Einführung in Transformer-Architekturen von Neuronalen Netzen

Ausarbeitung Integrationsseminar

im Rahmen der Prüfung zum

Bachelor of Science (B.Sc.)

des Studienganges Wirtschaftsinformatik Software Engineering

an der Dualen Hochschule Baden-Württemberg Mannheim

von

Alexander Meinecke

Abgabedatum:	27. Januar 2025
Bearbeitungszeitraum:	Dezember 2024 - 27. Januar 2025
Matrikelnummer, Kurs:	1522347, WWI22SEB
Ausbildungsfirma:	SAP SE Dietmar-Hopp-Allee 16 69190 Walldorf, Deutschland
Gutachter:	Alexander Lütke

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich meine Ausarbeitung Integrationsseminar mit dem Thema:

Einführung in Transformer-Architekturen von Neuronalen Netzen

gemäß § 5 der „Studien- und Prüfungsordnung DHBW Technik“ vom 29. September 2017 selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Mannheim, den 16. Dezember 2024

Meinecke, Alexander

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

BEHG	Brennstoffemissionshandelsgesetz
CBAM	Carbon Border Adjustment Mechanism (dt. Grenzausgleichsmechanismus)
DEHSt	Deutsche Emissionshandelsstelle
EPA	United States Environmental Protection Agency (dt. Umweltschutzbehörde der Vereinigten Staaten)
ETS	Emission Trading System (dt. Emissionshandelssystem)
ICAP	International Carbon Action Partnership (dt. Internationale Partnerschaft für Emissionshandel)
MSR	Market Stability Reserve (dt. Marktstabilitätsreserve)
nEHS	Nationales Emissionshandelssystem
TNAC	Total Number of Allowances in Circulation (dt. Anzahl der Zertifikate im Umlauf)

1 Einleitung

2 Grundlagen

3 Transformer-Architektur im Detail

Im ursprünglichen Werk *Attention Is All You Need* , in dem das grundlegende Transformer-Modell vorgestellt wurde, besteht ein Transformer aus zwei Hauptkomponenten: dem Encoder und dem Decoder. cite

Encoder und Decoder verarbeiten Text in Form von Tokens, die ganze Wörter oder Wortteile sein können. Jeder Token ist für den Transformer einzigartig und wird zunächst nur durch eine natürliche Zahl repräsentiert.

Der **Encoder** übersetzt die Eingabewörter in eine abstrahierte Repräsentation. Ziel ist es hierbei, die semantischen und syntaktischen Informationen des Textes zu extrahieren und zu kodieren. Die andere Komponente, der **Decoder**, nutzt diese Repräsentation, um passende Outputs zu generieren. Dabei berechnet er in jeder Iteration den nächsten am besten passenden Token, der zum Output hinzugefügt werden soll. Hierbei bezieht er auch den zuletzt generierten Token in die Berechnung des nächsten Tokens ein.

Grundlage für sowohl den Encoder als auch den Decoder ist das Self-Attention-Konzept, das im folgenden Abschnitt näher erläutert wird.

3.1 Wie funktioniert Attention?

Ein Attention-Zyklus kann als Übersetzungsfunktion von Eingabevektoren zu Ausgabevektoren verstanden werden.

3.1.1 Generierung von Embeddings

Grundlage jedes Attention-Zyklus sind Eingabe-Tokens. Um die mathematische Vorgehensweise besser zu veranschaulichen, wird im Folgenden der Satz „Ich sitze auf der Bank“ als Beispiel verarbeitet. Jedes dieser Wörter ist ein eigener Token, der vor der Eingabe in den Attention-Zyklus vom Transformer übersetzt wurde: [„Ich“, „sitze“, . . . , „Bank“]. Diese Tokens könnten für das Modell wie folgt aussehen: [„243“, „645“, . . . , „316“]. Die Herausforderung für den

Transformer besteht darin, aus dem Kontext der anderen Tokens zu erkennen, ob „Bank“ eine Sitzbank oder das Finanzinstitut Bank bedeutet.

Jeder Token bildet ein Schlüssel-Werte-Paar. Der korrespondierende Wert hinter einem Token ist ein Vektor, der die Bedeutung eines Tokens hinsichtlich mehrerer Dimensionen beschreibt. Diese Vektoren sind aus Trainingsdaten des Transformer-Modells entstanden.

Im ersten Schritt des Attention-Zyklus wird jeder Token in den dazugehörigen Vektor übersetzt. Diese Vektoren werden in der Matrix $\mathbf{X}$ gespeichert, wobei jede Zeile einen Token repräsentiert. Dieser Prozess wird als **Embedding** bezeichnet. Jeder dieser Vektoren hat gemäß der Literatur mindestens 512 Dimensionen. Es wird von einem $d_{\text{model}} = 512$ gesprochen. Es gilt: Je größer das d_{model}, desto präziser kann der Transformer die Zusammenhänge zwischen Tokens erkennen.

Wenn sich Tokens im Vektorraum nahe liegen, haben sie eher Gemeinsamkeiten im Vergleich zu Tokens, die weit auseinander liegen. Angenommen, es gäbe nur ein $d_{\text{model}} = 2$ für jeden Token, könnten diese zwei Dimensionen als Koordinaten genutzt werden, um Zusammenhänge visuell als Cluster in einem Koordinatensystem darzustellen. Hier wären beispielsweise die Tokens „Hund“ und „Katze“ nah beieinander.

Für das oben genannte Beispiel „Ich sitze auf der Bank“ nehmen wir der Übersichtlichkeit halber ein $d_{\text{model}} = 4$. So ergibt sich eine Embedding-Matrix $\mathbf{X}$ mit 5 Zeilen für 5 Tokens und 4 Spalten für jeweils 4 Dimensionen:

$$\mathbf{X} = \begin{bmatrix} 0.4 & 0.8 & 1.5 & 1.6 \\ 3.2 & 0.4 & 0.7 & 0.2 \\ 0.6 & 0.9 & 1.2 & 0.5 \\ 2.1 & 0.5 & 2.0 & 0.2 \\ 0.7 & 2.4 & 0.1 & 0.9 \end{bmatrix}$$

3.1.2 Lineare Transformation in Query-, Key- und Value-Matrizen

Die Embedding-Matrix $\mathbf{X}$ wird durch drei Gewichtungsmatrizen $\mathbf{W_Q}$, $\mathbf{W_K}$ und $\mathbf{W_V}$, die aus dem Training des Transformer-Modells stammen, in drei neue Matrizen transformiert:

$$\mathbf{Q} = \mathbf{X} \cdot \mathbf{W_Q}$$

$$\mathbf{K} = \mathbf{X} \cdot \mathbf{W_K}$$

$$\mathbf{V} = \mathbf{X} \cdot \mathbf{W_V}$$

Die drei Matrizen haben im Attention-Zyklus umgangssprachlich formuliert folgende Funktionen:

- **Query-Matrix (Q)**: Was fragt ein Token?
- **Key-Matrix (K)**: Welche Tokens im Kontext antworten am besten auf die Frage?
- **Value-Matrix (V)**: Erlernten Informationen über ein Token.

Im Beispiel könnten die jeweiligen Zeilen der Matrizen **Q**, **K**, **V** für das Token „Bank“ folgendermaßen aussehen:

$$Q_{\text{Bank}} = [1.0, 0.7, 0.9, 1.1], \quad K_{\text{Bank}} = [0.8, 0.6, 1.0, 0.9], \quad V_{\text{Bank}} = [0.9, 0.5, 0.7, 1.0]$$

3.1.3 Berechnung und Einbeziehung von Attention-Scores

Um die Relevanz zwischen Q und K zu messen, wird jeweils das Skalarprodukt zwischen jedem Tokenvektor von Q_{T} und K_{T} gebildet. Also im Beispiel wird unter anderem der Tokenvektor Q_{Bank} mit jedem Tokenvektor K_{T} multipliziert.

$$\begin{aligned}
\text{Score}_{\text{Bank,Ich}} &= [1.0, 0.7, 0.9, 1.1] \cdot [0.4, 0.5, 0.1, 0.3] \\
&= 0.4 + 0.35 + 0.09 + 0.33 && = 1.17 \\
\text{Score}_{\text{Bank,Sitze}} &= [1.0, 0.7, 0.9, 1.1] \cdot [0.9, 0.8, 0.75, 0.8] \\
&= 0.9 + 0.56 + 0.675 + 0.88 && = 3.015 \\
&\vdots \\
\text{Scores}_{\text{Bank}} &= [1.17, 3.015, 2.92, 1.12, 2.98]
\end{aligned}$$

Die berechneten Attentionscores müssen noch zwei Verfahren unterlaufen. Einmal ist das die Fokussierung und Normalisierung der Attentionscores mit der **Softmax-Funktion**.

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

x ist der jeweilige aktuell zu betrachtende Attention-Score-Vektor. i ist der aktuell zu betrachtende Werteindex in diesem Vektor. j ist die Gesamtanzahl an Werten in x.

Bei der **Fokussierung** werden höhere Attention-Score-Werte zwischen **Q** und **K** exponentiell bevorzugt. Analog dazu werden niedrigere Attention-Score-Werte exponentiell nach unten bewertet. Bei der zweiten Aufgabe der Softmax-Funktion, der **Normalisierung**, werden die Attention-Score-Werte pro Score-Vektor in Wahrscheinlichkeiten zwischen 0 und 1 transformiert, wobei die Summe jedes Attention-Score-Vektors immer 1 ist.

Damit die Softmax-Funktion aber optimal funktionieren kann, müssen die Werte in den Attention-Score-Vektoren erst einmal dimensioniert werden. Bei geläufigen Transformermodellen wird wie oben beschrieben, ein d_{model} von mindestens 512 verwendet. Durch diese großen Dimensionen entehen bei der Berechnung von den Attention-Scores durch die Aufsummierung bei der Bildung des Skalarprodukte sehr große Werte. Diese großen Werte sorgen dafür, dass die Softmax-Funktion viele Q-K-Beziehungen sehr hoch bewertet und so der Transformer nicht sich auf die tatsächlich vielversprechenden Verbindungen konzentrieren kann und so die Weiterverarbeitung ungenau wird. Diese Werte fallen bei dem oben gerechneten Beispiel nicht auf, da hier nur mit einem d_{model} von vier gerechnet wird.

Um hohe Attention-Score-Werte zu normalisieren, werden die Attention-Score-Vektor-Werte durch $\sqrt{d_{\text{model}}}$ geteilt und so für die Softmax-Funktion in einen stabilen Bereich gebracht.
So kann ein Transformer auch kleine Relevanzunterschiede in der Token-Beziehung berücksichtigen. Hier beispielsweise für das Attention-Score-Array von „Bank“:

$$\frac{\text{Scores}_{\text{Bank}}}{\sqrt{4}} = [0.585, 1.5075, 1.46, 0.56, 1.49]$$

$$\text{Softmax}\left(\frac{\text{Scores}_{\text{Bank}}}{\sqrt{4}}\right) = [0.107, 0.269, 0.256, 0.104, 0.264]$$

Damit diese nun umgewandelten Attention-Score-Wahrscheinlichkeiten in den weiteren Verarbeitungsschritten berücksichtigt werden können, werden sie mit der V-Matrix multipliziert. Das zeigt dem Modell, zu wie viel Prozent der erlernten Informationen zu einem Token im nächsten Schritt einfließen. Insgesamt sieht das Verfahren folgendermaßen aus:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_{\text{model}}}}\right)V$$