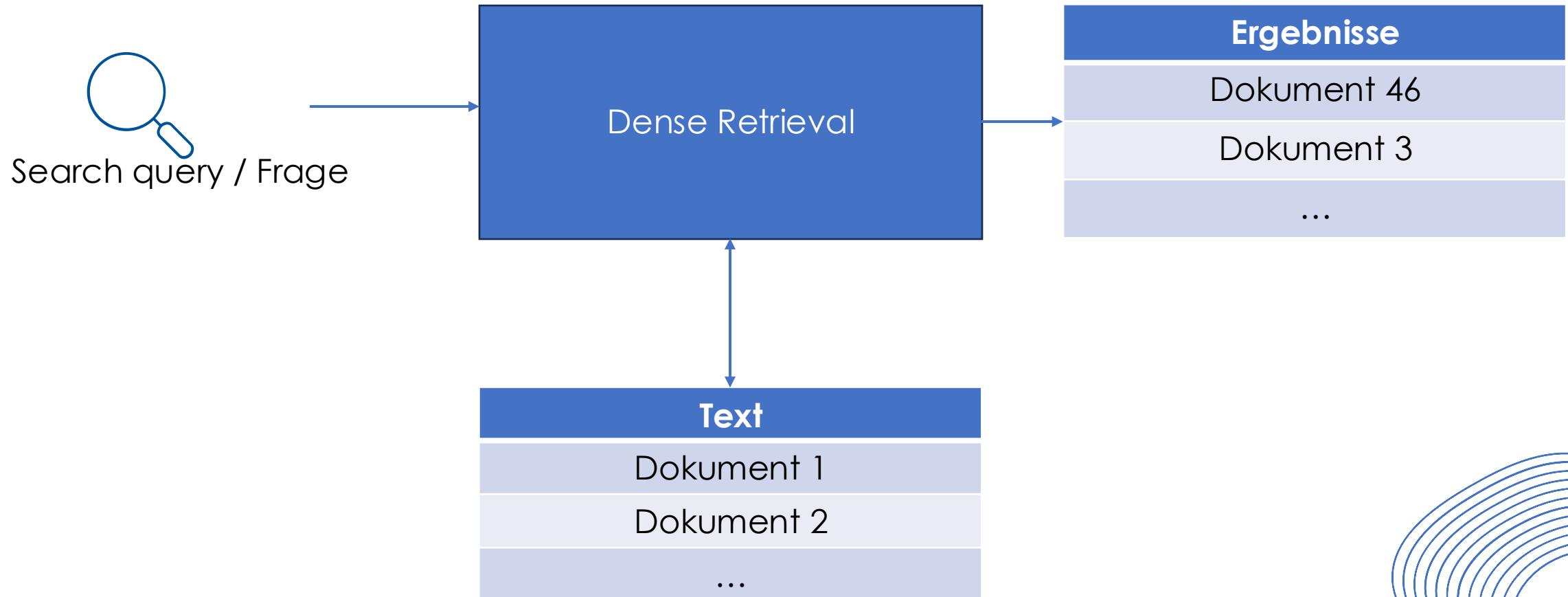
Anwendungsfaelle_und_Projekte/Klassifikation/demo_klassifi
kation_fine_tuning.py

Übung: Sentiment Analysis mit Klassifikationsmodell

Anwendungsfaelle_und_Projekte/Klassifikation/
uebung_klassifikation_fine_tuning.py

1. Suche auf Huggingface ein deutsches Bert Modell.
2. Binde das Modell ein und fine tune es auf dem Datensatz sepidmnorozy/German_sentiment zur Sentiment Analysis.

Projekt 2 – Semantic Search und Retrieval Augmented Generation



Dense Retrieval

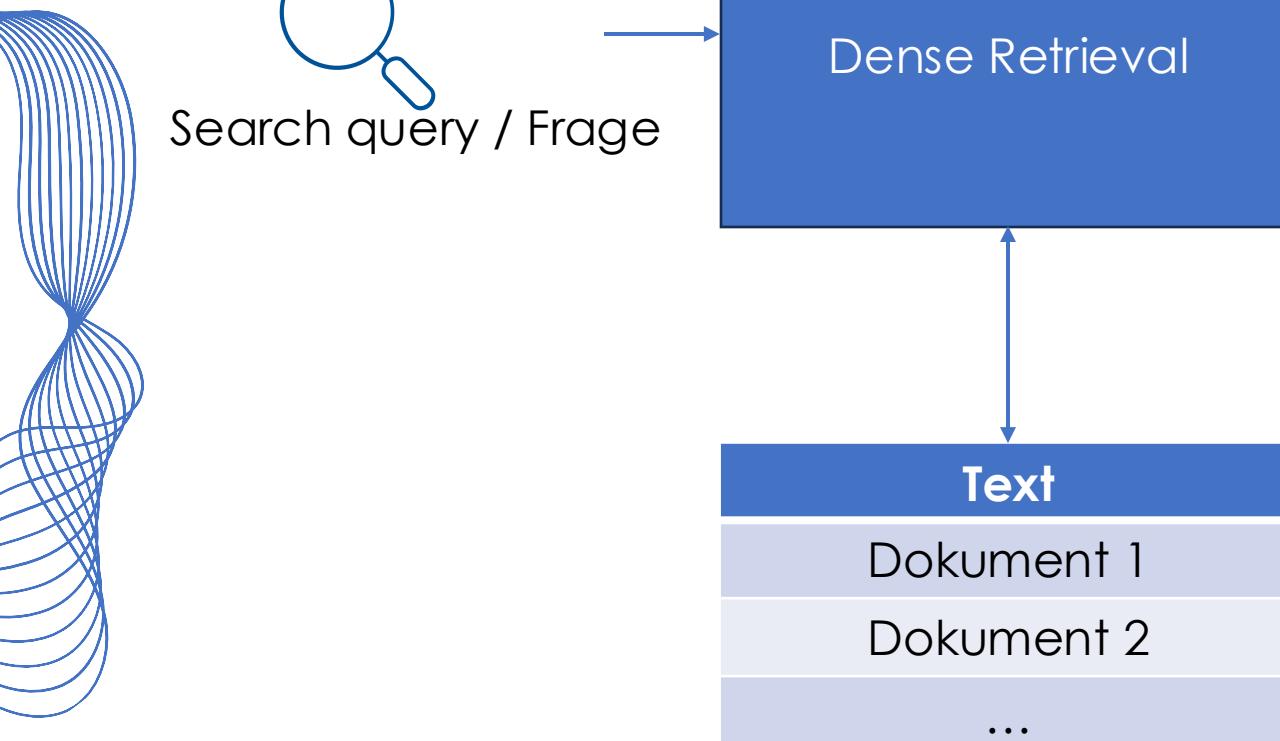
- Beim Dense Retrieval werden die **Ähnlichkeiten der Embedding Vektoren** verglichen.
- Alle relevanten **Dokumente** werden vorher **embedded und abgespeichert**. Dafür wird eine **Vektordatenbank** genutzt, die eine effiziente Suche ermöglicht.
- Die **Search Query** / Frage bildet das **zweite Embedding**.
- Die Embeddings aus der Datenbank, die am ähnlichsten zum Embedding der Search Query sind, werden zurückgegeben. Ähnlichkeit wird durch den Abstand im Embedding Raum bestimmt.



Code Demo: Embeddings in FAISS

Anwendungsfaelle_und_Projekte/Semantic_Search/demo_semantic_search.py

Retrieval Augmented Generation





Code Demo: Retrieval Augmented Generation

Anwendungsfaelle_und_Projekte/Semantic_Search/demo_retrieval_augmented_generation.py

Übung Semantic Search

1. Generiere Embeddings für einen Text deiner Wahl (z.B. einer Wikipedia Seite, die du mit `requests.get()` herunterlädst). Bereinige den Text gegebenenfalls sinnvoll (html Tags etc.)
2. Speichere die Embeddings mit FAISS ab.
3. Stelle Fragen an den Text und beobachte die Qualität der Ergebnisse.

Projekt 3 – Named Entity Recognition

- Named Entity Recognition (NER) bezeichnet die automatisierte **Extraktion und Klassifizierung** von benannten Entitäten wie **Personen, Orten und Organisationen** aus unstrukturierten Texten.
- Essentiell für den Aufbau von Wissensgraphen, indem strukturierte Informationen aus Textquellen gewonnen werden.
- **Unterstützt semantische Suchsysteme** durch die Identifikation relevanter Schlüsselwörter und deren Kategorisierung.
- Ein **grundlegender Task** im Natural Language Processing (NLP), der in vielen Anwendungen wie Informationsgewinnung, Textanalyse und maschinellem Verständnis eine zentrale Rolle spielt.
- Ermöglicht eine tiefere semantische Verarbeitung und Kontextualisierung von Textdaten.

Feintuning von vortrainierten Sprachmodellen (z. B. BERT) für NER

- **Vortrainierte Sprachmodelle** wie BERT bieten eine starke Ausgangsbasis durch das Erlernen von tiefen Sprachrepräsentationen aus großen Textkorpora.
- Feintuning beinhaltet das **Anpassen der finalen Schichten des Modells** auf die spezifische Aufgabe der Entitätserkennung, wodurch die Modellleistung gesteigert wird.
- Ermöglicht höhere Genauigkeit und Robustheit bei der NER, selbst mit begrenzten annotierten Daten.
- Industriestandard für anspruchsvolle NER-Aufgaben, da vortrainierte Modelle **bereits ein umfassendes Sprachverständnis** besitzen.

Anwendungsgebiete: Wissensgraphen, Informationsextraktion

- NER ist entscheidend für den Aufbau von **Wissensgraphen**, indem es strukturierte Informationen aus unstrukturierten Texten extrahiert und Beziehungen zwischen Entitäten herstellt.
- Unterstützt die **Informationsextraktion**, indem relevante Entitäten identifiziert und ihre Beziehungen zueinander analysiert werden.
- Verbessert die Effizienz semantischer Suchsysteme, indem Suchanfragen auf spezifische Entitäten ausgerichtet werden können.
- Anwendung in der Business Intelligence zur **Analyse von Markttrends, Kundenfeedback und Konkurrenzinformationen**.
- **Grundlage für fortgeschrittene NLP-Anwendungen** wie Fragebeantwortungssysteme, Empfehlungssysteme und automatisierte Inhaltsanalyse.



Code Demo: NER-Inferenz mit Hugging Face Pipeline

Anwendungsfaelle_und_Projekte/Named_Entity_Recognition/demo_ner_inferenz.py

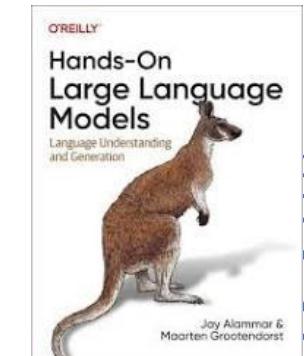
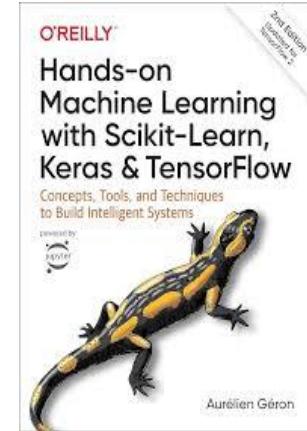


Code Demo: Feintuning von BERT auf NER- Datensatz

Anwendungsfälle_und_Projekte/Named_Entity_Recognitio
n/demo_ner_finetuning.py

Weitere Ressourcen

- Material von Andrej Karpathy:
 - <https://www.youtube.com/@AndrejKarpathy>
- Bücher:
 - Aurélien Géron: Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems
 - Jay Alammar & Maarten Grootendorst: Hands-On Large Language Models



Zum Schluss

- Abschließende Fragen?
- Dateien von Guacamole sichern
 - Feedback?

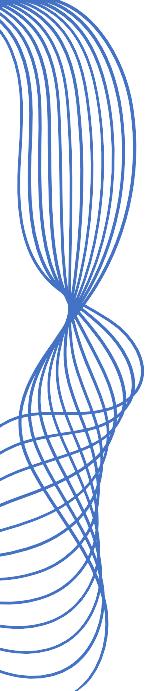
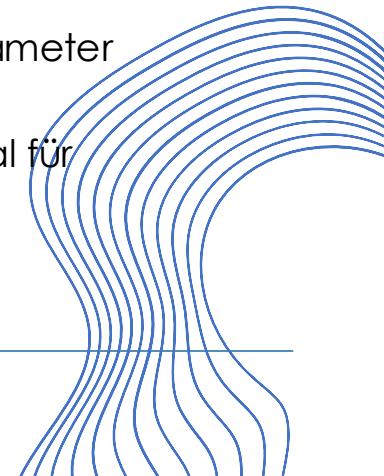


Ende

Bei Rückfragen: nk@data-convolution.de

Glossar

Glossar I

- 
- 
- **Accuracy:** Misst den Anteil korrekt klassifizierter Beispiele, sinnvoll bei ausbalancierten Datensätzen.
 - **Aktivierungsfunktionen:** Entscheiden, welche Signale weitergeleitet werden, um nichtlineare Beziehungen abzubilden. Beispiele sind Sigmoid, ReLU und Tanh.
 - **AUC (Area Under Curve):** Fasst die Modellgüte in einer einzelnen Kennzahl zusammen, basierend auf der ROC-Kurve.
 - **Autoscaling:** Passt die Anzahl der laufenden Instanzen automatisch basierend auf der aktuellen Nachfrage an.
 - **Back-Translation:** Übersetzt einen Satz zunächst in eine andere Sprache und anschließend zurück in die Ausgangssprache, um neue Varianten zu generieren.
 - **Backpropagation:** Verfahren, bei dem der am Ausgang gemessene Fehler rückwärts durch das Netzwerk propagiert wird, um die Gewichte iterativ zu korrigieren.
 - **Bag-of-Words (BoW):** Zählt die Häufigkeit von Wörtern ohne Rücksicht auf Reihenfolge oder Kontext.
 - **Batch Normalization:** Normalisiert die Aktivierungen jeder Schicht, um Schwankungen zu reduzieren und das Training zu beschleunigen.
 - **Batch Size:** Bestimmt die Anzahl der Trainingsbeispiele, die in einem Schritt zur Aktualisierung der Modellparameter verwendet werden.
 - **Bayesian Optimization:** Nutzt Modellierungen des Parameterraums, um gezielt Bereiche mit hohem Potenzial für Hyperparameter zu erkunden.

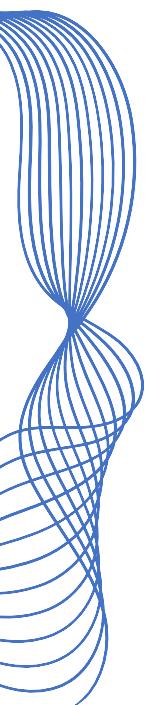
Glossar II

- **BERT:** Bidirektionales Transformer-Modell, das den Kontext von links und rechts eines Tokens berücksichtigt, um tiefe semantische Zusammenhänge zu lernen.
- **BLEU (Bilingual Evaluation Understudy):** Bewertet maschinelle Übersetzungen, indem es die Ähnlichkeit der generierten Sätze mit Referenzübersetzungen quantifiziert.
- **Caching:** Speichert bereits verarbeitete Daten im Arbeitsspeicher, um erneute Berechnungen zu vermeiden.
- **Canary Deployments:** Führen neue Modellversionen schrittweise ein, um das Risiko von Fehlern zu minimieren.
- **CIDEr (Consensus-based Image Description Evaluation):** Misst die Übereinstimmung zwischen generierten und Referenzbeschreibungen, besonders relevant für Bildbeschreibungsaufgaben.
- **CNNs (Convolutional Neural Networks):** Extrahieren lokale Muster (Kanten, Texturen) aus Bildern und finden Anwendung in der Bildklassifikation, Objekterkennung.
- **Cross Entropy:** Ein gängiges Maß für Klassifikationsfehler, das Wahrscheinlichkeitsverteilungen betrachtet.
- **Cross-Attention:** Schichten im Decoder beziehen Informationen aus dem Encoder-Ausgaberaum ein, um den generierten Output auf den Input-Kontext abzustimmen.
- **Cyclical Learning Rates:** Periodische Schwankungen der Lernrate helfen, aus lokalen Minima herauszufinden und die Konvergenz zu verbessern.

Glossar III

- **Data Parallelism:** Aufteilen von Batches auf mehrere GPUs, um das Training zu beschleunigen.
- **Datenaugmentation:** Erweitert den Trainingsdatensatz künstlich, um die Datenvielfalt und Robustheit des Modells zu erhöhen.
- **Dependency Parsing:** Identifiziert Abhängigkeitsbeziehungen zwischen Wörtern, etwa welches Wort Subjekt oder Objekt eines Verbs ist.
- **Dropout:** Schaltet Neuronen zufällig aus, um Overfitting zu vermeiden und die Robustheit des Modells zu erhöhen.
- **Early Stopping:** Beendet das Training bevor sich das Modell zu stark an die Trainingsdaten anpasst, um Overfitting zu vermeiden.
- **Elasticsearch:** Verwendet invertierte Indizes für schnelle Volltextsuche.
- **Encoder-Struktur:** Besteht aus gestapelten Attention- und Feed-Forward-Schichten, die Eingabe-Embeddings in kontextreiche Repräsentationen transformieren.
- **Epoche:** Ein vollständiger Durchlauf des gesamten Trainingsdatensatzes.
- **Exploding Gradients:** Der Gradient steigt rasant an, was zu instabilen Gewichtsupdates führt.
- **Extraktive Zusammenfassung:** Wählt wichtige Sätze aus dem Originaltext aus, um eine verkürzte Version zu erstellen.

Glossar IV

- 
- 
- **F1-Score:** Harmonisches Mittel aus Precision und Recall, besonders nützlich bei unausgewogenen Datensätzen.
 - **FastAPI:** Ein Python-Framework zur Erstellung von RESTful APIs.
 - **Fine-Tuning:** Feinabstimmung eines vortrainierten Modells auf eine neue Aufgabe.
 - **FP16 (16-Bit Floating Point):** Verwendet 16-Bit-Gleitkommazahlen, was das Training beschleunigt, aber die numerische Präzision verringert.
 - **FP32 (32-Bit Floating Point):** Verwendet 32-Bit-Gleitkommazahlen für genauere Berechnungen, ist aber ressourcenintensiver.
 - **GANs (Generative Adversarial Networks):** Bestehen aus einem Generator, der künstliche Daten erzeugt, und einem Diskriminator, der zwischen echten und künstlichen Daten unterscheidet.
 - **Generative Zusammenfassung:** Erstellt neue, komprimierte Textfassungen, die den Inhalt des Originals wiedergeben.
 - **Glorot-Initialisierung (Xavier):** Sorgt für ausgeglichene Varianzverteilungen in den Schichten.
 - **Gradientenabstieg:** Verfahren, bei dem die Gewichte in Richtung des abnehmenden Fehlers angepasst werden.
 - **Grid Search:** Durchsucht systematisch den Parameterraum für Hyperparameter-Tuning.

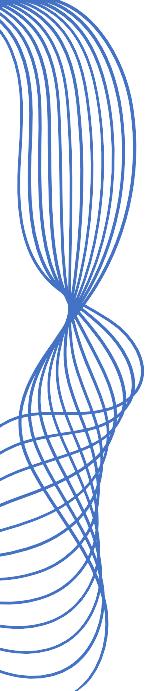
Glossar V

- **GRU (Gated Recurrent Unit)**: Eine erweiterte RNN-Variante mit internen Gating-Mechanismen zur besseren Erfassung von Langzeitabhängigkeiten.
- **GPT (Generative Pre-trained Transformer)**: Autoregressives Modell für Textgenerierung, das das nächste Wort auf Basis der bisherigen Tokens vorhersagt.
- **Haystack**: Ein Open-Source-Framework zur Entwicklung von Fragebeantwortungssystemen.
- **He-Initialisierung**: Speziell für ReLU-Schichten konzipiert und unterstützt schnelles, stabiles Training.
- **Hidden-Layer**: Extrahieren schrittweise abstrakte Merkmale aus den Eingangsdaten.
- **Hugging Face Transformers**: Bietet eine breite Auswahl an vortrainierten Modellen und Tools für NLP.
- **Input-Layer**: Nimmt die Rohdaten auf und gibt sie an die nächsten Schichten weiter.
- **Iteration**: Einzelner Schritt der Gewichtsaktualisierung pro Batch.
- **L1-Regularisierung**: Fügt Strafen für große Gewichte hinzu, um Overfitting zu vermeiden.
- **L2-Regularisierung**: Fügt Strafen für große Gewichte hinzu, um Overfitting zu vermeiden.
- **Layer Normalization**: Normalisiert Eingaben jeder Schicht, stabilisiert Training und begünstigt Konvergenz.

Glossar VI

- **Learning Rate:** Bestimmt die Größe der Gewichtsänderungen pro Schritt während des Trainings.
- **Learning Rate Schedules:** Steuern die Anpassung der Lernrate während des Trainings.
- **Lemmatisierung:** Reduziert Wörter auf ihre Grundform (Lemma).
- **Load Balancer:** Verteilen den eingehenden Traffic gleichmäßig auf mehrere Server, um Überlastungen zu verhindern.
- **Loss-Funktion:** Bestimmt über ihren Gradienten die Richtung des Lernschrittes. Beispiele sind MSE und Cross Entropy.
- **LSTM (Long Short-Term Memory):** Eine erweiterte RNN-Variante, die durch interne Gating-Mechanismen Langzeitabhängigkeiten besser erfasst.
- **MAE (Mean Absolute Error):** Betrachtet den absoluten Fehler und ist robuster gegenüber Ausreißern, oft bei Regressionen gebräuchlich.
- **Masked Language Modeling:** Maskiert Tokens im Input, um das Modell zu zwingen, kontextuelle Hinweise intensiver zu nutzen.
- **Masking:** Markiert gepaddete Bereiche, sodass das Modell diese ignorieren kann.
- **METEOR (Metric for Evaluation of Translation with Explicit ORdering):** Berücksichtigt Wortstämme, Synonyme und Wortreihenfolge stärker als BLEU.
- **MLflow:** Bietet ein zentrales UI, um Experimente, Parameter, Metriken und Artefakte zu verwalten.
- **MLOps:** Bezeichnet die Praktiken zur Automatisierung und zum Management von Machine-Learning-Modellen.

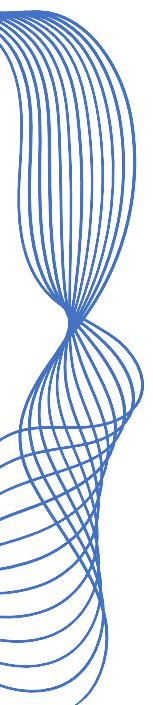
Glossar VII

- 
- 
- **Model Parallelism:** Zerlegung großer Modelle auf verschiedene GPUs oder Maschinen, um den Speicherbedarf je Gerät zu reduzieren.
 - **MSE (Mean Squared Error):** Wird häufig bei Regressionsaufgaben als Fehlermaß eingesetzt.
 - **Multi-Task Learning:** Trainiert mehrere verwandte Aufgaben gleichzeitig, um gemeinsame Strukturen besser auszunutzen.
 - **MUSE:** Mehrsprachige Embeddings, die Wörter aus verschiedenen Sprachen in einen gemeinsamen Vektorraum projizieren.
 - **N-Gramme:** Bilden Folgeeinheiten von mehreren Wörtern und berücksichtigen damit rudimentär die Wortreihenfolge.
 - **Named Entity Recognition (NER):** Automatisierte Extraktion und Klassifizierung von benannten Entitäten wie Personen, Orten und Organisationen.
 - **Next-Token Prediction:** Modell sagt das nächste Wort auf Basis des bisherigen Kontextes vorher.
 - **NLTK:** Ein umfangreiches Tool für NLP, stark im akademischen Kontext.
 - **Output-Layer:** Liefert die finale Vorhersage, beispielsweise eine Klasse oder einen numerischen Wert.
 - **Overfitting:** Modell passt sich zu stark an die Trainingsdaten an und verfehlt die Generalisierung.
 - **Padding:** Fügt künstliche Platzhalter hinzu, um alle Sequenzen auf eine einheitliche Länge zu bringen.

Glossar VIII

- **Perceptron:** Ein einfaches, lineares Modell für binäre Klassifikationsaufgaben.
- **Perplexity:** Misst, wie gut ein Sprachmodell Wahrscheinlichkeitsverteilungen von Wortfolgen abbildet.
- **Pooling Layer:** Reduziert die räumlichen Dimensionen und fasst Informationen zusammen.
- **POS-Tagging (Part-of-Speech-Tagging):** Weist jedem Wort seine Wortart zu (Verb, Nomen, Adjektiv usw.).
- **Positional Encoding:** Kodiert relative Wortpositionen, um den Transformer ein Verständnis für Wortreihenfolgen zu vermitteln.
- **Precision:** Definiert den Anteil der als positiv vorhergesagten Beispiele, die tatsächlich positiv sind.
- **Prefetching:** Lädt Daten im Hintergrund, während das Modell trainiert, um Leerlaufzeiten der GPU zu minimieren.
- **Prompts:** Dienen als Eingabeanweisungen, die das Verhalten und die Antworten des Modells steuern.
- **Random Search:** Probiert zufällig gewählte Parameterkombinationen für Hyperparameter-Tuning.
- **Recall:** Misst, wie viele der tatsächlich positiven Beispiele korrekt erkannt werden.
- **ReLU (Rectified Linear Unit):** Eine Aktivierungsfunktion, die einfach ist, zum Teil das Verschwinden von Gradienten verhindert und sehr verbreitet ist.
- **Residual Connections:** Leiten Informationen um tiefe Schichten herum und ermöglichen so stabileres Training.

Glossar IX

- 
- 
- **Retriever-Reader Workflow:** Teilt das Fragebeantwortungssystem in zwei Komponenten: den Retriever zur Dokumentsuche und den Reader zur Antwortextraktion.
 - **ROC-Kurve:** Illustriert das Verhältnis zwischen True Positive Rate und False Positive Rate.
 - **ROUGE (Recall-Oriented Understudy for Gisting Evaluation):** Fokussiert auf den N-Gramm-Overlap zwischen maschineller Zusammenfassung und Referenz.
 - **RNNs (Recurrent Neural Networks):** Verarbeiten Sequenzen schrittweise, indem sie Informationen über zeitliche Zusammenhänge in einem internen Zustand speichern.
 - **Self-Attention:** Ermöglicht, alle Tokens einer Sequenz parallel zu betrachten und relevante Zusammenhänge direkt zu gewichten.
 - **seq2seq-Transformer:** Nutzen ein Encoder-Decoder-Design, um Eingabesequenzen in Ausgabesequenzen zu transformieren.
 - **Sigmoid:** Wandelt Eingaben in Werte zwischen 0 und 1 um, ideal für Wahrscheinlichkeitsinterpretationen.
 - **SpaCy:** Bietet schnelle, produktionsreife Pipelines für Tokenisierung, POS-Tagging und Parsing.
 - **Stemming:** Verkürzt Wörter auf ihre Wortstämme, ohne auf grammatische Korrektheit zu achten.
 - **Step Decay:** Die Lernrate wird nach festgelegten Epochen reduziert.
 - **Stanza:** Ein NLP-Tool, das moderne, neuronale Modelle nutzt und mehrere Sprachen abdeckt.

Glossar X

- **Statische Wort-Embeddings:** Projizieren jedes Wort in einen kontinuierlichen Vektorraum, in dem semantisch ähnliche Wörter nah beieinander liegen. Beispiele sind Word2Vec und GloVe.
- **Synonym Replacement:** Ersetzt ausgewählte Wörter durch ihre Synonyme.
- **Tanh:** Gibt Werte zwischen -1 und 1 aus, ist um 0 zentriert und oft leistungsfähiger als Sigmoid.
- **Tensor Cores:** NVIDIA Tensor Cores sind für schnelle Matrixmultiplikationen optimiert und profitieren stark von Mixed Precision.
- **TensorBoard:** Tool zur Visualisierung von Loss- und Metrikverläufen.
- **TF-IDF (Term Frequency - Inverse Document Frequency):** Gewichtet seltene, aber aussagekräftige Wörter höher.
- **Tokenisierung:** Die Aufspaltung von Texten in Tokens (Wörter oder Subwörter).
- **Transformer-Modelle:** Setzen auf Self-Attention und umgehen die Nachteile von RNNs.
- **Transfer Learning:** Nutzt bereits erlernte Merkmalsrepräsentationen aus einem vortrainierten Modell.
- **Underfitting:** Das Modell ist zu einfach und erkennt die zugrundeliegenden Muster nicht ausreichend.
- **Validierungsmetriken:** Zeigen, wie gut das Modell auf unbekannte Daten generalisiert.
- **Vanishing Gradients:** Der berechnete Gradient nimmt mit zunehmender Schichtentiefe stark ab.

Glossar XI

- **Warmup:** Zu Beginn wird die Lernrate langsam angehoben.
- **Weights & Biases (W&B):** Eine cloudbasierte Lösung für das Tracken und Visualisieren von Experimenten.
- **WordPiece:** Ein Subwortverfahren, das seltene oder komplexe Wörter in häufig vorkommende Bestandteile zerlegt.
- **Word2Vec:** Ein Modell zur Erzeugung statischer Wort-Embeddings.
- **XLM-R:** Mehrsprachige Embeddings, die einen direkten Vergleich von Bedeutungen zwischen Sprachen ermöglichen.