großes Sprachmodell (LLM)

Autor und Herausgeber

Alexander Jung*, Department of Computer Science, Aalto University, Espoo, Finland (ORCID 0000-0001-7538-0990)
Editors: Konstantina Olioumtsevits (ORCID 0000-0003-2697-8659), Ekkehard Schnoor (ORCID 0000-0001-8252-1499), Aalto University, Espoo, Finland
*Corresponding author: alex.jung@aalto.fi

Zusammenfassung

Ein großes Sprachmodell (LLM) ist ein künstliches neuronales Netz (KNN), typischerweise mit Milliarden von Modellparameter, das eine Hypothese-Abbildung von einem Prompt zu einer Antwort implementiert. Das Training eines LLM erfolgt oft in Stufen: Vortraining verwendet selbstüberwachtes Lernen, wobei Rohtext in überlappende Fragmente aufgeteilt wird, die Datenpunkte ohne manuelle Annotation ergeben, und Feinabstimmung verwendet einen kleinen Datensatz manuell kuratierter Prompt-Antwort-Paare. LLM-Agenten erweitern LLMs über Text hinaus: Ein Programm interpretiert die Antwort, zum Beispiel als auszuführenden Code oder als auszuführendes Werkzeug, und kann das Ergebnis an den nächsten Prompt anhängen, sodass das LLM Aktionen auswählt, die das umgebende Programm ausführt.

Definition

Man betrachte ein chatbasiertes artificial intelligence (AI)-System wie ChatGPT, Claude, Gemini, DeepSeek oder Mistral: Ein Benutzer gibt eine Frage, den Prompt, ein und erhält eine Antwort, die Response. Das System, das die Antwort erzeugt, ist ein LLM: ein künstliches neuronales Netz (KNN), typischerweise ein Transformer (Vaswani et al., 2017) mit Milliarden von Modellparameter (Brown et al., 2020), das darauf trainiert ist, Ausgabe-Text basierend auf Eingabetext zu erzeugen. Jeder Datenpunkt dieser Lernaufgabe ist ein Textpaar, und das trainierte LLM implementiert eine Hypothese-Abbildung vom Prompt zur Response (Abb. 1). Ein Prompt und eine Response sind Folgen von Tokens, sodass der Merkmalsraum und der Labelraum dieser Lernaufgabe Mengen von Token-Folgen sind. Im Gegensatz dazu ist ein Text-Klassifikator, der ein Dokument auf eine von einer festen Menge von Labels abbildet, kein LLM: Er gibt ein Label zurück, keinen Text.

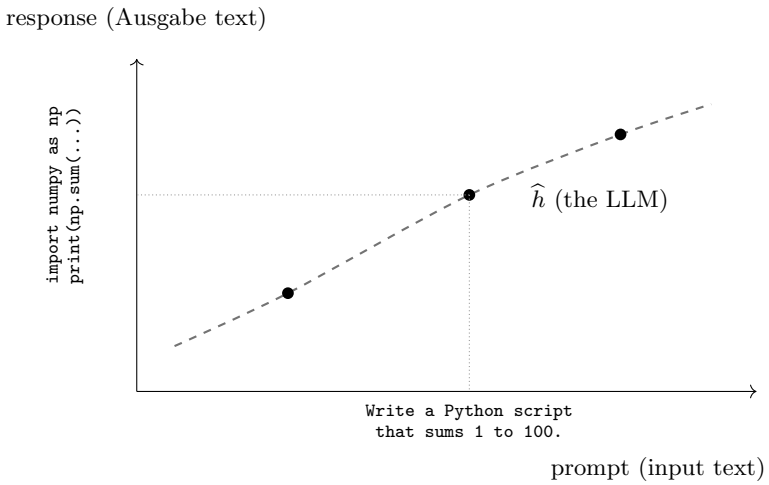


Abbildung 1: Ein LLM als Hypothese-Abbildung von Prompts zu Responses. Jeder ausgefüllte Kreis ist ein Datenpunkt, ein Paar aus Eingabetext und Ausgabe-Text; für einen der drei sind der Prompt und die Response ausgeschrieben. Die gestrichelte Kurve stellt die Hypothese $\widehat{h}$ dar, die das trainierte LLM implementiert: Sie weist jedem Prompt eine Response zu. Die Achsen sind schematisch, da der zugrunde liegende Merkmalsraum und Labelraum aus Token-Folgen bestehen, für die es keine natürliche Ordnung gibt.

Die meisten LLMs implementieren $\widehat{h}$ autoregressiv: Die Response wird Token für Token generiert, wobei jedes aus einer Wahrscheinlichkeitsverteilung über das vocabulary gezogen wird, die das Netzwerk aus dem Prompt und allen bisher generierten Tokens berechnet (Bengio et al., 2003; Brown et al., 2020). Alternative Konstruktionen existieren: Maskierte Wahrscheinlichkeitsmodelle prognostizieren entfernte Tokens und dienen der Textanalyse statt der Generierung (Devlin et al., 2019), und Diffusions-Sprachmodelle generieren viele Tokens parallel (Li et al., 2022).

Das Training eines LLM besteht aus mehreren Stufen. Eine Stufe verwendet typischerweise selbstüberwachtes Lernen: Ein gegebener Text wird in überlappende Fragmente aufgeteilt, und jedes Fragment entspricht einem Datenpunkt, dessen Label eines seiner Tokens ist und dessen Merkmale die verbleibenden Tokens sind. Aus Textfragmenten erstellte Trainingsdatensätze erfordern keine manuelle Annotation; eine Next-Token-Fragment-Konstruktion wird von Brown et al. (2020) verwendet und eine Masked-Token-Konstruktion von Devlin et al. (2019). Vortraining minimiert dann den durchschnittlichen Verlust über die Fragmente, d.h., es ist empirische Risikominimierung (ERM), wobei das Ziel ein kleiner erwarteter Verlust auf Textfragmenten außerhalb des Trainingsdatensatz ist. Eine weitere Stufe ist das Feinabstimmung, das einen kleinen Datensatz manuell kuratierter Prompt-Antwort-Paare verwendet

(Ouyang et al., 2022). Die vortrainierten Modellparameter sind das, was Feinabstimmung an jede Aufgabe anpasst; ein LLM, dessen Modellparameter auf diese Weise über viele Lernaufgaben hinweg wiederverwendet werden, ist ein foundation model (Bommasani et al., 2021).

Die LLM-inference, d.h. die Berechnung der Response für einen gegebenen Prompt, ist rechnerisch wesentlich günstiger als ihr Training (Kaplan et al., 2020, Abschn. 2.1). Dennoch ist die inference selbst rechnerisch aufwendig und wird oft unter Verwendung von cloud computing-Diensten implementiert. Ein LLM kann auch lokal ausgeführt werden: Open-Weight-LLMs, deren Modellparameter veröffentlicht sind (Touvron et al., 2023), können durch Quantisierung komprimiert werden (Dettmers et al., 2022, Abschn. 1). Die Quantisierung speichert jeden Modellparameter in wenigen Bits anstelle einer 16-Bit-floating-point number. Ein LLM mit Milliarden von Modellparameter läuft dann auf einem Laptop oder Telefon (siehe edge computing).

Der Ausgabe eines LLM allein ist nichts als eine Folge von Buchstaben, die die Response bilden. Ein LLM- (oder Coding-) Agent interpretiert die Response eines LLM als Anweisungen für auszuführende Berechnungen (Yao et al., 2023). Zum Beispiel kann der Prompt das LLM anweisen, eine Response zu erzeugen, die gültiger Quellcode in einer Programmiersprache wie Python ist, wie im ausgeschriebenen Datenpunkt von Abb. 1. Der gelieferte Quellcode kann dann ausgeführt werden, und das Ergebnis kann an den nächsten Prompt angehängt werden. Die Response kann gleichermaßen als Aufruf eines externen Werkzeugs gelesen werden, sodass das LLM den Text generiert, der die Aktionen spezifiziert, die das umgebende Programm ausführt.

Siehe auch: Token, vocabulary, Transformer, Attention, KNN, Worteinbettung, softmax function, natural language processing (NLP), foundation model, Vortraining, Feinabstimmung, Agent, cloud computing, edge computing.

Querverweise

Literatur

1. Bengio et al. (2003). A Neural Probabilistic Language Model. J. Mach. Learn. Res., 3(Feb), pp. 1137–1155.

2. Bommasani et al. (2021). On the Opportunities and Risks of Foundation Models. arXiv preprint arXiv:2108.07258. URL: https://arxiv.org/abs/2108.07258.

3. Brown et al. (2020). Language Models are Few-Shot Learners. In: Adv. Neural Inf. Process. Syst., pp. 1877–1901.

4. Dettmers et al. (2022). LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale. In: Adv. Neural Inf. Process. Syst. (NeurIPS), pp. 30318–30332.

5. Devlin et al. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: Proc. 2019 Conf. North American

Chapter Assoc. Computational Linguistics: Human Lang. Technologies, Volume 1 (Long and Short Papers), pp. 4171–4186.

6. Kaplan et al. (2020). Scaling Laws for Neural Language Models. arXiv preprint arXiv:2001.08361.

7. Li et al. (2022). Diffusion-LM Improves Controllable Text Generation. In: Adv. Neural Inf. Process. Syst. (NeurIPS), pp. 4328–4343.

8. Ouyang et al. (2022). Training Language Models to Follow Instructions with Human Feedback. In: Adv. Neural Inf. Process. Syst. (NeurIPS), pp. 27730–27744.

9. Touvron et al. (2023). LLaMA: Open and Efficient Foundation Language Models. arXiv preprint arXiv:2302.13971. URL: https://arxiv.org/abs/2302.13971.

10. Yao et al. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. In: Int. Conf. Learn. Represent. (ICLR).

11. Vaswani et al. (2017). Attention is all you need. In: Adv. Neural Inf. Process. Syst., pp. 5998–6008.