RNNs
LSTMs
Bi-LSTMs

Referentinnen: Chunxue Liu, An Li, Xiang Shi, Xingying Li
Dozent: Dr. Stefan Langer
Masterseminar: Classification and Clustering
Ludwig-Maximilians-Universität, München

Inhaltsverzeichnis:

RNN
(Recurrent Neural Network)

RNN (Recurrent Neural Network)

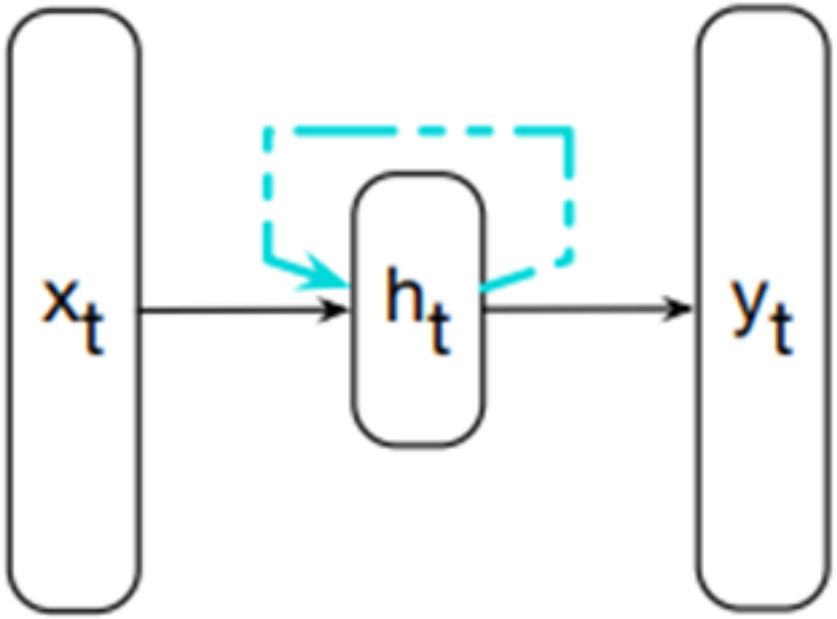
Definition:

Ein speziell für die Verarbeitung von sequenziellen Daten entwickeltes neuronales Netzwerk.

Merkmale:

1. Zeitabhängigkeit:

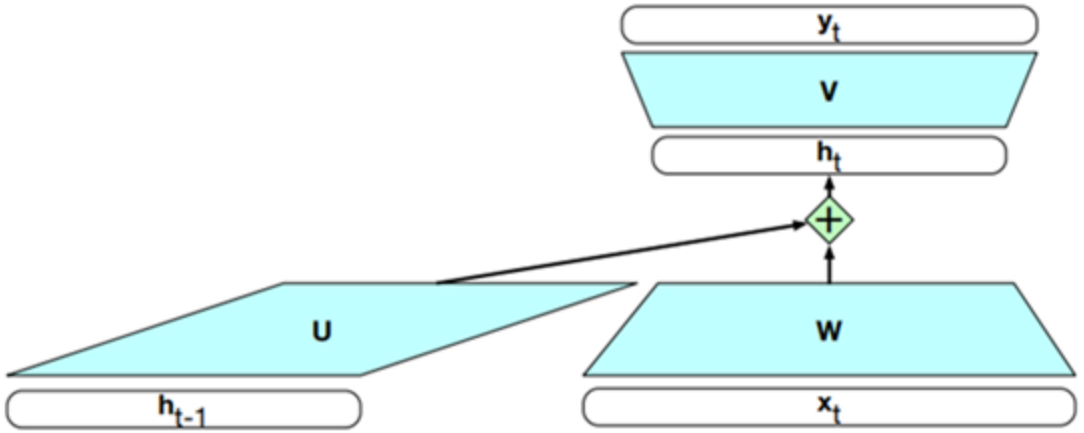
- Speichert Kontextinformationen durch rekurrente Verbindungen.
- Entscheidungen können von Informationen abhängen, die hunderte Wörter zuvor aufgetreten sind.



Quelle:J&M: Dan Jurafsky & James Martin, 2018. Speech and Language Processing (3rd ed. Draft).

Prozess in jedem Zeitschritt:

- Empfängt die Eingabe x_t.
- Kombiniert x_t mit dem vorherigen versteckten Zustand h_{t-1}, um h_t zu berechnen.
- Berechnet die Ausgabe y_t basierend auf h_t.

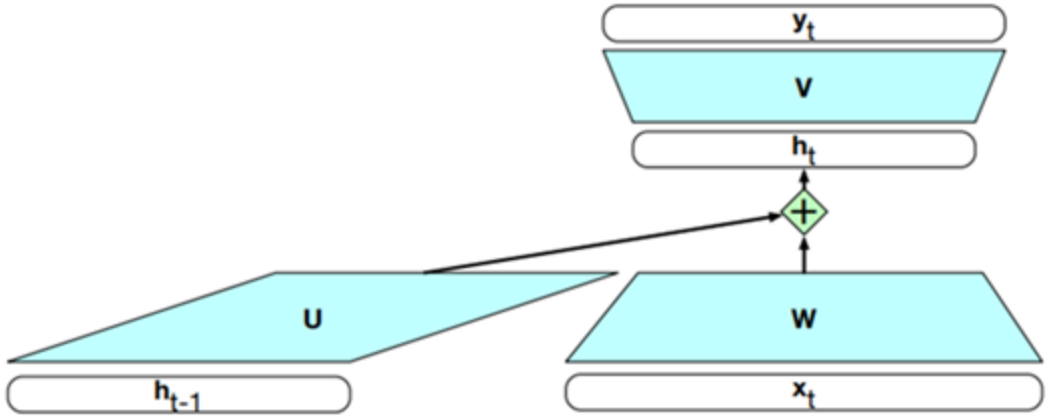


Quelle:J&M: Dan Jurafsky & James Martin, 2018. Speech and Language Processing (3rd ed. Draft)

Grundlegende Berechnungen des RNN

1. Die Berechnungsformel für den versteckten Zustand im Zeitschritt *t*:

$$h_t = g(Uh_{t-1} + Wx_t)$$



Quelle:J&M: Dan Jurafsky & James Martin, 2018. Speech and Language Processing (3rd ed. Draft)

h_{t-1}: Versteckter Zustand des vorherigen Zeitschritts.

x_t: Eingabe des aktuellen Zeitschritts.

g: Aktivierungsfunktion (z. B. tanh oder ReLU).

W: x_t wird vom Eingaberaum in den Raum der versteckten Schicht abgebildet.

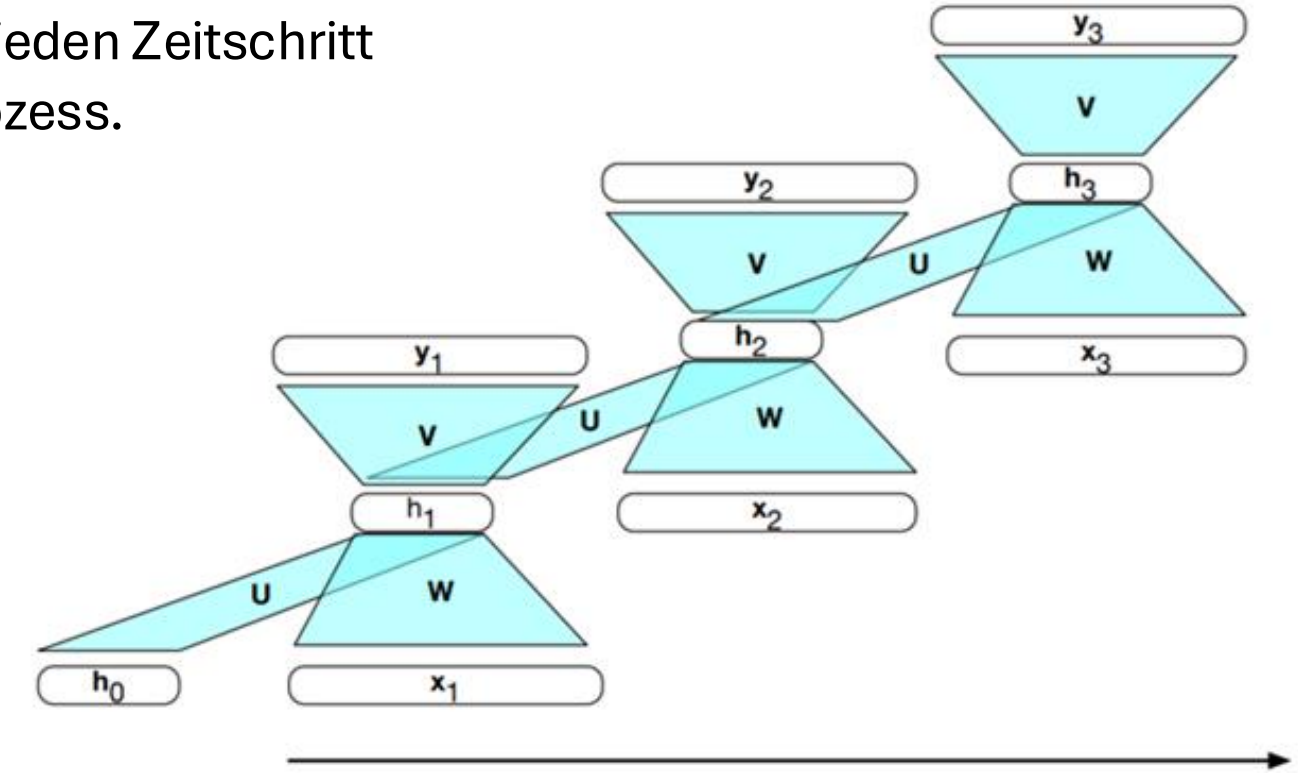
U: Der versteckte Zustand des vorherigen Zeitschritts h_{t-1} wird auf den aktuellen Zeitschritt abgebildet.

2. Berechnung der finalen Ausgabe:

- Zustandsabbildung: h_t wird durch V in den Ausgaberaum abgebildet.
- Softmax: Vh_t wird in die Wahrscheinlichkeitsverteilung y_t umgewandelt.

$$y_t = \text{softmax}(Vh_t)$$

- Der Berechnungsprozess eines RNN für jeden Zeitschritt ist ein wiederholender und rekursiver Prozess.
- Geteilte Gewichte W, U, V.

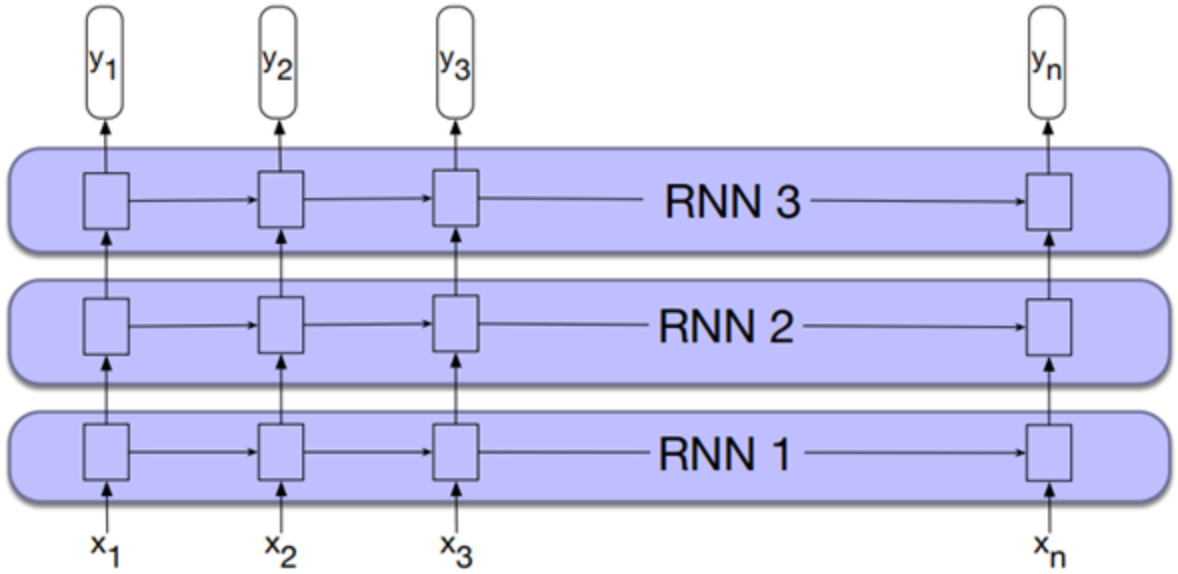


Quelle:J&M: Dan Jurafsky & James Martin, 2018. Speech and Language Processing (3rd ed. Draft)

Fortgeschrittene RNN-Architektur:

a) Stacked RNN

- Der versteckte Zustand (oder die Ausgabe) jeder RNN-Schicht dient als Eingabe für die nächste RNN-Schicht.

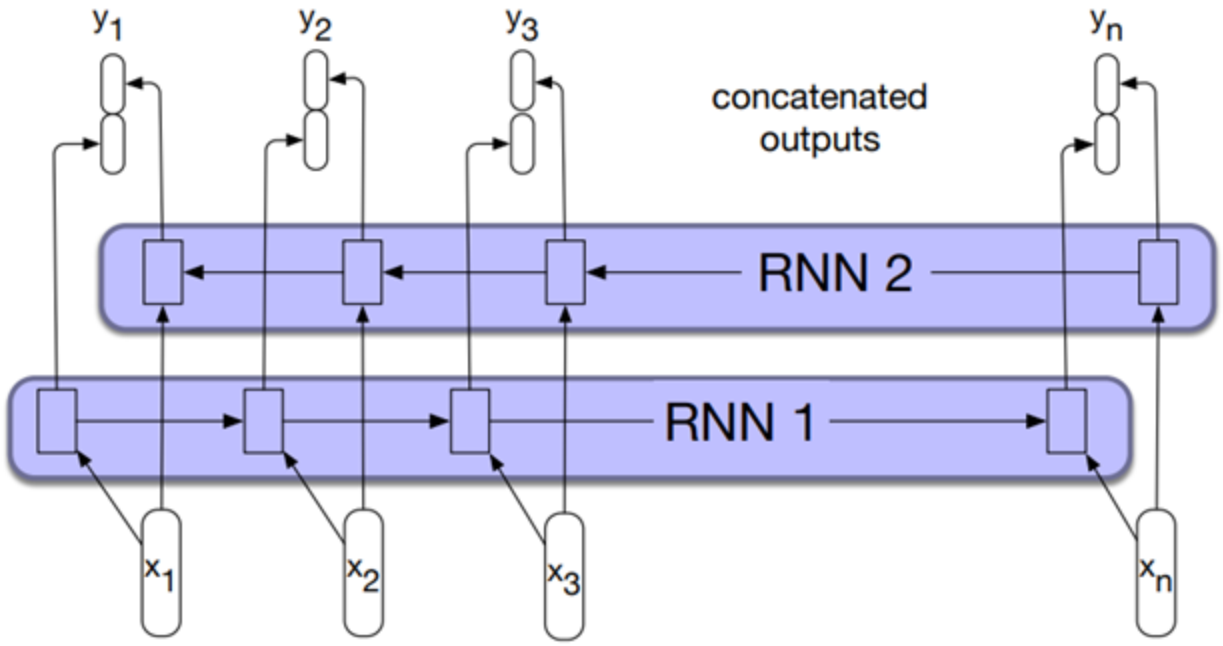


Quelle:J&M: Dan Jurafsky & James Martin, 2018. Speech and Language Processing (3rd ed. Draft)

b) Bidirectional RNN

- Gleichzeitige Verwendung von zwei RNNs, eines von links nach rechts und eines von rechts nach links, um den Kontext vor und nach dem aktuellen Punkt zu integrieren.

- Die endgültige Darstellung einer Eingabe x_k enthält die kumulierten Informationen von x0 bis x_{k-1} sowie von xn bis x_{k+1}.



Quelle:J&M: Dan Jurafsky & James Martin, 2018. Speech and Language Processing (3rd ed. Draft)

Einschränkungen von RNNs:

- **Das Problem des verschwindenden Gradienten:**

In langen Sequenzen nimmt der Gradient allmählich ab, wodurch das Netzwerk nicht in der Lage ist, langfristige Abhängigkeiten effektiv zu lernen.

In einem RNN ist die rekursive Beziehung des versteckten Zustands:

$$h_t = g(Uh_{t-1} + Wx_t)$$

Die Formel zur Gradientenweitergabe lautet:

$$\frac{\partial L}{\partial h_{t-k}} = \frac{\partial L}{\partial h_t} \cdot \prod_{i=t-k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}$$

Dabei gilt:

$$\frac{\partial h_i}{\partial h_{i-1}} = g'(Uh_{i-1} + Wx_i) \cdot U$$

Schlüsselelemente der Gradientenakkumulation:

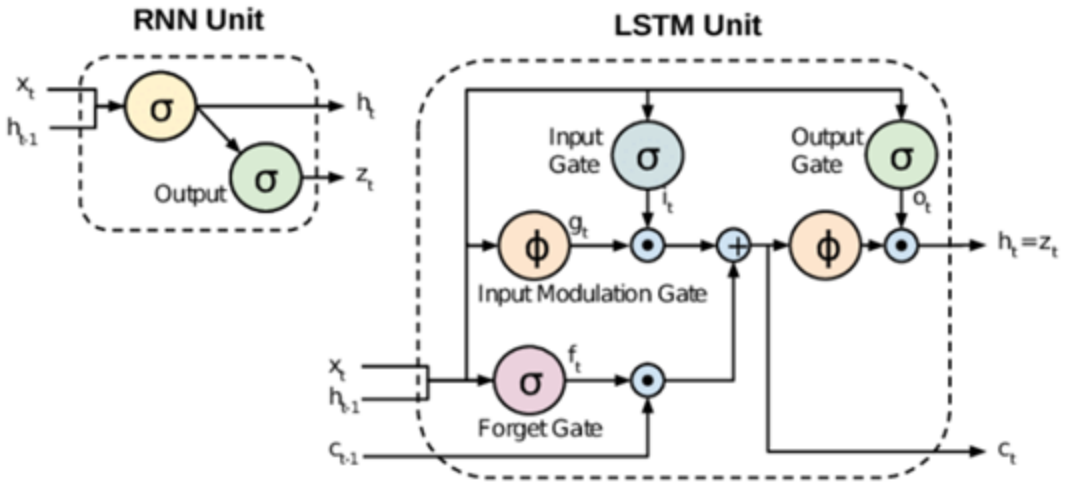
1. $g'(Uh_{i-1} + Wx_i)$: Ableitung der Aktivierungsfunktion.
2. U: Eigenschaften der Gewichtsmatrix.

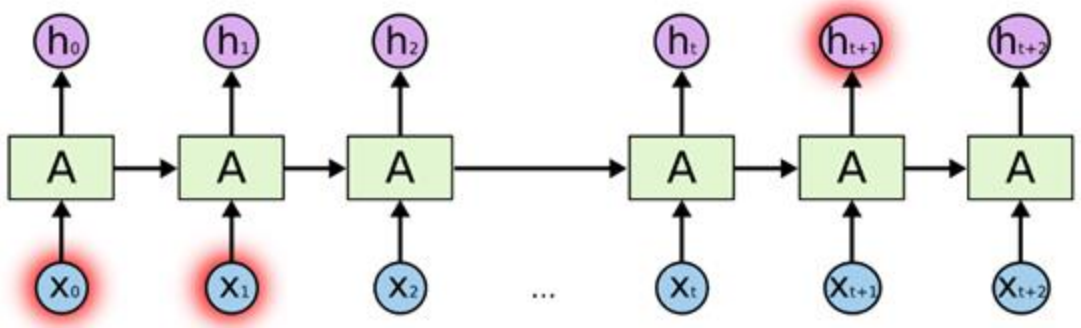
Verbesserungsmöglichkeiten:

- Einführung von LSTM oder GRU, um die Gedächtnisfähigkeit des Netzwerks zu stärken.

LSTM
(Long Short-Term Memory)

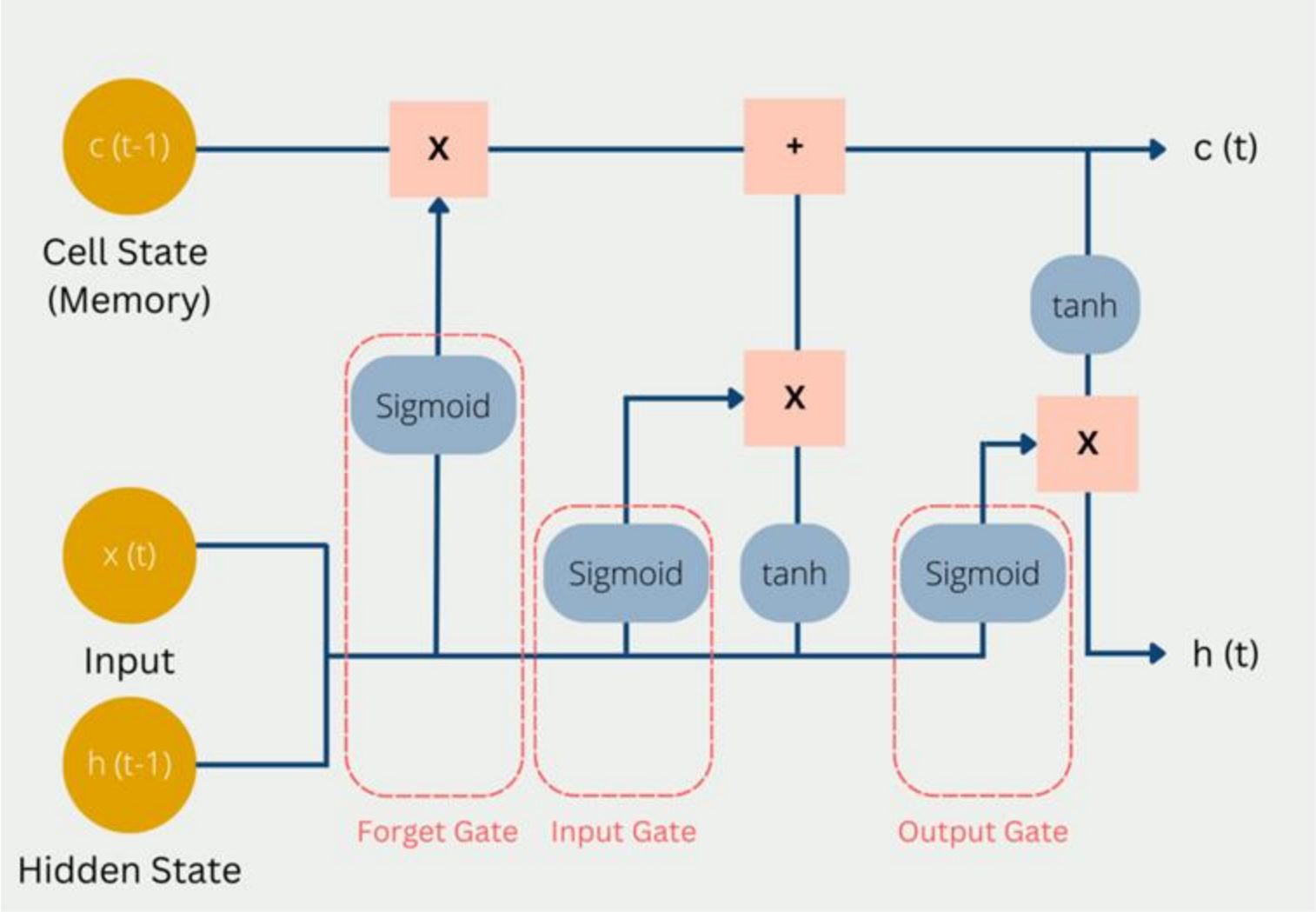
LSTM





Quelle: What is the main difference between RNN and LSTM | NLP | RNN vs LSTM – Data Science Duniya

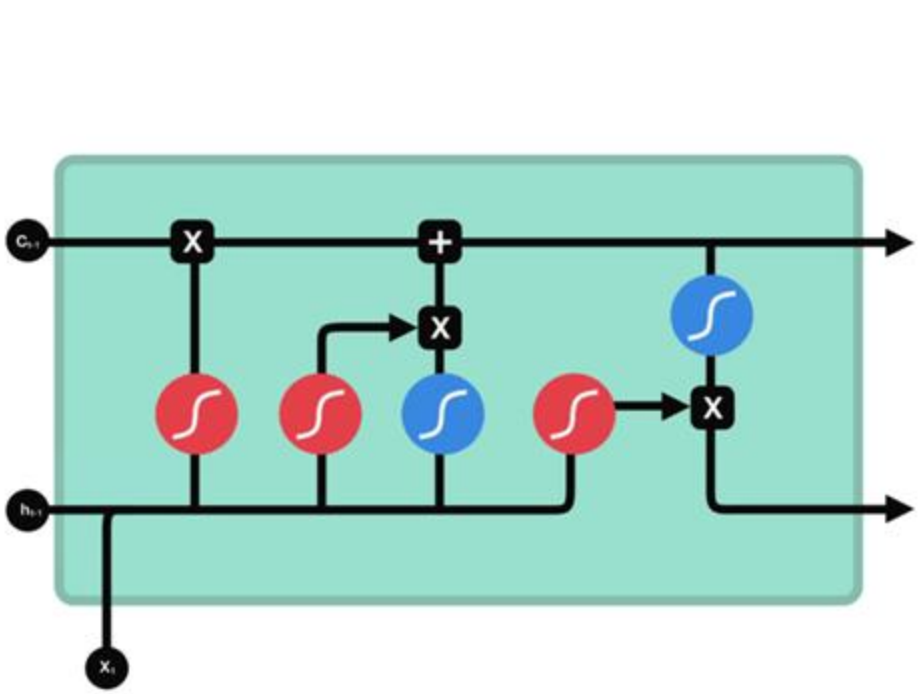
LSTM-Zellen

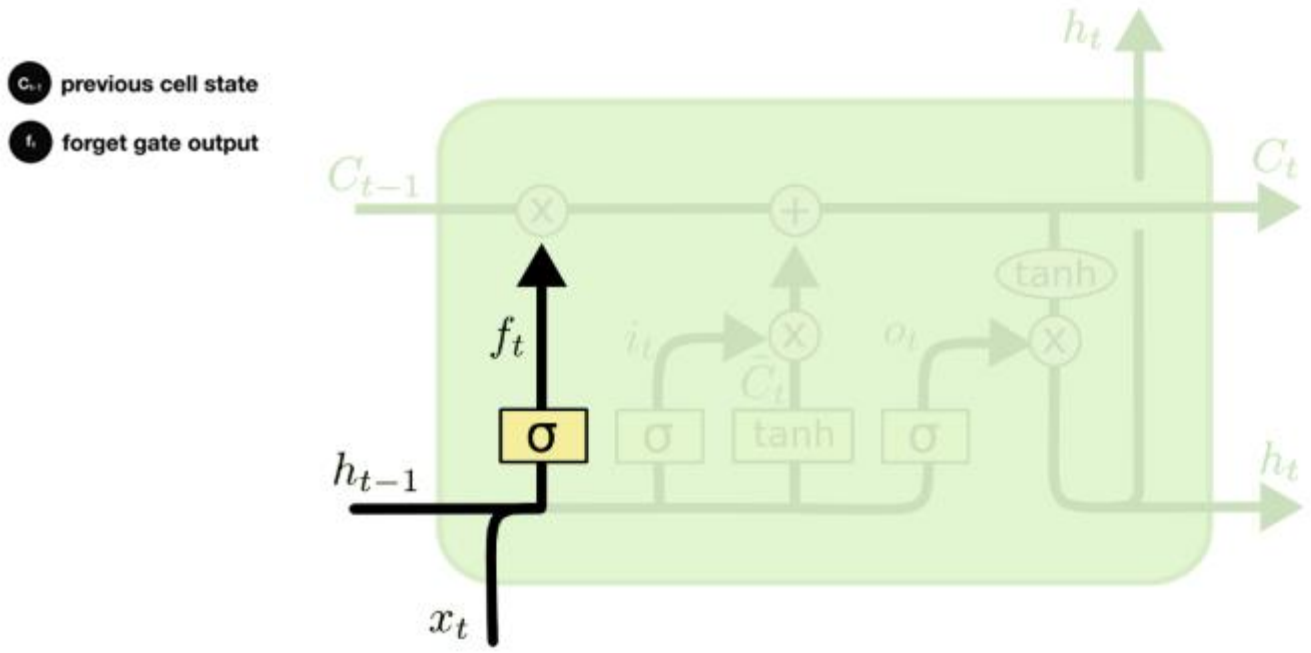


Quelle: databasecamp

Vergessens-Gate

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



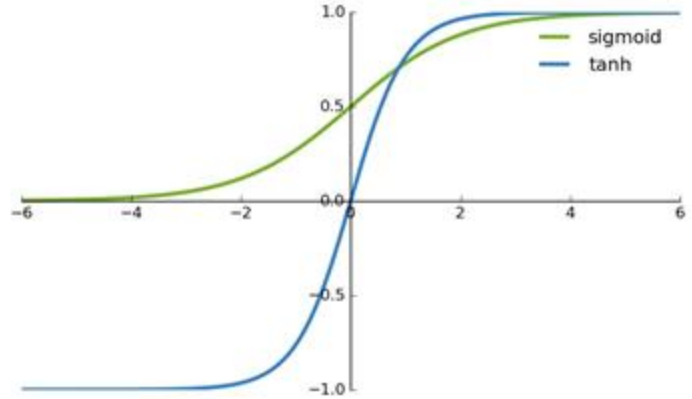


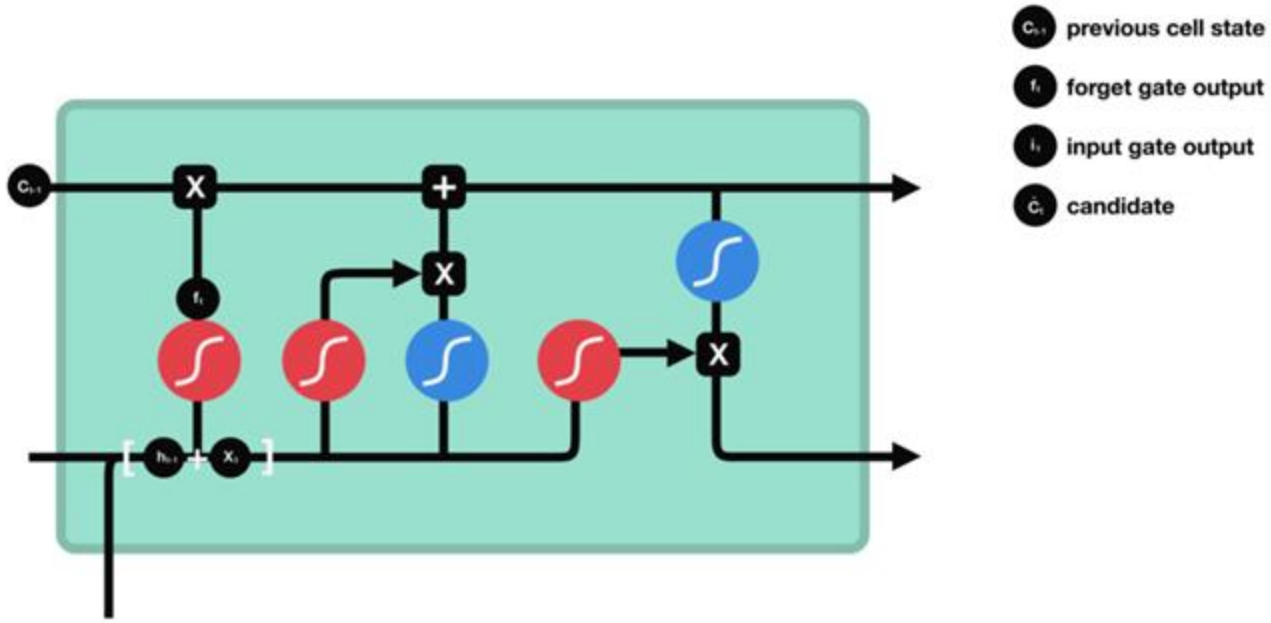
Quelle: https://blog.csdn.net/mary19831/article/details/129570030

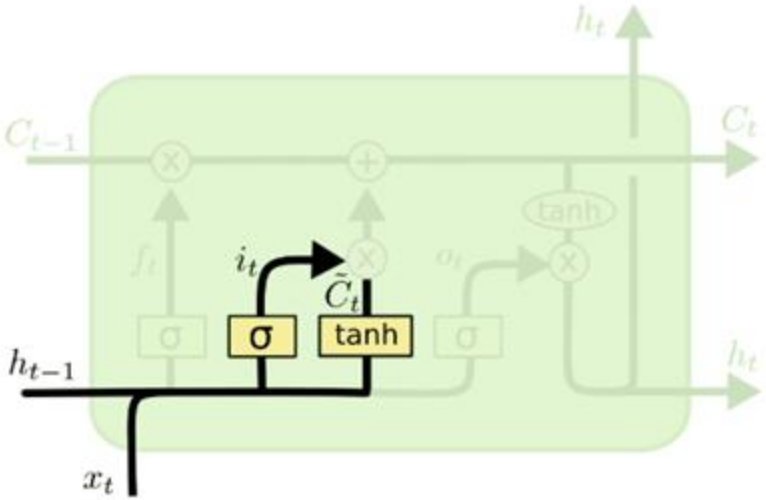
Quelle: https://blog.csdn.net/qq_39297053/article/details/136427425

Eingabe-Gate

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





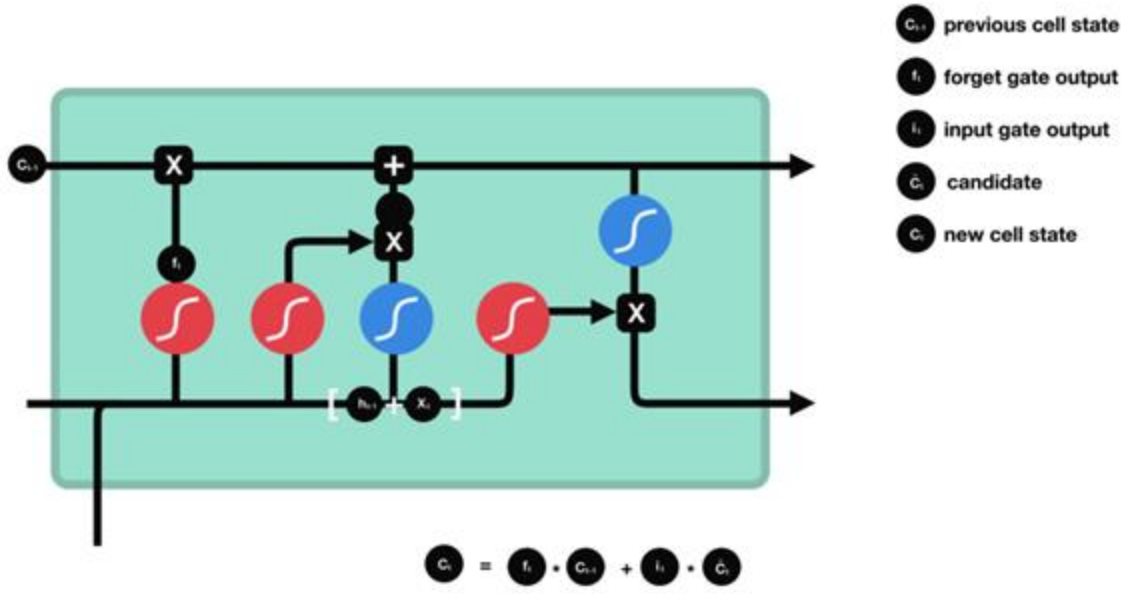


Quelle: https://blog.csdn.net/mary19831/article/details/129570030

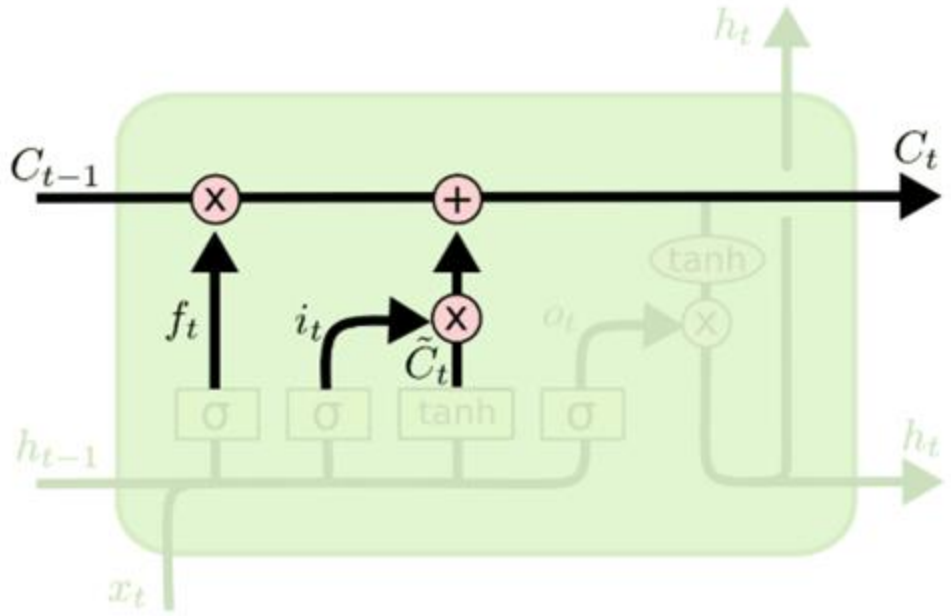
Quelle: https://blog.csdn.net/qq_39297053/article/details/136427425

Zellzustand

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



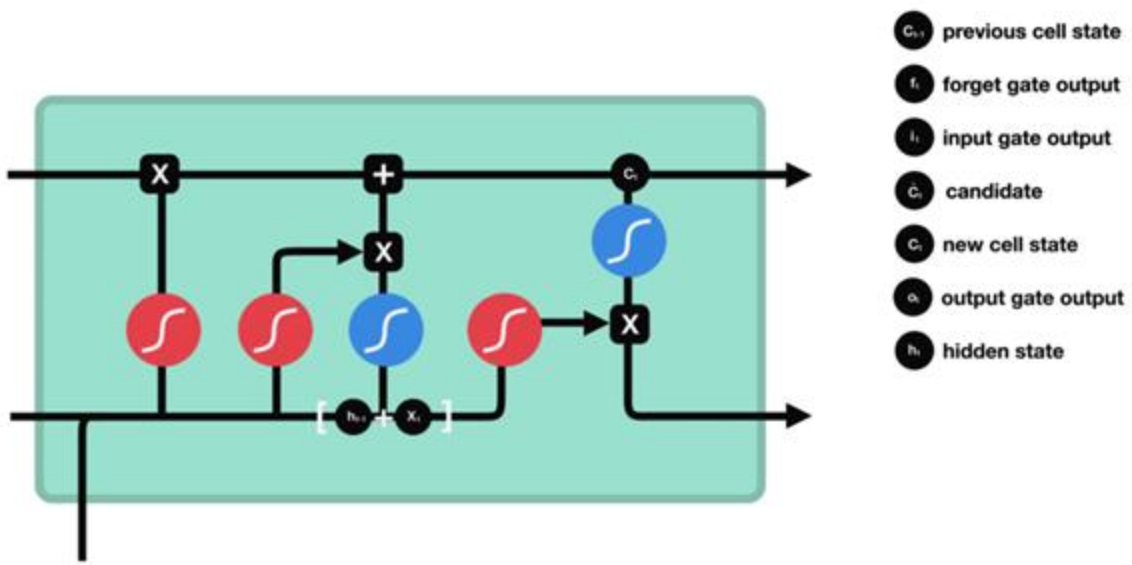
Quelle: https://blog.csdn.net/mary19831/article/details/129570030



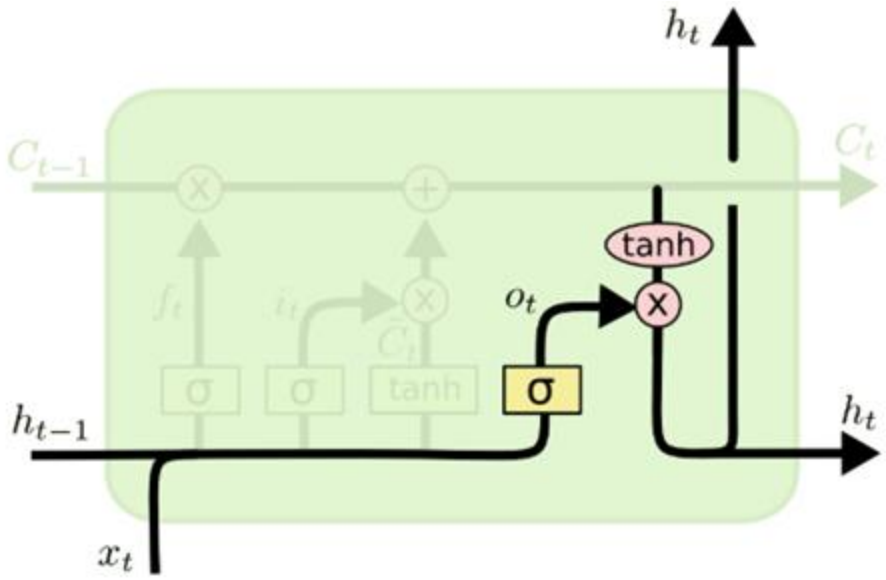
Quelle: https://blog.csdn.net/qq_39297053/article/details/136427425

Ausgabe-Gate

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



Quelle: https://blog.csdn.net/mary19831/article/details/129570030



Quelle: https://blog.csdn.net/qq_39297053/article/details/136427425

GRU

Eine vereinfachte Variante von LSTM

- Gated Recurrent Unit (GRU)
- 2 Gates: Reset und Update
- Schnelleres Training
- Kombiniert Langzeit- und Kurzzeitspeicher
- Ressourcenknappe oder kleinere Aufgaben

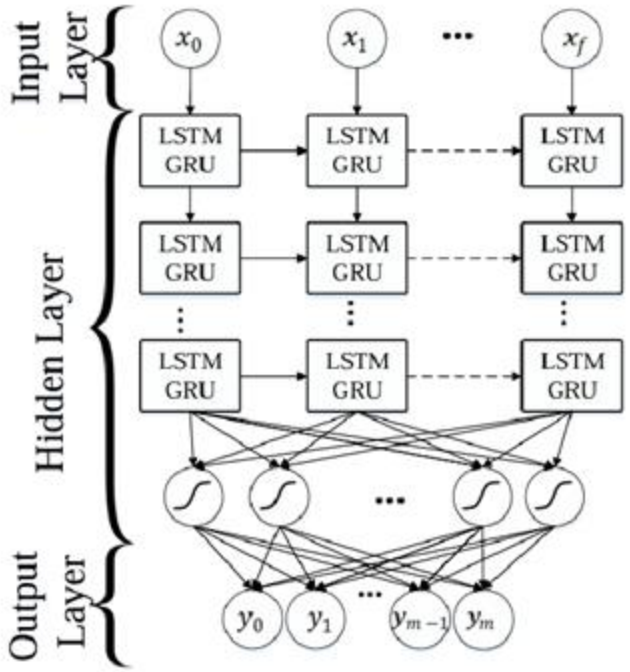
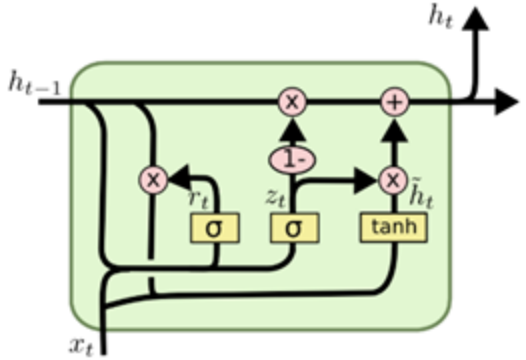


Figure 17. Standard long short-term memory (LSTM)/GRU recurrent neural networks.

Quelle: Kowsari, K., Meimandi, K. J., Heidarysafa, M., Mendu, S., Barnes, L., & Brown, D. (2020). Text Classification Algorithms: A Survey.



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

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

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

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

Quelle: dprogrammer.org/rnn-lstm-gru

Vision LSTM

- Eine Variante des klassischen LSTM-Modells für visuelle Daten
- Vision LSTM nutzt oft CNNs zur Extraktion von räumlichen Markmale, um räumlich-zeitliche Muster in visuellen Daten zu verarbeiten
- Gut geeignet für Anwendungen mit mittleren Sequenzen wie kurze Videoanalysen

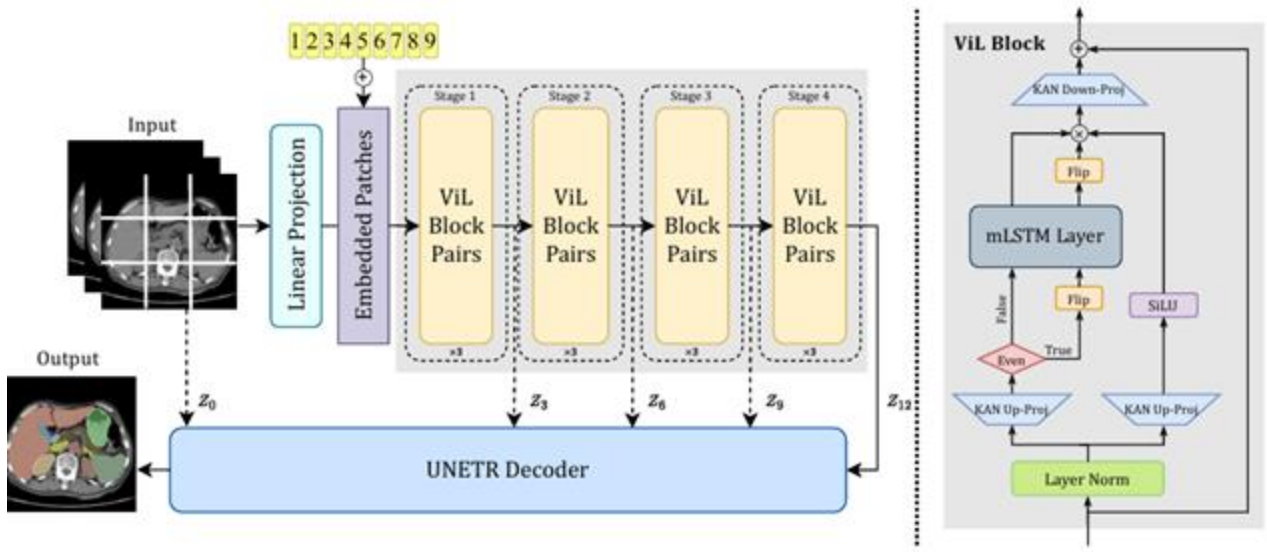


Fig. 1. Left: Overall architecture of UNETVL; Right: The internal structure of the ViL blocks.

Quelle: Guo, X., Dam, T., Dhamdhere, R., Modanwal, G., & Madabhushi, A. (2025). UNETVL: Enhancing 3D medical image segmentation with Chebyshev kernel powered Vision-LSTM. *Electrical Engineering and Computer Science Department, Michigan University*.

CNN-LSTM

Input layer
Conv1D
MaxPooling1D
LSTM
Dropout
Conv1D
MaxPooling1D
LSTM
Dropout
Conv1D
MaxPooling1D
LSTM
Dense

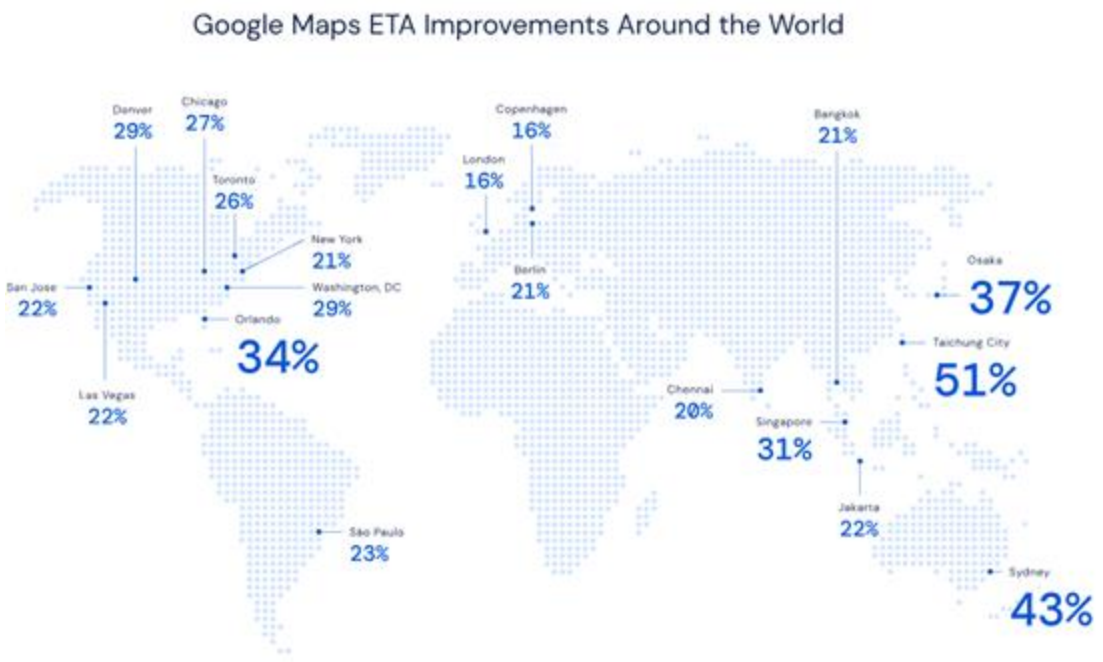
Quelle: indianexpress

Z.-x. Hu, B. Shen, Y. Hu, and C. Zhao, "Research on stock price forecast of general electric based on mixed CNN-LSTM model," *School of Computer Science, Hubei University*, Wuhan, China, Jan. 15, 2025.

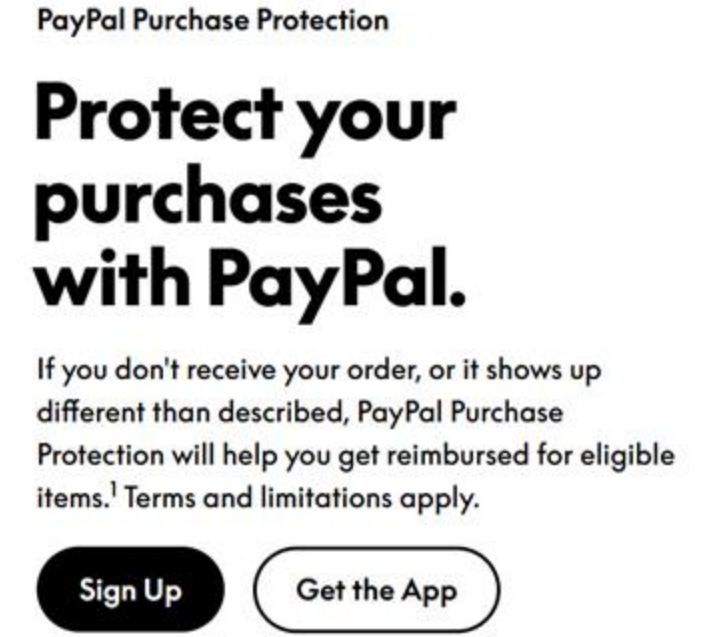
Vorteile & Anwendungen

Im Vergleich zu klassischen RNNs

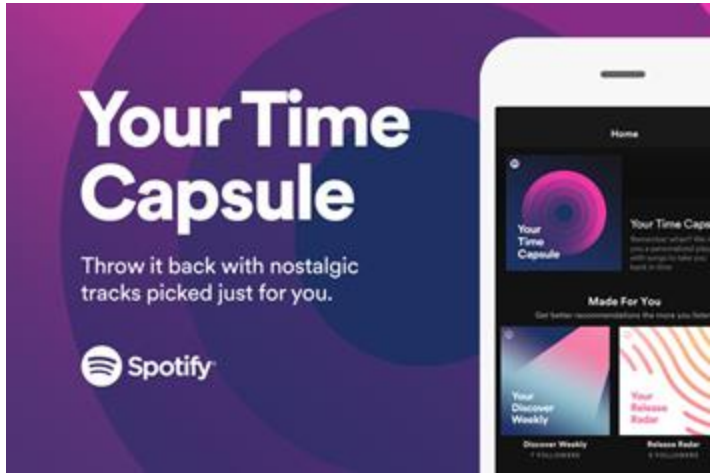
- Lösung des Vanishing Gradient Problems
- Modellierung von Langzeit- und Kurzzeitabhängigkeiten
- Kontrollierter Informationsfluss
- Verarbeitung längere Sequenzen



Quelle: reddit



Quelle: paypal



Quelle: spotify

Schwäche

- Schwierig bei langen Abhängigkeiten
- Begrenzte Parallelisierung
- Hoher Rechenaufwand
- Begrenzte Skalierbarkeit
- Langsamere Verarbeitung in Echtzeit-Anwendungen
- Abgelöst durch Transformer-Modelle

Quelle: apple

Quelle: replika

Bi-LSTM
(Bidirectional Long Short-Term Memory)

Was ist Bi-LSTM?

- Erweiterung des LSTM-Netzwerks
- Verarbeitet Daten in beide Richtungen (vorwärts & rückwärts)
- Erfasst vergangenen und zukünftigen Kontext

Wie funktioniert es?

Vorwärts

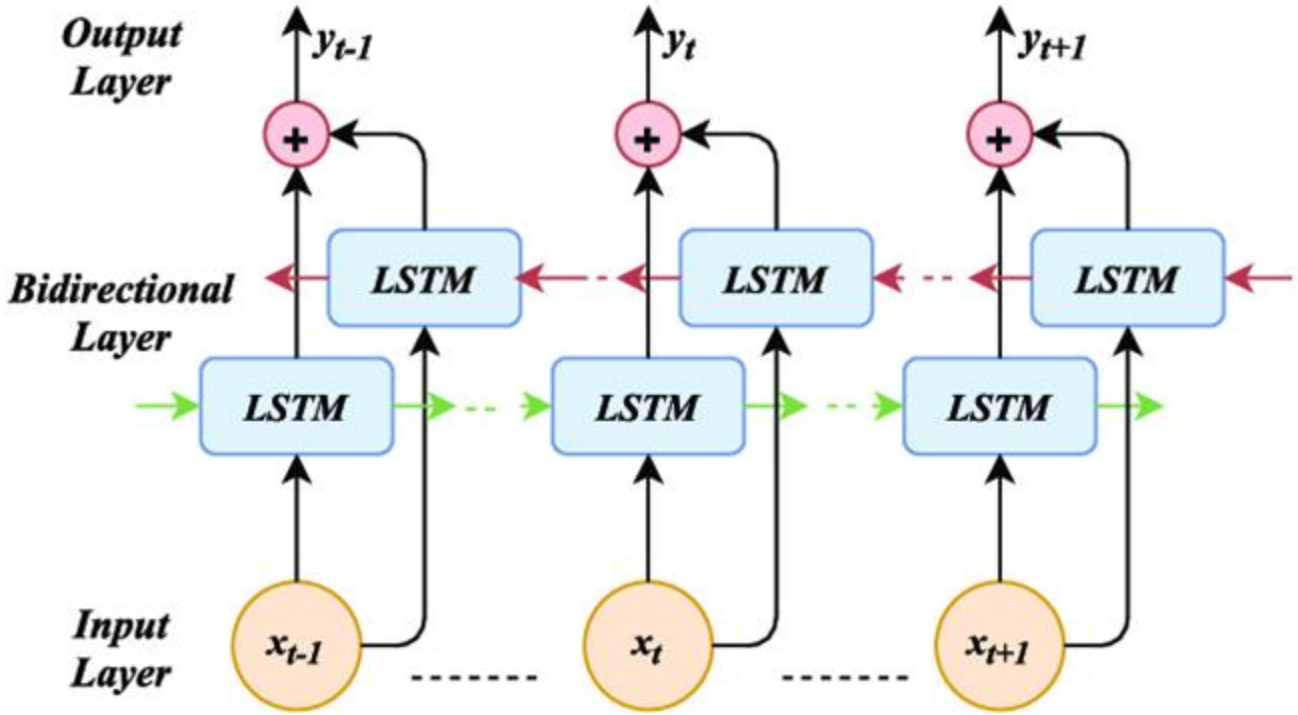
- Eingabe wird von der ersten bis zur letzten Zeiteinheit verarbeitet
- Berechnung des versteckten Zustands und Aktualisierung der Speicherzelle

Rückwärts

- Eingabe wird rückwärts verarbeitet
- Berechnung und Aktualisierung wie bei Vorwärts

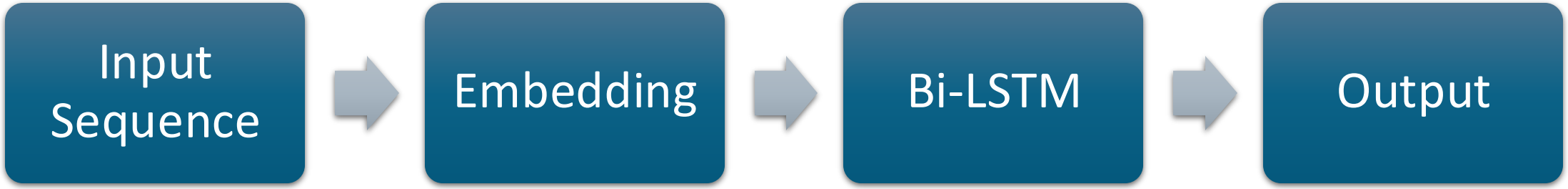
Kombination der Zustände

- Versteckte Zustände von Vorwärts- und Rückwärts-LSTM werden kombiniert



Quelle: https://www.researchgate.net/figure/Bidirectional-LSTM-model-showing-the-input-and-output-layers-The-red-arrows-represent_fig3_344554659

Architektur



$$p_t = p_t^f + p_t^b$$

wobei:

- p_t: Der finale Wahrscheinlichkeitsvektor des Netzwerks.
- p_t^f: Der Wahrscheinlichkeitsvektor des vorwärtsgerichteten LSTM-Netzwerks.
- p_t^b: Der Wahrscheinlichkeitsvektor des rückwärtsgerichteten LSTM-Netzwerks.

Erweiterungen der Bi-LSTM-Architektur

- **Zusätzliche Schichten:** Dense Layers erhöhen die Modellkapazität
- **Aufmerksamkeitsmechanismen (Attention):** Hervorhebung wichtiger Sequenzteile für bessere Leistung
- **Vorteil:** Mehr Flexibilität und höhere Leistungsfähigkeit für komplexe Aufgaben

Bi-LSTM Implementierung mit PyTorch

```python
import torch
import torch.nn as nn

# Define the custom Bi-LSTM model
class BiLSTMModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim=128, lstm_size=128, num_classes=3):
        super(BiLSTMModel, self).__init__()
        # Embedding layer
        self.embedding = nn.Embedding(num_embeddings=vocab_size, embedding_dim=embedding_dim)

        # Bi-LSTM layer
        self.lstm = nn.LSTM(input_size=embedding_dim, hidden_size=lstm_size,
                            num_layers=1, bidirectional=True, batch_first=True)

        # Fully connected output layer
        self.fc = nn.Linear(lstm_size * 2, num_classes)  # Multiply by 2 for bidirectional

    def forward(self, x):
        x = self.embedding(x)        # Embedding layer
        output, _ = self.lstm(x)     # Bi-LSTM layer
        x = output[:, -1, :]         # Use the last time step
        x = self.fc(x)               # Fully connected layer
        return x

# Hyperparameters
vocab_size = 5000
embedding_dim = 128
lstm_size = 128
num_classes = 3

# Initialize the model
model = BiLSTMModel(vocab_size, embedding_dim, lstm_size, num_classes)

# Print the model structure
print(model)
```

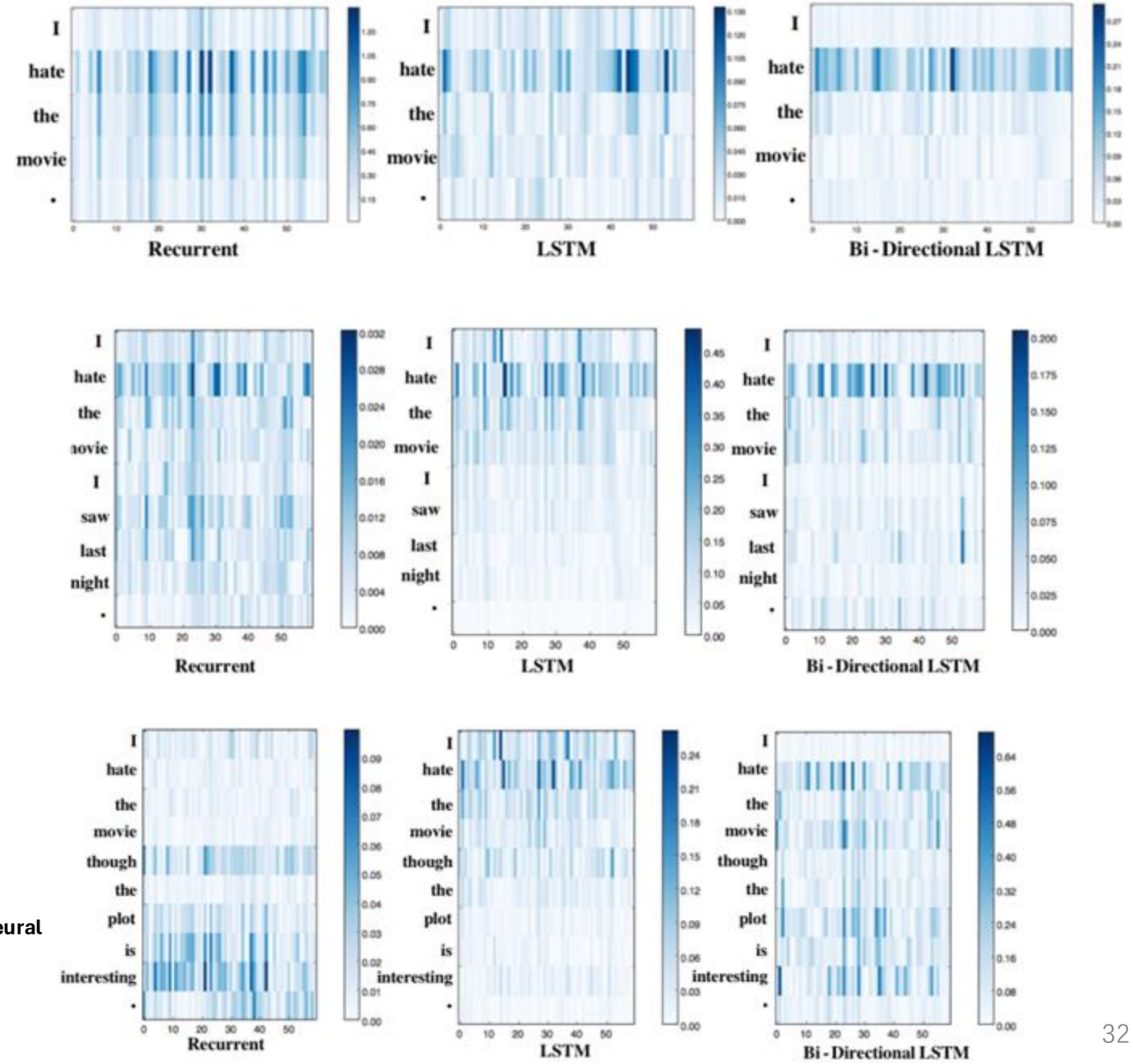
LSTM vs. Bi-LSTM

“ **Apple is something that …**”

- Apple is something that **competitors simply cannot reproduce**.
- Apple is something that **I like to eat**.

Recurrent vs. LSTM vs. Bi-LSTM

- I hate the movie
- I hate the movie **that I saw last night**
- I hate the movie **though the plot is interesting**



Related Work:

J. Li, X. Chen, E. Hovy & D. Jurafsky (2015). **Visualizing and Understanding Neural Models in NLP**. *arXiv:1506.01066*.

Bidirectional LSTM

- **Vorteile**
  - **Erfassung von Langzeitabhängigkeiten:** Verarbeitung der Sequenz in beide Richtungen für besseren Kontext.
  - **Verbesserte Leistung:** Höhere Genauigkeit bei NLP, Spracherkennung und Sentiment-Analyse.
  - **Flexible Architektur:** Erweiterbar mit Convolutional Layers und Attention-Mechanismen.

- **Nachteile**
  - **Hoher Rechenaufwand:** Erhöhte Komplexität und Ressourcenverbrauch.
  - **Hoher Datenbedarf:** Gefahr von Overfitting bei kleinen Datensätzen.
  - **Schwierige Interpretierbarkeit:** Begrenzte Nachvollziehbarkeit der Modellentscheidungen.

Modellinterpretierbarkeit – Captum
Sentiment Datensatz

Statisches Embedding/ Fine-tuning Embedding
Letter Datensatz

Das beste Ergebnis
News Datensatz

Captum

Captum (“comprehension” in Latin) ist eine Open-Source-Bibliothek für Modellinterpretierbarkeit, die auf PyTorch basiert.

Integrated Gradients bietet Forschern und Entwicklern eine einfache Möglichkeit zu verstehen, welche Merkmale zum Ergebnis eines Modells beitragen.

Model setup

Basierend auf den Texten im Trainingsdatensatz die Worthäufigkeit berechnen und ein Wörterbuch {Token -> Index} erstellen.

Max_len = 50
Embeding dim = 128
Hidden dim = 128
Batch size = 32
GPU

Masterpiece. Never has one movie captured the essence of the human spirit.

RNN

Prob(positive)=0.2049

- *Accuracy* schwankt ständig
- *Masterpiece* wurde vergessen
- Fokus liegt mehr auf den folgenden Wörtern
- Nur wenige Wörter berücksichtigt

```
Using device: cuda
Epoch 1/10, Loss=0.6936, Val Acc=0.5599
Epoch 2/10, Loss=0.6878, Val Acc=0.5257
Epoch 3/10, Loss=0.6748, Val Acc=0.5633
Epoch 4/10, Loss=0.6574, Val Acc=0.5652
Epoch 5/10, Loss=0.6129, Val Acc=0.6509
Epoch 6/10, Loss=0.5852, Val Acc=0.6198
Epoch 7/10, Loss=0.5922, Val Acc=0.6196
Epoch 8/10, Loss=0.5317, Val Acc=0.5579
Epoch 9/10, Loss=0.5471, Val Acc=0.6412
Epoch 10/10, Loss=0.5267, Val Acc=0.5658
```

LSTM

Prob(positive)=0.9133

Masterpiece wird mit stärkerem Gewicht versehen.
Never wird rot markiert. - negativ

```
Using device: cuda
Epoch 1/5, Loss=0.6646, Val Acc=0.6998
Epoch 2/5, Loss=0.5051, Val Acc=0.7663
Epoch 3/5, Loss=0.3805, Val Acc=0.7878
Epoch 4/5, Loss=0.2875, Val Acc=0.7858
Epoch 5/5, Loss=0.2056, Val Acc=0.7865
```

masterpiece never has one movie captured the essence of the human spirit

Bi-LSTM

Prob(positive)=0.9990

```
Using device: cuda
Epoch 1/5, Loss=0.5886, Val Acc=0.7512
Epoch 2/5, Loss=0.4290, Val Acc=0.7718
Epoch 3/5, Loss=0.3189, Val Acc=0.7870
Epoch 4/5, Loss=0.2108, Val Acc=0.7833
Epoch 5/5, Loss=0.1107, Val Acc=0.7797
```

- Die Abdeckung der Attribution ist breiter
- „Never“ wird grün markiert - positiv

masterpiece never has one movie captured the essence of the human spirit

Statisches Embedding/ Fine-tuning Embedding Letter Dataset

Die Auswirkungen verschiedener Embeddingsmethoden analysieren.

1. Datenverarbeitung

TF-IDF, Word2Vec, GloVe, S-BERT

2. Verwendete Modelle:

SimpleRNN, LSTM, BiLSTM

3. Ergebnisse

- Bei der Verwendung von statischen Embedding erreichte **TF-IDF** die beste Genauigkeit.
- Mit dynamischen Embedding **verbesserten sich die Ergebnisse** in Word2Vec, Glove und S-BERT.

Überlegung: Unterschiede zwischen dynamischen und statischen Embeddings.

Statisches Embedding

Model setup

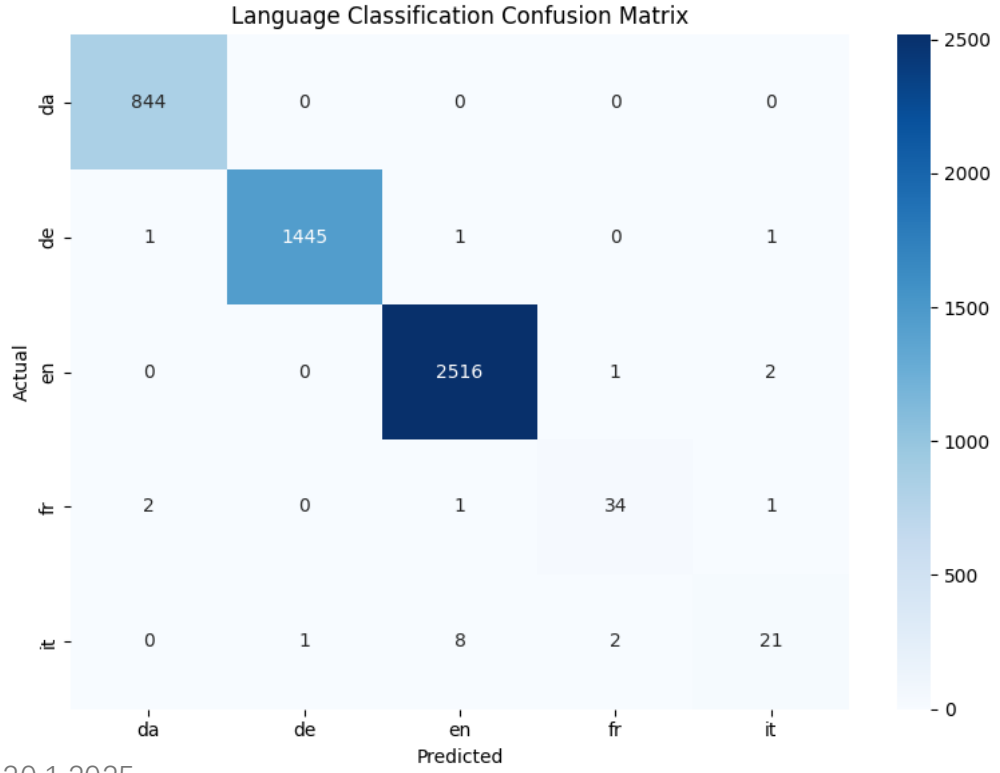
CPU
Epoch = 20
Hidden_dim = 128
Dropout = 0.5
Batch size = 64
Ctoss Entropyloss
Optimizer: Adam

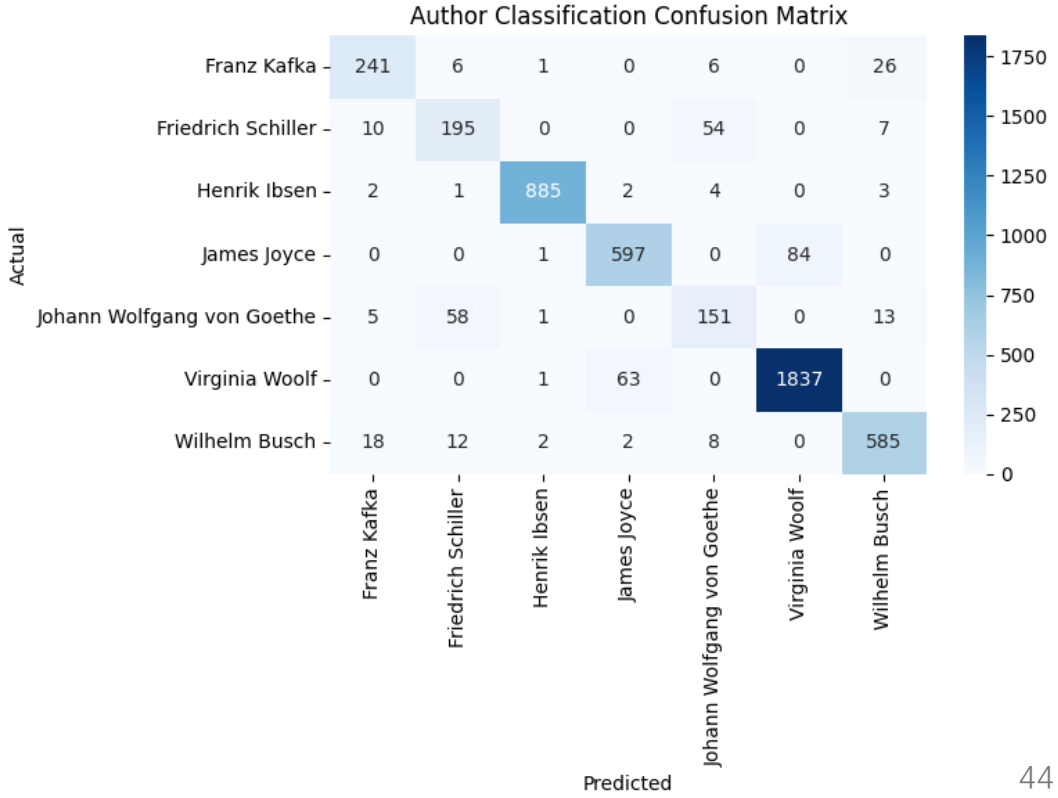
TF-IDF

from sklearn.feature_extraction.text import TfidfVectorizer
max_features = 5000

Input: Ein 5000-dimensionaler Feature-Vektor eines Textes

```
Language Classification Metrics:
Accuracy: 0.9957
F1 Score: 0.9955

Author Classification Metrics:
Accuracy: 0.9201
F1 Score: 0.9200
```





GloVe

- glove.6B.200d.txt
- Glove embeddings with 6 Billion words with each word being a 200 dimensional vector.

Statisches Embedding

```python
def sentence_to_glove(sentence, embeddings_index, embedding_dim=200):
    words = sentence.split()
    valid_embeddings = [embeddings_index[word] for word in words if word in embeddings_index]
    if len(valid_embeddings) > 0:
        return np.mean(valid_embeddings, axis=0)
    else:
        return np.zeros(embedding_dim)
```

Jeder Satz -> den Mittelwert der GloVe-Vektoren
Der Satz wird direkt als Vektor mit fester Dimension dargestellt.

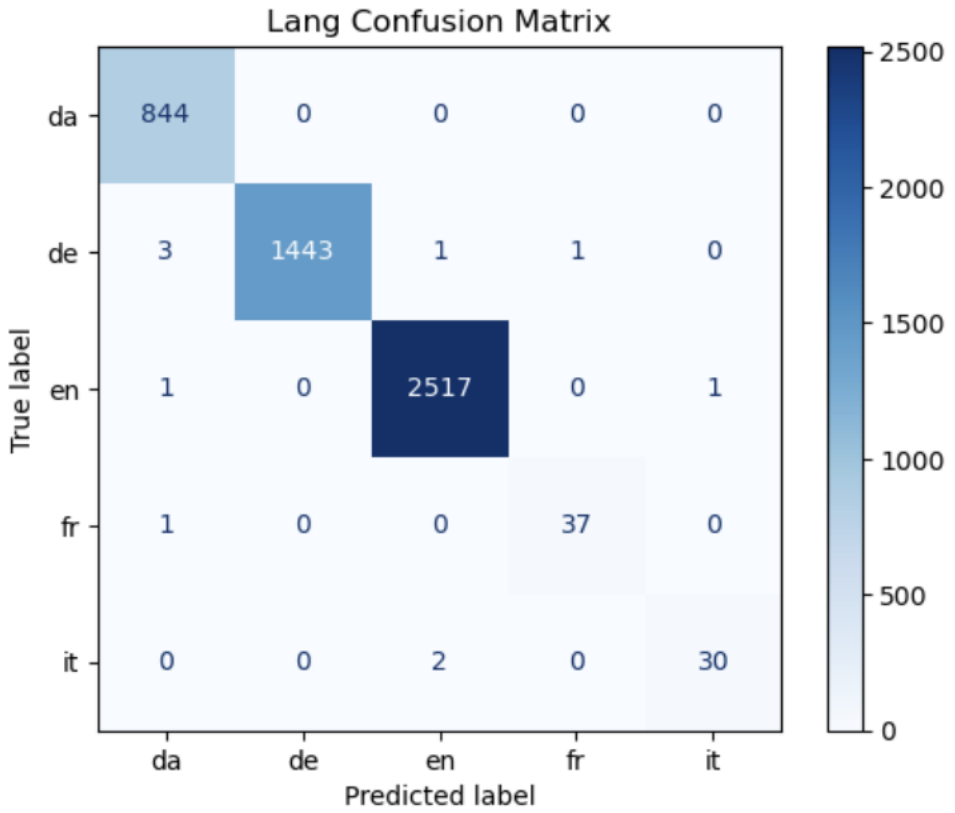
RNN

```
===== Author Classification Report =====
                          precision    recall  f1-score   support

           Wilhelm Busch       0.62      0.60      0.61       280
           Henrik Ibsen        0.59      0.53      0.56       266
             James Joyce       0.99      0.97      0.98       897
             Franz Kafka       0.81      0.81      0.81       682
       Friedrich Schiller      0.56      0.43      0.49       228
Johann Wolfgang von Goethe     0.93      0.93      0.93      1901
          Virginia Woolf       0.76      0.88      0.81       627

                accuracy                           0.85      4881
               macro avg       0.75      0.74      0.74      4881
            weighted avg       0.85      0.85      0.85      4881
```

```
===== Lang Classification Report =====
              precision    recall  f1-score   support

          da       0.99      1.00      1.00       844
          de       1.00      1.00      1.00      1448
          en       1.00      1.00      1.00      2519
          fr       0.97      0.97      0.97        38
          it       0.97      0.94      0.95        32

    accuracy                           1.00      4881
   macro avg       0.99      0.98      0.98      4881
weighted avg       1.00      1.00      1.00      4881
```

RNN


Author Confusion Matrix
Franz Kafka
Friedrich Schiller
Henrik Ibsen
James Joyce
Johann Wolfgang von Goethe
Virginia Woolf
Wilhelm Busch
True label
Predicted label
168 25 0 0 10 0 77
39 140 3 0 41 1 42
6 5 872 5 2 0 7
0 0 0 550 0 132 0
25 52 0 0 98 0 53
0 0 0 127 0 1774 0
34 14 2 1 23 0 553
1600
1400
1200
1000
800
600
400
200
0


Lang Confusion Matrix
da
de
en
fr
it
True label
Predicted label
844 0 0 0 0
3 1443 1 1 0
1 0 2517 0 1
1 0 0 37 0
0 0 2 0 30
2500
2000
1500
1000
500
0

LSTM

```
===== Author Classification Report =====
                          precision    recall  f1-score   support

           Wilhelm Busch       0.64      0.60      0.62       280
            Henrik Ibsen       0.53      0.66      0.59       266
             James Joyce       0.99      0.97      0.98       897
             Franz Kafka       0.87      0.74      0.80       682
       Friedrich Schiller       0.61      0.35      0.44       228
Johann Wolfgang von Goethe       0.91      0.96      0.94      1901
          Virginia Woolf       0.79      0.86      0.82       627

                accuracy                           0.85      4881
               macro avg       0.76      0.73      0.74      4881
            weighted avg       0.85      0.85      0.85      4881
```

```
===== Lang Classification Report =====
              precision    recall  f1-score   support

          da       1.00      1.00      1.00       844
          de       1.00      1.00      1.00      1448
          en       1.00      1.00      1.00      2519
          fr       0.93      1.00      0.96        38
          it       0.97      0.97      0.97        32

    accuracy                           1.00      4881
   macro avg       0.98      0.99      0.99      4881
weighted avg       1.00      1.00      1.00      4881
```

Bi-LSTM

```
===== Author Classification Report =====
                          precision    recall  f1-score   support

           Wilhelm Busch       0.63      0.63      0.63       280
            Henrik Ibsen       0.55      0.68      0.61       266
             James Joyce       1.00      0.97      0.98       897
             Franz Kafka       0.84      0.77      0.81       682
       Friedrich Schiller       0.59      0.38      0.46       228
Johann Wolfgang von Goethe       0.92      0.95      0.94      1901
          Virginia Woolf       0.81      0.86      0.83       627

                accuracy                           0.86      4881
               macro avg       0.76      0.75      0.75      4881
            weighted avg       0.86      0.86      0.86      4881
```

```
===== Lang Classification Report =====
              precision    recall  f1-score   support

          da       1.00      1.00      1.00       844
          de       1.00      1.00      1.00      1448
          en       1.00      1.00      1.00      2519
          fr       0.95      0.97      0.96        38
          it       0.97      0.97      0.97        32

    accuracy                           1.00      4881
   macro avg       0.98      0.99      0.99      4881
weighted avg       1.00      1.00      1.00      4881
```

Fine-tuning Embedding

```
embedding_matrix = np.zeros((vocab_size, embed_dim), dtype=np.float32)

for word, idx in token2idx.items():
    if word in glove_dict:
        embedding_matrix[idx] = glove_dict[word]
    else:
        embedding_matrix[idx] = np.random.normal(scale=0.6, size=(embed_dim,))
```

Max_length = 50

Für den Datensatz ein Wörterbuch erstellt (*token2idx*)

Für jedes Wort im Wörterbuch wurde, falls es in GloVe gefunden wurde, der entsprechende Vektor verwendet

Model setup

Batchsize = 32
hidden_dim = 128
Dropout = 0.5
lr = 1e-3
epochs = 5
optimizer = Adam
criterion = nn.CrossEntropyLoss()

```python
class RNNClassifier(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim, author_classes, lang_classes, embedding_matrix=None):
        super(RNNClassifier, self).__init__()

        # 1) embedding layer
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        if embedding_matrix is not None:
            self.embedding.weight.data.copy_(torch.from_numpy(embedding_matrix))

        # 2) Simple RNN
        self.rnn = nn.RNN(input_size=embed_dim, hidden_size=hidden_dim, batch_first=True)

        # 3) dropout
        self.dropout = nn.Dropout(0.5)

        # 4) fc
        self.author_fc = nn.Linear(hidden_dim, author_classes)
        self.lang_fc   = nn.Linear(hidden_dim, lang_classes)

    def forward(self, x):
        """
        x_shape: (batch_size, seq_len)
        """
        # embedding => (batch_size, seq_len, embed_dim)
        embedded = self.embedding(x)

        # RNN => output: (batch_size, seq_len, hidden_dim), hidden: (num_layers, batch_size, hidden_dim)
        output, hidden = self.rnn(embedded)

        # hidden[-1] get last layer => (batch_size, hidden_dim)
        hidden = self.dropout(hidden[-1])

        # classification
        author_output = self.author_fc(hidden)
        lang_output   = self.lang_fc(hidden)

        return author_output, lang_output
```

RNN

===== Author Classification Report =====

	precision	recall	f1-score	support
Franz Kafka	0.28	0.25	0.27	280
Friedrich Schiller	0.33	0.02	0.03	266
Henrik Ibsen	0.70	0.57	0.62	897
James Joyce	0.00	0.00	0.00	682
Johann Wolfgang von Goethe	0.16	0.07	0.09	228
Virginia Woolf	0.61	0.91	0.73	1901
Wilhelm Busch	0.35	0.53	0.42	627
accuracy			0.55	4881
macro avg	0.35	0.34	0.31	4881
weighted avg	0.45	0.55	0.48	4881

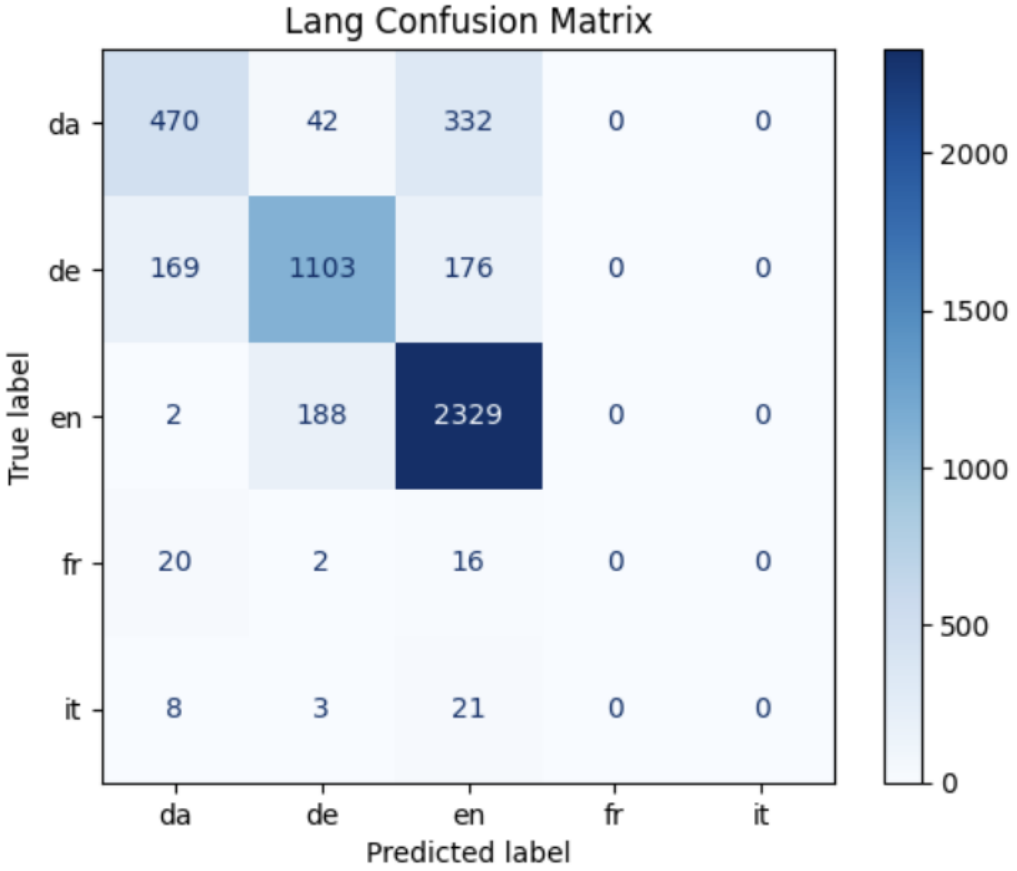
===== Lang Classification Report =====

	precision	recall	f1-score	support
da	0.70	0.56	0.62	844
de	0.82	0.76	0.79	1448
en	0.81	0.92	0.86	2519
fr	0.00	0.00	0.00	38
it	0.00	0.00	0.00	32
accuracy			0.80	4881
macro avg	0.47	0.45	0.46	4881
weighted avg	0.78	0.80	0.79	4881

RNN zeigt schlechte Ergebnisse nach 5 Epochen.

RNN


Author Confusion Matrix
Franz Kafka
Friedrich Schiller
Henrik Ibsen
James Joyce
Johann Wolfgang von Goethe
Virginia Woolf
Wilhelm Busch
True label
Predicted label
71 3 38 0 29 11 128
71 4 25 0 22 21 123
6 0 508 0 2 329 52
2 0 29 0 0 619 32
53 4 24 0 15 8 124
0 0 3 0 0 1739 159
49 1 102 0 25 115 335
1600
1400
1200
1000
800
600
400
200
0


Lang Confusion Matrix
da
de
en
fr
it
True label
Predicted label
470 42 332 0 0
169 1103 176 0 0
2 188 2329 0 0
20 2 16 0 0
8 3 21 0 0
2000
1500
1000
500
0

LSTM und Bi-LSTM zeigen im Vergleich zu früheren Modellen eine deutlich bessere Leistung.

LSTM

```
===== Author Classification Report =====
                          precision    recall  f1-score   support

              Franz Kafka       0.83      0.89      0.86       280
        Friedrich Schiller       0.68      0.74      0.71       266
             Henrik Ibsen       0.99      0.98      0.99       897
             James Joyce        0.93      0.95      0.94       682
Johann Wolfgang von Goethe       0.62      0.57      0.59       228
           Virginia Woolf       0.98      0.98      0.98      1901
            Wilhelm Busch       0.96      0.93      0.94       627

                 accuracy                           0.93      4881
                macro avg       0.86      0.86      0.86      4881
             weighted avg       0.93      0.93      0.93      4881
```

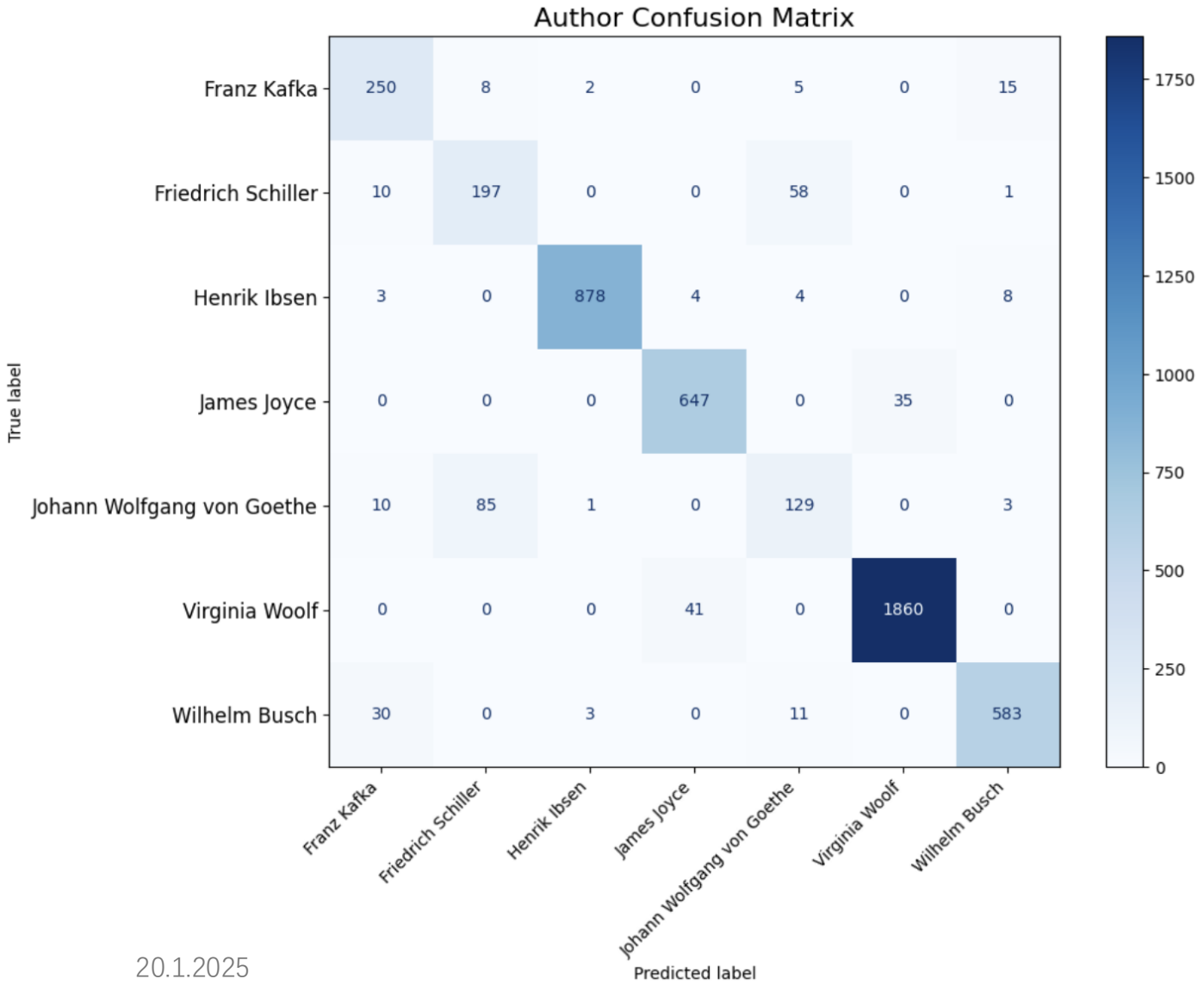
```
===== Lang Classification Report =====
              precision    recall  f1-score   support

          da       1.00      1.00      1.00       844
          de       1.00      1.00      1.00      1448
          en       1.00      1.00      1.00      2519
          fr       0.93      1.00      0.96        38
          it       0.97      0.91      0.94        32

    accuracy                           1.00      4881
   macro avg       0.98      0.98      0.98      4881
weighted avg       1.00      1.00      1.00      4881
```

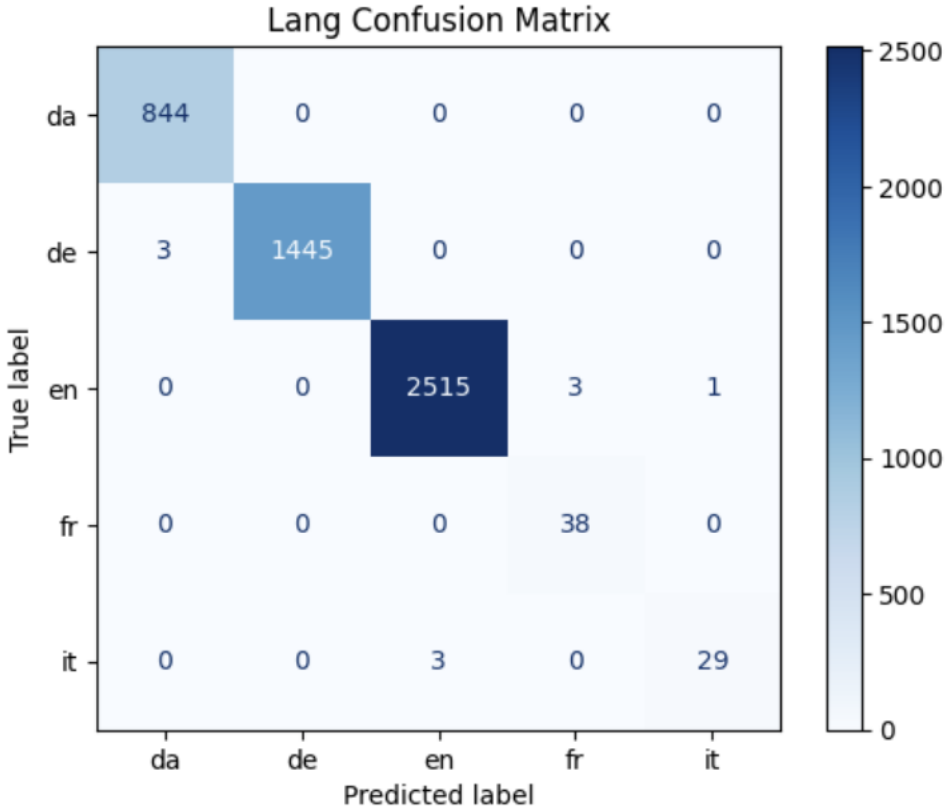
Bi-LSTM

```
===== Author Classification Report =====
                          precision    recall  f1-score   support

              Franz Kafka       0.91      0.87      0.89       280
        Friedrich Schiller       0.83      0.55      0.66       266
             Henrik Ibsen       0.96      0.99      0.97       897
             James Joyce        0.96      0.92      0.94       682
Johann Wolfgang von Goethe       0.58      0.81      0.67       228
           Virginia Woolf       0.97      0.99      0.98      1901
            Wilhelm Busch       0.95      0.93      0.94       627

                 accuracy                           0.93      4881
                macro avg       0.88      0.86      0.86      4881
             weighted avg       0.94      0.93      0.93      4881
```

```
===== Lang Classification Report =====
              precision    recall  f1-score   support

          da       1.00      1.00      1.00       844
          de       1.00      1.00      1.00      1448
          en       1.00      1.00      1.00      2519
          fr       0.95      0.97      0.96        38
          it       0.96      0.84      0.90        32

    accuracy                           1.00      4881
   macro avg       0.98      0.96      0.97      4881
weighted avg       1.00      1.00      1.00      4881
```

LSTM


Author Confusion Matrix
Franz Kafka
Friedrich Schiller
Henrik Ibsen
James Joyce
Johann Wolfgang von Goethe
Virginia Woolf
Wilhelm Busch
True label
Predicted label
250 8 2 0 5 0 15
10 197 0 0 58 0 1
3 0 878 4 4 0 8
0 0 0 647 0 35 0
10 85 1 0 129 0 3
0 0 0 41 0 1860 0
30 0 3 0 11 0 583
1750
1500
1250
1000
750
500
250
0


Lang Confusion Matrix
da
de
en
fr
it
True label
Predicted label
844 0 0 0 0
3 1445 0 0 0
0 0 2515 3 1
0 0 0 38 0
0 0 3 0 29
2500
2000
1500
1000
500
0

Statisches Embedding

• Jeder Satz wird durch GloVe in einen 200-dimensionalen Vektor komprimiert.
• Daher beträgt der Wert von *seq_len* nur 1.

Fine-tuning Embedding

• Ein Wörterbuch wird erstellt, und die GloVe-Vektoren der Tokens werden als Input ins Modell eingespeist.
• Bei der Erstellung des Wörterbuchs wird die Textlänge auf 50 begrenzt.
• Daher beträgt der Wert von seq_len 50.

Modellleistung

- Mit Embedding muss das Modell mehr Daten verarbeiten als ohne Embedding.
- Dynamische Embedding-Methoden laufen auf der CPU weniger effizient, was zu einer langsameren Ausführung führt.
- Schwächen von RNN wurden deutlich:

-> Informationen am Anfang werden leicht vergessen.

-> Langsame Konvergenzgeschwindigkeit.

-> Hohe Schwankungen des Verlustfunktionswerts.

News dataset

Sbert +RNN model architecture

Sbert

sentence-transformers/paraphrase-MiniLM-L6-v2

Headline+short description

Komprimieren die Semantik jedes Satzes auf den [CLS]-Token

```python
class FineTuneSBERTBase(nn.Module):
    def __init__(self, sbert_name, hidden_dim, output_dim):
        super(FineTuneSBERTBase, self).__init__()
        self.sbert = AutoModel.from_pretrained(sbert_name)
        self.hidden_dim = self.sbert.config.hidden_size
        self.fc = nn.Linear(self.hidden_dim, hidden_dim)
        self.out = nn.Linear(hidden_dim, output_dim)

    def extract_features(self, input_ids, attention_mask):
        outputs = self.sbert(input_ids=input_ids, attention_mask=attention_mask)
        cls_embedding = outputs.last_hidden_state[:, 0, :]
        return cls_embedding
```

Model Setup

epochs=20
Hidden_dim = 128
lr=1e-5
patience=3
Cross-Entropy loss
Adam optimizer

```python
class RNNFineTuneSBERT(FineTuneSBERTBase):
    def __init__(self, sbert_name, hidden_dim, output_dim):
        super(RNNFineTuneSBERT, self).__init__(sbert_name, hidden_dim, output_dim)
        self.rnn = nn.RNN(self.hidden_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, input_ids, attention_mask):
        # [CLS] embedding
        cls_embedding = self.extract_features(input_ids, attention_mask)

        cls_embedding = cls_embedding.unsqueeze(1)  # (batch_size, 1, hidden_dim)

        _, hidden = self.rnn(cls_embedding)  # (1, batch_size, hidden_dim)

        output = self.fc(hidden[-1])  # (batch_size, output_dim)
        return output
```

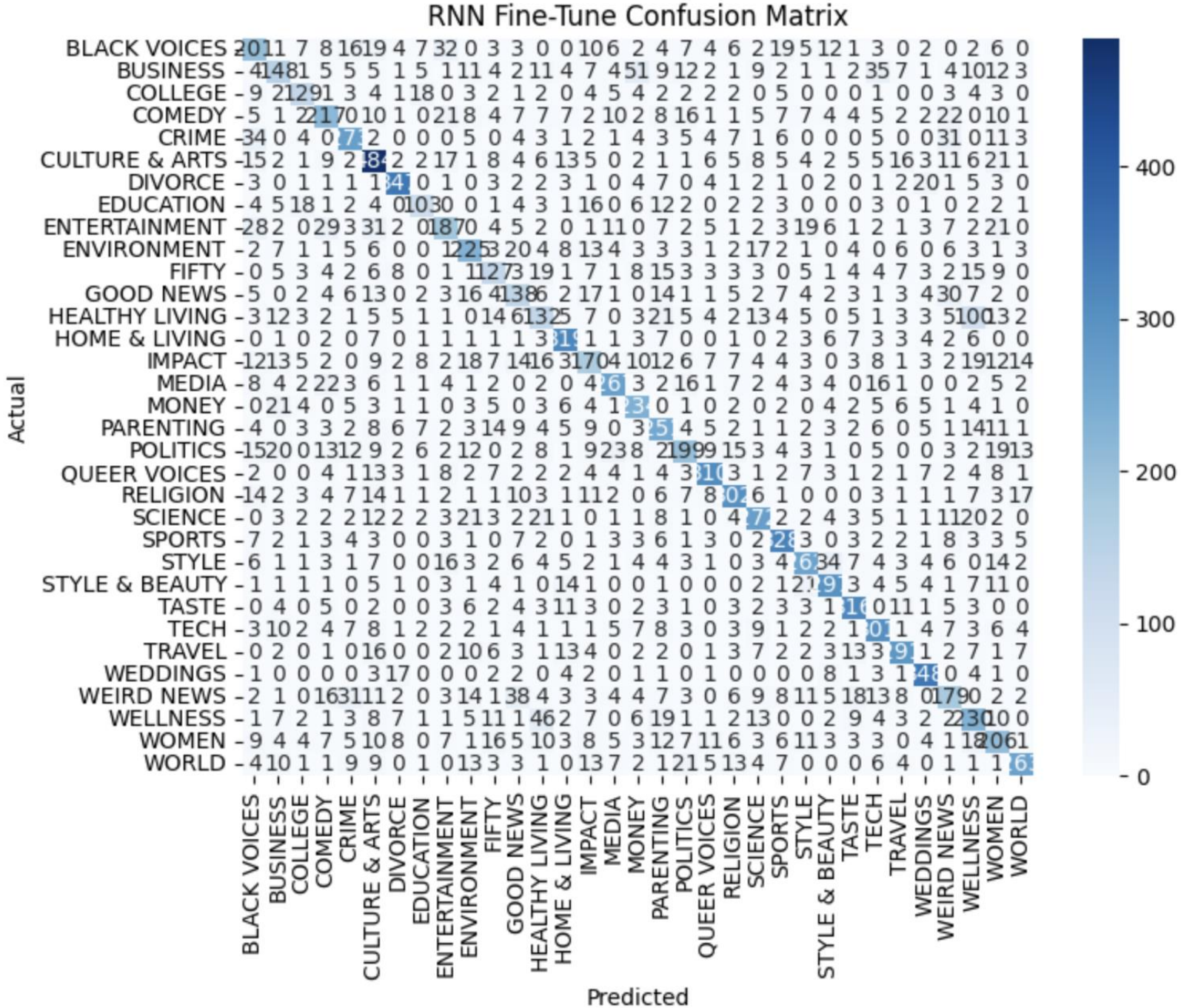
- **Lernraten-Scheduler:** Die Lernrate wird basierend auf dem Validierungsverlust dynamisch angepasst.
- **Early Stopping:** Drei Toleranzen, danach wird das Training gestoppt, wenn der Verlust den besten Verlust übersteigt.

```
scheduler.step(avg_val_loss)


if avg_val_loss < best_val_loss:
    best_val_loss = avg_val_loss
    patience_counter = 0
    torch.save(model.state_dict(), "best_model.pth")
    print("Best model saved.")
else:
    patience_counter += 1
    if patience_counter >= patience:
        print("Early stopping triggered.")
        break
```

```
Training RNN Fine-Tune Model
Epoch 1/20, Training Loss: 2.2417
Epoch 1/20, Validation Loss: 1.7172
Best model saved.
Epoch 2/20, Training Loss: 1.6175
Epoch 2/20, Validation Loss: 1.5277
Best model saved.
Epoch 3/20, Training Loss: 1.4133
Epoch 3/20, Validation Loss: 1.4472
Best model saved.
Epoch 4/20, Training Loss: 1.2809
Epoch 4/20, Validation Loss: 1.4094
Best model saved.
Epoch 5/20, Training Loss: 1.1713
Epoch 5/20, Validation Loss: 1.4016
Best model saved.
Epoch 6/20, Training Loss: 1.0781
Epoch 6/20, Validation Loss: 1.3873
Best model saved.
Epoch 7/20, Training Loss: 0.9945
Epoch 7/20, Validation Loss: 1.3862
Best model saved.
Epoch 8/20, Training Loss: 0.9146
Epoch 8/20, Validation Loss: 1.4136
Epoch 9/20, Training Loss: 0.8437
Epoch 9/20, Validation Loss: 1.4363
Epoch 10/20, Training Loss: 0.7781
Epoch 10/20, Validation Loss: 1.4453
Early stopping triggered.
RNN Fine-Tune Test Accuracy: 0.6282
```

RNN Fine-Tune Classification Report:

	precision	recall	f1-score	support
BLACK VOICES	0.50	0.51	0.51	392
BUSINESS	0.51	0.39	0.44	380
COLLEGE	0.64	0.61	0.62	212
COMEDY	0.58	0.54	0.56	399
CRIME	0.66	0.67	0.67	409
CULTURE & ARTS	0.64	0.72	0.68	673
DIVORCE	0.82	0.83	0.82	419
EDUCATION	0.61	0.52	0.56	198
ENTERTAINMENT	0.57	0.48	0.52	387
ENVIRONMENT	0.58	0.63	0.61	355
FIFTY	0.48	0.47	0.47	273
GOOD NEWS	0.44	0.45	0.45	305
HEALTHY LIVING	0.40	0.34	0.37	386
HOME & LIVING	0.73	0.83	0.77	386
IMPACT	0.49	0.42	0.45	400
MEDIA	0.72	0.68	0.70	395
MONEY	0.61	0.72	0.66	324
PARENTING	0.54	0.64	0.59	391
POLITICS	0.59	0.47	0.53	420
QUEER VOICES	0.78	0.75	0.76	415
RELIGION	0.72	0.69	0.70	440
SCIENCE	0.67	0.66	0.66	414
SPORTS	0.74	0.80	0.77	410
STYLE	0.67	0.63	0.65	413
STYLE & BEAUTY	0.72	0.76	0.74	391
TASTE	0.75	0.80	0.77	397
TECH	0.65	0.72	0.69	416
TRAVEL	0.75	0.72	0.73	405
WEDDINGS	0.80	0.87	0.83	400
WEIRD NEWS	0.50	0.44	0.47	408
WELLNESS	0.45	0.57	0.50	407
WOMEN	0.48	0.52	0.50	400
WORLD	0.76	0.65	0.70	404
accuracy			0.63	12824
macro avg	0.62	0.62	0.62	12824
weighted avg	0.63	0.63	0.63	12824


RNN Fine-Tune Confusion Matrix
Actual
Predicted
BLACK VOICES
BUSINESS
COLLEGE
COMEDY
CRIME
CULTURE & ARTS
DIVORCE
EDUCATION
ENTERTAINMENT
ENVIRONMENT
FIFTY
GOOD NEWS
HEALTHY LIVING
HOME & LIVING
IMPACT
MEDIA
MONEY
PARENTING
POLITICS
QUEER VOICES
RELIGION
SCIENCE
SPORTS
STYLE
STYLE & BEAUTY
TASTE
TECH
TRAVEL
WEDDINGS
WEIRD NEWS
WELLNESS
WOMEN
WORLD
400
300
200
100
0

Zukünftige Arbeit:

- Untersuchung der Auswirkungen von Parameteranpassungen auf die Modellergebnisse
- Analyse weiterer komplexer Modelle

Vielen Dank für Ihre Aufmerksamkeit!

Literaturverzeichnis

- Z.-x. Hu, B. Shen, Y. Hu, and C. Zhao, "Research on stock price forecast of general electric based on mixed CNN-LSTM model," *School of Computer Science, Hubei University*, Wuhan, China, Jan. 15, 2025.
- L., & Brown, D. (2020). Text Classification Algorithms: A Survey.
- Guo, X., Dam, T., Dhamdhere, R., Modanwal, G., & Madabhushi, A. (2025). UNETVL: Enhancing 3D medical image segmentation with Chebyshev kernel powered Vision-LSTM. *Electrical Engineering and Computer Science Department, Michigan University*.
- J&M: Dan Jurafsky & James Martin, 2018. Speech and Language Processing (3rd ed. Draft). Available here: https://web.stanford.edu/~jurafsky/slp3/
- https://www.geeksforgeeks.org/difference-between-a-bidirectional-lstm-and-an-lstm/
- https://medium.com/@souro400.nath/why-is-bilstm-better-than-lstm-a7eb0090c1e4
- https://www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm
- https://dagshub.com/blog/rnn-lstm-bidirectional-lstm/#lstm-vs-bidirectional-lstm
- https://medium.com/@anishnama20/understanding-bidirectional-lstm-for-sequential-data-processing-b83d6283befc#:~:text=Bi%2DLSTM%20(Bidirectional%20Long%20Short,context%20of%20the%20input%20sequence.
- J. Li, X. Chen, E. Hovy & D. Jurafsky (2015). Visualizing and Understanding Neural Models in NLP. *arXiv preprint arXiv:1506.01066*.