,

Statistical analysis on the hidden embedding of Flaubert's transformers

FOSSE Loïc2 GRAVIER Guillaume 1 NGUYEN Duc-Hau1
(1)IRISA, 263 Av. Général Leclerc, 35000 Rennes, France
(2) INSA Rennes , 20 Av. des Buttes de Coesmes, 35700 Rennes, France
guig@irisa.fr , loic.fosse@insa-rennes.fr , duc-hau.nguyen@irisa.fr

RÉSUMÉ

Les transformers sont des architectures neuronnales nouvelles, qui permettent de modifier les embeddings des mots en prenant en compte le contexte de la phrase, grâce (et seulement) au mécanisme d'attention. Cette prise en compte du contexte de la phrase nous a amené à nous poser divers questions et faire des analyses statistiques sur les embeddings des mots aux différents étages des transformers, afin de voir comment cette prise en compte du contexte agissait sur les embeddings. N'ayant trouvé aucune ressource sur une telle analyse, nous allons présenter dans ce rapport les différents résultats trouvés.

ABSTRACT

Transformers are new neural architectures that allow us to modify word embeddings by taking into account the context of the sentence, thanks to (and only thanks to) the attention mechanism. Taking into account the context of the sentence led us to ask ourselves various questions and to make statistical analyses on the embeddings of the words at the different stages of the transformers, in order to see how this taking into account of the context acts on the embeddings. As we did not find any resource on such an analysis, we will present in this report the different results found.

MOTS-CLÉS : Plongement lexical , Transformers , Flaubert , Cosinus , Apprentissage machine , statistique , vraisemblance .

KEYWORDS: Word Embedding , Transformers , Bert , Cosine , Machine Learning , statistics , likelihood.

Acknowledgements :

I wanted to start this report with the traditional thanks.

First of all I would like to thank Mr Gravier who accompanied me during all this project of initiation to research. His experience and his expertise in this field helped me enormously. He knew how to guide me in a world that I was discovering and he always knew how to find the right tracks of work. The numerous discussions we had allowed me to learn a lot about the objects I was handling and to understand them better.

Moreover, thanks to Mr Gravier I have the chance today to write a paper (that you are reading), which will be submitted to the conference **TALN** (Traitement Automatique des Langues Naturelles), in order to present all the results we found.

I also thank Mr Duc-Hau Nguyen, who is the PhD student with whom I had the chance to realize this project and who gave me a lot of time. He taught me a lot in the field of computer science and introduced me to many tools that will be very valuable for the rest of my career.

More generally, I would like to thank the whole team of IRISA, which has always welcomed me very well and which knew how to make a small place for me to work in the best possible conditions.

Table of content

1 Introduction :

Representing words as mathematical data, in order to translate them or to detect feelings, and all this with a simple computer. You are not reading the summary of the next terminator but the one of my research initiation project.

Alongside Mr Gravier and Mr Nguyen, we worked on neural architectures capable of performing such tasks : the transformers. These new architectures, coming straight out of Google's laboratories, have been much talked about lately. Through this report we propose a statistical analysis of the so-called hidden states of these architectures, in order to understand their functioning.

We are going to make here a presentation which essentially resumes the big steps that we followed during this project:

- definition of the mathematical models
- computer implementation
- research of results

This project was also for me the occasion to manipulate new computer tools which will be very useful for the continuation of my course, such as **Git**.

So I propose you to find all the codes that we could realize in the frame of this project in the following directory :

$GitPIR$

Enjoy your reading!

2 General Notion of Word Embedding

First of all we have to describe the notion of word Embedding.

Word Embedding is a general notion, which describes a way to represent words (textual data), using numerical data (vector).

This representation is necessary in the world of automatic language processing. Indeed, in this domain, we want to manipulate textual data and use automatic learning algorithms on them. All this is possible with numerical data, but not with textual data.

A similar example is image analysis. To analyze an image we represent this image as a matrix. This matrix contains numerical data relevant to our study (color shades, shades of gray, ...), which will allow us to manipulate this image with well chosen algorithms.

For the automatic language processing, we have to do the same, and find a way to represent our words as numerical data.

This representation will have to be coherent in the sense that the words whose meaning is close will have to be close, with the notion of proximity which will also have to be defined according to the space in which we represent the words.

Many methods to represent words in vectors have been developed. Here are some of them (the most known / used).

In this field it is often difficult to differentiate textual data from mathematical data that represent words. We thus introduce the following notations:

- a word in a sentence will be called a "token", it will be represented by a letter ("t").
- the representation in the form of embedding, will be noted w_t.

2.1 One hot

This vector representation of words is undoubtedly the easiest to understand. However it is also the least interesting. To build it, we have a large text with a set of words. We note V the number of distinct words in this text. To represent the words we will simply do a binary encoding. For each distinct word in the text, we decide to associate a binary vector of dimension V, with just one non-zero component, so that each word has a different non-zero component.

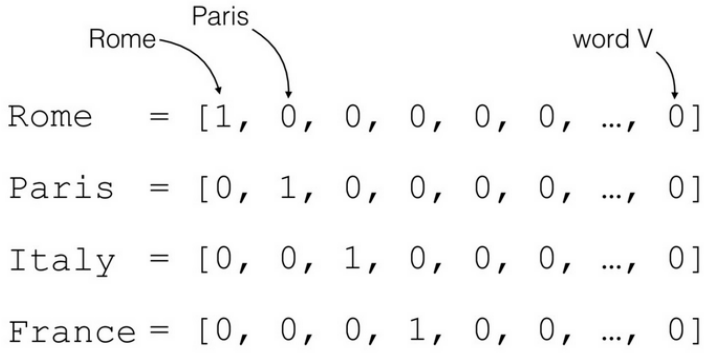


Figure 1: One hot encoding

This representation is thus very simple to understand and to use in practice, however it does not allow to distinguish if two words have a close meaning.

Moreover, if we perform a scalar product between two distinct words with this representation, we have :

$$< w_i, w_j >= 0 \quad \forall i \neq j$$

Thus we need to find a representation that allows us to identify words that are close in meaning. We need to find a representation of words that allows us to identify words that are close in terms of meaning from a simple scalar product. It is in this philosophy that the algorithms called word2Vec were developed.

2.2 Word2Vec

Word2Vec is a general term to describe algorithms that perform word embedding. These algorithms can be distinguished into two parts:

- the models with an architecture of type CBOW (continuous bag of words).
- the models with a skip-gram type architecture.

We will describe these two architectures, to better understand how they work.

2.2.1 CBOW

The general principle of this architecture is to guess a word from its context. Thus we will have a token t, called central, and we will define a neighborhood of this token. The neighborhood will be a set of tokens that are located before and after the central token.

Note: when we talk about the neighborhood of t, the token t is not included in this neighborhood.

To build embedding from a CBOW architecture, we need a text with a certain number of tokens. We note T, the number of token **unique** in this text.

We thus carry out a coding of type One hot, on this text. Thus for each token v, corresponds a vector oh_v, which is a binary vector of dimension T, of which only one component is nonzero, and we note v_k, the position of this nonzero component.

For each of these tokens v, we have corresponding embedding noted w_v which are vectors of dimension d, which are initialized randomly at the beginning. We constitute with these embedding the embedding matrix $W \in \mathbb{R}^{T \times d}$. To obtain the embedding of a given word, we just have to perform the following operation:

$$w_v = W^t \times oh_v$$

We then have the projection matrix noted $W' \in \mathbb{R}^{d \times T}$.

We thus carry out the following operations:

- step 1 : for all tokens v, which are in the neighborhood of t, we compute $w_v = W^t \times oh_v$
- step 2: we realize an empirical average on the whole of these vectors $h = \sum_{v \in V} \frac{w_v}{|V|}$, with V which indicates the neighborhood of the token v
- step 3 : $U = W'^t \times h$ projection of the vector h
- step 4 : $p(U) = (\frac{exp(U_k)}{\sum_{i=1}^{T} exp(U_i)})_{k \in \{1,...,T\}}$

This last vector $p(U)$, constitutes a probability measure on our unique token set. With the notation defined above, we have :

$$\boxed{p(U)_{t_k} = \frac{exp(U_{t_k})}{\sum_{i=1}^{T} exp(U_i)}}$$

This quantity can be interpreted as the probability that the token t, matches the token t that is in our text. The goal here is to make sure that for any t, each of these quantities is as close to 1 as possible.

This is achieved by modifying the matrices by W and W' by gradient back propagation.

2.2.2 Skip-gram

The skip-gram architecture is the exactly symmetric architecture of CBOW. Our goal this time is to find the neighborhood of a token t, knowing the embedding of this token t.

We thus have a text with a certain number of tokens that compose it. We have a number of unique tokens in this text. Each unique word can be represented with a One hot encoding, which we have presented previously.

We have for each of these token k a corresponding embedding $w_k \in \mathbb{R}^d$. Each of these embedding can be arranged in the embedding matrix $W \in \mathbb{R}^{T \times d}$, where T denotes the number of unique tokens in our text.

Thus to know the embedding of a word it is enough to carry out the following operation:

$$w_k = W^t \times oh_k$$

With $oh_k \in \{0,1\}^T$, which corresponds to the one hot encoding of the token k.

As for the CBOW architecture, we consider a number c, which will define the surrounding of a token t, that we will have chosen.

Indeed the surrounding of our word will be constituted of c tokens before and after our token t.

We introduce the context matrix noted $W' \in \mathbb{R}^{d \times T}$.

The operation is as follows:

- step 1 : $w_t = W^t \times oh_t \quad \in \mathbb{R}^d$
- step 2: $U = W'^t \times w_t \quad \in \mathbb{R}^T$
- step 3: $p(U) = (\frac{exp(U_k)}{\sum_{i=1}^T exp(U_i)})_{k \in \{1,...,T\}}$

$p(U)$ is a probability measure, associated to our one hot coding done on our original text.

Thus for each token v, in the neighborhood of our central token t, we note k_v, the index of its non-zero component in its one hot representation.

We thus interpret $p(U)_{k_v}$, as the probability that the token k, is indeed the token which is in the neighborhood of t, and in the right place.

With this model, we can define the log likelihood of our model:

$$\boxed{l(t) = log(\prod_{v \in \text{poximity}(t)} \frac{exp(U_{v_k})}{\sum_{i=1}^T exp(U_i)})}$$

We thus modify the matrices W and W', wanting to maximize the true likelihood of our model, using a back propagation algorithm.

Finally the embedding of the words of our text, will be able to be found in the matrix W.

2.2.3 The Word2Vec space

With the Word2Vec algorithms, we succeeded in obtaining continuous vector representations of our tokens. These representations have an interesting property which is the similarity of the cosines.

If we consider two tokens i and j, whose meaning is similar (e.g.: i = red, j = green), then we have :

$$\boxed{\frac{< w_i, w_j >}{||w_i||||w_j||} \approx 1}$$

Words that are close in meaning will be close in angle. We talk about cosine similarity measure. We will define it in more detail later.

3 The transformers

One our token are transformed into vectors (also call word embedding),in a well chosen space, we can manipulate them with the help of machine learning algorithms in order to transform these embedding and learn information about them (meaning of words, context of sentences, feelings expressed in the sentence, ...).

The most used are the LSTM (long short term memory) networks, and to a lesser extent today the recurrent networks or RNN (recurrent neural network).

In 2017, the Google Brain team published an article [1] that changed the world of NLP by introducing a new type of network, the transformers. This new architecture is original because it will essentially use the **attention mechanism**, which was until then only an auxiliary tool to improve to a lesser extent the network performances mentioned above.

The model presented in this article presents a structure in the form of encoder / decoder. It can be simplified in the form of the following diagram:

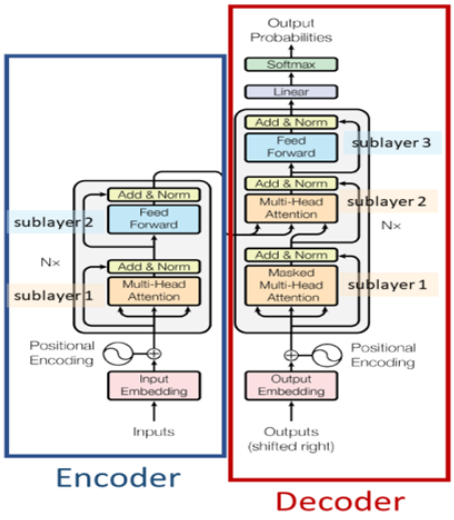


Figure 2: attention is all you need : architecture

This scheme seems at first sight very complex to understand. First of all, the transformers architecture only uses the encoder part. The decoder part will not be used here in this study. It is thus on the encoder part that we will concentrate in the following sections.

3.1 General principle of the transformers

Transformers are networks that are organized in layers (in the original version, we have 7 layers). First we have a first layer of embedding called non-contextual, using a first neural architecture like Word2Vec, in order to go into the embedding space. Then we have layers of "transformers" each of these layers is divided into 2 sub-layers with :

- a layer of multi-headed attention (which we will explain later)
- a feedforward layer

The general principle of transformers is to never transform the dimension of the embedding that we manipulate. We only transform the embedding (hence the name of transformers) of the words, in order to obtain, as we go along in the layers, embedding.

3.2 The positional Encoding

The first thing that appear on the schema, is the positional encoding.

The position of the embedding in the sentence, is an important information, which must be kept. This is why we transform our non-contextual embedding for the first time, so that it carries this information.

For this we create the **position function** ($PE(pos)$). This function takes into account the argument, pos which is the position of the considered token, within the sentence.

It returns a vector of dimension d, defined as follows:

$$\forall k \in \{0, \dots, (d-1)\} \;\; PE(pos)_k = \begin{cases} \sin \frac{pos}{10000^{\frac{2i}{d}}} & \text{where } \; k = 2i \\ \cos \frac{pos}{10000^{\frac{2i}{d}}} & \text{where } \; k = 2i+1 \end{cases}$$

note: we start indexing this position vector from 0. This is a technicality for the code.

Once we have realized this position vector, we will be able to use this information. To do this we carry out the following procedure.

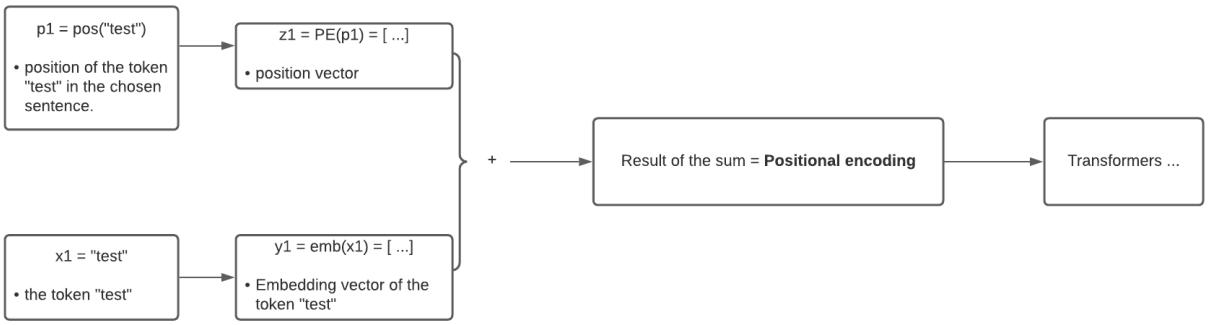


Figure 3: Keep the positional information

However, we may have a problem with this first method. Indeed it happens that the components of the embedding vector for a given token are very weak, and are thus not significant compared to the positional vector calculated from the sines and cosines.

When this happens when we sum the two vectors, we will overwrite the embedding value and we will lose a crucial information, namely the embedding of the token itself. To avoid this we make the following modification in the schema:

$$y_1 = emb(x_1) \times \sqrt{d}$$

The dimension of the model being very large, we will be able to increase the value of the components of our vector.

Now that we have described what happens upstream of the transformers architecture, let's go back to the heart of the architecture, with the attention mechanism.

3.3 The attention mechanism

First, we need to define the notion of attention. The attention between two embedding w_i and w_j in a given sentence, is a quantity noted $a_{i,j}$, which quantifies the link between these two embedding. If this quantity is large, then the two embedding have a strong link in the sentence (same meaning, same function in the sentence, ...). We also talk about attention weights to describe these quantities $a_{i,j}$. Attention is a mechanism that allows us to identify the key points, the important words in a sentence or in an image.

The attention mechanism is a mechanism that uses these famous attention weights. It is the heart of the architecture of transformers, through the multi-head attention layer.

In order to understand this so-called multi-head mechanism, we will go through the explanation of the single-head mechanism, which is the classical attention mechanism.

3.3.1 Single head mechanism

The general principle of the single-headed attention mechanism is quite simple. It consists in computing the attention weights between all the words in a given sentence. In this section, we will present the general formulas that are currently implemented in the Bert/Flaubert transformers.

Before starting the computation of our attention weights, we have our embedding matrix $X \in \mathbb{R}^{T \times d}$ where we recall that T is the number of tokens in the sentence and d is the embedding dimension. This matrix contains for a given sentence, the set of embedding of the different tokens. The embedding are ordered by row in this matrix.

In addition to these matrices, we have three new matrices, which are matrices whose values will be modified during the training:

$Q_\omega \in \mathbb{R}^{d \times T}$, the query matrix.

$K_\omega \in \mathbb{R}^{d \times T}$, the key matrix.

$V_\omega \in \mathbb{R}^{d \times T}$, the value matrix.

With these matrices, we will form the following matrix products:

$$Q = X \times Q_\omega \in \mathbb{R}^{T \times T}$$

$$K = X \times K_\omega \in \mathbb{R}^{T \times T}$$

$$V = X \times V_\omega \in \mathbb{R}^{T \times T}$$

We also introduce the function $softmax$, defined as follows:

$$\begin{array}{rccc} softmax & : & \mathbb{R}^T & \to & \mathbb{R}^T \\ & & (x_i)_{i \in \{1,\dots,T\}} & \mapsto & (\frac{e^{x_i}}{\sum_{k=1}^{T} e^{x_i}})_{i \in \{1,\dots,T\}} \end{array}$$

This function allows to obtain a discrete measure of probability from a vector.

Once these matrices are computed, the google brain team proposes a formula to compute the attention weights, which is the following formula:

$$\boxed{Attention(Q, K, V) = softmax(\frac{QK^t}{\sqrt{d}})V}$$

With here $softmax(\frac{QK^t}{\sqrt{d}})$, which is the application of the softmax function, to each row of the matrix $\frac{QK^t}{\sqrt{d}}$

By studying the dimensions, we obtain :

$$A = Attention(Q, K, V) \in \mathbb{R}^{T \times T}$$

More particularly, the vector $A(i, .)_{i \in \{1\dots,T\}} \in \mathbb{R}^T$ represents the attention weights of the token i, with respect to the rest of the sentence.

If we look at the set of formulas that are used here, we notice that the attention weights are relatively easy to obtain.

3.3.2 Multi-head mechanism

The multi-head attention mechanism, is a mechanism, which is essentially based on the single-head mechanism, but adding a subtlety.

As a reminder, the embedding we manipulate are d-dimensional embedding, with d often a very large number. The objective is to divide the X matrix of embedding into 12 blocks, as follows:

We will divide each embedding in 12 blocks. The block 1 corresponding to the first $\frac{d}{12}$ dimensions and so on. We finally have 12 embedding matrices, noted :

$$\{X_k\}_{k \in \{1,\dots,12\}} \in \mathbb{R}^{T \times \frac{d}{12}}$$

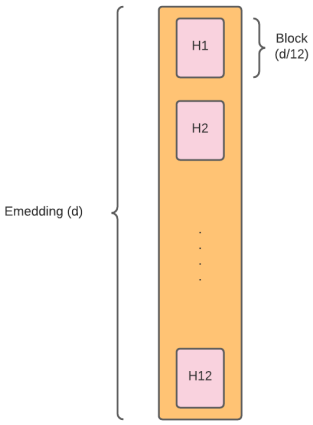


Figure 4: Block decomposition of each embedding

On each of these matrices, we will calculate the attention weights in the same way as before. We just need to increase the number of matrices. We have now for each block $k \in \{1, \ldots, 12\}$:

$Q_{\omega,k} \in \mathbb{R}^{\frac{d}{12} \times T}$, the query matrix.

$K_{\omega,k} \in \mathbb{R}^{\frac{d}{12} \times T}$, the key matrix.

$V_{\omega,k} \in \mathbb{R}^{\frac{d}{12} \times T}$, the value matrix.

Then in the same way, we will have new matrices:

$$Q_k = X_k \times Q_{\omega,k} \in \mathbb{R}^{T \times T}$$
$$K_k = X_k \times K_{\omega,k} \in \mathbb{R}^{T \times T}$$
$$V_k = X_k \times V_{\omega,k} \in \mathbb{R}^{T \times T}$$

note: this also explains why in the different implementations of Bert transformers, the dimension of the model is a multiple of 12 (512, 768, ...).

Finally we compute the attention points relative to the k blocks, always using the same formula:

$$A_k = softmax(\frac{Q_k K_k^t}{\sqrt{\frac{d}{12}}})V_k \in \mathbb{R}^{T \times T}$$

This new matrix A_k constitutes **an attention head**. Following this, we will perform a concatenation of the different heads of attention:

$$A = concat(A_1, \ldots, A_{12}) \in \mathbb{R}^{T \times 12T}$$

We now obtain a new matrix, but whose dimension does not correspond with the starting dimension of the X matrix of the embedding data. To come back to the dimensions of this matrix X, the google brain team, introduced a matrix $W^o \in \mathbb{R}^{12T \times d}$

And so the matrix $\tilde{X} = A \times W^o$, and the new matrix of embeddings, obtained thanks to the mechanism of multi-head attention.

note: The matrices indexed by w, are in fact the weight matrices, and constitute the parameters of our model. We can thus see that we have a very important number of parameters, even if it is only on the multi-head attention layer. Moreover we repeat this layer at least 6 times in the complete transformers. And on each layer the matrices are different. In a transformers architecture we have a considerable number of parameters that must be learned. This is the main problem of transformers today, their number of parameters is considerable, to update them it is necessary to use an important quantity of data and the calculation times are very important. For example, it is not possible to train a transformers architecture on a classical computer.

Finally the architecture of the bert transformers can be summarized in the following scheme, which is the scheme proposed by the team *Google Brain*.

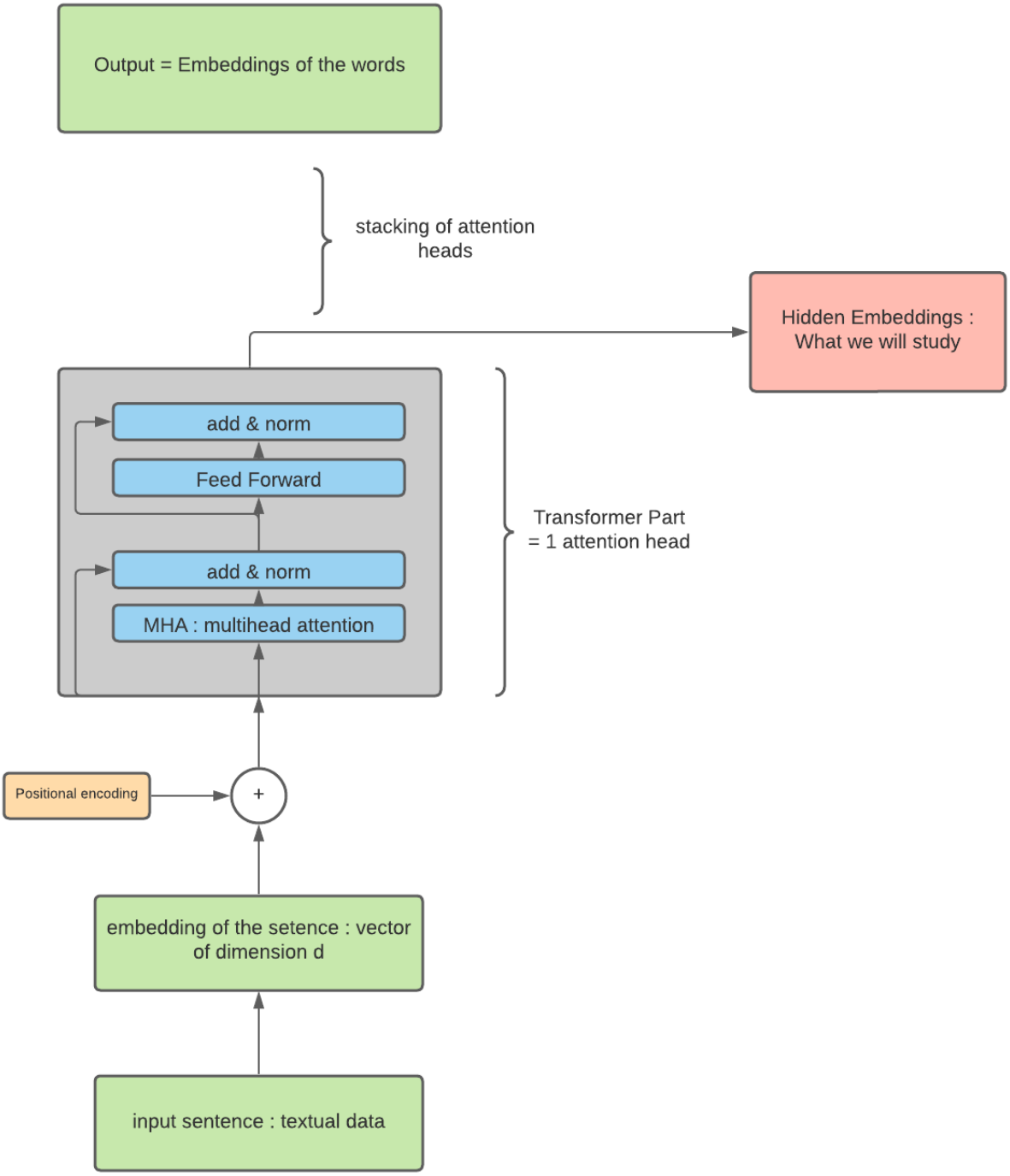


Figure 5: transformers architecture

3.4 The bert model

To understand how the formulas presented above work, we need to understand how the parameters of a transformer are learnt.

3.4.1 General principle for the training part

To train a transformers architecture, we need a considerable amount of textual data.

A simple way to have access to a large amount of data is to consider a set of web pages, making sure of course that all the text that will be used for training is in the same language.

Indeed a transformer will only be able to process textual data that are in the same language as its learning. This is why today we distinguish several transformers available on the market:

- Bert : trained with English data
- FlauBert , CamenBert : trained with French data.

We will thus consider a corpus, with a considerable number of words. In this corpus we are going to choose randomly some words that we are going to replace by the token "MASK". We apply in fact a mask in this text by hiding some words. We then make pass each sentence in the text, we have for each token "MASK", embedding which correspond.

$$t = "MASK" \quad \leftrightarrow \quad last\_embedding(t) = \begin{pmatrix} 1.7657 \\ \\ \vdots \\ \\ -8,675 \end{pmatrix}$$

Then, in the manner of the word2vec architecture, we will apply a log-linear classifier on these embedding, in order to estimate the probability that we have the right word (estimate the probability of making a good decision). We will actually try to predict the word that is located in the sentence in place of the token "MASK".. This probability must be close to 1. We try to maximize this probability for all the words in our corpus. This maximization is done by back propagation of the gradient.

3.5 Fine tuning

The fine tuning operation is a very important operation. So far we have described an architecture that allows to transform word embedding.

We would like to be able to perform a set of complex tasks from these vector representations, such as :

- the sentiment analysis
- word labeling
- search for a logical sequence of sentences
- classification of documents

These tasks are achievable from the previous architecture. We simply add layers to the output of our transformers, which will allow us to carry out these different tasks.

There are many layers that can be added to the output of the transformers. Usually it is one (or more) matrices and an activation function.

Here is an example of a way to do fine tuning on the Bert transformer.

For example, if we want to do sentiment analysis of a sentence, and we want to classify these sentences into two categories:

- 1 : the sentence is happy

- 0 : the sentence is sad

We consider $W^B \in \mathbb{R}^{T \times d}$, the matrix of embedding of the considered sentence obtained at the end of the Bert transformers. In this matrix, we consider only the first row. This row represents the token that we call $CLS = W^B(1, .)$. We add the matrix $C \in \mathbb{R}^{2 \times d}$, and we do the following operation:

$$p = softmax(C \times CLS)$$

With :

- $p(1)$: the probability that the sentence is happy
- $p(2)$: the probability that the sentence is sad

Thus if we have a set of n sentences for which we know the feelings, we note y_i, this binary quantity for the sentence i, we can thus define the log likelihood of our model:

$$l_n = \sum_{i=1}^{n} y_i \log p(1) + (1 - y_i) \log p(2)$$

The objective is to maximize this log-likelihood, and this operation is done by back propagation of the gradient. During this back propagation step, we update not only the parameters of the matrix C, but also all the parameters of our transformers

To have more details about how the transformer architecture works and how to train them, the author in [4] explain all of this.

3.6 Presentation of the subject

As we have presented, the transformers architecture is an architecture that is formed by stacking layers of MHA type, one above the other. Each of these layers is designed to modify the embedding of words in a sentence or a given text by taking into account the context of the sentence. After that we have a so-called contextual representation of the sentence. Then we repeat the operation, as many times as there are MHA layers in the transformers.

Our objective is to recover at each level of the Flaubert transformers the so-called contextual embedding (the hidden states), and to look at their behavior as we go up in the layers. We also talk about the rise in abstraction.

We have a starting hypothesis. Indeed, as at each level we start from a contextual representation, which we modify once again by using the context of the sentence, we assume that the set of word embedding converges towards a representation that represents the context of the sentence.

4 Metrics

As a reminder, our main objective is to retrieve the embedding obtained after each layer of the Bert transformers, and to characterize the proximity between these different embedding. Indeed, our basic hypothesis (or null hypothesis) is that all the embedding converge towards embedding that finally represent the context of the sentence.

This is the notion of proximity that we will define in this section.

4.1 The probabilistic framework

At first we can have the idea that all embedding converge to a single embedding noted w_0. Thus the context of the sentence would be represented by a point in the embedding space. To realize this, we can consider once again after each layer of our transformers, the embedding matrix $X \in \mathbb{R}^{T\times d}$, and compute :

$$V = var(X) \in \mathbb{R}^{d\times d}$$

The variance covariance matrix of our embedding matrix. To then quantitatively estimate the dispersion of our vectors in the embedding space, we will look at the quantity :

$$D = trace(V)$$

4.2 The cosine metric

As we have seen in the general presentation of the notion of embedding, many embedding will be built in such a way that words with similar meanings will see their embedding close in terms of angle. We thus introduce the cosine metric. We consider a sentence with T tokens inside. Let w_i and w_j be the embedding of two tokens ($i \neq j \in \{1, \dots, T\}$).

We have:

$$sim(w_i, w_j) = \frac{< w_i, w_j >}{||w_i||||w_j||}$$

Where $< ., . >$ refers to the standard dot product between vectors and $||.||$ the euclidean norm. The results of this metric can be interpreted as follows.

$sim(w_i, w_j) = 1$ the words are close in meaning

$sim(w_i, w_j) = 0$ the words are independent (no relationship)

$sim(w_i, w_j) = -1$ the words are opposite in meaning.

4.3 Metrics for the Embedding space

With the two previous metrics, we have characterized how words behave within the same sentence. It would be interesting to understand in a general way the space in which embedding live on each layer. It is in this goal that the authors of [2] and [3], have developed metrics, allowing to realize this task.

We are going to present you here these different metrics.

4.3.1 Erank

This metric is particularly used in computer vision and signal processing. It is expressed as follows:

$$\boxed{erank(W) = exp(-\sum_{i=1}^{d}(\frac{s_i}{\sum_{j=1}^{d} s_j}).log(\frac{s_i}{\sum_{j=1}^{d} s_j})}$$

With :

- W which is the matrix of embedding on a layer of the Bert transformers, it is a matrix which contains the embedding of a number N of words ($N >> d$). Thus with the convention of arranging our embedding in a row, we have $W \in \mathbb{R}^{N \times d}$
- $\{s_i\}_{i \in \{1,...,d\}}$ constitutes the singular values of the embedding matrix W.

This metric will inform us about the number of useful dimensions in our embedding space.

We have :

$\forall M \in \mathbb{R}^{N \times d}$ with $N > d$ we have : $erank(M) \in [1, D]$.

We must now be able to interpret the result of this metric. Thanks to [2] and [3], we have the following results:

- $erank(W)$ is close to 1, then we make a very poor use of our dimension set. Thus the matrix can be compressed into a matrix where the dimension of the vectors is 1. ($W \in \mathbb{R}^{N \times 1}$)
- $erank(W)$ close to d, indicates the opposite.

As it is mentioned in [2] and [3] , generally speaking, the $erank$ matrix will estimate the number of minimum dimensions that are useful in our embedding space. However it happens that this minimum dimension number is overestimated, in many applications.

4.3.2 Edim

To solve this problem, in [2] and [3], the authors propose the metric $Edim$. The edim (empirical dimension) metric will allow us to solve this overestimation problem. It is defined as follows:

$$\boxed{Edim(W) = \frac{||s||_p}{||s||_{\frac{p}{1-p}}}}$$

With $||x|_p = (\sum_{i=1}^{d}(|x_i|)^p)^{\frac{1}{p}}$.

5 Statistical analysis on Flaubert without fine-tuning

Now that we have set the context, and explained how the Flaubert transformers work, we are going to analyze the distribution of embedding as we go up in abstraction thanks to the metrics we presented in the previous section.

In this section, we will focus on the so-called raw Flaubert transform (without fine tuning).

We'll use the transformers from the Huggin face library. His characteristic are the following :

- embedding dimension : 768
- 1 layer of non-contextual embedding
- 12 MHA Layer
- 13 layer

5.1 Analysis of the Embedding in a single sentence

5.1.1 Probabilistic Framework

As we mentioned in the presentation of the probabilistic metric, our main idea is to consider that the context of the sentence is a point in our space of dimension 768.

Our objective is to try to see if the embedding of the tokens of a given sentence, converge towards a certain embedding (a certain vector/point), as we go up in the layers of the transformers.

To do this, we consider a first sentence:

"Alors, le monsieur lui donna son bateau, car elle était si mignonne qu'on ne pouvait rien lui refuser, et elle le fit transporter sur la corniche à côté de la petite maison."

We ran this sentence through the architecture of the Bert network, and retrieved the Embedding at the different layers of the architecture. For each layer of our network, we thus have an embedding matrix, on which we compute the trace of its variance covariance matrix, and here are the results obtained :

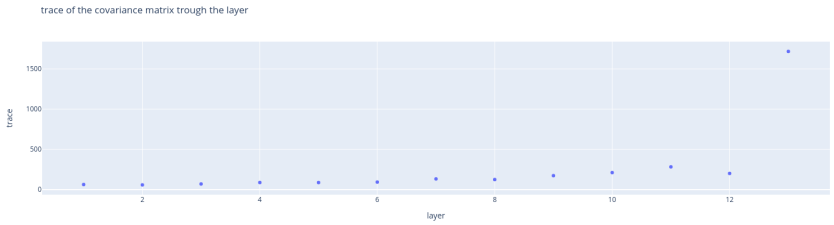
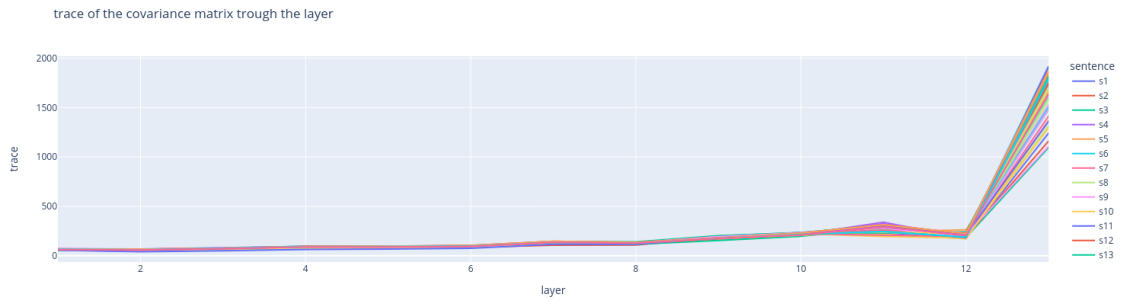
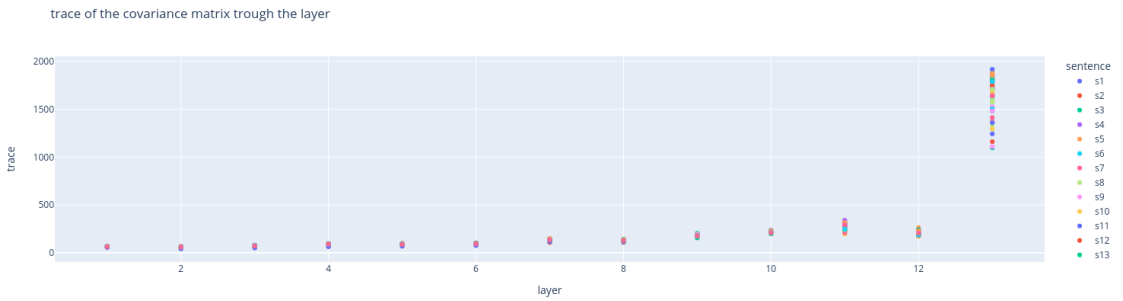


Figure 6: Trace through the layers

We observe a very interesting behavior which is that this trace explodes on the last layer. This may be a specific behavior of the sentence we considered. So we are going to do this same work again, but on a set of 52 different sentences, and here are the results we get:



(a) line plot



(b) scatter plot

Figure 7: Trace through layers for multiple sentences

We can thus see that whatever the considered sentence, we observe the same behavior, namely the explosion of the trace of the covariance matrix on the last layer the last layer acts as a dispersion layer. Moreover the evolution of the trace follows the same behavior, no matter the considered sentence. This is a first result that is interesting to note.

This explosion of the trace of the variance covariance matrix leads to a new question: Do all the values of the diagonal of this matrix explode, or do we have an explosion on some dimensions.

To answer this, we have thus, for each sentence, represented the diagonal of the variance covariance matrix of the embedding matrix on the last layer.

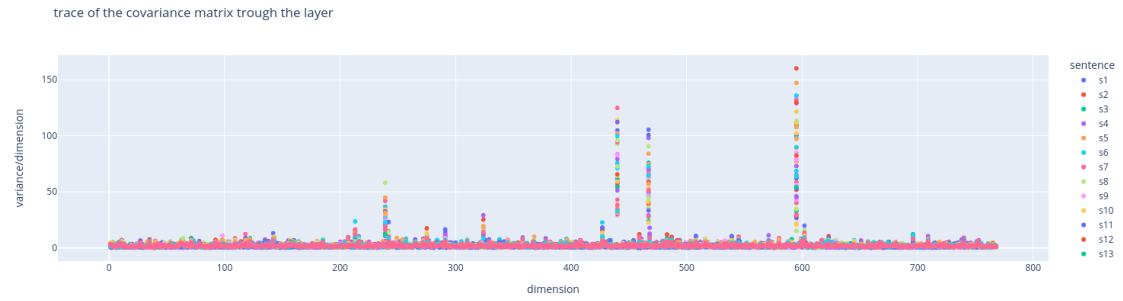


Figure 8: Diagonal on the last layer

We can see on this last graph, that the values of the diagonal, on the last layer, explode, but only on a handful of dimensions, and these dimensions are the same for each sentence.

5.1.2 Cosine similarity

We have thus observed the results given to us by the probabilistic point of view. This point of view does not allow us to conclude on any convergence of the embedding, quite the contrary.

We will now use the cosine similarity measure between the different embedding. In this new framework, we no longer assume that the context of the sentence is a point in space, but this time a direction. We want to see if the embedding of

the tokens align themselves in this famous direction.

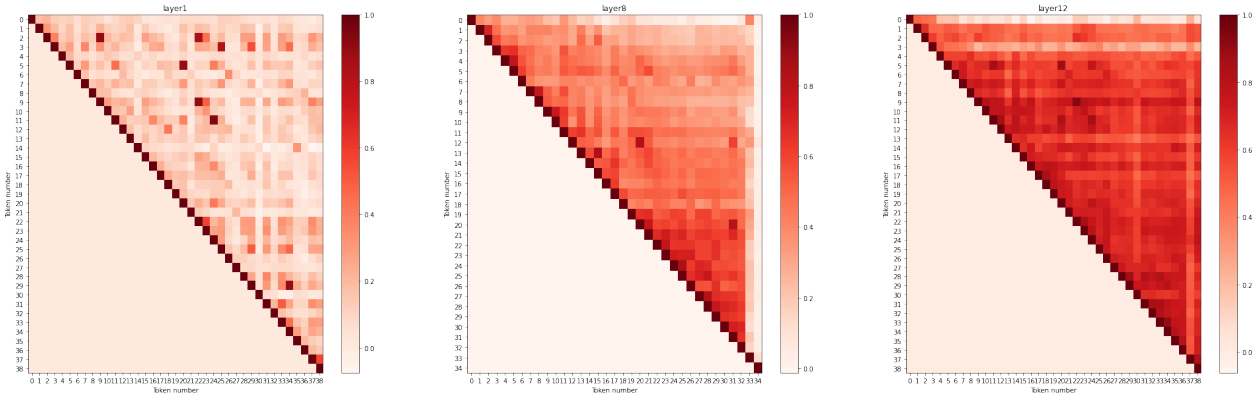
At first we will consider the same sentence as before:

"Alors, le monsieur lui donna son bateau, car elle était si mignonne qu'on ne pouvait rien lui refuser, et elle le fit transporter sur la corniche à côté de la petite maison."

This sentence will thus serve as our starting point. We are going to pass it inside the Flaubert's transformers, and get on each layer $k \in \{1, \ldots, 13\}$, the embedding of the different tokens of the sentence (we consider T, the number of tokens in this sentence). We will then form the similarity matrix :

$$W_{i,j}^{k} = sim(w_i, w_j) \text{ avec } i < j \in \{1, \ldots, T\}$$

If we look at the evolution of the quantities present in this matrix on layers 1 (non contextual embedding), 8 and finally 12, we obtain the following results:



(a) similarity matrix layer 1 (b) similarity matrix layer 8 (c) similarity matrix layer 12

Figure 9: Evolution of the similarity matrix trough the layers

On this figure, we can very well see that the image representation of the similarity matrix is more and more red. We observe a first convergence of the cosines towards the value of 1. The embedding of the tokens seem to align themselves as we go up in abstraction.

This progressive alignment is done on the first layers of our transformers. However, if we look at this matrix on layer 13 of our transformers, we obtain :

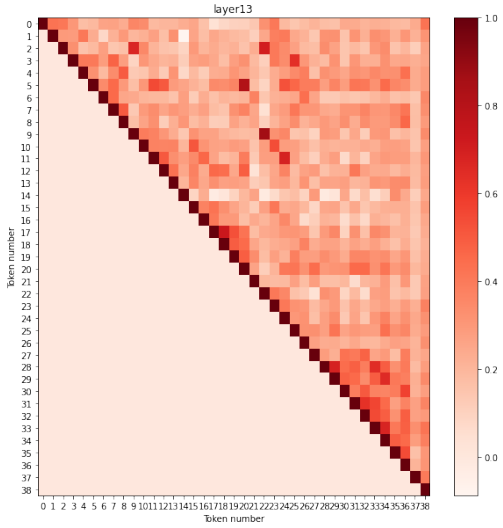


Figure 10: Similarity matrix on the last layer

On the last layer, the similarity matrix became clearer again. The last layer on this sentence had the effect of lowering the cosine value.

We can then ask ourselves if this effect is general, or if it is specific to the sentence we have chosen. To do this, we have selected a large number of sentences (here 52 sentences), and for each sentence, we have done the previous work, and for each layer, we have stored the cosine values, so that we can observe the way they are distributed according to the layers. We can thus draw the following graph:

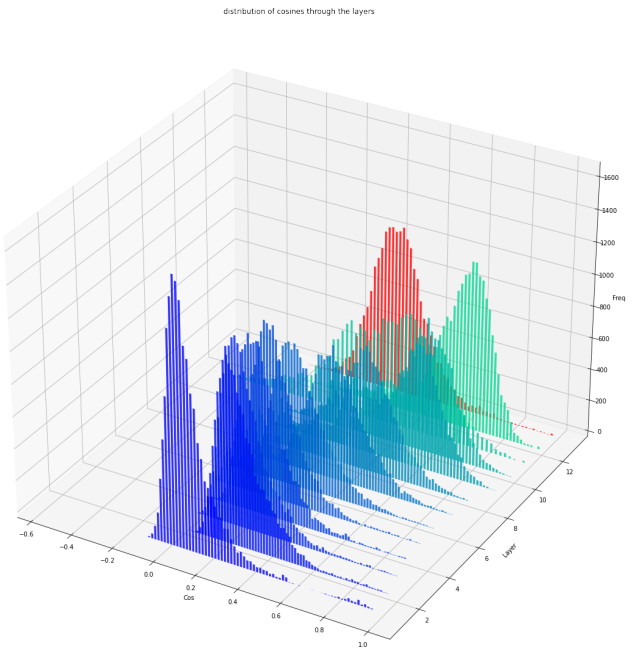


Figure 11: Cosine distribution through the layers

On this graph, we can clearly observe that the cosine values generally converge to 1 during the passage from layer 1 to layer 12. This means in vector terms, that our words are more and more aligned, we have words within a sentence that align on the same line in our 768 dimensional space. Then on the last layer, we find the dispersion effect observed during the study of the probabilistic framework. The cosines return to values close to 0, the words become independent again.

5.2 Analysis of the Embedding in different sentences

So far, we have analyzed words within a sentence. We will now look at a target word, but in different sentences, in order to see how embedding in different sentences behave in Flaubert's transformers. We will thus build a matrix set $\{W_i\}_{i\in\{1,...,13\}}$ for each layer i of our transformers, defined as follows:

$$W_i = \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,p} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ w_{768,1} & w_{768,2} & \cdots & w_{768,p} \end{pmatrix}$$

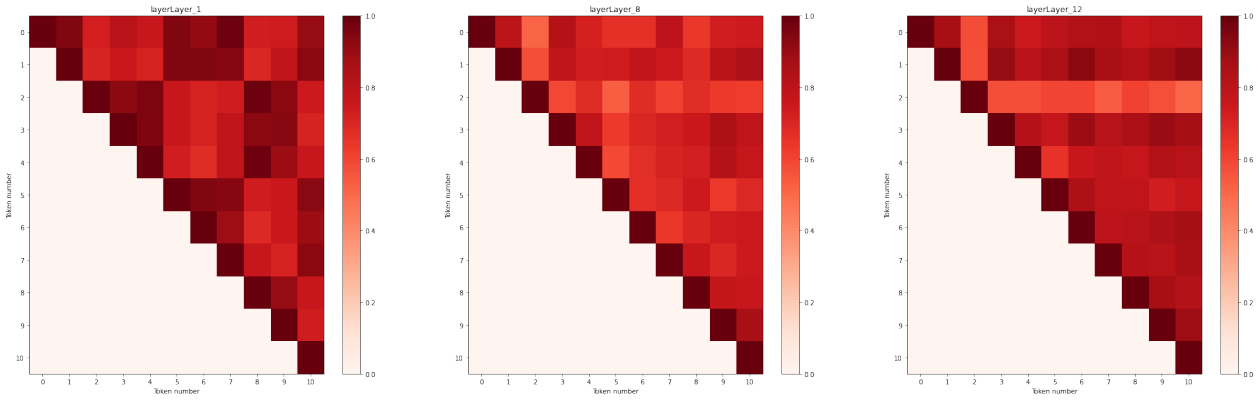
With $W_i(.,k)$, which is the embedding of the word we are studying at the i layer of the transformers, in the k sentence of our sentence set (we arrange the embedding in columns here).

5.2.1 Common noun

We will first look at the behavior of a target common noun in different sentences. For that, we have chosen the word "boat(x)". We have selected 11 sentences, in which this word appears. We transform each of these sentences, and we

can thus compute our set of $\{W_i\}_{i\in\{1,\dots,13\}}$ matrices.

We will then use the cosine similarity metric, in order to observe how the embedding behave with respect to each other. We thus form the same color matrices as in the previous section, but this time, the cosine is computed between the embedding of the word boat, but in different sentences. We obtain this on the first layers:



(a) similarity matrix layer 1 (b) similarity matrix layer 8 (c) similarity matrix layer 12

Figure 12: Evolution of the similarity matrix trough the layers (word : "bateau(x)"

On these layers we can see that despite the fact that the words are in different contexts, they remain aligned (cosine close to 1). However, when we look on the last layer, we have :

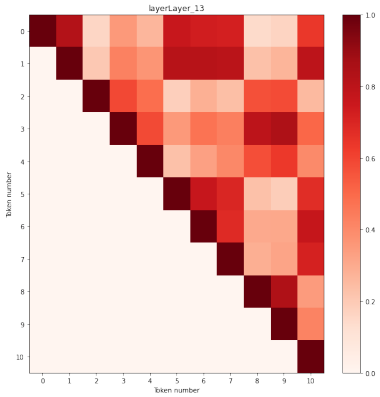


Figure 13: Similarity matrix (last layer)

We can clearly see on this last figure that on the last layer, the taking into account of the sentence, is important. Indeed in spite of the fact that the words are the same, we have cosine values which are different.

5.3 Analysis of the embedding space

5.3.1 Eigenvalues analysis on the last layer

A first natural way to analyze the space of embedding is to do an eigenvalue analysis of the variance-covariance matrix. This technique comes directly from the field of data analysis and more precisely from principal component analysis (PCA). For this purpose we have performed the following experiment:

We consider a sentence f, with a certain number of tokens T. We pass this sentence inside our Flaubert transformers, and we recover the embedding simply on the last layer.

We make our embedding matrix $X \in \mathbb{R}^{T\times 768}$, and on this matrix we estimate the variance covariance matrix $V \in \mathbb{R}^{768\times 768}$, then we compute the eigenvalues of this matrix.

We consider the modulus of these eigenvalues, and what will interest us is the decay of these modulus. Indeed, the stronger the decay, the less useful dimensions there are in our space, and so we can perhaps perform dimension reduction operations.

We have realized this experiment, with two sentences. The first sentence had a rather important number of tokens (148), and here are the results obtained:

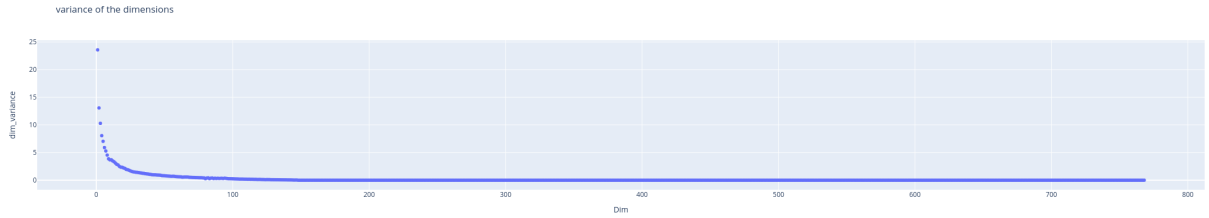


Figure 14: Eigenvalue decay (148 tokens)

Then we performed the same experiment, with a sentence where we have much less token (16), and here is what we can observe.

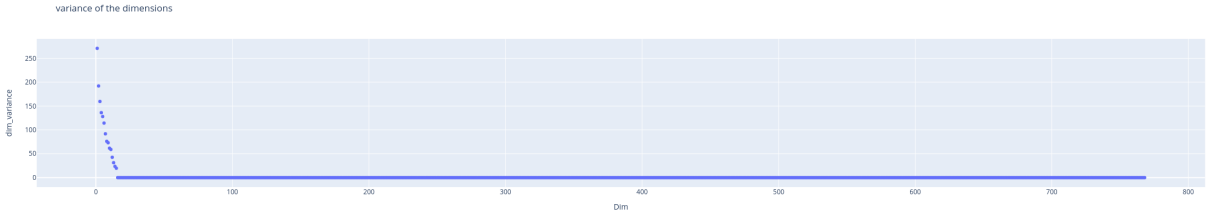


Figure 15: Eigenvalue decay (16 tokens)

We can see that on the sentence where we have much less token, the decrease of the eigenvalues is much more important, which will mean according to the classical interpretation in data analysis, that for a sentence where we have much less token, we need less dimensions to represent it in our embedding space. This seems quite coherent.

We performed this experiment on many other examples, and the conclusions were always the same.

However, it does not allow to characterize the embedding space in its totality, it simply allows to have an idea of the number of dimensions useful to represent a certain sentence. To characterize the space in its totality, one must use different measures.

5.3.2 Erank and Edim

We thus used a text with a considerable number of tokens, to form the embedding matrix W needed for these two metrics. We formed the embedding matrix with the embedding at the last level of the Flaubert transformers, and computed $erank(W_k)$ for all the k layers of the transformers the value of this metric. We obtain:

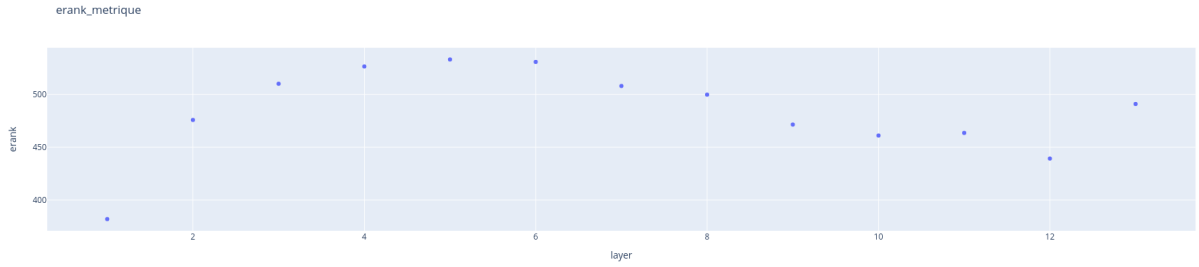


Figure 16: Erank through the layers

The values that our metric takes are relatively high sign that many dimensions are useful in our embedding space. To complete this study we computed the Edim metric for different values of p, and for each layer, and we obtain :

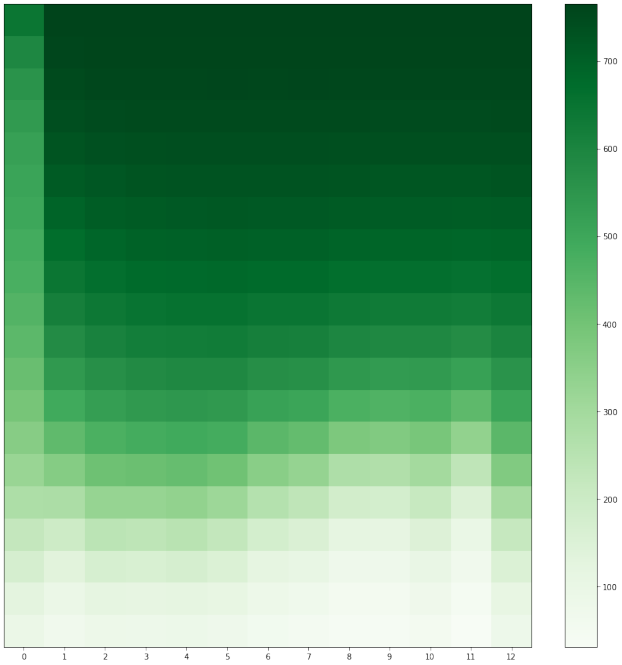


Figure 17: Edim through the layers

We can see that on each layer of multi-head attention, when p increases (increase on the y-axis), we have a value of metric which is higher and higher and which reaches the value of the dimension of the space $d = 768$.

If we combine these results, with the results found in [2] and [3], it means that to represent the set of our words in the embedding space used by the transformers, we use all the dimensions of the space.

5.4 Comparison with Bert's transformers

As we have seen in the section devoted to the training of transformers, what differentiates Bert and Flaubert is simply the language. Bert is trained on an English corpus, while Flaubert is trained on a French corpus.

So by retrieving Bert's transformers from the Huggin Face bookstore, we wanted to see if we could get results that are close to those found for Flaubert's transformers.

First, we looked at the evolution of the trace of the covariance matrix, over several sentences, and here is what we obtained:

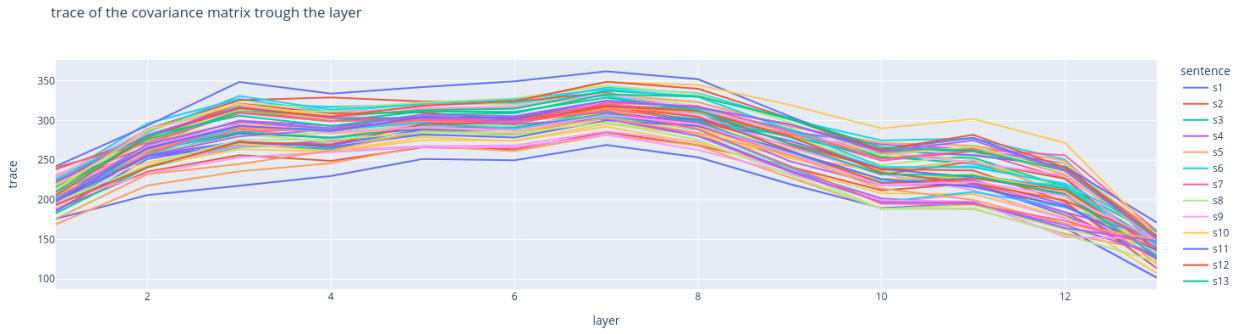


Figure 18: Trace for Bert

We can see that we do not get the same behavior as for the Flaubert transformers. Indeed on the latter, we had an explosion of the trace on the last layer. Here we observe the opposite, with a trace that strongly decreases on the last layer.

We have redone the same experiment that we did on the cosines with Flaubert, and here are the results obtained:

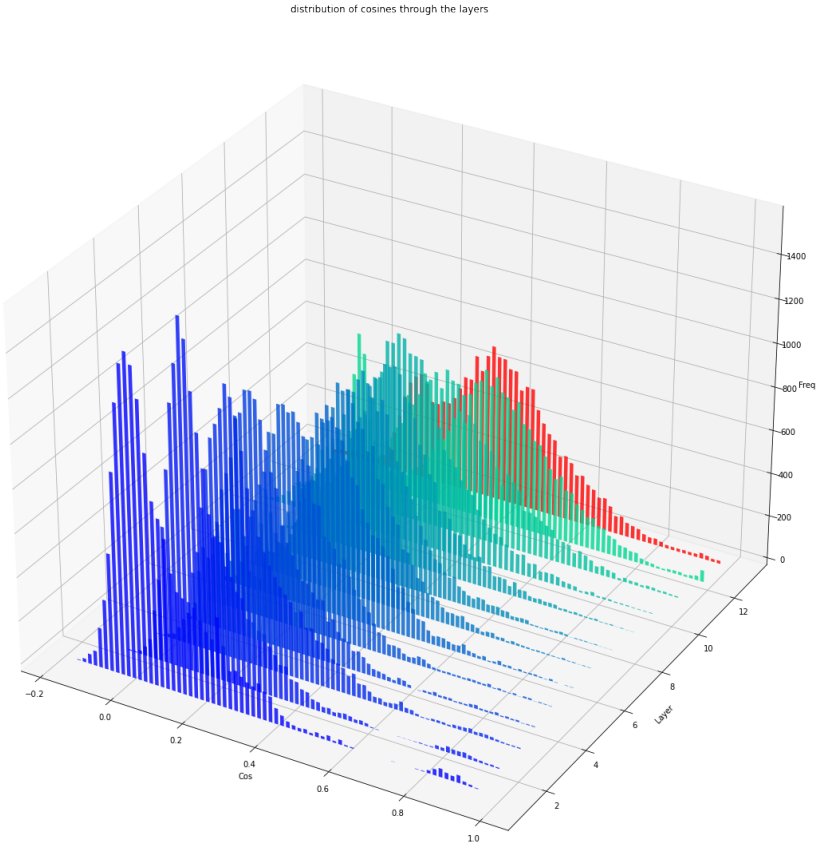


Figure 19: Evolution of the cosine for bert

We can see on this last graph, that we absolutely do not observe the same behavior as for Flaubert. The convergence to 1 of the cosines here is not really observable, and the last layer does not have the same effect at all.

These last results are interesting, indeed we can see that the learning language has a real impact on the behavior of the transform, which we did not suspect at all at the beginning.

6 Conclusion and opening

Through this project, I was able to integrate the world of artificial intelligence research for 6 months. During this period I was able to take in hand the notion of word embedding and learn about the state of the art in this field.

After that, we were able to take in hand the Flaubert/Bert transformers of the Huggin face bookstore, and to imagine a whole series of experiments, in order to verify the starting hypotheses that we fixed ourselves.

However, none of our experiments allowed us to confirm our hypothesis (on the contrary). We showed that the last layer of the Flaubert transform had a significant dispersion effect, which was not at all assumed at the beginning. More importantly, we were able to show that we did not find the same results at all using the Bert transform, which is surprising given that the architecture is the same only the training data changes.

However, we did not go to the end of our work since we wanted to perform the same analysis, but by performing a fine tuning on the Flaubert transformers.

The code to realize this fine tuning is ready and functional, but the transformers are very demanding in computational resources and we had to do a lot of work to deploy the code on the IRISA server. However, technical problems have slowed us down.

Our last objective is to perform this fine tuning operation to realize these analyses before submitting this article for the **TALN** conference in order to have a complete analysis.

References

[1] Niki Parmar Jakob Uskoreit Llion Jones Aidan N.Gomez Lukasz Kaiser Illia Polosukhin Ashish Vaswani, Noam Shazeer. Attention is all you need. 2017.

[2] Nihel Kooli Guillaume Gravier Robin Allesiardo François Torregrossa, Vincent Claveau. On the correlation of word embedding evalution metric. 2020.

[3] Vincent Claveau Nihel Kooli Guillaume Gravier François Torregrossa, Robin Allesiardo. A survey on training and evaluation of word embeddings. 2021.

[4] Denis Rothman. *Transformers for natural language processing*. Packt Publishing, 2021.