UNIKASSEL
VERSITÄT

SOFTWARE ENGINEERING
UNIVERSITÄT KASSEL

Large Language Models (LLMs): Eine kurze Einführung

Jens Kosiol

Sommersemester 2024

Was ist ein Large Language Model?

- Large Language oder Foundation Models sind extrem große neuronale Netze, die auf riesigen Datenmengen trainiert wurden.
- Ein LLM kann als eine mathematische Funktion verstanden werden, die, gegeben eine Zeichenkette, eine Wahrscheinlichkeitsverteilung dafür berechnet, welches Zeichen als nächstes folgt (Next Word Prediction).
- Allgemeiner lernt ein LLM, gegeben eine Kette von Token, das nächste Token vorherzusagen. Ein Token kann eine (kurze) Zeichenkette, aber auch eine Ton-, Video-, oder Bildsequenz repräsentieren (genannt multimodality).

Warum „Next Word Prediction“?

- Next Word Prediction erlaubt Self-supervised Learning, was bedeutet, dass große Mengen an Trainingsdaten (vergleichsweise) günstig zur Verfügung stehen (jeder Text).
- Ein Modell, das gelernt hat, das nächste Wort vorherzusagen, lässt sich vielfältig einsetzen:
  - Generierung von neuen Inhalten (Text, Code, …)
  - Zusammenfassung von Inhalten
  - Übersetzung von Text
  - Suche in Text
  - …
- Beobachtung: Hat ein Modell eine statistische Repräsentation von Sprache gelernt, ist es in der Lage, gewisse Transferleistungen zu erbringen. Beispiel: Sentimentanalyse

Was zeichnet Large Language Models aus?

- Größe (Anzahl der Parameter): LLMs sind neuronale Netze mit extrem hoher Anzahl an Layern und Knoten
- Trainingsprozess:
  - Training auf sehr großen Datenmengen
  - Aufwändige Struktur des Trainings mit einer Phase des self-supervised learnings (Pre-training) und Finetuning auf händisch entworfenem Trainingsmaterial (Alignment)
- Attention mechanism (Transformer-Architektur): Das Modell kann Input großer Länge verarbeiten und Zeichen als wichtig erkennen und beachten, die weit von der zu berechnenden Ausgabe entfernt sind [VSP+17].

Was macht ein LLM?

Ein LLM empfängt (typischerweise) eine Sequenz von Characters als Input und generiert eine Sequenz von Characters als Output. Grobablauf dabei:

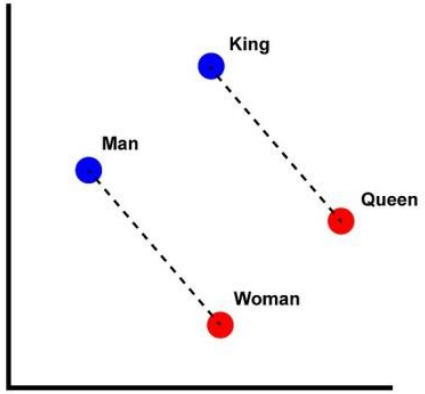
1. Die Eingabesequenz wird in eine Sequenz von Token zerlegt. Intuition: Token sind das Alphabet des LLM.
2. Jeder Token wird auf einen Vektor in einem hochdimensionalen Vektorraum abgebildet (embedding). Intuition: Numerische Repräsentation des Token, die die Semantik bewahrt, und auch Position in der Eingabesequenz abspeichert.
3. Die Menge der Vektoren durchläuft verschiedene Layer von sog. Attention-Heads und Feed-Forward Neuronalen Netzen. Intuition: Auf jeden Vektor wird übertragen, was er im Kontext der vor ihm vorkommenden Vektoren bedeutet.
4. Aus jedem Vektor wird eine Wahrscheinlichkeitsverteilung abgeleitet, welcher Token als nächstes kommt.
5. Der nächste Token wird auf Grundlage der Wahrscheinlichkeitsverteilung, die sich aus dem letzten Token der Eingabe ergibt, bestimmt, an die Eingabe angehängt und als neue Eingabe an das LLM geschickt.

Token und Tokenization

- Die Token sind das „Alphabet“, mit dem ein LLM arbeitet.
- Ein Token ist eine Zeichensequenz, die in dem Textkorpus, auf dem der Tokenizer entwickelt wurde, häufig auftaucht.
- Open AI Tokenizer:
  - Für englischen Text gilt durchschnittlich: Ein Token hat vier Zeichen und entspricht 0,75 Wörtern
  - Tokenalphabet für GPT 3.5 und GPT 4: > 100k
  - https://platform.openai.com/tokenizer

Word Embedding

- Die Darstellung von Worten als Vektoren (sodass Semantik repräsentiert wird) ist ein klassisches NLP-Problem.
- Kodierung von Worten/Token als Vektoren erlaubt das Anwenden mathematischer Operationen. Intuitionen:
  - Nähe (euklidische Distanz) oder auch hohes inneres Produkt korrespondieren mit ähnlicher Bedeutung.
  - Richtungsvektoren kodieren Äquivalenzen (der Unterschied zwischen Mann und König entspricht dem zwischen Frau und König).
  - Bedeutung von Sätzen, Paragraphen, … kann durch Aggregation der vorkommenden Worte berechnet werden.
- Die großen Anbieter von LLMs haben ihre eigenen neuronalen Netze zur Berechnung von Embeddings für ihr Tokenalphabet.



Bildquelle: Singerep, Wikimedia Commons, CC BY-SA 4.0 DEED

Logits

Am Ende hat ein LLM pro Eingabetoken einen Vektor der Länge des Tokenalphabets berechnet, dessen Einträge Logits heißen.

- Intuition: Jeder dieser Vektoren repräsentiert, gegeben die Token bis zu ihm, welcher Token folgen kann. Je höher der Eintrag an der Position eines Tokens ist, desto wahrscheinlicher ist es, dass dieses Token als nächstes folgt.
- Beispiel: Gegeben ein Tokenalphabet der Größe 5, könnte ein Vektor von Logits wie folgt aussehen: $[3.23, -7.1, 5.57, 0.39, 2.4]$
  Wurde dieser Vektor ausgehend vom 20. Eingabetoken berechnet, bedeutet, dies, dass gegeben die ersten 20 Token der Eingabesequenz wahrscheinlich das 3. Token des Alphabets als nächstes folgt.

Normalisierung und Sampling I

- Gegeben eine Eingabesequenz wird das nächste Token auf Grundlage der Logits des letzten Eingabetokens bestimmt (und wird dann das letzte Token der neuen Eingabe).
- Beobachtung: Logits bilden noch keine Wahrscheinlichkeiten:
  - Werte müssen nicht zwischen 0 und 1 liegen.
  - Werte müssen nicht zu 1 summieren.

Normalisierung und Sampling II: Softmax

- Die Softmax-Funktion (oder: normalisierte Exponentialfunktion) ist eine viel genutzte Funktion, um Logits in eine Wahrscheinlichkeitsverteilung zu überführen.
- Definition: Gegeben einen Vektor $z = [z_1, \dots, z_n] \in \mathbb{R}^n$ ist die Funktion definiert als

$$\mathrm{Softmax}(z)_j = \frac{\exp z_j}{\sum_{i=1}^{n} \exp z_i}.$$

- Beispiel: Anwendung von Softmax auf die Eingabe

$$[3.23, -7.1, 5.57, 0.39, 2.4]$$

ergibt (ungefähr)

$$[0.084, 2.75 * 10^{-6}, 0.87, 0.005, 0.037].$$

Normalisierung und Sampling III

- Es gibt weitere Möglichkeiten, aus Logits eine Wahrscheinlichkeitsverteilung über das Tokenalphabet zu bestimmen.
- Gegeben eine Wahrscheinlichkeitsverteilung, sind viele Samplingmethoden denkbar:
  - Sample das nächste Token gemäß der berechneten Wahrscheinlichkeiten.
  - Sample immer das wahrscheinlichste Token als nächstes.
  - Sample uniform gleichverteilt aus den 3 besten Token.
  - …
- Wir schauen uns gleich verschiedene Möglichkeiten, die Bestimmung der Wahrscheinlichkeitsverteilungen und/oder die Sampling-Methode anzupassen, im Zusammenhang mit Parametern an, die APIs bekannter LLMs anbieten.

Zweistufiger Trainingsprozess

- Pre-training: Self-supervised learning auf Texten aus dem Internet
  - In dieser Trainingsphase lernt das LLM die Aufgabe der next-word-prediction.
  - Die Trainingsdaten sind vergleichsweise billig (Web scraping)
  - Qualitätskontrolle der Daten ist minimal
  - Die Trainingsphase auf diesem Datensatz ist aufwändig und teuer.
  - Diese Trainingsphase wird seltener durchgeführt
- Fine-tuning: Reinforcement learning from human feedback
  - In dieser Trainingsphase wird trainiert, dass das LLM die Rolle eines Chatassistenten einnimmt, und versucht Alignment (Übereinstimmung mit humanen Werten) zu erreichen.
  - Trainingsdaten für diese Phase sind von hoher Qualität und teuer:
    - Nutzer interagieren mit dem LLM und bewerten die Interaktion
    - Datensätze mit prototypischen Interaktionen oder gescheiterten Interaktionen werden händisch entworfen
  - Trainingsphase kann öfter und leichter wiederholt werden

Interaktionsarten mit LLMs

- Chatassistent (ChatGPT und Co.)
  - Für viele LLMs gibt es einen Chatassistenten, der leichte Interaktion mit dem LLM erlaubt.
  - Keine Kontrolle über Parameter oder exakte Prompt (Systemprompt)
- API
  - Viele LLMs bieten Zugriff über eine API an
  - Größere Kontrolle
    - Zugriff auf manche Parameter
    - Größere Kontrolle über Systemprompt
- Indirekter Zugriff über Integration in Tools (z.B. GitHub Copilot)
  - Wenig Kontrolle über Prompt

Kontrollparameter von LLMs

Bei Nutzung über eine API bieten LLMs diverse Möglichkeiten, das Sampling zu steuern:

- Frequency penalty (GPT): logits von Wörtern, die bereits vorkommen, werden erhöht (oder gesenkt)
- Logit bias (GPT): Addiert einen ausgewählten Bias auf die Logit-Werte ausgewählter Token (basierend auf Token ID)
- Ausgabelänge (GPT, Gemini): Maximal mögliche Länge der Antwort (Modelle haben jeweils obere Schranke)
- Stop (GPT, Gemini): Strings/Token definieren, bei denen die Generierung von Text abbricht
- Temperatur (GPT, Gemini): Umrechnung der Logits in Wahrscheinlichkeiten wird gemäß der gewählten Temperatur angepasst.
- Top p (GPT, Gemini): Es wird nur aus den Token gesampelt, deren gemeinsame Wahrscheinlichkeit p erreicht (angefangen von den wahrscheinlichsten Token)
- Top k (Gemini): Beim Sampling werden nur die k Token mit den höchsten Logits beachtet

Temperatur

- Die Temperatur ist ein zusätzliche Parameter $T > 0$, um den die Softmax-Funktion ergänzt wird: Gegeben einen Vektor $z = [z_1, \dots, z_n] \in \mathbb{R}^n$ ist die Funktion definiert als

$$\text{Softmax}(z, T)_j = \frac{\exp \frac{z_j}{T}}{\sum_{i=1}^{n} \exp \frac{z_i}{T}}.$$

- Erhöhen der Temperatur sorgt für „gleichmäßigere“ Verteilung der Wahrscheinlichkeiten.
- Beispiel:

	$T = 0.01$	$T = 0.5$	$T = 1$	$T = 2$	$T = 5$
3.23	0	0.01	0.08	0.19	0.24
−7.1	0	0	0	0	0.03
5.57	1	0.99	0.87	0.63	0.39
0.39	0	0	0	0.05	0.14
2.4	0	0	0.04	0.13	0.2

Videoquellen zu LLMs

- Andrej Karpathy: https://www.youtube.com/@AndrejKarpathy
  - Allgemeinverständliches Erklärungsvideo zu LLMs
  - Videoerklärung zu Bau und Training von LLMs
  - Videoerklärung zum Bau eines Tokenizers
- 3Blue1Brown (Grant Sanderson): https://www.youtube.com/@3blue1brown
  - Videoserie zu Deep Learning, wo die beiden letzten Videos damit beginnen, LLMs und die Transformer-Architektur zu erklären

Wissenschaftliche Literatur

- [VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin: Attention is All you Need. NIPS 2017: 5998–6008