Conversational AI. Dialogsysteme, Chatbots, Assistenten

Veranstalter: Christoph Ringlstetter

Sitzung 2: LLMs

Was machen wir denn heute.

- Orga. Referate, Zulassung, Termine.
- Besprechung LLMs Basis: Jurafsky/Martin CH10
- Besprechung LLMs Transformer: Jurafsky/Martin CH9

Basis of the following introduction: the best introductory book on NLP

Speech and Language Processing. 3rd Edition 08/2024

CHAPTER

10 Large Language Models

"How much do we know at any time? Much more, or so I believe, than we know we know."

Agatha Christie, *The Moving Finger*

Fluent speakers of a language bring an enormous amount of knowledge to bear during comprehension and production. This knowledge is embodied in many forms, perhaps most obviously in the vocabulary, the rich representations we have of words and their meanings and usage. This makes the vocabulary a useful lens to explore the acquisition of knowledge from text, by both people and machines.

Estimates of the size of adult vocabularies vary widely both within and across languages. For example, estimates of the vocabulary size of young adult speakers of American English range from 30,000 to 100,000 depending on the resources used to make the estimate and the definition of what it means to know a word. What is agreed upon is that the vast majority of words that mature speakers use in their day-to-day interactions are acquired early in life through spoken interactions with caregivers and peers, usually well before the start of formal schooling. This active vocabulary (usually on the order of 2000 words for young speakers) is extremely

Einleitung.

Sprechen einer Sprache: Sprachproduktion/Sprachverstehen. V.a. Vokabular. Bedeutungen und Verwendung.

Aktives Vokabular von Kindern: ca 2000 Wörter <-> Erwachsene aktiv+passiv 30 – 100k

Lernrate von Kindern ca. 20 – 30 Wörter pro Tag.

Kipppunkt: „the rate of vocabulary growth exceeds the rate at which new words are appearing to the learner => das soll jetzt die distributional hypothesis unterstützen

=> akquirieren von Wissen aus Text: Pretraining. Das Resultat: LLM

Besonders transformativ für Aufgaben die Text erstellen müssen: Summarization, MT, QA, Chatbots

LLMs mit Transformern

gernerativeAI. Text generation, code generation, image generation.
Strategien: greedy decoding und sampling: jede NLP Task wird also Word Prediction eines LLMs modelliert.

Transformer (CH9) Elemente: transformer block, multi-head attention, language modelling head. Positonal encoding des Inputs.

LLMs mit Transformern

Ein transformerbasiertes LLM auf NLP Tasks anwenden: Bedingte Generierung
Erzeuge Text auf der Grundlage einer Inputsequenz => Kontext mehrere tausen Token lang. Das Modell hat jederzeit Zugang zum Kontext und zu allen bereits erzeugten Outputs.
Bsp. The sentiment of the sentence „I like Jackie Chan is: pos/neg

P(positive|I like…) or P(negative|I like…)

Bsp. Who wrote the book „The origin of species“
P(w| Who wrote the book: The origin of species) → Charles → feedback

LLMs mit Transformern

Summarization: LLM bekommt den text und ein Token: tl;dr -> too long didn‘t read. LLMs haben im pretraining das token oft genug vor einem Summary gesehen: können also Summarization gelernt haben.

Greedy Decoding: Erzeuge das wahrscheinlichste Wort gegeben der Kontext – lokal optimale Wahl – fast deterministisch: wenn der Kontext identisch ist wird dasselbe Wort erzeugt.

$$\hat{W}_t = \operatorname{argmax}_{w \in V} P(w \mid w_{<t})$$

Nicht für LLMs benutzt. V.a. in MT mit Beamsearch

Sampling zur Tokenerzeugung durch LLMs – Diversität

Decoding: wähle das nächste Wort basierend auf den Wahrscheinlichkeiten aller möglichen Wörter.
left to right – autoregressive generation, causal LM generation i.Ggs. Masked language models.

Sampling: gemäß der Verteilung eines Modells werden Zufallswörter gezogen. Iterativ gemäß der Wahrscheinlichkeit im Kontext. Indeterministisch.
=> wissen wir wie wir $P(w_t | w_{<t})$ aus dem LLM bekommen. Language Modelling Head.

Sampling zur Tokenerzeugung durch LLMs – Formalisierung

$x \sim P(x)$ chose x by sampling from distribution P(X)

i<-1

$w_i \sim P(w)$

while w_i != EOS

    i<-i+1

    $w_i \sim P(w_i | w_{<i})$

Random Sampling ist nicht optimal, da sehr unwahrscheinliche Wörter auch ausgewählt werden können -> long tail.

Sampling Methoden die sehr unwahrscheinliche Wörter vermeiden.

Sampling zur Tokenerzeugung durch LLMs – Adaptierte Methoden

Parameterisiertes Sampling: quality ⇔ diversity tradeoff.

Verstärkung der wahrscheinlichsten wörter. Korrekt, kohärent, faktisch passend aber langweilig, repetitiv.

Mittlere Wahrscheinlichkeit: Kreativ, Divers, falsch**.**

Top K Sampling: Abschneiden der Verteilung auf die Top k wahrscheinlichsten $w \in V$.

Renormalisieren auf eine Wahrscheinlichkeitsverteilung. Random Sampling von $P(X^K)$

Sampling zur Tokenerzeugung durch LLMs – Adaptierte Methoden

Nucleus oder Top-P Sampling. Das Problem der Top-K Methode. K ist statisch aber die Wahrscheinlichkeitsverteilung über die Wörter variiert je nach Kontext: mal ist nahezu die komplette Masse in den Top 10 mal mal ist die Verteilung flach => behalte die Top P Prozent der Wahrscheinlichkeitsmasse.

Temperature Sampling: Anpassung der Gestalt der Wahrscheinlichkeitsverteilung – Analogie zur Thermodynamik. Low temperature: smoothly steigern der Wahrscheinlichkeit der Wörter mit großer Warscheinlichkeit. Relativ verringern der Wahrscheinlichkeit der Wörter mit niedriger Wahrscheinlichkeit.

Y = softmax (u/$\boldsymbol{\tau}$) mit $\boldsymbol{\tau}$ < 1, 1, >1 und u logit Ausgabe des LLM Heads des Transformers

Pretraining Large Language Models – Algorithmus

Selfsupervised Training Algorithmus: Nimm einen Korpus und sage jeweils das nächste Wort vorher. Die natürliche Abfolge der Wörter der Text ist die Supervision – Objective.

Minimieren des Fehlers. Cross Entropy als Verlust. Differenz zwischen der vorhergesagten Verteilung und der korrekten Verteilung.

$L_{CE} = - \sum_{w \in V} y_t[w] \log \hat{y}_t[w]$. → indizes korrekt?

Language Modell: wir kennen das korrekte nächste Wort. One hot Vektor auf yt alle anderen sind 0

Cross Entropie wird durch die Wahrscheinlichkeitsmasse festgelegt die dem richtigen Wort zugemessen wird.

Pretraining Large Language Models -- Loss

CE Loss Simplified: negative logarithmierte Wahrscheinlichkeit welche das Modell dem nächsten Wort in der Trainingsequenz beimisst.

L_{CE} ($\hat{y}_t$ $y_{t)}$ = - log $\hat{y}_t$[w+1]. with $\hat{y}_{t}$ = $y_{t.}$

Jede Wortposition wt+1 nimmt als input die Sequenz w1:t aus dem Training. Truth.

Um den CE Loss zu berechnen Teacher Forcing anstatt nur die beste vorhergesagte Sequenz zu verwenden.

Pretraining Large Language Models – Training

Für jedes Item der Sequenz wir der Cross Entropy Loss berechnet.

Durchschnitt CE_Loss für die Batch.

Gewichte werden an den Durchschnitt CE_Loss über Grandient Decent angepasst: minimiere.

Jedes Training Item kann separat berechnet werden (anders RNN) => Parallelisierung möglich

Pretraining Large Language Models – Training Korpora

Riesig. Viele natürliche Beispiele für NLP Aufgaben, Analysen.

Training mit vollem Kontext Window. Common Crawl, Wikipedia, technical Bases, Books

Large Language Models – Safety and Quality Filtering

Quality Filter. Score auf jedes Trainingsdokument. Höher für Qualitätskorpora wie Wikipedia**.**
Vermeiden: PII Daten, Adult Content, Boilerplate: Formatierungscode etc
Duplikatentfernung.
Im Allgemeinen Verbesserung der Kuratierung. Vgl Llama Team 2024

Toxikalitäts-Erkennung: Modelle haben Klassifikationsfilter vorgeschaltet. Problem können dann toxischen Content nicht mehr so gut selbst entdecken.
Minderheitenausdrücke und Slang werden als toxisch erkannt.
Copyright Diskussion: „Fair Use“ Doktrin in den USA. Data consent, Privacy.

Large Language Models -- Fine Tuning

Domänen oder Aufgaben die nicht in den Pretraining Daten enthalten sind oder die verstärkt werden sollen. Gesundheitssektor, Rechtsthemen.

Multilingual e.g. Llama3 Sauerkraut -→ Link https://vago-solutions.ai/2024/04/23/neues-modell-llama-3-sauerkrautlm-8b-instruct/

Besondere Fähigkeiten des Modells: instruction finetuning

--> Arten

Einfach mit neuen Daten weiter trainieren

Arten: abhängig welche Parameter im Modell beeinflusst werden sollen: alle, einige, nur bestimmte Layer oder Circuits (?)

Alle: continued Pretraining. Langsam, teuer, catastrophic forgetting

Pretraining Large Language Models – Finetuning

Freeze: Parameter effizientes Finetunig. Nur ein Teil der Parameter wird trainiert. Bestimmte Layer etc.

Extrahead: Task spezifischer circuit. Nach dem Top Layer. Z.B. bei BERT eingesetzt. Masked Modell: auch heute für diskriminative Aufgaben manchmal besser und günstiger.

Supervised Finetunig –Instruction Finetuning: Für die sehr großen LLMs. Datenset aus Prompts und gewünschten Antworten: Q/A, Comands-Fullfilments

Pretraining Large Language Models – Evaluation von LLMs

Perplexity. Gute Modelle sind die die mit neuen Daten besser umgehen also ungesehenen Daten höhere Wahrscheinlichkeiten zuweisen.

Perplexität eines Models Θ auf ungesehenen Testdaten:
Inverse Wahrscheinlichkeit die Θ diesen Testdaten zuweist normalisiert durch die Länge des Testsets.

$$Perplexity_{\Theta}(w_{1:n}) = P_{\Theta}(w_{1:n})^{-1/n}$$

Um die Perplexität als Funktion der Wahrscheinlichkeiten entlang der Wörter des Testsets zu berechnen wird die Kettenregel angewandt.

Pretraining Large Language Models – Evaluation von LLMs

Kettenregel:

$$\text{Perplexity}_\theta(w_{1:n}) = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P_\theta(w_i|w_{<i})}}$$

Taskspezifische Maße: MT, Sentiment. Akkuratheit gegen Testkorpora.

Andere Faktoren: Wie gross, wie lange muss trainiert werden. Memory: GPU Fixkosten, Energieverbrauch

Bias: siehe Literatur, RealToxicityPrompts, BBQ

Leaderbords: Dynabench, HELM

Pretraining Large Language Models – Scaling

$$L(N) = \left(\frac{N_c}{N}\right)^{\alpha_N}$$
$$L(D) = \left(\frac{D_c}{D}\right)^{\alpha_D}$$
$$L(C) = \left(\frac{C_c}{C}\right)^{\alpha_C}$$

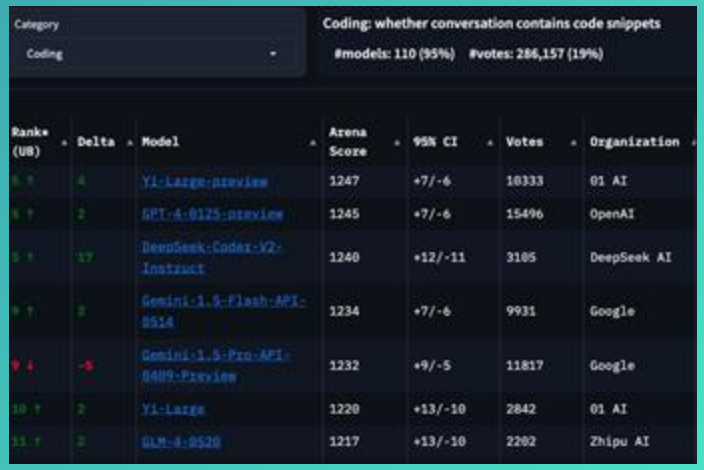
Llama3.1 405B 126 Layers, 128 heads.

15.6 Terabyte text |V| = 128k multilingual

Scaling laws für LLMs Kaplan 2020. Parameter, Daten, Cumputing Budget = GPU

Info zu Chinesischen Playern aus AI Report.

LLMSYS LEADERBORD



Pretraining Large Language Models – KV Cache: Verbesserung der Inferenz

Berechnung des Attention Vektors is sehr effizient machbar durch zwei Matrix Berechnungen siehe Kap. 9: $A = \text{softmax}(QK^T/d_k^{1/2})V$ alle gleichzeitig

Für Inferenz nicht durchführbar da wir iterativ das nächste Token erzeugen.

Für jedes neue Token x_i in der Sequenz müssen wir query, key und value über die Multiplikationen mit W^Q,W^K und W^V erzeugen.

Allerdings können Teile dieser Berechnungen gecacht werden.

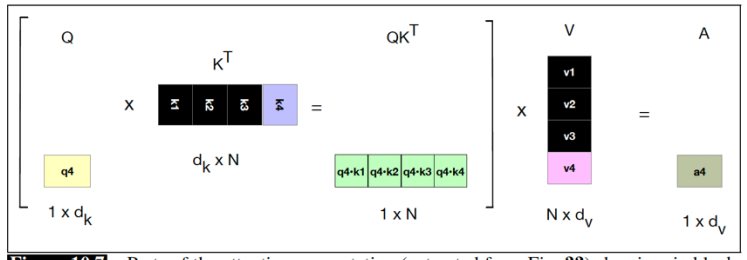


Figure 10.7 Parts of the attention computation (extracted from Fig. ??) showing, in black, the vectors that can be stored in the cache rather than recomputed when computing the attention score for the 4th token.

Jurafsky/Martin Kap. 10

Pretraining Large Language Models – Parameter Efficient Finetuning LORA

Geben dem LM zusätzliche Informationen zu einer neuen Domäne durch Finetuning. LLM→ Backpropagation durch 100 Layer, Milliarden von Parametern.
PEFT: nicht alle Parameter werden verändert.

LORA: Low-Rank-Adaptation.
Intuition: Transformer haben viele dichte (full connected) Layer die Matrix-Multiplikationen durchführen z.B. $W^Q W^K W^V W^O$ Attention Berechnung.
LORA: Diese einfrieren und stattdessen eine Low Rank Approximation mit neuen Daten updaten.

Pretraining Large Language Models --- Parameter Efficient Finetuning LORA

Brauchen Matrix Algebra: Dekompostion von Matrix W in Matrizen A und B mit A[Nxr] und B [rxd] und r << min [N,d]

Dann Finetuning W + ΔW –> W + BA

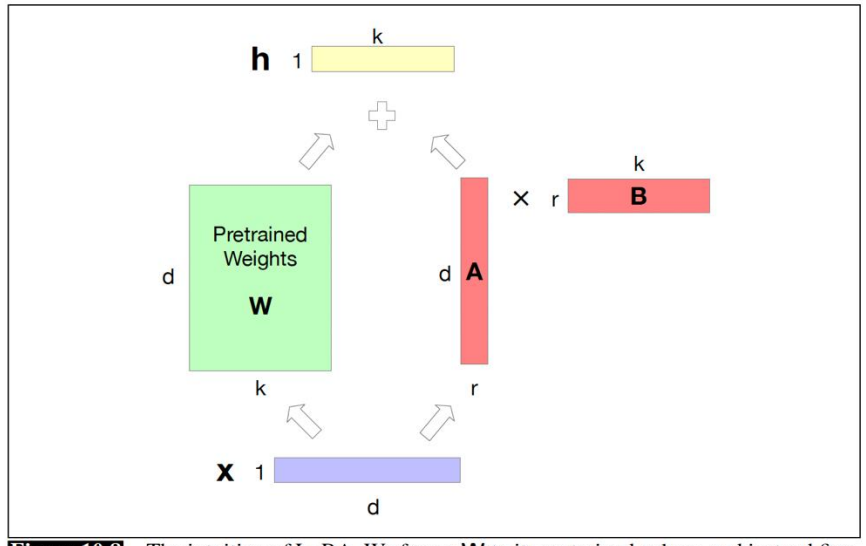


Figure 10.8 The intuition of LoRA. We freeze **W** to its pretrained values, and instead fine-tune by training a pair of matrices **A** and **B**, updating those instead of **W**, and just sum **W** and the updated **AB**.

Vorteile: Viel weniger Hardware Kosten

Forward Pass: h = xW + xAB

Weight Updates werden einfach zu c

Modularisierung: LORA Module könne addiert, subtrahiert, ausgetauscht werden

Large Language Models --- Potential Harms

Korrekthait see Kap 14 RAGs. bzw unsere Sitzung dazu

Toxicality, Stereotypen, negativer Bias

Copyright, Data Consent

PII -> Data Leaks

Missinformation durch böswillige Akteure

=> Eine Lösung: Modellcards mit exakter Aufschlüsselung der Trainingsdaten.

Replizierbar, Open Source

Pretraining Large Language Models – History

n-gram Modelle. Jelinek - IBM, Baker CMU.

MT, ASR

Then Toolkits Stolke Hatfield up to 5 –gram C++ libraries 2000s

Log Reg Predictor. Rosenfeld.

LSA/LSI Deerwester SVD on a term doc matrix – subsymbolic

First 300 dim vectors to represent words „embedding“

Benigo Neural language Models 2000,2003, 2006 next word as supervision signal

Extended to RNN 2010 surpassed 5-gram model Mikolov

2013/2013a word2vec use one hidden layer to create pretrained embeddings

Pretraining Large Language Models --- History

Static set used in furter tasks. Finetuning e.g. ELMO

Transformer Architecture Encoder – Decoder Vaswani „all you need“

2019 masked model BERT, Delvin

2019 autoregressive GPT2, Radford

Now: Vision, Speech Genetics etc Foundation Models

Chapter 9 – Jurafsky/Martin. Transformers

Transformers Jurafsky/Martin 9 – Introduction

Im Augenblick Standardarchitektur um LLMs zu erzeugen. Haben das Feld NLP völlig verändert.
Fokus: links-rechts, autoregressive, kausale Sprachmodellierung

Gegeben eine Sequenz von Eingabetoken werden die Ausgabetoken eins nach dem anderen vorhergesagt. Bedingung für die Modellierung ist der vorangegangene Kontext.

Transformers Jurafsky/Martin 9 – Introduction

Der Transformer ist ein neuronales Netz mit einer spezifischen Struktur die einen Mechanismus Names Self attention oder multi-headed Attention enthält.

=> Attention ermöglicht Kontextrepräsentation der Bedeutung eines Tokens. Integration der Bedeutung der Umgebung. Wie ist die Bezieung von Token über lange Distanzen hinweg.

Transformers Jurafsky/Martin 9 – Introduction

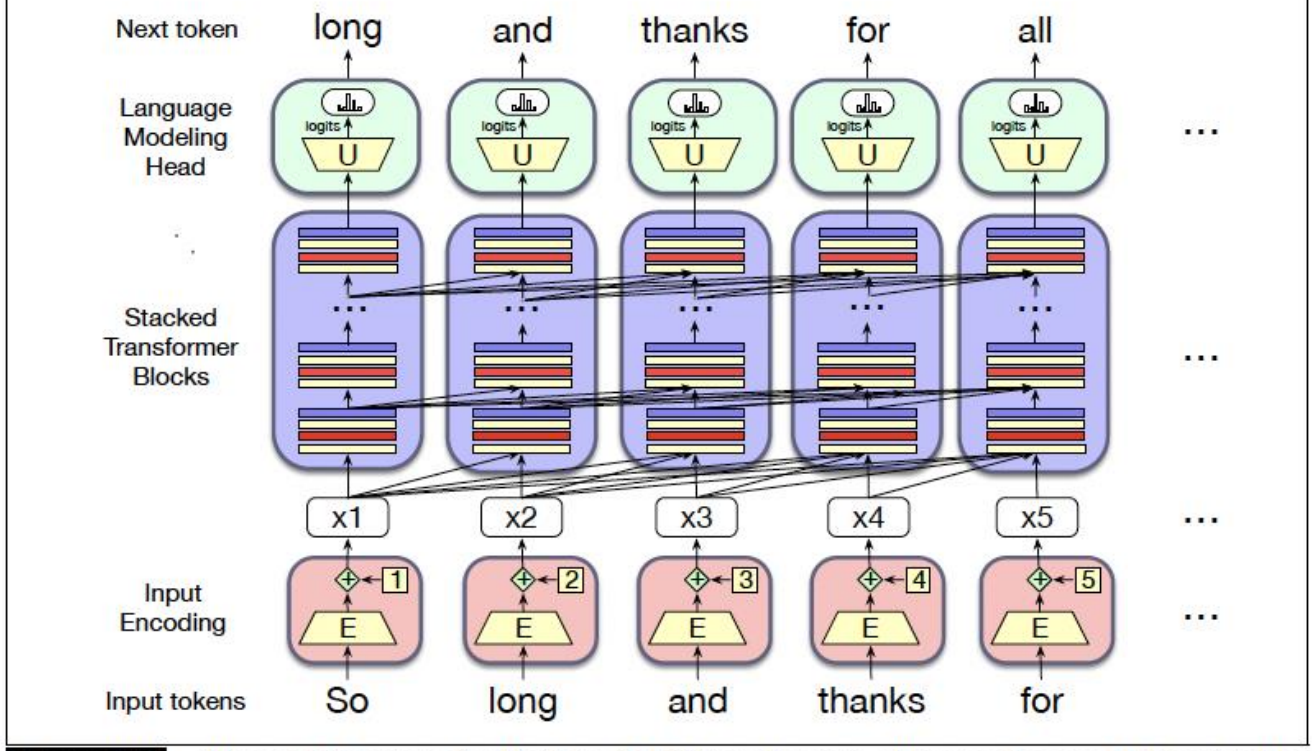


Figure 9.1 The architecture of a (left-to-right) transformer, showing how each input token get encoded, passed through a set of stacked transformer blocks, and then a language model head that predicts the next token.

Transformers Jurafsky/Martin 9 – Introduction

Vergleich zu Vorgängertechnologien: Word2Vec und andere statische Embeddings: Die Repräsentation eines Wortes bleibt unabhängig vom lokalen Kontext immer die selbe.

Pronomen Beispiel:
The Chicken didn‘t cross the road because it was too tired
The Chicken didn‘t cross the road because it was too wide

Links nach rechts Transformer: wenn wir it sehen kennen wir den entscheidenden lokalen Kontext noch nicht.

Transformers Jurafsky/Martin 9 – Introduction

Transformer erzeugen kontextualisierte Embeddings durch Attention.

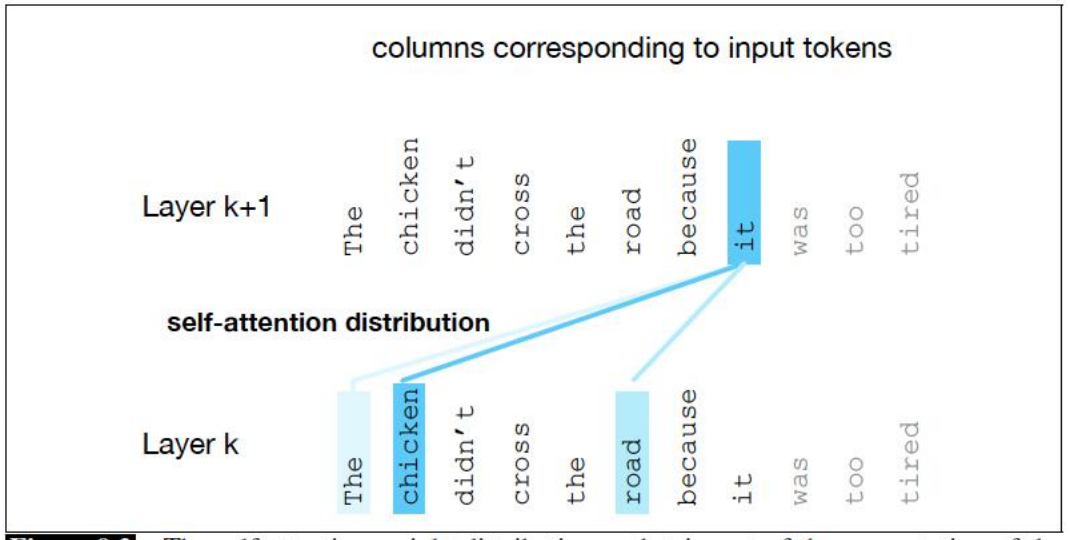


Figure 9.2 The self-attention weight distribution α that is part of the computation of the representation for the word *it* at layer $k+1$. In computing the representation for *it*, we attend differently to the various words at layer l, with darker shades indicating higher self-attention values. Note that the transformer is attending highly to the columns corresponding to the tokens *chicken* and *road* , a sensible result, since at the point where *it* occurs, it could plausibly corefers with the chicken or the road, and hence we'd like the representation for *it* to draw on the representation for these earlier words. Figure adapted from Uszkoreit (2017).

Transformers Jurafsky/Martin 9 – Attention formal

Attention formal: nimmt ein Input Token x_i und seinen vorangehenden Kontext x_1 … x_{i-1} und erzeugt einen Output a_i.

Der vorauslaufende Kontext kann ggf tausende von Token umfassen.

Attention Berechnung ist parallel für alle Token der Inputsequenz möglich.

$(x_1 \ldots\ldots x_n) \rightarrow (a_1,\ldots,a_n)$

Transformers Jurafsky/Martin 9 – Attention Simplifizierte Version

Attention als gewichtete Summe der Kontextvektoren.

$a_i = \sum_{j<=i} \alpha_{ij} x_j$

α_{ij} wieviel trägt x_j zur Attention von x_i bei.

Wie wird die Gewichtung α berechnet: proportional wie ähnlich das embedding des Vorgängers zum aktuell betrachteten Token ist.

Ähnlichkeit ist das Dot-Produkt das zwei Vektoren gleicher Dimension in einen Scalarwert $-\infty$ bis $+\infty$ mappt. Normalisierung dann per softmax.

$\alpha_{ij} = \text{softmax}(\text{score}(x_i, x_j))$ for all $j <= i$

und score $(x_i, x_j) = x_i \circ x_j$

Transformers Jurafsky/Martin 9 – Attention unsimplified

Single Attention Head: mit query, key und value Metrik. Dat Attention Head repräsentiert drei unterschiedliche Rollen die jedes Embedding im Attention Prozess spielt.

query: aktuelles Token verglichen zu den vorangegangenen.
key: vorangegangenes Token das mit dem aktuellen verglichen wird. Gewicht.
value: des vorangegangenen Inputs. Gewichtet und summiert um den Output des attention Mechanismus für den akuten String zu bestimmen.
w^Q, w^K, w^V → projektieren jeden input vektor xi in einen Repräsentanten seiner Rolle als key, query, value. $q_i = x_i\, w^Q$ | $k_i = x_i\, w^K$ | $v_i = x_i\, w^V$

Transformers Jurafsky/Martin 9 – Attention unsimplified

Verständnis: query und key um das Gewicht festzulegen. Value des vorhergehenden Elements der dann gewichtet aufsummiert wird um ai zu berechnen.

Gegeben die Projektionen: Berechne die Ähnlichkeit des aktuellen Elements xi mit einem vorhergehenden Element xj.

qi ° kj: muss skaliert werden in Bezug auf die Größe der Embeddings * $d_k^{1/2}$

dann αij = softmax (score (xi , xj)

und ai = $\sum_{j<=i} \alpha ij\ v_j$ in Dimension d_k Attentionvektor token x_i.

Transformers Jurafsky/Martin 9 – Attention Dimensionen

Input in die Attention ist xi und output ist ai

xi Dimension 1xd

output des transformer blocks und die vektoren der zwischenstadien (intermediate vectors) auch 1xd

Matrizen W^K, W^Q und W^V dxd_k

dk für query und key und value: im original paper 64, d = 512

dann wie zusammengerechnet? ai und xi : reshape Matrix W^O mit Dimension dk x d um wieder auf Dimension 1xd für ai zu kommen.

Transformers Jurafsky/Martin 9 – Attention Multihead Attention.

Intuition: jedes Head verarbeitet den Kontext von einem anderen Blickwinkel – different purposes: etwa verschiedene linguistisch beschreibbare Beziehungen zwischen Kontext und Current.

Die finalen Outputs der Heads am Ende des Transformers werden konkatiniert:

$a_i = (head^1 \oplus head^2 \oplus \dots head^n)\ W^O$

mit $W^O \in R^{hdvxd}$ **um auf Dimension [1xd] zurückzuführen.**

Transformers Jurafsky/Martin 9 – Transformer Blocks

Self Attention Berechnung ist der Kern eines Transformer Blocks.

Zusätzlich:

(1) Feedforward Layer

(2) Residual Connections => Residual Stream

(3) Normalisierungslayer

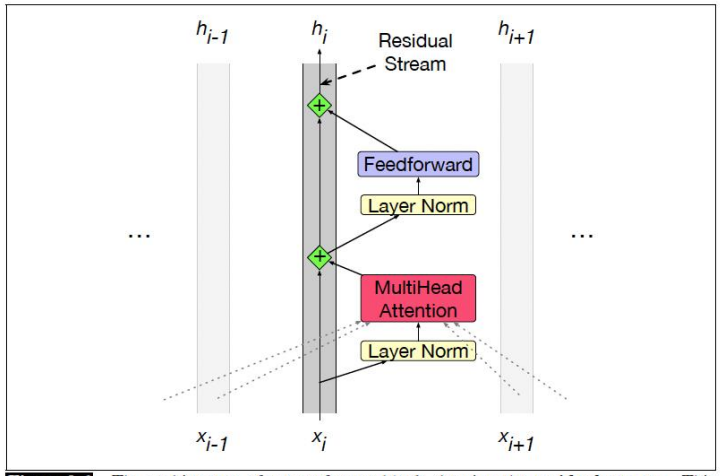


Figure 9.6 The architecture of a transformer block showing the **residual stream**. This figure shows the **prenorm** version of the architecture, in which the layer norms happen before the attention and feedforward layers rather than after.

Initialer Vektor für ein Token wird zunächst durch einen Normalisierungslayer verarbeitet, dann durch den Attention Layer dann wird der zum Input Vektor addiert und erneut normalisiert.

Transformers Jurafsky/Martin 9 – Transformer Blocks

Feedforward Layer: 2-layer Netzwerk mit dff > d z.B. 2048

$FFN(xi) = ReLU(x_iW_1+b_1)\ W_2 + b_2$

Normalisierung: hält die Werte im günstigen Bereich für gradientenbasierte Optimierung.
Variation des z-score aus der Statistik: Vektor des Token wird über Mittelwert und Standarabweichung sowie zwei lerbare Parameter normalisiert.

Transformers Jurafsky/Martin 9 – Transformer Blocks

Alles zusammen.

t^1_i = Layer Norm (x_i)

t^2_i = Multihead Attention (t^1_i [x^1_1 ….. x^1_{i-1}])

t^3_i = Add t^2_i + x_i

t^4_i = Layer Norm (t^3_i)

t^5_i = FFN(Layer Norm (t^4_i)

$h_i = t^5_i + t^3_i$

Information von aussen nur durch Multi-Head Attention Tokens.

Transformers Jurafsky/Martin 9 – Transformer Blocks

„We can view attention heads as literally moving information from the residual stream of a neighboring token into the current stream“

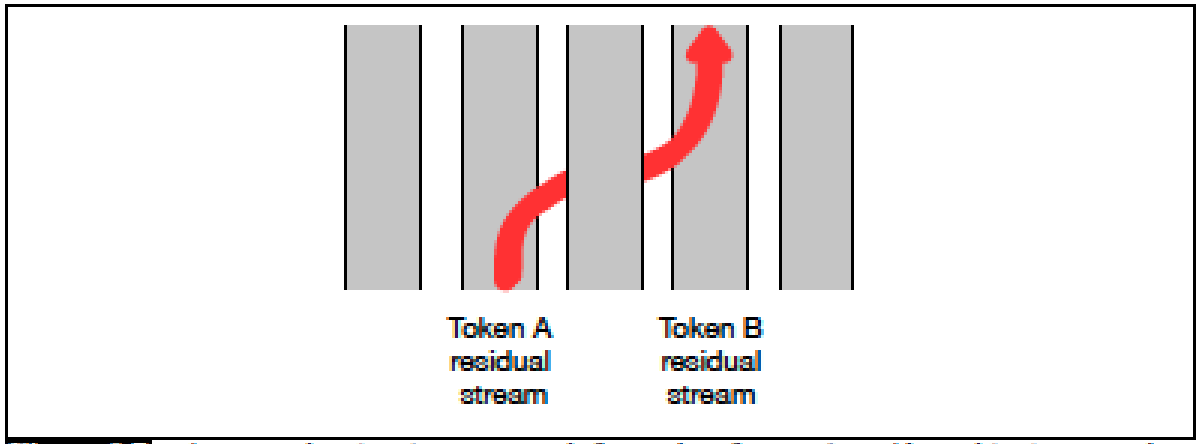


Figure 9.7 An attention head can move information from token A's residual stream into token B's residual stream.

Transformers Jurafsky/Martin 9 – Transformer Blocks

Stacking: Voraussetzung ist das input xi und output hi die selbe dimesion haben dadurch output wieder input für den nächsten Transformer Layer.
GPT-3 large: 96 layer.

Für die ersten Transformer Blocks repräsentiert der Residual Stream das aktuelle Token. In den höheren Blocks repräsentiert er schon das folgende Token da ja das Training immer auf die Vorhersage des nächsten Token abziehlt.

Transformers Jurafsky/Martin 9 – Parallelisierung der Berechnung

Ausgangspunkt: die Attentionberechnung für jedes Token ai ist unabhängig von der für alle anderen Token.

Damit kann die gesamte Berechnung des Blocks parallelisiert werden.

=> Matrix Routinen.

X |Nxd| mit N z.B. 32k Token. Jede Reihe das Embedding eines Tokens Dimension d

Spezifitäten: QK^T würde auch Ähnlichkeitswerte für Token erzeugen die der Query folgen => Mask Matrix die die rechtsliegenden Werte ausschaltet.

Transformers Jurafsky/Martin 9 – Parallelisierung der Berechnung

Attention ist quadratisch zur Länge des Inputs. Für jeden Layer müssen die Dot Products zwischen allen Input Paaren berechnet werden. Langer Kontext teuer.

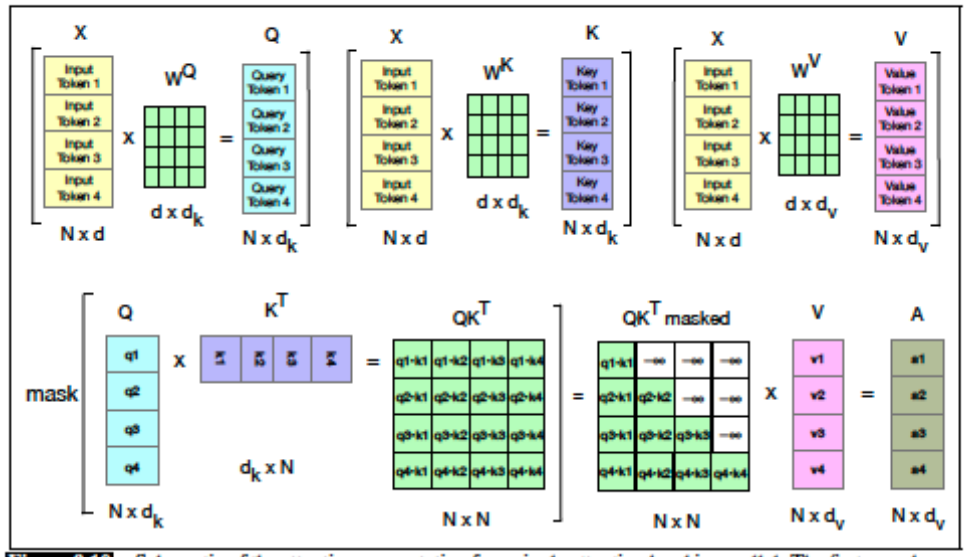


Figure 9.10 Schematic of the attention computation for a single attention head in parallel. The first row shows the computation of the **Q**, **K**, and **V** matrices. The second row shows the computation of $\mathbf{QK}^T$, the masking (the softmax computation and the normalizing by dimensionality are not shown) and then the weighted sum of the value vectors to get the final attention vectors.

Transformers Jurafsky/Martin 9 – Position Encodings

Wo kommt denn der X input überhaupt her?

Gegeben die Sequenz von N tokens (Kontextlänge)

X hat ein Embedding für jedes Token X:= |Nxd|

Der Transformer spaltet das Embedding: token e | position e

Das Token Embedding ist ein initiales Embedding das sich infolge des LLM trainings verändert.

Initiale token Embeddings werden aus dem Ebeddingstore geladen

E := |Vxd|

Nach Tokenisierung entweder Indizes oder one-hot Kodierung.

Transformers Jurafsky/Martin 9 – Position Encodings

Erzeugung der Input Matrix über one hot Vektoren und die ursprüngliche Embeddingmatrix.

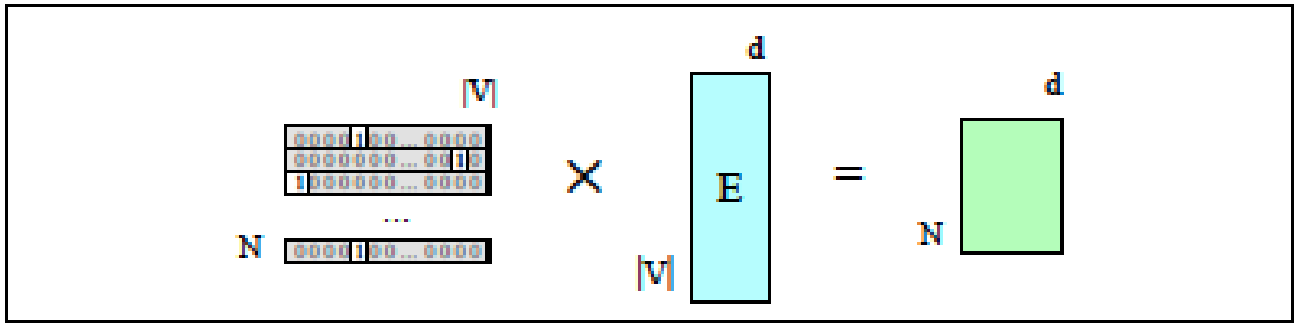


Figure 9.12 Selecting the embedding matrix for the input sequence of token ids W by multiplying a one-hot matrix corresponding to W by the embedding matrix $\mathbf{E}$.

Transformers Jurafsky/Martin 9 – Position Encodings

Kombination der Token Encodings mit Position Encodings um eine Inputsequenz zu bilden.

Einfachste Methode: Absolute Position.

Start mit random initialisierten postion Embeddings 1 …N die dann mitgelernt werden.

Matrix E_{pos} [1xN]

Addieren dann Input und Position.

Komplexer: Funktion die Entfernung zwischen den Token mit sin/cos Funktionen einfängt.

Noch komplexer relative Pos.

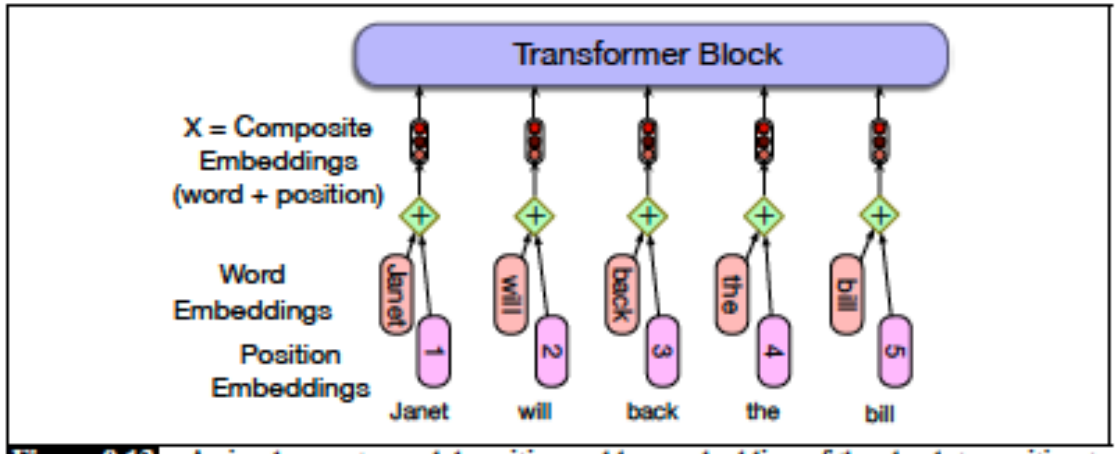


Figure 9.13 A simple way to model position: add an embedding of the absolute position to the token embedding to produce a new embedding of the same dimensionality.

Transformers Jurafsky/Martin 9 – Language Modelling Head

P(fish|Thanks for all the) : Distribution over V.

Nimm den Output der obersten Transformerschicht vom letzten Token N und sage N+1 vorher.

Also verwende das 1xd Token Embedding und erstelle die Wahrscheinlichkeitsverteilung über V an Position N+1.

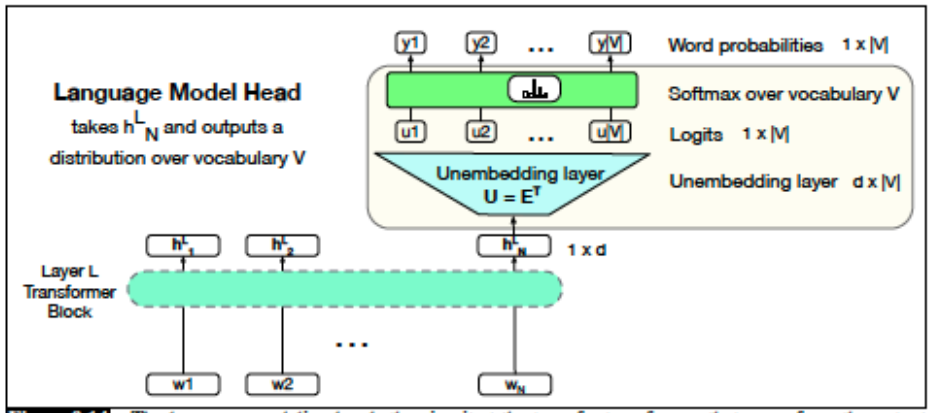


Figure 9.14 The language modeling head: the circuit at the top of a transformer that maps from the output embedding for token N from the last transformer layer ($\mathbf{h}^L_N$) to a probability distribution over words in the vocabulary V.

Transformers Jurafsky/Martin 9 – Language Modelling Head

Unembedding Layer:

$u = h^L_N E^T$ => Ähnlichkeit zu jedem $v \in V$

$y = \text{softmax}(u)$

Dann => Wordsampling. Ziehe ein Wort aus der Verteilung.

Transformers Jurafsky/Martin 9 – Language Modelling Head

Logit Lens: Können einen beliebigen Vektor aus dem Stack ziehen und eine sozusagen verfrühte Multiplikation mit der Embedding Matrix durchführen um zu sehen welche Verteilung zu diesem Punkt im Residual Stream vorliegt.

Transformers Jurafsky/Martin 9 – Summary

- Transformer sind Netzwerke basierend auf Multi-Head Attention: ein Eingabevektor xi wird zu einem Output ai durch gewichtetes einrechnen von Embeddings vorhergehender Tokes.
- Ein Transformer Block besteht aus einem Residual Stream durch den der Input aus einem vorhergehenden Layer dem nächsten Layer übergeben wird indem der Output verschiedener Komponenten addiert wird.
Multihead Attention + FFL + Layer normalization. Blöcke werden gestackt.
- Der Input eines Transformers ist das Token Embedding addiert zu einem Positional Encoding (repräsentiert die Position des Tokens in der Sequenz)
- LLMs werden aus Stacks von Transformerblöcken gelernt. Ein language model head bildet den Abschluss. Es verwendet eine Unembedding Matrix und den Output des Toplayers um mit dem Softmax Wortwahrscheinlichkeiten zu erzeugen.
- Transformerbasierte LLMs haben riesige Kontext Windows (Max 32768 tokens) konnen riesigen Kontext konsultieren um das nächste Wort vorherzusagen.

Decoder-Only or Encoder-Decoder? Interpreting Language Model as a Regularized Encoder-Decoder

Zihao Fu[1], **Wai Lam**[2], **Qian Yu**[3], **Anthony Man-Cho So**[2]
Shengding Hu[4], **Zhiyuan Liu**[4], **Nigel Collier**[1]

[1]Language Technology Lab, University of Cambridge,
[2]The Chinese University of Hong Kong, [3]JD.com
[4]Department of Computer Science and Technology, Tsinghua University
{zf268,nhc30}@cam.ac.uk, {wlam,manchoso}@se.cuhk.edu.hk
{hsd20, liuzy}@mails.tsinghua.edu.cn, yuqian81@jd.com

Our analysis is based on the prevalent Transformer architecture (Vaswani et al., 2017; Ott et al., 2019). As illustrated in Figure 2, we show the main components of the models and the detailed model structure can be found in Ott et al. (2019). In a seq2seq task, we denote the input source sequence as $s = [s_1, s_2, \cdots, s_{|s|}]$ and the target sequence as $t = [t_1, t_2, \cdots, t_{|t|}]$, where $|\cdot|$ is the length of the sequence and s_i is the ith source word. The positional token for s and t is denoted as $p_s = [1, 2, \cdots, |s|]$ and $p_t = [1, 2, \cdots, |t|]$. The attention layer is denoted as $\texttt{ATT}(Q, K, V)$, and $Z = \texttt{ATT}(Q, K, V) = \texttt{Softmax}(QW_Q W_K^\top K^\top / \sqrt{d}) V W_V = \texttt{Softmax}(QA^\top K^\top / \sqrt{d}) V W_V$, in which $A^\top = W_Q W_K^\top$ and $W_Q, W_K, W_V \in \mathbb{R}^{d \times d}$ are trainable parameters that transform input matrices into another space. $Q \in \mathbb{R}^{d_Q \times d}, K \in \mathbb{R}^{d_K \times d}, V \in \mathbb{R}^{d_V \times d}$ stand for the query, key and value matrices, d_Q, d_K, and d_V are the corresponding dimensions of the matrices while $Z \in \mathbb{R}^{d_Q \times d}$ is the output of the attention layer. Here, the $\texttt{Softmax}$ operation is applied to each row of the matrix concerned.

Atttention Degeneration: Measurement Setup

We begin our analysis by defining a notion of sensitivity. Intuitively, given a function $y = f(x)$, the sensitivity of x on y can be described as how the output vector y changes (Δy) when imposing a pertubation Δx on the input vector x. However, imposing different Δx leads to different Δy, we propose to study the upper bound on the ratio between the magnitude of x and y based on the following proposition:

Proposition 3.1. *Given a function $y = f(x)$ with a Jacobian matrix J_f, if we have a pertubation vector Δx and $y + \Delta y = f(x + \Delta x)$, then*

$$\frac{\|\Delta y\|}{\|\Delta x\|} \leq \|J_f\| + o(1). \tag{1}$$