



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Lineare-Algebra- Grundlagen für Transformer

Embeddings, Projektionen, Self-Attention und Vorhersage des
nächsten Tokens

18. Mai 2026

Grundlagen: Vektoren und Matrix-Sichtweisen



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Ziel und Notationskonventionen

Diese Folien fassen die lineare Algebra zusammen, die für Embeddings, Projektionen, Self-Attention (Selbstaufmerksamkeit) und die Vorhersage des nächsten Tokens benötigt wird.

Wir verwenden **beide** Konventionen:

- **Spaltenvektor-Konvention:** Vektoren in $\mathbb{R}^n$ werden als Spalten geschrieben.
- **Zeilenvektor-Konvention:** Vektoren werden als Transponierte geschrieben, also als Zeilen in $\mathbb{R}^{1 \times n}$.

In der Transformer-Praxis treten beide auf:

- Mathematik bevorzugt **Spaltenvektoren**
- Implementierungen sind natürlicher in **Zeilenform** — ein Token pro Zeile

Zentrale Lehraussage

Linksmultiplikation wirkt natürlich auf Spalten.
Rechtsmultiplikation wirkt natürlich auf Zeilen.

Skalarprodukt, Norm und Kosinus

Seien $x, y \in \mathbb{R}^n$.

Skalarprodukt

$$\langle x, y \rangle = x^\top y = \sum_{i=1}^n x_i y_i$$

bilinear, symmetrisch, positiv definit: $x^\top x \geq 0$, mit Gleichheit genau dann, wenn $x = 0$.

Euklidische Norm und Winkel

$$\|x\|_2 = \sqrt{x^\top x}, \quad \cos \theta = \frac{x^\top y}{\|x\|_2 \|y\|_2}$$

Der Kosinus misst die **Richtungsähnlichkeit** unabhängig von der Skalierung.

Eine Matrix ist beides: Spalten und Zeilen

Sei $A \in \mathbb{R}^{m \times n}$.

Spaltensicht:

$$A = (a_1, \dots, a_n), \quad a_j \in \mathbb{R}^m.$$

Zeilensicht:

$$A = \begin{pmatrix} r_1^\top \\ \vdots \\ r_m^\top \end{pmatrix}, \quad r_i \in \mathbb{R}^n.$$

Beide Sichtweisen sind äquivalent — und jede macht eine andere Identität offensichtlich.

Matrixmultiplikation



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Matrix mal Vektor: Zwei Interpretationen

Seien $A \in \mathbb{R}^{m \times n}$ und $x \in \mathbb{R}^n$.

Spaltenkombination:

$$Ax = x_1 a_1 + \cdots + x_n a_n.$$

Ax ist eine Linearkombination der Spalten von A.

Zeilenauswertung:

$$Ax = \begin{pmatrix} r_1^\top x \\ \vdots \\ r_m^\top x \end{pmatrix}.$$

Jeder Eintrag von Ax ist ein zeilenweises Skalarprodukt mit x.

Beide Aussagen sind gleichzeitig wahr.

Zeilenvektor mal Matrix

Seien $u^\top \in \mathbb{R}^{1 \times m}$ und $A \in \mathbb{R}^{m \times n}$. Dann ist $u^\top A \in \mathbb{R}^{1 \times n}$.

Zeilenkombination:

$$u^\top A = u_1 r_1^\top + \cdots + u_m r_m^\top.$$

Spaltenauswertung:

$$u^\top A = (u^\top a_1 \quad \cdots \quad u^\top a_n).$$

Zeilenvektor mal Matrix = Linearkombination der Zeilen von A .

Matrix mal Matrix

Seien $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{n \times p}$, also $AB \in \mathbb{R}^{m \times p}$.

Spaltenweise Sicht:

$$AB = (Ab_1, \dots, Ab_p).$$

Linksmultiplikation mit A transformiert jede *Spalte* von B .

Zeilenweise Sicht:

$$AB = \begin{pmatrix} r_1^\top B \\ \vdots \\ r_m^\top B \end{pmatrix}.$$

Rechtsmultiplikation mit B transformiert jede *Zeile* von A .

Die Kernidentität für die zeilenweise Transformer-Notation.

Eintragsweise Formel

Der (i, j) -Eintrag von AB ist

$$(AB)_{ij} = \sum_{k=1}^n A_{ik} B_{kj} = r_i^\top c_j,$$

wobei $r_i^\top$ die i -te Zeile von A und c_j die j -te Spalte von B ist.

Kernerkenntnis

Jeder Eintrag eines Matrixprodukts ist ein Zeile-Spalte-Skalarprodukt.

Matrix als Funktion — lineare Abbildungen

Jede Matrix $A \in \mathbb{R}^{m \times n}$ definiert eine Funktion

$$f_A : \mathbb{R}^n \rightarrow \mathbb{R}^m, \quad f_A(x) = Ax.$$

Definition (lineare Abbildung)

$f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ heißt **linear** genau dann, wenn für alle $x, y \in \mathbb{R}^n$ und $\alpha, \beta \in \mathbb{R}$

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y).$$

- f_A ist linear (direkter Nachweis über die Distributivität).
- Umgekehrt ist jede lineare Abbildung $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ durch eine eindeutige Matrix gegeben, sobald eine Basis fixiert ist: Spalte j von A ist $f(e_j)$.

Matrizen und lineare Abbildungen zwischen endlich-dimensionalen Räumen sind *dasselbe*.

Affine Abbildungen — linear plus Verschiebung

Definition (affine Abbildung)

$g : \mathbb{R}^n \rightarrow \mathbb{R}^m$ heißt **affin** genau dann, wenn $A \in \mathbb{R}^{m \times n}$ und $b \in \mathbb{R}^m$ existieren mit

$$g(x) = Ax + b.$$

Äquivalent: $g - g(0)$ ist linear.

- Linear $\subset$ affin (setze $b = 0$).
- Affine Abbildungen erhalten affine Kombinationen: $g(\sum_i \lambda_i x_i) = \sum_i \lambda_i g(x_i)$, sofern $\sum_i \lambda_i = 1$.
- **Konkreter Hinweis:** die Abbildung $h \mapsto hW_1 + b_1$ im FFN ist affin, nicht linear — der Bias b_1 bricht die Linearität.

Komposition rechtfertigt die Matrixmultiplikation

Seien $f_A : \mathbb{R}^n \rightarrow \mathbb{R}^m$ und $f_B : \mathbb{R}^m \rightarrow \mathbb{R}^p$ linear, mit Matrizen A, B . Dann gilt

$$(f_B \circ f_A)(x) = B(Ax) = (BA)x.$$

Die Matrixmultiplikation ist genau so *definiert*, damit dies funktioniert:

$$(BA)_{ij} = \sum_k B_{ik} A_{kj}.$$

Die Formel ist nicht beliebig — sie ist erzwungen durch die Forderung „Matrix der Komposition = Produkt der Matrizen“.

Abgeschlossenheit unter Komposition

- Die Komposition linearer Abbildungen ist **linear**.
- Die Komposition affiner Abbildungen ist **affin**:

$$B(Ax + b_1) + b_2 = (BA)x + (Bb_1 + b_2).$$

Beliebig viele lineare (oder affine) Schichten zu stapeln bleibt in derselben Klasse — genau deshalb braucht man Nichtlinearitäten.

Matrixverkettung als Funktion

Zwei Möglichkeiten, Matrizen zu größeren zusammenzusetzen:

Horizontale Stapelung. Für $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{m \times p}$:

$$[A \mid B] \in \mathbb{R}^{m \times (n+p)}, \quad [A \mid B] \begin{pmatrix} x \\ y \end{pmatrix} = Ax + By.$$

Liest A auf den ersten n Koordinaten, B auf den letzten p .

Vertikale Stapelung. Für $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{p \times n}$:

$$\begin{pmatrix} A \\ B \end{pmatrix} \in \mathbb{R}^{(m+p) \times n}, \quad \begin{pmatrix} A \\ B \end{pmatrix} x = \begin{pmatrix} Ax \\ Bx \end{pmatrix}.$$

Wendet A und B parallel auf denselben Input an und stapelt die Ausgaben.

Vorgriff: Vertikale Stapelungen liegen den per-Head-Ausgaben in MHA zugrunde; horizontale Stapelungen erscheinen bei fusionierten QKV-Projektionen.

Links- vs. Rechtsmultiplikation

- AX : wirkt auf **Spalten** » Spaltenwirkung
- XB : wirkt auf **Zeilen** » Zeilenwirkung

Transponierte, äußere Produkte, Gram-Matrizen



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Regeln für die Transponierte

Für passende Matrizen gilt

$$(A^\top)^\top = A, \quad (A + B)^\top = A^\top + B^\top, \quad (AB)^\top = B^\top A^\top.$$

Die letzte Identität ist besonders wichtig.

Beispiel: Attention-Score-Matrix

$$QK^\top = (XW_Q)(XW_K)^\top = XW_QW_K^\top X^\top.$$

Äußere Produkte und Rang-eins-Matrizen

Für $u \in \mathbb{R}^m$, $v \in \mathbb{R}^n$ hat das äußere Produkt

$$uv^\top \in \mathbb{R}^{m \times n}, \quad (uv^\top)_{ij} = u_i v_j$$

höchstens Rang 1.

Zerlegung eines Matrixprodukts in äußere Produkte. Sind $A = (a_1, \dots, a_n)$ und $B^\top = (b_1, \dots, b_n)$, so gilt

$$AB = \sum_{k=1}^n a_k b_k^\top.$$

Viele Matrixidentitäten lassen sich am natürlichsten als Summen von Rang-eins-Anteilen lesen.

Gram- und Ähnlichkeitsmatrizen

Sei $X \in \mathbb{R}^{T \times d}$.

Gram-Matrix der Zeilen von X :

$$XX^\top \in \mathbb{R}^{T \times T}, \quad (XX^\top)_{ij} = x_i^\top x_j.$$

Gram-Matrix der Spalten von X :

$$X^\top X \in \mathbb{R}^{d \times d}.$$

$XX^\top$ enthält *alle paarweisen zeilenweisen Skalarprodukte* — die strukturelle Blaupause der Attention-Scores.

Bilinearformen



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Die adjungierte Identität: $\langle Ax, y \rangle = \langle x, A^\top y \rangle$

Für $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$:

$$\langle Ax, y \rangle = (Ax)^\top y = x^\top A^\top y = \langle x, A^\top y \rangle,$$

mit $(AB)^\top = B^\top A^\top$ im mittleren Schritt.

Lesart

Die Transponierte wandert frei durch das Skalarprodukt: was A links von $\langle \cdot, \cdot \rangle$ tut, tut $A^\top$ rechts.

Genau dieser Schritt wird bei der Herleitung der Attention-Scores verwendet:

$$S_{ij} = (h_i^\top W_Q)(h_j^\top W_K)^\top = \langle W_Q^\top h_i, W_K^\top h_j \rangle.$$

Bilinearformen — verallgemeinerte Skalarprodukte

Definition

$b : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ heißt **bilinear**, wenn b in jedem Argument linear ist (bei festgehaltenem anderen).

Jede Bilinearform auf $\mathbb{R}^n$ besitzt die Form

$$b(x, y) = x^\top M y \quad \text{für ein eindeutiges } M \in \mathbb{R}^{n \times n}.$$

Das Skalarprodukt ist der Spezialfall $M = I$.

Erhalten vs. verloren

- Linearität in beiden Argumenten ✓.
- Symmetrie: nur falls $M = M^\top$.
- Positive Definitheit: nur falls M pos. semidefinit.

Attention-Scores sind genau dies mit $M = W_Q W_K^\top$ — im Allgemeinen nicht symmetrisch (gerichtete Attention).

Permutationsmatrizen



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Permutationsmatrizen

Eine **Permutationsmatrix** $P \in \{0, 1\}^{T \times T}$ besitzt genau eine 1 pro Zeile und pro Spalte.

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \quad P \begin{pmatrix} x_1^\top \\ x_2^\top \\ x_3^\top \end{pmatrix} = \begin{pmatrix} x_2^\top \\ x_3^\top \\ x_1^\top \end{pmatrix}.$$

Zwei Fakten, die wir wiederholt verwenden

- PX permutiert die **Zeilen** von X (und $XP^\top$ permutiert die Spalten).
- $P^\top P = I$: Permutationsmatrizen sind orthogonal.

Sie treten im Softmax-Äquivarianz-Lemma (nächster Abschnitt) und im Attention-Äquivarianz-Lemma (später, bei der Positional Encoding) auf.



Punktweise Nichtlinearitäten



Punktweise Nichtlinearitäten I — Sigmoid

Lineare und affine Abbildungen komponieren zu linear / affin. Damit ein gestapeltes Netz *mehr* berechnen kann, brauchen wir **Nichtlinearitäten**.

Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \quad \sigma : \mathbb{R} \rightarrow (0, 1).$$

Eigenschaften:

- Glatt, beschränkt, monoton wachsend.
- Sättigt: $\sigma'(x) \rightarrow 0$ für $|x| \rightarrow \infty$ » verschwindende Gradienten in tiefen Stapeln.
- Historischer Standard; weitgehend durch ReLU und seine glatten Varianten verdrängt.

Punktweise angewandt: $\sigma(z)$ für $z \in \mathbb{R}^n$ bedeutet σ auf jede Koordinate einzeln angewandt.

Punktweise Nichtlinearitäten II — ReLU und die „LU“-Familie

Alle werden **punktweise** auf Vektoren und Matrizen angewandt.

- **ReLU:** $\max(0, x)$. Schnell; keine Sättigung auf der positiven Seite. Risiko: *dying ReLU* (Neuronen bleiben bei 0 hängen).
- **Leaky ReLU:** $\max(\alpha x, x)$ für kleines $\alpha > 0$. Behebt das Dying-ReLU-Problem.
- **ELU / SELU:** glatter negativer Schwanz, $\alpha(e^x - 1)$ für $x < 0$.
- **GeLU:** $x \cdot \Phi(x)$, Φ = Verteilungsfunktion der Standardnormalverteilung. Glattes ReLU. Standard in BERT, GPT-2/3, ursprünglichem Transformer-FFN.
- **SiLU / Swish:** $x \cdot \sigma(x)$. In LLaMA u. a., meist als *SwiGLU* verpackt.

Was sie gemeinsam haben

Alle sind nichtlineare skalare Funktionen, elementweise angewandt; sie brechen die Linearität, sodass gestapelte Schichten mehr als nur eine einzige affine Abbildung ausdrücken können.



Softmax

Softmax — Definition und geometrisches Bild

Für $z \in \mathbb{R}^n$:

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}}.$$

Bildet $\mathbb{R}^n$ auf das offene Simplex $\{p_i > 0, \sum_i p_i = 1\}$ ab — glatte Näherung des $\arg \max$.

Drei Eigenschaften zum Merken

- **Verschiebungsinvarianz:** $\text{softmax}(z + c\mathbf{1}) = \text{softmax}(z)$ — nur Differenzen zählen (Trick: $\max_i z_i$ abziehen).
- **Temperatur:** $\text{softmax}(z/\tau) \gg$ für $\tau \rightarrow 0$ harter $\arg \max$; für $\tau \rightarrow \infty$ Gleichverteilung.
- **Äquivariant, nicht invariant:** Permutation von z permutiert $\text{softmax}(z)$ ebenso.

Keine punktweise Nichtlinearität — die Koordinaten sind über den Nenner gekoppelt.

Zeilenweises Softmax auf Matrizen

Für $X \in \mathbb{R}^{T \times T}$ sei $\text{softmax}(X)$ **zeilenweise** definiert:

$$\text{softmax}(X)_{ij} = \frac{e^{X_{ij}}}{\sum_{k=1}^T e^{X_{ik}}}.$$

Jede Zeile der Ausgabe ist eine Wahrscheinlichkeitsverteilung über die T Spaltenindizes.

Genau so in der Attention verwendet

Jede Zeile der Score-Matrix S wird zur Verteilung einer Query über die Keys.

Zeile vs. Spalte. Die Wahl der Achse ist entscheidend:

- Zeilenweises Softmax: $\text{softmax}(PX) = P \text{softmax}(X)$ für eine Permutation P (Zeilen sind unabhängig).
- Ein spaltenweises Softmax wäre ein völlig anderer Operator.

Lemma — Permutationskonjugation passiert das Softmax

Aussage. Für jede Permutationsmatrix $P \in \{0, 1\}^{T \times T}$ und jedes $X \in \mathbb{R}^{T \times T}$ gilt

$$\text{softmax}(PXP^\top) = P \text{softmax}(X) P^\top.$$

Beweis.

- **Zeilenhälfte:** $\text{softmax}(PX) = P \text{softmax}(X)$ — Softmax wirkt zeilenweise, und P sortiert lediglich um, welche Zeile wohin geht.
- **Spaltenhälfte:** $\text{softmax}(XP^\top) = \text{softmax}(X) P^\top$ — der Zeilennenner $\sum_k e^{X_{ik}}$ ist invariant unter Spaltenumordnung, sodass das Softmax jeder Zeile lediglich seine Einträge umetikettiert bekommt.
- **Komposition:** $\text{softmax}(PXP^\top) = P \text{softmax}(XP^\top) = P \text{softmax}(X) P^\top.$ ■

Warum das wichtig ist

Genau diesen Schritt überspringt der Beweis der Attention-Äquivarianz häufig. Mit diesem Lemma ist die Kette $A' = \text{softmax}(PSP^\top) = PAP^\top$ vollständig rigoros.

Self-Attention



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Zeilen- vs. spaltenweise Speicherung der Tokens

Zeilenweise: $X \in \mathbb{R}^{T \times d}$ — jeder Token ist eine Zeile.

$$X = \begin{pmatrix} x_1^\top \\ \vdots \\ x_T^\top \end{pmatrix}$$

Am natürlichsten für Erklärungen auf Implementierungsebene.

Spaltenweise: $\tilde{X} \in \mathbb{R}^{d \times T}$ — jeder Token ist eine Spalte.

$$\tilde{X} = (x_1, \dots, x_T) = X^\top.$$

Mathematisch äquivalent; sie unterscheiden sich nur darin, wo die Transponierte sitzt.

Query, Key, Value — woher die Namen kommen

Die Terminologie stammt aus dem **Information Retrieval** bzw. Dictionary-Lookup.

Klassischer Lookup: $d[\text{query}]$

Eine *Query* (Anfrage) wird mit gespeicherten *Keys* (Schlüsseln) verglichen; bei einem Treffer wird der zugehörige *Value* (Wert) zurückgegeben.

Self-Attention weicht diesen Lookup auf:

- jeder Key erhält ein *Gewicht* (nicht nur ein einzelner Treffer)
- die Gewichte werden per Softmax normalisiert
- die Ausgabe ist ein *gewichteter Mittelwert der Values*

In der *Self-Attention* stammen Q , K , V alle aus demselben X — jeder Token fragt: „*welche anderen Tokens sind Keys, die zu meiner Query passen?*“

Token $\leftrightarrow$ Zeile $\leftrightarrow$ Query / Key / Value

Jede Zeile von X, Q, K, V ist derselbe Token, betrachtet durch eine andere gelernte Linse.

Zeile	Token	Embedding $h_i^\top$	Query $h_i^\top W_Q$	Key $h_i^\top W_K$	Value $h_i^\top W_V$
1	king	$h_1^\top$	$q_1^\top$	$k_1^\top$	$v_1^\top$
2	man	$h_2^\top$	$q_2^\top$	$k_2^\top$	$v_2^\top$
3	woman	$h_3^\top$	$q_3^\top$	$k_3^\top$	$v_3^\top$
4	queen	$h_4^\top$	$q_4^\top$	$k_4^\top$	$v_4^\top$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
T	letzter Token	$h_T^\top$	$q_T^\top$	$k_T^\top$	$v_T^\top$

Zeile i von Q ist, was Token i *anfragt*; Zeile i von K ist, was er *anbietet*; Zeile i von V ist, was er *liefert*.

Self-Attention: Projektionen (zeilenweise)

Sei $X \in \mathbb{R}^{T \times d}$. Wähle Projektionsmatrizen

$$W_Q \in \mathbb{R}^{d \times d_k}, \quad W_K \in \mathbb{R}^{d \times d_k}, \quad W_V \in \mathbb{R}^{d \times d_v}.$$

Definiere

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V.$$

Rechtsmultiplikation transformiert jede Token-*Zeile* in eine Query-/Key-/Value-Zeile. Es ist also $Q, K \in \mathbb{R}^{T \times d_k}$ und $V \in \mathbb{R}^{T \times d_v}$.

Score-Eintrag als Skalarprodukt

Die Zeilen der projizierten Matrizen sind

$$\text{Zeile } i \text{ von } Q = h_i^\top W_Q, \quad \text{Zeile } j \text{ von } K = h_j^\top W_K.$$

Die Zerlegung

$$S_{ij} = q_i^\top k_j = (h_i^\top W_Q)(h_j^\top W_K)^\top = h_i^\top W_Q W_K^\top h_j = \langle W_Q^\top h_i, W_K^\top h_j \rangle.$$

Ein **Skalarprodukt** zwischen dem *gelernten Query-Vektor* $W_Q^\top h_i$ und dem *gelernten Key-Vektor* $W_K^\top h_j$, beide in $\mathbb{R}^{d_k}$.

(Der mittlere Schritt nutzt die Transponierten-Regel $(AB)^\top = B^\top A^\top$.)

Geometrische Lesart des Scores

Im Query-/Key-Unterraum $\mathbb{R}^{d_k}$ gilt

$$q_i^\top k_j = \|q_i\|_2 \|k_j\|_2 \cos \theta_{ij}.$$

Zwei geometrische Lesarten

1. **Projektion.** $q_i^\top k_j$ ist die Länge der Projektion von q_i auf die Richtung von k_j , skaliert mit $\|k_j\|_2$.
2. **Betrag $\times$ Ausrichtung.** Groß genau dann, wenn *beide* Vektoren lang sind *und* ähnlich zeigen.

„*Token i attended auf Token j* “ bedeutet: im gelernten Q/K-Unterraum hat die Query von i einen großen Anteil in Richtung des Keys von j .

Die vollständige Score-Matrix (Übersicht)

$$Q = \begin{pmatrix} h_1^\top W_Q \\ \vdots \\ h_T^\top W_Q \end{pmatrix}, \quad K^\top = (W_K^\top h_1 \quad \cdots \quad W_K^\top h_T).$$

$$QK^\top = \begin{pmatrix} h_1^\top W_Q W_K^\top h_1 & \cdots & h_1^\top W_Q W_K^\top h_T \\ \vdots & \ddots & \vdots \\ h_T^\top W_Q W_K^\top h_1 & \cdots & h_T^\top W_Q W_K^\top h_T \end{pmatrix}.$$

Jeder Eintrag ist ein Skalarprodukt im Query-/Key-Unterraum. Die Score-Matrix $S = QK^\top$ hängt *ausschließlich vom Input X* selbst ab.

Scores als gelernte Bilinearform

Sei $M = W_Q W_K^\top \in \mathbb{R}^{d \times d}$. Dann

$$S_{ij} = h_i^\top M h_j.$$

Die Score-Matrix ist eine *gelernte Bilinearform* auf dem Embedding-Raum.

- **Asymmetrie.** Im Allgemeinen $M \neq M^\top$, also $S_{ij} \neq S_{ji}$ — Attention ist *gerichtet*. („Die Katze jagt die Maus“: die Beziehung zwischen *Katze* und *Maus* ist nicht symmetrisch.)
- **Niedriger Rang.** $\text{rang}(M) \leq d_k$. Die Zerlegung von M in zwei $d \times d_k$ -Faktoren ist ein expliziter Rang-Engpass.
- **Warum zwei Matrizen, nicht eine.** W_Q und W_K können *unterschiedliche* Embeddings für „Fragen“ und „Befragt-Werden“ lernen — die Asymmetrie ist ein *Feature*, kein Preis.

Self-Attention: Scores und Ausgabe

Score-Matrix.

$$S = QK^{\top} \in \mathbb{R}^{T \times T}, \quad S_{ij} = q_i^{\top} k_j.$$

Alle paarweisen Query-Key-Skalarprodukte.

Scaled Dot-Product Attention.

$$A = \text{softmax}\left(\frac{QK^{\top}}{\sqrt{d_k}}\right).$$

Softmax wird **zeilenweise** angewandt; jede Zeile von A summiert sich zu 1.

Ausgabe.

$$\text{Attention}(X) = AV \in \mathbb{R}^{T \times d_v}.$$

Zeile i : $(AV)_{i,\cdot} = \sum_{j=1}^T A_{ij} v_j^{\top}$ — ein gewichteter Mittelwert der Value-Zeilen.

Warum der Faktor $1/\sqrt{d_k}$?

Varianzargument

Haben Query- und Key-Einträge eine Varianz der Ordnung 1, so hat $q_i^\top k_j$ eine Varianz der Ordnung d_k .

Skalierung mit $\sqrt{d_k}$:

- hält die Logits in einem numerisch stabilen Bereich,
- verhindert, dass das Softmax zu früh zu scharf wird.

Dieselbe Attention in spaltenweiser Notation

Sind die Tokens Spalten, $\tilde{X} \in \mathbb{R}^{d \times T}$, so werden die Projektionen **links** angewandt:

$$\tilde{Q} = W_Q \tilde{X}, \quad \tilde{K} = W_K \tilde{X}, \quad \tilde{V} = W_V \tilde{X}.$$

Die Score-Matrix wird zu

$$\tilde{Q}^\top \tilde{K} \in \mathbb{R}^{T \times T}.$$

Die beiden Notationen unterscheiden sich nur in der Platzierung der Transponierten; die zugrundeliegende Mathematik ist identisch.

Multi-Head Attention



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Von einem Head zu vielen — Multi-Head Attention

Multi-Head Attention (MHA) ist **kein grundlegend neuer Mechanismus**. Es ist die einfache Attention aus §4, parallel ausgeführt auf *disjunkten Teilen des Embedding-Raums*, mit einer abschließenden Misch-Projektion, die die Teile wieder zu einer d -dimensionalen Ausgabe zusammensetzt.

Drei strukturelle Änderungen gegenüber Single-Head

1. **Zerschneide den Embedding-Raum** in H Blöcke der Größe $d_k = d/H$.
2. **Q/K/V-Projektionen pro Head**, jeweils mit Form $d \times d_k$.
3. **Neue Output-Projektion** $W_O \in \mathbb{R}^{d \times d}$, die die Head-Ausgaben mischt.

Konkret (GPT-2 small): $d = 768$, $H = 12$, $d_k = 64$.

Aufteilung und Projektionen pro Head

Jeder Head $h = 1, \dots, H$ erhält eigene *schmalere* Projektionsmatrizen:

$$W_Q^{(\ell,h)}, W_K^{(\ell,h)}, W_V^{(\ell,h)} \in \mathbb{R}^{d \times d_k}.$$

Q/K/V-Parameterbudget bleibt unverändert

$$3 \cdot H \cdot d \cdot d_k = 3 \cdot H \cdot d \cdot (d/H) = 3d^2.$$

Multi-Head ist **nicht** „teurer pro Projektion“ — es ist *dasselbe* Parameterbudget, umverteilt auf H parallele Unterräume.

Der Skalierungsnenner ändert sich auf das pro Head verwendete $\sqrt{d_k}$ — jeder Head arbeitet nun in einem d_k -dimensionalen Unterraum, nicht im vollen d .

Die Output-Projektion W_O

Nachdem die Attention pro Head ausgeführt und die H pro Head erzeugten Ausgaben zurück auf Dimension d konkateniert wurden, mischt eine zusätzliche Matrix sie:

$$W_O^{(\ell)} \in \mathbb{R}^{d \times d}.$$

Das einzige „neue“ Element

W_O hat in der Single-Head Attention **kein Analogon**. Sie ist der Grund, weshalb Multi-Head Attention $4d^2$ Parameter hat, wo Single-Head $3d^2$ hatte: das zusätzliche d^2 ist genau W_O .

Parameterzählung: Single-Head vs. Multi-Head

Variante	Q / K / V	W_O	Gesamt pro Block
Single-Head	$3d^2$	—	$3d^2$
Multi-Head, H Heads, $d_k = d/H$	$3H \cdot d \cdot d_k = 3d^2$	d^2	$4d^2$

Die Anzahl der Q/K/V-Parameter ist in beiden gleich; Multi-Head fügt genau die Misch-Matrix W_O obendrauf.

Spielzeugbeispiel

$d = 6$, $H = 2$, $d_k = 3$: $4 \cdot 36 = 144$ MHA-Parameter pro Block. Unabhängig von T .

Warum diese Umstrukturierung wichtig ist

Jeder Head erzeugt seine eigene $T \times T$ -Routing-Matrix $A^{(h)}$.

Spezialisierung über Heads hinweg

Das Training animiert verschiedene Heads dazu, unterschiedliche Beziehungen zu entdecken: Koreferenz, Syntax, Position, Klammerübereinstimmung, ...

- Single-Head Attention kann mehrere gleichzeitige Routing-Muster nicht darstellen.
- Multi-Head bekommt das im Q/K/V-Rechenaufwand *im Wesentlichen umsonst*, zum Preis der zusätzlichen W_O -Projektion.

Der Forward-Pass ist die natürliche Verallgemeinerung der Single-Head-Variante: dieselben fünf Schritte (projizieren, scoren, skalieren, Softmax, gewichten), nun H -mal parallel, mit Konkatenation + Projektion am Ende.

MHA Forward-Pass: Projektionen pro Head

Sei $X \in \mathbb{R}^{T \times d}$ der Input zu Block ℓ . Für jeden Head $h = 1, \dots, H$:

$$Q^{(h)} = X W_Q^{(\ell, h)}, \quad K^{(h)} = X W_K^{(\ell, h)}, \quad V^{(h)} = X W_V^{(\ell, h)}.$$

Shape-Check.

$$\underbrace{(T \times d)}_X \cdot \underbrace{(d \times d_k)}_W \rightarrow (T \times d_k).$$

Es ist also $Q^{(h)}, K^{(h)}, V^{(h)} \in \mathbb{R}^{T \times d_k}$. ✓

MHA Forward-Pass: Scores, Softmax, gewichtete Values

Attention-Scores (pro Head):

$$S^{(h)} = \frac{Q^{(h)} (K^{(h)})^\top}{\sqrt{d_k}} \in \mathbb{R}^{T \times T}.$$

Softmax (zeilenweise; jede Zeile von $A^{(h)}$ summiert sich zu 1):

$$A^{(h)} = \text{softmax}(S^{(h)}) \in \mathbb{R}^{T \times T}.$$

Gewichtete Values:

$$Z^{(h)} = A^{(h)} V^{(h)} \in \mathbb{R}^{T \times d_k}.$$

Dieselben fünf Schritte wie bei Single-Head, H -mal parallel ausgeführt.

$T \times T$ ist eine Aktivierung, kein Gewicht

$S^{(h)}$ und $A^{(h)}$ haben Form $T \times T$ — sie sind aber **Aktivierungen**, keine Gewichte.

- Werden für jeden Input frisch berechnet; nie als Parameter gespeichert.
- Keine lernbare Matrix im gesamten Transformer hat die Form $T \times T$.
- Das T^2 in „Transformer skalieren quadratisch in der Sequenzlänge“ bezieht sich auf *Rechenzeit und Speicher* im Forward-Pass, nicht auf die Parameterzahl.

Heads konkatenieren und projizieren

Konkateniere die H pro Head erzeugten Ausgaben entlang der Feature-Dimension:

$$Z = [Z^{(1)} \mid Z^{(2)} \mid \dots \mid Z^{(H)}] \in \mathbb{R}^{T \times Hd_k} = \mathbb{R}^{T \times d}.$$

Mische die Teile mit der Output-Projektion:

$$U = Z W_O^{(\ell)} \in \mathbb{R}^{T \times d}.$$

Shape-Check: $(T \times d) \cdot (d \times d) \rightarrow (T \times d)$. ✓

W_O ist es, was Information, die im Unterraum eines Heads entdeckt wurde, in andere Koordinaten der Ausgabe einfließen lässt — nach diesem Schritt sind die Teile nicht mehr disjunkt.

MHA-Parameterzahl pro Block

- H Heads $\times$ 3 Projektionsmatrizen $\times (d \times d_k)$ Einträge je:

$$3H \cdot d \cdot d_k = 3H \cdot d \cdot (d/H) = 3d^2.$$

- Plus $W_O \in \mathbb{R}^{d \times d}$ mit d^2 Beiträgen.

Gesamt: $4d^2$ pro Block

Unabhängig von der Sequenzlänge T .

Spielzeug ($d = 6$, $H = 2$, $d_k = 3$): $4 \cdot 36 = 144$ MHA-Parameter pro Block.



Feed-Forward-Netzwerk



Der Feed-Forward-Block — wo das Wissen lebt

Nach der Multi-Head Attention hat jeder Token Information aus anderen Tokens aufgenommen. Attention hat ihre Aufgabe des *Sammelns* erledigt — viel *Nachdenken* war aber noch nicht dabei.

Attention produziert lediglich gewichtete Summen bestehender Vektoren. „Ein Pronomen, das Information über sein Bezugswort aufgenommen hat“ in „das Konzept des Tieres, auf das es verweist, angereichert mit Fakten über Tiere“ zu verwandeln — das ist die Aufgabe des FFN.

Attention vs. FFN, in einem Atemzug

Attention sammelt Information über Positionen hinweg.

FFN verarbeitet sie, pro Token, isoliert.

Das meiste, was wir „das Wissen des Modells“ nennen, lebt in den FFN-Gewichten.

FFN: Parameter und Expansionsfaktor

Ein zweischichtiges MLP, unabhängig auf jeden Token angewandt. Für jeden Block ℓ :

$$W_1^{(\ell)} \in \mathbb{R}^{d \times 4d}, \quad b_1^{(\ell)} \in \mathbb{R}^{4d},$$

$$W_2^{(\ell)} \in \mathbb{R}^{4d \times d}, \quad b_2^{(\ell)} \in \mathbb{R}^d.$$

Der Expansionsfaktor

Die erste Matrix expandiert die Dimension um den Faktor 4 (in neueren Architekturen manchmal 8): den Token auf $4\times$ seiner Größe aufblasen, eine Nichtlinearität anwenden, zurück komprimieren.

GPT-3 ($d = 12,288$): die Hidden-Dimension im FFN ist $4d = 49,152$.

FFN: Forward-Pass

Zeilenweise angewandt. Für jeden Token $h_t \in \mathbb{R}^d$:

$$\text{FFN}(h_t) = \varphi(h_t \cdot W_1 + b_1) \cdot W_2 + b_2 \in \mathbb{R}^d,$$

wobei φ eine punktweise Nichtlinearität ist (ReLU, GeLU, SwiGLU — die Wahl beeinflusst die Formen nicht).

Auf der gesamten Matrix (mit b_1, b_2 über die Zeilen gebroadcastet):

$$\text{FFN}(H) = \varphi(H \cdot W_1 + b_1) \cdot W_2 + b_2 \in \mathbb{R}^{T \times d}.$$

Shape-Check.

- $H \cdot W_1: (T \times d) \cdot (d \times 4d) \rightarrow (T \times 4d)$
- addiere b_1 , wende φ an: gleiche Form
- $\dots \cdot W_2: (T \times 4d) \cdot (4d \times d) \rightarrow (T \times d) \checkmark$

Positionsweise: kein Mischen über Tokens hinweg

W_1, W_2, b_1, b_2 sind *dieselben* Matrizen, die auf jede Zeile angewandt werden.

Die entscheidende Beobachtung

Das FFN ist eine **positionsweise** Abbildung: es behandelt jeden der T Tokens unabhängig, parallel. Nichts im FFN leitet Information über Token-Positionen hinweg.

Dies ist der mathematische Grund, weshalb Attention als eigener Mechanismus existieren muss — sie ist die *einzig*e Teilschicht, die Zeilen miteinander mischt.

Attention » Routing zwischen Tokens

FFN » Berechnung pro Token

FFN als Key-Value-Speicher

Eine nützliche Interpretation des FFN: ein pro Token wirkender **Key-Value-Speicher**.

- W_1 bewertet den Token gegen $4d$ interne „Keys“.
- φ steuert, welche Keys aktiv sind — eine weiche Maske, die sagt: „diese passen“.
- W_2 erzeugt die Ausgabe als gewichtete Kombination von $4d$ gespeicherten „Values“.

Routing vs. Speicherung

Attention ist Routing — sie braucht relativ wenige Parameter.

FFN ist Speicherung — es braucht viele.

Wo die Parameter liegen: Attention vs. FFN

Teilschicht	Parameter pro Block	Spielzeug ($d = 6$)
MHA ($Q/K/V + W_O$)	$4d^2$	144
FFN (W_1, W_2, b_1, b_2)	$8d^2 + 5d$	318

FFN hält rund zwei Drittel des Budgets pro Block

Für $d = 768$: MHA $\approx 2,4$ M, FFN $\approx 4,7$ M Parameter pro Block.

Über GPT-3s 96 Schichten: *zig Milliarden* Parameter, die der Transformation pro Token gewidmet sind.

Das meiste Modellwissen lebt in den FFN-Gewichten, nicht in der Attention.

Der vollständige Transformer-Block



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Den Block zusammensetzen (Pre-LN + Residuals)

Jeder Block wendet Layer-Normalisierung an, dann eine Teilschicht, dann ein Residual:

$$H' = H_{\ell-1} + \text{MHA}(\text{LN}_1(H_{\ell-1})),$$

$$H_{\ell} = H' + \text{FFN}(\text{LN}_2(H')).$$

Form-Erhaltung

Jede Zwischengröße liegt in $\mathbb{R}^{T \times d}$. Die Form bleibt durchgängig erhalten — Blöcke lassen sich beliebig stapeln.

Über die Tiefe des Modells: Attention » FFN » Attention » FFN » ...
sammeln, dann denken, dann sammeln, dann denken.

Gesamtparameter pro Block: $\approx 12d^2$

Summe über die Teilschichten:

$$\underbrace{4d^2}_{\text{MHA}} + \underbrace{4d}_{\text{zwei LNs}} + \underbrace{8d^2 + 5d}_{\text{FFN}} = 12d^2 + 9d \approx 12d^2.$$

Unabhängig von der Sequenzlänge T

Spielzeug ($d = 6$): $12 \cdot 36 + 9 \cdot 6 = 432 + 54 = 486$ Parameter pro Block.

Etwa $\frac{1}{3}$ Attention, $\frac{2}{3}$ FFN. L Blöcke übereinander geben insgesamt $\approx 12Ld^2$ Parameter im Transformer-Stack.

Positional Encoding — Motivation



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Der blinde Fleck — Attention sieht eine Bag-of-Words

Nimm den Satz `the cat sat on the mat` und durchmische die Wortreihenfolge.

Was ändert sich in der Attention?

Strukturell nichts. Die Menge der paarweisen Scores $q_i^\top k_j$ ist *identisch*, nur umetikettiert. Jeder Score hängt davon ab, **welche zwei Tokens** beteiligt sind — nie davon, **wo sie stehen**.

- Der Input der Attention ist mathematisch eine *ungeordnete Menge* von Token-Vektoren.
- Ohne Unterstützung ist der Transformer nur ein sehr teures Bag-of-Words-Modell.

Wir formulieren das gleich als Lemma — und beheben es danach.

Lemma — Self-Attention ist permutationsäquvariant

Rückblick. Self-Attention ist die Funktion

$$\text{Attention} : \mathbb{R}^{T \times d} \longrightarrow \mathbb{R}^{T \times d_v}$$

parametrisiert durch feste (gelernte) Matrizen $W_Q, W_K \in \mathbb{R}^{d \times d_k}$ und $W_V \in \mathbb{R}^{d \times d_v}$,
definiert durch

$$\text{Attention}(X) = \text{softmax}\left(\frac{(XW_Q)(XW_K)^\top}{\sqrt{d_k}}\right) XW_V,$$

mit **zeilenweise** angewandtem Softmax: $\text{softmax}(M)_{ij} = \exp(M_{ij}) / \sum_k \exp(M_{ik})$.

Lemma (Permutationsäquvarianz)

Für jede Permutationsmatrix $P \in \{0, 1\}^{T \times T}$ und jedes $X \in \mathbb{R}^{T \times d}$ gilt

$$\text{Attention}(PX) = P \cdot \text{Attention}(X) \in \mathbb{R}^{T \times d_v}.$$

Beweis des Lemmas

Setze $Q = XW_Q$, $K = XW_K$, $V = XW_V$, $S = QK^\top / \sqrt{d_k}$, $A = \text{softmax}(S)$, sodass $\text{Attention}(X) = AV$.

Schritt 1 (Projektionen). Matrixmultiplikation ist assoziativ, daher

$$Q' = (PX)W_Q = P(XW_Q) = PQ \in \mathbb{R}^{T \times d_k},$$

und analog $K' = PK \in \mathbb{R}^{T \times d_k}$, $V' = PV \in \mathbb{R}^{T \times d_v}$.

Schritt 2 (Scores). Mit $(AB)^\top = B^\top A^\top$:

$$S' = \frac{Q'(K')^\top}{\sqrt{d_k}} = \frac{(PQ)(PK)^\top}{\sqrt{d_k}} = P \cdot \frac{QK^\top}{\sqrt{d_k}} \cdot P^\top = PSP^\top \in \mathbb{R}^{T \times T}.$$

Schritt 3 (Softmax). Rechtsmultiplikation mit $P^\top$ permutiert die Spalten von S , der Zeilensummen-Nenner im Softmax hängt aber nicht von der Spaltenreihenfolge ab. Folglich gilt $\text{softmax}(SP^\top) = \text{softmax}(S) P^\top$. Linksmultiplikation mit P permutiert die Zeilen und passiert das Softmax trivial. Zusammen:

$$A' = \text{softmax}(PSP^\top) = P \text{softmax}(S) P^\top = PAP^\top \in \mathbb{R}^{T \times T}.$$

Schritt 4 (Ausgabe). Mit $P^\top P = I$ (P ist orthogonal):

$$\text{Attention}(PX) = A'V' = (PAP^\top)(PV) = PA(P^\top P)V = PAV = P \cdot \text{Attention}(X). \blacksquare$$

Was Äquivarianz für die Position bedeutet

Jede Ausgabezeile ist eine Funktion nur davon, *welche* Tokens vorhanden sind, niemals ihrer Reihenfolge.

Die angereicherte Darstellung des Tokens `cat`

Identischer Inhalt, ob die Sequenz
the cat sat, sat the cat oder cat sat the lautet
— nur sein *Zeilenindex* in der Ausgabematrix ändert sich.

- Die Ausgabe jedes Tokens hängt nur von der **Multimenge** der anderen Tokens ab.
- LayerNorm und FFN sind positionsweise, also ist der *gesamte* Block permutationsäquivalent.
- Blöcke zu stapeln behebt das nicht — Äquivarianz komponiert.

Konsequenz: Ohne zusätzliches Signal kann das Modell

„John liebt Mary“ und „Mary liebt John“ buchstäblich nicht unterscheiden.

Die Lösung — Position in die Tokens einspeisen

Wir wollen die Attention selbst nicht ändern — die Äquivarianz schenkt uns Parallelität, Längenverallgemeinerung und Verzicht auf Rekurrenz.

Der Trick

Ändere den **Input**, sodass derselbe Token an verschiedenen Positionen zu einem anderen Vektor wird:

$$X \rightsquigarrow X + E_{\text{pos}}, \quad E_{\text{pos}} \in \mathbb{R}^{T \times d}.$$

Im Allgemeinen ist nun

$$\text{Attention}(P(X + E_{\text{pos}})) \neq P \cdot \text{Attention}(X + E_{\text{pos}}),$$

weil $PE_{\text{pos}} \neq E_{\text{pos}}$. Die Äquivarianz wird **bewusst gebrochen**.

Wie wählen wir E_{pos} ? — nächster Abschnitt.

Positional Encoding — Verfahren



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Drei Standardansätze

Sei $X \in \mathbb{R}^{T \times d}$ die Token-Embedding-Matrix.

Verfahren	Wirkungsort	Parameter	Verwendet von
Sinusförmig	$H_0 = X + P_{\text{sin}}$ (berechnet)	0	„Attention Is All You Need“
Gelernt absolut	$H_0 = X + E_{\text{pos}}[0 : T]$	$n_{\text{max}} \cdot d$	GPT-2, GPT-3
RoPE	rotiere Q, K in der Attention	0	LLaMA, moderne Frontier-Modelle

Die Sichtweise ist entscheidend

Zwei davon modifizieren den **Input** ($H_0 = X + P$); RoPE *nicht* — es lässt $H_0 = X$ unangetastet und speist die Position später, innerhalb der Attention-Berechnung, ein. Es gibt keine einzige „ P -Matrix“, die alle drei abdeckt.

(3a) Sinusförmige Positionscodierung (Vaswani et al., 2017)

Die Positionsmatrix wird **berechnet**, nicht gelernt. Für Position $t = 0, \dots, T - 1$ und Frequenzindex $k = 0, \dots, d/2 - 1$:

$$P[t, 2k] = \sin\left(\frac{t}{10000^{2k/d}}\right), \quad P[t, 2k + 1] = \cos\left(\frac{t}{10000^{2k/d}}\right).$$

Dann ist $H_0 = X + P \in \mathbb{R}^{T \times d}$.

Drei angenehme Eigenschaften

- **Null Parameter.** P ist ein festes mathematisches Objekt.
- **Beliebige Länge.** Für ein neues T einfach neu auswerten — nichts zu redimensionieren.
- **Relative Positionen entstehen.** $P[t + \Delta, :]$ ist eine lineare Funktion von $P[t, :]$ (feste Rotation, nur von Δ abhängig), sodass das Modell relative Regeln aus absoluten Codierungen lernen kann.

(3b) Gelernt absolut (GPT-2 / GPT-3-Stil)

Eine zweite Lookup-Tabelle, indiziert nach **Position** statt nach Token-Identität:

$$E_{\text{pos}} \in \mathbb{R}^{n_{\text{max}} \times d}.$$

Für Eingabelänge $T \leq n_{\text{max}}$:

$$P = E_{\text{pos}}[0 : T, :] \in \mathbb{R}^{T \times d}, \quad H_0 = X + P.$$

Das eine n_{max} in der gesamten Architektur

Dies ist das *einzige* Gewicht, dessen Form von der maximalen Sequenzlänge abhängt.

- Parameterzahl: $n_{\text{max}} \cdot d$.
- Kürzerer Input: Präfix der Zeilen; der Rest bekommt in diesem Durchlauf null Gradient.
- Längerer Input: **keine Zeile zum Nachschlagen** — das Modell kann ihn buchstäblich nicht verarbeiten.

(3c) RoPE — Rotary Position Embeddings (Su et al., 2021)

Addiere nichts zu X . Stattdessen **rotiere** Q und K **innerhalb der Attention** um einen positionsabhängigen Winkel.

Für jeden aufeinanderfolgenden 2-Koordinaten-Block $(2k, 2k + 1)$ der Zeile q_t (und k_t) wende die 2D-Rotation an:

$$R(t \cdot \omega_k) = \begin{pmatrix} \cos(t\omega_k) & -\sin(t\omega_k) \\ \sin(t\omega_k) & \cos(t\omega_k) \end{pmatrix}, \quad \omega_k = \frac{1}{10000^{2k/d}}.$$

Warum es zum modernen Standard wurde

- **Null gelernte Parameter**, keine Matrix mit $n_{\max}$ -Form.
- Der Score $q_t^\top k_s$ hängt nur vom *relativen* Versatz $t - s$ ab — relative Codierung, nicht absolute.
- Extrapoliert über die Trainingslänge hinaus (mit nachlassender Qualität; abgemildert durch NTK-/YaRN-Skalierung).



Verwendet von LLaMA, Mistral, Qwen und den meisten aktuellen Frontier-Modellen.

Verbindung zum Äquivarianz-Lemma

Erinnerung: reine Self-Attention erfüllt $\text{Attention}(PX) = P \cdot \text{Attention}(X)$.

Positionsinformation entsteht erst durch *Brechen* dieser Symmetrie — die drei Verfahren tun das an unterschiedlichen Stellen.

Bruch auf der Input-Seite (sinusförmig, gelernt absolut)

Attention bleibt unverändert; der Input wird zu $X + P_{\text{pos}}$. Das Lemma gilt weiterhin — nur ist der Input nicht mehr permutationssymmetrisch, da $PP_{\text{pos}} \neq P_{\text{pos}}$.

Bruch auf der Attention-Seite (RoPE)

Attention selbst wird modifiziert: rotiert man Q, K um positionsabhängige Winkel, wird der Score $\tilde{S}_{ts} = q_t^\top R(s - t) k_s$ explizit positionsabhängig. Das Lemma beschrieb die *unveränderte* Funktion und greift nicht mehr.

Gleiches Ziel, zwei Bruchstellen.

Stand der Technik heute (2024–2026)

RoPE hat sich für autoregressive LLMs durchgesetzt

Praktisch jedes moderne Open-Weights-Modell: LLaMA 1/2/3, Mistral / Mixtral, Qwen 2/3, DeepSeek V2/V3, Yi, Gemma, Falcon, PaLM 2.

Die GPT-4-Familie ist unbestätigt, soll aber RoPE oder eine nahe Variante einsetzen.

Die anderen Verfahren kurz:

- **Gelernt absolut** (GPT-2/3, Original-BERT) — Altlast; harte $n_{\max}$ -Decke.
- **Sinusförmig** — hat die Skalierungsära nicht überlebt.
- **ALiBi** (BLOOM, MPT-7B) — anderes relatives Schema, aus dem Fokus.

Warum RoPE gewonnen hat: null Parameter, relative Positionierung, Längen-Extrapolation (mit NTK-/YaRN-Skalierung), saubere Komposition mit dem Attention-Skalarprodukt.

Vorbehalt: an der Long-Context-Front ($> 32k$ Tokens) braucht auch RoPE Skalierungstricks.



Ähnlichkeit, Orthogonalität, Projektionen



Skalarprodukt vs. Kosinus-Ähnlichkeit (cosine similarity)

$$x^\top y = \|x\|_2 \|y\|_2 \cos \theta.$$

Das Skalarprodukt hängt sowohl vom **Betrag** als auch von der **Richtung** ab.
Die *Kosinus-Ähnlichkeit* (engl. *cosine similarity*) entfernt den Betrag:

$$\text{cosim}(x, y) = \frac{x^\top y}{\|x\|_2 \|y\|_2}.$$

Skaleninvarianz. Für $\alpha, \beta > 0$:

$$\text{cosim}(\alpha x, \beta y) = \text{cosim}(x, y).$$

Die euklidische Distanz ist *nicht* skaleninvariant.

Hinweis: In deutschsprachiger Fachliteratur ist auch der englische Begriff *cosine similarity* weit verbreitet — beide Bezeichnungen sind üblich.

Kosinus-Ähnlichkeit (cosine similarity): geometrisches Bild

Mit normierten Versionen $\hat{x} = x/\|x\|_2$ und $\hat{y} = y/\|y\|_2$ gilt

$$\text{cosim}(x, y) = \hat{x}^\top \hat{y}.$$

Die Kosinus-Ähnlichkeit ist nichts anderes als das Skalarprodukt nach ℓ_2 -Normierung.

Auf der Einheitskugel ($\|x\|_2 = \|y\|_2 = 1$):

$$\|x - y\|_2^2 = 2 - 2 \text{cosim}(x, y).$$

Für normierte Vektoren liefern euklidische Distanz und Kosinus-Ähnlichkeit *äquivalente Reihenfolgen*.

Warum Skalarprodukt und nicht Kosinus?

Self-Attention nutzt $q_i^\top k_j$, *nicht* die Kosinus-Ähnlichkeit (cosine similarity) — und der Grund ist nicht bloß Rechenkostenersparnis.

Der Kosinus verschenkt einen gelernten Freiheitsgrad

W_Q und W_K lernen *sowohl* die **Richtung** als auch den **Betrag** von q_i und k_j . Eine erzwungene Einheitsnorm verwirft das.

Normen können sinnvoll codieren:

- $\|q_i\|$ — wie stark Token i irgendwohin *attended* will
- $\|k_j\|$ — wie laut Token j sich *anbietet*

Die numerische Stabilität des rohen Skalarprodukts wird durch Softmax plus die Skalierung $1/\sqrt{d_k}$ erledigt — dafür braucht es keine Kosinus-Normierung.

Warum nicht die euklidische Distanz?

Die Distanz erschiene ebenfalls natürlich, und

$$\|q_i - k_j\|_2^2 = \|q_i\|_2^2 + \|k_j\|_2^2 - 2 q_i^\top k_j.$$

Die Distanz reduziert sich also auf das Skalarprodukt plus zwei Selbstnormen. Warum nutzt man sie nicht?

- **Komposition.** $q_i^\top k_j = h_i^\top W_Q W_K^\top h_j$ ist eine saubere Bilinearform in den gelernten Gewichten; eine quadrierte Distanz verliert diese Form.
- **Softmax-Verträglichkeit.** Softmax erwartet *größer = relevanter*. Die Distanz ist für *weniger* ähnliche Paare größer, und ihr Offset $\|q_i\|^2 + \|k_j\|^2$ variiert pro Token.
- **Hochdimensionales Verhalten.** Bei großem d_k dominieren tendenziell die Selbstnorm-Terme und überdecken den paarweisen Term.

Warum keine allgemeinen ℓ_r -Normen?

$\|x - y\|_r$ für $r \neq 2$ ist eine legitime Distanz — aber sie beantwortet eine andere Frage.

- $\|x - y\|_2$: wie weit sind die Punkte auseinander?
- $\cos \theta$: wie ausgerichtet sind die Richtungen?

Warum $r = 2$ besonders ist:

$$\|x\|_2^2 = x^\top x.$$

Die euklidische Geometrie entspringt einem Skalarprodukt, das Zugang zu Winkeln, Orthogonalität, Projektionen und Gram-Matrizen liefert. Allgemeine ℓ_r -Normen interagieren nicht so natürlich mit der Matrixmultiplikation.

Orthogonalität und Projektionen

Orthogonalität. $x \perp y$ genau dann, wenn $x^\top y = 0$ — keine Komponente in der Richtung des jeweils anderen.

Skalarprojektion von x auf u :

$$\frac{x^\top u}{\|u\|_2}.$$

Vektorprojektion:

$$\text{proj}_u(x) = \frac{x^\top u}{u^\top u} u.$$

Skalarprodukte messen, wie stark ein Vektor in Richtung eines anderen zeigt — ein großes Query-Key-Skalarprodukt bedeutet starke Ausrichtung.

Zusammenfassung und Kernaussagen



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Zusammenfassung: Kernidentitäten

Matrix mal Vektor

$$Ax = \sum_{j=1}^n x_j a_j = \begin{pmatrix} r_1^\top x \\ \vdots \\ r_m^\top x \end{pmatrix}$$

Zeilenvektor mal Matrix

$$u^\top A = \sum_{i=1}^m u_i r_i^\top$$

Matrix mal Matrix

$$AB = (Ab_1, \dots, Ab_p) = \begin{pmatrix} r_1^\top B \\ \vdots \\ r_m^\top B \end{pmatrix}$$

Gram-Matrix: $(XX^\top)_{ij} = x_i^\top x_j$.

Zusammenfassung: Self-Attention-Pipeline

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

$$S = QK^\top$$

$$\text{Attention}(X) = \text{softmax}\left(\frac{S}{\sqrt{d_k}}\right) V$$

In einem Atemzug

Rechtsmultiplikation » Zeilenwirkung » Q/K/V-Projektionen » Gram-artiges $QK^\top$ »
Softmax + Value-Mittelung.

Fallstricke, auf die man achten sollte

- Zeilen und Spalten zu verwechseln
- die Dimensionsverträglichkeit zu vergessen
- zu denken, AB und BA seien meist dasselbe
- zu vergessen, dass $(AB)^\top = B^\top A^\top$, nicht $A^\top B^\top$
- Skalarprodukt-Ähnlichkeit mit euklidischer Distanz zu verwechseln
- anzunehmen, Kosinus-Ähnlichkeit (cosine similarity) und euklidische Distanz seien **ohne** Normierung äquivalent
- zu denken, Attention-Scores seien Kosinus-Ähnlichkeiten — es sind skalierte Skalarprodukte

Kernaussagen in einer Zeile

- Linksmultiplikation transformiert **Spalten**.
- Rechtsmultiplikation transformiert **Zeilen**.
- QK^T speichert alle Query-Key-Skalarprodukte.
- Kosinus vergleicht Richtung; Skalarprodukt vergleicht Richtung *und* Betrag.
- Self-Attention ist lineare Algebra plus Softmax.