Chatting about Transformers: an Introduction to Large Language Models

Gianfranco Bilardi

BOOST 2024
Bologna 19-30/08/24

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Two Major Discoveries

1. An efficiently computable stochastic model for natural languages.
2. Probabilistic generation produces meaningful text.

Machine Learning (a succint introduction)

Machine Learning (ML) is the discipline concerned with the automation of the development of scientific models.

1. Model = architecture *i.e.,* structure + parameters (human creativity)
2. Collection of data (human & machine)
3. Training, that is, the determination of the values of the parameters (machine).
4. Testing, that is, validation of the model w.r.t. the domain to be described (machine).
5. Inference, thas is, the utilization of the model (machine).

As of today, the algorithms deployed in the various phases are the result of human design, even if executed by machines. However, the behavior of the models (*e.g.,* during inference) is not always well understood.

Example: model of planet motion (Kepler, 1571-1630)

1. Architecture (Kepler's laws): a planet moves along an ellipse, with the sun occupying one of the focal points and the segment between the sun and the planet sweeping equal areas in equal times.
2. Parameters (7): direction perpendicular to the orbit (2); direction of the major axis (1); length of major and minor axes (2); position of the planet at some specified time (1); period of revolution (1).
3. Data: position of the planet at various times, measured by Ticho Brahe (1546-1601).
4. Training: determination of the laws of motion (ellipse + initial position + period) that best approximates data.
5. Testing: further measurements, over the centuries.
6. Inference: computation of the position of the planet in time, *e.g.*, to send a probe.

Statistical Models of Language

The statistical properties of language can be exploited for:

1. Criptography: reconstruct a secret code (from al-Kindi, c.801-873 A.C., and Caesar's (100-46) codes, to computational criptography (1970s-)).
2. Compression: encode a text so that it occupies less space , to save memory and bandwidth (information theory Shannon (1916-2001), souce coding).
3. Error correction: encode a text to facilitate detection and elimination of alterations (Shannon, channel coding).
4. Autocomplete: suggest possible continuations in the composition of a text (applications in word processors, mobile phones, ...), in particuar

   computing, for each word, the probability that it be the next one
5. Generation: generate a text, *e.g.*, in response to a "prompt".

Language Models

- Token dictionary: $\mathcal{T} = \{\tau_1, \dots, \tau_{|\mathcal{T}|}\}$.
- Text modeled as a stochastic sequence of tokens:

$$x_t \in \mathcal{T}, \qquad t = 0, 1, \dots$$

- Language model: The conditional probabilities of the next token given a context

$$Pr\left[x_t | x_0 x_1 \dots x_{t-1}\right] = Pr\left[x_t | \mathbf{c}_t\right], \qquad t = 0, 1, \dots$$

- Observation: For a given t, there are $|\mathcal{T}|^{t+1}$ probabilities. Typical values $\mathcal{T} = 2^{16}$; $t = 2^{11}$, implying $2^{2^{15}}$ probabilities.

Generating Text from the Model

1. Given an intial context, often referred to as the prompt, compute the probability distribution $\left[p_1, p_2, \ldots, p_{|\mathcal{T}|}\right]$ for the next token.
2. Generate the next token from the computed distribution:
    - Partition the real interval $[0, 1]$ into $|\mathcal{T}|$ subintervals, of respective sizes $p_1, p_2, \ldots, p_{|\mathcal{T}|}$.
    - Generate a random number ξ uniformly distributed in $[0, 1]$.
    - If ξ falls in the k-th interval, then output token τ_k.
3. Append output token to the input and iterate, until the "end-of-text" token is generated.

Observation. Starting twice with the same prompt will typically generate two different texts.

Embedding Models

An d- dimensional embedding is a function e that maps every sequence of tokens to a d-dimensional real vector:

$$e: \mathcal{T}^* \to \mathbb{R}^d.$$

For sequences of length 1 (one token), e is referred to as a word embedding.

Central Assumption

There are embeddings such that:

- The conditional probabilities are determined by the embedding as

$$Pr\left[x_t|\mathbf{c}_t\right] \propto \exp(e(\mathbf{c}_t) \cdot e(x_t));$$

- The embedding of a sequence is a function of the embeddings of its elements:

$$e(\mathbf{c}_t) = \gamma(e(x_0), \dots, e(x_{t-1})).$$

Matrix-Softmax Formulation

- Notation: Embedding vectors are represented as row vectors.
- Word-Embedding Matrix: $\mathbf{W}_{\mathcal{T}} \in \mathbb{R}^{d \times |\mathcal{T}|}$.

$$\mathbf{W}_{\mathcal{T}} = \left[e(\tau_1)^T, \ldots, e(\tau_{|\mathcal{T}|})^T \right],$$

  that is, the j-th column is the word embedding of τ_j.
- From token to embedding:

$$e(\tau_j) = (1 - \mathrm{hot} - \mathrm{encoding}(\tau_j))\mathbf{W}_{\tau}^T.$$

- From context embedding to next-token distribution:

$$\begin{aligned} \pi_t &= \left[Pr(\tau_1 | \mathbf{c}_t), \ldots, Pr(\tau_{|\mathcal{T}|} | \mathbf{c}_t) \right] \\ &= softmax(e(\mathbf{c}_t)\mathbf{W}_{\mathcal{T}}). \end{aligned}$$

The Transformer Model [1]

The context-embedding function γ, such that

$$e(\mathbf{c}_t) = \gamma(e(x_0), \ldots, e(x_{t-1})),$$

can be modeled/computed by a $(d \times t)$-Transformer Core, with the following high-level structure:

- Positional encoder
- A cascade of N Transformer blocks, all with the same structure, but with different coefficients, including:
  1. An Attention stage
  2. A Feed-forward stage
  3. Some Normalizations and Residual connections

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin: Attention is all you need. CoRR abs/1706.03762 (2017). Over 135,000 citations.

Transformer: Block Diagram (from Attention is all ...)

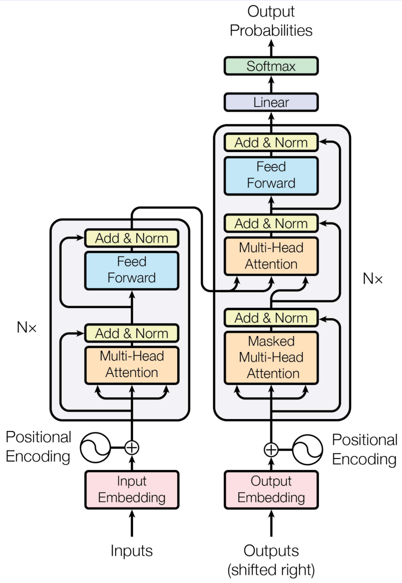

Figure 1: The Transformer - model architecture.

$d \times n$-(Self) Attention

1. Input:

$$\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_{n-1} \in \mathbb{R}^{1\times d}$$

2. Parameters: (query, key, and value weights)

$$\mathbf{W}_Q \in \mathbb{R}^{d\times d_k}, \quad \mathbf{W}_K \in \mathbb{R}^{d\times d_k}, \quad \mathbf{W}_V \in \mathbb{R}^{d\times d}$$

3. Output: For $i = 0, 1, \ldots, n-1$,

$$\mathbf{z}_i = \sum_{j=0}^{n-1} \alpha_{ij} \mathbf{y}_j \mathbf{W}_V,$$

$$\begin{aligned}
[\alpha_{i0}, \ldots, \alpha_{i,n-1}] &= softmax((\mathbf{y}_i \mathbf{W}_Q)(\mathbf{y_0} \mathbf{W}_K)^T, \ldots, (\mathbf{y}_i \mathbf{W}_Q)(\mathbf{y_{n-1}} \mathbf{W}_K)^T) \\
&= softmax(\mathbf{y}_i(\mathbf{W}_Q \mathbf{W}_K^T)\mathbf{y_0}^T, \ldots, \mathbf{y}_i(\mathbf{W}_Q \mathbf{W}_K^T)\mathbf{y}_{n-1}^T).
\end{aligned}$$

Multi-Head Attention

Let H, a divisor of d, be the number of heads, and $d_k = \frac{d}{H}$.

1. Input: $\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_{n-1} \in \mathbb{R}^{1\times d}$
2. Parameters: (query, key, and value weights). For $h = 0, \ldots H-1$

$$\mathbf{W}_Q^h \in \mathbb{R}^{d\times d_k}, \quad \mathbf{W}_K^h \in \mathbb{R}^{d\times d_k}, \quad \mathbf{W}_V^h \in \mathbb{R}^{d\times d_k}$$

3. Output: For $i = 0, 1, \ldots, n-1$ and $h = 0, 1, \ldots, H-1$

$$\begin{aligned} \mathbf{z}_i^h &= \sum_{j=0}^{n-1} \alpha_{ij}^h \mathbf{y}_j \mathbf{W}_V^h, \\ \mathbf{z}_i &= concatenate(z_i^0, z_i^1, \ldots, z_i^{H-1}), \end{aligned}$$

where

$$\left[\alpha_{i0}^h, \ldots, \alpha_{i,n-1}^h\right] = softmax((\mathbf{y}_i \mathbf{W}_Q^h)(\mathbf{y_0} \mathbf{W}_K^h)^T, \ldots, (\mathbf{y}_i \mathbf{W}_Q^h)(\mathbf{y_{n-1}} \mathbf{W}_K^h)^T)$$

$d \times n$-Feed Forward Stage

A set of n independent copies of a feed-forward network $FFN : \mathbb{R}^d \to \mathbb{R}^d$, each acting (independently and “in parallel”) on one of n embedding vectors:

$$FFN(\mathbf{z}) = \max(0, \mathbf{z}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2,$$

where

- $\mathbf{W}_1 \in \mathbb{R}^{d \times d_{ff}}$ is the weight matrix of a 1-stage neural network, with ReLU activation functions (specifically, d_{ff} neurons, each with d inputs);
- $\mathbf{b}_1 \in \mathbb{R}^{1 \times d_{ff}}$ is a bias vector;
- $\mathbf{W}_2 \in \mathbb{R}^{d_{ff} \times d}$ is the weight matrix of an affine transformation with $\mathbf{b}_2 \in \mathbb{R}^{1 \times d}$ as a bias vector.

$FFN(\mathbf{z})$ is typically normalized to a given **mean** and to a given **standard deviation**, themselves (learnable) parameters.

Positional Encoding

Transformer Core Symmetry: Applying a given permutation to the sequence of inputs (embedding vectors) results in the same permutation to be applied to the sequence of outputs.

To make the context-embedding function sensitive to the order of the tokens in the input, a (different) positional embedding vector is added to each word-embedding vector, before feeding the (modified) input to the transformer core.

The positional vectors can be determined by a suitable formula, or they can be made learnable.

Transformer Summary

1. Text is tokenized (*e.g.*, by the Byte Pair Encoding algorithm).
2. Each token is replaced with its d-dimensional word embedding vector.
3. A d-dimensional positional embedding vector is added to each word embedding vector.
4. A window of n (consecutive) vectors is processed by a $(d \times n)$-transformer core, that is, a sequence of N stages, each with
   - a multi-head attention substage;
   - a residual connection adding the input of the attention substage to its output;
   - normalization;
   - a feed-forward substage, followed by normalization;
   - a residual connection adding the input of the attention substage to its output;
   - normalization.
5. context-to-logit linear transform;
6. logit-to-probabilities softmax.

Training

- The training set is a set of context/next-token pairs $(\mathbf{c}, \tau)$.
- The contribution of a pair $(\mathbf{c}, \tau)$ to the loss function is the cross-entropy between:
  - the distribution produced by the transformer on input $\mathbf{c}$;
  - the (delta-type) distribution where token τ has probability 1 and all other tokens have probability 0;
  - minimizing the cross-entropy is equivalent to maximizing (w.r.t. the parameters θ) the function

$$\prod_{(\mathbf{c},\tau)} Pr_\theta\left(\tau|\mathbf{c}\right).$$

- Gradient descent (Cauchy, 1847) is applied, typically in the version specified by the ADAM algorithm[2].
- Back propagation is deployed, to efficiently compute the components of the gradient (*i.e.*, the partial derivatives of the loss function, w.r.t. the transformer parameters).

[2]Diederik P. Kingma, Jimmy Ba, ADAM: A Method for Stochastic Optimization, arXiv:1412.6980, (2014). Over 192,000 citations.

Computational Issues: Problem Size and Complexity

Hyper Parameters of (V)LLMs: representative values

- size of token dicionary $|\mathcal{T}|$: $O(2^{17}) = O(100,000)$.
- embedding dimension d: $2^{13} + 2^{12} = 12,288$.
- size of input window n: $2^{15} = 32,768$.
- number of blocks N: $\geq 96 = 2^6 + 2^5$.
- number of attention heads H: $12,288 : 128 = 96$.

Learnable Parameters: $\mathbf{W}_{|\mathcal{T}|}$ and, for each of the N levels, $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V, \mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b_2}$.

- number of learnable parameters: $O(|\mathcal{T}|d + 5Nd^2 + l.o.t.)$ $\approx O(10^{11}–10^{12})$ *floating point* numbers (typically, 1FP = 2 bytes).
- size of training set: $O(10^{11}–10^{12})$ tokens (1 token $\approx$ 2 bytes).
- computational effort for training:
- computational effort for inference:
- training cost: $> 10^8$ USD for ChatGPT-4, according to Sam Altman (CEO of OpenAI).

Computational Issues: Parallel Algorithms and Architectures

- The transformer is rich in matrix-matrix and matrix-vector multiplications.
- Fortunately, matrix-matrix multiplication exhibits
  - high degree of parallelism
  - (relatively) low communication requirements.
- Graphic Processing Units (GPUs) are well suited for matrix computations.
- Specialized accelerators for matrix computations are also being developed by various companies, often based on systolic algorithms and architectures (proposed in the early 1980s).

Computational Issues: Economic Impact

- The explosion in GPU utilization for AI is witnessed, *e.g.*, by NVIDIA growth in market capitalization. World ranking as of August 18:
  1. Apple $3.436T$
  2. Microsoft $3.110T$
  3. NVIDIA $3.064T$
  4. Alphabet $2.015T$
  5. Amazon $1.858T$
- Investments in new data centers are measured in tens G$/year.
- The energy requirements of AI computation is stressing the electrical grid, in some areas of the USA.
  As a reference, Microsoft has 5GW energy capacity.

The generative power of the Transformer[3]and its potential implications for semantics

1. Somewhat surpisingly the stochastic generation of text, driven by the transformer model produces output that is linguistically well formed and with plausibile content (although, non necessarily true).
2. Quite opposite interpretations of this discovery have been put forward:
    1. Stochastic parrot (E. Bender, 2022, N. Chomsky). In the inference phase, the model rehashes what was ingested in the training phase: no understanding whatsoever.
    2. Distributional semantics (L. Bloomfield 1933 e Z.S. Harris, 1951). Meaning emerges from stochastic correlations among words. "The meaning of a word lies in its usage" (L.Wittgenstein, 1953). Distribution of all continuations of a sentence (M. Trager e S. Soatto, 2024).
    3. Referential semantics. The meaning is in the world and words "point" to it (direct reference, J.S.Mill, 1843; S. Kripke, 1980) possibly through an intermediary (indirect reference, G. Frege, 1892).
3. Anyway, this discovery is revolutionizing artificial intelligence.

[3]GPT = Generative Pretrained Transformer

Some research issues

- Corpus encoding. Training can be viewed as an encoding of the training corpus into the parameters. Is this encoding lossy?
- Context encoding. Is it lossy?
- Impact of input variables. How much does each component of each word-embedding vector affect the output probabilities?
- Multiple realizations of the same I/O function.
  - $\forall A (W_Q W_K^T = (W_Q A^{-1})(W_K A^T)^T)$.
  - If positional embedding parameters are learnable, then the transformer core can realize each input-output function by $d!$ different parameter assignments.
  - Corollary: there are many equivalent minima.
- Breaking symmetry. Why positional embedding instead of asymmetric network architectures?
- Testing. How do we test probability estimates when it is extremely unlikely to observe any single event more than once?

Structure vs. Parameters

Question. Is there more information in the model structure or in (the values of) the model parameters?

Formulation. Assume we want to model a Boolean function $g : \{0,1\}^n \to \{0,1\}$ by means of a class of functions

$$f : \{0,1\}^n \times \{0,1\}^p \to \{0,1\},$$

by finding a parameter vector $\theta^* \in \{0,1\}^p$ such that

$$g(\mathbf{x}) = f(\mathbf{x}, \theta^*).$$

Suggestion. Find a function ζ such that

$$\begin{array}{llll} \textbf{if} & p < \zeta(n) & \textbf{then} & \text{there is more information in the structure,} \\ \textbf{if} & p > \zeta(n) & \textbf{then} & \text{there is more information in the parameters.} \end{array}$$