The Math behind transformers

Eddie Conti

2023/2024

Contents

1 Introduction

Large Language Models are AI systems capable of understanding and generating human language by processing vast amounts of text data. In recent years, specifically from 2017, the use of LLMs significantly increased thanks to the introduction of the Transformer architecture. This particular structure is characterized by several aspects as:

- Context-depending mechanism;
- Attention mechanism;
- Multitask-training using huge amounts of text;
- Parallel processing.

Even though the deep architecture is composed by several components, it can be summarized as a tool to find a proper embedding of the input and the output. The representation of text is the main aspect of LLM. Formally, given a vocabulary V of size $|V| = l$, an embedding is a function

$$E\colon w \to \mathbb{R}^d$$

so that a word is represented by a row vector with d entries and therefore a vocabulary V will be represented by a matrix $M \in \mathcal{M}_{\mathbb{R}}(l, d)$, where $\mathcal{M}_{\mathbb{R}}(l, d)$ is the space of real matrices of dimension $l \times d$. However, finding the proper function E is not obvious: the easiest case would be to to represent each word $v_i \in V$ with the unit vector $e_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ where 1 is in the $i-$th position. Unfortunately, this embedding is not suitable to understand and replicate human language. Indeed, if we perform $v_i \cdot v_j = 0$, namely every word are represented by orthogonal and so independent vectors. When representing the text with vectors we aim at capturing the similarity of the word based on the context: in this regards we expect that words as 'tree', 'leaf' are similar while 'train' and 'rock' are dissimilar. This is not the only goal desirable: usually language is mostly auto-referencial in the sense that we have words that refers/replace other words even though their meanings are not related. To be more clear, let us consider the following example

Luca is studying in Spain. He is very happy.

In the previous sentence, the word 'He' is replacing 'Luca' and we have somehow to capture this aspect of language. Furthermore, to be able to create an user interactive tool it is essential to capture this auto-referencial aspect at a high level. Everything is about embedding, or in other terms, finding a good way to represent your input so that your architecture can process and elaborate it. Transformer architecture meets these requirements and is able to produce embeddings that are suitable for a variety of tasks such as translation, text summarization, natural language generation and more. Nevertheless, the amount of parameters involved is extremely large and expensive. Moreover, the core of Transformer, the Self-Attention mechanism, lacks of a proper mathematical treatment. Given this limitations, this thesis delve into the aspects of transformers in mathematical therms.

In the first Chapter we analyze the architecture explaining each component in details. Then, we shift our attention to a posteriori approaches that reduce the complexity of the transformers architecture. In detail, we explore Sparse Attention, Linear Attention and matrix factorization approaches to scale linearly the complexity. In the third chapter, we focus on the mathematical and geometrical meaning of the attention formula. The experimental session in then described in this chapter and conclusion are drawn in the final chapter.

1.1 Seq2seq

Commonly, LLMs are used for text generation and encoder-decoder is the standard modeling paradigm for sequence-to-sequence tasks. The encoder reads source sequence and produces its representation; while the decoder uses source representation from the encoder to generate the target sequence. Given an input sequence $x = (x_1, \dots, x_n)$ we aim at finding the target sentence $y^* = (y_1^*, \dots, y_{n'}^*)$ that is the most probable given the input. Formally, the target sequence that maximizes the conditional probability $p(y|x)$:

$$y^* = \operatorname*{argmax}_y p(y|x).$$

In the case of language models, the output depends on the previous generated outputs, the probability of a sequence can be decomposed as

$$p(y_1, \dots, y_{n'}) = p(y_1) \cdot p(y_2|y_1) \dots p(y_{n'}|y_1, \dots, y_{n'-1}) = \prod_{t=1}^{n'} p(y_t|y_{<t})$$

and so taking into account the input and the parameters the aim is to determine

$$y^* = \operatorname*{argmax}_y \prod_{t=1}^{n'} p(y_t|y_{<t}, x).$$

Neural seq2seq models are trained to predict probability distributions of the next token given previous context (source and previous target tokens). At the timestep t a model predicts a probability distribution

$$p^{(t)} = p(*|y_1, \dots, y_{t-1}, x_1, \dots, x_n).$$

The target at this step is $p^* = \text{one-hot}(y_t)$, i.e., we want a model to assign probability 1 to the correct token, y_t, and zero to the rest. The standard loss function is the cross-entropy loss:

$$Loss(p^*, p) = -p^* \log(p) = -\sum_{i=1}^{|V|} p_i^* \log(p_i)$$

but since only one p_i^* (for the correct token y_t) is non-zero we get

$$Loss(p^*, p) = \log(p_{y_t}) = -\log(p(y_t|y_{<t}, x)).$$

Hence, at each step we maximize the probability a model assigns to the correct token. Now that we know how to the model is trained, we focus on how to generate the sentence, i.e, how to find y^* that satisfies

$$y^* = \operatorname*{argmax}_y \prod_{t=1}^{n'} p(y_t|y_{<t}, x).$$

The total amount of feasible solution is $|V|^n$ which is too large, hence usually there two approaches can be used to find an approximate solution: Greedy Decoding and Beam Search. The former, consists of generating at each a token with the highest probability. Even though it is a good baseline, mathematically

$$\underset{y}{\text{argmax}} \prod_{t=1}^{n'} p(y_t|y_{<t}, x) \neq \prod_{t=1}^{n'} \underset{y_t}{\text{argmax}}\, p(y_t|y_{<t}, x).$$

As a consequence, Beam Search, tries to tackle this issues by keeping track of several most probably hypotheses. At each step we continue each of the current hypotheses and pick top-N of them.

1.1.1 Capturing the context: Attention mechanism

So far we introduced the framework of LLMs and in particular seq2seq models. The simplest architecture consists of two RNNs (LSTMs): encoder RNN reads the source sentence, and the final state is used as the initial state of the decoder RNN. The hope is that the final encoder state "encodes" all information about the source, and the decoder can generate the target sentence based on this vector. However, the constraint of condensing all information into a single vector fails to capture the complexity of language and often proves insufficient for synthesizing the input. This motivates the introduction of the concept of attention (see paper [2]).
The attention mechanism is the foundation of the transformer architecture that will be discussed afterwards. The basic idea is to allow the model to weigh the importance of each word within a sequence according to context. In this way, the model is able to capture long-range word relationships without relying on rigid sequential structures. In simple terms, the attention-mechanism looks at an input sequence and decides at each step which other parts of the sequence are important. To solve the problem of the bottleneck in LSTMs, attention mechanism gives representations for all source tokens: instead of passing the last hidden state of the encoding stage, the encoder passes all the hidden states to the decoder.

In more details, figure 1 shows what the architecture performs: at every step given all the hidden states from the encoder, we produce a context vector that is concatenated to internal state and then we use a fully connected NN to produce the output word. In mathematical terms, denoted $s_1, \ldots, s_n$ all the encoder states and h_t the decoder state, we compute attention weights as

$$a_{t,k} = \frac{exp(\text{score}(h_t, s_k))}{\sum_{i=1}^{n} exp(\text{score}(h_t, s_i))} \quad k = 1, \ldots, n$$

where score is a scoring function that encodes how relevant is a source token for that target step. The most popular ways to compute attention scores are:

$$\text{Luong dot} \quad \text{score}(h_t, \hat{h}_s) = h_t^T \hat{h}_s$$
$$\text{Luong multiplicative} \quad \text{score}(h_t, \hat{h}_s) = h_t^T W_a \hat{h}_s$$
$$\text{Bahdanau} \quad \text{score}(h_t, \hat{h}_s) = v_a^T \tanh W_a[h_t; \hat{h}_s]$$

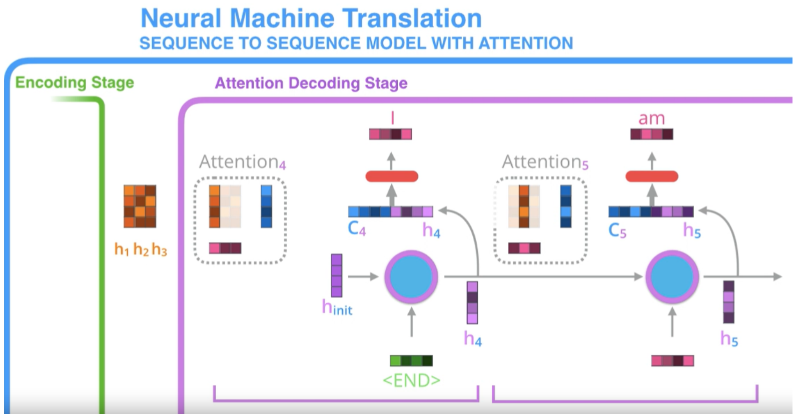


Figure 1: Details behind the mechanism

Therefore, the context vector is

$$c_t = a_{t,1}s_1 + \ldots + a_{t,m}s_m = \sum_{i=1}^{m} a_{t,i}s_i$$

here we concatenate c_t and h_t and compute

$$W[c_t; h_t]$$

where W is a matrix trained with all the other component that produces the output, i.e., the expected word.

1.2 The transformer architecture

A transformer is an architecture presented for the first time in the 2017 paper *Attention Is All You Need* for transforming one sequence into another one with the help of two parts: Encoder and Decoder.
Let us now explore the architecture in Figure 2. As we can see it is an Encoder-Decoder architecture: the former consists of encoding layers that process the input tokens iteratively one layer after another, while the latter consists of decoding layers that iteratively process the encoder's output as well as the decoder output's tokens so far.
The sequence is presented as a list of tokens x_i for $i = 1, \ldots, n$ usually as one-hot encodings. Since the vocabulary size l is very large we reduce the dimensionality by computing

$$X \cdot W^E, \quad W^E \in \mathbb{R}^{l \times d} \text{ and } d = 512$$

where the matrix W^E is trainable. Now we have a matrix that is representative of our input sequence of size (n, d). From now on we will omit that we actually work with tensors because the input is not a single sentence but rather a set of sentence of dimension $batch - size$. Lastly, we multiply $X \cdot W^E$ by $\sqrt{512}$ for stability reasons.

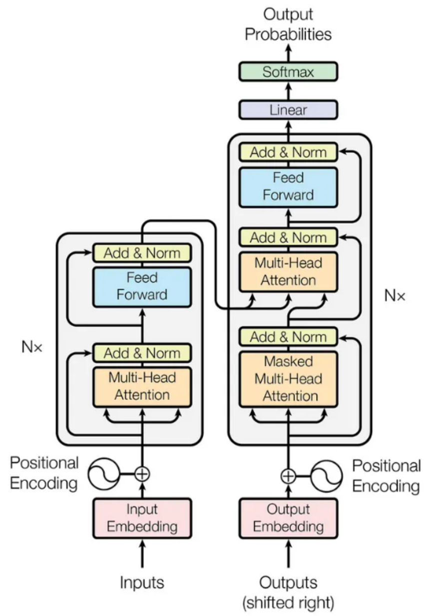


Figure 2: The transformer architecture. Image from [1].

1.2.1 Positional Encoding

As we already pointed out, all the tokens are presented to the Transformer simultaneously. However, this results in the loss of the original order of tokens in the input sequence. The purpose of positional encoding is to encapsulate the relative positions of tokens within a target sequence. Before presenting the original function let us delve our intuition on why positional encoding is necessary. The general idea is to allow the model to differentiate between words with similar meanings but different positions in the sentence. For example if we consider the following

Luca did not won the football tournament and he was happy;

Luca won the football tournament and he was not happy.

Without any additional information about the positions of the words, the model might treat the sentences similarly although the meaning is different. A first approach could be to add the index of the of the token to its representation: $x_i \mapsto x_i \cdot W^E + i$. However as we will see in the next pages we are going to perform matrix operations and exponentiation and $x_i \cdot W^E \ + i$ grows as the length of the sequence grows, which is dangerous as it can lead to exploding gradients. Hence, one possibility would be to normalize index i so it ranges values between $[0, 1]$. Unfortunately, also this strategy will confuse the model if the input text length varies since the result would contain different positional embeddings for the same positions for different input sequences. The solution presented in paper [1] involves using a sine and cosine function to create a d dimensional vector for each position in the sentence. The positional encoding is defined as a function $f\colon \mathbb{N} \to \mathbb{R}^d$ where the $i-$th entry is

$$f(t)^{(i)} = \begin{cases} \sin(\omega_k \cdot t) & \text{if } i = 2k, \\ \cos(\omega_k \cdot t) & \text{if } i = 2k+1 \end{cases} \tag{1}$$

with

$$\omega_k = \frac{1}{N^{2k/d}}.$$

Alternatively, it can be expressed as a complex valued function as

$$f(t)^{(i)} = e^{it/\omega_k} \quad k = 0, \ldots, d/2 - 1.$$

In paper [1] $N = 10000$, but in general is a free parameter that should be significantly larger than the biggest k that in our notation is $d/2 - 1$. The main idea behind this function is that it allows one to perform shifts as linear transformations and this is useful to express the relative position. For the sake of simplicity, let us fix $i = 2k$

$$\begin{aligned} f(t+s)^{(i)} &= \sin(\omega_k \cdot (t+s)) = \sin(\omega_k \cdot t + \omega_k \cdot s) \\ &= \sin(\omega_k \cdot t)\cos(\omega_k \cdot s) + \sin(\omega_k \cdot s)\cos(\omega_k \cdot t) \\ &= a\sin(\omega_k \cdot t) + b. \end{aligned}$$

Similarly, for $i = 2k+1$

$$\begin{aligned} f(t+s)^{(i)} &= \cos(\omega_k \cdot (t+s)) = \cos(\omega_k \cdot t + \omega_k \cdot s) \\ &= \cos(\omega_k \cdot t)\cos(\omega_k \cdot s) - \sin(\omega_k \cdot t)\sin(\omega_k \cdot s) \\ &= a'\cos(\omega_k \cdot t) + b', \end{aligned}$$

hence for a fixed offset s we can express the $(t+s)-$th position a linear combination of $t-$th position.
Furthermore, there is another relevant reason for this specific function: the output for a fixed t, i.e. for a fixed position, is a d dimensional vector

$$(\sin(\omega_1 \cdot t), \ldots, \sin(\omega_d \cdot t))$$

where t influences the oscillation of the sine/cosine function. As a consequence when changing the parameter t we obtain a different vector that is representative of that position of the word. The positional embedding is a matrix $P \in \mathbb{R}^{N \times d}$ and it is then added to $X \cdot W^E$ in a compatible way

$$X \cdot W^E + P[:n, :].$$

A dropout of 0.1 is applied to prevent overfitting.

1.2.2 Transformer Block

The crucial step of the architecture is the so called the transformer block consisting of several parts as shown in Figure 2. Let us denote, for simplicity, by $X \in \mathbb{R}^{n \times d}$ the position-aware word embedding. We can think of the transformer block as a function depending on some parameters θ

$$f_\theta \colon \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d}.$$

In this scenario, the name *transformer* captures that function f_θ is just transforming the input as the domain and codomain are the same. The block can be described in mathematical terms as it follows: Given X, then $f_\theta(X) = Z$, where:

$$\begin{aligned}
Q^{(h)}(x_i) &= W_{h,q}^T(x_i),\ K^{(h)}(x_i) = W_{h,k}^T(x_i),\ V^{(h)}(x_i) = W_{h,v}^T(x_i) \quad W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d\times k},\\
\alpha_{i,j}^{(h)} &= \text{softmax}_j\Big(\frac{Q^{(h)}(x_i)\cdot(K^{(h)}(x_j))^T}{\sqrt{k}}\Big),\\
u_i' &= \sum_{h=1}^{H} W_{c,h}^T \sum_{j=1}^{n} \alpha_{i,j}^{(h)} V^{(h)}(x_j), \qquad W_{c,h} \in \mathbb{R}^{k\times d},\\
u_i &= \text{LayerNorm}(x_i + u_i'; \gamma_1, \beta_1), \qquad \gamma_1, \beta_1 \in \mathbb{R}^d,\\
z_i' &= W_2^T \text{ReLU}(W_1^T u_i + b_1) + b_2, \qquad W_1 \in \mathbb{R}^{d\times m}, W_2 \in \mathbb{R}^{m\times d}, b_1, b_2 \in \mathbb{R}\\
z_i &= \text{LayerNorm}(u_i + z_i'; \gamma_2, \beta_2), \qquad \gamma_1, \beta_1 \in \mathbb{R}^d.
\end{aligned}$$

The LayerNorm function is defined as

$$\text{LayerNorm}(z; \gamma, \beta) = \gamma\frac{(z-\mu_z)}{\sigma_z + \epsilon} + \beta, \tag{2}$$

where ϵ is a fixed number as 10^{-6} to prevent division by 0

$$\mu_z = \frac{1}{k}\sum_{i=1}^{k} z_i, \qquad \sigma_z = \sqrt{\frac{1}{k}\sum_{i=1}^{k}(z_i - \mu_z)^2}.$$

Thus, given $x \in \mathbb{R}^{n\times d}$ we produce $z \in \mathbb{R}^{n\times d}$ such that $Z = f_\theta(X)$ where f describes the transformer block and the parameter θ consists of entries of weight matrices W along with the parameter of LayerNorm. In the general setting the final output is a stack of transformer blocks and can be derived as

$$f_{\theta_L} \circ \ldots \circ f_{\theta_1}(X) \in \mathbb{R}^{n\times d}.$$

In the paper the hyperparameters are $d = 512$, $k = 64$, $m = 2048$, $H = 8$ and $L = 6$. Let us now explore each step involved in the transformer block. The first equation is the formalization of the so called query, key and value vectors. First of all, three matrices $W_q, W_k, W_v \in \mathbb{R}^{d\times d}$ are randomly initialized: these matrices are then split to obtain sub-matrices of dimension (n, k) as it follows:

$$\begin{aligned}
W_q &\to W_{1,q} = W_q[:, : d_k], \ldots, W_{d/k,q} = W_q[:, d - d_k : d],\\
W_k &\to W_{1,k} = W_k[:, : d_k], \ldots, W_{d/k,k} = W_k[:, d - d_k : d],\\
W_v &\to W_{1,v} = W_v[:, : d_k], \ldots, W_{d/k,v} = W_k[:, d - d_k : d].
\end{aligned} \tag{3}$$

After reducing the dimensionality from (d, d) to (d, k), we compute query, key, value vectors as

$$Q^{(h)}(x_i) = W_{h,q}^T(x_i),\ K^{(h)}(x_i) = W_{h,k}^T(x_i). \tag{4}$$

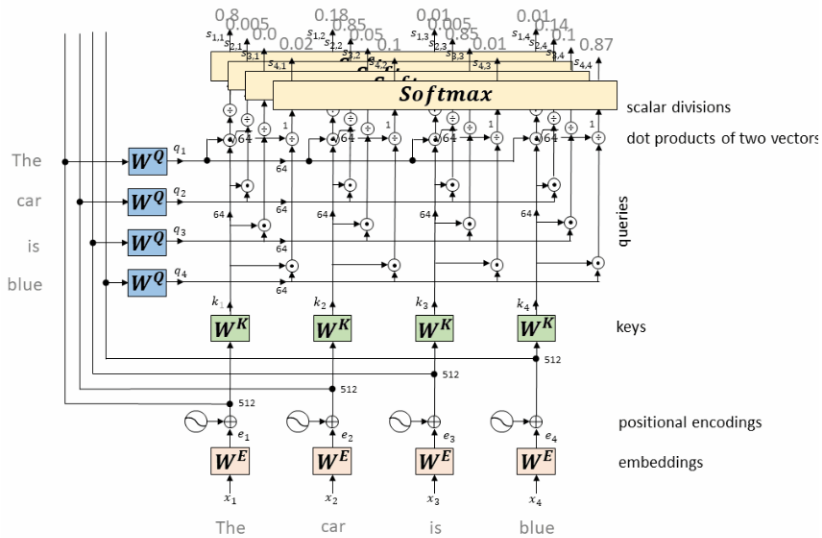


Figure 3: Embeddings, query and key vectors

We denote d/k with H i.e. the number of heads. The geometric and intuitive meaning of these matrices can be found in chapter 3. The second equation describes the self-attention mechanism

$$Attention(Q^{(h)}, K^{(h)}, V^{(h)}) = Softmax\Big(\frac{Q^{(h)}(K^{(h)})^T}{k}\Big)V^{(h)}, \tag{5}$$

and we usually refer to them as attention values, while the attention weights are

$$Softmax\Big(\frac{Q^{(h)}(K^{(h)})^T}{k}\Big) \tag{6}$$

We compute the attention for each "head" i.e. each splitting of W_q, W_k, W_v so that

$$head_i = Attention(Q^{(i)}, K^{(i)}, V^{(i)}), \quad i = 1, \ldots, H.$$

It is essential to observe that each $W_{i,q}, W_{i,k}, W_{i,v}$ is of dimension (n, k) so that is "looking" at the whole sentence but in different positions, i.e. is looking at different attributes of the embedded input. The next equation is the output of the so called Multi-head attention. If Q, K, V refers to the concatenation of $Q^{(h)}, K^{(h)}, V^{(h)}$ then the third equation can be expressed as

$$\text{MultiHead}(Q, K, V) = Concat(head_1, \ldots, head_h)W_c \tag{7}$$

where W_c is a matrix of dimension $(H \times k, d) = (d, d)$ and so the output of Multi-Head attention lies in $\mathbb{R}^{n\times d}$. The role of the matrix W_c is to extract the most valuable information from each head by weighting their outputs and this is why in linear expression can be split into $W_{c,h}$ for $h = 1, \ldots, H$.

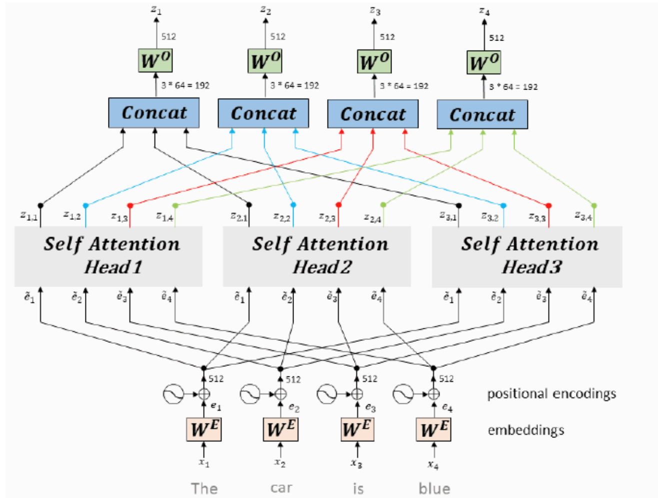


Figure 4: Multi-Head Attention. The matrix W^O is what we denoted with W_c

1.2.3 Motivation for scaling the dot products

Before proceeding with the transformer block, it is worthy to motivate the division by $\sqrt{k}$ in computing the attention weights. This factor is applied before the softmax because it leads to more stable gradients. To understand this, we recall that the softmax function is a vector function

$$\begin{aligned} Softmax\colon \mathbb{R}^n &\to \mathbb{R}^n \\ z_i &\mapsto s_i := \frac{e^{z_i}}{\sum_j e^{z_j}}. \end{aligned} \tag{8}$$

Now, if we compute the partial derivatives for $k \neq i$

$$\begin{aligned} \frac{\partial s_i}{\partial z_i} &= \frac{\partial}{\partial z_i}\frac{e^{z_i}}{\sum_j e^{z_j}} = \frac{e^{z_i}}{\sum_j e^{z_j}} + e^{z_i}\frac{\partial}{\partial z_i}\frac{1}{\sum_j e^{z_j}} = \frac{e^{z_i}}{\sum_j e^{z_j}} - \left(\frac{e^{z_i}}{\sum_j e^{z_j}}\right)^2 = s_i \cdot (1 - s_i), \\ \frac{\partial s_i}{\partial z_k} &= \frac{\partial}{\partial z_k}\frac{e^{z_i}}{\sum_j e^{z_j}} = e^{z_i}\partial z_k \frac{1}{\sum_j e^{z_j}} = \frac{-e^{z_i}e^{z_k}}{\left(\sum_j e^{z_j}\right)^2} = -s_i \cdot s_k. \end{aligned}$$

Therefore, the Jacobian of (8) can be expressed as

$$J_s = \begin{pmatrix} s_1 \cdot (1 - s_1) & -s_1 \cdot s_2 & \cdots & -s_1 \cdot s_n \\ -s_2 \cdot s_1 & -s_2 \cdot (1 - s_2) & \cdots & -s_2 \cdot s_n \\ \vdots & \vdots & \ddots & \vdots \\ -s_n \cdot s_1 & -s_n \cdot s_2 & \cdots & -s_n \cdot (1 - s_n). \end{pmatrix}$$

It is immediate to observe that the Jacobian becomes zero if occurs one among

$$s_1 = (1, 0, \ldots, 0),\ s_2 = (0, 1, 0, \ldots, 0), \ldots,\ s_n = (0, \ldots, 0, 1). \tag{9}$$

However, the softmax function is not scale invariant, which means that the higher we scale the inputs, the more the largest input dominates the output. Therefore, for large inputs the softmax function is generating outputs which closely resemble the values in (9). To tackle the problem of the vanishing gradient is necessary to rescale the dot products as the larger the dimension d of the key vectors and query vectors, the larger the dot products will tend to be.

1.2.4 Layer Normalization and Feed Forward

At this point the output of the Multi-Head Attention U' is combined with X coming from residual connection. The idea is to still consider the original embedding and to move smoothly through the architecture. Vectors u'_i and x_i are combined using the Layer Normalization function in (2). In this worthy to note that the parameter γ is used to avoid division by 0. The output is then

$$U = \gamma_1 \frac{X + U' - \mu_{x,u}}{\sigma_{x,u} + \epsilon} + \beta_1 \in \mathbb{R}^{n \times d}$$

At this point, we apply to each u_i a Feed Forward network consisting of two linear transformations with a ReLU activation in between. The dimensionality of input and output is d, and the inner-layer has dimensionality $m = 2048$ described by:

$$z'_i = \max(0; u_i W_1 + b_1) W_2 + b_2$$

where $W_1 \in \mathbb{R}^{d \times 2048}$, $W_2 \in \mathbb{R}^{2048 \times d}$ and $b_1, b_2 \in \mathbb{R}$. Finally, we again employ a residual connection and a normalization around the fully connected FFN analogously to as described above and produce the final embedding z_i for each x_i.
All of these operations can be sintetized using the function f_θ such that

$$f_\theta(X) = Z.$$

However, this is just a transformer block, but the encoder is made by L transformer blocks stacked, so that the final context aware embedding is

$$f_{\theta_L} \circ \ldots \circ f_{\theta_1}(X) \in \mathbb{R}^{n \times d}.$$

The general motivation for such a depth architecture is that in order to be able to capture the relationship between word, which is a crucial aspect as we said for translating and producing language, it is essential to look at the input text in many different ways. Similarly to what happens to images with convolutional NN, here each encoder is capturing a depth-dependent connection among the input. Language is extremely complex and heterogeneous and therefore we need a complex structure that is able to apprehend all facets of it.

1.2.5 The decoder

The decoder is responsible for sequentially generating output based on the information processed by the encoder. In its generality the structure is similar to the encoder's one, even though particular attention must be given to some aspects. First of all, the decoder

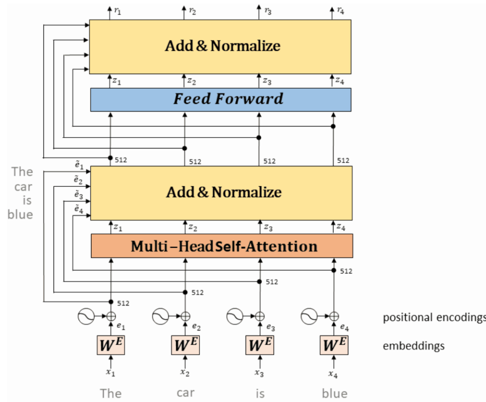


Figure 5: The transformer block

takes positional information and embeddings of the output sequence as its input. Here, during self-attention, the decoder employs a mask to ensure that each position can only be influenced by preceding positions in the sequence. This mask is crucial to ensure that the model generates sequential output correctly without "looking ahead" during generation. The masked- multi head attention is pretty similar to what we described before, however we apply a method to prevent computing attention scores for future words.

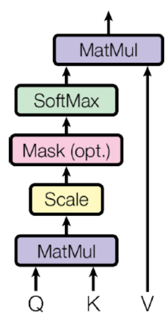


Figure 6: Details on how attention coefficients are computed

As Figure 6 points out the mask is applied after the scaling operation. Let A be the matrix of scaled values $a_{i,j}$ and M the masking matrix defined as

$$\begin{pmatrix} 0 & -\infty & -\infty & \cdots & -\infty \\ 0 & 0 & -\infty & \cdots & -\infty \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & -\infty \\ 0 & 0 & \cdots & \cdots & 0 \end{pmatrix}.$$

If we add matrix M to matrix S we obtain a matrix of masked scores whose elements are s_{ij} if $i \geq j$ and $-\infty$ otherwise. At this point if we apply the softmax to each row vector (the values for each token) if we have $-\infty$ then the output of the softmax will be 0. As a consequence, we end up with a matrix of the form

$$\begin{pmatrix} a'_{1,1} & 0 & 0 & 0 & 0 \\ a'_{2,1} & a'_{2,2} & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a'_{n-1,1} & a'_{n-1,2} & \cdots & a'_{n-1,n-1} & 0 \\ a'_{n,1} & a'_{n,2} & \cdots & \cdots & a'_{n,n} \end{pmatrix}.$$

The attention coefficients are computed as

$$\text{MaskedAttention}(Q,K,V) = \text{softmax}\left(M + \frac{QK^{\mathrm{T}}}{\sqrt{k}}\right)V. \tag{10}$$

It is worthy to note that even if in (10) we end up with a full matrix, the $j-$th row, i.e, the attention coefficients for input x_j, are computed as

$$(a'_{j,1}, \ldots, a'_{j,j}, 0, \ldots, 0) \cdot V.$$

In other words, they are linear combinations of $a'_{j,1}, \ldots, a'_{j,j}$ and this encodes exactly that the decoder is only influenced by preceding positions in the sequence. To sum up, the output of the first Multi-Head attention is a masked output vector with information on how the model should attend on the decoder's input.

In the second Multi-Head attention layer we use cross-attention so that the decoder interacts with the output of the encoder Z. If we denote again with X the input of the decoder, we apply the attention formula as

$$\text{softmax}\left(\frac{XZ^T}{\sqrt{k}}\right)Z$$

so that the query is from the decoder while key and values are from the encoder. This is consistent with what we said in the previous section. The query dialogue with the key from the encoder in order to capture the similarity between what will our output and our original input. Then to produce the output we apply the values from encoder that are trained to produce words given the similarity. Here, the architecture proceeds similarly to the encoder's one: it processes the inputs with feed forward layers and normalization. Residual connections are applied to move smoothly through the architecture for the back propagation.

Finally, the output let us say $Y \in \mathbb{R}^{n' \times d}$ where n' is the length of the output sequence, is projected into the vocabulary of size l by performing

$$Y \cdot P, \quad P \in \mathbb{R}^{d \times l}.$$

Here a softmax will produce probability scores between 0 and 1. We take the index of the highest probability score, and that equals to our predicted word. The decoder then takes the output, add's it to the list of decoder inputs, and continues decoding again until a <end> token is predicted. Therefore, the decoding steps produces a word that is the most probable according to the input and its context and the previous word generated: in this way we gained the ability to produce meaningful sentences that are related to the input sentence.

2 The amount of parameters in LLM

Transformers are the main architecture employed for LLM. Despite their notable capabilities in understanding and generating language such as GPT, a crucial aspect of these models lies in their substantial number of parameters, particularly when applied to the English language. The willing to capture the richness of linguistic context has led to architectures of considerable size, comprised of millions or even billions of parameters. As it is clear from Chapter 1, in order to comprehend language it is essential to use a huge variety of matrices (i.e. word representations) and long distance word relationship. The largest models can have a context window sized up to 32000. For example, GPT-4; while GPT-3.5 has a context window sized from 4000 to 16000, and legacy GPT-3 has had 2000 sized context window. Also, the significant economic costs in terms of computational resources and infrastructure must taken into account.
Within this context, the challenge emerges of balancing the expressive power of advanced language models with the need to optimize computational efficiency.

Name	Release Date	Number of parameters
GPT-1	June 2018	117 million
BERT	October 2018	340 million
GPT-2	February 2019	1.5 billion
GPT-3	May 2020	175 billion
LaMDA	January 2022	137 billion
Minerva	June 2022	540 billion

Table 1: List of some LLMs with the amount of parameters

As it is clear from Table (1) training transformer-based architectures can be expensive, especially for long inputs.

2.1 Transformers for long sequences: changing the attention

Transformer architecture suffers some limitations when it comes to long sequences as the computational complexity increases quadratically with the sequence length n. Indeed, most of the complexity of the architecture lies in the self-attention, where we produce the attention coefficients. Each token x_i is related to every other token x_j in order to capture the relationship between words. This means that the resulting complexity is $O(n^2)$. Most transformer models are fixed in their sequence length as in the case of BERT model [6] which is limited to 512 tokens. However, being able to cover a variety of tasks is a desirable aspect and when it comes to document summarization, DNA processing or any task that requires long sequences, the training becomes practically infeasible, due to the enormous computational cost. In the next pages we are going to cover different techniques that reduce this quadratically dependency on sequence length. These approaches try to switch from the quadratic dependency to a linear dependency.

2.1.1 Sparse attention

Sparse attention refers to an attention mechanism where the attention of each token is limited to a subset of other tokens. BigBird, a transformer model developed by Google Research (see [7]), implements a sparse attention mechanism that reduces the quadratic dependency to linear. As they proved, the proposed sparse attention can handle sequences of length up to $8x$ of what was previously possible using similar hardware. As a consequence, BigBird drastically improves performance on various NLP tasks such as question answering and summarization. To be more precise, BigBird is a combination of global attention, local attention and random attention. In particular, BigBird consists of three main parts:

- a set of g global tokens attending on all parts of the sequence;
- all tokens attending to a set of w local neighboring tokens;
- all tokens attending to a set of r random tokens.

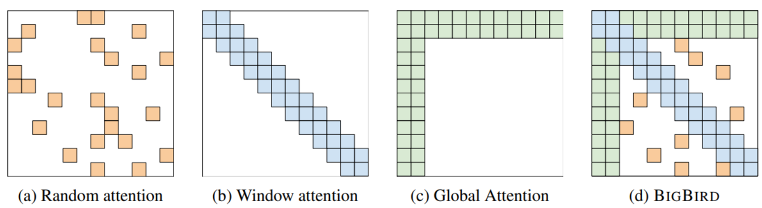


Figure 7: White color indicates absence of attention. (a) Random attention with $r = 2$, (b) sliding window attention with $w = 3$, (c) global attention with $g = 2$, (d) the combined BigBird model.

In mathematical terms, let $X = (x_1, \ldots, x_n) \in \mathbb{R}^{n \times d}$ an input sequence. The generalized attention mechanism is described by a directed graph D whose vertex set is $\{1, \ldots, n\}$. In this scenario, we denote by $N(i)$ the out-neighbors set of node i in D, or, in other terms, the set of inner products that the attention mechanism will consider. The generalized attention is defined as

$$\mathrm{ATTN}_D(X)_i = x_i + \sum_{h=1}^{H} \sigma\Big(Q^{(h)}(x_i) K^{(h)}(X_{N(i)})^T\Big) \cdot V^{(h)}(X_{N(i)}), \tag{11}$$

where $Q^{(h)}, K^{(h)} : \mathbb{R}^d \to \mathbb{R}^k$ (k is the dimension of each head) are query and key functions, $V_h : \mathbb{R}^d \to \mathbb{R}^d$ is a value function, σ is a scoring function such as softmax and H denotes the number of heads. Also note $X_{N(i)}$ corresponds to the matrix formed by only stacking $\{x_j : j \in N(i)\}$ and not all the inputs. It is clear that (11) reduces to (5) if $N(i)$ is full, in the sense that we are considering the relationship among all the input words. The general idea behind BigBird architecture is that most contexts within NLP and computational biology have data which displays a great deal of locality of reference. In this regards, fixed token x_i, tokens x_j for $i - w/2 \leq j \leq i + w/2$ are the most relevant for a context comprehension. Furthermore, adding random attention in the key function, allow to capturing less obvious or less frequent information that may be crucial for understanding the context. Furthermore, introducing randomness in attention can make the architecture more robust to variations in input data. Lastly, in contexts where relevant information may be distant or not immediately related, random attention could facilitate capturing broader and non-local relationships. To conclude, the architecture involves the use of global attention, i.e, tokens that attend to all tokens in the sequence and to whom all tokens attend to. Concretely, we choose a subset $G \subseteq \{1, \ldots, n\}$ such that we explore the relationship between x_i and x_j for $i \in G$ and all j. This latter element is added in order to connect indirectly any pair (x_i, x_j). In fact, the article is based on the mathematical result that every complete graph can be approximated by random graphs. In this scenario, these three elements aims at reproducing the original attention graph. As proven in Theorem 3 in [7], the complexity using sparse attention reduces to $O(n)$.

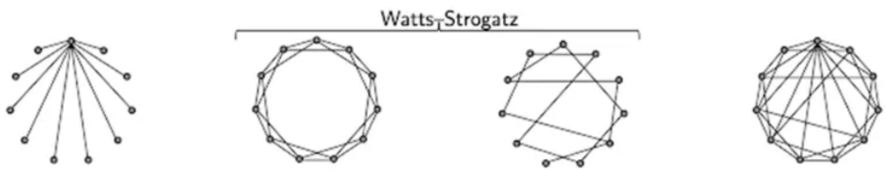
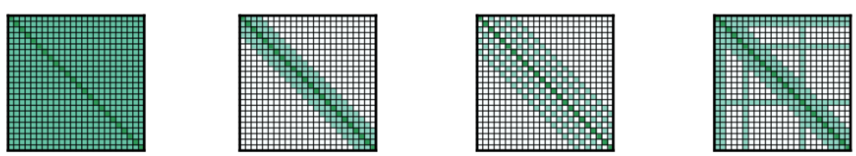


Figure 8: The effect of global, window and random attention in generating a sparse graph. Here, tokens represent nodes and the similarity scores calculated between tokens represent the edges. It is important to note that the number of edges is equal to the number of inner products that the attention mechanism will consider.

An alternative that shares similarities with BigBird is Longformer, presented in [8]. This architecture scales linearly with the input sequence, making it efficient for longer sequences. It is composed by three elements that combined form the sparse attention: sliding window, dilated sliding window and global attention.



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Figure 9: Visual explanation of the attention patterns in Longformer

First of all, given a fixed window size w, each token attends to $1/2w$ tokens on each side. Therefore, the computation complexity of this pattern is $O(n \times w)$, which scales linearly with input sequence length n. By using multiple stacked layers of such windowed attention, this results in a large receptive field that is able to capture information across the entire input. To further increase the receptive field without increasing computation, the sliding window can be "dilated" by gaps of size dilation d. In a transformer with L layers, assuming w, d fixed, the receptive field is $L \times d \times w$, which can reach tens of thousands of tokens even for small values of d. In paper [8] they underline that in multi-headed attention, each attention head computes a different attention score. Therefore settings with different

dilation configurations per head improves performance by allowing some heads without dilation to focus on local context, while others with dilation focus on longer context. In this way, we are "breaking down" the full attention in different experts that still aims at exploring the context at different scales.
Similarly to BigBird, we add global attention on few pre-selected input locations to allow enough flexibility to learn task-specific representations. Since the number of such tokens is small relative to and independent of n the complexity of the combined local and global attention is still $O(n)$. The attention coefficients are still computed as

$$Attention(Q, K, V) = Softmax\Big(\frac{QK^T}{\sqrt{d_k}}\Big)V,$$

but we use Q_s, K_s, V_s to compute attention scores of sliding window attention, while Q_g, K_g, V_g to compute attention scores for the global attention, that encodes what we described above. In paper [8] the authors emphasise the good performance of the model for long sequences, similarly to BigBird.

2.1.2 Linear attention

In paper [10] the authors propose a method that is able to scale linearly with respect to the dimension of the output. They introduce the so called *linear transformer* that is a reformulation of self-attention by using a kernel-based formulation. To follow their notation, let us denote

$$V' = Softmax\Big(\frac{QK^T}{\sqrt{d_k}}\Big)V.$$

and the $i-$th row of V' can be expressed as

$$V_i' = \frac{\sum_{j=1}^n e^{Q_i^T K_j} V_j}{\sum_{j=1}^n e^{Q_i^T K_j}}. \tag{12}$$

Equation (12) can be generalized to any similarity function as

$$V_i' = \frac{\sum_{j=1}^n \mathrm{sim}(Q_i, K_j) V_j}{\sum_{j=1}^n \mathrm{sim}(Q_i, K_j)} \tag{13}$$

where $\mathrm{sim}(Q_i, K_j)$ is a non-negative function that expresses how similar are two vectors. Before continuing it is essential to introduce the following definitions and results.

Definition 1. An Hilbert space H is a reproducing kernel Hilbert space (RKHS) if the evaluation functionals $F_x \colon H \to \mathbb{R}$ such that $F_x(f) = f(x)$ are continuous, i.e.

$$|F_x(f)| = |f(x)| \leq M||f|| \quad \forall f \in H$$

for a given $M \in \mathbb{R}$.

Invoking Riesz's representation theorem, every RKHS has a special function associated to it, namely the reproducing kernel:

Definition 2. A reproducing kernel is a function $K \colon X \times X \to \mathbb{R}$ such that

1. $K_x(\cdot) \in H$, $\forall x \in X$ and
2. $(f, K_x) = f(x)$, $\forall f \in H$ and $x \in X$.

The intuition for a reproducing kernel is a function that is able to reconstruct the elements of an Hilbert space. Given a measure μ,

$$\begin{aligned} f(x) = (f, K_x) &= \int_X f(x')K_x(x')\, d\mu(x') \\ &= \int_X f(x')K(x, x')\, d\mu(x'), \end{aligned}$$

which shows that K is a function that is able to determine the punctual value of f if it accesses to all the information about f in terms of similarity to the interested point. From the uniqueness in Riesz's representation theorem it is immediate to conclude the following

Theorem 2.1. *Every reproducing kernel K induces a unique RKHS, and every RKHS has a unique reproducing kernel.*

The interesting property of reproducing kernel is that they are a measure of similarity in a different (usually big) dimensional space. Indeed, in view of Mercer-Hilbert-Schmit theorem, it holds that a reproducing kernel can be expressed as

$$K(x, y) = \phi(x)^T\phi(y) \tag{14}$$

where $\phi\colon X \to F$ is a feature map with F an Hilbert space. Viceversa, every feature map defines a unique reproducing kernel according to (14).
Losing generality, we can assume sim to be a reproducing kernel, so that $\mathrm{sim}(a, b) = \phi(a)^T\phi(b)$. Therefore, substituting this in (13), we obtain

$$V_i' = \frac{\sum_{j=1}^n \phi(Q_i)^T\phi(K_j)V_j}{\sum_{j=1}^n \phi(Q_i)^T\phi(K_j)}. \tag{15}$$

In this way we have an attention computation consisting only on dot products. Using associativity in (15) we get

$$V_i' = \frac{\phi(Q_i)^T \sum_{j=1}^n \phi(K_j)V_j^T}{\phi(Q_i)^T \sum_{j=1}^n \phi(K_j)}.$$

Previous equation defines the *linear transformer* and it is crucial to underline that has a complexity of $O(n)$ because we can compute $\sum_{j=1}^n \phi(K_j)V_j^T$ and $\sum_{j=1}^n \phi(K_j)$ once and reuse them for every query and so n times. The same idea can be used for the masking such that the $i-$th position can only be influenced by a position j if and only if $j \leq i$, observing that we can express the attention as

$$V_i' = \frac{\sum_{j=1}^i \mathrm{sim}(Q_i, K_j)V_j}{\sum_{j=1}^i \mathrm{sim}(Q_i, K_j)}$$

and so

$$V_i' = \frac{\phi(Q_i)^T \sum_{j=1}^i \phi(K_j)V_j^T}{\phi(Q_i)^T \sum_{j=1}^i \phi(K_j)}.$$

An illustrative example is based on first-order approximation of Taylor expansion of exponential function:

$$e^{Q_i^T K_j} \approx 1 + Q_i^T K_j.$$

To ensure $1 + Q_i^T K_j \geq 0$ we can normalize Q_i and K_j using the l_2 norm so that $-1 \leq Q_i^T K_j \leq 1$. Equation (12) becomes

$$V_i' = \frac{\sum_{j=1}^{n} \left(1 + \left(\frac{Q_i}{||Q_i||_2}\right)^T \sum_{j=1}^{n} \left(\frac{K_j}{||K_j||_2}\right)\right) V_j}{\sum_{j=1}^{n} \left(1 + \left(\frac{Q_i}{||Q_i||_2}\right)^T \left(\frac{K_j}{||K_j||_2}\right)\right)}$$

and simplified as:

$$V_i' = \frac{\sum_{j=1}^{n} V_j + \left(\frac{Q_i}{||Q_i||_2}\right)^T \left(\frac{K_j}{||K_j||_2}\right) V_j}{n + \left(\frac{Q_i}{||Q_i||_2}\right)^T \sum_{j=1}^{n} \left(\frac{K_j}{||K_j||_2}\right)}$$

In paper [10] the authors show that their evaluation on image generation and automatic speech recognition demonstrates that linear transformer can reach the performance levels of transformer, while being up to three orders of magnitude faster during inference.

2.1.3 Matrix factorization

In matrix factorization we assume that the matrix corresponding to the self-attention values is low rank, i.e., not all items are independent of each other. Therefore, the matrix QK^T which is $n \times n$ is reduced to a matrix $n \times k'$ with $k' < n$ without significant loss in information. Since taking row-wise softmax of a square matrix of order n has complexity $o(n^2)$, reducing the matrix QK^T improves significantly the efficiency.
First of all in paper [13] the authors perform a singular values decomposition (SVD) on the matrix

$$P = Softmax\left(\frac{Q^{(h)}(K^{(h)})^T}{k}\right)$$

across different layers and different heads of the model. The results exhibit a clear long-tail spectrum distribution across each layer, head and task. This implies that most of the information of matrix P can be recovered from the first few largest singular values.

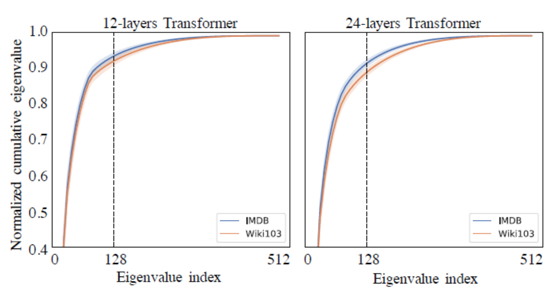


Figure 10: Spectrum analysis of the self-attention matrix for the two tasks performed

The experimental results are confirmed by the theoretical ones: the authors show that given the matrix P there exists a matrix $\tilde{P}$ with small approximation error, namely that

$$Pr(||P\omega^T - P\omega^T|| < \epsilon||P\omega^T||) > 1 - o(1) \tag{16}$$

for any ω values vector. In other words we are reading the approximation based on the action of that matrix on values vectors, i.e., the attention values. First of all, we can rewrite matrix P as

$$P = Softmax\underbrace{(\frac{Q^{(h)}(K^{(h)})^T}{k})}_{A} = exp(A) \cdot D_A^{-1}$$

where D_A is a diagonal matrix. Now $\tilde{P}$ is defined as $\tilde{P} = exp(A) \cdot D_A^{-1} R^T R$ where $R \in \mathbb{R}^{k' \times n}$ with i.i.d. entries. The proof that $\tilde{P}$ approximates in the sense of (16) P relies on JR lemma which states that

Lemma 2.1. *Given $0 < \varepsilon < 1$, a set X of m points in $\mathbb{R}^N$, and an integer $n > 8(\ln m)/\varepsilon^2$, there is a linear map $f : \mathbb{R}^N \to \mathbb{R}^n$ such that*

$$(1-\varepsilon)\|u-v\|^2 \leq \|f(u) - f(v)\|^2 \leq (1+\varepsilon)\|u-v\|^2$$

for all $u, v \in X$.

In simple terms, the lemma asserts that a set of points in a high-dimensional space can be embedded into a space of much lower dimension in such a way that distances between the points are nearly preserved.
So far we proved that matrix P can be approximated in low-rank terms. One straightforward ideas can be to use the SVD to approximate the attention weights, but as pointed out by the authors this will results in a much heavier architecture. As a consequence, invoking the JR lemma and recalling that applying a linear map can be thought as a matrix multiplication, they redefine attention values for a given head as

$$head_i = \underbrace{Softmax\Big(\frac{Q^{(i)}(E_i K^{(i)})^T}{k}\Big)}_{n \times k'} \underbrace{F_i V^{(i)}}_{k' \times d}. \tag{17}$$

In equation (17) we project the $(n \times k)$-dimensional key and values layers $K^{(i)}$, $V^{(i)}$ into $(k' \times k)$-dimensional projected key and value layers. Thus, if we can choose a very small projected dimension k', then we can significantly reduce the memory and space consumption.

3 Behind the scenes of Self-Attention

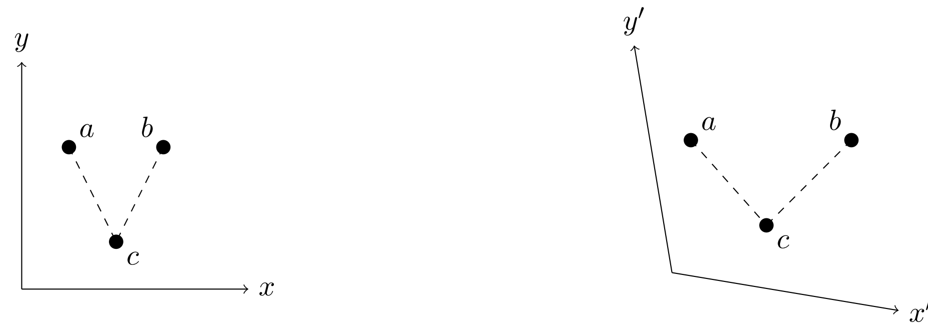
In this section we aim at exploring and unveiling the Self-Attention mechanism. First of all, the formula

$$Attention(Q, K, V) = Softmax\Big(\frac{QK^T}{\sqrt{k}}\Big)V$$

can be described in geometrical terms. For simplicity we omit the index referring to the head h. Matrices W_k and W_q are linear transformation that deforms the space. In multiplying QK^T we are performing,

$$W_q X \cdot (W_k X)^T$$

and since the dot product captures the similarity of two vectors, here we are computing how similar are the input vectors embedded in two appropriate linear spaces that are able to separate dissimilar words and cluster similar words. The softmax function is applied to normalise coefficients. Finally, the embedding that captures the similarity is transformed according to W_v. The motivation for that is, since our aim is to produce the next word in a sentence, the embedding defined by W_v suits this task since "knows" when two words appear in the same context because it is moving the space where words are either clustered or separated. To geometrically visualize the intuition, let us consider for the sake of simplicity two points $a, b \in \mathbb{R}^2$ and a third point c that is somehow in between a, b. We can think of c as a word that has multiple meaning and provided context it will shift towards either a or b.



When we provide context we are weighting each word according to all the other input words and so in our simple example, c will shift towards either a or b. It is clear that the second space works better in moving c according to the context. Finally, W_v is taking profit of this context aware embedding and capture long term relationship.

In its generality, given $q_1, \ldots, q_n \in \mathbb{R}^d$ query vectors, $k_1, \ldots, k_m \in \mathbb{R}^d$ key vectors and $v_1, \ldots, v_m \in \mathbb{R}^p$ value vectors, the attention mechanism computes a set of output vectors $o_1, \ldots, o_n \in \mathbb{R}^p$ by combining a transformation of these vectors. Formally,

$$o_j = \frac{1}{C} \sum_{i=1}^{m} f(q_j, k_i) g(v_i), \tag{18}$$

where $f(q_j, k_i)$ characterizes the relationship between queries and keys, while $g(\cdot)$ is a linear application such as $g(v_i) = W_v v_i \in \mathbb{R}^q$ where $W_v \in \mathbb{R}^{q \times p}$. The term $\sum_{i=1}^{m} f(q_j, k_i)$ is used as a normalization factor. In the case of transformers,

$$f(q_j, k_i) = \exp(\theta(q_j)\phi(k_i)^T)$$

where $\theta(\cdot), \phi(\cdot)$ are two linear transformation such as $\theta(q_j) = W_q q_j$ and $\phi(k_i) = W_k k_i$. The function f is generally referred to as *attention pooling* and it can be thought as a similarity measure. In general we can ask f to have various properties, the most common is that weights $f(q_j, k_i)$ form a convex combination leading to interpret them as probability coefficients. With this notation, the geometrical intuition can be expressed in analytical terms as it follows, assuming $f(x, y) = \langle x, y \rangle$,

$$o_j = \frac{1}{C}[\langle q_j, k_1 \rangle g(v_1) + \ldots + \langle q_j, k_m \rangle g(v_m)]$$

now if, for example, q_j attends mostly k_1, then

$$o_j \approx \frac{1}{C}[\langle q_j, k_1 \rangle g(v_1)] \approx g(v_1).$$

Therefore, we can think of attention scores as a "guide map" to attention values.

In paper [12] the authors delve into the multi-head attention mechanism and evaluate the contribution made by individual attention heads in the encoder to the overall performance of the model. They found that for a translation task, only a small number of heads are important for translation and these heads play interpretable "roles". They showed that most of the heads can be pruned without significant loss in quality, but after the training process. Inspired by their work, our aim is to explore at a deep level the self-attention mechanism.

3.1 Experiments on attention layer

In the experimental part we used the opus_books from HuggingFace from English to Italian, which included a large variety of texts, articles and web pages. To tokenize each sentence we used their tokenizers *WordLeverTrainer* and *Whitespace*. In order to understand at a basic level what is the attention layer doing, we decided to train the transformer for 20 epochs with just 1 layer and either 1 or 8 heads, to better understand the flow of information and prevent the depth of the neural network from altering the scores. The goal is to produce a correct translation of the sentence involved, so the attention scores reflect this task.
Figure 11 and 12 clearly show that there are few possible setting for the attention heads while performing translation, in according to paper [12]. Most of the heads show a diagonal structure in the matrix as when translating the token referencing to the word itself should be the most important to produce the translated token. Furthermore, it is possible to see that some heads show a stress on a specific token: this captures another aspect of translating. In producing language there are words that are giving the general meaning or context of the whole sentence or words that are referencing to other words such as pronouns. As a consequence, these layer with token attending a specific one are encoding this specific aspect. In Figure 12 head 2 and head 7 clearly exhibit this. Furthermore it is worthy to underline that all the matrix involved are full rank with low l_2 norm.

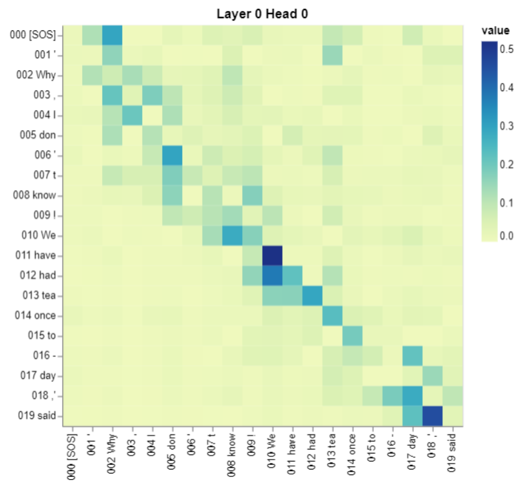


Figure 11: Attention visualization architecture with 1 layer and 1 head

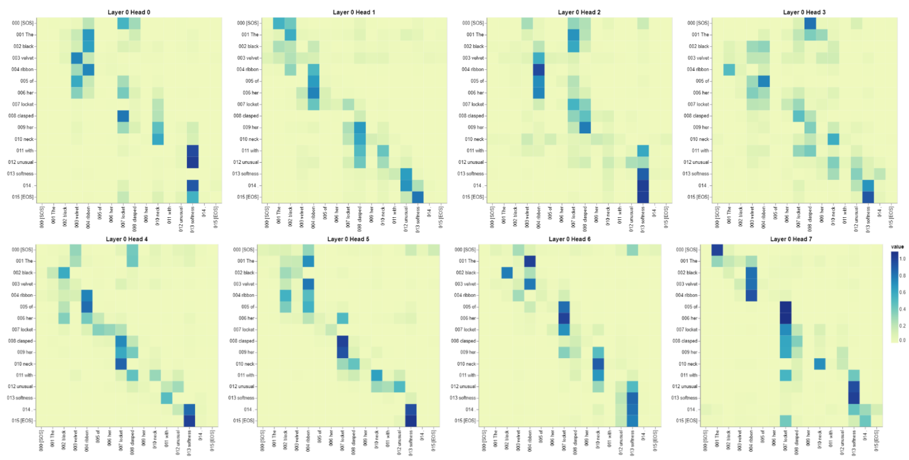


Figure 12: Attention visualization architecture with 1 layer and 8 heads

3.1.1 On the structure of attention scores

The experiments carried shows a specific structure in the attention pooling function for the transformer

$$f(Q, K) = Softmax\Big(\frac{QK^T}{\sqrt{k}}\Big).$$

Three general structured can be identified:

- positional heads;
- syntactic heads;
- rare tokens heads.

What is interesting is that the attention scores are never spread among the matrix but the values are mostly concentrated around the diagonal, up to a proper permutation of them.

Taking into consideration this and the roles of heads we propose the following formulation for the attention scores:

$$f(P, \tilde{E}) = (I_n P + \tilde{E}), \qquad P, \tilde{E} \in \mathbb{R}^{n \times n}, \quad P \in \Sigma \tag{19}$$

and therefore the new attention values

$$(I_n P + \tilde{E})V.$$

We write $I_n P$ instead of P to stress on the action of P in shifting the attention. The matrix $\tilde{E}$ is an sparse error matrix with elements $|\epsilon_{i,j}| \leq \epsilon$ so with very low values, while P is a permutation matrix with a specific structure we will discuss afterwards taken from a set of possible permutation Σ. Before describing the permutation we are dealing with and their properties let us clarify the intuition behind formula (19): the role of the permutation is to shift the positional heads, i.e. the diagonal, to the relevant structure that attention scores exhibit. In some sense, we are forcing query and keys to attend each other in a specific form such that encode the different roles of attention heads and so telling values where to look to get the context. After doing that we add an error matrix to introduce randomness and add elasticity to the formula.

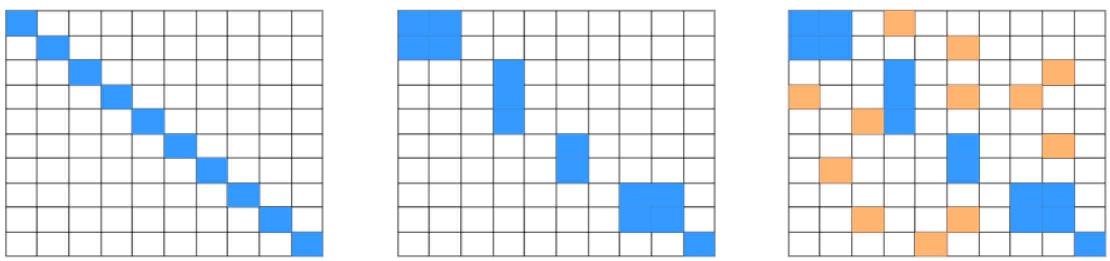


Figure 13: Effect of permutation and adding sparse error.

For example in Figure 13 we are forcing token $3, 4, 5$ to attend token 4.
The matrix P in (19) as a specific structure. We can define 3 family permutations that we are interested in: $\Sigma_1 = I_n$ to capture the positional head as we are dealing with mostly diagonal matrices, $\Sigma_2(w)$ for a windows size w which is made by appropriate matrices to capture syntactic heads, namely block matrices that correlate tokens to some other specific tokens as in Figure 12 for head $2, 4, 6$; and lastly Σ_3 representing permutation able to codify rare tokens and so that all token are attending a specific one as, almost, for head 7 in Figure 12. With this notation

$$\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3 \tag{20}$$

and the permutation P is taken among these possible sets. The windows size w is important to control the size of the blocks in Σ_2 and also it is consistent with the idea that syntactic relationships as verb-adjective are mainly close.

3.1.2 Description of permutation sets

In this section we are going to describe the sets that appears in (20). The set Σ_1 consists of I_n as we are shifting the attention to the token position, and so the resulting matrix should be diagonal. The set Σ_3 is related to rare token heads of the following form

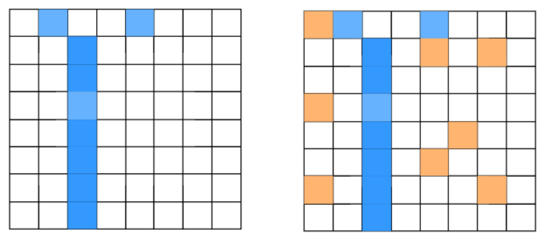


Figure 14: Exemplification of rare token heads.

These kind of permutation are characterized by two parameters: $\mathcal{T} = \{t_1, \dots, t_s\}$ the set of ordered tokens the attention focus on and their relative windows size $\mathcal{W} = \{w_1, \dots, w_s\}$ in the sense that token t_1 is attended by w_1 surrounding tokens and t_s by w_s surrounding tokens. It is sufficient to describe a single token with its window (t_1, w_1) as the general case would be just a concatenation of matrices. The base case, assuming t_1 in position i is represented by matrices of the form

$$\begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots \\ 0 & \cdots & \cdots & 1 & 0 \\ 0 & \cdots & \cdots & 0 & 1 \end{pmatrix}$$

where in the central part we have w vertical 1s in column i. If we call these kind of matrices $P_r \in \mathbb{R}^{w_1 \times w_1}$ of the form

$$\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & \cdots & 0 \end{pmatrix}$$

then the base case it is just a concatenation of I_{d_1}, P_r, I_{d_2} such that $d_1 + w + d_2 = n$ in the sense that we have the identity, then the rare token block and the identity.

Finally, the last set is Σ_2 in which we a structure close to a block diagonal matrix in which we have word, or group of words, attending other group of words. From the experiments it is possible to see that group of words attending other group of words are usually close, at least in English vocabulary, therefore an easy way to describe these matrices would be to start with a multi diagonal matrix and apply a "dropout" to obtain the desired structure.

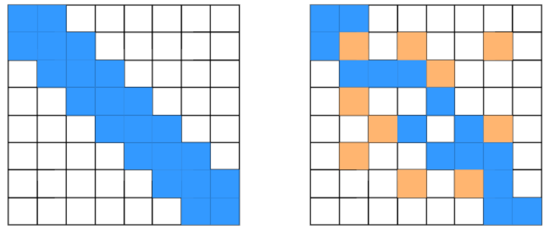


Figure 15: Exemplification of syntactic heads

The dropout can be applied also to create "separate groups" as shown in Figure 12 capturing that we have no relevant information as punctuation. In mathematical terms, given a matrix $B \in \mathbb{R}^{n\times n}$ such that

$$b_{i,j} = 0 \quad \text{if } i > j + w_1, \quad \text{or} \quad j > i + w_2; \quad w_1, w_2 \geq 0$$

where w_1, w_2 are the two bandwidths, then

$$P = \text{dropout}(B, p) \quad p \in (0, 1).$$

The three sets of matrices we defined can be thought, up to a proper permutation of rows as block-diagonal matrices with an error matrix. Indeed, for rare token matrices of the form it is sufficient to apply iteratively functions of the form

$$\sigma_i \colon (1, 0, \dots, 0) \to (0, \dots, 0, 1, 0, \dots, 0)$$

to make it diagonal. As a consequence, in mathematical terms, if we consider the space $\mathcal{X}$ defined as

$$\mathcal{X} = \{B + E_{rr} \text{ such that } B, E_{rr} \in \mathbb{R}^{n\times n}\}$$

where B is a band matrix of fixed bandwidth $w << n$ and E_{rr} is a sparse matrix such that

$$\max_{i,j} |e_{i,j}| \leq \epsilon, \quad \tilde{e}_{i,j} \in E_{rr}$$

we are projecting the attention weight heads on $\mathcal{X}$ and then adjusting its shape with a proper permutation.

Existence of solution: given the matrix H for head i we are aiming at finding $\tilde{X}$ such that

$$\tilde{X} = \underset{X\in\mathcal{X}}{\text{argmin}}\ |H - X| \tag{21}$$

where

$$|H - X| := \sum_{i,j} |h_{i,j} - x_{i,j}| \quad h_{i,j} \in H, \quad x_{i,j} \in X. \tag{22}$$

First of all we have to prove that $\mathcal{X}$ is a non-empty, closed and convex subset of $\mathcal{M}_n(\mathbb{R})$. The non emptiness is obvious, to prove that is convex it is sufficient to prove that

$$X_t = tX_1 + (1-t)X_2 \in \mathcal{X} \quad t \in (0, 1), \quad X_1, X_2 \in \mathcal{X}$$

but sum of band matrices is clearly a band matrices and sum of error matrices, according to our definition, is an error matrix, hence $X_t \in \mathcal{X}$. Finally, we are left to prove that $\mathcal{X}$ is close, i.e., given $X_n \in \mathcal{X}$ such that

$$X_n \to X \implies X \in \mathcal{X} \quad \text{for } n \to \infty.$$

From the definition of the norm in (22) we have that

$$\lim_{n\to\infty} \sum_{i,j} |x_{i,j}^{(n)} - x_{i,j}| = 0 \quad x_{i,j}^{(n)} \in X_n, \quad x_{i,j} \in X$$

and so, in particular,

$$\lim_{n\to\infty} \max_{i,j} |x_{i,j}^{(n)} - x_{i,j}| = 0$$

which implies the pointwise convergence

$$\lim_{n\to\infty} x_{i,j}^{(n)} = x_{i,j} \quad \forall i, j.$$

Now, if we split $X_n = B_n + E_{rr}^{(n)}$ and focusing on the band part, it is immediate to conclude from the point-wise convergence that

$$\lim_{n\to\infty} B_n$$

is a band matrix with bandwidth less or equal that w. Now, since we have pointwise convergence and the limit exists, it means that the limit on the error sparse matrix

$$\lim_{n\to\infty} E_{rr}^{(n)}$$

can be either a value less or equal than ϵ or 0. But since for any n the matrix is sparse, by definition of limit, the limit

$$\lim_{n\to\infty} e_{i,j}^{(n)}, \qquad e_{i,j}^{(n)} \in E_{rr}^{(n)}$$

is mostly 0. This conclude that $X \in \mathcal{X}$. From the fact that $\mathcal{M}_n(\mathbb{R}) \cong \mathbb{R}^{n\times n}$ and $(\mathbb{R}, ||\cdot||)$ is a Banach space, we conclude that $\mathcal{M}_n(\mathbb{R})$ with the norm

$$|M| = \sum_{i,j} |m_{i,j}| \quad m_{i,j} \in M, \quad M \in \mathcal{M}_n(\mathbb{R}),$$

is a Banach space. As a consequence we can conclude the existence and uniqueness of the solution to (21) recalling the following well-known result.

Theorem 3.1. *Let $\mathcal{H}$ be an Hilbert space with respect the norm $||\cdot||$. If C is a convex, closed, non-empty subset of $\mathcal{H}$ and q a point of $\mathcal{H}$. Then, there exists a unique $p \in C$ such that*

$$||p - q|| = dist(C, q) := \inf_{x\in C} ||x - q||.$$

References

[1] A. Vaswani et al., *Attention Is All You Need*, NIPS, 2017

[2] D. Bahdanau, K. Cho, Y. Bengio, *Neural Machine Translation by Jointly Learning to Align and Translate*, arXiv, 2014

[3] M.T. Luong, H. Pham, C. D. Manning, *Effective Approaches to Attention-based Neural Machine Translation*, arXiv, 2015

[4] R.S. Navid, *Introduction to Transformers*, Institute of Computational Perception CP Lectures, 2020

[5] A. Kak, C Bouman, *Transformers: Learning with Purely Attention Based Networks*, Lectures on Deep Learning Purdue University, 2023

[6] J. Devlin, M. W. Chang, K. Lee, K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, arXiv preprint, 2018

[7] M. Zaheer et al., *Big Bird: Transformers for Longer Sequences*, NIPS, 2020

[8] I. Beltagy, M. E. Peters, A. Cohan, *Longformer: The Long-Document Transformer*, arXiv, 2020

[9] I. Chalkidi et al., *An Exploration of Hierarchical Attention Transformers for Efficient Long Document Classification*, arXiv, 2022

[10] A. Katharopoulos et al., *Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention*, arXiv, 2020

[11] R. Li et al., *Linear Attention Mechanism: An Efficient Attention for Semantic Segmentation*, arXiv, 2020

[12] E. Voita et al., *Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned*, Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, 2019

[13] S. Wang et al., *Linformer: Self-Attention with Linear Complexity*, arXiv, 2020

[14] Y. Tay et al., *Long range arena: a benchmark for efficient transformers*, arXiv, 2020

[15] A. Zhang et al., Dive into Deep Learning, 2023

[16] S. Ji, Y. Xie, H. Gao, A Mathematical View of Attention Models in Deep Learning, Lectures at Texas A&M University, 2019

[17] D. Foschi, Appunti di Analisi 3, Lecture notes at Università degli studi di Ferrara, 2020