Big Data & Data Science

Transformer, LLM

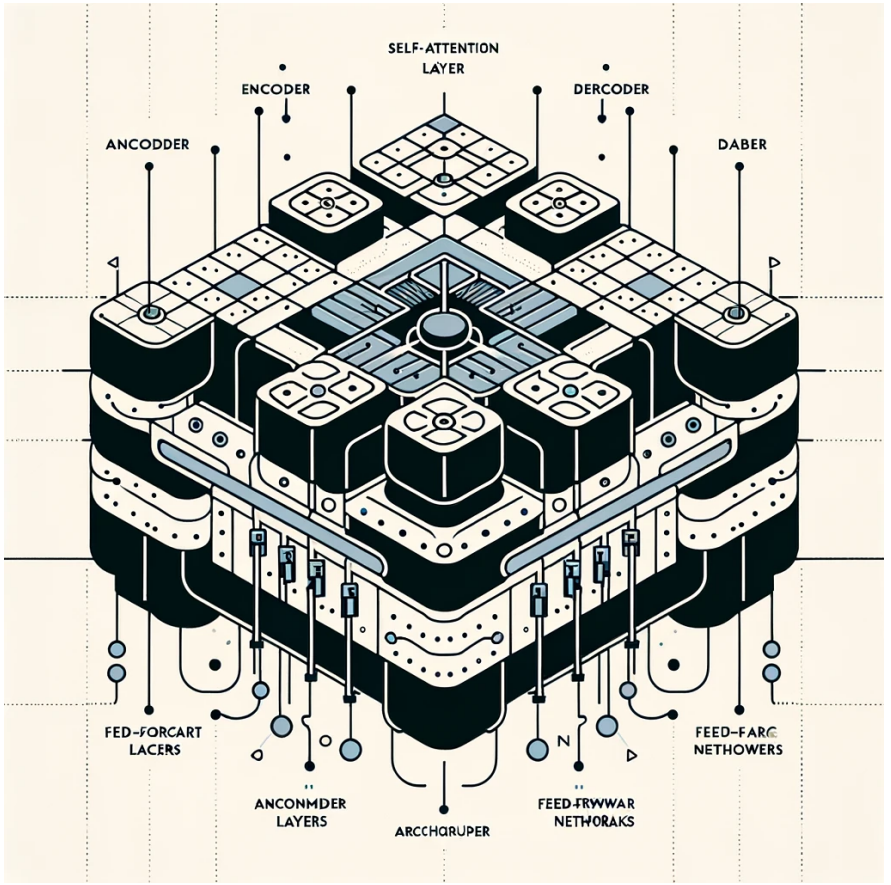
WS 2025/26

Prof. Dr. Klemens Waldhör

Transformer-Modelle
Einführung und Funktionsweise

Was ist ein Transformer-Modell?

- Transformer haben die NLP-Landschaft revolutioniert und RNNs und CNNs in vielen Aufgaben ersetzt.
- Hauptanwendung: Maschinelle Übersetzung, Textgenerierung, Sprachverständnis.



Architektur des Transformer-Modells

- Besteht aus zwei Hauptkomponenten: Encoder und Decoder.
- Der Encoder kodiert den Eingabetext in einen abstrakten, hochdimensionalen Raum.
- Der Decoder dekodiert diese Repräsentation und generiert den Ausgabetext.

Der Artikel

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kais and Polosukhin, I. 2017. Attention Is All You Need.

https://arxiv.org/abs/1706.03762

Architektur des Transformer-Modells

„ The dominant sequence transduction models are based on complex recu convolutional neural networks in an encoder-decoder configuration. The b performing models also connect the encoder and decoder through an atte mechanism. We propose a new simple network architecture, the Transfor based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show models to be superior in quality while being more parallelizable and requir significantly less time to train. Our model achieves 28.4 BLEU on the WM English-to-German translation task, improving over the existing best result including ensembles by over 2 BLEU. On the WMT 2014 English-to-Frenc translation task, our model establishes a new single-model state-of-the-ar score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of training costs of the best models from the literature. We show that the Tra generalizes well to other tasks by applying it successfully to English const parsing both with large and limited training data.”

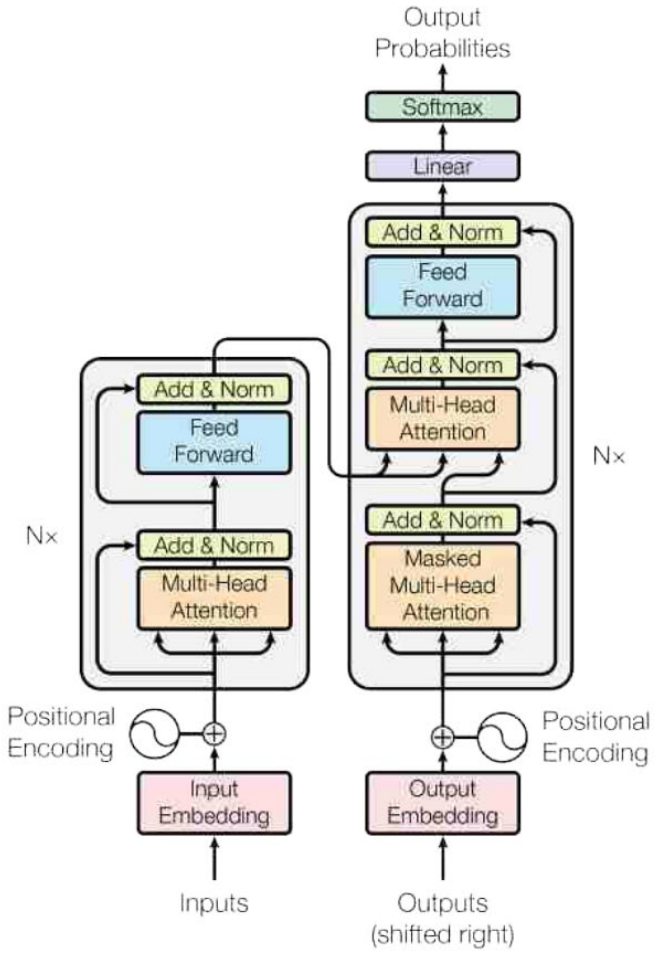


Figure 1: The Transformer - model architecture.

Der Artikel

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., and Polosukhin, I. 2017. **Attention Is All You Need**.

Architektur des Transformer-Modells

Encoder (N× wiederholt)
Jeder Encoder-Layer enthält:
Multi-Head Self-Attention – Tokens schauen aufeinander.
Add & Norm – Residual + LayerNorm stabilisieren.
Feedforward-Netz – verarbeitet Merkmale weiter.

Tokens werden in d_model-dimensionale Vektoren umgewandelt.
Positionale Codierung wird addiert, um Wortposition zu kodieren.
Ergebnis: semantischer Inhalt + Positionsinformation.

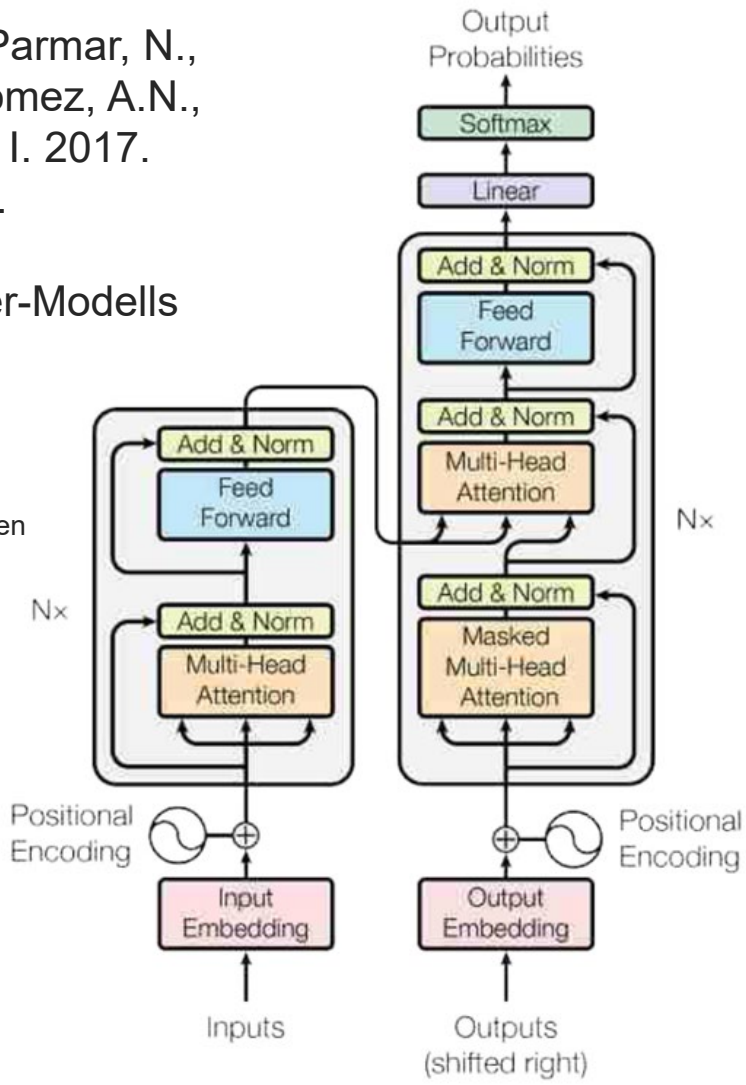


Figure 1: The Transformer - model architecture.

Decoder-Ausgabe wird in Vokabulardimension projiziert.
Softmax wählt wahrscheinlichstes nächstes Token.
(Softmax ist eine Funktion, die einen beliebigen Zahlenvektor in eine Wahrscheinlichkeitsverteilung umwandelt, indem sie exponentiert und anschließend normalisiert wird.)

Decoder (N× wiederholt)
Der Decoder hat drei Schritte:
Masked Self-Attention – kein Blick in die Zukunft.
Cross-Attention – schaut auf Encoder-Ausgabe. Feedforward + Add & Norm – Stabilität & Transformation.

Architektur der Transformer-Schichten

- Jede Encoder- und Decoder-Schicht besteht aus Sub-Layern:
  1. Multi-Head Self-Attention Mechanismus
  2. Punktweise vollständig verbundenes Feed-Forward Netzwerk
- Residual Connections (Sprungverbindungen) und Layer-Normalisierung werden nach jedem Sub-Layer angewendet.
- Attention ist ein Mechanismus, der für jedes Token gewichtet bestimmt, welche anderen Tokens im Kontext für seine Repräsentation relevant sind.
- Multi-Head Attention erlaubt dem Modell, Informationen aus verschiedenen Repräsentationsräumen zu kombinieren.
- Ein Repräsentationsraum ist der Vektorraum, in dem Tokens intern dargestellt werden. Das bedeutet, dass gleicher Text unter verschiedenen Blickwinkel betrachtet wird, etwa Syntax, Semantik, längere Abgängigkeiten.
- Das Feed-Forward Netzwerk ist für jede Position unabhängig und besteht aus zwei linearen Transformationen mit einer ReLU-Aktivierung dazwischen.

Typische Architektur eines Large Language Models (LLM)

Decoder-only

Moderne LLMs basieren fast ausschließlich auf der Transformer-Architektur (Decoder-only).

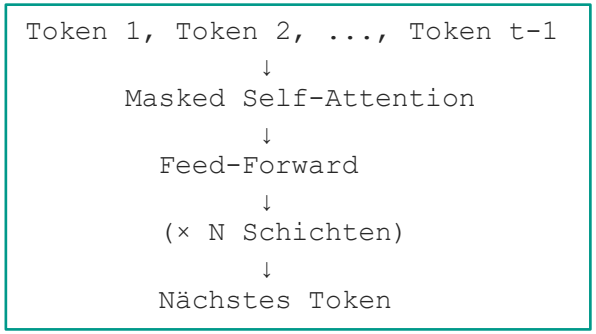
- Ein Modell besteht aus einer Stapelung identischer Transformer-Schichten.
- Die Modellgröße ergibt sich aus Anzahl der Schichten und deren Breite.
- Das Modell besteht ausschließlich aus Decoder-Schichten.
- Fast alle modernen LLMs (GPT, LLaMA, Mistral) sind Decoder-only.

Kein Encoder, keine Encoder–Decoder-Attention.

- Nur masked Self-Attention + Feed-Forward-Netze.
- Das Modell sieht ausschließlich vergangene Tokens.
- Zukünftige Tokens werden durch Maskierung ausgeblendet.

Der ursprüngliche Transformer besteht aus Encoder und Decoder.

- Encoder: verarbeitet Eingabetext vollständig.
- Decoder: erzeugt Ausgabetext Token für Token.



Typische Architektur eines Large Language Models (LLM)

Trainingsziel von Decoder-only Modellen

Vorhersage des nächsten Tokens (Next-Token-Prediction).

- Formal: P(w_t | w_1, …, w_{t-1}).
- Einheitliches Lernziel für alle Aufgaben.
- Architektonisch einfacher als Encoder–Decoder.
- Sehr gut skalierbar auf Milliarden Parameter.
- Ideal für Textgenerierung und Dialogsysteme.

Abgrenzung zu anderen Architekturen

- Encoder-only (z. B. BERT): Analyse & Klassifikation. BERT ist ein reines *Encoder-Modell*, das Text bidirektional versteht – aber keinen Text generiert.
- Encoder–Decoder (z. B. T5): Transformation von Text. T5 ist ein Transformer-Modell, bei dem jede Aufgabe als Text-→-Text-Problem formuliert wird.
- Decoder-only (z. B. GPT): Generierung von Text.

Zentrale Modellparameter

- Anzahl der Schichten (Layers): Anzahl der Transformer-Blöcke im Modell.
- Hidden Size (d_model): Dimension der Token-Repräsentation.
- Attention Heads: Parallele Aufmerksamkeitsmechanismen pro Schicht.
- Feed-Forward Dimension (d_ff): Anzahl der Neuronen im FFN-Teil.

Typische Architektur eines Large Language Models (LLM)

Parameter: Anzahl der Schichten

- Typische Werte: 12 (klein), 32 (Standard), 96+ (sehr groß).
- Mehr Schichten → tiefere Modellhierarchie
- Erlaubt komplexere sprachliche und semantische Abstraktionen.

Parameter: Hidden Size (d_model)

- Dimension des Vektors, der jedes Token repräsentiert.
  Typische Werte: 768, 4096, 12288.
  Bestimmt maßgeblich die Modellkapazität und Rechenkosten.

Parameter: Feed-Forward Network (FFN)

- Besteht aus zwei linearen Transformationen mit Nichtlinearität.
- Zwischendimension $d_{ff} \approx 4 \times d_{model}$.
- Enthält den Großteil der Modellparameter (ca. 60–70 %).

Parameter: Attention Heads

- Multi-Head Attention zerlegt d_model in mehrere Teilräume.
  Typische Werte: 12, 32, 96 Heads.
- Erlaubt parallele Betrachtung unterschiedlicher Kontextbeziehungen.

Beispiel: LLaMA / Mistral (7B)

32 Transformer-Schichten.

Hidden Size: 4096.

Attention Heads: 32.

FFN-Dimension: ca. 14.000.

Gesamtparameter: ca. 7 Milliarden.

Transformer-Modelle
Details

Encoder und Decoder

- Der Encoder verarbeitet die Eingabedaten parallel, was die Effizienz deutlich erhöht. Er verwendet Self-Attention-Mechanismen, um die Bedeutung verschiedener Wörter in der Eingabesequenz zu gewichten.
- Der Decoder erzeugt die Ausgabesequenz. Er nutzt sowohl Self-Attention als auch Encoder-Decoder-Attention-Mechanismen, um sich auf relevante Teile der Eingabe und der bereits generierten Ausgabe zu konzentrieren.

Embeddings und Modelldimensionen

- Embedding wandelt Wörter (diskrete Symbole) in kontinuierliche Vektoren um, die semantische Eigenschaften repräsentieren.

"mikrometeorit" → [0.12, -0.51, 0.33, ..., 0.04] (32 Werte)
"staub" → [-0.21, 0.77, 0.08, ..., -0.10] (32 Werte)

- Größe wird durch d_model (Hauptdimension des Transformer-Vektorraums) festgelegt, die vom Benutzer sinnvoll gewählt werden.
- Semantisch ähnliche Worte erhalten **datengetrieben** ähnliche Vektoren.
- Sie kommen in ähnlichen Kontexten gehäuft vor, daher statistische Ähnlichkeit im Kern, nicht wirklich Bedeutungsähnlichkeit im menschlichen Verständnis.
- Embeddings lernen Verteilungsähnlichkeit (Distributional Semantics): Wörter, die in ähnlichen Kontexten vorkommen, erhalten ähnliche Vektoren.

Wort	Bedeutungsähnlich zu	ähnliche Embedding-Vektoren
„Mikrometeorit“	„Staub“, „Partikel“, „kosmisch“	ja
„Erde“	„Planet“, „Mond“, „Himmelskörper“	ja
„und“	„oder“, „sowie“	ja

Embeddings und Modelldimensionen

Modelldimensionen (d_model)

- Sind eine zentrale Architektur-Entscheidung, Modelldimension muss überall konsistent sein. Embedding, Q/K/V, Attention, FFN, Output-Layer nutzen diese Dimension
  - 8–32 → Mini-Modelle
  - 64–256 → typische kompakte Modelle
  - 512–4096 → echte LLMs
- Die Modelldimension d_model legt fest, wie groß der semantische Repräsentationsraum des Modells ist. Je größer die Modelldimension, desto mehr Muster, Abhängigkeiten und Bedeutung kann das Modell abbilden – also höhere Leistungsfähigkeit – vorausgesetzt, genügend Trainingsdaten sind vorhanden.
- Daher ein größere Modelldimension bewirkt:

  ✓ Mehr Modellkapazität

  ✓ Bessere semantische Repräsentation

  ✗ Quadratisch mehr Parameter (O(d_model²))

  ✗ Höhere Trainingszeit & Overfitting-Risiko

Encoder und Decoder

Encoder

- Besteht aus mehreren Schichten.
- Jede Schicht hat zwei Hauptbestandteile:
  - Self-Attention-Mechanismus
  - Feed-Forward-Neuronales Netzwerk

  Hinzufügen von Residual Connections und Layer Normalization.
  - Eine Residual Connection (Skip Connection) fügt die ursprüngliche Eingabe wieder zum Ausgang einer Schicht hinzu, damit Informationen leichter durch das Netzwerk fließen und das Training stabiler wird.
  - Layer Normalization normalisiert die Aktivierungen einer Schicht pro Token, um den Wertebereich zu stabilisieren und das Training schneller und robuster zu machen.

Decoder

- Funktioniert ähnlich wie der Encoder, aber mit zusätzlichen Komponenten.
- Zwei Arten von Attention: Self-Attention und Encoder-Decoder-Attention.
- Output von Encoder wird genutzt, um den Kontext während der Generierung zu berücksichtigen.

Self-Attention

- Der Attention-Mechanismus ist ein zentrales Merkmal von Transformern. Er ermöglicht es dem Modell, sich beim Generieren jedes Wortes in der Ausgabesequenz auf unterschiedliche Teile der Eingabesequenz zu konzentrieren. Dadurch entstehen genauere und kohärentere Übersetzungen bzw. Textgenerierungen.
- Transformer besitzen von sich aus kein Verständnis für die Reihenfolge von Wörtern in einer Sequenz. Daher werden Positionsencodierungen zu den Eingabe-Embeddings hinzugefügt, um dem Modell Informationen über die Position jedes Wortes in der Sequenz zu geben.

Self Attention → Selbstaufmerksamkeit**

Selbstaufmerksamkeit ermöglicht es einem Modell, jedes Wort im Kontext des gesamten Satzes zu betrachten, anstatt isoliert, indem die Bedeutung der anderen Wörter in Beziehung zu jedem einzelnen Wort gewichtet wird.

- Erlaubt dem Modell, auf verschiedene Teile des Eingabetextes gleichzeitig zu achten.
- Berechnung der Aufmerksamkeit anhand von Query, Key und Value Matrizen.
- Multi-Head Attention: Mehrere Attention-Mechanismen werden parallel ausgeführt

Kernbegriffe

- Query (Q) – Was suche ich? Beschreibt die aktuelle Position, die gerade Aufmerksamkeit verteilen möchte.
- Key (K) – Wie relevant bin ich? Beschreibt jedes Wort im Satz im Hinblick darauf, ob es relevant für die aktuelle Query ist.
- Value (V) – Welche Information gebe ich weiter? Enthält die eigentliche Inhaltsinformation, die an die Query zurückgegeben wird.

Self-Attention

1. Input:

Menge von Vektoren, die die Wörter eines Satzes repräsentieren.

2. Query, Key, Value:

Jedes Wort wird in drei Vektoren transformiert –
Query (Q), Key (K) und Value (V).

3. Attention Scores

Berechnung der Aufmerksamkeitswerte über das Skalarprodukt von Q und K, gefolgt von einer Normalisierung.

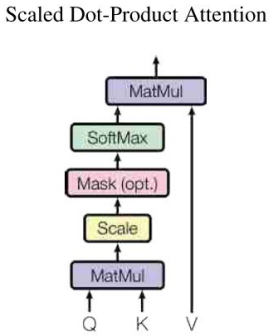
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

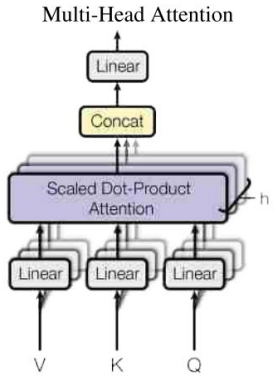
4. Gewichtete Summe

Für jedes Wort wird ein neuer Vektor erzeugt, basierend auf der gewichteten Summe der V-Vektoren.

5. Output

Neuer Vektor, der kontextabhängige aggregierte Information repräsentiert.





Beispielsatz: „NLP is amazing“

1. Eingabewortvektoren: NLP -> [1, 0, 1], is -> [0, 1, 0], amazing -> [1, 1, 0]
2. Lineare Transformationen zu Query, Key und Value Vektoren
3. Berechnung der Aufmerksamkeitswerte (Scores)
4. Anwendung der Softmax-Funktion
5. Gewichtete Summe der Value-Vektoren zur Berechnung der neuen Wortdarstellungen

Positionale Codierung

- Transformer-Modelle haben keine inhärente Reihenfolge.
- Positionale Codierung wird hinzugefügt, um die Position der Wörter im Satz zu berücksichtigen.
- Verwendet Sinus- und Cosinus-Funktionen basierend auf der Position des Wortes.
- Formeln:

  $PE(pos, 2i) = \sin(pos / 10000^{(2i/d\_model)})$

  $PE(pos, 2i+1) = \cos(pos / 10000^{(2i/d\_model)})$

Wort	Position (pos)
NLP	0
is	1
amazing	2

Beispielsatz: „NLP is amazing“

1. Wortvektoren: NLP -> [1, 0, 1, 0], is -> [0, 1, 0, 1], amazing -> [1, 1, 1, 1]
2. Positionale Codierungen berechnen:

   Position 0: [0, 1, 0, 1]

   Position 1: [sin(1), cos(1), sin(1), cos(1)]

   Position 2: [sin(2), cos(2), sin(2), cos(2)]
3. Positionale Codierungen zu den Wortvektoren addieren.

`i` ist die Dimensions-Nummer innerhalb des d_model-Vektors. d_model ist die Embedding-Dimension des Modells.

Training und Optimierung

- Verwendung von großen Datensätzen für das Training.
- Adam Optimizer wird oft genutzt.
  - Adam (Adaptive Moment Estimation) ist ein Optimierungsalgorithmus, der bei jedem Parameter die Lernrate automatisch anpasst, indem er sowohl den Mittelwert (1. Moment) als auch die Varianz (2. Moment) der Gradienten schätzt und dadurch schneller und stabiler konvergiert als klassisches Gradient Descent.
- Verlustfunktion: Normalerweise Kreuzentropieverlust für die Sequenzgenerierung.
  - Cross Entropy Loss misst die Güte einer Wahrscheinlichkeitsvorhersage, indem er hohe Strafen für falsche Klassen und niedrige Strafen für korrekte Klassen vergibt — basierend auf der negativen Log-Wahrscheinlichkeit des richtigen Tokens.

Anwendungen

- Maschinelle Übersetzung (z.B., Google Translate)
- Textgenerierung (z.B., GPT-Modelle)
- Sprachverständnis (z.B., BERT-Modelle)
- Frage-Antwort-Systeme, Textzusammenfassung

Zusammenfassung

- Transformer überwinden die Einschränkungen von RNNs und LSTMs, wie die schwierige Parallelisierung der Verarbeitung sowie Probleme mit weitreichenden Abhängigkeiten in Sequenzen.
- Transformer werden in vielen NLP-Aufgaben eingesetzt, z. B. maschinelle Übersetzung, Textzusammenfassung und Frage-Antwort-Systeme.
- Die Transformer-Architektur stellt einen bedeutenden Fortschritt im Bereich NLP dar, da sie eine höhere Effizienz und Effektivität bei der Verarbeitung sequenzieller Daten ermöglicht.

Modelllauf Mini Transformer Demo - Summarizer

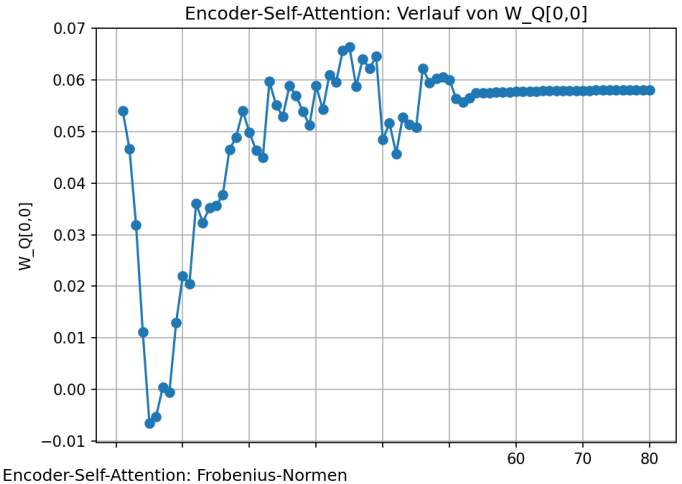
Folgende Folien basieren auf dem Demo-Programmtransf_ed_commented.py „Mini Transformer“.

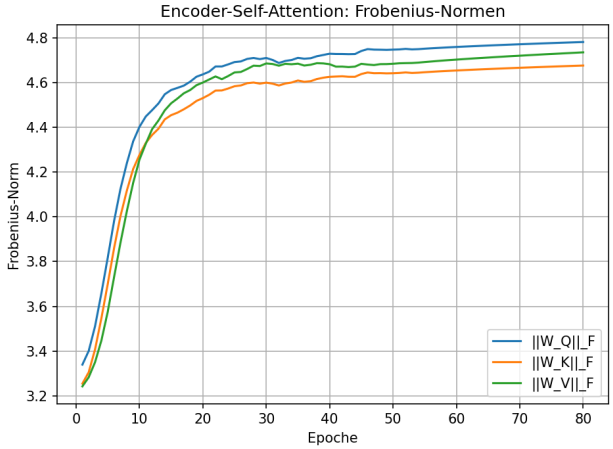
Das Programm lädt Mikrometeoriten-Artikel, zerlegt sie in kleine Textabschnitte, trainiert darauf einen Mini-Transformer und erzeugt anschließend eigenständig kurze Summaries.

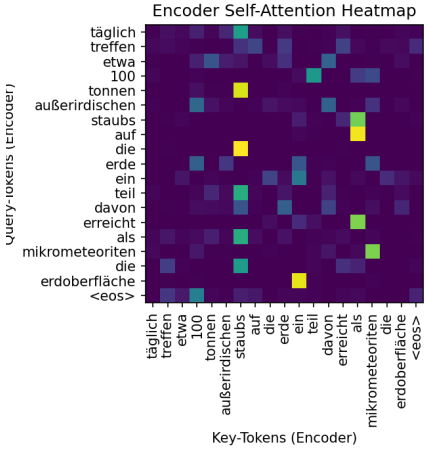
1. Datenaufbereitung: Artikel werden aus einem Ordner eingelesen, ggf. automatisch in Chunks von 6 Sätzen aufgeteilt, falls kein Summary vorhanden ist → automatisch generierte Mini-Summary, Tokenisierung + Aufbau eines Vokabulars
2. Encoder wandelt Text in Vektoren um (Embedding + Positionen), berechnet Self-Attention, baut ein „Gedächtnis“ des Eingabetextes auf
3. Decoder: startet mit <bos>nutzt, maskierte Self-Attention (nur Vergangenheit sichtbar), Cross-Attention zum Encoder (Decoder „fragt“ im Text nach), erzeugt Token für Token die Zusammenfassung
4. Training: klassisches seq2seq-Training mit Cross-Entropy, Gewichte von Q, K, V werden nach jeder Epoche gespeichert, Plots: Frobenius-Normen, einzelne Gewichte, Attention-Heatmaps
5. Generierung: nach dem Training kann zu jedem Mikrometeoriten-Text eine neue Zusammenfassung erzeugt werden, autoregressiv, bis <eos> erreicht
6. Debug- & Analysefunktionen: Encoder-Self-Attention-HeatmapDecoder-Cross-Attention-Heatmap, automatischer HTML-Report, der alle Ausgaben, Plots und Erklärtexte enthält

Modelllauf Mini Transformer Demo - Summarizer

```
(venv310) python transf_ed_commented.py
[dataset_loader] SuW_Sternenstaub_für_jeden_v06_TH_WK_TH.txt:
26 Chunks erzeugt.
[dataset_loader] dok1.txt: 4 Chunks erzeugt.
[dataset_loader] dok2.txt: 4 Chunks erzeugt.
[dataset_loader] dok3.txt: 4 Chunks erzeugt.
[dataset_loader] wega_appril23_v2.txt: 19 Chunks erzeugt.
[dataset_loader] Geladene Trainingspaare: 57
Starte Mini Encoder-Decoder mit:
  d_model = 32
  dim_ff  = 64
  epochs  = 80
  lr      = 0.001

Device          : cpu
Vokabulargröße  : 2040
Starte Training des Mini Encoder-Decoder-Transformers

Epoch   1 | Loss: 7.8595
Epoch  10 | Loss: 3.1393
Epoch  20 | Loss: 0.5821

…
```







Testtext:
täglich treffen etwa 100 tonnen außerirdischen staubs au
teil davon erreicht als mikrometeoriten die erdoberfläche

Generierte Zusammenfassung:
bachelor-thesis würde geräte und und und bzw auf basis haben worden
hat dieses händen zahllose des zahllose

Modelllauf Mini Transformer Demo - Summarizer

1. Embedding (d_model = 8)

Embedding-Vektor für „mikrometeorit“:

- E(mikrometeorit)=[0.20,−0.15,0.80,1.10,−0.55,0.30,0.05,0.90]
- Dieser Vektor ist die d_model-Repräsentation des Tokens.

2. Q / K / V Matrizen (8×8)

Beispielhafte Gewichtsmatrix W_Q (8×8):

- Matrixmultiplikation: Q = Embedding E × W_Q
- Analog: K = Embedding × W_K, V = Embedding × W_V
- Dies erzeugt Query-, Key- und Value-Vektoren gleicher Dimension.

$$W_Q = \begin{bmatrix} 0.4 & 0.2 & -0.1 & 0.0 & 0.5 & -0.3 & 0.1 & 0.2 \\ -0.2 & 0.3 & 0.4 & 0.1 & 0.0 & 0.2 & -0.1 & 0.1 \\ 0.1 & 0.5 & -0.2 & 0.3 & -0.1 & 0.0 & 0.4 & -0.3 \\ -0.3 & 0.1 & 0.2 & 0.5 & 0.3 & -0.2 & 0.1 & 0.0 \\ 0.5 & -0.1 & 0.3 & 0.2 & -0.4 & 0.1 & 0.2 & 0.4 \\ 0.2 & 0.1 & -0.3 & -0.2 & 0.1 & 0.6 & 0.5 & -0.1 \\ 0.4 & 0.3 & 0.2 & -0.1 & 0.0 & 0.1 & 0.5 & 0.2 \\ -0.1 & 0.2 & 0.3 & 0.4 & 0.1 & -0.2 & 0.1 & 0.6 \end{bmatrix}$$

Modelllauf Mini Transformer Demo - Summarizer

3. Query-Vektor (Q)

Ergebnis (Beispiel):

- Q = [0.64, 0.07, 0.32, 0.95, -0.21, 0.27, 0.49, 0.81]
- Aus der Multiplikation des Embeddings mit W_Q.

4. Key-Vektor (K)

K = [0.51, -0.12, 0.44, 1.02, -0.33, 0.15, 0.60, 0.75]

- Aus der Multiplikation des Embeddings mit W_K.

5. Value-Vektor (V)

V = [0.40, 0.10, 0.55, 0.85, -0.10, 0.33, 0.22, 0.91]

- Aus der Multiplikation des Embeddings mit W_V.

Modelllauf Mini Transformer Demo - Summarizer

6. Attention Score

$$\text{score} = \frac{Q \cdot K}{\sqrt{d_{model}}}$$

$$\begin{aligned} Q \cdot K &= 0.64 \cdot 0.51 + 0.07 \cdot (-0.12) + 0.32 \cdot 0.44 + 0.95 \cdot 1.02 + (-0.21) \cdot (-0.33) + 0.27 \cdot 0.15 + 0.49 \cdot 0.60 + 0.81 \cdot 0.75 \\ &= 0.3264 - 0.0084 + 0.1408 + 0.969 + 0.0693 + 0.0405 + 0.294 + 0.6075 \\ &= 2.4381 \end{aligned}$$

- Skalarprodukt Q · K = 2.4381
- Skalierung durch √d_model = 2.828 → Score = 0.862
- Softmax über einen Token → Gewicht = 1.0

Modelllauf Mini Transformer Demo - Summarizer

7. Kontextvektor

- Context = Gewicht × V = V
- C = [0.40, 0.10, 0.55, 0.85, -0.10, 0.33, 0.22, 0.91]
- Dieser Vektor trägt bereits Kontextinformation.

8. Feedforward Layer Output

$$FFN(x) = W_2 \cdot \mathrm{ReLU}(W_1 x + b_1) + b_2$$

Nach nichtlinearer Transformation:

- FFN(C) = [0.52, 0.01, 1.12, 0.75, -0.22, 0.44, 0.31, 0.83]
- Dies ist die finale Encoder-Repräsentation.

Modelllauf Mini Transformer Demo - Summarizer

9. Decoder Schritt 1 – Input <SOS>

- Embedding(<SOS>) = [-0.3, 0.2, 0.1, 0.5, -0.1, 0.4, 0.0, 0.2]
- Masked Self Attention findet statt, aber nur ein Token → trivial.

10. Decoder – Cross Attention

Decoder Q_d wird mit Encoder-K verglichen:

- Q_d=[0.12,−0.05,0.33,0.80,−0.10,0.25,0.04,0.50]
- Q_d · K = 1.889 → skaliert: 0.668 → Softmax = 1.0
- Kontext C_d = V = [0.40, 0.10, 0.55, 0.85, -0.10, 0.33, 0.22, 0.91]

11. Decoder nach Feedforward

Decoder-Ausgabe H:

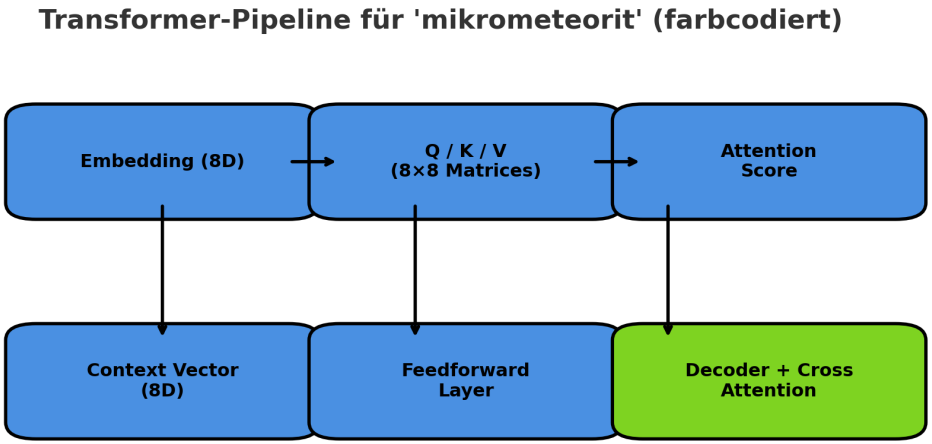
- [0.60, 0.08, 1.20, 0.90, -0.15, 0.41, 0.29, 0.96]
- Dieser Vektor wird in die Vocabulary-Projektion eingespeist.

Modelllauf Mini Transformer Demo - Summarizer

12. Projektion → Softmax

Logits (Beispiel): [-0.1, 1.5, 0.9, 0.2]

- Softmax → [0.06, 0.63, 0.26, 0.05]
- Wahrscheinlichstes Wort: „staub“



Zusammengefasst

- Embedding-Vektor
- Q / K / V Matrizen (8×8)
- Attention-Scores
- Context-Vektor
- Feedforward-Output
- Decoder: Masked Attention + Cross-Attention
- Output-Projektion → Softmax

Modelllauf Mini Transformer Demo – Variante Klasse bestimmen

```
(venv310) python transf_cls.py
MiniSentenceClassifier initialisiert:
  MiniSentenceClassifier initialisiert:
  Vokabulargröße : 97
  Klassen        : 2 → ['mikrometeorit_info', 'staub_statistik']
  d_model        : 32
  dim_ff         : 64
  num_layers     : 2
  Device         : cpu
Epoch   1 | Loss: 0.7356
Epoch   5 | Loss: 0.4356
Epoch  10 | Loss: 0.0734
Epoch  15 | Loss: 0.0115
Epoch  20 | Loss: 0.0060
Epoch  25 | Loss: 0.0040
Epoch  30 | Loss: 0.0029
Epoch  35 | Loss: 0.0023
Epoch  40 | Loss: 0.0019
Training abgeschlossen.

Text: Mikrometeoriten kann man auf Flachdächern finden.
  → vorhergesagte Klasse: staub_statistik
  → Wahrscheinlichkeiten: {'mikrometeorit_info': 0.03971893712878227, 'staub_statistik': 0.9602810144424438}

Text: Jeden Tag regnet tonnenweise Staub aus dem All auf die Erde.
  → vorhergesagte Klasse: mikrometeorit_info
  → Wahrscheinlichkeiten: {'mikrometeorit_info': 0.9985357522964478, 'staub_statistik': 0.0014642801834270358}

Text: Die Atmosphäre schützt uns vor vielen kleinen Staubteilchen.
  → vorhergesagte Klasse: mikrometeorit_info
  → Wahrscheinlichkeiten: {'mikrometeorit_info': 0.905990719795227, 'staub_statistik': 0.09400926530361176}
```

Gruppenaufgabe: Transformer-Modelle verstehen

Sie können als Basis das zur Verfügung gestellt Programm verwenden.

Aufgabe 1:

- Gemeinsame Anwendung recherchieren (z. B. Übersetzung, Chatbot, Agentic AI).
- Gruppen bearbeiten unterschiedliche Aspekte des Transformers.
- Ziel: Funktionsweise anhand eines konkreten Beispiels erklären.
- 10 Minuten

Aufgabe 2: Einzelaspekte Transformer

- Sh. Folgefolien
- 20 Minuten

Präsentation

- Alle Teilaufgaben zu einer gemeinsamen Präsentation
- Jede Gruppe präsentiert 3–5 Minuten.
- Definitionen, Grafiken, Mini-Beispiel.
- Kernerkenntnisse zur gewählten Anwendung.

Aufgabe 2: Gruppe 1 – Query, Key, Value & Gewichtsmatrizen

- Was sind Q, K, V?
- Wie entstehen sie durch lineare Transformationen?
- Wie werden W^Q, W^K, W^V gelernt?
- Mini-Beispiel mit Bezug zur Anwendung.

Aufgabe 2: Gruppe 2 – Self-Attention

- Berechnung der Attention Scores.
- Softmax und gewichtete Summen.
- Warum Self-Attention kontextsensitiv ist.
- Rechenbeispiel zur Anwendung.

Aufgabe 2: Gruppe 3 – Encoder

- Rolle des Encoders im Transformer.
- Mehrere Attention- und FFN-Schichten.
- Residuals & Layer Normalization.
- Welche Informationen der Encoder weitergibt.

Aufgabe 2: Gruppe 4 – Decoder

- Masked Self-Attention.
- Encoder–Decoder-Attention.
- Wie der Decoder das nächste Token generiert.
- Interaktion zwischen Encoder und Decoder.

Aufgabe 3: Weitere Transformer-Modelle

Neben LLMs, den klassischen Einsatzgebiet von Transformern, gibt es noch andere Varianten.

Erstellen Sie eine Liste weiterer Transformer-Modelle.