# Transformer gibt es nicht nur im Kino

Jonathan Arns
*Hochschule Mannheim*
Fakultät für Informatik
Paul-Wittsack-Str. 10
68163 Mannheim
jonathan.arns@stud.hs-mannheim.de

***Abstract*—abstract**

## I. Einleitung

Natural Language Processing (NLP), die Verarbeitung Menschlicher Sprache, ist ein Anwendungbereich für Machine Learning, der bereits erheblich von deep neural networks (DNN) profitiert hat. Die zwei dominanten Arten von DNN Architekturen dabei waren lange Zeit recurrent neural networks (RNN) und convolutional neural networks (CNN). [1]

2017 stellten Vaswani et al. [2] mit dem Transformer eine neue DNN Architektur vor, die seitdem unter Anderem große Aufmerksamkeit durch die erfolgreiche Verwendung in OpenAIs GPT-2 und GPT-3 Modellen erlangte.

## II. Sequence to Sequence Learning

Traditionell waren DNNs trotz ihrer hohen Flexibilität und Effektivität für viele Aufgaben beschränkt auf Probleme, deren Eingaben und Ausgaben sich sinvoll in Vektoren mit fester Länge codieren lassen, da neuronale Netze generell eine feste Anzahl an Eingabe- und Ausgabeneuronen haben. Das ist zwar für viele Klassifizierungsprobleme und in der Bildverarbeitung kein Problem, sehr wohl aber für beispielsweise NLP, da die Länge von Texten im Vorfeld nicht immer bekannt ist. [3]

Dieses Problem wird mittels Sequence to Sequence (Seq2Seq) Learning gelöst, mit dessen Hilfe beliebig lange Sequenzen von Elementen als Eingabe in vollkommen andere Sequenzen, anderer Länge und aus anderen Elementen, transformiert werden können. [3]

Die meisten Seq2Seq Modelle bestehen aus einem Encoder und einem Decoder [2]. Der Encoder codiert die Eingabe Sequenz in einer höher-dimensionale Vektorrepräsentation, die dann vom Decoder in eine Ausgabe Sequenz umgewandelt wird. Die Encoder und Decoder selbst sind in der Regel RNNs, beispielsweise mit der LSTM Architektur.

## III. Attention

Attention ist der wichtigste Mechanismus des Transformers. Attention erlaubt dem Transformer beispielsweise, Sätze nicht nur Wort für Wort zu übersetzen, sondern Zusammenhänge zu erkennen und grammatikalisch korrekte Sätze zu bilden. Abstrakt ist eine Attention Funktion eine Abbildung von einer Query und einer Menge von Key-Value Paaren zu einer Ausgabe. In einem Übersetzungskontext könnte das Wort, welches gerade übersetzt werden soll, die Query sein und alle Wörter in dem zu übersetzenden Satz die Keys und Values. Das Ergebnis signalisiert dann, welche Wörter bei der Übersetzung der Query zusätzlich besonders in betracht gezogen werden. Diese spezielle Konstellation, in der die Query, Keys und Values aus der gleichen Sequenz stammen, nennt sich Self-Attention. Diese kommt auch im Transformer zum Einsatz und ist beispielhaft in Abb. 1 visualisiert.

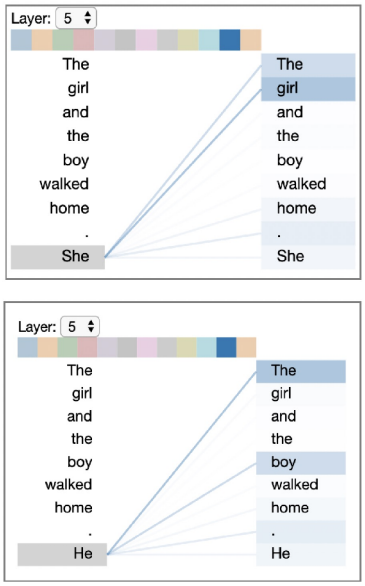


Fig. 1. Visualisierung von Self-Attention in GPT-2. Verbindungen von der Query (grau hinterlegt) zu den Keys zeigen Attention, dunklere Verbindungen sind höher gewichtet. Hier ist sichtbar, wie mittels Attention Referenzen auf frühere Wörter erkannt werden können. [4]

### A. Scaled dotproduct attention

Es gibt grundlegend zwei verschiedene Attention Funktionen, additive Attention und dot-product Attention. Der Transformer verwendet dot-product Attention, da diese trotz gleicher theoretischer Komplexität in der Praxis aufgrund von hoch optimiertem Matix-Multiplikations Code deutlich schneller ist. Abb. 2 zeigt den Aufbau eines scaled dot-product attention Moduls im Transformer. Die Berechnung der Attention Matrix erfolgt mit folgender Formel:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_K}})V \qquad (1)$$

Dabei ist $Q$ die Matrix der Query Vektoren, $K$ die Matrix der Key Vektoren und $V$ die Matrix der Value Vektoren. $Q$ ist entgegen der Intuition eine Matrix, da die Attention in der Praxis mittels Matrizenoperationen für mehrere Queries auf einmal berechnet wird.

Softmax, die normalisierte Exponentialfunktion, normalisiert alle Elemente eines Vektors auf Werte zwischen 0 und 1, so dass die Summe aller Werte 1 ergibt. Da die Exponentialfunktion verwendet wird, nähert sich das größte Element des Vektors 1 an, während sich, außer in sehr knappen Fällen, alle anderen Elemente 0 annähern. Sei $x$ ein Vektor mit den Elementen $x_1 ... x_n$, dann ist Softmax wie folgt definiert. [5]

$$Softmax(x)_j = \frac{e^{\beta x_j}}{\sum_{i=1}^{n} e^{\beta x_i}}, \beta \in \mathbb{R} \wedge \beta > 0 \qquad (2)$$

Da die Softmax Funktion in der Praxis nicht gut mit sehr großen Werten skaliert, werden die Scores mit der Wurzel der Dimension der Keys $\sqrt{d_k}$ skaliert, bevor Softmax berechnet wird. [2]

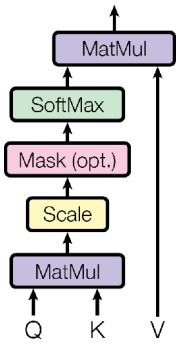


Fig. 2. Für die Berechnung der Scaled Dotproduct-Attention werden die Matrizen $Q$, $K$ und $V$ benötigt. Die Queries $Q$ und die Keys $K$ haben die Dimension $d_k$, die Values $V$ haben die Dimension $d_v$. Um die Attention zu berechnen, wird mittels Matrixmultiplikation der Queries mit den transponierten Keys pro Query Vektor ein Score für jeden Key gebildet. Diese Scores werden mit $\sqrt{d_k}$ skaliert und mit einer Softmax Funktion pro Query zu Gewichten für die Values normalisiert. Um die Attention Matrix zu erhalten werden die Values mit den ermittelten Gewichten multipliziert. Vor der Berechnung der Softmax Funktion werden optional Werte maskiert, die noch unerlaubte Verbindungen darstellen. Zu maskierende Werte werden auf $-\infty$ gesetzt. [2]

### B. Single-Head Attention

Bei der Definition der dot-product Attention bis hier her fehlt noch ein essentieller Teil des Attention Mechanismus. Es fehlt jede Art von Parametern, die beim Training des Modells erlernt werden und die den Eingaben eine Gewichtung verleihen. Die Matrizen $Q$, $K$ und $V$, die die Attention Funktion als Eingaben erwartet, sind nicht einfach eine direkte Repräsentation der Eingaben des Modells, sondern gewichtete Projektionen von diesen, die zur Berechnung der Attention gebildet werden. Hierzu werden die Parameter Matrizen $W^Q$, $W^K$ und $W^V$ benötigt, mit denen die Eingaben einer Attention Schicht multipliziert werden. $W^Q$, $W^K$ und $W^V$ werden zufällig initialisiert und beim Training des Modells erlernt. In einem Single-Head Attention Modul haben die Projektionen die Dimension $d_{model}$, die der Dimension der codierten Eingaben des Modells entspricht. [2]

### C. Multi-Head Attention

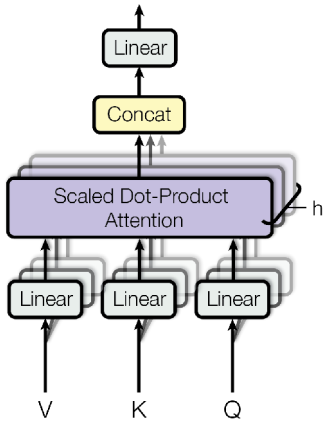


Fig. 3. Multi-Head Attention [2]

Der Transformer verwendet anstatt Single-Head Attention Multi-Head Attention. Der Unterschied besteht darin, dass es mehrere Parameter Matrizen $W_1^Q$ ... $W_h^Q$, $W_1^K$ ... $W_h^K$ und $W_1^V$ ... $W_h^V$ gibt, mit denen die Eingaben linear projeziert werden. So werden $h$ so genannte Attention Heads gebildet. Die Projektionen haben anstatt $d_{model}$ die Dimensionen $d_k$, $d_k$ und $d_v$, wobei $d_k$ und $d_v$ kleiner sein können, als $d_{model}$. Für jede Projektion wird dann einzeln die Attention berechnet, die Ergebnisse werden concatiniert und mit einer weiteren Parameter Matrix $W^O$ in die $d_{model}$-dimensionale Ausgabe des Multi-Head Attention Moduls projeziert. Dieser Aufbau ist in Abb. 3 dargestellt und entspricht der Formel:

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \qquad (3)$$

$$\text{mit } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \qquad (4)$$

Wobei die Projektionen mit $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ und $W^O \in \mathbb{R}^{hd_v \times d_{model}}$ definiert sind. [2]

Yin et al. [2] verwenden in ihrem Modell $h = 8$ und $d_k = d_v = \frac{d_{model}}{h}$, somit haben sie 8 kleinere Attention Heads bei ähnlichem Rechenaufwand im Vergleich zu einem $d_{model}$-dimensionalen Single-Head Attention Modul.

### D. Attention Beispiel

## IV. Modell Architektur

Fig. 4 zeigt die Architektur des Transformers. Der Transformer ist auto-regressiv, das bedeutet, dass jedes generierte Element der Ausgabe Sequenz ebenfalls als Eingabe Verwendet wird, um das nächste Element zu generieren [2]. Um eine vollständige Ausgabe zu erzeugen muss der Transformer also mehrfach ausgeführt werden.

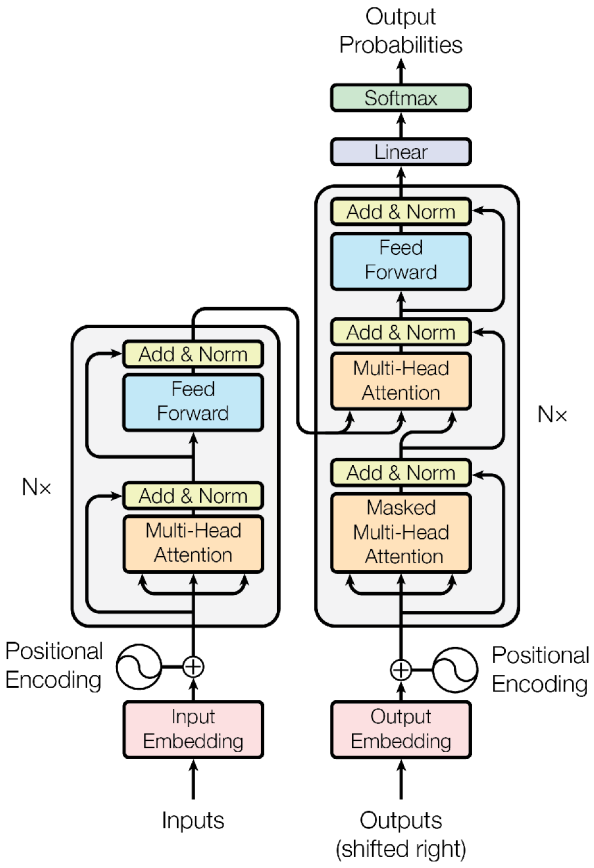


Fig. 4. Der Encoder (links) besteht aus einer Reihe von N identischen Schichten mit jeweils zwei Unterschichten. Die Eingaben einer Encoder Schicht fließen zuerst durch eine self-attention Schicht und dann durch ein feed-forward Netz.
Der Decoder (rechts) besteht auch aus einer Reihe von N identischen Schichten, jedoch mit jeweils drei Unterschichten. Die erste ist ebenfalls eine self-attention Schicht, deren Eingaben allerdings maskiert werden, um dem Modell keine Eingaben zu zeigen, die es nicht sehen sollte. Danach kommt eine Encoder-Decoder attention Schicht, die als Keys und Values die Ausgaben des Decoders und als Queries die Ausgaben der zuvor liegenden self-attention Schicht erhält. Zuletzt liegt wie beim Encoder ein feed-forward Netz.
Nach jeder Attention und feed-forward Schicht wird die Summe dessen Ausgaben und Eingaben zusammen normalisiert. Die normalisierten Werte dienen jeweils als Eingabe für die nächste Schicht.
Im NLP Anwendungen müssen die Eingaben für das Modell als erstes in Vektoren eingebettet werden. Da in Seq2Seq Modellen die Reihenfolge der Eingaben eine Rolle spielt, das Modell die Eingaben aber alle gemeinsam erhält, muss außerdem für jedes Element die Position in der Eingabe codiert werden. Der Transformer generiert für jedes Element einen Vektor, der dessen Position codiert, dazu stehen verschiedene Algorithmen, sowohl fix, als auch trainierbar, zur Verfügung.
Um ein Ergebnis zu erhalten, werden die Ausgaben des Decoders in einen one-hot codierten Vektor projeziert und mit der Softmax Funktion normalisiert. Der erzeugte Vektor enthält für jedes Element des Ausgabealphabets die Wahrscheinlichkeit, dass es das nächste Element in der Ausgabe Sequenz ist. [2]

### A. Anwendungen von Attention

Der Transformer verwendet den Attention Mechanismus auf drei verschiedene Arten.

*1) Encoder Self-Attention:* In Self-Attention Schichten werden die gleichen Werte als Keys, Values und Queries verwendet. Durch die Self-Attention Schichten können für jedes Element im Encoder Abhängigkeiten zu jedem Element der Eingabe, beziehungsweise der Ausgabe der vorherigen Encoder Schicht, dargestellt werden.

*2) Decoder Self-Attention:* Durch die Self-Attention Schichten im Decoder können für jedes Element im Decoder ebenfalls Abhängigkeiten zu jedem Element der Ausgabe bis hin zum aktuellen Element dargestellt werden. Verbindungen zu Elementen nach dem jeweiligen Element sind nicht erlaubt, damit Ausgaben tatsächlich nur auf der Eingabe und bereits generierten Elementen basieren. Unerlaubte Verbindung werden durch eine Maskierung von Werten in der Attention Funktion verhindert.

*3) Encoder-Decoder Attention:* Die Keys und Values der Encoder-Decoder Attention Schichten im Decoder sind die Ausgaben des Encoders. Als Queries werden jeweils die Ausgaben der zuvor liegenden Self-Attention Schicht im Decoder verwendet. Damit können Abhängigkeiten von jedem Element im Decoder zu jedem Element der Encoder Ausgabe dargestellt werden. Hier besteht also die Schnittstelle vom Encoder zum Decoder und entsprechend die Beziehung zwischen den Eingabe und Ausgabe Sequenzen.

### B. Feed-Forward Netze

## V. Training

Die Modell Parameter werden zufällig initialisiert.

## VI. Fazit

Das ist ein Fazit

## References


[1] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. “Comparative Study of CNN and RNN for Natural Language Processing”. In: *arXiv e-prints* (2017). arXiv: 1702.01923.

[2] Ashish Vaswani et al. “Attention is All You Need”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. ISBN: 9781510860964.

[3] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. “Sequence to Sequence Learning with Neural Networks”. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS’14. Montreal, Canada: MIT Press, 2014, pp. 3104–3112.

[4] Jesse Vig. “A Multiscale Visualization of Attention in the Transformer Model”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 37–42. DOI: 10.18653/v1/P19-3007. URL: https://www.aclweb.org/anthology/P19-3007.

[5] Steven Gold, Anand Rangarajan, et al. “Softmax to softassign: Neural network algorithms for combinatorial optimization”. In: *Journal of Artificial Neural Networks* 2.4 (1996), pp. 381–399.