Internship Log Book

Marius Ortega

April 24, 2023

Abstract

This logbook contains the full extent of my research and discovery during my internship. My final objective is the evaluate the Bert Score and Bart Score NLP metrics.

1 Introduction

In the context of my Master 1 internship, I will have to benchmark Bert Score and Bart Score metrics. Having only a limited experience in NLP Study, my first goal is to better understand NLP as a global subject.

To do so, I will study LSTM's internal structure, read and synthesize scientific articles, and then study and implement Bert and Bart Metrics.

2 Concepts Explanation

2.1 LSTM Internal Structure

This section aims at understanding the logical structure of an LSTM Cell. To do so, we will use Figure.1 as a visual support.

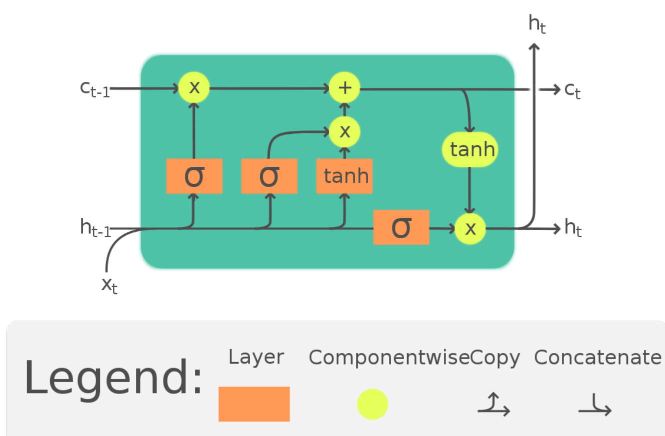

Figure 1: LSTM Cell's Logical Structure Diagram [11]

- **Variables and Symbols :**
  - h_{t-1} and h_t : Respectively the previous and current hidden states
  - c_{t-1} and c_t : Respectively the previous and current cell states
  - x_t : Input value
  - σ : Layer with sigmoid activation function
  - tanh : Layer with hyperbolic tangent activation function

- **Different components of a LSTM cell** :
  - Long Term Memory (Cell State) :

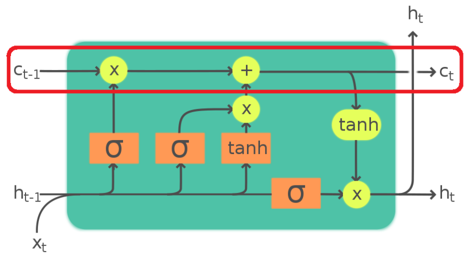

Figure 2: LSTM Cell's Long Term Memory (Cell State) [11]

    - ∗ Cell state isn't modified by any bias or weight
    - ∗ Prevents the gradient from vanishing or exploding
    - ∗ Only gets modified by a multiplicative and an additive component
  - Short Term Memory (Hidden State) :

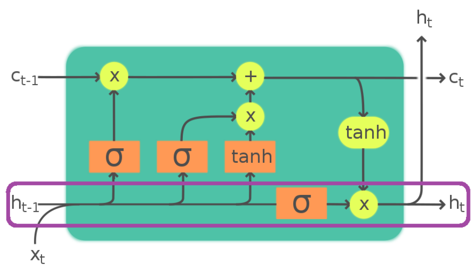

Figure 3: LSTM Cell's Short Term Memory (Hidden State) [11]

    - ∗ Gets summed with the input value to update the long term memory and create the new short term memory
- **Gates of a LSTM Cell** :
  - Forget Gate :

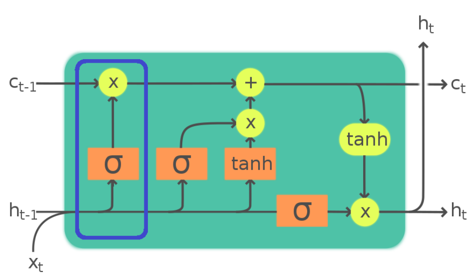

Figure 4: LSTM Cell's Forget Gate [11]

    - ∗ <u>Determines what percentage of the long term memory to keep</u>
    - ∗ Sums Hidden state and Input
    - ∗ Runs the result trough a Sigmoid layer which gives a factor $f \in [0, 1]$
    - ∗ Multiplies f with the previous long term memory c_{t-1}

– Input Gate :

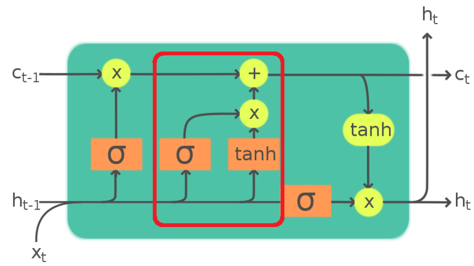

Figure 5: LSTM Cell's Input Gate [11]

- ∗ <u>Updates the long term memory</u>
- ∗ Sums Hidden state and Input
- ∗ Runs the result though a hyperbolic tangent layer to create a potential long term memory
- ∗ In parallel, runs the previous result through a sigmoid function
- ∗ Multiples both of these outputs to determine which percentage of the potential long term memory to keep
- ∗ Sums the previous long term memory and the potential one to finalize the update

– Output Gate :

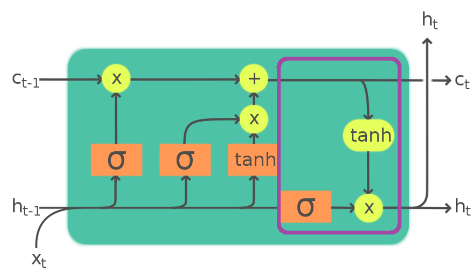

Figure 6: LSTM Cell's Output Gate [11]

- ∗ <u>Updates the short term memory</u>
- ∗ Runs the updated long term memory through a hyperbolic tangent layer to determine a potential short term memory
- ∗ Sums Hidden state and Input
- ∗ Runs it through a sigmoid layer
- ∗ Multiples the potential short term memory and the sigmoid layer output to determine what percentage of the potential short term memory to keep

- **Output of the LSTM cell** :
  - – The updated long term memory becomes the cell state of the next LSTM cell
  - – The update short term memory becomes the hidden state of the next LSTM cell

2.2 Transformers

2.2.1 RNN

"Recurrent Neural Network receives some input (which could be a word or character), feeds it through the network, and outputs a vector called the hidden state. At the same time, the model feeds some information back to itself through the feedback loop, which it can then use in the next step." as explained is the HuggingFace book dedicated to transformers [8]. Thus, it allows us to predict sequential data.

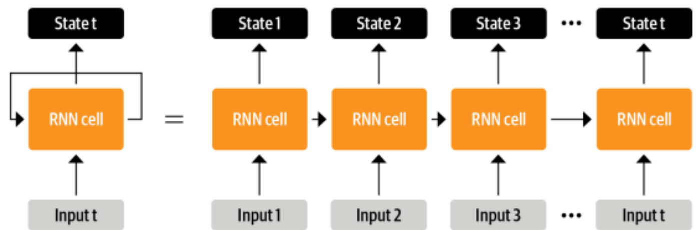

Figure 7: Basic RNN Schema [8]

2.2.2 Encoder-Decoder Framework and Seq2Seq models

We first need to discuss pros and cons of such a framework (see Figure.8).

- **Pros** :
  - Elegant and easy to understand.
  - Efficient on short sequences.
- **Cons** :
  - Creation of a bottleneck after the last hidden state of the encoder. This representation can become inefficient on long sequences.

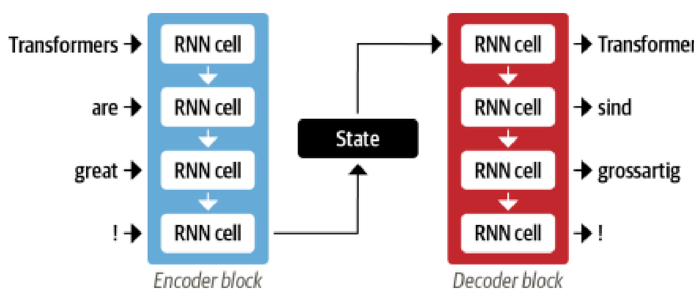

Figure 8: Encoder Decoder Framework schema [8]

2.2.3 Attention Mechanism

The first question to address is how to reduce the "bottleneck" effect :

- The initial idea is to extract the hidden state of each RNN cell so the decoder has access to all of the encoder's hidden states.
- However, this is a computationally very costly task. Attention provides a weighting of hidden states for each decoding step to reduce the quantity of information ingested at each step of the decoder.

In Figure.9, we is shown how the Encoder-Decoder computes the 3rd output state with the help of an attention mechanism.

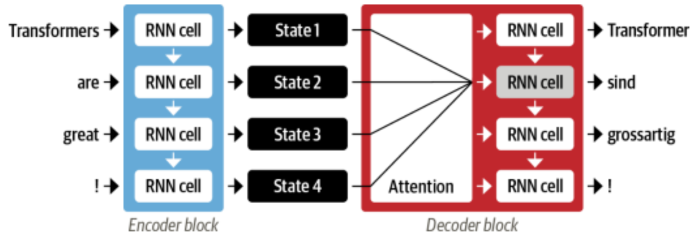

Figure 9: Encoder-Decoder with Attention Mechanism [8]

2.2.4 Transformers

General Definition :

- Encoder-Decoder Architecture.
- Assisted with Self-attention mechanism allowing parallelization of tasks (see Figure.10).

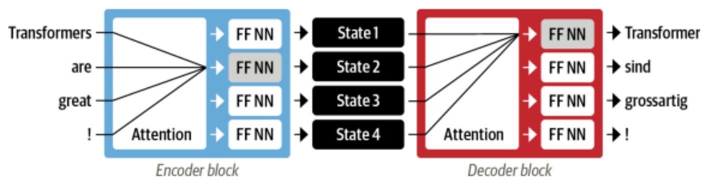

Figure 10: Transformer Basic architecture [8]

3 Articles Syntheses

3.1 ROUGE : A Package for Automatic Evaluation of Summaries

3.1.1 Rouge : Global Definition

- **Rouge** : Recall-Oriented Understudy for Gisting Evaluation.
  - Gisting : Use of machine Translation (MT) to translate foreign texts.
- Rouge **aims** and **main specs** :
  - Automatic evaluation of summaries.
  - Show correlation of it's results with human judgement.

3.1.2 Rouge-N : N-gram Co-Occurrence Statistics

N-gram : Contiguous sequence of n items from a given sample of text or speech

$$Rouge_N = \frac{\sum_{S\in\{ReferenceSummaries\}} \sum_{gram_n \in S} Count_{match}(gram_n)}{\sum_{S\in\{ReferenceSummaries\}} \sum_{gram_n \in S} Count(gram_n)}$$

- With :
  - n : length of the n-gram
  - $gram_n$: n-gram
  - $Count_{match}(gram_n)$: maximum number of n-grams co-occurring in a candidate summary and a set reference summaries
  - $Count(gram_n)$: number of n-grams occurring at the reference summary side

- **Comments** :
  - Recall-related measure because the denominator is equivalent to $TP + FN$
  - While numerator is equivalent to TP
- **Strengths** :
  - A candidate summary that contains words shared by more references is favored. We prefer when there is some kind of consensus among reference summaries.
  - Intuitive and reasonably simple to explain.

Multi-Reference Rouge-N :

$$Rouge_{N_{multi}} = argmax_i(Rouge_N(r_i, s))$$

- With :
  - r_i : i^{th} reference summary
  - s : candidate summary

3.1.3 Rouge-L : Longest Common Subsequence

- **Vocabulary** :
  - LCS : Longest Common Subsequence
  - LCSR : LCS Ratio : $LCSR = \frac{len(LCS(x,y))}{argmax(len(words_x), len(words_y))}$

- **Definition of a Subsequence** :
  - Let :
    - X a sequence, $X = [x_1, x_2, .., x_n]$
    - Z another sequence, $Z = [z_1, z_2, .., z_n]$
    - I strictly increasing sequence of X indices, $I = [i_1, i_2, .., i_n]$
  - Then :

$$\exists I, \forall j \in [1, k], x_{ij} = z_j \Rightarrow \text{Z is a subsequence of X}$$

- **Sentence-Level Rouge-L**:

$$\begin{cases} R_{LCS} = \frac{LCS(X,Y)}{m} & , m = length(X) \\ P_{LCS} = \frac{LCS(X,Y)}{n} & , n = length(Y) \\ \beta = \frac{P_{LCS}}{R_{LCS}} \\ Rouge_{L_{sentence}} = F_{LCS} = \frac{(1+\beta^2)R_{LCS}P_{LCS}}{R_{LCS}+\beta^2 P_{LCS}} \end{cases} \tag{1}$$

  - With :
    - X : reference summary
    - Y : candidate summary
    - LCS(X, Y) : length of the LCS of X and Y
  - Strengths of the sentence-level Rouge-N :
    - In contrast to Rouge-N, **Rouge-L takes into account the order of words**.
  - LCS cons :
    - Only takes the main in-sequence words while ignoring other alternatives (Rouge-N does not suffer this problem).

- **Summary-Level Rouge-L** :
  - Let :
    - C : Candidate sequence : $C = [c_1, c_2, \dots, c_n]$
    - r_i : Reference summary sentence
    - u : Number of sentences in the reference summary
  - Then :

$$\begin{cases} R_{LCS} = \frac{\sum_{i=1}^{u} LCS_{\cup}(r_i,C)}{m} & , m = length(X) \\ P_{LCS} = \frac{\sum_{i=1}^{u} LCS_{\cup}(r_i,C)}{n} & , n = length(Y) \\ \beta = \frac{P_{LCS}}{R_{LCS}} \\ Rouge_{L_{summary}} = F_{LCS} = \frac{(1+\beta^2)R_{LCS}P_{LCS}}{R_{LCS}+\beta^2 P_{LCS}} \end{cases} \tag{2}$$

  - Strength :
    - $Rouge_{L_{summary}}$ doesn't suffer from the same "tunnel vision" effect as $Rouge_{L_{sentence}}$.
    - It takes into account every matching LCS and subsequence even if they are not adjacent given that is only considers there union.

- **Normalized Pair-Wise Rouge-L** :
  - LCS_{MEAD} : Normalized Pair-Wise LCS.

$$\begin{cases} R_{LCS_{MEAD}} = \frac{\sum_{i=1}^{u} LCS_{\cup}(r_i,C)}{m} & , m = length(X) \\ P_{LCS_{MEAD}} = \frac{\sum_{i=1}^{u} LCS_{\cup}(r_i,C)}{n} & , n = length(Y) \\ \beta = \frac{P_{LCS_{MEAD}}}{R_{LCS_{MEAD}}} \\ LCS_{MEAD}(L_1, L_2) = \frac{(1+\beta^2)R_{LCS_{MEAD}}P_{LCS_{MEAD}}}{R_{LCS_{MEAD}}+\beta^2 P_{LCS_{MEAD}}} \end{cases} \tag{3}$$

  - Difference between LCS_{MEAD} and Rouge-L :
    - Normalized pairwise LCS takes the best LCS score while ROUGE-L takes the union LCS scores.

3.1.4 Rouge-W : Weighted Longest Common Subsequence

- **Vocabulary** :
  - WLCS : LCS but we keep in a dynamic 2D array the longest consecutive matches.
- **WLCS Algorithm** :

```
(1) For (i = 0; i <=m; i++)
      c(i,j) = 0  // initialize c-table
      w(i,j) = 0 // initialize w-table
(2) For (i = 1; i <= m; i++)
      For (j = 1; j <= n; j++)
        If x_i = y_j Then
          // the length of consecutive matches at
          // position i-1 and j-1
          k = w(i-1,j-1)
          c(i,j) = c(i-1,j-1) + f(k+1) – f(k)
          // remember the length of consecutive
          // matches at position i, j
          w(i,j) = k+1
        Otherwise
          If c(i-1,j) > c(i,j-1) Then
            c(i,j) = c(i-1,j)
            w(i,j) = 0          // no match at i, j
          Else c(i,j) = c(i,j-1)
            w(i,j) = 0          // no match at i, j
(3) WLCS(X,Y) = c(m,n)
```

Figure 11: WLCS Algorithm pseudo-code [5]

  - With :
    - c : dynamic table with c_{ij} storing the WLCS score ending at word x_i of X and at word y_j of Y.
    - f : function of consecutive matches at the table position c(i, j).
- **Rouge-W Formula** :

$$\begin{cases} R_{WLCS} = f^{-1}\left(\frac{WLCS(X,Y)}{m}\right) & , m = length(X) \\ P_{WLCS} = f^{-1}\left(\frac{WLCS(X,Y)}{n}\right) & , n = length(Y) \\ \beta = \frac{P_{WLCS}}{R_{WLCS}} \\ Rouge_W = F_{WLCS} = \frac{(1+\beta^2) R_{WLCS} P_{WLCS}}{R_{WLCS} + \beta^2 P_{WLCS}} \end{cases} \quad (4)$$

- **f weighting function** :

  $f(a+b) > f(a) + f(b), a, b \in N$

  - Examples :
    - $f(k) = \alpha k - \beta, \qquad k \geq 0, \alpha > 0, \beta > 0$
    - $f(k) = k^{\alpha}, \qquad \alpha > 1$
      - We prefer polynomials that can be inverse in closed form like k^2.
- **Strength of Rouge-W** :
  - It rewards consecutive matches in contrast to Rouge-L.

3.1.5 Rouge-S : Skip-Bigram Co-Occurrence Statistics

- **Vocabulary** :
  - Skip-Bigram : All combinations of possible bi-grams in a sentence.
- **Rouge-S Formula** :

$$\begin{cases} R_{SKIP2} = \frac{SKIP2(X,Y)}{C_2^m} & , m = length(X) \\ P_{SKIP2} = \frac{SKIP2(X,Y)}{C_2^n} & , n = length(Y) \\ \beta = \frac{P_{SKIP2}}{R_{SKIP2}} \\ Rouge_W = F_{SKIP2} = \frac{(1+\beta^2) R_{SKIP2} P_{SKIP2}}{R_{SKIP2} + \beta^2 P_{SKIP2}} \end{cases} \tag{5}$$

  - We can reduce the **skip distance** to only try to match word that are close in the sentence.
  - **Weakness of Rouge-S** :
    * Doesn't give any credit to a candidate sentence when there are no word co-occurring even if they have uni-grams in common.
- **Rouge-SU** :
  - We add uni-gram as counting unit to the basic Rouge-S to eliminate the main weakness of Rouge-S.

3.1.6 Evaluation of Rouge

Before considering the results of the conducted experiment, let's evaluate the quality of the protocol. To do so, we will focus 3 main aspects :

- **Datasets quality** :
  - Is it gold standard or often utilized for benchmarking ?
  - Is its size consequent enough ?
  - Is is biased or noisy (due to preprocessing choices) ?
- **Algorithm's Complexity** :
  - Spacial Complexity
  - Time Complexity
- **Parameters affecting the algorithm's robustness**.

We summarize these KPIs in the following table :

Table 1: Rouge's Evaluation conditions Study

(a) Dataset Study Table

Datasets KPIs	Comments
Benchmark-oriented or Gold Standard	Yes, they are gold standard and human annotated datasets widely used in scientific experiments
Size	Relatively small datasets : Only 1127 annotated summaries in DUC 2001,2002 and 2003 put together (considering all summaries size)
Biased or Noisy	No information were given relatively to the preprocessing but we know they used stop words to improve their algorithms performances

(b) Complexity Study Table

Complexities	Comments
Spatial	No information on the implementation
Time	No information on the benchmarking hardware and the algorithm run time

(c) Robustness to variation Study Table

Parameters	Comments
Stop words	Rouge is subject to noise disturbance : words like "I" or "the"
Summary Size	Some metrics like Rouge-S4 reach peak performance for very short summaries (10 words) but become inconsistent if the size comes to change
Method of Measurement	Only Pearson's correlation results are shown in the paper. However, the measure doesn't take into account non-linear relationships and Independence between variables.

3.1.7 Conclusion of Rouge's performances

Now that we took the context of the evaluation into account, we can study the experiment's results. Let's sum up which measures are the best depending of the required task :

- **Single document summarizing :**
  - Rouge-2, Rouge-L, Rouge-W, Rouge-S
- **Short/Headline Single document summarizing :**
  - Rouge-1, Rouge-L, Rouge-SU4, Rouge-SU9
- **Multi document summarizing :**
  - Rouge-1, Rouge-2, Rouge-S4, Rouge-S9, Rouge-SU4, Rouge-SU9
    (couldn't reach 90% correlation)

3.2 BERTScore : Evaluating Text Generation with BERT

3.2.1 Introduction to BERTScore

BERTScore is an automatic evaluation metric for text generation. It addresses two common downsides of n-gram-based metrics :

- Lack of robustness to match paraphrases.
  - BERTScore computes similarity using contextualized embedding.
  - Very efficient technique for paraphrase matching.
- Fail to capture distant dependencies and penalize semantically-critical ordering changes.
  - Contextualized embedding is designed to solve such problems.

3.2.2 Definition of BERTScore

In this section, we will explain how to calculate BERTScore based on Figure.8 :

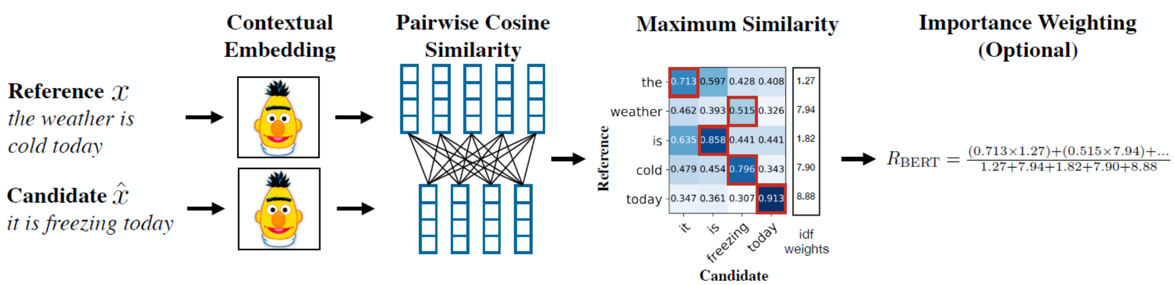

Figure 12: BERTScore Computing Process [16]

- **Contextual Embedding** :
  - Generates different vector representations for the same word in different sentences depending on the surrounding words.
  - Gives context to the embedding.
  - Unlike classical embedding that provides a single vector representation per word.
- **Pairwise Cosine Similarity Measurement** :
  - Let :
    * x_i : Reference token
    * $\hat{x}_j$: Candidate token
    * S : Cosine similarity between x_i and $\hat{x}_j$
  - Then :

$$S_{ij} = \frac{x_i^T \hat{x}_j}{||x_i^T||\ ||\hat{x}_j||}$$

  - Comment :
    * BERTScore implementation pre-normalizes vectors to prevent from calculating the denominator of S_{ij}.

- **BERTScore - Maximum Similarity** :
  Greedy matching is used to maximize the similarity scores. Each token is matched to the most similar token of the other sentence.
  - Let :
    - x_i : Reference token
    - $\hat{x}_j$: Candidate token
  - Then :

$$\begin{cases} Recall & R_{BERT} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} (x_i^T \hat{x}_j) \\ Precision & P_{BERT} = \frac{1}{|\hat{x}|} \sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} (x_i^T \hat{x}_j) \\ F1Score & F_{BERT} = 2 \frac{P_{BERT} \cdot R_{BERT}}{P_{BERT} + R_{BERT}} \end{cases} \tag{6}$$

- **Importance Weighting (Optional)** :
  - Observation :
    - Rare words can be more indicative for sentence similarity than common words.
    - Thus, we can use IDF (Inverse Document Frequency) to incorporate importance weighting to BERTScore.
    - In the same fashion as search engines.
  - Inverse Document Frequency :
    - Let :
      - M : Number of reference sentences
      - $\{x^{(i)}\}_{i=1}^{M}$: Reference Sentence
      - w : Word-piece token
    - Then :

$$idf(w) = -\log(\frac{1}{M} \sum_{i=1}^{M} (\mathbb{1}[w \in x^{(i)}]))$$

  - Recall with IDF :

$$R_{BERT} = \frac{\sum_{x_i \in x} idf(x_i) \max_{\hat{x}_j \in \hat{x}} (x_i^T \hat{x}_j)}{\sum_{x_i \in x} idf(x_i)}$$

    - idf score remains the same for a specific test set because it is computed based on reference sentences.
    - Plus-one smoothing is applied to handle unknown word pieces.

- **Baseline Re-scaling** :
  - In theory, BERTScore lies between -1 and 1 (because we use normalized vectors to compute similarity).
  - In practice, BERTScore lies in a much smaller range due to the learned geometry of the contextual embedding.
  - It doesn't affect its performance but makes it hardly readable.
  - Thus we re-scale it between 0 and 1 so it is more evenly distributed in this range. Let's take an example with the recall :

$$\hat{R}_{BERT} = \frac{R_{BERT} - b}{1 - b}$$

    - b : The empirical lower bound computed with Common Crawl Monolingual Datasets

3.2.3 BERTScore Experimental Setup

This section aims at evaluating the quality of the experimental setup of the BERTScore experiment described in this paper [16]. Given that BERTScore is trained to answer Machine Translation and Image Captioning problems, we will treat each of these aspects separately without forgetting about the transfer learning methods to obtain Contextual Embedding Models :

- **Contextual Embedding** :
  - Transfer Learning :
    - 24-layer $RoBERTa_{large}$ for English tasks
    - 12-layer $BERT_{chinese}$ for Chinese tasks
    - 12-layer cased multilingual $BERT_{multi}$ for other languages
  - Evaluation Datasets :
    - WMT16 : Considered gold standard dataset. However, is biased because most of the corpus of this dataset is composed of News Articles (meaning the training is made on well structured and well written data).
- **Machine Translation** :
  - Datasets quality :
    - Main evaluation dataset : WMT18
    - Other datasets : WMT16 and WMT17
    - All Gold standard datasets.
  - Evaluation of BERTScore quality :
    - Absolute Pearson correlation $|\rho(X,Y)| = |\frac{cov(X,Y)}{\sigma_X \times \sigma_Y}|$ [12]
      - Measures linear correlation between two variables.
    - Kendall rank correlation $\tau = \frac{2}{n(n-1)} \sum_{i<j} sgn(x_i - x_j)\ sgn(y_i - y_j)$ [10]
      - Often used as a statistic test to verify if two variables are statistically dependant.
    - Significance with Williams test [2]
      - Allows to determine whether or not the variation rate is significantly different from the hypothesis.
      - Equal to student test from $k = 1$
- **Image Captioning** :
  - Datasets quality :
    - Evaluation Dataset : COCO 2015, Captioning Challenge.
    - COCO Dataset is composed of descriptive texts as inputs.
    - Not considered a gold standard dataset but is a very popular benchmarking dataset and has been used extensively.
  - Evaluation of BERTScore quality :
    - Pearson correlation ρ
- **General performance** :
  - Speed :
    - BERTScore computes at an average speed of 192.5 candidate-reference pairs/second using a GTX-1080Ti GPU.
    - Computes 2998 sentences in 15.6s while SacreBLEU compute it in 5s.
    - Considering the size of BERTScore's pre-trained model, it is satisfyingly fast.

- Robustness :
  - Good resilience to word swapping, tested with PAWS dataset [17] :
    - Example of PAWS semantically identical swapped phrases : "Although interchangeable, the body pieces on the 2 cars are not similar." and "Although similar, the body parts are not interchangeable on the 2 cars.".
    - Example of PAWS semantically different swapped phrases : "Katz was born in Sweden in 1947 and moved to New York City at the age of 1." and "Katz was born in Sweden in 1947 and moved to New York City at the age of 1.".
  - Good resilience to paraphrase, tested with QQP dataset.
  - Especially when BERTScore models aren't trained on paraphrasing datasets.

3.2.4 BERTScore Results

- Machine Translation

Metric	en↔cs (5/5)	en↔de (16/16)	en↔et (14/14)	en↔fi (9/12)	en↔ru (8/9)	en↔tr (5/8)	en↔zh (14/14)
BLEU	.970/**.995**	.971/**.981**	**.986/.975**	.973/**.962**	.979/**.983**	**.657**/.826	.978/.947
ITER	.975/.915	.990/**.984**	.975/**.981**	**.996/.973**	.937/.975	**.861**/.865	.980/ –
RUSE	.981/ –	.997/ –	**.990/** –	.991/ –	**.988/** –	**.853/** –	**.981/** –
YiSi-1	.950/**.987**	.992/**.985**	.979/**.979**	.973/.940	**.991/.992**	**.958/.976**	.951/**.963**
P_{BERT}	.980/**.994**	**.998/.988**	**.990/.981**	.995/.957	.982/**.990**	**.791/.935**	.981/.954
R_{BERT}	**.998/.997**	.997/**.990**	.986/**.980**	**.997/.980**	**.995/.989**	.054/.879	**.990/.976**
F_{BERT}	**.990/.997**	**.999/.989**	.990/**.982**	**.998/.972**	**.990**/.990	**.499**/.908	**.988**/.967
F_{BERT} (idf)	.985/**.995**	**.999/.990**	**.992/.981**	.992/**.972**	**.991/.991**	**.826/.941**	**.989/.973**

Figure 13: BERTScore results on standard WMT18 dataset [16]

- Numbers in parentheses indicate the number of systems for each pair of language (i.e. the number of documents).

Metric	en↔cs	en↔de	en↔et	en↔fi	en↔ru	en↔tr	en↔zh
BLEU	.956/.993	.969/**.977**	**.981**/.971	.962/.958	.972/.977	.586/.796	.968/.941
ITER	.966/.865	.990/.978	.975/**.982**	.989/.966	.943/.965	.742/.872	.978/ –
RUSE	.974/ –	.996/ –	.988/ –	**.983/** –	.982/ –	.780/ –	.973/ –
YiSi-1	.942/.985	.991/.983	.976/.976	.964/.938	**.985/.989**	**.881/.942**	.943/.957
P_{BERT}	.965/.989	.995/.983	**.990/.970**	.976/.951	.976/.988	.846/.936	.975/.950
R_{BERT}	**.989/.995**	.997/**.991**	.982/**.979**	.989/**.977**	**.988/.989**	.540/**.872**	**.981/.980**
F_{BERT}	.978/**.993**	.998/.988	.989/.978	.983/.969	.985/.989	.760/.910	**.981**/.969
F_{BERT} (idf)	.982/.995	**.998**/.988	**.988**/.979	**.989**/.969	.983/.987	.453/.877	.980/.963

Figure 14: BERTScore results on 10K super-sampled WMT18 dataset [16]

- BERTScore outperforms all metrics (in terms of consistency).
- F_{BERT} is the most consistent. Kept for implementation.
- IDF not retained because the gain is negligible.

- Image Captioning :

Metric	M1	M2
BLEU	-0.019*	-0.005*
METEOR	0.606*	0.594*
ROUGE-L	0.090*	0.096*
CIDER	0.438*	0.440*
SPICE	0.759*	0.750*
LEIC	**0.939***	**0.949***
BEER	0.491	0.562
EED	0.545	0.599
CHRF++	0.702	0.729
CHARACTER	0.800	0.801
P_{BERT}	-0.105	-0.041
R_{BERT}	0.888	0.863
F_{BERT}	0.322	0.350
R_{BERT} (idf)	**0.917**	**0.889**

Figure 15: BERTScore results on 2015 COCO Captioning Challenge [16]

  - BERTScore outperforms task-agnostic based methods and n-grams based methods.
  - Only LEIC performs better than BERTScore because it is specifically design to work on COCO data.
  - However, we notice that the precision is particularly poor for image captioning. Only R_{BERT} is used instead of F_{BERT}. Though, F_{BERT} is a more balanced measure than R_{BERT}. Thus, we can't assess that BERTScore is efficient for image captioning.
  - In addition, it doesn't seem relevant to compute $R_{BERT}(idf)$ because inverse document frequency becomes consistent when the corpus is sufficiently big. In the context of image captioning, the corpus is small and the probability of finding two identical words decreases consequently.

3.2.5 BERTScore personal experimentation : Computation of IDF

In this subsection, we will try to understand how the calculation of the IDF is implemented in BERTScore's code [16]. The article only gives us the formula for Recall using IDF :

$$R_{BERT} = \frac{\sum_{x_i \in x} idf(x_i) \max_{\hat{x}_j \in \hat{x}} (x_i^T \hat{x}_j)}{\sum_{x_i \in x} idf(x_i)}$$

From there, we could ask ourselves multiple questions :

- Does Precision use gold summary vocabulary to be compiled ? If yes, this would be considered biased for reasons we will explain later.
- Is IDF calculated using only the reference vocabulary ?
- Should the vocabulary be common to reference and candidate for the computation of IDF ?
- How did researchers implement it ?

After reading the code in the utils.py file, we were able to answer these questions :

- **Calculation of maximum similarities** :

```
word_precision = sim.max(dim=2)[0]
word_recall = sim.max(dim=1)[0]
```

- **IDF normalization setup** :

```
hyp_idf.div_(hyp_idf.sum(dim=1, keepdim=True))
ref_idf.div_(ref_idf.sum(dim=1, keepdim=True))
precision_scale = hyp_idf.to(word_precision.device)
recall_scale = ref_idf.to(word_recall.device)
```

- **Precision and Recall computation** :

```
P = (word_precision * precision_scale).sum(dim=1)
R = (word_recall * recall_scale).sum(dim=1)
```

From the code above, we can see that Precision is indeed computed using gold hypothesis vocabulary while recall is calculated using reference's vocabulary. This discovery means that the computation of the metric uses gold information which consequently increases the score and perverts the result.
An alternative would be to only use reference vocabulary when computing the IDF of the hypothesis. If the word doesn't exist we give it the IDFScore of 1 so it doesn't influence the general score.

3.3 BARTScore : Evaluating Generated Text as Text Generation

In this section we will synthesis and explicit BARTscore measure theorized in the Carnegie Melon University [14].

3.3.1 Introduction to BARTScore

BARTScore is a NLP metric designed to perform text evaluation :

- Evaluates text through its probability of being generated in a particular context.
- Unsupervised metric

Global functioning of BARTScore :

- $s = s_1, s_2, \cdots, s_n$: Source
- $h = h_1, h_2, \cdots, h_m$: Hypothesis generated based on the source
- $r = r_1, r_2, \cdots, r_l$: Reference. Human-made comparison helping for evaluation

BARTScore's goal is to generate h based on s with one or multiple human-made references r used for for evaluation.

3.3.2 Definition of BARTScore

- **Seq2Seq Pretrained Model** :

  BART is used as backbone :

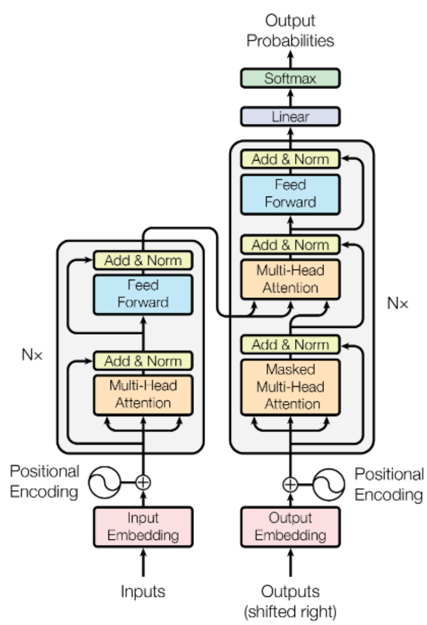

Figure 16: BART backbone [9]

  - Let a given Seq2Seq model with :
    - θ : Parameter of the Seq2Seq model.
    - $x = \{x_1, x_2, \cdots, x_n\}$: Source sequence of n tokens.
    - $y = \{y_1, y_2, \cdots, y_m\}$: Target sequence of m tokens.
  - Then :

$$p(y|x, \theta) = \prod_{t=1}^{m} p(y_t|y_{<t}, x, \theta)$$

- **BARTScore** :
  - Let :
    - $x = \{x_1, x_2, \cdots, x_n\}$: Source sequence of n tokens.
    - $y = \{y_1, y_2, \cdots, y_m\}$: Target sequence of m tokens.
    - $\omega = \{\omega_1, \omega_2, \cdots, \omega_m\}$: Weights for each token ($\omega_t = 1$ in the paper [14]).
  - Then :
$$BARTScore = \sum_{t=1}^{m} \omega_t log(p(y_t|y_{<t}, x, \theta))$$
  - 4 ways of computing BARTScore :
    - Faithfulness ($s \rightarrow h$) : How likely was the hypothesis generated from the source : $p(h|s, \theta)$
    - Precision ($r \rightarrow h$) : How likely could the hypothesis be constructed based on the gold reference : $p(h|r)$
    - Recall ($h \rightarrow r$) : How easily a gold reference could be generated from the hypothesis : $p(r|h)$
    - F1-score : $2\frac{Precision*Recall}{Precision+Recall}$
- **BARTScore Variants** :
  - Prompt :
  - Fine-Tuning Task :

3.3.3 BARTScore Experimental Setup

- **Tasks performed for evaluation** :
  - Unsupervised Matching : Measures the semantic equivalence between the reference and hypothesis by using a token-level matching functions in distributed representation space, such as BERTScore.
  - Supervised Regression : Introduces a parameterized regression layer, which would be learned in a supervised fashion to accurately predict human judgments.
  - Supervised Ranking : Learns a scoring function that assigns a higher score to better hypotheses than to worse ones.
  - Text Generation : High-quality hypothesis will be easily generated based on source or reference text or vice-versa.
- **Gold Standard human reference KPIs** :
  - Informativeness : How keys ideas are captured
  - Relevance : Consistency with respect to reference text
  - Fluency : Grammatical, semantic, formatting correctness
  - Coherence : How sentences make sense as a whole
  - Factuality : Is information rigorously taken from reference text
  - Semantic Coverage : How semantic content from reference are captured
  - Adequacy : How the output conveys the same meaning as the input
- **Additional information** :
  - Evaluation on 16 datasets over 7 different perspectives

3.3.4 BARTScore Results

4 Dataset Selection

4.1 Summary Evaluation

4.2 Fake Detection

4.3 Machine Translation

5 Various Experimentation

5.1 BERTScore performances with RoBERTa and Word2Vec embeddings

Even though BERTScore is already a relatively fast embedding method, we are curious to know how BERTScore would perform with a different embedding. In this context, we will compare it to a static embedding method and analyze differences.

5.1.1 KPIs, Datasets and Environment

- **KPIs of the study** :

  We will focus on two parameters for this study :

  - Run-time : How much time does the algorithms take to compute BERTScore for a fixed number of individuals.
  - BERTScore quality : We will focus on F1 score associated to each Reference/Candidate couple. The higher the F1 score, the more qualitative the BERTScore.

- **Dataset used** :

  We will use BillSUM dataset :

  - Gold Standard dataset.
  - Available in multiple illustrious libraries such as TensorFlow.
  - Consists of summaries of bills from the U.S. Congress and the State of California and their associated reference document.

- **Working Environment** :

  The results presented here have been computed using the following configuration :

  - Mother Board : MSI PRO Z690-A WIFI DDR4 ATX LGA1700
  - CPU : Intel Core i5-12600KF 3.7 GHz 10-Core
  - GPU : GeForce RTX 3060 Ti
  - Ram : 4x8 GiB

5.1.2 Calculation of BERTScores for both embeddings

We used pre-trained embeddings available on HuggingFace or proposed by default in the BERTScore library :

- **RoBERTa** : We used the BERTScore lib's default embedding for English language. 24-layers $RoBERTa_{large}$.
- **Word2Vec** : We used a Hugging Face Dataset (see reference).

5.1.3 Results

To minimize compilation time, I only perform the BERTScore of the first 3 individuals. To get an informal idea of performances.

	runtime
Roberta-24-layers	12.034703
Word2Vec	5.338707

Figure 17: RoBERTa VS Word2Vec runtime

Word2Vec is more than 2 twice as fast as RoBERTa for the 3 articles considered.

	Bert_P	Bert_R	Bert_F
0	0.874369	0.704338	0.780197
1	0.846002	0.728316	0.782760
2	0.873844	0.702725	0.778998

(a) RoBERTa's BERTScore quality

	W2V_P	W2V_R	W2V_F
0	0.574693	0.328203	0.417802
1	0.737639	0.645986	0.688777
2	0.544604	0.436672	0.484702

(b) Word2Vec's BERTScore quality

Figure 18: Informal BERTScore's embeddings quality comparison

Looking especially at the F1-score of each model, we notice that BERT embedding allows to get a higher and more consistent BERTScore than Word2Vec Embedding. From now on, we must decide if Word2Vec's loss in quality is acceptable because of the runtime improvement.

References

[1] Andrew Nguyen Deep Learning AI. Seq2Seq models basics. Youtube, 2018.

[2] Alain Carpentier. The Williams' Test, 2007.

[3] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

[4] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[5] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out (ACL 2004 Workshop)*, 2004.

[6] Josh Starmer StatQuest. Long Short-Term Memory (LSTM), Clearly Explained. Youtube, 2022.

[7] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *Arxiv*, 2014.

[8] Lewis Tunstall, Leandro von Werra, and Thomas Wolf. *Natural Language Processing with Transformers*. O-Reilly, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2022.

[9] Width. Bart text summarization, 2022.

[10] Wikipedia. Kendall rank correlation coefficient — Wikipedia, the free encyclopedia, 2023.

[11] Wikipedia. Long short-term memory, 2023.

[12] Wikipedia. Pearson correlation coefficient — Wikipedia, the free encyclopedia, 2023. [Online; accessed 7-April-2023].

[13] Yinfei Yang, Yuan Zhang, Chris Tar, and Jason Baldridge. PAWS-X: A Cross-lingual Adversarial Dataset for Paraphrase Identification. In *Proc. of EMNLP*, 2019.

[14] Weizhe Yuan, Graham Neubig, and Pengfei Liu. Bartscore : Evaluating generated text as text generation. In *Arxiv*, 2021.

[15] Weizhe Yuan, Graham Neubig, and Pengfei Liu. Bartscore: Evaluating generated text as text generation. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 27263–27277. Curran Associates, Inc., 2021.

[16] Tianyi Zhang*, Varsha Kishore*, Felix Wu*, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations*, 2020.

[17] Yuan Zhang, Jason Baldridge, and Luheng He. PAWS: Paraphrase Adversaries from Word Scrambling. In *Proc. of NAACL*, 2019.

[5] [11] [6] [16] [17] [13] [12] [10] [2] [15] [4] [3] [1] [7] [8] [9] [14]