Transformer und Multi-Head Attention

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

***Zusammenfassung*—Natural Language Processing ist ein wichtiger Anwendungbereich für Machine Learning, der von deep neural networks dominiert ist. Der Transformer wurde 2017 als neue deep learning Architektur für genau diesen Bereich vorgestellt und seitdem erfolgreich in mehreren sehr großen Sprach-Modellen wie GPT-2 und BERT verwendet. Dieser Artikel stellt die Transformer Architektur dar und veranschaulicht im Detail den Attention Mechanismus, der dem Transformer zu Grunde liegt.**

I. Einleitung

Natural Language Processing (NLP), die Verarbeitung menschlicher Sprache, hat bereits erheblich von deep neural networks (DNN) profitiert. 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 Modellen [3] erlangte.

Dieser Artikel erklärt die Transformer Architektur auf anschauliche Art und Weise. Dazu wird zunächst das Konzept von Sequence to Sequence Learning zusammengefasst, welches der Transformer aufgreift. Bevor die vollständige Modell Architektur dargestellt wird, liegt ein besonderer Fokus auf Attention, dem zentralen Mechanismus des Transformers. Diese wird auf ihre Bestandteile heruntergebrochen und von Grund auf erklärt und visualisiert.

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. 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. [4]

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 Abbildung 1 visualisiert.

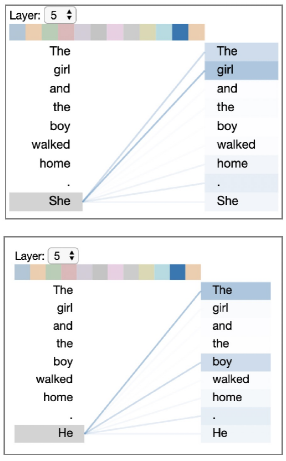

Abbildung 1. Visualisierung von Self-Attention in GPT-2. Verbindungen von der Query (grau hinterlegt) zu den Keys zeigen Attention, dunklere Verbindungen sind höher gewichtet. Hier ist sichtbar, wie mittels Attention Referenzen auf frühere Wörter erkannt werden können. [5]

A. Scaled Dotproduct Attention

Es gibt grundlegend zwei verschiedene Attention Funktionen, additive Attention und dotproduct Attention. Der Trans-

former verwendet dotproduct Attention, da diese trotz gleicher theoretischer Komplexität in der Praxis aufgrund von hoch optimiertem Matizenmultiplikations-Code deutlich schneller ist. Abbildung 2 zeigt den Aufbau eines scaled dotproduct Attention Moduls im Transformer. Die Berechnung der Attention Matrix erfolgt mit folgender Formel:

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

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. [6]

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

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]

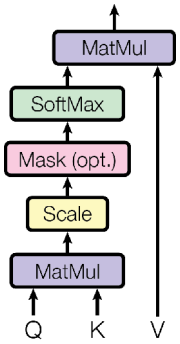

Abbildung 2. Für die Berechnung der scaled dotproduct Attention werden die Matrizen Q, K und V benötigt. Die Queries Q und die Keys K haben die Dimension d_k, die Values V haben die Dimension d_v. Um die Attention zu berechnen, wird mittels Matrixmultiplikation der Queries mit den transponierten Keys 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 dotproduct 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

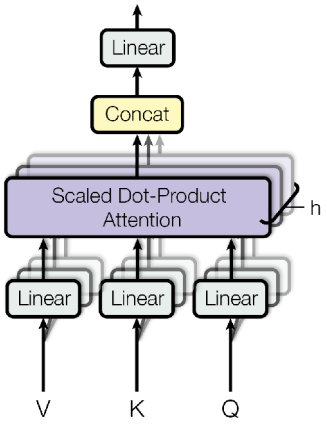

Abbildung 3. Für Multi-Head Attention werden die Eingaben h mal linear projeziert. Auf den Projektionen wird jeweils die scaled dotproduct Attention berechnet. Die Ergebnisse werden concatiniert und linear in eine d_{model}-dimensionale Ausgabe projeziert. [2]

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 Abbildung 3 dargestellt und entspricht der Formel:

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

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

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. Multi-Head Attention Visualisierung

In den Abbildungen 4, 5, 6, 7 wird die Funktionsweise eines Multi-Head Self-Attention Moduls visualisiert dargestellt, um die Transformation der verschiedenen Matrizen zu veranschaulichen.

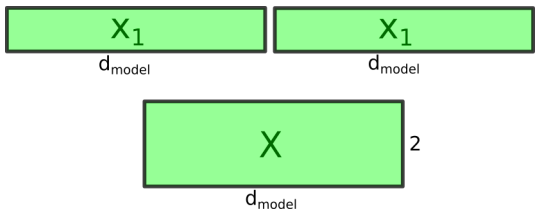

Abbildung 4. Sei die Eingabe eine Sequenz aus zwei Elementen, die in den Vektoren x_1 und x_2 mit je d_{model} Dimensionen codiert sind. Die Vektoren bilden zusammen die Matrix X, die als Eingabe für das Multi-Head Self-Attention Modul verwendet wird. (Abbildung nach [7])

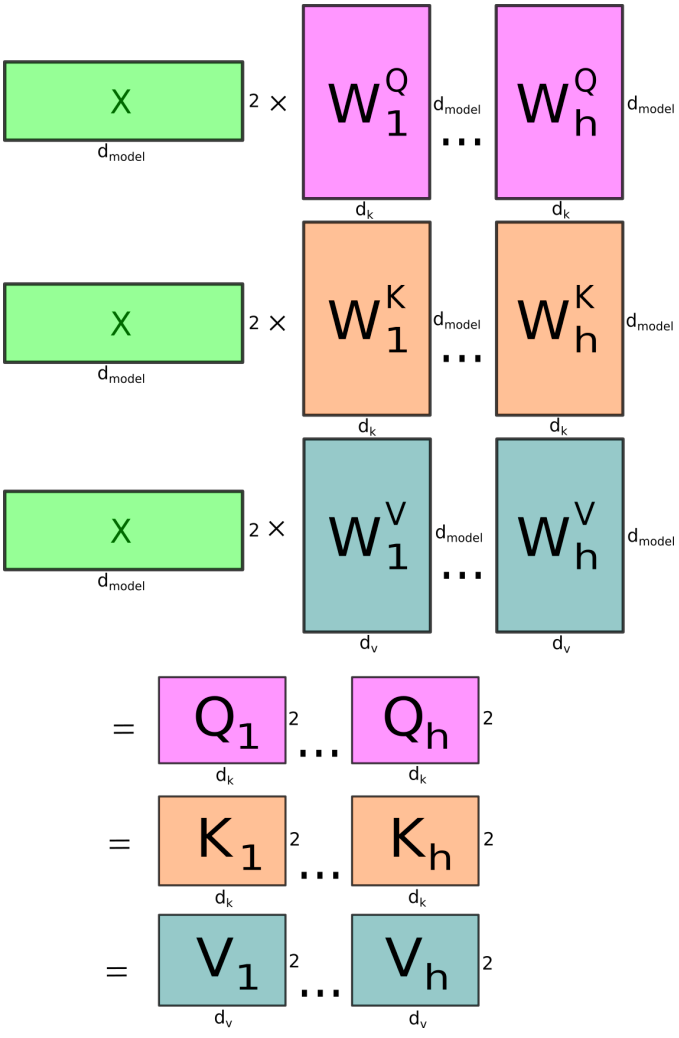

Abbildung 5. Da es sich um Multi-Head Attention handelt, wird X mit den Parameter Matrizen $W_1^Q ... W_h^V$ in h (Anzahl Heads) verschiedene Query, Key und Value Matrizen projeziert. Jede Matrix Q_i beinhaltet in diesem Fall zwei Queries. (Abbildung nach [7])

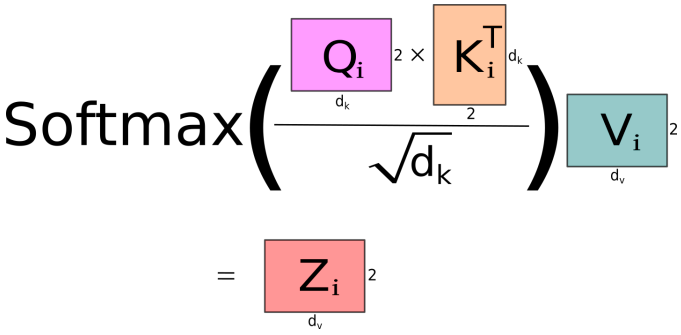

Abbildung 6. Für jeden Attention Head wird die scaled dotproduct Attention berechnet, das Ergebnis ist eine Matrix Z_i, die pro Query in Q_i einen Attention Vektor enthält. (Abbildung nach [7])

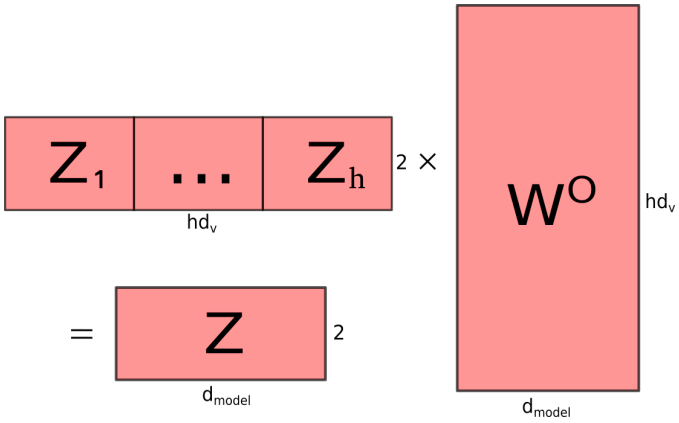

Abbildung 7. Um die Ergebnisse der Einzelnen Attention Heads zusammen zu führen, werden die Matrizen Z_1 bis Z_h concatiniert und mit der trainierten Parametermatrix W^O multipliziert. Das Ergebnis ist eine Matrix Z, die die Informationen aus allen Attention Heads kombiniert und as Ausgabe des Multi-Head Attention Moduls dient. (Abbildung nach [7])

IV. Modell Architektur

Abbildung 8 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.

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,

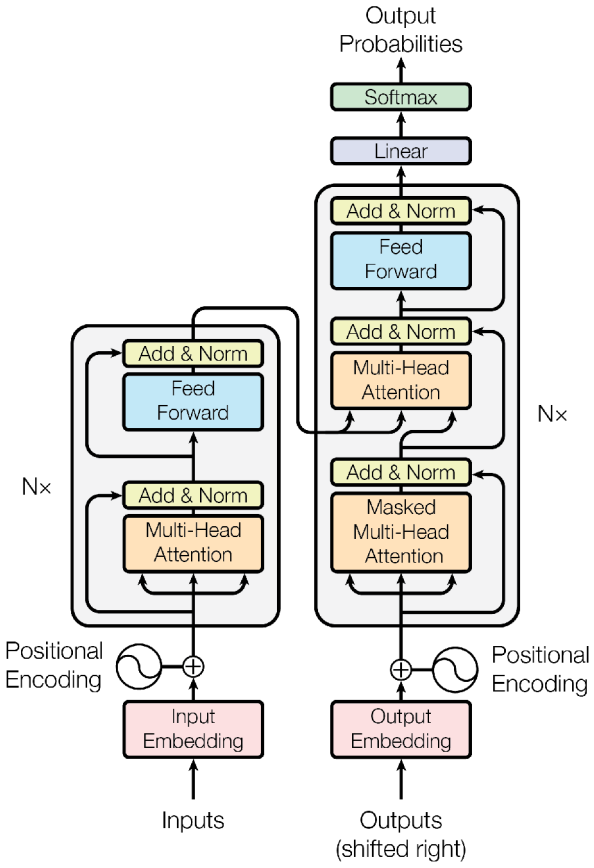

Abbildung 8. Der Encoder (links) besteht aus einer Reihe von N identischen Schichten mit jeweils zwei Unterschichten. Die Eingaben einer Encoder Schicht fließen zuerst durch eine self-Attention Schicht und dann durch ein feedforward Netz.
Der Decoder (rechts) besteht auch aus einer Reihe von N identischen 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 feedforward Netz.
Nach jeder Attention und feedforward 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 Logit-Vektor transformiert 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]

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. Feedforward Netze

Das feedforward Netz (FFN) in jeder Encoder und Decoder Schicht besteht aus Eingabe- und Ausgabeschichten mit $d_{model} = 512$ Dimensionen und einer verdeckten Schicht mit $d_{ff} = 2048$ Dimensionen. Auf den Ausgaben der ersten Schicht wird die ReLU Aktivierungsfunktion angewendet, bevor diese in die verdeckte Schicht fließen. ReLU ist 0 für negative Werte und gibt die Eingabe für positive Werte unverändert zurück, das entspricht der Formel $ReLU(x) = max(0, x)$ [8]. Somit bilden die feedforward Netze folgende lineare Transformation ab:

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{5}$$

Wobei W_i die erlernten Gewichte und b_i die erlernten Werte zur neuronen-spezifischen Verzerrung sind. Diese Transformation wird auf allen Eingabe-Elementen der FFN Schicht identisch und unabhängig voneinander durchgeführt und kann somit problemslos parallelisiert werden. Die einzelnen FFN Schichten haben allerdings ihre eigenen Gewichte und Verzerrungen. [2]

V. Training

Transformer können wie andere neuronale Netze auch mit unterschiedlichen Optimierungsfunktionen trainiert werden. Vaswani et al. [2] verwenden für ihren originalen Transformer die Adam Optimierungsfunktion, während OpenAIs GPT Modelle mit Stochastic Gradient Descent trainiert wurden [3].

Als Trainingsdaten werden Paare von Sequenzen benötigt, die jeweils eine gewünschte Transformation darstellen, beispielsweise Englische Sätze und die Deutschen Sätze, zu denen die Englischen Sätze übersetzt werden sollen. Da der Transformer seine eigenen Ausgaben als weitere Eingaben erwartet, um jeweils das nächste Element zu generieren, gibt es pro Sequenz-Paar mehrere Trainingsschritte, in denen der Transformer jeweils zusätzlich das nächste Element der erwarteten Ausgabe in der Decoder Eingabe erhält. Für jeden Trainingsschritt erhält der Transformer die komplette Eingabe Sequenz als Eingabe für den Encoder und für den Decoder die erwartete Ausgabe um ein Element nach rechts verschoben, damit die Eingabe das erwartete Element noch nicht enthält. Hat die erwartete Ausgabe Sequenz nur ein Element, erhält der Decoder nur ein Symbol, welches den Anfang einer Sequenz signalisiert.

VI. Verwendung

Ähnlich wie im Training sind bei der Verwendung des Transformers mehrere Durchläufe notwendig, um eine vollständige Sequenz zu generieren. Mit jedem Schritt erhält der Transformer seine eigene Ausgabe als Eingabe für den Decoder, um das nächste Element zu generieren. Zusätzlich wird nicht einfach in jedem Schritt das Element mit der höchsten Vorhersage-Wahrscheinlichkeit in der Ausgabe der letzten Softmax Schicht ausgegeben. Stattdessen verwendet der Transformer Beam Search [2], um mehrere mögliche Elemente zu behalten und erst später das beste auszuwählen.

Beam Search speichert jederzeit die besten n von m Sequenzen, verwirft die schlechtesten $m - n$ Sequenzen und sucht in der aktuell besten Sequenz weiter [9]. Zur Bewertung der Sequenzen wird im Transformer die Summe der Kostenfunktion aller Elemente verwendet. Vaswani et al. [2] verwenden für ihr Modell eine Beam Breite von $n = 4$. Mittels Beam Search können auch mehrere vollständige Ausgabe Sequenzen gefunden werden, somit können beispielsweise Übersetzungsmodelle mehrere mögliche Übersetzungen für einen Text generieren.

VII. Ausblick

Die Transformer Architektur, wie sie ursprünglich entwickelt und in diesem Artikel vorgestellt wurde, verwendet einen Encoder und einen Decoder, die beide jeweils aus einer Reihe von identisch aufgebauten Schichten bestehen. Transformer wurden seitdem für verschiedene andere Probleme verwendet. Dafür wurden neue, Transformer-basierte Architekturen entwickelt, die nur noch einen Stapel von Decoder oder Encoder Modulen verwenden und ohne das Gegenstück auskommen. OpenAIs GPT Modelle verwenden beispielsweise nur eine Reihe von Decoder Modulen, anstelle der zweigeteilten Architektur [3].

VIII. Fazit

Die größte Innovation des Transformers war die Verwendung von Self-Attention als Ersatz für komplexere rekurrente Schichten. Die Vorteile sind bessere Performance, geringerer Rechenaufwand beim Training des Modells und möglicherweise besser nachvollziehbare Ergebnisse. [2]

Angesichts der mächtigen neuen Sprach-Modelle wie GPT-2 [10] und BERT [11], die seit der Vorstellung des Transformers auf dessen Basis entstanden sind, lässt sich von einer kleinen Revolution der NLP Domäne sprechen. Heute ermöglichen solche vortrainierten, generellen Sprach-Modelle, mit geringem Aufwand und kleinen Datensätzen, mittels Fine-Tuning eines der genannten Modelle, kompetente NLP Systeme für eine breite Menge von Aufgaben zu schaffen. [12]

Literatur

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

[2] Ashish Vaswani u. a. "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, S. 6000–6010. ISBN: 9781510860964.

[3] Alec Radford, Karthik Narasimhan, Tim Salimans und Ilya Sutskever. "Improving language understanding by generative pre-training". In: *OpenAI* (2018).

[4] Ilya Sutskever, Oriol Vinyals und 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, S. 3104–3112.

[5] 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, Juli 2019, S. 37–42. DOI: 10.18653/v1/P19-3007. URL: https://www.aclweb.org/anthology/P19-3007.

[6] Steven Gold, Anand Rangarajan u. a. "Softmax to softassign: Neural network algorithms for combinatorial optimization". In: *Journal of Artificial Neural Networks* 2.4 (1996), S. 381–399.

[7] Jay Allamar. *The Illustrated Transformer*. http://jalammar.github.io/illustrated-transformer/ (aufgerufen am 26.02.2021). 2018.

[8] Prajit Ramachandran, Barret Zoph und Quoc V Le. "Searching for activation functions". In: *arXiv preprint arXiv:1710.05941* (2017).

[9] David Furcy und Sven Koenig. "Limited Discrepancy Beam Search". In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence*. IJCAI'05. Edinburgh, Scotland: Morgan Kaufmann Publishers Inc., 2005, S. 125–131.

[10] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei und Ilya Sutskever. "Language Models are Unsupervised Multitask Learners". In: *OpenAI* (2019).

[11] Ian Tenney, Dipanjan Das und Ellie Pavlick. "BERT rediscovers the classical NLP pipeline". In: *arXiv preprint arXiv:1905.05950* (2019).

[12] Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai und Xuanjing Huang. "Pre-trained models for natural language processing: A survey". In: *Science China Technological Sciences* (2020), S. 1–26.