

Werkzeuge für das wissenschaftliche Arbeiten

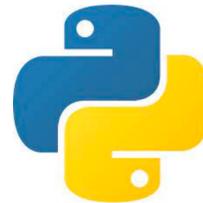
Python for Machine Learning and Data Science

Magnus Bender
bender@ifis.uni-luebeck.de
Wintersemester 2022/23

Inhaltsübersicht

1. Programmiersprache Python

- a) *Einführung, Erste Schritte*
- b) *Grundlagen*
- c) *Fortgeschritten*



2. Auszeichnungssprachen

- a) *LaTeX, Markdown*



3. Benutzeroberflächen und Entwicklungsumgebungen

- a) *Jupyter Notebooks lokal und in der Cloud (Google Colab)*

4. Versionsverwaltung

- a) *Git, GitHub*



5. Wissenschaftliches Rechnen

- a) *NumPy, SciPy*



6. Datenverarbeitung und -visualisierung

- a) *Pandas, matplotlib, NLTK*

7. Machine Learning (scikit-learn)

- a) *Grundlegende Ansätze (Datensätze, Auswertung)*
- b) *Einfache Verfahren (Clustering, ...)*

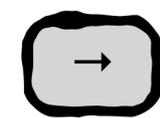


8. DeepLearning

- a) **TensorFlow, PyTorch, HuggingFace Transformers**

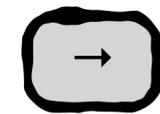


Themen



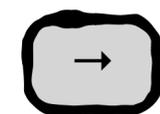
I. Projektaufgaben

1. Lösungsvorschlag Aufgabe 3
2. Lösungsvorschlag Aufgabe 4
3. Herangehensweise & Tipps Aufgabe 5



II. Deep Learning

1. Perzeptron
2. Mehrschichtige Netzwerke



III. Lehrevaluation



IV. Transformer Sprachmodelle

1. Idee
2. BERT & GPT mit Python

***Heute &
nächste Woche***



I. Projektaufgaben

1. Lösungsvorschlag Aufgabe 3

Projektaufgabe 3

„Git, Markdown und LaTeX“

- Vorstellung möglicher Lösungen
(werden nicht in Moodle hochgeladen!)



I. Projektaufgaben

2. Lösungsvorschlag Aufgabe 4

Projektaufgabe 4

„Datenverarbeitung und -darstellung“

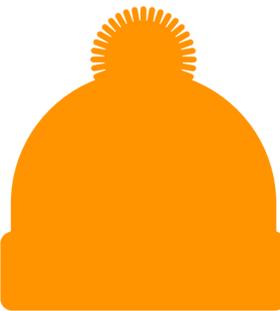
- Vorstellung möglicher Lösungen
(werden nicht in Moodle hochgeladen!)



I. Projektaufgaben

3. Herangehensweise & Tipps Aufgabe 5

„Bonusaufgabe“



Projektaufgabe 5

„Maschinelles Lernen und Data Science“

- Jupyter-Notebook
 - Z.B. in VS Code oder Google Colab
- 1. Visualisierung von Clustering
 - Zwei Datensätze gegeben
 - *K*-Means und DBSCAN
 - Vergleich der Verfahren
- 2. Sprachverarbeitung
 - Modell zur Anfragebeantwortung
 - 20 Newsgroups und *k*-Means oder Kosinusähnlichkeit
- Notebook **mit Ausgaben** im Moodle hochladen

Herangehensweise & Tipps

- 2er Gruppen
 - „erfolgreich bearbeitet“ oder „nicht erfolgreich bearbeitet“
- ## 1. Clustering
- Beispiele aus Vorlesung nutzen
 - Parameter aus dem Notebook für DBSCAN nutzen

2. Sprachmodell

- Training und Anfragebeantwortung in der Klasse ergänzen
- NLTK Vorverarbeitung
- k-Means erfordert Zuordnung der Cluster zu Kategorien
- Bewertet wird nicht, wie gut das Modell auf den Daten funktioniert

II.

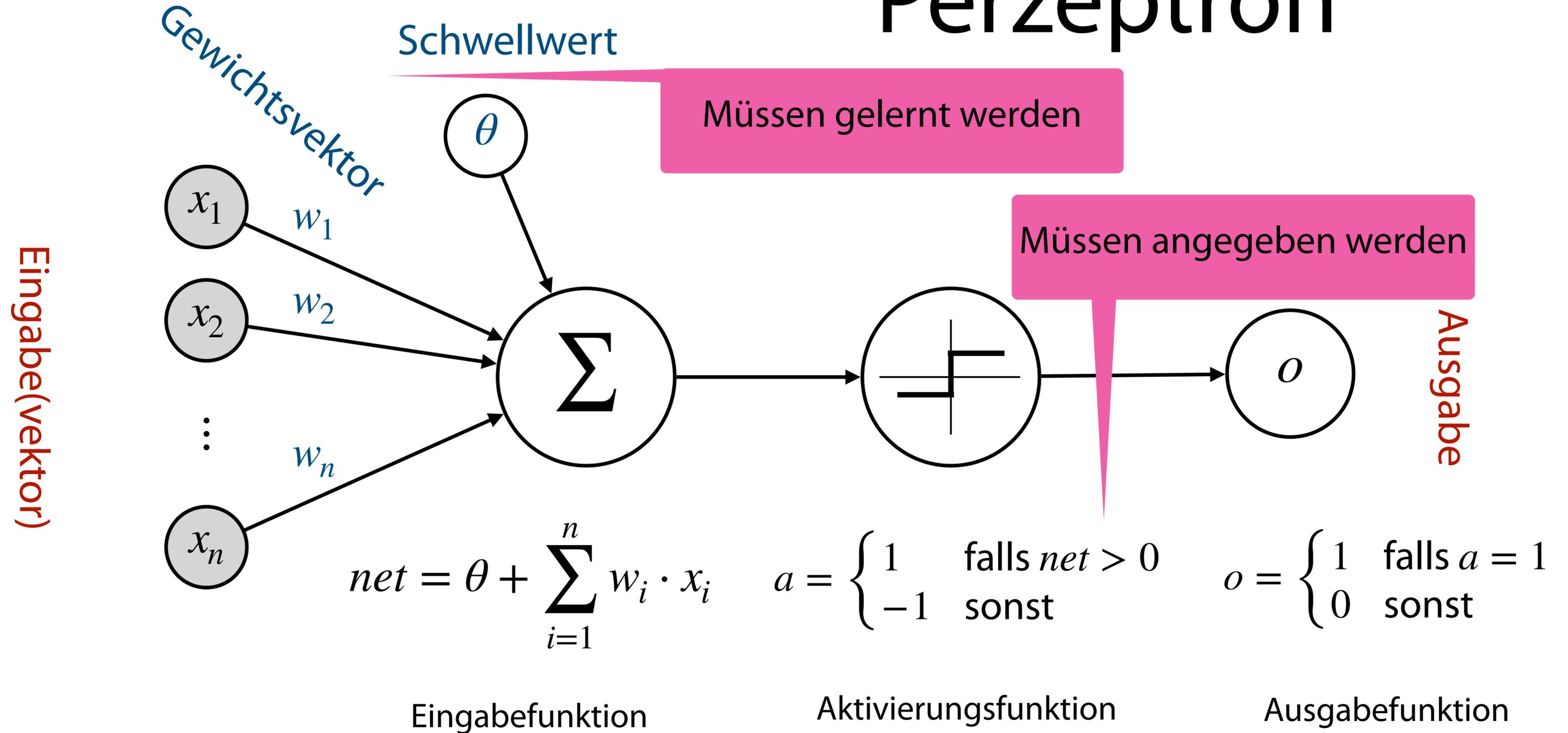
Deep Learning

1. Perzeptron

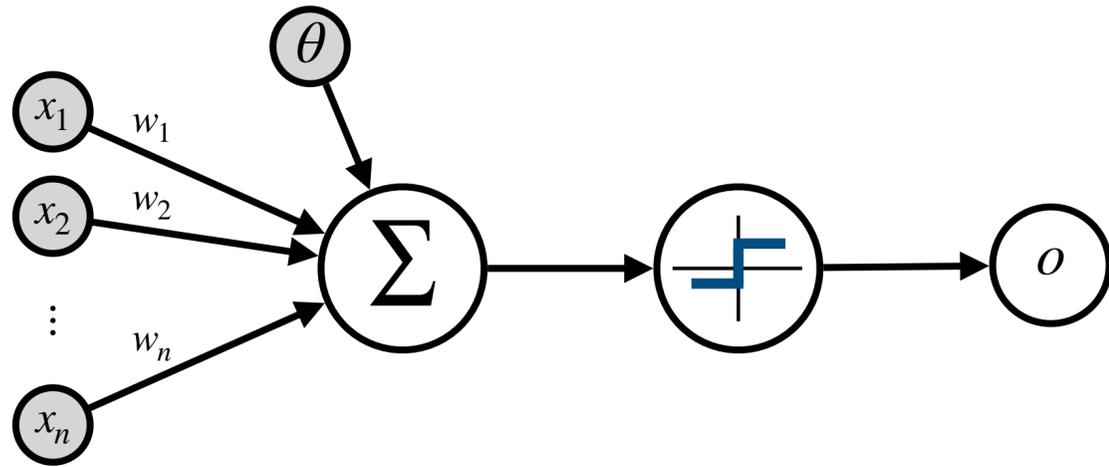
Danksagung

- Nachfolgende Folien sind teilweise übernommen aus folgenden Vorlesungen und Vorträgen
 - Prof. Ralf Möller: „Non-Standard Datenbanken und Data-Mining“
 - Dr. Marcel Gehrke, Prof. Ralf Möller: „Einführung in Web und Data Science“

Perzeptron



Perzeptron

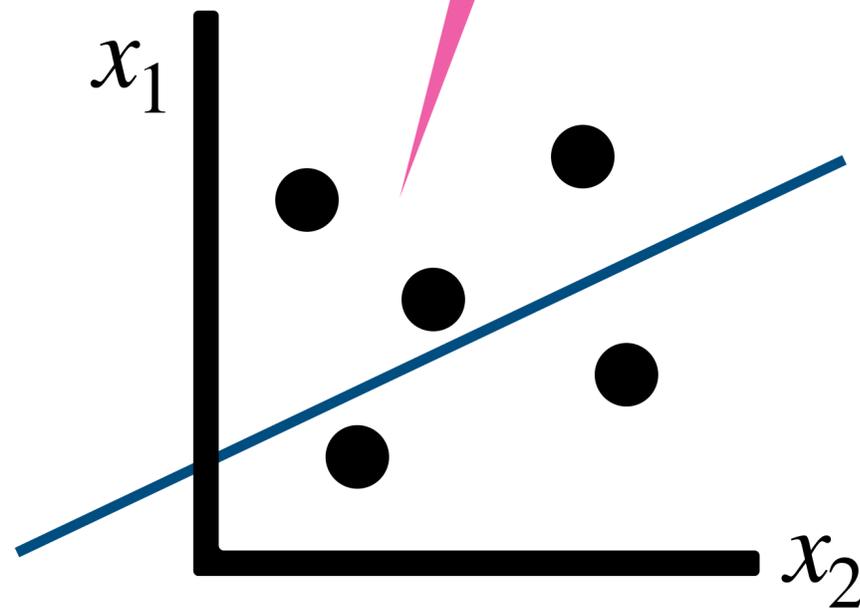


Skalarprodukt

Zusammengefasst

$$o(\vec{x}) = \begin{cases} 1 & \text{falls } \vec{x} \cdot \vec{w} > \theta \\ 0 & \text{sonst} \end{cases}$$

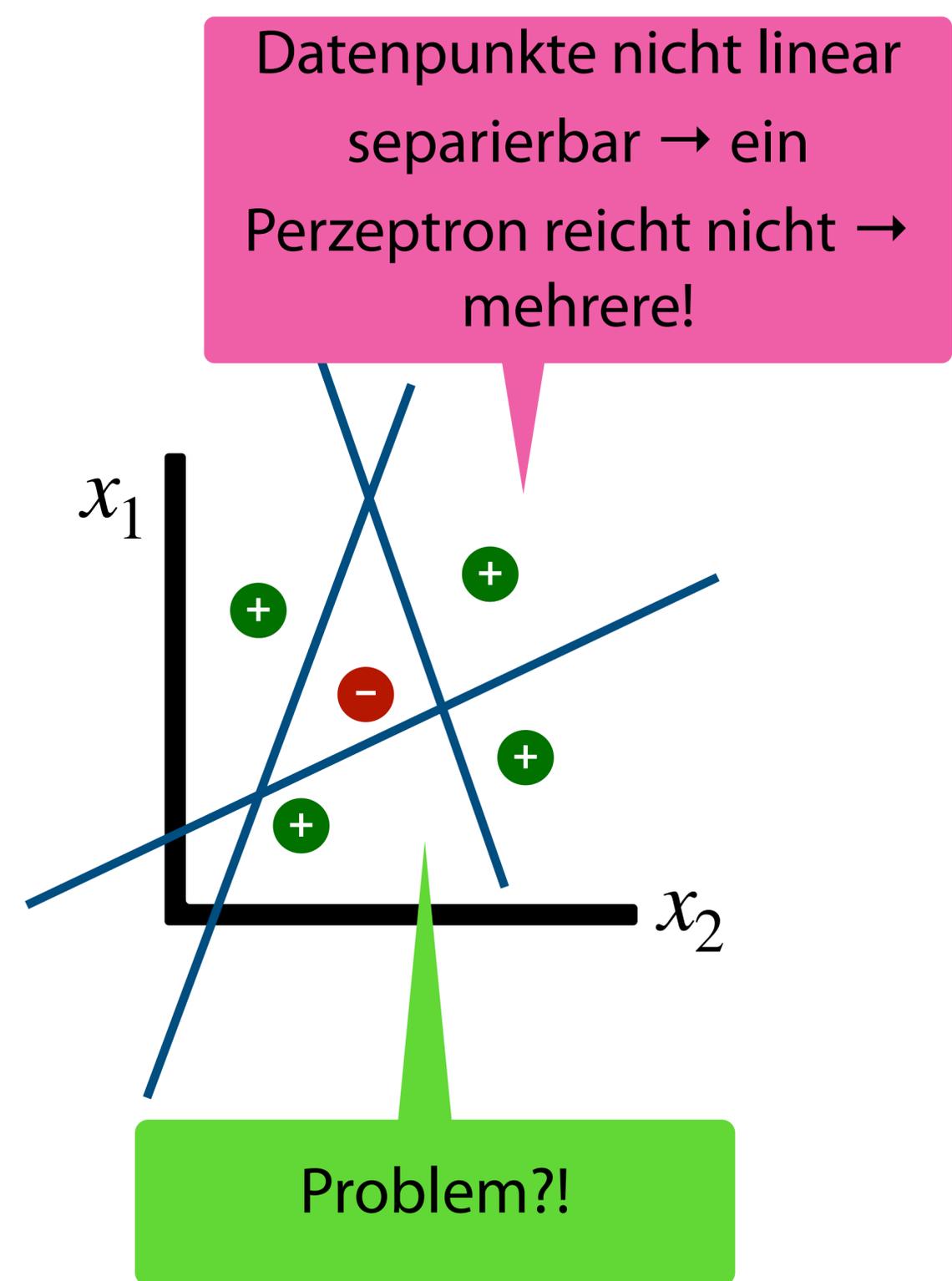
Hier z.B. 2 Dimensionen



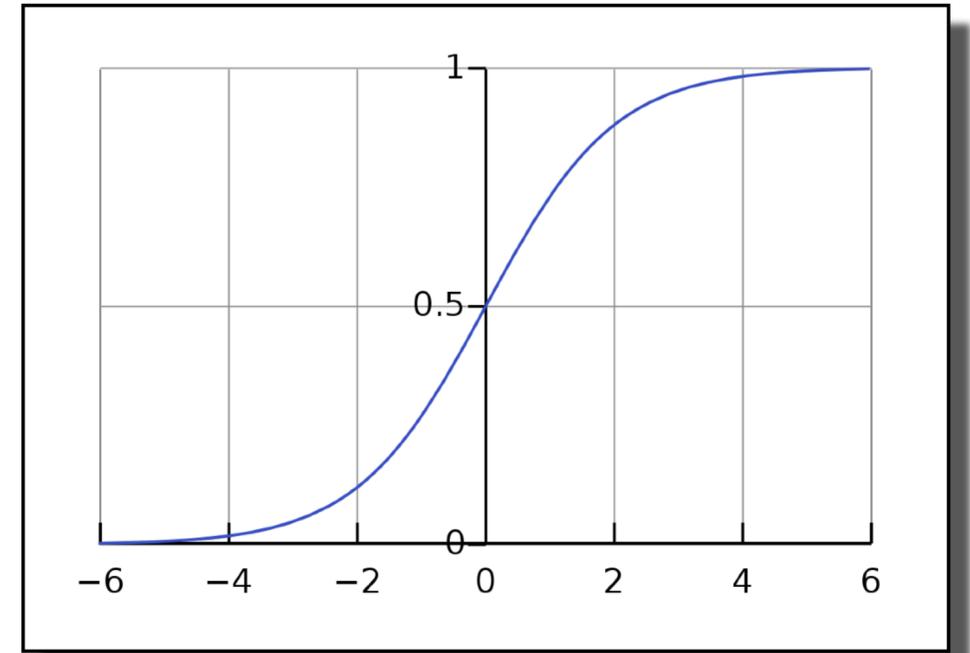
Lineare Funktion

Perzeptron

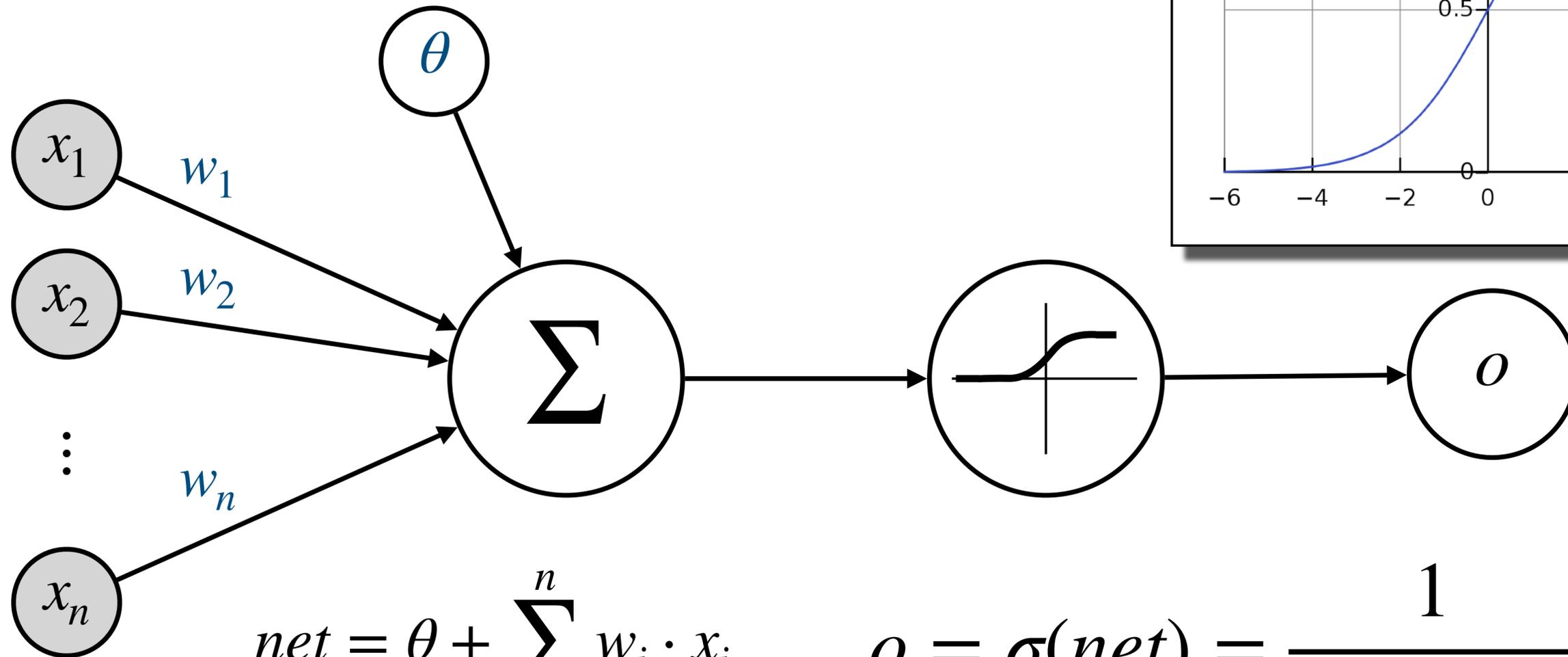
- Lineare Funktion zur Klassifikation
- Eingabe eines mehrdimensionalen Vektors \vec{x}
- Ausgabe $o \in \{0,1\}$ (häufig auch $o \in \{-1,1\}$)
- Parameter
 - Gewichte \vec{w} und Schwellwert θ
 - *Müssen bestimmt/ gelernt werden*



Perzeptron



Eingabe(vektor)



Ausgabe

$$net = \theta + \sum_{i=1}^n w_i \cdot x_i \quad o = \sigma(net) = \frac{1}{1 + e^{-net}}$$

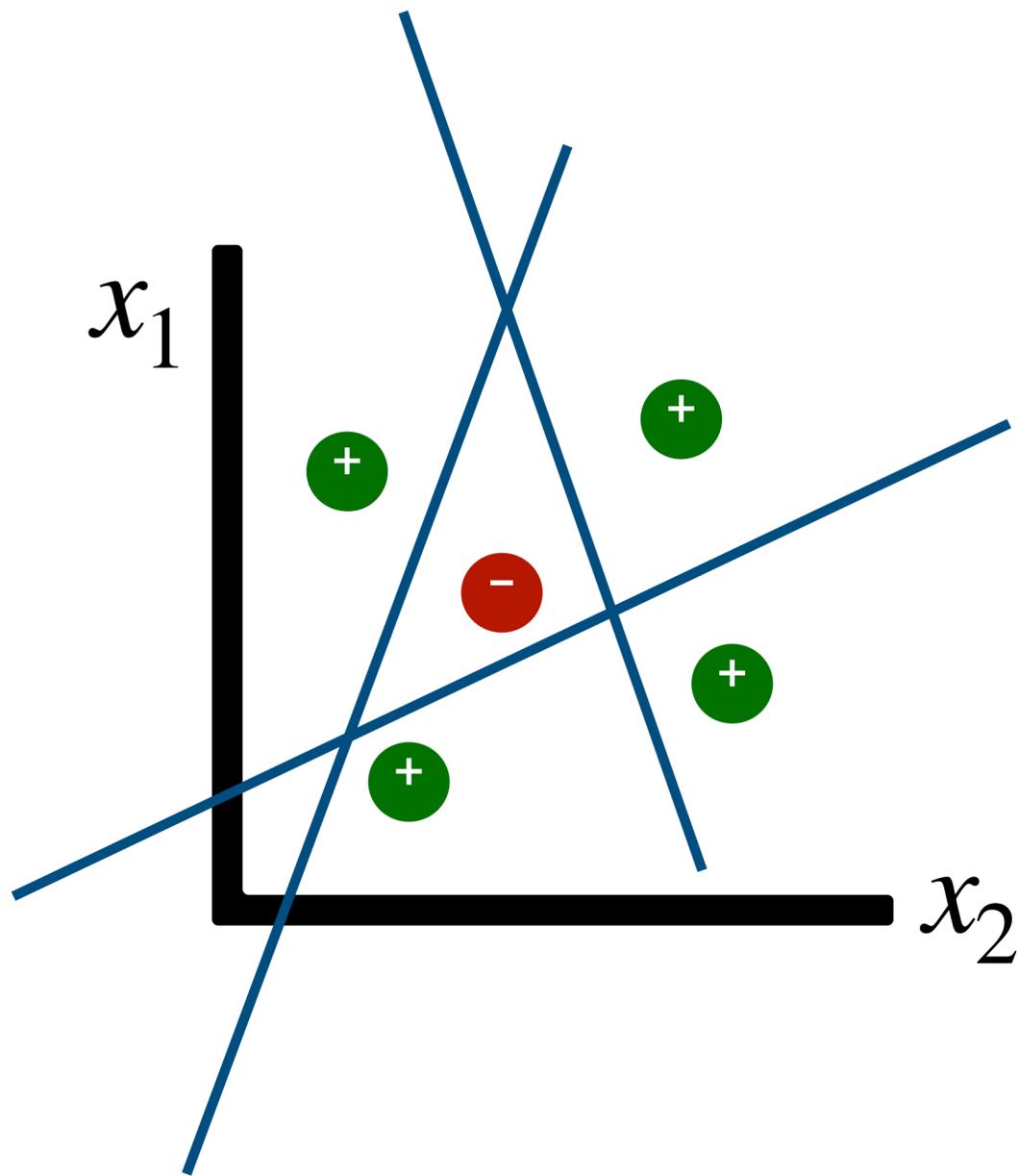
Gradient kann bestimmt werden!

II. Deep Learning

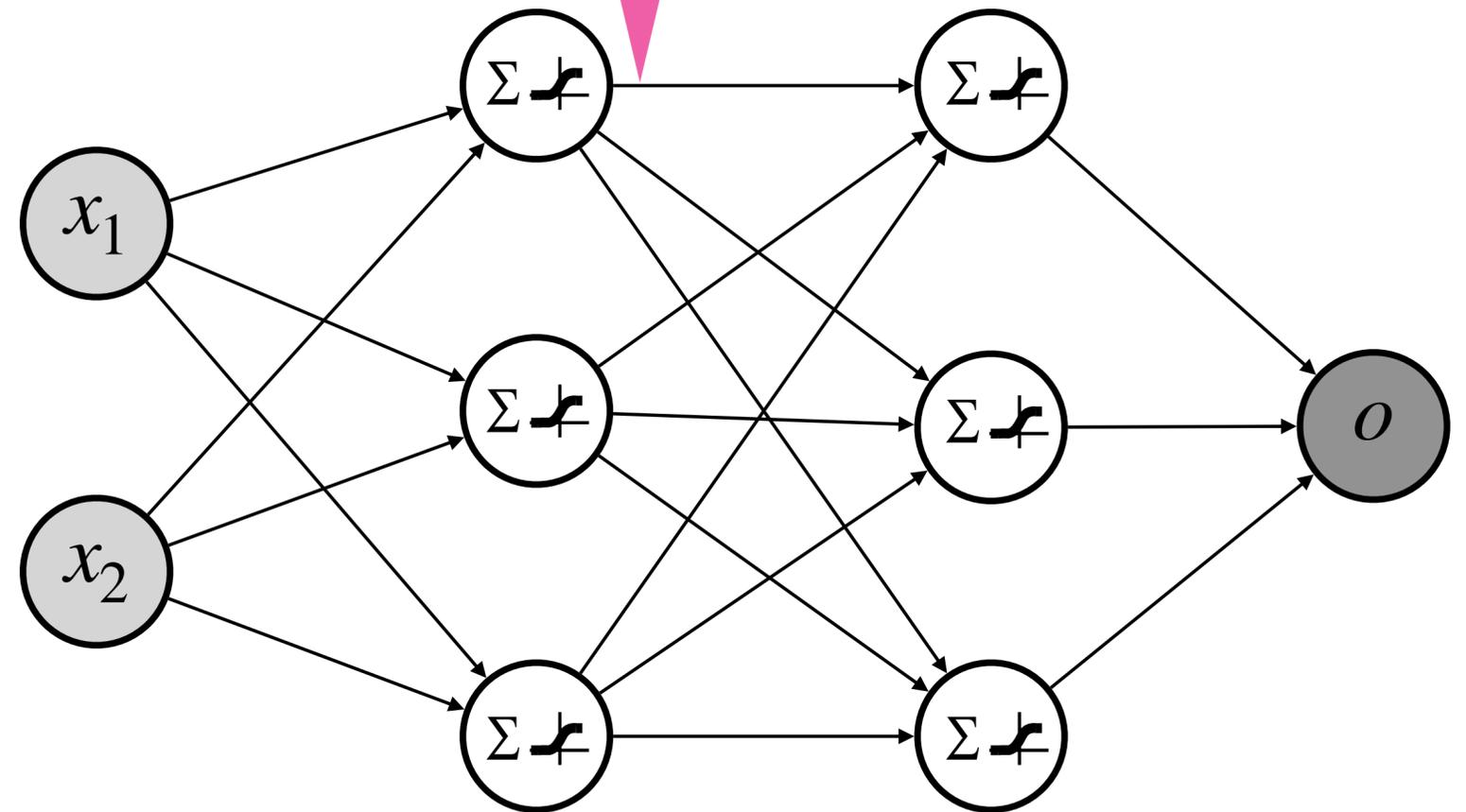
2. Mehrschichtige Netzwerke

Kombination von

Kombination linearer Klassifikatoren
→ mehrere Perzeptrone hintereinander
→ Kanten sind mit den Gewichten belegt
(ein \vec{v} pro Perzeoptron)



Eingabeschicht



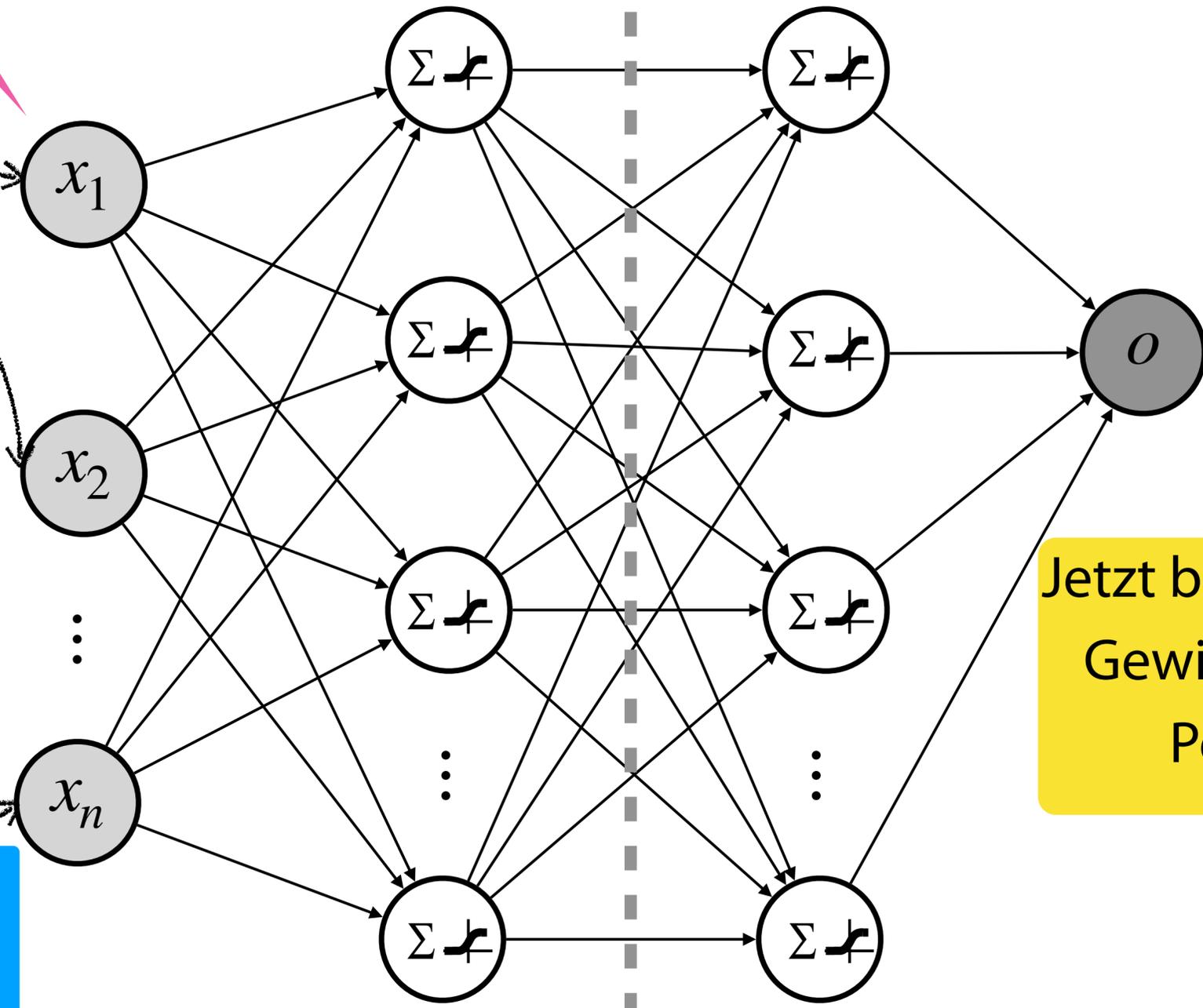
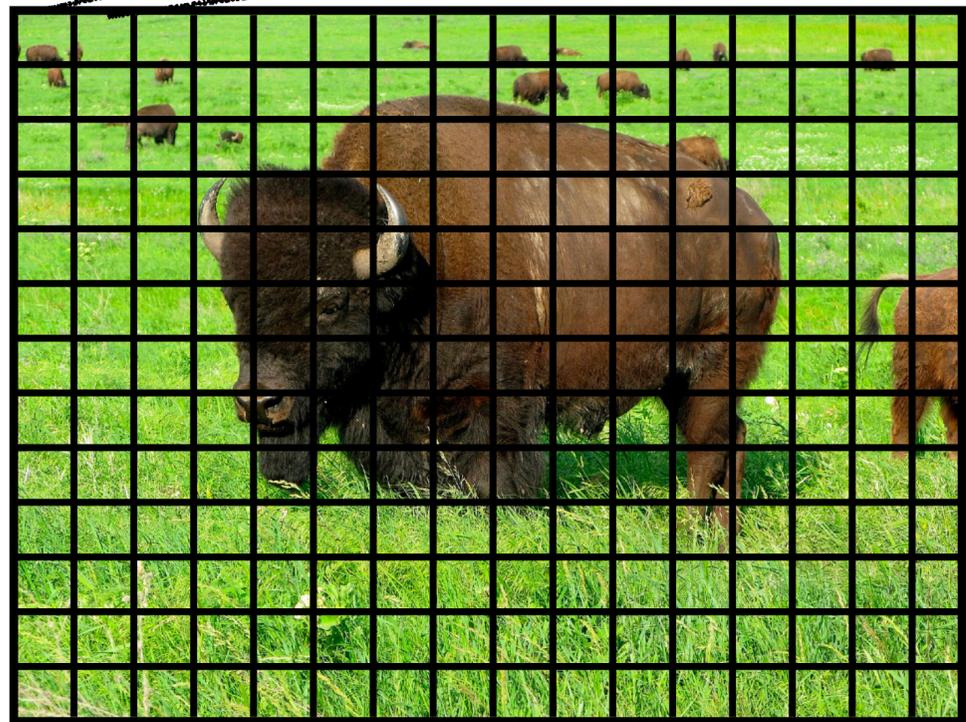
Ausgabeschicht

„Hidden Layer“

Beispiel

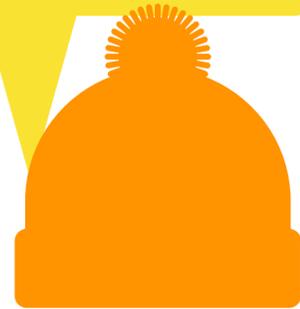
Eingabevektor mit Wert für jeden Pixel

Womöglich weitere „Hidden Layer“



„Wisent?“

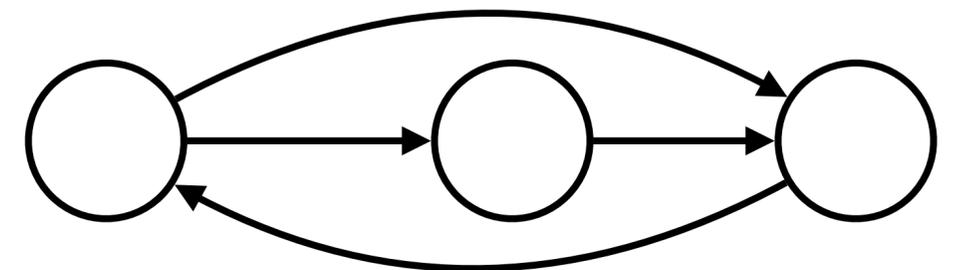
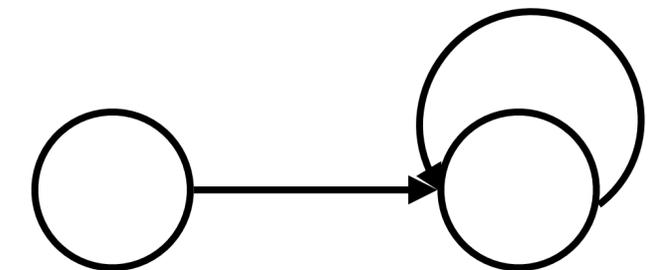
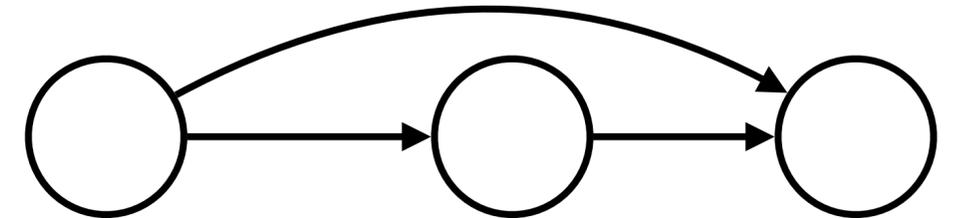
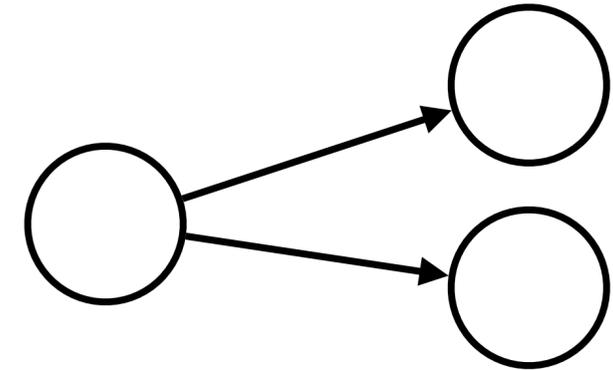
Jetzt brauchen wir die Gewichte für jedes Perzeptron!



- Gegeben viele Bilder verschiedener Tiere
- Ergebnis zu „Wisent?“ für jedes Bild

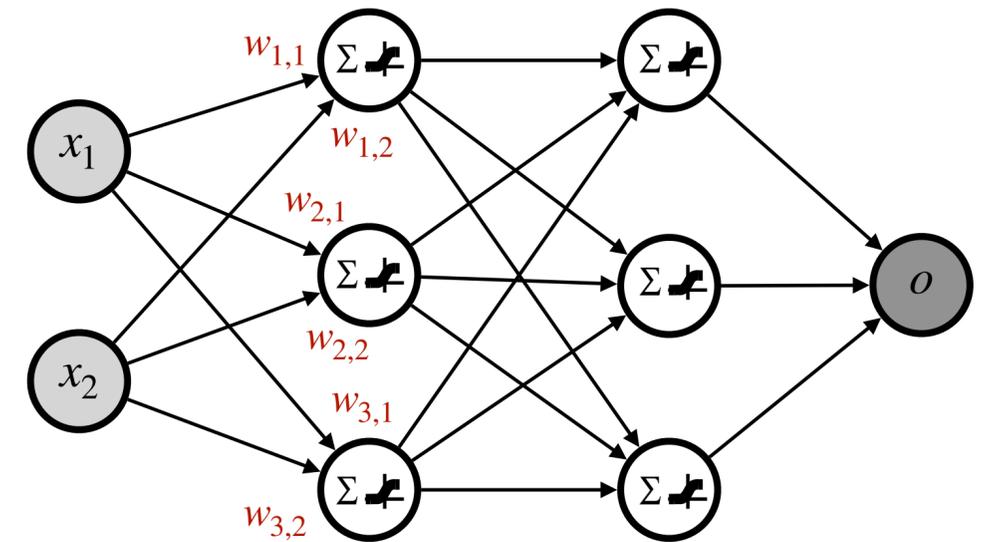
Lernen

- Zuvor festzulegen:
 - Netzwerk-Topologie
 - Anzahl Perzeptrone
 - Anzahl Schichten
 - Verbindungen zwischen den Perzeptronen und Schichten
 - Funktionen der Perzeptrone



Gewichte bestimmen

- Netzwerk festgelegt!
- Gesucht:
 - \vec{w}_j und θ_j für alle Perzeptrone j
- Gegeben:
 - Trainingsdaten
 - Vektoren \vec{x} und zugehörige Ausgabe $o_{\vec{x}}$



Lernen der Gewichte (Idee)

1. Gewichte zufällig initialisieren

2. Optimieren der Gewichte

i. Ergebnis (Ausgabe) für einen Trainingsdatensatz bestimmen

ii. Vergleich mit Zielausgabe

iii. Gewichte gemäß Fehler anpassen

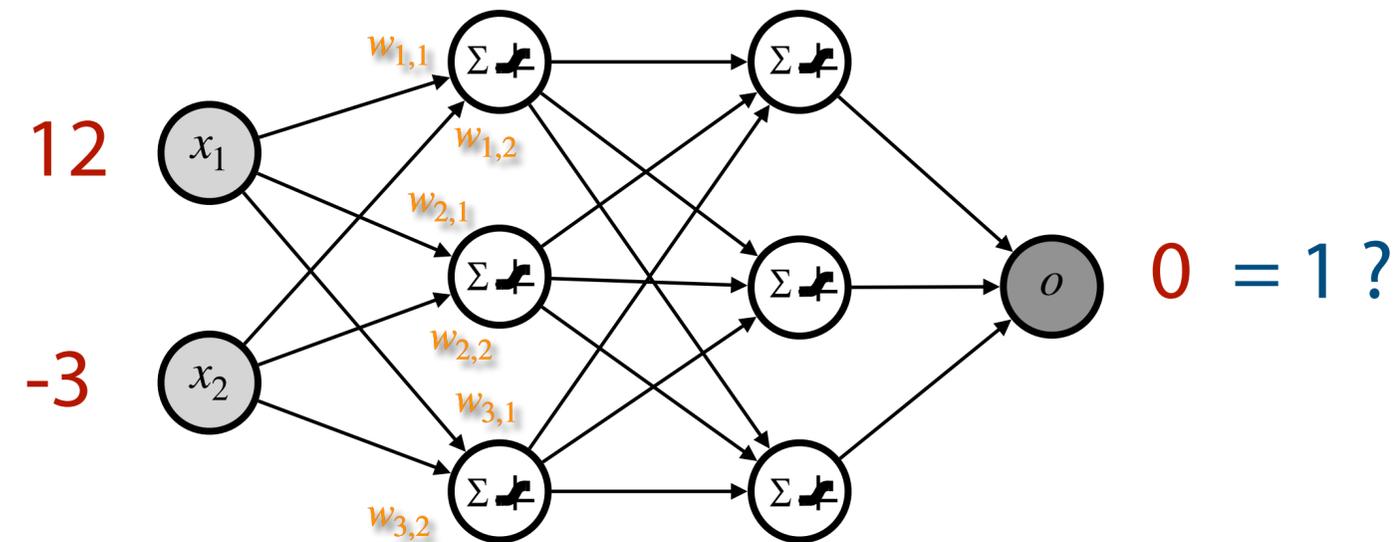
iv. Wiederholen, bis Fehler ausreichend klein

3. Netzwerk mit neuen Daten testen

Batch: Ausgabe bestimmen und Gewichte anpassen für mehrere Datensätze zusammen

Zufällig oder geht das besser?

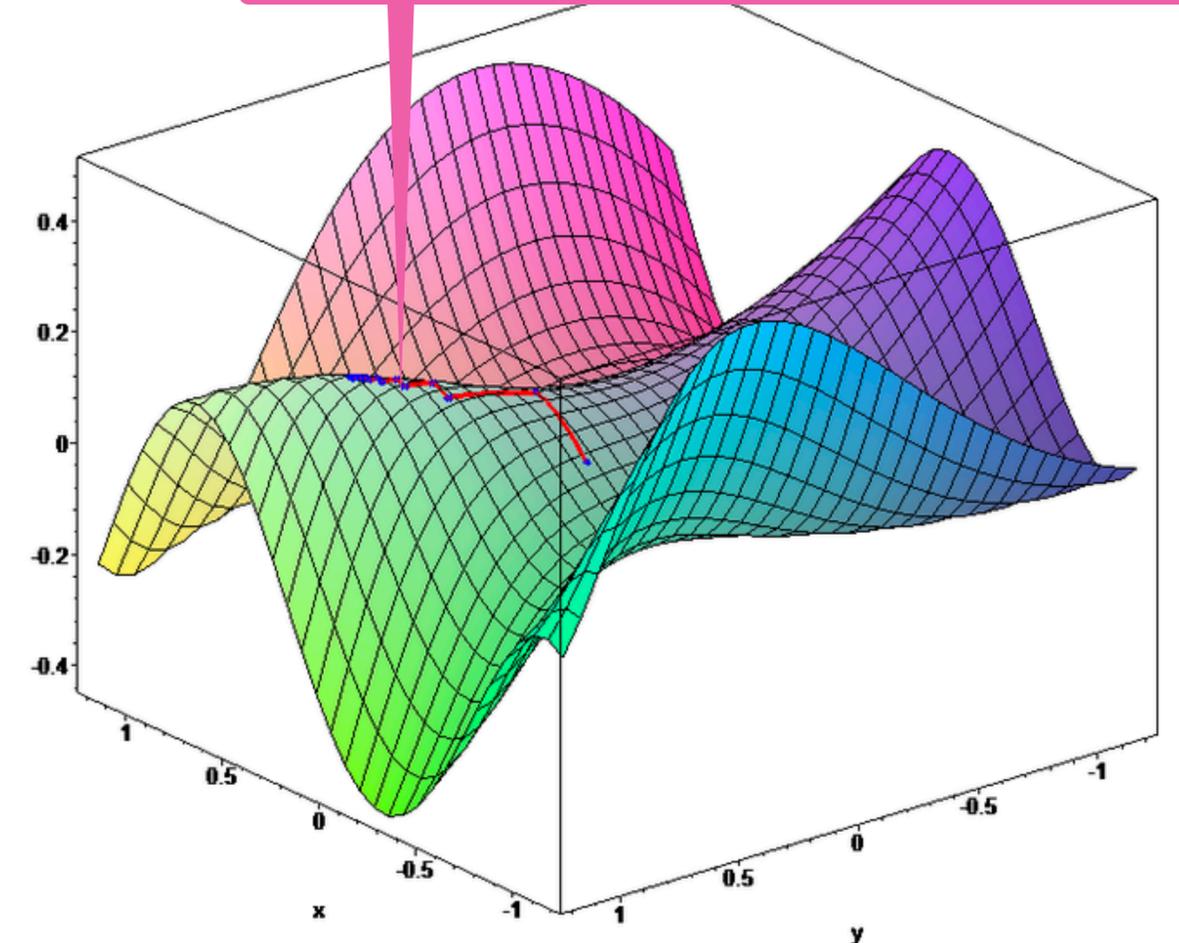
Epoche: Einmal alle Daten in den Trainingsdaten genutzt



Gradientenabstieg (Idee)

- „Wie sollte man die Gewichte anpassen, um den Fehler zu verringern?“
- Bestimmung einer Fehlerfunktion
 - Differenz zwischen Wunschausgabe und Netzwerkausgabe
 - Netzwerk ist eine Hintereinanderschaltung von Funktionen
 - Netzwerk selbst eine Funktion
- Gesucht: Gewichte, die Fehlerfunktion minimieren
 - Gradient bestimmt jeweils Richtung des steilsten Anstieg
 - ✓ Sigmoid-Funktion können wir ableiten
 - Steilster Abstieg gesucht (negativer Gradient)
- ✓ Mithilfe der Kettenregel über mehrere Schichten hinweg möglich (Backpropagation)

„Jeweils in Richtung des Tals bewegen.“



Mittels Python



- Pakete PyTorch (Facebook) und TensorFlow (Google)
- Modellierung von Netzwerken
- Durchführen des Lernens (z.B. Gradientenabstiegs)
- „Ausführen“ der Netzwerke
- Optimiert für Nutzung mit Grafikkarten (GPUs)
- Nutzung auf CPU möglich, aber deutlich langsamer



Treiber meist NVIDIA CUDA

Beispiel: PyTorch



- Python Paket
- <https://pytorch.org/get-started/locally/>
- Installation z.B. mit pip3 `install torch`
- Import

Passenden Befehl auf der Webseite nachschlagen

```
import torch
```

- Test (GPU Treiber)

Intern Nutzung von C++ und CUDA

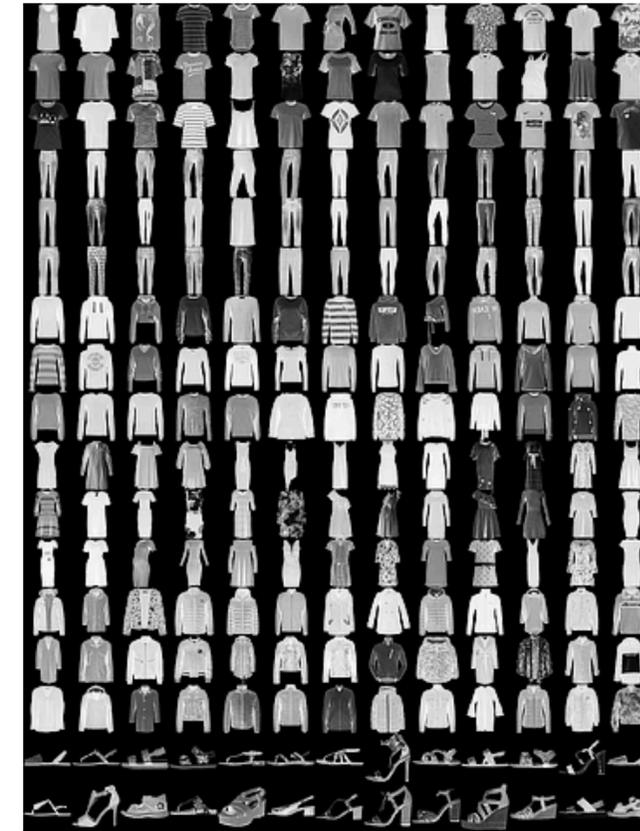
```
torch.cuda.is_available()
```

Beispiel

```
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor
```

```
training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor(),
)
```

```
test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor(),
)
```



Label Description

- | | |
|---|-------------|
| 0 | T-shirt/top |
| 1 | Trouser |
| 2 | Pullover |
| 3 | Dress |
| 4 | Coat |
| 5 | Sandal |
| 6 | Shirt |
| 7 | Sneaker |
| 8 | Bag |
| 9 | Ankle boot |

Daten laden, ähnlich wie schon
bei 20 Newsgroups mittels
SKLearn

Beispiel

Daten in Batches von 64 Bilder gruppieren

```
batch_size = 64
```

```
train_dataloader = DataLoader(training_data, batch_size=batch_size)  
test_dataloader = DataLoader(test_data, batch_size=batch_size)
```

```
for X, y in test_dataloader:  
    print(f"Shape of X [N, C, H, W]: {X.shape}")  
    print(f"Shape of y: {y.shape} {y.dtype}")  
    break
```

Einen Batch ansehen:
64 Bilder mit einer Klasse und
jeweils 28*28 Pixeln

```
Shape of X [N, C, H, W]: torch.Size([64, 1, 28, 28])  
Shape of y: torch.Size([64]) torch.int64
```

```
device = "cuda" if torch.cuda.is_available() else "cpu"  
print(f"Using {device} device")
```

Falls möglich, eine GPU nutzen

```
Using cuda device
```

Beispiel

Oberklasse für „Neuronales Netzwerk“ nutzen

```
class NeuralNetwork(nn.Module):  
    def __init__(self):  
        super().__init__()  
        self.flatten = nn.Flatten()  
        self.linear_relu_stack = nn.Sequential(  
            nn.Linear(28*28, 512),  
            nn.ReLU(),  
            nn.Linear(512, 512),  
            nn.ReLU(),  
            nn.Linear(512, 10)  
        )
```

Pixel nicht in 2 Dimensionen benötigt, also Eingabevektoren der Länge $28*28 = 644$

Netzwerk aus Modulen zusammenbauen (sequentiell hintereinander):
Eingabe 644 Pixel → Ausgabe eine von 10 Klassen

ReLU (rectified linear unit)
 $y = \max(0, x)$

Lineare Transformation der Daten $y = xA^T + b$

```
def forward(self, x):  
    x = self.flatten(x)  
    logits = self.linear_relu_stack(x)  
    return logits
```

Für einen Eingabevektor durchpropagieren.

```
model = NeuralNetwork().to(device)  
print(model(X).shape)  torch.Size([64, 10])
```

Batch X durch das Modell propagieren → Schätzung für jeden Klasse (Arg. Max. ist Vorhersage)

Fehlerfunktion, hier als Kreuzentropie

Beispiel

Fehlerminimierung, hier *Stochastic Gradient Descent*

```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)

def train(dataloader, model, loss_fn, optimizer):
    model.train()

    for X, y in dataloader:
        X, y = X.to(device), y.to(device)

        pred = model(X)
        loss = loss_fn(pred, y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Quasi approximativer Gradientenabstieg

Modell in den Trainingsmodus versetzen.

Vorhersage bestimmen und Fehler bestimmen

Über die Batches iterieren, dabei diese auf die GPU kopieren (Daten und Modell müssen auf gleicher Einheit sein).

- Gradienten der Modellparameter auf 0 setzen
- Fehler durch das Netzwerk „rückwärts“ propagieren (Gradienten neu bestimmt)
- Nutzung der Gradienten um Modellparameter zu verbessern

Beispiel

```
def test(dataloader, model, loss_fn):  
    size = len(dataloader.dataset)  
    num_batches = len(dataloader)  
    model.eval()
```

Modell in den Evaluationsmodus versetzen.

```
    test_loss, correct = 0, 0  
    with torch.no_grad():  
        for X, y in dataloader:  
            X, y = X.to(device), y.to(device)
```

Im folgenden Block keine Gradienten benötigt

```
            pred = model(X)  
            test_loss += loss_fn(pred, y).item()
```

Vorhersage bestimmen, Fehler als Wert speichern

```
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()
```

```
    test_loss /= num_batches  
    correct /= size
```

Normalisieren und
Werte ausgeben

Vorhergesagte Klasse bestimmen (Index mit max.
Wert) und mit Label vergleichen → 1en zählen

```
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")
```

Beispiel

- 5 Epochen:
- Modell mit Trainingsdaten in Batches trainieren
- Modell mit Testdaten prüfen

```
epochs = 5
for t in range(epochs):
    print(f"Epoch {t+1}\n-----")
    train(train_dataloader, model, loss_fn, optimizer)
    test(test_dataloader, model, loss_fn)
```

Epoch 1

Test Error:
Accuracy: 23.8%, Avg loss: 2.185232

Epoch 2

Test Error:
Accuracy: 49.3%, Avg loss: 1.952881

Epoch 3

Test Error:
Accuracy: 61.9%, Avg loss: 1.594316

Epoch 4

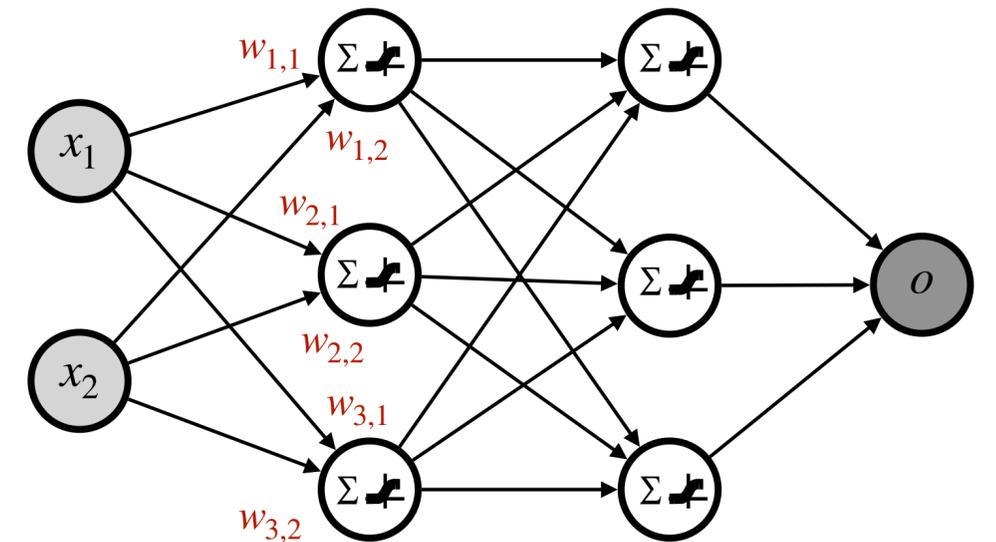
Test Error:
Accuracy: 63.6%, Avg loss: 1.308432

Epoch 5

Test Error:
Accuracy: 64.8%, Avg loss: 1.124939

Zwischenfazit

- Hintereinanderschaltung von (vielen) Perzeptronen
 - Kombination linearer Klassifikatoren
 - Bestimmung von Gewichten nötig → Gradienten nutzen
- PyTorch
 - Bietet benötigte Funktionalitäten
 - Zusammenstellen von Netzwerken
 - Lernen von Gewichten



III.

Lehrevaluation



Evaluation

ZENTRALE LEHREVALUATION

Evaluation im WiSe 2022, Teilnahme ist möglich bis zum: 01. April 2023, 02:00.

Bitte helfen Sie mit, die Qualität der Lehre an unserer Universität zu verbessern: Evaluieren Sie diesen Kurs anonym, am besten jetzt gleich. Danach erhalten Sie hier Zugriff auf die (Zwischen-)Ergebnisse.

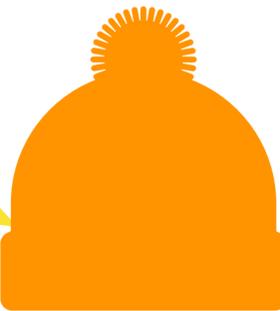
Diesen Kurs jetzt evaluieren

Je mehr mitmachen, desto besser!

Fragen zur Evaluation? [Alles über die zentrale Lehrevaluation](#)

Besprechung der Ergebnisse

Gerne später noch
teilnehmen!



IV.

Transformer Sprachmodelle

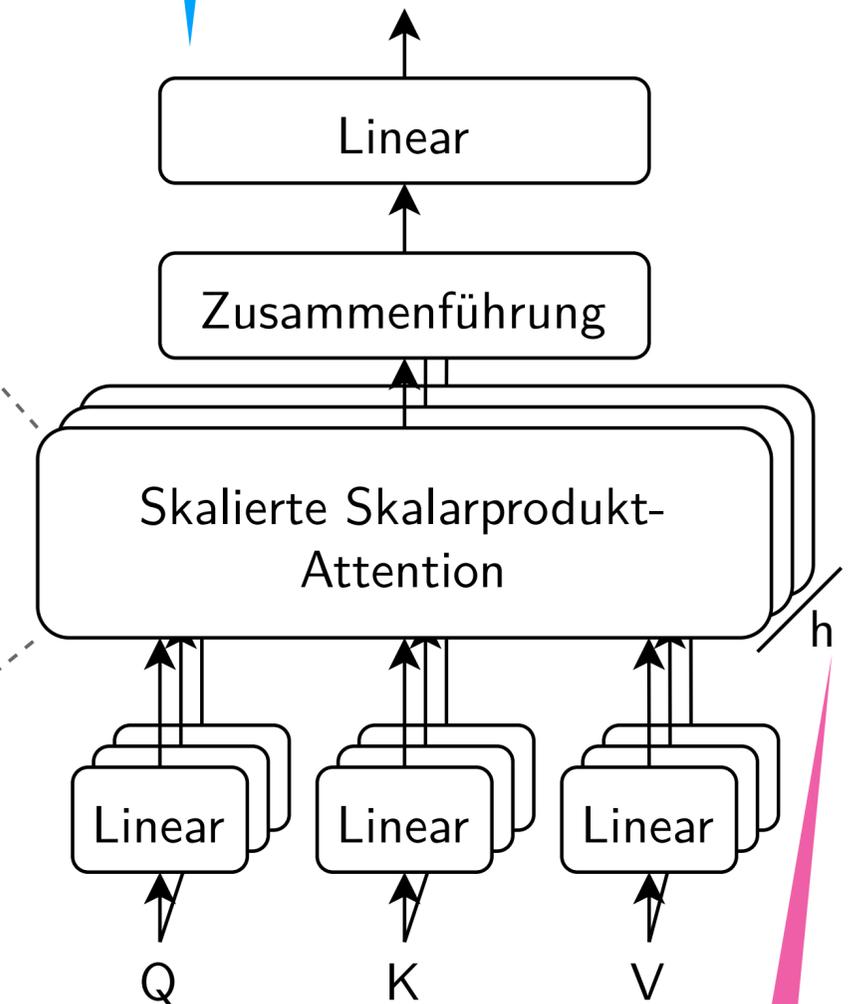
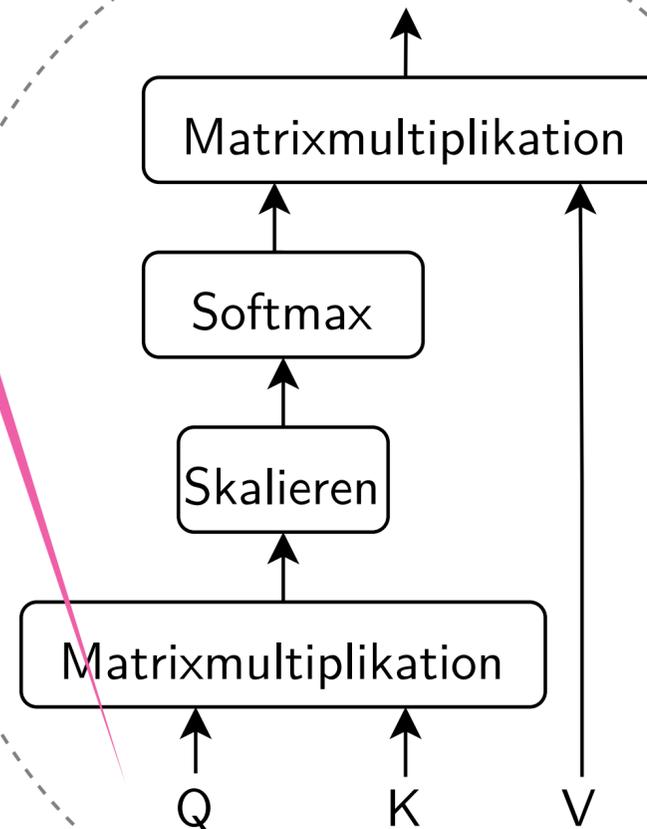
1. Idee

Drei Matrizen Q, K, V erzeugen drei Versionen des Eingabevektors

Attention

Kann für viele Eingabevektoren parallel berechnet werden (viele Wörter)

- Eingabevektor
- Z.B. für ein Wort
- Relevante Teile („Features“) erkennen
- Irrelevante Teile ausblenden
- Skalarprodukt zwischen zwei Ausgaben ergibt einen Wert der Übereinstimmung



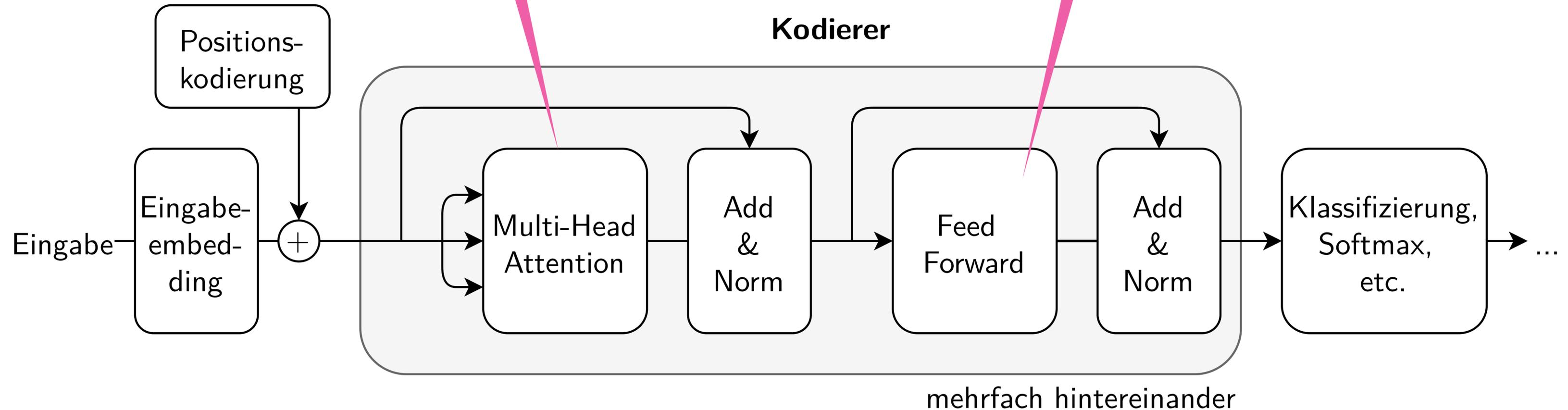
Aufteilen in h Bereiche $\rightarrow h$ (relevante) Relationen sollen gefunden werden können

Attention von der Folie zuvor

Transformer

Klassisches Perzeptron-Netzwerk

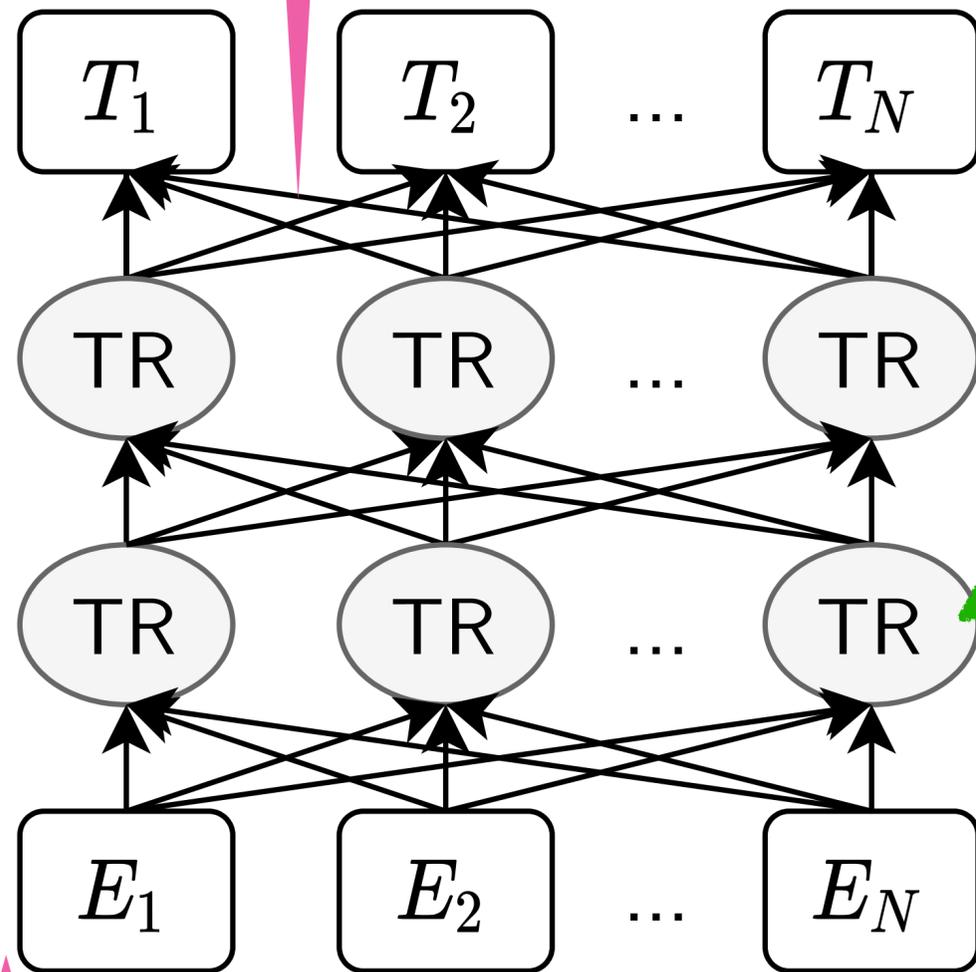
Kodierer



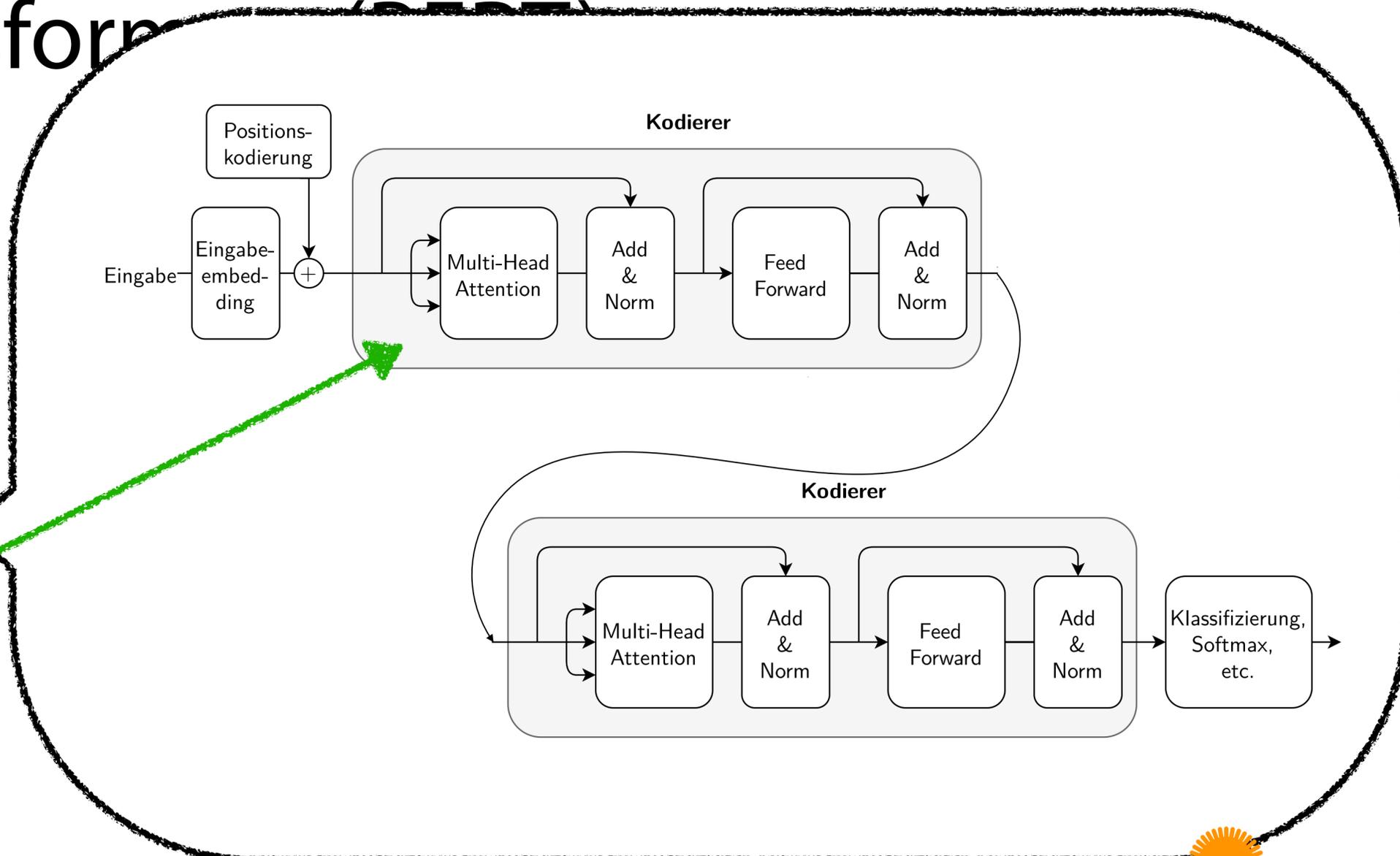
- Kernbaustein für Verarbeitung jedes Wortes (Token)
- Grauer Teil mehrfach hintereinander geschaltet

Bidirectional Encoder Representations from

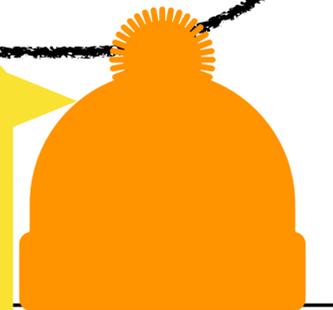
Schichten zwischen den verschiedenen Eingaben verbunden



Für jede Eingabe eine Folge von Transformerblöcken



Weiterhin Grundidee des Netzwerks, nur viel aufwendigere Module als Perzeptrone → Module in Modulen ...



BERT: Eingabe (Tokenize)

Text	Bisons are large animals.							They run fast.							
Tokenisierter Text	[CLS]	bison	##s	are	large	animals	.	[SEP]	they	run	fast	.	[SEP]		
Token IDs	101	22285	2015	2024	2312	4176	1012	102	2027	2448	3435	1012	102	0	...
Segment IDs	0	0	0	0	0	0	0	0	1	1	1	1	1	0	...
Position	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	...
Sequenzmaske	1	1	1	1	1	1	1	1	1	1	1	1	1	0	...

- Wörter werden mittels WordPiece in Vektoren übersetzt
- Position und „Segment“ werden hinzugefügt (Teil 1 oder 2)

Wörter werden gleichzeitig verarbeitet, Modell kann Position somit nur durch hinzuaddierten Vektor erkennen.

Großer Aufwand, fertige Modelle können heruntergeladen werden

ERT: Training

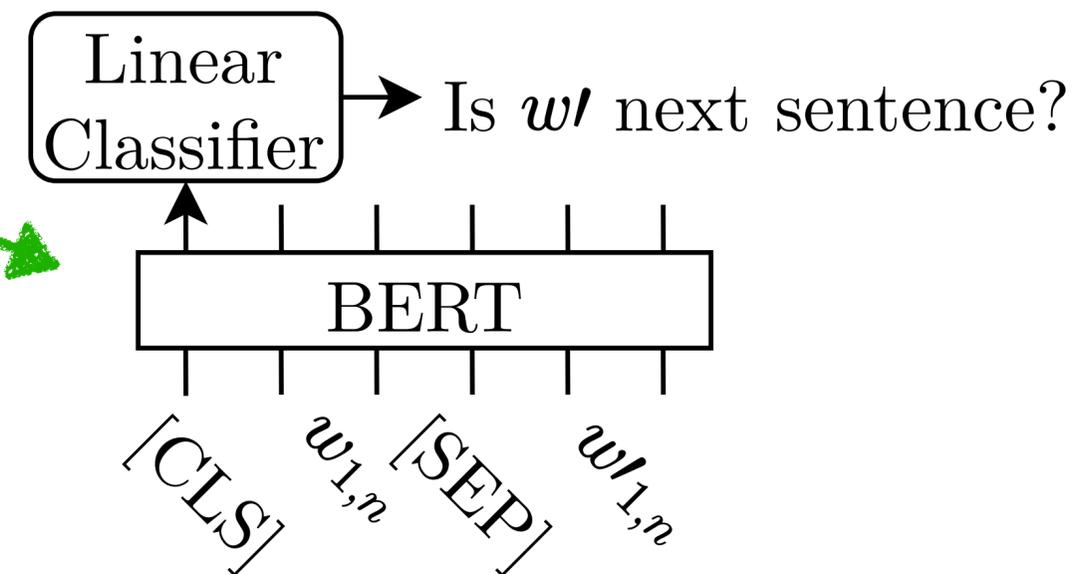
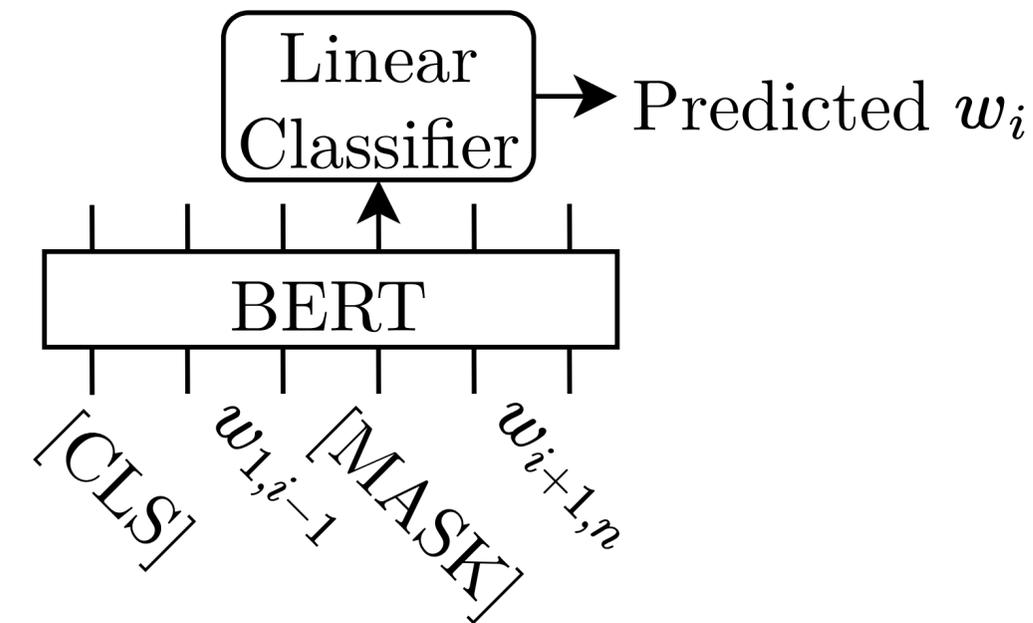
- Pre-Training

- Modell soll Sprache „lernen“
- Vorhersage eines maskierten Wortes
- Bestimmung, ob zwei Sätze nacheinander stehen

- Fine-Tuning

- Modell wird für bestimmten Anwendungsfall optimiert

Klein(er)er Aufwand, lernt man selbst auf eigenen Daten



BERT: Anwendung

Ein/ mehrere zusammenhängende Sätze

- Erste Vektorausgabe
- Repräsentation des ganzen Texts

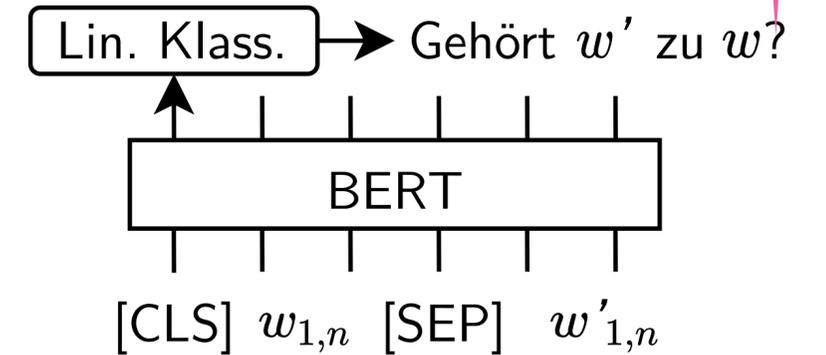
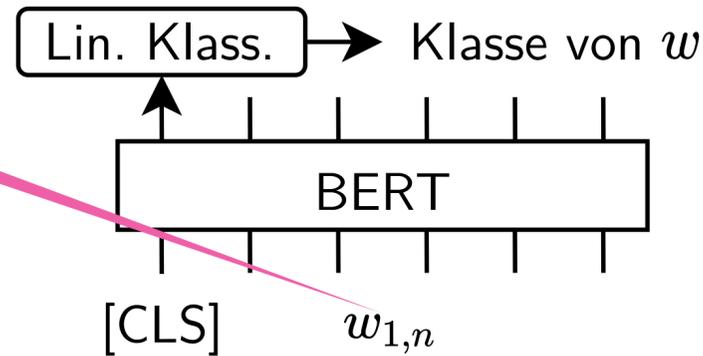
- Weitere Vektorausgaben
- Repräsentation des jeweiligen Wortes

Vorne Frage, hinten Text mit u.a. der Antwort (Ziel: Intervall der Worte mit Antwort finden).

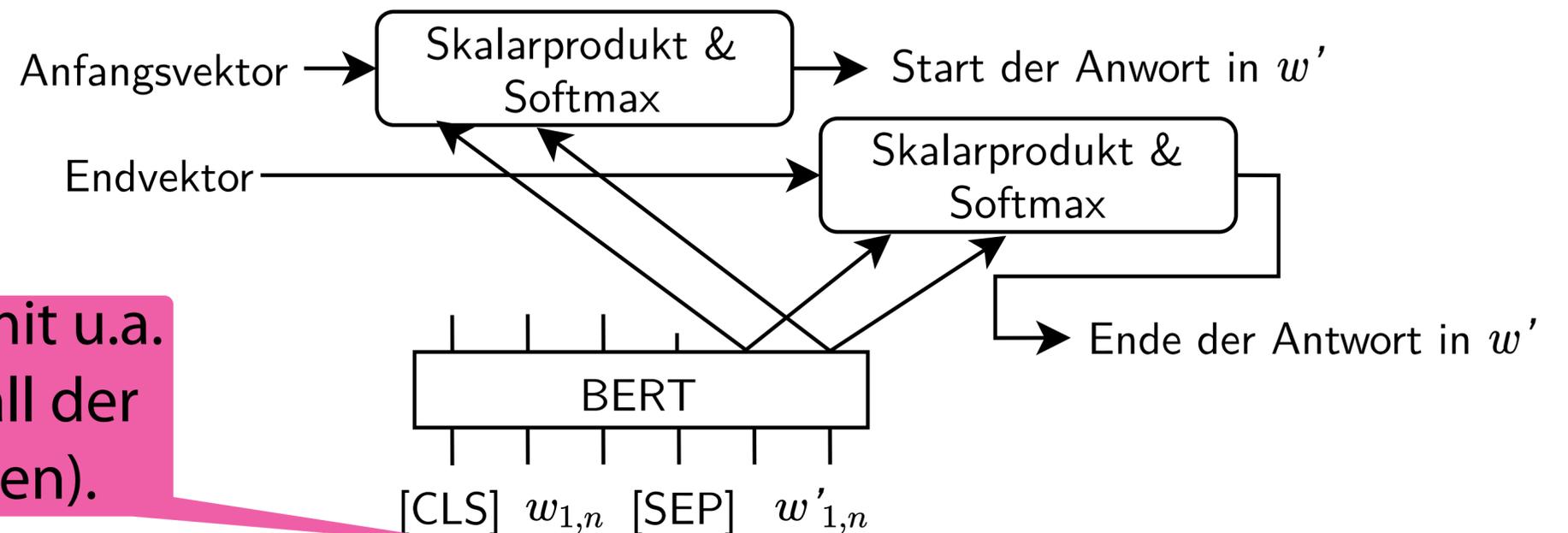
Relation zwischen zwei Sätzen gesucht

Nächster-Satz Vorhersage

Sequenzklassifizierung



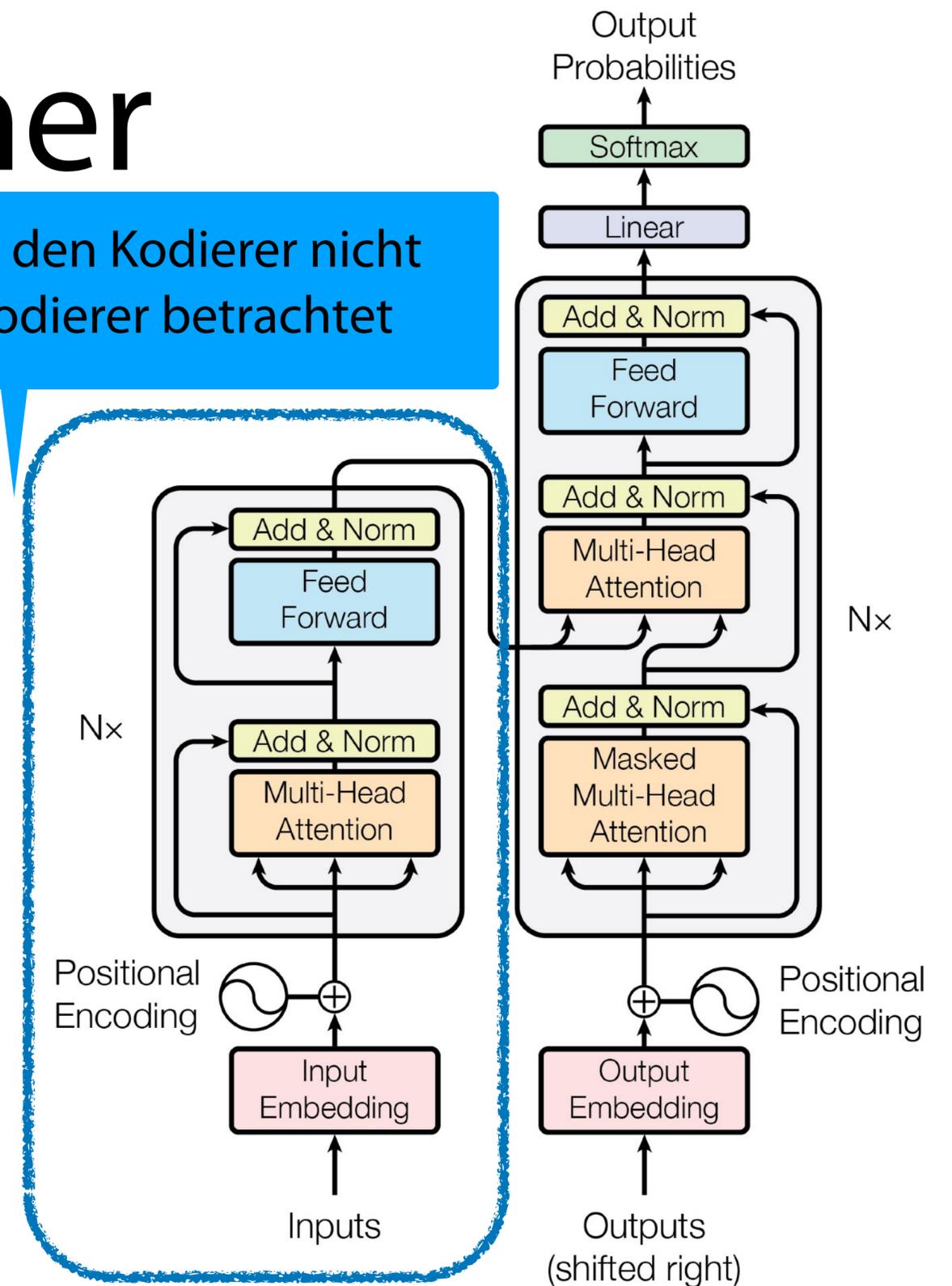
Fragebeantwortung



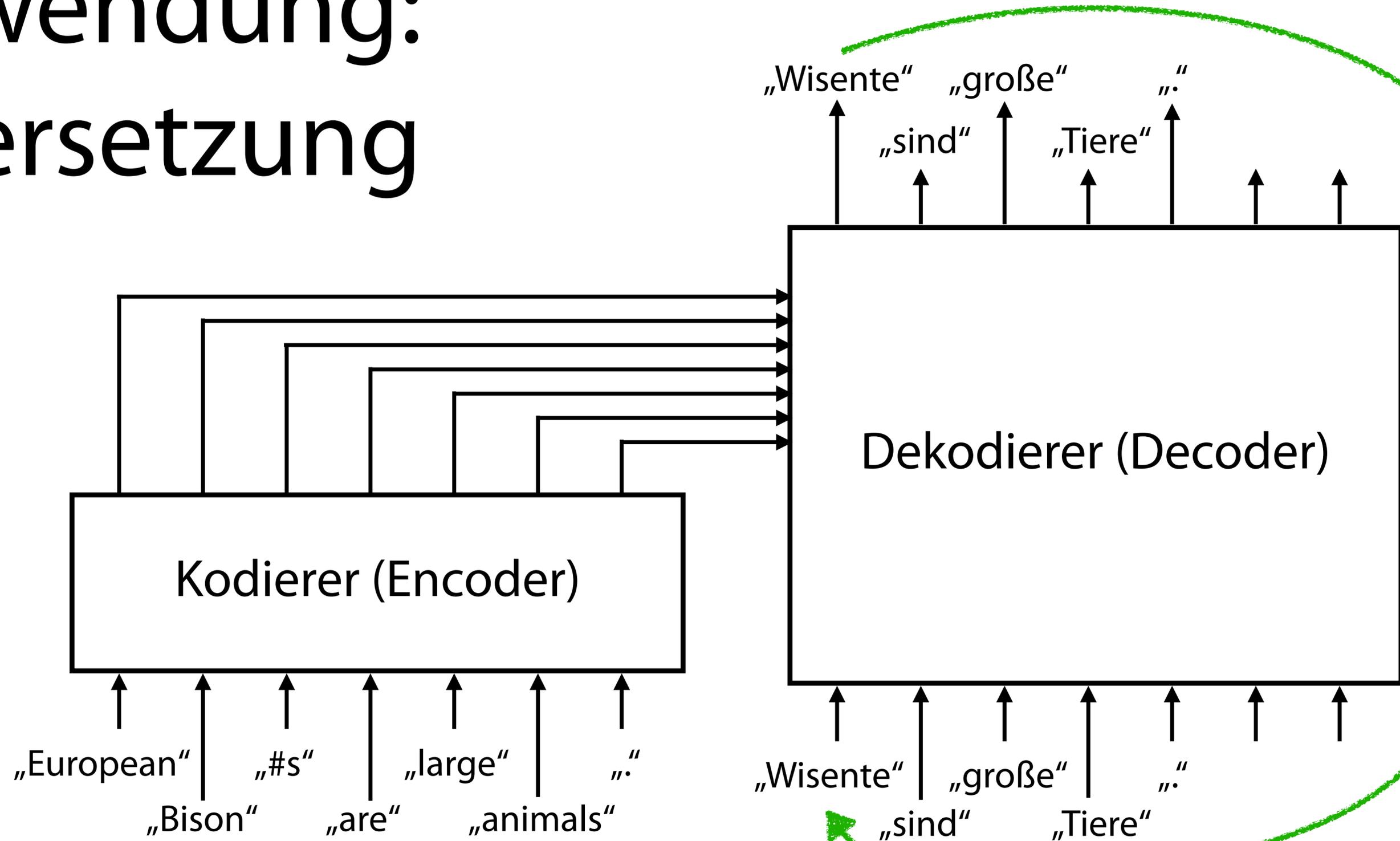
Transformer

Bisher nur den Kodierer nicht den Dekodierer betrachtet

- Kodierer erzeugt Vektorrepräsentation gegeben Text
- Dekodierer erzeugt Text gegeben Vektorrepräsentation
 - Übersetzung zwischen 2 Sprachen
 - Erzeugen „neuen“ Texts

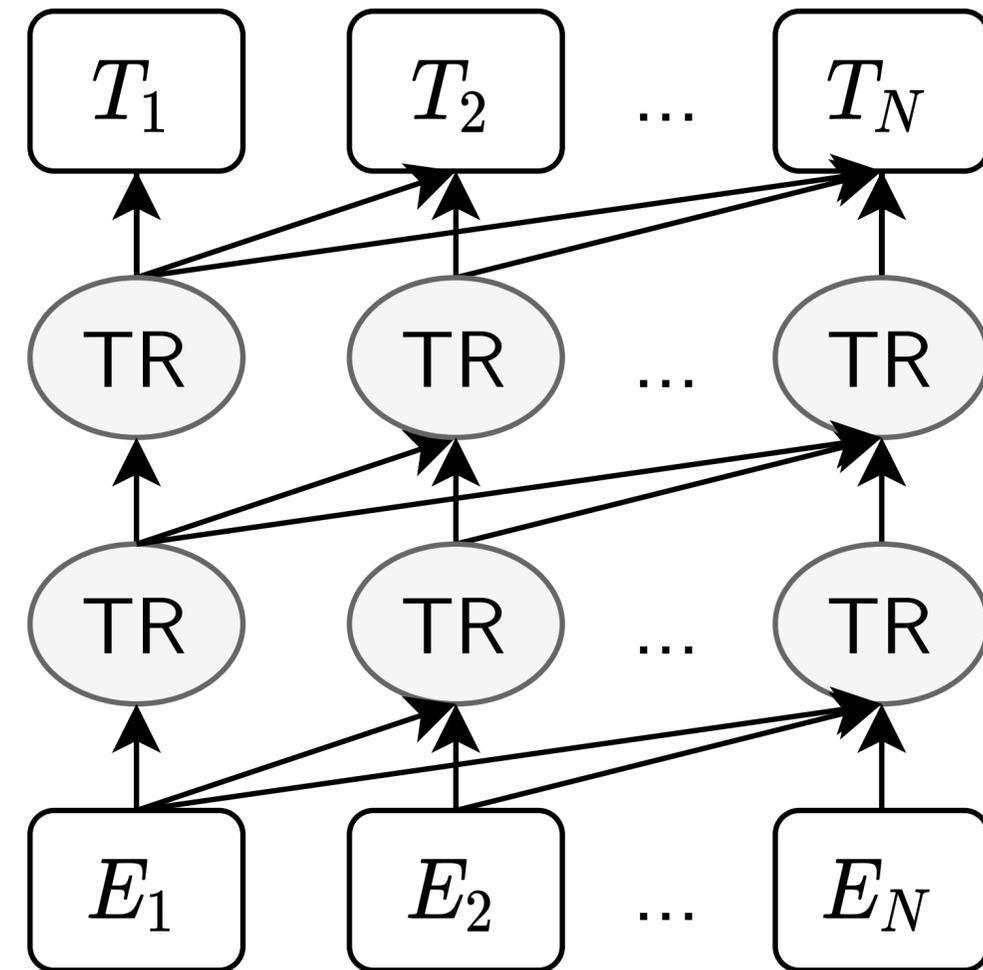


Anwendung: Übersetzung



Generative Pre-trained Transformer (GPT)

- Fokus auf Generation von Text
- Verbindungen nicht bidirektional, sondern nur von links nach rechts
- Bestimmung eines nächsten Wortes gegeben einen Satzanfang



IV.

Transformer Sprachmodelle

2. GPT & BERT mit Python

Installation



- Python Paket
- <https://huggingface.co/docs/transformers>
- Installation z.B. mit pip3 `install transformers`
- Vorher GPU-Backend `torch` oder `tensorflow` installieren
- CPU-Version mittels `pip install transformers[torch]` bzw. `transformers[tf-cpu]`
- Import z.B.

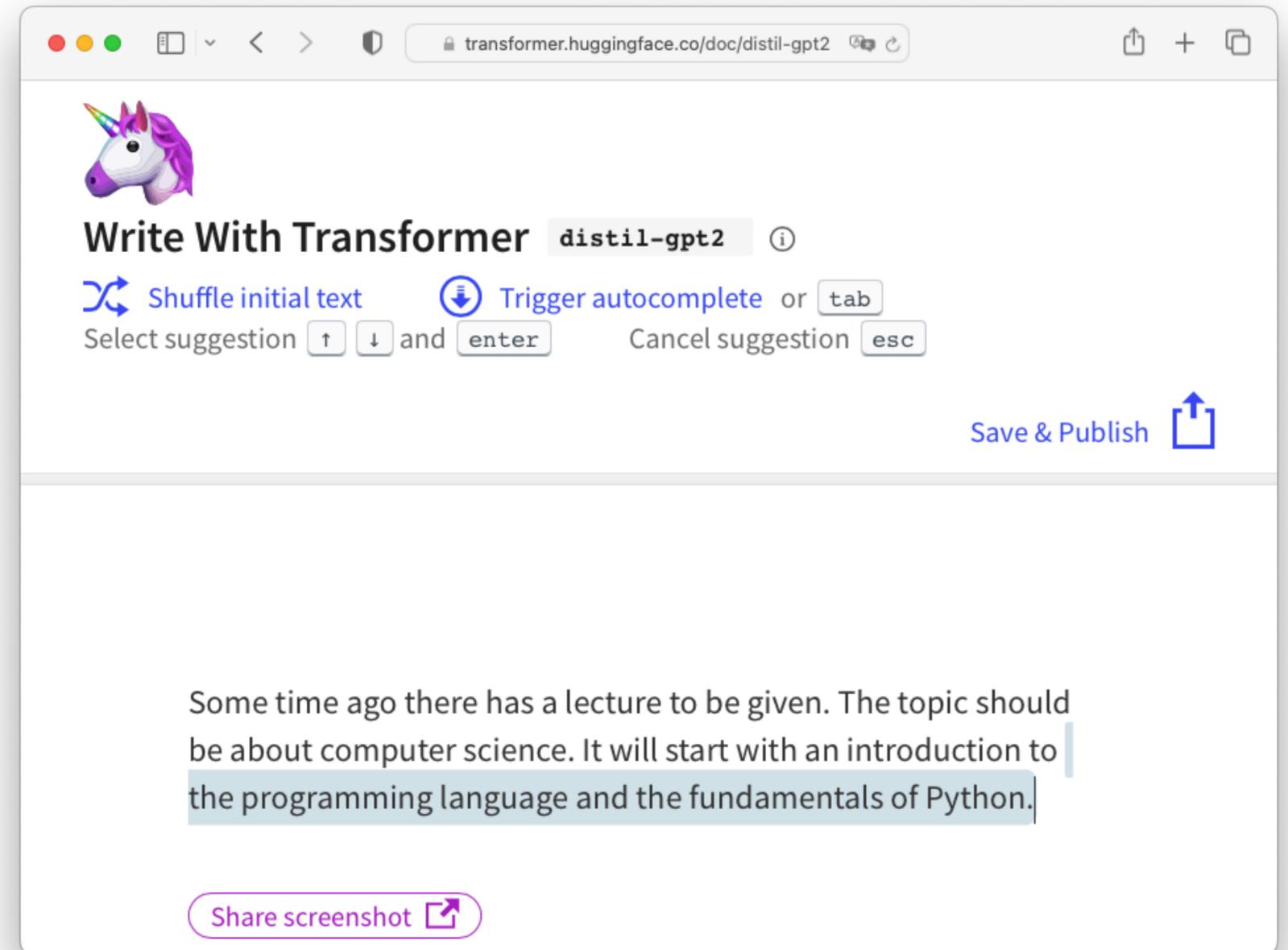
CUDA-Treiber und GPU benötigt,
geht auch in Docker

```
from transformers import pipeline  
from transformers import BertModel
```

Üblich: Benötigte Klassen oder Funktionen

Transformers

- Repository von Transformer Sprachmodellen und Datensätzen
 - <https://huggingface.co/models>
 - <https://huggingface.co/datasets>
- „Online-Demo“
 - <https://transformer.huggingface.co/doc/distil-gpt2>
- Python Paket mit Implementierungen verschiedener Modelle
 - <https://huggingface.co/docs/transformers/index>
 - Sowohl für PyTorch als auch Tensorflow



Beispiel I: BERT

```
from transformers import BertTokenizer, BertForNextSentencePrediction
import torch
```

```
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = BertForNextSentencePrediction.from_pretrained("bert-base-uncased")
```

```
prompt = "In Italy, pizza served in formal settings, such as at a restaurant, is presented unsliced."
next_sentence = "The sky is blue due to the shorter wavelength of blue light."
```

```
encoding = tokenizer(prompt, next_sentence, return_tensors="pt")
print(list(encoding.keys()))
```

```
['input_ids', 'token_type_ids', 'attention_mask']
```

```
outputs = model(**encoding, labels=torch.LongTensor([1]))
print(outputs.logits)
```

```
tensor([[ -3.0729,  5.9056]], grad_fn=<AddmmBackward0>)
```

```
print("random sentence" if outputs.logits[0, 0] < outputs.logits[0, 1] else "next sentence")
random sentence
```

Beispiel II: BERT

Pre-Trained Modell und Tokenizer
(herunter-)laden

```
from transformers import BertTokenizer, BertForNextSentencePrediction
import torch
```

```
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = BertForNextSentencePrediction.from_pretrained("bert-base-uncased")
```

Vorhersage ob zwei Sätze
zusammen passen.

```
prompt = "In Italy, pizza served in formal settings, such as at a restaurant, is presented unsliced."
next_sentence = "Thus, one might need a knife to cut the food."
```

Eingaben mittels Tokenizer für
BERT vorbereiten, siehe auch
[Folie 39](#).

```
encoding = tokenizer(prompt, next_sentence, return_tensors="pt")
print(list(encoding.keys()))
```

```
['input_ids', 'token_type_ids', 'attention_mask']
```

Eingabe an Modell übergeben, besteht aus
einem Batch mit einem Ergebnis.

```
outputs = model(**encoding, labels=torch.LongTensor([1]))
print(outputs.logits)
```

```
tensor([[ 5.7260, -5.1682]], grad_fn=<AddmmBackward0>)
```

Ausgabe auswerten

```
print("random sentence" if outputs.logits[0, 0] < outputs.logits[0, 1] else "next sentence")
next_sentence
```

BERT Fine-Tuning

20 Newsgroups Korpus → BERT soll Dokumente der 20 Kategorien erkennen

Erstellen einer Dataset-Klasse, die den Korpus passend ausgibt → Projektaufgabe 2 ;-)

Tokenizer für BERT

Auswahl einer sehr kleinen Teilmenge möglich → schneller, aber schlechtes Modell.

BERT für „unseren“ Anwendungsfall nutzen.

Ein Dokument (eine Eingabe) per Index abrufen.

Eingabe für BERT passend erstellen und auch das korrekte Label anfügen! (Wichtig: Länge vereinheitlichen, damit Batches gebildet werden können.)

Der Code ist nicht effizient, auch Tokenize sollte man z.B. in Batches durchführen!

Verlangt von der Schnittstelle

Magnus Bender | WiSe 2022/23

Werkzeuge für das wissenschaftliche Arbeiten

```
import torch
from sklearn.datasets import fetch_20newsgroups

class NewsgroupsData(torch.utils.data.Dataset):
    def __init__(self, tokenizer, train=True, reduce_size=True):
        self.tokenizer = tokenizer
        self.X, self.y = fetch_20newsgroups(
            remove=["headers", "footers", "quotes"],
            subset="train" if train else "test", return_X_y=True
        )
        if reduce_size:
            self.X, self.y = self.X[:50], self.y[:50]

    def __getitem__(self, index):
        item = self.tokenizer(self.X[index], truncation=True, padding="max_length")
        item['labels'] = self.y[index]
        return item

    def __len__(self):
        return len(self.X)
```

BERT Fine-Tuning

```
import numpy as np
from sklearn.metrics import accuracy_score
```

Evaluationsfunktion erstellen

```
def get_accuracy(eval_result):
    logits, labels = eval_result
    predicts = np.argmax(logits, axis=-1)
    return { "accuracy" : accuracy_score(labels, predicts) }
```

Vorhersage „ausrechnen“

Gewünschte Metriken berechnen und als Wörterbuch zurückgeben.

```
from transformers import (
    Trainer, TrainingArguments,
    BertTokenizer, BertForSequenceClassification
)
```

Passendes Modell laden und den Trainer

```
model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=20)
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
```

Pre-Trained Modell (herunter-)laden und Anzahl der Label festlegen!

BERT Fine-Tuning

```
trainer = Trainer(  
    model=model,  
    args=TrainingArguments(  
        output_dir="test_trainer",  
        overwrite_output_dir=True  
    ),  
    train_dataset=NewsgroupsData(tokenizer, train=True),  
    eval_dataset=NewsgroupsData(tokenizer, train=False),  
    compute_metrics=get_accuracy  
)
```

Trainer erstellen

- Pre-Trained Modell angeben
- Argumente angeben (z.B. Anzahl Epochen, Optimierer, Größe der Batches, ...)
- Datensätze für Training und Evaluation des Modelle (unsere Klasse von oben nutzen!)
- Funktion für die Metriken angeben

```
trainer.train()  
trainer.evaluate()
```

Training starten (zeigt u.A. einen Fortschrittsbalken an)

Dauert ohne eine GPU sehr lange ...

Evaluation durchführen, zeigt Ergebnisse am Ende an.

```
trainer.save_model("./my-model")  
BertForSequenceClassification.from_pretrained("./my-model")
```

Speichern des trainierten Modells (in einen Ordner).

Laden des selbst erstellten Modells.

Chat G

- Fine-Tuning von GPT 3.5
- Zusätzlich
 - Überwachtes Lernen
 - Reinforcement Learning (Feedback)

OP

What are the differences to GPT?

OP

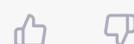


ChatGPT is a variant of GPT, but there are a few key differences between the two models.



1. Training data: ChatGPT is specifically trained on conversational data, such as dialogues and chat logs, whereas GPT is trained on a broader range of internet text. This makes ChatGPT better suited for generating responses in a conversational context.
2. Fine-tuning: ChatGPT is fine-tuned on conversational tasks, such as question answering and dialogue generation, whereas GPT is fine-tuned on a wide range of language tasks such as language translation, summarization and text generation.
3. Model size: ChatGPT is a smaller model than GPT, it has fewer parameters and therefore requires less computational power to run.
4. Output: ChatGPT is trained to generate human-like text specifically for chatbot applications, it can be fine-tuned to understand and generate text that is specific to a certain conversational topic (e.g. customer service, technical support, e-commerce) while GPT is more general and can be fine-tuned to generate text on a wide range of topics.

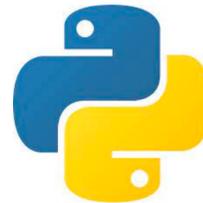
In summary, ChatGPT is a smaller, more specialized version of GPT, which is specifically designed for conversational tasks and fine-tuned on conversational data.



V. Zusammenfassung

Inhaltsübersicht

1. Programmiersprache Python
 - a) Einführung, Erste Schritte
 - b) Grundlagen
 - c) Fortgeschritten



2. Auszeichnungssprachen
 - a) LaTeX, Markdown

L^AT_EX



3. Benutzeroberflächen und Entwicklungsumgebungen
 - a) Jupyter Notebooks lokal und in der Cloud (Google Colab)

4. Versionsverwaltung
 - a) Git, GitHub



5. Wissenschaftliches Rechnen
 - a) NumPy, SciPy



6. Datenverarbeitung und -visualisierung
 - a) Pandas, matplotlib, NLTK

7. Machine Learning (scikit-learn)

- a) Grundlegende Ansätze (Datensätze, Auswertung)



- b) Einfache Verfahren (Clustering, ...)

8. DeepLearning

- a) TensorFlow, PyTorch, HuggingFace Transformers



Zusammenfassung

II. Deep Learning

1. Perzeptron

2. Mehrschichtige Netzwerke

IV. Transformer Sprachmodelle

1. Idee

2. BERT & GPT mit Python

- Abgabe der Aufgabe 5 am Freitag
- (Anmeldung im QiS)
- Teilnahme an der Evaluation im Moodle

~~Werkzeuge~~

Word2Vec (7b) und
RegEx (3a) wären noch
verfügbar.