Beyond Spell-checking: Word-checking

An attention based Transformer approach

Abdurrahman Shahid

A report presented for the Study and Research Work

(Travaux d'Etude et de Recherche - TER)

L3 MIASHS Cognitive Science

FST - Mathematics Department

University of Lille

France

2021

Abstract

Contents

CHAPTER 1

Introduction

This report is intended to present and discuss the work accomplished during the TER. In this chapter we outline the key ideas of this TER and introduce some of the concepts needed to understand the reminder of this report.

1.1 The general framework

Let's begin by presenting the issue we will try to solve throughout this report. As we grow up, we are able to make less and less mistakes when writing a sentence or a long text. However, some words are more difficult to spell correctly than others. Especially, words called *homophones* can be quite challenging. What are homophones ? **Homophones** are words that are pronounced the same, but they have different writings and different meanings. For instance, in the French language - the language used for all of the studies in this TER - the words "leur" and "leurs" are homophones. To spell properly these homophones, we must use information such as the context in which they occur. To make matters worse, when humans make mistakes, they are very confident that they have chosen the right word (see 3.2). Then we can wonder, how can we avoid these pitfalls ?

We can certainly treat such a question by leveraging knowledge from many different fields of study. For instance, we can use knowledge and concepts from linguistics to help us address the issue of choosing the right homophone. We can further use brain imaging

techniques to spot any differences between the condition where we choose the right form and the condition we don't. The latter methods could be, broadly speaking, categorized as the cognitive sciences approach. While cognitive sciences are arguably well suited for this task, we won't take this approach. Instead, we will take a more computer science oriented approach and tackle the issue of homophones by using **deep learning** (commonly abbreviated as **DL**) and the latest breakthrough in **natural language processing** (commonly abbreviated as **NLP**) known as the *Transformer* [8]. The Transformer has democratized and brought to a new level the use of *attention* mechanisms in NLP and sequential problems in general. More precisely, we wil try to answer the following question: **can we implement an effective attention based Transformer model for homophone correction ?**

We discuss what we mean by the term *effective*, and the terms *homophone correction* in chapter 3. Throughout this chapter, we will have a glimpse of what *attention* and *Tranformer model* mean, but their in-depth explanation is left for the next chapter.

1.2 DL and NLP prerequisites

In this section we will review basic concepts of DL and NLP needed to understand the material present in chapter 2 and chapter 3. Throughout this report, we won't get into too much details when it comes to mathematical theory or more technical methods usually employed in DL. Indeed, this is not the objective here and is beyond the scope of this report, in addition to the fact that many great resources on these topics already exist (see [4], and [2]). However, we will always try to give an intuitive explanation, enough to understand what the different components are used for. Also, additional explanations are given in the appendices. Most of the pages are dedicated to explain the in-depth functioning of the Transformer model and its building blocks, along with the experiment that have been done.

1.2.1 Deep Learning prerequisites

In deep learning we use terms like loss function, gradient, optimizer, true label, parameters, layers, units, activation function, etc. For an exhaustive survey see [4], [3] and [1]. We will mainly focus on the components used to implement the Transformer.

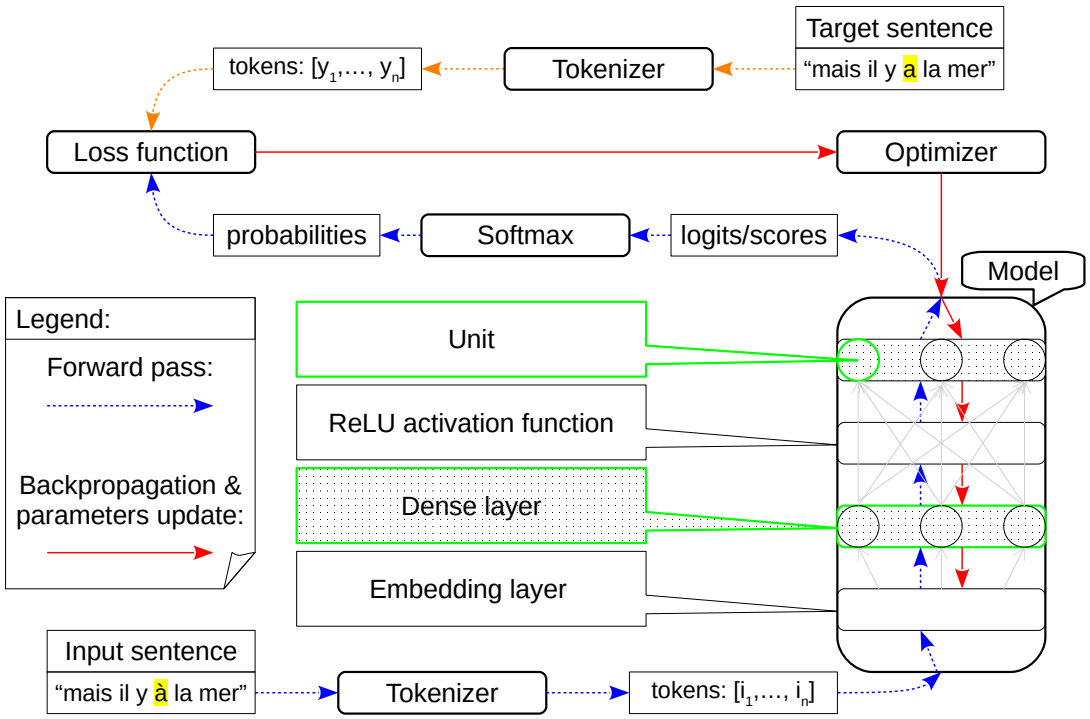


Figure 1.1: How different DL components can be related to each other during training. The model shown here is not the Transformer, and serves only as an illustration.

In the example in figure 1.1, we give the tokenized input (see 1.2.2 for more details) to the model, and then compare what the model outputs with the correct sentence i.e. target sentence. Target sentence is employed here to emphasize that, for each input sentence, we have the corresponding correct sentence (itself if it's already correct or the corresponding corrected sentence). Thus, the model is trained in a supervised setting. We want the model to correct the errors present - if any - in the input sentence, and output the corrected sentence. For instance in figure 1.1, the input contains an error whereas target sentence doesn't (see the highlighted terms "à" and "a"). That being said, let's review what each component in figure 1.1 does. First, the list of tokens (each token is a number) of length l goes through the **Embedding layer**, where each token is transformed into a vector of a given fixed dimension d called *embedding dimension*. Indeed, for every token in the vocabulary (see 1.2.2), the embedding layer has an associated vector of dimension d. For instance if $l = 11$ (there are 11 tokens) and $d = 64$, then the list of tokens is

transformed into a matrix X having 11 rows and 64 columns ($X \in \mathbb{R}^{11\times64}$). This newly created matrix is then fed to a **Dense layer**. Moreover, a dense layer consists of **units** and those *units* determine the shape of the weight matrix W representing the layer. What happens at the dense layer boils down to a matrix multiplication. Indeed, if we omit the bias term, and the number of units is equal to 16, then the following matrix multiplication occurs: $X \times W$ with $X \in \mathbb{R}^{11\times64}$ and $W \in \mathbb{R}^{64\times16}$ resulting in a matrix $X_{\text{new}} \in \mathbb{R}^{11\times16}$. Oftentimes an activation function is applied to a layer's output. The activation function used in figure 1.1 - and also by the Transformer - is called **ReLU** activation function. For every real number x, $\text{ReLU}(x) = x$ if $x \geq 0$ and $\text{ReLU}(x) = 0$ if $x < 0$. ReLU is applied *element-wise* to the previous matrix $X_{\text{new}} \in \mathbb{R}^{11\times16}$. Activation functions serve to inject non-linearity into our models. Finally, in figure 1.1, the output of the ReLU goes through a second Densely connected layer. If this second layer has $|V|$ units, then we get a new matrix $S \in \mathbb{R}^{11\times|V|}$ ($S = X_{\text{new}} \times W_2$ with $X_{\text{new}} \in \mathbb{R}^{11\times16}$ and $W_2 \in \mathbb{R}^{16\times|V|}$). As we will see in section 1.2.2, $|V|$ denotes the size of the vocabulary, i.e. the total number of tokens that make up the vocabulary being used by the Model. Notice that this time we didn't use an activation function after the dense layer. Thus, elements of S can be any real number, we call these numbers *logits*. Next, we use the **Softmax** function to get *probabilities*. We apply softmax along the rows of S and obtain the probability matrix P:

$$P = softmax_j(S) \quad \text{i.e.} \quad P_{ij} = \frac{\exp(S_{ij})}{\sum_{j=1}^{|V|}\exp(S_{ij})} \tag{1.1}$$

P_{ij} gives for the i^{th} output token, the probability that this token is the j^{th} token of the vocabulary. In our example (figure 1.1), $P \in \mathbb{R}^{11\times|V|}$, hence the model outputs probabilities for 11 tokens (corresponding to 11 rows in P). Intuitively, we can consider that the i^{th} output of the model, is the j^{th} token of the vocabulary, such that P_{ij} is the maximum i.e. $j = argmax_j(P_{i,.})$, for those who are familiar with this notation.

Furthermore, as we have the correct sentence i.e. target sentence, we also have the correct tokens i.e. target tokens, we tokeize the target sentence to get them. The same tokenizer is used to tokenize both the input sentence and the target sentence. Consequently, we can compare how well the model is doing for the task of correcting incorrect sentences. We compare the model's output - the probability matrix P - with the target tokens from the target sentence. This is accomplished by the **Loss function**, it takes as input both the list of target tokens and the probability matrix P, and it calculates the

loss, which is a measure of how different the model's output is from what is expected i.e. target sentence. We want this loss to be as low as possible. If we denote the loss function by L, and all the trainable parameters by θ, then parameters update is given by:

$$\theta_{\text{new}} = \theta_{\text{old}} - \epsilon \times \nabla_\theta L \quad \text{where} \quad \nabla_\theta L, \text{ is the gradient of } L \text{ w.r.t } \theta \tag{1.2}$$

We notice that, $\nabla_\theta L$ - obtained via the *backpropagation algorithm* - indicates the direction in which θ is moved, and ϵ specifies by how much it is moved. Additional update options can be configured and the **Optimizer** handles their implementation. This ends the quick review of the basic DL concepts needed to get the big picture of the problem. Next, we review some concepts related to NLP and the loss function in detail.

1.2.2 Natural Language Processing prerequisites

In order to train and evaluate a model, we have to first define a vocabulary. This vocabulary is linked to the way we *tokenize* textual data. Indeed, when we encode textual data using a **Tokeinzer**, the resulting token representation depends on the tokens the tokenizer is able to handle. For instance, a character-level tokenizer encodes each character separately, while a world-level tokenizer encodes each word separately. However, we will use for the Transformer a *subword-level* tokenizer. The subword tokenizer is first trained on a text corpus, then *the most frequent chunks* [see 2, for more details] of text are each associated with a number between 1 and $|V|$. Here, $|V|$ indicates the maximum number the tokenizer can emit. Moreover, at the beginning (respectively end) of a sentence, we add the $<SOS>$ token (respectively the $<EOS>$ token), see figure 1.2 for an example.

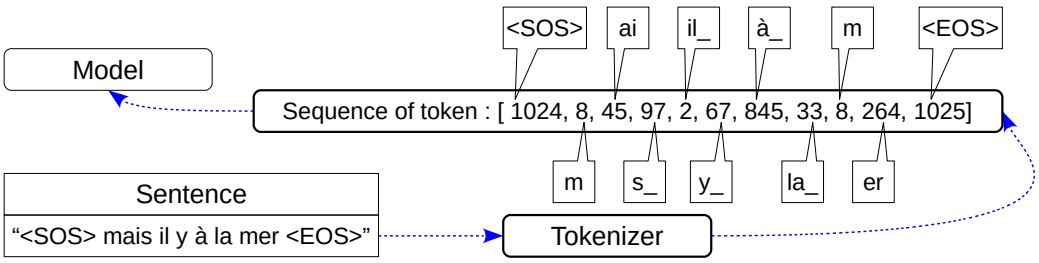


Figure 1.2: How Subword Tokenizer encodes sentences.

Finally, we describe the loss function L used in figure 1.1, and also used by the Transformer. More precisely, L is the *Sparse Categorical Cross-Entropy Loss*. We saw

in section 1.2.1, that this loss function takes as input the list of target tokens, and the probability matrix P from the model. Let's denote by y the list of target tokens, and $y_i \in \{1, ..., |V|\}$ the i^{th} token in the list. Moreover, l is both the length of y and the number of rows in P (in figure 1.1, $l = 11$). L is given by formula 1.3, and measures the loss for a pair input sentence/target sentence:

$$L(y, P) = -\sum_{i=1}^{l} \log(P_{i,y_i}) \tag{1.3}$$

We notice that, both the length of y and the number of rows in P must be equal (it's the same number l), we will see in the next section how it is ensured for the Transformer model. Additionally, if P_{i,y_i} tends to 1, then $\log(P_{i,y_i})$ tends to 0, and if P_{i,y_i} tends to 0, then $\log(P_{i,y_i})$ tends to $-\infty$ (then L tends to $+\infty$). Consequently, minimizing L forces P_{i,y_i} to be close to 1, i.e. the model gives a high probability for the i^{th} output token to be the i^{th} token in the target token list. Thus, the model *learns* to output what we expect it to output (the target tokens) given a certain input.

1.3 High-level picture of the problem setup

In this section, we look at the elements surrounding the training of the Tranfsormer. To begin with, the set-up used to train the Transformer is similar to the setting in figure 1.1, with changes only occurring at the model level. Indeed, as shown in figure 1.3, the Transformer is composed of two main blocks - an **Encoder** block and a **Decoder** block - along with a dense i.e. linear layer. Within these blocks, is implemented the *attention* mechanism. Moreover, *the Transformer takes as input two sequences*, one entering through the encoder and the other through the decoder. The encoder takes as input the list of tokens from the input sentence. However, the decoder takes as input the list of tokens from a truncated version of the target sentence. Indeed, from the entire target sentence, we extract two truncated sentences, one without the "<EOS>" token, the other without the "<SOS>" token. Both have the same length after tokenization, it's mandatory to use the loss function 1.3. For instance, in figure 1.3, from the target sentence "<SOS> mais il y a la mer <EOS>", we extract "mais il y a la mer <EOS>" and "<SOS> mais il y a la mer". The one without the "<EOS>" token is given as input to the decoder. The other one, without the "<SOS>" token is considered as the target sentence, and the resulting list of tokens (after tokenization) is given to the loss function.

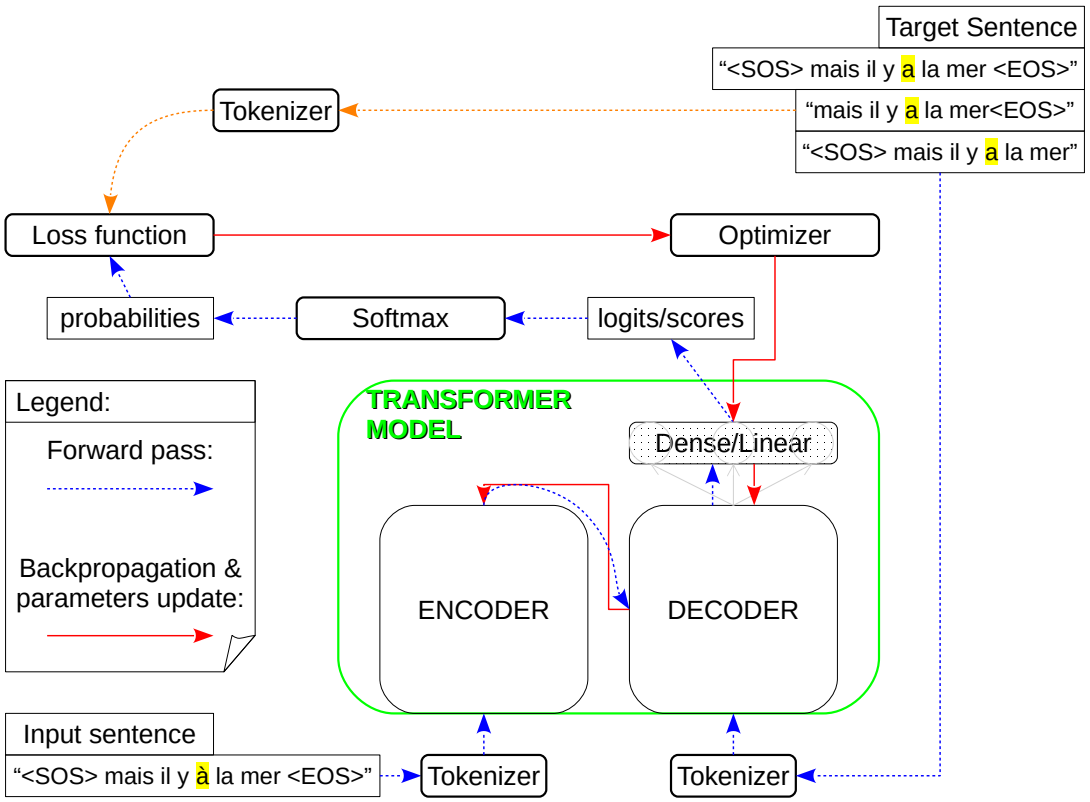


Figure 1.3: How the Transformer model fits into the big picture, with its Encoder and Decoder blocks along with a final linear layer. This figure illustrates the setting during the training phase.

We can also observe that the sequence given as input to the decoder is the sequence used as the target, but shifted by one token to the right. Since, the 1st token/word in the target sequence (for instance "mais" in figure 1.3) is the 2nd token/word in the sequence given as input to the decoder, and this observation applies to all following tokens/words in the target sequence. This is indicated in figure 2.1 in chapter 2 by "*Outputs (shifted right)*".

In the next chapter, we unpack the encoder and the decoder to see the building blocks that make them up. We also present how the Transformer is used during the evaluation/test phase.

CHAPTER 2

The Transformer Explained

In this chapter, we will see how the inputs are transformed within the Transformer model, and how the *attention* mechanism works. As we have seen in figure 1.3, there isn't one input but two. Indeed, the model takes as input the input sentence (at the encoder level), for instance "<SOS> mais il y à la mer <EOS>", and what we call *the decoder input*, for instance "<SOS> mais il y a la mer". At training time, from these two inputs, the model is trained to output "mais il y a la mer <EOS>", which is the target i.e. true label. We will illustrate the inner working of the Transformer by following what happens to our sentence from the previous chapter, from the moment it enters the model at the encoder level and the moment the model outputs probabilities for each output token. See section 1.3 for a refresher and the explanation of the meaning of "*Outputs (shifted right)*" at the decoder level in figure 2.1. Throughout this chapter we will describe in detail each component shown in figure 2.1, therefore you should refer to it as many times as necessary.

2.1 The Encoder Input

We first tokenize the sentence "<SOS> mais il y à la mer <EOS>", which gives us a list (or a vector) of tokens. Moreover, suppose that after tokenization the sentence is $l = 11$ tokens long as in figure 1.2. Recall that tokens are just numbers, each one representing a specific subword (see section 1.2.2 for more details). This list of tokens is represented by

"**Inputs**" and this is what goes into the encoder.

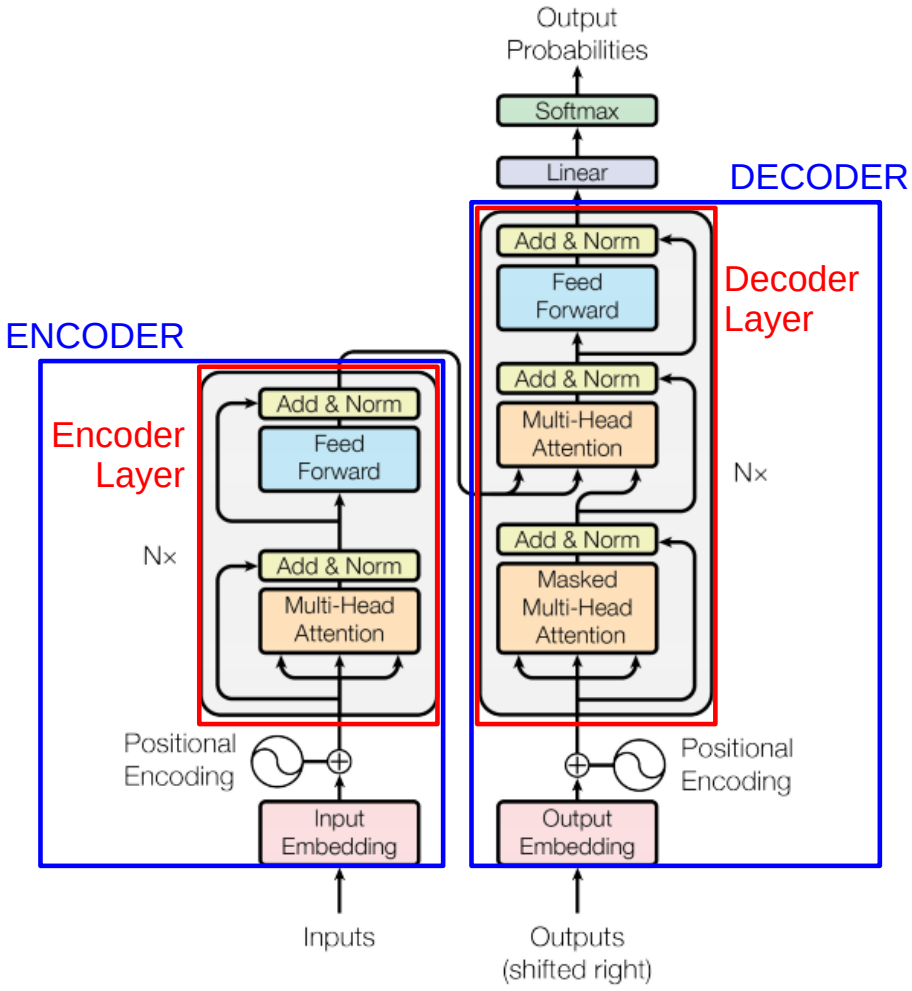


Figure 2.1: The Transformer - model architecture adapted from the original paper [8].

2.1.1 Input Embedding and Positional Encoding

The list of tokens goes through the *Embedding layer* (see 1.2.1, for the details) and we get a matrix $X \in \mathbb{R}^{l \times d_{model}}$. Here, d_{model} represents the dimension of the vectors representing each input token. If we set $d_{model} = 64$, then we get an "**Input Embedding**" matrix $X \in \mathbb{R}^{11 \times 64}$. Though we have encoded the information about words identity until now, we didn't encode the information about words position in the sentence. To inject the information about the position, we add X and a *sinusoidal* matrix R, with R having the same dimension as X. If $R \in \mathbb{R}^{l \times d_{model}}$, then each element of R is given by:

$$\forall i \in \{1, ..., l\}, \forall j \in \{1, ..., d_{model}\}, R_{ij} = \begin{cases} \sin(i/10000^{2j/d_{model}}) & \text{if } j \text{ is an even integer} \\ \cos(i/10000^{2j/d_{model}}) & \text{if } j \text{ is an odd integer} \end{cases}$$

R represents the "**Positional Encoding**". Let's call the resulting matrix $X_E \in \mathbb{R}^{l \times d_{model}}$, $X_E \in \mathbb{R}^{11 \times 64}$ in our example, encoding both the identity of the words and their position within the sentence. Then X_E enters the *Encoder Layer*. All the computations occurring within the latter can be repeated N times as indicated in figure 2.1.

2.1.2 Multi-Head Attention

First of all, **Multi-Head Attention** is made of multiple *Single-Head Attention*. Lets denote by *Attention* the calculations performed in a Single-Head Attention module and by h the number of Single-Head Attention composing the Multi-Head Attention module. The Attention module (and hence the Single-Head Attention) takes as input three matrices denoted by $Q \in \mathbb{R}^{l_q \times d_q}$, $K \in \mathbb{R}^{l_k \times d_k}$ and $V \in \mathbb{R}^{l_v \times d_v}$. We usually refer to these matrices by "Query" matrix, "Key" matrix and "Value" matrix respectively. In the encoder we obtain these matrices as follows:

$$Q = X_E W_Q \ , \quad K = X_E W_K \ , \quad V = X_E W_V \ , \quad \text{with } X_E \in \mathbb{R}^{l \times d_{model}} \tag{2.1}$$

with $W_Q \in \mathbb{R}^{d_{model} \times d_q}$, $W_K \in \mathbb{R}^{d_{model} \times d_k}$ and $W_V \in \mathbb{R}^{d_{model} \times d_v}$. Moreover, we have $depth = d_q = d_k = d_v = d_{model}/h$. If there is $h = 4$ heads (as in our implementation in chapter 3) then we have $depth = d_q = d_k = d_v = 64/4 = 16$. Single-Head Attention is then given by:

$$\text{Attention}(Q, K, V) = softmax_j(\frac{QK^T}{\sqrt{d_k}})V = AV, \quad \text{with } A = softmax_j(\frac{QK^T}{\sqrt{d_k}}) \tag{2.2}$$

More precisely, matrix $A \in \mathbb{R}^{l_q \times l_k}$ contains the **attention** weights. By using the softmax function along the columns we get, for each row of A, weights between 0 and 1. Afterwards, the matrix multiplication AV calculates a linear combination of the rows of $V \in \mathbb{R}^{l_v \times d_v}$ weighted by the attention weights found in A. Thus, if we denote by AV_i the i^{th} row in AV and V_i the i^{th} row in V then we have:

$$AV_i = \sum_{j=1}^{l_q} A_{ij} V_i \tag{2.3}$$

In other words, as each row in V represents a token in the token list, AV contains a new representation for each token that takes into account the information about the other tokens in the sentence. Therefore, the representation of a token or word is not fixed, it changes depending on the context (surrounding words) where it occurs. Attention weights

dictate how much each of the surrounding words/tokens should contribute for the new representation of a given word/token. Notice also that, $l_q = l_k = l_v = l = 11$ here. To summarize each Single-Head outputs a matrix that contains a new representation for each token and the matrices W_Q, W_K, and W_V are learned by the model during training.

As we have h heads, we also have h sets of matrices $\{W_Q^i, W_K^i, W_V^i\}$ for $i \in \{1, ..h\}$ that are learned by the model during training for a Multi-Head Attention layer. Each head outputs a matrix, we then concatenate all these matrices along the columns to get a single matrix that we finally multiply by another matrix $W_O \in \mathbb{R}^{hd_v \times d_{model}}$. At the end of the Multi-Head Attention layer we have a matrix $X_{MHA} \in \mathbb{R}^{l_q \times d_{model}}$. With our numeric example we have $X_{MHA} \in \mathbb{R}^{11 \times 64}$.

2.1.3 Add & Norm

Each **Add & Norm** layer, takes as input, both the input and the output of the previous layer. It adds up both and then normalizes the result. The normalization used is *Layer Normalization*, which standardizes the output along the columns for each row of the matrix obtained after the aforementioned addition.

2.1.4 Feed Forward

Feed Forward layer consists of two Dense layers with a ReLU activation in between. See section 1.2.1 for a refresher on Dense layers and ReLU activation function. The first Dense layer outputs a matrix $X_{FF1} \in \mathbb{R}^{l \times dff}$ (we use $dff = 256$ in chapter 3) and the second Dense layer outputs a matrix $X_{FF2} \in \mathbb{R}^{l \times d_{model}}$. The weight matrices associated with both of the Dense layer are trainable weights learned during training. After the Add & Norm layer following the Feed Forward layer in the Encoder we can either repeat once again the Encoder layer (the output will be fed anew as X_E) or we take the output as the Encoder's output. For convenience we will denote the Encoder's output by $H \in \mathbb{R}^{l \times d_{model}}$. We usually refer to it as the Encoder's *Hidden states*. H is then employed by the Decoder's Multi-Head Attention layer (see figure 2.1). More precisely H is used to calculate the key matrix K and value matrix V as we will see in section 2.2.3. Now lets see how the Decoder processes its input.

2.2 The Decoder Input

The Decoder takes as input the target sentence without the <EOS> token. In our example (figure 1.3) it takes as input "<SOS> mais il y a la mer". Suppose that after tokenization, the length of the input tokens list is $m = 10$ (recall that l indicates the length of the Encoder input and $l = 11$ in our example). This list of tokens is represented by "**Outputs (shifted right)**" in figure 2.1.

2.2.1 Output Embedding and Positional Encoding

As with the Encoder input, the vector of tokens is first transformed into a matrix. We then add to it the positional encoding matrix $R \in \mathbb{R}^{m \times d_{model}}$. We denote the resulting matrix by $X_D \in \mathbb{R}^{m \times d_{model}}$. X_D then enters the *Decoder Layer*. As with the Encoder Layer, all the computation occurring inside the Decoder Layer can be repeated N times.

2.2.2 Masked Multi-Head Attention

Everything is the same as the Multi-Head Attention in the Encoder except that we use X_D instead of X_E in equation 2.1 and $l_q = l_k = l_v = m = 10$ here. What is new is the "**Masked**" part. As we give to the Decoder what it has to output but shifted by one token, it can just learn to output what it takes as input but shifted by one token.Thus, it won't learn anything interesting but to shift a sequence. To prevent the model from "cheating", we apply a **mask** to the matrix A containing the attention weights in equation 2.2. This mask zeroes out all the coefficients above the main diagonal i.e. $A_{ij} = 0$, if $j > i$. See**??** for an illustration. Therefore, in order to output the i^{th} token, the model can't look ahead and can only attend to tokens that that comes before it in the Decoder input. For our example, we get a final matrix $X_{MMHA} \in \mathbb{R}^{10 \times 64}$.

2.2.3 Add & Norm, Multi-Head Attention and Feed Forward

The Add & Norm layer is the same as the one in the Encoder. We continue to denote by X_{MMHA} the output of this layer. What is new in the next Multi-Head Attention layer, is that $H \in \mathbb{R}^{l \times d_{model}}$ from the Encoder get involved in the computations along with $X_{MMHA} \in \mathbb{R}^{m \times d_{model}}$. Indeed, H is used to calculate K and V and formula 2.1 becomes:

$$Q = X_{MMHA} W_Q \ , \quad K = H W_K \ , \quad V = H W_V \tag{2.4}$$

with $Q \in \mathbb{R}^{m \times d_q}$, $K \in \mathbb{R}^{l \times d_k}$ and $V \in \mathbb{R}^{l \times d_v}$. As in the Encoder we have $depth = d_q = d_k = d_v = d_{model}/h = 64/4 = 16$ for our example. Everything else (equation 2.2 and 2.3) remain the same. Finally the output of the Multi-Head Attention goes through an Add & Norm layer and a Feed Forward layer. The latter outputs a matrix $D \in \mathbb{R}^{m \times d_{model}}$, see figure 2.1. We can either repeat the Decoder Layer one more time (D will be given as input to the Masked Multi-Head Attention layer) or we can pass D to a final Linear layer.

2.3 From the Decoder Output to Probabilities

Matrix D goes through a final Linear layer which has $|V|$ units. Thus, it outputs a matrix $S \in \mathbb{R}^{m \times |V|}$ containing the *scores*, see figure 2.1. Then S is softmaxed along the rows and we get a matrix $P \in \mathbb{R}^{m \times |V|}$ that contains probabilities, see equation 1.1. For the bigger picture of the problem setup see figure 1.3.

2.4 The Transformer at evaluation time

CHAPTER 3

Homophone Correction using The Transformer

The first two chapters set the stage for this last chapter of the report. Indeed, we will use the knowledge and expertise gained from the study of DL and the Transformer architecture to implement a homophone corrector. Therefore we will have an answer to the question stated in the introduction: **can we implement an effective attention based Transformer model for homophone correction ?**.

One of the criticisms often voiced about deep learning (DL) models is that they need to train on a large amount of data and therefore also require a lot of computing power. To give you an idea, the models in DL, and especially those in NLP, that make headlines with their exploits, are often trained for several weeks in a row on powerful machines, with data ranging from a few gigabytes to several hundred gigabytes, and these models have a number of parameters ranging from a few tens of millions to several hundreds of millions, some even reaching tens of billions of parameters !. Consequently, the trained model is deemed as ***effective*** because it is a relatively small model that does not need a large amount of data or computing power. The performances you will see were obtained with a model that took only *half an hour* to converge (i.e. it has learned the maximum it could learn with the data it was given) on a PC with GPU. Furthermore, the model was trained on *100,000 sentences* in total, the file containing these sentences requires *6 MB* on disk - the size of about 2 photos taken with a smartphone, which is remarkably small.

Also, the model has *431,490 parameters*, and takes up only *5 MB* on the disk.

Finally, the model was trained with 10 pairs of french homophones: (a, à), (est, et), (ces, ses), (ce, se), (ou, où), (la, là), (tout, tous), (leur, leurs), (ceux, ce), (cette, cet). Henceforth we will refer to the two homophones of a given pair as h_1 and h_2. The aim was to train ***an attention based Transformer model*** that correct sentences in which the wrong homophone is employed. For instance in figure 1.3 the input sentence contains "à" but it's incorrect because the right form here is "a". In the latter case, the model is expected to output the sentence by replacing "à" by "a". Thus, the model will take as input a sentence containing an incorrect homophone (h_1 or h_2) and output the sentence by correcting it and replacing the wrong homophone with the correct one (respectively h_2 or h_1). Thus we employ the model for ***homophone correction***. Of course if the sentence is already error-free i.e. the correct homophone is employed, then the model is expected to output exactly the same sentence as the one received as input.

3.1 Methodology

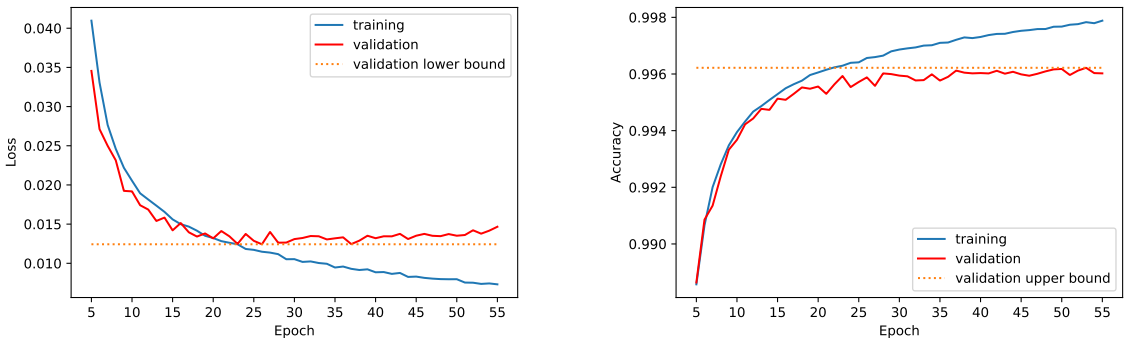
To assess the relevance of such a model, we compare the model's performance with human performance on the homophone correction task. The comparison metrics used are **Precision**, **Recall** and **Accuracy** derived from the counts of *true positive* (**TP**), *false positive* (**FP**), *true negative* (**TN**) and *false negative* (**FN**). For each pair of homophone (h_1, h_2), we have 4 possibilities. Among these 4 possibilities, there are 2 possibilities where one of the two homophones (say h_1) is present in the input sentence, and 2 possibilities where the other one (say h_2) is present. Indeed, h_1 (respectively h_2) can be present in the sentence and either the sentence is correct or the sentence is incorrect because it should have been written h_2 (respectively h_1) instead of h_1 (respectively h_2).

For each pair of homophone the model was trained on $10,000$ sentences; $2,500$ sentences for each of the 4 aforementioned possibilities. Then for each pair the model was tested on $2,000$ sentences; 500 sentences for each possibility. During training, a validation set comprising the same number of sentences as the test set was used to monitor model convergence. Additionally, notice that if the model is given a sentence where h_1 is present and the sentence is correct, it can either keep h_1 (TN) or replace h_1 with h_2 (FP). Now, if the model is given a sentence where h_1 is present and the sentence is incorrect, it can either keep h_1 (FN) or replace h_1 with h_2 (TP). The same reasoning applies to h_2.

See appendix **??** for detailed information on data selection and the online *Github repository* for all source code [6]. Besides, the trained Transformer have $N = 2$ (Encoder/Decoder) layers, $d_{model} = 64$, $dff = 256$ and number of heads $h = 4$. Moreover, human performance was measured through an *online form* [7]. They were also asked, for each question, to rate the level of confidence they had in their answer, on a scale ranging from 1 (not confident at all) to 10 (very confident). Subsequently, **exact binomial test** was performed to determine statistically significant differences between the model and human performance. See appendix **??** for a refresher on the metrics used and statistical hypothesis testing.

3.2 Results

First of all, the Transformer converged (the model learned as much as it could from the training data without overfitting) after 25 epochs, and 1 epoch took about 1 minute. Indeed, as we can see in figure 3.1a, at the beginning, both the training loss and the validation loss decrease. Then around epoch 25, the validation loss starts to increase and the training loss continues to decrease. Therefore, after epoch 25 the loss curves diverge. Notice that, the loss and the accuracy are calculated at the token level. As the model learns to output almost the exact same sentence that it takes as input - indeed only a small fraction of the tokens need to be changed - even if it learns nothing it can possibly be wrong for a very small proportion of the tokens. As a consequence, the accuracy on the training set and on the validation set in figure 3.1b are quite high.



(a) Training and Validation Loss curves.

(b) Training and Validation Accuracy curves.

Figure 3.1: The Transformer converges after 25 training epochs.

	Precision		Recall		Accuracy	
	Humans	Transformer	Humans	Transformer	Humans	Transformer
a/à	.9176	**.9831**	**.9398**	.8239	.9277	.8960
à/a	.9405	**.9953**	**.9518**	.8717	.9458	.9210
est/et	.9620	.9229	**.9157**	.8161	**.9398**	.8590
et/est	.9186	**.9870**	**.9518**	.6235	**.9337**	.7980
ces/ses	.8202	.8741	**.8795**	.7434	.8434	.8080
ses/ces	**.9859**	.9000	**.8434**	.6207	**.9157**	.7620
ce/se	.9625	**.9954**	.9277	.8810	.9458	.9250
se/ce	.9625	**.9930**	.9277	.8730	.9458	.9230
ou/où	.9474	.9440	.8675	**.9267**	.9096	.9230
où/ou	.9302	.9312	**.9639**	.8958	.9458	.9110
la/là	.9634	**.9978**	.9518	.9268	.9578	.9470
là/la	.9870	.9740	.9157	**.9819**	.9518	.9710
tout/tous	.9286	.9504	.7831	**.8909**	.8614	**.9120**
tous/tout	.8617	**.9430**	**.9759**	.9067	.9096	.9140
leur/leurs	.8657	**.9645**	.6988	**.9333**	.7952	**.9420**
leurs/leur	.9091	.9482	.8434	**.9597**	.8795	**.9480**
ceux/ce	.9753	.9350	.9518	.9819	.9639	.9500
ce/ceux	.9867	.9830	.8916	**.9448**	.9398	.9450
cette/cet	.9157	**.9757**	.9157	**.9679**	.9157	**.9660**
cet/cette	.9651	.9818	1.	.9739	.9819	.9750
Average	.9353	.9590	.9048	.8772	.9205	.9098
Advantage	1	9	8	7	3	4

Table 3.1: Comparison of Transformer performance with those of all respondents. The bold word at the beginning of each line indicates the homophone whose measurements are reported on the line, and the associated form is indicated after the slash. Statistically significant results (p-value < 0.05) are in bold.

Next, 83 people completed the online form, with an average age of 28 years (standard deviation is 15 years). For a detailed presentation of the respondents' profile see appendix **??**. In addition, *when respondents made an error (FP and FN), the average confidence*

level was ***8***, while when the answer was correct (TP and TN), the average confidence level was 9.5. Thus, even when they were wrong, they were very confident.

Finally, Table 3.1 compares the model's performance with human performance on the homophone correction task. Table 3.1 reads as follows: on the first line, for example, "***a***/*à*" means that the sentence proposed to the model or human contains the letter "*a*". The *Precision* on this line indicates then, the probability that the model or the human is right (the confidence that one can have on the modification made) when it changes the "*a*" into "*à*". The *Recall* indicates, among all the times when "*a*" had to be changed into "*à*" (because the sentence is not correct), the probability that the model or the human detects the error. Finally, the *Accuracy* indicates the probability that the model or the human gives the right answer, when "*a*" is present in the proposed sentence (all situations combined). For the second line, we are interested in the results when it is "*à*" that is proposed to the model or to the human. More generally on each line, the results are given in relation to the homophone highlighted in bold at the beginning of the line. The homophone after the slash indicates the word with which the model or human replaces the word in bold (when it replaces it, which is not always the case). Additional comparisons are provided in appendix **??**.

3.3 Discussion

For the comparison of results, one could only compare the average performance. However, comparing the model and the human for each condition of a pair of homophones, gives a more refined measure. Each pair (h_1, h_2) has two conditions, h_1/h_2 (for instance ***a***/*à*) and h_2/h_1 (for instance ***à***/*a*). Therefore, for the 10 pairs there are a total of 20 conditions corresponding to the 20 rows in Table 3.1.

The performance taken condition by condition (each row in Table 3.1) indicates that, for Precision, the model performs better than humans in 9 conditions, the human performs better than the model in 1 condition, and the difference is not statistically significant in 10 conditions. For Recall, the model performs better than humans in 7 conditions, the human performs better than the model in 8 conditions, and the difference is not statistically significant in 5 conditions. Finally, for Accuracy, the model performs better than humans in 4 conditions, the human performs better than the model in 3 conditions, and the difference is not statistically significant in 13 conditions.

Using this way of comparing, we have 60 points of comparison. What emerges from these results is a clear advantage (9-1) for the model for Precision, tight results for Recall (7-8) and Accuracy (4-3). Overall, *the model scores more points (20) than the humans (12), and "wins" against the humans on a score of 20-28-12*, with 28 indicating the number of points that could not be awarded to either because the difference was not statistically significant. We can add a further argument for the relevance of the model: *time*. Indeed, to evaluate the 40 sentences of the questionnaire, the Transformer took only 38 seconds (a little less than 1 second per sentence) - with a score of 37/40. In contrast, humans take much longer, with some respondents reporting 10 to 15 minutes to complete the questionnaire (the exact time taken by the respondents was not measured) - that is, 15 to 20 times longer than the Transformer.

Last but not least, we must discuss the limitations of the present study. In particular, *human performance was biased*. Indeed, the main bias is the way the answers were collected through the online questionnaire. When we make a mistake, we do so by writing it down the first time, then if we check again we may not detect the mistake (or change a word that was actually good), and thus leave the mistake (or introduce a new mistake) after checking. On the one hand, the questionnaire simulated more the 2$^{\text{nd}}$ condition, i.e. a review of what we have already written, since a sentence was presented and the respondent had to judge if it was correct or not. This led to better performance, since all the attention was focused on error detection. All the more so since, in the context of the questionnaire, *the respondents knew there would be errors* (this was explicitly indicated in the presentation of the questionnaire) and were therefore even more watchful. On the other hand, error checking was made even easier, as humans had only two response options, "oui, la phrase est correcte" (yes, the sentence is correct) and "non, la phrase n'est pas correcte, *il faut remplacer x par y*" (no, the sentence is not correct, *we should replace x with y*) where x is a homophone present in the sentence and y its associated homophone. Thus, thanks to the hint, the humans knew exactly where to focus their attention in the sentence, which is far from the real conditions of error correction during a rereading. *This may explain the good performance of humans on the Recall measure* which indicates the ability to detect an error when it is present. While the model, received as input only the sentence without any other indication, and had to output the sentence with correction if necessary. These biases are related to the way human performance was evaluated.

A second, but equally important, bias stems from the composition of the sample of respondents. Indeed, among the respondents, 48% have a level of education higher than BAC+2 (and 10% even have a level of education higher than BAC+5) see figure **??**, while according to INSEE [5, data for 2020], only 25% of the French people have a level of education higher than BAC+2. Moreover, it is also indicated that 18.5% of French people between 25 and 64 years old have no diploma (maximum the "Brevet des collèges"), whereas in the sample of respondents, all the people between 25 and 64 years old have at least the BAC. These two elements combined suggest that the sample is biased toward highly educated individuals. This bias arguably favored human performance.

Lastly, we discuss ideas for further studies extending what has been done here. We used only 10 pairs of homophone, so we could train a model on a much bigger number of pairs. We could also use the same approach but using pairs constituted of subword level units. For instance we can use pairs made of verb endings such as (-*ais*/-*ait*) that would simulate conjugation errors: ("je voul*ais*", "il voul*ait*") $\longmapsto$ ("je voul*ait*", "il voul*ais*"). At the time of writing these lines, a database referencing common mistakes in French doesn't exist. Therefore we could conduct studies in order to constitute a comprehensive database of typical errors that are made. Then we could train a model to corrupt correct sentences by introducing errors inspired by this database instead of injecting errors at random. Thus, we can get an even larger database which can be used to train efficient spell or word checkers, as it has been done for this TER.

3.4 Conclusion

The Transformer introduced by Vaswani et al. was a major breakthrough in Natural Language Processing. Since then, it has been successfully employed in other fields of Artificial Intelligence but also in problems that were not traditionally solved using Deep Learning. In addition to analyzing the Transformer in detail, we also used it to implement an effective word checker, which tackles the problem of homophone correction by leveraging the Transformer's attention mechanism. We then compared its performance with that of humans. Overall, the Transformer performs better and takes considerably less time than humans, even when humans have a significant advantage over the model in correcting homophone-related errors.

Bibliography

[1] François Chollet. *Deep Learning with Python*. Manning Publications, 2017. ISBN: 9781617294433.

[2] James H. Martin Dan Jurafsky. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, Third Edition*. URL: https://web.stanford.edu/~jurafsky/slp3/.

[3] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition*. O'Reilly Media, 2019. ISBN: 9781492032649.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. ISBN: 9780262035613.

[5] Insee. *Diplôme le plus élevé selon l'âge et le sexe*. 2020. URL: https://www.insee.fr/fr/statistiques/2416872#figure1_radio2.

[6] Abdurrahman Shahid. *Github repository*. URL: https://github.com/S-Abdu/Transformer_for_Word-Checking.

[7] Abdurrahman Shahid. *TER Form*. URL: https://forms.gle/zY2wgV51tR5YKEcn7.

[8] Ashish Vaswani et al. "Attention is all you need". In: *arXiv:1706.03762* (2017).

APPENDIX A

Word checker: Materiel Selection and Data Preprocessing

APPENDIX B

Attention Weights Visualization

APPENDIX C

Metrics and Statistical Hypothesis Testing

APPENDIX D

Additional Characteristics of Human Respondents

APPENDIX E

Additional Comparisons