LLM Workshop

Grigory Devadze

May 23, 2025

Einführung in LLMs

Was sind LLMs und wie funktionieren sie?

- LLM = Large Language Model, basierend auf Deep Learning, insbesondere der Transformer-Architektur (Vaswani et al., 2017)
- Kernmechanismus: Self-Attention
- Modelle lernen statistische Muster in Sprache und bilden Wahrscheinlichkeitsverteilungen
- Generieren Text durch Vorhersage des nächsten Tokens in einer Sequenz (autoregressive Generierung)
- Erfassen statistische Zusammenhänge, Bedeutungen und Strukturen in großen Textkorpora
- Werden auf Milliarden von Sätzen aus vielen Sprachen und Domänen vortrainiert
- Typische Aufgaben: Sprachverständnis, Sprachgenerierung, Übersetzung, Zusammenfassung, Codegenerierung, u.v.m.

Motivation: Warum LLMs?

- Menschen haben schon immer Texte geschrieben – unser Wissen und unsere Kommunikation sind größtenteils in Textform gespeichert
- Bereits heute gibt es riesige Mengen hochwertiger Textdaten, die genutzt werden können
- Immer mehr neue Textdaten: Webseiten, Chats, wissenschaftliche Literatur, Social Media
- Ziel: Maschinen sollen komplexe, natürliche Sprache verstehen, erzeugen und sinnvoll anwenden können

Motivation: Warum LLMs?

- Anwendungen: Chatbots, Übersetzer, Zusammenfassungen, Text-Analysetools
- Einfacheres (Vor-)Filtern, Analysieren und Strukturieren riesiger Informationsmengen
- Automatisierung vieler Aufgaben, wie Suchmaschinen-Optimierung, Kundenservice, medizinische Dokumentation, Codegenerierung
- Wettbewerbsdruck: KI-basierte Lösungen verschieben Marktstandards und Nutzererwartungen schnell

Grundprinzip: Transformer-Architektur

- Transformer-Architektur ersetzt RNNs/LSTMs durch Self-Attention und Positionsembeddings
- Jedes Token „achtet“ auf alle anderen Tokens im Kontext (Self-Attention)
- Multi-Head Attention: Mehrere parallele Attention-Mechanismen erfassen verschiedene Beziehungsarten
- Embedding Layer: Tokens werden in hochdimensionale Vektoren umgewandelt
- Positionsinformationen: Sinus- und Cosinusfunktionen kodieren die Reihenfolge der Tokens
- Feedforward-Layer, Residual-Verbindungen, Layer-Normalisierung stabilisieren und verbessern das Training

Tokenisierung

- Text wird in Tokens zerlegt (Wörter, Subwords, Zeichen)
- Subword-Tokenisierung (BPE, WordPiece, SentencePiece) ist Standard: Häufige Wörter bleiben intakt, seltene werden zerlegt
- Beispiel: “unbreakable” $\rightarrow$ [“un”, “break”, “able”]
- Tokenisierung beeinflusst Effizienz, Kosten und Kontextnutzung
- Vokabulargrößen: 32.000–100.000 Tokens (je nach Modell)
- Spezial-Tokens: <s>, </s>, <pad>, <unk>, <mask>, <bos>, <eos>
- Tokenisierungseffizienz variiert je nach Sprache (z.B. Deutsch: mehr Tokens pro Wort wegen Zusammensetzungen)

Self-Attention Mechanismus

- Jedes Token wird in drei Vektoren transformiert: Query (Q), Key (K), Value (V)
- Attention-Berechnung: Score = $Q \cdot K^T$, normalisiert durch Softmax, gewichtete Summe der Values
- Attention(Q,K,V) = $\text{softmax}(QK^T/d) \cdot V$
- Multi-Head: Modelliert verschiedene Kontextbeziehungen gleichzeitig

Autoregressive Generierung

- LLMs generieren Text Token für Token
- Bei jedem Schritt: Kontext $\rightarrow$ Wahrscheinlichkeitsverteilung $\rightarrow$ Auswahl des nächsten Tokens (greedy, sampling, beam search)
- Modelliert $P(\text{token}_t \mid \text{token}_1, \ldots, \text{token}_{t\text{-}1})$
- Softmax-Funktion wandelt logits in Wahrscheinlichkeiten um
- Temperatur-Parameter steuert Kreativität (niedrig = deterministisch, hoch = kreativ)

Kontext-Fenster und Skalierung

- Kontext-Fenster: Maximale Anzahl an Tokens, die das Modell verarbeiten kann (z.B. GPT-4: bis zu 128k Tokens)
- Aufmerksamkeitskomplexität: $O(n^2)$ mit Sequenzlänge n, limitierender Faktor für Kontextgröße
- Forschung zu effizienteren Attention-Mechanismen (Flash Attention, Sparse Attention)

Embedding Space

- Tokens werden in hochdimensionale Vektoren (z.B. 768–12.288 Dimensionen) abgebildet
- Semantische Nähe durch Vektorähnlichkeit (z.B. “König” - “Mann” + “Frau” “Königin”)
- Jede Transformer-Schicht transformiert diese Vektoren weiter

Emergente Fähigkeiten

- In-Context Learning: Lernen aus Beispielen im Prompt
- Chain-of-Thought: Schrittweise Problemlösung
- Instruction Following: Befolgen komplexer Anweisungen
- Diese Fähigkeiten entstehen emergent mit zunehmender Modellgröße

Limitierungen

- Keine echte Kausalität oder Weltverständnis, basiert auf statistischen Korrelationen
- Halluzinationen: Plausible, aber falsche Informationen
- Recency Bias: Übergewichtung neuerer Information im Kontext
- Keine dynamische Aktualisierung nach dem Training

Training von LLMs

- Selbstüberwachtes Lernen: Vorhersage des nächsten Tokens (Next Token Prediction) oder Masked Language Modeling (BERT)
- Trainingskorpus: Milliarden bis Billionen Tokens aus Webseiten, Büchern, wissenschaftlichen Artikeln, Code, Foren etc.
- Pre-Training: Modell lernt allgemeine Sprachmuster und Weltwissen
- Supervised Fine-Tuning (SFT): Anpassung an spezifische Aufgaben durch Beispiele
- RLHF (Reinforcement Learning from Human Feedback): Optimierung durch menschliches Feedback

Parameter und praktische Auswirkungen

- Parameter = "Gedächtnis" des Modells (abgespeicherte statistische Bedeutungen)
- Kleine Modelle: 1–10 Mrd. Parameter (Llama 3 8B, Mistral 7B)
- Mittlere Modelle: 10–100 Mrd. (Llama 3 70B, Mixtral 8x7B)
- Große Modelle: 100+ Mrd. (GPT-4, Claude 3 Opus)
- Mehr Parameter $\approx$ bessere Fähigkeiten (aber nicht linear)
- Größere Modelle: mehr Weltwissen, besseres Reasoning, komplexere Aufgaben
- Modellgröße, Datenmenge und Rechenleistung bestimmen die Leistungsfähigkeit (Scaling Laws, Kaplan et al. 2020)
- Emergente Fähigkeiten erscheinen erst ab kritischer Modellgröße

Hardware- und Infrastruktur-Anforderungen

- VRAM-Bedarf (Beispiele, FP16/4-bit):

Modellgröße	Präzision	VRAM benötigt
7B	FP16	~14 GB
7B	4-bit	~4 GB
70B	FP16	~140 GB
70B	4-bit	~38 GB

- Inferenz: Kleine Modelle (7B) auf Consumer-GPU (z.B. RTX 4090, 24 GB VRAM) möglich
- Training: Erfordert Cluster mit High-End-GPUs (A100, H100)
- Optimierung: Quantisierung (FP16, INT8, 4-bit), Modellparallelität, Offloading
- Software-Stack: PyTorch, Transformers, vLLM, GGML, llama.cpp, Text Generation Inference, LLaMA Factory

LLMs mit eigenen Daten verwenden – Drei Varianten

1. **Retrieval-Augmented Generation (RAG)**
   - Externe Wissensdatenbank (z.B. Vektordatenbank) wird mit eigenen Dokumenten befüllt
   - Dokumente werden vektorisiert (Embeddings) und bei Anfrage relevante Passagen abgerufen
   - LLM erhält Kontext aus der Datenbank und generiert darauf basierend die Antwort
   - Vorteile: Keine Modelländerung nötig, aktuelle und domänenspezifische Informationen möglich

LLMs mit eigenen Daten verwenden

2. **Fine-Tuning**
   - Vortrainiertes Modell wird gezielt auf eigene Daten/Anwendungsfälle weitertrainiert
   - Parameter-Efficient Fine-Tuning (PEFT): LoRA, QLoRA, Adapter-Layer
   - Benötigt sorgfältige Datenaufbereitung und Qualitätskontrolle
   - Relativ wenig Daten und kurze Trainingszeit reichen oft für große Effekte
   - Ergebnis: Modell liefert bessere Resultate für die gewünschte Aufgabe

LLMs mit eigenen Daten verwenden

3. **Prompt Engineering**
   - Optimierte Prompts, Few-Shot Learning, In-Context Learning, Chain-of-Thought Prompting
   - Keine Modelländerung, sondern gezielte Steuerung durch den Input
   - Effektiv für viele Aufgaben, insbesondere bei leistungsfähigen Modellen

Kommerzielle Modelle im Vergleich

- OpenAI: GPT-4, GPT-4 Turbo
  - Sehr hohe Leistungsfähigkeit, besonders bei komplexen Aufgaben, Reasoning, Code, Multimodalität (Bilder, Text)
  - GPT-4 Turbo: Schnellere und günstigere Variante, leicht geringere Qualität als GPT-4
- DeepSeek: DeepSeek V3, DeepSeek R1
  - V3: Starke Open-Source-Alternative, sehr gute Ergebnisse bei Textverständnis und Code
  - R1: Multimodale Fähigkeiten, hohe Effizienz, Open-Source-Ansatz
- Google: Gemini (Pro, Ultra), Bard
  - Gemini Ultra: Sehr großes Kontextfenster, starke Multimodalität, gute Integration in Google-Ökosystem

Kommerzielle Modelle im Vergleich

- Bard: Fokus auf Websuche und aktuelle Informationen, Integration in Google-Produkte
- Anthropic: Claude 3 Familie (Haiku, Sonnet, Opus)
  - Opus: Sehr großes Kontextfenster (200k+), hohe Sicherheit, starke Textverständnisfähigkeiten
  - Haiku/Sonnet: Schnellere, günstigere Varianten
- Cohere: Command
  - Fokus auf Enterprise-Anwendungen, hohe Zuverlässigkeit, Datenschutz

Open Source Modelle und deren Einsetzbarkeit

- Meta: Llama 3 (8B, 70B)
  - Sehr gute Open-Source-Modelle, starke Performance bei Text, Code, Reasoning
  - 8B: Für Edge/On-Premise, 70B: Für anspruchsvolle Aufgaben
- Mistral AI: Mistral 7B, Mixtral 8x7B
  - Mixtral: Mixture-of-Experts-Architektur, sehr effizient, starke Ergebnisse bei vielen Benchmarks
  - 7B: Sehr schnell, geringe Hardwareanforderungen
- Falcon, BLOOM
  - Falcon: Gute Performance, besonders bei englischem Text
  - BLOOM: Multilinguale Fähigkeiten, Open-Science-Ansatz

Open Source Modelle und deren Einsetzbarkeit

- DeepSeek V3, R1 (Open-Source-Variante)
  - V3: Sehr konkurrenzfähig zu GPT-3.5, Open-Source, gute Dokumentation
  - R1: Multimodal, Open-Source, für Forschung und Entwicklung geeignet
- Einsetzbarkeit:
  - Open-Source-Modelle sind günstiger, flexibler, können lokal betrieben werden (Datenschutz!)
  - Kommerzielle Modelle oft besser bei komplexen Aufgaben, aber teurer und mit API-Beschränkungen

Praxis-Test mit Prompts

- Standardisierte Prompts werden an verschiedene Modelle gegeben (z.B. Kreativaufgaben, Faktenabfrage, logische Schlussfolgerungen, Codegenerierung)
- Beispiel-Prompts:
  - "Schreibe eine kreative Kurzgeschichte über eine KI in der Zukunft."
  - "Erkläre den Unterschied zwischen Quantencomputern und klassischen Computern."
  - "Löse folgende Mathematikaufgabe: ... "
  - "Generiere Python-Code für eine Funktion, die ... "
  - "Fasse folgenden wissenschaftlichen Text zusammen: ..."

Reaktion der Modelle hinsichtlich Kreativität, Relevanz und Verständnis des Kontexts

- Kreativität: Wie originell und abwechslungsreich sind die Antworten?
- Relevanz: Wie gut passt die Antwort zur gestellten Frage?
- Kontextverständnis: Kann das Modell Informationen aus dem Prompt korrekt aufnehmen und weiterverarbeiten?
- Faktentreue: Sind die Antworten korrekt und nachvollziehbar?
- Sprachfluss: Wie natürlich und flüssig ist die Sprache?

Bias testen: Kontroverse Themen gezielt behandeln

- Prompts zu politischen, gesellschaftlichen oder ethischen Themen (z.B. Gender, Religion, Politik)
- Beobachtung: Wie neutral, ausgewogen oder voreingenommen sind die Antworten?
- Test: Werden bestimmte Meinungen bevorzugt? Gibt es Tendenzen zu bestimmten Sichtweisen?
- Dokumentation von auffälligen Biases und Halluzinationen

Bewertungsbogen mit Kriterien

- Kohärenz: Ist die Antwort logisch und in sich schlüssig?
- Genauigkeit: Stimmt die Antwort mit bekannten Fakten überein?
- Sprachfluss: Wie gut ist die sprachliche Qualität?
- Bias: Gibt es erkennbare Voreingenommenheiten?
- Kreativität: Wie originell ist die Antwort?
- Kontextverständnis: Wie gut wird der Kontext des Prompts erfasst?
- Ranking: Modelle werden nach diesen Kriterien bewertet (z.B. 1–5 Sterne pro Kriterium)

Fine-Tuning

Was ist Fine-Tuning?

- Der Prozess, ein vortrainiertes LLM punktgenau auf eine bestimmte Aufgabe oder einen speziellen Datensatz weiter anzupassen
- Erfolgt nach dem allgemeinen Pre-Training – Modell kennt Sprache schon, muss jetzt aber domänenspezifisch „Feinschliff“ bekommen
- Fine-Tuning ist nicht Pre-Training: Relativ wenig Daten & kurze Trainingszeit reichen oft für große Effekte!

Fine-Tuning

Warum wird Fine-Tuning gemacht?

- Vortrainierte Modelle sind sehr allgemein – sie „verstehen alles ein bisschen“, aber nichts 100% spezialisiert
- Für Aufgaben wie medizinische Texte, juristische Sprache, Programmcode o.Ä. braucht es domänenspezifisches Wissen
- Mithilfe von Fine-Tuning kann z.B. ein Chatbot speziell für Technik-Support, ein medizinisches Frage-Antwort-System oder ein Modell für deutsche Gesetzestexte entstehen

Fine-Tuning

Wie funktioniert Fine-Tuning praktisch?

1. Auswahl und Aufbereitung eines Task-spezifischen Datensatzes (z.B. Q&A, Chatprotokolle, Kategorisierungsdaten)
2. Modell wird für einige Epochen mit niedrigem Lernrate weiter trainiert $\rightarrow$ Modell „lernt“ Details oder Aufgabenlogik dazu
3. Overfitting vermeiden: Wenig Daten, daher gezielte Regularisierung und Monitoring nötig!
4. Ergebnis: Modell liefert bessere Resultate für die gewünschte Aufgabe, z.B. treffsicherere Antworten im Kundensupport

Fine-Tuning: Varianten und Einsatz

- Standard Fine-Tuning: Klassisches Weitertrainieren auf Taskdaten
- Instruction Tuning: Modell lernt, präzisen Anweisungen („User: ... System: ...“) zu folgen (z.B. Alpaca, ChatGPT)
- RLHF (Reinforcement Learning from Human Feedback): Nach Fine-Tuning weiterer Feinschliff durch menschliche Bewertungen/Präferenzen (z.B. Chatbots, sichere Textgenerierung)
- Adapter-Layer/LoRA: Modularisierte, ressourcenschonende Methoden, bei denen nicht das gesamte Modell verändert wird

Anpassung und Erweiterung bestehender Modelle

Briefing-Files

- System-Prompts und Briefing-Files steuern das Verhalten und die „Persönlichkeit“ eines LLMs.
- Beispiele:
  - Definition von Rollen („Du bist ein Marketing-Experte...“)
  - Vorgaben für Tonalität, Stil, Zielgruppe
  - Einsatz von Persona-Entwicklung für konsistente Antworten

Datenaufbereitung und Erstellung von Knowledge Files

- Datenaufbereitung:
  - Bereinigung, Anonymisierung, Formatierung der Daten
  - Qualitätskontrolle: Entfernen von Fehlern, Duplikaten, Bias
- Knowledge Files:
  - Strukturierte Wissenssammlungen (z.B. FAQs, Produktdatenblätter, interne Richtlinien)
  - Integration in Prompt-Ketten oder als Kontext für RAG
  - Ziel: LLM kann gezielt auf aktuelles und domänenspezifisches Wissen zugreifen

Beispiele

- AdBot für Anzeigen:
  - Automatisierte Generierung und Optimierung von Werbetexten für verschiedene Zielgruppen und Plattformen
- Data Analyst GPT:
  - Analyse von Unternehmensdaten, Generierung von Berichten, Visualisierungen und Handlungsempfehlungen
- Marketing GPT:
  - Erstellung von Kampagnen, Social-Media-Posts, E-Mail-Marketing-Texten, Zielgruppenansprache
- Social Media Poster:
  - Automatisierte Erstellung und Planung von Social-Media-Inhalten, Hashtag-Optimierung, Trend-Analyse
- Weitere Beispiele:
  - Juristischer Assistent, Medizinischer Chatbot, Technischer Support-Bot, HR-Assistant, u.v.m.

Komplett eigene LLMs trainieren

Ablauf

1. Datensammlung und -aufbereitung (Webscraping, Bücher, wissenschaftliche Artikel, Code, Foren, etc.)
2. Tokenisierung und Vokabularerstellung
3. Architektur-Design (Transformer, Anzahl Layer, Heads, Embedding-Größe, Parameterzahl)
4. Pre-Training auf riesigen Textmengen (Self-Supervised Learning)
5. Optional: Supervised Fine-Tuning auf spezifischen Aufgaben/Domänen
6. RLHF (Reinforcement Learning from Human Feedback) für Feinschliff
7. Evaluation und Benchmarking
8. Deployment (Bereitstellung als API, On-Premise, Cloud, Edge)

MosaicML

- Kommerzielle und Open-Source-Plattform für effizientes LLM-Training
- Features: Distributed Training, Checkpointing, Data Streaming, Optimierungen für Kosten und Geschwindigkeit
- Unterstützt Training von Modellen mit Hunderten Milliarden Parametern

Hugging Face Parquet Daten (Training)

- Parquet: Spaltenbasiertes, komprimiertes Datenformat, geeignet für große Textmengen
- Hugging Face Datasets: Viele offene Datensätze im Parquet-Format, einfaches Laden und Sharding

Databricks

- Plattform für verteiltes Datenmanagement und Machine Learning
- Unterstützt ETL, Datenaufbereitung, verteiltes Training, Monitoring
- Integration mit ML-Frameworks (PyTorch, TensorFlow, Hugging Face)

Fine-Tuning

- Nach dem Pre-Training: Anpassung an spezifische Aufgaben oder Domänen
- Daten: Eigene Beispiele, Unternehmensdaten, spezielle Aufgabenstellungen

Model Evaluierung und Deployment

- Evaluierung:
  - Benchmarks:
    - MMLU (Massive Multitask Language Understanding)
    - HELM (Holistic Evaluation of Language Models)
    - Eigene Tests
  - Menschliche Bewertung: Qualität, Faktentreue, Bias, Kreativität
  - Automatisierte Metriken: Perplexity (Güte der Wahrscheinlichkeitsverteilung), BLEU, ROUGE (Überschneidungen von Wortsequenze), Accuracy (Multiple-Choice- oder Klassifikationsaufgaben)
- Deployment:
  - Bereitstellung als API (REST, gRPC)
  - On-Premise, Cloud, Edge-Deployment