I. Einleitung

A. Einstieg

Ein immer wichtigeres Thema in unserer Gesellschaft, ist künstliche Intelligenz. Die Frage nach Sicherheit und Gefahren für die neue Generation und deren mentalen Existenz – also z.B. ob KI uns den Job stiehlt oder unsere mentale Entwicklung negativ beeinflusst - werden immer lauter. Diese Arbeit soll Aufklären und die Möglichkeiten und aber auch Gefahren von KI analysieren.

Um zu verstehen was die Gefahren und Möglichkeiten von KI sind, muss man zuallererst verstehen, was KI ist und wie sie sich verhält. Deshalb werden wir uns ein Transformer Model im kleinsten Detail anschauen und resultierend aus den dadurch gewonnenen Informationen zu der folgenden Themenfrage eine Antwort liefern.

“Welche Gefahren und Möglichkeiten bringt KI für unsere mentale Existentes mit sich?”

Dafür habe ich ein KI-Transformer-Sprachmodell vorbereitet, dass man selbst trainieren kann um die KI als System zu verstehen, da man Anhand eines funktionierendem Beispiels, verstehen kann, wie KI arbeitet, wie KI unseren Trainingsdaten versucht nachzuahmen, wie intelligent KI ist, was KI eigentlich ist und im Endeffekt, wie gefährlich KI wirklich für unsere mentale Existenz als Menschen ist. Wir können Training hautnah miterleben und sehen, wie KI lernt. Während des Analysierens, werden wir auf folgende Punkte achten, da sie wichtig für die Themenfrage sind:

B. Fragen und Ziele der Arbeit

1. Fragen

- Wie schnell können beim Training daten erlernt werden
- Wie schnell können neuronale Muster nachgeahmt werden
- Wie gut können solche Muster nachgeahmt werden
- Wie kreativ ist sie (also wie gut kann KI eigene Gedanken entwickeln)?
- Wie viel kann KI, also wie viel Einsatzgebiete hat KI?

2. Ziel

C. Gefahren und Möglichkeiten von KI

1. Gefahren

KI kann ein nützliches Tool sein, aber es kann auch einige gefahren mit sich bringen.
Einige der Gefahren sind:

Zum einen Datenschutz und Informationssicherheit. Dazu gehört das Hochladen vertraulicher Daten in KI-Tools kann zu Datenschutzverletzungen führen, insbesondere wenn die Server außerhalb der EU liegen.
Außerdem bestehen Risiken durch KI-geschützte Cyberangriffe und die unsachgemäße Nutzung von Information durch KI-Systeme.

Ein weiterer entscheidender Punkt ist die Diskriminierung und Voreingenommenheit: KI-Modelle können Diskriminierung fördern, wenn die Trainingsdaten bereits Vorurteile (z.B. rassistische, sexistische) enthalten.
Dies führt zu verzerrten Ergebnissen, die diskriminierende Entscheidungen zur Folge haben können.

Zudem zeigt sich, die Soziale und ethische Auswirkung: Also die Automatisierung kann zu Arbeitsplatzverlusten und Veränderungen auf dem Arbeitsmarkt führen.
Eine zu große Interaktion mit KI kann die zwischenmenschlichen Beziehungen beeinflussen.
Es bestehen ethische Herausforderungen, wie zum Beispiel die Verwendung von KI bei Personalentscheidungen oder die Gefahr, dass KI für missbräuchliche Zwecke eingesetzt wird.

Als letztes Argument sind die Existenzielle Risiken. Da wäre das Problem eine superintelligente KI hätte übermenschliche Fähigkeiten in Technologie, Strategie, Manipulation und Selbstverbesserung.
Sie könnte sich durch das Internet oder physische Systeme (z.B. Roboter) ausbreiten.
Einmal außer Kontrolle, könnten wir sie nicht mehr stoppen.

---

2. Möglichkeiten

Bei einer KI gibt es viele Pro Argumente. Diese wären:

Als erstes, die Automatisierung von Routineaufgaben: Die KI übernimmt langweilige, wiederholende oder gefährliche Tätigkeiten.
Als Beispiel wäre das, Fließbandarbeit, Lagerlogistik und Aktenprüfung.

Ein weiteres Argument ist, die Bessere Entscheidungen durch Datenanalyse: Die KI kann riesige Datenmengen auswerten und Muster erkennen, die Menschen übersehen.
Zum Beispiel: Frühzeitige Erkennung von Krankheiten durch KI in der Medizin.

Noch ein wichtigeres Argument sind die Fortschritte in der Medizin: KI hilft bei Diagnose, Medikamentenentwicklung, OP-Planung oder Patientenüberwachung.

Beispielsweise KI erkennt Hautkrebs auf Bildern mit höherer Trefferquote als machen Ärzt: innen.

Zum Schluss Lösungen für globale Probleme: KI kann bei der Bekämpfung von dem Klimawandel, Hunger oder Pandemien helfen.
Beispiele wären: Optimierung von Energieverbrauch oder Klimamodelle.

II. Künstliche neuronale Netzwerke

A. Aufklärung

Ein neuronales Netzwerk ist ein System aus vielen miteinander verbundenen Einheiten (Neuronen), das aus Daten lernt, um komplexe Aufgaben wie Bilderkennung, Sprachverarbeitung oder Prognosen zu lösen. Künstliche neuronale Netze sind Algorithmen, die dem menschlichen Gehirn nachempfunden sind. Außerdem ist eine KI ein neuronales Netzwerk. In dem neuronalen Netzwerk gibt es verschiedene Schichten wie beispielsweise die Eingabeschicht, die Verarbeitungsschicht (zwischen Schichten) und die Ausgabeschicht.

[Eingabe] → [Zwischenschicht] → [Ausgabe]

1. Was ist das?

Im genauen Sinne ist das neuronale Netzwerk ein Computerprogramm, das so tut, als wäre es ein kleines Gehirn. Es hilft Computern dabei, aus Beispielen zu lernen, anstatt dass man ihnen alles einzeln beibringt. Im Vergleich zu künstlich neuronalem Netzwerk gibt es in einem menschlichem Gehirn Neuronen, das sind Zellen, die Informationen weitergeben. In einem künstlichen neuronalen Netzwerk gibt es künstliche Neuronen – das sind Rechenpunkte, die Zahlen verarbeiten. Also diese künstlichen Neuronen sind miteinander verbunden – wie in einem Netzwerk und sie sprechen miteinander, um herauszufinden, wie sie eine Aufgabe lösen können.

2. Eingabeschicht

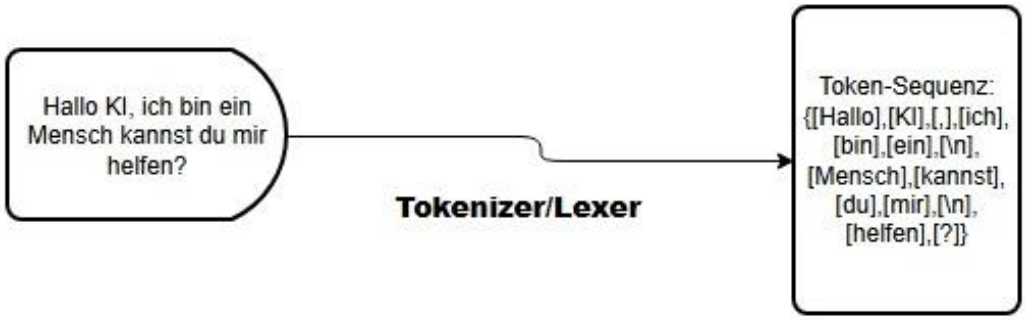
Die Eingabeschicht versorgt das neuronale Netz mit den notwendigen Informationen. Die Input-Neuronen verarbeiten die eingegebenen Daten und führen diese gewichtet an und dann die nächste Schicht weiter. Die Eingabeschicht nimmt Daten auf wie z.B. ein Bild von einer Katze. Das Bild besteht aus vielen Pixeln. Jeder hat z.B. einen Helligkeitswert oder eine Farbe – also eine Zahl. Diese Zahlen (z.B. 0 bis 255, Byte)

werden an die Neuronen in der Eingabeschicht übergeben. Das sieht dann ungefähr so aus.

Bild → Pixelwerte → Eingabeschicht → Weiterverarbeitung

Außerdem spielt der Token eine wichtige Rolle in der Eingabeschicht. Ein Token ist der Versuch die Eingabequelle in kleine Datenpakete zu zerlegen, um später (Verarbeitungsschicht) die Daten besser verarbeiten zu können. Bei einer Bild Verarbeitungs-KI entspricht ein Pixel, ein Token. Der Algorithmus, der die Eingabe Daten in Tokens zerlegt, wird als Tokenizer bezeichnet. Also die Eingabeschicht verarbeitet die Daten nicht, sondern sie leitet sie nur weiter. Außerdem übernimmt die zwischen Schicht die eigentliche Denkarbeit. Man sollte sich unbedingt merken das die Eingabeschicht ist wie der Briefkasten eines neuronalen Netzwerks – sie nimmt die Daten auf und reicht sie weiter an die nächste Schicht.

Das folgende Beispiel zeigt einen hypothetisch, realistisches Vorher-Nachher-Vergleich eines Tokenizers (Input-Benutzer-Text => Tokens). Was für uns wie unhandlich wirkt ist für den Computer ein wahrer Traum:



III. Zwischenschicht

A. Einleitung – Zwischenschicht

1. Zwischenschicht und Unterteilung

Die Zwischenschicht einer KI oder einem neuronalen Netzwerk allgemein ist das Herzstück derer. Hier findet die eigentliche Datenverarbeitung des neuronalen Netzwerkes statt. Sie ist der wichtigste, aber auch komplexeste Teil, weshalb man diese auch in Unterschichten oder Sub Layers aufgeteilt hat.
Folgendes Listing benennt die Zwischenschichten einmal deutlich:

Input → Embedding+Position → Transformer-Layer (x L) → Output
[-> Attention -> Feed Forward]

Ich werde versuchen, das Thema und die Funktion in den nächsten Unter Themen greifbar zu veranschaulichen, aber dennoch wissenschaftlich korrekt zu bleiben. Ich bitte deshalb um ihr Verständnis, wenn es kompliziert, wird:

B. Analyse der Zwischenschicht von Transformer-Modellen eines Text-Modell

1. Input Layer

Nachdem der Tokenizer uns eine Token Sequenz, die unsere Eingabe beschreibt, erzeugt hat, können wir nun mit der Weiterverarbeitung fortfahren. Dafür gibt die KI jeden Token einen Index (mathematisch versehen mit den Buchstaben *i*), diesen bekommt sie durch ein Vokabular (bei ChatGPT-5-mini ≈ 50.000 Tokens groß), dass alle möglichen Wörter oder Zeichen (falls diese nicht existieren, werden Rohe Charakter an die Zwischenschicht übergeben) der KI enthält. Ein KI-Model hat eine Dimensionsgröße, wobei jede Achse z.B. ein Ausdruck oder Klang eines Worts beschreibt. Die Anzahl der Dimensionen (also Achsen eines Vektors) wird mathematisch mit dem Buchstaben *D* versehen und wird auch Embedding-Dimension genannt. Schätzungsweise liegt diese bei ChatGPT-5-mini bei ca. 4.096 Demissionen. Um jeden Token nun also zu beschreiben (z.B. Ausdruckskraft oder Klang) besitzt eine KI eine Matrix, die diese Informationen beinhaltet. Diese Matrix heißt Embedding-Matrix (weshalb auch der Name Embedding-Dimension) und wird mathematisch mit dem Buchstaben *E* versehen. Sie wird aus mehreren Vektoren zusammengesetzt, die jeweils die Größe der Embedding-Dimension haben. Wie viele Vektoren *E* enthalten sind, hängt wiederum von der Vokabular Größe des Modelles ab. Es gilt also:

$$\mathbf{E} = \mathbf{E} \in \boldsymbol{R}^{\mathbf{V}\times\mathbf{D}}$$

... also die Embedding-Matrix ist die Sammlung von *x* Vektoren, mit der Embedding-Dimension Größe als Dimensionsanzahl, wobei *x* der Vokabular Größe entspricht.

Ziel ist alle Token Index Vektoren (vereint als Matrix ***T***) mit der Embedding-Matrix zu vermischen, genauer gesagt, dient der durch das Vokabular zugeteilten Index als Schlüssel zum eigentlichen Vektor, der das "Wort" oder genauer der Token beschreibt. Dabei gilt einfach formuliert:

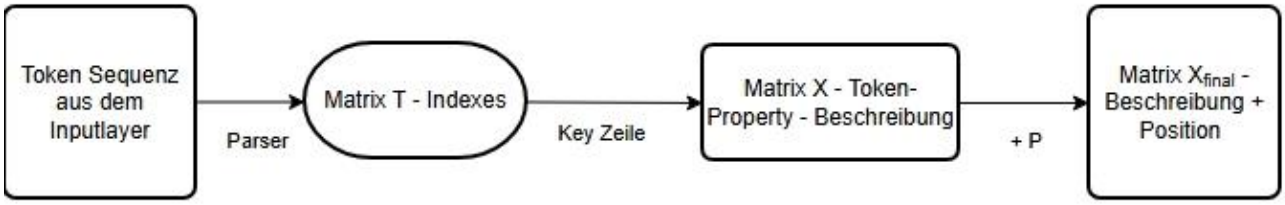
"Nehme den dir zugewiesenen Index, gehe zur der zum Index passenden Spalte der Embedding-Matrix und nehme den dort hinterlegten Vektor"

Nun bekommen wir eine neue Matrix namens ***X***. Diese enthält alle Token Vektoren, die wir durch das Verrechnen der Tokens mit der Embedding-Matrix erhalten haben. Das sorgt dafür, dass ähnliche Tokens (also z.B. ähnliche Wörter) ähnliche Werte haben und dadurch auch ähnlich von der KI behandelt werden.

Nun ist es wichtig, dass die KI aber die Positionen der Tokens weiß, um z.B. komplizierte Grammatik zu "verstehen". Beispielsweise ist es nicht egal, ob die KI: *"Hallo KI, ich bin ein Mensch!"* versteht, oder *"Hallo Mensch, ich bin ein KI!"*. Deshalb gibt es eine weitere Matrix (ja eine Matrix) namens ***P*** auch Position-Matrix. Also addieren wir einfach *X* mit *P* um auch die Position so effizient wie möglich zu beschreiben. Es gilt also:

$$\mathbf{X_{final} = X + P}$$

Zum Schluss des Input-Sub-Layers erhalten wir also eine Matrix ***X***, die aus mehreren Vektoren besteht, die aus der Position und den Eigenschaften des Tokens zusammengesetzt sind. Mit diesen Vektoren können wir weiterarbeiten. Das folgende Flussdiagramm soll nochmal den Schritt ‘Input-Sub-Layer’ zusammenfassen:



2. Transformer Layers – das Herz der Zwischenschicht

Vielleicht haben sie sich gefragt, wofür ChatGPT steht, oder jedenfalls das GPT? Hier die Auflösung:

G ... Generative
P ... Pre-Trained
T ... Transformer

Ja, was ist ein Transformer, wenn er sogar in ChatGPTs Namen steht (den ChatGPT ist ein Transformer Model)? Ein Transformer ist eine KI-Model-Art, die modellabhängig viele so-genannte Transformer-Layers enthält. Diese sind das eigentliche "Gehirn" der KI.
Sie werden wie schon gerade angesprochen wiederholt und diese ist modelabhänig. Z.B. hat ChatGPT-4 ca. 45 Transformerschichten. Zum Vergleich: ChatGPT-2 hat ca. 15.
Transformer-Layers haben 2 charakteristische Zwischen-Layers:

- Self-Attention Sub-Layer
- Feed Forward Sub-Layer

Diese werden jetzt genauer behandelt...

3. (Self-) Attention

Die Attention oder auch Self-Attention also auf Deutsch Selbst-Achtung ist unteranderem der Punkt, wo Wahrscheinlichkeiten ins Spiel kommen. Die KI versucht zu filtern, was wichtig für sie ist. Das hat nicht nur Performance Gründe, sondern auch Logik Gründe. Ohne diesen Schritt würde die KI nicht funktionieren. Denn unser menschliches Gehirn ist in diesem Punkt der KI ähnlich. Wir filtern aus. Z.B. ist das Gespräch mit dem Gegenüber wichtiger als das Vogelgezwitscher im Park. Unser Gehirn schätzt dieses nämlich als unwichtiger ein. Aber wie funktioniert das, dass ein Algorithmus werten kann?

Als Eingabe dieser Layer, dient die im letzten Layer entstandene Matrix *X*. Gegeben sind uns 3 vorttrainierte Matrizen W_Q, W_K, W_V. Sie werden mit *X* zusammenmultipliziert. Dabei entstehen die 3 Matrizen *Q, W und V*. Die Matrizen haben folgende (abstrakte) Funktion:

- **Q (Query)** = fragt: *Wen soll ich beachten?*
- **K (Key)** = sagt: *Wie relevant bin ich?*
- **V (Value)** = enthält die Informationen über das Token an sich

Um zu verstehen, wie diese Matrizen das Endergebnis beeinflussen, muss man in Relation zu jedem Token denken. Da *X* eine Sammlung aus allen Tokens ist, kann man durch das Multiplizieren mit den vortainierten Matrizen erzielen, dass man jedes Token in Relation denkt. Hier ein pseudo, Python ähnliches Skript:

```
for t in V:       # <- für jedes Token aus V
     Q.add(t*MQ) # errechne q
     K.add(t*MK) # errechne k
     V.add(t*MV) # errechne v
```

Der nächste Trick ist, dass multiplizieren, der Query Matrix mit dem Kehrwert von K. Dadurch bekommen wir eine neue Matrix namens ***S***. Vereinfacht gesagt enthält ***S*** für jeden Token die Information, wie stark es andere Tokens “beachten”, also wie wichtig andere in Relation zu sich selbst sind. Nun enthält ***S*** aber riesige inputabhängige Zahlen. Ein abstraktes stark entferntes Beispiel ist: Ein Nutzer ist wütend und redet mit “kräftigen” Wörter mit der KI. Das heißt die Werte der Achsen, die die ‘Aussagekraft’ der Wut beschreiben schlagen stark aus. Im Vergleich zu anderen Nutzereingaben, die z.B. interessiert wirken, ist das ein großer Unterschied und kann zu starken Störungen der Weiterverarbeitungen führen.

Deshalb müssen wir die Werte *normalisieren,* das heißt sie relativ zu anderen Tokens kleiner machen, ohne sie in Relation zu verändern. Wir bekommen eine neue Matrix namens ***A***. Sie enthält Wahrscheinlichkeiten. Die Formel dafür sei jetzt einfach so dahingeschmissen, aber sie ist wichtig, damit kein Irrsinn herauskommt:

A = softmax(S / sqwrt(dk))

A enthält jetzt die Information, wie stark Tokens andere Beachten und wie wichtig sie für sie sind. Jetzt wollen wir aber auch die Information: wie wichtig bin ich (der Token) für andere Tokens. Dafür verrechnen wir einfach *V* mit *A*. Also einfach $\boldsymbol{Z=A \cdot V}$. Wie gerade in der Formel gesehen, heißt die neue Matrix ***Z***. Sie enthält jetzt folgende Informationen:

- Wie wichtig ist jeder Token für den betrachteten Token (S)?
- Wie wichtig bin ich für den ganzen Kontext (V)?

Wer aufgepasst hat, dem wird auffallen, dass die Information, welche Informationen die Tokens beinhalten, verloren gegangen ist. Um das zu umgehen und unsere Matrix mit allen Informationen der Eingabesequenz zu füllen, müssen wir einfach *Z* mit *X* verrechnen. Dabei wird ein mathematisches Verfahren namens LayerNorm verwendet. Weitere Informationen finden Sie hier (Python Bibliothek namens PyTorch – Neuronale Netzwerke in Python): LayerNorm — PyTorch 2.9 documentation. In mathematischer Schreibweise sieht der Schritt wie folgt aus:

H1 =LayerNorm(X+Z)

H1 ist die kompakte Endmatrix des Attention-Layers oder “Selbstachtungs-Schicht". Sie enthält folgende Informationen über alle Tokens:

- Wie wichtig ist jeder Token für den betrachteten Token (S)?
- Wie wichtig bin ich für den ganzen Kontext (V)?
- Was beinhalte Ich als Token (T)?

Also alles, was man für den nächsten großen und wichtigen Layer(/s) brauch...

4. Feed-Forward Layer

Das Ziel des Feed-Forward Sub-Layers ist es, komplexere Merkmale, aus jeden Token Vektor herauszubekommen. Dabei wird auf jedes Token einzeln geschaut, da sie ja in der vorherigen Schicht schon verglichen worden sind.

Das erste Problem ist der Rechenraum.
Die Matrix ist zu klein, um umfangreichere Erkenntnisse über den Token und damit die Token Sequenz zu gewinnen. Deshalb vergrößern wir einfach die Größe der Matrix.
Dafür haben wir 2 vortrainierte Objekte: Eine Matrix namens W_1 und einen Wert namens b_1.
Wir bekommen eine neue Matrix namens U als Input nehmen wir die Ergebnis-Matrix aus dem vorherigem Layer H_1:

$$\mathbf{U = W_1 \cdot H_1 + b_1}$$

Das Problem: *U* ist linear, so kann das Modell keine komplexen Muster erkennen. Jetzt werden die Muster "verfeinert". Deshalb wird *U* nicht linear gemacht. Und zwar werden negative Werte ausgefiltert. Also zu 0 genormt. Das ist wichtig, da nur wichtige Information an die nächsten Schichten weitergeleitet werden. Mathematisch: **R=ReLU(U)**

Jetzt werden die Werte also der Rechenraum wieder verkleinert: **F=W2 ·R+b2**

Wir haben jetzt *F* eine Matrix, die unsere verarbeiteten Informationen des einzelnen Tokens enthält und verbinden jetzt das Ergebnis aus dem Attention-Layer (H1) und aus dem Feed-Forward Layer (F) zusammen:

$$\mathbf{H_2 = LayerNorm\ (H_1 + F)}$$

H_2 ist Input des nächsten Transformer Layers. Das wird dann *L* mal wiederholt. Wobei *L* die Anzahl an Transformer Layers ist.

5. Output Layer

Wir haben als Output der Transformer-Layers *H*. *H* ist eine Matrix, die das letzte Transformer Layer zurückgibt. *H* ist wie folgt aufgebaut:
(Anzahl Tokens × *D*)

Jetzt kommt der Interessante Part: Die Bildung der Output Tokens:

Zuerst müssen wir jedes Token in das Vokabular "projiziert". *H* enthält (wie X am Anfang) Informationen über Wort-Eigenschaften. Also z.B. wie wütend, formell oder interessiert klingt es. Diese Informationen sind wieder wie bei *X* aufgefächert in verschiedene Achsen der Matrix. Die Anzahl der Achsen beschreibt (zur Erinnerung) die Zahl *D*. Genauso wie die **Embedding-Matrix,** bloß umgekehrt verschieben wir jeden in *H* enthaltenen Token durch folgende Formel in neue Demission. So, dass sie die Worteigenschaften als Zahl repräsentieren:

$$\mathbf{z_n = h_n \cdot W_{out} + b_{out}}$$

Dabei wird jeder Token durchgegangen. h_n repräsentiert den betrachteten Token W_{out} und b_{out} sind vortrainierte Werte. Nun mögen wir aber Wahrscheinlichkeiten, um auch einen Zufälligen Output wiederzugeben. Z.B. antwortet ChatGPT immer ähnlich, aber mit verschiedenen Wörtern. Die Aussage der Antwort bleibt gleich. Wir nutzen deswegen wieder softmax um die Zahlen der Tokens in Wahrscheinlichkeiten umzurechnen, wobei die Summe aller Wahrscheinlichkeiten pro Token 1 ergibt:

$$\mathbf{p_n = softmax(z_n)}$$

P_n enthält die Wahrscheinlichkeit für jeden nachfolgenden Token der Token Sequenz nach dem betrachteten Token. Dann zieht man entweder den Wahrscheinlichsten Token, oder man zieht einen zufälligen Token, wobei der wahrscheinlichste Token auch wahrscheinlicher zu ziehen sind. Am Ende erhalten wir eine Sequenz aus Index, denen wir je einen Token zuordnen können. So erhalten wir eine Sequenz von Wörtern und Zeichen.

So wird aus "Hallo KI!" "Hallo Mensch!", ohne einen Menschen zu haben, der weiß, was er tut.

6. Mashine Learning

Machine Learning oder maschinelles Lernen ist ein Teilbereich der künstlichen Intelligenz. Der Schwerpunkt liegt dabei auf dem Trainieren von Computern, um aus Daten und Erfahrungen zu lernen und sich stets zu verbessern – anstatt explizit dafür programmiert zu werden. Beim Machine Learning werden Algorithmen darauf trainiert, Muster und Korrelationen in großen Datensätzen zu finden und auf Basis dieser Analyse die besten Entscheidungen zu treffen und Vorhersagen zu treffen.

Machine Learning ist fast der wichtigste Teil der KI-Entwicklung. An diesem Punkt wird der Einsatzbereich der KI entschieden und Sie so angepasst, dass sie ihren Zweck erfüllt.

Eine KI ist ein "Haufen" verketteter Matrizen und Gewichten. Dieses enthalten anfangs zufällige Werte. Da wir Tokens als Vektoren in einem gigantischen Koordinatensystem mit oft tausenden von Achsen, können wir diese mit Matrizen im Raum verschieben und so, wenn die Matrizen den Token korrekt verschieben, den perfekten Output erlangen und den Anschein eines Denkenden erwecken. Um aber diesen Punkt zu erreichen, müssen wir die Matrizen trainieren. Das heißt: Wir passen die Matrizen mit den enthaltenen Zufalls Werte so an, dass sie unseren Zweck erfüllen und bis sie die Tokens gewünscht transformieren. Hier alle zu trainierenden Matrizen und Parameter:

- Embedding
  - E (Token-Embedding)
  - P (Positions-Embedding)

- Self-Attention
  W_Q, W_K, W_V, W_O
  b_Q, b_K, b_V, b_O

- Feedforward (MLP)
  - W_1, W_2
  - b_1, b_2

- Layer Normalization
  - γ (Gamma)
  - β (Beta)

- Output Layer
  W_out und b_out

Aber wie funktioniert das Training intern? Wie benötigen große Daten Bibliotheken. Dieses enthalten eine hypothetische, beste Eingabe und die dazugehörige Antwort der KI. Wir nehmen die Eingabe, geben sie an das Modell, nehmen den Output und vergleichen den Output

der KI mit dem gewünschten Output. Dabei wird die Differenz berechnet und Anhand dieser alle Matrizen der KI angepasst. Die magische Formel zu Berechnung der Differenz (0-1) lautet wie folgt:

$$\mathbf{Loss} = -\sum_{i} \mathbf{y_i} * \mathbf{\log(p_i)}$$

Loss ... Differenzwert
Yi ... Erwünschtes Ergebnis
pi ... vom Modell vorhergesagte Wahrscheinlichkeit für Token *i*

Da wir jetzt die Differenz haben, können wir durch das Rückwerts-Rechnen das Mitwirken eines Parameters oder einer Matrix zum Fehler erfahren und diese durch folgende Formel optimieren:

$$\mathbf{W_{neu}} = \mathbf{W_{alt}} - \eta \cdot (\partial \mathbf{W} / \partial \mathbf{Loss})$$

η ... Lern-Rate – An welchem Lern-Schritt sich das Model befindet, also wie fein

Da einmal durchgehen nicht alles verändert und nicht die gesamte KI verbessert, wird dies bei großen Modellen wie ChatGPT milliardenfach wiederholt. Am Ende haben wir ein Abbild von unseren Wünschen und unseren Denkstrukturen im Gehirn. Ein Bewusstsein hat es aber nicht. Es ist nur ein Abbild vom gewünschten, eine riesige Formel, ein Algorithmus.

IV. Der Compiler als neuronales Netzwerk

A. Der Übersetzer für Code

1. Funktionsweise

2. Zusammenhänge mit KI

V. Testmodel KI

A. Eigene KI programmieren…

1. Wie funktioniert sie

VI. Wie mächtig ist KI?

A. Und wie stark ist sie bedrohlich?

1. Vergleich Mensch-KI

2. Reflektion „KI-Testmodel“

3. Möglichkeiten KI

VII. Quellen