Large Language Models: from RNNs to Attention. Technical analysis and introduction to hallucinations.

Eduardo De Filippis
www.eduardodefilippis.com

The advent of large language models (LLMs) based on the Transformer architecture marked a point of discontinuity in the history of natural language processing (NLP). While previous recurrent architectures (e.g., RNNs and LSTMs) treated language as a strictly linear sequence, suffering from the "vanishing gradient" problem over long dependencies, modern LLMs parallelize positional processing by introducing the concept of self-attention. It is worth noting that this holds true during training (thanks to masking). During inference (generation), the process remains inherently sequential (autoregressive) due to the dependency on the previous token.

However, reducing an LLM to a simple "chatbot" would be a mistake. In fact, from a statistical perspective, these systems are modelers of conditional probability distributions over discrete sequences of tokens.

Below is presented part of the mathematics behind the attention mechanism, positional injection, and the stochastic implications of decoding strategies. Overly technical details and clarifications are omitted to avoid weighing down the reading.

1 Formal and probabilistic definition

A language model is, in its essence, a function that assigns a probability to a sequence of words (tokens) $w_1, w_2, ..., w_T$. The objective during the pre-training phase (often self-supervised learning) is to maximize the likelihood of the joint distribution $P(w_1, ..., w_T)$, factorizable according to the chain rule:

$$P(w_1, ..., w_T) = \prod_{t=1}^{T} P(w_t | w_1, ..., w_{t-1})$$

The model learns the parameters θ that minimize the cross-entropy loss on the training corpus:

$$\mathcal{L}(\theta) = -\frac{1}{T}\sum_{t=1}^{T} \log P(w_t | w_{<t}; \theta)$$

It is specified that, in the Loss summation, $t = 1$ to T is often used, but the first token w_1 is usually a special token, such as <sos> or [CLS], which has no preceding context. Therefore, the prediction technically starts from the second token based on the first.

2 The Transformer architecture

While Attention is the most famous component, the effectiveness of the Transformer lies in the synergy of three fundamental components that allow it to overcome the limits of RNNs.

2.1 Scaled Dot-Product Attention

The innovation of the paper "Attention is All You Need" (Vaswani et al., 2017) consists in projecting each token into three vectors via weight matrices W^Q, W^K, W^V: **Query** (what I look for), **Key** (what I offer), **Value** (the content). The canonical formula is:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

Where the scaling term $\frac{1}{\sqrt{d_k}}$ prevents the saturation of the Softmax.

Furthermore, the architecture utilizes Multi-Head Attention. Consequently, the calculation is parallelized over h different "heads", allowing the model to simultaneously capture relationships in different semantic subspaces (e.g., one head focuses on syntax and another on anaphora).

2.2 Positional encoding

A critical point often overlooked is that the attention equation is "permutation invariant". That is, without explicit information, for the model, the sentence "The dog bites the man" is mathematically identical to "The man bites the dog". To remedy this, a positional vector P_t is added (or injected) to the token embedding E_t.

In modern architectures (such as LLaMA), "rotary positional embeddings" (RoPE) are used, which rotate the Query and Key vectors in complex space by an angle proportional to their absolute position m:

$$f(x, m) = xe^{im\theta}$$

In formal terms, for a pair of vector components (x_i, x_{i+1}), the RoPE transformation acts as a rotation in the plane:

$$\begin{pmatrix} f_q(x,m)_i \\ f_q(x,m)_{i+1} \end{pmatrix} = \begin{pmatrix} \cos m\theta_i & -\sin m\theta_i \\ \sin m\theta_i & \cos m\theta_i \end{pmatrix} \begin{pmatrix} x_i \\ x_{i+1} \end{pmatrix}$$

This allows the dot product QK^T to depend only on the relative distance between tokens, improving generalization on long sequences.

It is highlighted that, in reality, $\theta_i = 10000^{-2i/d}$. Where d is the head dimension (d_k), not the entire model dimension (d_{model}).
To be extremely precise, in RoPE the index i of θ refers to the pair of dimensions. If d is the head dimension, i ranges from 0 to $d/2 - 1$.
Furthermore, it is worth remembering that the value 10000 is the originally introduced standard base. In more recent implementations to extend the context window (such as in Llama 3 or Long-Context models), this base is often modified (e.g., 500,000).

2.3 Feed-Forward Networks (FFN)

After the attention block, the information passes through a dense network (MLP) that works "position-wise" (independently on each token). Thus, Attention serves to "mix" information between different words, while the FFN serves to process and "reason" about the collected information.

3 Practical application: implementation of Attention

To understand the $O(N^2)$ quadratic computational complexity, a from-scratch implementation of the attention mechanism is proposed below.

```python
import numpy as np
from scipy.special import softmax

###################

def scaled_dot_product_attention(query, key, value, mask=None):
    """
    Args:
        query: (batch_size, num_heads, seq_len_q, depth)
        key:   (batch_size, num_heads, seq_len_k, depth)
        value: (batch_size, num_heads, seq_len_k, depth_v)
        mask:  optional mask (look-ahead)
    """
    d_k = query.shape[-1]

###################

    # Step 1 - Matmul: (Batch, Heads, Seq_Q, Seq_K)
    # transpose K to align dimensions for the dot product
    scores = np.matmul(query, key.transpose(0, 1, 3, 2))

    # Step 2 - Scaling
    scores = scores / np.sqrt(d_k)

    # Step 3 - Masking
    # for autoregressive decoders
    if mask is not None:
        scores += (mask * -1e9)
        # technical note: this score setting assumes the mask is 1 where we want to obscure and 0 where we want to keep. Usually libraries (like PyTorch) use inverted boolean masks.

    # Step 4 - Softmax
    attention_weights = softmax(scores, axis=-1)
    # important technical note: The scipy.special.softmax function is safe, but if we were to implement it manually, we should subtract the maximum value (scores - np.max(scores)) to avoid exponential overflow. Using SciPy, this problem does not exist.

    # Step 5 - Output
    # weighted sum of values
    output = np.matmul(attention_weights, value)

    return output, attention_weights

###################

# dummy example
np.random.seed(2025)
seq_len, d_model = 4, 8
Q = np.random.rand(1, 1, seq_len, d_model)
K = np.random.rand(1, 1, seq_len, d_model)
V = np.random.rand(1, 1, seq_len, d_model)

output, weights = scaled_dot_product_attention(Q, K, V)
print(f"Output Shape: {output.shape}")
```

Listing 1: Low-level implementation of scaled dot-product attention

It is emphasized that masking is fundamental to avoid "data leakage". Indeed, during training, the token at time t must not be able to "see" the tokens $t + 1$.

4 The decoding dilemma and "hallucinations"

Once the probabilities (logits) are calculated, the model must select the next token.

4.1 Temperature (τ) and Top-P

Temperature modifies the shape of the distribution before sampling:

$$P_i = \frac{\exp(z_i/\tau)}{\sum_j \exp(z_j/\tau)}$$

With $\tau > 1$, the distribution flattens ("creativity" and hallucination risk). With $\tau < 1$, the peaks are accentuated. Top-P (nucleus sampling) selects the smallest set of tokens whose cumulative probability exceeds a threshold p (e.g., 0.9), dynamically adapting the pool of choices to the model's "confidence" in the current context. This prevents the dispersion of probabilities when the model is uncertain.

Hallucination is often statistical: the model samples from the "long tail" of the distribution or relies on learned patterns that do not correspond to factual reality (erroneous parametric knowledge).

5 Conclusions

LLMs are not knowledge databases, but probabilistic reasoning engines. Understanding the distinction between model weights (parametric knowledge) and the provided context (in-context knowledge) is essential. Future challenges focus on efficiency (e.g., quantization, pruning) and the integration of factual verification mechanisms to mitigate the stochastic nature of generation.