The Speaking Machine

Understanding Large Language Models

Author: Idrissi Kandri Badreddine

July 2, 2025

Author's Note

This Technical Booklet was written and realized by **Idrissi Kandri Badreddine** during my first year of engineering studies in *Information Systems and Big Data*.

It represents my very first research project, focused on providing a clear and structured understanding of **Large Language Models (LLMs)**. Through this work, I aim to break down the complexity behind these models and make them accessible, especially to students and newcomers in the world of artificial intelligence.

What makes this Booklet special is the passion and dedication I put into every page. I truly hope you find it insightful, useful, and inspiring particularly if you're taking your first steps into the fascinating world of AI.

Redaction: LaTeX
Visuals and Schemas: Designed using `app.diagrams.net`

Idrissi Kandri Badreddine
Student Engineer in Big Data and Information Systems

Contents

1 Introduction

In recent years, Large Language Models (LLMs) have revolutionized the way humans interact with machines. From answering questions and writing essays to generating code and engaging in natural conversations, these models are increasingly becoming essential tools in our daily lives. But behind the simplicity of typing a question and getting an instant response lies a complex and carefully engineered process one that blends mathematics, linguistics, data, and deep learning.

This Booklet aims to demystify that process.

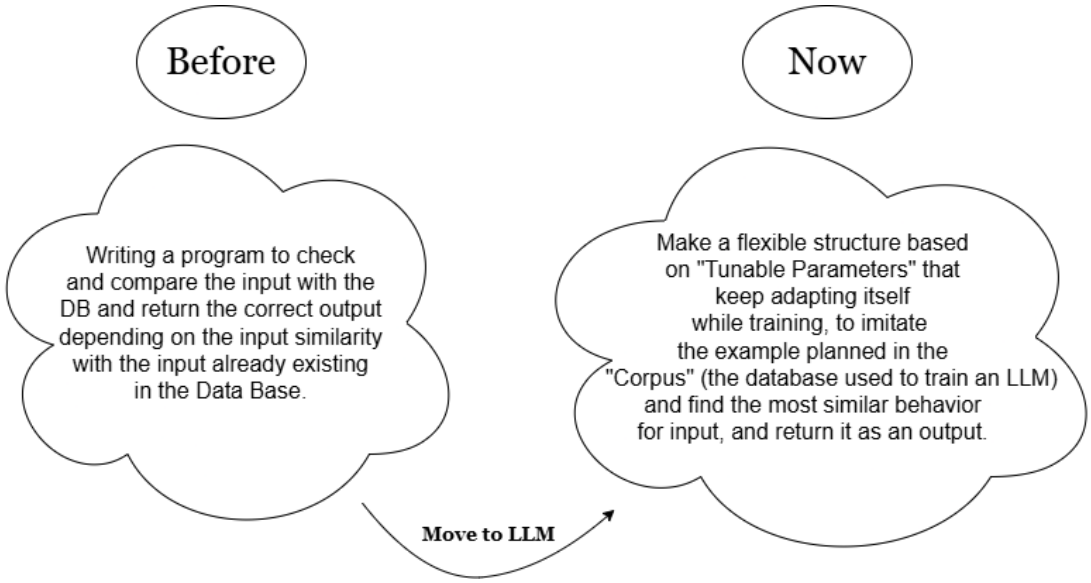
Rather than treating LLMs as black boxes, we will open them up and explore what truly happens inside. We'll walk step-by-step through the life of an LLM from how it is trained on massive text corpora to how it generates fluent, context-aware responses at inference time. Along the way, we'll break down core concepts such as tokenization, embeddings, attention mechanisms, loss functions, and gradient-based optimization, using clear explanations and intuitive examples.

Whether you are a curious learner, a student of AI, or an aspiring developer, this book is designed to give you both the theoretical understanding and the practical insight needed to appreciate the inner workings of modern language models.

Let us begin our journey into the fascinating world of LLMs where language meets intelligence, and where machines learn to speak.

2 Definition

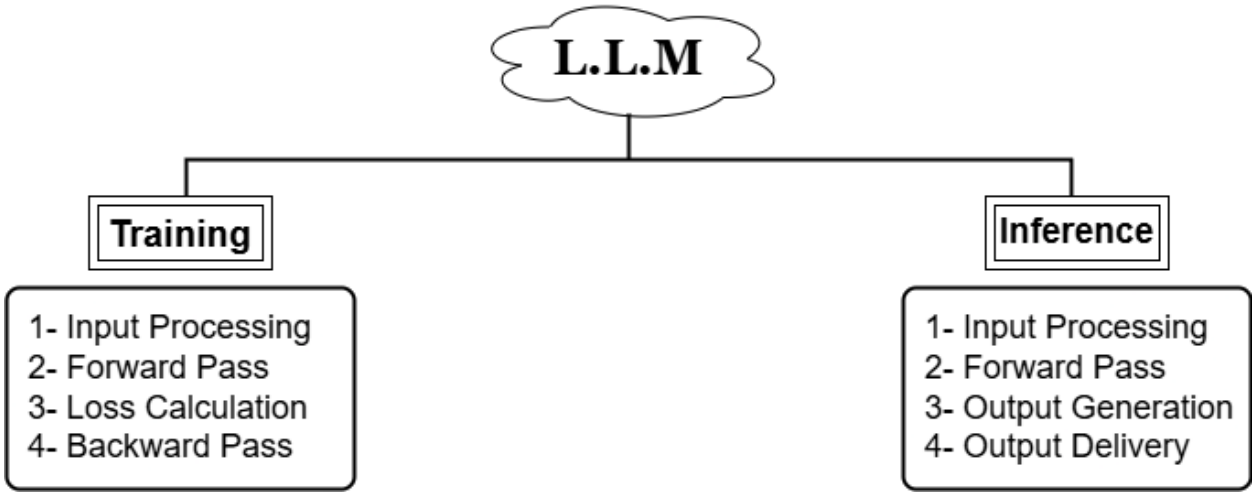
A Large Language Model (LLM) is a type of AI model trained on vast amounts of data using deep learning techniques, particularly Transformer architectures, with the goal of producing logical output from a given input



In Deep Learning: Tunable Parameters = **Weights**
=>Model Brain - Learn during training - Defines the model behaviors.

3 Process

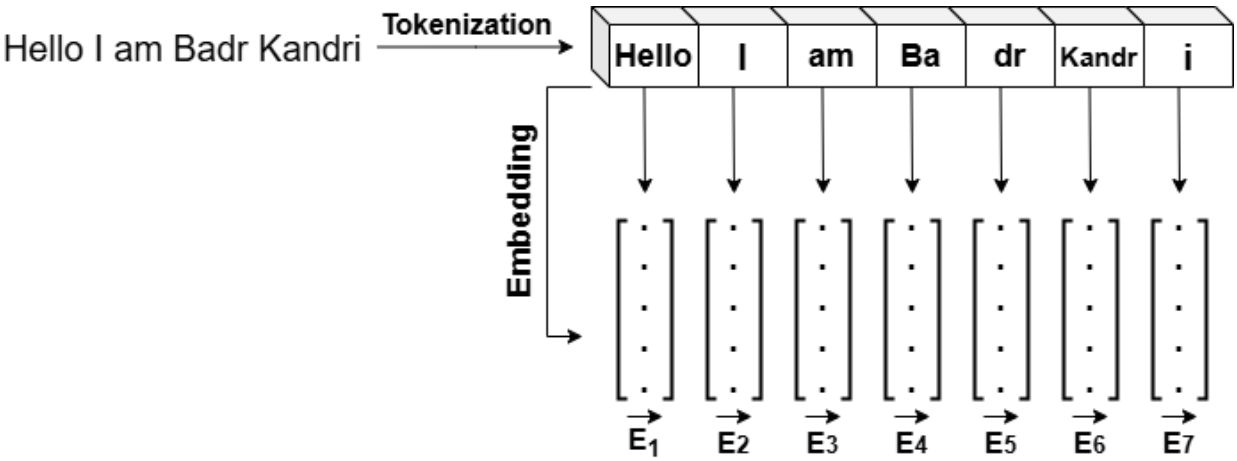
To fully understand all aspects of LLMs, we need to learn about its two main phases: training and inference.



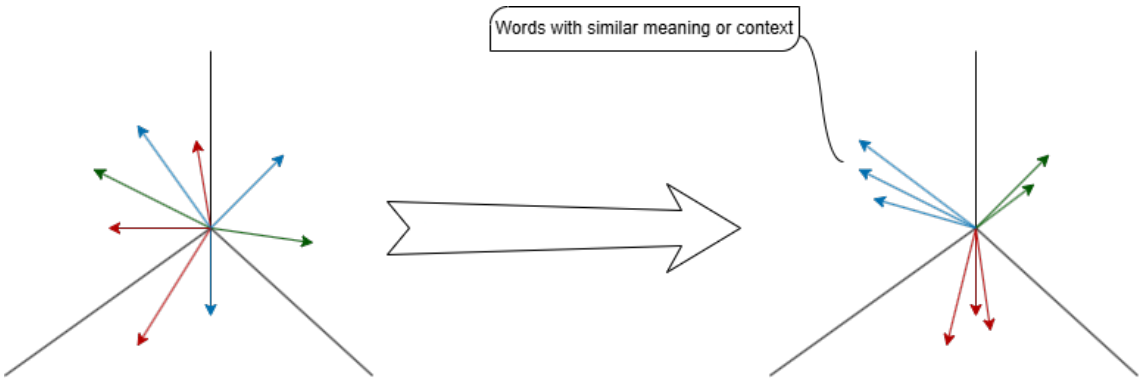
4 Inside the LLM

4.1 Input Processing

In the input processing phase, there are two steps. The first one is **Tokenization**, in this step, the model converts every slice of the input into tokens. These slices can be words, punctuation marks, or letters, it all depends on how the model was trained.
Then comes the **Embedding** step when the model converts each token into a vector with random numerical values, called an embedding. This vector contains two types of information: the slice's meaning (without considering the sentence context) and its position in the sentence.



OBJECTIF: Adjusting weights by updating the vectors to organize them based on their meaning.



4.2 Forward Pass

Forward Pass in an LLM (like ChatGPT) Is the phase where input tokens are passed through the Transformer architecture. After each token is converted into an embedding vector, through multiple layers of self-attention and MLP , the model captures the meaning and context of each token relative to others. The final output is a prediction (next word or full response), generated using the learned relationships all without changing the model's weights.

a Self-Attention

Self-attention is the step where each word looks at every other word in the sentence to understand their relationships and identify which ones influence its meaning.
This was a friendly definition of what really happens in this phase. Now, let's dive deeper into what actually happens mathematically and geometrically. And to do that, we're going to divide it into 8 steps.
Step 1: Each embedding generates a **Query vector** $\vec{Q}$, which allows it to interact with other embeddings in order to identify those that are most related to it.

$$W_Q \cdot \vec{E}_i = \vec{Q}_i$$

Step 2: Each embedding generates a **Key vector** $\vec{k}$ which contains the information needed to respond to queries from other embeddings.

$$W_K \cdot \vec{E}_i = \vec{K}_i$$

Step 3: Each $\vec{k_i}$ is multiplied by every $\vec{Q_i}$, producing dot product values. Large values imply a strong correlation between the embeddings.
so we say that **the key** $\vec{k_i}$ **attends to the query** $\vec{Q_i}$.
Geometrically, this corresponds to the two vectors being nearly aligned.

Step 4: (Masking) The dot product values are then multiplied by a masking matrix that sets certain values to zero. This prevents a token from attending to future positions and forces it to predict the next token based only on past tokens. This mechanism helps the model better understand the context of the input, leading to more accurate predictions.

Step 5: Each embedding generates a **Value vector** $\vec{V}$, which defines the information to be added to the original embedding $\vec{E}_i$ in order to obtain the updated embedding $\vec{E}'_i$.

$$W_V \cdot \vec{E}_i = \vec{V}_i$$

The W_V is the **Value Matrix**, and it is composed of an augmentation matrix and a reduction matrix, and the V_i is called the **Weight**.

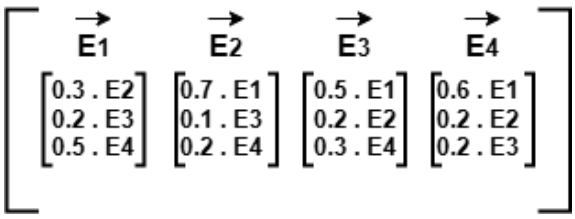
Step 6: The result from Step 4 (Masking) is then normalized into probabilities using the **Softmax** function, and subsequently multiplied by the value vectors.

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

Where:

- Q = Query matrix
- K = Key matrix
- V = Value matrix
- d_k = Dimension of the key vectors (used for scaling)

This produce an **Attention Pattern:**

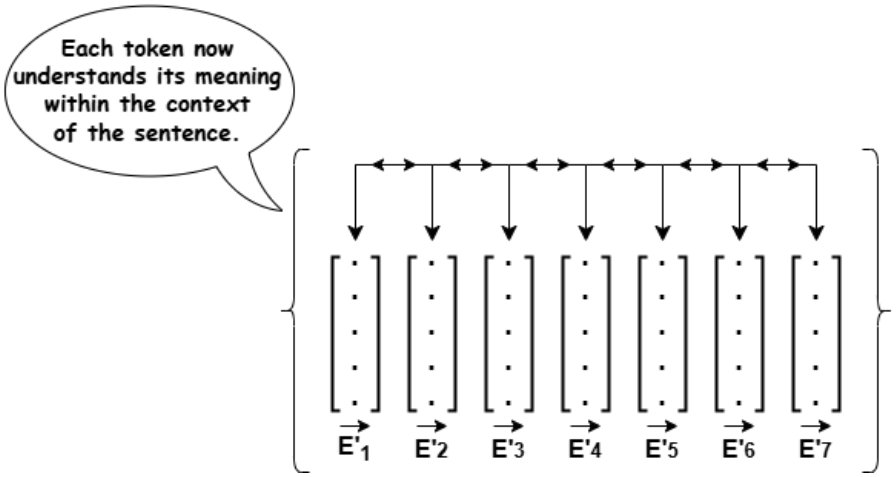


Step 7: Generate the contextualized embeddings.

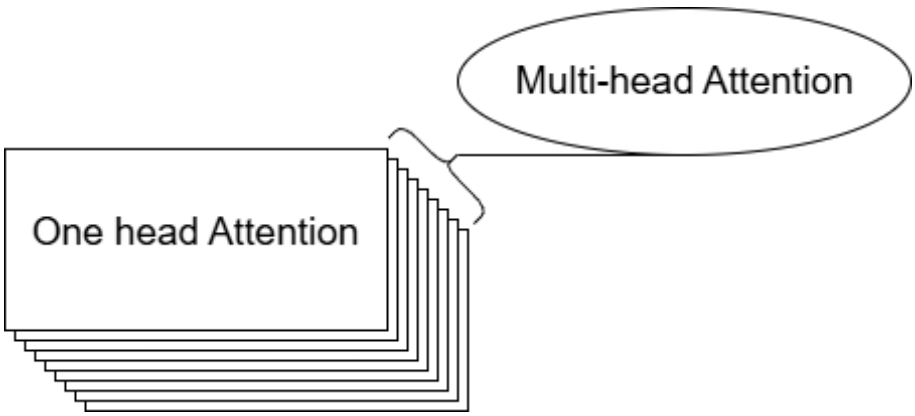
$$\vec{E}_i + \Delta\vec{E}_i = \vec{E}_i{}'$$

- $\vec{E}_i$ is the original embedding,
- $\Delta\vec{E}_i$ is the context-based adjustment (the change),
- $\vec{E}_i{}'$ is the updated, contextualized embedding.

Step 8: Generate the Matrix of contextualized embeddings (one row for each token).



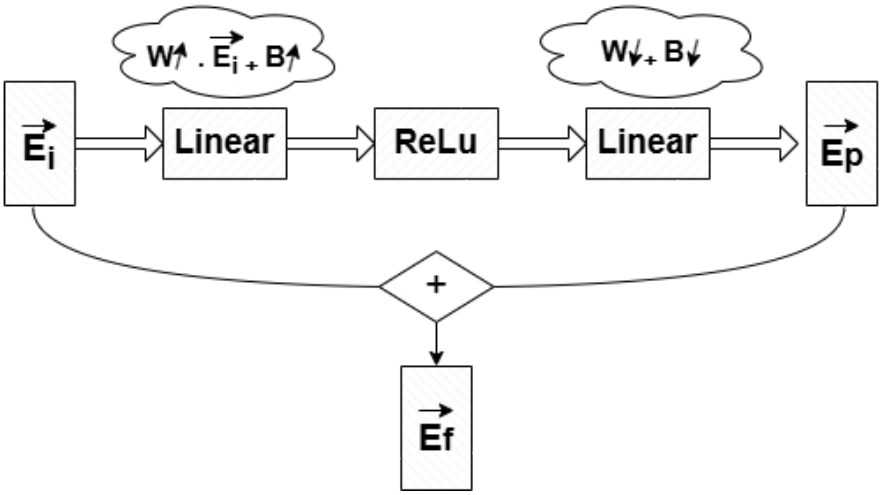
=> **RECAP:** Steps 1 to 8 describe the single-head attention process. However, in practice, this process happens multiple times in parallel and that's what we call multi-head attention.



Attention output : a Matrix

b Multi-Layer Perceptron (MLP)

After linking the embeddings with one another through the attention mechanism, the next step is to connect each embedding with the factual knowledge encoded in the LLM's corpus. This is precisely the role of the Multi-Layer Perceptron (MLP) component.
We will now explain how the LLM proceeds to enrich each embedding with external knowledge by following a series of well-defined steps. To make this process more intuitive, we summarize it in the following explanatory diagram:



Linear (1st): Define the direction of the embedding vector in the embedding space.
ReLU (Rectified Linear Unit): Reduce all the negative values to 0.
Linear (2nd): Define the vector that will be added to the input vector.

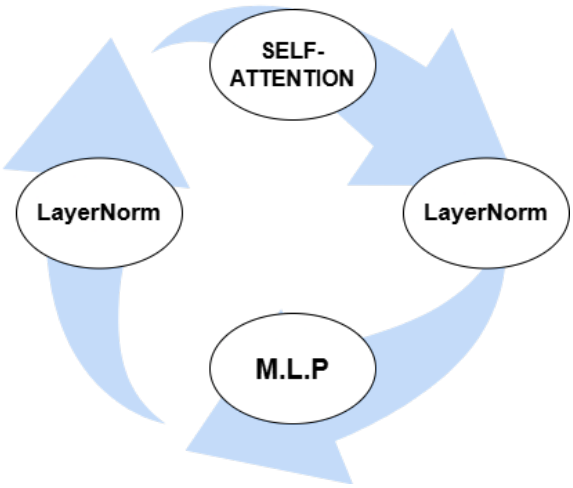
c Layer Normalization

This step is crucial to stabilize the tokens' embedding vectors and ensure consistent scale and better training or inference performance.
And we will do this by normalizing the MLP output:$\overrightarrow{E_f}$

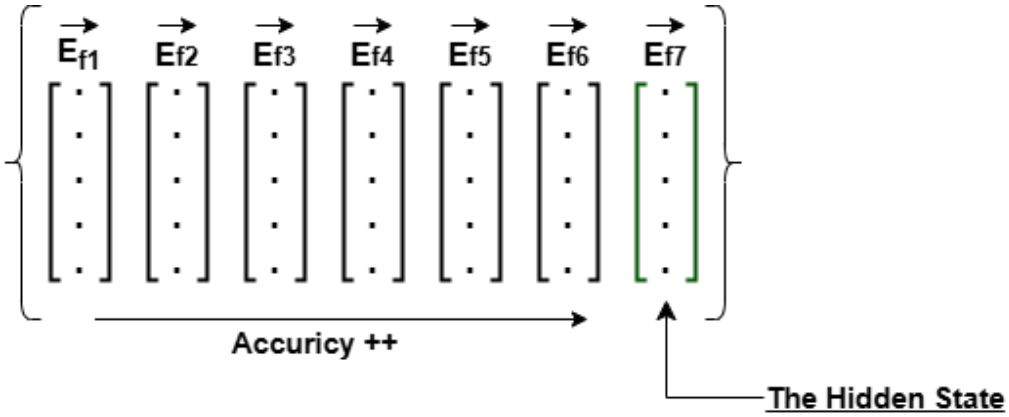
$$\text{Mean } (\mu): \quad \mu = \frac{1}{d}\sum_{i=1}^{d} x_i \qquad \text{Normalized:} \quad \text{Normalized} = \frac{E_y + \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

$$\text{Variance } (\sigma^2): \quad \sigma^2 = \frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2 \qquad \text{Final output } (Y): \quad Y = \text{Normalized} + B$$

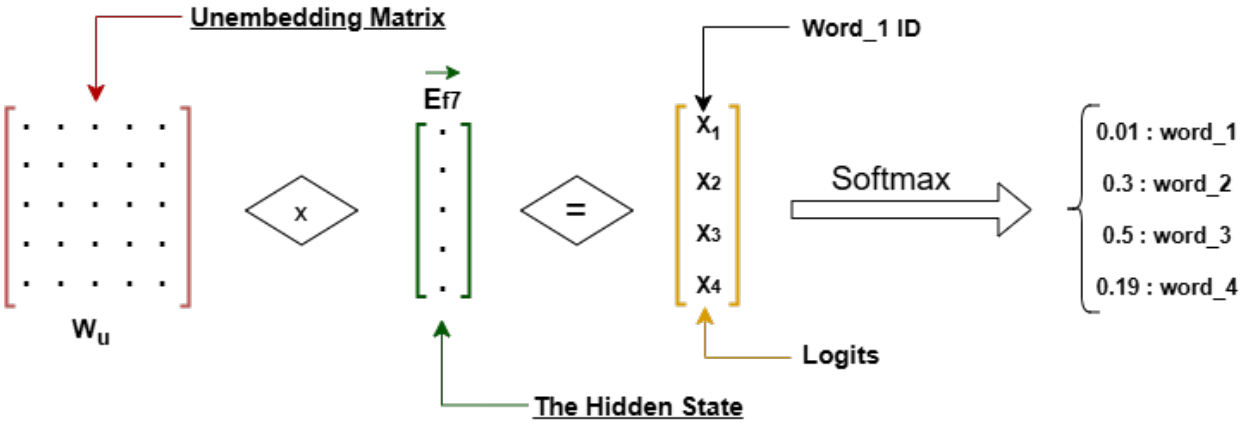
Important: This step is also between the self-attention and MLP.



This cycle repeats n times based on the model, and in the final iteration, the last layer-norm produces a final matrix. The desired output is the last vector of this matrix, referred to as the final hidden state.

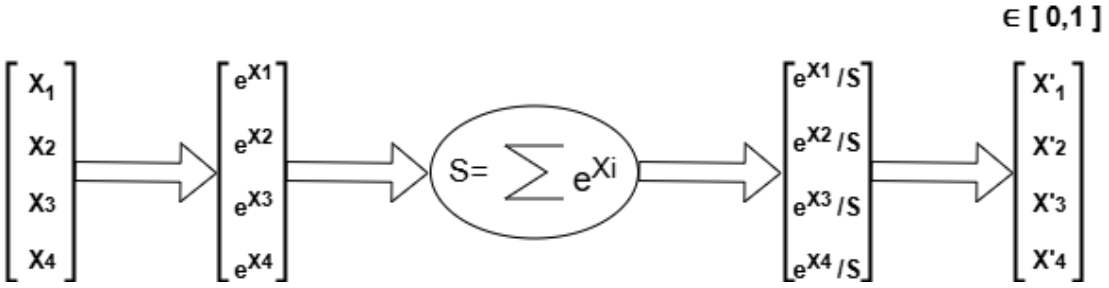


The model uses an unembedding matrix to transform the hidden state into a probability distribution over the vocabulary:



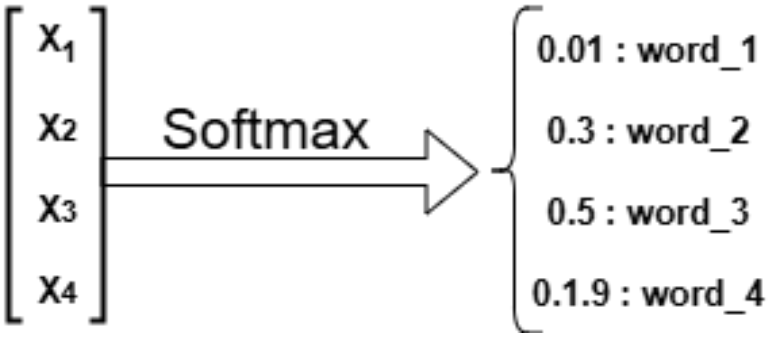
4.3 The Softmax Function

Following the forward pass, the hidden state becomes a vector of numbers within the interval $]-\infty, +\infty[$. Since predictions should represent probabilities, how can we transform this vector into a list of probabilities for the next token corresponding to each element?
This is where softmax is utilized in large language models (LLMs) to normalize the vector into a list of probabilities between [0 , 1].
so how does this function operate?



After applying the softmax function, all values in the vector now range between 0 and 1, allowing us to interpret them as probabilities that the large language model (LLM) uses to predict the next token. However, this approach has a limitation: the softmax function often produces values that are significantly different from one another. This makes it easy for the LLM to consistently select the token with the highest probability, which can reduce creativity to zero, as the model tends to predict the same word repeatedly.

To address this issue, researchers introduced a new parameter called "temperature" to the formula. Temperature is a small value that helps minimize the differences between the vector's values, resulting in more uniform probabilities. This encourages the LLM to select different tokens each time, rather than always choosing the highest one, thereby enhancing the model's creativity. To implement this temperature adjustment, each value in the vector is divided by the temperature parameter. Typically, a temperature value less than 1 increases focus on the most likely tokens, while a value greater than 1 promotes more diverse and creative outputs.



The first three steps are crucial in every process involving an LLM: tokenization, embedding, and attention-based processing. However, the rest of the process differs depending on whether the model is in **training** or **inference** (usage) mode.
In this section, we will tackle each process separately, starting first with the training phase.

4.4 LLM Training

To train an LLM, the next step is to take the list of predicted probabilities and compute the loss, which measures how "surprised" the model is when its predictions differ from the correct tokens. This loss quantifies the error between the predicted and actual tokens.
Then, the model performs backpropagation, which calculates the gradients (the direction and magnitude) of change needed, and uses these gradients to update the model's weights.
This process helps the model improve its predictions over time.

a Loss Calculation

To measure how "surprised" the model is when its predictions differ from the correct tokens, we compute the **Cross-Entropy Loss**, which quantifies the difference between the predicted probability distribution and the actual (true) distribution.
The cross-entropy loss is defined by the following formula:

$$\mathcal{L} = -\frac{1}{B \cdot T} \sum_{b=1}^{B} \sum_{t=1}^{T} \log P(y_t^{(b)} \mid y_{<t}^{(b)})$$

- B: batch size
- T: sequence length
- $y_t^{(b)}$: the target token at time step t for sample b
- $y_{<t}^{(b)}$: all previous tokens before t in sample b
- $P(y_t^{(b)} \mid y_{<t}^{(b)})$: the probability assigned by the model (after softmax) to the correct token

b Backpropagation

Now that the model knows how far off its predictions are compared to the correct tokens in the corpus ”thanks to the loss value” it needs to adjust itself accordingly. If the prediction was wrong, the model should change its internal parameters(weights) if it was right, it should reinforce that behavior. This process is known as optimization.

There are many optimization algorithms used in deep learning, such as **SGD**, **Adam**, and others. However, to keep things simple, we can summarize the core idea through the concept of **gradient** calculation.

By computing the gradient (the derivative of the loss with respect to each weight), the model determines how much each weight contributed to the error. The gradient acts like a guide, telling the model how to update its weights in order to reduce future errors and make better predictions over time.

$$\text{Gradient} = \hat{y} - y$$

- $\hat{y}$: the predicted probability (from the model, after softmax)
- y: the true label (usually one-hot encoded)

That’s it, you now have a solid understanding of what happens behind the scenes during the training of a large language model. From tokenization and embedding to attention mechanisms, loss calculation, gradient computation, and weight updates, we’ve explored how the model gradually learns to predict language accurately. But training is only half the story.

To complete our journey through the architecture and functioning of LLMs, we now shift our focus to the **inference phase**, the stage where the model is actually used in real-world applications. This is the typical scenario where a user types a prompt and instantly receives a coherent, often intelligent, response.

But what really happens under the hood when you interact with an LLM? How does the model interpret your input, generate meaningful text, and appear to communicate like a human? What mechanisms guide its generation of sentences, word by word, from learned patterns?

These are the questions we’ll dive into next to uncover how inference works, how the model produces text in real-time, and how it becomes a powerful tool for human-AI interaction.

4.5 LLM Inference

The inference step is the phase where the LLM is already trained and begins using what it has learned to assist humans with various tasks such as answering questions, generating content, or solving problems. Since the model has completed its training, we no longer evaluate or adjust its predictions, we assume it has been sufficiently trained to produce accurate and meaningful responses. Technically, the inference process involves two main stages: output generation and output delivery. In the output generation stage, the model receives the probabilities list from the **forward pass** step, processes it, and generates a response internally in the form of numerical vectors. Then, in the output delivery stage, these numerical outputs are decoded and converted into natural human language that the user can understand. In the following section, we will explore the first stage of inference: output generation.

a Output Generation

At this stage, each number in the probability distribution returned by the softmax function corresponds to a specific token (word or symbol) from the model's vocabulary(Corpus). The model selects one of these tokens based on a **decoding strategy** that reflects how it was trained, such as **Greedy search**, **Sampling**, or **Beam search**. Once a token is chosen, the model identifies its corresponding **ID** in the vocabulary. This token ID becomes the **final output** for that time step and is appended to the sequence of previously generated tokens.
This updated sequence, now including the newly generated token, becomes the input for the next generation step. Through this iterative process, the model generates text **token by token** until it reaches a **stopping condition**, such as a maximum length or an end-of-sequence token.

====> **Exemple of Output:** **ID=325("and")**

b Output Delivery

Now, let's tackle the final step of inference: **output delivery**. In the previous step, the model repeatedly generated tokens one by one, appending each new token ID to the input sequence until a complete sequence was formed. This process continues based on a stopping condition defined during training. As a result, the model produces a sequence of token IDs:

$$[325,\ 2023,\ 2003,\ 1037,\ 2742,\ 102]$$

Each of these IDs corresponds to a word or symbol in the model's vocabulary.

Okay, so now the model has completed the generation process and knows exactly which tokens it will add to the user's input. However, these tokens are still in the form of numerical IDs, which are not understandable to humans. To transform this sequence into natural language, the model must go through a final stage called **detokenization**. This process involves four key steps:

1. **ID to Token Mapping:** Each token ID is mapped back to its corresponding token (subword, word, or symbol) using the model's vocabulary. ID ==> Word
2. **Token Merging:** If the tokenizer used subword units, adjacent tokens are merged to reconstruct the original words. Play + ing ==> Playing
3. **Text Normalization:** Punctuation, spacing, and casing are corrected to ensure the output resembles fluent natural language.
4. **Final Output Rendering:** The resulting string is prepared for display or returned to the user as the final response.

5 Conclusion

In this Technical Booklet, we have explored in depth the internal functioning of Large Language Models (LLMs), covering both the **training** and **inference** phases. During the training phase, we saw how the model learns to understand and generate language through a series of critical steps: tokenization, embedding, attention mechanisms, probability prediction, loss computation, gradient calculation, and weight updates. This process allows the model to gradually improve its ability to predict the next token in a sequence, based on vast amounts of text data.

Once the model is trained, it enters the inference phase the stage where it is actually used by end users. We explained how, during inference, the model receives an input, generates outputs token by token through decoding strategies, and finally converts its internal numerical predictions into human-readable text. This final step involves detokenization, which transforms token IDs into coherent natural language.

Together, these two phases, training and inference represent the complete lifecycle of a large language model, from learning patterns in data to producing intelligent, human-like responses in real time. Understanding this full pipeline is essential not only for building effective language systems, but also for appreciating the remarkable complexity and design behind modern AI communication tools.

6 Contact

Connect with me on : **LinkedIn**
Check out my projects on : **GitHub**