

Stat 9911

Principles of AI: LLMs

Large Language Model Architectures 01

Edgar Dobriban

Department of Statistics and Data Science, the Wharton School, University of Pennsylvania

January 23, 2025



Plan

- ▶ We will discuss LLM architectures, and specifically attention and transformers.

Table of Contents

Large Language Model Architectures

Input and Output

- ▶ **Input:** Text
- ▶ **Output:** Text
- ▶ Examples: Question-answering ($Q \rightarrow A$), translation, summarization ($X_1, \dots, X_n \rightarrow S$), etc.
- ▶ Key strength: Generic format, covers many tasks.
- ▶ Origins in sequence-to-sequence (seq2seq) modeling ([Sutskever et al., 2014](#)).

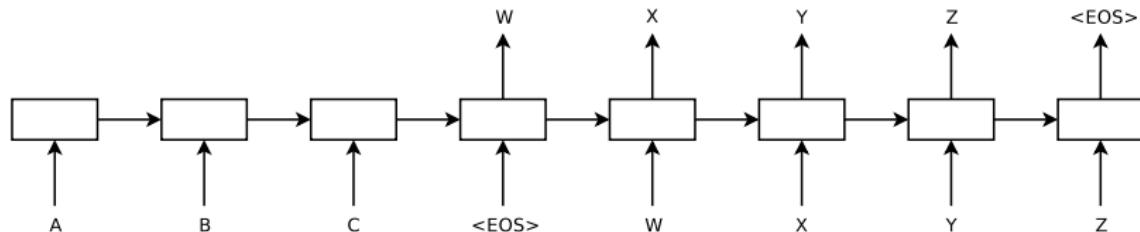


Figure: seq2seq ([Sutskever et al., 2014](#))

Operationalizing Input and Output

- ▶ Represent text as a sequence $S = (s_1, s_2, \dots, s_k)$ of symbols s_j from a fixed universe.
- ▶ Use autoregressive next-symbol/word prediction to reduce the task to supervised learning.
- ▶ Break the task into smaller subtasks:
 - ▶ $S \rightarrow o_1$
 - ▶ $S, o_1 \rightarrow o_2$
 - ▶ ...
 - ▶ $S, o_1, \dots, o_{m-1} \rightarrow o_m$

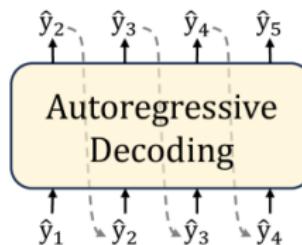


Figure: From Xia et al., 2022

Advantages and Disadvantages of the Approach

Advantages:

- ▶ Generic format.
- ▶ Predicting the next symbol one at a time is simpler.

Disadvantages:

- ▶ Sequential/Autoregressive processing is non-parallelizable.
- ▶ No efficient mechanism for backtracking mistakes. [Research direction: Diffusion LMs.]
- ▶ Exponentially decreasing correctness probability.

Drawbacks of Prediction Approach

- ▶ No immutable knowledge base or built-in behavior guarantees.
- ▶ No precise specifications, guarantees, or performance analysis.
Unlike much of
 - ▶ classical CS/engineering (e.g., programming languages, databases),
 - ▶ stats (probabilistic guarantees, e.g., confidence interval coverage).
- ▶ The popular phenomenon of "emergence" refers exactly to unpredictable behavior that appears spontaneously during training.
- ▶ Interventions via modifying representations still do not resolve problems.

Symbol Representation

- ▶ Options for symbols:
 - ▶ Letters/Characters (e.g., 'a', '+', '1').
 - ▶ Words (e.g., 'cat', 'dogs').
- ▶ Considerations
 1. The smallest symbol set could be characters (e.g., 255 ASCII).
 2. Could use words, a much larger set (e.g., 100K words across 1K languages).
 3. There is often redundancy in word-based systems. For example, words like 'cat' and 'cats' or 'dog' and 'dogs' are closely related, and the system should know this.
 4. This motivates an intermediate basic unit: tokens

Compromise for Symbol Representation: Tokens

- ▶ **Tokens** are subwords (e.g., ‘subwords’ → ‘[sub][word][s]’).
- ▶ See e.g., [OpenAI tokenizers](#)
- ▶ How construct them? Start with base vocab of chars. Find groupings of frequent pairs of characters, add them to vocab, repeat.
- ▶ See efficient tokenization methods like Byte Pair Encoding (BPE), [Sennrich et al. \(2016\)](#).
- ▶ Today’s vocabulary sizes: 50,000 to 100,000.
- ▶ Key to adapting LLM architectures to other modalities is to define tokens. E.g., for images define the tokens as ... patches.



Figure: Tokenization in Vision Transformers ([Dosovitskiy et al., 2021](#))

Issues Due to Tokenization

- ▶ Sub-token operations can be surprisingly hard: "How many rs are there in strawberry?" (More sophisticated version needed as of Jan 2025: how many xs
insswedurhwsiednsdfssjsjsaadsdadaszasazzxasxasdasdadwea)
- ▶ Inserting one space may disrupt model behavior (http:// vs http: //)
- ▶ See Andrej Karpathy's [video](#)

Learning a Next-Token Predictor

- ▶ Given any input text $x_{1:i-1}$, want to predict next token x_i .
- ▶ We will learn a predictor from a large dataset/corpus D of text.
- ▶ Data? The internet: Wikipedia, Arxiv, Reddit, Stack Exchange, other websites, books

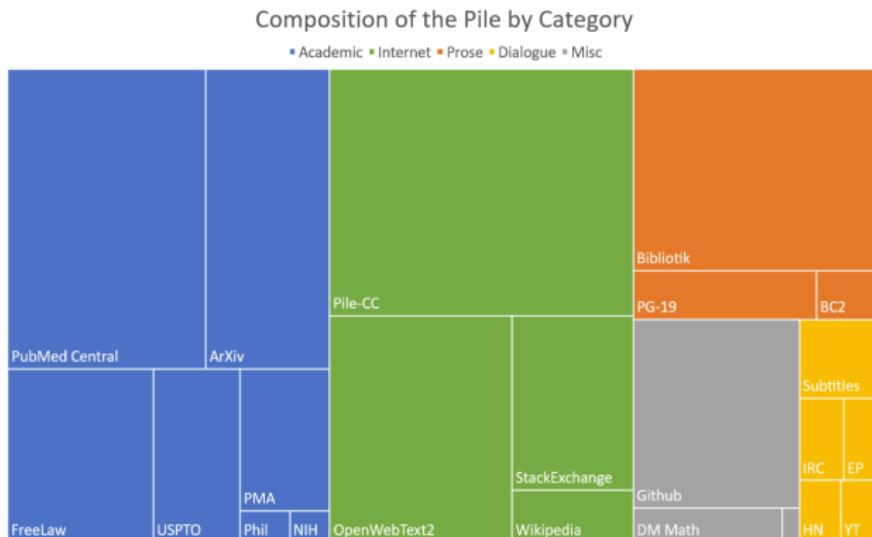


Figure: The Pile (800 GB) (Gao et al., 2020)

Aside: Copyrighted Training Data

- Data is known to (or allegedly does) include copyrighted and pirated content, e.g., the LibGen 'shadow library' (millions of books). The biggest IP theft in history?

With just two hours left before the fact discovery cut-off on Friday, December 13, 2024,

Meta produced some of the most incriminating internal documents it has produced to date relevant to Meta's copyright infringement claim and fair use defense, as well as Plaintiffs' proposed new claims. These documents concern Meta's torrenting and processing of pirated copyrighted works, including that: Meta's CEO, Mark Zuckerberg, approved Meta's use of the LibGen dataset notwithstanding concerns within Meta's AI executive team (and others at Meta) that LibGen is "a dataset we know to be pirated," Stein Reply Decl. ("Reply Ex."), Ex. A at 211699, 211702; top Meta engineers discussed accessing and reviewing LibGen data but hesitated to get started because "torrenting from a [Meta-owned] corporate laptop doesn't feel right 😊," Reply Ex. B at 204224; one of those engineers "filtered . . . copyright lines" and other data out of LibGen to prepare a CMI-striped version of it to train Llama, Reply Ex. C at 204220-21; and, by January 2024, Meta had already torrented (both downloaded and distributed) data from LibGen, Reply Ex. D.¹ And just yesterday, when asked about the type of piracy described in the TACC, Mr. Zuckerberg testified that such activity would raise "lots of red flags" and "seems like a bad thing." Reply Ex. E (Zuckerberg Dep. Tr.) at 102:10-14; 98:24-99:2.

Figure: From court case Kadrey vs Meta, via X

Empirical Next-Token Predictor

- ▶ Use empirical probability estimate (Shannon, 1948):

$$P(x_i | x_{1:i-1}) = \frac{\#(x \in D : x = x_{1:i})}{\#(x \subset D : x = x_{1:(i-1)})}$$

We can also approximate to a natural language by means of a series of simple artificial languages. The zero-order approximation is obtained by choosing all letters with the same probability and independently. The first-order approximation is obtained by choosing successive letters independently but each letter having the same probability that it does in the natural language.⁵ Thus, in the first-order approximation to English, E is chosen with probability .12 (its frequency in normal English) and W with probability .02, but there is no influence between adjacent letters and no tendency to form the preferred digrams such as TH, ED, etc. In the second-order approximation, digram structure is introduced. After a letter is chosen, the next one is chosen in accordance with the frequencies with which the various letters follow the first one. This requires a table of digram frequencies $p_i(j)$. In the third-order approximation, trigram structure is introduced. Each letter is chosen with probabilities which depend on the preceding two letters.

Figure: Shannon (1948)

Empirical Next-Token Predictor

3. THE SERIES OF APPROXIMATIONS TO ENGLISH

1. Zero-order approximation (symbols independent and equi-probable).
XFOML RXKHRJFFJUJ ZLPWCFWKCYJ
FFJEYVKCQSGXYD QPAAMKBZAACIBZLHJQD
2. First-order approximation (symbols independent but with frequencies of English text).
OCRO HLI RGWR NMIELWIS EU LL NBNSESBYA TH EEI
ALHENHTTPA OOBTTVA NAH BRL
3. Second-order approximation (digram structure as in English).
ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY
ACHIN D ILONASIVE TUCCOWE AT TEASONARE FUSO
TIZIN ANDY TOBE SEACE CTISBE
4. Third-order approximation (trigram structure as in English).
IN NO IST LAT WHEV CRATICT FROURE BIRS GROCID
PONDENOME OF DEMONSTURES OF THE REPTAGIN IS
REGOACTIONA OF CRE

5. First-Order Word Approximation. Rather than continue with tetragram, ⋯, n -gram structure it is easier and better to jump at this point to word units. Here words are chosen independently but with their appropriate frequencies.

REPRESENTING AND SPEEDILY IS AN GOOD APT OR
COME CAN DIFFERENT NATURAL HERE HE THE A IN
CAME THE TO OF TO EXPERT GRAY COME TO FUR-
NISHES THE LINE MESSAGE HAD BE THESE.

6. Second-Order Word Approximation. The word transition probabilities are correct but no further structure is included.

THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH
WRITER THAT THE CHARACTER OF THIS POINT IS
THEREFORE ANOTHER METHOD FOR THE LETTERS
THAT THE TIME OF WHO EVER TOLD THE PROBLEM
FOR AN UNEXPECTED

The resemblance to ordinary English text increases quite noticeably at
the end of the above steps.

Figure: Shannon (1948)

Challenges with Empirical Next-Token Predictor

- ▶ Empirical probability estimate

$$P(x_i|x_{1:i-1}) = \frac{\#(x \in D : x = x_{1:i})}{\#(x \subset D : x = x_{1:(i-1)})}$$

has some issues:

- ▶ Undefined for strings $x_{1:i-1} \notin D$.
- ▶ In particular, it is undefined for small semantic reformulations.

Alternative Approach

- ▶ Bengio et al. (2000) learn a probabilistic model with flexible, but restricted parameterization.
- ▶ Predict the next symbol/word in a sequence using a predictor function $f : V^{i-1} \rightarrow V$. At the outset, natural to think of deterministic predictors.
- ▶ Challenges:
 - ▶ Most natural loss function $I(f(x_{1:i-1}) \neq x_i)$ is discontinuous.
 - ▶ Discrete space of functions $\mathcal{F} : V^{i-1} \rightarrow V$ also discontinuous.

Machine Learning Solution

- ▶ Relax function space to output probability distributions over V . Model $x \mapsto P(x) = (P_1(x), \dots, P_{|V|}(x))$ becomes a probability distribution over V , $P(x) \in \Delta(V)$, for all x .
- ▶ Introduce a loss function for multi-class classification, where the outcome is $v \in V$, and the prediction is a probability distribution over V . E.g., logarithmic scoring rule:

$$\ell(P, v) = -\log P_v$$

- ▶ Aggregate over multiple data points by computing the empirical risk:

$$\sum_{x \in D} \sum_{j=1}^{|x|} \ell(P(x_{1:(j-1)}), x_j)$$

Machine Learning Solution

- ▶ For the log-loss, becomes

$$-\sum_{x \in D} \sum_{j=1}^{|x|} \log(P(x_{1:(j-1)})_{x_j}) = -\sum_{x \in D} \log(\tilde{P}(x))$$

where $\tilde{P}(x) = P(x_{1:(|x|-1)})_{x_{|x|}}$ is the probability of the string x under the language model

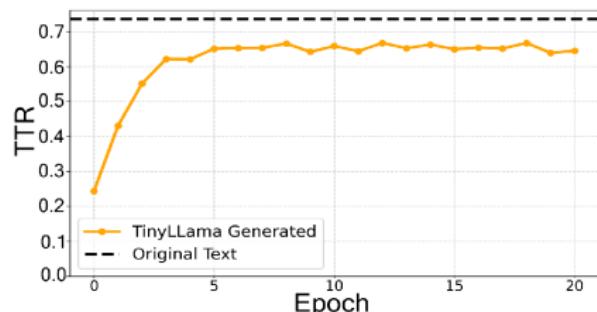
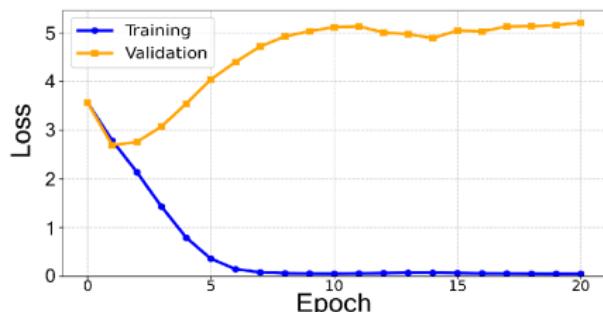
- ▶ Minimizing it over some class of P s amounts to maximum likelihood estimation

Accuracy vs Loss

- ▶ Ultimately we care about the accuracy. The loss is a differentiable surrogate. The two are not necessarily monotonically related, and this can lead to unexpected behaviors

THE HYPERFITTING PHENOMENON: SHARPENING AND STABILIZING LLMs FOR OPEN-ENDED TEXT GENERATION

This paper introduces the counter-intuitive generalization results of overfitting pre-trained large language models (LLMs) on very small datasets. In the setting of open-ended text generation, it is well-documented that LLMs tend to generate repetitive and dull sequences, a phenomenon that is especially apparent when generating using greedy decoding. This issue persists even with state-of-the-art LLMs containing billions of parameters, trained via next-token prediction on large datasets. We find that by further fine-tuning these models to achieve a near-zero training loss on a small set of samples – a process we refer to as hyperfitting – the long-sequence generative capabilities are greatly enhanced.



How to Parametrize P ?

- ▶ Any parametrization for multiclass classification can be considered (e.g., logistic regression, kernel classification, RNN).
- ▶ However, recall that the natural empirical distribution pools information in the original space of data.
- ▶ This does not account for semantics directly.
- ▶ Instead, aim to pool in a **semantic representation/ embedding space**.

Embeddings in Semantic Space

- ▶ Map tokens x_1, x_2, \dots, x_T into **embeddings/representations** $e_1, e_2, \dots, e_T \in \mathbb{R}^d$, which represent the "meaning" of tokens in a continuous vector space.

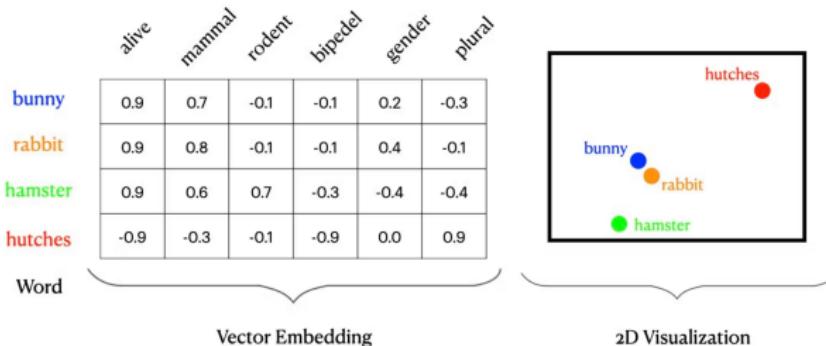


Figure: Source

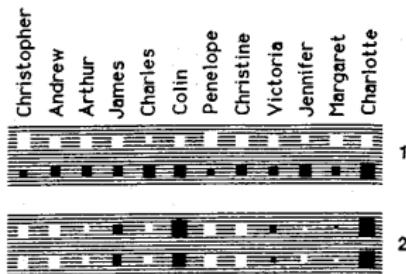
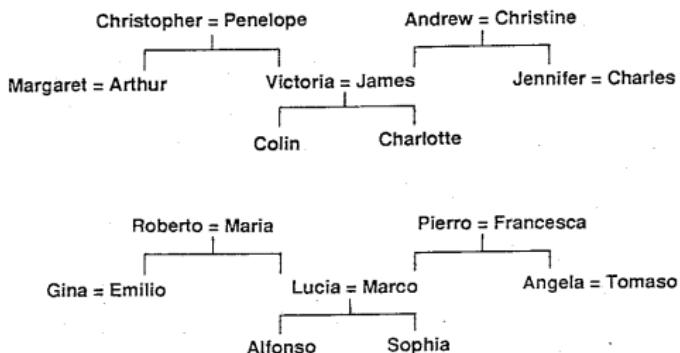
- ▶ Perform classification on the embeddings.
- ▶ Embeddings are learned during training.
- ▶ One of the most crucial ideas in AI!

The Origins of Distributed Representations

In: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*. Amherst, Mass. 1986, pages 1-12.

LEARNING DISTRIBUTED REPRESENTATIONS OF CONCEPTS

Geoffrey E. Hinton
Computer Science Department
Carnegie-Mellon University



Distributed Representations for Language Modeling

A Neural Probabilistic Language Model

Yoshua Bengio

BENGOY@IRO.UMONTREAL.CA

Réjean Ducharme

DUCHARME@IRO.UMONTREAL.CA

Pascal Vincent

VINCENTP@IRO.UMONTREAL.CA

Christian Jauvin

JAUVINC@IRO.UMONTREAL.CA

A goal of statistical language modeling is to learn the joint probability function of sequences of words in a language. This is intrinsically difficult because of the **curse of dimensionality**: a word sequence on which the model will be tested is likely to be different from all the word sequences seen during training. Traditional but very successful approaches based on n-grams obtain generalization by concatenating very short overlapping sequences seen in the training set. We propose to fight the curse of dimensionality by **learning a distributed representation for words** which allows each training sentence to inform the model about an exponential number of semantically neighboring sentences. The model learns simultaneously (1) a distributed representation for each word along with (2) the probability function for word sequences, expressed in terms of these representations. Generalization is obtained because a sequence of words that has never been seen before gets high probability if it is made of words that are similar (in the sense of having a nearby representation) to words forming an already seen sentence.

Figure: Bengio et al. (2000) uses distributed representations to learn a language model.

LLM Representations Encode Space

From Gurnee and Tegmark (2024).

The Attention Mechanism: Intuition

- ▶ Given embeddings e_1, e_2, \dots, e_T , learn which embeddings to focus on for next-token prediction.
- ▶ Example: "Anna was born on Tuesday. Bikram was born on Wednesday. What day followed Anna's birthday?"
 - ▶ How to solve this task?
 - ▶ Need day following Anna's birthday
 - ▶ Search input and memory for query "Anna's birthday"
 - ▶ From individual reps "Anna" + "'s" + "birth" + day construct contextualized representation for "Anna's birthday":
 - ▶ From "Anna" and "'s", construct rep for "Anna's"
 - ▶ From "Anna's" + "birthday", construct "Anna's birthday"
 - ▶ Find rep for related phrase "Anna was born" (key)
 - ▶ Look for information near that position: "on Tuesday" (value)
 - ▶ Propagate information "on Tuesday" to memory
 - ▶ Search for query "day following" + "on Tuesday" (key)
 - ▶ Retrieve rep of information "Wednesday" from internal memory

Attention Mechanism: Formulation

- ▶ Assign **attention weights** based on relevance (Bahdanau et al., 2015):

$$a(e_j, e_k) = \text{how much attention } e_j \text{ pays to } e_k$$

E.g., "Anna's birthday" should pay attention to "Anna was born"

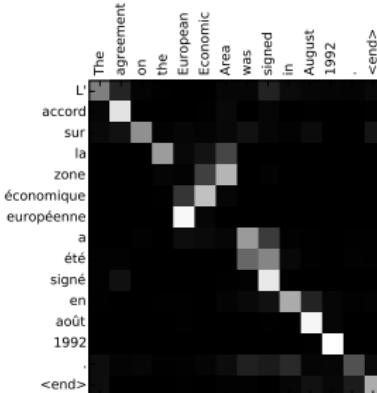


Figure: Bahdanau et al. (2015)

Attention Mechanism: Weighted Values

- ▶ Given the attention scores, compute a weighted sum of **values** V_i :

$$\hat{V}_j = \sum_{i \leq j} a(e_j, e_i) V_i$$

- ▶ E.g., Value of "Anna was born" could be "Tuesday", copied to location of "Anna's birthday"
- ▶ Values V_i are linearly transformed embeddings:

$$V_i = W_v e_i$$

- ▶ Could imagine directly using embeddings. Linear transforms
 1. allow us to focus on only the part of the embeddings that are directly relevant for next-token prediction (embeddings can carry other meaning as well; e.g., about the context, not directly relevant)
 2. reduce dimension

Inner-product Attention

- ▶ The attention weight $a(\cdot)$ is a measure of how similar certain aspects of embeddings e_j and e_i are. These aspects are captured in two quantities:
 1. What I'm looking for: a **query** q_j ; e.g., "Anna's birthday"
 2. What I have: a **key** k_i ; e.g., "Anna was born"
- ▶ Both are usually computed as linear transformations: $q_j = W_q e_j$, $k_i = W_k e_i$
- ▶ **Inner-product attention** (or, dot-product attn) ([Vaswani et al., 2017](#)):

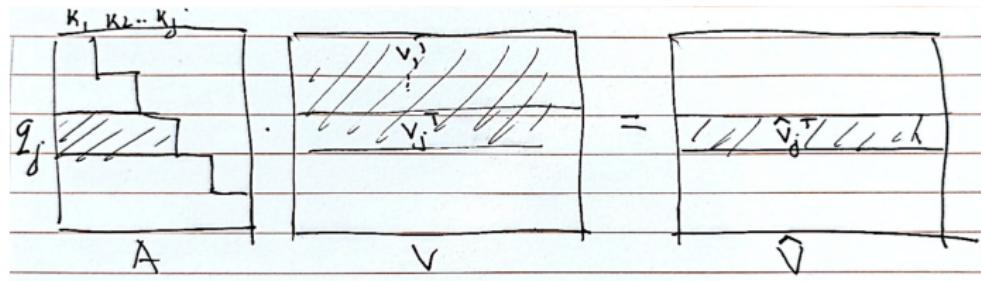
$$a(e_j, e_i) \propto \exp(q_j^\top k_i / \sqrt{d})$$

- ▶ Normalize using softmax. For $i \leq j$

$$a(e_j, e_i) = \frac{\exp(q_j^\top k_i / \sqrt{d})}{\sum_{i' \leq j} \exp(q_j^\top k_{i'} / \sqrt{d})}$$

The basis of nearly all modern LLMs.

Attention Operating on Values: Matrix Form



$$\hat{V}_j = \sum_{i \leq j} a(e_j, e_i) V_i = \sum_{i \leq j} a(q_j, k_i) V_i$$

Comments on Attention Weights

- ▶ Inner-product attention

$$a(e_j, e_i) = \frac{\exp(q_j^\top k_i / \sqrt{d})}{\sum_{i' \leq j} \exp(q_j^\top k_{i'} / \sqrt{d})}$$

- ▶ Where does \sqrt{d} come from? The typical size of an inner product between two random vectors with i.i.d. standardized entries.
- ▶ Numerical stability: first calculate the max of the terms $q_j^\top k_i$, and subtract it from all terms before the exponential.

Causal and Bidirectional Attention

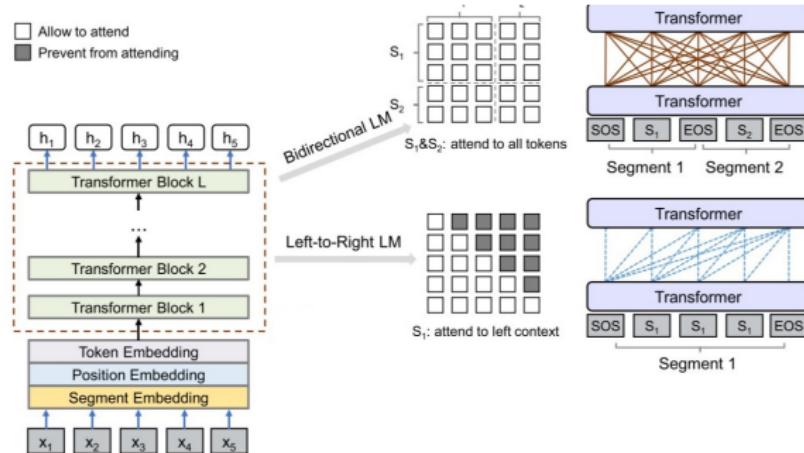


Figure: Source

- ▶ Causal/left-to-right attention:
 - ▶ Summation only over observed data (up to current index).
 - ▶ Usable for text generation via autoregression.
- ▶ Bidirectional attention (e.g., in BERT):
 - ▶ Summation over all keys (used, e.g., in classification tasks).

Value → Embedding & Multi-Head

- ▶ Maps values back into embedding space via W_o : $e'_i = W_o \hat{V}_i$.
- ▶ Use multiple **heads**, i.e., repeat the above operations multiple times to obtain $\hat{V}_i^{(j)}$, for several j s. Then concatenate them to obtain a feature of the same dim d .
- ▶ Thought to compute multiple solutions in parallel ([Xiong et al., 2024](#)).

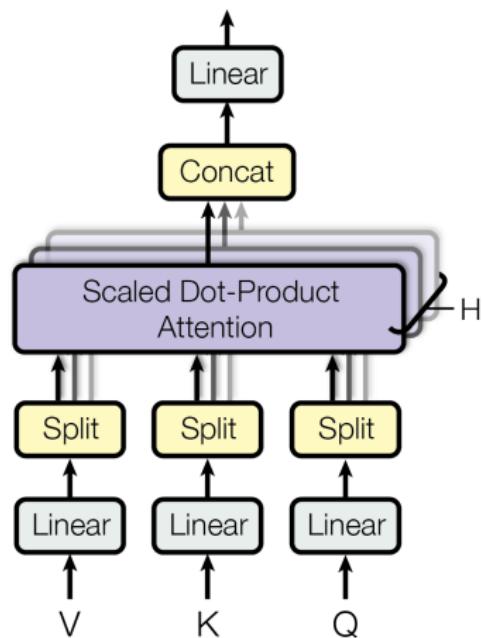


Figure: [Vaswani et al. \(2017\)](#)

Transformer Architecture Continued: Residual Layers

- ▶ Apply residual layers/skip connections ([He et al., 2016](#)) so that we make only a small update:

$$e_i \leftarrow e_i + e'_i$$

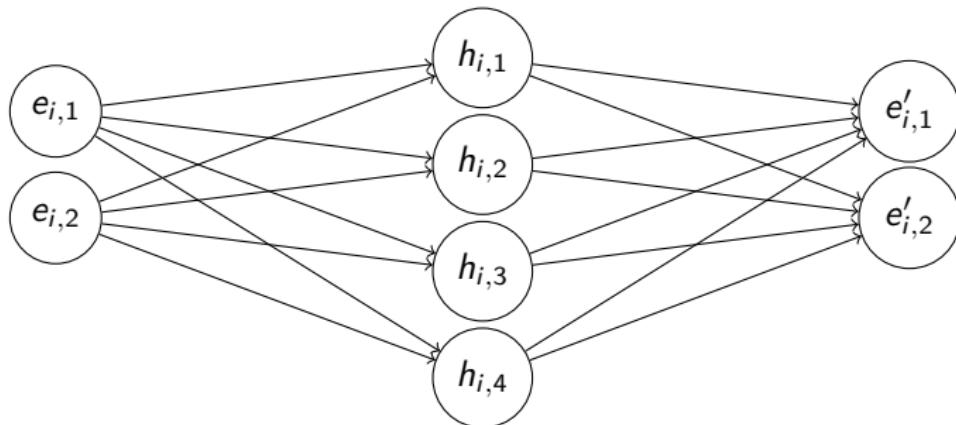
- ▶ Why? Makes it easy to copy data and propagate already computed information. [principle: make small changes at a time, iterate]

Feedforward/Multi-layer perceptron (MLP) layers

- ▶ Further process embeddings using feedforward networks. A small neural net applied to each embedding, e.g.,

$$e'_i = W_{\text{proj}} \sigma(W_1 e_i), \quad e_i \leftarrow e_i + e'_i,$$

where the dimension of the inner layer $W_1 e_i$ is larger by some factor (say four) than the dimension of the outer layer.



Interpreting MLP Layers

- ▶ These layers can be viewed as a form of un-normalized key-value memory (Geva et al., 2020).
 - ▶ Input: $e \in \mathbb{R}^d$, e.g., "day following Tuesday"
 - ▶ $\tilde{K}, \tilde{V} \in \mathbb{R}^{d \times d_m}$ form a neural memory (Sukhbaatar et al., 2015): d_m key-value pairs/memories.
 - ▶ Keys $\tilde{k}_i \in \mathbb{R}^d$, e.g., "day following Tuesday"
 - ▶ Values $\tilde{v}_i \in \mathbb{R}^d$, $i = 1, \dots, d_m$ e.g., "Wednesday"
 - ▶ Probability $P(\tilde{k}_i | e) \propto \exp(\tilde{k}_i^\top e)$. Expected value:

$$NM(e) = \sum_i P(\tilde{k}_i | e) \tilde{v}_i = \tilde{V} \cdot \text{softmax}(\tilde{K}^\top e)$$

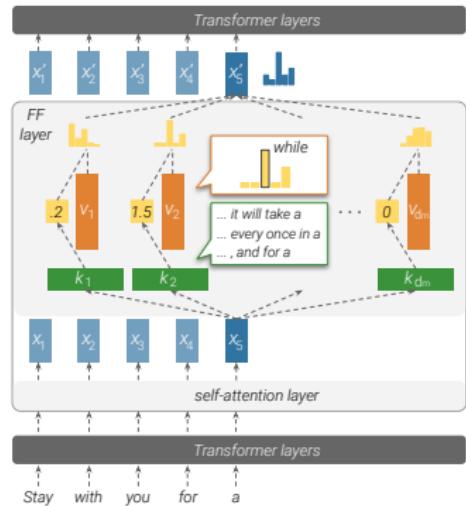


Figure: Geva et al. (2020)

MLP layers ctd.

- ▶ Instead, use unnormalized $\text{FF}(e) = \tilde{V}\sigma(\tilde{K}^\top e)$.
- ▶ Thought to store most of the knowledge: e.g., to answer "Michael Jordan plays the sport of ...", keys represent names, values represent professions; see also 3Blue1Brown [video](#).
- ▶ Difference from standard attention: Keys and values cannot depend on the input e . [principle: measure similarity via inner product].

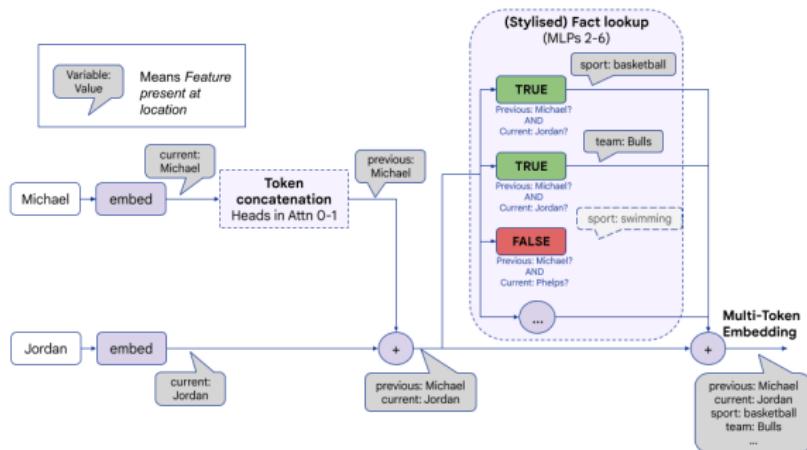


Figure: Source

Deep Architecture

- ▶ Use a deep network by repeating the process across multiple blocks.
- ▶ Attention updates and "contextualizes" representation by copying values from previous tokens
- ▶ MLP extracts knowledge associated with token embeddings
- ▶ Repeat to extract hierarchical concepts from the data.

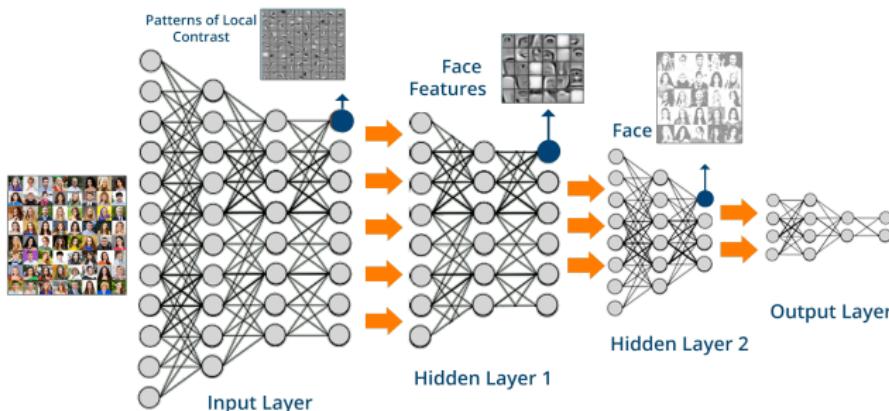


Figure: Source

Positional encoding

- ▶ How to take position of the input tokens into account? (Transformer does not do so by default),
- ▶ To address this, add a matrix Γ encoding location information to the embedding matrix E at the first layer. Has rows $\Gamma_1, \dots, \Gamma_T$ that represent the effect of token positions $1, 2, \dots, T$. Propagated through the layers.
- ▶ Can be fixed ([Vaswani et al. \(2017\)](#)) or learned (as in the GPT series in [Radford et al. \(2018, 2019\)](#); [Brown et al. \(2020\)](#)).

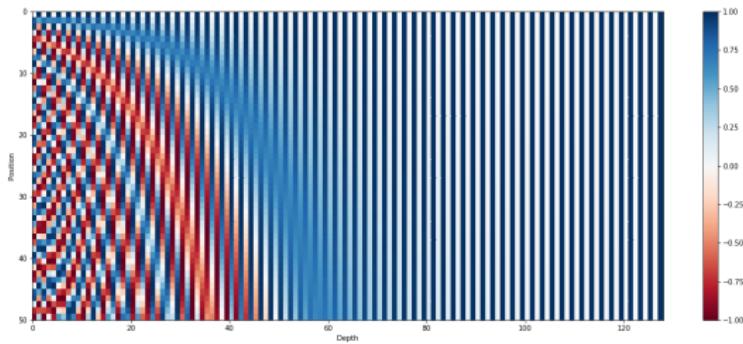


Figure: [Vaswani et al. \(2017\)](#)-style position embeddings ([Source](#))

Normalization and Dropout

- ▶ Layer normalization (Ba et al., 2016) is commonly used twice: before computing attention, and before the MLP.
- ▶ Embeddings: $E = [e_1, \dots, e_j]^\top$, a $j \times d$ matrix.
- ▶ LayerNorm $\text{LN}(E)$:

$$\text{LN}(e_i) = \frac{e_i - \mu_i}{\sigma_i} \cdot \gamma + \beta$$

where:

$$\mu_i = \frac{1}{d} \sum_{k=1}^d e_{ik}, \quad \sigma_i = \sqrt{\frac{1}{d} \sum_{k=1}^d (e_{ik} - \mu_i)^2 + c_0},$$

and $\gamma, \beta \in \mathbb{R}^d$ are learnable parameters, $c_0 > 0$ is a small constant for numerical stability.

- ▶ Dropout (Srivastava et al., 2014) is also used several times.

Illustration from Vaswani et al. (2017)

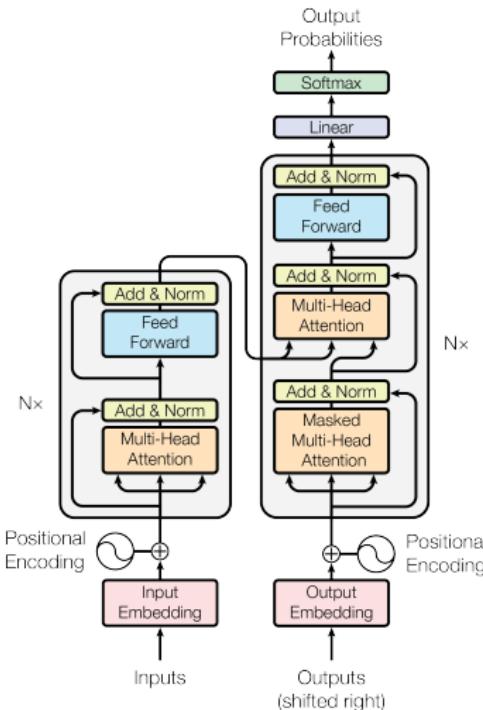


Figure: A more complicated encoder-decoder Transformer LM (Vaswani et al., 2017), with an additional cross-attention layer

A More Recent Illustration

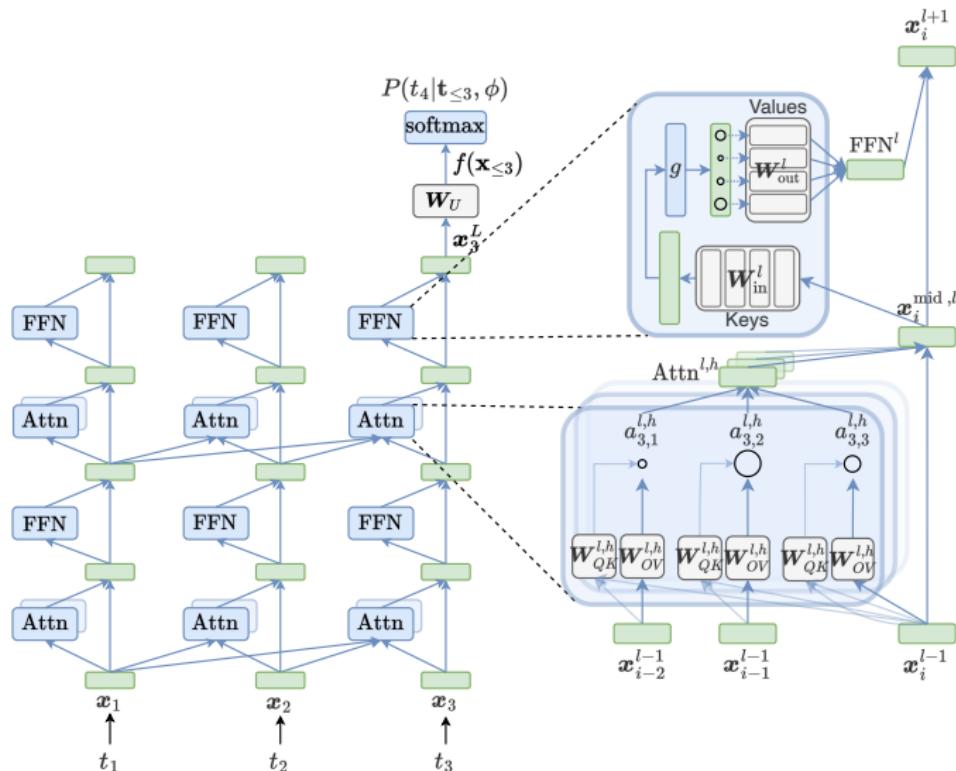


Figure: Transformer LM (Ferrando et al., 2024)

Next Token Prediction

- ▶ How to predict the next token given the embeddings e_1, e_2, \dots, e_j at the penultimate layer?
- ▶ Calculate pre-activations:

$$S = W_{\text{out}}[e_1, \dots, e_j],$$

where W_{out} is a $|V| \times d$ learned weight matrix, $[e_1, \dots, e_j]$ is $d \times j$, and obtain a matrix of size $|V| \times j$.

- ▶ Let $P_j = (P_{j,1}, \dots, P_{j,|V|})$ represent the predicted probability distribution over the possible tokens at position j .
- ▶ After calculating the pre-activations $S = (s_{k,i})_{k \leq |V|, i \leq j}$, apply a softmax function to convert them into probabilities. For all $k \leq |V|$

$$P_{j,k} = \frac{\exp(s_{k,j})}{\sum_{k'} \exp(s_{k',j})}$$

Next Token Prediction

- ▶ Equivalently, the probability of outputting token k at the $j + 1$ -st location is proportional to $\exp(W_{\text{out},k}^\top e_j)$.
- ▶ Also the probability of token k at the $l + 1$ -st location, for all $l \leq j$, is proportional to $\exp(W_{\text{out},k}^\top e_l)$. Self-consistent for varying j
- ▶ Leads to *KV caching*: for a speedup, store previous key-value pairs instead of recomputing them.
- ▶ Intuition: Given an input x , the probabilities over next tokens y are proportional to $p(y|x) \propto \exp(\phi(x)^\top \lambda(y))$, where $\phi(x)$ is the last-layer embedding of x , and $\lambda(y)$ is the readout representation of y

Matrix Representation of One-Head Attention

- ▶ Embeddings: $E = [e_1, \dots, e_T]^\top$, a $T \times d$ matrix. LayerNorm $E \leftarrow \text{LN}(E)$.
- ▶ Queries, Keys, Values:

$$Q = EW'_q, \quad K = EW'_k, \quad V = EW'_v$$

- ▶ Pre-attention: $Z = QK^\top / \sqrt{d}$.
- ▶ Causal attention: $A = \text{row-softmax}(Z \odot M)$ is a $T \times T$ matrix, where M is a lower-triangular $T \times T$ mask matrix with $M_{ij} = 1$ if $i \geq j$ and $-\infty$ otherwise.
- ▶ Intermediate embeddings: $\hat{E} = E + AVW'_o$. LayerNorm $\hat{E} \leftarrow \text{LN}(\hat{E})$.
- ▶ FFN: $\tilde{E} = \sigma(\hat{E}W'_1)W'_{\text{proj}}$. Residual update: $E \leftarrow \hat{E} + \tilde{E}$.
- ▶ Readout: calculate next-token probabilities $[\hat{p}_1, \dots, \hat{p}_T]^\top = \text{softmax}(EW'_{\text{out}})$, where W_{out} is $n_{\text{tokens}} \times d$.

Expressivity of Multi-Head Attention

- ▶ For one-head attention, $AVW'_o = AEW'_v W'_o$, so RHS transform is redundant: W_v or W_o can be removed.
- ▶ Two-head attention: $V = XW'_v$, $V = [V_1, V_2] = [EW'_{v,1}, EW'_{v,2}]$, and

$$\begin{aligned} O &= [A_1 V_1, A_2 V_2] W'_o = [A_1 E W'_{v,1}, A_2 E W'_{v,2}] \begin{bmatrix} W'_{o,1} \\ W'_{o,2} \end{bmatrix} \\ &= A_1 E W'_{v,1} W'_{o,1} + A_2 E W'_{v,2} W'_{o,2}. \end{aligned}$$

- ▶ W_o mixes together the results of $A_1 V_1$ and $A_2 V_1$, allowing the two heads to communicate.
- ▶ MHA has the same expressivity as $A_1 E C_1 + A_2 E C_2$, where C_1 and C_2 are $d \times d$ matrices of rank $d/2$.

In Code

- ▶ Need to process batches of datapoints, so have an extra dimension.
Work with tensors.
- ▶ See Andrej Karpathy's [video](#) "Let's build GPT"
- ▶ For readable code, see Andrej's [NanoGPT repo](#)

References

- J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016. URL <https://arxiv.org/abs/1607.06450>.
- D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.
- Y. Bengio, R. Ducharme, and P. Vincent. A neural probabilistic language model. *Advances in neural information processing systems*, 13, 2000.
- T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.
- A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=YicbFdNTTy>.
- J. Ferrando, G. Sarti, A. Bisazza, and M. R. Costa-jussà. A primer on the inner workings of transformer-based language models. *arXiv preprint arXiv:2405.00208*, 2024.
- L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

References

- M. Geva, R. Schuster, J. Berant, and O. Levy. Transformer feed-forward layers are key-value memories. *arXiv preprint arXiv:2012.14913*, 2020.
- W. Gurnee and M. Tegmark. Language models represent space and time. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=jE8xbmvFin>.
- K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- G. E. Hinton. Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, volume 1, page 12. Amherst, MA, 1986.
- A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training, 2018.
- A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners, 2019.
- R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1715–1725. Association for Computational Linguistics (ACL), 2016.
- C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

References

- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL <http://jmlr.org/papers/v15/srivastava14a.html>.
- S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus. End-to-end memory networks. *Advances in neural information processing systems*, 28, 2015.
- I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3104–3112, Cambridge, MA, USA, 2014. MIT Press.
- A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017. URL <https://arxiv.org/pdf/1706.03762.pdf>.
- Z. Xiong, Z. Cai, J. Cooper, A. Ge, V. Papageorgiou, Z. Sifakis, A. Giannou, Z. Lin, L. Yang, S. Agarwal, et al. Everything everywhere all at once: Llms can in-context learn multiple tasks in superposition. *arXiv preprint arXiv:2410.05603*, 2024.