Werkzeuge für das wissenschaftliche Arbeiten

Python for Machine Learning and Data Science

Magnus Bender
bender@ifis.uni-luebeck.de
Wintersemester 2023/24

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. Herangehensweise & Tipps Aufgabe 5
- II. Deep Learning
  1. Perzeptron
  2. Mehrschichtige Netzwerke
- III. Lehrevaluation
- IV. Lösungsvorschlag Projektaufgabe 4
- V. Transformer Sprachmodelle
  1. Idee
  2. BERT & GPT mit Python

Heute & nächste Woche

I.
Projektaufgaben

1. Lösungsvorschlag Aufgabe 3

Projektaufgabe 3

„Git, Markdown und LaTeX“

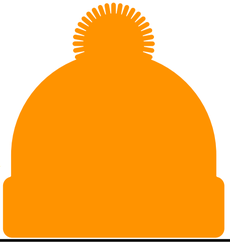
- Vorstellung möglicher Lösungen

  (werden nicht in Moodle hochgeladen!)

I.
Projektaufgaben

2. Herangehensweise & Tipps Aufgabe 5

„Bonusaufgabe“



Projektaufgabe 5

„Maschinelles Lernen und Data Science“

- Juypter-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

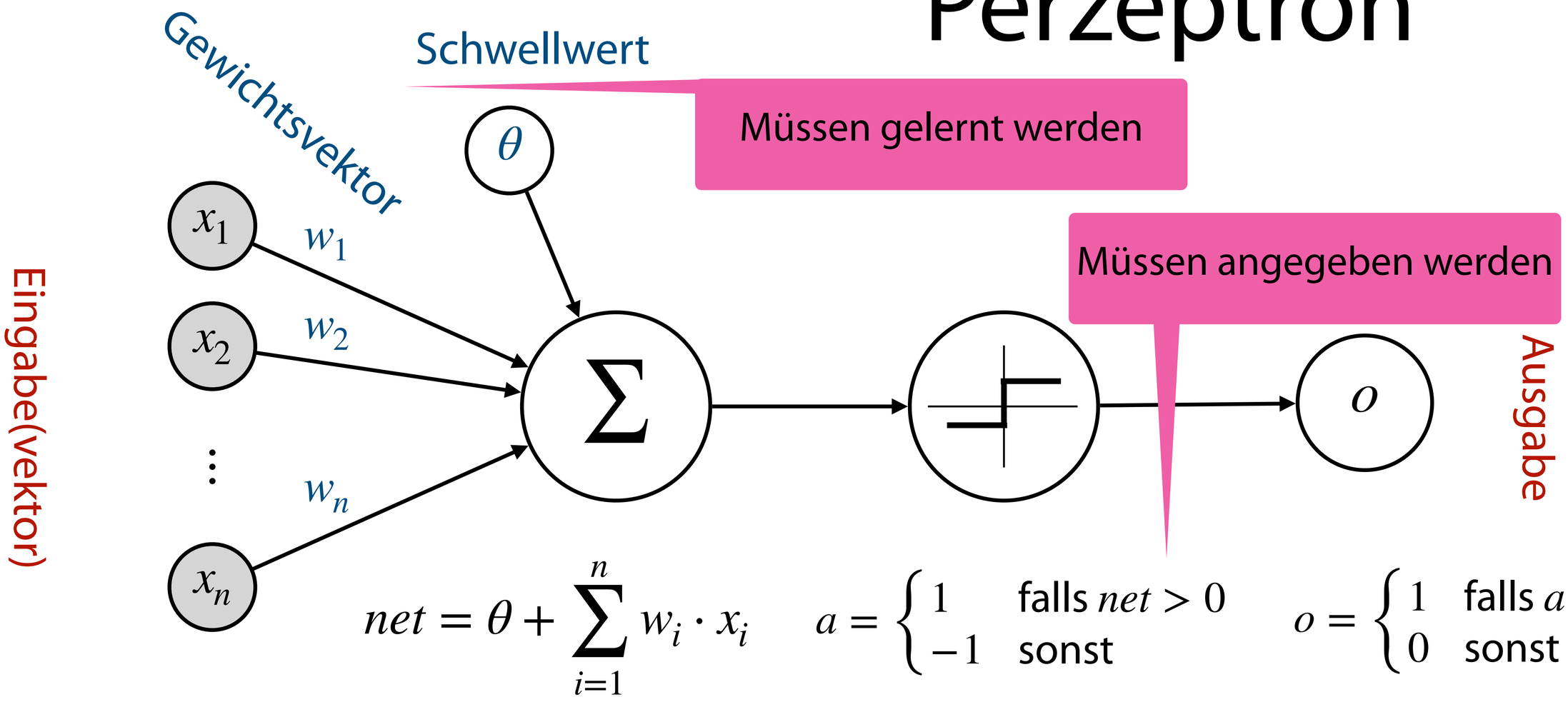
II.
Deep Learning

1. Perzeptron

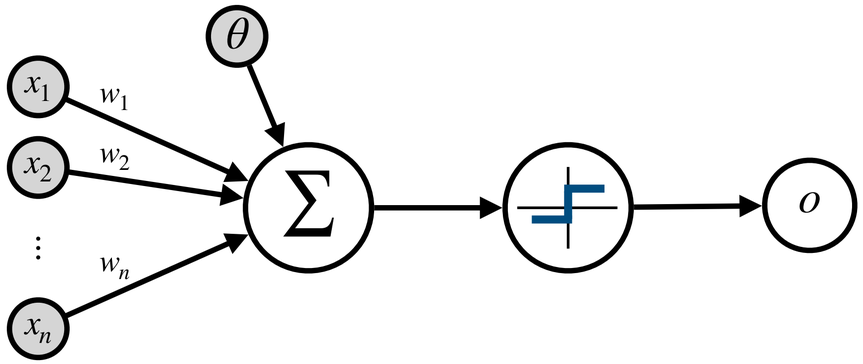
Danksagung

- 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


Gewichtsvektor
Schwellwert
Müssen gelernt werden
Müssen angegeben werden
Eingabe(vektor)
Ausgabe
θ
x_1
x_2
x_n
w_1
w_2
w_n
Σ
o
$net = \theta + \sum_{i=1}^{n} w_i \cdot x_i$
$a = \begin{cases} 1 & \text{falls } net > 0 \\ -1 & \text{sonst} \end{cases}$
$o = \begin{cases} 1 & \text{falls } a = 1 \\ 0 & \text{sonst} \end{cases}$
Eingabefunktion
Aktivierungsfunktion
Ausgabefunktion

Perzeptron



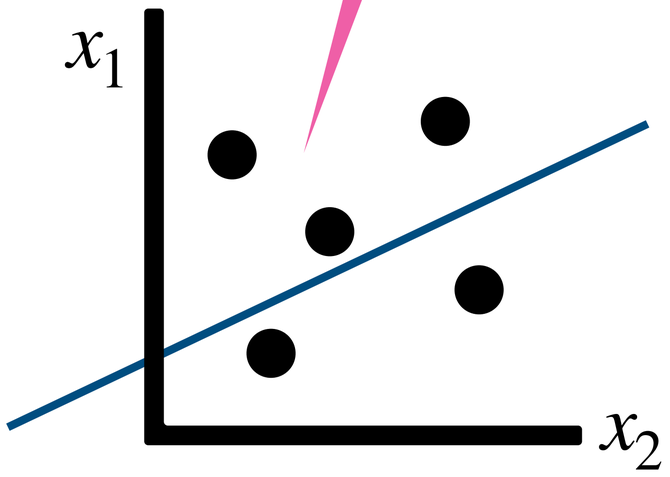
Zusammengefasst

Skalarprodukt

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

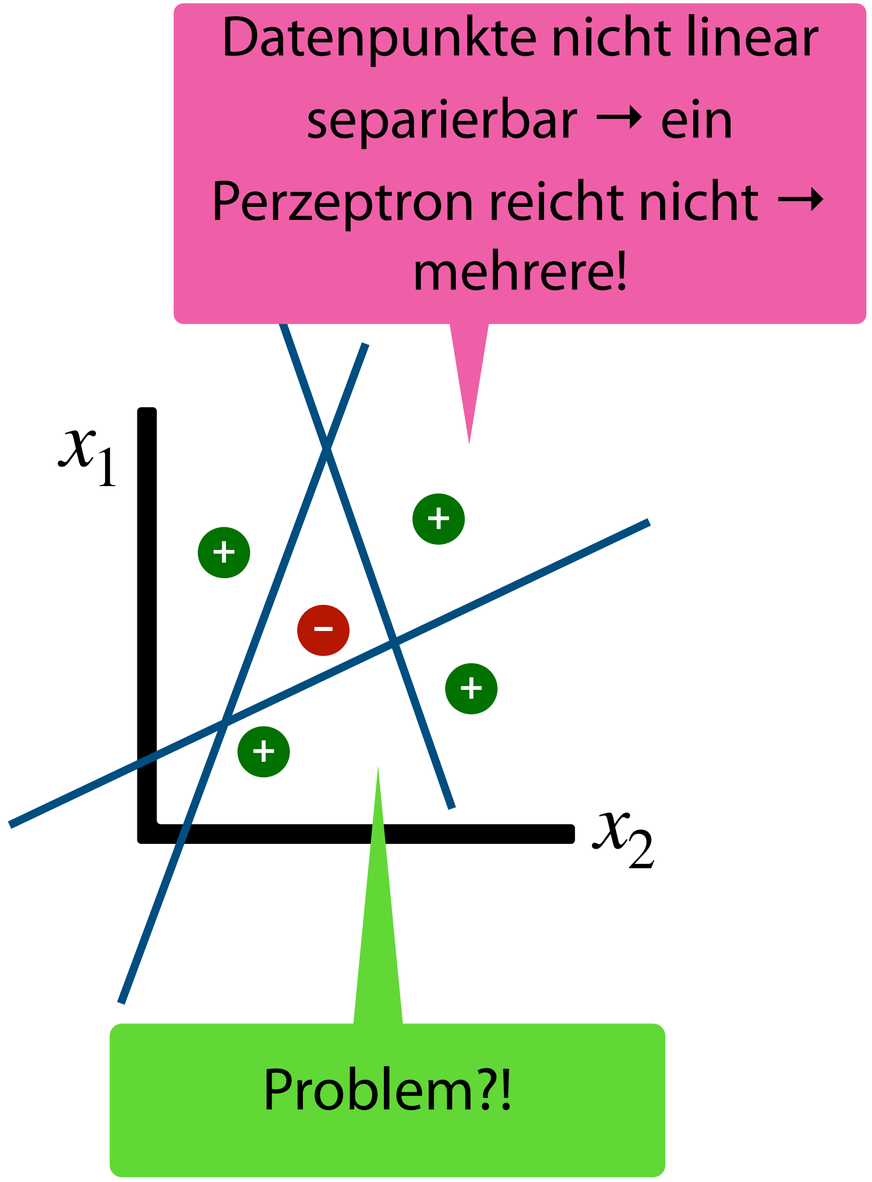
Lineare Funktion

Hier z.B. 2 Dimensionen

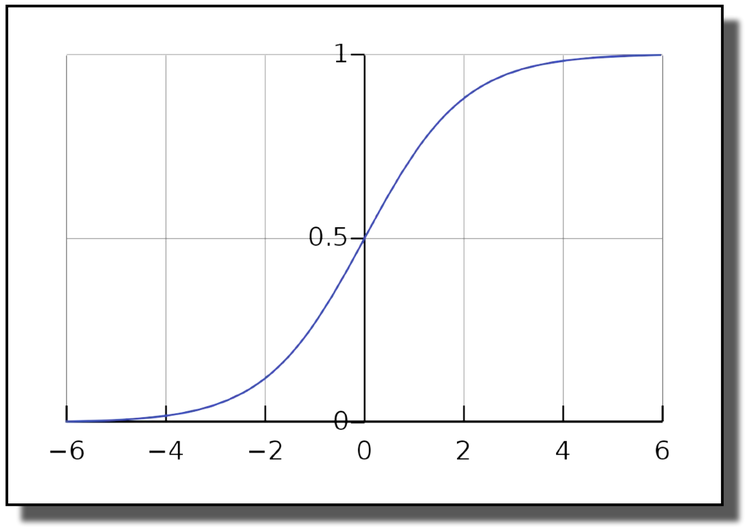


Perzeptron

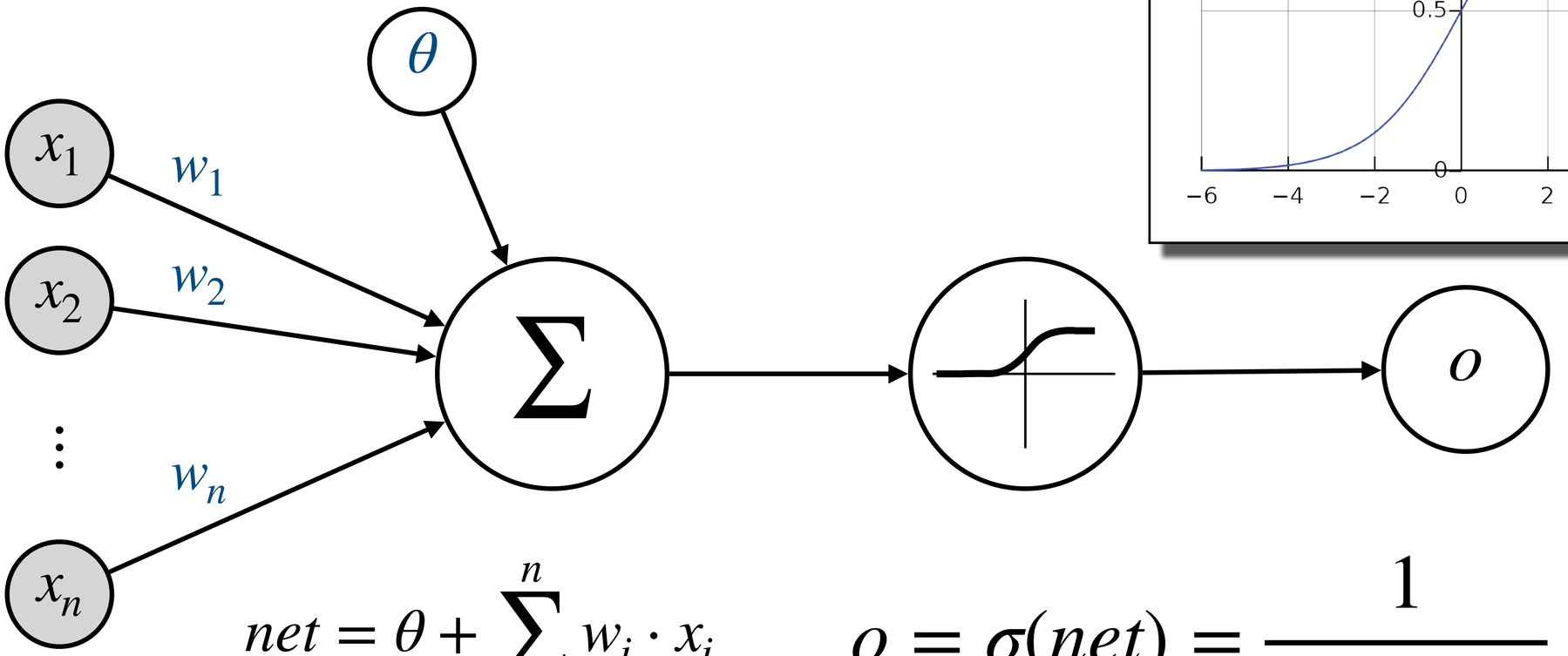
- Lineare Funktion zur Klassifikation
  - Eingabe eines mehrdimensionalen Vektors $\vec{x}$
  - Ausgabe $o \in \{0,1\}$ (häufig auch $o \in \{-1,1\}$)
- Parameter
  - Gewichte $\overrightarrow{w}$ und Schwellwert θ
  - *Müssen bestimmt/ gelernt werden*



Perzeptron



Eingabe(vektor)



Ausgabe

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

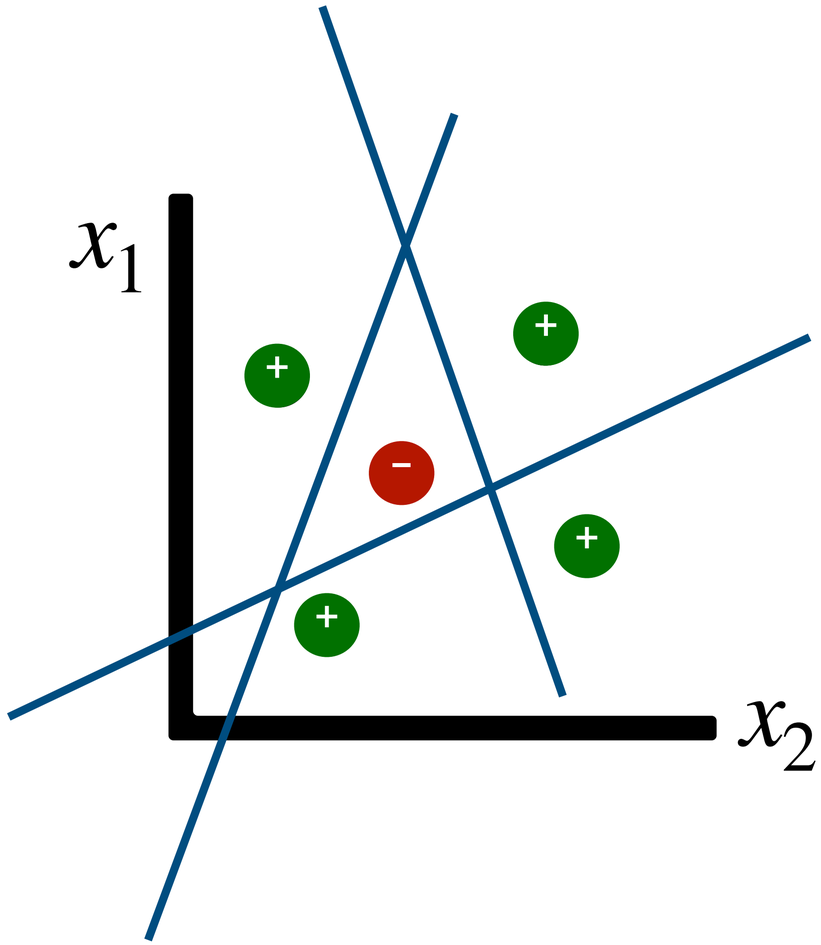
Gradient kann bestimmt werden!

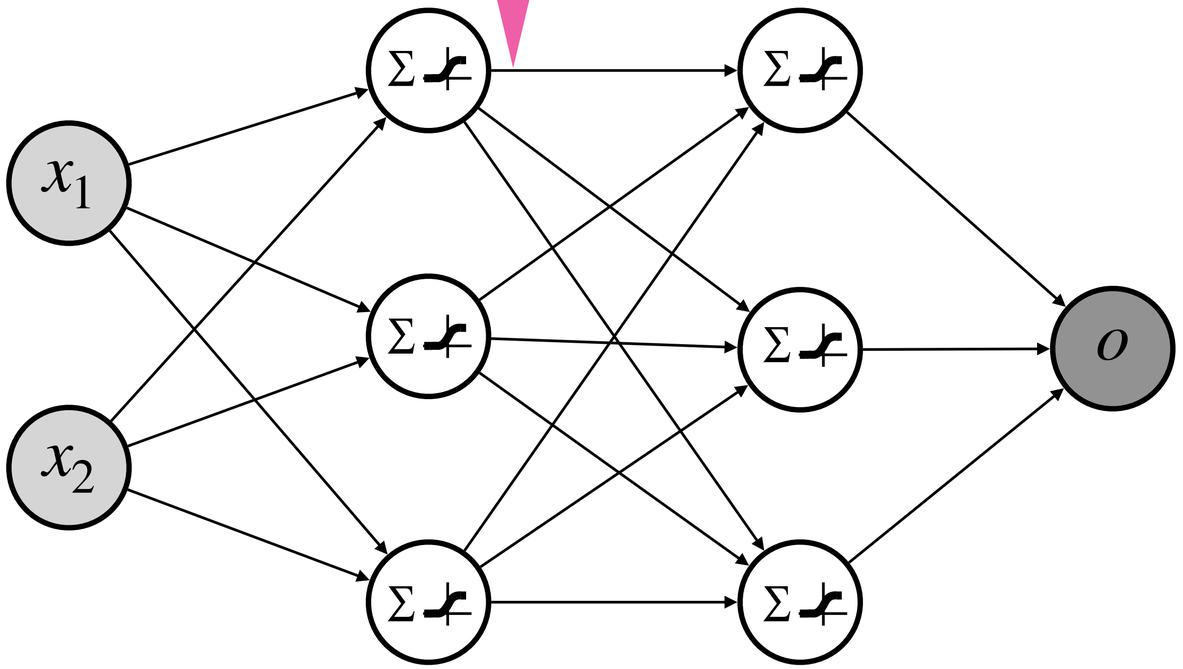
II.
Deep Learning

2. Mehrschichtige Netzwerke

Kombination von

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





Eingabeschicht

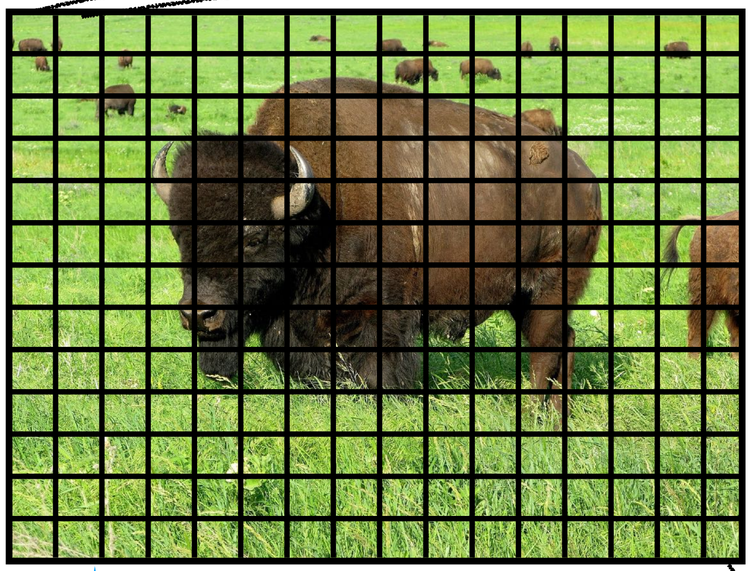
Ausgabeschicht

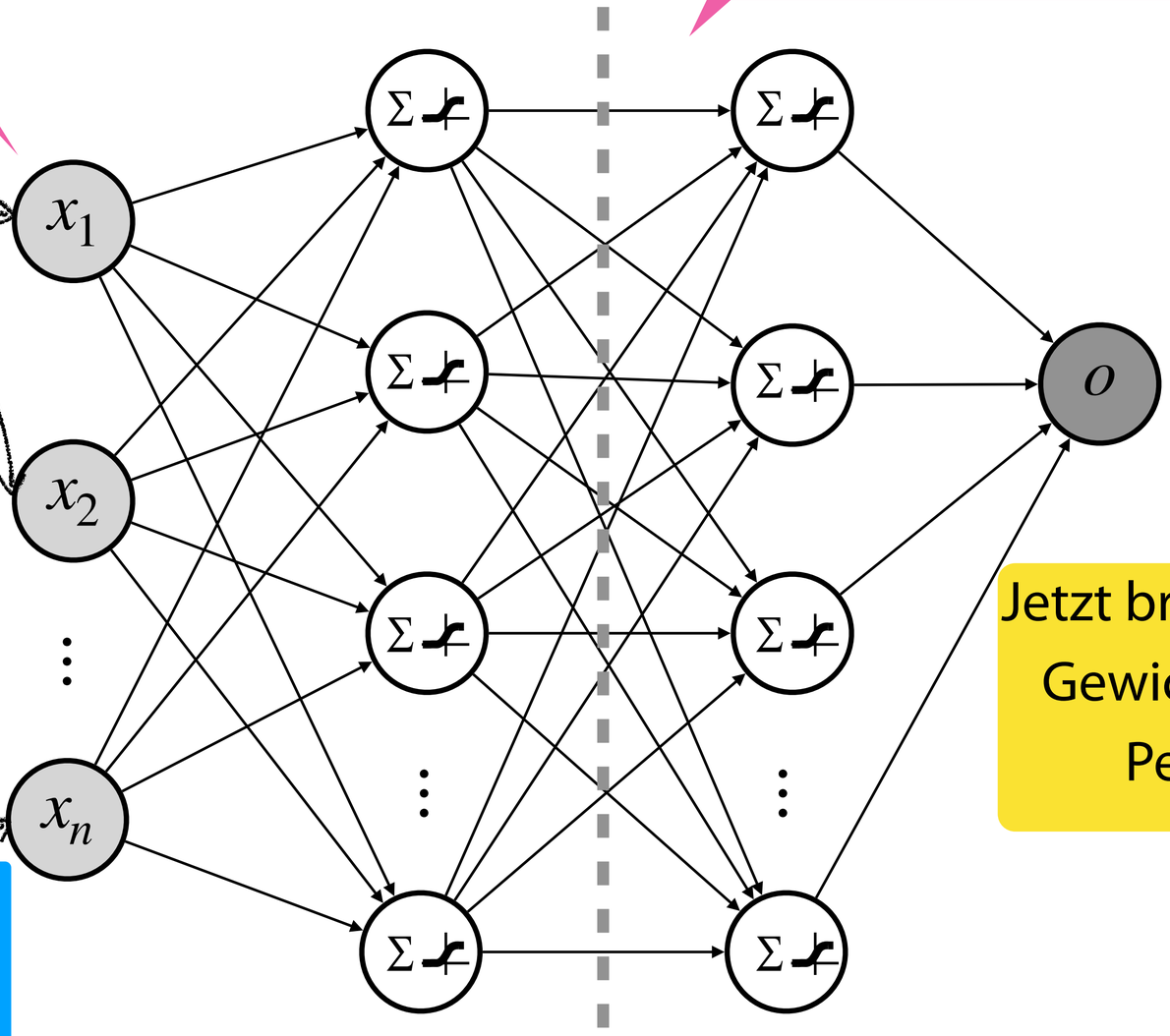
„Hidden Layer“

Beispiel

Eingabevektor mit Wert für jeden Pixel

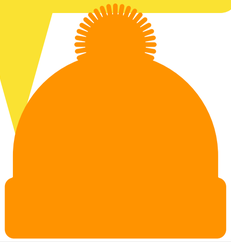
Womöglich weitere „Hidden Layer“





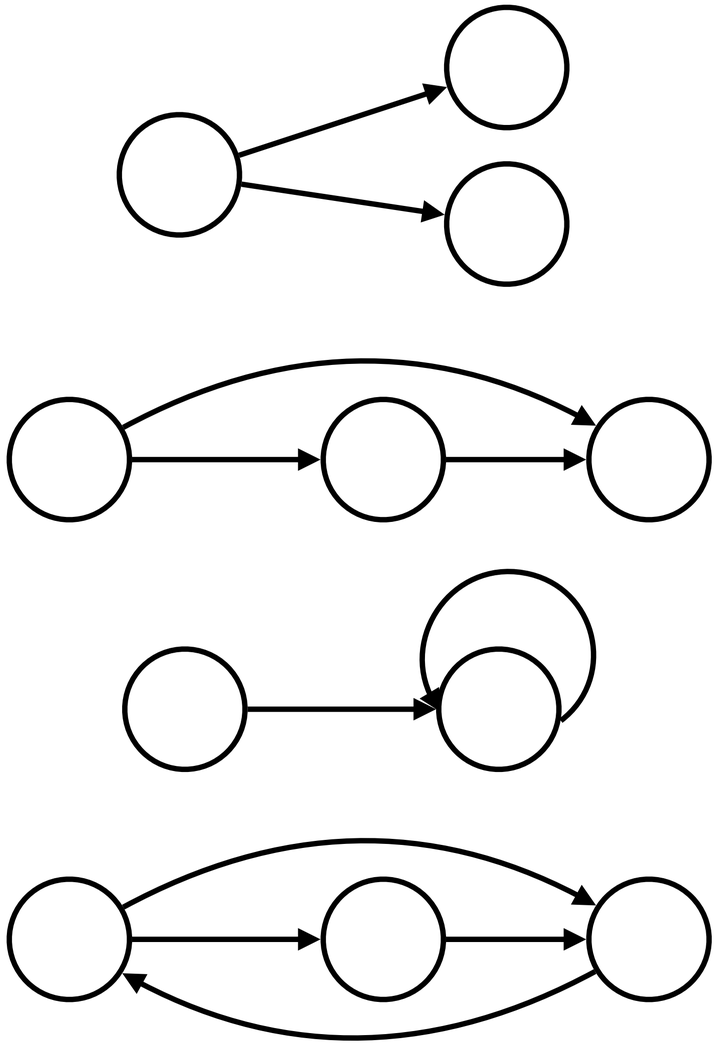
- Gegeben viele Bilder verschiedener Tiere
- Ergebnis zu „Wisent?“ für jedes Bild

Jetzt brauchen wir die Gewichte für jedes Perzeptron!



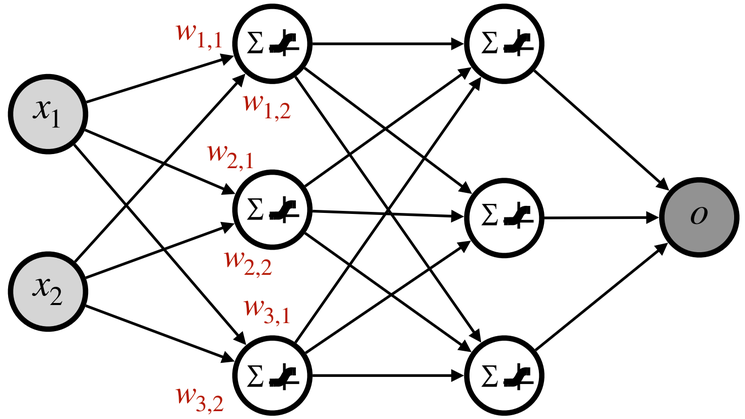
Lernen

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



Gewichte bestimmen

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



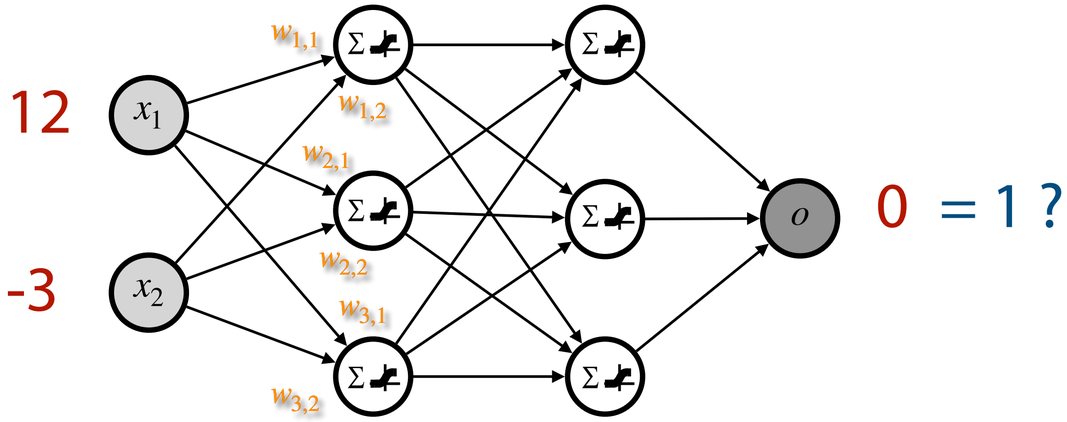
Lernen der Gewichte (Idee)

1. Gewichte zufällig initialisieren
2. Optimieren der Gewichte
   i. Ergebnis (Ausgabe) für einen Trainigsdatensatz 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 Trainigsdaten genutzt



Gradientenabstieg (Idee)

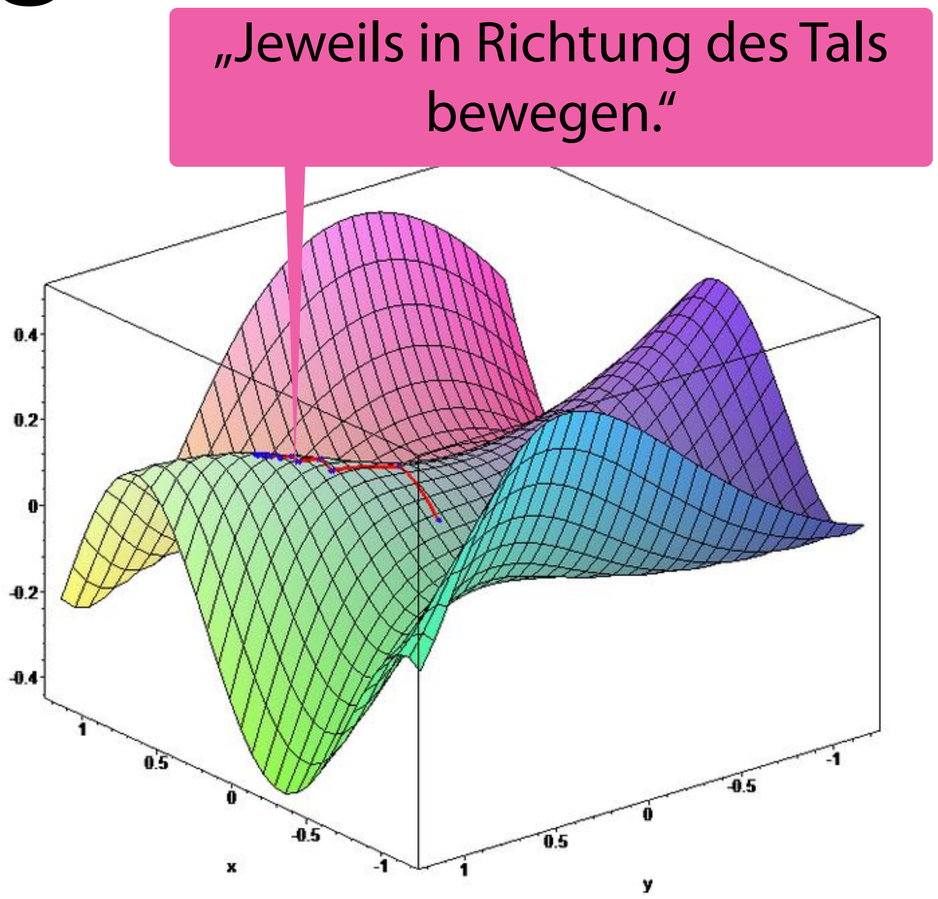
- „Wie sollte man die Gewichte anpassen, um den Fehler zu verringern?“
- Bestimmung einer Fehlerfunktion
  - Differenz zwischen Wunschausgabe und Netzwerkausgabe
  - Netzwerk ist eine Hintereinanderschaltung von Funktionen

    → 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)



https://commons.wikimedia.org/wiki/File:Gradient_ascent_%28surface%29.png

Mittels Python

- Pakete PyTorch (Facebook) und TensorFlow (Google)
  - Modellierung von Netzwerken
  - Durchführen des Lernes (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

```
import torch
```

- Test (GPU Treiber)

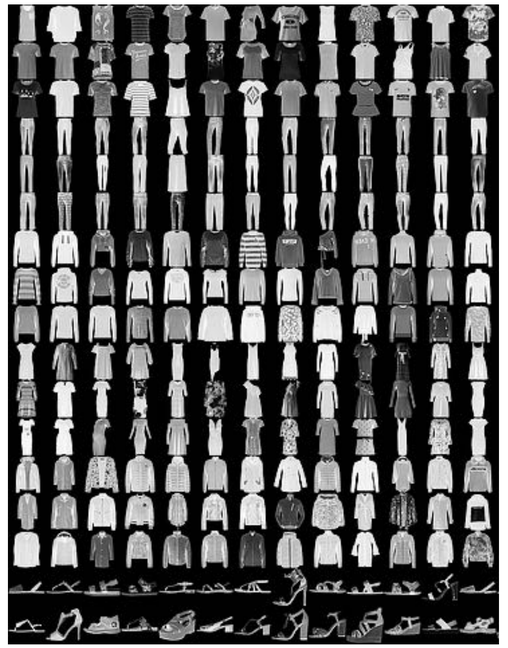
```
torch.cuda.is_available()
```

Passenden Befehl auf der Webseite nachschlagen

Intern Nutzung von C++ und CUDA

Beispiel

```python
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
```

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

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

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

```
Using cuda device
```

Falls möglich, eine GPU nutzen

Beispiel

```
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)
    )

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

model = NeuralNetwork().to(device)
print(model(X).shape)
```

```
torch.Size([64, 10])
```

Oberklasse für „Neuronales Netzwerk“ nutzen

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

Netzwerk aus Modulen zusammenbauen (sequentiell hintereinader):

Eingabe 644 Pixel → Ausgabe eine von 10 Klassen

ReLU (rectified linear unit)

$y = max(0,x)$

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

Für einen Eingabevektor durchpropagieren.

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

Beispiel

Fehlerfunktion, hier als Kreuzentropie

Fehlerminimierung, hier *Stochastic Gradient Descent*

Quasi approxiamtiver Gradientenabstieg

```
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()
```

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

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

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

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

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

    test_loss /= num_batches
    correct /= size

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

Modell in den Evaluationsmodus versetzen.

Im folgenden Block keine Gradienten benötigt

Vorhersage bestimmen, Fehler als Wert speichern

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

Normalisieren und Werte ausgeben

https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html

Beispiel

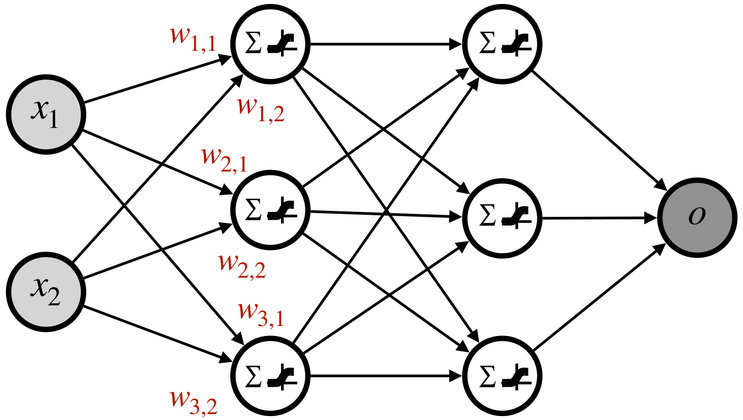
- 5 Epochen:
- Modell mit Trainigsdaten in Batches trainieren
- Modell mit Testdaten prüfen

```
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
```

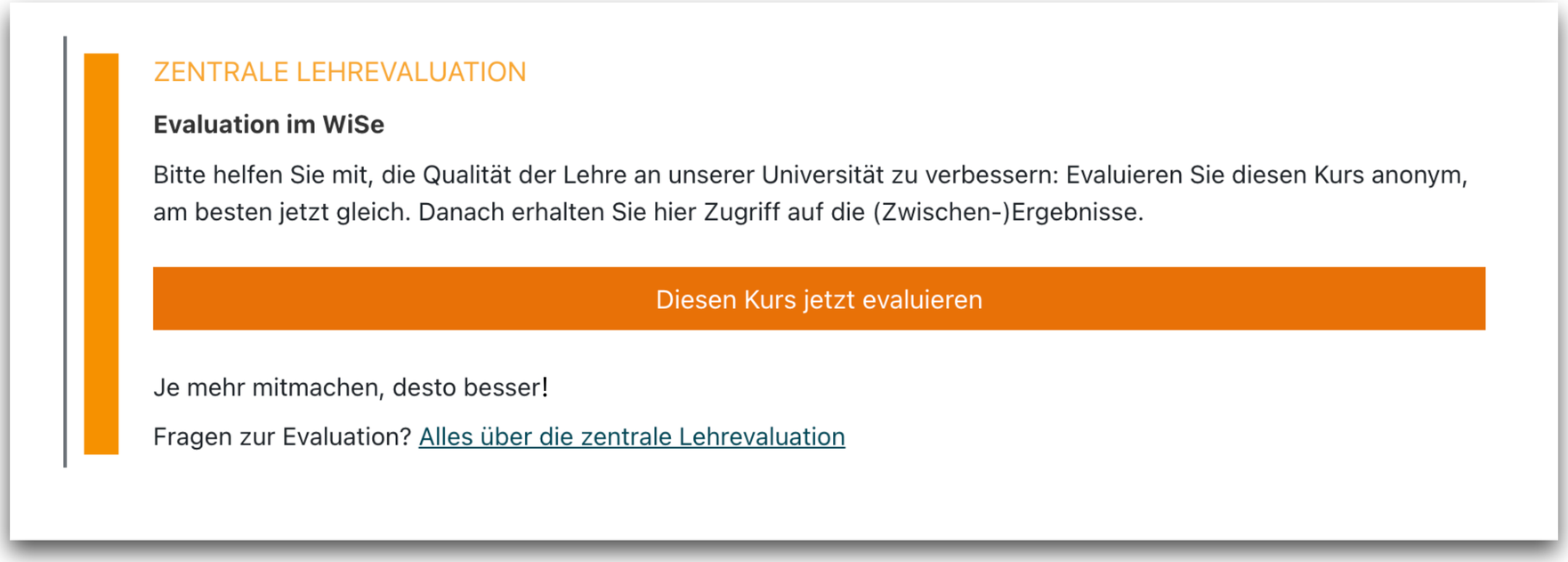
Zwischenfazit

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



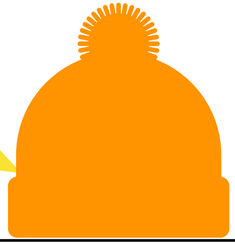
III.
Lehrevaluation

Evaluation



Besprechung der Ergebnisse

Gerne später noch teilnehmen!



IV.
Projektaufgaben

Lösungsvorschlag Aufgabe 4

Projektaufgabe 4

„Datenverarbeitung und -darstellung“

- Vorstellung möglicher Lösungen

  (werden nicht in Moodle hochgeladen!)

V.
Transformer Sprachmodelle

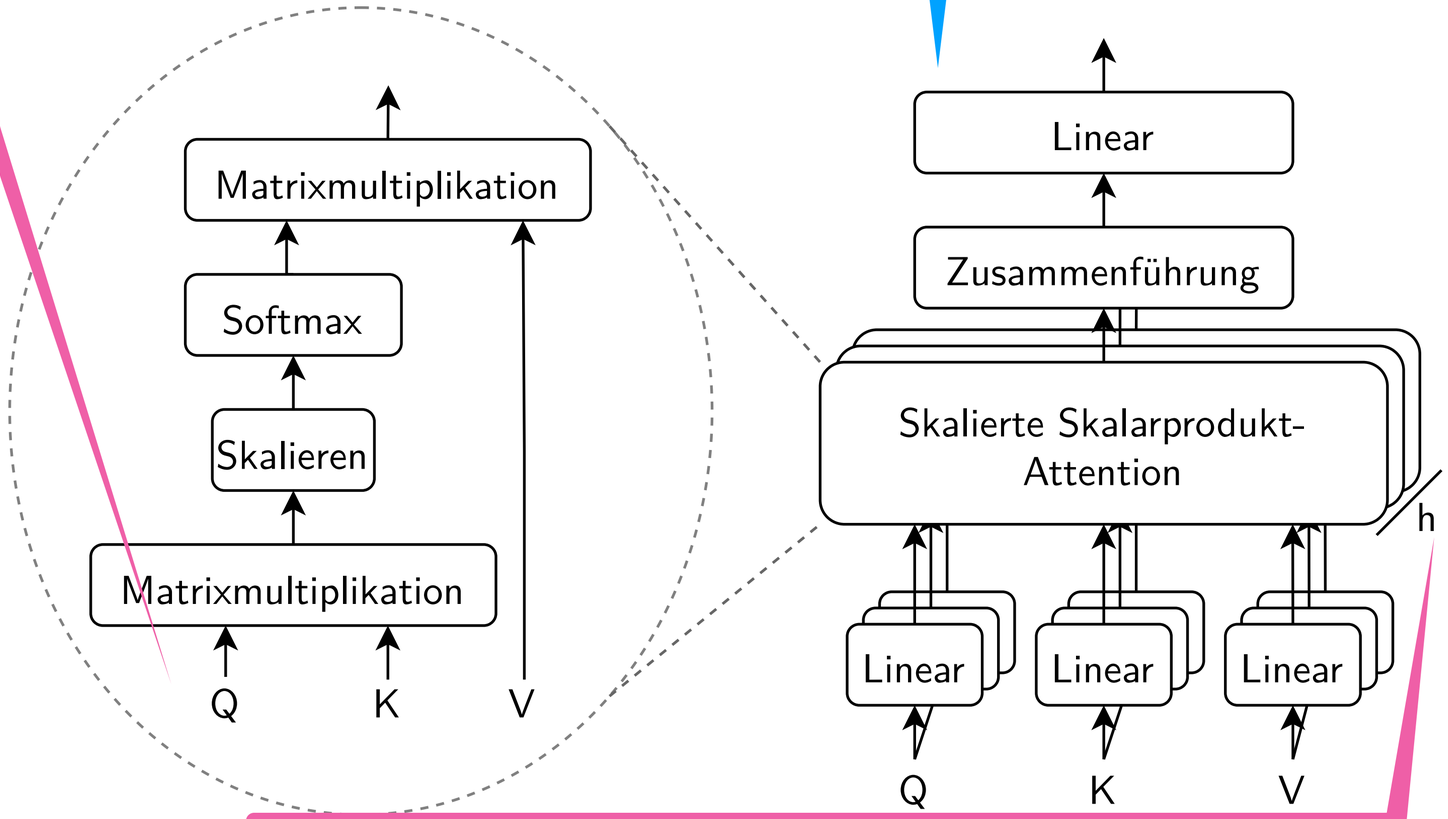
1. Idee

Attention

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

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

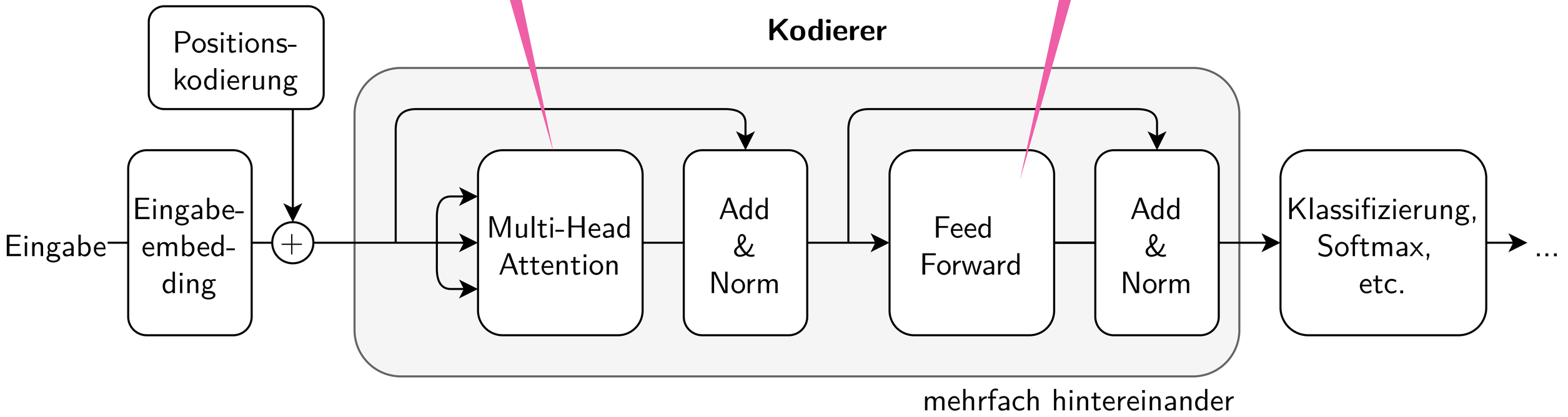


Aufteilen in *h* Bereiche → *h* (relevante) Relationen sollen gefunden werden können

Transformer

Attention von der Folie zuvor

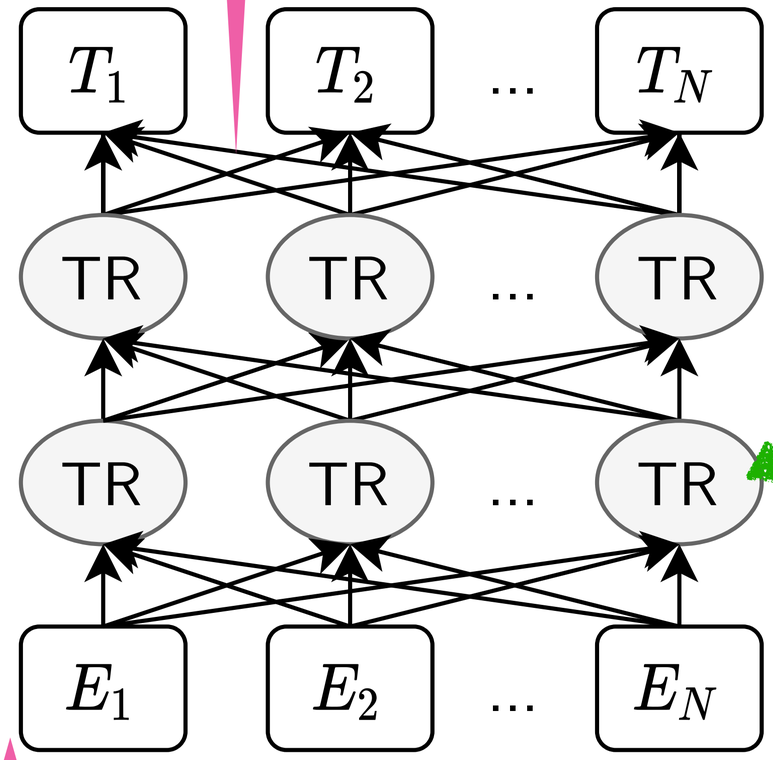
Klassisches Perzeptron-Netzwerk

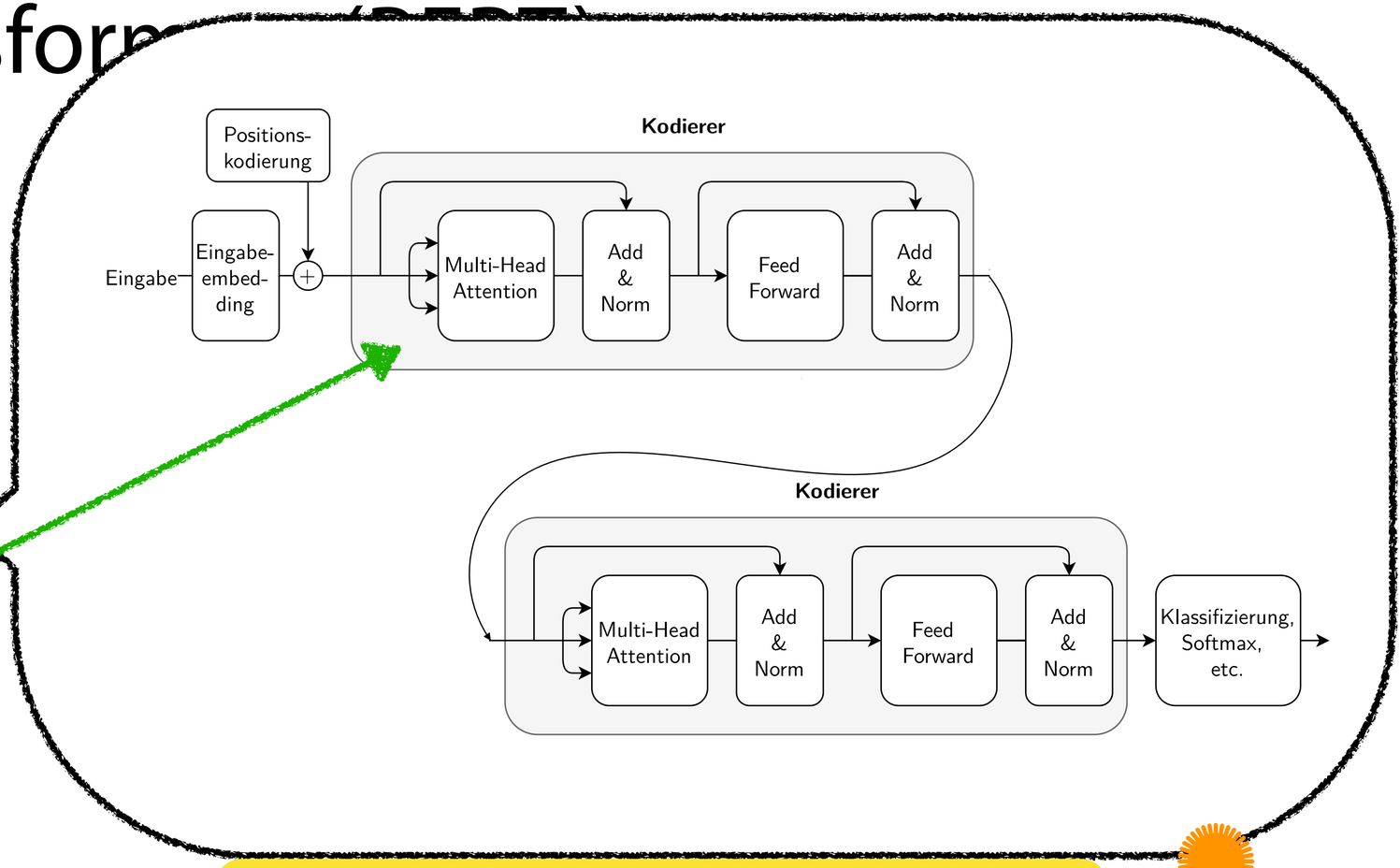


- Kernbaustein für Verarbeitung jedes Wortes (Token)
- Grauer Teil mehrfach hintereinander geschaltet

Bidirectional Encoder Representations from Transform

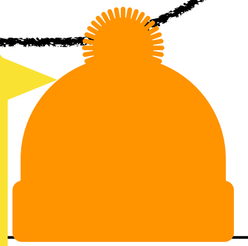
Schichten zwischen den verschiedenen Eingaben verbunden





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)

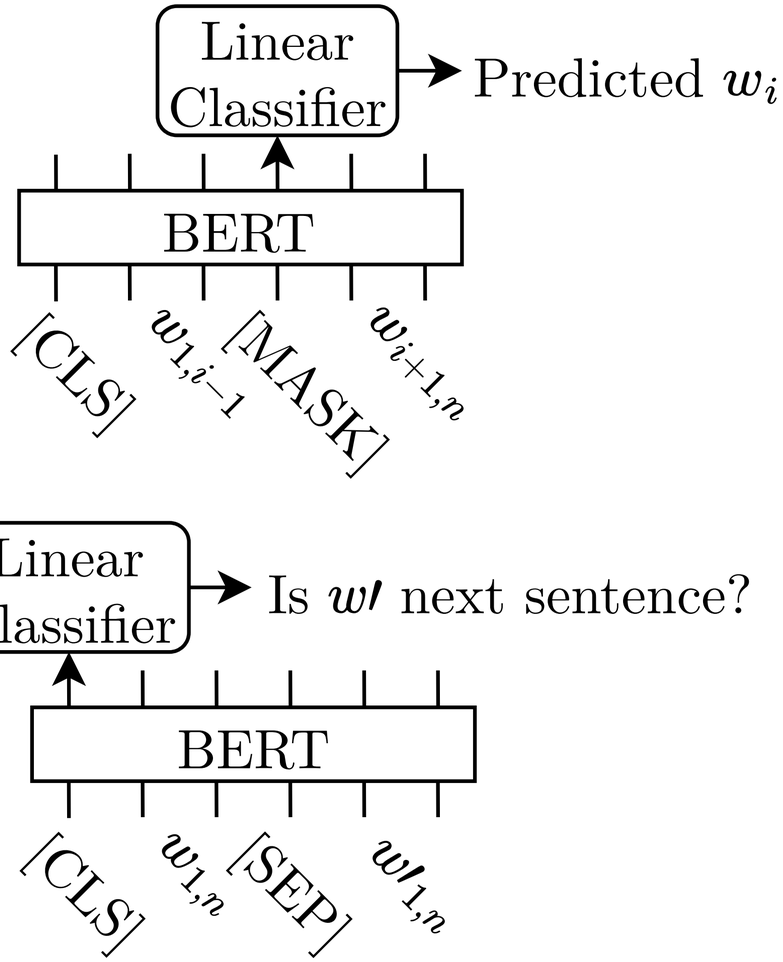
Wörter werden gleichzeitig verarbeitet, Modell kann Position somit nur durch hinzuaddierten Vektor erkennen.

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

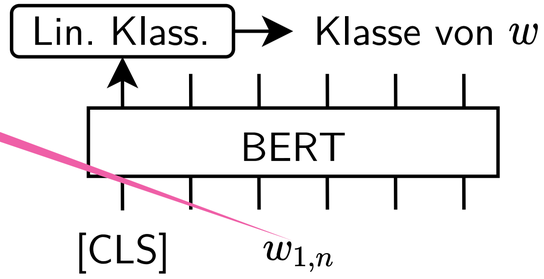


BERT: Anwendung

Ein/ mehrere zusammenhängende Sätze

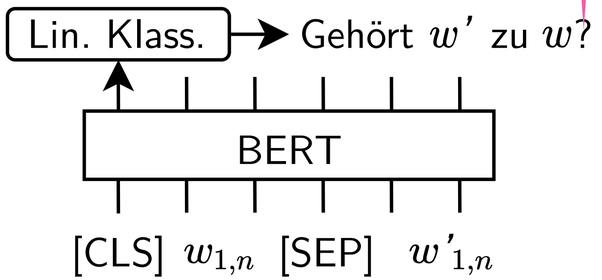
- Erste Vektorausgabe
  - Repräsentation des ganzen Texts
- Weitere Vektorausgaben
  - Repräsentation des jeweiligen Wortes

Sequenzklassifizierung

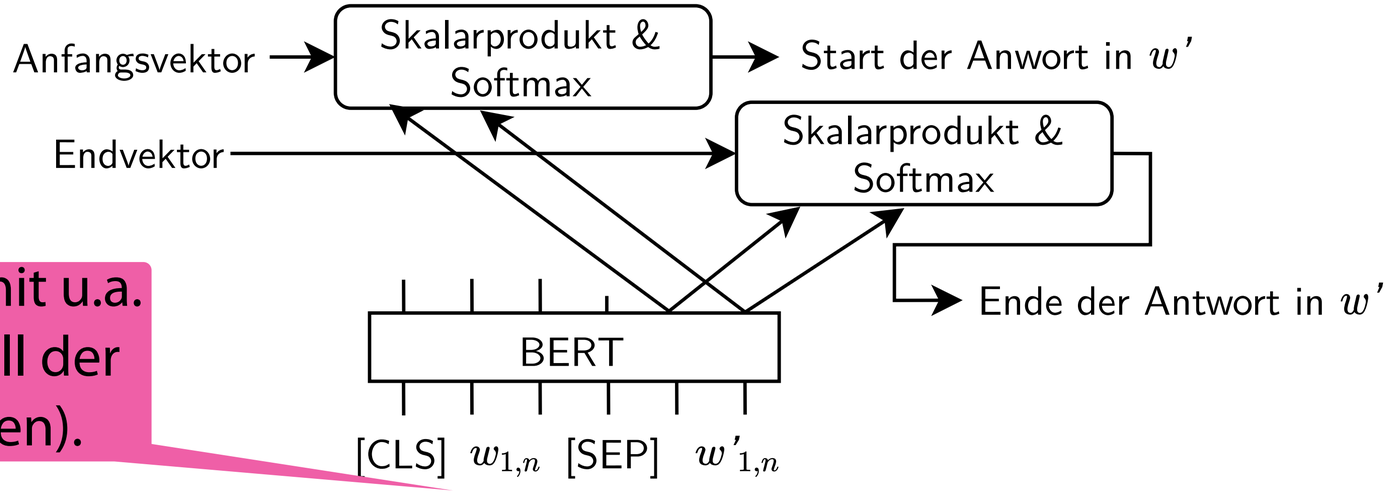


Relation zwischen zwei Sätzen gesucht

Nächster-Satz Vorhersage



Fragebeantwortung



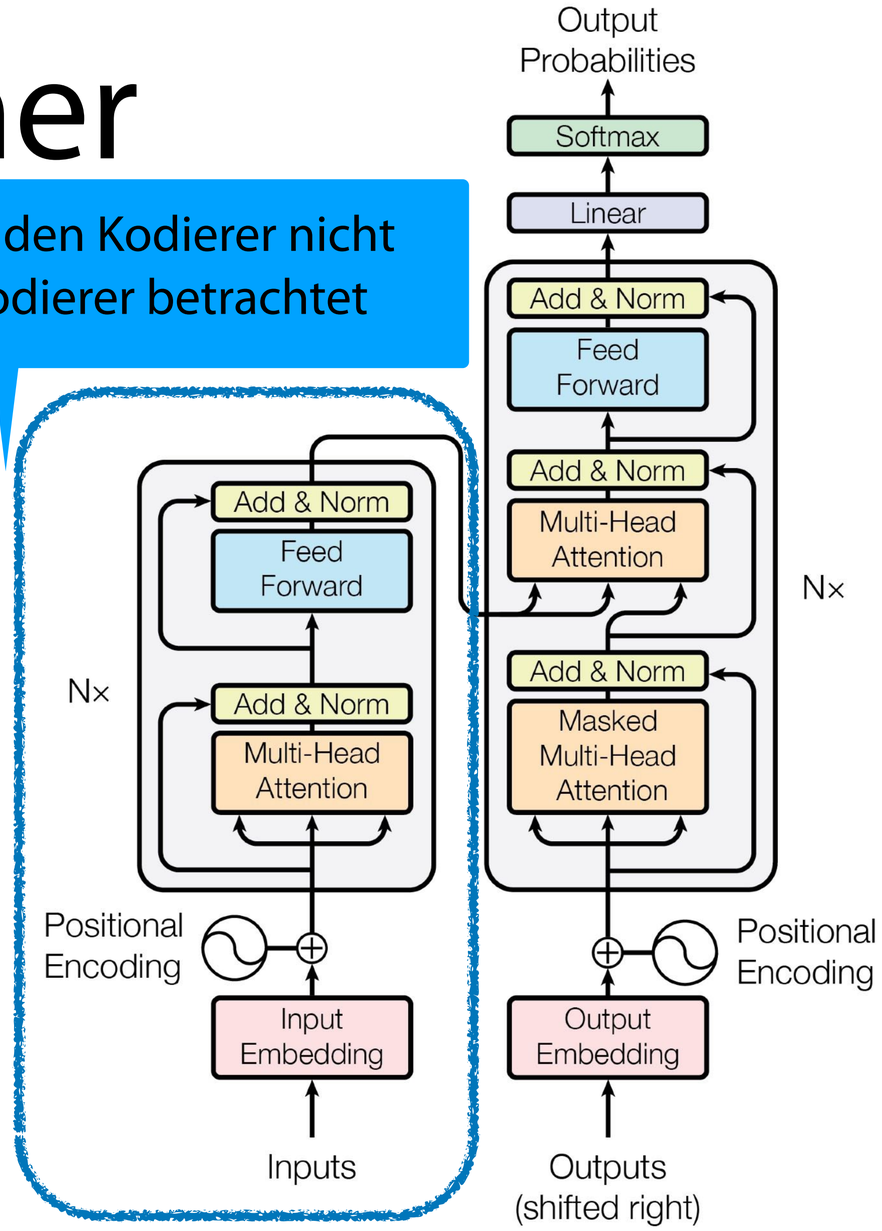
Vorne Frage, hinten Text mit u.a. der Antwort (Ziel: Intervall der Worte mit Antwort finden).

Transformer

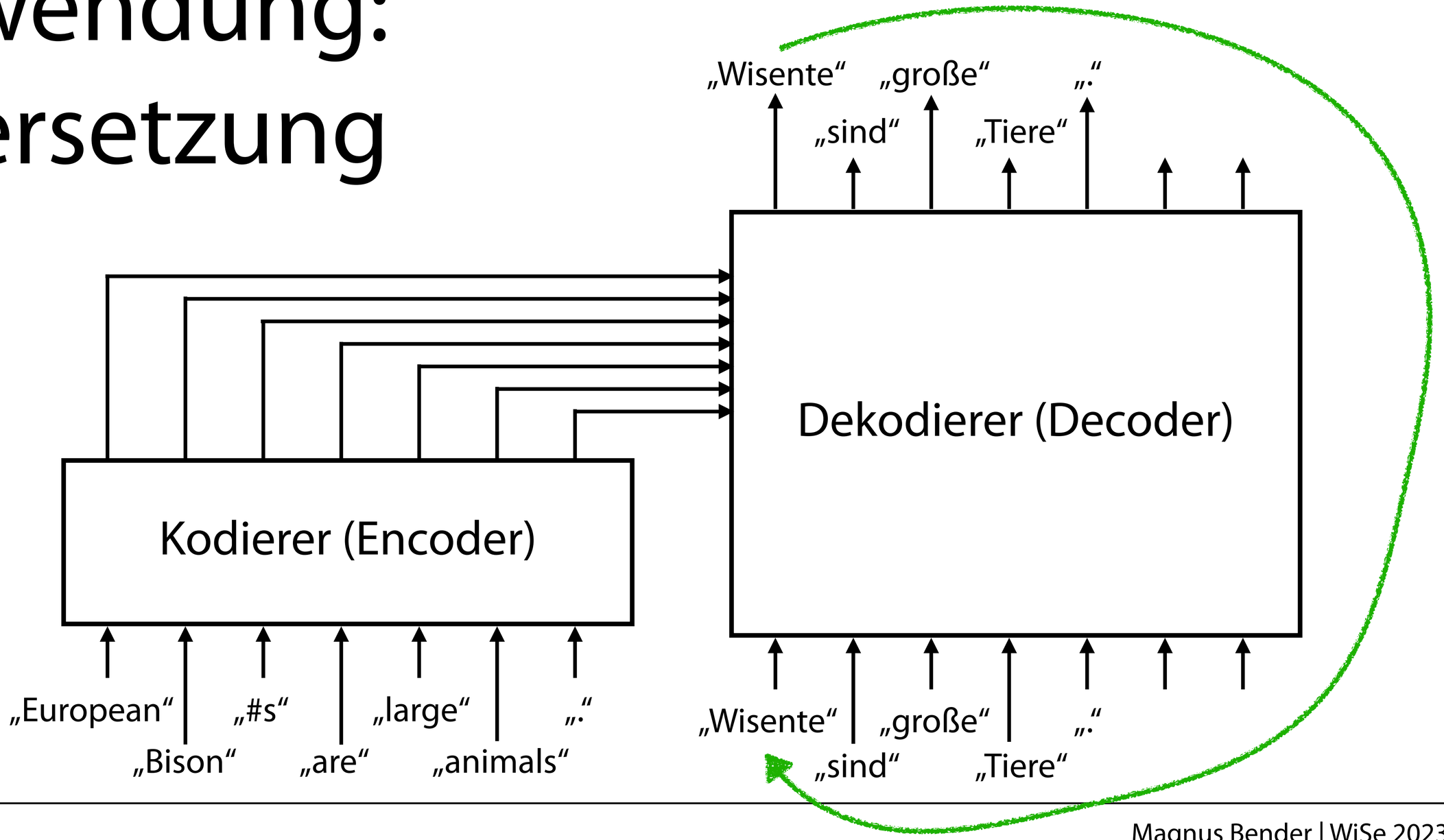
- Kodierer erzeugt Vektorrepräsentation gegeben Text
- Dekodierer erzeugt Text gegeben Vektorrepräsentation

  → Übersetzung zwischen 2 Sprachen

  → Erzeugen „neuen“ Texts

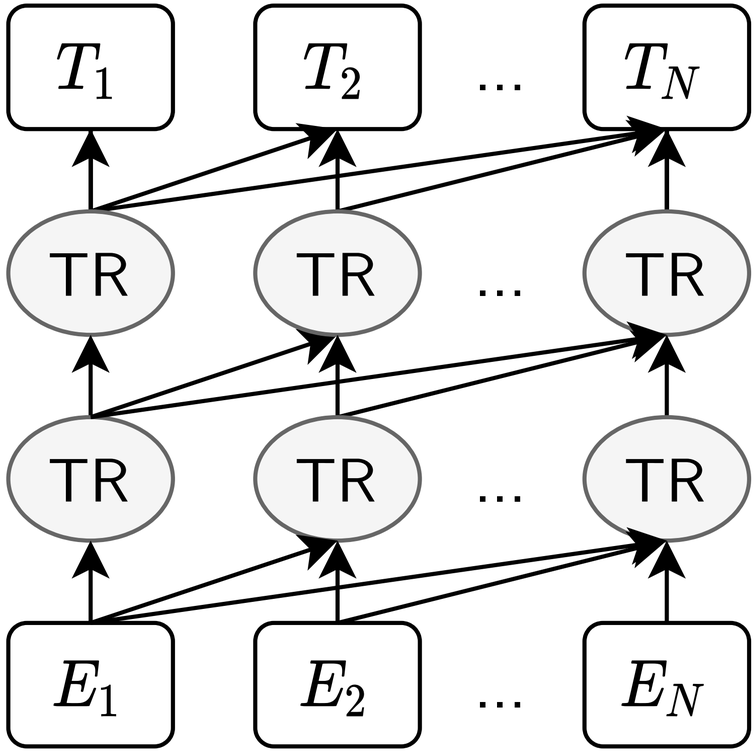


Anwendung: Übersetzung



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



Radford et al. „Improving Language Understanding by Generative Pre-Training“, **2018**
Radford et al. „Language Models are Unsupervised Multitask Learners“, **2019**
Brown et al. „Language Models are Few-Shot Learners“, **2020**

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.

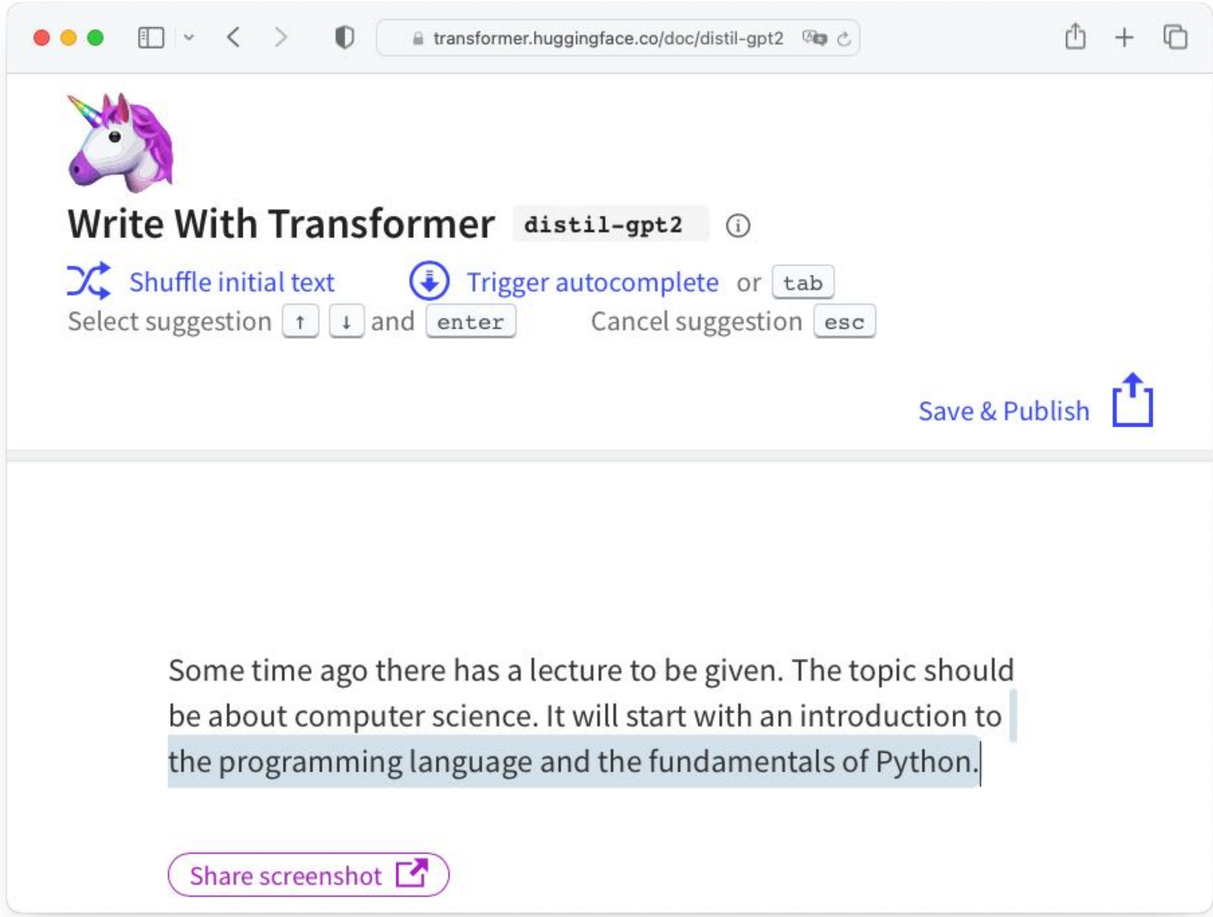
```
from transformers import pipeline
from transformers import BertModel
```

CUDA-Treiber und GPU benötigt, geht auch in Docker

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



Pipelines

```
from transformers import pipeline

classifier = pipeline("sentiment-analysis")
```

```
No model was supplied, defaulted to distilbert-base …
Using a pipeline without specifying a model name and
  revision in production is not recommended.
Downloading: 100%|████████████████████████████████| 629/629
Downloading: 100%|████████████████████████████████| 268M/268M
Downloading: 100%|████████████████████████████████| 48.0/48.0
Downloading: 100%|████████████████████████████████| 232k/232k
```

```
print(classifier("I like bisons."))
```

```
[{'label': 'POSITIVE', 'score': 0.9975264668464661}]
```

```
print(classifier("I hate to drop my plate."))
```

```
[{'label': 'NEGATIVE', 'score': 0.9376800656318665}]
```

Pipelines

```python
from transformers import pipeline
unmask = pipeline("fill-mask")
```

```
No model was supplied, defaulted to distilroberta-base …
Using a pipeline without specifying a model name
  and revision in production is not recommended.
Downloading: 100%|████████████████████| 480/480
Downloading: 100%|████████████████████| 331M/331M
Downloading: 100%|████████████████████| 899k/899k
Downloading: 100%|████████████████████| 456k/456k
Downloading: 100%|████████████████████| 1.36M/1.36M
```

```python
print(unmask.tokenizer.mask_token)
```

```
<mask>
```

```python
print(unmask("I went <mask> from school."))
```

```
[
  { 'I went home from school.' … },
  { 'I went away from school.' … },
  { 'I went back from school.' … }, …
]
```

Beispiel I: BERT

```python
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']
```

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

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

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

```
random sentence
```

Beispiel II: 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 = "Thus, one might need a knive to cut the food."

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

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

```
next sentence
```

Pre-Trained Modell und Tokenizer (herunter-)laden

Vorhersage ob zwei Sätze zusammen passen.

Eingaben mittels Tokenizer für BERT vorbereiten, siehe auch Folie 39.

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

Ausgabe auswerten

https://huggingface.co/docs/transformers/v4.25.1/en/model_doc/bert#transformers.BertForNextSentencePrediction

BERT Fine-Tuning

```python
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)
```

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.

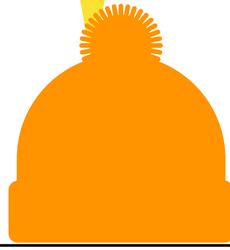
BERT für „unseren“ Anwendungsfall nutzen.

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



BERT Fine-Tuning

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

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

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

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

Evaluationsfunktion erstellen

Vorhersage „ausrechnen“

Gewünschte Metriken berechnen und als Wörterbuch zurückgeben.

Passendes Modell laden und den Trainer

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.train()
trainer.evaluate()


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

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

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

Dauert ohne eine GPU sehr lange …

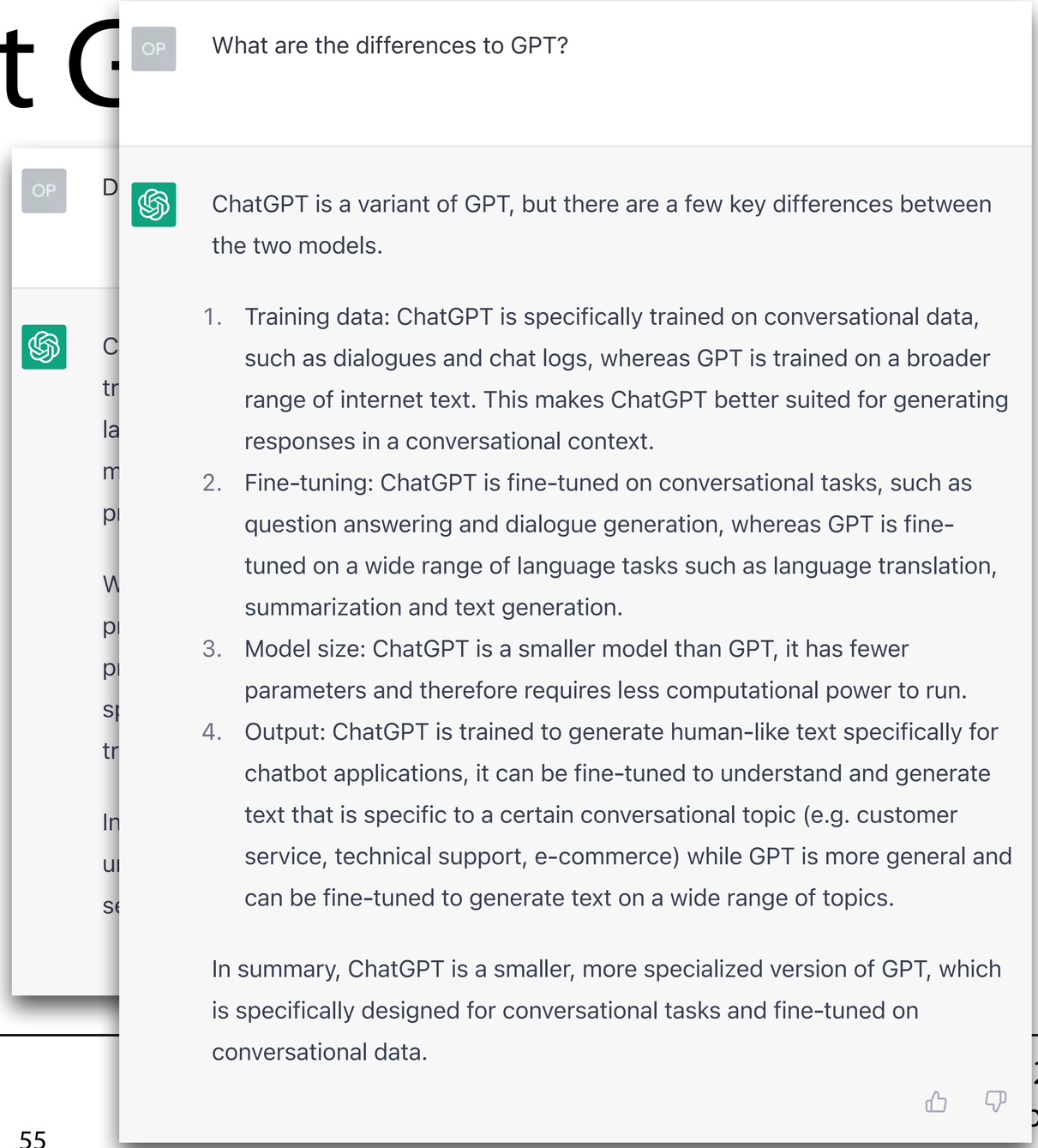
Evaluation durchführen, zeigt Ergebnisse am Ende an.

Speichern des trainierten Modells (in einen Ordner).

Laden des selbst erstellten Modells.

Chat G

- Fine-Tuning von GPT 3.5 (kostenfrei) oder GPT 4 (Abo)
  - GPT 4 ist „multimodal“, d.h., nicht nur Text
- Zusätzlich
  - Überwachtes Lernen
  - Reinforcement Learning (Feedback)



https://chat.openai.com/chat

V.
Zusammenfassung

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

Zusammenfassung

II. Deep Learning

1. Perzeptron

2. Mehrschichtige Netzwerke

V. Transformer Sprachmodelle

1. Idee

2. BERT & GPT mit Python

- **Abschlussquiz** ab heute 18 Uhr bis Freitag 18 Uhr
- Abgabe der (Bonus-)Aufgabe 5 am Freitag den 16. Februar

- Teilnahme an der Evaluation im Moodle

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